GPRS-Based Cinema Ticket Reservation System

Mihai Balan

Kongens Lyngby 2007
IMM-2007-7b

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Abstract

The purpose of this MSC project is to create a location-aware service for GPRS-enabled mobile devices. This service is called Cinema Ticket Reservation System and it can determine user's current position, allow users to search for movies in a given range from their current position, or reserve/purchase tickets. Users can pay for tickets using credit cards saved in a secure wallet embedded into the application, or e-money received as refund for the canceled tickets. An authentication mechanism based on the Needham-Schroeder protocol is implemented. A user-centered design is considered. Workshops and interviews are conducted with real users to build and evaluate different low and high-fidelity prototypes. GPRS is used as a network carrier for all client-server requests. J2ME, J2EE, Bouncy Castle cryptographic libraries, and postgreSQL DB are chosen as implementation technologies. Different optimization techniques are used to increase the overall system performance. The marketing strategies for lunching this service are analyzed. This prof-of-concept prototype shows how a user-centered approach can drive the design and implementation phases of a web service, and how several technologies can be merged together to create a successful service.

Preface

This thesis was prepared at Informatics Mathematical Modelling, the Technical University of Denmark in partial fulfillment of the requirements for acquiring the M.Sc. degree in Computer Science and Engineering.

This MSC project implements a mobile Location Aware Cinema Ticket Reservation Service using a user-centered design approach. Movie goers can search for movies in a given range from their current position, view movie details, rate movies, or reserve/purchase movie tickets using credit cards saved in their secure wallet, or e-money received as refund for previously canceled tickets.

Both the client and the server side are implemented. A user-centered design is considered for constructing different low and high-fidelity prototypes. Different workshops and user interviews are conducted for evaluating the prototypes. A final version of the prototype is proposed and implemented. Human-Computer Interaction, J2ME, J2EE, Tomcat, postgreSQL, Cryptography, etc are the technologies used used for implementation purposes.

The thesis consists of a report describing the implemented prototype, and the source code of the prototype.

Lyngby, August 2007

Mihai Balan

Acknowledgements

I would like to thank the following persons for their help and understanding during this thesis:

Jens Thyge Kristensen, IMM - DTU: for his kind help and inspiring ideas during the project.

Mirela Ramona Balan, my wife: for her understanding and patience while working on the thesis.

Lucia Burlacu, my mother: who never gives up.

(and last but not the least, God): for making this possible.

Contents

Chapter 1

Introduction

This MSC project implements a mobile Location Aware Cinema Ticket Reservation Service using a user-centered design approach. Movie goers can search for movies in a given range from their current position, view movie details, rate movies, or reserve/purchase movie tickets using credit cards saved in their secure wallet, or e-money received as refund for previously canceled tickets. *Human-Computer Interaction, J2ME, J2SE, J2EE, JDBC, Tomcat, Bouncy Castle cryptographic libraries, and postgreSQL DB* are used as implementation technologies. Both, the client side running on a GPRS-enabled mobile device, and the server side service are implemented. Workshops and interviews are conducted with real movie goers to achieve a very usable and user-friendly prototype. Several low and high fidelity prototypes are created and evaluated by real users. An authentication mechanism based on the *Needham-Schroeder protocol* is implemented to give users access to the service. User's sensitive data are stored encrypted in the mobile device. Different optimization techniques are applied to increase the overall service performance i.e. optimization of the application model initialization, unified communication protocol coupled with synchronous object passing during the network communication, prepared statements and connection pools to access the DB, design patterns, etc. The market perspectives of such a service are analyzed. Different solutions for selling this service are formulated.

The current version of the prototype is considered a prof-of-concept of a com-

mercial product. It shows how a user-centered approach can drive the design and implementation phases of any service, while optimization techniques can increase the overall system performance. The goal is to create a very usable and user friendly service that can be run on as many mobile devices as possible to allow movie goers to search for movies, reserve tickets, and offer a great deal of mobility to any movie goer. Both, the cinema industry and movie goers can gain benefits from this service.

1.1 Scenario

Alice and Christian are visiting Copenhagen. Until now they have seen most of the city's attractions. They notice a poster advertising a new movie while shopping. They would like to see it but they do not know where to buy tickets for that movie.

They remember the information and mobile phone application received in the *Tourism Info Center* at their arrival in Copenhagen. The application is called *Mobile Cinema Center* and it allows searching and purchasing tickets for different movies in the whole country. Directions on how to reach the cinema from the given the current position are also offered.

Alice opens the *Mobile Cinema Center* application on her mobile phone. A welcome screen is displayed and she is asked to introduce her credentials. She keys in the user name and password created in the Tourism Info Center and the system authenticates her. The application main menu is displayed. Alice notices that following entries in the main menu: *Find Movies*, *My Tickets*, *My Wallet*, *My Settings*, *Help*, and *EXIT*. She chooses *HELP* and she reads details about the application and the company that guaranties the application. She reads also about different topics as secure payment, secure wallet, etc. Alice goes back to the main menu and selects the *Find Movies* entry this time. A new screen opens and she is asked to key in her current position in the city i.e. *street*, *zip*, *city*, a *movie* she would like to see, a *range* value i.e. the maximum distance she is willingly to travel from her current position to the cinema, and a *date* for the selected movie. Alice keys in all that information and presses the *Search* button. A *progress bar* is displayed on the screen while the request is processed. After a few seconds, a list of all shows matching the given search criteria is displayed. Alice agrees with Christian on the show they would like to see, selects that show and presses the *Select Seats* button. A screen containing the cinema hall configuration is displayed. She can see that there are small red squares among some green ones. She notices the legend that explains the color markup. She can see the position of the screen and base of that she and Christian select two seats. Once this is done, she navigates to the *Option* menu and chooses the *Purchase Ticket(s)* option.
A *Ticket Summary* screen is displayed containing all ticket information i.e. movie, cinema name and address, cinema theater, row and seat no, price for each ticket, and the total price to be paid including all taxes. They verify the information and press the *Accept* button.
Several payment methods are displayed e.g. *Credit Card*, *My Wallet*, and *At the Cinema*. She remembers that there is a *My Wallet* entry in the main menu. But for the moment they would like only to reserve the tickets. Therefore, she chooses the *Pay at the Cinema* method and presses *Purchase*. A new progress

bar is displayed while the request is processed.
After a few seconds a *Payment Summary* screen is displayed. They can see all ticket information and the chosen payment method. They are also informed that they have to be at the cinema with at least 30 min before the show to pay for their tickets or the tickets will be canceled automatically by the system. They are also advised to keep their *m-tickets* to get access to the show. Alice presses the *OK* button and the application returns to the main menu.

Alice and Christian decide to go and see the movie. They open the application and select *My Tickets* option under *Main Menu*. The two previous reserved tickets are displayed. They select one ticket and check the ticket details to find out the address of the cinema. Once arrived at the cinema they open the application again, display the ticket and go to a check point where they slide the mobile on a scanner. The machine reads the ticket and reservation ID's and ask them for a payment method i.e. credit card or cash. Once the tickets are payed, two tickets are printed out by the machine stating the show, date/hour, seats, and total payed amount.

They liked the movie so much that after the show they decide to rate the movie. They open the *Mobile Cinema Center* once again, search for that movie, and choose the *Rate Movie* option. A new screen is opened and ask them to key in the user name and the rating score. When the submit button is pressed their score is recorded in the system and made available to the other users.

Chapter 2

Analysis

This section argues different choices taken during the analysis of the *GPRS-Based Cinema Ticket Reservation System*. It proposes and discusses different solutions to the identified problems and argues the chosen ones.

2.1 Identified Issues in the Cinema Ticket Reservation System

The following issues are identified during the analysis of the Cinema Ticket Reservation System:

- **Mobile Device Limitations**: limited device hardware i.e. small memory, low CPU power, short battery life, small display, limited input capability, etc.
- **MIDlet**: UI design for a CLDC 1.1/MIDP 2.0 based mobile device; build rich and user friendly UI; prevent UI lock-up during the network operations; make the information easy to read on a small screen; overcome the limited input capabilities.
- **User Centered Design of the Prototype**: involve users in all steps of the system design; obtain feed back from users.
- **Location-aware MIDlet**: determine user's current position; retrieve a list of all cinemas and movies in a certain range from the user's current position; hardware and software necessary for a location-aware application.
- **On-device data storing**: store user preferences and cinema service configuration parameters on the mobile device; secure the on-device data.
- **Secure Wallet**: secure saved users' sensitive data e.g. credit cards, on the mobile device.
- **MIDlet Configuration**: secure MIDlet configuration from the server side.
- **Server Side Data Storage**: store users' data and credit card information on the server side; security of credit card - based transactions over the Internet?
- **Mobile Device - Server-side Cinema Service Communication(SCS)**: communication between the MIDlet and SCS; error handling; and quality-of-service; minimum communication costs; guarantee the security and integrity of transactions; achieve a secure communication.
- **SCS - Server-side Data Storage Communication**: reduce the access time to retrieve the information from the server side data storage solution.
- **Security Concerns**: threats one can encounter; secure communication between the MIDlet and SCS; user authentication; send data over the air in a secure way; secure payment transactions.

- **Slow and Unreliable Networks**: slow and unreliable networks; overcome network latency and ensure a high quality of the service.
- **Cinema Hall Configuration** retrieve the cinema hall configurations together with the list of all shows or retrieve it only for the selected show;
- **Selecting Seats for a Show** make a request to SCS for each selected seats or select all seats and then make the reservation request to SCS.
- **Sending different data types between the SCS and MIDlet** sending different data types between the SCS and MIDlet in one single connection; overcome computation power issues when parsing Strings on the MIDlet

2.2 Solutions to the Identified Issues

This section formulates and analysis different solutions to the issues identified in section 2.1.

Mobile Device Limitations:
The solution of the Cinema Ticket Reservation System need to overcome all previous mentioned hardware limitations. This can be achieved by a good application and communication protocols design, use of lightweight libraries, optimizing the packaging process (e.g. including only those parts of the libraries the application uses, and taking advantage of the obfuscation process to replace class names and long variable names with shorter ones).

Due to the limited CPU power and short battery life, most of the operations have to be processed on the server side and the MIDlet used only for displaying the results. One can also overcome these issue by using OOP[1] best practices such as: minimizing object creation and disposing any unused objects, using design patterns, reusing objects rather then creating new ones, closing streams, network connections, and the record management system after use, opening the record management system once per application instance, etc.

The small display issue can be overcome by using a *One screen at a Time* approach i.e. long operations need to be split into small pieces.[23]

The limited input capabilities can be solved by reducing the amount of information user has to key in and provide different graphical components e.g. combo boxes that allow user to choose among several options instead of typing the required information. By keeping the users in mind and involving them in all steps of UI design, one can archive a well designed GUI and overcome all previous mentioned issues.

MIDlet:
Rich UI can be developed by taking advantage of both low level and high level components in CLDC 1.1/MIDP 2.0.

One must prevent the UI lock-up during network operations by using background threads for that. An animated gauge can be displayed to keep the user informed on the operation status at all time. A cancel button should be present in case the user would like to cancel the operation at any time.

The "One screen at a Time"[23] approach has to be taken into consideration

[1] Object Oriented Programming

due to the small display of the device.

By involving real users in the design and evaluation phases of the application, one can achieve a user friendly software product.

User Centered Design of the Prototype:
The application must have a user-centered approach i.e. the user domain, users and other stakeholders have to be defined. A design framework is chosen. Developers and users brainstorm on the given subject. Similar application are investigated and analyzed. Once that step is finished, initial requirement specifications are formulated. Different low and high fidelity prototypes are developed and evaluated by real users before the final design is decided. Interviews and workshops are conducted and the results used in the next step of the iterative development process. The overall goals of the evaluation, the questionnaires to be answered by users, guidelines for conducting the interview, and evaluation paradigms are defined and analyzed. Practical and ethical issues are dealt with. The results of the interviews and workshops are evaluated, interpreted and presented. These results are to be fed back into the design process of the application. Several alternative circles of design-evaluate-redesign are conducted. The number of required design circles is analyzed. Final requirement specifications are stated and a final prototype description is made. That can include a low or high fidelity prototype.

Location-aware application:
This is a location - aware application that can determine user's current position and display a list of all cinemas and movies in a certain range from that position. If the user selects a movie that he/she would like to watch, a request is made to the application OTA[2] and a list of all cinemas displaying that movie is shown on the user's mobile device.

A first solution involves an external GPS device that can be connected to the mobile device or a built in GPS chip to determine user's current position. By using a specialized API(JSR-179)[6] the MIDlet can communicate with the GPS device and retrieve user's current location. Based on the GPS data and application setup, all cinemas in the user's range can be found out. This solution is suitable only if the user has an external GPS device to connect it to his/her mobile phone. Therefore is not appropriate for the *targeted user group i.e. all cinema movie goers.*

A second solution involves a mobile device capable of using the *Location Acquisition API* and *JSR-179*. Location Acquisition API is included in S60 SDK from S60 2.6 (2.nd edition FP2) onwards. Moreover, the server side service used

[2]Over The Air

by the Location Acquisition API has to be provided by the network operator. There are not that many operators to provide this service and the cost is very high. (At this point the author is aware of a similar service provided by Vodafone UK and another operator in US). Thus, this solution is not suitable either for the application targeted user group.

A third solution involves the use of trilateration and it can be used by phones without GPS features. Trilateration is based on the signal-strength of the closest cell-phone towers. A network location service has to be provided by the network operator together with an API for validating and determining user's current position. Therefore, this solution, as the previous one is not realistic.

A forth solution consists in using third parties API's and services to determine user's current location. All these services are not free of charge, therefore a financial solution needs to be found out. The author has applied for a student license for such a service. Unfortunately, the *Company* providing that service has not agree on that. Due to the high financial implications e.g. one license for each mobile phone, this is not considered a desirable approach.

The last solution approaches this issue from another angle i.e. user can key in his/her current position (street, city, zip). Based on that the SCS can find all movies(cinemas) in the given range from the user's position and display them on the user's mobile. Then, the user can select the movie he/she would like to see. Several user friendly extensions can be implemented to overcome the limited input capabilities issues e.g. once a city is selected all streets can be displayed in a combo box; based on the given zip code, the city name is filled in automatically, and the other way around. *This is the solution chosen for the Location-aware issue.*

On-device data storing:
Mobile devices do not have a conventional file system for storying user's data. A possible solutions to this issue is to keep user's information stored in instance variables and make it available only to the current invocation of the application. When the user quits the application all previous saved data is lost.

A better approach need to be considered in case information has to be persistent from one invocation to another. The solution is to use the *Record Management System (RMS)* that gives MIDP applications local, on-device data persistence.[4] RMS can contain several *Record Stores*. A record store is a collection of records, each of them with a *unique ID(a key)*. The data to be stored is the application record store in a *Record* i.e. a *key-value pair*. The key is a long number from zero up, while the value can contain anything that a sequence of bytes can represent.[4] Each record store is uniquely identified for an application and is configured to be accessed only by the application who created.

From a security point of view, a stolen device containing sensitive data, user keys or credentials can pose a security risk to the whole system. Use of strong encryption with RMS is a must to protect the data.

The second solution i.e. RMS coupled with strong encryption is considered the final approach to the on-device data storing issue.

Secure Wallet:
A Secure Wallet feature must be implemented for storying user's credit cards and allow easy and secure ticket payment OTA.

A first approach consist in using RMS for storying data protected by a PIN code or a user name - password authentication. This solution is not considered secure enough because there are ways to extract the desired information from the RMS e.g. memory inspection under a microscope.

A second solution consist in using RMS coupled with strong encryption and PIN code-based authentication. This approach is not secure enough because a evildoer that gains access to a device containing credit card information can run dictionary attacks and find out the PIN code to the Secure Wallet.

A third solution uses RMS coupled with strong encryption for storying user's credit card data, PIN code-based authentication, and *PIN and credit card data safe reset triggers*. i.e. if the PIN code is entered wrong 3 times between 2 consecutive correct accesses to the wallet, the PIN code and all credit card data are reset. The number of times the PIN code is typed wrong can be saved in RMS and made available to every instance of the application. If the Secure Wallet is opened with the right PIN code, the number of PIN trials is reset to zero. This is considered the solution to the Secure Wallet issue.

MIDlet Configuration:
The Cinema MIDlet configuration can be done both by the user and server side service. The server side service can send a configuration message to the MIDlet containing all cinemas and movies in a city for one week. Thus, user can search offline for different movies. However, this can be a limited storage capability issues for some mobile devices. Therefore, this solution needs to be further analyzed during the implementation.

User can also configure the MIDlet to save his/her credentials and authenticate automatically when the application is opened. This solution can pose security risks to the application and user's sensitive data because an evildoer can impersonate as being the user and order tickets in his place. Therefore, this is not considered as a possible use case.

Server-side Data Storage:
Several approaches can be used in this case.
A first solution consists in using the file system(text, binary or .xml files) for storing the Cinema Service data. There are several disadvantages to this solution i.e. high access time to a file, complicated and slow mechanisms to search for data in a file, data manipulation (insert, update) implies that the file(s) have to be locked with an exclusive lock. This is not a suitable approach.

A better approach consists in using a *Database Management System(DBMS)*. A *relational DB* can be used for storying data on the server side. The access to the data is fast and based on indexes. Search and update operations are fast. Concurrent accesses are supported based on thre database driver implementation. A DBMS supports **isolation** i.e. transaction execute one at a time; **atomicity** i.e. transaction execute either completely or not; and **durability** the ability to recover from failures or errors of many types.[3] A DBMS provides a powerful API for accessing the data. A powerful query language is available to the developer. A file system has no such advantages. A DBMS is more powerful than a file system and supports flexible access to large amount of data. A relational database management system is chosen as the final solution for this issue. There are also OODB[3] that can be used for this purpose, but a RDBMS[4] is considered as the desired approach. OODB are not very common and they offer limited support.

Credit card data are not to be stored on the Cinema DB due to security considerations. A trusted third party service is to be used for credit card transactions.

MIDlet - Server-side Cinema Service Communication:
The communication between the mobile device and the SCS can be realized by using either *Socket* or *HTTP connections*.

Socket connections OTA involve that mobile devices that can establish a socket connections over carriers e.g. GPRS. This implies a devices supporting socket connections and an agreement with a GSM network provider in order to allow socket connections for the device. The later one is very difficult to achieve. Moreover, the solution should work on most of the devices and not only on those that support sockets.

The second approach, HTTP connections, can be used with all GPRS enabled mobiles without any need of extra configuration. The Server-side can be implemented using HTTP Servlets. This is the chosen solution for the communication between the MIDlet and the SCS.

[3]Object Oriented Databases
[4]Relational Database Management System

Errors can occur both on the server and client side. The user has to be informed by means of user-friendly messages that can help him recover from the error and deal with it in the most suitable way. Ticket payments have to be implemented as transaction for integrity reasons. All sensitive data has to be encrypted and only authenticated users can be allowed to access the system.

Communication costs can be kept at minimum by reducing the amount of information sent over GPRS. Also, the errors and messages sent from the server side to the MIDlet must include only an error code. The message is generated on the mobile device based on the received error code. There is a need of synchronization between the error codes on both ends.

A secure communication can be achieved by using authentication and encrypted data sent OTA. More details are depicted in Section 3.

The chosen solution to the MIDlet - SCS communication consists in using a GPRS carrier to send HTTP requests to several HTTP Servlets. An authentication service is implemented. In case user is authenticated, the SCS performs user's requests. Sensitive data must always be sent encrypted and both the server and client must have the means to decrypt the data based on the chosen authentication mechanism. Ticket payments are implemented as highly secure transactions. No credit card data is to be saved on the server side due to security reasons. A status code is sent from the server to the client side i.e. the status of the server side operation. The message to be displayed to the user is to be generated on the MIDlet based on the received status code.

SCS - DB Communication: *String connections* can be used to accessed the system DB. These are very expensive to use from the data access time point of view. Every time a request is made, a new connection to the DB is created, and the DB is queried for the desired data. Once the result is returned to the user, the DB connection is closed. This is a very expensive process due to the high time needed to establish a new DB connection for every new request. This is not the recommended approach for establishing a connection to the DB.

A second approach consists in using *connection pools* i.e. the servlet container[5] can provide the means for declaring resources that can be used to retrieved connections from a previously created connection pool. Therefore, every time a request is made, the connection pool is checked for any idle connections. If there are idle connections in the pool, a connection is retrieved from the pool and use to query the DB. When the result is returned to the client, the connection is returned back to the pool. Thus, the next request can take advantage of it. If there are no idle connections in the pool, the user can wait for a connection to

[5]In this case Tomcat is used as a web server for storying all servlets

become idle. This is the most optimum way of connecting and querying a DB due to the minimum amount of time spent in acquiring a new connection for every new request. In this case the connection are already created and the user can take advantage of that. This is the preferred solution for this issue. A high quality of the service can also be ensured.

Security Concerns:
A special section is reserved for dealing with security concerns in case of the Cinema Ticket Reservation System. More details about this topic are depicted in section 3.

Slow and Unreliable Networks:
The solution to this issue consists in using the on-device data techniques by means of RMS to avoid slow connections for obtaining user credentials, session keys or other configuration parameters. Reading/Writing network data need to be done using a buffering mechanism because reading/writing data byte by byte is very slow.[23] To ensure a high quality-of-service a RDBMS is to be used as storage solution. Connection pools are to improve the access the DB. A well known trusted third party service is to deal with all credit card transactions. User's trust can increase, and the integrity, reliability and security of the transactions are preserved.

2.3 Final Proposed Solution

Based on the previous formulated requirements and solutions depicted in sections 2.1 and 2.2, the final solution to the GPRS-Based Cinema Ticket Reservation System is stated.

A 3-Tier solution is to be implemented i.e. **Tier 1 - the MIDlet**, **Tier 2 - the Server-side Cinema Service**, and **Tier 3 - the Database**. Several third party services are used i.e. credit card transaction validation and user localization services.

A user-centered approach is considered for the design of the prototype. User interviews and workshops are conducted with real users. The results are used in the application design. Several alternative circles of design-evaluate-redesign are conducted.

Tier 1 - the MIDlet is built for CLDC 1.1/MIDP 2.0 enabled devices. The code follows all previous mentioned constraints. Design patterns, obfuscation, lightweight libraries, limited computations on the mobile device, minimizing object creation, closing network connections and record management system after use, opening the record management system once per application instance, using the *One screen at a Time* approach, and involving users in the design process of the application work flow and UI are considered when designing and implementing Tier 1. Data such as user credentials, keys, configuration parameters, etc are stored in the RMS. Sensitive data stored in RMS are encrypted. Rich UI are built. Background threads are used to prevent UI lockup during network operations. An animated gauge is displayed to keep the users informed on the operation status. All sensitive data sent OTA are encrypted according to the chosen security mechanism. Authentication is a must.

The **location-aware** solution allows user to key in his/her current position (street, city, zip). Based on this data the server side cinema service finds all movies(cinemas) in the given range from the user's position and displays them on the mobile device.

The **Secure Wallet** feature is implemented using RMS, strong encryption, PIN code authentication, and safe triggers that reset the PIN code and credit card data if the PIN is entered wrong 3 times. Thus, one can protect the content of the Secure Wallet against an evildoer who might have access to the device. Also, by reseting the secure wallet and PIN code, the user is ensured that he/she can use the secure wallet even if the PIN code is forgotten.

Tier 2 - the Server-side Cinema Service - the communication between Tier

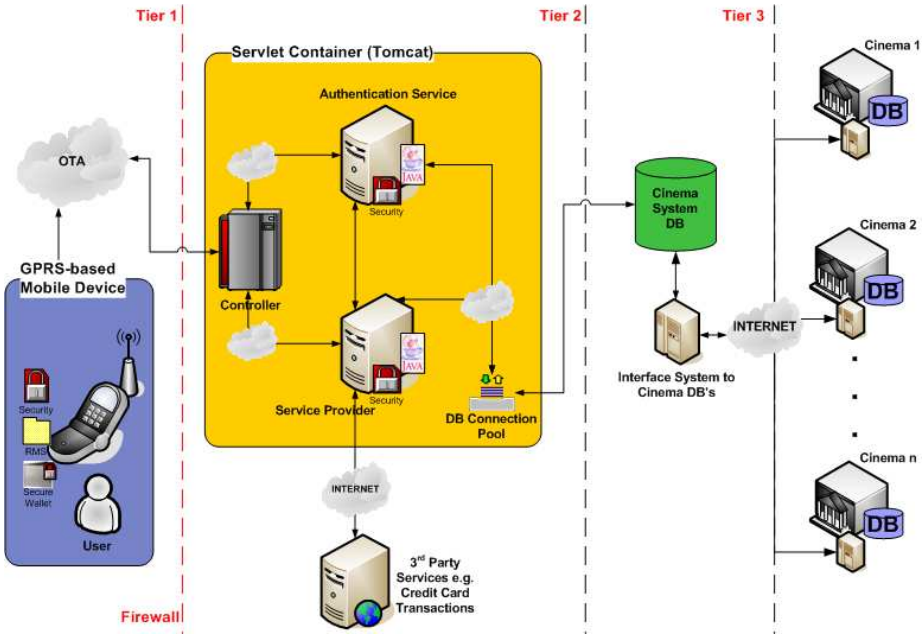


Figure 2.1: GPRS-Based Cinema Ticket Reservation System Diagram

1 and 2 is realized by means of HTTP connections. Java Servlets are used for this purpose.

An authentication mechanism is designed and implemented to allow secure transactions. Sensitive data is always sent encrypted and both ends have means to decrypt the data. Ticket payments are implemented as transactions and sensitive data encrypted. Only authenticated users can perform these operations. No credit card data is saved on the server side due to security reasons.

An error(status) code is sent from the server to the client side in every network response. The message to be displayed to the user is to be generated on the MIDlet based on the status code value to reduce communication costs.

Communication between Tier 2 and 3 is ensured by means of reusable DB connection pools. This provides a high quality of the service.

Tier 3 - the Server Side Data Storage
A relational database management system is used for storying the data. Credit card data are not saved on the DB due to security considerations. A trusted third party is used for credit card transactions.

The architecture of the chosen solution is depicted in figure 1.

The security considerations of the final solution are depicted in Section 3.

Chapter 3

Securing the Cinema Ticket Reservation System

Statement from the author: Some parts of the design and implementation of the security protocol has been reused from a previous project in Secure Mobile Services (course no. 34632). Two persons worked in equal amounts on that project during the Secure Mobile Services course. The author was one of them. The code fragments reused or adapted from that project are strictly marked with the name of both authors.

Security is divided into the security of the communication, protection against piracy, and the physical security of the mobile device. These are analyzed in the following chapters. Solutions to the traditional security concerns are also stated:

Data Integrity assures that identically the same piece of data is sent and received i.e. the correctness of the data being transmitted is not to be influenced by the underlying network operations and the data transmitted through the network cannot be changed by eavesdroppers in a meaningful way.

Confidentiality assures that only accredited parties are able to understand the data.[19]

Availability assures that only accredited subjects have an access to the resources. Two aspects of the availability should be considered:[19]

- Unauthorized subjects should not be granted an access to the resource.
- Authorized subjects should be granted an access in any case. Their access may not be prevented by any technical problems nor by malicious activities.

Non-repudiation assures that it may be proved the information sent by the user was originated by him and not by any object or subject in the system.[19]

3.1 Authentication Mechanism

An adequate authentication mechanism must be based on the previous stated security requirements. The chosen authentication mechanism involves:

- **Something that user possesses** - The characteristics of the system being designed do not allow for strong, physical authentication. Biometrical devices are not available yet for the most of todays mobile terminals. A key i.e. a piece of data can be used instead. The key must be preshared by the communicating parties. Experience shows that presharing of the binary cryptographic keys is too troublesome from the user point of view. Technically it requires additional protocols for secure exchange of the secret. Human being is able to memorize strings up to 12 digits long. Therefore the secure key would be a string much longer than this.[19]
- **Something that user knows** - PIN code, user name, and password.

An authentication mechanism based on the previous two factors is accepted for the application.

3.2 Protecting the Data Sent Over the Air

In order to preserve the data confidentiality and integrity, all sensitive communication between the mobile device and the server must be encrypted. It must be assumed that the communication may be intercepted at any point between the mobile user and the targeted service by any eavesdroppers i.e. single people or whole organizations that can be in possession of unlimited resources.

Therefore the encryption standards under consideration should sustain any known technological treatment. Sufficient strength is provided by the following standards:

1. **RSA** - asymmetric cryptography standard; Acceptable key length is 1024 bit, suggested 2048 bit.
2. **3DES** - symmetric cryptography standard; Acceptable key length is 54 bit, suggested 128 bit.
3. **AES** - symmetric cryptography standard 128 bit long key, provides the ideal security.

The characteristics of the mobile system constrains the usage of the cryptographical standards even more. The asymmetric cryptography requires a considerable amount of computation. Even in the case of very small messages sent from the mobile device, the delay could become unacceptable for the user. Additionally, the size of the keys used by asymmetric ciphers are too large for the mobile devices.

For the reasons explained above, only symmetric ciphers are used in this application. More details can be found in section 5.4. A possible security level is been reached by using only the symmetric ciphers.

3.3 Securing User's Private Data

It must be assumed that evildoers can get access to user's mobile device at any time. Therefore the access to user's sensitive data e.g. credit card information must be protected. An authentication system based on user name, password or PIN code allows trusted parties to access user's data. Moreover, sensitive data stored in the mobile device RMS must be encrypted using symmetric ciphers as mentioned in 3.2. This offers an extra level of security against evildoers with unlimited resources. On top of the symmetric encryption, safe triggers can be used for erasing user's sensitive data and PIN codes in case the user name - password or PIN code is entered wrong more then 3 times. This can offer protection against any dictionary attacks.

User's credit card information must not be stored in the system DB due to security reasons. There are third party service providers e.g. PAY PAL, etc. that can validate and execute credit card transactions. It is safer to use such a trusted 3rd party service instead of *reinventing the wheel*. Any communication inside the Tier 2 and between Tier 1 and 2 must be encrypted. Passwords are not to be stored in blank in the DB. Hashes of user names and passwords can be used instead. When an authentication request comes from the user, a hash of the given user name and password is computed and checked against the one stored in the DB. This is a better approach then storying encrypted password in the DB because of the high time needed to decrypt the password saved in the DB.

3.4 MIDlet Protection Against Piracy

The intellectual property represented by the mobile software, must be protected against reverse engineering and piracy. These threats can be mitigated by obfuscation of the source packages. The obfuscation process Obfuscation removes context from compiled code that humans (and reverse-engineering tools) would use to decipher the code's meaning. The trick is to remove this context from evil intentions while retaining complete execution integrity with the original program.[10] E.g. variable, methods and class names are replaced by shorter ones and without any human meaning, spaces are removed that the whole code appears as a very long sentence, etc.

Chapter 4

User-Centered Design of the Prototype

Human-computer interaction design based is a crucial aspect of any application made to be used by humans. It is an important practice to study and understand before an engineer enters the real world. Before a product is created several low and high fidelity prototypes are built. Market perspectives are to be analyzed before building the prototype. How will consumers react? Will it be easy, pleasing or efficient to use? Will it be attractive or appealing enough to be picked up in the first place? These are all very important questions and no matter how much one speculates how a product will do on the market, testing and user-evaluations are the closest and most effective way to be prepared for the delivery of a product.

Based on the previously stated scenario and application requirements, a high fidelity prototype of the Cinema Ticket Reservation System is created. Several concepts are used during the prototyping phase of the application such as: *brainstorm, conceptualize, prototype, workshops, interviews, and test.* Brainstorming is used in the incipient phase of the low fidelity prototype design. Developers and casual movie goers search similar application. They brainstorm on the given topics and try to find the pros and cons for each of them. Improvement ideas are always welcomed. Based on the brainstorming workshop, a first low fidelity prototype is created. This prototype is used in *3 repetitive*

cycles of design-evaluate-redesign. Users are presented with alternative designs of the cinema system application. All these designs are evaluated by real movie goers. The results of the user interviews are used to create the first high fidelity prototype during several design-evaluate-redesign phases. A final version of the prototype is then obtained. Detailed testing procedure are used to extract as much information from users as possible. The evaluation workshops and interviews performed by real movie goers on the Cinema Ticket Reservation System involve among other:

- evaluating, selecting, or redefining the right information to be displayed on the mobile device
- evaluating, selecting, or redesigning the right GUI layout and application flow to achieve application usability and user experience goals. 4.1.1
- evaluating or redefining command accessibility e.g. menu options, soft buttons, menu items or soft button priorities.
- measuring the time and number of errors while a movie goer achieved while performing a given task(scenario)

4.1 The Design Process

This section depicts the design process of the high and low fidelity prototypes of the Cinema Ticket Reservation Systems. Different steps from understanding the problem space until the final solution to the high fidelity prototype are analyzed. A user-centered approach is used as an interaction model because the Cinema Ticket Reservation System is an application addressed to different categories of stakeholders. Thus, it is important to involve them in all phases of the design process.

The following steps are used in the design process of the Cinema Ticket Reservation System prototype.

- Design Model Selection
- Defining the problem space. Identify needs and establish requirements
- Developing alternative designs. The Conceptual and Physical Design
- Interactive high fidelity prototyping
- Choosing the techniques for evaluation the design

The Design Model Selection
A model that focuses on users and allows any number of iterations for building a final product as close as possible to the user expectations and usability criteria [18] is chosen. This is the *Simple Lifecycle Model*. The graphical illustration of this model is depicted in fig. 4.1. Several cycles of design-evaluate-redesign are used in this model until a final product is obtained. The design process begins by identifying needs and establishing application requirements. A first version of the prototype is created and evaluated. After the evaluation, the requirements are refined and a new cycle of (re)design - evaluate takes place. This happens until the final version of the product is obtained.

Define the problem space. Identify needs and establish requirements
This is very important to understand the target users if a user-centered approach is considered.[18]. A brainstorming workshop is organized for this purpose. Before the actual workshop, designers search on the Internet for similar products, and analyze creatively the similar products. Pros and cons are depicted. Improvement ideas are welcomed. Different types of questions are used in understanding the problem space e.g. Is such an application useful to people? When it is useful and how? Can it help users in their every day life and extend the current way they are doing things? Are there any similar products? Are

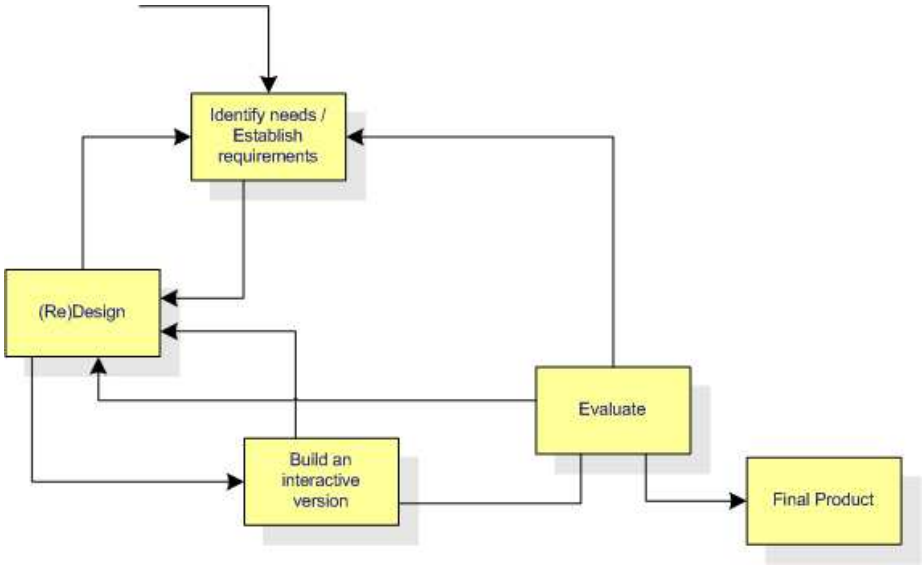


Figure 4.1: The simple lifecycle model[18]

there any problems with similar products? Which are the problems? How can one set this product into the every day use? etc.

After the individual brainstorming process the designers meet and share their findings among each others. A brainstorming workshop that involves real movie goers is prepared. The necessary logistics such as sheets of paper, post-its, colored pencils, markers, recording hardware e.g. camera, voice recorder, etc have to be available. A set of questions are also prepared for interviewing users. The application requirements represent the output of this step. Usability and user experience goals are formulated.[18]

Developing alternative designs
This is divided into two parts i.e. *conceptual design* and *physical design*.

The Conceptual Design defines the conceptual model of the Cinema Ticket Reservation System. It defines what the application should do, how it should look like, and what kind of interaction one can have.[18] Storyboarding, sketching ideas of the application, describing scenarios, and paper-based low-fidelity prototype are used for this purpose. As a result of of the conceptual design the following characteristics are obtained for the conceptual model:

- The conceptual model is based on activities such as exploring, browsing

(e.g. movies, cinemas), and instructing (e.g. searching for a movie)

- The GUI interaction paradigm is chosen because the application must provide a use friendly application, easy to use and powerful enough even for such a small screen.

- The conceptual model is based on several interface metaphors e.g. cinema, credit card, wallet, and ticket. Several of these metaphors have been identified during the first workshop. One can argue about the pros and cons of using metaphors in an application.[16]

- The **DECIDE framework** is considered for involving users in the design process and for evaluating different versions of the prototype.

New and better design ideas and requirements are obtained from the conceptual model e.g. UI components, icons, cinema theater layout, navigation mechanism and application flow, limited keyboard input issues, small screen device issues, help features, etc. All these requirements are used further on in building a user-centered low fidelity prototype.

The Physical Design defines the graphical details of the product(colors, layout, menus, images, icons, fonts, etc.) and the interaction with the application. Three iterations are used.

The first iteration is done during a workshop involving users that are colleagues of the designer. The second one is performed with real movie goers. Storyboarding, sketching ideas of the application, and describing scenarios are used for this purpose. A first paper - based prototype is considered. Different scenarios are used for users to interact with the application. Several problems are identified in this case. The application requirements are redefined based on the previous findings.

The second iteration involves real movie goers i.e. friends, colleagues, relatives of the designers, and even casual users that have never been in contact with the designers. The paper - based version of the prototype is redesign. New storyboarding ideas, scenarios, and alternative designs are presented during the workshop.

The third iteration involves real movie goers and another version of the paper based prototype. The GUI is very close of the one considered for a high fidelity prototype. This iteration reveals several problems concerning navigation, menus, cinema theater layout, and search functions. Several changes are performed such as:

- the black list of people is removed
- automatic cancellation of unpaid tickets with 30 min before the show is introduced as a feature in the system
- no history of the used tickets are kept in the mobile device for later use
- the wallet must not be blocked if the pin code is entered wrong more than three times

Interactive high fidelity prototyping
A functional interactive high fidelity prototype is created based on the results obtained from the previous workshops. Two user evaluations are used for obtaining the final version of the prototype. Several other iterations are performed in between the user evaluations by the programmer(the author in this case). Real movie goers are involved in the workshops for evaluating the prototypes. New requirements are obtained after the first iteration and several changes in the UI and functionality are performed such as:

- introduce a feature to save the user name and password values
- automatic login with the server side when the application is started

As mentioned in section 2.2 these new requirements may pose a security risk to the whole system and user's sensitive data. Therefore, these are evaluated later on during the implementation phase.

Choosing the techniques for evaluating the overall design
The evaluation process is based on the *DECIDE framework* and focuses on the user centered-design by using several design-evaluate-redesign circles. Based on the DECIDE framework, the steps defined in 4.1.1 are used during the evaluation process.

4.1.1 The DECIDE Framework

1. **Overall goals of the evaluation**
    - Let the users participate in the design process.
    - Get to know any issues users might have with different design solutions.
    - Specify new requirements for the project.
    - Check if the users' needs are understood.
    - Check if the application fulfills the *usability goals* i.e. effective, efficient, safe, easy to use, easy to learn, and easy to remember.
    - Check if the application fulfills the *user experience goals* i.e. fun, entertaining, satisfying, helpful, motivating, aesthetic, supportive or creative, rewarding, and emotional fulfilling.
    - Check to ensure the application is user friendly.
    - Check to ensure the application is easy to use, easy to navigate from one screen to another, and the feedback provided to users is clear enough and gives sufficient information.
    - Check to ensure the data provided to the users is clear enough, sufficient and it fulfills users' requirements.
    - Check if the application is consistent, and has a minimalist design i.e. avoids using information that is irrelevant or rarely needed.
    - Check to see if there is a match between the system and the real world.
2. **Questions to be answered by users**
    - **Pre-evaluation questionnaire** - use of both closed (predetermined range of answers) and open questionnaires.
    - **Post-evaluation questionnaire** - use of both closed (predetermined range of answers, Likert Scale) and open questionnaires.
    - **User interviews**
3. **Evaluation paradigms and techniques** - quick and dirty (observing users, asking users), field studies.
4. **Practical issues**
    - **Users** - 4 males and 4 females of different ages, backgrounds, experiences in using mobile phones, online stores, mobile commerce, mobile technologies; users go through the propose scenarios and redesign them by using post-its, if necessary. They can add their own ideas to the layouts that way.

- **Facilities and equipment** - printed forms (every user has its own copy; separate sheets for pre and post-evaluation), post-its (to let users add their own ideas), colored pencils, low-fidelity prototype (paper mock-up) based on scenarios of different tasks a user should fulfill when using the application.

5. **Ethical issues** - every questionnaire has a disclaimer placed at the bottom of the last page: *I hereby state that I am over 18 and I wish to participate in a program of research being conducted by ...*

6. **Evaluate, interpret, present the data and used it further on in a new version of the prototype** - heuristic evaluation is to be used. Trends and patterns are to be identified. Data is to be displayed graphically and perceptual for the close questions.

Several guidelines are used during the conceptual design workshop. More details can be found in *Appendix A*.

- Inform users about the study goal, the tasks to fulfill, the amount of time needed for this study, the data that is collected, and how the data is to be used.
- Give users an overall description of the application
- Inform users about the goal of the application they are to evaluate
- Inform users about any the ethical issues
- Inform users about the data gathering mechanisms used in this study e.g. observing users and conducting user interviews.

4.2 User Domain, Users and Other Stakeholders

User Domain
The Cinema Ticket Reservation System can be used by any movie goers to search movies and reserve cinema tickets. Everything that is needed is a mobile device that supports GPRS connection and CLDC1.1/MIDP2.0. Thus, the application target is the casual movie goer without any age limit that is not afraid to use new technologies and purchase products online. The targeted user can also be a tourist visiting a city. Once arrived at the tourism information office he/she receives the application on his/her mobile phone via several bluetooth antennas located inside the office. The targeted user can also be any person that is walking in the city and decides to watch a movie in a cinema; but it is inconvenient for her/him to go to the cinema and buy tickets. He/She can use his/her mobile device to search for movies and purchase tickets.

Users and Other Stakeholders
Several people are involved during the normal flow of the design and development processes. It is important to know the organizations involved in the development, production and testing to avoid any problem that could occur during these phases.

Several stakeholders are enumerated hereinafter:

- **the client** - the person who pays for the product
- **the customer** - the movie goers that are to use the product
- **project leader and manager, designers, developers, testers, certification organizations, competitors**
- **others**

Selecting users for the interviews
When choosing users for the interviews one has to focus on the application target group. As the prototype target group are movie goers with a mobile device, all kinds of users are needed. People from the following groups are chosen:

- **Average people** - People with little or no background in computer science. But they are used with a mobile phone - mostly for phoning and text messaging.
- **Experienced people** - People with some experience in computer science. It is important to find out, which features they are interested in, as they are in the targeted group, as well.

- **Both sexes** - It is important that both girls, boys, men and women are able to use the application as all of these form the movie goers group.
- **Different ages** - Movie goers are persons of different ages

4.3 Design Ideas and Data Gathering Mechanisms

The purpose of the Cinema Ticket Reservation System is to allow movie goers to search for movies and reserve movie tickets even when they are not in front of a computer. Movie goers can be tourist just arrived in a new city. They can receive this application for free from the tourism information office. They can also be represented local people who decide to see a movie while they are shopping or having lunch in the city. They open the application on the mobile phone, search for a movie and book the tickets. The product is supposed to have the same functionality as any web based applications for cinema ticket reservations. User can pay for the tickets via mobile phone or once arrived at the cinema. Moreover, a social network of all movie goers is proposed to be created by exchanging impressions about movies. As mentioned later on in the report, the users do not like the idea of writing movie reviews due to the limited input capabilities of todays mobile device. Therefore, this feature is not considered for implementation.

Design ideas are collected by brainstorming on this topic, searching on the Internet for similar application, exploring them, talking to people involved in the cinema industry and with real movie goers. Based on the results, a first sketch of the prototype and the corresponding scenarios are created. Several ideas are redefined and a low-fidelity prototype is built. The prototype is no more then a set of GUI drawings on small cards and arranged in a particular order on a piece of paper to emulate the real application work flow. This prototype is used to conduct a first workshop that involves real movie goers. They are involved in the design process from the very beginning i.e. arranging the layout of the application screens and the flow among them, the information displayed on the screen, etc. They are encouraged to come with new and constructive ideas by *talking loud*. Alternative prototypes and scenarios are presented to the users. Scenarios based on cards are also displayed during the workshop for the evaluation of the application work flow. *Pre-evaluation*(before users could see the prototype) and *post-evaluation*(after users have seen the prototype) questionnaires are used to gather data from the users during and after the conceptual design workshop. This workshop provides useful information about they GUI layout, application flow, information displayed on the screen, useful and useless features, and new application requirements. Based on the feedback received from the users several changes are made to the prototype as one can see in section 4.5. The fake information had to be separated from the real one. The low-fidelity prototype, the scenarios, the questionnaires, and the results of the workshops are displayed in *Appendix B*.

The final version of the low-fidelity prototype is created based on the feedback received from the users durinf the conceptual design workshop. This prototype is also evaluated with different users using short questionnaires and the *think aloud technique*[1].

The *first high-fidelity prototype* is built once the final version of the low-fidelity one is agreed. A *physical design workshop* is conducted. The application is running on a laptop and it is evaluated by one user at a time for a better quality of the workshop output data. The think aloud technique and user interviews are chosen for the prototype evaluation. Users are asked to perform different scenarios using the prototype e.g. search for a movie, rate a movie, add/delete a credit card, purchase tickets using different payment methods, etc. The feedback obtained from the users is analyzed and used later on in the final electronic version of this prototype.

The changes performed to the prototype include:

- several updates of the GUI layout and message windows - the text is aligned left aligned or centered. The size of the displayed icons is reduced. Some soft buttons used e.g. in the credit card view screen are updated, etc.[1]
- the content of several Options menus are rearranged based on the most likely option to be selected as the first entry, followed by the second most likely option, etc.
- the ticket reservation flow is simplified
- the users complains about the speed necessary to open the wallet so the wallet initialization is to be changed.
- the progress gauge is customized and includes also textual messages informing the user about the current operation either over the network or not.

[1] users are asked to express aloud their opinions about the prototype

4.4 Initial Requirement Specifications

Identifying needs and establishing requirements is a crucial part of any design process.[18]. One has to understand the users as much as possible, and involve them in the development process of any product made for the users. One must not forget that IT projects failure mainly come from unclear objectives and requirements. The application requirements before the conducted workshops are presented in here. The refined application requirements after the workshops have been conducted with real users are depicted in section 4.5.

Functional Requirements

- The application provides a user name and password authentication mechanism. Authentication is done with the server side and a token received and used per session.

- The application allows users to find movies in a certain range from a given position. Users can enter a street name, city or zip, a range and a date for which they would like to search for movies.

- The application displays a list of all movies in that area, the cinemas where the movies are played and the hours for all shows. Users can seats for a particular show(movie, date, and hour). User can also see information about the selected cinema i.e. cinema name, distance and map to that cinema from the given position(section 9).

- Users can reserve one or more seats for a movie. The cinema hall configuration is displayed by using a color code map i.e. red - booked seats, green - free seats, black - unavailable seats, blue - user's current selection. Users can navigate from one seat to another by using a yellow cursor. Users can select a seat by pressing the OK button on the phone or the select button situated between the arrow keys of the device.

- The application allows users to enter discount information about the chosen tickets before purchasing them e.g. student, pensioner, VIP discount. Users can enter the chosen discount type and the student card/pensioner card/coupon number.

- The application displays a reservation summary together with the total amount to be payed and prompts users to accept the ticket payment or not. The payment is done in a highly secure way. No credit card information is saved on the server side.

- The application provides several payment methods i.e. paying at the cinema, by a new credit card, or by previous saved credit cards in the phone memory.
- The application displays the billing info after the payment is done. It displays the payed total amount, and an info message. The message tells users to keep their ticket ID's to get access to the movie, and bring any discount card or coupon they might have used to obtain discount tickets.
- The application saves a black list of people. This list is used to store people who do not cancel a previous made reservation in case they cannot attend a show and the ticket has not been payed out. If people do that for three times, the application is locked and they cannot use it anymore unless they pay for the previous unused tickets.
- Users can cancel a previous made reservation(1 or many tickets) even if the tickets are payed or not. If the tickets are payed, users can get their money refunded.
- The application saves, up to 10 tickets. These tickets have not been used before. The ticket name contains the Ticket ID.
- The application keeps up to 5 used tickets. User can delete any used tickets manually.
- Once a user selects a movie he/she can view details about that movie. The movie details includes the movie poster, too.
- Users can read/write reviews about a selected movie. The review title contains the movie, short review description and the movie rating score.
- Users can watch trailers for a selected movie.
- The application provides a feature for storing users' credit cards on the mobile phone in a highly secure way. This feature is called *Secure Wallet*.
- The application provides a PIN code based authentication method in order to access the content of the Secure Wallet. If the PIN code is entered wrong three times the secure wallet is locked and the users cannot access it anymore.
- If the authentication procedure for the Secure Wallet is successful, users can add/delete/view credit cards.
- The applition stores the following information about users' credit cards: *Credit Card Nickname*, *Credit Card No.*, *Owner*, *Expiration Date*, *Security Code*.

- Users can identify the saved credit cards in the phone memory by using the *Credit Card Nickname*.
- The application main menu contains the following entries: *Search for a Movie*, *Manage Not Used Tickets*, *Manage Used Tickets*, *Manage My Wallet*, *Help*, and *EXIT*.
- The application provides users with *Help* information on each screen. This information helps users in solving any misunderstandings or recovering from any possible errors.
- The *Message Info Screens* provide users with enough information about a specific action or request.
- An *EXIT* option is placed on almost every screen to leave the application.

Data Requirements
The application must have access to all cinemas' DBs to search for movies and reserve tickets. A highly secure 3rd party payment service such as Pay-Pall is used for all credit card transactions.

Environmental Requirements
The environmental requirements are not really specific. The Cinema Ticket Reservation System has to be available to everyone with a mobile phone that supports GPRS connections and CLDC1.1/MIDP 2.0. It has to be fast and reliable. The GUI is simple and not overwhelming due to the hardware limitations of mobile devices.

User Requirements
The product is designed for a high number of people, even unexperienced Internet shoppers. Every movie goer that can use his/her mobile phone to play a game, send messages, or browse the Internet must be able to reserve movie tickets using this prototype.

Usability Requirements
This are one of the most important requirements. The application is fast and easy to use even for unexperienced people with a mobile device. Users have to learn to use this application very fast and fulfill any scenario based on recalling rather then thinking. User must have confidence in the security provided be the application. Therefore a PAY-PAL icon is displayed during all credit card transactions. The product must be competitive.

4.5 Final Requirement Specifications and Prototype

After the workshops conducted with real movie goers data are evaluated, new requirement specifications defined, and old ones updated. This section depicts the final requirement specifications embedded into the final prototype of the Cinema Ticket Reservation System.

4.5.1 Authenticating into the Mobile Application

User enters his/her credentials i.e. user name and password. The authentication is done with the server side. If user is authenticated, a token is returned and used for the current session, and a message is displayed on the user's mobile device for 2 second, followed by the application main menu. The following entries are present in the main menu i.e. *Find Movies*, *My Settings*, *My Tickets*, *My Wallet*, *Help*, and *EXIT*.

The main menu layout is list of entries with a meaningful icon attached to every entry. When an item is highlighted in the menu, a *mouse-over effect* is created.

The splash screen, authentication UI, and the main menu layout can be seen in fig. 4.2

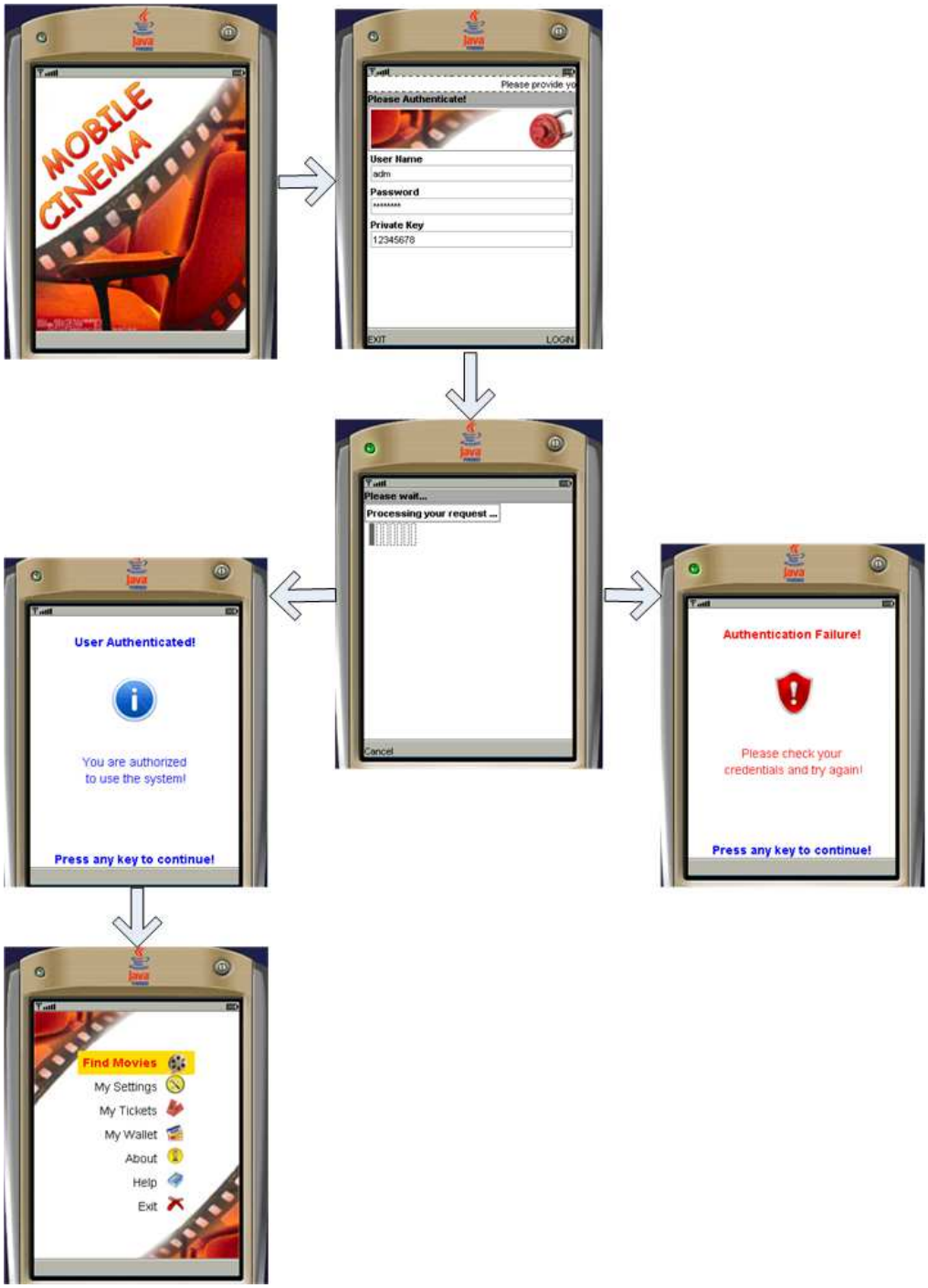


Figure 4.2: The Mobile Application Authentication GUI and Main Menu

4.5.2 Find Movies

This scenario is split into two screens for easy navigation. **In the first screen** users can search for movies in a certain range from a given position. They can enter a *movie*, *street*, *city* or *zip*, a *range*, and a *date* to find all movies in that range on the given date. The *date* is selected by using a *calendar* like feature for an increase usability. The *movie name* allows users to search for a particular movie in the given area on a given date. If a *city name* is entered the corresponding *zip code* is automatically found and displayed; and the other way around. Users can also search for all movies in a city by entering only either a *city* or a *zip code*. The following searching criteria can be used:

- find a particular movie in a certain range from the given position and on a given date
- find a particular movie in a given city and on a given date
- find all movies in a given city and on a given date
- find all movies in a certain range from the given position and on a given date

Once the movies are found, the application displays a list of movies, a list of the cinemas where the movies are played, a list of the hours for each show, and a cinema info field *on the second screen*. The movie select box is the first component in the UI form. Users can select and reserver tickets for a particular show. Users can also see information about the selected cinema i.e. cinema name, distance and map to that cinema from the given position.(section 9) The *movie*, *cinema*, and *show hour* display default values. (One can use as default values e.g. the latest movie, the closest cinema to the user, and respectively the first show hour when that movie is to be played in that cinema. These are a part of a possible future work on the application as mentioned in section 9). The *date field* is visible all the time.

User-friendly messages are displayed in case of an error or the search did not returned any data.

User can choose now to *select seats for movies, view movie descriptions, or rate movies*. The most obvious commands for this scenario have direct access e.g. SELECT SEATS, SEARCH. They are represented by soft buttons. The *Movie Details, Rate Movie, Main Menu, Help, and EXIT options* are depicted as items in the OPTION menu. If user choose to see a movie description, a screen containing the following details is displayed: *movie poster, name, duration, year,*

language, type, parent classification,country, director, artists, user rating, and movie short description.

The screens used during this scenario are depicted in fig. 4.3

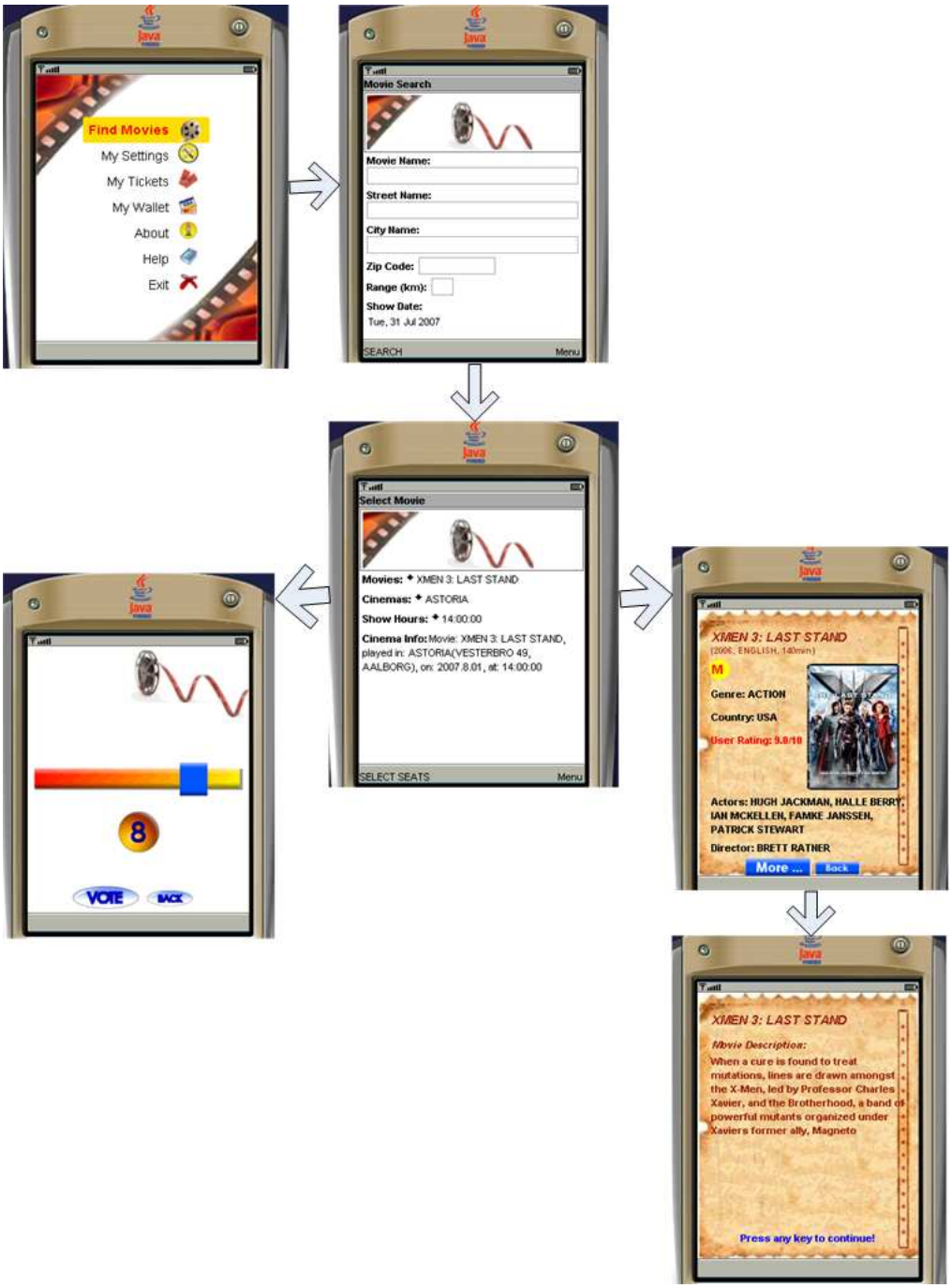


Figure 4.3: Find Movies GUI

4.5.3 Ticket Reservation

This scenario is split into several screens for easy navigation. After a movie is selected in the movie list and the SELECT SEATS button pressed, the cinema hall configuration is displayed. The seats are depicted using a color code i.e. *red - booked seats, green - free seats, and blue - user's current selected seats.* A legend explaining the color code is displayed at the top of the screen.

Users can select seats by using two text boxes i.e. *Row(1 .. No. of rows) and Seat(1 .. No. of seats).* The user enters the corresponding values into those 2 text boxes and press the *(De)Select* button. The chosen seats are highlighted in blue if they are not already booked.

Another method for selecting seats uses a rectangular yellow cursor. The user can jump from one seat to another using the keys *1(left),3(right),2(up),8(down), and 0(select/deselect seat).* Two buttons are displayed on the bottom of the screen i.e *BACK and RESERVE.* The navigation between these buttons is done using the *arrow and OK keys. This method is chosen.*

When the *RESERVE* button is pressed, the seats are selected on the system and a *Ticket Discount* screen is displayed. The allowed discount values are: child, student, pensioner, voucher. The screen displays a list of all tickets, a list of available discounts, the total price to be payed, and a list of details about each ticket. When a discount value is chosen for a selected ticket, the ticket price and the total price to be paid are updated automatically. The user can choose to select any discounts or not.

If the *ACCEPT* button is pressed a *Ticket Payment screen is displayed.* It allows users to select the desired payment method to pay for the reserved tickets. It displays the total amount to be payed together with a list of payment methods. The following payment methods are available:

- **At the Cinema** - the user only reserve the tickets and he/she has to be at the cinema with at least 30 min before the show and pay for the tickets. Otherwise, the system will cancel the tickets automatically within 30 min before the show. Tickets reserved with this method cannot be canceled by the user.

- **Secure Wallet** - if this method is selected, the *My Wallet Authentication* screen is displayed. Once the user is authenticated he/she can select any credit card from the wallet and press the *PURCHASE* button to pay for the tickets. The tickets purchased using this method can be canceled by

the user at any time and refunded as e-money[2].

- **Credit Card** - when this method is selected a payment form displaying credit card data has to be filled in. This data is not saved in the phone memory or DB. It is used only to collect input from the user regarding his/her credit card data. The tickets purchased via this method have the same status as the one purchased with the *Secure Wallet* method.
- **E-Money** - in case of this payment method, a new screen is shown. It displays the total amount to be payed, the list of available payment methods, the available amount of e-money, a message stating if there are enough e-money to pay for the tickets or not. In the later case, a select box for a secondary payment method is used. This secondary payment method is used for paying the difference between the total amount and the amount of e-money. The following secondary payment methods are allowed: *At the Cinema, Secure Wallet, and Credit Card* with the same meanings as explained above.

Once the *PURCHASE* button is pressed any credit card is verified and the payment done. A *Billing Details* screen is displayed. The following information is made available to the user:

- Total payed amount including taxes.
- A list of purchased ticket details. Each entry represents a ticket and provides the following information: *cinema name and address, movie name, show date and hour, row and seat no, discount type, ticket price, and payment method.*
- A message informing the user that all tickets are saved in the phone memory and they can be found in *My Tickets* entry in the *Main Menu.* Users are also informed that they can enter the movie using the saved tickets, and they can cancel any tickets at any time. Only payed tickets are refunded as e-money.

The normal flow of purchasing a ticket is: *Main Menu* $\mapsto$ *Find Movies* $\mapsto$ *Select Movie* $\mapsto$ *Select Seats* $\mapsto$ *Display Ticket Summary and Add any Discount Information if necessary* $\mapsto$ *Choose Payment Method* $\mapsto$ *Display Billing Details* $\mapsto$ *Main Menu.*

The screens used during this scenario are depicted in fig. 4.4

[2]E-Money is a term chosen by the author for responding to the money refund of canceled tickets. This money is electronic money. It cannot be transfered to a real bank account or use anywhere else except the cinemas. They can be used both for purchasing tickets and goods inside the cinema

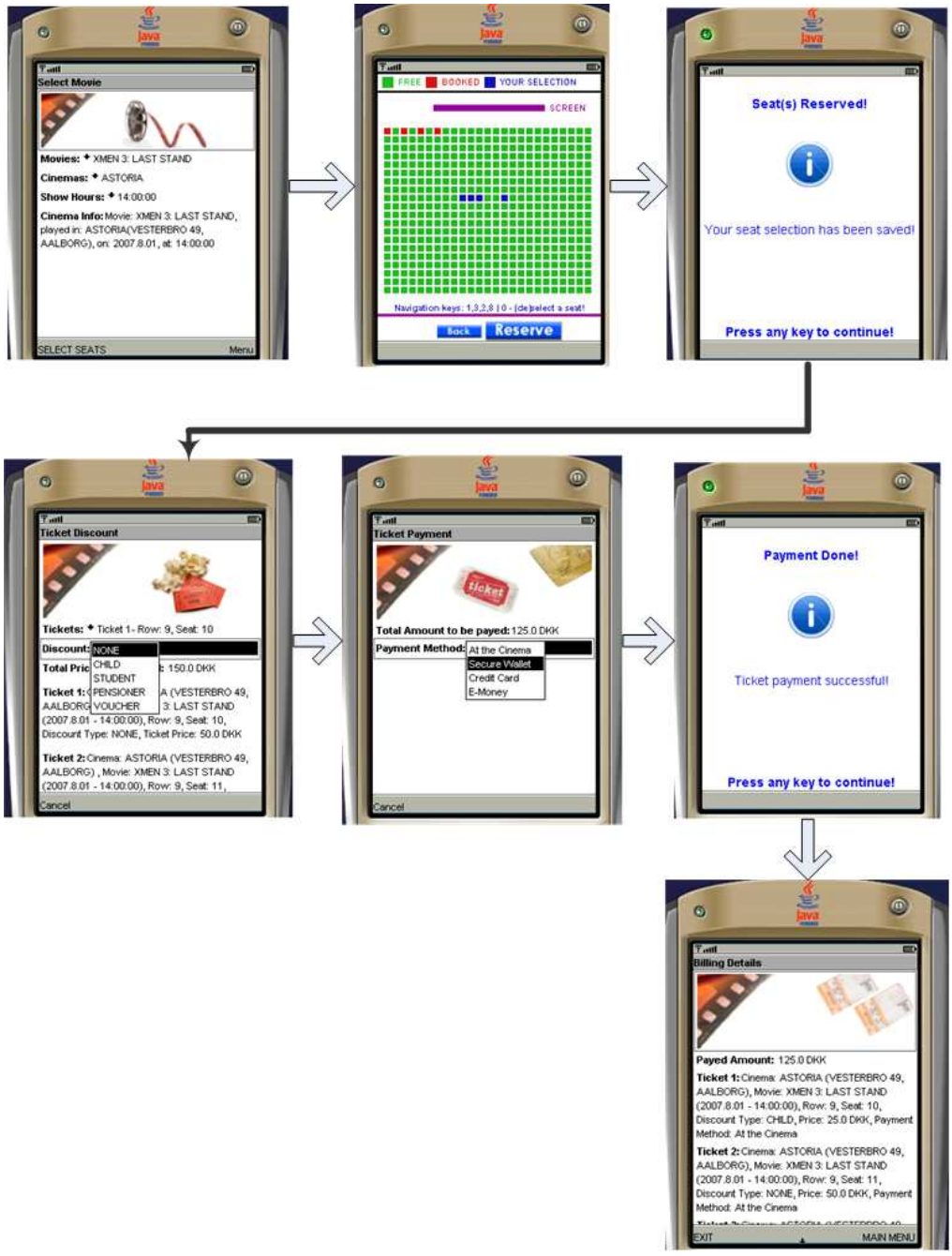


Figure 4.4: Ticket reservation GUI

4.5.4 My Tickets

User can see the reserved tickets under the *My Tickets* entry in the *Main Menu*. The application saves, up to 10 tickets. A list of all tickets is displayed. Every entry in the ticket list has the following format: *Movie,Cinema(Date-Hour),Row,Seat*. User can choose to see details about the tickets using the *VIEW* button or to *cancel* any of the reserved tickets by using the *Cancel Tickets* entry in the OPTIONS menu. Only tickets payed by credit card can be canceled. All other tickets are canceled automatically within 30 minutes before the show. In case of credit card payed tickets, the money are refunded using e-money. The exact amount of e-money corresponding to the real amount of currency is refunded. User can use this money only to purchase cinema tickets later on or buy goods in the cinema. A bar code is displayed for each for easy access to the cinema in case a bar code scanner is placed in the cinema for automatic ticket issuing, payment, and movie access. Only tickets that have not been used are kept in the phone memory. The rest of the tickets(tickets used to see a movie) are deleted automatically at application startup.

If the *VIEW* button is pressed a graphical representation of the ticket is displayed. The layout tries to resemble to a real cinema ticket following the *ticket metaphor*. The following information is displayed: *Ticket ID, a bar code encoding the ticket information, cinema name and address, movie, date, hour, row no, seat no, discount type, payment method, and ticket price.*

If the *Cancel Tickets* entry in the OPTIONS menu is chosen, a YES-NO type dialog widow is displayed asking the users if they are sure that they want to cancel the selected ticket. If the YES button is pressed, the ticket is canceled and money refunded. If the ticket cannot be canceled and info message is displayed.

There is no need to keep a black list of people that do not cancel a previous made reservation when they cannot attend a show and the ticket is not payed. The server side service cancels any reserved and not payed tickets within 30 minutes before the show.

The screens used during this scenario are depicted in fig. 4.5

Figure 4.5: My Tickets GUI

4.5.5 My Wallet

The application provides a feature for *storing users' credit cards* on the mobile phone in a highly secure way. This feature is called *My Wallet*.

A PIN code based authentication method is set up to access the content of the Secure Wallet(My Wallet). If the PIN code is entered wrong three times the wallet content and the PIN code are reseted i.e. all data stored in the wallet are removed. A new PIN code can be set up or the old one changed in case of successful authentication, respectively. When *My Wallet* is used for the first time, a screen is displayed to set up the PIN code. That screen displays a text box to enter the PIN code and another one to verify it. After the PIN code is saved, the *My Wallet Authentication Screen* is displayed. If a PIN code is already set up, the *My Wallet Authentication Screen* is displayed directly when accessing the wallet.

If the wallet authentication is successful, a list of all available credit cards is displayed. Every entry in the list has the following layout: *credit card type, credit card nickname*. The credit card type is depicted as a small picture similar to the real credit card type.

Users can choose to *add/delete/edit/view credit cards*. When a new credit card is saved to the wallet, or an old one deleted/edited, an info message is displayed. The message is made available until the user presses any key on his device. The message informs the user that *THE CREDIT CARD NICKNAME HAS BEEN SAVED/DELETED/UPDATED SUCCESFULY OR NOT!*. In case the user tries to delete a credit card, a dialog screen containing two buttons *YES and No*i s displayed. The message states *Do you really want to the CREDIT CARD NICKNAME?*. This allows users to be in full control of the sensitive operations that affect their personal data.

If a new credit card is to be stored, a form is displayed to enter the required credit card information. The application stores the following data about users' credit cards: *Credit Card Nickname, Credit Card Number, Expiration Date: i.e month - year, Bank Name, Emergency Phone No, Credit Card Type(e.g. VISA, MASTER CARD), PIN, and CW2*. Thus, one can use *My Wallet* for more than paying for a cinema ticket. He/She can store his/her credit card data together with the security code for online payments, and the PIN code needed to use the credit card in any ATM.

When the user chooses to visualize details about a selected credit card, a screen with a credit card-like layout is displayed. All previous mentioned credit card data is displayed together with a small credit card icon that identifies the credit

card type. Three soft buttons are presented i.e. *PIN, CW2, and BACK*. If the *PIN or CW2* buttons are selected, the PIN and CW2 codes are displayed respectively. The *BACK* button displays the credit card list in the wallet.

The screens used during this scenario are depicted in fig. 4.6

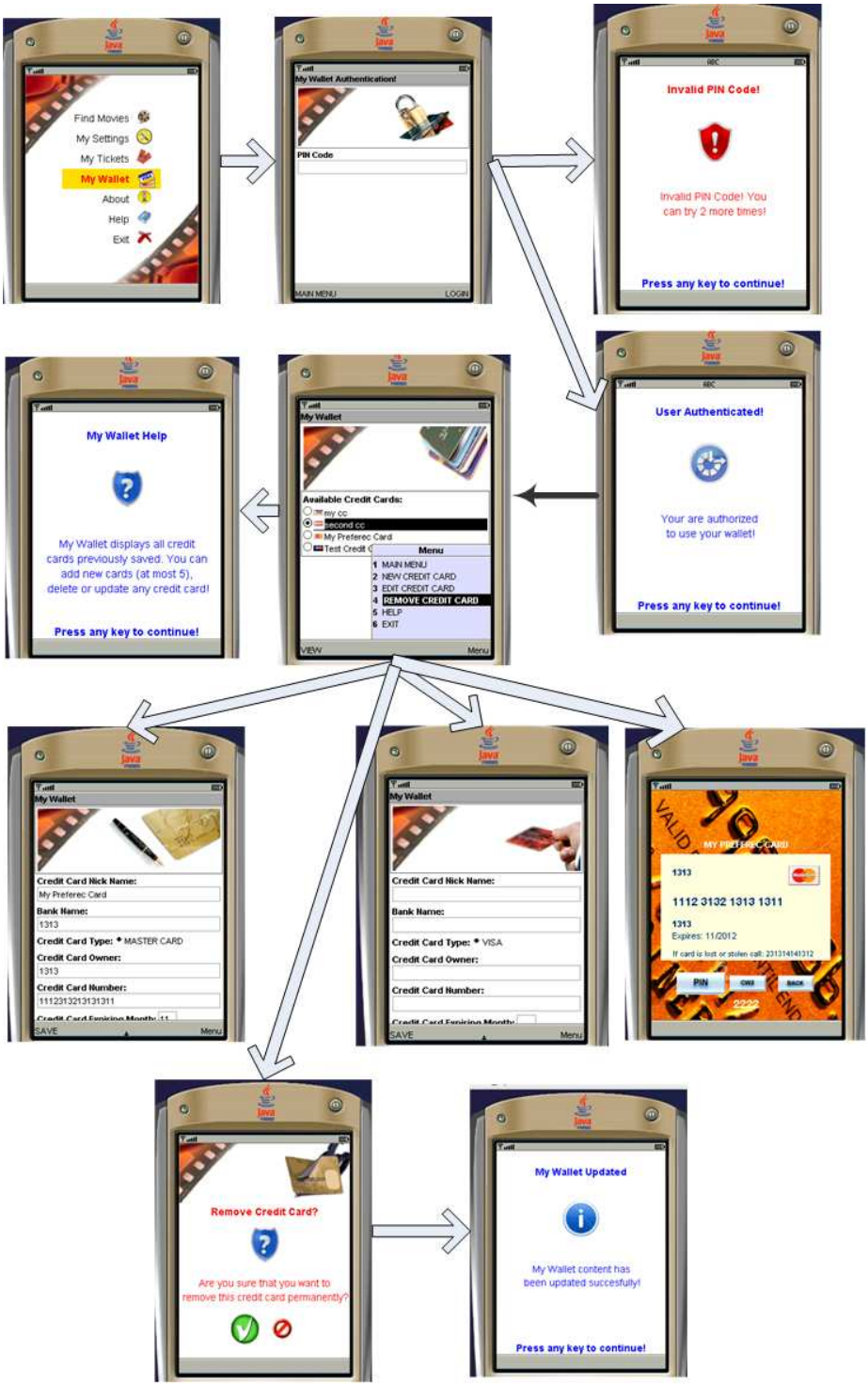


Figure 4.6: My Wallet GUI

4.5.6 My Settings

The application provides a *My Settings* feature that allows users to *change the application password, PIN code for the wallet, and visual theme.*

If the *Change Application Password* is chosen a screen is displayed and the user is prompted to enter his/her old password, the new password, and to reenter the new password to verify it. Once the *Change* button is selected, the password is changed.

In case of the *Change My Wallet PIN* the layout and functionality is as mentioned above.

If users would like to *change the application theme* i.e. colors, layout, images, and text font, they can choose between a red and a blue theme. The changes are performed instantly and the new look of the main menu displayed.

The screens used during this scenario are depicted in fig. 4.7

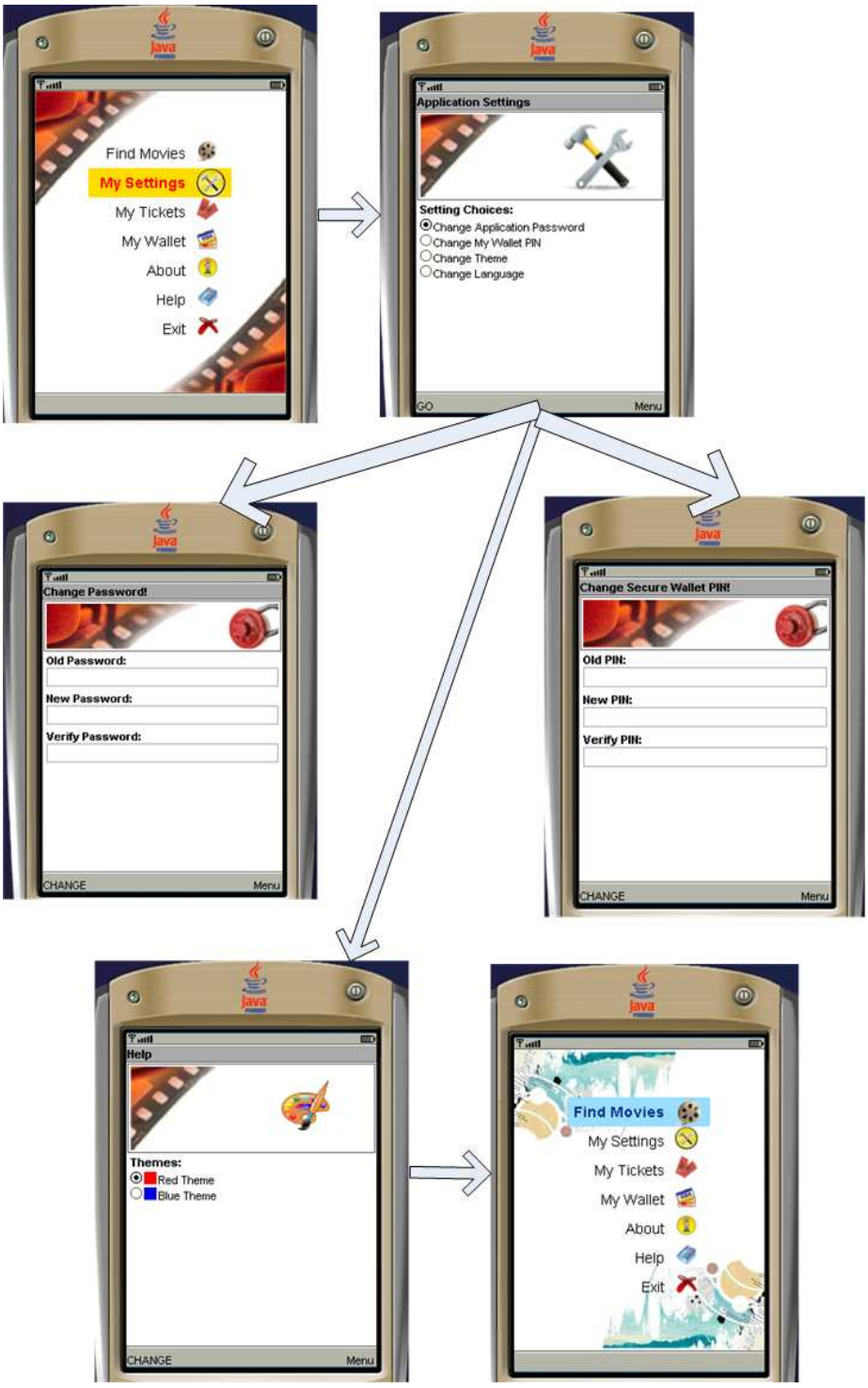


Figure 4.7: My Settings GUI

4.5.7 Help

The application provides users with *Help* information on each of the screens. This information can help users in solving any misunderstandings or recovering from any possible errors.

A *Help* entry is also depicted in the *Main Menu*. Several topics are displayed in a select box. Users can choose among the following topics: *My Wallet, My Tickets, My Settings, Find Movie, Login, and Credit Card Security*. When a topic is chosen in the select box, topic details are depicted in another text field.

The screens used during this scenario are depicted in fig. 4.8

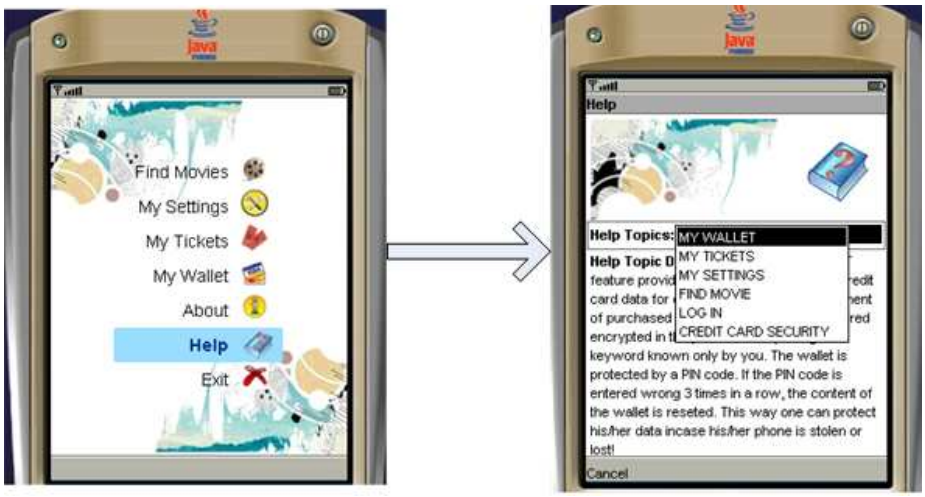


Figure 4.8: The Help GUI

4.5.8 Exit

This command performs the necessary clean up and exits the main application.

The screens used during this scenario are depicted in fig. 4.9

Figure 4.9: Application Exit GUI

4.5.9 Other Design Decisions

The interviewed users do not want to read/write movie reviews on their mobile phone due to limited input capabilities. Therefore, the *movie review* feature is not taken into consideration further on.

They do not want to watch trailers for the selected movies. The users' explanation is, and I quote *it's to slow and costs too much*. Therefore, the *movie trailer* feature is not taken into consideration further on.

The *Message Info Screens* provide users with enough information about a specific action or request.

4.6 When does it end?

Once an idea is presented, prototyped, and tested numerous times in varying ways, when can designers sit back and say "The product is ideal to the market."? Truly the product will never be perfect; someone will always have a complaint about one thing or another and personal interests will change from user to user. Even the same user could change their opinion on a product during a certain period of time, going back on what they had originally said.

The reality lies in the effort put into the research. Going as far as to recognize cognitive reactions, aesthetic visuals, data collection and user analysis will give an excellent idea as to where your design must go in order to please the target market and make an ideal product. However, considering the facts above, people will change, technology will change, and so the product must change. So the real value, the most important thing that can come from all Interaction Design work, is the data records.[18]

When the world changes and products need to follow suit, nothing will serve a designer better than a well documented design history of similar products. In this way testing will not need to be repetitive and new ideas can be constructed from historical patterns in user requests.

Interaction Design, while time consuming, serves a critical purpose in making the most valuable and accurate product, avoiding valuable development time on programs that will never survive on the market.[18]

Chapter 5

The Design of the Cinema Ticket Reservation System

One of the most important design issue when developing client - server applications is the overall system architecture. There are 2 models that can be used for describing such a system i.e. the *two-tier model* and the *n-tier model*. Both model are discussed. The *n-tier model* is chosen in this case.

The two-tier model
This is the classical client-server model where the client knows how to access the DB server. This results in a tight coupling between the client and the server leading to important issues such as: difficult system maintenance and low scalability. Changes made in the server or DB will crash the client. This model is suitable for LAN environments.

The n-tier model
This model has a client tier, one or more server tiers, and a middle layer used to maintain the DB connections. The middle tier is usually implemented by a web server e.g. Tomcat. This system is scalable by adding new middle or server tiers, provides easy support for authentication, but it adds some extra complexity to the system. A *three-tier system* is chosen as the architectural solution for the Cinema Ticket Reservation System.

5.1 Design of the Relational Database

As mentioned in section 2.3 a *Relational Database Management System* is used for server side persistent data storying in case of the Cinema Ticket Reservation System. A RDBMS has several advantages over the regular file system such as:[3]

- **Concurrent Accesses** - supported by the database driver implementation;
- **Isolation** - transaction execute one at a time;
- **Atomicity** - transaction execute either completely or not;
- **Durability** - the ability to recover from failures or errors of many types;
- **API** - a powerful API for accessing the data;
- **SQL** - a powerful query language available to the developer;
- it supports flexible access to large amount of data.

Independent of the chosen RDBMS, the DB design is similar for all RDBMS e.g. PostgreSQL, MySQL, Oracle, Firebird, Predator, etc.

The design process of the cinema system DB begins by analyzing and defining the requirements for the data storage. The information that need to be stored is investigated and the relationships among different entities are defined. Once the requirements are clear, an *E/R (Entity-Relationship) Data Model* is created and the corresponding schema defined. Further on, the E/R model is mapped to a *Relational Data Model*. The model is further normalized into the *Boyce-Codd Normal Form*. Needs for *decompositions* are analyzed. *Primary and foreign keys*, *indexes*, *constraints*, *queries*, *stored procedures*, and *transactions* are analyzed and defined. The DB modeling and implementation process is depicted in fig. 5.1.[3]

Figure 5.1: The DB modeling and implementation process[3]

5.1.1 DB Requirements Specifications

The Cinema Ticket Reservation System uses as server side persistent storage a RDBMS. The overall system uses data from several cinema centers, every one of them with multiple theaters where movies are played. Each cinema entity stores the address and has a connection to its theaters. Every theater defines the number of rows and seats per row in the theater. Every seat contains information that connects it to the cinema, theater, and a show. It also has a status attribute i.e. free or booked.

A movie entity must be defined for storing movie details such as: movie name, duration, genre, language, year, poster, description, actors, directors, etc. A movie can be shown in more than one theater at a time.

A movie goer can login into the Cinema Ticket Reservation System, search for movies and cinemas, view movie details, purchase/cancel tickets or rate different movies. Therefore, movie goers credentials are stored in the DB. Any movie goer can use several payment methods when purchasing tickets, such as: credit card, e-money, or at the cinema. Any movie goer can also cancel any credit card payed ticket. When a ticket is canceled the money are refunded as e-money. A new concept of *e-money* is introduced. E-money are money refunded in case a movie goer cannot attend a show and he/she cancels his/her tickets. The e-money are stored into the movie goers DB account. They cannot be exchanged into real currency, but they can be used for purchasing other cinema tickets or goods inside the cinema.

Movie goer can buy discount tickets e.g. child, student, pensioner, and possessor of a voucher. Any movie goer can store on the system cinema tickets for more than one show.

A show can be played at different hours during the day. Each show has its own price range depending on the cinema, theater, movie, hour and day of the show. Discounts are subtracted from the base price for each show.

5.1.2 The Entity-Relationship Data Model

Based on the requirements stated in section 5.1.1, entity-sets and relationships among them are defined. The information about the entity-sets that are stored in the DB is defined i.e. attributes, entity-sets. One - one, many - one, and many - many relationships are analyzed and defined. Subclasses and weak entity-sets are identified. Conversion of any relationships into entity sets is investigated. Additional constraints are defined e.g. primary and foreign keys, single value constraints, referential integrity constraints, domain and general constraints. The system ER diagram can be seen in fig. 5.2.

Several week entity sets are defined i.e. *Rating*, *Shows*, and *CinemaHalls* to avoid data redundancy in the DB and preserve the DB design principles when constructing the DB structure i.e. redundancy, simplicity, faithfulness, and choosing the right relationships or element[3]. It is considered that the keys of these entity sets are made of attributes belonging to other entity sets e.g. *Rating* has the primary key made of both the primary keys from *Users* and *Movies*; *Shows* has the primary key composed of the primary keys of *ShowLocation* and *ShowTime* entity sets; a tuple in *CinemaHalls* is identified by the *Cinema primary key* and a *hallID attribute of CinemaHalls*.

The following *One-One Relationships* are defined i.e. *PaymentMethod - Reservations*, and *Tickets - BookedSeats*.

The following *Many-One Relationships* are defined i.e. *Cinema - CinemaHalls*, *Movies - ShowLocation*, *HourOfShow - ShowTime*, *DateOfShow - ShowTime*, *Reservations - Tickets*, *Reservations - Shows*, *BookedSeats - Shows*, *ShowDiscounts - Tickets*, *Shows - Prices*, *ShowLocation - CinemaHalls*, *Reservations - Users*, and *Reservations - Shows*.

The following *Many-Many Relationships* are defined i.e. *DiscountSchema - Shows*.

The *primary keys* are represented in fig. 5.2 by underlined attributes. *Referential integrity* is enforced by forbidding deletion of a referenced entity and allowing deletion of all entities referenced by a deleted entity, or by enforcing the update of a referenced entity. Referential integrity is imposed between *Cinemas - CinemaHalls, CinemaHalls - ShowLocation, Movies - ShowLocation, Tickets - Booked Seats, Users - Reservations, and Movies - Ratings*. Domain constraints that restrict the value of an attribute to be in a given interval are defined on the attribute level.

More details about the meaning of each entity-set can be seen in section 5.1.3.

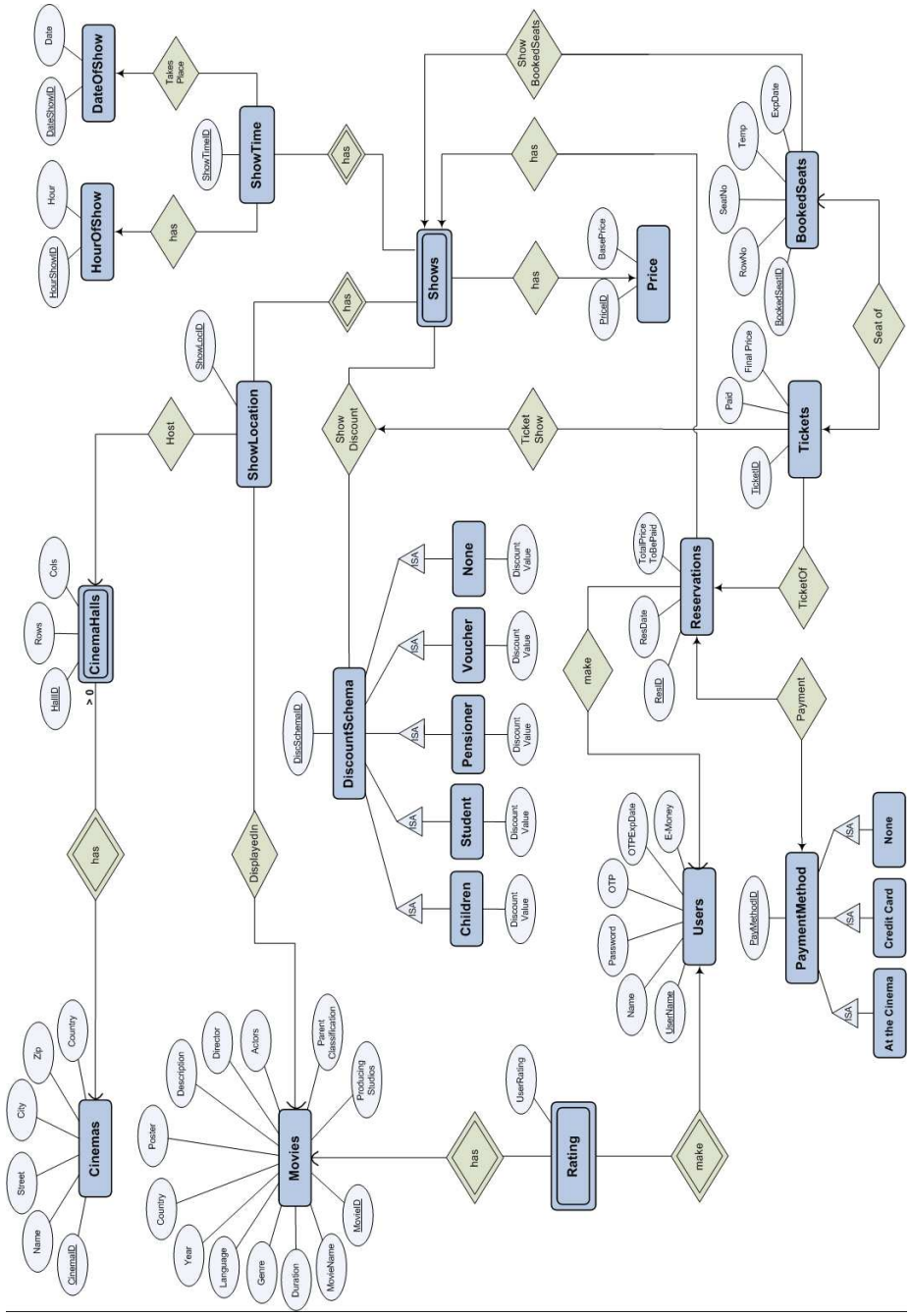


Figure 5.2: The Cinema System Entity-Relationship Model

5.1.3 The Relational Data Model

The Relational Data Model is constructed around a component called *relation*i.e. a 2 dimensional table, as its primary concept. *Attributes* are properties of a relation. A relation together with its attributes represent a *schema*.

As depicted in fig. 5.1, after creating the E/R Model one must convert this into a *Relational Data model*. This is done by using several conversion rules such as:

- all subclasses are converted into relations using the *Null values rule* i.e. the whole ISA tree corresponds to a relation. The following ISA relationships adhere to this conversion: (PaymentMethod - At the Cinema, Credit Card, None) and (DiscountSchema - Children, Student, Pensioner, Voucher, None).
- There is no need to implement each subclass as a different relation. This way, the previous subclasses are translated into values with the same name for a *PaymentType* or *DiscountType* attribute in the *PaymentMethod* and *DiscountSchema* relations, respectively.
- Many to many relationships are translated into relations e.g. Discount Schema - Shows into ShowDiscount.
- One-One and Many-One relationships are translated into foreign/primary keys into the entity sets involved in the relationship. This way, there is no need of decomposition for that relation i.e. redundancy is avoided.
- Weak entity-sets are converted into relations having the primary key composed of all primary keys of helping entity sets e.g. *CinameHalls*, *Shows*, and *Rating*.

Primary keys are depicted below as underlined attributes with a continuous line, while the foreign keys are depicted in italic.

Considering all previous mentioned transformation rules, the following relational data model can be stated.

CINEMAS - Entity sets that contains all cinemas in the system

- CinemaID - the primary key. This key uniquely identifies a cinema tuple in the Cinemas entity set.
- CinemaName - the official name of the cinema

- Street - the street where the cinema is located
- City - the city where the cinema is located
- Zip - the zip code of the city where the cinema is located
- Country - the country where the city is located

CINEMA HALLS - Theaters that belongs to different cinemas. This is a weak entity set.

- CinemaID - the first component of the primary key
- HallID - the second component of the primary key. Together with CinemaID they form the primary key for this entity set and uniquely identify a tuple.
- Rows - the number of rows in a theater
- Cols - the number of seats on a row in a theater

MOVIES - The movies shown in different cinemas

- MovieID - the primary key. It uniquely identifies a tuple in the entity.
- MovieName - the official name of the movie
- Duration - the duration of the movie in minutes
- Genre - the genre of the movie e.g. action, thriller, etc
- ParentClassification - parent classification e.g. allowed to minors, allowed between 12 and 15, etc
- Language - the language spoken in the movie
- Year - the year when the movie was produced
- MovieCountry - the country of the studio producing the movie
- Poster - the movie poster
- Description - a short description of the movie
- ProducingStudios - the name of the producing studio
- Director - the movie director

- Actors - a list of movie actors

SHOW LOCATION - entity set for defining the location of a show with respect to the cinema, theater and movie. A movie can be played in more than one theater/cinema at a time.

- ShowLocationID - the primary key. It uniquely identifies a tuple in the entity.
- *CinamaID* - It is a foreign key in here and connects ShowLocation with CinemaHalls. This is the primary key of CinemaHalls entity set.
- *HallID* - It is a foreign key in here and connects ShowLocation with CinemaHalls. This is the primary key of CinemaHalls entity set.
- *MovieID* - It is a foreign key in here and connects ShowLocation with Movies. This is the primary key of the Movies entity set.

DATE OF SHOW - defines all dates when the shows are played

- DateShowID - the primary key. It uniquely identifies a tuple in the entity.
- DateOfShow - the dates when the shows are displayed

HOUR OF SHOW - defines all hours when shows are played

- HourShowID - the primary key. It uniquely identifies a tuple in the entity.
- HourOfShow - the hours when the shows are displayed

SHOW TIME - entity set that defines a unique combination of date-hour for each shows. It combines data from DateOfShow and HourofShow entity sets

- ShowTimeID - the primary key. It uniquely identifies a tuple in the entity.
- *DateShowID* - It is a foreign key in here and connects ShowLocation with DateOfShow. This is the primary key of DateOfShow entity set.
- *HourShowID* - It is a foreign key in here and connects ShowLocation with HourOfShow. This is the primary key of HourOfShow entity set.

PRICES - entity set defining the base prices for all shows

- PriceID - the primary key. It uniquely identifies a tuple in the entity.
- BasePrice - the base prices for all shows

SHOWS - entity set that contains all shows users can book tickets for. This is a weak entity set.

- ShowLocationID - the first component of the primary key. It gets this part of the key from the ShowLocation entity set.
- ShowTimeID - the second component of the primary key. It gets this part of the key from the ShowTime entity set. Together with ShowLocationID they form the primary key of this entity set.
- *PriceID* - It is a foreign key in here and connects Shows and Prices. This is the primary key of the Price entity set.

DISCOUNT SCHEMA - entity set that contains the discount types and values to be applied for the final ticket price.

- DiscSchemaID - primary key that uniquely identifies a tuple in the entity.
- DiscountType - discount type i.e. children, student, pensioner, voucher or none
- DiscountValue - the percentage value of the discount to be subtracted from the ticket base price

SHOW DISCOUNT - entity set connecting the Shows with the DiscountSchema. Each show has its own discount schema. This is the result of a many to many relationship.

- ShowLocationID - the first component of the primary key. It gets this part of the key from the ShowLocation entity set.
- ShowTimeID - the second component of the primary key. It gets this part of the key from the ShowTime entity set.
- DiscSchemaID - the third component of the primary key. It gets this part of the key from the DiscountSchema entity set. Together with the first 2 components they form the primary key of this entity set.

USERS - entity set containing users' data

- UserName - the primary key. It uniquely identifies a user in the system.
- Name - The real name if the user
- Password - The password used to authenticate the user together with the user name.
- E-money - The amount of e-money users have as result of ticket refunds.

RATING - entity set that contains movie rating scores per user. One user can have only one rating per movie.

- UserName - the first component of the primary key. It gets this part of the key from the Users entity set.
- MovieID - the second component of the primary key. It gets this part of the key from the Movies entity set. Together with the first key component forms the primary key of this entity set.
- UserRating - The given score for a movie. It ranges from 0 to 10.

PAYMENT METHOD - entity set that defines the payment methods movie goers can choose to pay for the reserved tickets

- PaymentMethodID - the primary key. It uniquely identifies a payment method in the system.
- PaymentMethodType - the payment method type i.e. credit card, at the cinema, or voucher.

RESERVATIONS - entity set that holds all reservation details in the system. A reservation can contain more than one ticket.

- ResID - the primary key. It uniquely identifies a reservation in the system.
- ResDate - the date when the reservation is made
- TotalPriceToBePaid - the total price to be paid for the reserved tickets
- *PaymentMethodID* - It is a foreign key in here and connects Reservations with PaymentMethod. This is the primary key of the PaymentMethod entity set. It is used to identify the payment method for this reservation.

- *UserName* - It is a foreign key in here and connects Reservations with Users. This is the primary key of the Users entity set. It is used to identify the user who made the reservation.
- *ShowLocationID* - It is a foreign key in here and connects Reservations with ShowLocation. This is the primary key of the ShowLocation entity set. It is used together with the following foreign key to uniquely identify the show for this reservation.
- *ShowTimeID* - It is a foreign key in here and connects Reservations with ShowTime. This is the primary key of the ShowTime entity set.

BOOKED SEATS - entity set containing all booked seats by different users for different shows

- BookedSeatID - the primary key. It uniquely identifies a booked seat in the system.
- RowNo - The row no of the booked seat in the theater
- SeatNo - The position of the booked seat on the row in the theater
- ExpDate - date field used for automated cancellation of booked seats via pay at the cinema method in case users do not show for the movie. These seats are cancelled with 45 min before the show.
- *ShowLocationID* - It is a foreign key in here and connects Reservations with ShowLocation. This is the primary key of the ShowLocation entity set. It is used together with the following foreign key to uniquely identify the show for this reservation.
- *ShowTimeID* - It is a foreign key in here and connects Reservations with ShowTime. This is the primary key of the ShowTime entity set.

TICKETS - entity set containing all tickets reserved by all users for different shows

- TicketID - primary key that uniquely identifies a ticket in the system.
- FinalPrice - the final price for one ticket.
- Paid - boolean attribute indicating if the ticket has been payed with a credit card or not.

- *ShowLocationID* - It is a foreign key in here and connects Tickets with ShowLocation. This is the primary key of the ShowLocation entity set. It is used together with the following foreign key to uniquely identify the show for this reservation.
- *ShowTimeID* - It is a foreign key in here and connects Tickets with ShowTime. This is the primary key of the ShowTime entity set.
- *DiscountSchemaID* - It is a foreign key in here and connects Tickets with DiscountSchema. This is the primary key of the DiscountSchema entity set and uniquely identifies the discount value for this ticket.
- *ResID* - It is a foreign key in here and connects Tickets with Reservations. This is the primary key of the Reservations entity set and uniquely identifies the reservation no this ticket belongs to.
- *BookedSeatID* - It is a foreign key in here and connects Tickets with BookedSeas. This is the primary key of the BookedSeas entity set and uniquely identifies the seat corresponding to this ticket.

5.1.4 Needs of Decomposition and Normalization

During the design of the E/R Model as depicted in section 5.1.2, a *Boyce-Codd Normal form* is achieved i.e. if a set of attributes of a relation determine another attribute, it has to determine all attributes of that relation.[3] The DB normalization used the decomposition methods of 1st, 2nd and 3rd normal form.

- **1st normal form** - no repeating elements or group of elements is fulfilled. Each entity set has a unique key to uniquely identify any tuple in the entity set. There are no duplicates in the entity sets and no redundancies in the DB.
- **2nd normal form** - no partial dependencies on a concatenated key[3] is fulfilled except the ZipCode attribute in the Cinema entity set. The solution is to create a new table that connects a zip code to one city. This involves an extra complexity level in the queries for extracting data out of the DB. Therefore, the current solution is considered as acceptable.
- **3rd normal form** - all tuples in the entity sets should depend only on the primary key and not on any other attributes. Considering the proposed DB structure, this normal form is also fulfilled.

One can state that the DB schema depicted in section 5.1.3 is normalized and fulfills the Boyce-Codd Normal form. No needs of decomposition are further identified.

5.2 The Design of the Mobile Client Application

The following chapters depicts the different solutions considered during the design process of the mobile client application and argues the chosen ones.

5.2.1 On Device Data Storing and Application Configuration

This mobile client is designed to be used by more than one user on the same mobile device i.e. different users can access the same mobile application and save their own credit cards, tickets, or preferences without interfering with other users, and without any security issues. Therefore each user has access to its own memory space. No memory space intersections is allowed due to security reasons. That can be achieved by using the built in data storage solution provided by the Record Management System. A *different record store* is to be used for each user - the memory separation is achieved by default.

Sensitive data such as credit cards is stored encrypted in user's own record store. User's data is therefore protected against any brute force attacks.

The solution for storing data in RMS allows easy access to the data. The complexity of the search, read, and write operations must is minimum and the speed maximum, respectively.

Thus, a *hashmap* like solution is chosen i.e. all data is stored in RMS based on a *(key, value)* pair construction. Every key uniquely identifies a particular property or data saved in RMS. All keys must be unique. A key has a predefined form i.e. a 3-letter word e.g. *CC1* - first user's credit card, *USR* - user name, etc.

In order to preserve the *hashmap* characteristics of the proposed solution special attentions is taken when data is written or updated in RMS. If a new *(key, value)* pair is added to RMS, the application must ensure that key has not been saved in RMS, in the first place. If the key already exists, the old value is deleted and the new value written to RMS. In case of update operations, the application deletes the entry with the given key and writes again the updated value. Otherwise, one might end up having two entries with the same key.

The above proposed solution is also used for storing *configuration parameters of the mobile client* e.g. *number of reserved tickets, number of saved credit cards, user private key, default application theme ID, etc.* The client reads these parameters at startup and initializes the application in the background. Caching of search results can also be enabled by saving a particular configuration parameter in RMS.

5.2.2 My Secure Wallet

My Wallet feature ought to provide strong security and protection of user sensitive data i.e. credit card information. The access to this feature is protected by a *PIN code* authentication mechanism on top of the main application authentication. This provides a double layer of security to access the wallet. Moreover, all data saved in the wallet is encrypted and accessible only if the user is authenticated. A maximum number of six credit cards can be stored in the wallet. This is needed for low memory usage considerations. A possible future work as described in section reffuturework can allow to set up the max no. of credit cards, dynamically based on the mobile device memory characteristics.

Brute force and dictionary attacks are consider. Let us assume the following scenario: a movie goer loses its mobile phone. Another person finds the mobile phone and starts playing with it. While looking through the notes he finds a note with the password and private key to access the Mobile Cinema Applications. He writes down both of them and navigates to applications/MobileCinema. He gets access to the application by using the previous found credentials. He notices in the main menu an entry called *My Wallet* and tries to access it but is prompted for a PIN code. Now he starts to key in different combinations. After 1 hour of trial and errors he manages to log in and see all credit cards saved in the wallet.

To prevent such situations, a solutions must be found.

A first solution allows user to try to login to the secure wallet feature for at most 3 times. If he does not succeed that, the wallet is locked and it cannot be used anymore. An alternative solution can be provided to allow the secure wallet unlocking from the server side. This introduce an overhead of security and authentications. Moreover, during the interviews conducted with real movie goers they disliked the idea of having a feature they cannot used just because they made a mistake when entering the PIN code.

Therefore, a second solution is proposed. A movie goer is allowed to try to login to the secure wallet for at most 3 times. If he does not succeed that the wallet and the access PIN code are erased. The movie goer can setup a new PIN code and save again his/her credit cards in the wallet. This solution ensure the security of movie goer sensitive data by protecting it against any brute force attack and preserve the application usability in the same time.

The second solution is chosen as final solution to the secure wallet feature.

Movie goers can add credit cards to the wallet, update any credit card information, or permanently delete any credit card from the wallet. They can navigate

through the credit cards via a menu displaying the *credit card types* e.g. VISA, American Express, etc. as an image, and the chosen *nick names*. When a new credit card is added to the wallet, the movie goer must assign a nick name to it. That nick name is to be used in the wallet credit card list.

When movie goers access the wallet for the first time they must setup the access PIN code. Once the PIN code is set up they can log in and use the wallet. The PIN code is saved in RMS as a configuration parameter. That makes the PIN code persistent among different instances of the application for the same user name.

5.2.3 The Ticket Manager

The ticket manager feature provides access to all tickets saved in the phone memory by the movie goer. Easy identification of the saved tickets is possible. The tickets are displayed as menu list. Each entry in the menu list represents a purchased ticket and contains enough data to identify that ticket i.e. *movie, cinema, show date/hour, row and seat.*

Movie goers can view ticket details or cancel any paid tickets. The view function of the ticket manager allows movie goers to use their mobile devices on a scanner present at the entry point of the cinema theater in order to pay for the tickets or get access to the selected show. Therefore they do not need to purchased the paper back version of the ticket. Environmental issues are addressed in that way.

Movie goers can purchase movie tickets using several payment methods i.e. *credit card*, *pay at cinema*, or *E-Money*. Only tickets paid using the credit card payment method can be canceled. Usually, a cinema ticket purchased on-line cannot be canceled. Thus, movie goers will lose their money in case they cannot attend the show. This solution is chosen in the first working prototype of the application. After several user interviews, a new use case is proposed by the interviewed movie goers. *They expressed their wishes to be refunded for any canceled paid ticket.*

A possible solution to this new use case consists in saving user credit card data on the server side or asking user to provide an account number during the ticket purchase operation. If movie goers cancel a ticket, the given account no will be used for refunding purposes. This solution introduces some overhead such as: bank transactions, fees, etc. It might be that the fee for transferring the money into the movie goer's account is bigger than the ticket price. This solution is not acceptable for any cinema.

A new approach is proposed by introducing a new concept i.e. *e-money*. This is electronic money that is refunded to the movie goers when they cancel a previously paid ticket using the credit card payment method. The amount of user's e-money is saved on the server side in user's account and also on his/her mobile device. Movie goers can use their e-money to purchase new tickets, or buy different goods inside the cinema e.g. candies, pop corn, soda, etc. The e-money cannot be transfered from one user to another or from one account to another. They also cannot be exchanged in real currency. This solution is considered as the final approach to the ticket refund issue.

The ticket manager view function allows movie goers to view ticket details such

as *ticket ID, movie name, show date/hour, cinema name and address, cinema theater, row and seat.* All this information is displayed in a user friendly way. This has to follow the *ticket metaphor* depicted in the conceptual model from section 4.1. Every ticket also displays a bar code image that encodes the ticket details. If cinemas have scanners at the cinema theater entries, movie goers can use their mobile phones on the scanners. The scanners will read the ticket bar code information and the movie goers can enter the shows. A small keyboard can be made available to allow movie goers to key in the ticket IDs' in case the bar code cannot be read. The bar code feature can also be used when purchasing a ticket at the cinema i.e. the movie goer will slide the mobile phone on top of a scanner or key in the ticket ID. He/she will be asked to insert a credit card or the right amount of real money. Thus, the ticket is payed and the correct change returned. The payment system will be similar to the ones used for purchasing train/bus tickets.

After 30 minutes a show has begun, the tickets purchased for that show are considered expired. The mobile ticket manager deletes any expired tickets from the phone memory at the application start up and free any unused memory.

5.3 Mobile Device Client - Server Side Service Communication Protocol

The communication between the Mobile Device and the Server Side Cinema Service is split in eight parts based on the requests sent by the movie goer i.e. *authenticate*, *change password*, *search movies*, *reserve/purchase tickets*, *cancel tickets*, *rate movies*, *view movie details*, and *background cinema theater update*.

The overall system architecture is depicted in fig. 5.3

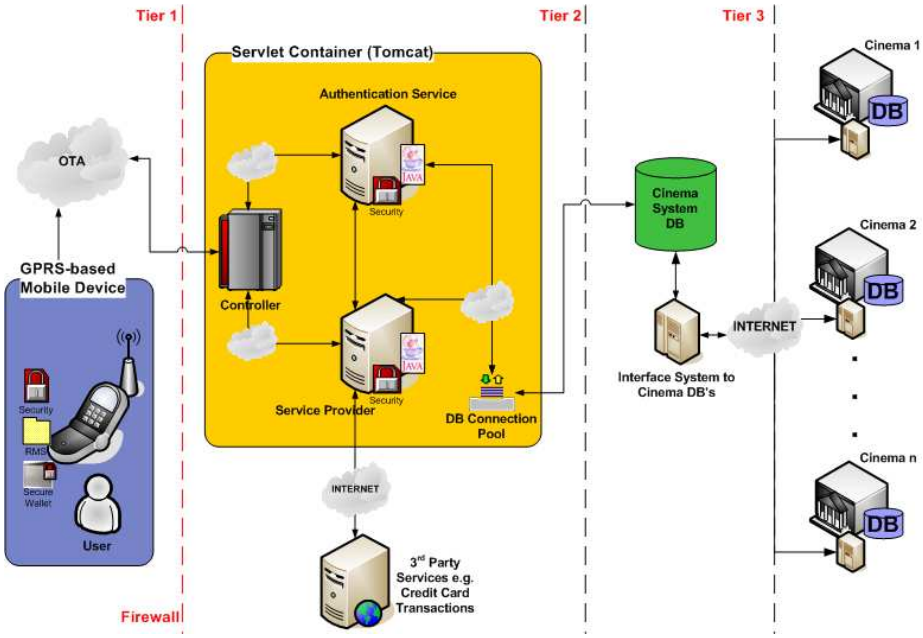


Figure 5.3: The Architecture of the Cinema Ticket Reservation System

To minimize data-traffic, and thereby cost and waiting time, the protocol focuses on minimizing the amount of sent data and not on easy human readability. The communication between the client and the server is done using HTTP POST requests.

General Considerations about the Communication Protocol

- The communication between the client and server is done by means of object passing i.e. request ad response objects are transferred and inter-

preted on both ends. Once a request is sent from the client side, a request object is embedded into the POST request. On the server side, the request object is interpreted, and the desired information extracted. The corresponding business logic is performed for that type of request and a response object is sent back on the same HTTP channel. The response object is interpreted on the client side and the result displayed to the movie goer.

- The server will send back only the response code and no content when that is NOT necessary. This is important in order to keep the data-traffic at minimum and separate the business logic from the view. The client will present the message to the user function of the response code e.g. *authentication successful, ticket canceled successfully, etc.*
- The server will send back a response code and content when that is necessary e.g. *list of movies, cinema hall configuration and status, reservation information, reviews, trailers,* etc.
- If the server experiences an internal error it should return a *500 Status Code (HTTP_INTERNAL_ERROR)* to help the client cope with this error.
- If an invalid protocol step is received the server should return a *501 (HTTP_NOT_IMPLEMENTED) Status Code* to help the client cope with this error.
- If an error occurs while paying for the tickets the server should return a *420 Error Status Code* to help the client cope with this error.
- If an error occurs while decrypting the request the server should return a *16 Error Status Code* to help the client cope with this error.
- If an error occurs while reading/writing the request object to the network the server should return a *14/15 Error Status Code* to help the client cope with this error.
- If an error occurs while reading/writing the response object to the network the server should return a *11/12 Error Status Code* to help the client cope with this error.
- If a request is executed successfully the server should return a *200 OK Status Code* together with a *secondary status response code (operation response code)* to help the client cope with this error. The operation response code can be a *2xx OK Value* in case the operation is successful, or a *4xx Error Value* in case the operation failed e.g. *no movies found matching the searching criteria, user cannot be authenticated, etc.* The *2xx* and *4xx* values are split among all eight previously mentioned types of requests as depicted below. That way the client can distinguish among

the error types and display the appropriate error message to the movie goer. More details about the status code values and significations can be found in the following chapters.

5.3.1 Movie Goer Authenticates

An authentication request can be made during the client-server communication. This is depicted in fig. 5.4. More details about the authentication protocol can be found in section 6.3.2. The protocol can be summarized in the following steps:

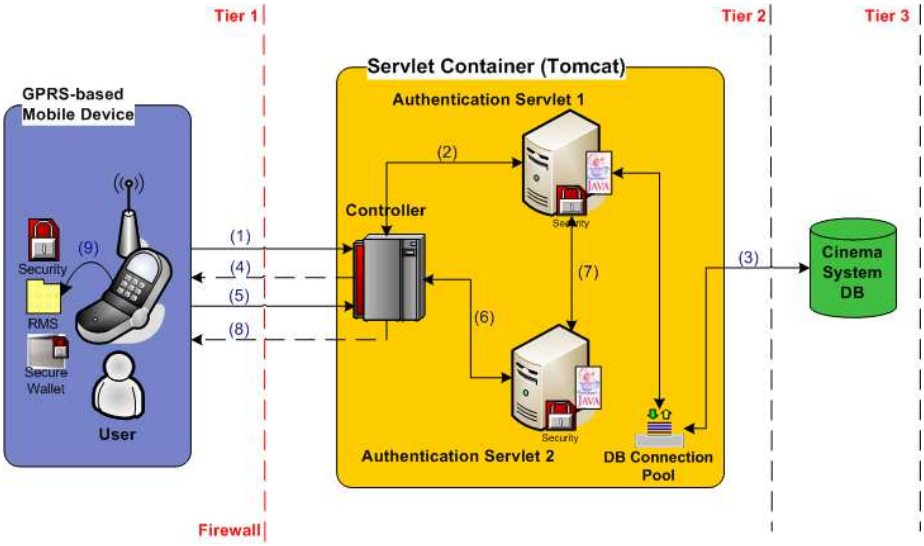


Figure 5.4: The Communication Protocol for Movie Goers Authentication

1. The movie goer starts the application. After 2-3 seconds while the splash screen is displayed, the movie goer is presented with the authentication screen. User introduces his/her *credentials(user name and password)* and presses the *Authenticate* button. The request is sent as an HTTP request over GPRS to the central entry point of the Cinema Service i.e. *Cinema Central Controller Servlet*. (1)

2. The central controller checks the type of request and forwards the request to the corresponding worker servlet i.e. either*Authentication Servlet 1*(2) or *Authentication Servlet 2* (5), depending on the authentication step. If the movie goer is authenticated keys are exchanged. The key is saved into the user's RMS for further requests. (9).

3. A server side status code, together with the key and the total amount of e-money in user's account are sent back to the mobile device client as a response. (8)

4. The main menu is displayed if the authentication is successful or an error message otherwise.

The following data is sent to the server as a request object during the POST request: **user name**, and **password**.

The following data is received from the server as a response object to the previous made request:

- **Operation Status Code** - one of the following:
  - **221** - *AUTHENTICATION E-MONEY OK* - user is authenticated and the amount of e-money sent to the client
  - **401** - *USER NOT AUTHENTICATED* - user is not authenticated. Wrong user name or password
  - **421** - *AUTHENTICATION E-MONEY ERROR* - an error occurred while performing the user authentication
- **E-Money** - the amount of e-money user has in his/her account. In case the operation status code is 401 or 421, no e-money value is set by the server in the response object.

5.3.2 Movie Goer Changes the Application Password

A change password request can be made during the client-server communication. This is depicted in fig. 5.5. The protocol can be summarized in the following steps:

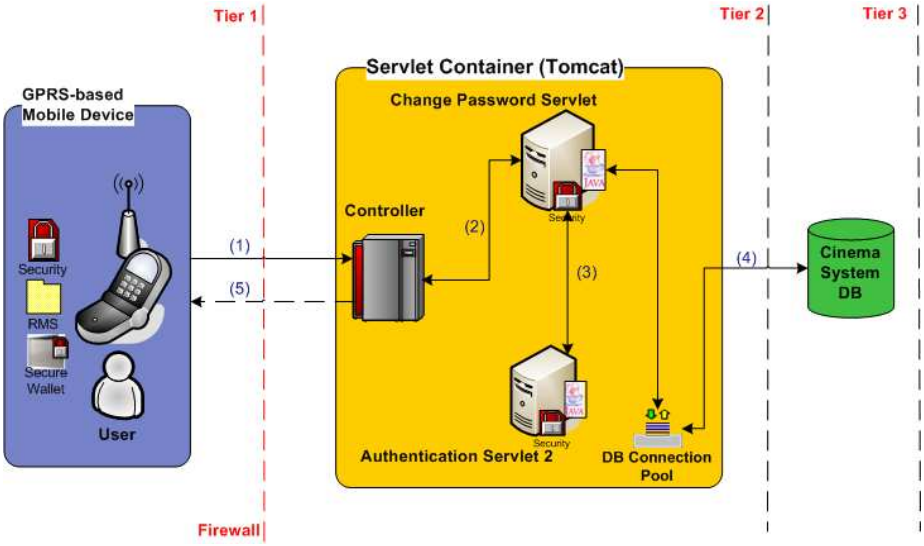


Figure 5.5: The Communication Protocol for Changing Movie Goer's Password

1. The movie goer is authenticated and the *Main Menu* is displayed on the mobile device screen.

2. The movie goer would like to change the application password and selects *My Settings* option in the *Main Menu*.

3. The *My Settings* main screen is displayed and the movie goer selects the *Change Password* option. User enters his *user name*, *old password*, *new password*, and the *verify password* and presses the *Change Password* button. The request is sent to the *Cinema Central Controller Servlet* over GPRS. (1)

4. The central controller checks the type of request and forwards the request to the corresponding worker servlet i.e. *Change Password Servlet*. (2) The worker servlet checks the movie goer's key with the *Authentication Servlet 2*. (3) If the key is valid and the supplied credentials correct, it and updates his/her password in the DB. (4)

5. The worker servlet sends a server side status code back to the mobile device and the corresponding info message is displayed on the screen. (5)

The following data is sent to the server as a request object during the POST request:

- **user name**
- **old password**
- **new password**
- **verified password**

The following data is received from the server as a response object to the previous made request:

- **Operation Status Code** - one of the following:
  - **202** - *PASSWORD CHANGED* - user is authenticated and the password changed in the system
  - **402** - *PASSWORD NOT AUTHENTICATED* - user is not authenticated. Wrong user name or password

5.3.3 Movie Goer Searches For a Movie

A search movie request can be made during the client-server communication. This is depicted in fig. 5.6. The protocol can be summarized in the following steps:

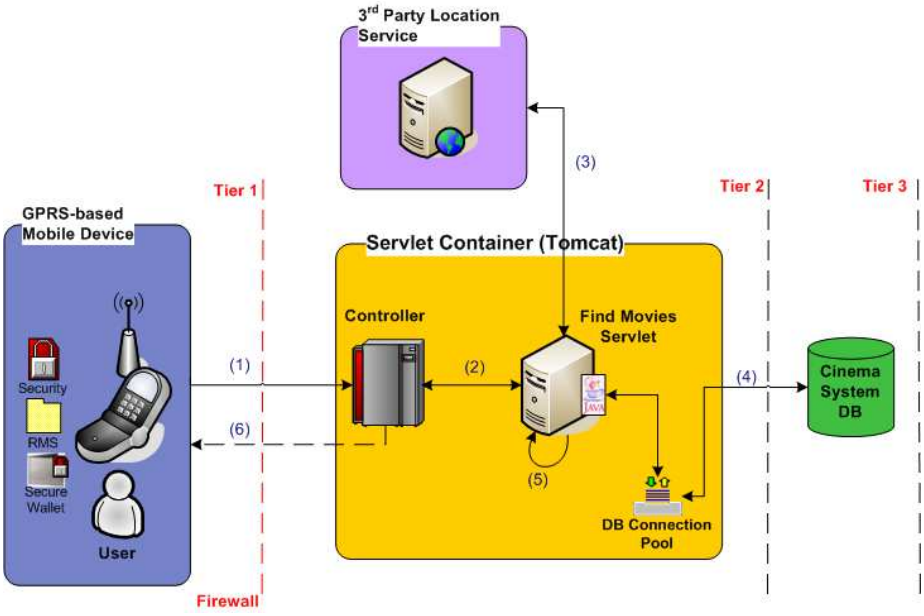


Figure 5.6: The Communication Protocol for Searching Movies

1. The movie goer is authenticated and the *Main Menu* displayed on the mobile device screen.
2. He/she would like to place a search movie request. He/she selects the *Find Movies* entry in the *Main Menu*.
3. The *User Location* screen is displayed and the movie goer enters his/her current position i.e. *street*, *zip*, *city*, *range*, and *date* and presses the *Search* button. The request is sent to the *Cinema Central Controller Servlet* over GPRS. (1)
4. The central controller checks the type of request and forwards the request to the corresponding worker servlet i.e. *Find Movies Servlet*. (2)
5. The worker servlet contacts a *third party cinema location service* e.g. *krak.dk* and returns all cinemas in the given range from the user. (3)

6. The worker servlet computes a list of all movie for the previous returned cinema list by the 3rd party location service. This is done by searching against the DB for all shows played in the given cinemas. (5)
7. The worker servlet returns a status code together with the list of shows that fulfills users' searching criteria. (6)
8. A new screen is opened and the list of shows displayed i.e. *movie name, cinema, show hour, and cinema info*. User browses through the list and selects a show. He can *Reserve Seats for a show*, *View Details about a movie*, *Refine searching criteria*, or *Rate a movie*.

The following data is sent to the server as a request object during the POST request:

- **movie**
- **street**
- **city**
- **zip**
- **range**
- **date**

The following data is received from the server as a response object to the previous made request:

- **Operation Status Code** - one of the following:
  - **205** - *FIND MOVIES CRITERIA 1 MOVIES FOUND* - movies found accordingly to the given searching criteria
  - **405** - *FIND MOVIES CRITERIA 1 MOVIES NOT FOUND* - movies not found accordingly to the given searching criteria
  - **206** - *FIND MOVIES CRITERIA 2 MOVIES FOUND* - movies found accordingly to the given searching criteria
  - **406** - *FIND MOVIES CRITERIA 2 MOVIES NOT FOUND* - movies not found accordingly to the given searching criteria
  - **207** - *FIND MOVIES CRITERIA 3 MOVIES FOUND* - movies found accordingly to the given searching criteria

  - **407** - *FIND MOVIES CRITERIA 3 MOVIES NOT FOUND* - movies not found accordingly to the given searching criteria
  - **208** - *FIND MOVIES CRITERIA 4 MOVIES FOUND* - movies found accordingly to the given searching criteria
  - **408** - *FIND MOVIES CRITERIA 4 MOVIES NOT FOUND* - movies not found accordingly to the given searching criteria
  - **209** - *FIND MOVIES CRITERIA 5 MOVIES FOUND* - movies found accordingly to the given searching criteria
  - **409** - *FIND MOVIES CRITERIA 5 MOVIES NOT FOUND* - movies not found accordingly to the given searching criteria
  - **419** - *MOVIE LOCATION SERVICE ERROR* - an error occurred during the communication with the 3rd party cinema location service
  - **420** - *MOVIE LOCATION SERVICE NO DATA* - no cinemas can be found by the the 3rd party cinema location service matching user's given position

- **no. of found movies**
- **movies** - a list of all found movies containing the following details: *movie Name, hour, cinema, city, street, showLocationID, showTimeID*

5.3.4 Movie Goers Reserve/Purchase Tickets

A reserve/purchase movie tickets request can be made during the client-server communication. This is depicted in fig. 5.7. The protocol can be summarized in the following steps:

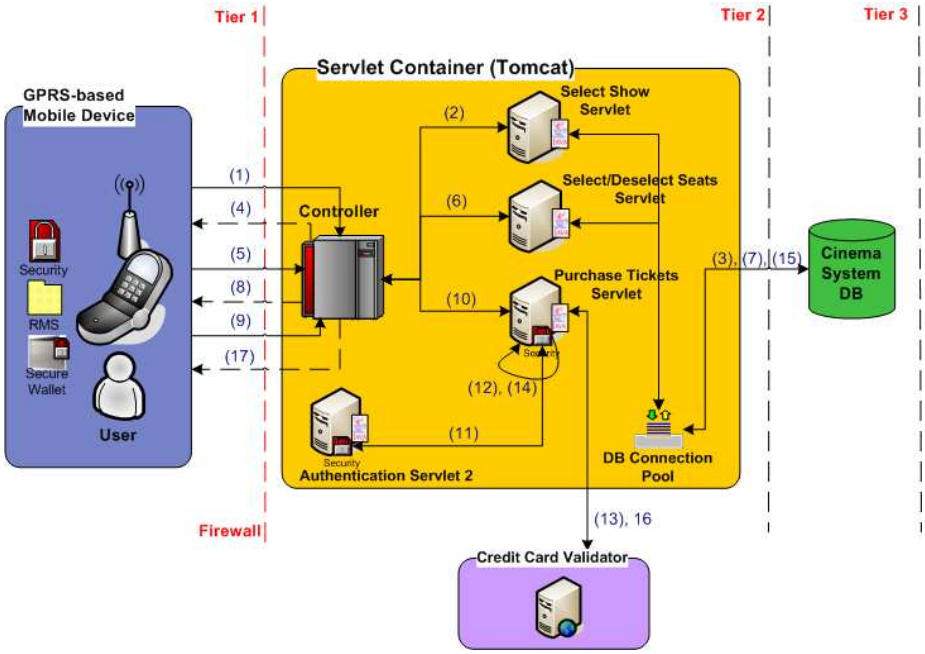


Figure 5.7: The Communication Protocol for Reserving/Purchasing Cinema Tickets

1. The movie goer is authenticated. A *Find Movie request* has just been placed. A list of all shows is displayed on the mobile device.

2. The movie goer selects one of the shows and presses the *Select Seats* button. The request is sent to the *Cinema Central Controller Servlet* over GPRS. (1)

3. The central controller checks the type of request and forwards the request to the corresponding worker servlet i.e. *Select Show Servlet*. (2)

4. The worker servlet performs a search against the DB for the cinema theater configuration (i.e number of seats, rows, free or booked seats), ticket base price, and discount values. (3)

5. If the show is found the requested data together with a status code is sent back to the mobile device. (4)

6. The cinema theater configuration is displayed graphically on the mobile device i.e. a matrix of rows x seats with seats colored in red(already booked) and green(free). A cursor allows movie goers to jump from one seat to another and select/deselect the desired seats. Once the seat is selected it is highlighted in blue i.e. user current selection.

7. The movie goer can select as many seats as needed. Once the seats selected and the *Select* button pressed, a request is sent to the *Cinema Central Controller Servlet* over GPRS. (5)

8. The central controller checks the type of request and forwards the request to the corresponding worker servlet i.e. *Select Deselect Seats Servlet*. (6)

9. The worker servlet performs a search against the DB to check if the selected seats are still free or not.[1] If the seats are free, they are marked as booked in the DB. (7)

10. A status code together with the cinema theater updated configuration is sent back to the mobile device. (8) This can be used to display the configuration screen in case user's selected seats have been reserved in the meanwhile.

11. A *Ticket Summary and Discount* screen is displayed if the seats are booked successfully. The movie goer can see details about the reserved tickets, the price for each ticket, the total price to be payed, and the discount type. The user can choose any ticket discount information that he/she is entitled to. The user can choose to select any discounts or not.

12. If the *Accept* button is pressed a *Ticket Payment screen is displayed*. It allows users to select the desired payment method to pay for the reserved tickets as mentioned in section 4.5.3.

13. Once the movie goers presses the *Purchase* button a secure request is built by encrypting the data to be sent using the key received during the authentication and sent to the *Cinema Central Controller Servlet* over GPRS. (9)

14. The central controller checks the type of request and forwards the request to the corresponding worker servlet i.e. *Purchase Tickets Servlet*. (10)

15. The worker servlet talks to the *Authentication Servlet 2* and verifies the user's key. (11) If the key is valid, the user is authenticated and the

[1] It might happend that 2 users are reserving seats in the same time and both of them would like to reserve the same seats

transaction continues. If not, an error message is returned and no money withdrawn from user's credit card.

16. If user is authenticated, the worker servlet decrypts the request and verifies the payment method. (12)
17. If the payment method is *credit card* the worker makes a *Check Credit Card Validity* request to a 3rd party trusted service such as Pay-Pal. (13)
18. The 3rd party credit card validation service checks if the credit card is valid or not and returns a response to the worker servlet.
19. If the card is valid, the worker servlet generate a unique *Reservation ID* and unique *Ticket ID's*. (14) It makes the reservation persistent in the DB, marks the reservation as *Payed* (15), and withdraws the money for the tickets via the 3rd party trusted credit card validation service. (16) No credit card info is saved on the Server Side Cinema Service.
20. If the payment method is *at the cinema* the worker servlet makes the reservation permanent in the DB and marks it as *Not Payed*. (15)
21. A status code together with the reservation ID, ticket ID's, ticket details, total payed price, and left e-money are sent back to the mobile device by the worker servlet. (17)
22. A *Billing Details* screen is displayed on the mobile device containing all previous mentioned data. The reservation ID, ticket ID's and left e-money are stored in the movie goer's mobile device.

Three requests are made from the client to the server during the purchase tickets protocol step i.e. *reserve seats*, *purchase tickets*, and *reject payment*. The last request is made in case the purchase tickets operations is canceled by the user. In order to preserve the integrity of the cinema system, the later request is made.

The Reserve Seats Request
The following data is sent to the server as a request object during the reserve seats POST request:

- **command type** - select or deselect seats
- **showLocationID** - key in the DB to identify the show. The cinema theater can be determine via the show
- **showTimeID** - key in the DB to identify the show. The cinema theater can be determine via the show

- **seats** - the seats selected by the user

The following data is received from the server as a response object to the previous made request:

- **Operation Status Code** - one of the following:
  - **210** - *SEATS SELECTED OK* - seats selected successfully
  - **211** - *SEATS DESELECTED OK* - seats deselected successfully
  - **410** - *SEATS SELECTED ERROR*- an error occurred while selecting the seats
  - **411** - *SEATS DESELECTED ERROR* - an error occurred while deselecting the seats
- **booked seats** - movie goer booked seats by the server

The Purchase Tickets Request
The following data is sent to the server as a request object during the purchase tickets POST request:

- **user name** - the user name of the movie goer to access the system
- **password** - the password of the movie goer to access the system
- **showLocationID** - key in the DB to identify the show
- **showTimeID** - key in the DB to identify the show
- **seats** - the booked seats
- **discounts** - the discount types for all selected seats
- **creditCardType** - the credit card type used for payment. If no credit card is used this value is empty.
- **creditCardNo** - the credit card no used for payment. If no credit card is used this value is empty.
- **creditCardExpDate** - the credit card expiring date used for payment. If no credit card is used this value is empty.
- **creditCardCW2** - the credit card security code used for payment. If no credit card is used this value is empty.

- **reservationDate** - the reservation date
- **purchaseMethod** - the payment method

The following data is received from the server as a response object to the previous made request:

- **Operation Status Code** - one of the following:
  - **212** - *PURCHASE TICKETS OK* - the tickets have been purchased
  - **401** - *USER NOT AUTHENTICATED* - the user cannot be authenticated be the server side. Wrong user name or password.
  - **412** - *PURCHASE TICKETS ERROR* - an error occurred while trying to purchase the tickets
  - **413** - *PURCHASE TICKETS INVALID CREDIT CARD* - movie goer's credit card is invalid
- **reservation ID** - the current reservation ID
- **total Price** - the total amount payed for the tickets
- **E-money** - user's e-money
- **ticket IDs** - the ticket IDs'
- **ticket Prices** - the prices for all tickets

The Reject Payment Request
The following data is sent to the server as a request object during the reject payment POST request:

- **showLocationID** - key in the DB to identify the show
- **showTimeID** - key in the DB to identify the show
- **seats** - the seats reserved by the user

The following data is received from the server as a response object to the previous made request:

- **Operation Status Code** - one of the following:

- **216** - *REJECT PAYMENT OK* - the previous made user reservation is canceled
- **416** - *REJECT PAYMENT ERROR* - an error occurred while trying to cancel user's reservation

5.3.5 Movie Goers Cancel Tickets

A cancel ticket request can be made during the client-server communication. This is depicted in fig. 5.8. The protocol can be summarized in the following steps:

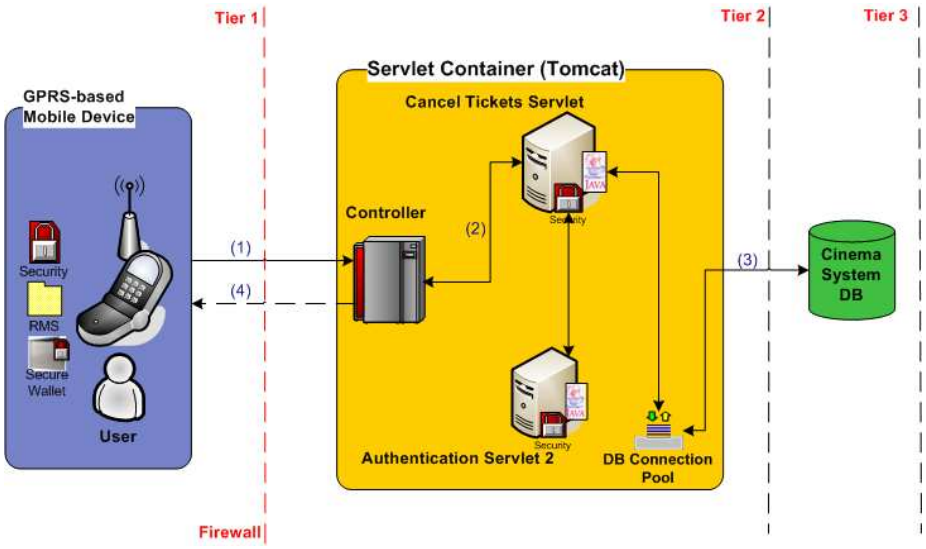


Figure 5.8: The Communication Protocol for Canceling Tickets

1. The movie goer is authenticated and the *Main Menu* is displayed on the mobile device screen.
2. He/she would like to cancel a previous purchased ticket and selects *My Tickets* entry in the *Main Menu*.
3. A list of all reserved/purchased tickets is displayed. The movie goer selects a ticket and presses the *Cancel Tickets* button.[2]. The cancel tickets request is sent to the *Cinema Central Controller Servlet* over GPRS. (1)
4. The central controller checks the type of request and forwards the request to the corresponding worker servlet i.e. *Cancel Tickets Servlet*. (2)
5. The worker servlet deletes the ticket in the DB and any other reservation information; it updates the amount of user's e-money with the total amount of money payed for the canceled ticket. (3)

[2]Only tickets payed by credit card can be canceled. The tickets reserved using the *pay at the cinema* payment method are canceled automatically by the system within 30 min before the show if they are still unpaid

6. The worker returns a status code together with the total amount of e-money to the mobile device. (4)
7. A *Tickets Canceled!* message is displayed on the mobile device and the ticket list is updated together with the amount of e-money. An error message is displayed otherwise.

The following data is sent to the server as a request object during the POST request:

- **user name** - the movie goer's user name to access the system
- **password** - the movie goer's password to access the system
- **reservation ID** - the reservation ID to be canceled
- **ticket IDs** - the ticket IDs to be canceled

The following data is received from the server as a response object to the previous made request:

- **Operation Status Code** - one of the following:
  - **214** - *CANCEL TICKETS OK* - the tickets are canceled successfully
  - **414** - *CANCEL TICKETS ERROR* - an error occurred while trying to cancel the tickets

5.3.6 Movie Goer Rates a Movie

A rate movie request can be made during the client-server communication. This is depicted in fig. 5.9. The protocol can be summarized in the following steps:

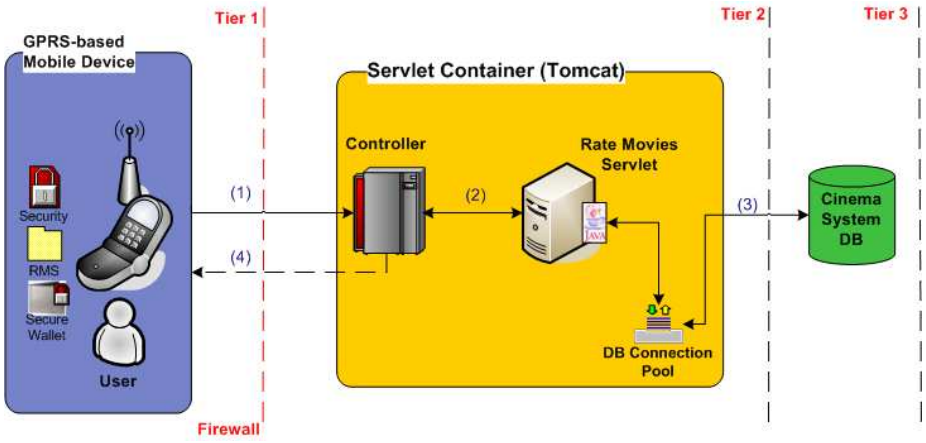


Figure 5.9: The Communication Protocol for Rating a Movie

1. The movie goer is authenticated and a *Find Movie request* has just been made. The list of all shows is displayed on the mobile device.

2. The movie goer selects one of the shows and chooses the *Rate Movie* entry in the *Option* menu. A new screen is displayed. The movie goers selects a mark from 1 to 10 for that movie and presses the *Rate Movie* button. The request is sent to the *Cinema Central Controller Servlet* over GPRS. (1)

3. The central controller checks the type of request and forwards the request to the corresponding worker servlet i.e. *Rate Movie Servlet*. (2)

4. The worker servlet performs a search against the DB and verifies the user name and credentials. If the user is authenticated it insert or updates the rating score for that movie.[3] (3)

5. A status code is returned to the mobile device and a *Movie Rated Successfully* message is displayed. An error message is displayed otherwise. (4)

[3] A movie goer is not allowed to have more than one rating for the same movie

The following data is sent to the server as a request object during the POST request:

- **user name** - the movie goer's user name to access the system
- **password** - the movie goer's password to access the system
- **showLocationID** - the show ID in the DB
- **movie score** - the rating given by the movie goer

The following data is received from the server as a response object to the previous made request:

- **Operation Status Code** - one of the following:
  - **218** - *RATE MOVIE OK* - the movie is rating successfully
  - **418** - *RATE MOVIE ERROR* - an error occurred while rating the movie

5.3.7 Movie Goers View Movie Details

The communication between the client and server when the movie goer makes a view movie request is depicted in fig. 5.10. The protocol can be summarized in the following steps:

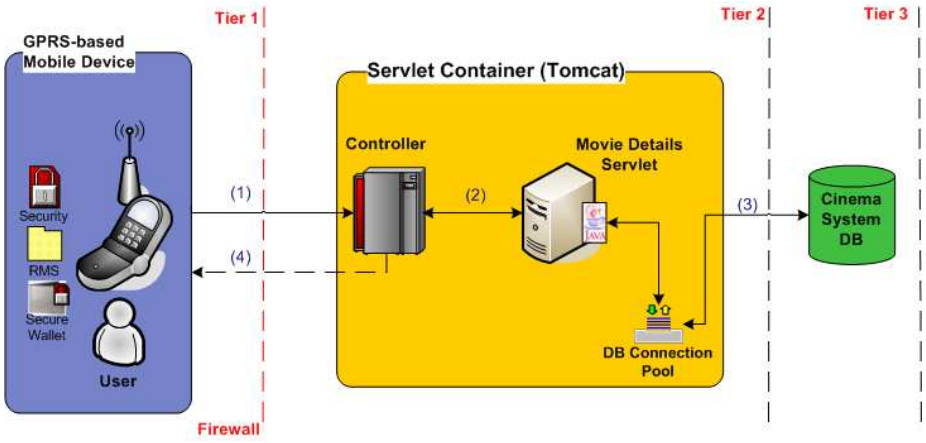


Figure 5.10: The Communication Protocol for Viewing Movie Details

1. The movie goer is authenticated and a *Find Movie request* has just been made. The list of all shows is displayed on the mobile device.

2. The movie goer selects one of the shows and chooses the *Movie Details* entry in the *Option* menu. A get movie details request is sent to the *Cinema Central Controller Servlet* over GPRS. (1)

3. The central controller checks the type of request and forwards the request to the corresponding worker servlet i.e. *Movie Details Servlet*. (2)

4. The worker servlet performs a search against the DB (3), and returns the movie details corresponding to the requested movie together with a status code to the mobile device. (4)

5. The selected movie details are displayed graphically on the mobile screen i.e. *Movie Name*, *Genre*, *Duration*, *Parent Classification*, *Language*, *Year*, *Country*, *User Rating*, *Description*, *Director*, *Actors*, and *the Movie Poster*.

The following data is sent to the server as a request object during the POST request:

- **showLocationID** - the show ID in the DB. The movie ID can be determined from the show ID.

The following data is received from the server as a response object to the previous made request:

- **Operation Status Code** - one of the following:
  - **217** - *MOVIE DETAILS OK* - the movie details OK
  - **417** - *MOVIE DETAILS ERROR* - an error occurred while trying to retrieve the movie details

5.3.8 Background Cinema Theater Update

The communication between the client and server during the background cinema theater configuration update is depicted in fig. 5.11. The protocol can be summarized in the following steps:

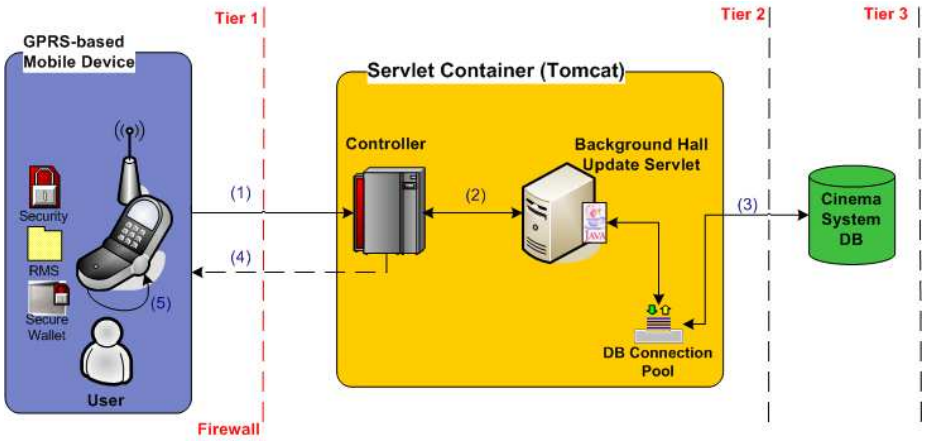


Figure 5.11: The Communication Protocol for Background Cinema Theater Configuration Update

1. The movie goer is authenticated, and he/she has just selected a show to book tickets for. The cinema theater configuration is displayed on the mobile device.

2. Every 15 seconds, a background thread sends request to the server side for obtaining the cinema theater configuration. A get cinema theater configuration request is sent to the *Cinema Central Controller Servlet* over GPRS. (1)

3. The central controller checks the type of request and forwards the request to the corresponding worker servlet i.e. *Background Hall Update Servlet*. (2)

4. The worker servlet performs a search against the DB (3), and returns that particular cinema theater configuration together with a a status code to the mobile device. (4)

5. The current cinema theater configuration display is updated and the customer can see any new booked seats in the meanwhile. (5)

The following data is sent to the server as a request object during the POST request:

- **showLocationID** - key in the DB to identify the show. The cinema theater can be determine via the show
- **showTimeID** - key in the DB to identify the show. The cinema theater can be determine via the show

The following data is received from the server as a response object to the previous made request:

- **Operation Status Code** - one of the following:
  - **204** - *UPDATE SHOW FOUND* - cinema theater configuration updated successfully
  - **404** - *UPDATE SHOW NOT FOUND* - an error occurred while updating the cinema theater configuration
- **booked seats** - all booked seats in the selected cinema theater

5.4 Securing the Communication between the client and the server

Statement from the author: Some parts of the design and implementation of the security protocol has been reused from a previous project in Secure Mobile Services (course no. 34632). Two persons worked in equal amounts on that project during the Secure Mobile Services course. The author was one of them. The code fragments reused or adapted from that project are strictly marked with the name of both authors.

Security is very important in the Cinema Ticket Reservation System. An authentication protocol is designed to provide strong security. Two solutions are proposed and analyzed by following the requirements specified in section 3.

The first solutions provides acceptable security but it has some important minuses. The second solution provides a solution to all four security requirements as mentioned in section 3 i.e. *Data Integrity*, *Confidentiality*, *Availability* and *Non-repudiation*. Both protocols are built on the concept of *Single Sign On server*. The second solution is implemented.

There are three parties involved in the protocols:

Trent - the first authentication server, trusted arbitrator in the protocol.

Alice - user connecting from the mobile terminal.

Bob - the second authentication server that checks further request from the user using the preshered key with Trent.

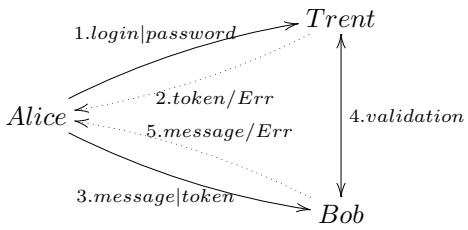


Figure 5.12: Simple Authentication protocol, using Single Sign On server.

5.4.1 Single Sign On Server with the use of two preshered keys

The protocol consists of the following steps:

1. **Alice** is sending an encrypted challenge to **Trent** . The challenge contains the login and password.
2. **Bob** is sending an encrypted response to **Alice** . The response contains a serial number of the user and the one time password, together known as a *token*.
3. **Alice** is sending an encrypted message to **Bob** . This message is already a part of their intended communication. The message is encrypted with a different key than the one used in the authentication step. Message contains the *token* and a payload.
4. **Bob** is authenticating **Alice** by challenging the **Trent** . Authentication server responds with a short message indicating the success or the failure of authentication.
5. **Bob** sends a response to **Alice** . It contains either a response data or information about failure of the authentication.

The protocol is visualized in figure 2.

The protocol, consists of several security threats, relevant from the cryptographical point of view as well as from the user point of view. In the section 3, it is explained that the key used in the encryption of the messages should be represented as a sufficiently long string. Two keys are required, one for the encryption of the authentication and the other one for encryption of the communication between user and service provider. Presharing of two keys increases the complexity of the solution on the user side.

Moreover, the following security cons are present in the protocol:

- Keys used for the encryption of messages can't be very long from the user point of view i.e. strings provided by the users. They are used in the generation of the valid key objects physically used for the encryption. Those keys are therefore considered as weak. Moreover, they are not generated randomly, but reused. Even in case of the use of symmetric ciphers proposed in section 3, weak strings used in the generation of the key objects used for encryption make them much weaker. Strings are built of ASCII characters what introduces additional repeatability.
- If one of the keys is compromised, the whole system becomes compromised.
- If login and password are compromised, then the whole system again becomes compromised.
- If the protocol used for the communication with **Bob** is known, then the attack on the encrypted messages becomes a partially known plain-text attack. Together with the weak key it makes it breakable. Therefore we can't say that the protocol provides full *confidentiality*.
- The protocol is vulnerable to the replay attacks. If the message is intercepted in the step 3, then it may be reused. In case of the system used for the control of industrial robot, it may cause the serious harm. It is possible to partially mitigate the threats by the introduction of serial numbers of messages sent by the protocol between the user and service provider. The authentication part of the protocol would still be vulnerable. If an attacker sends an authentication challenge again to its destination, then the one time password would be changed, preventing accredited user form accessing the service. Therefore the protocol do not grant the *availability* requirement to the system.

Depending on the security level required, the protocol provides a sufficient security level.

5.4.2 Single Sign On Server with the use of Needham-Schroeder Protocol

The use of Needham-Schroeder concept, improves the security and usability of the concept introduced in section 5.4.1. The following protocol is an interpretation of this concept applied to the project purpose.

The protocol consists of the following steps:

1. **Alice** sends to **Trent** a challenge consisting of the following elements: login and password (A), address of the targeted system (B) and a random number (R_a).
$$A, B, R_a$$

2. **Trent** checks the credentials of **Alice** in the local data base. He grabs the user serial number and one time password (*token*). **Trent** also checks whether the random number received, have not been used before. He generates a salt value - reasonably large number of byte array. The salt value would be used on the client side and on the Service side for creation of the session key object. **Trent** encrypts the random salt value (S) and the *token* (T) with the key preshared with **Bob** (E_b). Then he encrypts the random number received from **Alice** (R_a), address of the service provider (B) ,the salt value (S), the serial number of **Alice** (S_t) and message encrypted with *Bob's* key, with the key preshared with **Alice** . He sends the message back to **Alice** .
$$E_a(R_a, B, S_t, S, E_b(S, T))$$

3. **Alice** decrypts the message and extracts the Salt value (S) and her serial number (S_t). She confirms that R_a is the same random number that she sent to **Trent** in the step 1. Then she sends to **Bob** the message that **Trent** encrypted for him.
$$E_b(S, T)$$

4. **Bob** decrypts the message and extracts the Salt value and the *token* sent previously by **Alice** . He confirms that Alice credentials are correct by communicating with **Trent** .

5. If the credentials of **Alice** are correct then **Bob** generates a *session key* object with the use of the salt value and Serial number of the user, contained in the token. He generates another random number. He encrypts the number with the session key. He sends the result to **Alice** .
$$E_k(R_b)$$

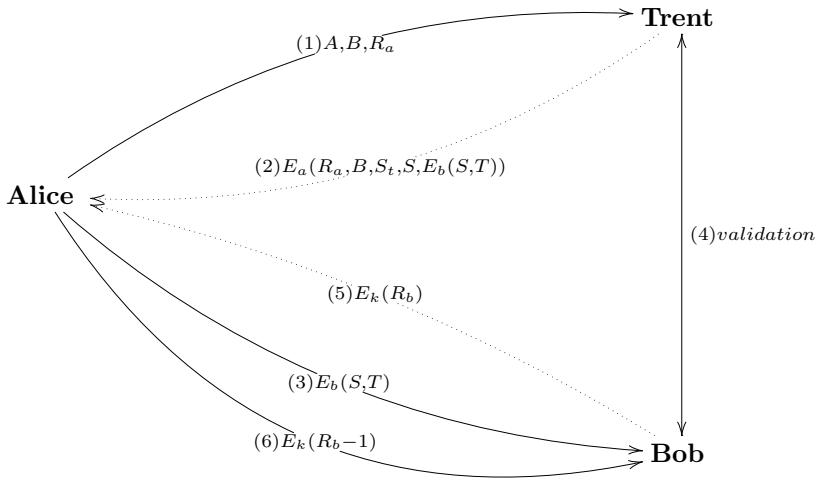


Figure 5.13: Authentication with the use of Single Sign On Server and Needham-Schroeder Protocol

6. **Alice** receives the message from Bob. She creates the session key from the salt value and serial number she received from **Trent** . She decrypts the random number that was sent by **Bob** . She subtracts one from it, encrypts it once again and sends back to **Bob** .

$$E_k(R_b - 1)$$

The last step of the protocol is there in order to prevent the replay attacks. The protocol is at best visualized in figure 3.

The above protocol is implemented. It contains several improvements in relation to the previous protocol.

- The protocol is not vulnerable to the replay attacks during the communication with **Trent** . Moreover the inclusion of the random number in the message makes it practically impossible to decrypt, even if the preshared key used at this stage is weak.
- All three factors of the authentication must be compromised in order to compromise the system. Attacker must know the user name, password and a key preshared between **Trent** and **Alice** .
- The session key is used only for one communication session. This is enforced by the checking the random password from the *token* with **Trent** .

- The session key is 128 bit key. It is generated from **securely random** salt value and a serial number of **Alice** . The session key may be considered as *strong*. The algorithm used is *AES*. Therefore it may be stated that even if attacker knows plain contents of the message, he is not able to perform a successful attack on the encrypted message.
- All the messages involved in the communication between **Alice** and **Bob** contain a timestamp. Sufficient approximation of this timestamp should be a serial number. It prevents from the replay attacks.
- Authentication of each message by **Bob** is no longer necessary. Possession of the session key by **Alice** is sufficient proof of her authenticity.

The above protocol provides all four security requirements stated in section 3 i.e. *Data Integrity*, *Confidentiality*, *Availability* and *Non-repudiation*.

Chapter 6

Cinema Ticket Reservation System Implementation

This section depicts the implementation process of the Cinema Ticket Reservation System i.e. mobile client, server side services, and database. The following technologies are used for implementation purposes: *J2ME, J2EE (Servlets, Java Beans, JDBC), J2SE, Cryptographic Libraries(The Legion of Bouncy Castle), postgreSQL, and stored procedures.* Both the client and server implementations follow several *design patterns* that provide a fast and optimized code, such as: *MVC, Facade, Controller, Singleton, Template Method, Refactoring, Abstract Coupling, etc.* The implementation details of the final solution proposed in sections 2.3, 4.5, and 5 are depicted in the following chapters.

Several use cases are identified in the overall system architecture. They are depicted in fig. 6.1 as ellipses. Several actors are also identified i.e. *the movie goer, the mobile device inner clock, the credit card validating service, the cinema server side service.* They are the ones performing the use cases. The connection between the actors and use cases is depicted with a continuous line from the actor to the ellipse(use case). A line with an arrow at one of ends indicate that the use case the arrow points at is used by the use case that the arrow comes from e.g. *a cancel reservation use case uses the authenticate use case (user must be authenticated before canceling a ticket)*

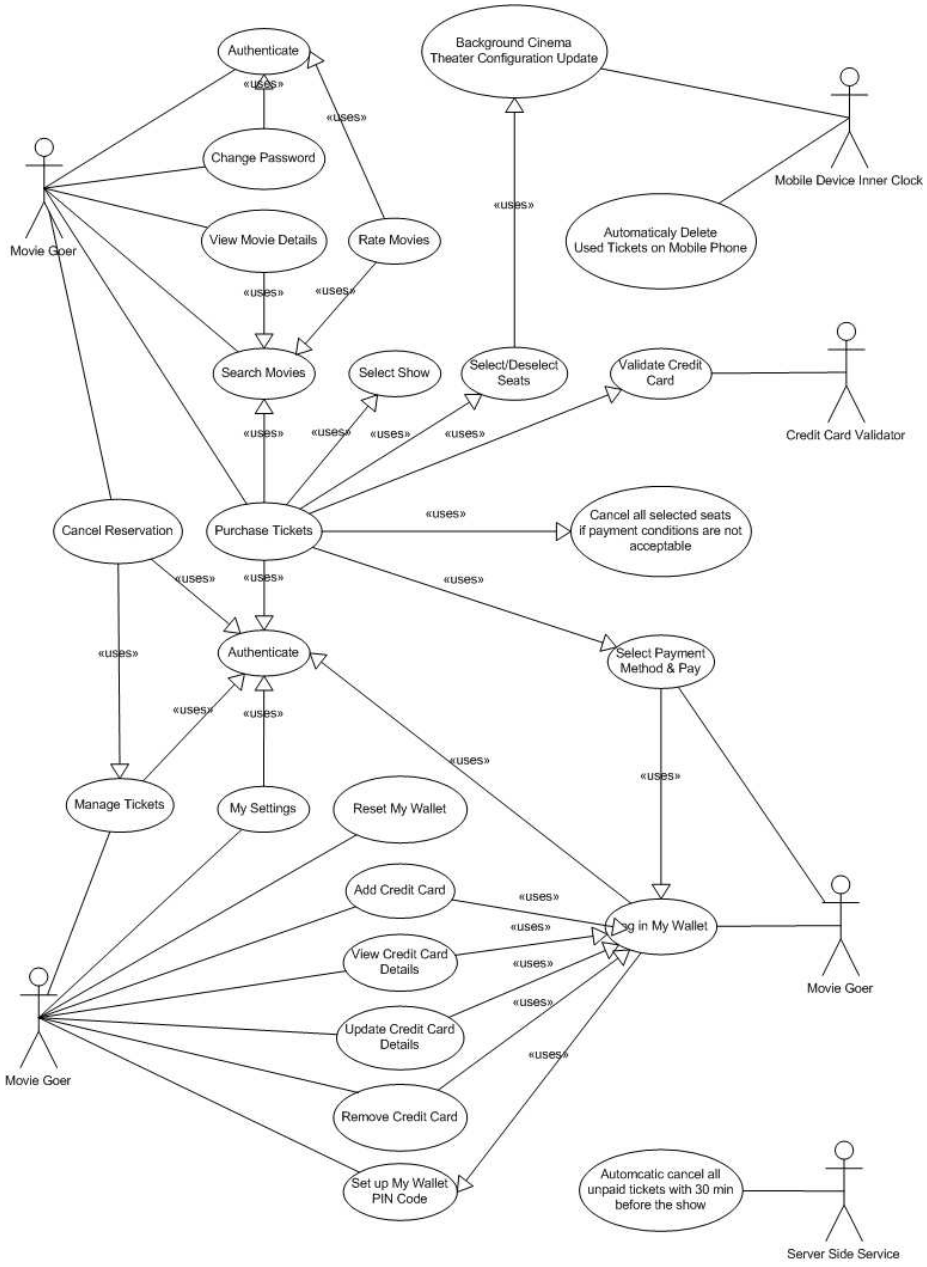


Figure 6.1: The Use Cases Identified for the Cinema Ticket Reservation System

6.1 Technologies used for the Cinema Service Implementation

Several technologies such as J2ME, J2EE(Java Servlets, JDBC), SQL, PL/SQL, etc are used for implementing the cinema ticket reservation service. They are shortly depicted in this chapter.

J2ME
J2ME(Java 2 Micro Edition) is the Java platform aimed squarely at consumer devices with limited horsepower. J2ME offers a very flexible and robust platform for developing mobile applications on small devices and a great deal of mobility to the consumer devices i.e. browsing, downloading, and installing Java application. Before J2ME all consumer devices where static by their nature. There is a broad range of consumers and embedded devices that use J2ME e.g. PDAs, mobile phones, card readers, etc and J2ME tries to accommodate all of them.[17] J2ME contains a lightweight JVM, a minimum set of core classes, and lightweight substitutes for standard Java libraries.[23]

In order to support the broad range of PDAs', enhanced mobile phones, and other consumer devices, *Configurations* and *Profiles* are introduced. The configuration defines a particular JVM, language features, and libraries. It defines which devices can use certain aspects of the language and it applies to all parts of the devices e.g. CPU, screen, networking, memory, etc. A profile is an extension of a configuration providing development libraries for a specific device. There are two configuration supported by J2ME i.e.

CDC(Connected Device Configuration)

- 512 kilobytes (minimum) memory for running Java[23]
- 256 kilobytes (minimum) for runtime memory allocation[23]
- used by home devices and embedded devices that have possible persistent network connectivity and bandwidth[23]

CLDC(Connected Limited Device Configuration)(this is the chosen configuration for implementing the cinema service)

- 128 kilobytes memory for running Java[23]
- 32 kilobytes memory for runtime memory allocation[23]

- it is the mostly used configuration my devices with limited memory and computation power, limited wireless network connectivity e.g. battery operated mobile devices

MIDP
Provides lightweight libraries for implementing J2ME applications on mobile devices such as enhanced mobile phones by providing access to *GUI components* and *Record Management System(persistent data storage)*. J2ME applications written using the MIDP profile are called *MIDlets*. The MIDlet lifecycle is depicted in fig. 6.2

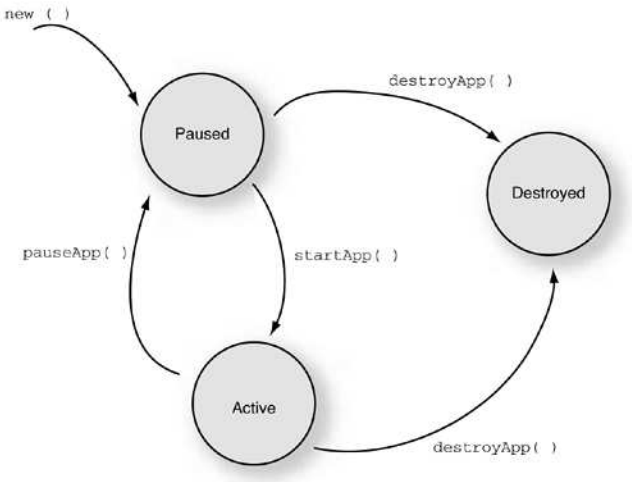


Figure 6.2: MIDlet Lifecycle[12]

There are two types of GUI components introduced by MIDP i.e. *low-level* and *high-level* depicted in fig. 6.3. The GUI implemented in the Mobile Cinema Service is using both low and high-level J2ME components.

The *low-level components* are extended from the *Canvas* class and offers more means of customizing the look and feel of the application GUI via colors, fonts, drawings, etc.

The *high-level components* are extended from the *Screen* class and contains *Alerts, Forms, Lists, and TextBoxes*. These and their subclasses are the most common components one can use to build a simple GUI. In case a customized approach is needed, one has to use the low-level GUI.

The *Record Management System*

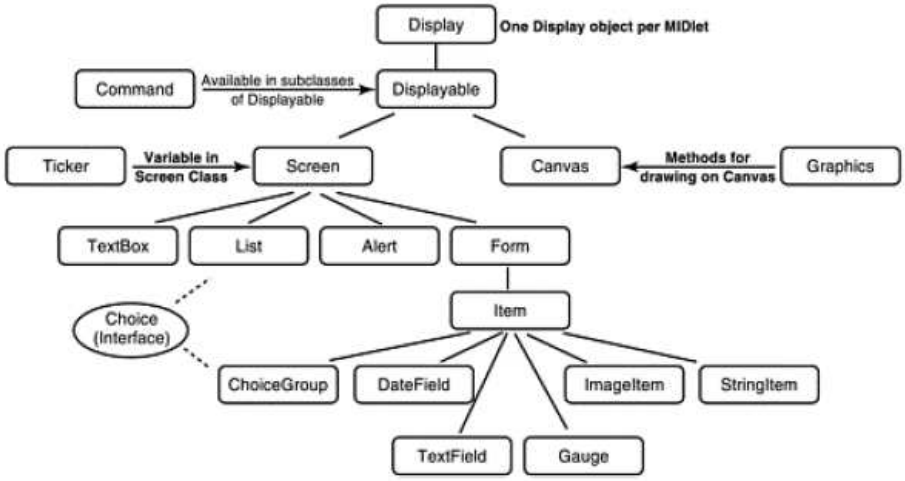


Figure 6.3: GUI components and the hierarchy among them[17]

RMS is a persistent data storage in the MIDlet profile via a *Record Store (RS)*. One can read, write, or search for data in the RS. A MIDlet can have zero or many RSs' but it can access only the RSs' it created. Data is stored inside a RS on a *record* base. A record store has the structure of a *HashMap* i.e. all records are stored on a key-value basis. The key is an integer number while the value is a byte array. A RS can be opened, closed, or deleted.

HTTP Java Servlets
A HTTP Java Servlet can be considered as a small extension to a Web Server that can be loaded dynamically and add portability and flexibility to the server. They are similar to any CGI scripts but they are more efficient and scalable. Servlets run in a servlet container that manages their state. Tomcat is used as a servlet container in the cinema service.

HTTP Java Servlet are extended from the *javax.servlet.http.HttpServlet* class. An HTTP servlet must override one of the following two methods:

- **doGet(HttpServletRequest request, HttpServletResponse response)**
  - serves GET request send by the client to the server
- **doPost(HttpServletRequest request, HttpServletResponse response)**
  - serves POST request send by the client to the server

The *request and response objects* represents the client request and the servlet

response, respectively. The *request* give access to parameters, headers, etc sent by the client. The *response* is used by the servlet to return data to the client as soon as the business logic is performed.

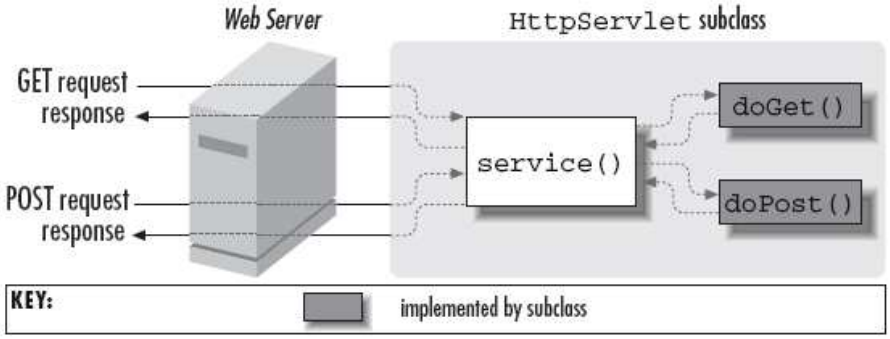


Figure 6.4: An HTTP servlet handling GET and POST requests[14]

JDBC(Java Database Connectivity)
JDBC opens a database-independent API that allows clients to execute SQL statements against a DB. Much of the functionality is provided by the DB-dependent driver that must be loaded by JDBC. There are several ways one can establish a connection to the DB i.e. *connection strings and connection pools*.

A JDBC Connection String is created to every time a query is made to the DB. The SQL statement is run and the connection disposed. When a new request comes, a new connection must be created. This approach reduces the overall system performance.

In case of a *Connection Pool*, the system performance increases because the web server creates a connection pool at application startup and for each request to the DB a connection that is already crated is retrieved from the pool, used to query the DB, and returned back to the pool for later requests. One can notice the increased performance of the system due to the fact that for each DB request an existing connection is used and not a new connection created every time.

A *PreparedStatement* can be used for SQL statements that are executed multiple times with different values.[21]. If a request is to be sent several times from the client side, a prepared statement can increase the system performance. The result returned by the statement is a *Result Set*. The desired data can be retrieved out of the result set and sent to the client.

6.2 Mobile Application Implementation

The mobile application is implemented using J2ME(networking, threads, low and high level UI components, data streams, RMS, etc.), Java Beans, and the cryptographic libraries from Bouncy Castle.

Several design patters such as *Singleton, MVC, template method, facade, abstract coupling, refactoring*, etc. are used to make the source code easy to read and update, and to increase the application performance. The mobile application source code can be found in *Appendix D1.*

The sequence diagrams describing the mobile client implementation are depicted in *Appendix C1.* The sequence diagrams for the server side service and the client-server communication protocol implementation can be found in *Appendix C2.*

6.2.1 The UI-Call Model

The UI model of the Mobile Cinema application is composed of several screens depicted in section 4.5. The navigation among these screens is implemented by means of menu selection, soft button actions, or links.

- **Splash Screen** - displayed at application startup
- **Authentication**
- **Main Menu**
- **Purchase Ticket screens**
  - **Movie Search**
  - **Select Show**
  - **Select Seats**
  - **Reservation Summary and Set Discount**
  - **Choose Payment**
  - **Billing Info**
- **Movie Details**
- **Movie Rating**
- **Ticket Manager screens**
  - **Ticket Manager Main Menu**
  - **Ticket Viewer**
- **My Wallet screens**
  - **My Wallet Authentication**
  - **My Wallet Main Menu**
  - **Add New Credit Card**
  - **Edit Credit Card**
  - **View Credit Card**
- **Settings screens**
  - **Change Application Password**
  - **Change My Wallet PIN**

  - **Set Application Theme**

- **Application Main Help screen** - different help screens specific to different places in the applications
- **Low and high level message info screens** - used to inform the user about the result of any operation that takes place on the server side or locally.
- **Progress Bar screens** - used during network and memory operations

An abstract class *GenericGUI.java* is the super class for all *high level screens*. Two design patterns are used for implementing this behavior i.e. *abstract coupling and template method*. The *GenericGUI.java* class provides a static *Displayable* object i.e. *the screen* and a static *Display* where the *screen* is to be displayed on. It implements the *showScreen()* and *prepareScreen()* methods but delegates several abstract methods(hook methods) to be implemented by its subclasses such as:

- **getScreen()** - return the current screen object
- **initModel()** - initialize the current screen model. This is called before the view is created.
- **createView()** - creates the screen i.e UI components and append them to the screen.
- **updateView()** - updates the UI in case of command action or listeners.
- **commandAction(Command c, Displayable s)** - deals with different command actions such as navigating to another screen, submitting a request over the network, etc.

That way the screen model initialization, the view creation/update, or the command action implementation are defined by each of the concrete classes extending the *GenericGUI.java* class. The classes extended from the *GenericGUI.java* class can be seen in the mobile client class diagram in *Appendix C*.

Several *low level screens* are also created. The low level UI such as *Canvas* provides a more flexible way to create and customizable the look and feel of the application GUI. It provides access to the key events with connection to the key codes directly linked to the concrete keys on the device keyboard, allows to declare listeners, commands, and an abstract *paint()* method that must be implemented by its subclasses. All low-level screens created in the Mobile Cinema

extend the *Canvas* class and implement their own functionality for the *paint()* method. Any initialization operations are implemented in the class constructor, while the paint() method is responsible for drawing the canvas content. The *keyPressed(int keyCode)* method checks for the code of the mobile device pressed key and defines custom action for different keys such as: navigate to another screen, submit a request over the network, update the main menu look and feel, navigate through the main menu, exit the application, etc. The Canvas can be seen as an *Java Applet*.

Due to the lack of builtin low-level info message screens and dialog window components in J2ME, 4 types of customized info message screens are designed and implemented i.e. *yes-no dialog windows (DialogWindow.java)*, *OK dialog windows(CanvasAlert.java)*, *info windows(CanvasAlert.java)*, and *customized alerts(DialogWindow.java)*. All these customized components are extended from

Figure 6.5: Yes-No Dialog Windows

the Canvas and provide custom implementation for the *paint()* method. These dialog windows are split into 4 parts i.e. title, icon, message, and action buttons or links. The navigation between the action buttons or the links is always done using the left and right arrows, while the selection is implemented by the fire button on the keypad. Command actions are implemented in order to capture the events of the pressed key and perform the appropriate actions.

Yes-No Dialog Windows (fig. 6.5)
These are created to collect users' input to different actions such as removing a credit card from my wallet, canceling a ticket, exit the application, etc. User is presented with a Yes - No choice. If he/she chooses Yes, the action is executed and an info message is displayed informing the user about the status of the action (action completed successfully or failed). If user chooses No, the action

is canceled and the previous screen is displayed. The title, icon, message, and the action buttons or links are customized by instantiating the object with the corresponding constructor. A button rollover look and feel is implemented when user selects one of the action buttons. This is implemented by either swapping 2 images when a button is selected or by changing the font size used to display the link when that link is selected.

OK Dialog Windows (fig. 6.6) This type of window is used to display the

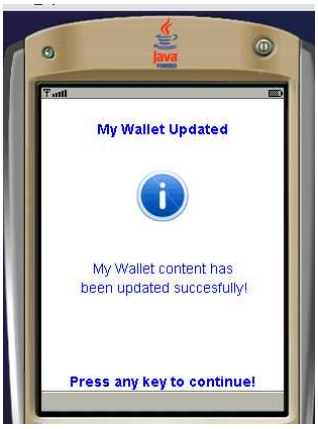


Figure 6.6: OK Dialog Windows

outcome of an action but the user must acknowledge the outcome e.g. the ticket payment is done and the user is informed about this. He/she must be able to see the result of this action and leave the screen only by pressing the select button. A single action button (OK) or link is used for this purpose. When the button is selected, the current screen is left and a new one displayed e.g. Main Menu, My Wallet Menu, etc.

Info Windows (fig. 6.7)
These types of windows are used to display an info message to the user for a fixed period of time e.g. 2 sec. A *Timer thread* is used in the background to count the no. of seconds passed from the time when the info window was displayed. When the given amount of time is reached, the info window is closed and a new window displayed on the mobile display. The display time is set up in the class constructor. There are 3 types of info windows used in the client application i.e. *info, warn, and alert.* A color code is used for displaying these alert windows. Blue for info, and red for warn and error. Different meaningful icons are used for each of the windows in order to suggest its meaning visually.

Info windows (fig. 6.7) are used for displaying an *action done* message. This is use in case the action is performed successfully. (Positive response to the Authentication procedure).

Figure 6.7: Info Windows

Warning windows (fig. 6.8) are used for displaying warning messages e.g. (The wallet is full. No more credit cards can be added.) *Error windows (fig. 6.8)* are

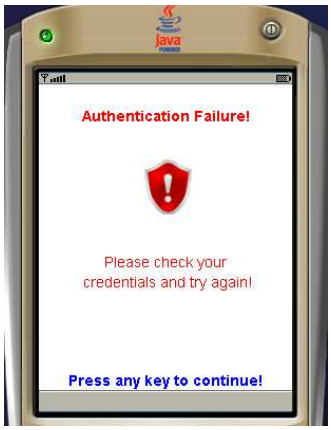


Figure 6.8: Warning and Error Windows

used in case an error occurs during a client or server side operation. These info windows are strictly connected to the *HTTP status code* sent from the server to the client side during a network operation and to the *operation result code* as

depicted in section 5.3. Based on the *HTTP status and operation result codes*, a customized info window (info, warn, or error) is displayed.

Customized Dialog Windows (fig. 6.9) These represents extensions of the *YES-NO Dialog Windows* that provide a wider choice of actions to the user.

Customized components for collecting user input
As mentioned before, this application is designed to provide a very user - friendly look and feel. Therefore, low-level components are preferred to be used due to the highly customization one can perform on their look and feel. Unfortunately, there is no low-level component for collecting user input as text. A prototype for such a component used to collect user input as numbers is developed and used for rating movies. Different approaches are analyzed. The implemented solution is chosen due to its limited usage through the application. A first proposal

Figure 6.9: Customized Dialog Windows

represents an alphabet based approach where all characters are displayed at the button of the screen and the user can navigate among the letters using the arrow keys. A character can be selected by pressing the fire button. Therefore user textual input can be collected on a low-level UI using different layouts. The alphabet layout displayed on the screen is just a matter of design.

A second approach consists in implementing a High-level Text Box look and feel as a low-level component. It is very important to distinguish between the key pres event for selecting a letter or digit(multiple pres of a key) and the key pres for actually typing a letter or digit in the input box. The time between pressing and releasing the key has to be measured and dealt with accordingly. This approach is out of the project scope but it can be considered as a possible future work as mentioned in 9. An extension to the existing J2ME low level

components can be created.

A third approach is a more styled version of the first proposal but it involves only digits due to the movie rating system requirements. A slide bar approach is considered as depicted in fig. 6.10. User can move a slider on a bar and function of the slider's position a digit from 1 to 10 is displayed. A vote button is depicted on the bottom of the screen. When the button is pressed a request is made to the server side and the movie rated. The swapping process of the 1 to 10 digits is implemented by swapping the previously created .png images of all digits function of the current selection, pressed arrow, and previous or next digit. This solution is considered and implemented for the customized movie rating component. **Movie Details UI**

Figure 6.10: Movie Rating Custom Component

This is a special case from the UI point of view. The movie poster has to feet on a particular percentage of the screen for any mobile phone. Therefore it is important to know the size of the screen i.e. width and height. When a movie details request is made the screen width and height are sent together with the movie ID. On the server side, a Java Bean retrieves the movie poster from the DB, resizes it and sends it together with the movie details embedded into a Movie Details Java Bean. That way the image can feet on the screen of any mobile device.

The following movie details are depicted on the screen: movie title, year, language, poster, user rating, actors, directors, movie type, and a short description. Two solutions are considered for that. The first solution consists in using a form and trying to fit all information in it. This solution takes advantage of the auto-scroll feature of the form i.e. when the amount of information exceed the screen size, one can scroll to see the rest of info by using the UP and DOWN keys.

This solution involves using high level components. Therefore, colors cannot be used for this screen.

A second approach is to use two Canvases to display the movie details. (fig. 6.11) The first Canvas is used for the movie title, year, language, poster, user rating, actors, directors, and movie type, while the second one is used for the movie short description. Navigation between the canvases and back to the parent screen is implemented. From the usability point of view 4 this is considered a better approach because it is more difficult to read a lot of information in one screen than to organize and split the information into several screens and provide navigation among the screens. This is the chosen solution for the Movie Details UI implementation. Several *helper classes* are implemented for:

Figure 6.11: Movie Details Screens

- updating the GUI
- drawing the text message on the info message screens using different colors, font types and sizes, function of the message type(*info message*, *warning message*, or *error message*)
- creating and adding different images to the screens as icons or buttons
- swapping images depending of the user action or key pressed
- creating the main menu look and feel on a Canvas
- implementing a mouse over - effect similar to the Java Script one
- etc.

6.2.2 The Thread - Based Model

As mentioned before, UI lock-up occurs during network operations. Three different threads are used on the client side to overcome this issue. They perform network communications, keep users informed about the network communication status, and display/update the UI.

The UI lock-up is overcome by using a *background worker thread* for performing all network operations. *BackgroundTask.java* class is the super class for all worker threads used for performing network operations.*(Appendix C)* The template design pattern is used and it provides customized tasks to be executed during network operations. Therefore a *runTask()* hook method present in the BacgroundTask class is implemented by all worker threads extending this class. The network operations are performed inside this method. When the network operations are finished, a new screen is displayed e.g. a info, error or warning screen to inform users about the network operation result, or a new screen that displays the data received from the server side e.g. movie details.

A second thread is used to display an *animated gauge* to keep the user informed about the status of the current network operation. A *ProgressGauge* high-level J2ME component is used for this purpose. (fig.6.12) It starts a background thread and displays an animated gauge during the network operations. A customized message is also displayed on the screen depending on the network operation that is performed to achieve a more user-friendly GUI. A third thread

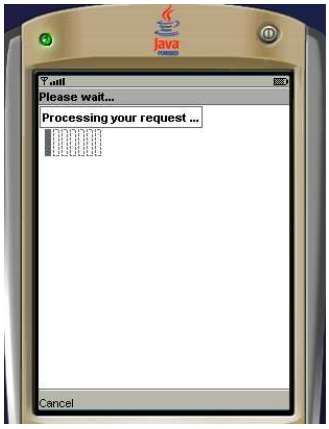


Figure 6.12: Network Communication Progress Gauge

is used for displaying and updating the progress gauge status to keep the user informed about the status of the ongoing network operations.

6.2.3 The Data Model

To overcome the issues mentioned in 2.1 and 2.2 two data models are used:

- **The Network Data Communication Model** - used for network communication operations. It must overcome all issue generated by slow and unreliable networks.
- **The On-Device Data Storage Model** - used for storing different data on the mobile device to reduce the amount of network communications, make data persistent among different sessions, provide an optimized way for reading/writing data from/to the device memory, etc.

6.2.3.1 The Network Data Communication Model

The communication between the mobile device and the server side service is realized by means of *HTTP POST requests* over GPRS. A HTTP connection is established between the client and the server. Different types of requests can be sent from the client to the server. Each request is uniquely identified by a *protocol step - PRT* as mentioned in section 6.4.3. Several request properties are specified i.e. *User-Agent, Content-Language, Connection, Content-Length* before sending the request to the server. *OutputStream/InputStream* are used for sending/receiving binary data to/from the server side. The data is not sent byte by byte, but rather using specific methods of the mentioned streams for sending primitive data types. Sending data byte by byte decreases the application performance. Once a response is received from the server side, the status code is retrieved *HttpConnection.getResponseCode()*. If the status code is an error i.e. and error occurred during the network communication (e.g. connection lost) an alert is displayed on the device screen. If the status code is *HttpConnection.HTTP_OK* the data is retrieved and the appropriate screen or alert is displayed.

What is important to be mentioned is the format used for sending/receiving data to/from the network. Several approaches are considered.

A first solution is to send/receive data byte-by-byte. This results in a performance drop during the network communication operations. Therefore, it is not the preferred solution for data communication over the network.

A second approach consist in using *OutputStream/InputStream* and buffer the data when sending/receiving. This solution does not provide any clear format for the data. One can use this option to send strings, but the data model that has to be send/received is rather complicated. Therefore, sending/receiving very big strings will decrease the application performance due to heavy parsing of the response content on the client side.

The chosen solution consists in implementing a unified data communication model between the client and the server. This is implemented by means of *Java Beans*. Two types of Java Beans are used for exchanging data over the network:

- **Request Java Beans** - encapsulates the request data from the client to the server
- **Response Data Beans** -encapsulates the response data sent from the server to the client

Java Beans are reusable components that should follow the following requirements:

- implement the *Serializable* interface in order to be read/written to a storage or network
- they provide a non-argument constructor
- all their properties must be private and accessible via their *accessor methods (getX())*
- all their properties can be set only via their *mutator methods (setX() methods)*
- they must implement the *toString(), equals(), and hash()* methods
- when needed the *toHTML(), or toXML()* methods can be implemented

Both the client and the server must share the same Java Bean-based communication model or data can be corrupted during the communication process. During the implementation procedure of the Java Beans-based model the *Serializable interface* rule had to be broken due to the *missing Serializable interface in J2ME*. This issue gives rise to several other issues such as: how to keep a unified model, and how to serialize/deserialize the data.
The solution consists in creating Java Beans objects that can serialize/deserialize by themselves synchronously.

This is achieved via 2 methods:

- **public void writeBean(DataOutputStream dataStream)**
- **public static Object_Bean readBean(DataInputStream dataStream)**, where *Object_Bean* is either a Request or Response Bean, depending of the type of the bean.

The *writeBean(...) method* serialize its properties over the network in a portable way via the given *DataOutputStream* using methods corresponding to the primitive data types e.g. boolean, char, String, float, int long, double, etc. If one of the bean properties is a more complex structure e.g. *String Array*, the complex property has to be split into subcomponents having as type a primitive type, and each primitive subcomponent serialized to the network using one of the DataOutputStream methods. On the other end, the complex structure has to be recreated i.e. from the primitive data to the complex format.

An example of such a structure is represented by serializing a *String[] ticketID* in the *Cancel_Tickets_Req_Bean*. The array is serialized in 2 steps i.e.

- the number of elements in the array is serialized first
- each String element in the array is serialized over the network in a for loop

When serializing *byte[]* e.g. *byte[] password* in the *Cancel_Tickets_Req_Bean*, the serialization is done in 2 steps, too. The password is sent as a byte[] over the network because it is encrypted before sending and the result of the encryption is a byte[]:

- the size of the byte array is serialized first
- followed by the whole byte[] using the *write()* method that writes the whole array

For a two column array, the same algorithm applies, but the no. of rows and columns have to be serialized before writing all elements in the array.
One might ask why it is necessary to serialize the number of elements in the array before serializing the array. The answer is that the number of elements in the array is to be used to reconstruct the StringArray when it is deserialized at the other end.

The *readBean(...) method* deserializes the data written by the *writeBean(...)* method. The process is the opposite of the one described above. Every primitive type is read from an *InputDataStream* or complex structures are recreated.

The deserialization algorithm for the *String[] ticketID* in the *Cancel_Tickets_Req_Bean* contains the following steps:

- the corresponding Request Bean is instantiated e.g. *Cancel_Tickets_Req_Bean*
- the number of elements in the array is read first from the stream
- the *noOfTickets property* of the request bean is set
- a new String[] array is created with a size equal to the number of elements read from the stream
- the *ticketID[]* property of the request bean is set

- each String element of the array is then read over the network in a for loop and added to the *ticketID[]* property, etc.
- then the request bean object is returned by the method for later use.

One can notice the synchronous order in which data is serialized/deserialized i.e. the same order is preserved when serializing/deserializing the data e.g. userName, password.length, password, etc.

```
writeBean(...){
...
dataStream.writeUTF(userName);
dataStream.writeInt(password.length);
dataStream.write(password);
...
}

Cancel_Tickets_Req_Bean readBean(...){

Cancel_Tickets_Req_Bean canceledTicketReqBean = new Cancel_Tickets_Req_Bean();
canceledTicketReqBean.userName = dataStream.readUTF();
byte[] password = new byte[dataStream.readInt()];
dataStream.readFully(password);
canceledTicketReqBean.password = password;
...
return canceledTicketReqBean;
}
```

6.2.3.2 On-Device Data Storage Model Using RMS

In order to reduce the amount of network communications, save persistent data among different sessions, and provide an optimized way to read/write data from/to the device memory, a solution has to be found for storing the data on the mobile device.

As mentioned in sections 2.3 and 5 the considered approach is to use the *RMS (Record Management System)* for the on-device data storing. Data is stored in RMS on a record base inside a *Record Store (RS)*. A record store has the structure of a *HashMap* i.e. all records are stored on a key-value basis. The key is an integer number while the value is a byte array. Reading and writing operation on the record store are possible only when the RS is opened. Any data can be write/read in/from the RS, but memory issues have to be considered. A MIDlet can access only the record store that it created.

Accessing the RMS is an expensive operation. Therefore, a solution has to be found for optimizing the access to RMS. After several experiments, the following solution is chosen:

- all application parameters are read from RMS during the initialization phase of the application and stored in static variables. A progress gauge is displayed during the initialization operations to keep the user informed about the ongoing operations as mentioned in section 6.2.5.
- when one of the parameters stored in RMS is needed, its value is retrieved directly from the corresponding static variable instead of accessing the RMS again. Thus, the total amount of time spent accessing the RMS is kept at minimum. *The total access time to RMS is optimized by reading ones and using many times the read values stored in RMS.*
- every time a new object is saved in RMS, it is also saved in a static variable e.g. when a new credit card object is created, it is saved in RMS, and the *Credit Card Array* that contains all credit cards is updated i.e. one more entry is added to the array.
- every time an object is deleted from RMS, the variable that holds that object is set to *null*. e.g. when a credit card is removed, it is deleted from RMS, and the Credit Card Array that contains all credit cards is updated i.e. the element corresponding to that Credit Card is removed
- when an application parameter saved in RMS is updated, the variable that holds that object is updated, too. A search operation is performed against RMS. If that parameter is found, it is deleted and the new value written

to RMS. Otherwise, the parameter is written directly to RMS. RMS is implemented as a hash map. If new values are added without removing the old ones, one can end up in a situation where a parameter has two different values - not quite the desired result!

- search operations - values are stored in RMS under a key-value format. Unfortunately, the key is a long value generated by the JVM. There is no control over the key value. Therefore, a new storing format had to be found in order to search for information. The solution consists in creating a three characters key e.g. *CC1 - credit card no. 1* and prefixing the parameter value with the key. One can say that the new solution is a hash map of hash maps.

Original RMS: x - value
Customized RMS: x - key:value

When a search is performed against RMS, an *Iterator* is used to traverse the RMS, and the value of each record is matched against the given key. When such a record is found the value is returned.

The application must to support several user profiles on the same device. Several solutions are analyzed for this use case.

A first solution consists in using one *RecordStore* and keep user data separated using a customized format for storing data i.e. username:key:value. The read-/write/search operations time increase in this case. Security issues have to be dealt with because user can access each other data in that case.

The chosen solution takes advantage of the default built in data separation in RMS i.e. each MIDlet can access only the record store it created. The solution consists in creating a RS with a different name for each user. It is important the application can recall the name of the RS based on the authenticated user. Therefore, the username is chosen as a name for the user's RS. When the application opens, the authentication form is displayed first. User enters his/her credentials and submits the form. Before the server side authentication, the application tries to open the user's RS. If that is successful, the given password is checked against the one saved in the RS. If the credentials are correct, the application is initialized by reading all parameters from the RS and storing them in static variables. Then, an authentication request is made to the server side service. If user is authenticated, he/she gets full control of the application. As one can see, a double authentication (local and remote) is used in order to access the application.

An *RMSOperations.java* class is provided for all RMS operations

- open, close, display, delete the RMS
- search, insert, replace, delete encrypted/unencrypted values into RMS based on a given customized key
- find or delete all encrypted/unencrypted items
- match a part of a given key e.g. all tickets or credit cards saved in RMS

Storing users' credit card and ticket data Sensitive data such as credit cards, are stored encrypted in user's RS, while the non-sensitive one e.g. tickets, are not encrypted before saving them in the RS. The same indexing format is used for storing both the credit cards and tickets. The key used to identify the object is made of a fix part and a counter. The fix part is CC for a credit card and TT for a ticket, respectively. The counter starts from 0 and it is used to distinguish among the same items. Search, read, write, delete operations are based on the partial key.

Accessing encrypted information in RMS can decrease the application performance. Therefore the access optimization techniques described in the beginning can improve the application performance. Different test are performed for this purpose.

6.2.4 Implementation of the Authentication Algorithm

This section is inspired by the project written during the Secure Mobile Service course (34632) in DTU. The same idea has been used for that authentication protocol. Several changes are made for this case.

The sequence diagram describing the authentication protocol can be found in *Appendix C2*.

Security Implementation Considerations
As mentioned before, security is implemented by means of cryptography and local and remote authentication protocols.

Bouncy Castle cryptographical libraries are chosen for the implementation. They provide fast and reliable implementation of the number of symmetric and asymmetric ciphers. Successful use of these tools require understanding of their technical details.

Encryption of the initial authentication steps
As described in 5.4.2, initial communication between the mobile application and authentication server is encrypted with the key preshared between **Alice** and **Trent** , namely E_a. Encapsulated message that is relayed by **Alice** to **Bob** is encrypted with the key preshared between **Bob** and **Trent** , E_b.

The cipher employed at this stage is **DES**.

Procedures used in the encryption of data with this procedure, may be found in the *Encryptor.java* and *AesKey.java* classes. The *Encryptor.java* class is instantiated with the key parameter, provided by the user. It contains methods for encryption/decryption of *String* or *byte[]* input and output objects.

Encryption of the communication between the client and the server
The encryption of messages between the *Mobile Device* and *Server Side* is accomplished with the use of AES cipher. The key object ($ParametersWithIV$) is created with the use of two elements provided to the user and to the server by the authentication i.e. *random salt value and serial number.*

Implementation of the Authentication Algorithm
As described previously, the authentication protocol involves the participation of the client in all steps:

A, B, R_a - the challenge to the authentication server. The random number is generated on the mobile phone as a numerical value of the type *long*. It's

concatenated with user name and password. Fields are delimited by the char: *;*. The string is encrypted by the key entered by the user. The key is stored in the record store, as it should be sufficiently long and impossible to memorize. It is assumed that the string representation of the key is provided to the user through a safe channel.

$E_a(R_a, B, S_t, S, E_b(S, T))$ - The response from the authentication server. The data intended for the client is extracted after decryption with the pre-shared key. A session key object is generated as a set of parameters with the initialization vector, for the symmetric cipher. The last part of the message is extracted from the response as *byte[]* array. It is sent further to the *Authentication_Servlet_2* without any transformations ($E_b(S, T)$).

$E_k(R_b)$ - The random number sent by the servlet. If it was properly created, it is possible to decrypt it with previously generated AES key. The number is modified and the result sent back.

6.2.5 Application Initialization

As mentioned in 6.2.3.2 reading/writing encrypted data to RMS can decrease the application performance due to the encryption/decryption operations performed on the data. Under these conditions the access optimization techniques described in section 6.2.3.2 can improve the application performance. Different test are performed for this purpose. The results of the experiments are depicted in this section. Several solutions are analyzed.

The first solution reads the data stored in RMS on the fly. When the data is required it is read and used in the application to either initialize a UI screen, send over the network or perform different other operations. During the initialization phase of the application only the user name, password, key, and UI theme are read. The ticket and credit card information saved in RMS are opened on the fly. This results in a very fast initialization process, but every time the user opens the ticket manager to visualize any ticket details, or he/she uses the secure wallet to view/use any credit card, the application performance decreases due to the very expensive encryption/decryption operations for retrieving the credit card objects out of RMS. Once the wallet or the ticket manager is close, and opened again, the application reads the same data from RMS again and again i.e *not a very efficient solution.*

The second solution consists in reading the data stored in RMS on the fly, but storying the read data in static variables accessible to the whole application. The initialization phase of the application is done in the same way as for the first solution. Using this solution, the repetitive read/write RMS operations are eliminated. Data is read only once but used many times. Unfortunately, several extra control statements have to be used to check if the desired data is available or not i.e. the code size increases while the application speed decreases. An alternative solution is found.

The chosen solution implements a 2 way control of the data stored in RMS i.e. from RMS to the application (*initialization*), and from the application to RMS (*update*). These are represented by the *InitModel.java* and *UpdateModel.java* classes.

During the initialization phase of the application, the record store corresponding to the given user name is opened and all required parameters during the application life time are read and the corresponding static variables initialized. A *Progress Gauge* is displayed during the initialization process to keep the user informed about the ongoing operations. If the user name cannot be found, an new empty RS is opened to be used (usually when the user opens the application for the first time). If the user name is found, the corresponding RS is opened

and all parameters retrieved. The access to RMS is optimized for better performance and follows the following steps. The access operations to encrypted, unencrypted data are grouped together, respectively.

- Open user's RS based on the given user name or create a new one if username not found
- Get all user's tickets and initialize the corresponding static variables.
- Get user name and initialize the corresponding static variable.
- Get user's selected theme and initialize the corresponding static variable. If no theme has been selected(application opened for the first time, set the red theme as default in the RS).
- Get user's private key. Check if user's try to change the key and update accordingly. Also, initialize the static variable to hold the key.
- Create the instance of the *Decryptor* class used to decrypt the encrypted information stored in RMS. Thus, the *Decryptor* is instantiated only once and not every time the access to the encrypted RMS info is required.
- Get user's encrypted credit card information from RMS, decrypt them and initialize the corresponding static variables.
- Get user's encrypted password from RMS, decrypt it and initialize the corresponding static variable.
- Get user's encrypted secure wallet PIN from RMS, decrypt it and initialize the corresponding static variable.
- Delete any used tickets stored in RMS and update the corresponding static variables.

As mentioned before, the initialization phase of the application is optimized by:

- Grouping read/write operation to unencrypted and encrypted data saved in RMS, respectively.
- Saving the data out from RMS into static variables.
- Instantiating the *Decryptor* class only once.
- Keeping all initialization operations in one place, and reducing the control structures used in the code.

The second way of controlling the data stored in RMS, the *Update model* is used when update operations are executed on the data saved in RMS e.g. add a new credit card, remove a credit card, edit a credit card, add a new ticket, or cancel a ticket. The *Update model* is also optimized, but all the operations provided in here must be executed on the demand. Several tests are performed. The application performance does not decrease. A progress gauge is displayed every time an update operation is executed. Usually, the time is less than 1 sec for regular operations and up to 2 seconds for credit card related operations.

6.2.6 Location-aware Movie Search Service

The Mobile Cinema application is implemented as Location-aware application i.e. it allows users to enter their current position. Then, users can make requests to the server side and retrieve a list of all movies in a given range from the user matching different searching criteria e.g. movie that start with XM and are located in Copenhagen. Different search criteria can be submitted. The server side service interpret all of them and returns a list of found movie or an error message in case no movies can be found.

The location-aware feature is meant to be provided by a 3rd party service via an external API. The Server Side Service connects to the Location-aware service via the given API, verifies user's given position and returns a possible list of close positions in case the given position cannot be found. If the user position is verified a list of all cinemas in the given range from the user is returned. The server side service can use this list and perform a search for the required movie(s). A list of found movies is returned then to the user.

Several location-aware services have been contacted to provide an evaluation license for their service e.g. *eniro.dk*, *krak.dk*, *www.viamichelin.com*, etc. None of them were willingly to offer student licenses to their location service. Therefore, an alternative had to be found i.e. to emulate the location-aware service. The emulated service validates all given user' positions and give a list of all cinemas in the given city. The cinema list is used further one by the Cinema Server Side Service to find the movies matching user's given criteria.

Several other solution can also be used for the location - aware client:

- The latest mobile phone from Nokia S60 provides a *Location-aware API (JSR 179)* that can be used to determine user's current position. The Mobile Cinema application is meant to be used by as many users as possible. Therefore, making this application available for a limited amount of mobile devices does not feat in chosen marketing strategy 8
- A GPS device can be used together with a mobile phone to determine the current location. But, not many users have a GPS device connected to their phone. Thus, only a limited number of customers are able to use the application. This is not a desirable solution from the chosen marketing strategy point of view 8
- *Cell ID* i.e. the cell that the movie goer's mobile device is connected to. The accuracy depends on the cell size. It can be implemented both network based and device based. [5]

- *Cell ID + Timing Advance* i.e. the time delay between the mobile and serving base station. Accuracy is 500 meters. They are available into the network. [5]
- *Signal Strength Based* - Measure signal strength from the control channels of several Base Stations. If signal levels from 3 different *Base Stations* are known, it is possible to calculate the location .[5]
- Keyboard based - user keys in its current position
- Camera based - user takes pictures of a road sign and send it to an MMS server. [5]

The keyboard based solution is chosen in this case together with a 3rd party location-aware service that returns a list of all cinema in a given range from the movie goer's current position.

6.2.7 My Secure Wallet

My Wallet feature is meant to be used together with the Mobile Cinema application or as a standalone application by extracting it from the current application. All sensitive information stored in the wallet is encrypted. Different optimization techniques are used to increase the access time to My Wallet and its features. Optimization is very important especially when dealing with encryption-decryption operation that require high computation power.

The sequence diagrams describing the possible use cases for the Secure Wallet feature can be found in *Appendix C1*.

The following UI screens are implemented *MyWalletMainMenu, MyWalletAddNewCC, MyWalletEditCC, MyWalletAuthenticationScreen, and MyWalletViewCC*. The first four UI screens are extended from the *GenericGUI* class. The last one extends the Canvas and displays credit card information graphically.

By extending from the *GenericGui* class, most of the functionality is already implemented. These classes are only implementing the hook methods that define the particular functionality for each one of them. Therefore, the *initMOdel(), updateModel(), createView(), and commandAction()* are implemented by these subclasses.

My Wallet Authentication
My Wallet is protected by an PIN-based authentication. The high level J2ME components to be placed on the screen are created in the *createView()* method of the *MyWalletAuthenticationScreen*. If a PIN code value is not saved in user's RS i.e. My Wallet is used for the first time, a PIN code set up screen is displayed. Once the PIN is set up, the user is presented with the authentication screen. He keys in his PIN code and if authenticated, my wallet model is initialized and the main menu displayed. In case authentication is not successful for 3 times during the current session, the wallet PIN and content is reseted as mentioned in section 5.2.2. Thus, an empty wallet is made available to the user. Switching between the setup PIN and authentication screen is implemented in the *createView()* method of the authentication screen by checking if the PIN value is empty or not. Also, the menu commands are changed dynamically function of the displayed screen.

My Wallet Main Menu
The Wallet model is initialized in the *initModel()* of the *MyWalletMainMenu* class. This displays all available credit cards previously saved by the user in its own RS. Several solution are analyzed for the initialization procedure of my wallet as mentioned in section 6.2.5.

The first solution does not initialize the client model during the application startup but on the fly. When My Wallet is opened, the model initialization is triggered and the wallet initiated. Every time when the wallet is opened, the same initialization is performed. This is done by reading all encrypted credit card information from RMS, decrypting and saving them into an array of CreditCards. This operation is very expensive from the computation power point of view. Also, the high initialization time makes the wallet a non-user friendly feature.

A better approach consists in initializing the model when the wallet is opened for the first time, and all conf parameters saved in static variables accessible to the whole application. But this is not considered as a good design.

Therefore, the chosen solution, as mentioned in section 6.2.5 is to initialize only once and use the initialized data many times. As a result, the wallet model is initialized at application client start up, decrypted and stored in static variables. When My Wallet is opened, the model is initialized from the static variables. This is done in less then 0.4 sec. This is considered the best solution from the design and usability point of views.

After the model is initialized, the view is created in the *createView()* method. Menu commands are created and appended to the form for *add a new credit card, edit/remove credit card, help, exit, and back operations*. Also, a *ChoiceGroup* is created and appended to the form. This is used to display all credit cards available in user's own RS.

All credit cards are saved in RMS as *encrypted byte[]*. This is achived by creating a *Credit Card Java Bean (CreditCardBean.java)* with the following properties as String variables *CCNickName, CCBank, CCEmergencyPhone, CCOwner, CCType, CCNumber, CCExpDateMonth, CCExpDateYear, CCCW2, and CCPIN. setX() and getX()* methods are made available for getting/setting property values. As mentioned in section 6.2.3.1, the serializable interface is not available in J2ME. Therefore, this Java Bean is implemented to serialize/deserialize by itself using *DataOutputStream/DataInputStream* to write/read primitive data types.

Using Java Bean objects to write/read data to/from RMS is a best practice. Structuring the input/output data reduces the computation time and power needed otherwise e.g. to parse a very big and complicated string. The application source code is also more structured and manageable.

My Wallet - Add New Credit Card

The UI screen used for this operation extends the *GenericGui.java* class. No model initialization is performed in here. The high-level J2ME components to

be displayed on the screen are created in the *createView()* method by initializing all *TextFields* components for collecting user input. The command action are set in the *commandAction()* method. When user presses the *SAVE* button, the credit card data format is verified e.g. credit card no. must a 16 digits no.; the year is a 4 digits no. between 2007 and 2012; the month is a 2 digit no. between 1 and 12. If the data is valid, it is further on encrypted, the *credit card java bean* created, and serialized to the user's RS. The static variables that store my wallet model are updated. During this operation, a progress gauge is displayed. The updated model is made available to the whole application in that way. In case the validation fails, an error message is displayed.

My Wallet - Edit Credit Card
This functionality is similar to the *Add new credit card* operation. The same form used for adding a new credit card is displayed to the user but the components are populated with the selected credit card data. Once the user presses the *UPDATE* button, the form is validated and function of the validation results, the data is further on encrypted, the *credit card java bean* created, and the application model updated.

My Wallet - Remove Credit Card
A credit card can be removed from the wallet by selecting it in the wallet menu and choosing *Remove Credit Card* option in the wallet option menu list. A *Yes-No* dialog window is displayed. If the movie goer chooses *Yes* the credit card is removed from RMS, and the model updated using. Both the RMS and static variables are updated. An *Info Message Window* is displayed. The message is displayed until the users presses any key. On the other hand, if *No* is chosen, the previous screen is displayed.

My Wallet - View Credit Card Details
The UI screen for display the credit card information extends the Canvas J2ME component. Its look and feel resemble a real credit card and provides access to other sensitive data such as *credit card PIN code and CW2*. The credit card representation is build dynamically i.e. to fit on any screen size. The text and shapes are drawn following the same principle. Unfortunately, the background is an image. Therefore, different images must to be used for different mobile screen sizes. Another approach can be considered by using the *GameCanvas, TiledLayer or Sprite* components. The background can be created in this case dynamically, but the image has to be built out of small pieces as a puzzle. For the current prototype, the author considers this approach as appropriate. Image optimization operations can be conducted as future work on the project as mentioned in section 9.
The credit card is created in the *paint(Graphics g)* method where all dynamic painting takes place. Several other helper classes are used for this purpose e.g. *GuiWalletHelper.java* and *MyWalletTools.java*.

The user interaction is done by following a similar design with the *Yes-No Dialog Window* i.e. buttons are used to display the PIN and CW2. A button rollover effect is implemented. A bi-directional circular navigation among the buttons is created i.e. one can go from the first button to the last one by pressing the left arrow; and the other way around.

My Wallet - Help
It extends the J2ME Canvas and displays help information about the wallet features and functionality in a user friendly graphical format.

6.2.8 The Ticket Manager

All ticket operations are performed via the ticket manager. Displaying, storing, canceling or deleting tickets are the functions implemented by the manager. Different optimization techniques are used to increase the access time to the tickets stored in RMS.

The sequence diagrams describing the possible use cases for the Ticket Manager feature can be found in *Appendix C1.*

When the application model is initialized at start up all available tickets are read from the user's RS. During the model initialization process, the ticket expiring date and hour are checked. If the ticket has expired it is removed from RMS and the model updated. Memory issues are addressed in that way. Several Java class implement the ticket manager i.e. *MyTicketMainMenu, MyTicketViewTKT and MyTicketTools.*

The Ticket Manager - Main Menu
The ticket manager main menu look and feel is similar to the My Wallet main menu. Both of them are based on the same logic. Several differences are notable between them:

- The Ticket Manager displays the acquired tickets. Every time a ticket is acquired it is automatically saved in userś own RS and the application model updated.
- The Ticket Manager uses a *TicketBean* Java object to operate on the ticket level and provide structure to the read/write operations.
- The tickets are not encrypted in RMS. Therefore the time to load all tickets is smaller than for loading all credit cards. Still, the model is initialized during the application start up, to achieve data and logic separation and better performance.

The *TicketBean* Java Bean stores all properties that can identify a ticket i.e. *tktID, tktReservationID, tktCinema, tktCinemaAddress, tktCinemaTheater, tktMovie, tktShowDate, tktShowHour, tktSeat, tktRow, tktDiscountType, tktPrice, tktPurchaseMethod, tktReservationDate, and tktStatus. setX() and getX()* methods are available to manipulate the property values. The TicketBean serializes/deserializes by itself.

The Ticket Manager - View Ticket Option
The UI screen used to display the ticket details extends the Canvas J2ME com-

ponent. Its look and feel resemble a real cinema ticket. The ticket UI is dynamic created to fit any screen size. The text and shapes are drawn following the same principle. Unfortunately, the background is an image. Therefore, different images need to be used for different mobile screen sizes. Another approach can be considered by using the *GameCanvas, TiledLayer or Sprite* components. The background can be dynamic created in this case, but the image has to be built out of tiles, as a small puzzle. For the current prototype level, the author considers this approach as appropriate. Image optimization operations can be conducted as future work as mentioned in 9.
The user interaction is implemented in a similar meaner to the *Yes-No Dialog Window*. A button rollover effect is implemented.

Every ticket displays a *bar code image* i.e. graphical representation of the ticket information as mentioned in section 5.2.3.
The bar code image can either be generated on the mobile device as a byte[] and then given to an Image object, or it can be created on the server side and returned together with the ticket details response. Benchmarking tests have to be performed. The test have to record the time used for generating a bar code image on the mobile phone comparing to generating the image on the server side and transferring it via GPRS to the client. The price to be payed for sending the the image has to be taken into consideration, too. For the current prototype, the bar code image is a static image. Generating a bar code image is a complicated task. It requires access to a bar code reader in order to check the image validity. 3rd party API's are available for generating bar code images.

The Ticket Manager - Cancel Ticket Option
A ticket can be canceled from both the ticket manager main menu or ticket details screen. A *Yes-No Dialog Window* pops up. If the movie goer chooses *Yes*, a *Cancel_Ticket_Req_Bean* is instantiated and sent to the server side. A response is received as a *Response_Msg_Bean*. An info message is displayed, function of the response code value. If the ticket is canceled successfully on the server side, the client deletes the tickets from RMS and updates the model.

The Ticket Manager - Help
It extends the Canvas and displays help information about the ticket manager features and functionalities in a user friendly graphical format.

6.2.9 Application Settings

Several options are implemented for setting different customizable application parameters such as:

- the application access password
- the wallet PIN code
- the application theme
- the application language. This is considered as a possible future work as mentioned in section 9

The sequence diagrams describing the possible use cases for My Settings feature can be found in *Appendix C1.*

Application Settings - Change Password
Movie goers can change their application password in *Main Menu* ↦ *Application Settings* ↦ *Change Password*. The Change Password UI displays 4 **TextFields** for entering the user name, the old, new and verified password, respectively. Once the *CHANGE* button is pressed, the user's credentials are checked. If the given credentials match the ones saved in user's own RS, and the new and verified passwords match, a *Change_Password_Req_Bean* is created and sent to the server. A *Response_Msg_Bean* is received by the client and an info message displayed function of the operation status code. If the operation was successful, the application model is updated.

Application Settings - Change My Wallet PIN code
Movie goers can change my wallet PIN code in *Main Menu* ↦ *Application Settings* ↦ *Change My Wallet PIN*. The Change PIN UI displays 4 *TextFields* for entering the user name, the old, new and verified PIN code, respectively. When the *CHANGE* button is pressed, the credentials used to access the wallet are checked against the ones saved in RMS. If they match, the old PIN code is deleted from RMS and the encrypted new PIN saved into RMS. The application model is also updated. Otherwise, an error message is displayed.

Application Settings - Change Theme
This option allows users to change the current application theme. By default the application theme is the *RED THEME*. For the current prototype once can choose between 2 themes i.e. *RED THEME* and *BLUE THEME*. The theme selection is implemented in the *ChangeThemeGUI.java* class. The themes are displayed using a *ChoiceGroup* appended to a form. When one of the theme is

selected, the *application_theme_update* is triggered and the theme is changed to the selected one. The application model is also updated.

When the application theme is changed a set of updates of the GUI look and feel are preformed:

- The main menu look and feel are changed
- The images displayed on every screen are replaced
- The colors used for drawing strings and shapes are redefined

This is done by setting a configuration parameter in the application. Every times one of the screens initializes, it reads that configuration parameter and displays the appropriate images or uses the corresponding colors.

Application Settings - Change Language
This is not implemented due to insufficient time, but it is still analyzed in here. This feature can be implemented by setting a *locale* parameter to the whole application i.e. the language to be used. The titles used for different J2ME components, default values of different text fields, the text of info message screens and buttons, and all other components displaying textual information must be parametrized. Function of the *locale* value, each screen loads the needed parametrized values, and displays the component using the textual representation of the selected language.

Storing the different language textual representations is another issue. One can use Java constants or save them in .xml files that can be loaded and parsed function of the chosen language. The .xml file-based solution is to be chosen in this case.

6.3 Security Implementation Considerations

This section depicts the choices made during the implementation phase of the Cinema System security layer.

The security of the Cinema System is provided by means of an *authentication protocol* presented in section 5.4.2 combined with use of *cryptography* for encrypting user sensitive data stored on the mobile device or sent OTA.

The Legion of the Bouncy Castle API [7] is chosen for implementing the system security. They provide a fast and reliable number of ciphers and appropriate methods for the underling security.

6.3.1 Encryption of the Authentication Protocol

The authentication protocol is perform with the use of two Java Servlets i.e. *Authentication Servlet 1*, and *Authentication Servlet 2*. These two Java Servlets correspond to **Trent** and **Bob** , respectively 5.4.2.

Encryption of the Communication between the Mobile Client and Authentication Servlet 1(Trent)
This part describes the initial steps of the authentication protocol.

As described in section 5.4.2 the communication between the mobile client and *Authentication Servlet 1 (**Trent**)* is encrypted with the key preshared between **Alice** and **Trent** , namely E_a. The encapsulated message that is relayed by **Alice** to **Bob** is encrypted with the key preshared between **Bob** and **Trent** , E_b.

A *DES* cypher is chosen for this purpose and it is used as a *Block cipher.*[7]

The methods used for data encryption in this case can be found in the *Encryptor.java* class. This class has a constructor that is instantiated with they key parameter provided by the user. It contains methods for encryption/decryption of input objects of type *String* or *byte[]* returning either objects of type *String* or *byte[]*. Methods for creating a *ParametersWithIV* key, and AES encryption/decryption operation are also implemented.

```
public Encryptor(byte[] key){

cipher = new  PaddedBufferedBlockCipher(
new CFBBlockCipher(new DESEngine(),8));

this.key = new KeyParameter (key);
}
```

Encryption of the Communication between the Mobile Client and Authentication Servlet 2(Bob)
This section depicts the encryption of sensitive data that is to be sent to the worker servlets during authentication, change password, and purchase tickets requests. The encrypted communication between the mobile device and the *Autentication Servlet 2* is realized by means of AES ciphers run in **Cipher Block Chaining Mode**.[7]

As mentioned in section 5.4.2, the key object (*ParametersWithIV*) is created

with the use of two elements(*a random salt value and user's serial number*) provided both to the user and *Autentication Servlet 2* by the authentication service.

```
public ParametersWithIV createKey(String salt, String key){
PBEParametersGenerator generator =
new PKCS12ParametersGenerator(new SHA1Digest());

generator.init(
PBEParametersGenerator.PKCS12PasswordToBytes(key.toCharArray()),
salt.getBytes(), 1024);

// Generate a 128 bit key w/ 128 bit IV
ParametersWithIV ret =
(ParametersWithIV)generator.generateDerivedParameters(128, 128);

return ret;
}

...

public byte[] encryptWithAES(ParametersWithIV key, byte[] msg){...}

public byte[] decryptWithAES(ParametersWithIV key, byte[] result){...}
```

The issue with this implementation is that if one bit is corrupted during the transmission, then the whole message becomes unreadable. Security is very important for protecting users' private data. Therefore, one can assume that the communication channels do not corrupt the data.

6.3.2 The Implementation of the Authentication Protocol

The authentication protocol is implemented using two Java Servlets i.e. *Authentication Servlet 1 & 2*. The first servlet implements the initial steps of user authentication protocol, while *Authentication Servlet 2* deals with the remaining steps. Both servlets receive the client request via the *Cinema Central Controller Servlet* that is the entry point(request dispatcher) of the server side service. More details are depicted in section 6.4.1. The sequence diagram depicting the authentication protocol implementation can be found in *Appendix C2*.

Authentication Servlet 1
The implementation details of *Authentication Servlet 1* are depicted below:

- It receives the initial authentication request from the mobile client in the *doPost()* method. The encrypted message sent by the client is extracted by the servlet as *byte[]* using an *InputStream*. The message consists of a challenge made of *user name*, *password*, *address of the second authentication servlet*, and a *random number*.
- The message is decrypted using a DES cipher and the key preshared with the mobile client.
- User's credentials are extracted and checked against the DB using a *PreaparedStatement* obtained from a *connection pool* object. If the credentials are verified it retrieves the *userID* and *randomID* values from the DB. This two values are concatenated into a *token* and stored into the *JVM system properties* for further use by *Authentication Servlet 2*.
- It checks if the *random number* sent by the client has not been used before and if the address of the second authentication servlet is correct. In case this conditions are fulfilled the response message is formed.
- It forms the message for **Bob** i.e. $E_b(S, T)$. It generates a long random number and concatenates it with a token made of *userID* and *randomID*. The result is encrypted with a DES cipher.
- The other elements of the response message i.e. *user's e-money*, the *random no.* sent by the client, the *address of the second authentication servlet*, the *userID*, and the *long random no.* are generated, concatenated, and converted it into *byte[]* forming the *first part of the message response*.
- The *first part of the message response* is concatenated with the *message for **Bob*** and all this encrypted with the key preshared with the mobile client.

- The encrypted message is then sent to the mobile device.
- If the user is not authenticated against the DB, or any other exceptions occur during the first step of the authentication process, an error code is sent back to the mobile device.

This servlet has practically three main methods that provide the business logic:

- **init()** - initiates the servlet state and retrieve the DB connection pool from the JNDI. More details about the connection pools are depicted in section 6.5.3.
- **doPost()** - provides the main communication functionalities between the client and the server. Extracts the client message from the *ServletInputStream* and calls the *returnAuthenticationResponse()* method to process the initial authentication steps.
- **returnAuthenticationResponse()** - performs the first steps of the authentication protocol as described above and creates the response to be sent to the client.

Authentication Servlet 2
The following methods provide the business logic around this servlet:

- **init()** - initiates the servlet.
- **doPost()** - provides the main communication functionalities between the client and the server. Extracts the client message from the *ServletInputStream* and calls the *handleTheProtocol()* method to process the rest of the authentication steps.
- **handleTheProtocol()** - performs the final steps of the authentication protocol based on the protocol step received from the client side i.e. *PRT1* or *PRT2*. Each encrypted message received by this servlet has the following format *protocol step + encrypted message*. The protocol step is a four-character identifier.

  1. **PRT1** - The servlet is challenged with the message relayed from the *Authentication Servlet 1* i.e.$E_b(S,T)$. It extracts the *salt value* and the *token* from the message by manipulating the *raw array object*. This operation is necessary not to corrupt the cipher padding. The *token* is checked against *Authentication Servlet 1* via the *JVM system properties*. If the result is successful a *session key* is generated based

on the *token* and the *salt value*. A *large random number* is generated and encrypted with the session key. The encrypted random number is sent to the mobile client.

2. **PRT2** - This is the last step of the authentication procedure. The encrypted message received from the client is decrypted using the previously generated *session key* and the extracted random number. If the value is equal to the value generated in *PRT1* minus one, the user is authenticated and sensitive requests such as *purchase tickets* can be processed.

The encryption/decryption methods are provided by the *Encryptor.java* class mentioned in section 6.3.1.

The *token* shared between *Authentication Serlvet 1 & 2* via the *JVM system properties* is used only for prototyping purposes. It is not safe from the security point of view to store sensitive data into the JVM and have them widely opened for a malicious attacker. Also, this solution is not functional in case each servlet is running in different JVM's. Alternative solutions can be developed:

- storying the token in an *.xml configuration file*
- storying the token as an encrypted in a *java object* in a *binary file*
- storying the token encrypted in a *DB as byte[] or Blob*[1]

Also, the communication between the two authentication servlets can be done via *Java RMI*. This can be part of a future work as mentioned in section 9.

[1]Binary large Object

6.4 Cinema Ticket Reservation Service Implementation

As mentioned in section 5 a *3-tier model* is chosen as architectural solution for the Cinema Ticket Reservation System. The second tier implementation is displayed hereinafter. Tomcat is used as a container for the Java Servlets[2].

J2EE (Servlets, Java Beans, JDBC), Bouncy Castle cryptographic libraries, and J2SE technologies are used for implementing the server side service together with several design patterns such as: *MVC, Facade, Singleton, Template Method, Refactoring, Abstract Coupling, Iterator*, etc. The class diagram is depicted in Appendix C.

[2]Servlets are generic server extensions that can be dynamically loaded when needed by the web server. HTTP requests/responses are used to communicate with the clients

6.4.1 Integration of Design Patterns

Model-View-Controller Design Pattern
The Cinema Ticket Reservation System follows a *Model-View-Controller* architecture as depicted in fig. 2.1 combined with a *Facade* approach. The *MVC* design pattern has the advantage of separating the business logic from the presentation. Thus, all business logic can be performed on the server while the results are only displayed on the mobile device. This suits very well to the limited computation power and battery life issues of today's mobile devices. It also improves the system scalability and management

The server side service has a common entry point(*the Controller i.e. Cinema Central Controller Servlet*) for all HTTP requests made by the client. The *Controller* checks the type of request e.g. reserve tickets request, and dispatches the requests to the appropriate *Worker*. The designated worker processes the request and performs the required business logic.[3] A response is then sent back to the client. *The Worker* uses *Java Beans*[4] to perform the business logic(DB connections, encryption, decryption, etc) and updates the application *model*.

The Model is represented by *request/response Java Beans* used for sending / receiving data to/from the client/server. These Java Beans contain all parameters mentioned in the design section of the protocol steps 5.3. They represents the objects used for sending the request/response data as mentioned in section 5.3. When a request is made he following business logic is performed by a worker servlet:, the parameters embedded in the request as properties of the java beans are retrieved by the worker, deserializing the java bean. The required business logic is performed:

- the request bean is deserialized and casted to the appropriate type
- the request bean properties are decrypted if necessary
- a prepared statement is built to query the DB using the request bean properties
- obtain a connection pool to the DB
- the DB is queried and a result together with an SQL status code[5] is retrieved

[3]The forwarding is based on a set of mappings. It increases the system scalability and separation among tiers.
[4]A Java Bean is a reusable software component where all its properties are accessed using setter and getter methods. It implements the Serializable interface and has a no-argument constructor
[5]it represents the operation status code mentioned in section 5.3

- parse the result from the DB query and build the *response Java Bean*[6]
- the response bean is serialized to the network on the same communication channel

The application MVC is depicted in fig. 6.13. *The worker servlets* are built using the *Template Method* and *Facade* design patterns.

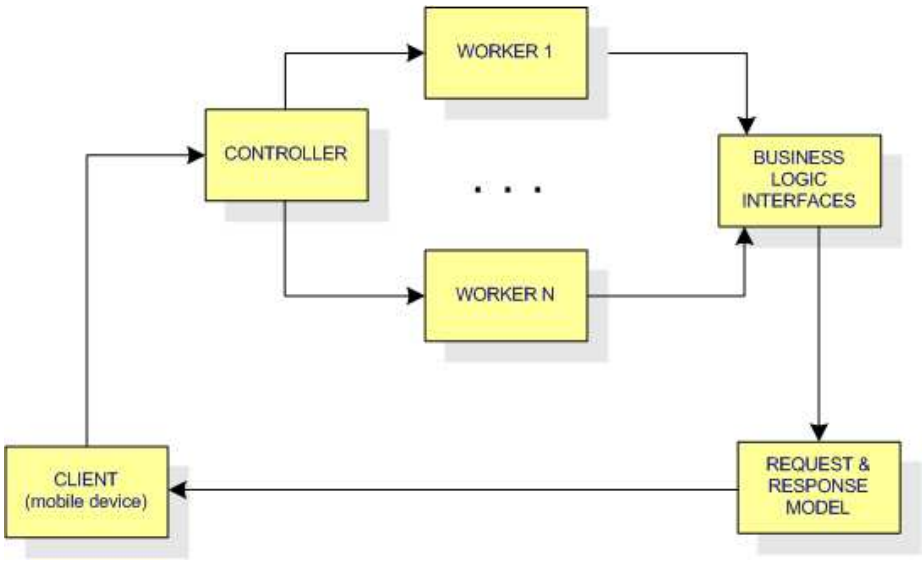


Figure 6.13: Application MVC components and interaction

The Template Method Design Pattern

The *Template Method* defines the skeleton of an algorithm in a method, deferring some steps to subclasses, thus allowing the subclasses to redefine certain steps of the algorithm.[22]. A *Generic Workwer Servlet* implements the invariant part of the workers algorithm by defining three *abstract hook methods*. The hook methods are implemented in the concrete classes that *extend* from the Generic Worker i.e. *Background Hall Update Servlet, Cancel Tickets Servlet, Change Password Servlet, Find Movies Servlet, Movie Details Servlet, Purchase Tickets Servlet, Rate Movie Servlet, Reject Payment Servlet, Select Deselect Seats Servlet, and Select Show Serlvet.* The hook methods defined by the Generic Worker are depicted below:

[6]HTTP is a stateless protocol

- **getRequestBean(...)** - cast the request Java Bean received from the client to the appropriate type

- **setSQLStatement(...)** - set the SQL statement that is to be executed against the DB using *PreparedStatement*. Each DB request is made using *connection pools* server by the *Generic Worker Servlet*. More details about the connection pool mechanism are depicted in section 6.5.3.

- **parseSQLResponse(...)** - parse the result of the SQL statement and creates response to be sent to the client

The Facade Design Pattern and Abstract Coupling
The *Facade and Abstract Coupling* design patterns hide the complexity of the business logic and provides a simple interface to the worker from where the worker can access all subsystems to perform the needed business logic. Also, it promotes a weak coupling between the workers and all subsystems i.e. the subsystem implementation can be changed without changing the worker code. These design patterns reduce the system complexity by structuring the system into subsystems.

The system is split into several subsystems:

- **The Request Data Model** implemented by the *RequestDataModel.java* class. It defines the business logic for processing the request java beans received from the client. Parameters are extracted and DB *PreparedStatements* to be executed are created.

- **The Response Data Model** implemented by the *ResponseDataModel.java* class. It defines the business logic for processing the sql results from the DB and it creates the reponse java beans to be sent to the client.

- **The Servlet Operations Bean** - retrieves the request java beans sent by the client and constructs the response to be sent to the mobile device including the response java bean.

- **The SQL Operation Bean** - sets the sql statement to be executed against the DB, the parameters for the SQL statement, the connection pool, and executes the *PreparedStatement* against the DB, etc.

- **The Encryptor** - performs encryption - decryption operation using the Bouncy Castle API.

- **other helper classes**

The interfaces used for accessing the previously mentioned subsystems are defined by *Client To Facade*, *Facade To Model*, *Client To Facade Interface*, *Request Model Interface*, and *Request Model Interface* interfaces.

The Singleton Design Pattern is used when only one instance of a particular class must be available per session. The Singleton is implemented by using a *Logger* to log different client-server interactions, method calls, etc. *Log4j* is used to log different actions in the server side implementation. Messages defined as *DEBUG, INFO, WARN, ERROR, or FATAL* are logged. A *log4j.properties* configuration file is created under *$CATALINA_HOME/Cinema Controller/WEB-INF/classes/* directory. Different configuration parameters such as logger output level e.g. DEBUG, logging file name, size, and layout of the logged message are defined. The *log4j.properties* file is depicted in Appendix E1.

The server side source code is *refactored* using the following methods:

- previously mentioned design patterns
- minimizing object creation and reusing old ones
- grouping common functionality in a method and using that method instead of repeating the code
- break big methods into smaller ones
- use of inheritance and polymorphisms
- use of a controller
- use of self-explanatory method names
- etc.

6.4.2 Communication Protocol Implementation

The communication between the mobile client and the server is realized by means of *HTTP connections* using the implementation of the *HttpConnection-interface* of the mobile device JVM. Request can be sent from the mobile phone and responses received. Section 5.3 describes eight types of requests that are sent from the mobile device to the server side. POST and GET requests are used for communication. *Background threads* are used for the network communication between the mobile client and the server side to prevent UI lock-up as mentioned in section 6.2.2.

The sequence diagrams describing the communication protocol and server side service implementation are depicted in *Appendix C2*.

Using Java Beans for client - server communication
Two solutions are analyzed for the client-server communication protocol implementation.

The first solution consists in sending/receiving buffered binary data using *OutputStream/InputStream*. Header request properties are set up before sending the data. Once a response is received, the response code is retrieved using the *HttpConnection.getResponseCode()* method. Function of the response code value i.e. error or OK *HttpConnection.HTTP_OK* the response data is displayed on the mobile screen. Once the response is retrieved, the network connection and the OutputStream/InputStream are closed. The problem with this solution lays in the data format to be sent/received. The data cannot be manipulated in an easy way - it does not have any structural definition e.g. a concatenated string. If a very complex string e.g. cinema theater info such as discounts, prices, booked seats, shows, etc. is received, a huge amount of computation is needed on the mobile client to parse the string. Due to the hardware limitations mentioned in the previous sections this approach is considerate not acceptable.

The second approach consists in defining a structural format for the data to be sent/received i.e. *Java Beans*.[7] The computation power required for extracting the data from the bean is limited. Therefore Java Beans can overcome the hardware limitation issues of the mobile client. This is the chosen solution for implementing the communication protocol between the client and the server.

The Java Beans used for sending/receiving data from/to the mobile device follow a particular design due to different software limitation of the mobile JVM e.g. streams operations. A template of the implemented beans is depicted below.

[7] *Java Beans* are reusable software components used to encapsulate many objects into a single object, so that the bean can be passed around rather than the individual objects.[13]

```
public class TemplateJavaBean{

private int    property1 = 0;
private String property2 = "";

public TemplateJavaBean(){}

public void setProperty1(int property1){
this.property1 = property1;
}

public void setProperty2(String property2){
this.property2 = property2;
}

public int getProperty1(){
return property1;
}

public String getProperty2(){
return property2;
}

public void writeBean(DataOutputStream dataStream)
throws IOException {

dataStream.writeInt(property1);
dataStream.writeUTF(property2);
}

public static TemplateJavaBean readBean(DataInputStream dataStream)
throws IOException{

TemplateJavaBean templateBean = new TemplateJavaBean();
templateBean.property1        = dataStream.readInt();
templateBean.property2        = dataStream.readUTF();
return templateBean;
}

public String toString(){...}

public boolean equals(Object object){...}
}
```

The *TemplateJavaBean* follows all Java Bean conventions i.e. it contains a no argument constructor, has a set of properties i.e. *property1* of type *int* and *property2* of type *String*. *Set* and *Get* methods are defined for both properties for accessing/mutating the property values i.e. *setProperty1()/getProperty1()*, and *setProperty2()/getProperty2()* respectively. The *toString()* and *equals()* methods are defined. What makes this java bean different from other beans are the *writeBean()* and *readBean()* methods. One can say that this bean *serializes/deserializes itself to/from the network*.

The Communication Protocol Steps

1. The mobile client sends a request to the server side by opening a *HTTP-Connection* and obtaining a *DataOutputStream* from the connection to write the data to the network.
2. A *Request Java Bean (RQJB)* corresponding to that particular type of request[8] is created and the RQJB properties are set using the *setX()* methods e.g. A *Change Password Request Bean* has as properties *user name*, *old password*, and *new password*. The *setUserName(), setOldPassword(), and setNewPassword()* methods are called for setting the *Change Password Request Bean* properties.
3. Once the properties are set, the *writeBean()* method of the bean is called. This method gets as argument the previously opened *DataOutputStream* and writes the bean properties to the stream using specific methods for each primitive data type e.g. *writeInt()* for *Integers* or *writeUTF()* for *Strings*, etc.
4. Once the data is sent to the network the output stream is closed and the mobile client application is waiting for an answer from the server side.
5. The request sent from the mobile is received by the *Controller Servlet* on the the server side. It determines the type of request and dispatches the request to the appropriate *Worker Servlet* to perform the requested business logic. A *RequestDispatcher* object pointing to the worker Servlet is created and the *forward(request, response)* method is called on the *RequestDispatcher* object to forward the request to the corresponding worker.
6. The worker calls different components(java classes) that implement different parts of the required business logic e.g. decryption, SQL statement settings, SQL parameter set up, SQL query execution, SQL Result parsing, creating the *Response Java Bean* and sending the response to the client. Based on the *Facade* and *MVC* design patterns all business logic is hidden and simple interfaces are provided for the worker interaction.

[8]There are 8 types of Request Java Beans defined for each type of request

7. The worker dispatches all business logic steps to the corresponding components who retrieve the RQJB properties by using the *readBean()* method of the RQJB. This method returns a new RQJB object that it is a 100% identical copy of the RQJB sent by the client.

8. The worker sets the SQL statement to be executed against the DB e.g. *SELECT * FROM cinema.Change_Password(?, ?, ?)* by executing the corresponding stored procedure in a *PreparedStatement*. The worker calls another Facade method and sets the parameters of the *PreparedStatement* by reading the RQJB properties using the corresponding *getX()* methods. The RQJB is passed as an argument to the Facade method that sets the stored procedure parameters together with the *PreparedStatement* object. *There is a 1:1 mapping between the RQJB properties and the Stored Procedure input parameters* as depicted in section 6.5.2. This improves the system performance and makes the protocol transparent and easy to maintain.

9. The worker dispatches the business logic to the next component that executes the *PreparedStatement* and retrieves the *ResultSet*

10. The Facade method that creates the *Response Java Bean (RSJB)* is called. Each RSJB has a property called *responseCode* that represent the response code of the operation performed against the DB or on the server e.g. success, error, other value, exception on the server. There is a 1:1 mapping between the *responseCode* property, the operation status code given by the stored procedure execution, and the messages displayed on the mobile device. This is done by sharing the server side response codes with the mobile client. As mentioned in the design section a simple status code (3 digit number) can be received and translated into an user friendly message on the mobile device.

11. The RSJB parameters are set and the worker calls the corresponding Facade method for setting the *HTTPStatusCode* and the header information to be sent to the client. Then, the *RSJB* is serialized to the network using the *writeBean()* method.

12. A response is received on the mobile client from the server side. If the communication is successful i.e. *status code* is *HttpConnection.HTTP_OK* the client opens a *DataInputStream* and reads the RSJB. The *readBean()* method of the RSJB creates a new RSJB object that is a 100% identical copy of the original. The RSJB properties are retrieved using the corresponding *getX()* methods and the UI is updated. An info message is generated and displayed based on the RSJB *responseCode*.

General considerations about the Java Beans used during the communication protocol:

The RSJB's are created using an *inheritance* relationship among them. A superclass RSJB having only one property i.e. the *responseCode*, the corresponding *setX()*and *getX()* methods, and the *readBean()/writeBean()* for writing/reading the status code to/from the network, is implemented.[9]. All other RSJB's corresponding to the previously mentioned eight types of request extend the *Response Msg Bean* by adding extra properties, *setX()* and *getX()* methods for the new properties, and by *overriding* the *readBean()*, *writeBean()*, *toString()*, and *equals()* methods. The *readBean()* method returns a *Response Msg Bean* object that can be casted to the appropriate type on the client side if necessary. This provides flexibility to the application and allow source code refactorization on both client and server sides. The implemented RQJB and RSJB are depicted in the *Appendix D1.*

Besides the primitive types used as properties in the RQJB/RSJB there are several properties of other data types then primitive, used for containing complex data that is to be sent/received e.g. *1-dimension and 2-dimension arrays of primitive types*, and *byte array* for sending/receiving encrypted data or movie posters. There is no default read/write method on these types. Several solutions are considered.

The chosen solution consists in adding an extra property i.e. the *array length* to the Java Bean. This property is write/read to/from the network, too. When the array has to be serialized/deserialized the array length is read first. Then, all primitive elements in the array are written/read using the corresponding write/read methods for the primitive types in a *for loop.*

In case of a *byte array*, the *byte array length* is set as a property in the RQJB/RSJB. If the byte array property e.g. *movie poster* in the *Movie Details Response Bean* is written to the network, first the *moviePoster.length* property is written, followed by the whole movie poster

```
dataStream.writeInt(moviePoster.length);
dataStream.write   (moviePoster);
```

If the *movie poster* property in the *Movie Details Response Bean* is read from the network, the *moviePoster.length* property is read first, a movie poster byte array object is created with a dimension equal to the poster length, followed by reading fully the movie poster in the newly created byte array object, and setting the movie poster property.

```
byte[] poster = new byte[dataStream.readInt()];
```

[9]This is the Response Msg Bean

```
dataStream.readFully(poster);
movieDetailsBean.moviePoster = poster;
```

6.4.3 Server Side Components

Server Side Components Overview
The Server Side Cinema Service contains several components for dealing with the requests sent from the mobile device.

The *Cinema Central Controller* and *Cinema Workers* components are implemented by using *Java Servlets*. They extend the *HTTPServlet* class. Due to the *HTTP communication* choice between the mobile device and the server side the *HTTPServlets* are the best option.

Request and Response Java Beans are created for receiving/sending data from/to the mobile device following a strict data structure.

Other Java Beans are created for different operations performed by the worker servlets. The methods that perform the required business logic are available to the *Workers* via a *Facade design pattern* where an interface is made available to the caller and all business logic is hidden behind that. Among these java beans one can find the following components:

- **Encryptor** - used for encryption/decryption operations
- **RequestDataModel** - for retrieving the request data out of the request beans and setting up the stored procedure parameters
- **ResponseDataModel** - parse the formated SQL result and creates the RSJB to be sent to the client
- **SQL Operations Bean** - performs different SQL operations e.g. setting stored procedure parameters, obtaining the connection pool, creating and executing the *PreparedStatement* that contains the given stored procedure
- **Servlet Operations Bean** - set the HTTP request, response objects, get the RQJB from the request, set the response header parameters and sends the RSJB to the client
- **Other helper classes** - for performing different operations. Besides these components a *credit card validation* and *movie location services* components are defined.

The application also defines several *constant classes* such as:

- **Error Code Constants** - deals with error occurred in the server application. This class is also available on the mobile device.

- **Parsing Constants** - constants for different parsing operations
- **Protocol Step Constants** - identifies the client-server communication protocol steps. This class is also available on the mobile device.
- **SQL Return Codes** - declare all operation status codes returned by executing the DB stored procedures. This class is also made available on the mobile device.

Based on the *Error Code Constants* and *SQL Return Codes* the info message to be displayed to the client is created on the fly based on the operation status code received from the server. The communication bandwidth and money issues are addressed in this case.

A *Cinema Service Exception* class is defined for Cinema System specific exceptions. It defines exceptions that contains a message, the class/method/position where the exception is thrown, and the value accordingly to the Error Code Constants class. These exceptions are thrown by the Java Beans or helper classes and caught inside the *Controller* or *Workers* Java Servlets.

The Cinema Central Controller Servlet
It initiate a *custom logging category* based on log4j for logging purposes as mention in Section 6.4.1. All requests coming from the mobile device are serviced by the *doPost()* method. Inside this method the request is processed by determining the type of request based on the *Protocol Step Constants* class. A *RequestDispatcher* object pointing to the corresponding worker servlet is created and the request forwarded to that worker. The *forward(...)* method takes 2 parameters i.e. the request and response objects, respectively. Any exception that might be thrown by the workers is caught and logged in here. The response code is set to the error code value of the caught exception.

The Generic Worker Servlet
It defines the generic worker servlet i.e. the super class for all worker servlets. The template methods provided in here i.e. *getRequestBean()*, *setSQLStatement()*, and *parseSQLResponse()* must be implemented as hook methods in the concrete worker classes. The implementation is based on the *Template Method*, *MVC*, and *Facade* design patterns.

The following business logic is performed inside the *Generic Worker Servlet* by calling methods made available via an interface based on the Facade design pattern, as mentioned in the previous section.

- initialize the *Connection Pool* using the *JNDI* in the *init()* method

- deserialize the *Request Java Bean*
- sets the stored procedure name to be executed and the parameter list
- executes the stored procedure by assigning it to a *PreparedStatement*
- retrieves and parse the SQL result
- construct the *Response Java Bean* out of the parsed SQL response
- serialize the *Response Java Bean* to the network

All client requests are serviced via the *doPost()* method.

The Concrete Worker Servlets
There are ten concrete worker servlets i.e. *Background Hall Update*, *Cancel Tickets*, *Change Password*, *Find Movies*, *Movie Details*, *Purchase Tickets*, *Rate Movie*, *Reject Payment*, *Select Deselect Seats*, and *Select Show* Servlets. The source code for this classes can be found in *Appendix D2*. Each worker servlet serves a particular request e.g. *Change Password Servlet* is assigned to a *Change Password request*. There are cases when more than one worker serves the same request i.e. *Select Show*, *Select Deselect Seats*, and *Purchase Tickets* Servlets service the *Reserves Tickets request*.

All worker servlets extend the *The Generic Worker Servlet* and they must implement the template methods provided in the generic servlet i.e. *getRequestBean()*, *setSQLStatement()*, and *parseSQLResponse()* as hook methods. This approach is based on the *Template Method* design patterns.

The worker servlet calls the *init()* method of the super class in order to get access to the *DataSource* object for retrieving the *Connection Pool* object. All client requests for the given request are dealt with in the *doPost()* method by calling the *doPost()* method of the superclass. The hook methods are implemented. The *getRequestBean(Object requestBean)* accepts as parameter a *RQJB Object* and returns a RQJB casted to the appropriate request type e.g.

```
protected Change_Password_Req_Bean getRequestBean(Object requestBean){

Change_Password_Req_Bean chgPswdBean =
(Change_Password_Req_Bean) requestBean;
return chgPswdBean;
}
```

The *setSQLStatement()* method sets the name of the stored procedure to be executed against the DB. The query to be used by the *PreparedStatement* object is created e.g.

```
protected String setSQLStatement() {

return "SELECT * FROM cinema.Change_Password(?, ?, ?)";
}
```

The *parseSQLResponse(Vector sqlResult)* method calls another method to parses the SQL result and create the RSJB to be sent to the client by using the interface provided via the *Response data Model Facade* e.g.

```
protected Response_Msg_Bean parseSQLResponse(Vector sqlResult) {

FacadeToModel facade = new FacadeToModel();
return facade.setResponseChangePasswordBean(sqlResult);
}
```

In case of the Purchase Tickets request, *Reservation ID* and *Ticket ID's* are generated on the server side as random UUID's that are unique along all JVM's in the *RequestDataModel* class. UUID's are 128-bit **U**niversally **U**nique **ID**entifiers. Alternative solutions can be considered as part of a future work as mentioned in section 9.

```
public String generateUUID(){
String uuid = UUID.randomUUID().toString();
uuid = uuid.replaceAll("-", "").replaceAll(":", "");
}
```

Request and Response Java Bean
The RQJB and RSJB are depicted in the previous section. The beans source code can be found in *Appendix D1.*

6.4.4 Concurrency Issues

Making the Controller and Worker Servlets Thread-Safe
A Java Servlet has two primary methods that service the request i.e. *doGet()* and *doPost()*. When the servlet is compiled, one single method that incorporates the whole business logic is generated i.e. *service()*. This method must be thread safe.

The Tomcat servlet container creates, executes, and disposes the servlets. It checks if an instance of that servlet is running, and if not it loads the servlet, create an instance of it, and initialize it by calling its *init()* method. If the container has to dispose the server, the servlet *destroy()* method is called. In case a client request is received, the servlet *service()* method intercepts the request, perform the implemented business logic and sends a response back to the client on the same communication channel. This is called the *Servlet Life Cycle*.[15] Each servlet has only one instance loaded into a JVM during its life cycle.

Tomcat allows multiple threads to use that servlet instance simultaneously. Because a servlet has only one instance in a JVM this can cause concurrency problems when two or more requests access the servlet instance and write/read instance class variables defined in the servlet.

There are several solutions to this issue i.e. synchronizing the access to the instance variables or have unique variables for each request thread.

The first solution involves partial or total synchronization of the block accessing the class variables i.e. only one thread can access that variable at a time, while the others have to wait until the first thread exits the synchronized block or method. It is not a very good idea to synchronize the whole method but only those critical parts of the method due to performance issues. This can be done by using a *mutex(mutual exclusion)* variable shared by all threads.[11]

The second solution consists in declaring the *servlet as thread-safe by implementing the SingleThreadModel interface*. This tells Tomcat that only one thread can access the servlet *service()* method at a given time. This is a very expensive solution from the performance point of view and it is not acceptable.

The third solution i.e. *unique variables for each thread* can be implemented by *changing the instance variables to local ones i.e. moving them from the class to the methods*. Now, for each incoming request a new variable is created. The problem with this implementation is that there might be other methods that were using the class variable and now they do not have access to it. *The solution*

to this issue consists in changing the method declaration and pass that variable as a parameter in the method. This is the chosen solution for dealing with the servlet concurrency issues in the Cinema Ticket Reservation System.

Using the pgSQl Driver in a Multithreaded Servlet Environment
The pgSQL driver is thread safe. If a thread tries to use the connection, it is paused until the current operation is over. [21] Thus, one does not need to take any extra measures about concurrent DB access issues when using the pgSQL driver inside Java Beans and other application. Performance problems occur when used with Servlets. If several threads are performing queries against the DB, all have to be paused but one. The solution to this issue consist in using connection pool to the DB with servlets. Every thread uses its own connection to query the DB. The connection is marked as busy and released back into the pool once the thread has finished. Dead connections can be removed from the pool by the pool manager. There is also a performance issue in case of using connection pools i.e. it increases the load on the server because a new session is created for each Connection object.[21] Therefore care must be taken when tunning the connection pool settings. The optimal performance is obtained when the pool is in a stable state i.e. has enough connection to service all concurrent requests without creating new physical connections. [2] The pool can be tuned up to increase its size over time by dynamically creating new physical connection during rush hours and reducing its size outside the rush hours.

6.5 Database Implementation

Several open-source databases can be used for making the Cinema Ticket Reservation System data persistent. MySQL and PostgreSQL are the ones taken into consideration. There are different pros and cons of using either one RDBMS or the other. Use of foreign keys, sub-selects, transactions, procedural language for stored procedures(pl/pgsql), full joins, constraints, cursors, and views make PostgreSQL the choice for DB system implementation of the Cinema Ticket Reservation System. MySQL lacks on the previous mentioned features even if its performance is better then pgSQL. Similar performance can also be obtained with pgSQL but one has to take advantage of prepared statements. Also, pgSQl is more flexible and the data storage mechanism is thread safe. MySQL storage mechanism is thread safe only if InnoDB is used. Both DB's support JDBC, ODBC, indexes, have reach data types, and use large objects i.e. LOB and OID. [21], [9]

Different challenges are encountered during the implementation process such as storing movie posters in the DB, JDBC driver version issues, date objects, sending array as parameters to a stored procedure via JDBC, connection pool set up, etc. All these issues are analyzed and the chosen solutions argued. The chosen JDBC implementation solution considers the system performance as a first priority.

6.5.1 Creating Entities

The *Relational Data Model* diagram depicted in section 5.1.3 is implemented using PostgreSQL. The SQL queries for creating the relations, and inserting data into the DB can be found in *Appendix D3*. This section depicts the DB implementation and exemplifies with several entity sets.

Table Creation
A schema called *Cinema* is created and all entity sets are added to this schema. Every entity set contains constraints e.g. primary and foreign keys, not-null constraints, attribute constraints, referential integrity constraints, domain and general constraints.

All tables in the DB are created based on the following pattern:

```
CREATE TABLE table_name (

- Attribute Declaration

- Constraint List Declaration:
   - primary key
 - foreign keys
 - referential integrity constraints
 - domain constraints
 - single value constraints

);

Declaration of Singleattribute Indexes;
Declaration of Multiattribute  Indexes;
```

As an example, the *Cinema Halls* relation is depicted below:

```
CREATE TABLE CinemaHalls (
CinemaID  SERIAL,
HallID    VARCHAR(20),
Rows      NUMERIC(2, 0),
Cols      NUMERIC(2, 0),

  --  KEY      CONSTRAINT DEFINITIONS
  CONSTRAINT CinemaHalls_PK_CinemaID_HallID  PRIMARY KEY (CinemaID, HallID),
```

```
  CONSTRAINT CinemaHalls_FK_HallID
     FOREIGN KEY (CinemaID) REFERENCES Cinemas (CinemaID)
    ON DELETE CASCADE
    ON UPDATE CASCADE,

  --  CHECK    CONSTRAINT DEFINITIONS
  CONSTRAINT CinemaHalls_CHECK_Rows    CHECK (Rows > 0),
  CONSTRAINT CinemaHalls_CHECK_Cols    CHECK (Cols > 0),

  --  NOT NULL CONSTRAINT DEFINITIONS
  CONSTRAINT CinemaHalls_NOT_NULL_Rows CHECK (Rows IS NOT NULL),
  CONSTRAINT CinemaHalls_NOT_NULL_Cols CHECK (Cols IS NOT NULL)
);

--  INDEX DEFINITIONS
CREATE INDEX CinemaHalls_IDX_CinID_HallID
ON CinemaHalls (CinemaID, HallID);
```

Primary Keys Constraints, Many - Many Relationships, and Weak Entity Sets A *primary key* can be made either of one or many attributes e.g.

```
CinameHalls table (CONSTRAINT CinemaHalls_PK_CinemaID_HallID
 PRIMARY KEY (CinemaID, HallID))
```

contains the primary key of the *Cinema* table and the *HallID* primary key of the *Cinema Halls* table. It is used to uniquely identify a tuple in the table therefore the primary key has to be unique by definition. A multi attribute primary key is made of all primary keys of the tables involved in the many-many relationship. The *Show Discount* entity set is an example of a many - many relationship between *Discount Schema* and *Shows*. The primary key is formed by the primary keys of both entity sets involved in the relationship

```
CONSTRAINT ShowDiscount_PK_ShowLoc_ShowTime_DiscID
 PRIMARY KEY (ShowLocationID, ShowTimeID, DiscountSchemaID)
```

The first two attributes of the key come from the primary key of the *Shows* entity set, while the last one comes from the *Discount Schema*.

A weak entity is an entity where a tuple cannot be uniquely identified based on the existing attributes. Therefore, some other entity sets come with their

primary keys and form the primary key of the weak entity set to uniquely identify a tuple. *Cinema Halls* is an example of a weak entity set. The tuples cannot be uniquely identified based on the *HallID* because this represents a name given to the cinema hall e.g. Night Hall, Xtreme Surround, etc. One can find the same *cinema hall id* in more than one cinema. Therefore the primary is key is made of the *Cinema* primary key and the *HallID* attribute.

Foreign Key Constraints, One - Many Relationships, and Referential Integrity

The foreign key constraints are the second most important kind of constraints and state that certain attributes must make sense. [3] A foreign key occur in an entity set as a result of a *one-many relationship* where the primary key from the *one* side is translated into a foreign key on the *many* side. As an example, the *Show Location* entity set is involved in a one-may relationship with the *Movies* and *Cinema Halls* entity sets. The primary keys in both *Movies* and *Cinema Halls* are translated into foreign keys into the *Show Location* entity set.

```
CREATE TABLE ShowLocation(

...

  CONSTRAINT ShowLocation_FK_CinemaID_HallID
   FOREIGN KEY (CinemaID, HallID)
REFERENCES CinemaHalls(CinemaID, HallID)
   ON DELETE CASCADE
   ON UPDATE CASCADE

  CONSTRAINT ShowLocation_FK_MovieID
   FOREIGN KEY (MovieID)
    REFERENCES Movies(MovieID)
 ON DELETE CASCADE
 ON UPDATE CASCADE

...

);
```

One can notice the *Referential Integrity - Cascade Policy* constraints enforced on both foreign keys. The referential integrity on the first foreign key implies two things i.e.

- if a tuple in *Cinema Halls* is deleted, the changes must be reflected in the

Show Location entity set by deleting all tuples from *Show Location* that reference the deleted tuple in *Cinema Halls* i.e. *ON DELETE CASCADE*

- if a tuple in *Cinema Halls* is updated, the changes must be reflected in the *Show Location* entity set by updating all tuples from *Show Location* that reference the updated tuple in *Cinema Halls* i.e. *ON UPDATE CASCADE*

There are two more *Referential Integrity Policies* i.e. *default policy* where all changes are rejected and the *Set-null policy*. The later one is also used in the DB implementation e.g. the *Shows* entity set. The *Shows* entity set is a weak entity set who's primary key is made of the primary keys of the supporting sets i.e. *ShowLocation*, *ShowTime*, and *Prices*. The set-null policy states in this case: if a tuple with a particular *PriceID* value is deleted in *Prices*, all tuples in *Shows* that referenced that *PriceID* value are updated by replacing that *PriceID* value with *null*. A *Dangling Tuple* is created in this case.

```
CREATE TABLE Shows(

...

  CONSTRAINT Shows_FK_PriceID FOREIGN KEY (PriceID)
         REFERENCES PricesPriceID)
      ON DELETE SET NULL
      ON UPDATE CASCADE

...

);
```

Not-NULL and Attribute-based Constraints
By enforcing this constraint tuples that contain a given attribute are not allowed to have that attribute equal to NULL. PostgreSQl does not allow this type of constraint but there is a work around this issue by using the *attribute based constraints* i.e. *CHECK constraints* where a condition following the *CHECK* keyword has to be fulfilled for every value of that attribute. This can be exemplified by using the *Discount Schema* entity set where the *Discount Type and Value* attributes are enforced to be different then null.

```
CREATE TABLE DiscountSchema(

...
```

```
  CONSTRAINT DiscountSchemaID_NOT_NULL_DiscountType
      CHECK (DiscountType  IS NOT NULL),

  CONSTRAINT DiscountSchemaID_NOT_NULL_DiscountValue
      CHECK (DiscountValue IS NOT NULL),

  CONSTRAINT DiscountSchemaID_CHECK_DiscountValue
      CHECK ((DiscountValue >= 0.0) AND (DiscountValue <= 1.0))

  ...
```

As one can see all constraints have names for easier management i.e. deleting or updating them. This can be considered as a best practice when using constraints.

Indexes
Indexes are very important for the overall performance of a DB. It can be very expensive to scan all tuples of a relation that match a certain condition without any index. [3] Indexes can speed up queries in which a value for the attributes defined as indexes is involved.[10] But, indexes make insertion, deletions, an updates more time consuming. Therefore, a trade off has to be accepted when designing the DB. Indexes can be defined both on single and multiple attributes. By default a unique index is defined on any primary key in pgSQL.

Due to a limited no of updates in the Cinema DB and an extensive no. of queries against the DB the searching performance is considered high priority[11]. Therefore indexes are created for the most frequently attributes used in queries.

```
CREATE TABLE Users (
...
);

  CREATE INDEX Users_IDX_OTP         ON Users (OTP);
  CREATE INDEX Users_IDX_Credentials ON Users (UserName, Password);
```

Views
There are pros and cons of using views. Views can give simple and immediate

[10] if an index is defined on multiple attributes the search has to be performed in the same attribute order

[11] Most of the system updates are conducted every Sunday night. The only inserts and deletions occur when a user makes/cancels a reservation

results, encapsulate very complex calculations and commonly used joins. On the other hand, views are difficult to track, optimize, document, maintain, and do not improved query performance. It is also difficult to keep up with source application changes. The Cinema DB does not take advantage of views. It uses stored procedures due to extensive client-server interaction with a number of input/output parameters. It is author's choice not to use views in the stored procedures due to the complex joins that may be involved in view creation. Another approach is considered by using a more structured definition of the stored procedures.

Storing Large Objects in the DB
There are three solutions for storing large objects in the DB such as the movie posters.

The first approach uses an attribute of type *OID*[12] for storing large files in a DB. Based on the LargeObject API images can be stored and retrieved within a transaction mode. The procedure for inserting a large object into a DB involves several steps

- set a transaction
- create a Large Object Manager to perform all operations with
- create a new the large object
- open the newly created large object for writing
- open the image file and copy the data from the image file to the large object
- close the large object
- insert the row in the DB

The process of retrieving the large object from the DB has the same level of complexity. Therefore, this is not the chosen solution.

The second solution stores the images on the hard disk as files and saves in the DB the path to that image. Thus, every time a movie poster is required, the DB is queried for the path to the movie poster and the file is read from the hard disk and send to the client. This is not a very optimal solution due to the increased access time for reading files from the hard disk rather then a DB. Due to this issues, this is not an optimal solution for the DB implementation.

[12] Object Identifier

The chosen solution stores the movie posters as *BYTEA* in a DB table. The poster is inserted in the DB by reading the image bytes in a stream and transferring them to the corresponding Response Java Bean property. Retrieving a *BYTEA* image is even easier. Due to the API simplicity and improved performance, this approach is considered the chosen solution. The *Movies* table definition is depicted below.

```
CREATE TABLE Movies (

MovieID    SERIAL,
MovieName  VARCHAR(30),
...
Poster     BYTEA,
...

);
```

The Java implementation can be seen in the *MoviePosterHelper.java* and *DBConnTool.java* classes. A *Movie Details* stored procedure is created to query the DB. (*Appendix D3*)

6.5.2 Stored Procedures and Transactions

Stored Procedures
As mentioned in 6.5.1 stored procedures are used due to extensive client-server interaction with a high number of parameters for sending/retrieving data to/from the client side, respectively. *PL/pgSQL* is used as procedural language for implementing the stored procedures used in the cinema system. A procedural language for defining stored procedures reduces thr network and communication overhead due to extensive querying from the client side. The stored procedures have numerous advantages:

- they are pre-compiled - they increasing the system performance by running faster
- they are stored in the DB, therefore they can have direct access to the data
- they can controll transaction and embed all the business logic inside them reducing the faulty business logic in the client applications
- if many SQL statements are embedded into a stored procedure they are all executed at one time instead of one SQL statement at a time. [8] This can improve the performance and flexibility of the application.
- Exceptions can be thrown and caught in a stored procedure
- Transactions can be dealt with inside a stored procedure
- PL/pgSQL is a standalone programming language that has access to all pgSQl data types, operators an functions

In case of the Cinema Ticket Reservation System, a stored procedure is defined for each client-server protocol step e.g. an *Authenticate stored procedures* for the authentication use case, a *Rate Movie stored procedure* for the request with the same name, etc. Some of the stored procedures are used inside of more complex ones e.g. the *Authenticate* stored procedure is used inside the *Cancel Tickets* stored procedure for user authentication before canceling any tickets. This proves that code refactoring can also be used when implementing stored procedures.

All stored procedures used in the cinema system follow a strict pattern. They are depicted in *Appendix D3* The lines preceded by a double dash are comments in the pattern.

```
SET search_path TO cinema, public;
SET DATESTYLE TO ISO;

CREATE TYPE output_data_type AS(
...
);

CREATE TYPE user_defined_data_type AS(
...
);

CREATE OR REPLACE FUNCTION fnct_name(list_of_input_parameter_types)
 RETURNS output_data_type AS '

  DECLARE
      ...

  BEGIN
     [statements ...]
  END;

' LANGUAGE 'plpgsql';
```

The schema used for storying the relations and the ISO date format are set. Then, the output data type is created if there is a need to return a more complex structure e.g. result sets. Other user defined data types are created to store the result of a query, for example. The stored procedure is then declared together with the input and output parameter types by using the *CREATE OR REPLACE FUNCTION* construction. Any data declarations are performed inside the procedure after the *DECLARATION* keyword. The stored procedure business logic is represented by blocks of statements (SQL, conditional, etc) placed between the *BEGIN* and *END* keywords. These statements can include calls to other stored procedures which results can be used inside; loops on tuples and cursors can be used. The result of a stored procedure is returned using the *RETURN* keyword.

The list of all stored procedures implemented in the Cinema DB is depicted below:

- **Authenticate** - authenticates users and returns a status_code i.e. user authenticated(201) or user not authenticated(401). It accepts 2 input arguments i.e. *UserName* and *Password*.

- **Authenticate_E_Money** - authenticates the user and returns a status code (*221 - OK*, or *401 - user not authenticated*) followed by the amount of e-money in user's account in case status_code = 221. It accepts 2 input arguments i.e. *user_name* and *password*.

- **Background_Cinema_Hall_Update** - finds all booked seats, ticket base price, discount values, and all booked seats for a given show. It accepts 2 input arguments i.e. *ShowLocationID* and *ShowTimeID* that identify a show. It returns a status_code (*204 - show found*, *404 - show not found*) followed by a *list of all booked seats* if *status_code = 204*.

- **Cancel_Tickets** - checks if user is authenticated and in that case cancel the given reservation or ticket(s) that have been purchase by using the *CARD payment method*. Refund those tickets by means of e-money. It accepts as input 4 arguments i.e. *UserName*, *Password*, *ReservationID*, and an *array of [TicketIDs]*. It returns a status_code i.e. *214 - tickets canceled*, *201 - user authenticated*, *401 - user not authenticated*, or *414 - error*. This stored procedure is implemented in a transaction mode.

- **Cancel_Unpaid_Tickets_Before_Show** - cancels all reserved tickets by using the *Pay at the Cinema method* that have not been purchased yet. This is done by a background thread on the server side within 30 minutes before the show It returns a status code i.e. *415 - error*, *215 - all unpaid tickets for the corresponding shows are canceled*. This stored procedure is implemented in a transaction mode.

- **Change_Password** - accepts 3 input arguments: *UserName*, *Old_Password*, and *New_password*. It returns a status code i.e. *402 - user not authenticated*, or *202 - password changed*.

- **Compute_Price_And_Maybe_Pay** - checks if the user is authenticated and in that case computes the price for each ticket and the final price to be paid. Only if the payment method is CARD pay for the tickets. It saves user's Ticket ID's and reservation ID in the DB. It accepts as input 7 arguments i.e. *UserName*, *Password*, *ShowLocationID*, *ShowTimeID*, *an array of [SelectedRowNo, SelectSeatNo]*,*an array of [DiscountTypes]*, *reservationID*, *an array of [TicketIDs]*, *reservation_date*, *payment_method*, and *isCreditCardValid*. It returns a list of *RESERVATION_ID*, *TOTAL PRICE TO BE PAID*, *LEFT_E_MONEY*, *TICKET_ID's*, and *PRICE* for each TICKET_ID. It also returns a status_code i.e. *212 - price computed successfully and reservation saved*, *201 - user authenticated*, *401 - user not authenticated*, *412 - transaction error*, or *413 - invalid credit card*. This stored procedure is implemented in a transaction mode.

- **Display_Cinema_Hall_Conf** - finds the cinema hall configuration for the given show. It accepts 2 input arguments i.e. *ShowLocationID* and

ShowTimeID. It returns a status_code followed by a list of *base price*, *discount values[]*,*no. of rows for the cinema hall*, *no. of columns for the cinema hall*, and a *list of all booked seats* if status_code=203. The status_code values are: *203 - Show found according to the given criteria*, and *403 - Show NOT found according to the given criteria*.

- **Find_Movies_Criteria** - Finds a particular movie based on a given searching criteria. It accepts 4 input arguments i.e. *MovieName*, an *array of CinemaID's*, *city name*, and *show date*. It returns a status_code followed by a list of all Movies that are displayed by the given CinemaID's on the given date. The status_code is: *205 - Movies found according to the criteria* or *405 - Movies NOT found according to the given criteria*.

- **Movie_Details** - retrieves all info about a movie. It accepts as input 1 arguments i.e. *ShowLocationID*. It returns a status_code (*417 - error* or *217 - OK*) followed by the movie info.

- **Movie_Location_Service** - finds all cinemas in a certain range from user's given position. It retrieves the list of CinemaId's. It accepts 5 input arguments i.e. *range(¿=0)*, *movie name*, *street*, *city*, *zip*. It returns a status_code(*219 - Cinemas found* or *419 - Cinemas not found*) followed by a list of all found CinemaId's.

- **Rate_Movie** - checks if user is authenticated and if so, it rates the movie. If there is an entry for that user and movie in the rating table, update the user rating score. Else create a new entry. It accepts as input 4 arguments i.e. *UserName*, *Password*, *ShowLocationID*, and *UserRatingScore*. It returns a status_code i.e. *201 - user authenticated*, *401 - user not authenticated*, *418 - error*, or *218 - rating done* .

- **Reject_Payment_Cancel_Selected_Seats** - cancels all selected seats by user in case he/she does not want to accept the payment conditions. It accepts as input 3 arguments i.e. *ShowLocationID*, *ShowTimeID*, and an array of *[row, seat]*. It returns a status_code i.e *416 - error* or *216 - all seats are canceled*. This stored procedure is implemented in a transaction mode.

- **Select_Deselect_Many_Seats** - checks if the user's selected seats are still free when user presses *(DE)SELECT* in the Seat Selection form on the mobile device. It makes user's seat selection persistent in the DB. If user presses the *(DE)SELECT* button for the seats that he/she has just selected, it removes the previous selected seat from the DB. It accepts 5 input arguments i.e. CommandCode e.g "SELECT = 1 / DESELECT = 2", ShowLocationID, ShowTimeID, and an array of [row, seat]. It returns a status_code(210 - seat Selected, 211 - seat Deselected, 410 - error OR Seat Already Selected, 411 - error OR Seat Already Deselected) followed by a

list of all booked seats including the latest ones. This stored procedure is implemented in a transaction mode.

Transactions
Transactions can be created inside stored procedures. They are the solution to the *Serialization*(operations run serially) and *Atomicity*(certain combinations of DB operations need to be done atomically) DB issues.[3] Transactions group several operations in a group that must be executed atomically or not. Moreover, all operations inside transactions need to be run in a serializable mode.

Transaction can be created inside stored procedure by using the following construction *BEGIN - END* keywords that mark the stored procedure beginning and ending.

```
BEGIN

[statements to be executed in a transaction mode]

EXCEPTION
WHEN OTHERS THEN

[deal with the exception in here]

END;
```

Transactions are used in several stored procedures defined in the cinema DB such as *Cancel_Tickets*, *Cancel_Unpaid_Tickets_Before_Show*, *Compute_Price_And_Maybe_Pay*, *Reject_Payment_Cancel_Selected_Seats*, and *Select_Deselect_Many_Seats*.

6.5.3 Java DataBase Connectivity

As mentioned in session 6.4 Tomcat is used as web server and Sevlet container. All Java classes used for querying the DB are deployed on Tomcat. Several settings are performed to be possible to query the DB from Tomcat. The JDBC driver is made available to Tomcat for establishing connection and executing query against the pgSQL DB. The JDBC API provides access to connections, prepared statements, stored procedures, cursors, etc. A *Type 4 driver* is used i.e. it is written in Java, so it can be taken anywhere, and used anywhere as long as TCP/IP is available, because the driver only connects via TCP/IP. [8] The JDBC driver is made available to Tomcat by copying it under *$CATALINA_HOME/common/lib* folder, where *$CATALINA_HOME is the Tomcat installation folder*. This is the place where actually Tomcat looks for the DB driver when it is started. It loads the driver at every start up for all applications in its context.

JDBC Driver issues
During the implementation one major issues is identified when using the previous mentioned driver i.e. a *Missing dimension exception* is thrown when JDBC is calling a stored procedures that uses PL/pgSQL with an array parameter as input argument. The reason is the pgSQL JDBC driver does not have support for using *setObject* with array types. [21]. Using the 7.4 driver version instead of the 8.0 driver solves the issue. No functionality is lost in the system by downgrading to JDBC driver v7.4.

Simple JDBC Connection vs Connection Pool
Another performance issue occurred during the design phase of the system. How to connect to the DB and still have a good performance? There are two solutions for this issue i.e use a *simple JDBC Connection*, or use a *connection pool* from Tomcat. In order to increase the system performance, the second solution is chosen.

In case of a *simple JDBC Connection* every time a request is made to the DB, a connection is created, used for running the SQL statements and then disposed. It is known that most of the performance issues with this connectivity mode comes from the long time needed to establish a new connection for every request. This approach reduces the overall system performance.

In case of a *Connection Pool*, the system performance increases because Tomcat creates a connection pool at application startup and for each request to the DB a connection that is already crated is retrieved from the pool, used to query the DB, and returned back to the pool for later requests. One can notice the increased performance of the system due to the fact that for each DB request

an existing connection is used and not a new connection created every time. In case all connections are used there are two possible solutions:

- a new connection can be created and hand it in to the user
- the user can be put on a waiting state until a connection is returned to the pool

The connection pool has to be initialized in the *init()* method of the servlet by using JNDI to retrieve the *DataSource* object defined and in the previous .xml configuration files (context.xml, web.xml). The servlet retrieves a connection pool via the *getConnection()* method of the *DataSource* object.

- **context.xml** - it contains the settings for the Connection Pool such as: name of the accessed resource i.e pgSQl DB, type = DataSource, credentials to connect to the DB, JDBC URL to the DB, driver class name, max number of active connections, max number of idle connections, etc. More details can be seen in *Appendix E1.*
- **web.xml** - it contains the resource configuration for the connection pool and any servlet mapping. More details can be seen in *Appendix E2.*

Prepared Statements
A *PreparedStatement* is used for SQL statements that are executed multiple times with different values.[21]. If several find movie requests are sent by one client one after another, a prepared statement can increase the system performance as mentioned in section 6.5. Prepared statements are used in the Cinema System for executing all request depicted in section 6.5.2.

```
// set the SQL statement to be prepared and executed
// the 3 ? represent the parameters to be set up
String sqlStmt = "SELECT * FROM cinema.Change_Password(?, ?, ?)";

// get the connection from the pool
Connection conn = getPooledConnection();

// prepare the sql statement to be executed
PreparedStatement pgPsqlStmt = conn.prepareStatement(sqlStmt);

// set the parameter values for the prepared statement
pgPsqlStmt.setString(1, "my user name");
```

```
pgPsqlStmt.setString(2, "my old password");
pgPsqlStmt.setString(3, "my new password");

// execute the prepared statement
rs = pgPsqlStmt.executeQuery();

// get the results
...
```

The *executeQuery()* method of a prepared statement takes no arguments because a qyeru has already been assigned to a prepared statement. The result of a prepared statement execution is a *ResultSet* object. Parameters are passed to a prepared statement by using question marks instead of the parameter names, and then binding the values to those parameters. That can be done via methods such as *setDate(x1, y1)*, *setString(x2, y2)*, etc. It is very important to bind the right value and type to the given parameter i.e. the *y1* value has to be of type *java.sql.Date* while the *y2* value of type *String*.

The details of using the pgSQl driver in a multithreaded servlet environment are depicted in Section 6.4.4.

Chapter 7

Overall System Testing

Several methods are used for testing the overall *Mobile Cinema Ticket* system functionality. A systematic approach is used during the testing process. *Functional* and *structural* tests are performed. The application is split in 3 components i.e. client, server, and DB. Each of these components are tested separately and then the whole system is tested by testing the communication protocol among those three components.

It is important to perform both functional and structural tests. Some errors can be found with the functional tests, while others during the structural ones. While the functional test focuses on the output, the structural one checks the internal structure of the program i.e. if branches, for/while loops, switch cases, etc.

Moreover, a usability evaluation is performed with real users to identify any usability issues in the GUI. The usability test cannot be performed by using automated testing. Real users are needed to interact with the application and express their opinion about the controls, GUI look&feel, user friendliness, different tasks to perform, etc. Users try to perform several predefined tasks on the system by using different scenarios that reflect the real world tasks.

The DB is the first component tested. Functional and structural tests are executed against the stored procedures. Several test cases are built and the

expected results compared with the actual results obtained by executing the procedure with the given data. Several errors are found and fixed.

The server side is tested using a small web UI for sending GET request over HTTP. Different test cases are executed. Several errors are found and fixed. The DB and server parts are also tested during the client side implementation.

The client side is the last but not the least component tested. Functional and structural tests are used for this purpose. Several errors are found and fixed. A *usability evaluation* of the client UI and functionality is performed as mentioned in section 2.3.

More details about the overall system testing can be found in the following chapters.

The purpose of these tests is to reveal any errors in the system. When errors are found they are corrected and the tests performed again (including previous successful test, since the correction might have changed unexpected parts of the system). For this reason it does not make sense to document all performed tests in details - it is more the test strategy, that is important. The downside of this approach is that it cannot guarantee that the system works perfectly. A test performed over a long period of time can be more exhaustive. As, this application is just a prototype and the available time does not permit such a test, it is possible that some errors are still present in the application.

The system is not fully tested using structural testing. Functional and usability tests are performed against the whole application. The reason is that the author considers this application as a prototype that shows how such a system can be implemented and tested. But one has to keep in mind that it is only a prototype. A more detailed testing process is considered out of the project scope.

A more exhaustive test is feasible if the application is to be lunched on the market. In that case, automated tests can be performed for both structural and functional testing. Also, more users can be involved during the usability evaluation. There are different approaches to the testing methods depending on the available resources i.e. money, people, time in the real world. Some applications are heavily tested before being released on the market. Automated tests and special trained testers are used for that. Other applications are not that heavily tested. They are released to the user and errors are reported while users are using the application. From time to time, a new software update is released.

7.1 Functional Tests

The goal of the functional testing is to make sure that the program solves the problem it is supposed to solve. [20] Several test cases using input values and expected ones are created. The output given by the test case execution is compared with the expected one. If they do not match an error is found. These tests can be automated by creating a small script to execute all test cases and check the results.

Most of the functional tests in the cinema ticket application are performed visually. The GUI is tested in a similar way. Several errors are found and fixed.

Several examples of functional tests are depicted below. They show how the application can be tested using the functional approach. Eight test cases are used for the functional test of the Change password use case. The server is tested by logging different debug outputs and application states. The DB is tested by running the stored procedure and testing the output with the expected one. Operation status codes are also checked. On the mobile client, the test are performed both visually and using debug statements.

Test Cases for the Change Password Use Case:

1. The user name and password are correct. The user can be authenticated. The new password and the verify one match. Thus, the password is changed and an OK response is returned.

2. The user name is correct but the password is not. The user cannot be authenticated. The new password and the verify one match. Thus, the password is not changed and a *User not authenticated. Invalid user name or password* message is returned.

3. The user name is not correct but the password is. The user cannot be authenticated. The new password and the verify one match. Thus, the password is not changed and a *User not authenticated. Invalid user name or password* message is returned.

4. Both the user name and password are incorrect. The user cannot be authenticated. The new password and the verify one match. Thus, the password is not changed and a *User not authenticated. Invalid user name or password* message is returned.

5. The user name is correct but the password is incorrect. The user cannot be authenticated. The new password and the verify one do not match. Thus,

the password is not changed and a *Passwords do not match* message is returned.

6. The password is correct but the user name is incorrect. The user cannot be authenticated. The new password and the verify one do not match. Thus, the password is not changed and a *Passwords do not match* message is returned.

7. Both user name and password are incorrect. The user cannot be authenticated. The new password and the verify one do not match. Thus, the password is not changed and a *Passwords do not match* message is returned.

8. The user name and password are correct. The user is authenticated. But, the new password and the verify one do not match. Thus, the password is not changed and a *Passwords do not match* message is returned.

9. The user name and password are correct. Both passwords match. But, the new password and the verify have a size smaller then 8 characters. Thus, the password is not changed and a *The password has to be at least 8 characters long* message is returned.

10. The user name is correct but the or password is not. Both passwords match. But, the new password and the verify have a size smaller then 8 characters. Thus, the password is not changed and a *The password has to be at least 8 characters long* message is returned.

11. The user name is incorrect and the password is correct. Both passwords match. But, the new password and the verify have a size smaller then 8 characters. Thus, the password is not changed and a *The password has to be at least 8 characters long* message is returned.

12. The user name and password are correct. The passwords do not match. But, the new password and the verify have a size smaller then 8 characters. Thus, the password is not changed and a *The passwords do not match* message is returned.

13. The user name is incorrect and the password is correct. The passwords do not match. But, the new password and the verify have a size smaller then 8 characters. Thus, the password is not changed and a *The passwords do not match* message is returned.

14. The user name is correct but the password is incorrect. The passwords do not match. But, the new password and the verify have a size smaller then 8 characters. Thus, the password is not changed and a *The passwords do not match* message is returned.

15. The user name and password are incorrect. The passwords do not match. But, the new password and the verify have a size smaller then 8 characters. Thus, the password is not changed and a *The passwords do not match* message is returned.

No.	Input	Output	Expected Output
1	user = adm oldpassword = 87654321 newpassword = 12345678 verifpassword = 12345678	Password Changed Successfully	Password Changed Successfully
2	user = adm oldpassword = 87654321 newpassword = 34567890 verifpassword = 34567890	User not authenticated Invalid user name or password	User not authenticated Invalid user name or password
3	user = zzz oldpassword = 12345678 newpassword = 45678901 verifpassword = 45678901	User not authenticated Invalid user name or password	User not authenticated Invalid user name or password
4	user = qqq oldpassword = 87654321 newpassword = 45678901 verifpassword = 45678901	User not authenticated Invalid user name or password	User not authenticated Invalid user name or password
5	user = adm oldpassword = 87654321 newpassword = 45678901 verifpassword = 78901234	Passwords do not match	Passwords do not match
6	user = qqq oldpassword = 12345678 newpassword = 45678901 verifpassword = 78901234	Passwords do not match	Passwords do not match
7	user = qqq oldpassword = 87654321 newpassword = 45678901 verifpassword = 78901234	Passwords do not match	Passwords do not match

No.	Input	Output	Expected Output
8	user = adm oldpassword = 12345678 newpassword = 45678901 verifpassword = 78901234	Passwords do not match	Passwords do not match
9	user = adm oldpassword = 12345678 newpassword = 456 verifpassword = 456	The password has to be at least 8 characters long	The password has to be at least 8 characters long
10	user = adm oldpassword = 87654321 newpassword = 456 verifpassword = 456	The password has to be at least 8 characters long	The password has to be at least 8 characters long
11	user = zzz oldpassword = 12345678 newpassword = 456 verifpassword = 456	The password has to be at least 8 characters long	The password has to be at least 8 characters long
12	user = adm oldpassword = 12345678 newpassword = 456 verifpassword = 45678	Passwords do not match	Passwords do not match
13	user = zzz oldpassword = 12345678 newpassword = 456 verifpassword = 45678	Passwords do not match	Passwords do not match
14	user = adm oldpassword = 87654321 newpassword = 456 verifpassword = 45678	Passwords do not match	Passwords do not match
15	user = zzz oldpassword = 87654321 newpassword = 456 verifpassword = 45678	Passwords do not match	Passwords do not match

7.2 Structural Tests

As mentioned in section 7, the goal of the structural testing is to ensure a correct internal structure of the application i.e. all if branches are executed, the for/while loops are execute zero, at least once, or many time, the switch cases branches are reached, etc. The same approach is used as in case of the functional test i.e. a set of input test data is used together with the expected output. The outputs from executing the program based on the given test cases are compared with the expected ones. If there is a match, the test cases are passed.

An example of structural test conducted for the server application is depicted below. The *processRequest(...)* method of the *Cinema Central Controller Servlet* is used for this purpose. *Appendix D2* The if statements that are to be tested in this case are numbered from 1 to 13 in the source code. These statements are checking if the protocol step sent by the client has a match on the server or not. If yes, the url to the appropriate worker servlet is built. Otherwise, an exception is thrown and an error returned to the client.

Test No.	Input	Input Property	Output	Expected Output
1	AT1	Valid Protocol Step	Authentication Servlet 1	Authentication Servlet 1
2	AT2	Valid Protocol Step	Authentication Servlet 2	Authentication Servlet 2
3	CGP	Valid Protocol Step	Change Password Servlet	Change Password Servlet
4	MOV	Valid Protocol Step	Find Movies Servlet	Find Movies Servlet
5	SHW	Valid Protocol Step	Select Show Servlet	Select Show Servlet
6	BHU	Valid Protocol Step	Background Hall Update Servlet	Background Hall UpdateServlet
7	SDS	Valid Protocol Step	Select Deselect Seats Servlet	Select Deselect Seats Servlet
8	PTC	Valid Protocol Step	Purchase Tickets Servlet	Purchase Tickets Servlet
9	CCT	Valid Protocol Step	Cancel Tickets Servlet	Cancel Tickets Servlet
10	REJ	Valid Protocol Step	Reject Payment Servlet	Reject Payment Servlet
11	RTM	Valid Protocol Step	Rate Movie Servlet	Rate Movie Servlet
12	DET	Valid Protocol Step	Movie Details Servlet	Movie Details Servlet
13	TEST	INVALID Protocol Step	INVALID PROTOCOL STEP sent by the MIDLET	INVALID PROTOCOL STEP sent by the MIDLET

7.3 Usability Evaluation of the Mobile Client

As mentioned in sections 2.3 and 7, the usability evaluation is performed with real users that are asked to perform several predefined tasks on the whole system. This is done on the first version of the high fidelity prototype. The feedback received from the users is interpreted and used in the second version of the high fidelity prototype. The feedback received from the user contains among others:

- a new feature for changing the secure wallet PIN code is added
- the access time to the secure wallet and ticket manager is increased
- the error message windows look is improved
- etc.

Several scenarios are created and performed by all users. The scenarios must be simple because they reflect on the real world tasks.

1. **Save/delete/edit/view a credit card on the secure wallet** - it is important the user perform this task without problems.
2. **Book two cinema tickets for a movie tomorrow in Copenhagen as close as possible by your place** - this is the main task for the application. It is necessary that the user is able to perform this task without any problems.
3. **Change the application password** - a very simple task
4. **Cancel one ticket out of the previous two** - the user must be able to perform this task. It can be considered as one of the main tasks.
5. **View ticket details** - the user must be able to perform this task. It can be considered as one of the main tasks.
6. **Change the application theme**
7. **View the details about Spider Man 3**
8. **Rate the movie XMEN3**

After performing all given tasks, the users are interviewed. Several issues are revealed while they where performing the given tasks. There are several advantages to this type of testing such as:[18]

- The study is performed in a real-world situation.
- Issues are found that could not have been predicted.
- Issues that will occur when the product is in real use are discovered.

Unfortunately, the chosen method has some disadvantages, too. Being performed on a mobile device, it is time consuming, and the screen is to small to follow the user all the time. But this can be overcome by using the *think aloud* technique.[1].

[1] the user express his thought about the application with loud voice

Chapter 8

Market Perspective

This section depicts the *market perspectives* of the Cinema Ticket Reservation System. Nordisk Film, the biggest player in the danish cinema market, provides an online ticket reservation service i.e. *biobooking.dk*. Movie goers can browse movies based on the movie name or cinema, check movie details, select the desired seats in the theater, and purchase one or more tickets with different discount options.

The Mobile Cinema Ticket Reservation System provides the same functionality but on a mobile device. Moreover, the system is extended withe several extra features. The system offers a great deal of mobility to any movie goer.

The system can also be used by tourists who visit a city. They can received the application from the *Tourism Information Office* via bluetooth, download from the Internet or from a stationary PC in the office, free of charge. They also received a user name and password that can be changed later on. The system can be extended to provide a map service that shows the route from the current location of the movie goer to the desired cinema.

Several innovative ideas are introduces by the Mobile Cinema Ticket Reservation System

- **Bar code ticket** - The information embedded in the cinema ticket can

also be displayed in a bar code format and read at the cinema to pay for the ticket or get access to the show as mentioned in the previous sections. A different approach consists in using bluetooth when reading the ticket information in the cinema. The mobile phone and the ticket reader machine can be pair over bluetooth and exchange information. The issue with this approach is to limit the range of the bluetooth reader, avoid being spamed via bluetooth, or having your ticket information stolen by evil doers.

- **Social Network** - a network of all movie goers using this system. They can rate movies and read/write reviews for different movies or cinema/theaters. This network can also be connected to other movie DB systems such as *www.imdb.com*
- **The Secure Wallet** - movie goers can store their credit cards in a highly secure way. This feature can be used no only by a movie goer.

The Mobile Cinema Ticket Reservation System tries to improve the current service offered by the cinemas:

- it provides fast access to the movie DB
- it helps movie goers to avoid long queues and prior visits to the cinema for checking movies and purchasing tickets
- it helps movie goers to pick the closest cinema to the given position
- it allows movie goers to search for movies, purchase the tickets, and enter the shows - using one application.

8.1 Selling the Service

The main features provided by this service are similar to the ones offered by other web based services e.g. *biobooking.dk*. There several extra features that extend the existing services as mentioned before.

In order to create a successful service, special marketing actions need to be taken for promoting the service and making it available to all movie goers. 3rd party companies can advertise their company and products e.g. dynamically created company logo displayed on the ticket or small icons on the application forms. Pop-up windows can be displayed on the mobile screen based on the chosen cinema where different small shops e.g. Burger King, can advertise their products, etc.

Another way of making this service a success would be to extend the functionality mentioned in section 8 and allow users to gain access to the movies without a paper back ticket by using a ticket reader machines placed inside the cinemas. The movie goer can also use these machines to pay for the ticket or enter the show, as mentioned before.

Bluetooth hot spots can be placed in the cinema and the application enhanced with a bluetooth - aware feature. The cinema could broadcast movie trailers or other information and the mobile application could display them. One has to take extra care about spamming via bluetooth. Therefore a communication protocol between the mobile device application and the cinema hot spots has to be designed to avoid the spamming issues and misuse of the bluetooth technology.

By providing movie goers a secure user friendly application, a reach GUI, a very modern look and feel, and the unique functionalities mentioned in section 8, the *Mobile Cinema Ticket Reservation Service* can gain a large market.

Chapter 9

Future Work

Several decisions are taken during the analyze, design, and implementation phases of the Cinema Ticket Reservation System to fulfill the application requirements depicted in section 4.5. During the implementation process different solutions are considered and several improvements stated. Due to the limited time for implementing this prototype the author could not focus on including those improvement ideas into the current version of the prototype. They are considered as part of a possible future work on the project.

- When a list of all movies matching the given searching criteria is displayed, two new fields can be displayed on the UI i.e. the distance from the current position to the cinema, and a map, respectively as mentioned in section 4.4.
- When a list of all movies matching the given searching criteria is displayed, the *movie*, *cinema*, and *show hour* can display predefined default values such as: the latest movie, the closest cinema to the user, the first show hour when that movie is to be played in that cinema, etc. (section 4.5.2)
- The max no. of credit cards allowed to be saved in my wallet can be dynamically set up based on the mobile device memory characteristics. (section 5.2.2)

- A customized High-level Text Box look and feel can be implemented as a low-level component. Thus, an extension to the existing J2ME low level components can be created as mentioned in section 6.2.1.
- When displaying the credit card information or ticket details, *GameCanvas, TiledLayer or Sprite* J2ME components can be used for dealing with image optimization issues. The background can be created in this case dynamically by building the image out of small pieces as a puzzle. (sections 5.2.2, 6.2.8).
- A feature that allows movie goers to chage the application language can be implemented as described in section 6.2.9.
- The communication between the *Authentication Servlet 1* and *Authentication Servlet 2*can be implemented using *Java RMI* as depicted in section 6.3.2.
- In case of the Purchase Tickets request, *Reservation ID* and *Ticket ID's* are generated on the server side as random UUID's that are unique along all JVM's in the *RequestDataModel* class. UUID's are 128-bit **U**niversally **U**nique **ID**entifiers. Alternative solutions can be considered as part of a future work as mentioned in section 6.4.3
- A ticket bar code image embedding the whole ticket information can be generated on the server side and displayed on the mobile device. Different solutions have to be analyzed and a benchmarking has to be be performed between generating the bar code ticket on the client, and generating the bar code ticket on the server and sending it to the client, respectively.
- A list of all shows in a city can be cached on the mobile device. The mobile client has to determine the city where the movie goer is located and check the validity of the cache every time the application is opened.

Chapter 10

Conclusion

The prototype of a location - aware application for purchasing cinema tickets is implemented for GPRS-enable mobile devices. Both the client and the server side are implemented following the application requirements in section 4.5.

As a location - aware service[1], it displays a list of all shows (movies, cinemas, and show hours) in a certain range from the movie goer's current position. Movie goers can enter their position (city, zip, street). They can select a movie from a list, view details about that movie, rate the movie, or reserve/buy tickets for it. Movie goers can use several payment methods such as: pay by credit card, at the cinema, or e-money. If they decide to purchase the tickets they can pay using a previously saved credit card in a *My Wallet* feature. The movie goers can authenticate on the wallet using the PIN code. All credit card information is stored encrypted in the wallet. Once the tickets are reserved/purchased movie goers can view them using the ticket manager. Any purchased tickets can be canceled at any time and the money refunded as e-money. This money can be used to purchase other cinema tickets or goods inside the cinema.

The design process of this service has a user-center approach due to its targeted user group i.e. movie-goers that use mobile phones. Different low and high fidelity prototypes are developed and evaluated by real movie-goers. Brainstorming sessions, workshops, and interviews are conducted. The results are

[1] it can determine the movie goer's current position

interpreted and used in the next step of the iterative development process. An interactive process of design-evaluate-redesign is chosen.

An authentication mechanism based on the Needham-Schroeder Protocol coupled with strong encryption is implemented to give movie goers access to both the client and server side service. The sensitive data sent over the network i.e. credit cards, is encrypted with a key preshared between the client and the server. The user's sensitive data is stored encrypted on the mobile device e.g. credit card data, passwords, and PIN codes.

The mobile client is implemented using J2ME. A reach and dynamic GUI is developed providing movie goers with a powerful, user-friendly, and easy to use interface. Reach information screens inform the user on the status of different operations. The GUI layout is designed during the workshops and interviews conducted with real movie goers. One can say that it is designed by the users for the users. Optimization techniques are applied to increase the application performance. Different benchmarking tests are performed and the best solutions chosen.

A My Wallet feature is proposed and implemented for storying movie goers' credit cards. All credit cards are stored encrypted. Safe triggers that erase all credit card information in case an attacker tries to gain access to the wallet are implemented. Movie goers can save, delete, edit, or view any credit cards stored in the wallet. The application is designed to be used independently from the cinema service.

A server side service is implemented to respond to the client requests. Java Servlets, Java Beans, JDBC, and Bouncy Castle cryptographic libraries are used for this purpose. A postgreSQL DB is chosen for storying the system data. Stored procedures and java prepared statements are used for querying the DB. In order to increase the system performance, connection pools are preferred when connecting from a java servlet to the DB. Concurrency issues are addressed.

A unified communication protocol between the client and the server is created. Java Beans that can serialize/deserialize by themselves are exchanged. Different hardware and software limitations on the mobile client are addressed that way.

In order to improve the system scalability and management, design patterns are used when implementing both the client and the server e.g. MVC, Facade, Singleton, Template Method, Refactoring, Abstract Coupling, Iterator, etc. Several other refactoring methods are considered.

The system is optimized for best performance. Several optimization solutions are chosen for the client, server, and DB respectively.

The service and its components are tested using functional and structural tests. Several errors are found and corrected. The usability evaluation of the system is conducted with real users. Users are asked to perform several predefined tasks on the high fidelity prototypes. The feedback received is interpreted and used in the following version of the prototype.

The market perspectives of the cinema ticket reservation service are analyzed. Different solutions for making this service available to movie goers are proposed. Several choices for selling this service are discussed.

A future work section presents possible improvements to the system functionality and usability.

Altogether, the current prototype of the cinema ticket reservation service shows how a user centered approach can drive the design and implementation phases of any service while optimization techniques can increase the overall system performance. The current version of the prototype is considered a prof-of-concept that depicts how Human-Computer Interaction Design, Cryptography, Java, DB, and Marketing Strategies can be combined to create a successful mobile service.

Appendix A

Guidelines for the conceptual design workshop

1. **Inform users about**
   (a) **The goal of the study**
      i. The goal is to evaluate a paper based prototype of a mobile application used for buying cinema tickets on mobile phones via a GPRS connection.
      ii. Let the users participate in the design process.
      iii. Get to know any issues users might have with our specific ideas.
      iv. Specify new requirements for the project.
      v. Check if the user needs have been understood.
      vi. Check if the application fulfills the usability goals i.e. effective, efficient, safe, easy to use, easy to learn, and easy to remember.
      vii. Check if the application fulfills the user experience goals i.e. fun, entertaining, satisfying, helpful, motivating, aesthetic, supportive or creative, rewarding, and emotional fulfilling.
      viii. Check to ensure the application is user friendly.
      ix. Check to ensure the application is easy to use, easy to navigate from one screen to another, and the feedback provided to users is clear enough and gives sufficient information.

x. Check to ensure the data provided to the users is clear enough, sufficient and meet users' requirements.

xi. Check if the application is consistent, and has a minimalist design i.e. avoids using information that is irrelevant or rarely needed.

xii. Check to see if there is a match between the system and the real world.

(b) **The tasks users will be asked to fulfill**

i. Users are asked to go through several scenarios that simulate the interaction with the real application. User can use post-its and colored pencils to add their own ideas to the scenarios at any time.

ii. Users are asked to fill in two questionnaire forms -before and after the evaluation.

iii. Users are asked to answer to different questions regarding the previously shown UI

(c) **The amount of time needed for this study**

i. The user will be asked to take part in this study for at most 60 minutes.

(d) **The data that is collected**

i. The user experience and thoughts of using the prototype is recorded. His responses to the provided questionnaires will be also collected.

(e) **How this data is to be used**

i. The recorded data is to be analyzed, interpreted anonymously and feed back into the design process for an improved version of the prototype. Several design - evaluate - redesign cycles are to be used before starting the real implementation.

2. **Give users an overall description of the application**

(a) This experiment evaluates a paper-based prototype of an application for buying cinema tickets by using a GPRS enabled mobile phone. This is a location - aware application i.e. user enters his current position, a date for which he/she wants to find a movie and a range i.e. the radius of the area around him where he/she would like to search for movies/cinemas. A list of all cinemas and movies in the given range from the user's current position is sent back to the user. The user can select a movie that he/she wants to watch. The configuration of the cinema hall where the movie is played is displayed on the mobile device screen. The user can select and reserve any free seats. User reviews and trailers ca also be read/watch.

User can choose to pay for the ticket by using the mobile device or he/she can pay once arrived at cinema. A secure wallet feature is developed to allow ticket payments by using the mobile device. The secure wallet stores credit card information in an encrypted form and provides secure connections for making the payments. An authentication procedure is used to get access to the secure wallet content.
User can see all his/her previous made reservations and cancel any one of them.
User can also see all his/her previous bought tickets. He/She can delete any one of them.
A black list of people is to be built. The list is used to store people who do not cancel a previous made reservation in case they cannot attend the show and the ticket has not been paid.

3. **Inform users about the goal of the application they are to evaluate**
   (a) Evaluating a low fidelity prototype of the m-Cinema Reservation System
4. **Inform users about the ethical issues**
   (a) Users' name won't be revealed at any time;
   (b) The collected information is confidential;
   (c) Users can leave the experiment when they feel uncomfortable;
   (d) Users can ask questions during the experiment.
5. **Inform users about the data gathering mechanisms used in this study**
   (a) **Observing users**
      i. **The users will be observed while they are performing different tasks with the low fidelity prototype of the application. Notes will be taken and the users are asked to express their thoughts loudly all the time they interact with the prototype.**
   (b) **Conducting user interviews**
      i. Pre/post questionnaires need to be filled in by the users before/after the prototype evaluation.
      ii. Users are asked to answer to several questions regarding the previously evaluated UI

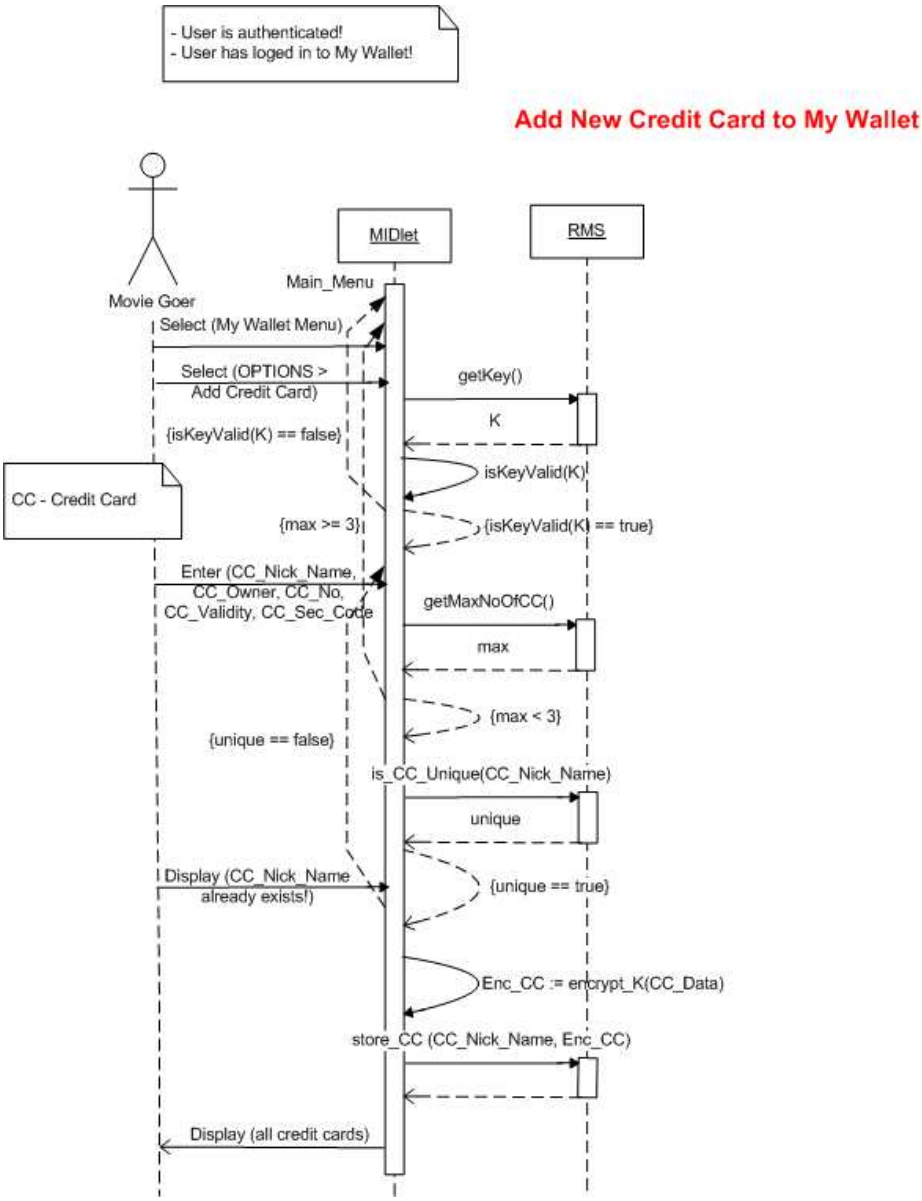
Appendix B

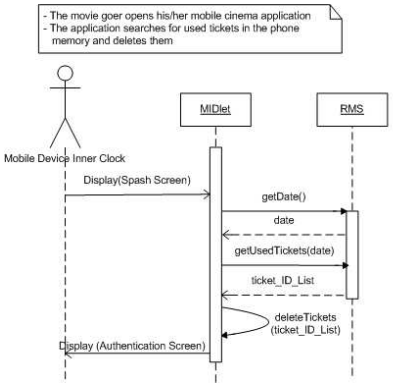
Conceptual Design Workshop Questionaires and Results

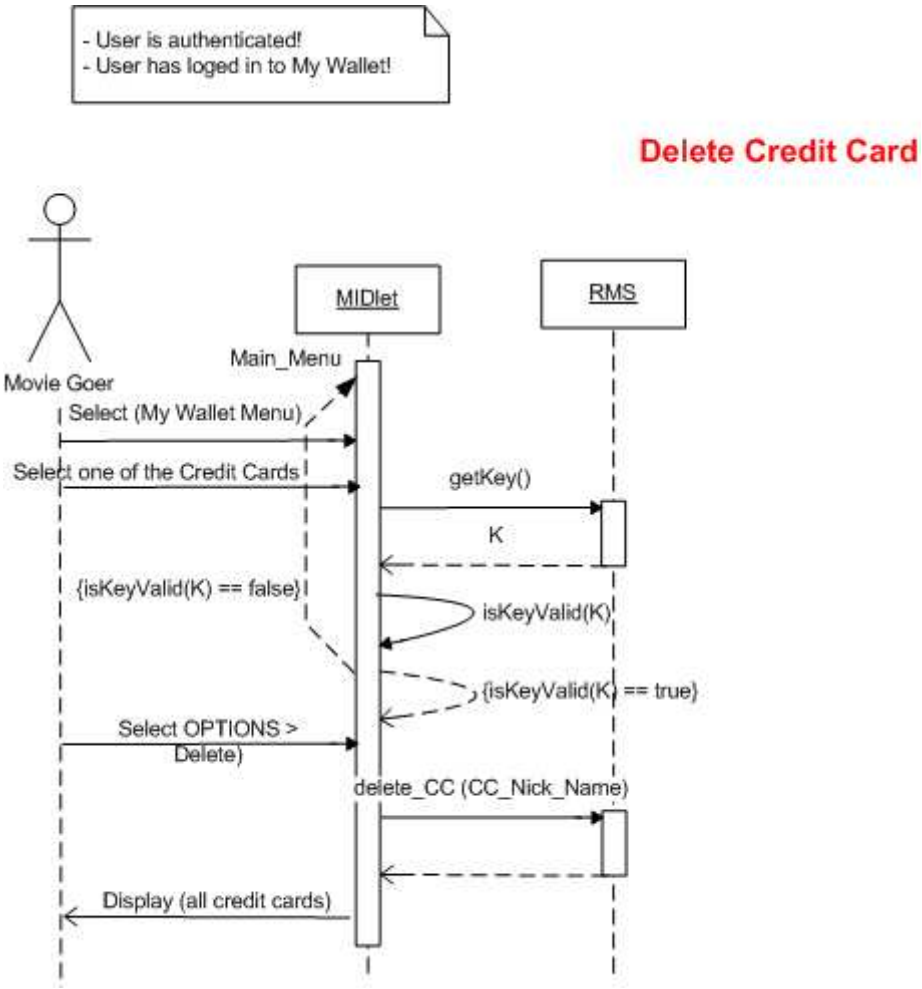
Appendix C

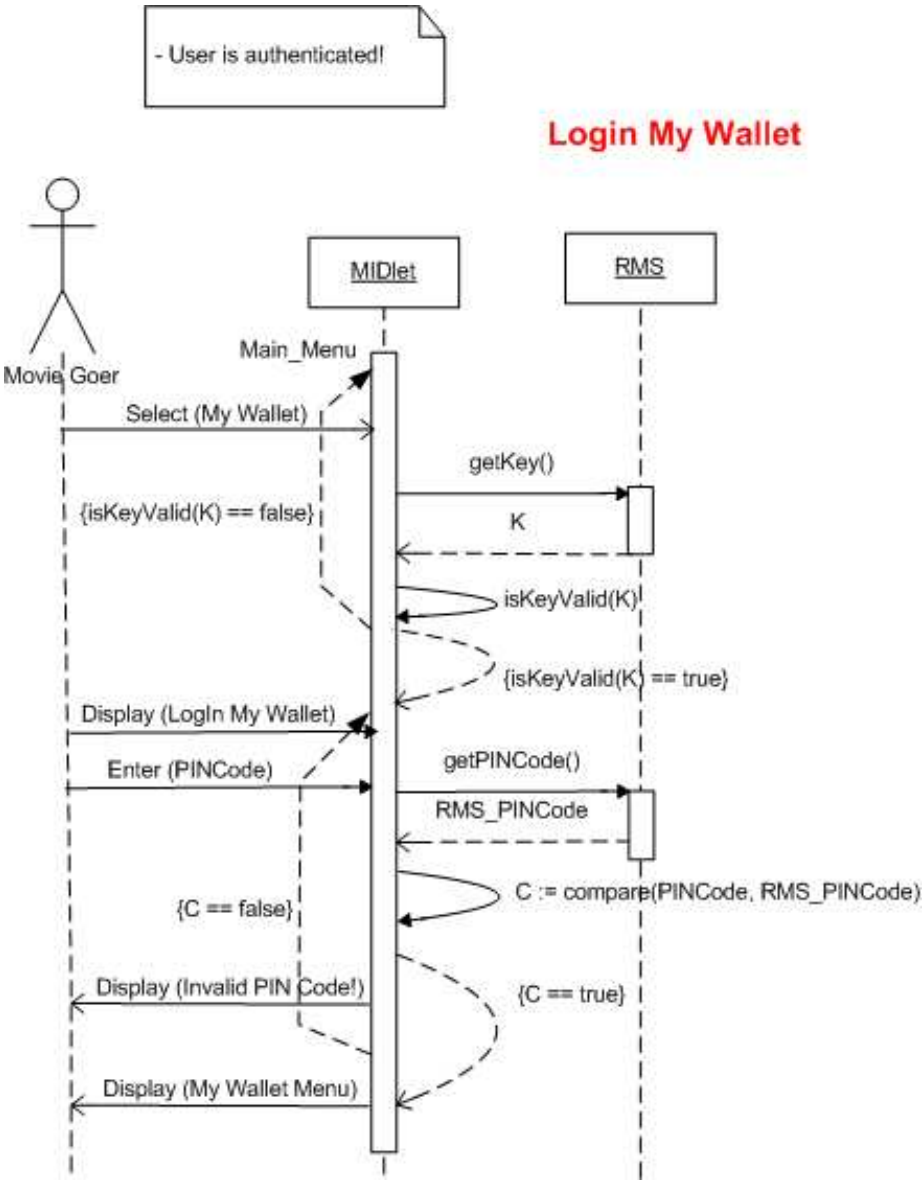
Sequence Diagrams of the System

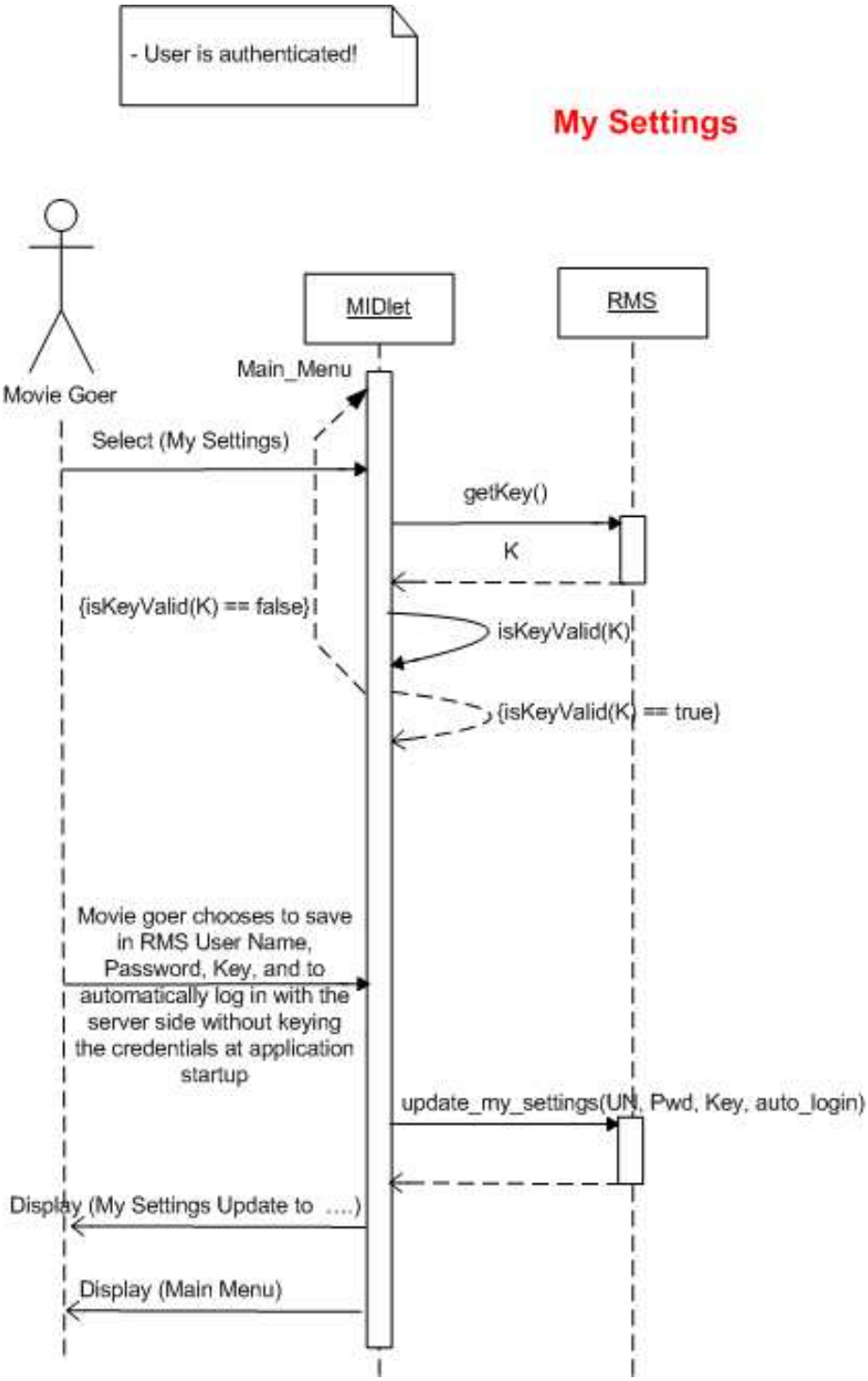
C.1 Mobile Client Sequence Diagrams

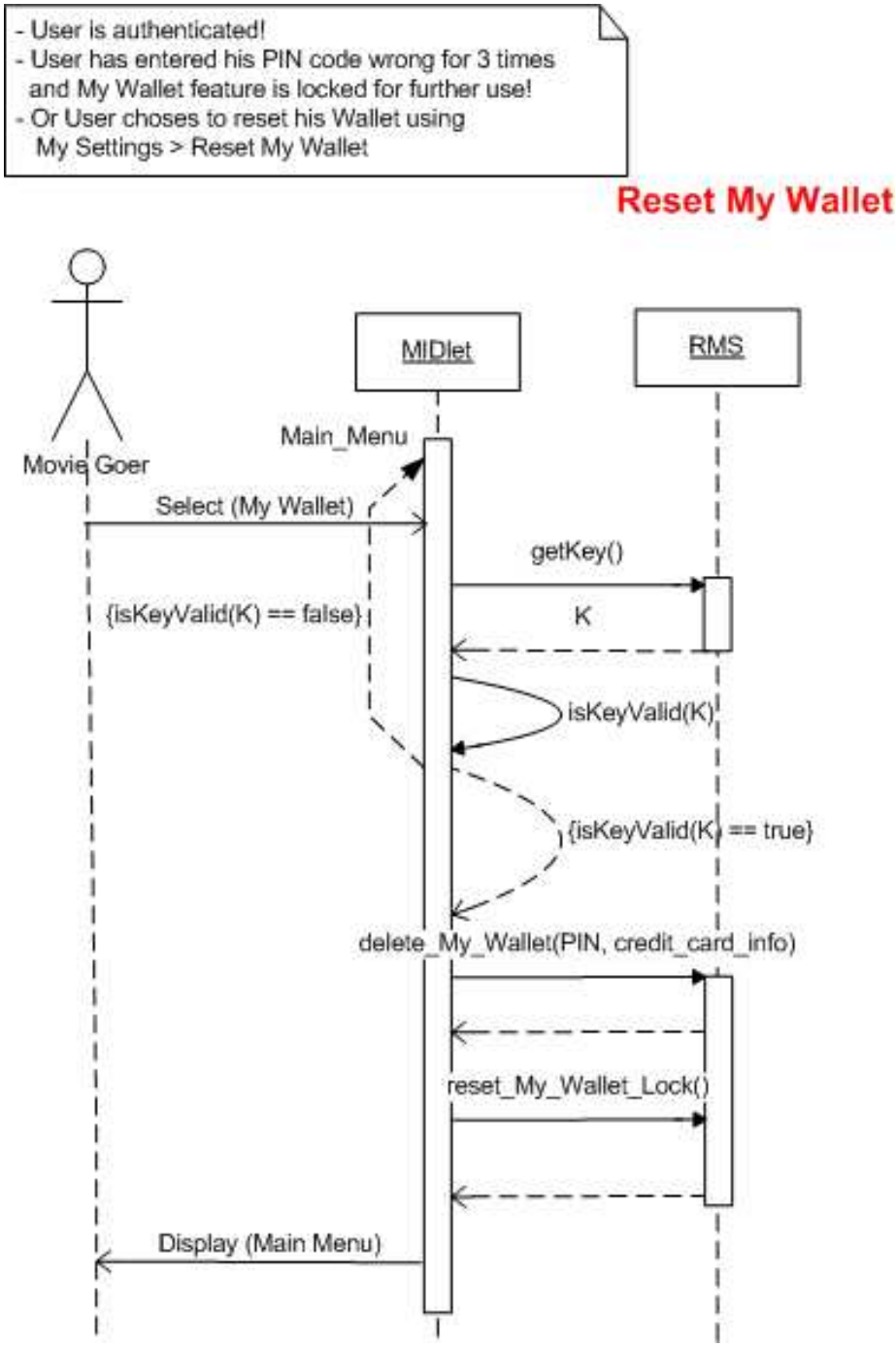

- User is authenticated!
- User has loged in to My Wallet!
Add New Credit Card to My Wallet
Movie Goer
MIDlet
RMS
Main_Menu
Select (My Wallet Menu)
Select (OPTIONS > Add Credit Card)
getKey()
K
{isKeyValid(K) == false}
isKeyValid(K)
CC - Credit Card
{max >= 3}
{isKeyValid(K) == true}
Enter (CC_Nick_Name, CC_Owner, CC_No, CC_Validity, CC_Sec_Code
getMaxNoOfCC()
max
{max < 3}
{unique == false}
is_CC_Unique(CC_Nick_Name)
unique
Display (CC_Nick_Name already exists!)
{unique == true}
Enc_CC := encrypt_K(CC_Data)
store_CC (CC_Nick_Name, Enc_CC)
Display (all credit cards)

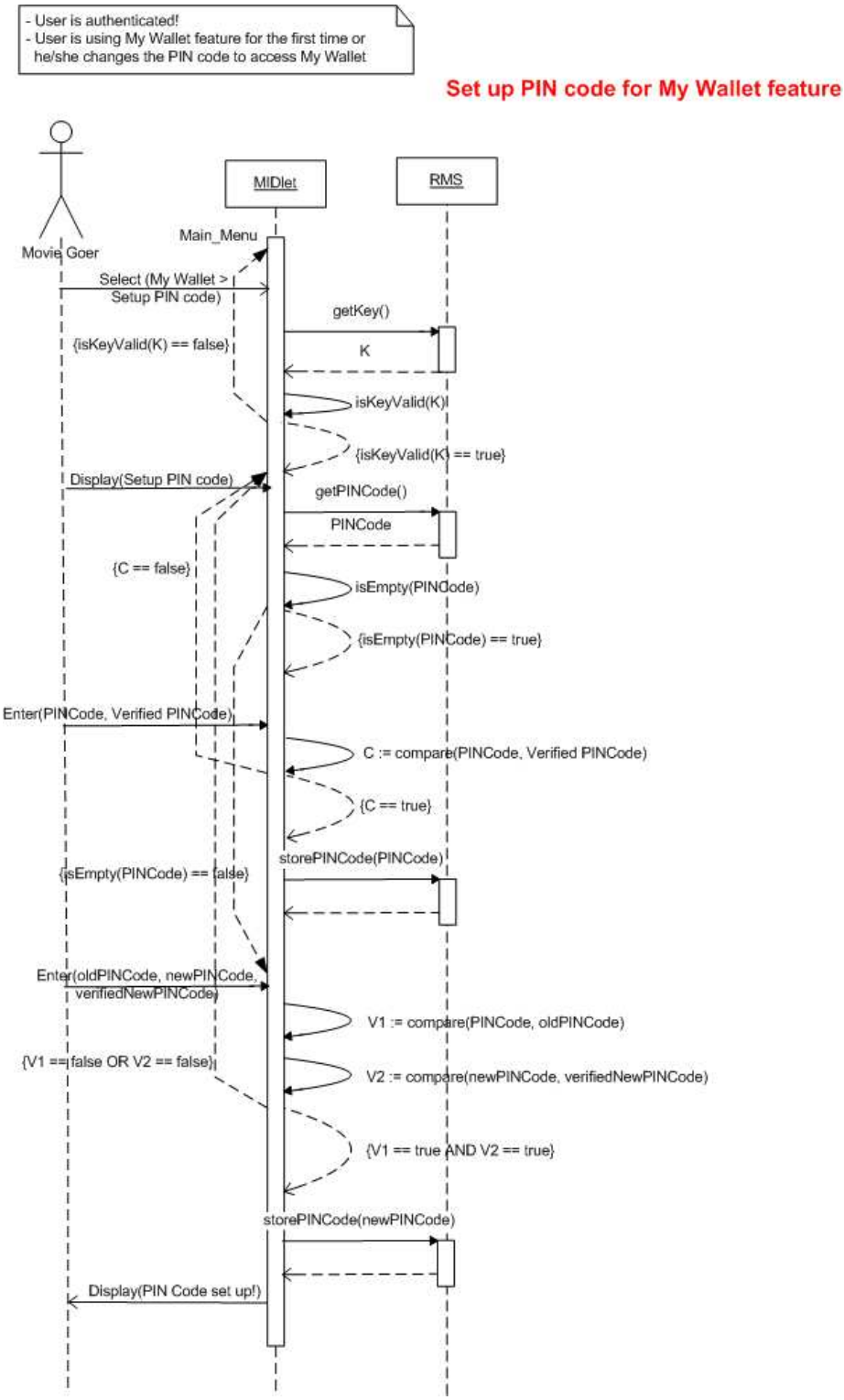

- The movie goer opens his/her mobile cinema application
- The application searches for used tickets in the phone memory and deletes them
Automatically Deletion of Used Tickets from the Mobile Device
MIDlet
RMS
Mobile Device Inner Clock
Display(Spash Screen)
getDate()
date
getUsedTickets(date)
ticket_ID_List
deleteTickets
(ticket_ID_List)
Display (Authentication Screen)

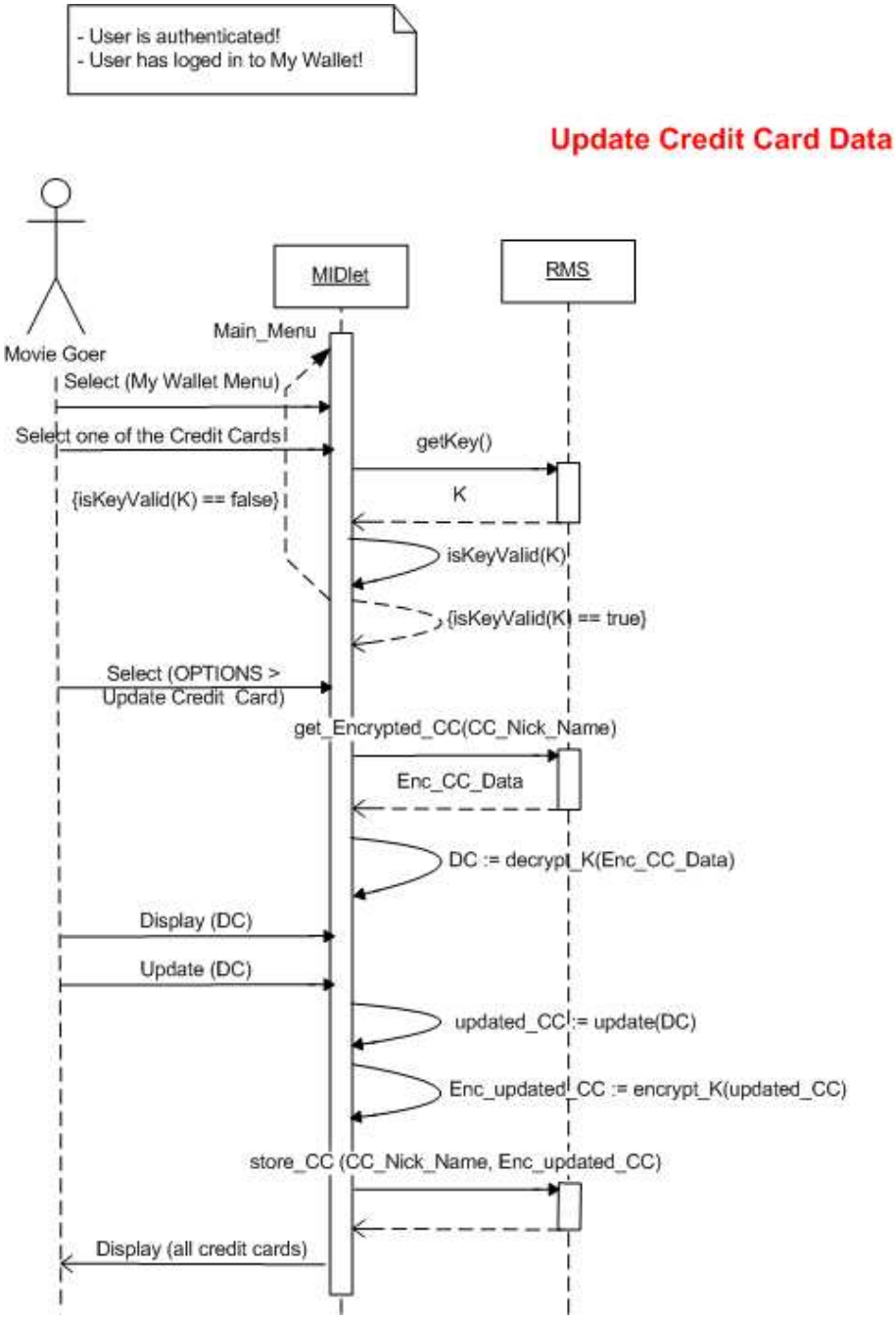

- User is authenticated!
- User has loged in to My Wallet!
Delete Credit Card
MIDlet
RMS
Movie Goer
Main_Menu
Select (My Wallet Menu)
Select one of the Credit Cards
getKey()
K
{isKeyValid(K) == false}
isKeyValid(K)
{isKeyValid(K) == true}
Select OPTIONS > Delete)
delete_CC (CC_Nick_Name)
Display (all credit cards)

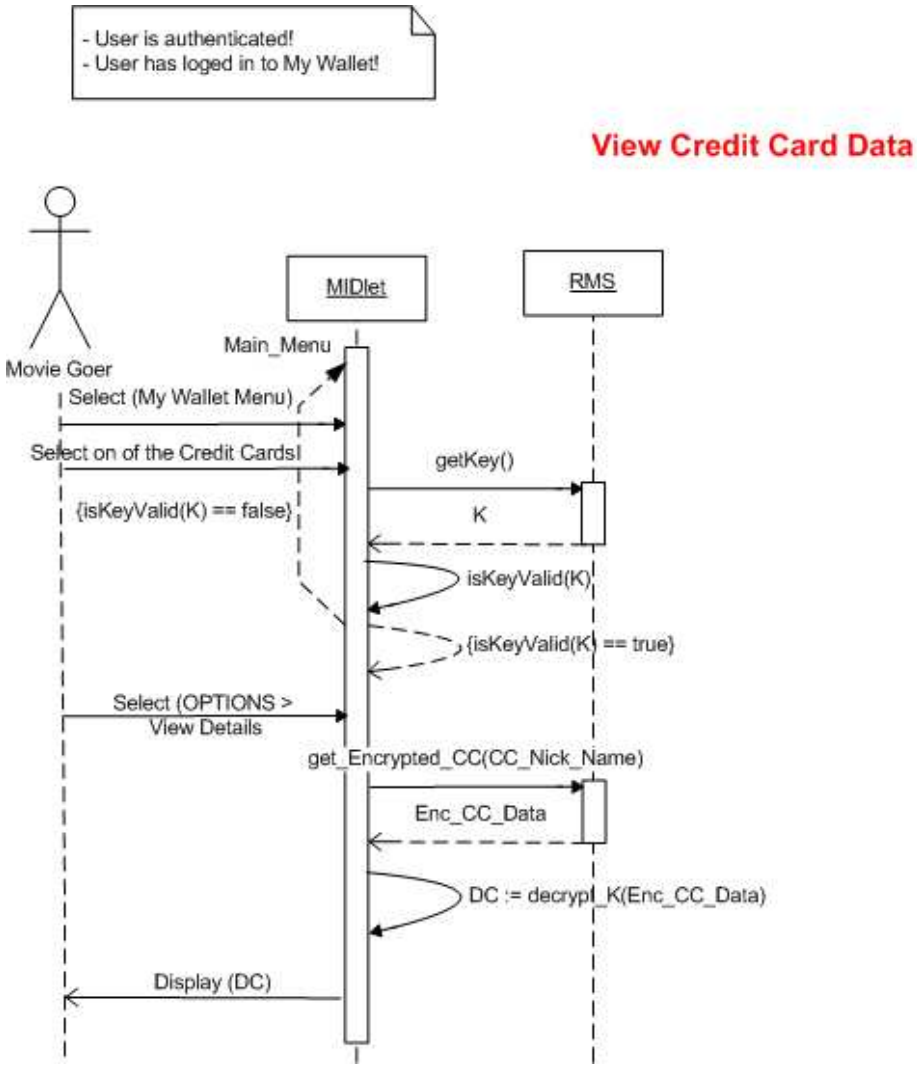

- User is authenticated!
Login My Wallet
MIDlet
RMS
Movie Goer
Main_Menu
Select (My Wallet)
getKey()
{isKeyValid(K) == false}
K
isKeyValid(K)
{isKeyValid(K) == true}
Display (LogIn My Wallet)
Enter (PINCode)
getPINCode()
RMS_PINCode
C := compare(PINCode, RMS_PINCode)
{C == false}
Display (Invalid PIN Code!)
{C == true}
Display (My Wallet Menu)

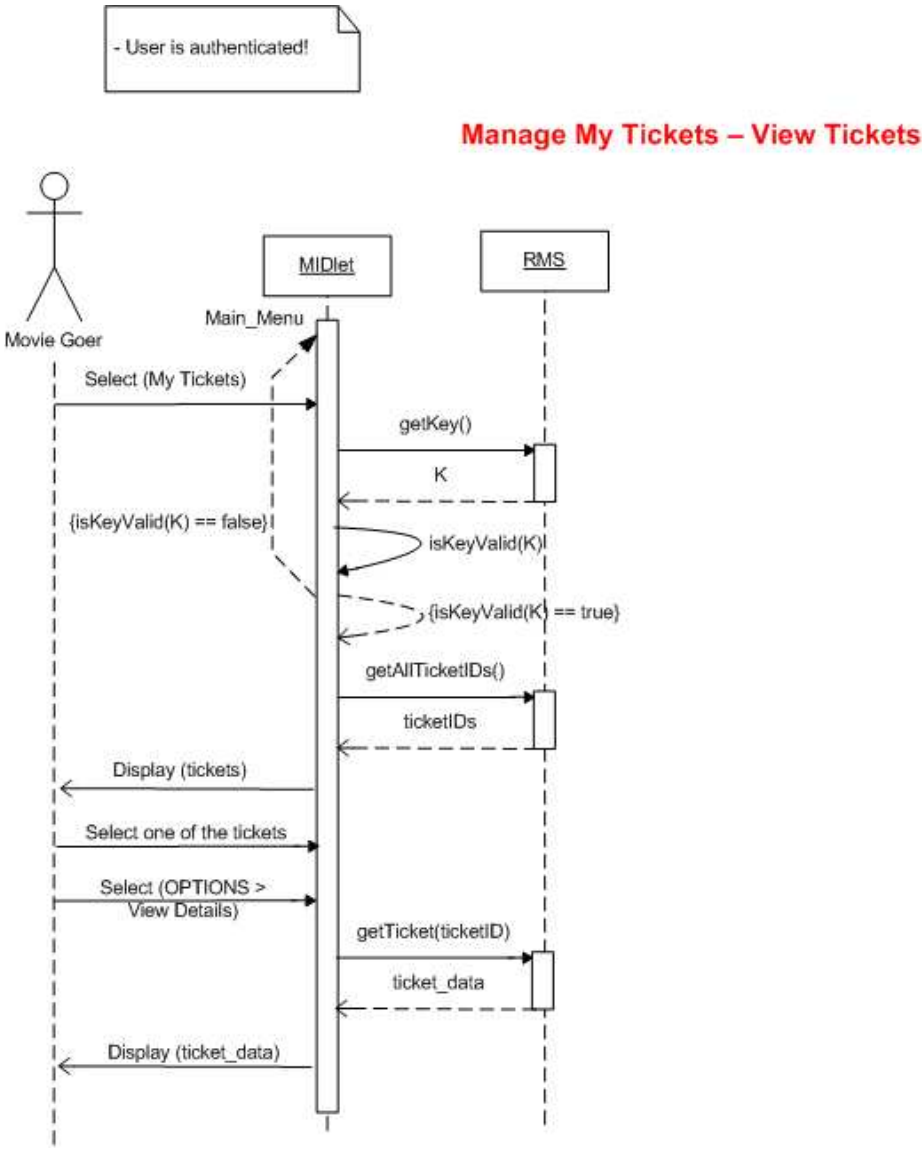

- User is authenticated!
My Settings
Movie Goer
MIDlet
RMS
Main_Menu
Select (My Settings)
getKey()
K
{isKeyValid(K) == false}
isKeyValid(K)
{isKeyValid(K) == true}
Movie goer chooses to save in RMS User Name, Password, Key, and to automatically log in with the server side without keying the credentials at application startup
update_my_settings(UN, Pwd, Key, auto_login)
Display (My Settings Update to ....)
Display (Main Menu)


- User is authenticated!
- User has entered his PIN code wrong for 3 times
and My Wallet feature is locked for further use!
- Or User choses to reset his Wallet using
My Settings > Reset My Wallet
Reset My Wallet
MIDlet
RMS
Main_Menu
Movie Goer
Select (My Wallet)
getKey()
{isKeyValid(K) == false}
K
isKeyValid(K)
{isKeyValid(K) == true}
delete_My_Wallet(PIN, credit_card_info)
reset_My_Wallet_Lock()
Display (Main Menu)

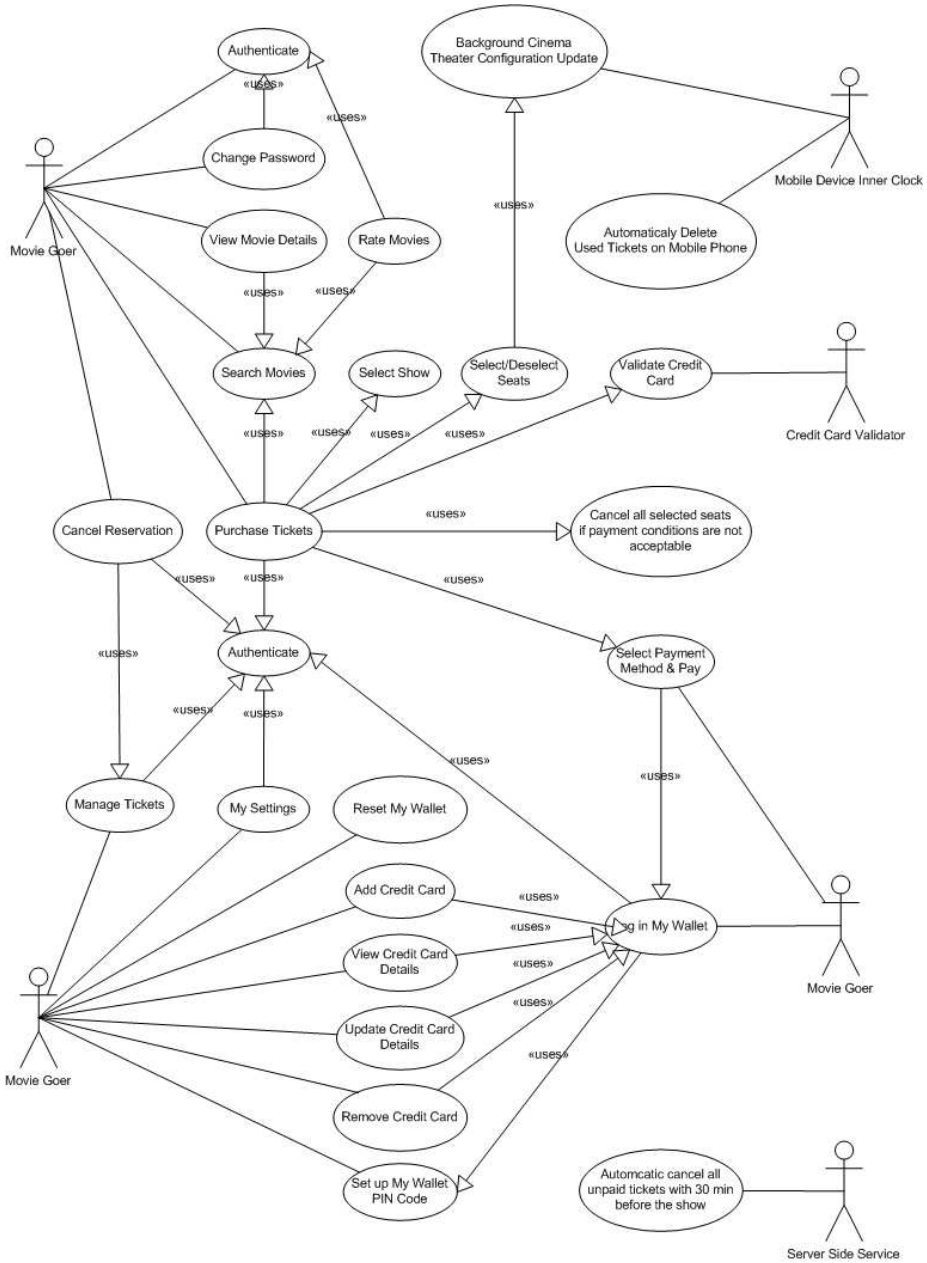

- User is authenticated!
- User is using My Wallet feature for the first time or he/she changes the PIN code to access My Wallet
Set up PIN code for My Wallet feature
Movie Goer
MIDlet
RMS
Main_Menu
Select (My Wallet > Setup PIN code)
getKey()
K
{isKeyValid(K) == false}
isKeyValid(K)
{isKeyValid(K) == true}
Display(Setup PIN code)
getPINCode()
PINCode
{C == false}
isEmpty(PINCode)
{isEmpty(PINCode) == true}
Enter(PINCode, Verified PINCode)
C := compare(PINCode, Verified PINCode)
{C == true}
storePINCode(PINCode)
{isEmpty(PINCode) == false}
Enter(oldPINCode, newPINCode, verifiedNewPINCode)
V1 := compare(PINCode, oldPINCode)
{V1 == false OR V2 == false}
V2 := compare(newPINCode, verifiedNewPINCode)
{V1 == true AND V2 == true}
storePINCode(newPINCode)
Display(PIN Code set up!)

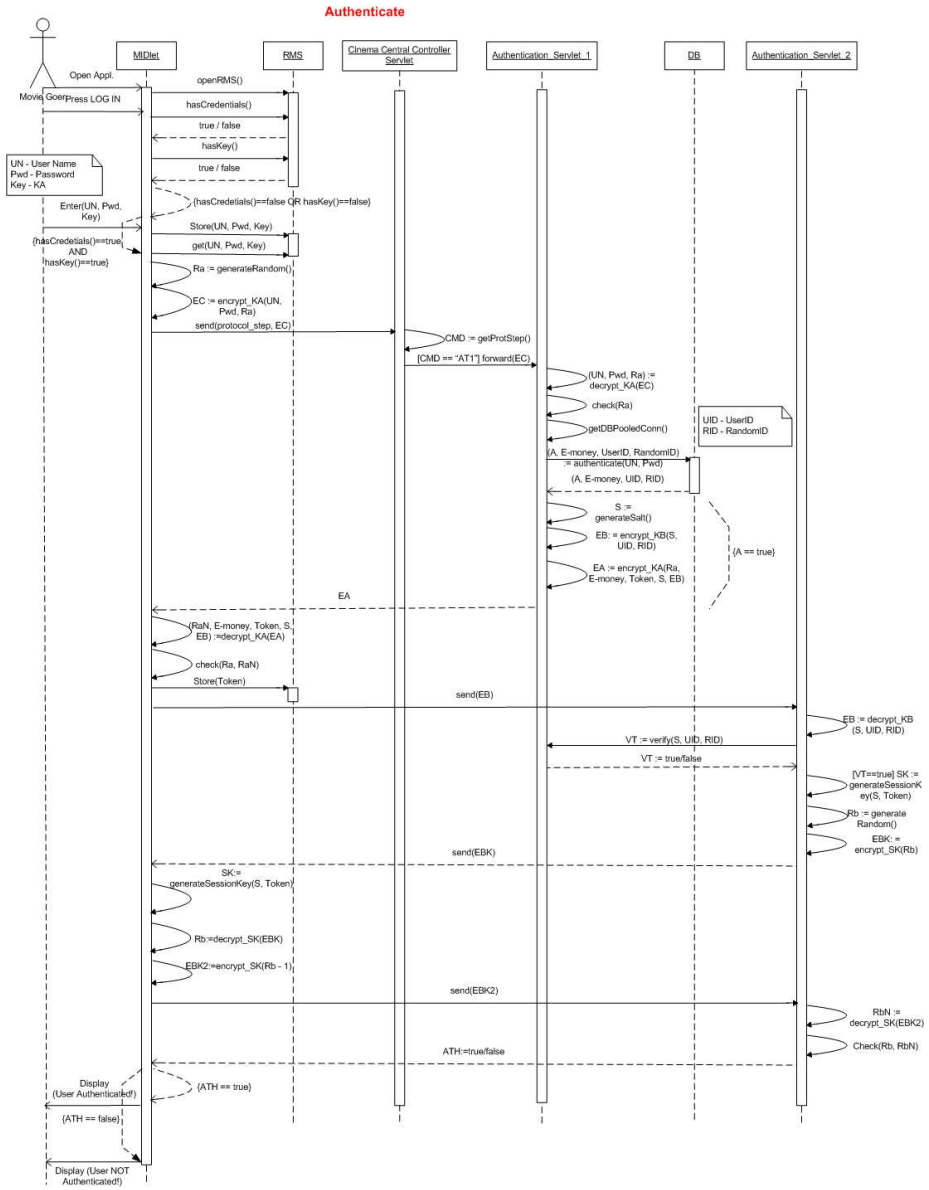

- User is authenticated!
- User has loged in to My Wallet!
Update Credit Card Data
Movie Goer
MIDlet
RMS
Main_Menu
Select (My Wallet Menu)
Select one of the Credit Cards
getKey()
K
{isKeyValid(K) == false}
isKeyValid(K)
{isKeyValid(K) == true}
Select (OPTIONS >
Update Credit Card)
get_Encrypted_CC(CC_Nick_Name)
Enc_CC_Data
DC := decrypt_K(Enc_CC_Data)
Display (DC)
Update (DC)
updated_CC := update(DC)
Enc_updated_CC := encrypt_K(updated_CC)
store_CC (CC_Nick_Name, Enc_updated_CC)
Display (all credit cards)


- User is authenticated!
- User has loged in to My Wallet!
View Credit Card Data
Movie Goer
MIDlet
RMS
Main_Menu
Select (My Wallet Menu)
Select on of the Credit Cards
getKey()
K
{isKeyValid(K) == false}
isKeyValid(K)
{isKeyValid(K) == true}
Select (OPTIONS >
View Details
get_Encrypted_CC(CC_Nick_Name)
Enc_CC_Data
DC := decrypt_K(Enc_CC_Data)
Display (DC)

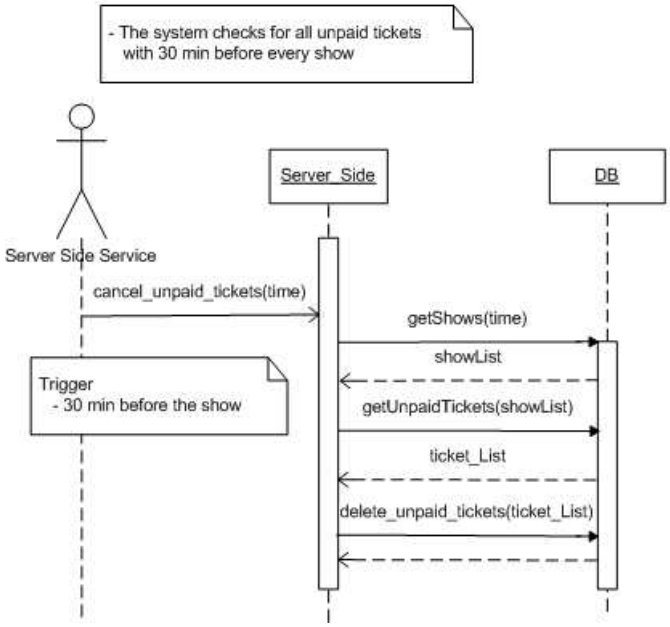

- User is authenticated!
Manage My Tickets – View Tickets
Movie Goer
MIDlet
RMS
Main_Menu
Select (My Tickets)
getKey()
K
{isKeyValid(K) == false}
isKeyValid(K)
{isKeyValid(K) == true}
getAllTicketIDs()
ticketIDs
Display (tickets)
Select one of the tickets
Select (OPTIONS > View Details)
getTicket(ticketID)
ticket_data
Display (ticket_data)

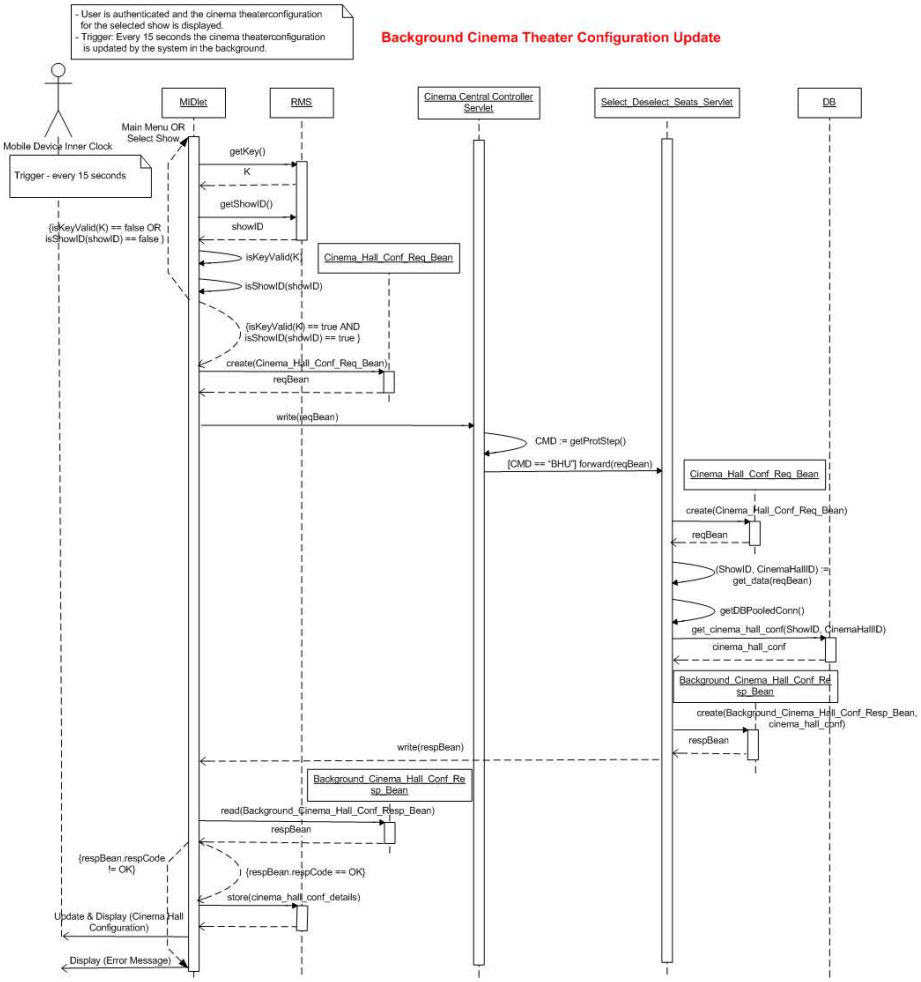
C.2 Server Side and Communication Protocol Sequence Diagrams

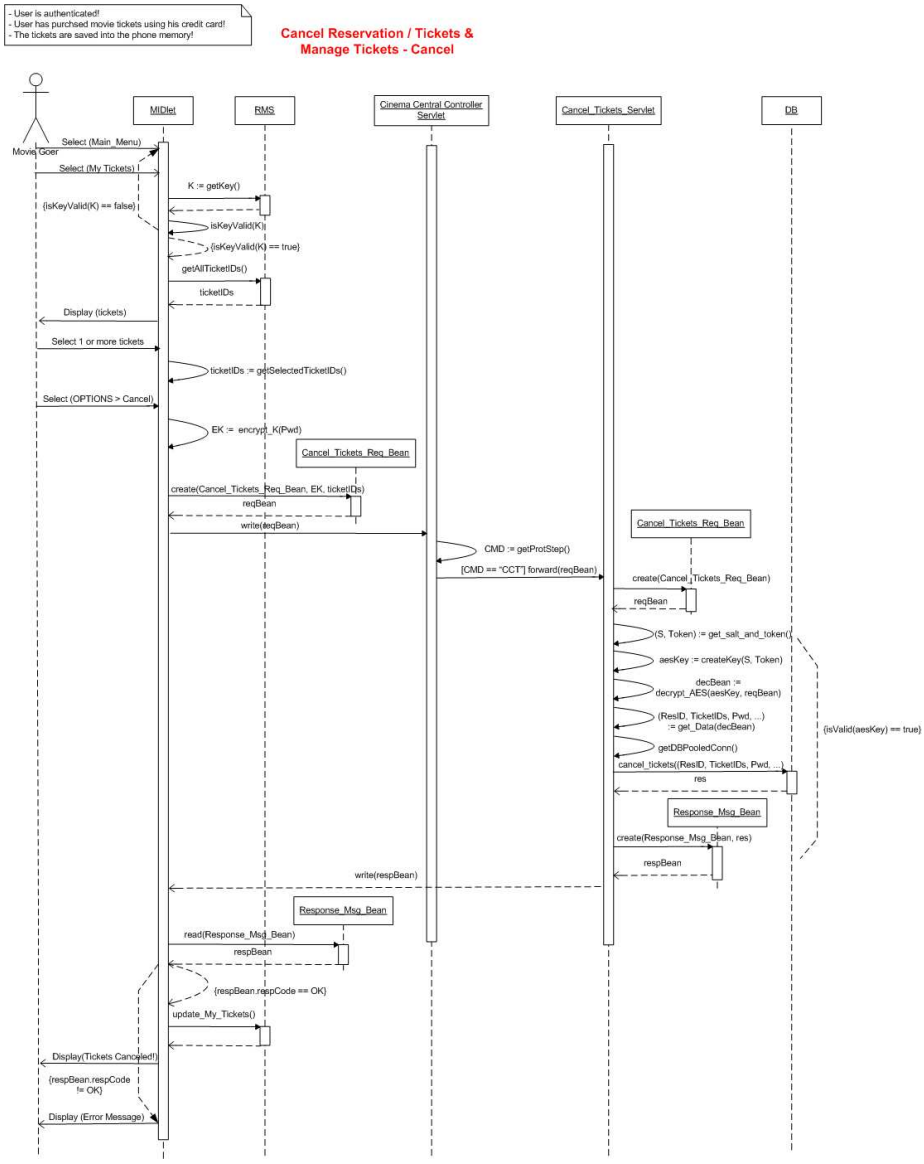

Authenticate
Background Cinema Theater Configuration Update
Mobile Device Inner Clock
Change Password
Movie Goer
View Movie Details
Rate Movies
Automaticaly Delete Used Tickets on Mobile Phone
Search Movies
Select Show
Select/Deselect Seats
Validate Credit Card
Credit Card Validator
Cancel Reservation
Purchase Tickets
Cancel all selected seats if payment conditions are not acceptable
Authenticate
Select Payment Method & Pay
Manage Tickets
My Settings
Reset My Wallet
Add Credit Card
g in My Wallet
Movie Goer
View Credit Card Details
Update Credit Card Details
Movie Goer
Remove Credit Card
Set up My Wallet PIN Code
Automcatic cancel all unpaid tickets with 30 min before the show
Server Side Service
«uses»

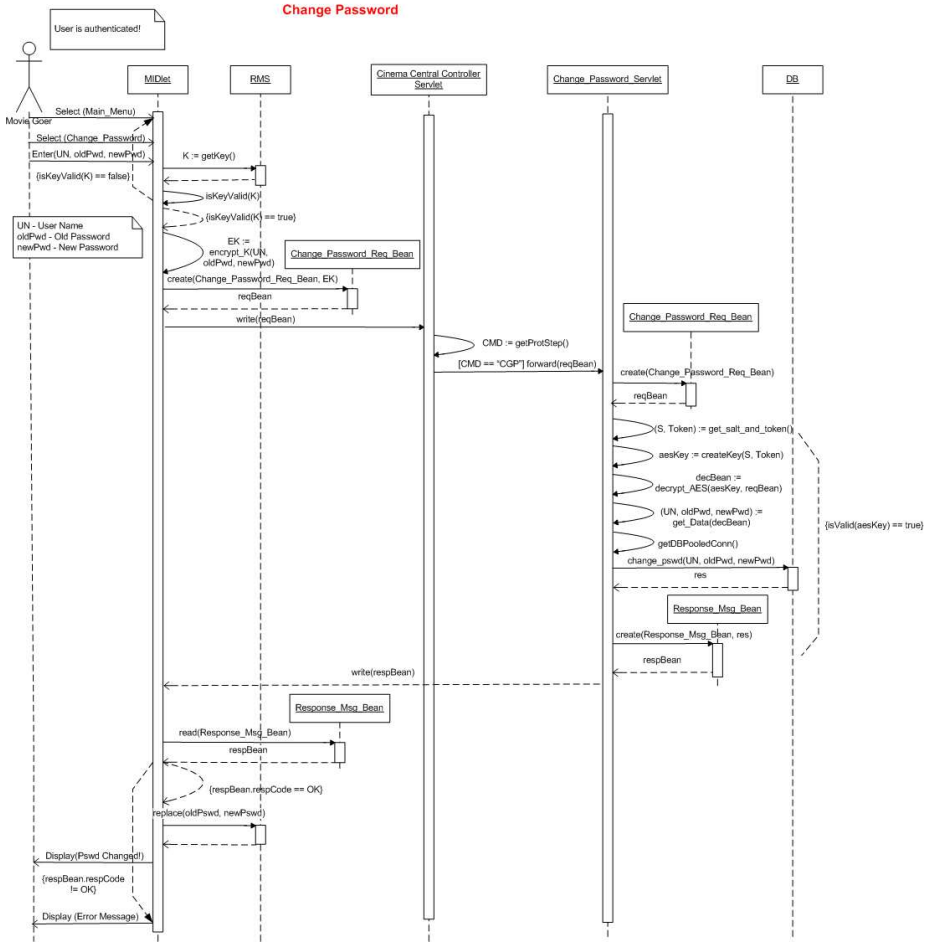

Authenticate
Movie Goer
MIDlet
RMS
Cinema Central Controller Servlet
Authentication_Servlet_1
DB
Authentication_Servlet_2
Open Appl.
openRMS()
Press LOG IN
hasCredentials()
true / false
hasKey()
true / false
UN - User Name
Pwd - Password
Key - KA
{hasCredetials()==false OR hasKey()==false}
Enter(UN, Pwd, Key)
Store(UN, Pwd, Key)
{hasCredetials()==true AND hasKey()==true}
get(UN, Pwd, Key)
Ra := generateRandom()
EC := encrypt_KA(UN, Pwd, Ra)
send(protocol_step, EC)
CMD := getProtStep()
[CMD == "AT1"] forward(EC)
(UN, Pwd, Ra) := decrypt_KA(EC)
check(Ra)
getDBPooledConn()
UID - UserID
RID - RandomID
(A, E-money, UserID, RandomID) := authenticate(UN, Pwd)
(A, E-money, UID, RID)
S := generateSalt()
EB: = encrypt_KB(S, UID, RID)
{A == true}
EA := encrypt_KA(Ra, E-money, Token, S, EB)
EA
(RaN, E-money, Token, S, EB) :=decrypt_KA(EA)
check(Ra, RaN)
Store(Token)
send(EB)
EB := decrypt_KB (S, UID, RID)
VT := verify(S, UID, RID)
VT := true/false
[VT==true] SK := generateSessionKey(S, Token)
Rb := generate Random()
EBK: = encrypt_SK(Rb)
send(EBK)
SK:= generateSessionKey(S, Token)
Rb:=decrypt_SK(EBK)
EBK2:=encrypt_SK(Rb - 1)
send(EBK2)
RbN := decrypt_SK(EBK2)
Check(Rb, RbN)
ATH:=true/false
Display (User Authenticated!)
{ATH == true}
{ATH == false}
Display (User NOT Authenticated!)

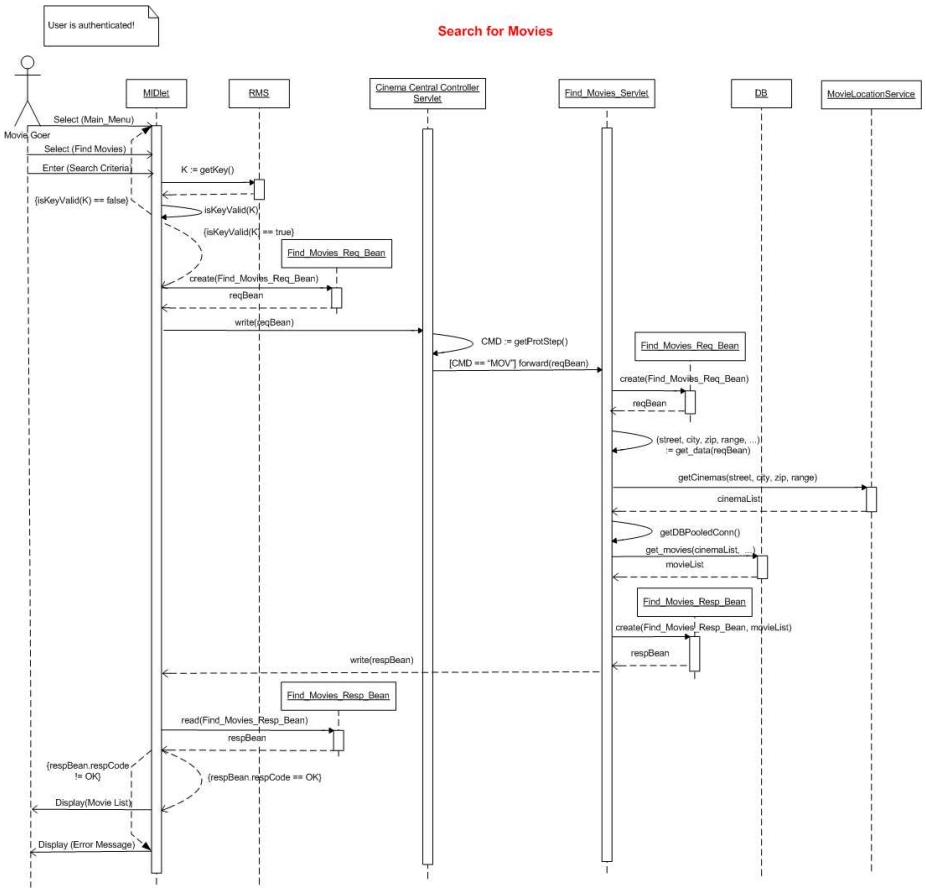
Automatically cancel all unpaid tickets with 30 min before the show

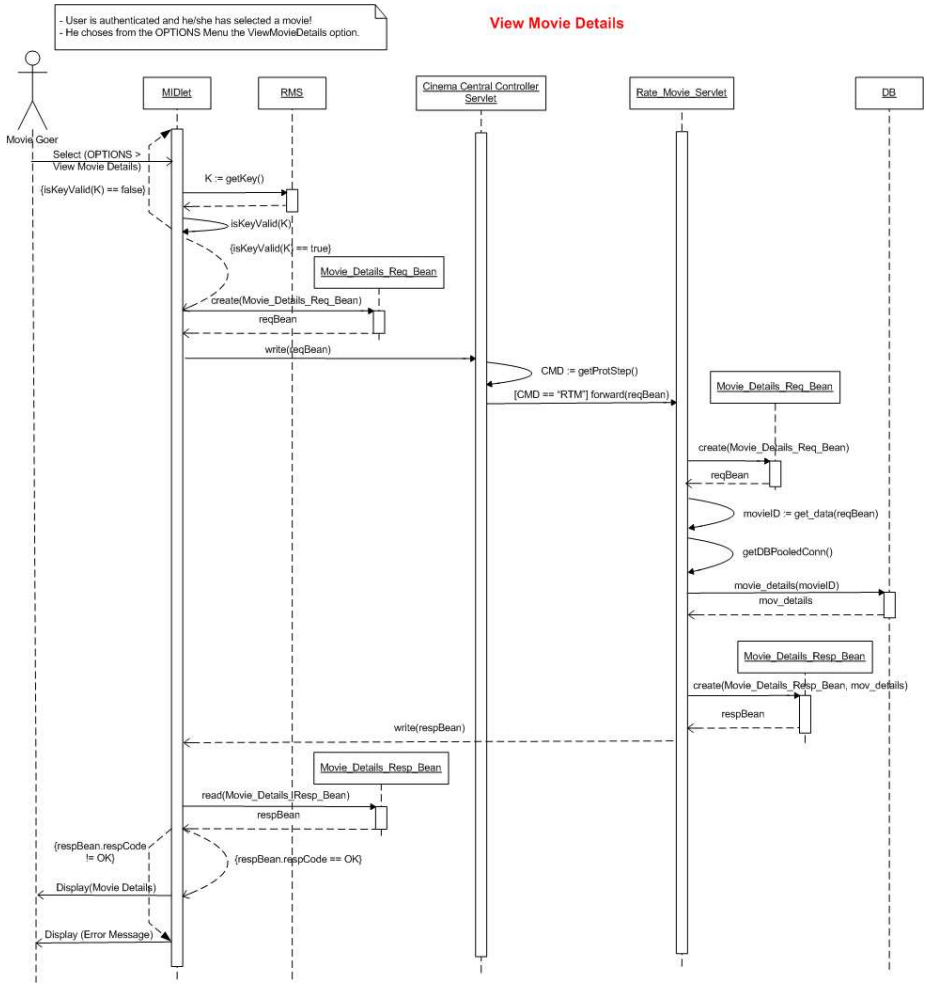


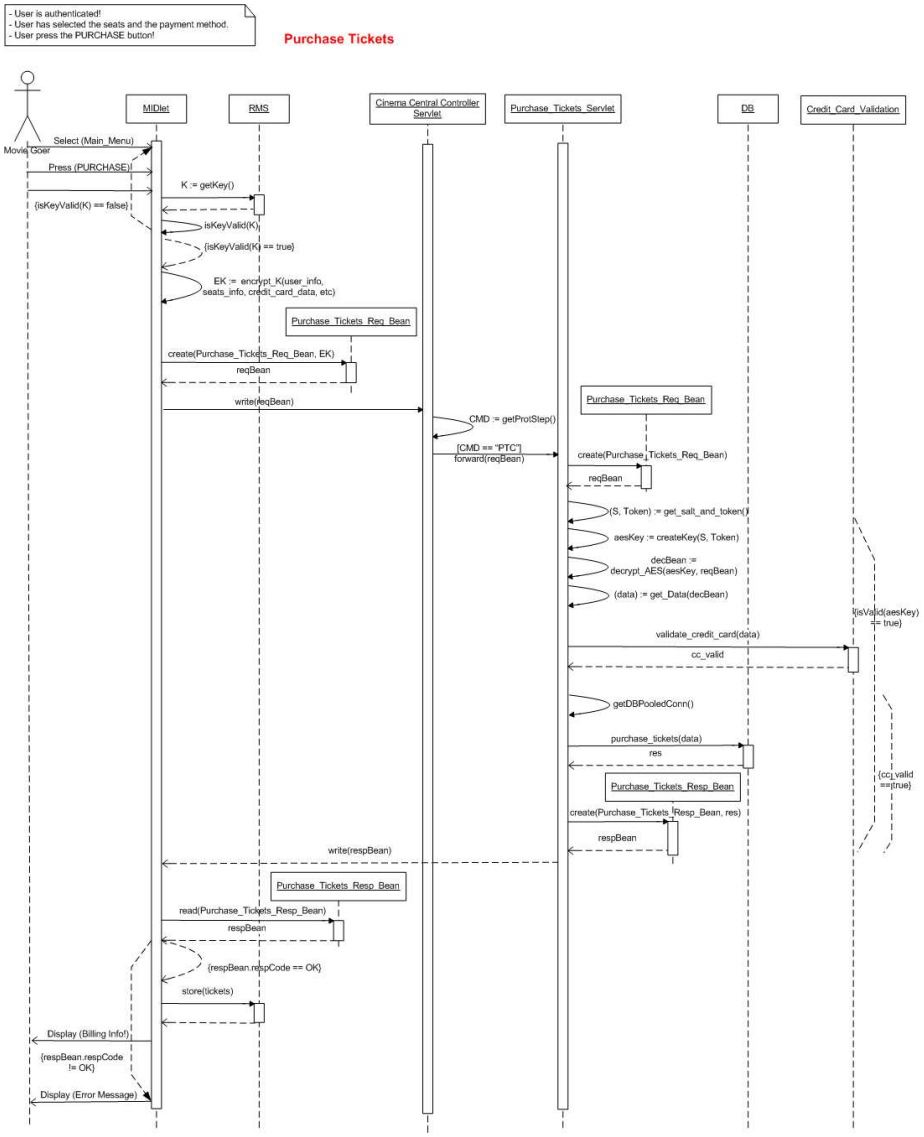

- User is authenticated and the cinema theaterconfiguration for the selected show is displayed.
- Trigger: Every 15 seconds the cinema theaterconfiguration is updated by the system in the background.
Background Cinema Theater Configuration Update
Mobile Device Inner Clock
MIDlet
RMS
Cinema Central Controller Servlet
Select_Deselect_Seats_Servlet
DB
Main Menu OR Select Show
Trigger - every 15 seconds
getKey()
K
getShowID()
showID
{isKeyValid(K) == false OR isShowID(showID) == false }
isKeyValid(K)
Cinema_Hall_Conf_Req_Bean
isShowID(showID)
{isKeyValid(K) == true AND isShowID(showID) == true }
create(Cinema_Hall_Conf_Req_Bean)
reqBean
write(reqBean)
CMD := getProtStep()
[CMD == "BHU"] forward(reqBean)
Cinema_Hall_Conf_Req_Bean
create(Cinema_Hall_Conf_Req_Bean)
reqBean
(ShowID, CinemaHallID) := get_data(reqBean)
getDBPooledConn()
get_cinema_hall_conf(ShowID, CinemaHallID)
cinema_hall_conf
Background_Cinema_Hall_Conf_Resp_Bean
create(Background_Cinema_Hall_Conf_Resp_Bean, cinema_hall_conf)
respBean
write(respBean)
Background_Cinema_Hall_Conf_Resp_Bean
read(Background_Cinema_Hall_Conf_Resp_Bean)
respBean
{respBean.respCode != OK}
{respBean.respCode == OK}
store(cinema_hall_conf_details)
Update & Display (Cinema Hall Configuration)
Display (Error Message)

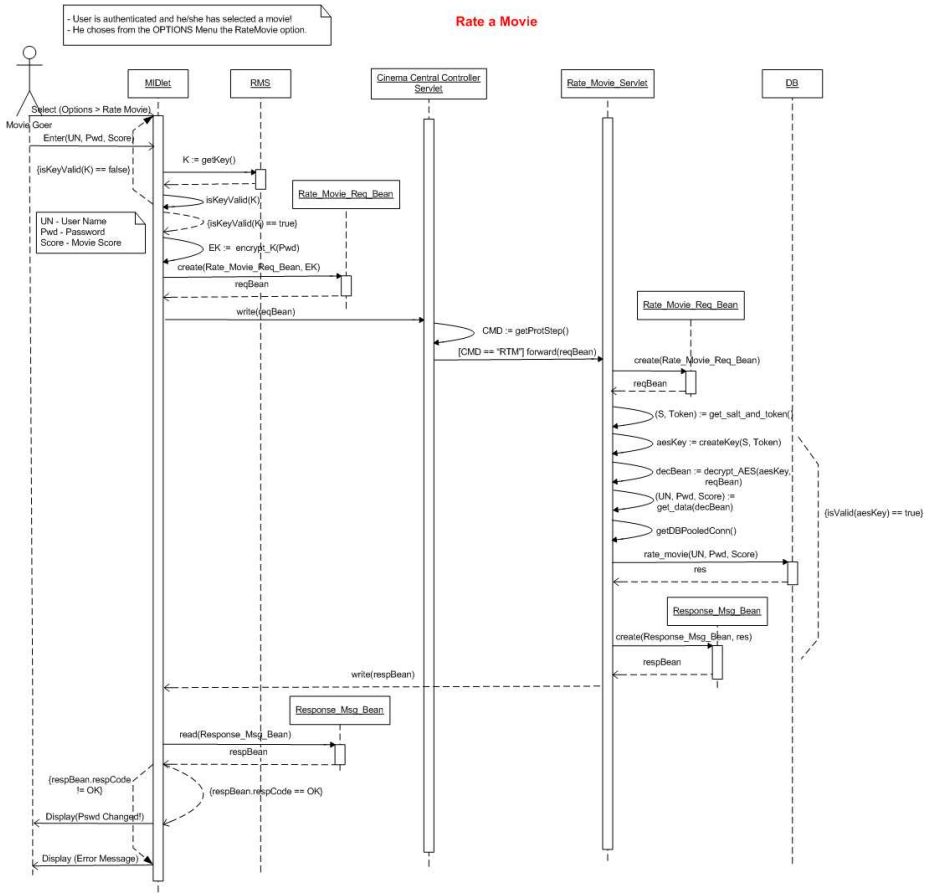

- User is authenticated!
- User has purchsed movie tickets using his credit card!
- The tickets are saved into the phone memory!
Cancel Reservation / Tickets &
Manage Tickets - Cancel
Movie Goer
MIDlet
RMS
Cinema Central Controller Servlet
Cancel_Tickets_Servlet
DB
Select (Main_Menu)
Select (My Tickets)
K := getKey()
{isKeyValid(K) == false}
isKeyValid(K)
{isKeyValid(K) == true}
getAllTicketIDs()
ticketIDs
Display (tickets)
Select 1 or more tickets
ticketIDs := getSelectedTicketIDs()
Select (OPTIONS > Cancel)
EK := encrypt_K(Pwd)
Cancel_Tickets_Req_Bean
create(Cancel_Tickets_Req_Bean, EK, ticketIDs)
reqBean
write(reqBean)
CMD := getProtStep()
[CMD == "CCT"] forward(reqBean)
Cancel_Tickets_Req_Bean
create(Cancel_Tickets_Req_Bean)
reqBean
(S, Token) := get_salt_and_token()
aesKey := createKey(S, Token)
decBean :=
decrypt_AES(aesKey, reqBean)
(ResID, TicketIDs, Pwd, ...)
:= get_Data(decBean)
getDBPooledConn()
cancel_tickets((ResID, TicketIDs, Pwd, ...)
res
{isValid(aesKey) == true}
Response_Msg_Bean
create(Response_Msg_Bean, res)
respBean
write(respBean)
Response_Msg_Bean
read(Response_Msg_Bean)
respBean
{respBean.respCode == OK}
update_My_Tickets()
Display(Tickets Canceled!)
{respBean.respCode
!= OK}
Display (Error Message)

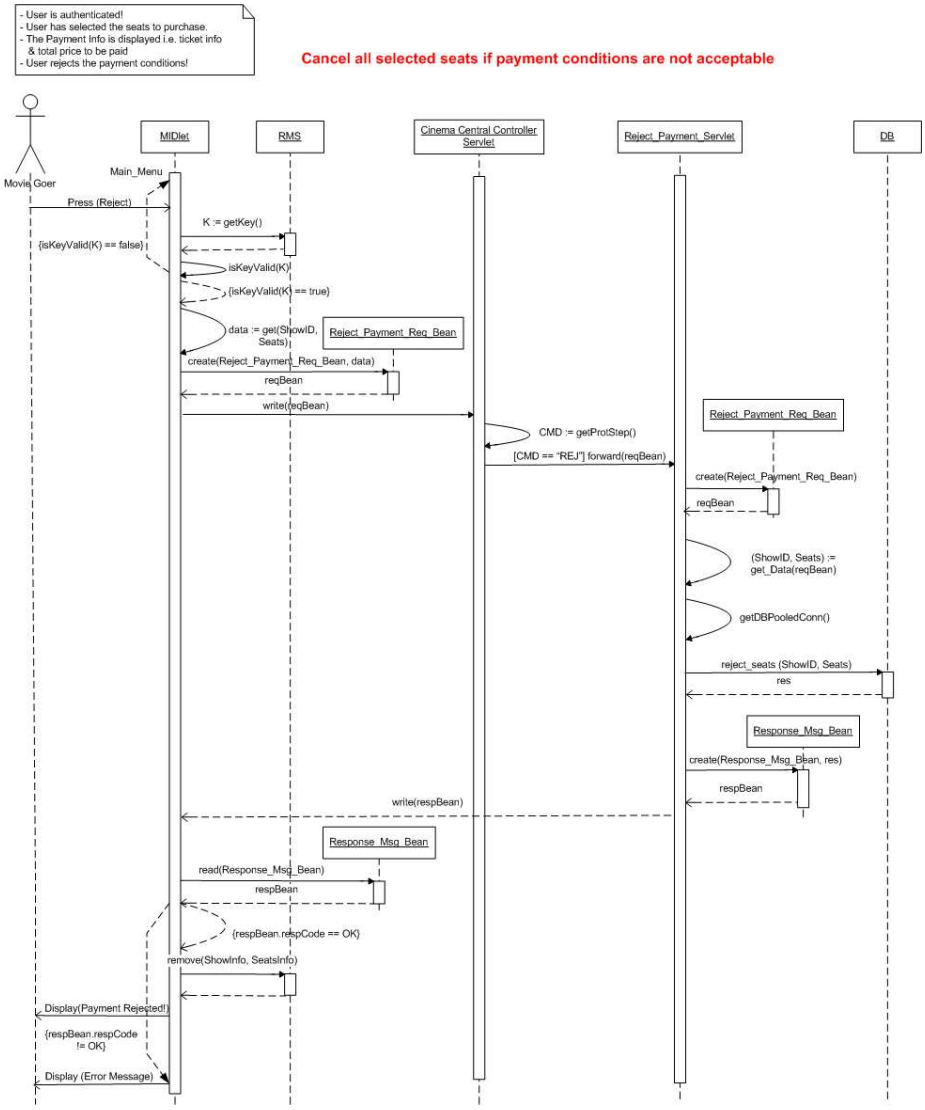

Change Password
User is authenticated!
MIDlet
RMS
Cinema Central Controller Servlet
Change_Password_Servlet
DB
Movie Goer
Select (Main_Menu)
Select (Change_Password)
Enter(UN, oldPwd, newPwd)
K := getKey()
(isKeyValid(K) == false)
isKeyValid(K)
(isKeyValid(K) == true)
UN - User Name
oldPwd - Old Password
newPwd - New Password
EK := encrypt_K(UN, oldPwd, newPwd)
Change_Password_Req_Bean
create(Change_Password_Req_Bean, EK)
reqBean
write(reqBean)
Change_Password_Req_Bean
CMD := getProtStep()
[CMD == "CGP"] forward(reqBean)
create(Change_Password_Req_Bean)
reqBean
(S, Token) := get_salt_and_token()
aesKey := createKey(S, Token)
decBean := decrypt_AES(aesKey, reqBean)
(UN, oldPwd, newPwd) := get_Data(decBean)
getDBPooledConn()
change_pswd(UN, oldPwd, newPwd)
res
{isValid(aesKey) == true}
Response_Msg_Bean
create(Response_Msg_Bean, res)
respBean
write(respBean)
Response_Msg_Bean
read(Response_Msg_Bean)
respBean
(respBean.respCode == OK)
replace(oldPswd, newPswd)
Display(Pswd Changed!)
(respBean.respCode != OK)
Display (Error Message)

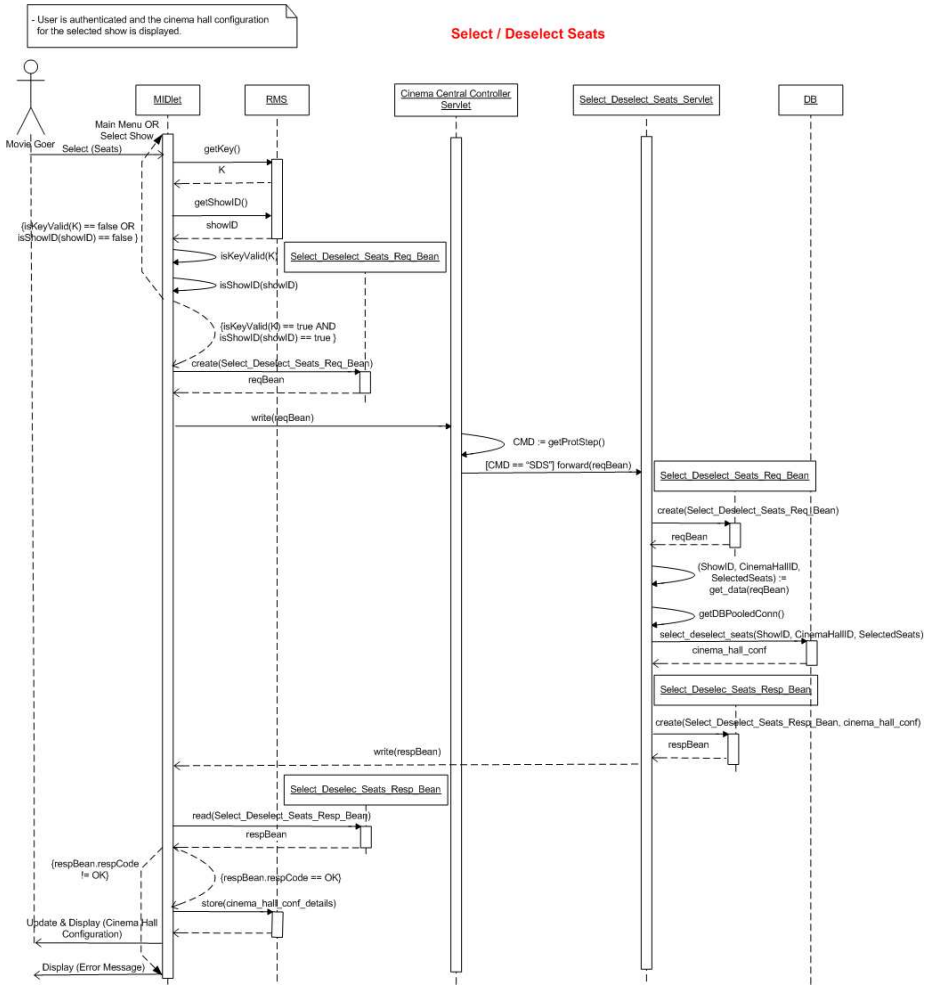

User is authenticated!
Search for Movies
Movie Goer
MIDlet
RMS
Cinema Central Controller Servlet
Find_Movies_Servlet
DB
MovieLocationService
Select (Main_Menu)
Select (Find Movies)
Enter (Search Criteria)
K := getKey()
{isKeyValid(K) == false}
isKeyValid(K)
{isKeyValid(K) == true}
Find_Movies_Req_Bean
create(Find_Movies_Req_Bean)
reqBean
write(reqBean)
CMD := getProtStep()
[CMD == "MOV"] forward(reqBean)
Find_Movies_Req_Bean
create(Find_Movies_Req_Bean)
reqBean
(street, city, zip, range, ...) := get_data(reqBean)
getCinemas(street, city, zip, range)
cinemaList
getDBPooledConn()
get_movies(cinemaList, ...)
movieList
Find_Movies_Resp_Bean
create(Find_Movies_Resp_Bean, movieList)
respBean
write(respBean)
Find_Movies_Resp_Bean
read(Find_Movies_Resp_Bean)
respBean
{respBean.respCode != OK}
{respBean.respCode == OK}
Display(Movie List)
Display (Error Message)

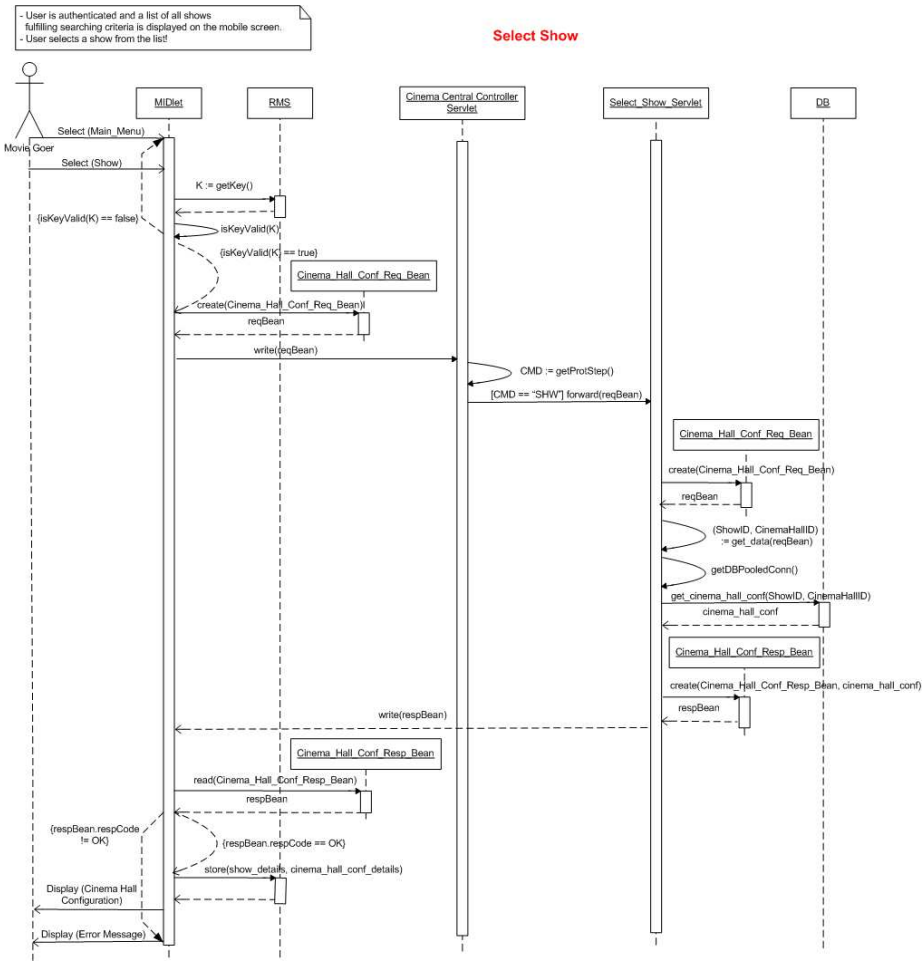

View Movie Details
- User is authenticated and he/she has selected a movie!
- He choses from the OPTIONS Menu the ViewMovieDetails option.
Movie Goer
MIDlet
RMS
Cinema Central Controller Servlet
Rate_Movie_Servlet
DB
Select (OPTIONS > View Movie Details)
{isKeyValid(K) == false}
K := getKey()
isKeyValid(K)
{isKeyValid(K) == true}
Movie_Details_Req_Bean
create(Movie_Details_Req_Bean)
reqBean
write(reqBean)
CMD := getProtStep()
[CMD == "RTM"] forward(reqBean)
Movie_Details_Req_Bean
create(Movie_Details_Req_Bean)
reqBean
movieID := get_data(reqBean)
getDBPooledConn()
movie_details(movieID)
mov_details
Movie_Details_Resp_Bean
create(Movie_Details_Resp_Bean, mov_details)
respBean
write(respBean)
Movie_Details_Resp_Bean
read(Movie_Details_Resp_Bean)
respBean
{respBean.respCode != OK}
{respBean.respCode == OK}
Display(Movie Details)
Display (Error Message)


- User is authenticated!
- User has selected the seats and the payment method.
- User press the PURCHASE button!
Purchase Tickets
Movie Goer
MIDlet
RMS
Cinema Central Controller Servlet
Purchase_Tickets_Servlet
DB
Credit_Card_Validation
Select (Main_Menu)
Press (PURCHASE)
K := getKey()
{isKeyValid(K) == false}
isKeyValid(K)
{isKeyValid(K) == true}
EK := encrypt_K(user_info, seats_info, credit_card_data, etc)
Purchase_Tickets_Req_Bean
create(Purchase_Tickets_Req_Bean, EK)
reqBean
write(reqBean)
CMD := getProtStep()
[CMD == "PTC"] forward(reqBean)
Purchase_Tickets_Req_Bean
create(Purchase_Tickets_Req_Bean)
reqBean
(S, Token) := get_salt_and_token()
aesKey := createKey(S, Token)
decBean := decrypt_AES(aesKey, reqBean)
(data) := get_Data(decBean)
{isValid(aesKey) == true}
validate_credit_card(data)
cc_valid
getDBPooledConn()
purchase_tickets(data)
res
Purchase_Tickets_Resp_Bean
create(Purchase_Tickets_Resp_Bean, res)
respBean
{cc_valid ==true}
write(respBean)
Purchase_Tickets_Resp_Bean
read(Purchase_Tickets_Resp_Bean)
respBean
{respBean.respCode == OK}
store(tickets)
Display (Billing Info!)
{respBean.respCode != OK}
Display (Error Message)


- User is authenticated and he/she has selected a movie!
- He choses from the OPTIONS Menu the RateMovie option.
Rate a Movie
MIDlet
RMS
Cinema Central Controller Servlet
Rate_Movie_Servlet
DB
Movie Goer
Select (Options > Rate Movie)
Enter(UN, Pwd, Score)
K := getKey()
{isKeyValid(K) == false}
isKeyValid(K)
Rate_Movie_Req_Bean
{isKeyValid(K) == true}
UN - User Name
Pwd - Password
Score - Movie Score
EK := encrypt_K(Pwd)
create(Rate_Movie_Req_Bean, EK)
reqBean
write(reqBean)
CMD := getProtStep()
[CMD == "RTM"] forward(reqBean)
Rate_Movie_Req_Bean
create(Rate_Movie_Req_Bean)
reqBean
(S, Token) := get_salt_and_token()
aesKey := createKey(S, Token)
decBean := decrypt_AES(aesKey, reqBean)
(UN, Pwd, Score) := get_data(decBean)
getDBPooledConn()
rate_movie(UN, Pwd, Score)
res
{isValid(aesKey) == true}
Response_Msg_Bean
create(Response_Msg_Bean, res)
respBean
write(respBean)
Response_Msg_Bean
read(Response_Msg_Bean)
respBean
{respBean.respCode != OK}
{respBean.respCode == OK}
Display(Pswd Changed!)
Display (Error Message)


- User is authenticated!
- User has selected the seats to purchase.
- The Payment Info is displayed i.e. ticket info & total price to be paid
- User rejects the payment conditions!
Cancel all selected seats if payment conditions are not acceptable
Movie Goer
MIDlet
RMS
Cinema Central Controller Servlet
Reject_Payment_Servlet
DB
Main_Menu
Press (Reject)
K := getKey()
{isKeyValid(K) == false}
isKeyValid(K)
{isKeyValid(K) == true}
data := get(ShowID, Seats)
Reject_Payment_Req_Bean
create(Reject_Payment_Req_Bean, data)
reqBean
write(reqBean)
CMD := getProtStep()
[CMD == "REJ"] forward(reqBean)
Reject_Payment_Req_Bean
create(Reject_Payment_Req_Bean)
reqBean
(ShowID, Seats) := get_Data(reqBean)
getDBPooledConn()
reject_seats (ShowID, Seats)
res
Response_Msg_Bean
create(Response_Msg_Bean, res)
respBean
write(respBean)
Response_Msg_Bean
read(Response_Msg_Bean)
respBean
{respBean.respCode == OK}
remove(ShowInfo, SeatsInfo)
Display(Payment Rejected!)
{respBean.respCode != OK}
Display (Error Message)


- User is authenticated and the cinema hall configuration for the selected show is displayed.
Select / Deselect Seats
Movie Goer
MIDlet
RMS
Cinema Central Controller Servlet
Select_Deselect_Seats_Servlet
DB
Main Menu OR Select Show
Select (Seats)
getKey()
K
getShowID()
showID
{isKeyValid(K) == false OR isShowID(showID) == false }
isKeyValid(K)
Select_Deselect_Seats_Req_Bean
isShowID(showID)
{isKeyValid(K) == true AND isShowID(showID) == true }
create(Select_Deselect_Seats_Req_Bean)
reqBean
write(reqBean)
CMD := getProtStep()
[CMD == "SDS"] forward(reqBean)
Select_Deselect_Seats_Req_Bean
create(Select_Deselect_Seats_Req_Bean)
reqBean
(ShowID, CinemaHallID, SelectedSeats) := get_data(reqBean)
getDBPooledConn()
select_deselect_seats(ShowID, CinemaHallID, SelectedSeats)
cinema_hall_conf
Select_Deselec_Seats_Resp_Bean
create(Select_Deselect_Seats_Resp_Bean, cinema_hall_conf)
respBean
write(respBean)
Select_Deselec_Seats_Resp_Bean
read(Select_Deselect_Seats_Resp_Bean)
respBean
{respBean.respCode != OK}
{respBean.respCode == OK}
store(cinema_hall_conf_details)
Update & Display (Cinema Hall Configuration)
Display (Error Message)


- User is authenticated and a list of all shows fulfilling searching criteria is displayed on the mobile screen.
- User selects a show from the list!
Select Show
MIDlet
RMS
Cinema Central Controller Servlet
Select_Show_Servlet
DB
Movie Goer
Select (Main_Menu)
Select (Show)
K := getKey()
{isKeyValid(K) == false}
isKeyValid(K)
{isKeyValid(K) == true}
Cinema_Hall_Conf_Req_Bean
create(Cinema_Hall_Conf_Req_Bean)
reqBean
write(reqBean)
CMD := getProtStep()
[CMD == "SHW"] forward(reqBean)
Cinema_Hall_Conf_Req_Bean
create(Cinema_Hall_Conf_Req_Bean)
reqBean
(ShowID, CinemaHallID) := get_data(reqBean)
getDBPooledConn()
get_cinema_hall_conf(ShowID, CinemaHallID)
cinema_hall_conf
Cinema_Hall_Conf_Resp_Bean
create(Cinema_Hall_Conf_Resp_Bean, cinema_hall_conf)
respBean
write(respBean)
Cinema_Hall_Conf_Resp_Bean
read(Cinema_Hall_Conf_Resp_Bean)
respBean
{respBean.respCode != OK}
{respBean.respCode == OK}
store(show_details, cinema_hall_conf_details)
Display (Cinema Hall Configuration)
Display (Error Message)

Appendix D

Source Code of the System

D.1 Mobile Client Application

```
package constants;

/**
 * Defines all credit card types available for online payment
 *
 * @author Mihai Balan (s031288)
 *
 */
public final class CreditCardTypes {

  public static final String CC_VISA        = "VISA";
  public static final String CC_VISA_ELECTRON = "VISE";
  public static final String CC_AMERICAN_EXPRESS = "AMEX";
  public static final String CC_MASTER_CARD = "MAST";
  public static final String CC_DANKORT     = "DANK";
  public static final String CC_MAESTRO     = "MAES";
  public static final String CC_DINERS_CLUB = "DINE";

  public static final String[] CC_TYPES = {"VISA", "VISA␣ELECTRON", "
      AMERICAN␣EXPRESS", "MASTER␣CARD", "DANKORT", "MAESTRO", "DINERS␣
      CLUB"};

}

package constants;

/**
 * Defines the types of alert that can be displayed on the screen
 *
 * @author Mihai Balan (s031288)
 *
 */
public final class CustomAlertTypes{

  public static final int ALERT_ERROR = 10;

  public static final int ALERT_WARNING = 11;

  public static final int ALERT_INFO = 12;
}

package constants;

import javax.microedition.io.HttpConnection;
```

```
/**
 * Declare the SQL errors trigered by executing the
 * given SQL statements on the pgsql DB.
 *
 * All this errors are defined for and created in the
 * stored procedure that are called.
 *
 * It realizes a mapping between the SQL errors and the HttpServlet
     status codes
 *
 * @author Mihai Balan - s031288
 *
 */
public final class Error_Code_Constants {

  /**
   * User cannot be authenticated
   * SQL Error code = 401
   */
  //public static final int USER_NOT_AUTHENTICATED = HttpServletResponse.
      SC_UNAUTHORIZED;

  /**
   * User is authenticated
   * SQL Error code = 201
   */
  //public static final int USER_AUTHENTICATED = HttpServletResponse.
      SC_CREATED;

  /**
   * Error while executing the SQL statement
   * SQL Error code = 4xx
   */
  public static final int ERROR_IN_SQL = HttpConnection.HTTP_BAD_REQUEST;

  /**
   * Statement executed sucesfully and data retrieved
   * SQL Error code = 200
   */
  public static final int OK = HttpConnection.HTTP_OK;

  /**
   * There is no error in the SQL statement but
   * no data could be found acoocrdingly to the
   * given criteria
   *
   * SQL Error code = 410
```

```
 */
//public static final int DATA_NOT_FOUND = HttpConnection.
    HTTP_NOT_FOUND;

/**
 * Invalid Credit Card data
 *
 * SQL Error code = 406
 */
//public static final int INVALID_CREDIT_CARD = HttpConnection.
    HTTP_NOT_ACCEPTABLE;

/**
 * Incorrect Protocol Step sent by the client to the servlet
 *
 * Error Code = 501;
 */
public static final int INVALID_PROTOCOL_STEP = HttpConnection.
    HTTP_NOT_IMPLEMENTED;

/**
 * An exception or error that occurs unexpected in the Cinema Service
 *
 * value = 500
 */
public static final int INTERNAL_SERVER_ERROR = HttpConnection.
    HTTP_INTERNAL_ERROR;

/**
 * Unknown error - very unlikely to occur
 *
 * SQL Error Code = 417
 */
public static final int UNKNOWN_ERROR = HttpConnection.
    HTTP_EXPECT_FAILED;

/**
 * The response bean that is creted is either null or invalid
 *
 * value = 10
 */
public static final int INVALID_RESPONSE_BEAN = 10;

/**
 * An error occurs while trying to serialize the response bean to the
     MIDlet
 *
```

```
   * value = 11
   */
  public static final int ERROR_IN_SERIALIZING_RESPONSE_BEAN = 11;

  /**
   * An error occurs while trying to deserialize the request bean from
       the MIDlet
   *
   * value = 12
   */
  public static final int ERROR_IN_DESERIALIZING_RESPONSE_BEAN = 12;

  /**
   * An error occurs while trying to serialize the response bean to the
       MIDlet
   *
   * value = 15
   */
  public static final int ERROR_IN_SERIALIZING_REQUEST_BEAN = 14;

  /**
   * An error occurs while trying to deserialize the request bean from
       the MIDlet
   *
   * value = 15
   */
  public static final int ERROR_IN_DESERIALIZING_REQUEST_BEAN = 15;

  /**
   * An error occurs while trying to decrypt the content of a message
       sent by the user
   *
   * value = 16
   */
  public static final int ERROR_IN_DECRYPTING = 16;


  /**
   * An error occurs while trying to pay for the purchased
   * tickets. The credit card data is valid but
   * the payment service is down or a network error occured
   */
  public static final int ERROR_WHILE_PAYING = 420;

}// end class


package constants;
```

```
/**
 * Declare general constants used for identifying the protocl steps in
     the
 * client - server communication
 *
 * @author Mihai Balan - s031288
 *
 */
public final class Protocol_Step_Constants {

  /**
   * Authentication Protocol Step 1
   */
  public static final String PRT_STEP_AUTHENTICATION_1 = "AT1";

  /**
   * Authentication Protocol Step 2
   */
  public static final String PRT_STEP_AUTHENTICATION_2 = "AT2";

  /**
   * Authentication Protocol Step 3
   */
  //public static final String PRT_STEP_AUTHENTICATION_2 = "AT2";

  /**
   * Change Password Protocol Step
   */
  public static final String PRT_STEP_CHANGE_PASSWORD = "CGP";

  /**
   * Find Movies Protocol Step
   */
  public static final String PRT_STEP_FIND_MOVIES = "MOV";

  /**
   * Select Show and Display Cinema Hall Configuration Protocol Step
   */
  public static final String
      PRT_STEP_SELECT_SHOW_AND_DISPLAY_CINEMA_HALL_CONF = "SHW";

  /**
   * Background Cinema Hall Update Protocol Step
   */
  public static final String PRT_STEP_BACKGROUND_CINEMA_HALL_UPDATE = "
      BHU";
```

```
  /**
   * Select / Deselect Seats Protocol Step
   */
  public static final String PRT_STEP_SELECT_DESELECT_SEATS = "SDS";

  /**
   * Purchase Tickets Protocol Step
   */
  public static final String PRT_STEP_PURCHASE_TICKETS = "PTC";

  /**
   * Cancel Tickets / Resservation Protocol Step
   */
  public static final String PRT_STEP_CANCEL_TICKETS = "CCT";

  /**
   * Reject Payment Protocol Step
   */
  public static final String PRT_STEP_REJECT_PAYMENT = "REJ";

  /**
   * Rate Movie Protocol Step
   */
  public static final String PRT_STEP_RATE_MOVIE = "RTM";

  /**
   * Movie Details Protocol Step
   */
  public static final String PRT_STEP_MOVIE_DETAILS = "DET";


} // end class

package constants;

/**
 * Defines all purchase method types available for online payment
 *
 * @author Mihai Balan (s031288)
 *
 */
public final class PurchaseMethosConstants {

  public static final String PM_CARD = "CARD";
  public static final String PM_CINEMA = "CINEMA";
  public static final String PM_EMONEY = "EMONEY";
```

```
}

package constants;


/**
 * Declare the return codes from the SQL stored procedures
 * for the protocl steps.
 *
 * @author Mihai Balan - s031288
 *
 */
public final class SQL_Return_Codes {

  /*-------------------------------------------------------------------*/
  /**
   * Authenticate Stored Procedure - User authenticated
   */
  public static final int AUTHENTICATE_PRT_USER_AUTHENTICATED = 201;

  /**
   * Authenticate Stored Procedure - User NOT authenticated
   */
  public static final int AUTHENTICATE_PRT_USER_NOT_AUTHENTICATED = 401;

  /*-------------------------------------------------------------------*/
  /**
   * Change Password Stored Procedure - Password changes successfuly
   */
  public static final int CHANGE_PASSWORD_PRT_PASSWORD_CHANGED = 202;

  /**
   * Change Password Stored Procedure - User NOT authenticated
   */
  public static final int CHANGE_PASSWORD_PRT_USER_NOT_AUTHENTICATED
       = 402;

  /*-------------------------------------------------------------------*/
  /**
   * Display Cinema Hall Configuration Stored Procedure
   *  - Show found according to the given criteria
   */
  public static final int DISP_CINEMA_HALL_CONF_PRT_SHOW_FOUND = 203;

  /**
   * Display Cinema Hall Configuration Stored Procedure
   *  - Show NOT found according to the given criteria
   */
```

```
public static final int DISP_CINEMA_HALL_CONF_PRT_SHOW_NOT_FOUND = 403;

/*-------------------------------------------------------------------*/
/**
 * Background Cinema Hall Configuration Update Stored Procedure
 *  - Show found according to the given criteria
 */
public static final int BCKG_CINEMA_HALL_UPDATE_PRT_SHOW_FOUND = 204;

/**
 * Background Cinema Hall Configuration Update Stored Procedure
 *  - Show NOT found according to the given criteria
 */
public static final int BCKG_CINEMA_HALL_UPDATE_PRT_SHOW_NOT_FOUND
     = 404;


/*-------------------------------------------------------------------*/
/**
 * Find Movies Criteria 1 Stored Procedure
 *  - Movies found according to the given criteria
 */
public static final int FIND_MOVIES_CRIT_1_PRT_MOVIES_FOUND = 205;

/**
 * Find Movies Criteria 1 Stored Procedure
 *  - Movies NOT found according to the given criteria
 */
public static final int FIND_MOVIES_CRIT_1_PRT_MOVIES_NOT_FOUND = 405;

/*-------------------------------------------------------------------*/
/**
 * Find Movies Criteria 2 Stored Procedure
 *  - Movies found according to the given criteria
 */
public static final int FIND_MOVIES_CRIT_2_PRT_MOVIES_FOUND = 206;

/**
 * Find Movies Criteria 2 Stored Procedure
 *  - Movies NOT found according to the given criteria
 */
public static final int FIND_MOVIES_CRIT_2_PRT_MOVIES_NOT_FOUND = 406;

/*-------------------------------------------------------------------*/
/**
 * Find Movies Criteria 3 Stored Procedure
 *  - Movies found according to the given criteria
```

```
 */
public static final int FIND_MOVIES_CRIT_3_PRT_MOVIES_FOUND = 207;

/**
 * Find Movies Criteria 3 Stored Procedure
 *  - Movies NOT found according to the given criteria
 */
public static final int FIND_MOVIES_CRIT_3_PRT_MOVIES_NOT_FOUND = 407;

/*-----------------------------------------------------------------*/
/**
 * Find Movies Criteria 4 Stored Procedure
 *  - Movies found according to the given criteria
 */
public static final int FIND_MOVIES_CRIT_4_PRT_MOVIES_FOUND = 208;

/**
 * Find Movies Criteria 4 Stored Procedure
 *  - Movies NOT found according to the given criteria
 */
public static final int FIND_MOVIES_CRIT_4_PRT_MOVIES_NOT_FOUND = 408;

/*-----------------------------------------------------------------*/
/**
 * Find Movies Criteria 5 Stored Procedure
 *  - Movies found according to the given criteria
 */
public static final int FIND_MOVIES_CRIT_5_PRT_MOVIES_FOUND = 209;

/**
 * Find Movies Criteria 5 Stored Procedure
 *  - Movies NOT found according to the given criteria
 */
public static final int FIND_MOVIES_CRIT_5_PRT_MOVIES_NOT_FOUND = 409;

/*-----------------------------------------------------------------*/
/**
 * Select Deselect Many Seats Stored Procedure
 *  - Seats Selected
 */
public static final int SELECT_DESELECT_SEATS_PRT_SEATS_SELECTED_OK
     = 210;

/**
 * Select Deselect Many Seats Stored Procedure
 *  - Seats Deselected
 */
```

```
public static final int SELECT_DESELECT_SEATS_PRT_SEATS_DESELECTED_OK
     = 211;

/**
 * Select Deselect Many Seats Stored Procedure
 *  - Error when selecting seats or Seats already selected
 */
public static final int SELECT_DESELECT_SEATS_PRT_SEATS_SELECTED_ERROR
     = 410;

/**
 * Select Deselect Many Seats Stored Procedure
 *  - Error when deselecting seats or Seats already deselected
 */
public static final int
    SELECT_DESELECT_SEATS_PRT_SEATS_DESELECTED_ERROR= 411;

/*-----------------------------------------------------------------*/
/**
 * Compute Price and Maybe Pay stored Procedure
 *  - Ticket Price computed successfuly and reservation saved
 *  in the DB
 */
public static final int PURCHASE_TICKETS_PRT_OK = 212;

/**
 * Compute Price and Maybe Pay stored Procedure
 *   - User autnenticated
 */
public static final int PURCHASE_TICKETS_PRT_USER_AUTNENTICATED= 201;

/**
 * Compute Price and Maybe Pay stored Procedure
 *  - User NOT Authenticated
 */
public static final int PURCHASE_TICKETS_PRT_USER_NOT_AUTNENTICATED
     = 401;

/**
 * Compute Price and Maybe Pay stored Procedure
 *  - Error when trying to pay for the tickets
 *  (Transaction error)
 */
public static final int PURCHASE_TICKETS_PRT_ERROR = 412;

/**
 * Compute Price and Maybe Pay stored Procedure
```

```
 *  - Invalid Credit Card
 */
public static final int PURCHASE_TICKETS_PRT_INVALID_CREDIT_CARD = 413;

/*-----------------------------------------------------------------*/
/**
 * Cancel Tickets Stored Procedure
 *  - Tickets canceled
 */
public static final int CANCEL_TICKETS_PRT_OK = 214;

/**
 * Cancel Tickets Stored Procedure
 *  - User Authenticated
 */
public static final int CANCEL_TICKETS_PRT_USER_AUTHENTICATED = 201;

/**
 * Cancel Tickets Stored Procedure
 *  - User Not Authenticated
 */
public static final int CANCEL_TICKETS_PRT_USER_NOT_AUTHENTICATED
     = 401;

/**
 * Cancel Tickets Stored Procedure
 *  - Error while canceling the tickets
 */
public static final int CANCEL_TICKETS_PRT_ERROR= 414;

/*-----------------------------------------------------------------*/
/**
 * Cancel Unpaid Tickets Before Show Stored Procedure
 *  - All unpaid tickets are canceled before the show
 */
public static final int CANCEL_UNPAID_TICKETS_PRT_OK = 215;

/**
 * Cancel Unpaid Tickets Before Show Stored Procedure
 *  - Error when trying to cancel the unpaid tickets
 */
public static final int CANCEL_UNPAID_TICKETS_PRT_ERROR = 415;

/*-----------------------------------------------------------------*/
/**
 * Reject Payment Stored Procedure
 *  - Payment is rejected and all seats are canceled
```

```
 */
public static final int REJECT_PAYMENT_PRT_OK = 216;

/**
 * Reject Payment Stored Procedure
 *  - Error when rejecting the payment
 */
public static final int REJECT_PAYMENT_PRT_ERROR = 416;

/*-------------------------------------------------------------------*/
/**
 * Get Movie Details Stored Procedure
 *  - Movie Details Selected
 */
public static final int MOVIE_DETAILS_PRT_OK = 217;

/**
 * Get Movie Details Stored Procedure
 *  - Error when retrieving the movie details
 */
public static final int MOVIE_DETAILS_PRT_ERROR = 417;

/*-------------------------------------------------------------------*/
/**
 * Rate Movie Stored Procedure
 *  - Movie Rated
 */
public static final int RATE_MOVIE_PRT_OK = 218;

/**
 * Rate Movie Stored Procedure
 *  - Error when rating the movie
 */
public static final int RATE_MOVIE_PRT_ERROR = 418;

/**
 * Rate Movie Stored Procedure
 *  - User authenticated
 */
public static final int RATE_MOVIE_PRT_USER_AUTHENTICATED = 201;

/**
 * Rate Movie Stored Procedure
 *  - User not authenticated
 */
public static final int RATE_MOVIE_PRT_USER_NOT_AUTHENTICATED = 401;
```

```
  /*-----------------------------------------------------------------*/
  /**
   * Movie Location Service Stored Procedure
   *  - Cinemas found according to the given criteria
   */
  public static final int MOVIE_LOCATION_SERVICE_OK = 219;

  /**
   * Movie Location Service Stored Procedure
   *  - Error while searching for Cinemas according to the given criteria
   */
  public static final int MOVIE_LOCATION_SERVICE_ERROR = 419;

  /**
   * Movie Location Service Stored Procedure
   *  - Cinemas not foundaccording to the given criteria
   */
  public static final int MOVIE_LOCATION_SERVICE_NO_DATA = 420;

  /*-----------------------------------------------------------------*/
  /**
   * Authentication & E_Money Stored Procedure
   *  - User authenticated and E-money retrieved
   */
  public static final int Authentication_E_MONEY_PRT_OK = 221;

  /**
   * Authentication & E_Money Stored Procedure
   *  - Error while retrieving the e-money amount
   */
  public static final int Authentication_E_MONEY_ERROR = 421;

  /**
   * Authentication & E_Money Stored Procedure
   *  - User NOT authenticated
   */
  public static final int
      Authentication_E_MONEY_PRT_USER_NOT_AUTHENTICATED = 401;
  /*-----------------------------------------------------------------*/

}

package constants;

/**
 * Defines the application setting parameters
 *
 * @author Mihai Balan (s031288)
```

```
 *
 */
public final class SystemConstants {

  public static final int MAX_NO_CREDIT_CARDS = 6;
  public static final int MAX_NO_TICKETS      = 10;
  public static final int CREDIT_CARD_EXP_YEAR_MIN = 2007;
  public static final int CREDIT_CARD_EXP_YEAR_MAX = 2015;
  public static final String NATIONAL_CURRENCY = "DKK";

}

package cryptography;

import org.bouncycastle.crypto.params.ParametersWithIV;

/**
 * Constructs an AES Key used for encryption - decryption with AES
 *
 * @author Mihai Balan(s031288), Wojciech Dobrowolski
 *
 */
public class AesKey {

  static public AesKey getInstance(){
    return theInstance;
  }

  protected AesKey(){

  }

  public void setKey(ParametersWithIV key){
    this.AesKey = key;
  }

  public ParametersWithIV getKey(){
    return this.AesKey;
  }

  ParametersWithIV AesKey = null;
  public static AesKey theInstance = new AesKey();
}

package cryptography;

import org.bouncycastle.crypto.*;
import org.bouncycastle.crypto.digests.SHA1Digest;
```

```
import org.bouncycastle.crypto.engines.*;
import org.bouncycastle.crypto.generators.PKCS12ParametersGenerator;
import org.bouncycastle.crypto.modes.*;
import org.bouncycastle.crypto.paddings.PaddedBufferedBlockCipher;
import org.bouncycastle.crypto.params.*;

/**
 * Provides methods for performing different
 * encryption - decryption operations
 *
 *  @author Mihai Balan(s031288), Wojciech Dobrowolski
 *
 */
public class Encryptor {

  private PaddedBufferedBlockCipher cipher;
  private KeyParameter key;

  public Encryptor(byte[] key){
    cipher = new PaddedBufferedBlockCipher(
         new CFBBlockCipher(
           new DESEngine(),8));
    this.key = new KeyParameter (key);
  }


  /**
   * Initialize the cryptographic engine.
   * The string should be at least 8 chars long.
   */
  public Encryptor( String key ){
       this( key.getBytes() );
    }

    /**
     * Private routine that does the gritty work.
     *
     */
    private byte[] callCipher( byte[] data )
                      throws CryptoException {
      System.out.println("Cipher␣has␣been␣called␣...");
        int size = cipher.getOutputSize( data.length );
        byte[] result = new byte[ size ];
        int olen = cipher.processBytes( data, 0, data.length, result, 0 )
            ;
        olen += cipher.doFinal( result, olen );
```

```
        if( olen < size ){
            byte[] tmp = new byte[ olen ];
            System.arraycopy( result, 0, tmp, 0, olen );
            result = tmp;
        }

        return result;
    }

    /**
     * Encrypt arbitrary byte array, returning the
     *  encrypted data in a different byte array.
     */
    public synchronized byte[] encrypt( byte[] data )
                throws CryptoException {
        if( data == null || data.length == 0 ){
            return new byte[0];
        }

        cipher.init( true, key );
        return callCipher( data );
    }



    /**
     * Encrypts a string
     */
    public byte[] encryptString( String data )
                throws CryptoException {
        if( data == null || data.length() == 0 ){
            return new byte[0];
        }

        return encrypt( data.getBytes() );
    }


    /**
     * Decrypts arbitrary data
     */
    public synchronized byte[] decrypt( byte[] data )
                throws CryptoException {
        if( data == null || data.length == 0 ){
          System.out.println("Data␣turned␣out␣to␣be␣null␣...");
            return new byte[0];
        }
```

```
      cipher.init( false, key );
      System.out.println("Decrypted␣data:␣" );
      return callCipher( data );
  }



  /**
   * Decrypts a string that was previously encoded
   * using encryptString.
   */
  public String decryptString( byte[] data )
                throws CryptoException {
      if( data == null || data.length == 0 ){
        System.out.println("Something␣happened␣with␣the␣data␣...");
          return "";
      }
      System.out.println("Decrypting␣data␣..." + data.toString());
      return new String( decrypt( data ));
  }


  public ParametersWithIV createKey(String salt, String key){
  PBEParametersGenerator generator =
    new PKCS12ParametersGenerator(new SHA1Digest());
  generator.init(
      PBEParametersGenerator.PKCS12PasswordToBytes(key.toCharArray()),
      salt.getBytes(), 1024);
  // Generate a 128 bit key w/ 128 bit IV
  ParametersWithIV ret =
    (ParametersWithIV)generator.generateDerivedParameters(128, 128);
  return ret;
}


  public byte[] encryptWithAES(ParametersWithIV key, byte[] msg){
  BufferedBlockCipher cipher = new PaddedBufferedBlockCipher(
      new CBCBlockCipher(new AESFastEngine()));
  cipher.init(true, key);
  byte[] result = new byte[cipher.getOutputSize(msg.length)];
  int len = cipher.processBytes(msg, 0,
      msg.length, result, 0);
  try {
    cipher.doFinal(result, len);
  } catch (CryptoException ce) {
    System.out.println("Encryption␣with␣AES␣error␣...");
    ce.printStackTrace();
```

```
    }
    return result;
  }

  public byte[] decryptWithAES(ParametersWithIV key, byte[] result){
    BufferedBlockCipher cipher = new PaddedBufferedBlockCipher(
        new CBCBlockCipher(new AESFastEngine()));
    cipher.init(false, key);
    byte[] res =
      new byte[cipher.getOutputSize(result.length)];
    int leng = cipher.processBytes(result, 0,
        result.length, res, 0);
    try {
      cipher.doFinal(res, leng);
    } catch (CryptoException ce) {
      System.out.println("Decryption␣with␣AES␣error␣...");
      ce.printStackTrace();
    }
    return res;
  }

}

package gui.authentication;

import gui.GenericGUI;
import gui.customdialogwindows.CanvasAlert;
import gui.customdialogwindows.DialogWindow;
import gui.mainmenu.MenuScreen;

import java.io.IOException;
import javax.microedition.lcdui.*;

import networkoperations.NetworkCommunicationFacade;

import model.init.InitModel;

import start.Start;
import constants.CustomAlertTypes;
import cryptography.Encryptor;

/**
 *  Displays the screen used to enter user credentials, encryption key,
 *  Authentication Server URL, Secure Server URL.
 *  It performs the user authentication.
 *  It extends the GenericGUI super class
 *
 * @author Mihai Balan (s031288)
```

```
 *
 */
public class AuthenticationGUI extends GenericGUI{

  // the authentication screen
  private static Displayable screen = null;

  //the starting point of the application
  public static Start startingPoint;


  //  the exit and select commands
  private static Command exitCommand;
  private static Command loginCommand;

  // user, password, and key text boxes
  private TextField key;
  private TextField user;
  private TextField passw;
  private Image imgUp;
  private ImageItem imgThemeUp;

  private String textFieldUser = "";
  private String textFieldPassw = "";
  private String textFieldKey = "";

  // predifined authentication values to be displayed and used
  private String userField = "adm";
  private String passwField = "12345678";
  private String keyField = "12345678";

  /** reference to the Encryptor class in order
   to perform encryption/decryption operations */
  public Encryptor encryptor = null;



  /**
   * Constructs an instance of the class
   */
  public AuthenticationGUI () {
    this.textFieldUser = "User␣Name";
    this.textFieldPassw = "Password";
    this.textFieldKey = "Private␣Key";

  }
```

```
/**
 * Returns the displayable authentication screen
 * @return screen Returns the Authentication screen
 */
public Displayable getScreen() {
  return screen;
}

/**
 * Function of the pressed button i.e. exit or Login
 * the user returns to the main menu or the Authentication
 * is performed by using the Authenticate class in order to
 * perform network operation (for authentication purposes)
 * on a separate thread.
 *
 * @param c The executed command
 * @param s The main menu form
 *
 */
public void commandAction(Command c, Displayable s) {

  CanvasAlert alert;
  try {
    // go back to main menu
    if (c == exitCommand) {

      // cleanBeforeExit();
      DialogWindow reallyExit = new DialogWindow(
          display,
          getScreen(),
          "Exit␣Application?",
          "Are␣you␣sure␣that␣you␣want␣to␣exit␣the␣application?",
          "question",
          "/dialogIcons/exitTheme",
          new MenuScreen().startingpoint);

      display.setCurrent(reallyExit);

    }// check if the key is valid and add it to the record store
    else if (c == loginCommand) {
      System.out.println("-----␣Authentication␣--␣Proceed␣command␣
          pressed");
      keyField  = key.getString();
      userField = user.getString();
      passwField = passw.getString();

      if (keyField.length() < 8){
```

```
        // display an alarm if the key is shorter that 8 characters
        alert = new CanvasAlert(
            display,
            getScreen(),
            "Invalid␣Key!",
            "The␣Key␣has␣to␣be␣at␣least␣8␣characters␣long!␣Please␣
                provide␣the␣key␣obtained␣when␣you␣received␣the␣
                application!",
            "error",
            CustomAlertTypes.ALERT_ERROR);

       }else{

        // initiates the midlet data model from RMS
        // i.e. keys, username, password, tickets, etc.

        InitModel.initModelFromRMS(user.getString(), passw.getString(),
            key.getString());
        Runtime runtime = Runtime.getRuntime();
        long t = runtime.freeMemory();
        System.out.println("************************Memory␣before␣auth:"
             + t);

        NetworkCommunicationFacade.authenticate(display, getScreen(),
            new MenuScreen(), user.getString(), passw.getString(), key.
            getString());

        clean();
        alert = null;
        long t1 = runtime.freeMemory();
        System.out.println("************************Memory␣after␣auth:"
             + t1);


       }
      }
    } catch (Exception e) {
      e.printStackTrace();
      alert = new CanvasAlert(
           display,
           getScreen(),
           "Key␣Not␣Saved!",
           "Error␣while␣saving␣the␣key␣to␣the␣phone␣memory!",
           "error",
           CustomAlertTypes.ALERT_ERROR);
```

```
    display.setCurrent(alert);
  }

}// end CommandAction()


protected void initModel() throws Exception {

  Runtime runtime = Runtime.getRuntime();
  long t = runtime.freeMemory();
  System.out.println("**********************Memory␣before␣auth␣init:"
       + t);


}// end initModel()


/**
 * Creates the Authentication Screen
 *
 * @throws Exception
 *
 */
protected void createView() throws Exception {
  exitCommand = new Command("EXIT", Command.EXIT, 1);
  loginCommand = new Command("LOGIN", Command.OK, 1);

  // create the text fields and add them to the form
  user  = new TextField(textFieldUser, userField, 40, TextField.ANY);
  passw = new TextField(textFieldPassw, passwField, 40, TextField.
      PASSWORD);
  key   = new TextField(textFieldKey, keyField, 40, TextField.ANY);

  try{
    imgUp = Image.createImage("/theme_red/authentication/theme_up.png")
        ;
    imgThemeUp = new ImageItem("", imgUp, Item.LAYOUT_TOP | Item.
        LAYOUT_CENTER , "Theme_Img_Up");

  }catch(IOException ioe){
    System.out.println("Theme␣image␣exception!");
  }

  Ticker ticker = new Ticker("Please␣provide␣your␣user␣name,␣password␣
      and␣private␣key!");
  screen = new Form("Please␣Authenticate!");
  ((Form)screen).setTicker(ticker);
```

```
    ((Form)screen).append(imgUp);
    ((Form)screen).append(user);
    ((Form)screen).append(passw);
    ((Form)screen).append(key);

    // add the commands to the form
    screen.addCommand(exitCommand);
    screen.addCommand(loginCommand);
  }

  /**
   * Update the view - maybe refresh the fields
   *
   */
  protected void updateView() throws Exception {}

  private void clean(){

    key = null;
    user = null;
    passw = null;
    imgUp = null;
    imgThemeUp = null;
    exitCommand = null;
    loginCommand = null;
    textFieldUser = null;
    textFieldPassw = null;
    textFieldKey = null;
    userField  = null;
    passwField = null;
    keyField   = null;
    encryptor = null;
    screen = null;
    System.gc();

  }


}// end class

package gui.customdialogwindows;


import gui.GUIHelper;

import java.util.*;
import javax.microedition.lcdui.*;
```

```
import start.Start;
import tools.ImageProcessing;

/**
 * Creates a customizable alert screen using different low level
 * components and a more user-friendly look and feel.
 *
 * @author s031288, Mihai Balan
 */
public class CanvasAlert extends Canvas {

  // the display to draw on
  private Display   display;

  private Displayable next;
  // a thread used to display the image for a no. of seconds

  // properties of the alert
  private String title;
  private String msg;
  private String image;
  private int    alertType;
  private long   alertDuration;
  private boolean timerAlert = false;
  private Start  midlet;
  private boolean pressKey = true;


  private Timer timer = new Timer();

  /**
   * Constructs a canvas based alert screen containing textual
   * information about the alert and an image
   *
   * @param display The display to draw on
   * @param next    The next screen to be displayed after the image
   * @param title   The title of the alart message
   * @param msg     The message of the alert
   * @param image   The image to be displayed with the alert
   * @param alertType The alert type @see constants.CustomAlertTypes
   */
  public CanvasAlert( Display display, Displayable next, String title,
      String msg, String image, int alertType){

    this.display   = display;
    this.next      = next;
    this.title     = title;
```

```
    this.msg       = msg;
    this.image     = image;
    this.alertType = alertType;

    display.setCurrent(this);
  }

  public CanvasAlert( Display display, Displayable next, String title,
      String msg, String image, int alertType, boolean show){

    this.display   = display;
    this.next      = next;
    this.title     = title;
    this.msg       = msg;
    this.image     = image;
    this.alertType = alertType;

  }


public CanvasAlert( Display display, Displayable next, boolean pressKey,
     String title, String msg, String image, int alertType){

    this.display   = display;
    this.next      = next;
    this.pressKey  = pressKey;
    this.title     = title;
    this.msg       = msg;
    this.image     = image;
    this.alertType = alertType;

  }

  /**
   * Constructs a canvas based alert screen containing textual
   * information about the alert and an image.
   * It uses a Tinmer thread to display the canvas alert
   * for a given period of time
   *
   * @param display     The display to draw on
   * @param next        The next screen to be displayed after the image
   * @param timerAlert  If the alarm is to be displayed for a no of sec
   *                    and then cancelled automatocally
   * @param alertDuration The duration in ms the alarm is displayed
   * @param title       The title of the alart message
   * @param msg         The message of the alert
   * @param image       The image to be displayed with the alert
```

```
 * @param alertType   The alert type @see constants.CustomAlertTypes
 */
public CanvasAlert( Display display, Displayable next, boolean
    timerAlert, long alertDuration, String title, String msg, String
    image, int alertType){

  this.display     = display;
  this.next        = next;
  this.timerAlert  = timerAlert;
  this.alertDuration = alertDuration;
  this.title       = title;
  this.msg         = msg;
  this.image       = image;
  this.alertType   = alertType;

  display.setCurrent(this);

}


/**
 * Constructs a canvas based alert screen containing textual
 * information about the alert and an image.
 * It is used as a good bye screen.
 * It uses a Tinmer thread to display the canvas alert
 * for a given period of time.
 *
 * @param display     The display to draw on
 * @param next        The next screen to be displayed after the image
 * @param timerAlert  If the alarm is to be displayed for a no of sec
 *                    and then cancelled automatocally
 * @param alertDuration The duration in ms the alarm is displayed
 * @param title       The title of the alart message
 * @param msg         The message of the alert
 * @param image       The image to be displayed with the alert
 * @param alertType   The alert type @see constants.CustomAlertTypes
 */
public CanvasAlert( Display display, Displayable next, boolean
    timerAlert, long alertDuration, String title, String msg, String
    image, int alertType, Start midlet){

  this.display     = display;
  this.next        = next;
  this.timerAlert  = timerAlert;
  this.alertDuration = alertDuration;
  this.title       = title;
  this.msg         = msg;
```

```
  this.image       = image;
  this.alertType   = alertType;
  this.midlet      = midlet;

  display.setCurrent(this);
}


/** The image and text displaying takes place in here
 * Get the splash image from a .png file, convert it to a byte array
 * and then display it on the scree
 *
 * @param g The graphocs to draw on
 */
protected void paint( Graphics g ){
  Image splash;
  int w = getWidth();
  int h = getHeight();

  try {
    ImageProcessing imgProcc = new ImageProcessing();
    splash = imgProcc.getImage(image);

    // clear the background
    g.setColor(255, 255, 255);
    g.fillRect(0, 0, getWidth(), getHeight());

    // draw the image
    g.drawImage(splash, w/2, h/3,
        Graphics.VCENTER | Graphics.HCENTER);

    // set the font and color of the alert title and msg
    // and draw them on the screen
    CanvasAlertTools alertTool = new CanvasAlertTools(g, alertType, w,
        h);
    alertTool.drawTitle(title, pressKey);

    if(title.equals("My␣Tickets␣Help")){

      alertTool.drawMessage(msg, Font.SIZE_MEDIUM);

    }else if(title.equals("Invalid␣UI␣entries!")){

      Font msgFont = Font.getFont(
          Font.FACE_PROPORTIONAL,
          Font.STYLE_BOLD,
          Font.SIZE_MEDIUM);
```

```
      g.setFont(msgFont);
      g.setColor(255,0,0);

      GUIHelper.dynamicDrawMessage(g, msgFont, 0, 0, 0, msg, w, h
          , 10, 11*h/20 );

    } else if(title.equals("Master␣Thesis␣Project!")){

      Font msgFont = Font.getFont(
          Font.FACE_PROPORTIONAL,
          Font.STYLE_BOLD,
          Font.SIZE_MEDIUM);

      g.setFont(msgFont);
      g.setColor(0,0,255);

      GUIHelper.dynamicDrawMessage(g, msgFont, 0, 0, 0, msg, w, h
          , 10, 11*h/20 );

    } else{

      alertTool.drawMessage(msg);

    }// end if()

  } catch (Exception e) {
    e.printStackTrace();
    // if the image cannot be drawn, write some text
    g.drawString("Mobile␣Cinema", w/2, h/2,
        Graphics.BASELINE | Graphics.HCENTER);
  }
}
/**
 * The splash screen disapears when any key is pressed
 *
 * @param keyCode The code of the pressed key
 */
protected void keyPressed(int keyCode){
  if(pressKey){
    dismiss();
  }
}

/**
 * Displays the image for a period of time and then goes to the main
     menu
```

```
   */
  protected void showNotify(){
    if(timerAlert)
      timer.schedule(new CountDown(), alertDuration);
  }

  /**
   * Cancel the timer and set the screen to the next screen set up before
   *
   */
  private void dismiss(){
    if (timerAlert)
      timer.cancel();

    // in case the canvas alert is not a good bye screen
    if(midlet == null){
      display.setCurrent(next);

      // if the canvas alert is a good bye screen exit the application
    }else {
      midlet.destroyApp(false);
      midlet.notifyDestroyed();
    }
  }

  /**
   * Thread used to count down the no. of seconds
   * to display the image
   *
   * @author s031288, Mihai Balan
   *
   */
  private class CountDown extends TimerTask {
    public void run(){
      dismiss();
    }
  }// end CountDown

}// end class

package gui.customdialogwindows;

import constants.CustomAlertTypes;
import javax.microedition.lcdui.*;

/**
 * Performs several computations for tokenizing a msg into words
 * and drawing the text on the canvas by splitting the text
```

```
 * into full words
 *
 * @author s031288, Mihai Balan
 *
 */
public class CanvasAlertTools{

  private Graphics g  = null;
  private int alertType = 0;
  private int w       = 0;
  private int h       = 0;


  /**
   * Constructor for the tool
   *
   * @param g The graphics object to draw onto the Canvas
   * @param alertType @see constants.AlertTypes
   * @param w The width of the screen
   * @param h The heght of the screen
   */
  public CanvasAlertTools(Graphics g, int alertType, int w, int h){
    this.g       = g;
    this.alertType = alertType;
    this.w       = w;
    this.h       = h;
  }


  /**
   * Set the title and msg color function of the alert type
   * i.e. red for error and blue for info
   */
  public void setTextColor(){

    if(alertType == CustomAlertTypes.ALERT_ERROR || alertType ==
        CustomAlertTypes.ALERT_WARNING){
      g.setColor(255,0,0); //red
    }

    if(alertType == CustomAlertTypes.ALERT_INFO){
      g.setColor(0, 0, 255); // blue
    }

  }// end setTextColor(int alertType)
```

```
/**
 * Set the font and color of the alert msg, based on
 * the alert type and draw it on the canvas. It also
 * tokenizes the msg into words and make some computations
 * to feet the text and full words on the screen
 *
 * @param msg The message to be displayed on the mobile screen
 */
public void drawMessage(String msg){

  // define the msg font
  Font msgFont = Font.getFont(
      Font.FACE_PROPORTIONAL,
      Font.STYLE_PLAIN,
      Font.SIZE_LARGE);

  // set the message font and color
  g.setFont(msgFont);
  setTextColor();

  // calculate msg to be drawn by tokenizing on words and draw the
       message
  int msgWidth  = msgFont.stringWidth(msg);
  int charHeight = msgFont.getHeight();
  int noRows    = (int)msgWidth/(w-2) + 1;
  int msgLength = msg.length();

  int from = 0;
  String m = "";
  int to = 0;

  for(int i=0; i<noRows; i++){

    if(i == (noRows-1)){
      m = msg.substring(from, msgLength);
      g.drawString(m, w/2, 11*h/20 + i*charHeight, Graphics.TOP |
           Graphics.HCENTER);

    } else{
      to = msg.lastIndexOf(32, (i+1)*(int)(i + msgLength)/noRows);
      m = msg.substring(from +(i==0?0:1), to);
      if (msgFont.stringWidth(m) > (w - 2)){
        to = msg.lastIndexOf(32, to-i);
      }
      g.drawString(m, w/2, 11*h/20 + i*charHeight, Graphics.TOP |
           Graphics.HCENTER);
    }
```

```
    from = to;

  }// end for()

}// end drawMessage()

/**
 * Set the font and color of the alert msg, based on
 * the alert type and draw it on the canvas. It also
 * tokenizes the msg into words and make some computations
 * to feet the text and full words on the screen
 *
 * @param msg The message to be displayed on the mobile screen
 * @param fontSize The size of the font used to draw the message
 */
public void drawMessage(String msg, int fontSize){

  // define the msg font
  Font msgFont = Font.getFont(
      Font.FACE_PROPORTIONAL,
      Font.STYLE_PLAIN,
      fontSize);

  // set the message font and color
  g.setFont(msgFont);
  setTextColor();

  // calculate msg to be drawn by tokenizing on words and draw the
      message
  int msgWidth  = msgFont.stringWidth(msg);
  int charHeight = msgFont.getHeight();
  int noRows    = (int)msgWidth/(w-2) + 1;
  int msgLength = msg.length();

  int from = 0;
  String m = "";
  int to = 0;

  for(int i=0; i<noRows; i++){

    if(i == (noRows-1)){
      m = msg.substring(from, msgLength);
      g.drawString(m, w/2, 11*h/20 + i*charHeight, Graphics.TOP |
          Graphics.HCENTER);

    } else{
```

```
        to = msg.lastIndexOf(32, (i+1)*(int)(i + msgLength)/noRows);
        m = msg.substring(from +(i==0?0:1), to);
        if (msgFont.stringWidth(m) > (w - 2)){
          to = msg.lastIndexOf(32, to-i);
        }
        g.drawString(m, w/2, 11*h/20 + i*charHeight, Graphics.TOP |
            Graphics.HCENTER);
      }

      from = to;

    }// end for()

  }// end drawMessage()

  /**
   * Set the font and color of the alert title based on
   * the alert type and draw it on the canvas
   *
   * @param title The title to be displayed on the alert screen
   */
  public void drawTitle(String title, boolean pressKey){
    //defines the font used for title and msg
    Font titleFont = Font.getFont(
        Font.FACE_PROPORTIONAL,
        Font.STYLE_BOLD,
        Font.SIZE_LARGE);

    int charHeight = titleFont.getHeight();

    g.setFont(titleFont);
    setTextColor();
    g.drawString(title, w/2, h/14, Graphics.TOP | Graphics.HCENTER);

    g.setFont(titleFont);
    g.setColor(0,0,255); //blue

    if(pressKey){
      g.drawString("Press␣any␣key␣to␣continue!", w/2, h-charHeight/4,
          Graphics.BASELINE | Graphics.HCENTER);
    }

  }// end drawTitle()


}// end class

package gui.customdialogwindows;
```

```
import gui.GUIHelper;
import gui.mytickets.MyTicketTools;
import gui.mytickets.MyTicketsMainMenu;

import javax.microedition.lcdui.*;

import org.bouncycastle.asn1.ocsp.Request;

import constants.CustomAlertTypes;
import constants.Protocol_Step_Constants;

import networkoperations.SendMessage;


import model.beans.otherbeans.CreditCardBean;
import model.beans.otherbeans.TicketBean;
import model.beans.requestbeans.Cancel_Tickets_Req_Bean;
import model.update.UpdateModel;

import start.Start;


/**
 * Creates a customizable alert screen using different low level
 * components and a more user-friendly look and feel.
 *
 * @author Mihai Balan
 */
public class DialogWindow extends Canvas {

  // the display to draw on
  private Display   display;

  private Displayable previous;
  private Displayable next;

  // properties of the alert
  private String title;
  private String msg;
  private String iconName;
  private String bckgImgName;
  private Start  midlet = null;

   // NOT highlighted buttons
  private String[] optionDeselected = {
      "/dialogIcons/yesDeselected.png",
```

```
    "/dialogIcons/noDeselected.png"
};

// the highlighted buttons
private String[] optionSelected = {
    "/dialogIcons/yesSelected.png",
    "/dialogIcons/noSelected.png"
};

// the images for building the YES and NO options
Image[] deselectedImgs;
Image[] selectedImgs;

// the curent selected option i.e. YES or NO
private int selectedOptionIndex = 0;

ChoiceGroup cgCreditCard;
ChoiceGroup cgTickets;
CreditCardBean[] walletCC;
TicketBean[]   tickets;
private Cancel_Tickets_Req_Bean cancelTicketReqBean;

/**
 * Constructs a canvas based alert screen containing textual
 * information about the alert and an image
 *
 * @param display The display to draw on
 * @param next    The next screen to be displayed after the image
 * @param title   The title of the alart message
 * @param msg     The message of the alert
 * @param image   The image to be displayed with the alert
 * @param alertType The alert type @see constants.CustomAlertTypes
 */
public DialogWindow(Display display, Displayable previous,
    String title, String msg,
    String iconName, String bckgImgName,
    Start midlet){

  this.display     = display;
  this.previous    = previous;
  this.title       = title;
  this.msg         = msg;
  this.iconName    = iconName;
  this.bckgImgName = bckgImgName;
  this.midlet      = midlet;

  selectedImgs  = GUIHelper.createSelectedButtons(optionSelected);
```

```
  deselectedImgs = GUIHelper.createDeselectedButtons(optionDeselected);

  display.setCurrent(this);

}// end DialogWindow()


public DialogWindow(Display display,
    Displayable previous, Displayable next,
    String title, String msg,
    String iconName, String bckgImgName,
    Start midlet){

  this.display     = display;
  this.previous    = previous;
  this.next        = next;
  this.title       = title;
  this.msg         = msg;
  this.iconName    = iconName;
  this.bckgImgName = bckgImgName;
  this.midlet      = midlet;

  selectedImgs  = GUIHelper.createSelectedButtons(optionSelected);
  deselectedImgs = GUIHelper.createDeselectedButtons(optionDeselected);

  display.setCurrent(this);

}// end DialogWindow()


public DialogWindow(Display display, Displayable previous,
    String title, String msg,
    String iconName, String bckgImgName,
    Start midlet,
    ChoiceGroup cgCreditCard, CreditCardBean[] walletCC){

  this.display     = display;
  this.previous    = previous;
  this.title       = title;
  this.msg         = msg;
  this.iconName    = iconName;
  this.bckgImgName = bckgImgName;
  this.midlet      = midlet;
  this.cgCreditCard = cgCreditCard;
  this.walletCC    = walletCC;

  selectedImgs  = GUIHelper.createSelectedButtons(optionSelected);
```

```
  deselectedImgs = GUIHelper.createDeselectedButtons(optionDeselected);

  display.setCurrent(this);

}// end DialogWindow()

public DialogWindow(
    Display display,
    Displayable previous,
    Displayable next,
    String title,
    String msg,
    String iconName,
    String bckgImgName,
    Start midlet,
    ChoiceGroup cgTickets,
    TicketBean[] tickets){

  this.display     = display;
  this.previous    = previous;
  this.next        = next;
  this.title       = title;
  this.msg         = msg;
  this.iconName    = iconName;
  this.bckgImgName = bckgImgName;
  this.midlet      = midlet;
  this.cgTickets  = cgTickets;
  this.tickets    = tickets;

  selectedImgs  = GUIHelper.createSelectedButtons(optionSelected);
  deselectedImgs = GUIHelper.createDeselectedButtons(optionDeselected);

  display.setCurrent(this);

}// end DialogWindow()

public DialogWindow(
    Display display,
    Displayable previous,
    String title,
    String msg,
    String iconName,
    String bckgImgName,
    Start midlet,
    ChoiceGroup cgTickets,
    TicketBean[] tickets){
```

```
  this.display     = display;
  this.previous    = previous;
  this.title       = title;
  this.msg         = msg;
  this.iconName    = iconName;
  this.bckgImgName = bckgImgName;
  this.midlet      = midlet;
  this.cgTickets  = cgTickets;
  this.tickets    = tickets;

  selectedImgs  = GUIHelper.createSelectedButtons(optionSelected);
  deselectedImgs = GUIHelper.createDeselectedButtons(optionDeselected);

  display.setCurrent(this);

}// end DialogWindow()


/** The image and text displaying takes place in here
 * Get the splash image from a .png file, convert it to a byte array
 * and then display it on the scree
 *
 * @param g The graphics to draw on
 */
protected void paint( Graphics g ){

  int w = getWidth();
  int h = getHeight();

  try {
    // draw the background
    GUIHelper.drawBackground(g, bckgImgName, w, h);

    // draw the title
    GUIHelper.drawTitle(g, title, w, h);

    // draw the icon
    GUIHelper.drawIcon(g, iconName, w, h);

    // draw the dialog message on the screen
    int buttonHeightPos = GUIHelper.drawMessage(g, msg, w, h);

    // draw the buttons
    GUIHelper.drawButtons(g,
        deselectedImgs, selectedImgs,
        w, h,
        buttonHeightPos,
```

```
        selectedOptionIndex);

  } catch (Exception e) {

    System.out.println("Eception␣in␣dialog␣window:" + e.getMessage());
    e.printStackTrace();

    // if the image cannot be drawn, write some text
    g.drawString("Mobile␣Cinema", w/2, h/2,
        Graphics.BASELINE | Graphics.HCENTER);
  }
}
/**
 * Update the value of the current selected option in the list
 * and repaint the screen.
 *
 * @param keyCode The code of the pressed key
 */
protected void keyPressed( int keyCode ){

  if ((getGameAction(keyCode) == Canvas.LEFT)){

    if(selectedOptionIndex - 1 >= 0){

      selectedOptionIndex--;
      repaint();

    } else if(selectedOptionIndex == 0){

      selectedOptionIndex = optionDeselected.length - 1;
      repaint();
    }

  } else if ((getGameAction(keyCode) == Canvas.RIGHT)){

    if(selectedOptionIndex - 1 >= 0){

      selectedOptionIndex--;
      repaint();

    } else if(selectedOptionIndex == 0){

      selectedOptionIndex = optionDeselected.length - 1;
      repaint();
    }

  } else if ((getGameAction(keyCode) == Canvas.FIRE)){
```

```
// if this is used to exit the application
if (title.equals("Exit␣Application?")){

  if(selectedOptionIndex == 0)
    // cleanUpBeforeExit();
    exitApplication();
  else if(selectedOptionIndex == 1)
    display.setCurrent(previous);

} // end if (exit)


// if this canvas is used as a dialog box
// for deleting a credit card
if (title.equals("Remove␣Credit␣Card?")){

  if(selectedOptionIndex == 0){

    UpdateModel.deleteCreditCardAndUpdateAllCreditCards(cgCreditCard
        , display);

}
  else if(selectedOptionIndex == 1){
    display.setCurrent(previous);
  }

}// end if (delete CC)


// if this canvas is used as a dialog box
// for cacelling a ticket
if (title.equals("Cancel␣Ticket?")){

  if(selectedOptionIndex == 0){


      boolean[] cgSelected = new boolean[cgTickets.size()];
      cgTickets.getSelectedFlags(cgSelected);
      int selectedIndex = cgTickets.getSelectedIndex();

      try{
        if(Start.tickets[selectedIndex].getTKTPurchaseMethod().
            equals("At␣the␣Cinema")){
          CanvasAlert alert = new CanvasAlert(
              display,
              new MyTicketsMainMenu().prepareScreen(),
```

```
                "Ticket␣cannot␣be␣canceled!",
                "Tickets␣purchased␣using␣AT␣THE␣CINEMA␣payment␣method␣
                    cannot␣be␣canceled!",
                "error",
                CustomAlertTypes.ALERT_ERROR);

            }else{
              String[] tktID = {Start.tickets[selectedIndex].getTKTID()
                  };

              cancelTicketReqBean = new Cancel_Tickets_Req_Bean();

              cancelTicketReqBean.setUserName(Start.userName);
              cancelTicketReqBean.setNoOfTickets(1);
              cancelTicketReqBean.setReservationID(Start.tickets[
                  selectedIndex].getTKTReservationID());
              cancelTicketReqBean.setTicketID(tktID);

              SendMessage sm = new SendMessage(
                  display,
                  Protocol_Step_Constants.PRT_STEP_CANCEL_TICKETS,
                  previous,
                  cancelTicketReqBean);
              sm.setCancelTicketsData(next, cgTickets, tickets,
                  cancelTicketReqBean);
              sm.go();

            }// end if(payment method)

          }catch(Exception e){
            System.out.println("Exception␣in␣Dialog␣window␣when␣trying␣
                to␣cancel␣a␣ticket!");
            e.printStackTrace();
          }

        }else if(selectedOptionIndex == 1){
          display.setCurrent(previous);
        }

      }// end if (cancel TKT)

    } // end if(FIRE)

  }// end keyPressed


  /**
```

```
   * Exit the application
   *
   */
  private void exitApplication(){
    if(midlet != null){
      midlet.destroyApp(false);
      midlet.notifyDestroyed();
    }
  }// end exitApplication()

}// end class

package gui.help;

import java.io.IOException;
import javax.microedition.lcdui.*;

import start.Start;

import gui.GenericGUI;
import gui.customdialogwindows.CanvasAlert;
import gui.customdialogwindows.DialogWindow;
import gui.mainmenu.MenuScreen;
import constants.CustomAlertTypes;


/**
 *  Displays the help main screen
 *
 *  It extends the GenericGUI super class
 *
 * @author s031288, Mihai Balan
 *
 */
public class MainHelpGUI extends GenericGUI implements ItemStateListener{

  // the help screen
  private static Displayable screen = null;

  //  the commands
  private static Command backCommand;
  private static Command exitCommand;

  // UI components
  private ChoiceGroup helpTopicsUI;
  private StringItem helpDetailsUI;
  private Image     imgUp;
  private ImageItem imgThemeUp;
```

```
private String[] helpTopicsValues = {};
private String[] helpDetailsValues = {};
private String helpDetailValue = "";

/**
 * Constructs an instance of the class
 */
public MainHelpGUI(){
}


/**
 * Returns the displayable MainHelpGUI screen
 * @return screen Returns the MainHelpGUI screen
 */
public Displayable getScreen() {
  return screen;
}

/**
 *
 * @param c The executed command
 * @param s The main menu form
 *
 */
public void commandAction(Command c, Displayable s) {
  try {

    if (c == backCommand){
      display.setCurrent(new MenuScreen());

    }// end if (c == mainCommand)

    if (c == exitCommand){
      DialogWindow reallyExit = new DialogWindow(
          display,
          new MainHelpGUI().getScreen(),
          "Exit␣Application?",
          "Are␣you␣sure␣that␣you␣want␣to␣exit␣the␣application?",
          "question",
          "/dialogIcons/exitTheme",
          new MenuScreen().startingpoint);

      display.setCurrent(reallyExit);

    } // end if (c == exitCommand)
```

```
  } catch (Exception e) {
    e.printStackTrace();
    CanvasAlert keyErrorAlert = new CanvasAlert(
        display,
        getScreen(),
        "Main␣Help␣Error!",
        "Error␣in␣the␣Main␣Help␣Screen!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    display.setCurrent(keyErrorAlert);
  }
}


/**
 * Initialize the model
 */
protected void initModel() throws Exception {
  // populate the combo box and string items
  helpTopicsValues = new String[6];
  helpTopicsValues[0] = "MY␣WALLET";
  helpTopicsValues[1] = "MY␣TICKETS";
  helpTopicsValues[2] = "MY␣SETTINGS";
  helpTopicsValues[3] = "FIND␣MOVIE";
  helpTopicsValues[4] = "LOG␣IN";
  helpTopicsValues[5] = "CREDIT␣CARD␣SECURITY";

  helpDetailsValues = new String[helpTopicsValues.length];
  helpDetailsValues[0] =
    "MY␣SECURE␣WALLET␣feature␣provides␣a␣mean␣for␣storing␣"       +
    "your␣credit␣card␣data␣for␣easy␣access␣and␣online␣payment␣of␣" +
    "purchased␣cinema␣tickets.␣All␣data␣is␣stored␣encrypted␣in␣the␣" +
    "phone␣memory␣using␣a␣keyword␣known␣only␣by␣you.␣The␣wallet␣is␣" +
    "protected␣by␣a␣PIN␣code.␣If␣the␣PIN␣code␣is␣entered␣wrong␣3␣times
        ␣" +
    "in␣a␣row,␣the␣content␣of␣the␣wallet␣is␣reseted.␣This␣way␣one␣can␣"
         +
    "protect␣his/her␣data␣incase␣his/her␣phone␣is␣stolen␣or␣lost!";

  helpDetailsValues[1] =
    "MY␣TICKETS␣feature␣allows␣saving␣and␣visualizing␣"           +
    "your␣purchased␣tickets.␣You␣can␣see␣all␣ticket␣details␣or␣"  +
    "cancel␣tickets.␣By␣canceling␣any␣of␣the␣tickets␣the␣money␣you␣" +
    "spent␣on␣them␣are␣refunded␣to␣you␣as␣electronic␣money␣"      +
    "that␣you␣can␣use␣to␣buy␣new␣tickets␣or␣pay␣for␣different␣"   +
```

```
    "items␣purchased␣inside␣the␣cinema";

  helpDetailsValues[2] =
    "MY␣SETTINGS␣feature␣allows␣to␣tune␣the␣application␣by␣changing␣" +
    "the␣theme␣or␣setting␣different␣options.";

  helpDetailsValues[3] =
    "The␣FIND␣MOVIES␣feature␣allows␣you␣to␣search␣for␣movies␣in␣a␣given
        ␣" +
    "range␣from␣your␣current␣possition␣or␣any␣given␣position.␣"    +
    "A␣position␣is␣defined␣by␣a␣street␣name,␣city␣name␣or␣zip,␣"   +
    "and␣a␣range␣in␣km␣i.e.␣the␣radius␣of␣the␣area␣where␣you␣want␣" +
    "to␣search␣for␣a␣movie.␣A␣movie␣name␣can␣also␣be␣used.␣Otherwise,␣"
         +
    "all␣found␣movies␣matching␣your␣searching␣criteria␣are␣returned.␣"
        +
    "A␣date␣for␣the␣show␣has␣to␣be␣provided,␣also.␣Once␣you␣have␣" +
    "selected␣the␣seats,␣you␣can␣choose␣to␣pay␣for␣the␣tickets␣using" +
    "␣4␣payment␣methods␣i.e.␣AT␣THE␣CINEMA␣(you␣pay␣in␣cash␣or␣by␣
        credit" +
    "card␣once␣arived␣at␣the␣cinema␣by␣showing␣the␣tickets␣saved␣in␣" +
    "your␣phone);␣CREDIT␣CARD(you␣have␣to␣enter␣the␣data␣for␣the␣credit
        ␣" +
    "card␣you␣want␣to␣use␣for␣payment.␣The␣payment␣is␣done␣in␣a␣very␣"
        +
    "secure␣way.␣No␣credit␣card␣data␣is␣stored!);␣SECURE␣WALLET␣(using
        ␣" +
    "any␣of␣the␣previous␣credit␣cards␣saved␣in␣your␣secure␣wallet);" +
    "␣or␣E-MONEY␣(the␣electronic␣money␣refunded␣when␣cancelling␣a␣
        ticket.)";

  helpDetailsValues[4] =
    "The␣LOG␣IN␣screen␣performs␣the␣authentication␣with␣a␣web␣service
        .␣" +
    "This␣ensure␣that␣all␣your␣private␣data␣is␣safe!";

  helpDetailsValues[5] =
    "All␣credit␣card␣data␣is␣kept␣in␣the␣phone␣memory␣strongly␣"   +
    "encrypted.␣Nobody␣except␣you␣can␣access␣the␣credit␣card␣
        information␣"+
    "using␣your␣selected␣private␣keyword␣and␣PIN␣code!";

  helpDetailValue = helpDetailsValues[0];

} //end initModel()

/**
```

```
 * Creates the Ticket Discount Screen
 *
 * @throws Exception
 *
 */
protected void createView() throws Exception {
  backCommand  = new Command("MAIN␣MENU", Command.OK, 0);
  exitCommand  = new Command("EXIT",    Command.EXIT, 1);

  helpTopicsUI = new ChoiceGroup("Help␣Topics:", Choice.POPUP,
      helpTopicsValues, null);
  helpDetailsUI = new StringItem("Help␣Topic␣Details:␣",
      helpDetailValue, Item.PLAIN);

  try{
    imgUp = Image.createImage("/" + Start.themeDir + "/main_help/
        MainHelpTheme.png");
    imgThemeUp = new ImageItem("", imgUp, Item.LAYOUT_TOP | Item.
        LAYOUT_CENTER , "Theme_Img_Up");

  }catch(IOException ioe){
    System.out.println("Ticket␣Discount␣image␣exception!");
  }

  screen = new Form("Help");
  ((Form)screen).append(imgUp);
  ((Form)screen).append(helpTopicsUI);
  ((Form)screen).append(helpDetailsUI);

  // add the commands to the form
  screen.addCommand(backCommand);
  screen.addCommand(exitCommand);

  // add item listener
  ((Form)screen).setItemStateListener(this);
}

/**
 * Update the view - maybe refresh the fields
 *
 */
protected void updateView() throws Exception {
  initModel();
  createView();
}

/**
```

```
   * Trigered when the state of any of the UI elements changes
   * Actions can be taken based on the selected UI item e.g. update
       another
   * UI component when selecting another one.
   */
  public void itemStateChanged(Item item){
    int selectedTopic = 0;

    if (item.getLabel().equals("Help␣Topics:")){
      boolean[] ticketSelected = new boolean[helpTopicsUI.size()];
      helpTopicsUI.getSelectedFlags(ticketSelected);
      selectedTopic = helpTopicsUI.getSelectedIndex();
      helpDetailValue = UpdateMainHelpScreen.setAndUpdateHelpTopicDetails
          (helpDetailsUI, selectedTopic, helpDetailsValues);

    }// end if (item.getLabel().equals("Help Topics:"))

  }// end itemStateChanged()

}// end class

package gui.help;

import javax.microedition.lcdui.StringItem;

/**
 * Updates the Main Help UI components based on the selected element
 * in the Help topics choice group
 *
 * @author Mihai Balan (s031288)
 *
 */
public class UpdateMainHelpScreen {

  /**
   * Set the help topic details function of the selected help topic
   *
   * @param movie Movie name
   * @param showDate Show date
   * @param showHour Show hour
   * @param reservedSeats Reserved seats
   * @param selectedTicket Selected ticket in the choice group
   * @param selectedDiscount Selected discount in the choice group
   * @param ticketsDiscountValue All ticket discount values
   * @param discountValue Standard discoutn values
   * @param ticketInfoItems The ticket details UI components
   * @param ticketInfo The values used to populate the ticketInfoItems
   *
```

```
   */
  public static String setAndUpdateHelpTopicDetails(
      StringItem helpDetailsUI,
      int selectedTopic,
      String[] helpDetailsValues){

    String helpDetailValue = helpDetailsValues[selectedTopic];
    helpDetailsUI.setText(helpDetailValue);

    return helpDetailValue;

  }// end setAndUpdateTicketDiscount()


}// end class

package gui.mainmenu;

import javax.microedition.lcdui.Font;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.game.Sprite;
import javax.microedition.lcdui.Image;

import start.Start;

/**
 * Performs different helping operations for the main menu
 * such as: drawing the background, menu items, etc
 *
 * @author Mihai Balan (s031288)
 *
 */
public class MenuGenerationHelper{

  /**
   * Get the position of the longest entry in the menu
   *
   * @param mainMenuItems The menu items
   * @return pos The position of the longest menu entry in the menu
   */
  public static int getPositionLongestMenuEntry(String[] mainMenuItems){

    int max = 0;
    int pos = 0;

    for(int i = 0; i < mainMenuItems.length; i++){
      if(max < mainMenuItems[i].length()){
        max = mainMenuItems[i].length();
```

```
      pos = i;
    }
  }
  return pos;

} // end getPositionLongestMenuEntry()

/**
 * Generates and draws the main menu background
 *
 * @param g       The graphical object to paint on
 * @param backImage The background image to be used
 * @param width   The width of the canvas
 * @param height  The height of the canvas
 */
public static void drawBackground(Graphics g, Image backImage, int
    width, int height){

  // erase the canvas backgound
  g.setColor(0x00FFFFFF);
  g.fillRect(0, 0, width, height);

  // create and draw the background
  g.drawImage(
      backImage,
      0, 0,
      Graphics.TOP | Graphics.LEFT);

  g.drawRegion(
      backImage,
      0, 0,
      backImage.getWidth(), backImage.getHeight(),
      Sprite.TRANS_ROT180,
      width, height,
      Graphics.BOTTOM | Graphics.RIGHT);

}// end drawBackground()


/**
 * Draws the selected menu item on the screen by using a new font,
 * red color, a bigger image and an orange background for
 * the whole item
 *
 * @param g               The graphical object to draw on
 * @param mainMenuItems   The menu Entries
 * @param selectedImages  The source of selected images for the menu
```

```
 * @param posOfLongestEntry The position of the longest entry in the
     menu
 * @param selectedItemColor The color used for drawing the selected
     item
 * @param selectedItemFont The font used for drawing the selected item
 * @param canvasWidth     The canvas width
 * @param menuStartHeight The position where the menu starts to be
     drawn
 * @param spacing         The spacing between rows
 * @param itemPositionInMenu The current menu item
 */
public static void drawSelectedMenuItem(Graphics g, String[]
    mainMenuItems, Image[] selectedImages, int posOfLongestEntry, int
    selectedItemColor, Font selectedItemFont, int canvasWidth, int
    menuStartHeight, int spacing, int itemPositionInMenu){

  g.setFont (selectedItemFont);
  if(Start.themeName.equals("red")){
    g.setColor(255, 215, 0);

  }else if(Start.themeName.equals("blue")){
    g.setColor(152, 220, 255);

  }

  // draw the rectangle around the text and image
  g.fillRoundRect(
      (canvasWidth - selectedItemFont.stringWidth(mainMenuItems[
          posOfLongestEntry]) - 50)/2,
      menuStartHeight + (itemPositionInMenu * selectedItemFont.
          getHeight()) + itemPositionInMenu*spacing - 4,
      selectedItemFont.stringWidth(mainMenuItems[posOfLongestEntry])
           + 50,
      3*selectedItemFont.getHeight()/2,
      3, 3);

  // draw the image
  g.drawImage(selectedImages[itemPositionInMenu],
      3*canvasWidth/5 + 10,
      menuStartHeight + (itemPositionInMenu * selectedItemFont.
          getHeight()) + itemPositionInMenu*spacing -4,
      Graphics.LEFT | Graphics.TOP);

  // set a new color for the font and draw the highlighted menu entry
  if(Start.themeName.equals("red")){
    g.setColor(selectedItemColor);
```

```
  }else if(Start.themeName.equals("blue")){
    g.setColor(143, 179, 251);
    g.setColor(0, 50, 130);

  }

  g.drawString(
      mainMenuItems[itemPositionInMenu],
      (3*canvasWidth/5 - selectedItemFont.stringWidth(mainMenuItems[
          itemPositionInMenu])),
      menuStartHeight + (itemPositionInMenu * selectedItemFont.
          getHeight()) + itemPositionInMenu*spacing,
      Graphics.TOP | Graphics.LEFT);

} // end drawSelectedMenuItem()


/**
 * Draws the deselected menu item on the screen by using a new font,
 * black color, a small image and no background for the whole item
 *
 * @param g                The graphical object to draw on
 * @param mainMenuItems    The menu Entries
 * @param deselectedImages   The source of selected images for the menu
 * @param posOfLongestEntry The position of the longest entry in the
     menu
 * @param deselectedItemColor The color used for drawing the deselected
      item
 * @param deselectedItemFont The font used for drawing the deselected
     item
 * @param selectedItemFont The font used for drawing the selected item
 * @param canvasWidth      The canvas width
 * @param menuStartHeight  The position where the menu starts to be
     drawn
 * @param spacing          The spacing between rows
 * @param itemPositionInMenu The current menu item
 */
public static void drawDeselectedMenuItem(Graphics g, String[]
    mainMenuItems, Image[] deselectedImages, int posOfLongestEntry, int
     deselectedItemColor, Font deselectedItemFont, Font
    selectedItemFont, int canvasWidth, int menuStartHeight, int spacing
    , int itemPositionInMenu){
  g.setFont (deselectedItemFont);
  g.setColor(deselectedItemColor);

  // draw the image
  g.drawImage(deselectedImages[itemPositionInMenu],
```

```
        3*canvasWidth/5 + 10,
        menuStartHeight + (itemPositionInMenu * selectedItemFont.
            getHeight()) + itemPositionInMenu*spacing - 4, // -4
        Graphics.LEFT | Graphics.TOP);

    // draw the not highlighted menu entries
    g.drawString(
        mainMenuItems[itemPositionInMenu],
        (3*canvasWidth/5 - deselectedItemFont.stringWidth(mainMenuItems[
            itemPositionInMenu])),
        menuStartHeight + (itemPositionInMenu * selectedItemFont.
            getHeight()) + itemPositionInMenu*spacing,
        Graphics.TOP | Graphics.LEFT);

  } // end drawDeselectedMenuItem()

}// end class

package gui.mainmenu;

import java.io.*;
import java.util.Vector;

import javax.microedition.lcdui.Canvas;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Font;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.Image;

import constants.CustomAlertTypes;

import gui.customdialogwindows.CanvasAlert;
import gui.customdialogwindows.DialogWindow;
import gui.help.MainHelpGUI;
import gui.mytickets.MyTicketsMainMenu;
import gui.mywallet.MyWalletAuthenticationGUI;
import gui.purchasetickets.step1moviesearch.SearchMoviesGUI;
import gui.settings.SettingsGUI;

import rms.RMSOperations;
import start.Start;


/**
 * Generates the main menu of the application as a Canvas
 * and draws the menu entries using strings and images.
 *
 * @author s031288, Mihai Balan
```

```
 *
 */
public class MenuScreen extends Canvas {

  // the starting point of the application
  public static Start startingpoint;
  public static Display display;

  // set the fonts used to displaye the menu
  private Font deselectedItemFont = Font.getFont(Font.FACE_SYSTEM,
      Font.STYLE_PLAIN, Font.SIZE_LARGE);

  private Font selectedItemFont = Font.getFont(Font.FACE_SYSTEM,
      Font.STYLE_BOLD, Font.SIZE_LARGE);

  // the image used to set up the background
  Image imgUp = null;

  // set the color for the selected/deselected entries in the menu
  private int deselectedItemColor = 0x00000000;
  private int selectedItemColor = 0x00FF0000;

  // screen width and height
  static int width;
  static int height;

  // the height where the menu starts
  private int menuStartHeight;

  // the spacing between menu items
  private int spacing = selectedItemFont.getHeight()/2;

  // the menu items
  private String[] mainMenuItems = {
      "Find␣Movies",
      "My␣Settings",
      "My␣Tickets",
      "My␣Wallet",
      "About",
      "Help",
      "Exit",
      };

  // the menu images when the entry is not highlighted
  private String[] mainMenuImagesDeselected = {
      "searchIconMoviesDeselected.png",
      "settingsIconDeselected.png",
```

```
    "ticketsIconDeselected.png",
    "walletIconDeselected.png",
    "aboutIconDeselected.png",
    "helpIconDeselected.png",
    "exitIconDeselected.png",
    };

// the menu images when any entry is highlighted
private String[] mainMenuImagesSelected = {
    "searchIconMoviesSelected.png",
    "settingsIconSelected.png",
    "ticketsIconSelected.png",
    "walletIconSelected.png",
    "aboutIconSelected.png",
    "helpIconSelected.png",
    "exitIconSelected.png"
    };

private Image[] selectedImages = new Image[mainMenuImagesSelected.
    length];
private Image[] deselectedImages = new Image[mainMenuImagesSelected.
    length];

// the selected menu item (the menu starts from the first item)
private int selectedMenuItem = 0;
private int pos = 0;


/**
 * The constructor
 */
public MenuScreen() {

  try {
    imgUp = Image.createImage("/" + Start.themeDir + "/main_menu/
        menu_up.png");
    for(int i =0; i < mainMenuImagesSelected.length; i ++){
      selectedImages [i] = Image.createImage("/" + Start.themeDir + "/
          main_menu/" + mainMenuImagesSelected[i]);
      deselectedImages[i] = Image.createImage("/" + Start.themeDir + "/
          main_menu/" + mainMenuImagesDeselected[i]);
    }

  } catch (IOException ioe) {
    System.out.println("Exception␣in␣creating␣the␣background␣image␣and␣
        the␣main␣menu␣ones!");
  }
```

```
  // get the width and height of the canvas
  width = getWidth();
  height = getHeight();

  // Calculate the Start Height of Menu
  menuStartHeight = (height - ((mainMenuItems.length -1) * (
      selectedItemFont.getHeight() + spacing )))/2 - 10;

  // get the length of the longest entry in the main menu
  pos = MenuGenerationHelper.getPositionLongestMenuEntry(mainMenuItems)
      ;



}// end MenuScreen()



/**
 * Paint/repaint the menu items on the canvas
 *
 * @param Graphics g - The graphical object to draw on
 */
public void paint(Graphics g) {

  // draw the main menu backgound
  MenuGenerationHelper.drawBackground(g, imgUp, width, height);

  // generate and print the main menu
  for (int i = 0; i < mainMenuItems.length; i++) {

    // check if the current entry is selected
    // and draw the menu item highlighted
    if (i == selectedMenuItem) {

      MenuGenerationHelper.drawSelectedMenuItem(
          g,
          mainMenuItems,
          selectedImages,
          pos,
          selectedItemColor,
          selectedItemFont,
          width,
          menuStartHeight,
          spacing,
          i);
```

```
    } else {
      // if the meu entry is not selected
      // draw it as not highlighted
      MenuGenerationHelper.drawDeselectedMenuItem(
          g,
          mainMenuItems,
          deselectedImages,
          pos,
          deselectedItemColor,
          deselectedItemFont,
          selectedItemFont,
          width,
          menuStartHeight,
          spacing,
          i);

    }// end (if)

  }// end for()

  if(Start.needMovieAlert){
    Start.needMovieAlert = false;
    CanvasAlert invalidUI = new CanvasAlert(
      display,
      new MenuScreen(),
      "Movie today!",
      "You have tickets for a movie today!",
      "error",
      CustomAlertTypes.ALERT_WARNING);
  }

}// end paint()



/**
 * Capture the pressed key when navigating
 * through the menu or selecting any menu item
 * It creates a menu where you can scroll among
 * the menu entries in a continuous circle
 *
 * @param keyCode The key pressed
 */
protected void keyPressed(int keyCode) {

  if ((getGameAction(keyCode) == Canvas.UP)){
```

```
    if(selectedMenuItem - 1 >= 0){

      selectedMenuItem--;
      repaint();

    } else if(selectedMenuItem == 0){

      selectedMenuItem = mainMenuItems.length-1;
      repaint();
    }
  }

  if ((getGameAction(keyCode) == Canvas.DOWN)){
    if(selectedMenuItem + 1 < mainMenuItems.length){

      selectedMenuItem++;
      repaint();

    } else if(selectedMenuItem == mainMenuItems.length-1){

      selectedMenuItem = 0;
      repaint();
    }
  }

  if (getGameAction(keyCode) == Canvas.FIRE){


    // display the selected screen
    if(mainMenuItems[selectedMenuItem].equals("Find␣Movies") ){

      new SearchMoviesGUI().showScreen();

    } else if(mainMenuItems[selectedMenuItem].equals("My␣Settings") ){

      try{
        Vector v= RMSOperations.displayRecStore();
        for(int i=0; i<v.size(); i++)
          System.out.println(v.elementAt(i));

      }catch(Exception e){
        e.printStackTrace();
      }

      new SettingsGUI().showScreen();

    } else if(mainMenuItems[selectedMenuItem].equals("My␣Tickets") ){
```

```
        (new MyTicketsMainMenu()).showScreen();

      } else if(mainMenuItems[selectedMenuItem].equals("My␣Wallet") ){

        (new MyWalletAuthenticationGUI()).showScreen();

      } else if(mainMenuItems[selectedMenuItem].equals("About") ){

        CanvasAlert invalidUI = new CanvasAlert(
            display,
            new MenuScreen(),
            "Master␣Thesis␣Project!",
            "This␣is␣the␣MASTER␣THESIS␣GRADUATION␣PROJECT␣of␣MIHAI␣BALAN(
                s031288)␣graduating␣the␣COMPUTER␣SCIENCE␣MASTER␣PROGRAM␣-␣
                KNOLEDGE␣AND␣REQUIREMENTS␣ENGINEERING␣at␣DENMARK␣TECHNICAL
                ␣UNIVERISTY␣-␣IMM␣department,␣in␣August␣2007!",
            "info",
            CustomAlertTypes.ALERT_INFO);

      } else if(mainMenuItems[selectedMenuItem].equals("Help") ){

        (new MainHelpGUI()).showScreen();

      } else if(mainMenuItems[selectedMenuItem].equals("Exit") ){

        DialogWindow reallyExit = new DialogWindow(
            display,
            new MenuScreen(),
            "Exit␣Application?",
            "Are␣you␣sure␣that␣you␣want␣to␣exit␣the␣application?",
            "question",
            "/dialogIcons/exitTheme",
            startingpoint);

        display.setCurrent(reallyExit);

      } // end if(EXIT)

    } //end if(FIRE)

  }// end keyPressed()

} // end class

package gui.moviedetails;

import java.io.IOException;
```

```
import javax.microedition.lcdui.Canvas;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Font;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.ImageItem;
import javax.microedition.lcdui.Item;
import javax.microedition.lcdui.game.Sprite;

import model.beans.responsebeans.Movie_Details_Resp_Bean;

import gui.GUIHelper;
import start.Start;

/**
 *
 * Constructs and displays the movie description UI
 *
 * @author Mihai Balan (s031288)
 *
 */
public class ViewMovieDescriptionGUI extends Canvas {

  // the display to draw on
  private Display   display;

  private Displayable next;

  private Graphics g;

  private Image   backImg;
  private ImageItem backImgItem;

  private Movie_Details_Resp_Bean movRespBean;

  private int showLocationID;

  private Start midlet;

  //  define the msg font
  private Font msgFontBoldMedium = Font.getFont(
      Font.FACE_PROPORTIONAL,
      Font.STYLE_BOLD,
      Font.SIZE_MEDIUM);
```

```
private Font msgFontBoldItalianMedium = Font.getFont(
    Font.FACE_PROPORTIONAL,
    Font.STYLE_BOLD | Font.STYLE_ITALIC,
    Font.SIZE_MEDIUM);

private Font msgFontItalicBoldLarge = Font.getFont(
    Font.FACE_PROPORTIONAL,
    Font.STYLE_ITALIC | Font.STYLE_BOLD,
    Font.SIZE_LARGE);


// the images for building the YES and NO options
private Image[] deselectedImgs;
private Image[] selectedImgs;

private int textX = 15;
private int textY = 20;
private int spacing = 10;


public ViewMovieDescriptionGUI(Display display, Displayable next,
     Movie_Details_Resp_Bean movRespBean, int showLocationID){

  this.display      = display;
  this.next         = next;
  this.movRespBean  = movRespBean;
  this.showLocationID = showLocationID;

  display.setCurrent(this);

}


/**
 * The image and text displaying takes place in here
 *
 * @param g The graphics to draw on
 */
protected void paint(Graphics g){

  int w = getWidth();
  int h = getHeight();
  resetCoordinates();

  try {
    // clear the background
    g.setColor(255, 255, 255);
```

```
    g.fillRect(0, 0, getWidth(), getHeight());

    backImg    = Image.createImage("/theme_red/ticket/ticketBackground.
        png");
    backImgItem = new ImageItem("", backImg, Item.LAYOUT_TOP | Item.
        LAYOUT_RIGHT , "screen␣background");

    // draw the background
    g.drawRegion(
        backImg,
        0, 0,
        backImg.getWidth(), backImg.getHeight(),
        Sprite.TRANS_MIRROR,
        0, 0,
        Graphics.TOP | Graphics.LEFT);


    // write movie name
    g.setFont(msgFontItalicBoldLarge);
    g.setColor(160, 40, 18);
    textY = GUIHelper.dynamicDrawMessage(g, msgFontItalicBoldLarge
        , 160, 40, 18, movRespBean.getMovieName().toUpperCase(), w, h,
        textX, textY);


    g.setFont(msgFontBoldItalianMedium);
    g.setColor(160, 40, 18);
    textY = GUIHelper.dynamicDrawMessage(g, msgFontBoldItalianMedium
        , 160, 40, 18, "Movie␣Description:", w, h, textX, textY +
        spacing);

    // write movie description
    g.setFont(msgFontBoldMedium);
    g.setColor(160, 40, 18);
    textY = GUIHelper.dynamicDrawMessage(g, msgFontBoldItalianMedium
        , 160, 40, 18, movRespBean.getMovieDescription(), w - 3*textX
        /2, h, textX, textY + spacing/3);

    g.setFont(msgFontBoldMedium);
    g.setColor(0, 0, 255);
    g.drawString("Press␣any␣key␣to␣continue!", w/2, h-3*spacing,
        Graphics.TOP | Graphics.HCENTER);


  } catch(IOException ioe){
    System.out.println("Movie␣details␣image␣exception!");
    // if the image cannot be drawn, write some text
```

```
      g.drawString("Movie␣description␣IOException", w/2, h/2,
          Graphics.BASELINE | Graphics.HCENTER);

    } catch (Exception e) {
      System.out.println("Movie␣details␣exception!");
      g.drawString("Movie␣description␣Exception", w/2, h/2,
          Graphics.BASELINE | Graphics.HCENTER);
    }
  }

  /**
   * Update the value of the current selected option in the list
   * and repaint the screen.
   *
   * @param keyCode The code of the pressed key
   */
  protected void keyPressed( int keyCode ){
    if(midlet == null)
      display.setCurrent(next);

  }// end keyPressed

  private void resetCoordinates(){
    textX = 15;
    textY = 20;
  }

}// end class

package gui.moviedetails;

import java.io.IOException;

import javax.microedition.lcdui.Canvas;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Font;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.ImageItem;
import javax.microedition.lcdui.Item;
import javax.microedition.lcdui.game.Sprite;

import model.beans.responsebeans.Movie_Details_Resp_Bean;

import start.Start;
import gui.GUIHelper;
import gui.purchasetickets.step4discountandreservationsummary.
```

```
    UpdateTicketDiscountAndReservationSummaryScreen;

/**
 * Construct the Movie Details UI and displayes the movie
 * details in a nice graphical way. Everything is built dinamically
 * function of the screen size.
 *
 * @author Mihai Balan (s031288)
 *
 */
public class ViewMovieDetailsGUI extends Canvas {

  // the display to draw on
  private Display   display;
  private Displayable next;

  private Graphics g;

  private Image   backImg;
  private ImageItem backImgItem;
  private Image   movieImg;
  private ImageItem movieImgItem;

  private Movie_Details_Resp_Bean movRespBean;

  private int showLocationID;

  private Start midlet;

  // the curent selected option i.e. CANCEL, BACK
  private int selectedOptionIndex = 0;

  // NOT highlighted buttons
  private String[] optionDeselected = {
      "/MovieDetailsButtons/newMoreDeselected.png",
      "/MovieDetailsButtons/newBackDeselected.png"
  };

  // the highlighted buttons
  private String[] optionSelected = {
      "/MovieDetailsButtons/newMoreSelected.png",
      "/MovieDetailsButtons/newBackSelected.png"
  };

  //   define the msg font
  private Font msgFontBoldLarge = Font.getFont(
      Font.FACE_PROPORTIONAL,
```

```
    Font.STYLE_BOLD,
    Font.SIZE_LARGE);

private Font msgFontBoldMedium = Font.getFont(
    Font.FACE_PROPORTIONAL,
    Font.STYLE_BOLD,
    Font.SIZE_MEDIUM);

private Font msgFontItalicBoldLarge = Font.getFont(
    Font.FACE_PROPORTIONAL,
    Font.STYLE_ITALIC | Font.STYLE_BOLD,
    Font.SIZE_LARGE);

private Font msgFontBoldSmall = Font.getFont(
    Font.FACE_PROPORTIONAL,
    Font.STYLE_BOLD,
    Font.SIZE_SMALL);

// the images for building the YES and NO options
private Image[] deselectedImgs;
private Image[] selectedImgs;

private int textX = 15;
private int textY = 20;
private int spacing = 10;
private int w, h;


public ViewMovieDetailsGUI( Display display, Displayable next,
    Movie_Details_Resp_Bean movRespBean, int ShowLocationID){

  this.display      = display;
  this.next         = next;
  this.movRespBean  = movRespBean;
  this.showLocationID = showLocationID;

  selectedImgs  = GUIHelper.createSelectedButtons(optionSelected);
  deselectedImgs = GUIHelper.createDeselectedButtons(optionDeselected);
  w = getWidth();
  h = getHeight();
  display.setCurrent(this);
}


/**
 * The image and text displaying takes place in here
 *
```

```
 * @param g The graphics to draw on
 */
protected void paint(Graphics g){


  resetCoordinates();

  try {
    // clear the background
    g.setColor(255, 255, 255);
    g.fillRect(0, 0, getWidth(), getHeight());

    backImg    = Image.createImage("/theme_red/ticket/ticketBackground.
        png");
    backImgItem = new ImageItem("", backImg, Item.LAYOUT_TOP | Item.
        LAYOUT_RIGHT , "screen␣background");

    // create the image from the byte[] stored in the Movie Details
        Resp bean
    movieImg    = Image.createImage(movRespBean.getMoviePoster(), 0,
        movRespBean.getMoviePoster().length);
    movieImgItem = new ImageItem("", movieImg, Item.LAYOUT_TOP | Item.
        LAYOUT_RIGHT , "movie␣poster");

    // draw the background
    g.drawRegion(
        backImg,
        0, 0,
        backImg.getWidth(), backImg.getHeight(),
        Sprite.TRANS_MIRROR,
        0, 0,
        Graphics.TOP | Graphics.LEFT);

    // draw the movie poster
    g.drawImage(
        movieImg,
        w-21, textY + msgFontItalicBoldLarge.getHeight() + 2*spacing,
        Graphics.TOP | Graphics.RIGHT);

    // write movie name
    g.setFont(msgFontItalicBoldLarge);
    g.setColor(160, 40, 18);
    GUIHelper.dynamicDrawMessage(g, msgFontItalicBoldLarge
        , 160, 40, 18, movRespBean.getMovieName().toUpperCase(), w, h,
        textX, textY);

    // write year, language, duration
```

```
g.setFont(msgFontBoldSmall);
textY += msgFontItalicBoldLarge.getHeight();
textY = GUIHelper.dynamicDrawMessage(g, msgFontBoldSmall
    , 160, 40, 18, "(" + movRespBean.getMovieYear() + ",␣" +
    movRespBean.getMovieLanguage() + ",␣" + movRespBean.
    getMovieDuration() + "min)", w - textX, h, textX, textY);

// write the parrent classification
g.setColor(255, 255, 0);
g.fillArc(textX - 3, textY + 3, msgFontBoldLarge.getHeight() + 4,
    msgFontBoldLarge.getHeight() + 3, 0, 360);
g.setColor(255, 0, 0);
g.setFont(msgFontBoldLarge);
textY = GUIHelper.dynamicDrawMessage(g, msgFontBoldLarge
    , 160, 40, 18, movRespBean.getMovieParentClassification(), w -
    textX, h, textX, textY+4);



// write genre, country, user rating
g.setColor(0, 0, 0);
textY += spacing;
g.setFont(msgFontBoldMedium);
textY = GUIHelper.dynamicDrawMessage(g, msgFontBoldMedium
    , 160, 40, 18, "Genre:␣" + movRespBean.getMovieGenre(), w -
    textX - movieImg.getWidth(), h, textX, textY);
textY += spacing;
textY = GUIHelper.dynamicDrawMessage(g, msgFontBoldMedium
    , 160, 40, 18, "Country:␣" + movRespBean.getMovieCountry(), w
     - textX - movieImg.getWidth(), h, textX, textY);
g.setColor(255, 0, 0);
textY += spacing;
textY = GUIHelper.dynamicDrawMessage(g, msgFontBoldMedium
    , 160, 40, 18, "User␣Rating:␣" +
    UpdateTicketDiscountAndReservationSummaryScreen.
    formatTotalPrice(Double.parseDouble(movRespBean.
    getMovieUserRating())) + "/10", w - textX - movieImg.getWidth()
    , h, textX, textY);

// write actors and director
g.setColor(0, 0, 0);
textY = 10 + msgFontItalicBoldLarge.getHeight() + spacing +
    movieImg.getHeight();
textY = GUIHelper.dynamicDrawMessage(g, msgFontBoldMedium
    , 160, 40, 18, "Actors:␣" + movRespBean.getMovieActors().
    toUpperCase(), w - textX, h, textX, textY + 2 * spacing +5);
textY = GUIHelper.dynamicDrawMessage(g, msgFontBoldMedium
```

```
        , 160, 40, 18, "Director:␣" + movRespBean.getMovieDirector().
        toUpperCase(), w - textX, h, textX, textY + spacing/2);

    // draw the buttons
    GUIHelper.drawCCViewButtons(g,
        deselectedImgs, selectedImgs,
        w, h,
        w/2, textY - 10,
        selectedOptionIndex);

  } catch(IOException ioe){
    System.out.println("Movie␣details␣image␣exception!");
    // if the image cannot be drawn, write some text
    g.drawString("Movie␣details␣IOException", w/2, h/2,
        Graphics.BASELINE | Graphics.HCENTER);

  } catch (Exception e) {
    System.out.println("Movie␣details␣exception!");
    g.drawString("Movie␣details␣Exception", w/2, h/2,
        Graphics.BASELINE | Graphics.HCENTER);
  }
}

/**
 * Update the value of the current selected option in the list
 * and repaint the screen.
 *
 * @param keyCode The code of the pressed key
 */
protected void keyPressed( int keyCode ){

  if ((getGameAction(keyCode) == Canvas.LEFT)){

    if(selectedOptionIndex > 0){

      selectedOptionIndex--;
      resetCoordinates();
      repaint();

    } else if(selectedOptionIndex == 0){

      selectedOptionIndex = optionDeselected.length - 1;
      resetCoordinates();
      repaint();
    }

  } else if ((getGameAction(keyCode) == Canvas.RIGHT)){
```

```
    if(selectedOptionIndex < optionDeselected.length - 1){

      selectedOptionIndex++;
      resetCoordinates();
      repaint();

    } else if(selectedOptionIndex == optionDeselected.length - 1){

      selectedOptionIndex = 0;
      resetCoordinates();
      repaint();
    }

  } else if ((getGameAction(keyCode) == Canvas.FIRE)){

    // show movie description
    if(selectedOptionIndex == 0){
      display.setCurrent(new ViewMovieDescriptionGUI(display, display.
          getCurrent(), movRespBean, showLocationID));


      // go back to Select show screen
    }else if(selectedOptionIndex == 1){


       Runtime runtime = Runtime.getRuntime();
       long t = runtime.freeMemory();
       System.out.println("**********************Memery␣before:" + t)
           ;
       clean();
       long t1 = runtime.freeMemory();
       System.out.println("**********************Memery␣after:" + t1)
           ;


      if(midlet == null) {
        display.setCurrent(next);
      }
    }

  } // end if(FIRE)

}// end keyPressed


private void resetCoordinates(){
```

```
    textX = 15;
    textY = 20;
  }// end resetCoordinates()

  private void clean(){

    backImg = null;
    backImgItem = null;
    movieImg = null;
    movieImgItem = null;
    movRespBean = null;
    optionDeselected = null;
    optionSelected = null;
    msgFontBoldLarge = null;
    msgFontBoldMedium = null;
    msgFontItalicBoldLarge = null;
    msgFontBoldSmall = null;
    deselectedImgs = null;
    selectedImgs = null;

    g = null;
    System.gc();
  }


}// end class

package gui.mytickets;

import java.io.IOException;
import javax.microedition.lcdui.*;

import gui.GenericGUI;
import gui.customdialogwindows.CanvasAlert;
import gui.customdialogwindows.DialogWindow;
import gui.mainmenu.MenuScreen;
import start.Start;
import constants.CustomAlertTypes;


/**
 *  Displays My Ticket main menu and shows
 *  all Tickets(TKTs) saved in RMS
 *
 *  It extends the GenericGUI super class
 *
 * @author s031288, Mihai Balan
 *
```

```
 */
public class MyTicketsMainMenu extends GenericGUI{

  // the authentication screen
  private static Displayable screen = null;

  //the starting point of the application
  public static Start startingPoint;

  //  the exit and select commands
  private static Command viewCommand;
  private static Command cancelCommand;
  private static Command menuCommand;
  private static Command helpCommand;
  private static Command exitCommand;

  private Image imgUp;
  private ImageItem imgThemeUp;

  private String ccChoiceTitle = "Available␣Tickets:";

  // stores ticket IDs and images for the choice group
  private String[] tktIDs;
  private Image[] tktImages;


  // the choice group for displaying all credit cards
  private ChoiceGroup cgTickets;

  /**
   * Constructs an instance of the class
   */
  public MyTicketsMainMenu () {}


  /**
   * Returns the displayable authentication screen
   * @return screen My Tickets Maine Menu screen
   */
  public Displayable getScreen() {
    return screen;
  }

  /**
   * Function of the chosen menu option i.e. view, cancel, help, main
       menu, or exit
   * one of the functionalities with the same name is executed i.e.
```

```
 * view ticket details, cancel a ticket, display help,
 * go back to appl. main menu, and exit the appl, respectively.
 *
 * @param c The executed command
 * @param s The main menu form
 *
 */
public void commandAction(Command c, Displayable s) {
  try {

    if (c == viewCommand){

      if(cgTickets.size() > 0){
        boolean[] cgSelected = new boolean[cgTickets.size()];
        cgTickets.getSelectedFlags(cgSelected);
        int selectedTKTs = cgTickets.getSelectedIndex();

        new MyTicketViewTKT(display, getScreen(), Start.tickets[
            selectedTKTs], cgTickets);

      } else if(cgTickets.size() == 0){

        CanvasAlert help = new CanvasAlert(
            display,
            new MyTicketsMainMenu().prepareScreen(),
            "No␣Tickets␣Available!",
            "There␣are␣no␣tickets␣available␣to␣display!",
            "question",
            CustomAlertTypes.ALERT_INFO);

      }// end if(cgTickets.size() == 0)

    }// end if (c == viewCommand)

    if (c == cancelCommand){

      if(cgTickets.size() > 0){
        DialogWindow reallyDelete = new DialogWindow(
            display,
            getScreen(),
            new MyTicketsMainMenu().prepareScreen(),
            "Cancel␣Ticket?",
            "Are␣you␣sure␣that␣you␣want␣to␣cancel␣this␣ticket␣
                permanently?",
            "question",
            "/theme_red/ticket/cancelTicketTheme",
            new MenuScreen().startingpoint,
```

```
        cgTickets,
        Start.tickets
    );

    display.setCurrent(reallyDelete);

  }else if(cgTickets.size() == 0){

    CanvasAlert help = new CanvasAlert(
        display,
        getScreen(),
        "No␣Tickets␣Available!",
        "There␣are␣no␣tickets␣available␣to␣cancel!",
        "question",
        CustomAlertTypes.ALERT_INFO);

  }// end if(cgTickets.size() == 0)

}// end if (c == cancelCommand)

// go back to main menu
if (c == menuCommand) {
  display.setCurrent(new MenuScreen());

}// end if (c == menuCommand)

if (c == helpCommand){
  CanvasAlert help = new CanvasAlert(
      display,
      new MyTicketsMainMenu().prepareScreen(),
      "My␣Tickets␣Help",
      "My␣Tickets␣displays␣all␣tickets␣available␣for␣different␣shows
          .␣You␣can␣cancel␣any␣payed␣ticket␣at␣any␣time.␣The␣money␣
          is␣refunded␣via␣electronic␣money.␣You␣can␣use␣this␣money␣
          to␣buy␣tickets␣or␣different␣products␣in␣the␣cinema.",
      "question",
      CustomAlertTypes.ALERT_INFO);

} // end if (c == helpCommand)

if (c == exitCommand){
  DialogWindow reallyExit = new DialogWindow(
      display,
      new MyTicketsMainMenu().prepareScreen(),
      "Exit␣Application?",
      "Are␣you␣sure␣that␣you␣want␣to␣exit␣the␣application?",
      "question",
```

```
          "/dialogIcons/exitTheme",
          new MenuScreen().startingpoint);

      display.setCurrent(reallyExit);

    } // end if (c == exitCommand)

  }catch (Exception e) {
    e.printStackTrace();
    CanvasAlert keyErrorAlert = new CanvasAlert(
        display,
        getScreen(),
        "My␣Tickets␣Error!",
        "Error␣while␣displaying␣all␣Tickets!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    display.setCurrent(keyErrorAlert);
  }
}

/**
 * Initialize the model an open the record store
 */
protected void initModel() {
  try{

    // if there are any tickets in RMS
    // construct the choice group and display it
    tktIDs   = new String[Start.maxTTSaved];
    tktImages = new Image [Start.maxTTSaved];

    if (Start.maxTTSaved > 0){
      String imgName = "";
      for (int i = 0; i < Start.maxTTSaved; i++){
        tktIDs[i] = Start.tickets[i].getTKTMovie() + "␣(" + Start.
            tickets[i].getTKTCinema() + "␣-␣" + Start.tickets[i].
            getTKTShowDate() +"␣-␣" + Start.tickets[i].getTKTShowHour()
             + "),␣Row:␣" + Start.tickets[i].getTKTRow() + ",␣Seat:␣"
             + Start.tickets[i].getTKTSeat();
        imgName = "/Tickets/cgTicketImg.png";
        tktImages[i] = Image.createImage(imgName);
      } // end for

    } else{
      ccChoiceTitle = "There␣are␣no␣tickets␣available!";
    }
```

```
  }catch(Exception e){
    System.out.println("Exception␣in␣My␣Tickets␣Menu␣INIT␣method!");
    e.printStackTrace();
  }

} // end initModel()

/**
 * Creates the Authentication Screen
 *
 * @throws Exception
 *
 */
protected void createView() throws Exception {

  viewCommand  = new Command("VIEW",       Command.EXIT, 0);
  cancelCommand = new Command("CANCEL␣TICKET", Command.SCREEN, 3);
  menuCommand  = new Command("MAIN␣MENU",  Command.SCREEN, 2);
  helpCommand  = new Command("HELP",       Command.SCREEN, 4);
  exitCommand  = new Command("EXIT",       Command.SCREEN, 5);

  cgTickets = new ChoiceGroup(ccChoiceTitle, Choice.EXCLUSIVE, tktIDs,
      tktImages);

  try{
    imgUp = Image.createImage("/" + Start.themeDir + "/ticket/tktTheme.
        png");
    imgThemeUp = new ImageItem("", imgUp, Item.LAYOUT_TOP | Item.
        LAYOUT_CENTER , "Theme_Img_Up");

  }catch(IOException ioe){
    System.out.println("My␣Tickets␣image␣exception!");
  }


  screen = new Form("My␣Wallet");
  ((Form)screen).append(imgUp);
  ((Form)screen).append(cgTickets);

  // add the commands to the form
  screen.addCommand(viewCommand);
  screen.addCommand(menuCommand);
  screen.addCommand(cancelCommand);
  screen.addCommand(helpCommand);
  screen.addCommand(exitCommand);

}// end createView()
```

```
  /*
   * Update the view - maybe refresh the fields
   *
   */
  protected void updateView() throws Exception {
    initModel();
    createView();
  }

}// end class

package gui.mytickets;

import gui.customdialogwindows.CanvasAlert;

import java.io.IOException;

import javax.microedition.lcdui.ChoiceGroup;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.ImageItem;
import javax.microedition.lcdui.Item;

import model.beans.otherbeans.TicketBean;


import rms.RMSOperations;
import constants.CustomAlertTypes;
import constants.SystemConstants;
import start.Start;

/**
 * Helper class that performs different operations
 * for My Tickets feature
 * i.e. removing a selected ticket from RMS
 *
 * @author s031288, Mihai Balan
 *
 */
public class MyTicketTools {


  /**
   * Draw the bar code image coresponding to the ticket ID
   *
```

```
   * @param g      The graphical object to draw on
   * @param ticketID The ticket ID
   * @param startX The x coordinate to place the image
   * @param startY The y coordinate to place the image
   * @throws IOException
   */
  public static void drawBarCode(
      Graphics g,
      String ticketID,
      int startX,
      int startY) throws IOException{

    Image barCodeImg       = Image.createImage("/barCodeExample.png");
    ImageItem barCodeImgItem = new ImageItem("", barCodeImg, Item.
        LAYOUT_TOP | Item.LAYOUT_CENTER , "Theme_Img_Up");

    g.drawImage(barCodeImg, startX , startY, Graphics.TOP | Graphics.
        HCENTER);

  } // end drawBarCode()


} // end class

package gui.mytickets;

import javax.microedition.lcdui.*;

import model.beans.otherbeans.TicketBean;


import gui.GUIHelper;
import gui.customdialogwindows.DialogWindow;
import gui.mainmenu.MenuScreen;
import constants.SystemConstants;
import start.Start;

/**
 * Display the selected ticket data in
 * a very user friendly way i.e. the same format as
 * a regular cinema ticket
 *
 * @author s031288, Mihai Balan
 */
public class MyTicketViewTKT extends Canvas {

  // the display to draw on
  private Display   display;
```

```
private Displayable next;

private Graphics g;

// properties of the alert
private String title;

private Image   ticketImg;
private ImageItem ticketImgItem;

private TicketBean tktBean;

private Start midlet;

// the curent selected option i.e. CANCEL, BACK
private int selectedOptionIndex = 1;

 // NOT highlighted buttons
private String[] optionDeselected = {
    "/TicketViewButtons/cancelDeselected.png",
    "/TicketViewButtons/backDeselected.png"
};

// the highlighted buttons
private String[] optionSelected = {
    "/TicketViewButtons/cancelSelected.png",
    "/TicketViewButtons/backSelected.png"
};

private ChoiceGroup cgTickets;

// the images for building the YES and NO options
private Image[] deselectedImgs;
private Image[] selectedImgs;

private int selectedTicket;
private int textX = 25;
private int textY = 10;
private int spacing = 10;

/**
 * Constructor for the ticket view screen
 *
 * @param display The display to draw on
 * @param next  The next screen to be displayed after the CC data
 * @param tktBean The ticket bean containing all data about the
```

```
     selected TKT
 */
public MyTicketViewTKT( Display display, Displayable next, TicketBean
    tktBean, ChoiceGroup cgTickets){

  this.display  = display;
  this.next     = next;
  this.tktBean  = tktBean;
  this.cgTickets = cgTickets;

  selectedImgs  = GUIHelper.createSelectedButtons(optionSelected);
  deselectedImgs = GUIHelper.createDeselectedButtons(optionDeselected);

  // get the selected ticket ID
  boolean[] cgSelected = new boolean[cgTickets.size()];
  cgTickets.getSelectedFlags(cgSelected);
  selectedTicket = cgTickets.getSelectedIndex();

  display.setCurrent(this);
}


/**
 * The image and text displaying takes place in here
 *
 * @param g The graphocs to draw on
 */
protected void paint( Graphics g ){

  int w = getWidth();
  int h = getHeight();

  try {
    // clear the background
    g.setColor(255, 255, 255);
    g.fillRect(0, 0, getWidth(), getHeight());

    ticketImg    = Image.createImage("/theme_red/ticket/ticketBackground
        .png");
    ticketImgItem = new ImageItem("", ticketImg, Item.LAYOUT_TOP | Item
        .LAYOUT_RIGHT , "Theme_Img_Up");

    // draw the theme image
    g.drawImage(ticketImg, 0 , 0,
       Graphics.TOP | Graphics.LEFT);

    // define the msg font
```

```
Font msgFontBoldLarge = Font.getFont(
    Font.FACE_PROPORTIONAL,
    Font.STYLE_BOLD,
    Font.SIZE_LARGE);

Font msgFontBoldMedium = Font.getFont(
    Font.FACE_PROPORTIONAL,
    Font.STYLE_BOLD,
    Font.SIZE_MEDIUM);

Font msgFontBoldSmall = Font.getFont(
    Font.FACE_PROPORTIONAL,
    Font.STYLE_BOLD,
    Font.SIZE_SMALL);

Font msgFontPlainSmall = Font.getFont(
    Font.FACE_PROPORTIONAL,
    Font.STYLE_PLAIN,
    Font.SIZE_SMALL);

// write ticketID
//g.setColor(0, 50, 125);
g.setColor(255, 255, 255);
g.setFont(msgFontBoldSmall);
g.drawString(tktBean.getTKTID().toUpperCase(), w - 16, textY + 6 ,
    Graphics.TOP | Graphics.RIGHT);

g.setColor(160, 40, 18);
// bar code image generation
MyTicketTools.drawBarCode(g, tktBean.getTKTID().toUpperCase(), w
    /2, textY + spacing + msgFontBoldLarge.getHeight());

// draw Cinema name and address
g.setFont(msgFontBoldMedium);
// TO DO CHANGE this back to the below
//int imgHeight = ticketImg.getHeight();
int imgHeight = 70;
textY += imgHeight + spacing;
g.drawString(tktBean.getTKTCinema().toUpperCase(), textX, textY,
    Graphics.TOP | Graphics.LEFT);
g.setFont(msgFontBoldSmall);
textY += msgFontBoldMedium.getHeight();
g.drawString(tktBean.getTKTCinemaTheater().toUpperCase(), textX,
    textY, Graphics.TOP | Graphics.LEFT);

// tokenize the cinema address in order to fint into the screen
    width
```

```
    textY += msgFontBoldMedium.getHeight();
    GUIHelper.dynamicDrawMessage(g, msgFontPlainSmall, 160, 40, 18,
        tktBean.getTKTCinemaAddress(), w - 35, h, textX, textY);

    // draw Movie name, date, hour, row, seat
    textY = h/2 + 2*spacing;
    g.setFont(msgFontBoldLarge);
    g.drawString(tktBean.getTKTMovie(), w/2, textY, Graphics.BASELINE|
        Graphics.HCENTER);

    textY += 5*msgFontBoldLarge.getHeight()/6;
    g.setFont(msgFontBoldMedium);
    g.drawString("Date:␣" + tktBean.getTKTShowDate() + ",␣Hour:␣" +
        tktBean.getTKTShowHour(), w/2, textY, Graphics.BASELINE|
        Graphics.HCENTER);

    textY += 5*msgFontBoldLarge.getHeight()/6;
    g.setFont(msgFontBoldMedium);
    g.drawString("Row:␣" + tktBean.getTKTRow() + ",␣Seat:␣" + tktBean.
        getTKTSeat(), w/2, textY, Graphics.BASELINE| Graphics.HCENTER);

    // draw discount, purchase method and price
    textY += msgFontBoldLarge.getHeight() + spacing;
    g.setFont(msgFontBoldSmall);
    g.drawString("Dicount␣Type:␣" + tktBean.getTKTDiscountType(), textX
        , textY, Graphics.TOP| Graphics.LEFT);

    textY += 5*msgFontBoldSmall.getHeight()/6;
    g.setFont(msgFontBoldSmall);
    g.drawString("Purchase␣Method:␣" + tktBean.getTKTPurchaseMethod(),
        textX, textY, Graphics.TOP| Graphics.LEFT);

    textY += 5*msgFontBoldSmall.getHeight()/6;
    g.setFont(msgFontBoldMedium);
    g.drawString("Price:␣" + tktBean.getTKTPrice() + "␣" +
        SystemConstants.NATIONAL_CURRENCY, textX, textY, Graphics.TOP|
        Graphics.LEFT);

    // draw the buttons
    GUIHelper.drawCCViewButtons(g,
       deselectedImgs, selectedImgs,
       w, h,
       w/2, textY -5,
       selectedOptionIndex);

  } catch (Exception e) {
    System.out.println("View␣CC␣exception!");
```

```
    g.drawString("Mobile␣Cinema␣Exception", w/2, h/2,
        Graphics.BASELINE | Graphics.HCENTER);
  }
}

/**
 * Update the value of the current selected option in the list
 * and repaint the screen.
 *
 * @param keyCode The code of the pressed key
 */
protected void keyPressed( int keyCode ){

  if ((getGameAction(keyCode) == Canvas.LEFT)){

    if(selectedOptionIndex > 0){

      selectedOptionIndex--;
      resetCoordinates();
      repaint();

    } else if(selectedOptionIndex == 0){

      selectedOptionIndex = optionDeselected.length - 1;
      resetCoordinates();
      repaint();
    }

  } else if ((getGameAction(keyCode) == Canvas.RIGHT)){

    if(selectedOptionIndex < optionDeselected.length - 1){

      selectedOptionIndex++;
      resetCoordinates();
      repaint();

    } else if(selectedOptionIndex == optionDeselected.length - 1){

      selectedOptionIndex = 0;
      resetCoordinates();
      repaint();
    }

  } else if ((getGameAction(keyCode) == Canvas.FIRE)){

      if(selectedOptionIndex == 0){
```

```
        try{
          // cancel the ticket
          DialogWindow reallyDelete = new DialogWindow(
              display,
              new MyTicketViewTKT(display, new MyTicketsMainMenu().
                  prepareScreen(), Start.tickets[selectedTicket],
                  cgTickets),
              new MyTicketsMainMenu().prepareScreen(),
              "Cancel␣Ticket?",
              "Are␣you␣sure␣that␣you␣want␣to␣cancel␣this␣ticket␣
                  permanently?",
              "question",
              "/theme_red/ticket/cancelTicketTheme",
              new MenuScreen().startingpoint,
              cgTickets,
              Start.tickets
              );

          display.setCurrent(reallyDelete);

        }catch(Exception e){
          System.out.println("Exception␣in␣My␣Ticket␣View␣TKT␣when␣
              trying␣to␣cancel␣the␣ticket");
          e.printStackTrace();
        }

      }else if(selectedOptionIndex == 1){
        if(midlet == null)
          display.setCurrent(next);
      }

    } // end if(FIRE)

  }// end keyPressed

  private void resetCoordinates(){
    textX = 25;
    textY = 10;
  }

}// end class

package gui.mywallet;

import gui.customdialogwindows.CanvasAlert;

import java.util.*;
import java.io.*;
```

```
import javax.microedition.lcdui.*;

import constants.CustomAlertTypes;

/**
 * This class is used to perform network operations on
 * a separate thread. Each class that performs network operations,
 * (e.g. Authenticate)extends this class.
 * An animated progress gauge is displayed while the
 * network operations are performed.
 *
 * @author Mihai Balan, Wojciech Dobrowolski
 *
 */
public abstract class BackgroundUpdate extends TimerTask {

  protected Display   display;
  protected Displayable nextScreen;
  protected Displayable prevScreen;
  protected String    title;

  protected boolean needAlert = false;
  private String alertTitle = "";
  private String alertMessage = "";

  private Thread workerThread;
  private boolean isWrkStopped;

  /**
   * Constructor - initialize the display and
   * creates the worker thread
   *
   * @param display The current display
   */
  public BackgroundUpdate (Display display) {
    this.display = display;
    workerThread = new Thread(this);
  }

  /**
   * Starts the worker thread for performing the network
   * operations in the background
   */
  public void go () {
    // set the flag to worker alive
    isWrkStopped = false;
    // start the worker thread
```

```
  System.out.println("-----Background␣Task␣--␣The␣worker␣thread␣for␣
      performing␣network" +
  "␣operations␣in␣backgruond␣has␣been␣started!");
  workerThread.start();
}

/**
 * Stop the worker thread by setting the pririty to MIN
 */
public void stop () {
  // set the flag to worker stopped
  isWrkStopped = true;
  System.out.println("-----Background␣Task␣--␣The␣worker␣thread␣
      priority␣set␣to␣MINIMUM");
  // stops the worker thread
  workerThread.setPriority(Thread.MIN_PRIORITY);
}

/**
 * Create the animated gauge and
 * call the template method runTask()
 * that is implemented by the derived classes.
 * In case of network communication exception
 * or other exception, the application catches
 * the exceptions and display an alert.
 */
public void run() {

  //ProgressGauge pg = null;

  try {
    // Construct and start the gauge
    // The gauge is started in the init() method of the ProgressGauge
    System.out.println("-----Background␣Task␣--␣Animated␣gauge␣created"
        );
    //pg = new ProgressGauge(this, title, display, prevScreen);
    // start the task implemented by the derived classes
    runTask ();
    System.out.println("-----Background␣Task␣--␣The␣runTask()␣method␣
        implemented␣" +
    "by␣the␣classes␣that␣extend␣Background␣Task␣is␣called");
  } catch (IOException ioe) {
    // an alert need to be displayed
    needAlert   = true;
    alertTitle  = "Communication␣Error!";
    alertMessage = "Please␣check␣your␣network␣or␣server␣setup!";
```

```
    nextScreen = prevScreen;
    System.out.println("-----Background␣Task␣--␣IO␣ERROR");
    System.out.println("-----Background␣Task␣--␣Background␣task␣IO␣
        Error");
    ioe.printStackTrace();
  } catch (Exception e) {

    // an alert need to be displayed
    needAlert   = true;
    alertTitle  = "Unknown␣Error!";
    alertMessage = "Please␣contact␣customer␣support!";

    // return to the previous screen in case of error
    nextScreen = prevScreen;
    System.out.println("-----Background␣Task␣--␣Background␣task␣Error")
        ;
    System.out.println("-----Background␣Task␣--␣ERROR");
    e.printStackTrace();

  } finally {
    // Since pg could callback and reset "stopped" when its
    // Cancel key is pressed, we'd better check.
    System.out.println("-----Background␣Task␣--␣before␣if(!STOPPED)");
    if (!isWrkStopped) {
      // in case an alert was displayed
      if ( needAlert ){
        System.out.println("-----Background␣Task␣--␣Progress␣Gauge␣
            stoped");
      //  pg.stop();

        // create the alert but do not display it
        // let the progress gauge to display it after it stoped
        CanvasAlert alert = new CanvasAlert(
            display,
            prevScreen,
            alertTitle,
            alertMessage,
            "error",
            CustomAlertTypes.ALERT_ERROR,
            false);
        display.setCurrent(alert);

      //  pg.setNextScreen(alert, nextScreen);

      } else {

        System.out.println("-----Background␣Task␣--␣Progress␣Gauge␣
```

```
        stoped");
  //  pg.stop();
  //  pg.setNextScreen(nextScreen);

  /*  CanvasAlert alert = new CanvasAlert(
        display,
        prevScreen,
        "1111",
        "1111",
        "error",
        CustomAlertTypes.ALERT_ERROR,
        false);
    display.setCurrent(alert);
    */
  }
  System.out.println("-----Background␣Task␣--␣Progress␣Gauge␣stoped
      ");
  // notify the progress gauge to quit
//  pg.stop();

  /*CanvasAlert alert = new CanvasAlert(
      display,
      prevScreen,
      "22222",
      "1111",
      "error",
      CustomAlertTypes.ALERT_ERROR,
      false);
  display.setCurrent(alert);
  */
}
try{
CanvasAlert alert = new CanvasAlert(
    display,
    new MyWalletMainMenu().prepareScreen(),
    "My␣Wallet␣Updated",
    "My␣Wallet␣content␣has␣been␣updated␣succesfully!",
    "info",
    CustomAlertTypes.ALERT_INFO,
    false);
display.setCurrent(alert);
}catch(Exception e){

  System.out.println("-----Exception␣in␣here");
  e.printStackTrace();
}
```

```
    }
  }

  /**
   * Template method that need to be implemented in the derived classes.
   * The actual task is implemented in this method by the derived class
   *
   * @throws Exception
   */
  public abstract void runTask () throws Exception;
}

package gui.mywallet;

import model.beans.otherbeans.CreditCardBean;
import constants.SystemConstants;

/**
 * Performs some helping operation for displaying the Secure Wallet
 * e.g. validating UI, formating no of decimals for the final price, etc
 *
 * @author Mihai Balan, s031288
 *
 */
public class GUIWalletHelper {

  public static boolean validWalletAddNewCardUI(CreditCardBean ccBean){
    boolean valid = true;

    if (ccBean.getCCNickName().equals(""))
      valid = false;

    if (ccBean.getCCOwner().equals(""))
      valid = false;

    if (ccBean.getCCNumber().equals(""))
      valid = false;

    if (ccBean.getCCNumber().length() != 16)
      valid = false;

    if (ccBean.getCCCW2().length() < 3)
      valid = false;

    try{
      if (Integer.parseInt(ccBean.getCCExpDateMonth()) > 12
        || Integer.parseInt(ccBean.getCCExpDateMonth()) < 1)
```

```
      valid = false;

    if (Integer.parseInt(ccBean.getCCExpDateYear()) < SystemConstants.
        CREDIT_CARD_EXP_YEAR_MIN
       || Integer.parseInt(ccBean.getCCExpDateYear()) > SystemConstants
           .CREDIT_CARD_EXP_YEAR_MAX)

      valid = false;

  } catch(NumberFormatException nfe){
    valid = false;
    System.out.println("Number␣format␣exception␣in␣GUIWallet␣Helper");
    nfe.printStackTrace();
  }
  catch(Exception e){
    valid = false;
    System.out.println("Exception␣in␣GUIWallet␣Helper");
    e.printStackTrace();
  }

  return valid;

}// end validWalletAddNewCardUI()


public static boolean validPayCreditCardUI(CreditCardBean ccBean){
  boolean valid = true;

  if (ccBean.getCCNumber().equals(""))
    valid = false;

  if (ccBean.getCCNumber().length() != 16)
    valid = false;

  if (ccBean.getCCCW2().length() < 3)
    valid = false;

  try{
    if (Integer.parseInt(ccBean.getCCExpDateMonth()) > 12
      || Integer.parseInt(ccBean.getCCExpDateMonth()) < 1)

      valid = false;

    if (Integer.parseInt(ccBean.getCCExpDateYear()) < SystemConstants.
        CREDIT_CARD_EXP_YEAR_MIN
       || Integer.parseInt(ccBean.getCCExpDateYear()) > SystemConstants
           .CREDIT_CARD_EXP_YEAR_MAX)
```

```
        valid = false;

    } catch(NumberFormatException nfe){
      valid = false;
      System.out.println("Number␣format␣exception␣in␣GUIWallet␣Helper");
      nfe.printStackTrace();
    }
    catch(Exception e){
      valid = false;
      System.out.println("Exception␣in␣GUIWallet␣Helper");
      e.printStackTrace();
    }

    return valid;

  }// end validWalletAddNewCardUI()

  /**
   * Format the Credit card number according to the standard
   *
   * @param ccNumber The Credit card no
   *
   * @return Return a CC no in the standard format i.e.
   *         4 chars sperated by space ...
   */
  public static String formatCCNumber(String ccNumber){
    return ccNumber.substring(0,4) + "␣"
          + ccNumber.substring(4,8) + "␣"
          + ccNumber.substring(8,12) + "␣"
          + ccNumber.substring(12,16);

  }// end formatCCNumber()

}// end class

package gui.mywallet;

import gui.GenericGUI;
import gui.customdialogwindows.CanvasAlert;
import gui.customdialogwindows.DialogWindow;
import gui.mainmenu.MenuScreen;

import java.io.IOException;

import javax.microedition.lcdui.*;

import model.beans.otherbeans.CreditCardBean;
```

```
import model.update.UpdateModel;

import org.bouncycastle.crypto.CryptoException;



import rms.RMSOperations;
import start.Start;

import constants.CreditCardTypes;
import constants.CustomAlertTypes;
import constants.SystemConstants;
import cryptography.Encryptor;

/**
 *  Add a new credit card and save it to RMS
 *
 *  It extends the GenericGUI super class
 *
 * @author s031288, Mihai Balan
 *
 */
public class MyWalletAddNewCC extends GenericGUI{

  // the add new CC screen
  private static Displayable screen = null;

  //  the commands
  private static Command backCommand;
  private static Command saveCommand;
  private static Command mainCommand;
  private static Command helpCommand;
  private static Command exitCommand;

  // user, password, and key text boxes
  private TextField ccNickName;
  private TextField ccOwner;
  private TextField ccNo;
  private TextField ccValidMonth;
  private TextField ccValidYear;
  private TextField ccCW2;
  private TextField ccPIN;
  private TextField ccBank;
  private TextField ccEmergencyPhone;
  private ChoiceGroup ccType;

  private Image     imgUp;
```

```
private ImageItem imgThemeUp;

private String textCCNickName = "";
private String[] textCCType    = {
    "VISA",
    "VISA␣ELECTRON",
    "AMERICAN␣EXPRESS",
    "MASTER␣CARD",
    "DANKORT",
    "MAETRO",
    "DINNERS␣CLUB"};

private String[] imgCCType     = {
    CreditCardTypes.CC_VISA,
    CreditCardTypes.CC_VISA_ELECTRON,
    CreditCardTypes.CC_AMERICAN_EXPRESS,
    CreditCardTypes.CC_MASTER_CARD,
    CreditCardTypes.CC_DANKORT,
    CreditCardTypes.CC_MAESTRO,
    CreditCardTypes.CC_DINERS_CLUB};


/** Reference to the Encryptor class in order
 to perform encryption/decryption operations */
public Encryptor encryptor = null;

/**
 * Constructs an instance of the class
 */
public MyWalletAddNewCC () {
}

/**
 * Returns the displayable authentication screen
 * @return screen Returns the WalletAuthenticationScreen screen
 */
public Displayable getScreen() {
  return screen;
}

/**
 * Function of the chosen command
 * the user saves the CC in RMS, go back to myWallet menu,
 * main menu, displays help or exit
 *
 * @param c The executed command
 * @param s The main menu form
```

```
 *
 */
public void commandAction(Command c, Displayable s) {
  try {

    if (c == saveCommand) {

      // get selected cc type
      boolean[] ccSelected = new boolean[ccType.size()];
      ccType.getSelectedFlags(ccSelected);
      int selected = ccType.getSelectedIndex();

      CreditCardBean ccBean = new CreditCardBean();
      ccBean.setCCNickName    (ccNickName.getString());
      ccBean.setCCOwner       (ccOwner.getString());
      ccBean.setCCType        (imgCCType[selected]);
      ccBean.setCCNumber      (ccNo.getString());
      ccBean.setCCExpDateMonth (ccValidMonth.getString());
      ccBean.setCCExpDateYear (ccValidYear.getString());
      ccBean.setCCCW2         (ccCW2.getString());
      ccBean.setCCPIN         (ccPIN.getString());
      ccBean.setCCBank        (ccBank.getString());
      ccBean.setCCEmergencyPhone (ccEmergencyPhone.getString());

      // if the data entered is valid save the CC to RMS
      if (GUIWalletHelper.validWalletAddNewCardUI(ccBean)){
        try{

          UpdateModel.addNewCreditCardAndUpdateAllCreditCards(display,
              ccBean);

        }catch (IOException ioe){
          System.out.println("IOException␣when␣writing␣the␣CC␣" + ioe.
              getMessage());
          ioe.printStackTrace();

        }catch (CryptoException ce){
          System.out.println("CryptoException␣when␣writing␣the␣CC␣" + ce
              .getMessage());
          ce.printStackTrace();

        }catch (Exception e){
          System.out.println("Exception␣when␣writing␣the␣CC␣" + e.
              getMessage());
          e.printStackTrace();
        }
```

```
  }else{
    CanvasAlert ss = new CanvasAlert(
        display,
        getScreen(),
        "Incorrect␣data!",
        "Please␣check␣that␣the␣exp.␣month␣is␣between␣1-12," +
          "␣exp.␣year␣is␣between"
          + SystemConstants.CREDIT_CARD_EXP_YEAR_MIN + "␣-␣20"
          + SystemConstants.CREDIT_CARD_EXP_YEAR_MAX + ""
          + ",␣and␣CW2␣has␣exactly␣3␣characters!",
        "error",
        CustomAlertTypes.ALERT_WARNING);
  }

}// end if (c == saveCommand)


if(c == backCommand){
  display.setCurrent(new MyWalletMainMenu().prepareScreen());
}// end if (c == backCommand)

if(c == mainCommand){
  display.setCurrent(new MenuScreen());
}// end if (c == mainCommand)

if (c == helpCommand){
    CanvasAlert help = new CanvasAlert(
        display,
        getScreen(),
        "My␣Wallet␣Help",
        "My␣Wallet␣displays␣all␣credit␣cards␣previously␣saved.␣You␣
            can␣add␣new␣cards␣(at␣most␣5),␣delete␣or␣update␣any␣
            credit␣card!",
        "question",
        CustomAlertTypes.ALERT_INFO);

  }// end if (c == helpCommand)

  if (c == exitCommand){
    DialogWindow reallyExit = new DialogWindow(
        display,
        getScreen(),
        "Exit␣Application?",
        "Are␣you␣sure␣that␣you␣want␣to␣exit␣the␣application?",
        "question",
        "/dialogIcons/exitTheme",
        new MenuScreen().startingpoint);
```

```
        display.setCurrent(reallyExit);

      } // end if (c == exitCommand)

  } catch (Exception e) {
    e.printStackTrace();
    CanvasAlert keyErrorAlert = new CanvasAlert(
        display,
        getScreen(),
        "New␣Credit␣Card␣Error!",
        "Error␣while␣adding␣a␣new␣Credit␣Card␣into␣the␣phone␣memory!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    display.setCurrent(keyErrorAlert);
  }
}

/**
 * Initialize the model an open the record store
 */
protected void initModel() throws Exception {
}

/**
 * Creates the Authentication Screen
 *
 * @throws Exception
 *
 */
protected void createView() throws Exception {
  saveCommand  = new Command("SAVE",   Command.EXIT, 0);
  backCommand  = new Command("BACK",   Command.SCREEN, 2);
  mainCommand  = new Command("MAIN␣MENU", Command.SCREEN, 3);
  helpCommand  = new Command("HELP",   Command.SCREEN, 4);
  exitCommand  = new Command("EXIT",   Command.SCREEN, 5);

  ccNickName     = new TextField("Credit␣Card␣Nick␣Name:", "", 40,
      TextField.ANY);
  ccOwner        = new TextField("Credit␣Card␣Owner:␣",    "", 40,
      TextField.ANY);
  ccNo           = new TextField("Credit␣Card␣Number:",    "", 16,
      TextField.NUMERIC);
  ccValidMonth   = new TextField("Credit␣Card␣Expiring␣Month:", "", 2,
      TextField.NUMERIC);
  ccValidYear    = new TextField("Credit␣Card␣Expiring␣␣␣␣Year:", ""
```

```
      , 4, TextField.NUMERIC);
  ccCW2          = new TextField("Credit␣Card␣Security␣Code:␣", "", 3,
      TextField.ANY);
  ccPIN          = new TextField("Credit␣Card␣PIN␣Code:␣", "", 4,
      TextField.ANY);
  ccBank         = new TextField("Bank␣Name:␣", "", 40, TextField.ANY);
  ccEmergencyPhone = new TextField("Emergencu␣Phone␣No:␣", "", 12,
      TextField.PHONENUMBER);

  ccType      = new ChoiceGroup("Credit␣Card␣Type:", Choice.POPUP,
      textCCType, null);

  try{
    imgUp = Image.createImage("/" + Start.themeDir + "/mywallet/
        walletAddTheme.png");
    imgThemeUp = new ImageItem("", imgUp, Item.LAYOUT_TOP | Item.
        LAYOUT_CENTER , "Theme_Img_Up");

  }catch(IOException ioe){
    System.out.println("Wallet␣Theme␣image␣exception!");
  }


  screen = new Form("My␣Wallet");
  ((Form)screen).append(imgUp);
  ((Form)screen).append(ccNickName);
  ((Form)screen).append(ccBank);
  ((Form)screen).append(ccType);
  ((Form)screen).append(ccOwner);
  ((Form)screen).append(ccNo);
  ((Form)screen).append(ccValidMonth);
  ((Form)screen).append(ccValidYear);
  ((Form)screen).append(ccCW2);
  ((Form)screen).append(ccPIN);
  ((Form)screen).append(ccEmergencyPhone);

  // add the commands to the form
  screen.addCommand(saveCommand);
  screen.addCommand(backCommand);
  screen.addCommand(mainCommand);
  screen.addCommand(helpCommand);
  screen.addCommand(exitCommand);
}

/*
 * Update the view - maybe refresh the fields
 *
```

```
   */
  protected void updateView() throws Exception {
    initModel();
    createView();
  }

}// end class

package gui.mywallet;

import gui.GenericGUI;
import gui.customdialogwindows.CanvasAlert;
import gui.mainmenu.MenuScreen;

import java.io.IOException;

import javax.microedition.lcdui.*;


import rms.RMSOperations;
import start.Start;
import constants.CustomAlertTypes;
import cryptography.Encryptor;

/**
 *  Displays the screen used to enter the PIN code to authenticate
 *  into My Wallet feature
 *
 *  It extends the GenericGUI super class
 *
 * @author s031288, Mihai Balan
 *
 */
public class MyWalletAuthenticationGUI extends GenericGUI{

  // the authentication screen
  private static Displayable screen = null;

  //the starting point of the application
  public static Start startingPoint;

  //  the exit and select commands
  private static Command backCommand;
  private static Command loginCommand;

  // user, password, and key text boxes
  private TextField PIN;
  private TextField verifPIN;
```

```
private Image imgUp;
private ImageItem imgThemeUp;

private String textFieldPIN= "";
private String textFieldVerifPIN = "";

private String userPIN = "";
private String userVerifPIN = "";

private boolean normalLogin = true;
private CanvasAlert alert;

// the number of times a user can enter the
// PIN wrong. If the PIN is entered wrong more then
// 3 times, My Wallet content is deleted and the PIN reset.
public static int pinTrials = 3;


/** reference to the Encryptor class in order
 to perform encryption/decryption operations */
public Encryptor encryptor = null;

/**
 * Constructs an instance of the class
 */
public MyWalletAuthenticationGUI () {
  this.textFieldPIN    = "PIN␣Code";
  this.textFieldVerifPIN = "Verify␣PIN␣Code";

}

/**
 * Returns the displayable authentication screen
 * @return screen Returns the WalletAuthenticationScreen screen
 */
public Displayable getScreen() {
  return screen;
}

/**
 * Function of the pressed button i.e. exit or Login
 * the user returns to the main menu or the WalletAuthenticationScreen.
 *
 * @param c The executed command
 * @param s The main menu form
 *
 */
```

```
public void commandAction(Command c, Displayable s) {
  try {
    // go back to main menu
    if (c == backCommand) {
      display.setCurrent(new MenuScreen());

    }// check if the key is valid and add it to the record store
    else if (c == loginCommand && normalLogin) {
      pinTrials = UpdateWalletGUI.walletAuthentication(PIN, display,
          pinTrials);

    } else if(c == loginCommand && !normalLogin){
      userPIN      = PIN.getString ();
      userVerifPIN = verifPIN.getString();

      if(!userPIN.equals(userVerifPIN)){
        alert = new CanvasAlert(
            display,
            getScreen(),
            "PIN␣Codes␣don't␣match!",
            "The␣provided␣PIN␣Codes␣do␣not␣match!␣Please␣try␣again!",
            "error",
            CustomAlertTypes.ALERT_ERROR);

      } else {
        // save the encrypted PIN code into RMS
        RMSOperations.writeEncryptedRecord("PIN:", userPIN.getBytes());
        Start.walletPin = userPIN;

        alert = new CanvasAlert(
            display,
            new MyWalletAuthenticationGUI().prepareScreen(),
            "PIN␣code␣Set␣up!",
            "Your␣Wallet␣has␣been␣setup␣to␣use␣the␣new␣PIN␣code!␣Now␣you
                ␣can␣use␣your␣wallet!",
            "OK",
            CustomAlertTypes.ALERT_INFO);

      }  // end if(!userPIN.equals(userVerifPIN))

    }// end else if(c == loginCommand && !normalLogin)

  } catch (Exception e) {
    e.printStackTrace();
    alert = new CanvasAlert(
        display,
        getScreen(),
```

```
        "Key␣Not␣Saved!",
        "Error␣while␣saving␣the␣PIN␣code␣into␣the␣phone␣memory!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    display.setCurrent(alert);
  }
}

/**
 * Initialize the model an open the record store
 */
protected void initModel() throws Exception {
  String rmsPIN = Start.walletPin;

  // if PIN code not found in RMS (user logs in for the first time in
      RMS)
  if(!rmsPIN.equals("")){
    normalLogin = true;
  } else {
    normalLogin = false;
  }

}// end initModel()

/**
 * Creates the Authentication Screen
 *
 * @throws Exception
 *
 */
protected void createView() throws Exception {

  backCommand = new Command("MAIN␣MENU", Command.EXIT, 1);

  try{
    imgUp = Image.createImage("/" + Start.themeDir + "/mywallet/
        walletAuthTheme.png");
    imgThemeUp = new ImageItem("", imgUp, Item.LAYOUT_TOP | Item.
        LAYOUT_CENTER , "Theme_Img_Up");

  }catch(IOException ioe){
    System.out.println("Wallet␣Theme␣image␣exception!");
  }

  String formName = "";
```

```
    // create the text fields and add them to the form
    if(normalLogin){
      formName    = "My␣Wallet␣Authentication!";
      PIN         = new TextField(textFieldPIN, "", 40, TextField.
          PASSWORD);
      loginCommand = new Command("LOGIN", Command.OK, 1);

    } else{
      formName    = "My␣Wallet␣PIN␣Code␣Setup!";
      PIN         = new TextField(textFieldPIN,   "", 40, TextField.
          PASSWORD);
      verifPIN    = new TextField(textFieldVerifPIN, "", 40, TextField.
          PASSWORD);
      loginCommand = new Command("SUBMIT", Command.OK, 1);
    }

    screen = new Form(formName);
    ((Form)screen).append(imgUp);

    if(normalLogin){
      ((Form)screen).append(PIN);
    }else{
      ((Form)screen).append(PIN);
      ((Form)screen).append(verifPIN);
    }

    // add the commands to the form
    screen.addCommand(backCommand);
    screen.addCommand(loginCommand);
  }

  /*
   * Update the view - maybe refresh the fields
   *
   */
  protected void updateView() throws Exception {
    initModel();
    createView();
  }

}// end class

package gui.mywallet;

import gui.GenericGUI;
import gui.customdialogwindows.CanvasAlert;
import gui.customdialogwindows.DialogWindow;
import gui.mainmenu.MenuScreen;
```

```
import java.io.IOException;

import javax.microedition.lcdui.*;

import model.beans.otherbeans.CreditCardBean;
import model.update.UpdateModel;

import org.bouncycastle.crypto.CryptoException;



import rms.RMSOperations;
import start.Start;
import constants.CreditCardTypes;
import constants.CustomAlertTypes;
import constants.SystemConstants;
import cryptography.Encryptor;

/**
 *  Edit CC data and save it to RMS
 *
 *  It extends the GenericGUI super class
 *
 * @author s031288, Mihai Balan
 *
 */
public class MyWalletEditCC extends GenericGUI{

  // the edit CC screen
  private static Displayable screen = null;

  //  the commands
  private static Command backCommand;
  private static Command saveCommand;
  private static Command mainCommand;
  private static Command helpCommand;
  private static Command exitCommand;

  // user, password, and key text boxes
  private TextField ccNickName;
  private TextField ccOwner;
  private TextField ccNo;
  private TextField ccValidMonth;
  private TextField ccValidYear;
  private TextField ccCW2;
```

```
private TextField ccPIN;
private TextField ccBank;
private TextField ccEmergencyPhone;
private ChoiceGroup ccType;

private Image     imgUp;
private ImageItem imgThemeUp;

private String[] textCCType     = {
    "VISA",
    "VISA␣ELECTRON",
    "MASTER␣CARD",
    "AMERICAN␣EXPRESS",
    "DANKORT",
    "MAETRO",
    "DINNERS␣CLUB"};

private String[] imgCCType      = {
    CreditCardTypes.CC_VISA,
    CreditCardTypes.CC_VISA_ELECTRON,
    CreditCardTypes.CC_MASTER_CARD,
    CreditCardTypes.CC_AMERICAN_EXPRESS,
    CreditCardTypes.CC_DANKORT,
    CreditCardTypes.CC_MAESTRO,
    CreditCardTypes.CC_DINERS_CLUB};

private boolean[] ccSel         =  new boolean[7];
private String  textCCNickName = "";
private String  textCCOwner     = "";
private String  textCCNo        = "";
private String  textCCValidMonth = "";
private String  textCCValidYear = "";
private String  textCCCW2       = "";
private String  textCCPIN       = "";
private String  textCCBank          = "";
private String  textCCEmergencyPhone = "";

private CreditCardBean ccBean;
private int ccRMSIndex;


/** reference to the Encryptor class in order
 to perform encryption/decryption operations */
public Encryptor encryptor = null;

/**
 * Constructs an instance of the class
```

```
 */

public MyWalletEditCC () {
}


public MyWalletEditCC (CreditCardBean ccBean, int ccRMSIndex) {
  this.ccBean    = ccBean;
  this.ccRMSIndex = ccRMSIndex;
}

/**
 * Returns the displayable authentication screen
 * @return screen Returns the WalletAuthenticationScreen screen
 */
public Displayable getScreen() {
  return screen;
}

/**
 * Function of the chosen command
 * the user saves the edited CC in RMS, go back to myWallet menu,
 * main menu, displays help or exit
 *
 * @param c The executed command
 * @param s The main menu form
 *
 */
public void commandAction(Command c, Displayable s) {
  try {

    if (c == saveCommand) {

      // get selected cc type
      boolean[] ccSelected = new boolean[ccType.size()];
      ccType.getSelectedFlags(ccSelected);
      int selected = ccType.getSelectedIndex();

      CreditCardBean ccBean = new CreditCardBean();
      ccBean.setCCNickName    (ccNickName.getString());
      ccBean.setCCOwner       (ccOwner.getString());
      ccBean.setCCType        (imgCCType[selected]);
      ccBean.setCCNumber      (ccNo.getString());
      ccBean.setCCExpDateMonth (ccValidMonth.getString());
      ccBean.setCCExpDateYear (ccValidYear.getString());
      ccBean.setCCCW2         (ccCW2.getString());
      ccBean.setCCPIN         (ccPIN.getString());
```

```
  ccBean.setCCBank         (ccBank.getString());
  ccBean.setCCEmergencyPhone (ccEmergencyPhone.getString());

  // if the data entered is valid save the CC to RMS
  if (GUIWalletHelper.validWalletAddNewCardUI(ccBean)){
    try{

      ccRMSIndex = UpdateModel.editCreditCardAndUpdateAllCreditCards
          (ccBean, ccRMSIndex, display);

    }catch (IOException ioe){
      System.out.println("IOException␣when␣updating␣the␣CC␣" + ioe.
          getMessage());
      ioe.printStackTrace();

    }catch (CryptoException ce){
      System.out.println("CryptoException␣when␣updating␣the␣CC␣" +
          ce.getMessage());
      ce.printStackTrace();

    }catch (Exception e){
      System.out.println("Exception␣when␣updating␣the␣CC␣" + e.
          getMessage());
      e.printStackTrace();
    }

  }else{
    CanvasAlert ss = new CanvasAlert(
        display,
        getScreen(),
        "Incorrect␣data!",
        "Please␣check␣that␣the␣exp.␣month␣is␣between␣1-12," +
          "␣exp.␣year␣is␣between␣200"
          + SystemConstants.CREDIT_CARD_EXP_YEAR_MIN + "␣-␣20"
          + SystemConstants.CREDIT_CARD_EXP_YEAR_MAX + ""
          + ",␣and␣CW2␣has␣exactly␣3␣characters!",
        "error",
        CustomAlertTypes.ALERT_WARNING);
  }

}// end if (c == saveCommand)

if(c == backCommand){
  display.setCurrent(new MyWalletMainMenu().prepareScreen());
}// end if (c == backCommand)

if(c == mainCommand){
```

```
      display.setCurrent(new MenuScreen());
    }// end if (c == mainCommand)

    if (c == helpCommand){
        CanvasAlert help = new CanvasAlert(
            display,
            getScreen(),
            "My␣Wallet␣Help",
            "My␣Wallet␣displays␣all␣credit␣cards␣previously␣saved.␣You␣
                can␣add␣new␣cards␣(at␣most␣5),␣delete␣or␣update␣any␣
                credit␣card!",
            "question",
            CustomAlertTypes.ALERT_INFO);

      }// end if (c == helpCommand)

      if (c == exitCommand){
        DialogWindow reallyExit = new DialogWindow(
            display,
            getScreen(),
            "Exit␣Application?",
            "Are␣you␣sure␣that␣you␣want␣to␣exit␣the␣application?",
            "question",
            "/dialogIcons/exitTheme",
            new MenuScreen().startingpoint);

        display.setCurrent(reallyExit);

      } // end if (c == exitCommand)

  } catch (Exception e) {
    e.printStackTrace();
    CanvasAlert keyErrorAlert = new CanvasAlert(
        display,
        getScreen(),
        "Update␣Credit␣Card␣Error!",
        "Error␣while␣updating␣a␣Credit␣Card␣data!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    display.setCurrent(keyErrorAlert);
  }
}

/**
 * Initialize the model an open the record store
 */
```

```
protected void initModel() throws Exception {
  for (int i = 0; i < imgCCType.length; i++){
    if (ccBean.getCCType().equals(imgCCType[i]))
      ccSel[i] = true;
    else
      ccSel[i] = false;
  } // end for()

} //end initModel()

/**
 * Creates the Authentication Screen
 *
 * @throws Exception
 *
 */
protected void createView() throws Exception {
  saveCommand  = new Command("SAVE",    Command.EXIT, 0);
  backCommand  = new Command("BACK",    Command.SCREEN, 2);
  mainCommand  = new Command("MAIN␣MENU", Command.SCREEN, 3);
  helpCommand  = new Command("HELP",    Command.SCREEN, 4);
  exitCommand  = new Command("EXIT",    Command.SCREEN, 5);

  ccNickName     = new TextField("Credit␣Card␣Nick␣Name:",   ccBean.
      getCCNickName(),    40, TextField.ANY);
  ccOwner        = new TextField("Credit␣Card␣Owner:␣",      ccBean.
      getCCOwner(),       40, TextField.ANY);
  ccNo           = new TextField("Credit␣Card␣Number:",      ccBean.
      getCCNumber(),      16, TextField.NUMERIC);
  ccValidMonth   = new TextField("Credit␣Card␣Expiring␣Month:", ccBean.
      getCCExpDateMonth(), 2, TextField.NUMERIC);
  ccValidYear    = new TextField("Credit␣Card␣Expiring␣␣␣␣Year:",
      ccBean.getCCExpDateYear(), 4, TextField.NUMERIC);
  ccCW2          = new TextField("Credit␣Card␣Security␣Code:␣", ccBean.
      getCCCW2(),      3,  TextField.ANY);
  ccPIN          = new TextField("Credit␣Card␣PIN␣Code␣␣␣␣␣:␣", ccBean.
      getCCPIN(),      4,  TextField.ANY);
  ccBank         = new TextField("Bank␣Name:␣",              ccBean.
      getCCBank(),        40, TextField.ANY);
  ccEmergencyPhone = new TextField("Emergencu␣Phone␣No:␣",   ccBean.
      getCCEmergencyPhone(), 12, TextField.PHONENUMBER);

  ccType         = new ChoiceGroup("Credit␣Card␣Type:", Choice.POPUP,
      textCCType, null);
  ccType.setSelectedFlags(ccSel);

  try{
```

```
      imgUp = Image.createImage("/" + Start.themeDir + "/mywallet/
          walletEditTheme.png");
      imgThemeUp = new ImageItem("", imgUp, Item.LAYOUT_TOP | Item.
          LAYOUT_CENTER , "Theme_Img_Up");

    }catch(IOException ioe){
      System.out.println("Wallet␣Theme␣image␣exception!");
    }


    screen = new Form("My␣Wallet");
    ((Form)screen).append(imgUp);
    ((Form)screen).append(ccNickName);
    ((Form)screen).append(ccBank);
    ((Form)screen).append(ccType);
    ((Form)screen).append(ccOwner);
    ((Form)screen).append(ccNo);
    ((Form)screen).append(ccValidMonth);
    ((Form)screen).append(ccValidYear);
    ((Form)screen).append(ccCW2);
    ((Form)screen).append(ccPIN);
    ((Form)screen).append(ccEmergencyPhone);

    // add the commands to the form
    screen.addCommand(saveCommand);
    screen.addCommand(backCommand);
    screen.addCommand(mainCommand);
    screen.addCommand(helpCommand);
    screen.addCommand(exitCommand);

  }// end createView()

  /*
   * Update the view - maybe refresh the fields
   *
   */
  protected void updateView() throws Exception {
    initModel();
    createView();
  }

}// end class

package gui.mywallet;

import java.io.IOException;
import javax.microedition.lcdui.*;
```

```
import model.beans.otherbeans.TicketBean;
import model.beans.requestbeans.Cinema_Hall_Conf_Req_Bean;
import model.beans.requestbeans.Select_Deselect_Seats_Req_Bean;
import model.beans.responsebeans.Cinema_Hall_Conf_Resp_Bean;
import model.beans.responsebeans.Select_Deselect_Seats_Resp_Bean;


import gui.GenericGUI;
import gui.customdialogwindows.CanvasAlert;
import gui.customdialogwindows.DialogWindow;
import gui.mainmenu.MenuScreen;
import start.Start;
import constants.CustomAlertTypes;
import constants.SystemConstants;

/**
 *  Displays My Wallet UI that shows all Credit Cards saved in RMS
 *
 *  It extends the GenericGUI super class
 *
 * @author s031288, Mihai Balan
 *
 */
public class MyWalletMainMenu extends GenericGUI{

  // the authentication screen
  private static Displayable screen = null;

  //the starting point of the application
  public static Start startingPoint;

  //  the exit and select commands
  private static Command viewCommand;
  private static Command addCommand;
  private static Command deleteCommand;
  private static Command editCommand;
  private static Command menuCommand;
  private static Command helpCommand;
  private static Command exitCommand;
  private static Command purchaseCommand;
  private static Command backCommand;

  private Image imgUp;
  private ImageItem imgThemeUp;

  // store the CC from RMS
  //private CreditCardBean[] walletCC = null;
```

```
private String ccChoiceTitle = "Available␣Credit␣Cards:";

// max no of CC saved in RMS
//public static int maxCCSaved = 0;

// the choice group for displaying all credit cards
private ChoiceGroup cgCreditCard;

// stores the names and images for all CC in the menu
private Image[] ccImages = null;
private String[] ccNickNames = null;

private Displayable                 parentScreen;
private String[]                    ticketsDiscountValue;
private String                      paymentMethodValue;
private String                      amountToPay;
private TicketBean[]                cinemaTickets;
private Cinema_Hall_Conf_Req_Bean   cineHallConfReqBean;
private Cinema_Hall_Conf_Resp_Bean  cineHallConfRespBean;
private Select_Deselect_Seats_Req_Bean selDeselectSeatsReqBean;
private Select_Deselect_Seats_Resp_Bean selDeselectSeatsRespBean;


/**
 * Constructs an instance of the class
 */
public MyWalletMainMenu () {
  this.parentScreen = new Form("Other");
}


public MyWalletMainMenu (Displayable parentScreen) {
  this.parentScreen = parentScreen;
}

public MyWalletMainMenu (
    Displayable                 backScreen,
    String[]                    ticketsDiscountValue,
    String                      paymentMethodValue,
    String                      amountToPay,
    TicketBean[]                cinemaTickets,
    Cinema_Hall_Conf_Req_Bean   cineHallConfReqBean,
    Cinema_Hall_Conf_Resp_Bean  cineHallConfRespBean,
    Select_Deselect_Seats_Req_Bean selDeselectSeatsReqBean,
    Select_Deselect_Seats_Resp_Bean selDeselectSeatsRespBean) {
```

```
  this.parentScreen          = backScreen;
  this.ticketsDiscountValue  = ticketsDiscountValue;
  this.paymentMethodValue    = paymentMethodValue;
  this.amountToPay           = amountToPay;
  this.cinemaTickets         = cinemaTickets;
  this.cineHallConfReqBean   = cineHallConfReqBean;
  this.cineHallConfRespBean  = cineHallConfRespBean;
  this.selDeselectSeatsReqBean = selDeselectSeatsReqBean;
  this.selDeselectSeatsRespBean = selDeselectSeatsRespBean;

}


/**
 * Returns the displayable authentication screen
 * @return screen Returns the WalletAuthenticationScreen screen
 */
public Displayable getScreen() {
  return screen;
}

/**
 * Function of the chosen menu option i.e. add, edit, delete, help,
     main menu, or exit
 * one of the functionalities with the same name is executed i.e.
 * add a new credit card, edit a previous saved credit card, remove a
     credit card,
 * display help, go back to appl main menu, and exit the appl,
     respectively.
 *
 * @param c The executed command
 * @param s The main menu form
 *
 */
public void commandAction(Command c, Displayable s) {
  try {

    if (c == viewCommand){
      boolean[] cgSelected = new boolean[cgCreditCard.size()];
      cgCreditCard.getSelectedFlags(cgSelected);
      int selectedCC = cgCreditCard.getSelectedIndex();

      new MyWalletViewCC(display, getScreen(), Start.creditCards[
          selectedCC]);

    }// end if (c == viewCommand)
```

```
if (c == addCommand){
  // check if the max no of CC's is reached and does not allow
  // to add new CC in that case
  if (Start.maxCCSaved == SystemConstants.MAX_NO_CREDIT_CARDS){
    CanvasAlert ss = new CanvasAlert(
        display,
        new MyWalletMainMenu().prepareScreen(),
        "Max␣no␣of␣Credit␣Cards␣reached!",
        "You␣can␣save␣up␣to␣5␣credit␣cards␣in␣your␣wallet!",
        "error",
        CustomAlertTypes.ALERT_WARNING);

  } else{
    new MyWalletAddNewCC().showScreen();

  } // end if (MAX_NO_CREDIT_CARDS)

}// end if (c == addCommand)

if (c == deleteCommand){
  DialogWindow reallyDelete = new DialogWindow(
      display,
      new MyWalletMainMenu().prepareScreen(),
      "Remove␣Credit␣Card?",
      "Are␣you␣sure␣that␣you␣want␣to␣remove␣this␣credit␣card␣
          permanently?",
      "question",
      "/theme_red/mywallet/walletCancelTheme",
      new MenuScreen().startingpoint,
      cgCreditCard,
      Start.creditCards
  );

  display.setCurrent(reallyDelete);

}// end if (c == deleteCommand)

if (c == editCommand){
  boolean[] cgSelected = new boolean[cgCreditCard.size()];
  cgCreditCard.getSelectedFlags(cgSelected);
  int selectedIndex = cgCreditCard.getSelectedIndex();

  new MyWalletEditCC(Start.creditCards[selectedIndex],
      selectedIndex).showScreen();

}// end if (c == editCommand)
```

```
// go back to main menu
if (c == menuCommand) {
  display.setCurrent(new MenuScreen());
}// end if (c == menuCommand)

if (c == helpCommand){
  CanvasAlert help = new CanvasAlert(
      display,
      getScreen(),
      "My␣Wallet␣Help",
      "My␣Wallet␣displays␣all␣credit␣cards␣previously␣saved.␣You␣can
          ␣add␣new␣cards␣(at␣most␣5),␣delete␣or␣update␣any␣credit␣
          card!",
      "question",
      CustomAlertTypes.ALERT_INFO);

} // end if (c == helpCommand)

if (c == exitCommand){
  DialogWindow reallyExit = new DialogWindow(
      display,
      getScreen(),
      "Exit␣Application?",
      "Are␣you␣sure␣that␣you␣want␣to␣exit␣the␣application?",
      "question",
      "/dialogIcons/exitTheme",
      new MenuScreen().startingpoint);

  display.setCurrent(reallyExit);

} // end if (c == exitCommand)


// in case the screen is called by Ticket Payment
if (c == purchaseCommand){

  if(parentScreen.getTitle().equals("Ticket␣Payment")){
    // get the selected CC index
    boolean[] cgSelected = new boolean[cgCreditCard.size()];
    cgCreditCard.getSelectedFlags(cgSelected);
    int selectedCC = cgCreditCard.getSelectedIndex();

    PaymentViaWallet.payViaWallet(
        display,
        getScreen(),
        ticketsDiscountValue,
        paymentMethodValue,
```

```
            amountToPay,
            Start.creditCards[selectedCC],
            cinemaTickets,
            cineHallConfReqBean,
            cineHallConfRespBean,
            selDeselectSeatsReqBean,
            selDeselectSeatsRespBean);

      }// end if(parentScreen.getTitle().equals("Ticket Payment"))

    } // end if (c == purchaseCommand)


    // in case the screen is called by Ticket Payment
    if (c == backCommand){
      if(parentScreen.getTitle().equals("Ticket␣Payment")){
        display.setCurrent(parentScreen);
      }
    } // end if (c == backCommand)

  }catch (Exception e) {
    e.printStackTrace();
    CanvasAlert keyErrorAlert = new CanvasAlert(
        display,
        getScreen(),
        "My␣Wallet␣Error!",
        "Error␣while␣displaying␣all␣Credit␣Cards!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    display.setCurrent(keyErrorAlert);
  }
}

/**
 * Initialize the model an open the record store
 */
protected void initModel() {
  try{
    //walletCC = null;
    //walletCC = RMSOperations.getAllCreditCards();
    //maxCCSaved = walletCC.length;

    // if there are any credit cards in the wallet
    // construct the choice group and display it
    ccNickNames = new String[Start.maxCCSaved];
    ccImages   = new Image [Start.maxCCSaved];
```

```
    if (Start.maxCCSaved > 0){
      String imgName = "";
      for (int i = 0; i < Start.maxCCSaved; i++){
        ccNickNames[i] = Start.creditCards[i].getCCNickName();
        imgName = "/CCTypes/" + Start.creditCards[i].getCCType() + ".png
            ";
        ccImages[i] = Image.createImage(imgName);
      } // end for

    } else{
      ccChoiceTitle = "There␣are␣no␣credit␣cards␣in␣the␣wallet!";
    }
  }catch(Exception e){
    System.out.println("Exception␣in␣init");
    e.printStackTrace();
  }

} // end initModel()

/**
 * Creates the Authentication Screen
 *
 * @throws Exception
 *
 */
protected void createView() throws Exception {
  viewCommand    = new Command("VIEW",           Command.EXIT, 0);
  purchaseCommand = new Command("PURCHASE",      Command.EXIT, 0);
  backCommand    = new Command("BACK",           Command.EXIT, 1);
  menuCommand    = new Command("MAIN␣MENU",      Command.SCREEN, 2);
  addCommand     = new Command("NEW␣CREDIT␣CARD", Command.SCREEN, 3);
  editCommand    = new Command("EDIT␣CREDIT␣CARD", Command.SCREEN, 4);
  deleteCommand  = new Command("REMOVE␣CREDIT␣CARD", Command.SCREEN, 5);
  helpCommand    = new Command("HELP",           Command.SCREEN, 6);
  exitCommand    = new Command("EXIT",           Command.SCREEN, 7);
  cgCreditCard = new ChoiceGroup(ccChoiceTitle, Choice.EXCLUSIVE,
      ccNickNames, ccImages);

  try{
    imgUp = Image.createImage("/" + Start.themeDir + "/mywallet/
        walletMainTheme.png");
    imgThemeUp = new ImageItem("", imgUp, Item.LAYOUT_TOP | Item.
        LAYOUT_CENTER , "Theme_Img_Up");
  }catch(IOException ioe){
    System.out.println("Wallet␣Theme␣image␣exception!");
  }
```

```
    screen = new Form("My Wallet");
    ((Form)screen).append(imgUp);
    ((Form)screen).append(cgCreditCard);

    // in case the screen is initialized from
    // the Ticket Payment Screen
    if(parentScreen.getTitle().equals("Ticket Payment")){
      // add the commands to the form
      screen.addCommand(purchaseCommand);
      screen.addCommand(backCommand);

    }else if(parentScreen.getTitle().equals("Other")){
      // add the commands to the form
      screen.addCommand(menuCommand);
      screen.addCommand(viewCommand);
      screen.addCommand(addCommand);
      screen.addCommand(deleteCommand);
      screen.addCommand(editCommand);
      screen.addCommand(helpCommand);
      screen.addCommand(exitCommand);

    }// end if(parentScreen.getTitle().equals("Ticket Payment"))

  }

  /*
   * Update the view - maybe refresh the fields
   *
   */
  protected void updateView() throws Exception {
    initModel();
    createView();
  }

}// end class

package gui.mywallet;


import gui.GUIHelper;

import java.io.IOException;

import javax.microedition.lcdui.*;

import model.beans.otherbeans.CreditCardBean;
```

```
import start.Start;


/**
 * Display the selected credit card data in
 * a very user friendly way i.e. the same format as
 * regular credit card
 *
 * @author s031288, Mihai Balan
 */
public class MyWalletViewCC extends Canvas {

  // the display to draw on
  private Display   display;

  private Displayable next;

  private Graphics g;

  private Image   ccTypeImg;
  private ImageItem ccTypeImgItem;

  private Image   bckgImg;
  private ImageItem bckgImgItem;

  private CreditCardBean ccBean;

  private Start midlet;

  // the curent selected option i.e. PIN, CW2, BACK
  private int selectedOptionIndex = 0;

  // NOT highlighted buttons
  private String[] optionDeselected = {
      "/CCViewButtons/pinDeselected.png",
      "/CCViewButtons/cw2Deselected.png",
      "/CCViewButtons/backDeselected.png"
  };

  // the highlighted buttons
  private String[] optionSelected = {
      "/CCViewButtons/pinSelected.png",
      "/CCViewButtons/cw2Selected.png",
      "/CCViewButtons/backSelected.png"
```

```
};

// the images for building the YES and NO options
private Image[] deselectedImgs;
private Image[] selectedImgs;

private int ccHeight = 0;
private int ccWidth = 0;
private int startX = 0;
private int startY = 0;
private int ccX = 0;
private int ccY = 0;
private int textX = 0;
private int textY = 0;

private boolean drawSelection = false;

/**
 * Constructor for the credit card view screen
 *
 * @param display The display to draw on
 * @param next  The next screen to be displayed after the CC data
 * @param ccBean The credit card bean containing all data about the
     selected CC
 */
public MyWalletViewCC( Display display, Displayable next,
    CreditCardBean ccBean){

  this.display = display;
  this.next   = next;
  this.ccBean = ccBean;

  selectedImgs  = GUIHelper.createSelectedButtons(optionSelected);
  deselectedImgs = GUIHelper.createDeselectedButtons(optionDeselected);

  display.setCurrent(this);
}


/** The image and text displaying takes place in here
 *
 * Get the splash image from a .png file, convert it to a byte array
 * and then display it on the scree
 *
 * @param g The graphocs to draw on
 */
protected void paint( Graphics g ){
```

```
int w = getWidth();
int h = getHeight();

ccWidth = w - 20;
ccHeight = 2*h/3;
startX = 10;
startY = 50;
ccX = startX + 2;
ccY = startY + 30;

try {

  ccTypeImg    = Image.createImage("/CCTypesBig/" + ccBean.getCCType()
       +".png");
  ccTypeImgItem = new ImageItem("", ccTypeImg, Item.LAYOUT_TOP | Item
       .LAYOUT_RIGHT , "Theme_Img_Up");

  bckgImg    = Image.createImage("/theme_red/mywallet/ccBackground.
      png");
  bckgImgItem = new ImageItem("", bckgImg, Item.LAYOUT_TOP | Item.
       LAYOUT_RIGHT , "Theme_Img_Up");

  // draw the background
  g.drawImage(bckgImg, 0, 0,
      Graphics.TOP | Graphics.LEFT);

  // draw the CC
  g.setColor(255, 255, 225);
  g.fillRect(ccX, ccY, ccWidth - 6, ccHeight - 58);

  // draw the CC type image
  g.drawImage(ccTypeImg, w - 27 , ccY + 15,
      Graphics.TOP | Graphics.RIGHT);

  // draw the text on the CC

  // define the msg font
  Font msgFontBold = Font.getFont(
      Font.FACE_PROPORTIONAL,
      Font.STYLE_BOLD,
      Font.SIZE_LARGE);

  Font msgFontBoldMedium = Font.getFont(
      Font.FACE_PROPORTIONAL,
      Font.STYLE_BOLD,
      Font.SIZE_MEDIUM);
```

```
Font msgFontPlain = Font.getFont(
    Font.FACE_PROPORTIONAL,
    Font.STYLE_PLAIN,
    Font.SIZE_SMALL);

Font msgFontPlainMedium = Font.getFont(
    Font.FACE_PROPORTIONAL,
    Font.STYLE_PLAIN,
    Font.SIZE_MEDIUM);

// set the message font and color
textX = ccX + 15;
textY = ccY + 15;

g.setColor(255, 255, 255);
g.setFont(msgFontBoldMedium);
g.drawString(ccBean.getCCNickName().toUpperCase(), w/2, 60,
    Graphics.TOP | Graphics.HCENTER);

g.setColor(0, 50, 125);
g.drawString(ccBean.getCCBank().toUpperCase(), textX, textY ,
    Graphics.TOP | Graphics.LEFT);

g.setFont(msgFontBold);
g.drawString(GUIWalletHelper.formatCCNumber(ccBean.getCCNumber()),
    textX, textY + 35, Graphics.TOP | Graphics.LEFT);

g.setFont(msgFontBoldMedium);
g.drawString(ccBean.getCCOwner().toUpperCase(), textX, textY
     + 45 + 20, Graphics.TOP | Graphics.LEFT);

g.setFont(msgFontPlainMedium);
g.drawString("Expires:␣" + ccBean.getCCExpDateMonth() + "/"+ ccBean
    .getCCExpDateYear(), textX, textY + 45 + 20 + 15 , Graphics.TOP
     | Graphics.LEFT);

g.setFont(msgFontPlain);
g.drawString("If␣card␣is␣lost␣or␣stolen␣call:␣" + ccBean.
    getCCEmergencyPhone(), textX, textY + 45 + 20 + 15 + 25 ,
    Graphics.TOP | Graphics.LEFT);


// draw the buttons
GUIHelper.drawCCViewButtons(g,
    deselectedImgs, selectedImgs,
    w, h,
```

```
        92, textY + 45 + 20 + 15 + 15 + 19,
        selectedOptionIndex);

    // draw the PIN code, or CW2 if the corespondent button is pressed
    if(drawSelection)
      drawSelection(g, msgFontBold, w/2, h - 15 );

  } catch(IOException ioe){
    System.out.println("CC␣type␣image␣exception!");
    // if the image cannot be drawn, write some text
    g.drawString("Mobile␣Cinema␣IOException", w/2, h/2,
        Graphics.BASELINE | Graphics.HCENTER);

  } catch (Exception e) {
    System.out.println("View␣CC␣exception!");
    g.drawString("Mobile␣Cinema␣Exception", w/2, h/2,
        Graphics.BASELINE | Graphics.HCENTER);
  }
}

/**
 * Update the value of the current selected option in the list
 * and repaint the screen.
 *
 * @param keyCode The code of the pressed key
 */
protected void keyPressed( int keyCode ){

  if ((getGameAction(keyCode) == Canvas.LEFT)){

    if(selectedOptionIndex > 0){

      selectedOptionIndex--;
      repaint();

    } else if(selectedOptionIndex == 0){

      selectedOptionIndex = optionDeselected.length - 1;
      repaint();
    }

  } else if ((getGameAction(keyCode) == Canvas.RIGHT)){

    if(selectedOptionIndex < optionDeselected.length - 1){

      selectedOptionIndex++;
      repaint();
```

```
    } else if(selectedOptionIndex == optionDeselected.length - 1){

      selectedOptionIndex = 0;
      repaint();
    }

  } else if ((getGameAction(keyCode) == Canvas.FIRE)){

    if(selectedOptionIndex == 0){
      drawSelection = true;
      repaint();

    }else if(selectedOptionIndex == 1){
      drawSelection = true;
      repaint();

    }else if(selectedOptionIndex == 2){
      if(midlet == null)
        display.setCurrent(next);
    }

  } // end if(FIRE)

}// end keyPressed

/**
 * Draw selected item i.e. pin code, or cw2
 */
private void drawSelection(Graphics g, Font f, int x, int y){

  g.setColor(255, 255, 255);
  g.setFont(f);

  if(selectedOptionIndex == 0){
    g.drawString(ccBean.getCCPIN(), x, y,
        Graphics.BASELINE | Graphics.HCENTER);

  }else if(selectedOptionIndex == 1){
    g.drawString(ccBean.getCCCW2(), x, y,
        Graphics.BASELINE | Graphics.HCENTER);
  }

  drawSelection = false;

}// end drawSelection()
```

```
}// end class

package gui.mywallet;

import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;

import model.beans.otherbeans.CreditCardBean;
import model.beans.otherbeans.TicketBean;
import model.beans.requestbeans.Cinema_Hall_Conf_Req_Bean;
import model.beans.requestbeans.Select_Deselect_Seats_Req_Bean;
import model.beans.responsebeans.Cinema_Hall_Conf_Resp_Bean;
import model.beans.responsebeans.Select_Deselect_Seats_Resp_Bean;



import gui.purchasetickets.step5chooseticketpayment.
    UpdateTicketPaymentScreen;

/**
 * If the payment methos is CREDIT CARD FROM SECURE WALLET
 * this class performs the online payement
 *
 * @author Mihai Balan, s031288
 *
 */
public class PaymentViaWallet {

  /**
   * Performs the ticket payments with the server side using a
   * credit card from the seciure wallet
   *
   * @param display
   * @param backScreen
   * @param ticketsDiscountValue
   * @param paymentMethodValue
   * @param amountToPay
   * @param creditCard
   * @param cinemaTickets
   * @param cineHallConfReqBean
   * @param cineHallConfRespBean
   * @param selDeselectSeatsReqBean
   * @param selDeselectSeatsRespBean
   * @throws Exception
   */
  public static void payViaWallet(
      Display                     display,
      Displayable                 backScreen,
```

```
      String[]                    ticketsDiscountValue,
      String                      paymentMethodValue,
      String                      amountToPay,
      CreditCardBean              creditCard,
      TicketBean[]                cinemaTickets,
      Cinema_Hall_Conf_Req_Bean   cineHallConfReqBean,
      Cinema_Hall_Conf_Resp_Bean  cineHallConfRespBean,
      Select_Deselect_Seats_Req_Bean selDeselectSeatsReqBean,
      Select_Deselect_Seats_Resp_Bean selDeselectSeatsRespBean) throws
          Exception{


    UpdateTicketPaymentScreen.payOverNetwork(
        display,
        backScreen,
        ticketsDiscountValue,
        paymentMethodValue,
        creditCard.getCCType(),
        creditCard.getCCNumber(),
        creditCard.getCCExpDateMonth(),
        creditCard.getCCExpDateYear(),
        creditCard.getCCCW2(),
        amountToPay,
        cinemaTickets,
        cineHallConfReqBean,
        cineHallConfRespBean,
        selDeselectSeatsReqBean,
        selDeselectSeatsRespBean);

  }// end payViaWallet()

}// end class

package gui.mywallet;

import gui.customdialogwindows.CanvasAlert;

import java.io.IOException;

import javax.microedition.lcdui.ChoiceGroup;
import javax.microedition.lcdui.Display;

import model.beans.otherbeans.CreditCardBean;

import org.bouncycastle.crypto.CryptoException;

import rms.RMSOperations;
import start.Start;
```

```
import constants.CustomAlertTypes;
import constants.SystemConstants;

public class UpdateWalletBackground extends BackgroundUpdate {

  ChoiceGroup cgCreditCard;
  Display display;
  CanvasAlert alert;

  public UpdateWalletBackground(ChoiceGroup cgCreditCard, Display display
      ){
    super(display);
    this.cgCreditCard = cgCreditCard;
  }

  public void runTask() throws Exception {
    // TODO Auto-generated method stub

    boolean[] cgSelected = new boolean[cgCreditCard.size()];
    cgCreditCard.getSelectedFlags(cgSelected);
    int selectedIndex = cgCreditCard.getSelectedIndex() + 1;

    try{
      RMSOperations.deleteItems("CC" + selectedIndex + ":");

      // save the remaining CCs
      int j = 0;
      CreditCardBean[] updatedWalletCC = new CreditCardBean[Start.
          maxCCSaved - 1];

      for (int i = 0; i < Start.maxCCSaved; i++){
        if(i != (selectedIndex-1)){
          updatedWalletCC[j] = Start.creditCards[i];
          ++j;
        }
      }

      // delete all CCs
      for (int i = 1; i <SystemConstants.MAX_NO_CREDIT_CARDS + 1 ; i++)
        RMSOperations.deleteItems("CC" + i + ":");

      // save the remaining CCs back to RMS
      for (int i = 1; i <= updatedWalletCC.length ; i++){
        RMSOperations.writeEncryptedRecord("CC" + i + ":",
            updatedWalletCC[i-1].getBytes());
      }
```

```
  RMSOperations.deleteItems("CCN:");
  RMSOperations.writeRecord("CCN:", String.valueOf((Start.maxCCSaved
       - 1)));

  // update the Start.creditCards and Start.maxCCSaved
  // to make them available to the whole application
  // and improve appl performance by reducing the access to RMS
  if (updatedWalletCC.length > 0){
    System.out.println("-----in␣here␣1");

    Start.creditCards = null;
    System.out.println("-----in␣here␣2");
    Start.maxCCSaved -= 1;
    System.out.println("-----in␣here␣3");
    Start.creditCards = new CreditCardBean[Start.maxCCSaved];
    System.out.println("-----in␣here␣4");

    for (int i = 0; i < Start.maxCCSaved; i++){
      Start.creditCards[i] = new CreditCardBean();
      Start.creditCards[i] = updatedWalletCC[i];
      System.out.println("-----in␣here␣5␣+␣" + i);
    }
    System.out.println("-----in␣here␣6");
  }
  System.out.println("-----in␣here␣7");

/*  alert = new CanvasAlert(
      display,
      new MyWalletMainMenu().prepareScreen(),
      "Credit Card Removed!",
      "The credit card has been removed from your wallet!",
      "OK",
      CustomAlertTypes.ALERT_INFO);*/
  System.out.println("-----in␣here␣8");

}catch (IOException ioe){
  System.out.println("IOException␣when␣deleting␣the␣CC␣" + ioe.
       getMessage());
  ioe.printStackTrace();

}catch (CryptoException ce){
  System.out.println("CryptoException␣when␣deleting␣the␣CC␣" + ce.
       getMessage());
  ce.printStackTrace();

}catch (Exception e){
  System.out.println("Exception␣when␣deleting␣the␣CC␣" + e.getMessage
```

```
          ());
      e.printStackTrace();

    } // end try - catch

  }

}

package gui.mywallet;

import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.TextField;

import model.beans.otherbeans.TicketBean;
import model.beans.requestbeans.Cinema_Hall_Conf_Req_Bean;
import model.beans.requestbeans.Select_Deselect_Seats_Req_Bean;
import model.beans.responsebeans.Cinema_Hall_Conf_Resp_Bean;
import model.beans.responsebeans.Select_Deselect_Seats_Resp_Bean;



import gui.customdialogwindows.CanvasAlert;
import rms.RMSOperations;
import start.Start;
import constants.CustomAlertTypes;

/**
 * Updates Wallet UI function of the authentication result
 *
 * @author s031288, Mihai Balan
 *
 */
public class UpdateWalletGUI {

  /**
   * Check if the provided PIN code matches the one found in RMS
   * and display a message accordingly
   */
  public static int walletAuthentication(TextField PIN, Display display,
      int pinTrials) throws Exception{
    String userPIN = PIN.getString();
    //String rmsPIN = new String(RMSOperations.getDecryptedItem("PIN:"));

    if(userPIN.equals(Start.walletPin)){
      CanvasAlert authOK = new CanvasAlert(
          display,
```

```
        new MyWalletMainMenu().prepareScreen(),
        true,
        2000,
        "User␣Authenticated!",
        "Your␣are␣authorized␣to␣use␣your␣wallet!",
        "OK",
        CustomAlertTypes.ALERT_INFO);

  } else{
    if (pinTrials > 0){
      --pinTrials;
      CanvasAlert ss = new CanvasAlert(
          display,
          new MyWalletAuthenticationGUI().prepareScreen(),
          "Invalid␣PIN␣Code!",
          "Invalid␣PIN␣Code!␣You␣can␣try␣" + pinTrials + "␣more␣times!",
          "error",
          CustomAlertTypes.ALERT_ERROR);
    }

    // in case a user enters the PIN wrong more then 3 time
    // reset the wallet and pin
    if (pinTrials == 0){
      RMSOperations.resetMyWallet();

      CanvasAlert ss = new CanvasAlert(
          display,
          new MyWalletAuthenticationGUI().prepareScreen(),
          "My␣Wallet␣reseted!",
          "The␣PIN␣has␣been␣entered␣wrong␣more␣then␣3␣times.␣My␣Wallet␣
              content␣has␣been␣reseted!",
          "error",
          CustomAlertTypes.ALERT_ERROR);
    }

  }// end if(userPIN.equals(rmsPIN))

  return pinTrials;

}// end walletAuthentication()


/**
 * check if the provided PIN code matches the one found in RMS
 * and display a message accordingly
 */
public static int walletAuthenticationTicketPayment(
```

```
  Display                       display,
  Displayable                   backScreen,
  String[]                      ticketsDiscountValue,
  String                        paymentMethodValue,
  String                        walletPin,
  String                        amountToPay,
  int                           pinTrials,
  TicketBean[]                  cinemaTickets,
  Cinema_Hall_Conf_Req_Bean     cineHallConfReqBean,
  Cinema_Hall_Conf_Resp_Bean    cineHallConfRespBean,
  Select_Deselect_Seats_Req_Bean selDeselectSeatsReqBean,
  Select_Deselect_Seats_Resp_Bean selDeselectSeatsRespBean) throws
      Exception{

CanvasAlert alert;

if(walletPin.equals(Start.walletPin)){
  alert = new CanvasAlert(
      display,
      new MyWalletMainMenu(
          backScreen,
          ticketsDiscountValue,
          paymentMethodValue,
          amountToPay,
          cinemaTickets,
          cineHallConfReqBean,
          cineHallConfRespBean,
          selDeselectSeatsReqBean,
          selDeselectSeatsRespBean).prepareScreen(),
          true,
          2000,
          "User␣Authenticated!",
          "Your␣are␣authorized␣to␣use␣your␣wallet!",
          "OK",
          CustomAlertTypes.ALERT_INFO);

} else{
  if (pinTrials > 0){
    --pinTrials;
    CanvasAlert ss = new CanvasAlert(
        display,
        backScreen,
        "Invalid␣PIN␣Code!",
        "Invalid␣PIN␣Code!␣You␣can␣try␣" + pinTrials + "␣more␣times!",
        "error",
        CustomAlertTypes.ALERT_ERROR);
  }
```

```
      // in case a user enters the PIN wrong more then 3 time
      // reset the wallet and pin
      if (pinTrials == 0){
        RMSOperations.resetMyWallet();
        alert = new CanvasAlert(
            display,
            new MyWalletAuthenticationGUI().prepareScreen(),
            "My␣Wallet␣reseted!",
            "The␣PIN␣has␣been␣entered␣wrong␣more␣then␣3␣times.␣My␣Wallet␣
                content␣has␣been␣reseted!",
            "error",
            CustomAlertTypes.ALERT_ERROR);
      }

    }// end if(userPIN.equals(rmsPIN))

    return pinTrials;

  }// end walletAuthentication()

}// end class

package gui.purchasetickets.step1moviesearch;

import gui.GUIHelper;

/**
 * Implements some helping functionality used in the serach for movies
     scenario
 * such as: extracting a unique list of elements from an array, etc
 *
 * @author s031288, Mihai Balan
 *
 */
public class MovieSearchHelper {

  /**
   * Returns a single dim array containing unique values
   * e.g. movie names, shouw hours, cinemas, etc.
   *
   * It does that by:
   * constructing a bi dim array containing the values from
   * the column in the original array that holds the data
   * to be extracted
   * and a flag (0 or 1) used for "pseudo" sorting the array
   * by removing the duplicates from the given column
   *
```

```
 * If the element in the column is marked with 1 it means that it has
     been found before
 * therefore it is a duplicate and it is not gonna be added to the
 * final list. If the element is marked with 0 it is either the first
     occurence
 * of the element name in the array or it is unique
 */
public static String[] getUniqueValues(String[][] originalArray, int
    elementColumn){

  String[]  uniqueElements;
  String[][] tempElements = new String[originalArray.length][2];

  // get all elements from the original array

  // set the flag of all elements in the array to "0"
  for(int i = 0; i < originalArray.length; i++){
    tempElements[i][0] = originalArray[i][elementColumn].trim();
    tempElements[i][1] = "0";

  }// end for(i)

  // the number of unique elements
  int count = 0;


  for(int i = 0; i < tempElements.length; i++){

    // if an element has the flag = 0
    // i.e. unique or first time encountered
    if(tempElements[i][1].equals("0")){
      // count it
      ++count;

      // and check in the remaining string to see if there
      // is any other element with the same value.
      //If yes, mark the next ones as duplicates i.e. flag = 1;
      for(int j = i+1; j < tempElements.length; j++){
        if(tempElements[j][0].equals(tempElements[i][0])){
          tempElements[j][1] = "1";

        }// end if

      }// end for (i = j)

    }// end if()
```

```
  }// end for (i=0)

  // construct the array of unique elements
  uniqueElements = new String[count];
  int index = 0;

  // get all elements in the bi dim array marked as unique(0)
  // and add them to the array of unique elements
  for(int i = 0; i < tempElements.length; i++){

    if(tempElements[i][1].equals("0")){
      uniqueElements[index++] = tempElements[i][0];

    }
  }

  return uniqueElements;

}// end getUniqueValues()



/**
 * Remove the duplicates from a String array and returns the newly
     formed array
 *
 * @param The input array
 */
public static String[] getUniqueValues(String[] inputArray){

  String[] uniqueElements;
  String[][] tempElements = new String[inputArray.length][2];

  // get all elements from the original array

  // set the flag of all elements in the array to "0"
  for(int i = 0; i < inputArray.length; i++){
    tempElements[i][0] = inputArray[i].trim();
    tempElements[i][1] = "0";

  }// end for(i)

  // the number of unique elements
  int count = 0;


  for(int i = 0; i < tempElements.length; i++){
```

```
    // if an element has the flag = 0
    // i.e. unique or first time encountered
    if(tempElements[i][1].equals("0")){
      // count it
      ++count;

      // and check in the remaining string to see if there
      // is any other element with the same value.
      //If yes, mark the next ones as duplicates i.e. flag = 1;
      for(int j = i+1; j < tempElements.length; j++){
        if(tempElements[j][0].equals(tempElements[i][0])){
          tempElements[j][1] = "1";

        }// end if

      }// end for (i = j)

    }// end if()

  }// end for (i=0)

  // construct the array of unique elements
  uniqueElements = new String[count];
  int index = 0;

  // get all elements in the bi dim array marked as unique(0)
  // and add them to the array of unique elements
  for(int i = 0; i < tempElements.length; i++){

    if(tempElements[i][1].equals("0")){
      uniqueElements[index++] = tempElements[i][0];

    }
  }

  return uniqueElements;

}// end getUniqueValues()

/**
 * Returns a single dim array containing unique movie names
 */
public static String[] getUniqueCinemaNamesFromSorted(String[][]
    reqMovies){

  String[] uniqueMoviesTmp = new String[reqMovies.length];
```

```
  String[] uniqueMovies = null;
  // get all movie names from the server response

  int count = 0;

  // set the flag of all elements in the array to "0"
  for(int i = 0; i < reqMovies.length -1; i++){

    if(!reqMovies[i][0].trim().equals(reqMovies[i+1][0].trim())){

      uniqueMoviesTmp[count++] = reqMovies[i][0];

      if(i == reqMovies.length-2){
        uniqueMoviesTmp[count++] = reqMovies[i+1 ][0];

      }
    }else{
      if(i == reqMovies.length-2)
        uniqueMoviesTmp[count++] = reqMovies[i+1 ][0];
    }

  }// end for(i)

  System.out.println("count:␣"+count);

  uniqueMovies = new String[count];

  for(int i = 0; i < uniqueMovies.length; i++){
    uniqueMovies[i] = uniqueMoviesTmp[i];
  }



  return uniqueMovies;

}// end getUniqueCinemaNamesFromSorted()



/**
 * Convert the selected show date under a different format
 *
 * @return Show date under a different format
 */
public static String parseDate(String date){

  String tmpDate[];
```

```
String retDate = "";
String month  = "";

tmpDate = GUIHelper.tokenizeString(date);

if (tmpDate[1].toLowerCase().trim().equals("Jan".toLowerCase())){
  month = "1";
}

if (tmpDate[1].toLowerCase().trim().equals("Feb".toLowerCase())){
  month = "2";
}
if (tmpDate[1].toLowerCase().trim().equals("Mar".toLowerCase())){
  month = "3";
}

if (tmpDate[1].toLowerCase().trim().equals("Apr".toLowerCase())){
  month = "4";
}
if (tmpDate[1].toLowerCase().trim().equals("May".toLowerCase())){
  month = "5";
}

if (tmpDate[1].toLowerCase().trim().equals("Jun".toLowerCase())){
  month = "6";
}
if (tmpDate[1].toLowerCase().trim().equals("Jul".toLowerCase())){
  month = "7";
}

if (tmpDate[1].toLowerCase().trim().equals("Aug".toLowerCase())){
  month = "8";
}
if (tmpDate[1].toLowerCase().trim().equals("Sep".toLowerCase())){
  month = "9";
}

if (tmpDate[1].toLowerCase().trim().equals("Oct".toLowerCase())){
  month = "10";
}
if (tmpDate[1].toLowerCase().trim().equals("Nov".toLowerCase())){
  month = "11";
}

if (tmpDate[1].toLowerCase().trim().equals("Dec".toLowerCase())){
  month = "12";
}
```

```
    retDate = tmpDate[5] + "." + month + "." +tmpDate[2];

    return retDate;

  }// end parseDate()


}// end class

package gui.purchasetickets.step1moviesearch;


import java.io.IOException;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.DateField;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.ImageItem;
import javax.microedition.lcdui.Item;
import javax.microedition.lcdui.TextField;

import start.Start;


import model.beans.requestbeans.Find_Movies_Req_Bean;
import networkoperations.SendMessage;

import gui.GenericGUI;
import gui.customdialogwindows.CanvasAlert;
import gui.customdialogwindows.DialogWindow;
import gui.mainmenu.MenuScreen;

import constants.CustomAlertTypes;
import constants.Protocol_Step_Constants;

/**
 * Construct the search movie GUI
 *
 * @author Mihai Balan, s031288
 *
 */
public class SearchMoviesGUI extends GenericGUI{

  private static Displayable screen = null;

  //  the commands
```

```
private static Command searchCommand;
private static Command mainCommand;
private static Command helpCommand;
private static Command exitCommand;

// text boxes for UI
private TextField movie;
private TextField street;
private TextField city;
private TextField zip;
private TextField range;
private DateField date;


private Image     imgUp;
private ImageItem imgThemeUp;


/**
 * Constructs an instance of the class
 */
public SearchMoviesGUI() {
}

/**
 * Returns the displayable authentication screen
 * @return screen Returns the SearchMovie screen
 */
public Displayable getScreen() {
  return screen;
}

/**
 * Function of the chosen command
 * the user performs the search against the server side,
 * go to the main menu, displays help or exit
 *
 * @param c The executed command
 * @param s The main menu form
 *
 */
public void commandAction(Command c, Displayable s) {

  CanvasAlert help;

  try {
```

```
if (c == searchCommand) {

  // get the data from the UI and create the
  // Find_Movies_Req_Bean
  Find_Movies_Req_Bean findMovReqBean = new Find_Movies_Req_Bean();
  findMovReqBean.setMovie (movie.getString());
  findMovReqBean.setStreet(street.getString());
  findMovReqBean.setCity (city.getString());
  findMovReqBean.setZip (zip.getString());
  findMovReqBean.setRange (range.getString());
  findMovReqBean.setDate (MovieSearchHelper.parseDate(date.getDate
      ().toString()));

  // sends the request over the network and render the response
  SendMessage sm = new SendMessage(
      display,
      Protocol_Step_Constants.PRT_STEP_FIND_MOVIES,
      getScreen(),
      findMovReqBean);

  sm.go();


  Runtime runtime = Runtime.getRuntime();
  long t = runtime.freeMemory();
  System.out.println("**********************Memery␣before␣search␣
      movie:" + t);
  findMovReqBean = null;
  clean();
  long t1 = runtime.freeMemory();
  System.out.println("**********************Memery␣after␣search␣
      movie:" + t1);



}// end if (c == saveCommand)

if(c == mainCommand){
  display.setCurrent(new MenuScreen());
}// end if (c == mainCommand)

if (c == helpCommand){
  help = new CanvasAlert(
      display,
      getScreen(),
      "Movie␣Search␣Help",
      "It␣allows␣searching␣for␣a␣given␣movie,␣on␣a␣given␣day,␣in␣a␣
```

```
                given␣range␣from␣the␣keyed␣in␣position␣i.e.␣street␣&␣city
                !",
            "question",
            CustomAlertTypes.ALERT_INFO);

      }// end if (c == helpCommand)


      if (c == exitCommand){
        DialogWindow reallyExit = new DialogWindow(
            display,
            getScreen(),
            "Exit␣Application?",
            "Are␣you␣sure␣that␣you␣want␣to␣exit␣the␣application?",
            "question",
            "/dialogIcons/exitTheme",
            new MenuScreen().startingpoint);

        display.setCurrent(reallyExit);

      } // end if (c == exitCommand)

    }catch (Exception e) {
      e.printStackTrace();
      help = new CanvasAlert(
          display,
          getScreen(),
          "Movie␣Searching␣Error!",
          "Error␣in␣the␣Search␣Movies␣Screen!",
          "error",
          CustomAlertTypes.ALERT_ERROR);

      display.setCurrent(help);
    }
  }

  /**
   * Initialize the model an open the record store
   */
  protected void initModel() throws Exception {
  }

  /**
   * Creates the Authentication Screen
   *
   * @throws Exception
   *
```

```
 */
protected void createView() throws Exception {
  searchCommand = new Command("SEARCH", Command.EXIT, 0);
  mainCommand  = new Command("MAIN␣MENU", Command.SCREEN, 2);
  helpCommand  = new Command("HELP",    Command.SCREEN, 3);
  exitCommand  = new Command("EXIT",    Command.SCREEN, 4);

  movie  = new TextField("Movie␣Name:", "", 40, TextField.ANY);
  street = new TextField("Street␣Name:␣", "", 40, TextField.ANY);
  city   = new TextField("City␣Name:␣", "", 40, TextField.ANY);
  zip    = new TextField("Zip␣Code:␣", "", 8, TextField.NUMERIC);
  range  = new TextField("Range␣(km):␣", "", 2, TextField.NUMERIC);
  date   = new DateField("Show␣Date:␣", DateField.DATE);

  try{
    imgUp = Image.createImage("/" + Start.themeDir + "/movie_search/
        moviesTheme.png");
    imgThemeUp = new ImageItem("", imgUp, Item.LAYOUT_TOP | Item.
        LAYOUT_CENTER , "Theme_Img_Up");

  }catch(IOException ioe){
    System.out.println("Search␣for␣Movies␣Theme␣image␣exception!");
  }


  screen = new Form("Movie␣Search");
  ((Form)screen).append(imgUp);
  ((Form)screen).append(movie);
  ((Form)screen).append(street);
  ((Form)screen).append(city);
  ((Form)screen).append(zip);
  ((Form)screen).append(range);
  ((Form)screen).append(date);
  date.setDate(new java.util.Date());

  // add the commands to the form
  screen.addCommand(searchCommand);
  screen.addCommand(mainCommand);
  screen.addCommand(helpCommand);
  screen.addCommand(exitCommand);
}

/*
 * Update the view - maybe refresh the fields
 *
 */
protected void updateView() throws Exception {
```

```
    initModel();
    createView();
  }


  private void clean(){
    System.gc();
  }

}

package gui.purchasetickets.step2selectshow;

import gui.GenericGUI;
import gui.customdialogwindows.CanvasAlert;
import gui.customdialogwindows.DialogWindow;
import gui.mainmenu.MenuScreen;
import gui.purchasetickets.step1moviesearch.MovieSearchHelper;
import gui.purchasetickets.step1moviesearch.SearchMoviesGUI;
import gui.ratemovie.RateMovieGUI;

import java.io.IOException;

import javax.microedition.lcdui.*;

import start.Start;

import model.beans.requestbeans.Cinema_Hall_Conf_Req_Bean;
import model.beans.requestbeans.Find_Movies_Req_Bean;
import model.beans.requestbeans.Movie_Details_Req_Bean;
import model.beans.responsebeans.Cinema_Hall_Conf_Resp_Bean;
import model.beans.responsebeans.Find_Movies_Resp_Bean;
import model.beans.responsebeans.Movie_Details_Resp_Bean;
import networkoperations.NetworkCommunicationFacade;
import networkoperations.NetworkResposeFacade;
import networkoperations.SendMessage;


import constants.CustomAlertTypes;
import constants.Protocol_Step_Constants;
import constants.SystemConstants;


/**
 *  Select a show from the list returned by the server
 *  as a result of a movie search operation
 *
 *  It extends the GenericGUI super class
```

```
 *
 * @author s031288, Mihai Balan
 *
 */
public class SelectShowGUI extends GenericGUI implements
    ItemStateListener{

  private static Displayable screen = null;

  //  the commands
  private static Command selectCommand;
  private static Command backCommand;
  private static Command movieCommand;
  private static Command rateCommand;
  private static Command mainCommand;
  private static Command helpCommand;
  private static Command exitCommand;

  // UI components
  private ChoiceGroup cinemas;
  private ChoiceGroup movies;
  private ChoiceGroup showHours;
  private StringItem cinemaInfo;
  private Image     imgUp;
  private ImageItem imgThemeUp;

  private String[] moviesValue  = {};
  private String[] cinemasValue = {};
  private String[] showHoursValue = {};
  private String[] showInfo     = {};
  private String cinemaInfoValue = "";

  private Find_Movies_Resp_Bean findMoviesRespBean;
  private Find_Movies_Req_Bean findMovReqBean;
  private Cinema_Hall_Conf_Req_Bean cinemaHallConfReqBean;
  private Movie_Details_Req_Bean movDetailsReqBean;

  private String[][] reqMovies;

  private CanvasAlert alert;


  /**
   * Constructs an instance of the class
   */
  public SelectShowGUI(){
  }
```

```
public SelectShowGUI (
    Find_Movies_Req_Bean findMovReqBean,
    Find_Movies_Resp_Bean findMoviesRespBean) {

  System.out.println("-----␣IN␣select␣SHOW␣GUI");

  this.findMoviesRespBean = findMoviesRespBean;
  this.findMovReqBean   = findMovReqBean;
}

/**
 * Returns the displayable authentication screen
 * @return screen Returns the SelectShow screen
 */
public Displayable getScreen() {
  return screen;
}

/**
 * Function of the chosen command
 * the user view movie details, choose to select book for the selected
     movie,
 * go back to the search movie screen, main menu, displays help or exit
 *
 * @param c The executed command
 * @param s The main menu form
 *
 */
public void commandAction(Command c, Displayable s) {
  try {

    // get selected cinema, movie, show hour, showLocationID,
        showTimeID
    boolean[] moviesSelectedTmp = new boolean[movies.size()];
    movies.getSelectedFlags(moviesSelectedTmp);
    int selectedMovie = movies.getSelectedIndex();

    boolean[] cinemasSelectedTmp = new boolean[cinemas.size()];
    cinemas.getSelectedFlags(cinemasSelectedTmp);
    int selectedCinema = cinemas.getSelectedIndex();

    boolean[] showHourSelectedTmp = new boolean[showHours.size()];
    showHours.getSelectedFlags(showHourSelectedTmp);
    int selectedShowHour = showHours.getSelectedIndex();
```

```
if (c == selectCommand) {

  // if UI is valid i.e. an element in each choice group is seleted
  // make a request to get the cinema hall configuration from the
      server side
  // Else, display error message
  if(selectedCinema== -1 || selectedMovie == -1 || selectedShowHour
       == -1){

    alert = new CanvasAlert(
        display,
        getScreen(),
        "Invalid␣UI␣entries!",
        "Please␣check␣that␣you␣have␣selected␣the␣movie,␣cinema␣and␣
            show␣hour!",
        "error",
        CustomAlertTypes.ALERT_WARNING);

  }else{

    // protection against saving more than 10 ticketrs in the memory
    if(Start.maxTTSaved < SystemConstants.MAX_NO_TICKETS){
      cinemaHallConfReqBean = new Cinema_Hall_Conf_Req_Bean();
      cinemaHallConfReqBean.setShowLocationID(Integer.parseInt(
          showInfo[5]));
      cinemaHallConfReqBean.setShowTimeID (Integer.parseInt(showInfo
          [6]));

      System.out.println(cinemaHallConfReqBean.toString());

      SendMessage sm = new SendMessage(
          display,
          Protocol_Step_Constants.
              PRT_STEP_SELECT_SHOW_AND_DISPLAY_CINEMA_HALL_CONF,
          getScreen(),
          cinemaHallConfReqBean);

      sm.setShowInfo(showInfo);
      sm.go();

      Runtime runtime = Runtime.getRuntime();
      long t = runtime.freeMemory();
      System.out.println("**********************Memery␣before␣
          select␣show␣gui␣-␣hall␣conf:" + t);
      miniClean();
      long t1 = runtime.freeMemory();
      System.out.println("**********************Memery␣after␣
```

```
            select␣show␣gui␣-␣hall␣conf:" + t1);


      }else{

        alert = new CanvasAlert(
            display,
            getScreen(),
            "Max␣no␣of␣tickets␣reached!",
            "The␣maximum␣no␣of␣tickets␣that␣you␣can␣save␣in␣the␣memory
                ␣is␣10.␣Please␣buy␣the␣comercial␣version␣for␣unlimited
                ␣tickets!",
            "error",
            CustomAlertTypes.ALERT_WARNING);

      }// end if( SystemConstants.MAX_NO_TICKETS)

    }// end if (UI VALID)

  }// end if (c == selectCommand)


  if(c == movieCommand){
    if(selectedCinema== -1 || selectedMovie == -1 || selectedShowHour
        == -1){
      alert = new CanvasAlert(
          display,
          getScreen(),
          "Invalid␣UI␣entries!",
          "Please␣check␣that␣you␣have␣selected␣the␣movie,␣cinema␣and␣
              show␣hour!",
          "error",
          CustomAlertTypes.ALERT_WARNING);

    }else{

      movDetailsReqBean = new Movie_Details_Req_Bean();
      movDetailsReqBean.setShoLocationID(Integer.parseInt(showInfo[5])
          );

      SendMessage sm = new SendMessage(display,
          Protocol_Step_Constants.PRT_STEP_MOVIE_DETAILS, getScreen()
          , movDetailsReqBean);
      sm.go();

      Runtime runtime = Runtime.getRuntime();
      long t = runtime.freeMemory();
```

```
    System.out.println("************************Memery␣before␣select
        ␣show␣gui␣-␣mov␣det:" + t);
    miniClean();
    long t1 = runtime.freeMemory();
    System.out.println("************************Memery␣after␣select␣
        show␣gui␣-␣mov␣det:" + t1);

  }// end if (c == movieCommand)
}

if(c == rateCommand){
  if(selectedCinema== -1 || selectedMovie == -1 || selectedShowHour
      == -1){
    alert = new CanvasAlert(
        display,
        getScreen(),
        "Invalid␣UI␣entries!",
        "Please␣check␣that␣you␣have␣selected␣the␣movie,␣cinema␣and␣
            show␣hour!",
        "error",
        CustomAlertTypes.ALERT_WARNING);

  }else{

    RateMovieGUI rateMovieGUI = new RateMovieGUI(display, getScreen
        (), Integer.parseInt(showInfo[5]));
    Runtime runtime = Runtime.getRuntime();
    long t = runtime.freeMemory();
    System.out.println("************************Memery␣before␣select
        ␣show␣gui␣-␣rate␣mov:" + t);
    miniClean();
    long t1 = runtime.freeMemory();
    System.out.println("************************Memery␣after␣select␣
        show␣gui␣-␣rate␣mov:" + t1);

  }// end if (c == UI VALID)

}// end if (c == rateCommand)


if(c == backCommand){

  Runtime runtime = Runtime.getRuntime();
  long t = runtime.freeMemory();
  System.out.println("************************Memery␣before␣select␣
      show␣gui:" + t);
  clean();
```

```
      long t1 = runtime.freeMemory();
      System.out.println("************************Memery␣after␣select␣
          show␣gui:" + t1);


      display.setCurrent(new SearchMoviesGUI().getScreen());
    }// end if (c == backCommand)

    if(c == mainCommand){
      display.setCurrent(new MenuScreen());
    }// end if (c == mainCommand)


    if (c == helpCommand){
      alert = new CanvasAlert(
          display,
          new SelectShowGUI().getScreen(),
          "Select␣Show␣Help",
          "Allows␣to␣select␣a␣show␣and␣book␣tickets␣later␣on␣for␣this␣
              show,␣or␣view␣details␣about␣the␣selected␣movie!",
          "question",
          CustomAlertTypes.ALERT_INFO);

    }// end if (c == helpCommand)


    if (c == exitCommand){
      DialogWindow reallyExit = new DialogWindow(
          display,
          new SelectShowGUI().getScreen(),
          "Exit␣Application?",
          "Are␣you␣sure␣that␣you␣want␣to␣exit␣the␣application?",
          "question",
          "/dialogIcons/exitTheme",
          new MenuScreen().startingpoint);

      display.setCurrent(reallyExit);

    } // end if (c == exitCommand)

  } catch (Exception e) {
    e.printStackTrace();
    alert = new CanvasAlert(
        display,
        getScreen(),
        "Select␣Show␣Error!",
        "Error␣in␣the␣Select␣Show␣Screen!",
```

```
        "error",
        CustomAlertTypes.ALERT_ERROR);

    display.setCurrent(alert);
  }
}

/**
 * Initialize the model
 */
protected void initModel() throws Exception {

  System.out.println("-----␣IN␣INIT␣select␣SHOW␣GUI");
  reqMovies = findMoviesRespBean.getMovies();

  // populate the movie choice group component
  System.out.println("-----␣TRYING␣TO␣POPULATE␣select␣SHOW␣GUI");
  moviesValue = MovieSearchHelper.getUniqueValues(reqMovies,0);
  for(int i=0; i<moviesValue.length; i++)
    System.out.println("--␣movies:␣" + moviesValue[i]);

  System.out.println("-----␣AFTER␣POPULATING␣select␣SHOW␣GUI");


} //end initModel()

/**
 * Creates the Authentication Screen
 *
 * @throws Exception
 *
 */
protected void createView() throws Exception {

  System.out.println("-----␣BEFORE␣CREATING␣THE␣SHOW␣GUI␣VIEW");

  selectCommand = new Command("SELECT␣SEATS", Command.EXIT, 0);
  backCommand  = new Command("BACK",       Command.SCREEN, 2);
  movieCommand = new Command("MOVIE␣DETAILS", Command.SCREEN, 3);
  rateCommand  = new Command("RATE␣MOVIE", Command.SCREEN, 4);
  mainCommand  = new Command("MAIN␣MENU",  Command.SCREEN, 5);
  helpCommand  = new Command("HELP",       Command.SCREEN, 6);
  exitCommand  = new Command("EXIT",       Command.SCREEN, 7);

  movies    = new ChoiceGroup("Movies:",   Choice.POPUP, moviesValue,
      null);
  cinemas   = new ChoiceGroup("Cinemas:",  Choice.POPUP, cinemasValue,
```

```
    null);
showHours = new ChoiceGroup("Show␣Hours:", Choice.POPUP,
    showHoursValue, null);
cinemaInfo = new StringItem("Cinema␣Info:", cinemaInfoValue, Item.
    PLAIN);

try{
  imgUp = Image.createImage("/" + Start.themeDir + "/movie_search/
      moviesTheme.png");
  imgThemeUp = new ImageItem("", imgUp, Item.LAYOUT_TOP | Item.
      LAYOUT_CENTER , "Theme_Img_Up");

}catch(IOException ioe){
  System.out.println("Select␣Movie␣image␣exception!");
}

screen = new Form("Select␣Movie");
((Form)screen).append(imgUp);
((Form)screen).append(movies);
((Form)screen).append(cinemas);
((Form)screen).append(showHours);
((Form)screen).append(cinemaInfo);

// add the commands to the form
screen.addCommand(selectCommand);
screen.addCommand(backCommand);
screen.addCommand(movieCommand);
screen.addCommand(rateCommand);
screen.addCommand(mainCommand);
screen.addCommand(helpCommand);
screen.addCommand(exitCommand);

// populate all UI componenets with default data
cinemasValue  = UpdateSelectMovieScreen.updateCinemaCG (moviesValue
    [0], cinemasValue, cinemas, showHours, cinemaInfo, reqMovies);
System.out.println("-----␣AFTER␣updating␣cinema␣values");
showHoursValue = UpdateSelectMovieScreen.updateShowHourCG(moviesValue
    [0], cinemasValue[0], showHoursValue, showHours, cinemaInfo,
    reqMovies);
System.out.println("-----␣AFTER␣updating␣show␣hour␣values");
showInfo      = UpdateSelectMovieScreen.updateCinemaInfo(moviesValue
    [0], cinemasValue[0], showHoursValue[0], findMovReqBean.getDate()
    , cinemaInfo, reqMovies);
System.out.println("-----␣AFTER␣updating␣SHOW␣info␣values");
((Form)screen).setItemStateListener(this);

System.out.println("-----␣AFTER␣CREATING␣THE␣SHOW␣GUI␣VIEW");
```

```
}

/*
 * Update the view - maybe refresh the fields
 *
 */
protected void updateView() throws Exception {
  initModel();
  createView();
}

/**
 *
 */
public void itemStateChanged(Item item){

  if (item.getLabel().equals("Movies:")){
    boolean[] movieSelected = new boolean[movies.size()];
    movies.getSelectedFlags(movieSelected);
    int selectedMov = movies.getSelectedIndex();

    // populate the cinema, show hour combo boxes and show info text
        field based on the selected movie name
    cinemasValue  = UpdateSelectMovieScreen.updateCinemaCG(moviesValue[
        selectedMov], cinemasValue, cinemas, showHours, cinemaInfo,
        reqMovies);
    showHoursValue = UpdateSelectMovieScreen.updateShowHourCG(
        moviesValue[selectedMov], cinemasValue[0], showHoursValue,
        showHours, cinemaInfo, reqMovies);
    showInfo      = UpdateSelectMovieScreen.updateCinemaInfo(moviesValue
        [selectedMov], cinemasValue[0], showHoursValue[0],
        findMovReqBean.getDate(), cinemaInfo, reqMovies);

  }// end if (item.getLabel().equals("Movies:"))

  if (item.getLabel().equals("Cinemas:")){
    boolean[] movieSelected = new boolean[movies.size()];
    movies.getSelectedFlags(movieSelected);
    int selectedMov = movies.getSelectedIndex();

    boolean[] cinemaSelected = new boolean[cinemas.size()];
    cinemas.getSelectedFlags(cinemaSelected);
    int selectedCin = cinemas.getSelectedIndex();

    // populate the shoHours and combo box and show infor text field
        based on the selected movie name and cinema
```

```
    showHoursValue = UpdateSelectMovieScreen.updateShowHourCG(
        moviesValue[selectedMov], cinemasValue[selectedCin],
        showHoursValue, showHours, cinemaInfo, reqMovies);
    showInfo      = UpdateSelectMovieScreen.updateCinemaInfo(moviesValue
        [selectedMov], cinemasValue[selectedCin], showHoursValue[0],
        findMovReqBean.getDate(), cinemaInfo, reqMovies);

  }// end if (item.getLabel().equals("Cinemas:"))



  if (item.getLabel().equals("Show␣Hours:")){
    boolean[] movieSelected = new boolean[movies.size()];
    movies.getSelectedFlags(movieSelected);
    int selectedMov = movies.getSelectedIndex();

    boolean[] cinemaSelected = new boolean[cinemas.size()];
    cinemas.getSelectedFlags(cinemaSelected);
    int selectedCin = cinemas.getSelectedIndex();

    boolean[] hourSelected = new boolean[showHours.size()];
    showHours.getSelectedFlags(hourSelected);
    int selectedHour = showHours.getSelectedIndex();

    showInfo = UpdateSelectMovieScreen.updateCinemaInfo(moviesValue[
        selectedMov], cinemasValue[selectedCin], showHoursValue[
        selectedHour], findMovReqBean.getDate(), cinemaInfo, reqMovies)
        ;

  }// end if (item.getLabel().equals("Show Hours:"))

}// end itemStateChanged()

private void clean(){
  findMoviesRespBean = null;
  findMovReqBean = null;
  cinemaHallConfReqBean = null;
  movDetailsReqBean = null;
  reqMovies = null;
  cinemas = null;
  movies = null;
  showHours = null;
  cinemaInfo = null;
  imgUp = null;
  imgThemeUp = null;

  moviesValue    = null;
```

```
    cinemasValue    = null;
    showHoursValue  = null;
    showInfo        = null;
    cinemaInfoValue = null;
    alert = null;
    System.gc();
  }

  private void miniClean(){
    cinemaHallConfReqBean = null;
    movDetailsReqBean = null;
    alert = null;
    System.gc();
  }

}// end class

package gui.purchasetickets.step2selectshow;

import gui.purchasetickets.step1moviesearch.MovieSearchHelper;

import javax.microedition.lcdui.ChoiceGroup;
import javax.microedition.lcdui.StringItem;

/**
 * Updates the UI components from the select show UI
 * function of the selected elements in the choice groups
 *
 * @author Mihai Balan, s031288
 *
 */
public class UpdateSelectMovieScreen {

  /**
   * Update the cinema choice group every time when a new
   * value is selected from the movie choise group.
   * The choice group is populated with the unique names
   * of the cinema playing the selected movie.
   *
   * @param cinemas The cinema choice group
   */
  public static String[] updateCinemaCG(
      String selectedMovie, String[] cinemasValue,
      ChoiceGroup cinemas, ChoiceGroup showHours,
      StringItem cinemaInfo, String[][] reqMovies){

    // remove the duplicates from the cinema list
    cinemasValue = MovieSearchHelper.getUniqueValues(
```

```
      UpdateSelectMovieScreen.getCinemas(selectedMovie, reqMovies)
          );

  cinemas.deleteAll();
  showHours.deleteAll();
  cinemaInfo.setText("");

  // populate the choice group with the unique elements
  for(int i = 0; i < cinemasValue.length; i++){
    cinemas.append(cinemasValue[i], null);
  }

  return cinemasValue;

}// end updateCinemaCG()


/**
 *
 *
 * @param showHours The show hour choice group
 */
public static String[] updateShowHourCG(
    String selectedMovie,
    String selectedCinema,
    String[] showHoursValue,
    ChoiceGroup showHours,
    StringItem cinemaInfo,
    String[][] reqMovies){

  // remove the duplicates from the show hour list and populate
  // the show hour combo box with the unique elements
  showHoursValue = MovieSearchHelper.getUniqueValues(
      UpdateSelectMovieScreen.getShowHours(selectedMovie,
          selectedCinema, reqMovies)
      );

  showHours.deleteAll();
  cinemaInfo.setText("");

  // populate the choice group with the unique elements
  for(int i = 0; i < showHoursValue.length; i++){
    showHours.append(showHoursValue[i], null);
  }

  return showHoursValue;
```

```
}// end updateShowHourCG()


/**
 *
 *
 * @param cg The show hour choice group
 */
public static String[] updateCinemaInfo(
    String selectedMovie,
    String selectedCinema,
    String selectedShowHour,
    String showDate,
    StringItem cinemaInfo,
    String[][] reqMovies){

  String[] showInfo = UpdateSelectMovieScreen.getShowInfo(selectedMovie
      , selectedCinema, selectedShowHour, showDate, reqMovies);
  cinemaInfo.setText("Movie:␣" + selectedMovie + ",␣played␣in:␣" +
      selectedCinema + "(" + showInfo[4] + "),␣on:␣"+ showDate + ",␣at
      :␣" + selectedShowHour);
  //cinemaInfo.setText("Movie: " + selectedMovie + ", played in: " +
      selectedCinema + "(" + showInfo[4] + "), on: "+ "date" + ", at
      : " + selectedShowHour);

  return showInfo;


}// end updateCinemaInfo()


/**
 * Returns an array of all cinemas where the given movie is played
 *
 *
 * @param movie The selected movie
 * @param reqMovies The bi dim movie array contained by the Find Movie
     Res Bean
 * @return The list of all cinemas that play the given movie
 */
public static String[] getCinemas(String movie, String[][] reqMovies){

  int index = 0;
  String[] tmpCinemas = new String[reqMovies.length];
  String[] cinemas;

  // finds all cinemas that play the selected movie
```

```
  for(int i = 0; i < reqMovies.length; i++){
    if(reqMovies[i][0].trim().equals(movie)){
      tmpCinemas[index++] = reqMovies[i][2];
    }

  }// end for()

  cinemas = new String[index];
  for(int i = 0; i < index; i++){
    cinemas[i] = tmpCinemas[i];
  }// end for()

  return cinemas;

}// end getCinemas()


/**
 * Returns an array of all show hours
 * for the slected movie played in the selected cinema
 *
 * @param movie The selected movie
 * @param cinema The selected cinema
 * @param reqMovies The bi dim movie array contained by the Find Movie
     Res Bean
 * @return The list of all show hours that play the given movie in the
     given cinema
 */
public static String[] getShowHours(String movie, String cinema, String
    [][] reqMovies){

  int index = 0;
  String[] tmpShowHours = new String[reqMovies.length];
  String[] showHours;

  // finds all show hours for the selected movie in the selected cinema
  for(int i = 0; i < reqMovies.length; i++){
    if((reqMovies[i][0].trim().equals(movie)) && (reqMovies[i][2].trim
        ().equals(cinema))){
      tmpShowHours[index++] = reqMovies[i][1];
    }

  }// end for()

  showHours = new String[index];
  for(int i = 0; i < index; i++){
    showHours[i] = tmpShowHours[i];
```

```
  }// end for()

  return showHours;

}// end getShowHours()


/**
 *
 * @param movie  The selected movie
 * @param cinema The selected cinema
 * @param showHour The selected show hour
 * @param reqMovies The bi dim movie array contained by the Find Movie
     Res Bean
 * @return The cinema info i.e. cinema address, showLocationID,
     showTimeID
 *        for the given selected in the selected cinema played at the
     selected show hour
 */
public static String[] getShowInfo(String movie, String cinema, String
    showHour, String showDate, String[][] reqMovies){

  String[] showInfo = new String[7];

  String address      = "";
  String showLocationID = "";
  String showTimeID   = "";

  // finds the cinema address for the show matching the input criterita
  for(int i = 0; i < reqMovies.length; i++){
    if((reqMovies[i][0].trim().equals(movie))
        && (reqMovies[i][1].trim().equals(showHour))
          && (reqMovies[i][2].trim().equals(cinema))){

      address       = reqMovies[i][4] + ",␣" + reqMovies[i][3];
      showLocationID = reqMovies[i][5];
      showTimeID    = reqMovies[i][6];
    }

  }// end for()

  showInfo[0] = movie;
  showInfo[1] = cinema;
  showInfo[2] = showDate;
  //showInfo[2] = MovieSearchHelper.parseDate(showDate);
  showInfo[3] = showHour;
```

```
    showInfo[4] = address;
    showInfo[5] = showLocationID;
    showInfo[6] = showTimeID;

    return showInfo;

  }// end getCinemaInfo()

}// end class

package gui.purchasetickets.step3selectseats;

import javax.microedition.lcdui.Canvas;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Font;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.Image;

import constants.CustomAlertTypes;
import constants.Protocol_Step_Constants;
import constants.SystemConstants;

import model.beans.otherbeans.TicketBean;
import model.beans.requestbeans.Cinema_Hall_Conf_Req_Bean;
import model.beans.requestbeans.Select_Deselect_Seats_Req_Bean;
import model.beans.responsebeans.Cinema_Hall_Conf_Resp_Bean;
import networkoperations.SendMessage;

import start.Start;
import gui.GUIHelper;
import gui.customdialogwindows.CanvasAlert;



public class SelectSeatsGUI extends Canvas {

  // the display to draw on
  private Display   display;
  private Displayable next;
  private Graphics g;

  private TicketBean                 genericTicket;
  private Cinema_Hall_Conf_Req_Bean  cineHallConfReqBean;
  private Cinema_Hall_Conf_Resp_Bean cineHallConfRespBean;
  private Select_Deselect_Seats_Req_Bean selDeselectSeatsReqBean;

  private Start midlet;
```

```
// the curent selected option i.e. RESERVE, BACK
private int selectedOptionIndex = 1;

// NOT highlighted buttons
private String[] optionDeselected = {
    "/MovieDetailsButtons/newBackDeselected.png",
    "/MovieDetailsButtons/newReserveDeselected.png"
};

// the highlighted buttons
private String[] optionSelected = {
    "/MovieDetailsButtons/newBackSelected.png",
    "/MovieDetailsButtons/newReserveSelected.png"
};

//  define the msg font
private Font msgFontBoldSmall = Font.getFont(
    Font.FACE_PROPORTIONAL,
    Font.STYLE_BOLD,
    Font.SIZE_SMALL);

// the images for building the YES and NO options
private Image[] deselectedImgs;
private Image[] selectedImgs;

private int startX   = 5;
private int startY   = 5;
private int spacing  = 5;
private int rectWidth = msgFontBoldSmall.getHeight();
private int rectHeight = msgFontBoldSmall.getHeight();
private int w = getWidth();
private int h = getHeight();
private int seatSize = 0;
private int buttonY = 35;
private int[][] seatMatrix;
private int startPosX = 0;
private int startPosY = 0;
private int rowNo = 0;
private int colNo = 0;

private CanvasAlert warnMsg;


public SelectSeatsGUI(
    Display                 display,
    Displayable             next,
```

```
    TicketBean              genericTicket,
    Cinema_Hall_Conf_Req_Bean cineHallConfReqBean,
    Cinema_Hall_Conf_Resp_Bean cineHallConfRespBean){

  this.display           = display;
  this.next              = next;
  this.genericTicket     = genericTicket;
  this.cineHallConfRespBean = cineHallConfRespBean;
  this.cineHallConfReqBean = cineHallConfReqBean;

  selectedImgs  = GUIHelper.createSelectedButtons(optionSelected);
  deselectedImgs = GUIHelper.createDeselectedButtons(optionDeselected);

  seatMatrix = createSeatMatrix();

  display.setCurrent(this);

}// end constructor()


/**
 * Construct and dipsplay the cinema hall
 *
 * @param g The graphics to draw on
 */
protected void paint(Graphics g){


  resetCoordinates();

  try {
    // clear the background
    g.setColor(255, 255, 255);
    g.fillRect(0, 0, getWidth(), getHeight());

    // draw the legend
    startY = drawLegend(g);

    // draw the screen
    startY = drawScreen(g);
    startY += msgFontBoldSmall.getHeight();

    // draw the cinema hall configuration
    drawSeats(g);

    // draw select Message
    drawSelectMessage(g);
```

```
    // draw the buttons
    GUIHelper.drawCCViewButtons(g,
        deselectedImgs, selectedImgs,
        w, h,
        w/2, h - buttonY,
        selectedOptionIndex);

  } catch (Exception e) {
    System.out.println("Cinema hall Configuration exception!");
    g.drawString("Cinema hall Configuration Exception", w/2, h/2,
        Graphics.BASELINE | Graphics.HCENTER);
  }

}// end paint()


/**
 * Update the value of the current selected option in the list
 * and repaint the screen.
 *
 * @param keyCode The code of the pressed key
 */
protected void keyPressed( int keyCode ){

  if ((getGameAction(keyCode) == Canvas.LEFT)){

    if(selectedOptionIndex > 0){

      selectedOptionIndex--;
      resetCoordinates();
      repaint();

    } else if(selectedOptionIndex == 0){

      selectedOptionIndex = optionDeselected.length - 1;
      resetCoordinates();
      repaint();
    }

  } else if ((getGameAction(keyCode) == Canvas.RIGHT)){

    if(selectedOptionIndex < optionDeselected.length - 1){

      selectedOptionIndex++;
      resetCoordinates();
      repaint();
```

```
    } else if(selectedOptionIndex == optionDeselected.length - 1){

      selectedOptionIndex = 0;
      resetCoordinates();
      repaint();
    }

  } else if ((getGameAction(keyCode) == Canvas.FIRE)){

    // reserved button pressed
    if(selectedOptionIndex == 1){
      int[][] reservedSeats = getReservedSeats();

      // in case the reserve button is pressed
      // and the no of tickets saved in the phone memory plsu
      // the current ones do not exceed the allowed limit
      // make the request to the network
      if(((Start.maxTTSaved + reservedSeats.length) <= SystemConstants.
          MAX_NO_TICKETS)
         && (reservedSeats.length > 0)){

        selDeselectSeatsReqBean = new Select_Deselect_Seats_Req_Bean();
        selDeselectSeatsReqBean.setCommmand(1);
        selDeselectSeatsReqBean.setShowLocationID(cineHallConfReqBean.
            getShowLocationID());
        selDeselectSeatsReqBean.setShowTimeID(cineHallConfReqBean.
            getShowTimeID());
        selDeselectSeatsReqBean.setSeats(reservedSeats);
        selDeselectSeatsReqBean.setSeatsNoRows(reservedSeats.length);
        selDeselectSeatsReqBean.setSeatsNoCols(reservedSeats[0].length);

        System.out.println(selDeselectSeatsReqBean.toString());

        try{
          SendMessage sm = new SendMessage(display,
              Protocol_Step_Constants.PRT_STEP_SELECT_DESELECT_SEATS,
              display.getCurrent(), selDeselectSeatsReqBean);
          sm.setNextScreenAfterSeatSelectionConfParams(
               reservedSeats,
               genericTicket,
               cineHallConfReqBean,
               cineHallConfRespBean,
               selDeselectSeatsReqBean);
          sm.go();

        }catch(Exception e){
```

```
          warnMsg = new CanvasAlert(
              display,
              display.getCurrent(),
              "Seat␣selection␣network␣error!",
              "Exception␣while␣sending␣the␣select␣seats␣request␣over␣the
                  ␣network!",
              "warn",
              CustomAlertTypes.ALERT_WARNING);

          e.printStackTrace();

        }// end try-catch()

      }else if((Start.maxTTSaved + reservedSeats.length) >
          SystemConstants.MAX_NO_TICKETS){

        warnMsg = new CanvasAlert(
            display,
            display.getCurrent(),
            "Ticket␣limit␣exceeded!",
            "You␣are␣allowed␣to␣store␣max.␣" + SystemConstants.
                MAX_NO_TICKETS + "␣tickets␣in␣the␣memory!␣"+
            "You␣have␣" + Start.maxTTSaved + "␣tickets␣till␣now!",
            "warn",
            CustomAlertTypes.ALERT_WARNING);

      }else if(reservedSeats.length > 0){

        warnMsg = new CanvasAlert(
            display,
            display.getCurrent(),
            "No␣seats␣selected!",
            "Please␣choose␣a␣seat␣before␣trying␣to␣reserv␣it!",
            "warn",
            CustomAlertTypes.ALERT_WARNING);

      }// end if(reservedSeats.length > 0)

    // go back to Select show screen
    }else if(selectedOptionIndex == 0){
      if(midlet == null)
        display.setCurrent(next);
    }

  }else if (keyCode == Canvas.KEY_NUM0){
    // if seat selected - deselect it
    // if seat is not selected then select it
```

```
    if(seatMatrix[rowNo][colNo] == 2)
      seatMatrix[rowNo][colNo] = 0;
    else if(seatMatrix[rowNo][colNo] == 0)
      seatMatrix[rowNo][colNo] = 2;

  } else if (keyCode == Canvas.KEY_NUM1){
    if(colNo == 0)
      colNo = cineHallConfRespBean.getCols()-1;
    else
      --colNo;

    repaint();

  }else if (keyCode == Canvas.KEY_NUM3){
    if(colNo == cineHallConfRespBean.getCols()-1)
      colNo = 0;
    else
      ++colNo;

    repaint();

  }else if (keyCode == Canvas.KEY_NUM2){
    if(rowNo == 0)
      rowNo = cineHallConfRespBean.getRows()-1;
    else
      --rowNo;

    repaint();

  }else if (keyCode == Canvas.KEY_NUM8){
    if(rowNo == cineHallConfRespBean.getRows()-1)
      rowNo = 0;
    else
      ++rowNo;

    repaint();

  }// end if Key

}// end keyPressed


/**
 * Create the seat matrix to be displayed on the screen
 * @return
 */
private int[][] createSeatMatrix(){
```

```
  // mtarix seat populated witz 0 for free seat and 1 for booked seat
  int[][] seatMatrixTmp = new int[cineHallConfRespBean.getRows()][
      cineHallConfRespBean.getCols()];

  // hold the booked seats
  int[][] seatMatrixBooked = cineHallConfRespBean.getAllBookedSeats();

  // initialize the seat matrix with 0
  for(int rows = 0; rows < cineHallConfRespBean.getRows(); rows++){
    for(int cols = 0; cols < cineHallConfRespBean.getCols(); cols++) {
      seatMatrixTmp[rows][cols] = 0;
    }
  }

  // mark in the seat matrix the booked seats with 1
  for(int i = 0; i < cineHallConfRespBean.getAllBookedSeatsRows(); i++)
      {
    seatMatrixTmp[seatMatrixBooked[i][0] - 1][seatMatrixBooked[i
        ][1] - 1] = 1;
  }

  return seatMatrixTmp;

}// end creatSeatMatrix()


/**
 * Draw the seat legend on the screen
 *
 */
private int drawLegend(Graphics g){

  g.setColor(0, 0, 0);
  g.fillRect(0, 0, w, 2);

  g.setFont(msgFontBoldSmall);
  g.setColor(31, 191, 31);
  g.fillRoundRect(startX, startY, rectWidth, rectHeight, 1, 1);
  startX += rectWidth + spacing;
  g.drawString("FREE", startX, startY, Graphics.TOP | Graphics.LEFT);
  startX += msgFontBoldSmall.stringWidth("FREE") + spacing;

  g.setColor(228, 12, 12);
  g.fillRoundRect(startX, startY, rectWidth, rectHeight, 1, 1);
  startX += rectWidth + spacing;
  g.drawString("BOOKED", startX, startY, Graphics.TOP | Graphics.LEFT);
```

```
  startX += msgFontBoldSmall.stringWidth("BOOKED") + spacing;

  g.setColor(13, 22, 193);
  g.fillRoundRect(startX, startY, rectWidth, rectHeight, 1, 1);
  startX += rectWidth + spacing;
  g.drawString("YOUR␣SELECTION", startX, startY, Graphics.TOP |
      Graphics.LEFT);

  g.setColor(0, 0, 0);
  startY += msgFontBoldSmall.getHeight() + 3;
  g.fillRect(0, startY, w, 2);

  return startY;

}// drawLegend()


/**
 * Draw the whole screen
 *
 */
private int drawScreen(Graphics g){

  g.setColor(150, 0, 150);
  startY += 15;
  g.fillRect(w/4, startY, w/2, 7);
  g.drawString("SCREEN", 3*w/4 + 5, startY - 2, Graphics.TOP | Graphics
      .LEFT);

  return startY;

}// drawScreen()


/**
 * Draw the seats
 *
 */
private int drawSeats(Graphics g){
  int leftWidth = w - 2*spacing;
  int leftHeight = h - startY - 2*spacing - buttonY;

  int seatW = (leftWidth)/cineHallConfRespBean.getCols();
  int seatH = (leftHeight)/cineHallConfRespBean.getRows();

  if(seatW <= seatH)
    seatSize = seatW;
```

```
  else
    seatSize = seatH;

  int spacingX = (w - 2*spacing - cineHallConfRespBean.getCols()*
      seatSize)/2;

  int spacingY = (leftHeight - cineHallConfRespBean.getRows()*seatSize)
      /2;

  int startSeatX = (w - cineHallConfRespBean.getCols()*seatSize)/2;
  int startSeatY = startY + spacing + spacingY;

  startPosX = startSeatX;
  startPosY = startSeatY;

  for(int rows = 0; rows <cineHallConfRespBean.getRows(); rows++){
    for(int cols = 0; cols <cineHallConfRespBean.getCols(); cols++) {
      if(seatMatrix[rows][cols] == 0){
        // free seat
        g.setColor(31, 191, 31);
      }else if(seatMatrix[rows][cols] == 1){
        // booked seat
        g.setColor(228, 12, 12);
      }else if(seatMatrix[rows][cols] == 2){
        // your selection
        g.setColor(13, 22, 193);
    }
      g.fillRect(startSeatX + (seatSize * cols), startSeatY + (seatSize
          * rows), seatSize -2, seatSize -2);
    }
  }

  g.setColor(150, 0, 150);
  g.fillRect(0, h-buttonY + 6, w, 3);

  // draw selected seat
  g.setColor(13, 22, 193);
  g.fillRect(startPosX+ (seatSize * colNo), startPosY+ (seatSize *
      rowNo), seatSize -2, seatSize -2);

  return 0;

}//drawSeats()


/**
 * Draw navigation key message
```

```
 * @param g
 */
private void drawSelectMessage(Graphics g){
  g.setColor(13, 22, 193);
  g.drawString("Navigation␣keys:␣1,3,2,8␣|␣0␣-␣(de)select␣a␣seat!", w
      /2, h-buttonY + 3, Graphics.BASELINE | Graphics.HCENTER);

}// end drawSelectMessage()

/**
 * Reset the drawing coordinates when canvas is repainted
 *
 */
private void resetCoordinates(){
  startX = 5;
  startY = 5;

}// end resetCoordinates()


/**
 * Get the selected seats
 *
 */
private int[][] getReservedSeats(){

  int count = 0;

  // get the no of reserved seats
  for(int rows = 0; rows <cineHallConfRespBean.getRows(); rows++){
    for(int cols = 0; cols <cineHallConfRespBean.getCols(); cols++) {
      if(seatMatrix[rows][cols] == 2){
        ++count;
      }
    }
  }// end for(rows)

  // get the reserved seats
  int[][] reservedSeats = new int[count][2];
  count = 0;

  for(int rows = 0; rows <cineHallConfRespBean.getRows(); rows++){
    for(int cols = 0; cols <cineHallConfRespBean.getCols(); cols++) {
      if(seatMatrix[rows][cols] == 2){
        reservedSeats[count] [0] = rows + 1;
        reservedSeats[count++][1] = cols + 1;
      }
```

```
      }
    }// end for(rows)

    return reservedSeats;

  }// end getReservedSeats()


}// end class

package gui.purchasetickets.step4discountandreservationsummary;

import java.io.IOException;
import javax.microedition.lcdui.*;

import start.Start;

import networkoperations.SendMessage;

import model.beans.otherbeans.TicketBean;
import model.beans.requestbeans.Cinema_Hall_Conf_Req_Bean;
import model.beans.requestbeans.Reject_Payment_Req_Bean;
import model.beans.requestbeans.Select_Deselect_Seats_Req_Bean;
import model.beans.responsebeans.Cinema_Hall_Conf_Resp_Bean;
import model.beans.responsebeans.Select_Deselect_Seats_Resp_Bean;


import gui.GenericGUI;
import gui.customdialogwindows.CanvasAlert;
import gui.purchasetickets.step5chooseticketpayment.
    ChooseTicketPaymentGUI;

import constants.CustomAlertTypes;
import constants.Protocol_Step_Constants;


/**
 *  Set the ticket discount value for each ticket
 *
 *  It extends the GenericGUI super class
 *
 * @author s031288, Mihai Balan
 *
 */
public class TicketDiscountAndReservationSummaryGUI extends GenericGUI
    implements ItemStateListener{

  // the edit CC screen
```

```
private static Displayable screen = null;

//  the commands
private static Command acceptCommand;
private static Command mainCommand;
private static Command helpCommand;
private static Command exitCommand;

// UI components
private ChoiceGroup ticketsUI;
private ChoiceGroup discountsUI;
private StringItem[] ticketInfoUI;
private StringItem totalPriceUI;
private Image      imgUp;
private ImageItem  imgThemeUp;

private double[] ticketPrices;
private double totalPrice = 0.0;
private double ticketBasePrice;
private int[][] reservedSeats;
private double[] discountDoubleValues;
private String[] ticketsValue     = {};
private String[] ticketsDiscountValue = {};
private String[] ticketInfoItems  = {};
private String[] discountValue    = {"NONE", "CHILD", "STUDENT", "
    PENSIONER", "VOUCHER"};

private TicketBean                  genericTicket;
private TicketBean[]                cinemaTickets;
private Cinema_Hall_Conf_Req_Bean   cineHallConfReqBean;
private Cinema_Hall_Conf_Resp_Bean  cineHallConfRespBean;
private Select_Deselect_Seats_Req_Bean selDeselectSeatsReqBean;
private Select_Deselect_Seats_Resp_Bean selDeselectSeatsRespBean;


/**
 * Constructs an instance of the class
 */
public TicketDiscountAndReservationSummaryGUI(){
}


public TicketDiscountAndReservationSummaryGUI (
    int[][]                     reservedSeats,
    TicketBean                  genericTicket,
    Cinema_Hall_Conf_Req_Bean   cineHallConfReqBean,
    Cinema_Hall_Conf_Resp_Bean  cineHallConfRespBean,
```

```
    Select_Deselect_Seats_Req_Bean selDeselectSeatsReqBean,
    Select_Deselect_Seats_Resp_Bean selDeselectSeatsRespBean
) {

  this.reservedSeats         = reservedSeats;
  this.genericTicket         = genericTicket;
  this.discountDoubleValues  = cineHallConfRespBean.getDiscountValues()
      ;
  this.ticketBasePrice       = cineHallConfRespBean.getBasePrice();
  this.cineHallConfReqBean   = cineHallConfReqBean;
  this.cineHallConfRespBean  = cineHallConfRespBean;
  this.selDeselectSeatsReqBean = selDeselectSeatsReqBean;
  this.selDeselectSeatsRespBean = selDeselectSeatsRespBean;

}


/**
 * Returns the displayable authentication screen
 * @return screen Returns the SelectShow screen
 */
public Displayable getScreen() {
  return screen;
}

/**
 * Function of the chosen command
 * the user can skip the ticket discount info,
 * or select to purchase the tickets
 *
 * @param c The executed command
 * @param s The main menu form
 *
 */
public void commandAction(Command c, Displayable s) {
  try {

    if (c == acceptCommand){

      for(int i = 0; i < reservedSeats.length; i++){
        cinemaTickets[i].setTKTDiscountType(ticketsDiscountValue[i]);
        cinemaTickets[i].setTKTPrice(
            UpdateTicketDiscountAndReservationSummaryScreen.
            formatTotalPrice(ticketPrices[i]));
        cinemaTickets[i].setTKTRow (Integer.toString(reservedSeats[i
            ][0]));
        cinemaTickets[i].setTKTSeat(Integer.toString(reservedSeats[i
```

```
            ][1]));
      }

      display.setCurrent(new ChooseTicketPaymentGUI(
          totalPrice,
          getScreen(),
          ticketsDiscountValue,
          cinemaTickets,
          cineHallConfReqBean,
          cineHallConfRespBean,
          selDeselectSeatsReqBean,
          selDeselectSeatsRespBean).prepareScreen());

    }// end if (c == acceptCommand)


      if (c == mainCommand){
        rejectPayment("main");

      }// end if (c == mainCommand)

      if (c == helpCommand){
        CanvasAlert help = new CanvasAlert(
            display,
            display.getCurrent(),
            "Ticket␣Discount␣Help",
            "Allows␣to␣set␣a␣discount␣type␣for␣every␣reserved␣seat.␣NONE
                ␣is␣set␣by␣default.␣If␣one␣do␣not␣need␣this␣he␣can␣
                choose␣SKIP.",
            "question",
            CustomAlertTypes.ALERT_INFO);

      }// end if (c == helpCommand)


      if (c == exitCommand){
        // TO DO - cancel the selected seats before exit
        rejectPayment("exit");

      } // end if (c == exitCommand)

  } catch (Exception e) {
      e.printStackTrace();
      CanvasAlert keyErrorAlert = new CanvasAlert(
          display,
          getScreen(),
          "Ticket␣Discount␣Error!",
```

```
        "Error␣in␣the␣Ticket␣Discount␣Screen!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    display.setCurrent(keyErrorAlert);
  }
}

/**
 * Initialize the model an open the record store
 */
protected void initModel() throws Exception {
  // create the cinema tickets
  cinemaTickets = new TicketBean[reservedSeats.length];

  for(int i = 0; i < reservedSeats.length; i++){
    cinemaTickets[i] = new TicketBean();
    cinemaTickets[i].setTKTCinema(genericTicket.getTKTCinema());
    cinemaTickets[i].setTKTCinemaAddress(genericTicket.
        getTKTCinemaAddress());
    cinemaTickets[i].setTKTDiscountType(genericTicket.
        getTKTDiscountType());
    cinemaTickets[i].setTKTMovie(genericTicket.getTKTMovie());
    cinemaTickets[i].setTKTReservationDate(genericTicket.
        getTKTReservationDate());
    cinemaTickets[i].setTKTRow(genericTicket.getTKTRow());
    cinemaTickets[i].setTKTSeat(genericTicket.getTKTSeat());
    cinemaTickets[i].setTKTShowDate(genericTicket.getTKTShowDate());
    cinemaTickets[i].setTKTShowHour(genericTicket.getTKTShowHour());
  }

  // populate the Tickets Combo Box
  ticketsValue       = new String[reservedSeats.length];
  ticketsDiscountValue = new String[reservedSeats.length];

  for(int i = 0; i < reservedSeats.length; i++){
    ticketsValue[i] = "Ticket␣" + (i+1) + "-␣Row:␣" + reservedSeats[i
        ][0] + ",␣Seat:␣" + reservedSeats[i][1];
  }

  // initialize the ticket discount with NONE
  for(int i = 0; i < reservedSeats.length; i++){
    ticketsDiscountValue[i] = discountValue[0];
  }

  // init the ticket info
  ticketInfoUI   = new StringItem[reservedSeats.length];
```

```
  ticketInfoItems = new String[reservedSeats.length];

  for(int i = 0; i < ticketInfoItems.length; i++){
    ticketInfoItems[i] = genericTicket.getTKTMovie() +
    "␣(" + genericTicket.getTKTShowDate() +
    "␣-␣" + genericTicket.getTKTShowHour() +
    "),␣Row:␣" + reservedSeats[i][0] +
    ",␣Seat:␣" + reservedSeats[i][1] +
    ",␣Discount␣Type:␣" + ticketsDiscountValue[i];
  }

} //end initModel()

/**
 * Creates the Ticket Discount Screen
 *
 * @throws Exception
 *
 */
protected void createView() throws Exception {
  acceptCommand = new Command("ACCEPT", Command.EXIT, 0);
  mainCommand  = new Command("MAIN␣MENU", Command.SCREEN, 2);
  helpCommand  = new Command("HELP",   Command.SCREEN, 3);
  exitCommand  = new Command("EXIT",   Command.SCREEN, 4);

  ticketsUI    = new ChoiceGroup("Tickets:", Choice.POPUP, ticketsValue
      , null);
  discountsUI  = new ChoiceGroup("Discount:", Choice.POPUP,
      discountValue, null);
  ticketPrices = new double[ticketsValue.length];

  for(int i = 0; i < ticketInfoItems.length; i++){
    ticketInfoUI[i] = new StringItem("Ticket␣" + (i+1) +":",
        ticketInfoItems[i], Item.PLAIN);
    ticketPrices[i] = (1.0 -
        UpdateTicketDiscountAndReservationSummaryScreen.
        getDiscountvalue(ticketsDiscountValue[i], discountDoubleValues)
        )*ticketBasePrice;
    totalPrice += ticketPrices[i];
    ticketInfoUI[i].setText("Cinema:␣" + genericTicket.getTKTCinema()
         + "␣(" + genericTicket.getTKTCinemaAddress() + ")␣" + ",␣Movie
        :␣" + genericTicket.getTKTMovie() + "␣(" + genericTicket.
        getTKTShowDate() + "␣-␣" + genericTicket.getTKTShowHour() + ")
        ,␣Row:␣" + reservedSeats[i][0] + ",␣Seat:␣" + reservedSeats[i
        ][1] + ",␣Discount␣Type:␣" + ticketsDiscountValue[i] + ",␣
        Ticket␣Price:␣" + ticketPrices[i] + "␣DKK");
  }
```

```
  totalPriceUI = new StringItem("Total␣Price␣To␣Be␣Payed:␣", totalPrice
       + "␣DKK", Item.PLAIN);

  try{
    imgUp = Image.createImage("/" + Start.themeDir + "/ticket/tktTheme.
        png");
    imgThemeUp = new ImageItem("", imgUp, Item.LAYOUT_TOP | Item.
        LAYOUT_CENTER , "Theme_Img_Up");

  }catch(IOException ioe){
    System.out.println("Ticket␣Discount␣image␣exception!");
  }

  Spacer[]spacerUI = new Spacer[ticketInfoItems.length];
  screen = new Form("Ticket␣Discount");

  ((Form)screen).append(imgUp);
  ((Form)screen).append(ticketsUI);
  ((Form)screen).append(discountsUI);
  ((Form)screen).append(totalPriceUI);
  for(int i = 0; i < ticketInfoItems.length; i++){
    spacerUI[i] = new Spacer(100, 3);
    ((Form)screen).append(spacerUI[i]);
    ((Form)screen).append(ticketInfoUI[i]);
  }

  // add the commands to the form
  screen.addCommand(acceptCommand);
  screen.addCommand(mainCommand);
  screen.addCommand(helpCommand);
  screen.addCommand(exitCommand);

  ((Form)screen).setItemStateListener(this);
}

/**
 * Update the view - maybe refresh the fields
 *
 */
protected void updateView() throws Exception {
  initModel();
  createView();
}

/**
 * Trigered when the state of any of the UI elements changes
```

```
 * Actions can be taken based on the selected UI item e.g. update
     another
 * UI component when selecting another one.
 */
public void itemStateChanged(Item item){

  int selectedTicket = 0;

  if (item.getLabel().equals("Tickets:")){
    boolean[] ticketSelected = new boolean[ticketsUI.size()];
    ticketsUI.getSelectedFlags(ticketSelected);
    selectedTicket = ticketsUI.getSelectedIndex();

    UpdateTicketDiscountAndReservationSummaryScreen.resetDiscount(
        selectedTicket, discountsUI, discountValue,
        ticketsDiscountValue);

  }// end if (item.getLabel().equals("Movies:"))


  if (item.getLabel().equals("Discount:")){
    boolean[] ticketSelected = new boolean[ticketsUI.size()];
    ticketsUI.getSelectedFlags(ticketSelected);
    selectedTicket = ticketsUI.getSelectedIndex();

    boolean[] discountSelected = new boolean[discountsUI.size()];
    discountsUI.getSelectedFlags(discountSelected);
    int selectedDiscount = discountsUI.getSelectedIndex();

    // set the discount type for the selected ticket
    // and update ticket info items
    ticketsDiscountValue =
        UpdateTicketDiscountAndReservationSummaryScreen.
    setAndUpdateTicketDiscount(
        genericTicket,
        reservedSeats,
        ticketPrices,
        selectedTicket,
        selectedDiscount,
        ticketsDiscountValue,
        discountValue,
        ticketInfoItems,
        ticketInfoUI);

    ticketPrices = UpdateTicketDiscountAndReservationSummaryScreen.
    setAndUpdateTicketPrices(
        genericTicket,
```

```
        reservedSeats,
        ticketPrices,
        discountDoubleValues,
        ticketBasePrice,
        selectedTicket,
        selectedDiscount,
        ticketsDiscountValue,
        discountValue,
        ticketInfoItems,
        ticketInfoUI);

    totalPrice = UpdateTicketDiscountAndReservationSummaryScreen.
    setAndUpdateTotalPriceToBePayed(
        ticketPrices,
        totalPriceUI);

  }// end if (item.getLabel().equals("Cinemas:"))


}// end itemStateChanged()


/**
 * Constructs the Reject_Payment_Req_Bean and sends the request
 * over the network.
 *
 * Function of the response code and starting point
 * it displays either the main menu
 * or exits the application id the request comes when user wants to
     exit.
 *
 * @param nextScreenName The name of the screen that should be
 * displayed in case the payment is rejected sucesfuly
 *
 * @throws Exception
 */
private void rejectPayment(String nextScreenName) throws Exception{

  Reject_Payment_Req_Bean reqBean = new Reject_Payment_Req_Bean();
  reqBean.setShowLocationID(selDeselectSeatsReqBean.getShowLocationID()
      );
  reqBean.setShowTimeID  (selDeselectSeatsReqBean.getShowTimeID());
  reqBean.setSeatsNoCols (selDeselectSeatsReqBean.getSeatsNoCols());
  reqBean.setSeatsNoRows (selDeselectSeatsReqBean.getSeatsNoRows());
  reqBean.setSeats       (selDeselectSeatsReqBean.getSeats());

  SendMessage sm = new SendMessage(
```

```
        display,
        Protocol_Step_Constants.PRT_STEP_REJECT_PAYMENT,
        getScreen(),
        reqBean);
    sm.setRejectPaymentData(nextScreenName);
    sm.go();

  }// rejectPayment()

}// end class

package gui.purchasetickets.step4discountandreservationsummary;


import javax.microedition.lcdui.ChoiceGroup;
import javax.microedition.lcdui.StringItem;

import model.beans.otherbeans.TicketBean;



/**
 * Updates the disocunt UI components based on the
 * selected items in the components
 *
 * @author Mihai Balan, s031288
 *
 */
public class UpdateTicketDiscountAndReservationSummaryScreen {

  /**
   * Set the ticket discount values
   * function of the seleted ticket and selected discoutn value.
   * Also, updates the UI and displays updated ticket details
   *
   * @param movie Movie name
   * @param showDate Show date
   * @param showHour Show hour
   * @param reservedSeats Reserved seats
   * @param selectedTicket Selected ticket in the choice group
   * @param selectedDiscount Selected discount in the choice group
   * @param ticketsDiscountValue All ticket discount values
   * @param discountValue Standard discoutn values
   * @param ticketInfoItems The ticket details UI components
   * @param ticketInfo The values used to populate the ticketInfoItems
   *
   * @return ticketsDiscountValue - All ticket discount values (updated
       ones)
```

```
 */
public static String[] setAndUpdateTicketDiscount(
    TicketBean genericTicket,
    int[][] reservedSeats,
    double[] ticketPrices,
    int selectedTicket,
    int selectedDiscount,
    String[] ticketsDiscountValue,
    String[] discountValue,
    String[] ticketInfoItems,
    StringItem[] ticketInfoUI){

  ticketsDiscountValue[selectedTicket] = discountValue[selectedDiscount
      ];
  ticketInfoUI[selectedTicket].setText("Cinema:␣" + genericTicket.
      getTKTCinema() + "␣(" + genericTicket.getTKTCinemaAddress() + ")
      ␣" + ",␣Movie:␣" + genericTicket.getTKTMovie() + "␣(" +
      genericTicket.getTKTShowDate() + "␣-␣" + genericTicket.
      getTKTShowHour() + "),␣Row:␣" + reservedSeats[selectedTicket
      ][0] + ",␣Seat:␣" + reservedSeats[selectedTicket][1] + ",␣
      Discount␣Type:␣" + ticketsDiscountValue[selectedTicket] + ",␣
      Ticket␣Price:␣" + ticketPrices[selectedTicket] + "␣DKK");
  return ticketsDiscountValue;

}// end setAndUpdateTicketDiscount()

/**
 * Update the ticket prices fucntion of the chosen discount type
 *
 * @return The updated ticket prices
 */
public static double[] setAndUpdateTicketPrices(
    TicketBean genericTicket,
    int[][] reservedSeats,
    double[] ticketPrices,
    double[] discountDoubleValues,
    double ticketBasePrice,
    int selectedTicket,
    int selectedDiscount,
    String[] ticketsDiscountValue,
    String[] discountValue,
    String[] ticketInfoItems,
    StringItem[] ticketInfoUI){

  ticketPrices[selectedTicket] = formatDoublePrice((1.0 -
      UpdateTicketDiscountAndReservationSummaryScreen.getDiscountvalue(
      ticketsDiscountValue[selectedTicket], discountDoubleValues))*
```

```
      ticketBasePrice);
  ticketInfoUI[selectedTicket].setText("Cinema:␣" + genericTicket.
      getTKTCinema() + "␣(" + genericTicket.getTKTCinemaAddress() + ")
      ␣" + ",␣Movie:␣" + genericTicket.getTKTMovie() + "␣(" +
      genericTicket.getTKTShowDate() + "␣-␣" + genericTicket.
      getTKTShowHour() + "),␣Row:␣" + reservedSeats[selectedTicket
      ][0] + ",␣Seat:␣" + reservedSeats[selectedTicket][1] + ",␣
      Discount␣Type:␣" + ticketsDiscountValue[selectedTicket] + ",␣
      Ticket␣Price:␣" + ticketPrices[selectedTicket] + "␣DKK");

  return ticketPrices;

}// end setAndUpdateTicketPrices()

/**
 * Updates the total price to be payed
 * @return The total price to be payed
 */
public static double setAndUpdateTotalPriceToBePayed(
    double[] ticketPrices,
    StringItem totalPriceUI){

  double totalPrice = 0.0;

  for(int i = 0; i < ticketPrices.length; i++){
    totalPrice += ticketPrices[i];
  }

  totalPriceUI.setText(formatDoublePrice(totalPrice) + "␣DKK");

  return new Double(formatDoublePrice(totalPrice));

}// end setAndUpdateTotalPriceToBePayed()



/**
 * return the discount value based on the discount type
 *
 */
public static double getDiscountvalue(String discountType, double[]
    discountValues){

  if(discountType.equals("CHILD"))
    return discountValues[0];

  else if(discountType.equals("STUDENT"))
```

```
    return discountValues[1];

  else if(discountType.equals("PENSIONER"))
    return discountValues[2];

  else if(discountType.equals("VOUCHER"))
    return discountValues[3];

  return 0.0; // DISCOUNT = NONE

}// end getDiscountvalue()

/**
 * Returns the ticket price wit 2 decimals only
 *
 * @param ticketPrice The ticket price as a double with n decimals
 * @return The ticket price wit 2 decimals only
 */
private static double formatDoublePrice(double ticketPrice){

  String strPrice = new Double(ticketPrice).toString();
  int index = strPrice.indexOf(".");

  if ((strPrice.length() - (index + 1)) > 2){
    strPrice = strPrice.substring(0, index + 3);
  }

  return Double.parseDouble(strPrice);

}// end formatDoublePrice()


/**
 * Returns the ticket price wit 2 decimals only
 *
 * @param ticketPrice The ticket price as a double with n decimals
 * @return The ticket price wit 2 decimals only
 */
public static String formatTotalPrice(double ticketPrice){

  String strPrice = new Double(ticketPrice).toString();
  int index = strPrice.indexOf(".");

  if ((strPrice.length() - (index + 1)) > 2){
    strPrice = strPrice.substring(0, index + 3);
  }
```

```
    return strPrice;

  }// end formatDoublePrice()

  public static void resetDiscount(int selectedTicket, ChoiceGroup
      discountsUI, String[] discountValue, String[] ticketsDiscountValue)
      {

    String currentDiscValue = "";

    for(int i = 0; i < ticketsDiscountValue.length; i++){
      if (i == selectedTicket){
        currentDiscValue = ticketsDiscountValue[i];
      }
    }

    for(int i = 0; i < discountValue.length; i++){
      if (currentDiscValue.equals(discountValue[i])){
        discountsUI.setSelectedIndex(i, true);
      }
    }

  }// end resetDiscount()

}// end class

package gui.purchasetickets.step5chooseticketpayment;

import java.io.IOException;
import javax.microedition.lcdui.*;

import networkoperations.SendMessage;

import model.beans.otherbeans.TicketBean;
import model.beans.requestbeans.Cinema_Hall_Conf_Req_Bean;
import model.beans.requestbeans.Reject_Payment_Req_Bean;
import model.beans.requestbeans.Select_Deselect_Seats_Req_Bean;
import model.beans.responsebeans.Cinema_Hall_Conf_Resp_Bean;
import model.beans.responsebeans.Select_Deselect_Seats_Resp_Bean;



import rms.RMSOperations;
import start.Start;

import gui.GenericGUI;
import gui.customdialogwindows.CanvasAlert;
import gui.customdialogwindows.DialogWindow;
```

```
import gui.mainmenu.MenuScreen;

import constants.CreditCardTypes;
import constants.CustomAlertTypes;
import constants.Protocol_Step_Constants;

import cryptography.Encryptor;


/**
 *  Allows customer to select the payment method
 *  to pay for the reserve tickets i.e. at the cinema,
 *  existing credit cards in the secure wallet, refunded e-money,
 *  or new credit card
 *
 *  It extends the GenericGUI super class
 *
 * @author s031288, Mihai Balan
 *
 */
public class ChooseTicketPaymentGUI extends GenericGUI implements
    ItemStateListener{

  // the ticket payment screen
  private static Displayable screen = null;

  //  the commands
  private static Command purchaseCommand;
  private static Command backCommand;
  private static Command mainCommand;
  private static Command helpCommand;
  private static Command exitCommand;

  // UI components
  private TextField walletPinUI;
  private TextField ccNoUI;
  private TextField ccValidMonthUI;
  private TextField ccValidYearUI;
  private TextField ccCW2UI;
  private ChoiceGroup paymentMethodUI;
  private ChoiceGroup creditCardTypeUI;
  private ChoiceGroup eMoneyPayMethodUI;
  private StringItem eMoneyUI;
  private StringItem amountToPayUI;
  private StringItem eMoneyInfoUI;
  private StringItem eMoneyPayMethodTitleUI;
  private Image     imgUp;
```

```
private ImageItem imgThemeUp;

private Displayable backScreen;

private double totalPrice = 0.0;
private String[] paymentMethodValues = {"At␣the␣Cinema", "Secure␣Wallet
    ", "Credit␣Card", "E-Money"};
private String[] creditCardTypeValues = CreditCardTypes.CC_TYPES;
private String[] eMoneyPayMethodValues = {"At␣the␣Cinema", "Secure␣
    Wallet", "Credit␣Card"};

//private Encryptor encryptor;
private boolean emptyWallet;
private boolean enoughEMoney = false;
private String  rmsPIN;
//private String  refundedEMoney;
private String  eMoneyInfoText;

private String[] ticketsDiscountValues;
private TicketBean[]                cinemaTickets;
private Cinema_Hall_Conf_Req_Bean   cineHallConfReqBean;
private Cinema_Hall_Conf_Resp_Bean  cineHallConfRespBean;
private Select_Deselect_Seats_Req_Bean selDeselectSeatsReqBean;
private Select_Deselect_Seats_Resp_Bean selDeselectSeatsRespBean;

// the number of times a user can enter the
// PIN wrong. If the PIN is entered wrong more then
// 3 times, My Wallet content is deleted and the PIN reset.
public static int pinTrials = 3;



/**
 * Constructs an instance of the class
 */
public ChooseTicketPaymentGUI(){
}


public ChooseTicketPaymentGUI (
    double                      totalPrice,
    Displayable                 backScreen,
    String[]                    ticketsDiscountValues,
    TicketBean[]                cinemaTickets,
    Cinema_Hall_Conf_Req_Bean   cineHallConfReqBean,
    Cinema_Hall_Conf_Resp_Bean  cineHallConfRespBean,
    Select_Deselect_Seats_Req_Bean selDeselectSeatsReqBean,
```

```
    Select_Deselect_Seats_Resp_Bean selDeselectSeatsRespBean) {

  this.totalPrice            = totalPrice;
  this.backScreen            = backScreen;
  this.ticketsDiscountValues = ticketsDiscountValues;
  this.cinemaTickets         = cinemaTickets;
  this.cineHallConfReqBean   = cineHallConfReqBean;
  this.cineHallConfRespBean  = cineHallConfRespBean;
  this.selDeselectSeatsReqBean = selDeselectSeatsReqBean;
  this.selDeselectSeatsRespBean = selDeselectSeatsRespBean;

}

/**
 * Returns the displayable ChooseTicketPaymentGUI screen
 * @return screen Returns the ChooseTicketPaymentGUI screen
 */
public Displayable getScreen() {
  return screen;
}

/**
 * Chosen command
 *
 * @param c The executed command
 * @param s The main menu form
 *
 */
public void commandAction(Command c, Displayable s) {
  try {

    if (c == purchaseCommand){

      // verify the credit card data and make the payment if CC data OK
      pinTrials = UpdateTicketPaymentScreen.getPaymentMethodAndPay(
                  display,
                  getScreen(),
                  ticketsDiscountValues,
                  paymentMethodValues,
                  paymentMethodUI,
                  creditCardTypeUI,
                  creditCardTypeValues,
                  walletPinUI,
                  ccNoUI,
                  ccValidMonthUI,
                  ccValidYearUI,
                  ccCW2UI,
```

```
                eMoneyUI,
                String.valueOf(totalPrice),
                eMoneyPayMethodUI,
                eMoneyPayMethodValues,
                pinTrials,
                cinemaTickets,
                cineHallConfReqBean,
                cineHallConfRespBean,
                selDeselectSeatsReqBean,
                selDeselectSeatsRespBean);

    }// end if (c == purchaseCommand)


    if (c == backCommand){
      display.setCurrent(backScreen);
    }// end if (c == backCommand)


    if (c == mainCommand){
      rejectPayment("main");

    }// end if (c == mainCommand)

    if (c == helpCommand){
      CanvasAlert help = new CanvasAlert(
          display,
          display.getCurrent(),
          "Ticket␣Payment␣Method",
          "One␣can␣select␣to␣pay␣for␣the␣tickets␣using␣his␣credit␣card␣
              in␣secure␣wallet,␣refunded␣e-money,␣or␣when␣arraiving␣at␣
              the␣cinema.",
          "question",
          CustomAlertTypes.ALERT_INFO);

    }// end if (c == helpCommand)

    if (c == exitCommand){
      rejectPayment("exit");

    } // end if (c == exitCommand)

  } catch (Exception e) {
    e.printStackTrace();
    CanvasAlert keyErrorAlert = new CanvasAlert(
        display,
        getScreen(),
```

```
        "ChooseTicketPaymentGUI␣Error!",
        "Error␣in␣the␣ChooseTicketPaymentGUI␣Screen!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    display.setCurrent(keyErrorAlert);
  }
}

/**
 * Initialize the model an access the record store
 */
protected void initModel() throws Exception {
  // get e-money, and secure wallet PIN from RMS
  //refundedEMoney = RMSOperations.getItem("EMN:");

  if(Start.emoney.equals(""))
    Start.emoney = "0.0";

  if((totalPrice - Double.parseDouble(Start.emoney)) > 0.0){
    enoughEMoney  = false;
    eMoneyInfoText = "You␣do␣not␣have␣sufficient␣e-money␣to␣purchase␣
        the␣tickets.␣Please␣use␣one␣of␣the␣payment␣methods␣depicted␣
        below,␣too!";

  } else{
    enoughEMoney  = true;
    eMoneyInfoText = "You␣have␣sufficient␣e-money␣to␣purchase␣your␣
        tickets!";
  }

  //String authKey = RMSOperations.getItem("KEY:");
  //encryptor = new Encryptor(authKey);

  //rmsPIN = new String(RMSOperations.getDecryptedItem("PIN:"));

  // if PIN code not found in RMS (user logs in for the first time in
      RMS)
  if(!Start.walletPin.equals("")){
    emptyWallet = false;
  } else {
    emptyWallet = true;
  }

} //end initModel()

/**
```

```
 * Creates the ChooseTicketPaymentGUI Screen
 *
 * @throws Exception
 *
 */
protected void createView() throws Exception {
  purchaseCommand = new Command("PURCHASE", Command.EXIT, 0);
  backCommand    = new Command("BACK",    Command.SCREEN, 2);
  mainCommand    = new Command("MAIN␣MENU", Command.SCREEN, 3);
  helpCommand    = new Command("HELP",    Command.SCREEN, 4);
  exitCommand    = new Command("EXIT",    Command.SCREEN, 5);

  paymentMethodUI = new ChoiceGroup("Payment␣Method:", Choice.POPUP,
      paymentMethodValues, null);
  creditCardTypeUI = new ChoiceGroup("Credit␣Card␣Type:", Choice.POPUP
      , creditCardTypeValues, null);
  eMoneyPayMethodUI = new ChoiceGroup("Secondary␣Payment:", Choice.
      POPUP, eMoneyPayMethodValues, null);

  walletPinUI    = new TextField("Secure␣Wallet␣PIN:",       "", 40,
      TextField.PASSWORD);
  ccNoUI         = new TextField("Credit␣Card␣Number:",      "", 16,
      TextField.NUMERIC);
  ccValidMonthUI = new TextField("Credit␣Card␣Expiring␣Month:", "", 2,
      TextField.NUMERIC);
  ccValidYearUI  = new TextField("Credit␣Card␣Expiring␣␣␣␣Year:", ""
      , 4, TextField.NUMERIC);
  ccCW2UI        = new TextField("Credit␣Card␣Security␣Code:␣", "", 3,
      TextField.ANY);

  amountToPayUI = new StringItem("Total␣Amount␣to␣be␣payed:", Double.
      toString(totalPrice) + "␣DKK");
  eMoneyUI      = new StringItem("Available␣E-Money:", Start.emoney);
  eMoneyInfoUI  = new StringItem("", eMoneyInfoText);

  eMoneyPayMethodTitleUI = new StringItem("Pay␣the␣price␣difference␣
      using:", "");

  try{
    imgUp = Image.createImage("/" + Start.themeDir + "/ticketpayment/
        tktPayTheme.png");
    imgThemeUp = new ImageItem("", imgUp, Item.LAYOUT_TOP | Item.
        LAYOUT_CENTER , "Theme_Img_Up");

  }catch(IOException ioe){
    System.out.println("Ticket␣Discount␣image␣exception!");
  }
```

```
  screen = new Form("Ticket␣Payment");
  ((Form)screen).append(imgUp);
  ((Form)screen).append(amountToPayUI);
  ((Form)screen).append(paymentMethodUI);

  UpdateTicketPaymentScreen.updateTicketPaymentView(screen,
      paymentMethodUI, creditCardTypeUI, walletPinUI, ccNoUI,
      ccValidMonthUI, ccValidYearUI, ccCW2UI, eMoneyUI, amountToPayUI,
      imgUp, eMoneyPayMethodUI, eMoneyInfoUI, enoughEMoney,
      eMoneyPayMethodTitleUI, emptyWallet);

  // add the commands to the form
  screen.addCommand(purchaseCommand);
  screen.addCommand(backCommand);
  screen.addCommand(mainCommand);
  screen.addCommand(helpCommand);
  screen.addCommand(exitCommand);

  ((Form)screen).setItemStateListener(this);

}

/**
 * Update the view - maybe refresh the fields
 *
 */
protected void updateView() throws Exception {
  initModel();
  createView();
}

/**
 * Trigered when the state of any of the UI elements changes
 * Actions can be taken based on the selected UI item e.g. update
     another
 * UI component when selecting another one.
 */
public void itemStateChanged(Item item){

  if (item.getLabel().equals("Payment␣Method:")){
    UpdateTicketPaymentScreen.updateTicketPaymentView(screen,
        paymentMethodUI, creditCardTypeUI, walletPinUI, ccNoUI,
        ccValidMonthUI, ccValidYearUI, ccCW2UI, eMoneyUI, amountToPayUI
        , imgUp, eMoneyPayMethodUI, eMoneyInfoUI, enoughEMoney,
        eMoneyPayMethodTitleUI, emptyWallet);
```

```
  }// end if (item.getLabel().equals("Payment Method:"))

  if ((item.getLabel().equals("Secondary␣Payment:"))){
    UpdateTicketPaymentScreen.updateSecondaryEMoneyTicketPaymentView(
        screen, paymentMethodUI, creditCardTypeUI, walletPinUI, ccNoUI
        , ccValidMonthUI, ccValidYearUI, ccCW2UI, eMoneyUI,
        amountToPayUI, imgUp, eMoneyPayMethodUI, eMoneyInfoUI,
        enoughEMoney, eMoneyPayMethodTitleUI, emptyWallet);

  }// end if (item.getLabel().equals("Payment Method:"))

}// end itemStateChanged()


/**
 * Constructs the Reject_Payment_Req_Bean and sends the request
 * over the network.
 *
 * Function of the response code and starting point
 * it displays either the main menu
 * or exits the application id the request comes when user wants to
     exit.
 *
 * @param nextScreenName The name of the screen that should be
 * displayed in case the payment is rejected sucesfuly
 *
 * @throws Exception
 */
private void rejectPayment(String nextScreenName) throws Exception{

  Reject_Payment_Req_Bean reqBean = new Reject_Payment_Req_Bean();
  reqBean.setShowLocationID(selDeselectSeatsReqBean.getShowLocationID()
      );
  reqBean.setShowTimeID  (selDeselectSeatsReqBean.getShowTimeID());
  reqBean.setSeatsNoCols (selDeselectSeatsReqBean.getSeatsNoCols());
  reqBean.setSeatsNoRows (selDeselectSeatsReqBean.getSeatsNoRows());
  reqBean.setSeats       (selDeselectSeatsReqBean.getSeats());

  SendMessage sm = new SendMessage(
      display,
      Protocol_Step_Constants.PRT_STEP_REJECT_PAYMENT,
      getScreen(),
      reqBean);
  sm.setRejectPaymentData(nextScreenName);
  sm.go();

}// rejectPayment()
```

```
}// end class

package gui.purchasetickets.step5chooseticketpayment;


import gui.customdialogwindows.CanvasAlert;
import gui.mywallet.GUIWalletHelper;
import gui.mywallet.UpdateWalletGUI;
import gui.purchasetickets.step6billinginfo.BillingInfoGUI;

import javax.microedition.lcdui.ChoiceGroup;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.StringItem;
import javax.microedition.lcdui.TextField;


import rms.RMSOperations;
import start.Start;

import model.beans.otherbeans.CreditCardBean;
import model.beans.otherbeans.TicketBean;
import model.beans.requestbeans.Cinema_Hall_Conf_Req_Bean;
import model.beans.requestbeans.Purchase_Tickets_Req_Bean;
import model.beans.requestbeans.Select_Deselect_Seats_Req_Bean;
import model.beans.responsebeans.Cinema_Hall_Conf_Resp_Bean;
import model.beans.responsebeans.Purchase_Tickets_Resp_Bean;
import model.beans.responsebeans.Select_Deselect_Seats_Resp_Bean;
import networkoperations.NetworkCommunicationFacade;
import networkoperations.NetworkResposeFacade;
import networkoperations.SendMessage;

import constants.CustomAlertTypes;
import constants.Protocol_Step_Constants;
import constants.PurchaseMethosConstants;
import constants.SystemConstants;
import cryptography.Encryptor;

/**
 * Update the ticket payment UI components based on the user interaction
 *
 * @author Mihai Balan, s031288
 *
 */
public class UpdateTicketPaymentScreen {
```

```
/**
 * Get the selected index corresponding to the payment method
 *
 */
private static int selectedTicketPaymentMethod(ChoiceGroup
    paymentMethodUI){

  boolean[] ticketSelected = new boolean[paymentMethodUI.size()];
  paymentMethodUI.getSelectedFlags(ticketSelected);
  return paymentMethodUI.getSelectedIndex();

}// end selectedTicketPaymentMethod()


/**
 * Get the selected index corresponding to the credit card type
 *
 */
private static int selectedCreditCardType(ChoiceGroup creditCardTypeUI)
    {

  boolean[] ccSelected = new boolean[creditCardTypeUI.size()];
  creditCardTypeUI.getSelectedFlags(ccSelected);
  return creditCardTypeUI.getSelectedIndex();

}// end selectedCreditCardType()



/**
 * Updates the view function of the selected payment method
 */
public static void updateTicketPaymentView(
    Displayable screen,
    ChoiceGroup paymentMethodUI,
    ChoiceGroup creditCardTypeUI,
    TextField  walletPinUI,
    TextField  ccNoUI,
    TextField  ccValidMonthUI,
    TextField  ccValidYearUI,
    TextField  ccCW2UI,
    StringItem eMoneyUI,
    StringItem amountToPayUI,
    Image      imgUp,
    ChoiceGroup eMoneyPayMethodUI,
```

```
  StringItem eMoneyInfoUI,
  boolean    enoughEMoney,
  StringItem eMoneyPayMethodTitleUI,
  boolean    emptyWallet){

// payment method is AT THE CINEMA
if(selectedTicketPaymentMethod(paymentMethodUI) == 0){
  ((Form)screen).deleteAll();
  ((Form)screen).append(imgUp);
  ((Form)screen).append(amountToPayUI);
  ((Form)screen).append(paymentMethodUI);

}// end if()

// payment method is SECURE WALLET and the wallet is not empty
if((!emptyWallet) && (selectedTicketPaymentMethod(paymentMethodUI)
     == 1)){
  ((Form)screen).deleteAll();
  ((Form)screen).append(imgUp);
  ((Form)screen).append(amountToPayUI);
  ((Form)screen).append(paymentMethodUI);
  ((Form)screen).append(walletPinUI);

// payment method is SECURE WALLET and the wallet is empty
}else if((emptyWallet) && (selectedTicketPaymentMethod(
    paymentMethodUI) == 1)){
  ((Form)screen).deleteAll();
  ((Form)screen).append(imgUp);
  ((Form)screen).append(amountToPayUI);
  ((Form)screen).append(paymentMethodUI);
  ((Form)screen).append(new StringItem("", "Your␣Secure␣Wallet␣is␣
      empty!␣Please␣use␣At␣the␣Cinema,␣Credit␣Card,␣or␣E-Money␣
      payment␣method␣instead!"));

}// end if()

// payment method is NEW CREDIT CARD
if(selectedTicketPaymentMethod(paymentMethodUI) == 2){
  ((Form)screen).deleteAll();
  ((Form)screen).append(imgUp);
  ((Form)screen).append(amountToPayUI);
  ((Form)screen).append(paymentMethodUI);
  ((Form)screen).append(creditCardTypeUI);
  ((Form)screen).append(ccNoUI);
  ((Form)screen).append(ccValidMonthUI);
  ((Form)screen).append(ccValidYearUI);
  ((Form)screen).append(ccCW2UI);
```

```
  }// end if()

  // payment method is REFUNDED E-MONEY
  if(selectedTicketPaymentMethod(paymentMethodUI) == 3){
    ((Form)screen).deleteAll();
    ((Form)screen).append(imgUp);
    ((Form)screen).append(amountToPayUI);
    ((Form)screen).append(paymentMethodUI);
    ((Form)screen).append(eMoneyUI);
    ((Form)screen).append(eMoneyInfoUI);

    // if there are insuficient e-money display
    // other payment methods to pay for the rest.
    if (!enoughEMoney){
      ((Form)screen).append(eMoneyPayMethodTitleUI);
      ((Form)screen).append(eMoneyPayMethodUI);

    }

  }// end if()

}// end updateTicketPaymentView()



/**
 * Updates the view function of the selected payment method
 */
public static void updateSecondaryEMoneyTicketPaymentView(
    Displayable screen,
    ChoiceGroup paymentMethodUI,
    ChoiceGroup creditCardTypeUI,
    TextField  walletPinUI,
    TextField  ccNoUI,
    TextField  ccValidMonthUI,
    TextField  ccValidYearUI,
    TextField  ccCW2UI,
    StringItem eMoneyUI,
    StringItem amountToPayUI,
    Image      imgUp,
    ChoiceGroup eMoneyPayMethodUI,
    StringItem eMoneyInfoUI,
    boolean    enoughEMoney,
    StringItem eMoneyPayMethodTitleUI,
    boolean    emptyWallet){
```

```
// payment method is AT THE CINEMA
if(selectedTicketPaymentMethod(eMoneyPayMethodUI) == 0){
  ((Form)screen).deleteAll();
  ((Form)screen).append(imgUp);
  ((Form)screen).append(amountToPayUI);
  ((Form)screen).append(paymentMethodUI);
  ((Form)screen).append(eMoneyUI);
  ((Form)screen).append(eMoneyInfoUI);

  // if there are insuficient e-money display
  // other payment methods to pay for the rest.
  if (!enoughEMoney){
    ((Form)screen).append(eMoneyPayMethodTitleUI);
    ((Form)screen).append(eMoneyPayMethodUI);
  }

}// end if()


// payment method is SECURE WALLET
if(selectedTicketPaymentMethod(eMoneyPayMethodUI) == 1){
  ((Form)screen).deleteAll();
  ((Form)screen).append(imgUp);
  ((Form)screen).append(amountToPayUI);
  ((Form)screen).append(paymentMethodUI);
  ((Form)screen).append(eMoneyUI);
  ((Form)screen).append(eMoneyInfoUI);

  // if there are insuficient e-money display
  // other payment methods to pay for the rest.
  if (!enoughEMoney){
    ((Form)screen).append(eMoneyPayMethodTitleUI);
    ((Form)screen).append(eMoneyPayMethodUI);
  }

  if(!emptyWallet){
    ((Form)screen).append(walletPinUI);

  }else{
    ((Form)screen).append(new StringItem("", "Your␣Secure␣Wallet␣is␣
        empty!␣Please␣use␣At␣the␣Cinema,␣or␣Credit␣Card␣payment␣
        method␣instead!"));

  }
}// end if()
```

```
  // payment method is NEW CREDIT CARD
  if(selectedTicketPaymentMethod(eMoneyPayMethodUI) == 2){
    ((Form)screen).deleteAll();
    ((Form)screen).append(imgUp);
    ((Form)screen).append(amountToPayUI);
    ((Form)screen).append(paymentMethodUI);
    ((Form)screen).append(eMoneyUI);
    ((Form)screen).append(eMoneyInfoUI);

    // if there are insuficient e-money display
    // other payment methods to pay for the rest.
    if (!enoughEMoney){
      ((Form)screen).append(eMoneyPayMethodTitleUI);
      ((Form)screen).append(eMoneyPayMethodUI);

    }
    ((Form)screen).append(creditCardTypeUI);
    ((Form)screen).append(ccNoUI);
    ((Form)screen).append(ccValidMonthUI);
    ((Form)screen).append(ccValidYearUI);
    ((Form)screen).append(ccCW2UI);

  }// end if()

}// end updateSecondaryEMoneyTicketPaymentView()


private static void verifyCreditCard(
    Display                     display,
    Displayable                 backScreen,
    String[]                    ticketsDiscountValue,
    String                      paymentMethodValue,
    String                      creditCardTypeValue,
    String                      ccNo,
    String                      ccValidMonth,
    String                      ccValidYear,
    String                      ccCW2,
    String                      eMoney,
    String                      amountToPay,
    TicketBean[]                cinemaTickets,
    Cinema_Hall_Conf_Req_Bean   cineHallConfReqBean,
    Cinema_Hall_Conf_Resp_Bean  cineHallConfRespBean,
    Select_Deselect_Seats_Req_Bean selDeselectSeatsReqBean,
    Select_Deselect_Seats_Resp_Bean selDeselectSeatsRespBean) throws
        Exception{
```

```
CreditCardBean ccBean = new CreditCardBean();

ccBean.setCCNumber      (ccNo);
ccBean.setCCExpDateMonth (ccValidMonth);
ccBean.setCCExpDateYear (ccValidYear);
ccBean.setCCCW2         (ccCW2);

// if the data entered make the payment
if (!GUIWalletHelper.validPayCreditCardUI(ccBean)){

  CanvasAlert ss = new CanvasAlert(
      display,
      backScreen,
      "Incorrect␣data!",
      "Please␣check␣that␣the␣exp.␣month␣is␣between␣1-12," +
      "␣exp.␣year␣is␣between␣200"
      + SystemConstants.CREDIT_CARD_EXP_YEAR_MIN + "␣-␣20"
      + SystemConstants.CREDIT_CARD_EXP_YEAR_MAX + ""
      + ",␣and␣CW2␣has␣exactly␣3␣characters!",
      "error",
      CustomAlertTypes.ALERT_WARNING);

}else{

  if (!paymentMethodValue.equals("E-Money")){

    // make the payment with the server side
    payOverNetwork(
      display,
      backScreen,
      ticketsDiscountValue,
      paymentMethodValue,
      creditCardTypeValue,
      ccNo,
      ccValidMonth,
      ccValidYear,
      ccCW2,
      amountToPay,
      cinemaTickets,
      cineHallConfReqBean,
      cineHallConfRespBean,
      selDeselectSeatsReqBean,
      selDeselectSeatsRespBean);

  }else{

    payOverNetwork(
```

```
            display,
            backScreen,
            ticketsDiscountValue,
            paymentMethodValue,
            creditCardTypeValue,
            ccNo,
            ccValidMonth,
            ccValidYear,
            ccCW2,
            String.valueOf((Double.parseDouble(amountToPay) - Double.
                parseDouble(eMoney))),
            cinemaTickets,
            cineHallConfReqBean,
            cineHallConfRespBean,
            selDeselectSeatsReqBean,
            selDeselectSeatsRespBean);

        }// end if (!paymentMethodValue.equals("E-Money"))

      }// end if (!GUIWalletHelper.validWalletAddNewCardUI(ccBean))

    }// end verifyCreditCard()

    /**
     * Get the payment method and based on that
     * pay over the network for the selected tickets
     */
    public static int getPaymentMethodAndPay(
        Display                     display,
        Displayable                 backScreen,
        String[]                    ticketsDiscountValue,
        String[]                    paymentMethodValues,
        ChoiceGroup                 paymentMethodUI,
        ChoiceGroup                 creditCardTypeUI,
        String[]                    creditCardTypeValues,
        TextField                   walletPinUI,
        TextField                   ccNoUI,
        TextField                   ccValidMonthUI,
        TextField                   ccValidYearUI,
        TextField                   ccCW2UI,
        StringItem                  eMoneyUI,
        String                      amountToPay,
        ChoiceGroup                 eMoneyPayMethodUI,
        String[]                    eMoneyPayMethodValues,
        int                         pinTrials,
        TicketBean[]                cinemaTickets,
        Cinema_Hall_Conf_Req_Bean   cineHallConfReqBean,
```

```
  Cinema_Hall_Conf_Resp_Bean   cineHallConfRespBean,
  Select_Deselect_Seats_Req_Bean selDeselectSeatsReqBean,
  Select_Deselect_Seats_Resp_Bean selDeselectSeatsRespBean) throws
      Exception{

// payment method is AT THE CINEMA
if(selectedTicketPaymentMethod(paymentMethodUI) == 0){

  // set ticket paymenet method
  cinemaTickets = setTicketPaymentMethod(cinemaTickets,
      paymentMethodValues[selectedTicketPaymentMethod(paymentMethodUI
      )]);

  payOverNetwork(
      display,
      backScreen,
      ticketsDiscountValue,
      paymentMethodValues[selectedTicketPaymentMethod(paymentMethodUI)
          ],
      "",
      "",
      "",
      "",
      "",
      amountToPay,
      cinemaTickets,
      cineHallConfReqBean,
      cineHallConfRespBean,
      selDeselectSeatsReqBean,
      selDeselectSeatsRespBean);

}// end if()

// payment method is SECURE WALLET
if(selectedTicketPaymentMethod(paymentMethodUI) == 1){

  // set ticket paymenet method
  cinemaTickets = setTicketPaymentMethod(cinemaTickets,
      paymentMethodValues[2]);

  pinTrials = UpdateWalletGUI.walletAuthenticationTicketPayment(
         display,
         backScreen,
         ticketsDiscountValue,
         paymentMethodValues[2], // se to credit card payment method
         walletPinUI.getString(),
         amountToPay,
```

```
            pinTrials,
            cinemaTickets,
            cineHallConfReqBean,
            cineHallConfRespBean,
            selDeselectSeatsReqBean,
            selDeselectSeatsRespBean);

    }// end if() SECURE WALLET

    // payment method is CREDIT CARD
    if(selectedTicketPaymentMethod(paymentMethodUI) == 2){

      // set ticket paymenet method
      cinemaTickets = setTicketPaymentMethod(cinemaTickets,
          paymentMethodValues[selectedTicketPaymentMethod(paymentMethodUI
          )]);

      verifyCreditCard(
          display,
          backScreen,
          ticketsDiscountValue,
          paymentMethodValues[selectedTicketPaymentMethod(paymentMethodUI)
              ],
          creditCardTypeValues[selectedCreditCardType(creditCardTypeUI)],
          ccNoUI.getString(),
          ccValidMonthUI.getString(),
          ccValidYearUI.getString(),
          ccCW2UI.getString(),
          eMoneyUI.getText(),
          amountToPay,
          cinemaTickets,
          cineHallConfReqBean,
          cineHallConfRespBean,
          selDeselectSeatsReqBean,
          selDeselectSeatsRespBean);

    }// end if()

    // payment method is REFUNDED E-MONEY
    if(selectedTicketPaymentMethod(paymentMethodUI) == 3){

      // set ticket paymenet method
      cinemaTickets = setTicketPaymentMethod(cinemaTickets,
          eMoneyPayMethodValues[selectedTicketPaymentMethod(
          eMoneyPayMethodUI)]);

      pinTrials = payUsingSecondaryPaymentMethod(
```

```
            display,
            backScreen,
            ticketsDiscountValue,
            paymentMethodValues[selectedTicketPaymentMethod(
                paymentMethodUI)],
            creditCardTypeValues[selectedCreditCardType(creditCardTypeUI
                )],
            ccNoUI.getString(),
            ccValidMonthUI.getString(),
            ccValidYearUI.getString(),
            ccCW2UI.getString(),
            amountToPay,
            eMoneyUI.getText(),
            eMoneyPayMethodUI,
            eMoneyPayMethodValues,
            walletPinUI.getString(),
            pinTrials,
            cinemaTickets,
            cineHallConfReqBean,
            cineHallConfRespBean,
            selDeselectSeatsReqBean,
            selDeselectSeatsRespBean);


  }// end if()

  return pinTrials;

}// getPaymentMethodAndPay()

/**
 *
 * Get the secondary payment method when first payment method is
 * e-money and the difference between the amount to pay and
 * available e-money is > 0. i.e. teh rest of the money
 * needs to be payed either at the cinema or using a credit card
 */
public static int payUsingSecondaryPaymentMethod(
    Display                     display,
    Displayable                 backScreen,
    String[]                    ticketsDiscountValue,
    String                      paymentMethodValue,
    String                      ccType,
    String                      ccNo,
    String                      ccValidMonth,
    String                      ccValidYear,
    String                      ccCW2,
```

```
  String                        amountToPay,
  String                        eMoney,
  ChoiceGroup                   eMoneyPayMethodUI,
  String[]                      eMoneyPayMethodValues,
  String                        walletPin,
  int                           pinTrials,
  TicketBean[]                  cinemaTickets,
  Cinema_Hall_Conf_Req_Bean     cineHallConfReqBean,
  Cinema_Hall_Conf_Resp_Bean    cineHallConfRespBean,
  Select_Deselect_Seats_Req_Bean selDeselectSeatsReqBean,
  Select_Deselect_Seats_Resp_Bean selDeselectSeatsRespBean) throws
      Exception{

// at the cinema secondary payment (e-money + rest at the cinema)
if(selectedTicketPaymentMethod(eMoneyPayMethodUI) == 0){

  payOverNetwork(
      display,
      backScreen,
      ticketsDiscountValue,
      eMoneyPayMethodValues[0],
      "",
      "",
      "",
      "",
      "",
      String.valueOf((Double.parseDouble(amountToPay) - Double.
          parseDouble(eMoney))),
      cinemaTickets,
      cineHallConfReqBean,
      cineHallConfRespBean,
      selDeselectSeatsReqBean,
      selDeselectSeatsRespBean);

}// end if(0)


// secure wallet secondary payment (e-money + rest using secure
    wallet)
if(selectedTicketPaymentMethod(eMoneyPayMethodUI) == 1){

  pinTrials = UpdateWalletGUI.walletAuthenticationTicketPayment(
      display,
      backScreen,
      ticketsDiscountValue,
      eMoneyPayMethodValues[2],
      walletPin,
```

```
        String.valueOf((Double.parseDouble(amountToPay) - Double.
            parseDouble(eMoney))),
        pinTrials,
        cinemaTickets,
        cineHallConfReqBean,
        cineHallConfRespBean,
        selDeselectSeatsReqBean,
        selDeselectSeatsRespBean);

  }// end if(1)


  // new credit card data secondary payment (e-money + rest using a new
       credit card data)
  if(selectedTicketPaymentMethod(eMoneyPayMethodUI) == 2){

    verifyCreditCard(
        display,
        backScreen,
        ticketsDiscountValue,
        eMoneyPayMethodValues[2],
        ccType,
        ccNo,
        ccValidMonth,
        ccValidYear,
        ccCW2,
        eMoney,
        amountToPay,
        cinemaTickets,
        cineHallConfReqBean,
        cineHallConfRespBean,
        selDeselectSeatsReqBean,
        selDeselectSeatsRespBean);

  }// end if(2)

  return pinTrials;

}// end payUsingSecondaryPaymentMethod()


/**
 * Performs the ticket payment with the server side service
 */
public static void payOverNetwork(
    Display                     display,
    Displayable                 backScreen,
```

```
  String[]                    ticketsDiscountValue,
  String                      paymentMethodValue,
  String                      ccType,
  String                      ccNo,
  String                      ccValidMonth,
  String                      ccValidYear,
  String                      ccCW2,
  String                      totalAmountToPay,
  TicketBean[]                cinemaTickets,
  Cinema_Hall_Conf_Req_Bean   cineHallConfReqBean,
  Cinema_Hall_Conf_Resp_Bean  cineHallConfRespBean,
  Select_Deselect_Seats_Req_Bean selDeselectSeatsReqBean,
  Select_Deselect_Seats_Resp_Bean selDeselectSeatsRespBean) throws
      Exception{


Purchase_Tickets_Req_Bean reqBean = new Purchase_Tickets_Req_Bean();

System.out.println("----------␣PURCHSE␣TICKET␣REQ␣BEAN␣BEFORE");
System.out.println(ccType + "\n" + ccNo + "\n" +ccValidMonth + "\n"
     + ccValidYear + "\n" + ccCW2 + "\n" );
System.out.println("----------␣PURCHSE␣TICKET␣REQ␣BEAN␣AFTER");
System.out.println("*****************************PAYING␣USING␣E-
    MONEY␣-␣TOTAL␣TO␣PAY:␣" + totalAmountToPay);

reqBean.setUserName      (Start.userName);
reqBean.setShowLocationID (selDeselectSeatsReqBean.getShowLocationID
    ());
reqBean.setShowTimeID    (selDeselectSeatsReqBean.getShowTimeID());
reqBean.setSeatsNoRows   (selDeselectSeatsReqBean.getSeatsNoRows());
reqBean.setSeatsNoCols   (selDeselectSeatsReqBean.getSeatsNoCols());
reqBean.setSeats         (selDeselectSeatsReqBean.getSeats());
reqBean.setDiscounts     (ticketsDiscountValue);
reqBean.setReservationDate (cinemaTickets[0].getTKTReservationDate())
    ;

if(paymentMethodValue.equals("At␣the␣Cinema")){
  reqBean.setPurchaseMethod(PurchaseMethosConstants.PM_CINEMA);

}else if(paymentMethodValue.equals("Credit␣Card")){
  reqBean.setPurchaseMethod(PurchaseMethosConstants.PM_CARD);

}else if(paymentMethodValue.equals("E-Money")){
  reqBean.setPurchaseMethod(PurchaseMethosConstants.PM_EMONEY);

}// end if
```

```
    SendMessage sm = new SendMessage(display, Protocol_Step_Constants.
        PRT_STEP_PURCHASE_TICKETS, backScreen, reqBean);
    sm.setCreditCardData(ccType, ccNo, ccValidMonth, ccValidYear, ccCW2);
    sm.setCinemaTickets(cinemaTickets);
    sm.go();

  }// end payOverNetwork()

  /**
   * Set the ticket payment method fucntion of the selected UI values
   */
  private static TicketBean[] setTicketPaymentMethod(TicketBean[]
      cinemaTickets, String paymentMethodValue){

    for(int i= 0; i < cinemaTickets.length; i++){
      if(paymentMethodValue.equals("Secure␣Wallet"))
        cinemaTickets[i].setTKTPurchaseMethod("Credit␣Card");
      else
        cinemaTickets[i].setTKTPurchaseMethod(paymentMethodValue);

    }

    return cinemaTickets;

  }// end setTicketPaymentMethod()

}// end class

package gui.purchasetickets.step6billinginfo;

import java.io.IOException;
import javax.microedition.lcdui.*;

import start.Start;

import model.beans.otherbeans.TicketBean;
import model.beans.responsebeans.Purchase_Tickets_Resp_Bean;



import gui.GenericGUI;
import gui.customdialogwindows.CanvasAlert;
import gui.customdialogwindows.DialogWindow;
import gui.mainmenu.MenuScreen;
import gui.purchasetickets.step4discountandreservationsummary.
    UpdateTicketDiscountAndReservationSummaryScreen;

import constants.CustomAlertTypes;
```

```
/**
 *  Display the billing info after the payment has been done
 *
 *  It extends the GenericGUI super class
 *
 * @author s031288, Mihai Balan
 *
 */
public class BillingInfoGUI extends GenericGUI{

  // the Billing info screen
  private static Displayable screen = null;

  //  the commands
  private static Command mainCommand;
  private static Command exitCommand;

  // UI components
  private StringItem[] ticketInfoUI;
  private StringItem totalPriceUI;
  private StringItem infoMsgUI;
  private Image      imgUp;
  private ImageItem  imgThemeUp;
  private Spacer     spacer;

  private String totalPrice = "";
  private String[] ticketInfoValues = {};

  private TicketBean[] cinemaTickets;
  private Purchase_Tickets_Resp_Bean respBean;



  /**
   * Constructs an instance of the class
   */
  public BillingInfoGUI(){
  }

  public BillingInfoGUI(TicketBean[] cinemaTickets,
      Purchase_Tickets_Resp_Bean respBean){

    this.cinemaTickets = cinemaTickets;
    this.respBean    = respBean;
```

```
}// end 2nd constructor


/**
 * Returns the displayable billing info
 *
 */
public Displayable getScreen() {
  return screen;
}

/**
 * Function of the chosen command
 * the user can go back to the main menu
 * or exit the application
 *
 * @param c The executed command
 * @param s The main menu form
 *
 */
public void commandAction(Command c, Displayable s) {
  try {

    if (c == mainCommand){
      display.setCurrent(new MenuScreen());
    }// end if (c == mainCommand)

    if (c == exitCommand){
      // TO DO - cancel the selected seats before exit
      DialogWindow reallyExit = new DialogWindow(
          display,
          new BillingInfoGUI().getScreen(),
          "Exit␣Application?",
          "Are␣you␣sure␣that␣you␣want␣to␣exit␣the␣application?",
          "question",
          "/dialogIcons/exitTheme",
          new MenuScreen().startingpoint);

      display.setCurrent(reallyExit);

    } // end if (c == exitCommand)

  } catch (Exception e) {
    e.printStackTrace();
    CanvasAlert keyErrorAlert = new CanvasAlert(
        display,
        getScreen(),
```

```
        "Billing␣Info␣Error!",
        "Error␣in␣the␣Billing␣Info␣Screen!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    display.setCurrent(keyErrorAlert);
  }
}

/**
 * Initialize the model an open the record store
 */
protected void initModel() throws Exception {

  // get and set the ticket data from the response bean
  cinemaTickets = BillingInfoHelper.updateTickets(cinemaTickets,
      respBean);

  // save the tickets into RMS
  BillingInfoHelper.saveTicketsToRMS(cinemaTickets, respBean);

  // construct the ticket info to be displayed on the screen
  ticketInfoValues = new String[cinemaTickets.length];
  ticketInfoUI    = new StringItem[cinemaTickets.length];
  totalPrice      = UpdateTicketDiscountAndReservationSummaryScreen.
      formatTotalPrice(respBean.getTotalPrice());
  ticketInfoValues = BillingInfoHelper.buildTicketInfo(cinemaTickets,
      ticketInfoValues, respBean);

} //end initModel()

/**
 * Creates the Ticket Discount Screen
 *
 * @throws Exception
 *
 */
protected void createView() throws Exception {
  mainCommand  = new Command("MAIN␣MENU", Command.OK, 0);
  exitCommand  = new Command("EXIT",   Command.EXIT, 1);

  for(int i = 0; i < ticketInfoValues.length; i++){
    ticketInfoUI[i] = new StringItem("Ticket␣" + (i+1) +":",
        ticketInfoValues[i], Item.PLAIN);
  }

  totalPriceUI = new StringItem("Payed␣Amount:␣", totalPrice + "␣DKK",
```

```
      Item.PLAIN);
  spacer      = new Spacer(100, 10);
  infoMsgUI   = new StringItem("", "", Item.PLAIN);
  infoMsgUI.setText("All␣tickets␣are␣saved␣in␣your␣phone␣memory.␣" +
      "They␣can␣be␣found␣in␣Main␣Menu␣under␣My␣Tickets.␣" +
      "You␣can␣enter␣the␣movies␣by␣using␣the␣saved␣tickets.␣" +
      "You␣can␣also␣pay␣for␣the␣tickets␣at␣the␣cinema␣in␣case␣" +
      "you␣you␣have␣used␣the␣Pay␣At␣Cinema␣payment␣method." +
      "␣In␣case␣you␣cancel␣any␣of␣the␣purchased␣tickets,␣the␣ticket" +
      "␣price␣is␣refunded␣using␣electronic␣money␣that␣you␣can␣use␣to␣
          purchase␣new␣tickets.");


  try{
    imgUp = Image.createImage("/" + Start.themeDir + "/billing_info/
        billingInfoTheme.png");
    imgThemeUp = new ImageItem("", imgUp, Item.LAYOUT_TOP | Item.
        LAYOUT_CENTER , "Theme_Img_Up");

  }catch(IOException ioe){
    System.out.println("Ticket␣Discount␣image␣exception!");
  }

  screen = new Form("Billing␣Details");
  ((Form)screen).append(imgUp);
  ((Form)screen).append(totalPriceUI);
  for(int i = 0; i < ticketInfoValues.length; i++){
    ((Form)screen).append(ticketInfoUI[i]);
  }
  ((Form)screen).append(spacer);
  ((Form)screen).append(infoMsgUI);

  // add the commands to the form
  screen.addCommand(mainCommand);
  screen.addCommand(exitCommand);

}

/**
 * Update the view - maybe refresh the fields
 *
 */
protected void updateView() throws Exception {
  initModel();
  createView();
}
```

```
}// end class

package gui.purchasetickets.step6billinginfo;

import model.beans.otherbeans.TicketBean;
import model.beans.responsebeans.Purchase_Tickets_Resp_Bean;
import rms.RMSOperations;
import start.Start;
import gui.purchasetickets.step4discountandreservationsummary.
    UpdateTicketDiscountAndReservationSummaryScreen;

/**
 * Performs several helping operation after the ticket payment has been
     done
 * e.g. updating the billing info UI, saving the tickets in RMS, etc
 * @author Mihai Balan, s031288
 *
 */
public class BillingInfoHelper {

  /**
   * Updates the tickets info on the billing UI
   */
  public static TicketBean[] updateTickets(
      TicketBean[]             cinemaTickets,
      Purchase_Tickets_Resp_Bean respBean){

    String tiktPrice = "";
    for(int i = 0; i < cinemaTickets.length; i++){
      cinemaTickets[i].setTKTReservationID(respBean.getReservationID());
      cinemaTickets[i].setTKTID(respBean.getTicketIDs()[i]);
      tiktPrice = UpdateTicketDiscountAndReservationSummaryScreen.
          formatTotalPrice(respBean.getTicketPrices()[i]);
      cinemaTickets[i].setTKTPrice(tiktPrice);
      cinemaTickets[i].setTKTCinemaTheater("");
      cinemaTickets[i].setTKTStatus("");

    }// end for()

    return cinemaTickets;

  }// end updateTickets()


  /**
   * Save the tickets to RMS
   */
  public static void saveTicketsToRMS(
```

```
  TicketBean[]              cinemaTickets,
  Purchase_Tickets_Resp_Bean respBean) throws Exception{

for(int i = 0; i < cinemaTickets.length; i++){
  RMSOperations.writeByteItem("TT" + (Start.maxTTSaved + i) + ":",
      cinemaTickets[i]);
}

// update the no of tickets saved in RMS
RMSOperations.deleteItems("TTN:");
RMSOperations.writeRecord("TTN:", String.valueOf((Start.maxTTSaved +
    cinemaTickets.length)));

// save the e-money to RMS
RMSOperations.deleteItems("EMN:");
RMSOperations.writeRecord("EMN:",
    UpdateTicketDiscountAndReservationSummaryScreen.formatTotalPrice(
    respBean.getLeftEmoney()));

// if there are no tickets saved in the memory
if(Start.maxTTSaved == 0){
  Start.tickets = null;
  Start.maxTTSaved += cinemaTickets.length;
  Start.tickets = new TicketBean[Start.maxTTSaved];

  for(int i = 0; i < cinemaTickets.length; i++){
    Start.tickets[i] = new TicketBean();
    Start.tickets[i] = cinemaTickets[i];

  }// end for()

}// end if(Start.maxTTSaved == 0)

// if there are already tickets in the memory
else if(Start.maxTTSaved > 0){

  TicketBean[] tempTickets = new TicketBean[Start.maxTTSaved];

  for(int i = 0; i < Start.maxTTSaved; i++){
    tempTickets[i] = new TicketBean();
    tempTickets[i] = Start.tickets[i];

  }// end for()

  Start.tickets = null;
  Start.maxTTSaved += cinemaTickets.length;
```

```
      Start.tickets = new TicketBean[Start.maxTTSaved];

      for(int i = 0; i < tempTickets.length; i++){
        Start.tickets[i] = new TicketBean();
        Start.tickets[i] = tempTickets[i];
      }

      for(int i = tempTickets.length; i < Start.maxTTSaved; i++){
        Start.tickets[i] = new TicketBean();
        Start.tickets[i] = cinemaTickets[i - tempTickets.length];
      }

    }// end if(Start.maxTTSaved > 0)

  }// end saveTicketsToRMS()


  /**
   * Build the ticket info details from the ticket bean and response bean
   */
  public static String[] buildTicketInfo(
      TicketBean[]             cinemaTickets,
      String[]                 ticketInfoValues,
      Purchase_Tickets_Resp_Bean respBean){

    for(int i = 0; i < ticketInfoValues.length; i++){
      ticketInfoValues[i] =
        "Cinema:␣" +
        cinemaTickets[i].getTKTCinema()     + "␣(" +
        cinemaTickets[i].getTKTCinemaAddress() + "),␣Movie:␣" +
        cinemaTickets[i].getTKTMovie()      + "␣(" +
        cinemaTickets[i].getTKTShowDate()   + "␣-␣" +
        cinemaTickets[i].getTKTShowHour()   + "),␣Row:␣" +
        cinemaTickets[i].getTKTRow()        + ",␣Seat:␣" +
        cinemaTickets[i].getTKTSeat()       + ",␣Discount␣Type:␣" +
        cinemaTickets[i].getTKTDiscountType() + ",␣Price:␣" +
        cinemaTickets[i].getTKTPrice()      + "␣DKK,␣Payment␣Method:␣" +
        cinemaTickets[i].getTKTPurchaseMethod();

    }// end for()

    return ticketInfoValues;

  }// end buildTicketInfo()

}// end class

package gui.ratemovie;
```

```
import java.io.IOException;

import javax.microedition.lcdui.Canvas;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.ImageItem;
import javax.microedition.lcdui.Item;

import constants.Protocol_Step_Constants;

import model.beans.requestbeans.Rate_Movie_Req_Bean;
import networkoperations.SendMessage;

import rms.RMSOperations;
import start.Start;
import gui.GUIHelper;

/**
 * Builds the Rate Movie UI
 *
 * @author Mihai Balan, s031288
 *
 */
public class RateMovieGUI extends Canvas {

  // the display to draw on
  private Display   display;
  private Displayable next;
  private Graphics g;

  private Image   scaleImg;
  private ImageItem scaleImgItem;
  private Image   cursorImg;
  private ImageItem cursorImgItem;
  private Image   scoreImg;
  private ImageItem scoreImgItem;
  private Image   themeImg;

  private int showLocationID;
  private Start midlet;

  // the curent selected option i.e. CANCEL, BACK
  private int selectedOptionIndex = 0;
```

```
// NOT highlighted buttons
private String[] optionDeselected = {
    "/theme_red/movie_rating/voteDeselected.png",
    "/theme_red/movie_rating/backDeselected.png"
};

// the highlighted buttons
private String[] optionSelected = {
    "/theme_red/movie_rating/voteSelected.png",
    "/theme_red/movie_rating/backSelected.png"
};

// the images for building the YES and NO options
private Image[] deselectedImgs;
private Image[] selectedImgs;
private int step = 0;
private int count = 5;


public RateMovieGUI( Display display, Displayable next, int
     showLocationID){

  this.display      = display;
  this.next         = next;
  this.showLocationID = showLocationID;

  selectedImgs  = GUIHelper.createSelectedButtons(optionSelected);
  deselectedImgs = GUIHelper.createDeselectedButtons(optionDeselected);

  display.setCurrent(this);
}


/**
 * The image and text displaying takes place in here
 *
 * @param g The graphics to draw on
 */
protected void paint(Graphics g){

  int w = getWidth();
  int h = getHeight();

  try {

    // clear the background
    g.setColor(255, 255, 255);
```

```
g.fillRect(0, 0, getWidth(), getHeight());

themeImg    = Image.createImage("/theme_red/movie_rating/voteTheme.
    png");

scaleImg    = Image.createImage("/theme_red/movie_rating/scale.png"
    );
scaleImgItem = new ImageItem("", themeImg, Item.LAYOUT_VCENTER, "
    scale");

cursorImg    = Image.createImage("/theme_red/movie_rating/cursor.png
    ");
cursorImgItem = new ImageItem("", themeImg, Item.LAYOUT_VCENTER , "
    cursor");

scoreImg    = Image.createImage("/theme_red/movie_rating/score" +
    count + ".png");
scoreImgItem = new ImageItem("", themeImg, Item.LAYOUT_VCENTER , "
    score");

step = 2*cursorImg.getWidth()/3;

// draw the theme
g.drawImage(
    themeImg,
    w, 0,
    Graphics.TOP | Graphics.RIGHT);

// draw the scale
g.drawImage(
    scaleImg,
    w/2, h/2,
    Graphics.VCENTER | Graphics.HCENTER);

// draw the cursor
g.drawImage(
    cursorImg,
    w/2 + (count-5)*step, h/2,
    Graphics.VCENTER | Graphics.HCENTER);

// draw the score ball
g.drawImage(
    scoreImg,
    w/2, 3*h/4 -15,
    Graphics.VCENTER | Graphics.HCENTER);

// draw the buttons
```

```
    GUIHelper.drawCCViewButtons(g,
        deselectedImgs, selectedImgs,
        w, h,
        w/2, h - 50,
        selectedOptionIndex);

  } catch(IOException ioe){
    System.out.println("Movie␣rating␣image␣exception!");
    // if the image cannot be drawn, write some text
    ioe.printStackTrace();
    g.drawString("Movie␣details␣IOException", w/2, h/2,
        Graphics.BASELINE | Graphics.HCENTER);

  } catch (Exception e) {
    System.out.println("Movie␣rating␣exception!");
    e.printStackTrace();
    g.drawString("Movie␣details␣Exception", w/2, h/2,
        Graphics.BASELINE | Graphics.HCENTER);
  }
}

/**
 * Update the value of the current selected option in the list
 * and repaint the screen.
 *
 * @param keyCode The code of the pressed key
 */
protected void keyPressed( int keyCode ){

  if ((getGameAction(keyCode) == Canvas.LEFT)){

    if(count > 0){
      --count;
      repaint();

    } else if(count == 0){
      count = 10;
      repaint();
    }

  } else if ((getGameAction(keyCode) == Canvas.RIGHT)){

    if(count < 10){
      ++count;
      repaint();

    } else if(count == 10){
```

```
      count = 0;
      repaint();
    }

  } if ((getGameAction(keyCode) == Canvas.UP)){

    if(selectedOptionIndex > 0){

      selectedOptionIndex--;
      repaint();

    } else if(selectedOptionIndex == 0){

      selectedOptionIndex = optionDeselected.length - 1;
      repaint();
    }

  } else if ((getGameAction(keyCode) == Canvas.DOWN)){

    if(selectedOptionIndex < optionDeselected.length - 1){

      selectedOptionIndex++;
      repaint();

    } else if(selectedOptionIndex == optionDeselected.length - 1){

      selectedOptionIndex = 0;
      repaint();
    }

  } else if ((getGameAction(keyCode) == Canvas.FIRE)){

    // show movie description
    if(selectedOptionIndex == 0){
      try{
      // send the requst throught the network to the server side
      //NetworkCommunicationFacade netCommFacade = new
          NetworkCommunicationFacade();

      Rate_Movie_Req_Bean rateMovieReqBean = new Rate_Movie_Req_Bean();
      String userName = RMSOperations.getItem("USR:");
      byte[] password = RMSOperations.getByteItem("PSW:");

      rateMovieReqBean.setUserName(userName);
      rateMovieReqBean.setPassword(password);
      rateMovieReqBean.setShowLocationID(showLocationID);
      rateMovieReqBean.setMovieScore(count);
```

```
        System.out.println(rateMovieReqBean.toString());

        SendMessage sm = new SendMessage(
            display,
            Protocol_Step_Constants.PRT_STEP_RATE_MOVIE,
            next,
            rateMovieReqBean);

        sm.go();

        //Response_Msg_Bean rateMovieRespbean = netCommFacade.rateMovie(
            rateMovieReqBean);

        /*Response_Msg_Bean rateMovieRespBean = TestConstruct.
            constrRateMovie();
        NetworkResposeFacade netRespFacade = new NetworkResposeFacade();

        netRespFacade.displayRateMovieResponse(
            display,
            display.getCurrent(),
            new SelectShowGUI().getScreen(),
            rateMovieReqBean,
            rateMovieRespBean);
            */

        }catch(Exception e){
          e.printStackTrace();
          System.out.println("Exception␣while␣trying␣to␣construct␣" +
              "the␣communication␣data␣for␣rating␣the␣movie!");
        }

      // go back to Select show screen
      }else if(selectedOptionIndex == 1){
        if(midlet == null)
          display.setCurrent(next);
      }

    } // end if(FIRE)

  }// end keyPressed

}// end class

package gui.settings;

import gui.GenericGUI;
import gui.customdialogwindows.CanvasAlert;
```

```
import gui.customdialogwindows.DialogWindow;
import gui.mainmenu.MenuScreen;

import java.io.IOException;
import javax.microedition.lcdui.*;

import networkoperations.SendMessage;

import model.beans.requestbeans.Change_Password_Req_Bean;

import start.Start;
import constants.CustomAlertTypes;
import constants.Protocol_Step_Constants;
import cryptography.Encryptor;

/**
 *  Displays the screen used to allow movie goer to change the
 *  password used to access the application
 *
 *  It extends the GenericGUI super class
 *
 * @author Mihai Balan (s031288)
 *
 */
public class ChangePasswordGUI extends GenericGUI{

  // the main screen
  private static Displayable screen = null;

  //the starting point of the application
  public static Start startingPoint;


  //  the exit and select commands
  private static Command changeCommand;
  private static Command backCommand;
  private static Command mainCommand;
  private static Command helpCommand;
  private static Command exitCommand;

  // user, password, and key text boxes
  private TextField oldPassw;
  private TextField newPassw;
  private TextField newPasswVerify;
  private Image   imgUp;
  private ImageItem imgThemeUp;
```

```
private CanvasAlert alert;

/** reference to the Encryptor class in order
 to perform encryption/decryption operations */
public Encryptor encryptor = null;

Change_Password_Req_Bean changePswdReqBean;


/**
 * Constructs an instance of the class
 */
public ChangePasswordGUI() {}

/**
 * Returns the displayable change password screen
 *
 */
public Displayable getScreen() {
  return screen;
}

/**
 * Function of the pressed button i.e. exit or change user password
 *
 * @param c The executed command
 * @param s The main menu form
 *
 */
public void commandAction(Command c, Displayable s) {

  try {

    if (c == changeCommand) {

      if(!newPassw.getString().equals(newPasswVerify.getString())){
        alert = new CanvasAlert(
            display,
            getScreen(),
            "The␣passwords␣do␣not␣match!",
            "The␣password␣and␣verify␣password␣do␣not␣match.␣Please␣try␣
                again!!",
            "error",
            CustomAlertTypes.ALERT_ERROR);

      }else{
```

```
    if (newPassw.getString().length() < 8){

      // display an alarm if the psw is shorter that 8 characters
      alert = new CanvasAlert(
          display,
          getScreen(),
          "Invalid␣Password!",
          "The␣password␣has␣to␣be␣at␣least␣8␣characters␣long!",
          "error",
          CustomAlertTypes.ALERT_ERROR);

    }else{
      // save the password on the server side and to RMS
      changePswdReqBean = new Change_Password_Req_Bean();
      changePswdReqBean.setUserName(Start.userName);

      SendMessage sm = new SendMessage(display,
          Protocol_Step_Constants.PRT_STEP_CHANGE_PASSWORD,
          getScreen(), changePswdReqBean);
      sm.setChangePasswordData(oldPassw.getString(), newPassw.
          getString());
      sm.go();

    }// end if (newPassw.getString().length() < 4)

  }// end if(!newPassw.getString().equals(newPasswVerify.getString
      ())){

}else if (c == backCommand) {
  display.setCurrent(new SettingsGUI().prepareScreen());
  //clean();

}else if (c == mainCommand) {
  display.setCurrent(new MenuScreen());
  //clean();

}else if (c == helpCommand) {
  alert = new CanvasAlert(
      display,
      getScreen(),
      "Change␣Password␣Help",
      "Allows␣to␣change␣the␣password␣used␣to␣access␣the␣application␣
          and␣login␣to␣the␣remote␣service!",
      "question",
      CustomAlertTypes.ALERT_INFO);

}else if (c == exitCommand) {
```

```
      DialogWindow reallyExit = new DialogWindow(
          display,
          getScreen(),
          "Exit␣Application?",
          "Are␣you␣sure␣that␣you␣want␣to␣exit␣the␣application?",
          "question",
          "/dialogIcons/exitTheme",
          new MenuScreen().startingpoint);
      display.setCurrent(reallyExit);

    }// end if(c== command)

  } catch (Exception e) {
    e.printStackTrace();
    alert = new CanvasAlert(
        display,
        getScreen(),
        "Change␣Password␣Error!",
        "Error␣while␣trying␣to␣change␣the␣password!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    display.setCurrent(alert);
  }

}// end CommandAction()


protected void initModel() throws Exception {
}// end initModel()


/**
 * Creates the Screen
 *
 * @throws Exception
 *
 */
protected void createView() throws Exception {

  changeCommand = new Command("CHANGE", Command.EXIT, 0);;
  backCommand  = new Command("BACK",   Command.SCREEN, 1);;
  mainCommand  = new Command("MAIN␣MENU", Command.SCREEN, 2);;
  helpCommand  = new Command("HELP",   Command.SCREEN, 3);;
  exitCommand  = new Command("EXIT",   Command.SCREEN, 4);

  // create the text fields and add them to the form
```

```
    oldPassw      = new TextField("Old␣Password:␣", "", 40, TextField.ANY
        );
    newPassw      = new TextField("New␣Password:␣", "", 40, TextField.ANY
        );
    newPasswVerify = new TextField("Verify␣Password:␣", "", 40, TextField
        .ANY);

    try{
      imgUp = Image.createImage("/" + Start.themeDir + "/authentication/
          theme_up.png");
      imgThemeUp = new ImageItem("", imgUp, Item.LAYOUT_TOP | Item.
          LAYOUT_CENTER , "Theme_Img_Up");

    }catch(IOException ioe){
      System.out.println("Theme␣image␣exception!");
    }

    screen = new Form("Change␣Password!");
    ((Form)screen).append(imgUp);
    ((Form)screen).append(oldPassw);
    ((Form)screen).append(newPassw);
    ((Form)screen).append(newPasswVerify);

    // add the commands to the form
    screen.addCommand(changeCommand);
    screen.addCommand(backCommand);
    screen.addCommand(mainCommand);
    screen.addCommand(helpCommand);
    screen.addCommand(exitCommand);

  }

  /**
   * Update the view - maybe refresh the fields
   *
   */
  protected void updateView() throws Exception {}

}// end class

package gui.settings;

import java.io.IOException;
import javax.microedition.lcdui.*;

import start.Start;

import gui.GenericGUI;
```

```
import gui.customdialogwindows.CanvasAlert;
import gui.customdialogwindows.DialogWindow;
import gui.mainmenu.MenuScreen;
import constants.CustomAlertTypes;


/**
 *  Displays the help main screen
 *
 *  It extends the GenericGUI super class
 *
 * @author s031288, Mihai Balan
 *
 */
public class ChangeThemeGUI extends GenericGUI{

  // the help screen
  private static Displayable screen = null;

  //  the commands
  private static Command changeCommand;
  private static Command backCommand;
  private static Command mainCommand;
  private static Command helpCommand;
  private static Command exitCommand;

  // UI components
  private ChoiceGroup themeUI;
  private Image     imgUp;
  private ImageItem imgThemeUp;
  private String[]  themeValues = {"Red␣Theme", "Blue␣Theme"};
  private Image[]   themeImages  = null;
  private CanvasAlert alert;

  /**
   * Constructs an instance of the class
   */
  public ChangeThemeGUI(){
  }


  /**
   * Returns the displayable MainHelpGUI screen
   * @return screen Returns the MainHelpGUI screen
   */
  public Displayable getScreen() {
    return screen;
```

```
}

/**
 *
 * @param c The executed command
 * @param s The main menu form
 *
 */
public void commandAction(Command c, Displayable s) {
  try {
    if (c == changeCommand) {

      UpdateSettings.changeTheme(display, themeUI);

    }else if (c == backCommand) {
      display.setCurrent(new SettingsGUI().prepareScreen());
      //clean();

    }else if (c == mainCommand) {
      display.setCurrent(new MenuScreen());
      //clean();

    }else if (c == helpCommand) {
      alert = new CanvasAlert(
          display,
          getScreen(),
          "Change␣Theme␣Help",
          "Allows␣to␣change␣the␣application␣theme␣by␣choosing␣among␣
              several␣themes!",
          "question",
          CustomAlertTypes.ALERT_INFO);

    }else if (c == exitCommand) {
      DialogWindow reallyExit = new DialogWindow(
          display,
          getScreen(),
          "Exit␣Application?",
          "Are␣you␣sure␣that␣you␣want␣to␣exit␣the␣application?",
          "question",
          "/dialogIcons/exitTheme",
          new MenuScreen().startingpoint);
      display.setCurrent(reallyExit);

    }// end if(c== command)

  } catch (Exception e) {
    e.printStackTrace();
```

```
    CanvasAlert keyErrorAlert = new CanvasAlert(
        display,
        getScreen(),
        "Change␣Theme␣Error!",
        "Error␣in␣the␣Change␣Theme␣Screen!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    display.setCurrent(keyErrorAlert);
  }
}


/**
 * Initialize the model
 */
protected void initModel() throws Exception {
  themeImages = new Image [themeValues.length];
  themeImages[0] = Image.createImage("/theme_images/red.png");
  themeImages[1] = Image.createImage("/theme_images/blue.png");

} //end initModel()

/**
 * Creates the Ticket Discount Screen
 *
 * @throws Exception
 *
 */
protected void createView() throws Exception {
  changeCommand = new Command("CHANGE", Command.EXIT, 0);;
  backCommand  = new Command("BACK",   Command.SCREEN, 1);;
  mainCommand  = new Command("MAIN␣MENU", Command.SCREEN, 2);;
  helpCommand  = new Command("HELP",   Command.SCREEN, 3);;
  exitCommand  = new Command("EXIT",   Command.SCREEN, 4);

  themeUI = new ChoiceGroup("Themes:", Choice.EXCLUSIVE, themeValues,
      themeImages);

  try{
    imgUp = Image.createImage("/" + Start.themeDir + "/settings/
        ChangeTheme.png");
    imgThemeUp = new ImageItem("", imgUp, Item.LAYOUT_TOP | Item.
        LAYOUT_CENTER , "Theme_Img_Up");

  }catch(IOException ioe){
    System.out.println("Ticket␣Discount␣image␣exception!");
```

```
    }

    screen = new Form("Help");
    ((Form)screen).append(imgUp);
    ((Form)screen).append(themeUI);

    // add the commands to the form
    screen.addCommand(changeCommand);
    screen.addCommand(backCommand);
    screen.addCommand(mainCommand);
    screen.addCommand(helpCommand);
    screen.addCommand(exitCommand);

  }

  /**
   * Update the view - maybe refresh the fields
   *
   */
  protected void updateView() throws Exception {
    initModel();
    createView();
  }

}// end class

package gui.settings;

import gui.GenericGUI;
import gui.customdialogwindows.CanvasAlert;
import gui.customdialogwindows.DialogWindow;
import gui.mainmenu.MenuScreen;

import java.io.IOException;
import javax.microedition.lcdui.*;

import rms.RMSOperations;
import start.Start;
import constants.CustomAlertTypes;
import cryptography.Encryptor;

/**
 *  Displays the screen used to allow movie goer to change the
 *  PIN  used to access the secure wallet
 *
 *  It extends the GenericGUI super class
 *
 * @author Mihai Balan (s031288)
```

```
 *
 */
public class ChangeWalletPINGUI extends GenericGUI{

  // the main screen
  private static Displayable screen = null;

  //the starting point of the application
  public static Start startingPoint;


  //  the exit and select commands
  private static Command changeCommand;
  private static Command backCommand;
  private static Command mainCommand;
  private static Command helpCommand;
  private static Command exitCommand;

  // user, password, and key text boxes
  private TextField oldPassw;
  private TextField newPassw;
  private TextField newPasswVerify;
  private Image   imgUp;
  private ImageItem imgThemeUp;

  private CanvasAlert alert;
  private boolean normalLogin = true;

  // the number of times a user can enter the
  // old PIN wrong. If the old PIN is entered wrong more then
  // 3 times, My Wallet content is deleted and the PIN reset.
  public static int newPinTrials = 3;


  /** reference to the Encryptor class in order
   to perform encryption/decryption operations */
  public Encryptor encryptor = null;

  /**
   * Constructs an instance of the class
   */
  public ChangeWalletPINGUI() {}

  /**
   * Returns the displayable change pin screen
   *
   */
```

```
public Displayable getScreen() {
  return screen;
}

/**
 * Function of the pressed button i.e. exit or change PIN
 *
 * @param c The executed command
 * @param s The main menu form
 *
 */
public void commandAction(Command c, Displayable s) {

  try {

    if (c == changeCommand) {

      if(!newPassw.getString().equals(newPasswVerify.getString())){
        alert = new CanvasAlert(
            display,
            getScreen(),
            "The␣PINs␣do␣not␣match!",
            "The␣PIN␣and␣verify␣PIN␣do␣not␣match.␣Please␣try␣again!!",
            "error",
            CustomAlertTypes.ALERT_ERROR);

      }else{

        if (newPassw.getString().length() < 3){

          // display an alarm if the pin is shorter than 3 characters
          alert = new CanvasAlert(
              display,
              getScreen(),
              "Invalid␣PIN!",
              "The␣PIN␣has␣to␣be␣at␣least␣3␣characters␣long!",
              "error",
              CustomAlertTypes.ALERT_ERROR);

        }else{

          verifyAndSavePIN();

        }// end if (newPassw.getString().length() < 3)

      }// end if(!newPassw.getString().equals(newPasswVerify.getString
          ()){
```

```
    }else if (c == backCommand) {
      display.setCurrent(new SettingsGUI().prepareScreen());

    }else if (c == mainCommand) {
      display.setCurrent(new MenuScreen());

    }else if (c == helpCommand) {
      alert = new CanvasAlert(
          display,
          getScreen(),
          "Change␣Wallet␣PIN␣Help",
          "Allows␣to␣change␣the␣PIN␣used␣to␣access␣the␣secure␣wallet!",
          "question",
          CustomAlertTypes.ALERT_INFO);

    }else if (c == exitCommand) {
      DialogWindow reallyExit = new DialogWindow(
          display,
          getScreen(),
          "Exit␣Application?",
          "Are␣you␣sure␣that␣you␣want␣to␣exit␣the␣application?",
          "question",
          "/dialogIcons/exitTheme",
          new MenuScreen().startingpoint);
      display.setCurrent(reallyExit);

    }// end if(c== command)

  } catch (Exception e) {
    e.printStackTrace();
    alert = new CanvasAlert(
        display,
        getScreen(),
        "Change␣PIN␣Error!",
        "Error␣while␣trying␣to␣change␣the␣secure␣wallet␣PIN!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    display.setCurrent(alert);
  }

}// end CommandAction()


protected void initModel() throws Exception {
```

```
  // if PIN code not found in RMS (user logs in for the first time in
      RMS)
  if(!Start.walletPin.equals("")){
    normalLogin = true;
  } else {
    normalLogin = false;
  }
}// end initModel()


/**
 * Creates the Screen
 *
 * @throws Exception
 *
 */
protected void createView() throws Exception {

  changeCommand = new Command("CHANGE", Command.EXIT, 0);;
  backCommand  = new Command("BACK",   Command.SCREEN, 1);;
  mainCommand  = new Command("MAIN␣MENU", Command.SCREEN, 2);;
  helpCommand  = new Command("HELP",   Command.SCREEN, 3);;
  exitCommand  = new Command("EXIT",   Command.SCREEN, 4);

  // create the text fields and add them to the form
  oldPassw      = new TextField("Old␣PIN:␣", "", 40, TextField.ANY);
  newPassw      = new TextField("New␣PIN:␣", "", 40, TextField.ANY);
  newPasswVerify = new TextField("Verify␣PIN:␣", "", 40, TextField.ANY)
      ;

  try{
    imgUp = Image.createImage("/" + Start.themeDir + "/authentication/
        theme_up.png");
    imgThemeUp = new ImageItem("", imgUp, Item.LAYOUT_TOP | Item.
        LAYOUT_CENTER , "Theme_Img_Up");

  }catch(IOException ioe){
    System.out.println("Theme␣image␣exception!");
  }

  screen = new Form("Change␣Secure␣Wallet␣PIN!");
  ((Form)screen).append(imgUp);
  ((Form)screen).append(oldPassw);
  ((Form)screen).append(newPassw);
  ((Form)screen).append(newPasswVerify);

  // add the commands to the form
```

```
  screen.addCommand(changeCommand);
  screen.addCommand(backCommand);
  screen.addCommand(mainCommand);
  screen.addCommand(helpCommand);
  screen.addCommand(exitCommand);

}

/**
 * Update the view
 *
 */
protected void updateView() throws Exception {
  initModel();
  createView();
}



/**
 * Check if the old pin is valid and change the pin code
 * In case the old pin is entered wrong more than 3 times,
 * te pin and secure wallet are reset
 *
 * @throws Exception
 */
private void verifyAndSavePIN() throws Exception{

  if(oldPassw.getString().equals(Start.walletPin)){

    // save the encrypted PIN code into RMS
    RMSOperations.writeEncryptedRecord("PIN:", newPassw.getString().
        getBytes());
    Start.walletPin = newPassw.getString();

    alert = new CanvasAlert(
        display,
        new SettingsGUI().prepareScreen(),
        "PIN␣code␣Set␣up!",
        "Your␣Wallet␣has␣been␣setup␣to␣use␣the␣new␣PIN␣code!",
        "OK",
        CustomAlertTypes.ALERT_INFO);

  }else{

    if (newPinTrials > 0){
```

```
        --newPinTrials;
        CanvasAlert ss = new CanvasAlert(
            display,
            getScreen(),
            "Invalid␣PIN␣Code!",
            "Invalid␣PIN␣Code!␣You␣can␣try␣" + newPinTrials + "␣more␣times
                !",
            "error",
            CustomAlertTypes.ALERT_ERROR);
      }

      // in case a user enters the old PIN wrong more then 3 time
      // reset the wallet and pin
      if (newPinTrials == 0){

        //RMSOperations.resetMyWallet();

        CanvasAlert ss = new CanvasAlert(
            display,
            new SettingsGUI().prepareScreen(),
            "My␣Wallet␣reseted!",
            "The␣PIN␣has␣been␣entered␣wrong␣more␣then␣3␣times.␣My␣Wallet␣
                content␣has␣been␣reseted!",
            "error",
            CustomAlertTypes.ALERT_ERROR);
      }// end if(newPinTrials == 0)

    }// end if(oldPassw.getString().equals(Start.walletPin))

  }// end verifyAndSavePIN()

}// end class

package gui.settings;

import java.io.IOException;
import javax.microedition.lcdui.*;

import networkoperations.SendMessage;

import model.beans.otherbeans.TicketBean;
import model.beans.requestbeans.Cinema_Hall_Conf_Req_Bean;
import model.beans.requestbeans.Reject_Payment_Req_Bean;
import model.beans.requestbeans.Select_Deselect_Seats_Req_Bean;
import model.beans.responsebeans.Cinema_Hall_Conf_Resp_Bean;
import model.beans.responsebeans.Select_Deselect_Seats_Resp_Bean;
```

```
import rms.RMSOperations;
import start.Start;

import gui.GenericGUI;
import gui.customdialogwindows.CanvasAlert;
import gui.customdialogwindows.DialogWindow;
import gui.mainmenu.MenuScreen;

import constants.CreditCardTypes;
import constants.CustomAlertTypes;
import constants.Protocol_Step_Constants;

import cryptography.Encryptor;


/**
 *  Allows customer to select the payment method
 *  to pay for the reserve tickets i.e. at the cinema,
 *  existing credit cards in the secure wallet, refunded e-money,
 *  or new credit card
 *
 *  It extends the GenericGUI super class
 *
 * @author s031288, Mihai Balan
 *
 */
public class SettingsGUI extends GenericGUI{

  // the main screen
  private static Displayable screen = null;

  //  the commands
  private static Command submitCommand;
  private static Command mainCommand;
  private static Command helpCommand;
  private static Command exitCommand;

  // UI components
  private ChoiceGroup settingsUI;
  private Image     imgUp;
  private ImageItem imgThemeUp;

  private String[] settingsValues = {"Change␣Application␣Password", "
      Change␣My␣Wallet␣PIN", "Change␣Theme", "Change␣Language"};

  /**
```

```
 * Constructs an instance of the class
 */
public SettingsGUI(){}



public Displayable getScreen() {
  return screen;
}

/**
 * Chosen command
 *
 * @param c The executed command
 * @param s The main menu form
 *
 */
public void commandAction(Command c, Displayable s) {
  try {

    if (c == submitCommand){
      UpdateSettings.applySettings(display, settingsUI);

    }// end if (c == submitCommand)


    if (c == mainCommand){
      display.setCurrent(new MenuScreen());

    }// end if (c == mainCommand)

    if (c == helpCommand){
      CanvasAlert help = new CanvasAlert(
          display,
          display.getCurrent(),
          "Settings␣Help",
          "The␣application␣access␣password␣and␣my␣secure␣wallet␣PIN␣code
              ␣can␣be␣changed␣in␣here.␣You␣can␣also␣choose␣among␣several
              ␣themes.",
          "question",
          CustomAlertTypes.ALERT_INFO);

    }// end if (c == helpCommand)

    if (c == exitCommand){
      DialogWindow reallyExit = new DialogWindow(
          display,
```

```
          getScreen(),
          "Exit␣Application?",
          "Are␣you␣sure␣that␣you␣want␣to␣exit␣the␣application?",
          "question",
          "/dialogIcons/exitTheme",
          new MenuScreen().startingpoint);
      display.setCurrent(reallyExit);

    } // end if (c == exitCommand)

  } catch (Exception e) {
    e.printStackTrace();
    CanvasAlert alert = new CanvasAlert(
        display,
        getScreen(),
        "Settings␣Error!",
        "Error␣in␣the␣SettingstGUI␣Screen!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    display.setCurrent(alert);
  }
}

/**
 * Initialize the model an access the record store
 */
protected void initModel() throws Exception {

} //end initModel()

/**
 * Creates the ChooseTicketPaymentGUI Screen
 *
 * @throws Exception
 *
 */
protected void createView() throws Exception {
  submitCommand = new Command("GO", Command.EXIT, 0);
  mainCommand  = new Command("MAIN␣MENU", Command.SCREEN, 1);
  helpCommand  = new Command("HELP",    Command.SCREEN, 2);
  exitCommand  = new Command("EXIT",    Command.SCREEN, 3);

  settingsUI   = new ChoiceGroup("Setting␣Choices:", Choice.EXCLUSIVE,
      settingsValues, null);

  try{
```

```
      imgUp = Image.createImage("/" + Start.themeDir + "/settings/
          SettingsTheme.png");
      imgThemeUp = new ImageItem("", imgUp, Item.LAYOUT_TOP | Item.
          LAYOUT_CENTER , "Theme_Img_Up");

    }catch(IOException ioe){
      System.out.println("Ticket␣Discount␣image␣exception!");
    }

    screen = new Form("Application␣Settings");
    ((Form)screen).append(imgUp);
    ((Form)screen).append(settingsUI);

    // add the commands to the form
    screen.addCommand(submitCommand);
    screen.addCommand(mainCommand);
    screen.addCommand(helpCommand);
    screen.addCommand(exitCommand);

  }// end createView()

  /**
   * Update the view - maybe refresh the fields
   *
   */
  protected void updateView() throws Exception {
    initModel();
    createView();
  }

}// end class

package gui.settings;

import gui.mainmenu.MenuScreen;

import javax.microedition.lcdui.ChoiceGroup;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;

import rms.RMSOperations;
import start.Start;

public class UpdateSettings {

  /**
   * Get the selected index corresponding to the settings option
   *
```

```
 */
private static int selectedSettingsOption(ChoiceGroup settingsUI){

  boolean[] settingSelected = new boolean[settingsUI.size()];
  settingsUI.getSelectedFlags(settingSelected);
  return settingsUI.getSelectedIndex();

}// end selectedSettingsOption()


/**
 * Get the seleted settings option and display the
 * corresponding screen
 *
 */
public static void applySettings(Display display, ChoiceGroup
    settingsUI) throws Exception{

  switch (selectedSettingsOption(settingsUI)) {

  case 0:
    display.setCurrent(new ChangePasswordGUI().prepareScreen());
    break;

  case 1:
    ChangeWalletPINGUI pinSetup = new ChangeWalletPINGUI();
    ChangeWalletPINGUI.newPinTrials = 3;
    display.setCurrent(pinSetup.prepareScreen());

    break;

  case 2:
    display.setCurrent(new ChangeThemeGUI().prepareScreen());
    break;

  case 3:

    break;

  default:
    break;
  }


}// end applySettings()

/**
```

```
   * Change the application theme based on the user selected theme
   *
   * @param themeUI The list of all themes
   */
  public static void changeTheme(Display display, ChoiceGroup themeUI){

  switch (selectedSettingsOption(themeUI)) {

  case 0:
    Start.themeDir = "theme_red";
    Start.themeName = "red";
    display.setCurrent(new MenuScreen());
    break;

  case 1:
    Start.themeDir = "theme_blue";
    Start.themeName = "blue";
    display.setCurrent(new MenuScreen());
    break;

  default:
    break;
  }

  }// end changeTheme()

}// end class

package gui.splashscreen;

import java.util.*;
import javax.microedition.lcdui.*;



/**
 * Displays an image on the screen for several seconds
 * when the application starts.
 *
 * @author Mihai Balan, s031288
 */
public class SplashScreen extends Canvas {

  // the display to draw on
  private Display   display;

  private Displayable next;
```

```
// a thread used to display the image for a no. of seconds
private Timer timer = new Timer();

Image splash;

int width = 0;
int height = 0;

/**
 * Constructs the splash screen setting the display and
 * the next screen to be displayed after the image
 * @param display The display to draw on
 * @param next The next screen to be displayed after the image
 */
public SplashScreen( Display display, Displayable next ){

  Runtime runtime = Runtime.getRuntime();
  long t = runtime.freeMemory();
  System.out.println("**********************Memory␣before␣clean␣splsh
      ␣screen:" + t);

  this.display = display;
  this.next   = next;
  display.setCurrent(this);

}

/**
 * The image and text displaying takes place in here
 * Get the splash image from a .png file, convert it to a byte array
 * and then display it on the scree
 * @param g The graphocs to draw on
 */
protected void paint( Graphics g ){

  width = getWidth ();
  height = getHeight ();

  try {
    // erase the canvas backgound
    g.setColor(0x00FFFFFF);
    g.fillRect(0, 0, width, height);

    //ImageProcessing imgProcc = new ImageProcessing();
    splash = Image.createImage("/theme_red/splash/splash.png");

    // draw the image
```

```
    g.drawImage(splash, width/2, height/2,
        Graphics.VCENTER | Graphics.HCENTER);

  } catch (Exception e) {
    // if the image cannot be drawn, write some text
    g.drawString("Mobile␣Cinema", width/2, height/2,
        Graphics.BASELINE | Graphics.HCENTER);
  }
}
/**
 * The splash screen disapears when any key is pressed
 * @param keyCode The code of the pressed key
 */
protected void keyPressed( int keyCode ){
  dismiss();
}

/**
 * Displays the image for 8 seconds and then goes to the main menu
 */
protected void showNotify(){
  timer.schedule( new CountDown(), 8000 );
}

/**
 * Cancel the timer and set the screen to the next screen set up before
 *
 */
private void dismiss(){
  timer.cancel();
  display.setCurrent(next);
  clean();
  Runtime runtime = Runtime.getRuntime();
  long t1 = runtime.freeMemory();
  System.out.println("**********************Memory␣after␣clean␣splsh␣
      screen:" + t1);
}

/**
 * Thread used to count down the no. of seconds
 * to display the image
 *
 * @author Mihai Balan, s031288
 *
 */
private class CountDown extends TimerTask {
  public void run(){
```

```
      dismiss();
    }
  }

  private void clean(){
    next = null;
    timer = null;
    splash = null;
    System.gc();


  }
}

package gui;

import gui.customdialogwindows.CanvasAlert;

import javax.microedition.lcdui.*;

import constants.CustomAlertTypes;


/**
 * This is the super class for ProvideKey, WriteMessage,
 * MainMenu, and MyAlert. It implements the common functionality
 * for all this classes. The specific functionlity for
 * each of these classes is implemented in the template methods:
 * initModel(),createView (), updateView (), and commandAction().
 *
 * @author s031288, Mihai Balan
 *
 */
public abstract class GenericGUI implements CommandListener {

  // the screen
  protected static Displayable screen = null;

  // Set from outside at beginning
  public static Display display;


  /**
   *  Returns the screen object from the derived class
   *
   * @return screen The screen object from the derived class
   */
  public abstract Displayable getScreen();
```

```
/**
 * If the screen is displayed for the first time,
 * call the init method for the model and then crete the view.
 * Else, just update the view
 *
 * @return Displyable item to be shown on the screen
 * @throws Exception In case of errors
 */
public Displayable prepareScreen () throws Exception {
  if ( getScreen() == null ){
    initModel();
    createView();
  } else {
    updateView();
  }

  getScreen().setCommandListener ((CommandListener) this);
  return getScreen();
}


/**
 * Displays the prepared screen
 *
 */
public void showScreen(){

  try {
    display.setCurrent(prepareScreen());

  } catch (Exception e){

    e.printStackTrace();

    CanvasAlert alert = new CanvasAlert(
        display,
        getScreen(),
        "Error␣while␣displaying␣the␣screen",
        "An␣error␣occured␣while␣the␣screen␣was␣displayed",
        "error",
        CustomAlertTypes.ALERT_ERROR);
  }

} // end showScreen()
```

```
  /**
   *  Initialize the model
   */
  protected abstract void initModel () throws Exception;

  /**
   * Create the view to be displayed
   *
   * @throws Exception
   */
  protected abstract void createView () throws Exception;

  /**
   * Update the view e.g. when an operation affect the components
   * displayed on the screen
   *
   * @throws Exception
   */
  protected abstract void updateView () throws Exception;

  /**
   * Check the chosen command and performs the corespoding action
   */
  public abstract void commandAction(Command c, Displayable s);

}

package gui;

import java.io.IOException;

import javax.microedition.lcdui.Font;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.Image;

/**
 * Performs different helping operations for the GUI elements
 * e.g. drawing the screen background, icons, buttons, messages, etc
 *
 * @author Mihai Balan
 *
 */
public class GUIHelper{

  /**
   * Generates and draws the background
   *
```

```
 * @param g       The graphical object to paint on
 * @param backImage The background image to be used
 * @param width   The width of the canvas
 * @param height  The height of the canvas
 */
public static void drawBackground(Graphics g,
    String bckgImgName,
    int width, int height) throws IOException{

  // erase the canvas backgound
  g.setColor(0x00FFFFFF);
  g.fillRect(0, 0, width, height);

  // create and draw the background
  Image backImage = Image.createImage(bckgImgName + ".png");

  g.drawImage(
      backImage,
      0, 0,
      Graphics.TOP | Graphics.LEFT);

}// end drawBackground()


/**
 * Generates and draws the icon
 *
 * @param g       The graphical object to paint on
 * @param iconName The icon image to be used
 * @param width   The width of the canvas
 * @param height  The height of the canvas
 */
public static void drawIcon(Graphics g,
    String iconName,
    int width, int height) throws IOException{

  // create and draw the icon
  Image icon = Image.createImage("/dialogIcons/" + iconName + ".png");

  g.drawImage(
      icon,
      width/2, 2*height/5 + 20,
      Graphics.VCENTER | Graphics.HCENTER);

}// end drawIcon()

/**
```

```
 * Set the font and color of the alert msg, based on
 * the alert type and draw it on the canvas. It also
 * tokenizes the msg into words and make some computations
 * to feet the text and full words on the screen
 *
 * @param msg The message to be displayed on the mobile screen
 */
public static int drawMessage(Graphics g,
    String msg,
    int weight, int height){

  // define the msg font
  Font msgFont = Font.getFont(
      Font.FACE_PROPORTIONAL,
      Font.STYLE_PLAIN,
      Font.SIZE_LARGE);

  // set the message font and color
  g.setFont(msgFont);
  g.setColor(255,0,0);

  // calculate msg to be drawn by tokenizing on words and draw the
      message
  int msgWidth  = msgFont.stringWidth(msg);
  int charHeight = msgFont.getHeight();
  int noRows    = (int)msgWidth/(weight-2) + 1;
  int msgLength = msg.length();

  int from = 0;
  String m = "";
  int to = 0;

  // the postion where the buttons are to be displayed
  int retInt = 0;
  int startY = 11*height/20 + 20;

  for(int i=0; i<noRows; i++){

    if(i == (noRows-1)){
      m = msg.substring(from, msgLength);
      g.drawString(m, weight/2, startY + i*charHeight, Graphics.TOP |
          Graphics.HCENTER);

    } else{
      to = msg.lastIndexOf(32, (i+1)*(int)(i + msgLength)/noRows);
      m = msg.substring(from +(i==0?0:1), to);
      if (msgFont.stringWidth(m) > (weight - 2)){
```

```
        to = msg.lastIndexOf(32, to-i);
      }
      g.drawString(m, weight/2, startY + i*charHeight, Graphics.TOP |
          Graphics.HCENTER);
    }

    from = to;

    if (i == noRows - 1)
      retInt = 11*height/20 + i*charHeight;

  }// end for()

  return retInt;

}// end drawMessage()


/**
 * Set the font and color of the alert msg, based on
 * the alert type and draw it on the canvas. It also
 * tokenizes the msg into words and make some computations
 * to feet the text and full words on the screen
 *
 * @param msg The message to be displayed on the mobile screen
 */
public static int drawMessage(Graphics g,
    String msg,
    int width, int height, int x, int y){

  // define the msg font
  Font msgFont = Font.getFont(
      Font.FACE_PROPORTIONAL,
      Font.STYLE_PLAIN,
      Font.SIZE_SMALL);

  // set the message font and color
  g.setFont(msgFont);
  g.setColor(160, 40, 18);

  // calculate msg to be drawn by tokenizing on words and draw the
      message
  int msgWidth  = msgFont.stringWidth(msg);
  int charHeight = msgFont.getHeight();
  int charWidth = msgFont.getSize();
  System.out.println("size:␣" + charWidth);
  int noRows    = (int)msgWidth/(width-2) + 1;
```

```
  int msgLength = msg.length();

  int from = 0;
  String m = "";
  int to = 0;

  // the postion where the text are to be displayed
  int retInt = 0;
  int startY = y;

  for(int i=0; i<noRows; i++){

    if(i == (noRows-1)){
      m = msg.substring(from, msgLength);
      g.drawString(m, x-2, startY + i*charHeight, Graphics.TOP |
          Graphics.LEFT);

    } else{
      to = msg.lastIndexOf(32, (i+1)*(int)(i + msgLength)/noRows);
      m = msg.substring(from +(i==0?0:1), to);

      if (msgFont.stringWidth(m) > (x - 2)){
        to = msg.lastIndexOf(32, to-i);
      }

      g.drawString(m, x, startY + i*charHeight, Graphics.TOP | Graphics
          .LEFT);
    }

    from = to;

    if (i == noRows - 1)
      retInt = y + i*charHeight;

  }// end for()

  return retInt;

}// end drawMessage()


/**
 * Set the font and color of the msg.
 * Tokenizes the message into words in such way that
 * displays full words on one line. Splits all message
 * into several lines. Each line feets in the screen width.
 *
```

```
 * @param g     The graphical object to draw on
 * @param msgFont The type of font used for drawaing the message
 * @param colorR The red component of the RGB for the color used to
     write the text with
 * @param colorG The green component of the RGB for the color used to
     write the text with
 * @param colorB The blue component of the RGB for the color used to
     write the text with
 * @param msg   The message to be displayed on the mobile screen
 * @param width The width of the screen
 * @param height The height of the screen
 * @param startX The X position where the test starts to be drawn
 * @param stratY The Y position where the test starts to be drawn
 */
public static int dynamicDrawMessage(
    Graphics g,
    Font msgFont,
    int colorR, int colorG, int colorB,
    String msg,
    int width, int height,
    int startX, int startY){

  // set the message font and color
  //g.setFont(msgFont);
  //g.setColor(colorR, colorG, colorB);

  int startPos  = 0;
  int lineLength = 0;
  boolean printed = false;


  // tokenize the string in words
  String[] words = GUIHelper.tokenizeString(msg);

  for (int i = 0; i < words.length; i++){

    if((lineLength + msgFont.stringWidth(words[i] + "␣")) < width){

      lineLength += msgFont.stringWidth(words[i] + "␣");
      printed = false;

    }else{

      //print words from [startPos ... (i-1)]
      String strToPrint = "";
      for (int j = startPos; j < i; j++){
        strToPrint += words[j] + "␣";
```

```
      }

      g.drawString(strToPrint, startX, startY, Graphics.TOP | Graphics.
          LEFT);

      startPos = i;
      lineLength = 0;
      i = i - 1;
      printed = true;
      startY += msgFont.getHeight();

    } // end if

  }// end for()

  if (!printed){
    // print words from [startPos ... (words.length - 1)]
    String strToPrint = "";
    for (int j = startPos; j < words.length; j++){
      strToPrint += words[j] + "␣";
    }

    g.drawString(strToPrint, startX, startY, Graphics.TOP | Graphics.
        LEFT);

  }// end if(!printed)

  return startY + msgFont.getHeight();

}// end drawMessage()


/**
 * Count the no of words in a text message under a genral format.
 * This has to be implemented due to missing String Tokenizer in J2ME
 *
 * @param msg Thse string to count words in
 */
public static int countWords(String msg){

  int startPos = 0;
  int endPos  = 0;
  int count = 0;
  String subStr = "";

  while(startPos < msg.length()){
```

```
    endPos = msg.indexOf(32, startPos);
    if(startPos == endPos){
      ++startPos;
      continue;
    }

    if(endPos == -1){
      ++count;
      break;
    }

    subStr = msg.substring(startPos, endPos);

    if(subStr.equals("␣")){
      continue;

    }else{
      ++count;
    }
    startPos = endPos+1;

  }// end while()

  return count;

}// end countWords()


/**
 * Tokenize a string on space and returns the words
 * as an array of strings
 *
 *  @param msg Thse string to tokenize
 */
public static String[] tokenizeString(String msg){

  int startPos = 0;
  int endPos   = 0;
  int count    = 0;
  String subStr = "";

  String words[] = new String[GUIHelper.countWords(msg)];


  while(startPos < msg.length()){

    //System.out.println("startPos 1: " + startPos);
```

```
    endPos = msg.indexOf(32, startPos);
    if(startPos == endPos){
      ++startPos;
      continue;
    }

    if(endPos == -1){
      words[count] = msg.substring(startPos);
      ++count;
      break;
    }

    subStr = msg.substring(startPos, endPos);

    if(subStr.equals("␣")){
      continue;
    }else{
      words[count] = subStr;
      ++count;
    }

    startPos = endPos+1;

  } // end while()

  return words;

}// end tokenizeString();


/**
 * Set the font and color of the alert title based on
 * the alert type and draw it on the canvas
 *
 * @param title The title to be displayed on the alert screen
 */
public static void drawTitle(Graphics g,
    String title,
    int width, int height){

  //defines the font used for title
  Font titleFont = Font.getFont(
      Font.FACE_PROPORTIONAL,
      Font.STYLE_BOLD,
      Font.SIZE_LARGE);
```

```
  g.setFont(titleFont);
  g.setColor(255,0,0);
  g.drawString(title, width/2, height/4 + 10, Graphics.TOP | Graphics.
      HCENTER);

}// end drawTitle()


/**
 * Creates the deselected images based on the given names
 *
 * @param optionDeselected The name of the deselected images
 * @return The Deselected Images
 */
public static Image[] createDeselectedButtons(String[] optionDeselected
    ){

  //create the selected
  Image[] deselectedImgs = new Image[optionDeselected.length];

  try{
    for (int i = 0; i < optionDeselected.length; i++)
      deselectedImgs[i] = Image.createImage(optionDeselected[i]);

  }catch(IOException ioe){
    System.out.println("Exception␣when␣creating␣the␣selected␣buttons:␣"
        + ioe.getMessage());
  }

  return deselectedImgs;

} // end createSelectedButtons()


/**
 * Creates the selected images based on the given names
 *
 * @param optionSelected The name of the selected images
 * @return The Selected Images
 */
public static Image[] createSelectedButtons(String[] optionSelected){

  //create the selected
  Image[] selectedImgs = new Image[optionSelected.length];

  try{
    for (int i = 0; i < optionSelected.length; i++)
```

```
      selectedImgs[i] = Image.createImage(optionSelected[i]);

  }catch(IOException ioe){
    System.out.println("Exception␣when␣creating␣the␣selected␣buttons:␣"
        + ioe.getMessage());
  }

  return selectedImgs;

} // end createSelectedButtons()


/**
 * Draw the YES and NO Buttons on the canvas function of the selected
     option.
 * A highlighted image is drawn for the coresponding selected image
 *
 * @param g                The graphical object to draw on
 * @param deselectedImgs   The deselected Images
 * @param selectedImgs     The selected Images
 * @param width            The width of the canvas
 * @param height           The height of the canvas
 * @param buttonHeightPos  The y coordinate where the buttons are to be
     drawn
 * @param selectedOptionIndex The current selected option in the YES-NO
     option list
 */
public static void drawButtons(Graphics g,
    Image[] deselectedImgs, Image[] selectedImgs,
    int width, int height,
    int buttonHeightPos,
    int selectedOptionIndex){

  for (int i = 0; i < deselectedImgs.length; i++) {

    // check if the current entry is selected
    // and draw the menu item highlighted
    if (i == selectedOptionIndex) {
      // draw the image
      g.drawImage(selectedImgs[i],
          width/2 + 2*(2*i-1)*deselectedImgs[i].getWidth()/3,
          buttonHeightPos + 17*deselectedImgs[i].getHeight()/10 + 20,
          Graphics.HCENTER | Graphics.VCENTER);

    } else{
      // draw the image
      g.drawImage(deselectedImgs[i],
```

```
          width/2 + 2*(2*i-1)*deselectedImgs[i].getWidth()/3,
          buttonHeightPos + 17*deselectedImgs[i].getHeight()/10 + 20,
          Graphics.HCENTER | Graphics.VCENTER);
    } // end if()

  } // end for()

} // end drawButtons()


/**
 * Draw the YES and NO Buttons on the canvas function of the selected
     option.
 * A highlighted image is drawn for the coresponding selected image
 *
 * @param g                The graphical object to draw on
 * @param deselectedImgs   The deselected Images
 * @param selectedImgs     The selected Images
 * @param width            The width of the canvas
 * @param height           The height of the canvas
 * @param buttonHeightPos  The y coordinate where the buttons are to be
      drawn
 * @param selectedOptionIndex The current selected option in the YES-NO
      option list
 */
public static void drawCCViewButtons(Graphics g,
    Image[] deselectedImgs, Image[] selectedImgs,
    int width, int height,
    int buttonXPos,
    int buttonYPos,
    int selectedOptionIndex){

  for (int i = 0; i < deselectedImgs.length; i++) {

    // check if the current entry is selected
    // and draw the menu item highlighted
    if (i == selectedOptionIndex) {
      // draw the image
      g.drawImage(selectedImgs[i],
          buttonXPos + 2*(2*i-1)*deselectedImgs[i].getWidth()/3,
          buttonYPos + 17*deselectedImgs[i].getHeight()/10,
          Graphics.HCENTER | Graphics.VCENTER);

    } else{
      // draw the image
      g.drawImage(deselectedImgs[i],
          buttonXPos + 2*(2*i-1)*deselectedImgs[i].getWidth()/3,
```

```
            buttonYPos + 17*deselectedImgs[i].getHeight()/10,
            Graphics.HCENTER | Graphics.VCENTER);
      } // end if()

    } // end for()

  } // end drawCCViewButtons()


}// end class

package model.beans.otherbeans;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;


/**
 * Constructs a Credit Card object used for transaction
 * purposes. This object is stored encrypted in RMS by using
 * a (key, value) approach, where the key is the CCNickName
 * and the value is all other propertis of the CC object.
 *
 * @author s031288, Mihai Balan
 *
 */
public class CreditCardBean {

  // Credit Card(CC) properties
  private String CCNickName;
  private String CCBank;
  private String CCEmergencyPhone;
  private String CCOwner;
  private String CCType;
  private String CCNumber;
  private String CCExpDateMonth;
  private String CCExpDateYear;
  private String CCCW2;
  private String CCPIN;

  public CreditCardBean(){}

  // Set methods tpo set the Credit Card properties
  public void setCCNickName(String CCNickName){
    this.CCNickName = CCNickName;
```

```
}

public void setCCBank(String CCBank){
  this.CCBank = CCBank;
}

public void setCCEmergencyPhone(String CCEmergencyPhone){
  this.CCEmergencyPhone = CCEmergencyPhone;
}

public void setCCOwner(String CCOwner){
  this.CCOwner = CCOwner;
}

public void setCCType(String CCType){
  this.CCType = CCType;
}

public void setCCNumber(String CCNumber){
  this.CCNumber = CCNumber;
}

public void setCCExpDateMonth(String CCExpDateMonth){
  this.CCExpDateMonth = CCExpDateMonth;
}

public void setCCExpDateYear(String CCExpDateYear){
  this.CCExpDateYear = CCExpDateYear;
}

public void setCCCW2(String CCCW2){
  this.CCCW2 = CCCW2;
}

public void setCCPIN(String CCPIN){
  this.CCPIN = CCPIN;
}

// Get methods to obtain the Cresit Card properties
public String getCCNickName(){
  return this.CCNickName;
}

public String getCCBank(){
  return this.CCBank;
}
```

```
public String getCCEmergencyPhone(){
  return this.CCEmergencyPhone;
}

public String getCCOwner(){
  return this.CCOwner;
}

public String getCCType(){
  return this.CCType;
}

public String getCCNumber(){
  return this.CCNumber;
}

public String getCCExpDateMonth(){
  return this.CCExpDateMonth;
}

public String getCCExpDateYear(){
  return this.CCExpDateYear;
}

public String getCCCW2(){
  return this.CCCW2;
}

public String getCCPIN(){
  return this.CCPIN;
}

public String toString(){

  return
    this.CCNickName   + ";" +
    this.CCBank       + ";" +
    this.CCOwner      + ";" +
    this.CCType       + ";" +
    this.CCNumber     + ";" +
    this.CCExpDateMonth + ";" +
    this.CCExpDateYear  + ";" +
    this.CCEmergencyPhone + ";" +
    this.CCCW2        + ";" +
    this.CCPIN;
}
```

```
/**
 * Constructs a String[] representation of the CC object
 * in order to be used with output streams to write
 * the whole CC object into RMS
 *
 * @return String array representation of the CC object
 */
private String[] toArray(){

  String[] ccStringArray = new String[10];

  ccStringArray[0] = CCNickName;
  ccStringArray[1] = CCOwner;
  ccStringArray[2] = CCType;
  ccStringArray[3] = CCNumber;
  ccStringArray[4] = CCExpDateMonth;
  ccStringArray[5] = CCExpDateYear;
  ccStringArray[6] = CCCW2;
  ccStringArray[7] = CCPIN;
  ccStringArray[8] = CCBank;
  ccStringArray[9] = CCEmergencyPhone;

  return ccStringArray;

}


/**
 * Constructs a byte representation of a given Credit Card Array Object
 * using output streams to be able to save the CC to the RMS
 *
 * @param CC The Credit Card Array Object
 * @return The byte representation of the Credit Card Array Object
 * @throws IOException
 * @throws org.bouncycastle.crypto.CryptoException In case encryption
     fails
 */
public byte[] getBytes() throws IOException, org.bouncycastle.crypto.
    CryptoException{

  // get the array representation of the CC object
  String[] ccArray = toArray();

  // Write data into an internal byte array
  ByteArrayOutputStream strmBytes = new ByteArrayOutputStream();

  // Write Java data types into the above byte array
```

```
  DataOutputStream strmDataType = new DataOutputStream(strmBytes);

  for (int i = 0; i < ccArray.length; i++)
    // Write the CC properties as a Java String
    strmDataType.writeUTF(ccArray[i]);

  // Clear any buffered data
  strmDataType.flush();

  // Get stream data into byte array to be written into RMS
  byte[] byteCreditCardData = strmBytes.toByteArray();

  strmBytes.close();
  strmDataType.close();

  return byteCreditCardData;

} // end getBytes()


/**
 * Decrypts the encrypted credit card data retrieved from RMS
 * and reads all CC properties using InputStreams.
 * It constructs and returns a new CreditCardBean object
 * to be used later on.
 *
 * @param ccDataEncrypted The encrypted credit card data retrieved from
      RMS
 * @param decryptor 2see Encryptor
 * @return A CreditCardBean object
 * @throws IOException
 * @throws org.bouncycastle.crypto.CryptoException In case decryption
     fails
 */
public CreditCardBean getCCObject(byte[] ccData) throws IOException,
    org.bouncycastle.crypto.CryptoException{

  // Read from the specified byte array
  ByteArrayInputStream strmBytes = new ByteArrayInputStream(ccData);

  // Read Java data types from the above byte array
  DataInputStream strmDataType = new DataInputStream(strmBytes);

  // create the CC object and set its properties based on
  //the primitive data type i.e. String previously saved in RMS
  CreditCardBean ccBean = new CreditCardBean();
```

```
    ccBean.setCCNickName     (strmDataType.readUTF());
    ccBean.setCCOwner        (strmDataType.readUTF());
    ccBean.setCCType         (strmDataType.readUTF());
    ccBean.setCCNumber       (strmDataType.readUTF());
    ccBean.setCCExpDateMonth (strmDataType.readUTF());
    ccBean.setCCExpDateYear  (strmDataType.readUTF());
    ccBean.setCCCW2          (strmDataType.readUTF());
    ccBean.setCCPIN          (strmDataType.readUTF());
    ccBean.setCCBank         (strmDataType.readUTF());
    ccBean.setCCEmergencyPhone (strmDataType.readUTF());

    strmBytes.close();
    strmDataType.close();

    return ccBean;

  }//  end getCCObject()

} // end class

package model.beans.otherbeans;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;

/**
 * Constructs a Ticket object that stores information
 * about the show customer purchased the ticket for
 *
 * @author s031288, Mihai Balan
 *
 */
public class TicketBean {

  private String tktID;
  private String tktReservationID;
  private String tktCinema;
  private String tktCinemaAddress;
  private String tktCinemaTheater;
  private String tktMovie;
  private String tktShowDate;
  private String tktShowHour;
  private String tktSeat;
  private String tktRow;
  private String tktDiscountType;
```

```
private String tktPrice;
private String tktPurchaseMethod;
private String tktReservationDate;
private String tktStatus;


// SET METHODS
public void setTKTID(String tktID){
  this.tktID = tktID;
}

public void setTKTReservationID(String tktReservationID){
  this.tktReservationID = tktReservationID;
}

public void setTKTCinema(String tktCinema){
  this.tktCinema = tktCinema;
}

public void setTKTCinemaAddress(String tktCinemaAddress){
  this.tktCinemaAddress = tktCinemaAddress;
}

public void setTKTCinemaTheater(String tktCinemaTheater){
  this.tktCinemaTheater = tktCinemaTheater;
}

public void setTKTMovie(String tktMovie){
  this.tktMovie = tktMovie;
}

public void setTKTShowDate(String tktShowDate){
  this.tktShowDate = tktShowDate;
}

public void setTKTShowHour(String tktShowHour){
  this.tktShowHour = tktShowHour;
}

public void setTKTSeat(String tktSeat){
  this.tktSeat = tktSeat;
}

public void setTKTRow(String tktRow){
  this.tktRow = tktRow;
}
```

```
public void setTKTDiscountType(String tktDiscountType){
  this.tktDiscountType = tktDiscountType;
}

public void setTKTPrice(String tktPrice){
  this.tktPrice = tktPrice;
}

public void setTKTStatus(String tktStatus){
  this.tktStatus = tktStatus;
}

public void setTKTPurchaseMethod(String tktPurchaseMethod){
  this.tktPurchaseMethod = tktPurchaseMethod;
}

public void setTKTReservationDate(String tktReservationDate){
  this.tktReservationDate = tktReservationDate;
}

// GET METHODS
public String getTKTID(){
  return this.tktID;
}

public String getTKTReservationID(){
  return this.tktReservationID;
}

public String getTKTCinema(){
  return this.tktCinema;
}

public String getTKTCinemaAddress(){
  return this.tktCinemaAddress;
}

public String getTKTCinemaTheater(){
  return this.tktCinemaTheater;
}

public String getTKTMovie(){
  return this.tktMovie;
}

public String getTKTShowDate(){
  return this.tktShowDate;
```

```
}

public String getTKTShowHour(){
  return this.tktShowHour;
}

public String getTKTSeat(){
  return this.tktSeat;
}

public String getTKTRow(){
  return this.tktRow;
}

public String getTKTDiscountType(){
  return this.tktDiscountType;
}

public String getTKTPrice(){
  return this.tktPrice;
}

public String getTKTStatus(){
  return this.tktStatus;
}

public String getTKTPurchaseMethod(){
  return this.tktPurchaseMethod;
}

public String getTKTReservationDate(){
  return this.tktReservationDate;
}


public String toString(){

  return
    "Ticket␣ID:␣␣␣␣␣␣␣␣␣␣␣␣␣" + this.tktID          + ";\n" +
    "Ticket␣ReservationID:␣␣" + this.tktReservationID + ";\n" +
    "Ticket␣Cinema:␣␣␣␣␣␣␣␣" + this.tktCinema      + ";\n" +
    "Ticket␣Cinema␣Address:␣" + this.tktCinemaAddress + ";\n" +
    "Ticket␣Theater:␣␣␣␣␣␣␣" + this.tktCinemaTheater + ";\n" +
    "Ticket␣Movie:␣␣␣␣␣␣␣␣␣" + this.tktMovie     + ";\n" +
    "Ticket␣ShowDate:␣␣␣␣␣␣␣" + this.tktShowDate    + ";\n" +
    "Ticket␣ShowHour:␣␣␣␣␣␣␣" + this.tktShowHour  + ";\n" +
    "Ticket␣Seat:␣␣␣␣␣␣␣␣␣␣" + this.tktSeat        + ";\n" +
```

```
    "Ticket␣Row:␣␣␣␣␣␣␣␣␣␣␣" + this.tktRow       + ";\n" +
    "Ticket␣Discount:␣␣␣␣␣␣␣" + this.tktDiscountType + ";\n" +
    "Ticket␣Price␣␣␣␣␣␣␣␣␣␣" + this.tktPrice       + ";\n" +
    "Ticket␣Purchase:␣␣␣␣␣␣␣" + this.tktPurchaseMethod + ";\n" +
    "Ticket␣Res␣Date:␣␣␣␣␣␣␣" + this.tktReservationDate + ";\n" +
    "Ticket␣Status:␣␣␣␣␣␣␣␣␣" + this.tktStatus      + ";\n";
}

/**
 * Constructs a String[] representation of the TKT object
 * in order to be used with output streams to write
 * the whole TKT object into RMS
 *
 * @return String array representation of the TKT object
 */
private String[] toArray(){

  String[] tktStringArray = new String[15];

  tktStringArray[0] = this.tktID;
  tktStringArray[1] = this.tktReservationID;
  tktStringArray[2] = this.tktCinema ;
  tktStringArray[3] = this.tktCinemaAddress;
  tktStringArray[4] = this.tktCinemaTheater;
  tktStringArray[5] = this.tktMovie ;
  tktStringArray[6] = this.tktShowDate;
  tktStringArray[7] = this.tktShowHour;
  tktStringArray[8] = this.tktSeat;
  tktStringArray[9] = this.tktRow;
  tktStringArray[10] = this.tktDiscountType;
  tktStringArray[11] = this.tktPrice;
  tktStringArray[12] = this.tktPurchaseMethod;
  tktStringArray[13] = this.tktReservationDate;
  tktStringArray[14] = this.tktStatus;

  return tktStringArray;

}


/**
 * Constructs a byte representation of a given TicketArray Object
 * using output streams to be able to save the ticket to the RMS
 *
 * @return The byte representation of the Ticket Array Object
 * @throws IOException
 */
```

```
public byte[] getBytes() throws IOException{

  // get the array representation of the TKT object
  String[] tktArray = toArray();

  // Write data into an internal byte array
  ByteArrayOutputStream strmBytes = new ByteArrayOutputStream();

  // Write Java data types into the above byte array
  DataOutputStream strmDataType = new DataOutputStream(strmBytes);

  for (int i = 0; i < tktArray.length; i++)
    // Write the TKT properties as a Java String
    strmDataType.writeUTF(tktArray[i]);

  // Clear any buffered data
  strmDataType.flush();

  // Get stream data into byte array to be written into RMS
  byte[] byteTKTData = strmBytes.toByteArray();

  strmBytes.close();
  strmDataType.close();

  return byteTKTData;

} // end getBytes()


/**
 * Reads all TKT properties using InputStreams.
 * It constructs and returns a new Ticketbean object
 * to be used later on.
 *
 * @param tktData The ticket data retrieved from RMS
 * @return A TicketBean object
 * @throws IOException
 */
public TicketBean getTKTObject(byte[] tktData) throws IOException{

  // Read from the specified byte array
  ByteArrayInputStream strmBytes = new ByteArrayInputStream(tktData);

  // Read Java data types from the above byte array
  DataInputStream strmDataType = new DataInputStream(strmBytes);

  // create the TKT object and set its properties based on
```

```
    //the primitive data type i.e. String previously saved in RMS
    TicketBean tktBean = new TicketBean();

    tktBean.setTKTID           (strmDataType.readUTF());
    tktBean.setTKTReservationID (strmDataType.readUTF());
    tktBean.setTKTCinema       (strmDataType.readUTF());
    tktBean.setTKTCinemaAddress (strmDataType.readUTF());
    tktBean.setTKTCinemaTheater (strmDataType.readUTF());
    tktBean.setTKTMovie        (strmDataType.readUTF());
    tktBean.setTKTShowDate     (strmDataType.readUTF());
    tktBean.setTKTShowHour     (strmDataType.readUTF());
    tktBean.setTKTSeat         (strmDataType.readUTF());
    tktBean.setTKTRow          (strmDataType.readUTF());
    tktBean.setTKTDiscountType (strmDataType.readUTF());
    tktBean.setTKTPrice        (strmDataType.readUTF());
    tktBean.setTKTPurchaseMethod (strmDataType.readUTF());
    tktBean.setTKTReservationDate (strmDataType.readUTF());
    tktBean.setTKTStatus       (strmDataType.readUTF());

    strmBytes.close();
    strmDataType.close();

    return tktBean;

  }//  end getTKTObject()

}// end class

package model.beans.requestbeans;

import java.io.*;


/**
 * Request Java Bean sent by the MIDlet to the server side
 * for authentication against the DB.
 *
 * The Request Bean contains the parameters for executing the SQL Queries
 * on the server side.
 *
 * This bean is created after the request from the MIDlet is read
 * and before creating the SQL parameter list to execute the particular
 * SQL Query agains the DB
 *
 * @author Mihai balan - s031288
 *
 */
public class Authentication_1_Req_Bean{
```

```
// ============================================================================

//                            PROPERTIES
// ============================================================================


/**
 * User name
 */
private String userName = "";

/**
 * User's password
 */
private String password = "";

/**
 * Constructor
 */
public Authentication_1_Req_Bean(){}

// ============================================================================

//                            SET METHODS
// ============================================================================

/**
 * Set user's name
 *
 * @param userName
 */
public void setUserName(String userName){
  this.userName = userName;
}

/**
 * Set user's password
 *
 * @param password User's password
 */
public void setPassword(String password){
  this.password = password;
}


// ============================================================================
```

```
//                            GET METHODS
// ============================================================================

/**
 * Get User's name
 * @return User name
 */
public String getUserName(){
  return userName;
}

/**
 * Get User's password
 * @return User's password
 */
public String getPassword(){
  return password;
}


// ============================================================================

//                  READ/WRITE METHODS
// ============================================================================


/**
 * Write the user name and password to the network
 *
 * @param dataStream The stream used for writing the data
 * @throws IOException
 */
public void writeBean(DataOutputStream dataStream) throws IOException {
  dataStream.writeUTF(userName);
  dataStream.writeUTF(password);

} // end writeBean()

/**
 * Read the user name and password
 * from the network and cosntruct the Authentication_1_Req_Bean
 *
 * @param dataStream The stream used for reading the data
 * @return The read Authentication_1_Req_Bean
 * @throws IOException
 */
```

```
  public static Authentication_1_Req_Bean readBean(DataInputStream
      dataStream) throws IOException{
    Authentication_1_Req_Bean authBean = new Authentication_1_Req_Bean();
    authBean.userName  = dataStream.readUTF();
    authBean.password  = dataStream.readUTF();

    return authBean;

  } // end readBean()


  /**
   * Return the string representation of the Authentication_1_Req_Bean
   *
   * @return The string representation of the Authentication_1_Req_Bean
   */
  public String toString(){

    String res = "----␣Authentication_1_Req_Bean␣-----\n";

    res += "-------------------------------------------\n";
    res += "User␣Name:␣␣␣␣" + userName  + "\n";
    res += "Old␣Password:␣" + password + "\n";
    res += "-------------------------------------------\n";

    return res;

  } // end toString()

  /**
   * Compares two Authentication_1_Req_Bean objects
   * @param object A Authentication_1_Req_Bean object
   */
  public boolean equals(Object object) {
    return object != null
    && (object instanceof Authentication_1_Req_Bean
        && (object == this
            || (((Authentication_1_Req_Bean) object).getUserName().equals(
                userName))
            && (((Authentication_1_Req_Bean) object).getPassword().equals(
                password))));
  } // end equals()

} // end class

package model.beans.requestbeans;

import java.io.*;
```

```
/**
 * This is a Cancel_Tickets_Req_Bean that contains information about
 * the tickets user wants to cancel. These tickets have been purchased
     using
 * the "CARD" payment method
 * This Bean is sent by the MIDlet to the server side.
 *
 * The Request Bean contains the parameters for executing the SQL Queries
 * on the server side.
 *
 * This bean is created after the request from the MIDlet is read
 * and before creating the SQL parameter list to execute the particular
 * SQL Query agains the DB
 *
 * @author Mihai Balan - s031288
 *
 */
public class Cancel_Tickets_Req_Bean{

  // ============================================================================

  //                           PROPERTIES
  // ============================================================================

  //private static final long serialVersionUID = 1L;

  /** User name */
  private String userName = "";

  /** User's encrypted password */
  private byte[] password = null;

  /** No of tickets that are to be cancelled */
  private int noOfTickets = 0;

  /** User Reservation ID */
  private String reservationID = "";

  /** User reserved ticket IDs */
  private String ticketID[] = null;

  public Cancel_Tickets_Req_Bean() {}

  // ============================================================================
```

```
//                          SET METHODS
// ============================================================================


public void setUserName(String userName){
  this.userName = userName;
}

public void setPassword(byte[] password){
  this.password = password;
}

public void setNoOfTickets(int noOfTickets){
  this.noOfTickets = noOfTickets;
}

public void setReservationID(String reservationID){
  this.reservationID = reservationID;
}

public void setTicketID(String[] ticketID){
  this.ticketID = ticketID;
}

// ============================================================================

//                          GET METHODS
// ============================================================================


public String getUserName(){
  return userName;
}

public byte[] getPassword(){
  return password;
}
public int getNoOfTickets(){
  return noOfTickets;
}

public String getReservationID(){
  return reservationID;
}

public String[] getTicketID(){
  return ticketID;
```

```
}

// ============================================================================

//                  READ/WRITE METHODS
// ============================================================================

/**
 * Write the canceled ticket bean properties to the network
 *
 * @param dataStream The DataStreamOutput to write the movie details to
 */
public void writeBean(DataOutputStream dataStream) throws IOException {
  dataStream.writeUTF(userName);

  if (password != null){
    dataStream.writeInt(password.length);
    dataStream.write(password);
  }else{
    dataStream.writeInt(0);
  }

  dataStream.writeInt(noOfTickets);

  dataStream.writeUTF(reservationID);

  // write the elements in the ticketIDs[]
  for (int i = 0; i < noOfTickets; i++)
    dataStream.writeUTF(ticketID[i]);

} // end writeBean()


/**
 * Read the canceled tickets bean from the network and
 * creates the Cancel_Tickets_Req_Bean bean to store all details.
 * This bean is to be used later on to extract the parameters for
 * running the Cancel_Tickets stored procedure on the server side
 *
 * @param dataStream The DataStreamInput to read the cinema hall conf
     details
 * @return Cancel_Tickets_Req_Bean that stores all cinema hall conf
     details
 * @throws IOException
 */
public static Cancel_Tickets_Req_Bean readBean(DataInputStream
    dataStream) throws IOException {
```

```
  Cancel_Tickets_Req_Bean canceledTicketReqBean = new
      Cancel_Tickets_Req_Bean();
  System.out.println("---------␣In␣the␣Purchase_Tickets_Req_Bean␣-␣
      before␣reading");

  canceledTicketReqBean.userName    = dataStream.readUTF();

  // read encrypted password
  byte[] password = new byte[dataStream.readInt()];
  dataStream.readFully(password);
  canceledTicketReqBean.password = password;

  canceledTicketReqBean.noOfTickets = dataStream.readInt();

  canceledTicketReqBean.reservationID = dataStream.readUTF();

  // read all ticketID values
  canceledTicketReqBean.ticketID = new String[canceledTicketReqBean.
      noOfTickets];
  for (int i = 0; i < canceledTicketReqBean.noOfTickets; i++)
    canceledTicketReqBean.ticketID[i] = dataStream.readUTF();

  return canceledTicketReqBean;

} // end readBean()


/**
 * Return the string representation of the Cancel_Tickets_Req_Bean
 *
 * @return The string representation of the Cancel_Tickets_Req_Bean
 */
public String toString(){

  String res = "----␣Cancel_Tickets_Req_Bean␣----\n";
  String ticketIDStr = "";

  for (int i = 0; i < noOfTickets; i++){
    ticketIDStr += ticketID[i] + "␣|␣";
  }
  res += "-----------------------------------------\n";
  res += "User␣Name:␣␣␣␣␣␣" + userName  + "\n";
  res += "Password:␣␣␣␣␣␣␣" + password  + "\n";
  res += "No␣Of␣Tickets:␣␣" + noOfTickets + "\n";
  res += "Reservation␣ID:␣" + reservationID + "\n";
  res += "Ticket␣ID's:␣␣␣␣" + ticketIDStr + "\n";
```

```
    res += "-----------------------------------------\n";

    return res;

  } // end toString()


  /**
   * Compare 2 Cancel_Tickets_Req_Bean objects
   *
   * @param object a Cancel_Tickets_Req_Bean objects
   */
  public boolean equals(Object object) {
    return object != null
    && (object instanceof Cancel_Tickets_Req_Bean
        && (object == this
            || (((Cancel_Tickets_Req_Bean) object).getUserName() ==
                userName)
            && (((Cancel_Tickets_Req_Bean) object).getNoOfTickets() ==
                noOfTickets)
            && (((Cancel_Tickets_Req_Bean) object).getReservationID() ==
                reservationID)));

  } // end equals()

}// end class

package model.beans.requestbeans;

import java.io.*;


/**
 * Request Java Bean sent by the MIDlet to the server side
 * to change user's password in the DB.
 *
 * The Request Bean contains the parameters for executing the SQL Queries
 * on the server side.
 *
 * This bean is created after the request from the MIDlet is read
 * and before creating the SQL parameter list to execute the particular
 * SQL Query agains the DB
 *
 * @author Mihai balan - s031288
 *
 */
public class Change_Password_Req_Bean{
```

```
// ============================================================================

//                            PROPERTIES
// ============================================================================


/**
 * User name
 */
private String userName = "";

/**
 * User's encrypted old password
 */
private byte[] oldPassword = null;

/**
 * User's encrypted new password
 */
private byte[] newPassword = null;


/**
 * Constructor
 */
public Change_Password_Req_Bean(){}

// ============================================================================

//                            SET METHODS
// ============================================================================

/**
 * Set user's name
 *
 * @param userName
 */
public void setUserName(String userName){
  this.userName = userName;
}

/**
 * Set user's old encrypted password
 *
 * @param oldPassword User's old password
 */
public void setOldPassword(byte[] oldPassword){
```

```
  this.oldPassword = oldPassword;
}

/**
 * Set user's new encrypted password
 *
 * @param newPassword User's new password
 */
public void setNewPassword(byte[] newPassword){
  this.newPassword = newPassword;
}

// ============================================================================

//                           GET METHODS
// ============================================================================

/**
 * Get User's name
 * @return User name
 */
public String getUserName(){
  return userName;
}

/**
 * Get User's old encrypted password
 * @return User's old password
 */
public byte[] getOldPassword(){
  return oldPassword;
}

/**
 * Get user's new encrypted password
 * @return User's new password
 */
public byte[] getNewPassword(){
  return newPassword;
}


// ============================================================================

//                  READ/WRITE METHODS
// ============================================================================
```

```
/**
 * Write the user name, old password, and new password to the network
 *
 * @param dataStream The stream used for writing the data
 * @throws IOException
 */
public void writeBean(DataOutputStream dataStream) throws IOException {
  dataStream.writeUTF(userName);

  if (oldPassword != null){
    dataStream.writeInt(oldPassword.length);
    dataStream.write(oldPassword);
  } else {
    dataStream.writeInt(0);
  }

  if (newPassword != null){
    dataStream.writeInt(newPassword.length);
    dataStream.write(newPassword);
  } else {
    dataStream.writeInt(0);
  }

} // end writeBean()

/**
 * Read the user name, old password, and new password
 * from the network and cosntruct the Change_Password_Req_Bean
 * sent by the MIDlet
 *
 * @param dataStream The stream used for reading the data
 * @return The read Change_Password_Req_Bean
 * @throws IOException
 */
public static Change_Password_Req_Bean readBean(DataInputStream
    dataStream) throws IOException{
  Change_Password_Req_Bean chgPswdBean = new Change_Password_Req_Bean()
      ;
  chgPswdBean.userName  = dataStream.readUTF();

  // used for reading the old & new password data
  byte[] oldPassword = new byte[dataStream.readInt()];
  dataStream.readFully(oldPassword);
  chgPswdBean.oldPassword = oldPassword;

  byte[] newPassword = new byte[dataStream.readInt()];
```

```
    dataStream.readFully(newPassword);
    chgPswdBean.newPassword = newPassword;

    return chgPswdBean;

  } // end readBean()


  /**
   * Return the string representation of the Change_Password_Req_Bean
   *
   * @return The string representation of the Change_Password_Req_Bean
   */
  public String toString(){

    String res = "----␣Change_Password_Req_Bean␣-----\n";

    res += "-----------------------------------------\n";
    res += "User␣Name:␣␣␣␣" + userName  + "\n";
    res += "Old␣Password:␣" + oldPassword + "\n";
    res += "New␣Password:␣" + newPassword + "\n";
    res += "-----------------------------------------\n";

    return res;

  } // end toString()

  /**
   * Compares two Change_Password_Req_Bean objects
   * @param object A Change_Password_Req_Bean object
   */
  public boolean equals(Object object) {
    return object != null
    && (object instanceof Change_Password_Req_Bean
        && (object == this
            || (((Change_Password_Req_Bean) object).getUserName().equals(
                userName))));
  } // end equals()

} // end class

package model.beans.requestbeans;

import java.io.*;


/**
 * Request Java Bean sent by the MIDlet to the server side
```

```
 * to retrieve the Cinema Haal Configuration for the given Show.
 * A show is uniquely identified by using ShowLocationID and ShowTimeID
 *
 * The Request Bean contains the parameters for executing the SQL Queries
 * on the server side.
 *
 * This bean is created after the request from the MIDlet is read
 * and before creating the SQL parameter list to execute the particular
 * SQL Query agains the DB
 *
 * @author Mihai balan - s031288
 *
 */
public class Cinema_Hall_Conf_Req_Bean{

  // ============================================================================

  //                            PROPERTIES
  // ============================================================================


  /**
   * Movie ShowLocationID
   */
  private int showLocationID = 0;

  /**
   * Movie ShowTimeID
   */
  private int showTimeID = 0;


  /**
   * Constructor
   */
  public Cinema_Hall_Conf_Req_Bean(){}

  // ============================================================================

  //                            SET METHODS
  // ============================================================================

  /**
   * Set movie showLocationID
   *
   * @param showLocationID
   */
```

```
public void setShowLocationID(int showLocationID){
  this.showLocationID = showLocationID;
}

/**
 * Set movie showTimeID
 *
 * @param showTimeID Movie showTimeID
 */
public void setShowTimeID(int showTimeID){
  this.showTimeID = showTimeID;
}


// ============================================================================

//                            GET METHODS
// ============================================================================

/**
 * Get Movie showLocationID
 * @return showLocationID
 */
public int getShowLocationID(){
  return showLocationID;
}

/**
 * Get Movie showTimeID
 * @return showTimeID
 */
public int getShowTimeID(){
  return showTimeID;
}


// ============================================================================

//                   READ/WRITE METHODS
// ============================================================================


/**
 * Write the movie showLocationId and showTimeID to the network
 *
 * @param dataStream The stream used for writing the data
 * @throws IOException
```

```
 */
public void writeBean(DataOutputStream dataStream) throws IOException {
  dataStream.writeInt(showLocationID);
  dataStream.writeInt(showTimeID);

} // end writeBean()

/**
 * Read the movie showLocationId and showTimeID
 * from the network and cosntruct the Cinema_Hall_Conf_Req_Bean
 * sent by the MIDlet
 *
 * @param dataStream The stream used for reading the data
 * @return The read Cinema_Hall_Conf_Req_Bean
 * @throws IOException
 */
public static Cinema_Hall_Conf_Req_Bean readBean(DataInputStream
    dataStream) throws IOException{
  Cinema_Hall_Conf_Req_Bean cinHallConfBean = new
      Cinema_Hall_Conf_Req_Bean();
  cinHallConfBean.showLocationID = dataStream.readInt();
  cinHallConfBean.showTimeID     = dataStream.readInt();
  return cinHallConfBean;

} // end readBean()



/**
 * Return the string representation of the Cinema_Hall_Conf_Req_Bean
 *
 * @return The string representation of the Cinema_Hall_Conf_Req_Bean
 */
public String toString(){

  String res = "----␣Cinema_Hall_Conf_Req_Bean␣----\n";

  res += "-----------------------------------------\n";
  res += "ShowLocationID:␣" + showLocationID + "\n";
  res += "ShowTimeID:␣␣␣␣␣" + showTimeID + "\n";
  res += "-----------------------------------------\n";

  return res;

} // end toString()
```

```
  /**
   * Compares two Cinema_Hall_Conf_Req_Bean objects
   * @param object A Cinema_Hall_Conf_Req_Bean object
   */
  public boolean equals(Object object) {
    return object != null
    && (object instanceof Cinema_Hall_Conf_Req_Bean
        && (object == this
            || (((Cinema_Hall_Conf_Req_Bean) object).getShowLocationID()
                 == showLocationID)
            && (((Cinema_Hall_Conf_Req_Bean) object).getShowTimeID() ==
                showTimeID)));
  } // end equals()

} // end class

package model.beans.requestbeans;

import java.io.*;


/**
 * Request Java Bean sent by the MIDlet to the server side
 * to find movies based on the MOVIE LOCATION SERVICE (MLS).
 * The request from the client goes to MLS where all or the given movie
 * is found in the given range from user given location.
 * MLS returns a list of all cinemas in the given range from the user.
 * The cinema list is further used by the Cinema Controller to
 * retrieve the movie(s) requested by users.
 *
 * The Request Bean contains the parameters for MLS and Cinema Controller
 *
 * This bean is created after the request from the MIDlet is read
 * and before creating the SQL parameter list to execute the
 * SQL Query fro finding the movie(s) agains the DB based on the
 * results returned by MLS
 *
 * @author Mihai balan - s031288
 *
 */
public class Find_Movies_Req_Bean{

  // ============================================================================

  //                           PROPERTIES
  // ============================================================================
```

```
/** Movie name */
private String movie = "";

/** Street name of user's current location */
private String street = "";

/** City name of user's current location */
private String city = "";

/** Zip code of user's current location */
private String zip = "";

/** Range from the current user's location to find the movies */
private String range = "";

/** Date for the movies/shows */
private String date = "";

/**
 * Constructor
 */
public Find_Movies_Req_Bean(){}

// ================================================================================

//                            SET METHODS
// ================================================================================


public void setMovie(String movie){
  this.movie = movie;
}

public void setStreet(String street){
  this.street = street;
}

public void setCity(String city){
  this.city = city;
}

public void setZip(String zip){
  this.zip = zip;
}

public void setRange(String range){
  this.range = range;
```

```
}

public void setDate(String date){
  this.date = date;
}

// ========================================================================

//                           GET METHODS
// ========================================================================

public String getMovie(){
  return movie;
}

public String getStreet(){
  return street;
}

public String getCity(){
  return city;
}

public String getZip(){
  return zip;
}

public String getRange(){
  return range;
}

public String getDate(){
  return date;
}


// ========================================================================

//                  READ/WRITE METHODS
// ========================================================================


/**
 * Write bean properties to the network
 *
 * @param dataStream The stream used for writing the data
 * @throws IOException
```

```
 */
public void writeBean(DataOutputStream dataStream) throws IOException {
  dataStream.writeUTF(movie);
  dataStream.writeUTF(street);
  dataStream.writeUTF(city);
  dataStream.writeUTF(zip);
  dataStream.writeUTF(range);
  dataStream.writeUTF(date);

} // end writeBean()

/**
 * Read the bean properties
 * from the network and cosntruct the Find_Movies_Req_Bean
 * sent by the MIDlet
 *
 * @param dataStream The stream used for reading the data
 * @return The read Find_Movies_Req_Bean
 * @throws IOException
 */
public static Find_Movies_Req_Bean readBean(DataInputStream dataStream)
     throws IOException{
  Find_Movies_Req_Bean findMovBean = new Find_Movies_Req_Bean();
  findMovBean.movie = dataStream.readUTF();
  findMovBean.street = dataStream.readUTF();
  findMovBean.city = dataStream.readUTF();
  findMovBean.zip  = dataStream.readUTF();
  findMovBean.range = dataStream.readUTF();
  findMovBean.date = dataStream.readUTF();
  return findMovBean;

} // end readBean()


/**
 * Return the string representation of the Find_Movies_Req_Bean
 *
 * @return The string representation of the Find_Movies_Req_Bean
 */
public String toString(){

  String res = "----␣Find_Movies_Req_Bean␣----\n";

  res += "-----------------------------------------\n";
  res += "Movie:␣␣␣␣␣" + movie + "\n";
  res += "Street:␣␣␣␣" + street + "\n";
  res += "City:␣␣␣␣␣␣" + city + "\n";
```

```
    res += "Zip:␣␣␣␣␣␣␣" + zip + "\n";
    res += "Range:␣␣␣␣␣" + range + "\n";
    res += "Show␣Date:␣" + date + "\n";
    res += "-----------------------------------------\n";

    return res;

  } // end toString()


  /**
   * Compares two Find_Movies_Req_Bean objects
   * @param object A Find_Movies_Req_Bean object
   */
  public boolean equals(Object object) {
    return object != null
    && (object instanceof Find_Movies_Req_Bean
        && (object == this
            || (((Find_Movies_Req_Bean) object).getMovie().equals(movie))
            && (((Find_Movies_Req_Bean) object).getStreet().equals(street)
                )
            && (((Find_Movies_Req_Bean) object).getCity().equals(city))
            && (((Find_Movies_Req_Bean) object).getZip().equals(zip))
            && (((Find_Movies_Req_Bean) object).getRange().equals(range))
            && (((Find_Movies_Req_Bean) object).getDate().equals(date))));
  } // end equals()

} // end class

package model.beans.requestbeans;

import java.io.*;


/**
 * Request Java Bean sent by the MIDlet to the server side
 * to retrieve the requested movie details from the DB.
 *
 * The Request Bean contains the parameters for executing the SQL Queries
 * on the server side.
 *
 *
 * This bean is created after the request from the MIDlet is read
 * and before creating the SQL parameter list to execute the particular
 * SQL Query agains the DB
 *
 * @author Mihai balan - s031288
 *
```

```
 */
public class Movie_Details_Req_Bean{

  // ============================================================================

  //                            PROPERTIES
  // ============================================================================

  /**
   * Movie Location ID i.e. Cinema, Date and Hour
   */
  private int showLocationID = 0;

  /**
   * Constructor
   *
   */
  public Movie_Details_Req_Bean(){}


  // ============================================================================

  //                            SET METHODS
  // ============================================================================


  /**
   * Set Movie ShowLocationID
   * @param showLocationID Movie ShowLocationID as in the DB
   */
  public void setShoLocationID(int showLocationID){
    this.showLocationID = showLocationID;
  }

  // ============================================================================

  //                            GET METHODS
  // ============================================================================

  /**
   * Get Movie ShowLocationID
   * @param showLocationID Movie ShowLocationID as in the DB
   */
  public int getShowLocationID(){
    return showLocationID;
  }
```

```
// ============================================================================

//                  READ/WRITE METHODS
// ============================================================================

/**
 * Write the movie ShowLocationID to the network
 *
 * @param dataStream The stream used for writing the data
 * @throws IOException
 */
public void writeBean(DataOutputStream dataStream) throws IOException {
  dataStream.writeInt(showLocationID);

} // end writeBean()

/**
 * Read the movie ShowLocationID from the network
 * and construct the Movie_Details_Req_Bean sent by the MIDlet
 *
 * @param dataStream The stream used for reading the data
 * @return The retrieved Movie_Details_Req_Bean
 * @throws IOException
 */
public static Movie_Details_Req_Bean readBean(DataInputStream
    dataStream) throws IOException {
  Movie_Details_Req_Bean movDetReqBean = new Movie_Details_Req_Bean();
  movDetReqBean.showLocationID = dataStream.readInt();
  return movDetReqBean;

} // // end readBean()


/**
 * Return the string representation of the Movie_Details_Req_Bean
 *
 * @return The string representation of the Movie_Details_Req_Bean
 */
public String toString(){

  String res = "----␣Movie_Details_Req_Bean␣----\n";

  res += "-----------------------------------------\n";
  res += "ShowLocationID:␣" + showLocationID + "\n";
  res += "-----------------------------------------\n";
```

```
    return res;

  } // end toString()

  /**
   * Compares two Movie_Details_Req_Bean objects
   * @param object A Movie_Details_Req_Bean object
   */
  public boolean equals(Object object) {
    return object != null
    && (object instanceof Movie_Details_Req_Bean
        && (object == this
            || ((Movie_Details_Req_Bean) object).getShowLocationID() ==
                showLocationID));
  } // end equals()

} // end class

package model.beans.requestbeans;

import java.io.*;


/**
 * This is a Purchase_Tickets_Req_Bean that contains information about
 * the tickets user wants to purchase.
 * This Bean is sent by the MIDlet to the server side.
 *
 * The Request Bean contains the parameters for executing the SQL Queries
 * on the server side.
 *
 * This bean is created after the request from the MIDlet is read
 * and before creating the SQL parameter list to execute the particular
 * SQL Query agains the DB
 *
 * @author Mihai Balan - s031288
 *
 */
public class Purchase_Tickets_Req_Bean{

  // ============================================================================

  //                            PROPERTIES
  // ============================================================================

  //private static final long serialVersionUID = 1L;

  /** User name */
```

```
private String userName = "";

/** User's encrypted password */
private byte[] password = null;

/** ShowLocationID used to identify a show in the DB */
private int showLocationID = 0;

/** ShowTimeID used to identify a show in the DB */
private int showTimeID = 0;

/** No of rows in the array of seats[][] */
private int seatsNoRows = 0;

/** No of cols in the array of seats[][] */
private int seatsNoCols = 0;

/** All Reserved Seats by the user that are to be canceled */
private int seats[][] = null;

/** The discount types for the reserved seats */
private String discounts[] = null;

/** Credit Card Type e.g. VISA, Eurocard */
private byte[] creditCardType = null;

/** Credit Card no */
private byte[] creditCardNo = null;

/** Credit Card ExpDate i.e. month - year */
private byte[] creditCardExpDate = null;

/** Credit Card CW2 */
private byte[] creditCardCW2 = null;

/** Reservation Date */
private String reservationDate = "";

/** Reservation Date */
private String purchaseMethod = "";


public Purchase_Tickets_Req_Bean() {}

// ============================================================================

//                           SET METHODS
```

```
// ============================================================================


public void setUserName(String userName){
  this.userName = userName;
}

public void setPassword(byte[] password){
  this.password = password;
}

public void setShowLocationID(int showLocationID){
  this.showLocationID = showLocationID;
}

public void setShowTimeID(int showTimeID){
  this.showTimeID = showTimeID;
}

public void setSeatsNoRows(int seatsNoRows){
  this.seatsNoRows = seatsNoRows;
}

public void setSeatsNoCols(int seatsNoCols){
  this.seatsNoCols = seatsNoCols;
}

public void setSeats(int[][] seats){
  this.seats = seats;
}

public void setDiscounts(String[] discounts){
  this.discounts = discounts;
}

public void setCreditCardType(byte[] creditCardType){
  this.creditCardType = creditCardType;
}

public void setCreditCardNo(byte[] creditCardNo){
  this.creditCardNo = creditCardNo;
}

public void setCreditCardExpDate(byte[] creditCardExpDate){
  this.creditCardExpDate = creditCardExpDate;
}
```

```
public void setCreditCardCW2(byte[] creditCardCW2){
  this.creditCardCW2 = creditCardCW2;
}

public void setReservationDate(String reservationDate){
  this.reservationDate = reservationDate;
}

public void setPurchaseMethod(String purchaseMethod){
  this.purchaseMethod = purchaseMethod;
}

// ============================================================================

//                           GET METHODS
// ============================================================================


public String getUserName(){
  return userName;
}

public byte[] getPassword(){
  return password;
}
public int getShowLocationID(){
  return showLocationID;
}

public int getShowTimeID(){
  return showTimeID;
}

public int getSeatsNoRows(){
  return seatsNoRows;
}

public int getSeatsNoCols(){
  return seatsNoCols;
}

public int[][] getSeats(){
  return seats;
}

public String[] getDiscounts(){
  return discounts;
```

```
}

public byte[] getCreditCardType(){
  return creditCardType;
}

public byte[] getCreditCardNo(){
  return creditCardNo;
}

public byte[] getCreditCardExpDate(){
  return creditCardExpDate;
}

public byte[] getCreditCardCW2(){
  return creditCardCW2;
}

public String getReservationDate(){
  return reservationDate;
}

public String getPurchaseMethod(){
  return purchaseMethod;
}

// ================================================================================

//                  READ/WRITE METHODS
// ================================================================================

/**
 * Write the purchased tickets bean properties to the network
 *
 * @param dataStream The DataStreamOutput to write the movie details to
 */
public void writeBean(DataOutputStream dataStream) throws IOException {
  dataStream.writeUTF(userName);

  // write encrypted password
  if(password != null){
    dataStream.writeInt(password.length);
    dataStream.write(password);
  } else{
    dataStream.writeInt(0);
  }
```

```
dataStream.writeInt(showLocationID);
dataStream.writeInt(showTimeID);
dataStream.writeInt(seatsNoRows);
dataStream.writeInt(seatsNoCols);

// write all elements in the seats array
for (int i = 0; i < seatsNoRows; i++)
  for (int j = 0; j < seatsNoCols; j++){

    dataStream.writeInt(seats[i][j]);
}

// write the elements in the discounts[]
for (int i = 0; i < seatsNoRows; i++)
    dataStream.writeUTF(discounts[i]);

// write encrypted creditCardType
if(creditCardType != null){
  dataStream.writeInt(creditCardType.length);
  dataStream.write(creditCardType);
} else{
  dataStream.writeInt(0);
}

//write encrypted creditCardNo
if(creditCardNo != null){
  dataStream.writeInt(creditCardNo.length);
  dataStream.write(creditCardNo);
} else{
  dataStream.writeInt(0);
}

// write encrypted creditCardExpDate
if(creditCardExpDate != null){
  dataStream.writeInt(creditCardExpDate.length);
  dataStream.write(creditCardExpDate);
} else{
  dataStream.writeInt(0);
}

// write encrypted creditCardCW2
if(creditCardCW2 != null){
  dataStream.writeInt(creditCardCW2.length);
  dataStream.write(creditCardCW2);
} else{
  dataStream.writeInt(0);
}
```

```
  dataStream.writeUTF(reservationDate);
  dataStream.writeUTF(purchaseMethod);

} // end writeBean()


/**
 * Read the purchesed tickets bean from the network and
 * creates the Purchase_Tickets_Req_Bean bean to store all details.
 * This bean is to be used later on to extract the parameters for
 * running the Compute_Price_And_Maybe_Pay stored procedure on the
     server side
 *
 * @param dataStream The DataStreamInput to read the cinema hall conf
     details
 * @return Purchase_Tickets_Req_Bean that stores all cinema hall conf
     details
 * @throws IOException
 */
public static Purchase_Tickets_Req_Bean readBean(DataInputStream
    dataStream) throws IOException {

  Purchase_Tickets_Req_Bean purchaseTicketReqBean = new
      Purchase_Tickets_Req_Bean();
  System.out.println("---------␣In␣the␣Purchase_Tickets_Req_Bean␣-␣
      before␣reading");

  purchaseTicketReqBean.userName    = dataStream.readUTF();

  // read encrypted password
  byte[] password = new byte[dataStream.readInt()];
  dataStream.readFully(password);
  purchaseTicketReqBean.password    = password;

  purchaseTicketReqBean.showLocationID = dataStream.readInt();
  purchaseTicketReqBean.showTimeID = dataStream.readInt();
  purchaseTicketReqBean.seatsNoRows = dataStream.readInt();
  purchaseTicketReqBean.seatsNoCols = dataStream.readInt();

  // read all seats values
  purchaseTicketReqBean.seats = new int[purchaseTicketReqBean.
      seatsNoRows][purchaseTicketReqBean.seatsNoCols];
  for (int i = 0; i < purchaseTicketReqBean.seatsNoRows; i++)
    for (int j = 0; j < purchaseTicketReqBean.seatsNoCols; j++){
      purchaseTicketReqBean.seats[i][j] = dataStream.readInt();
  }
```

```
  // read all discount values
  purchaseTicketReqBean.discounts = new String[purchaseTicketReqBean.
      seatsNoRows];
  for (int i = 0; i < purchaseTicketReqBean.seatsNoRows; i++)
     purchaseTicketReqBean.discounts[i] = dataStream.readUTF();

  // read encrypted creditCardType
  byte[] creditCardType = new byte[dataStream.readInt()];
  dataStream.readFully(creditCardType);
  purchaseTicketReqBean.creditCardType = creditCardType;

  // read encrypted creditCardNo
  byte[] creditCardNo = new byte[dataStream.readInt()];
  dataStream.readFully(creditCardNo);
  purchaseTicketReqBean.creditCardNo = creditCardNo;

  // read encrypted creditCardExpDate
  byte[] creditCardExpDate = new byte[dataStream.readInt()];
  dataStream.readFully(creditCardExpDate);
  purchaseTicketReqBean.creditCardExpDate = creditCardExpDate;

  // read encrypted creditCardCW2
  byte[] creditCardCW2 = new byte[dataStream.readInt()];
  dataStream.readFully(creditCardCW2);
  purchaseTicketReqBean.creditCardCW2 = creditCardCW2;

  purchaseTicketReqBean.reservationDate = dataStream.readUTF();
  purchaseTicketReqBean.purchaseMethod = dataStream.readUTF();

  return purchaseTicketReqBean;

} // end readBean()


/**
 * Return the string representation of the Purchase_Tickets_Req_Bean
 *
 * @return The string representation of the Purchase_Tickets_Req_Bean
 */
public String toString(){

  String res = "----␣Purchase_Tickets_Req_Bean␣----\n";
  String seatsStr = "";
  String discountStr = "";

  for (int i = 0; i < seatsNoRows; i++){
```

```
    for (int j = 0; j < seatsNoCols; j++){
      seatsStr += seats[i][j] + "␣␣";
    }
    seatsStr += "|␣";
  }

  for (int i = 0; i < seatsNoRows; i++){
    discountStr += discounts[i] + "␣|␣";
  }

  res += "-----------------------------------------------\n";
  res += "User␣Name:␣␣␣␣␣␣␣␣␣␣␣␣␣" + userName     + "\n";
  res += "Password:␣␣␣␣␣␣␣␣␣␣␣␣␣" + password     + "\n";
  res += "ShowLocationID:␣␣␣␣␣␣␣" + showLocationID + "\n";
  res += "ShowTimeID:␣␣␣␣␣␣␣␣␣␣␣" + showTimeID   + "\n";
  res += "No␣Of␣Rows:␣␣␣␣␣␣␣␣␣␣" + seatsNoRows  + "\n";
  res += "No␣Of␣Cols:␣␣␣␣␣␣␣␣␣␣" + seatsNoCols  + "\n";
  res += "Seats:␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣" + seatsStr     + "\n";
  res += "Discounts:␣␣␣␣␣␣␣␣␣␣␣␣" + discountStr  + "\n";
  res += "Credit␣Card␣Type:␣␣␣␣␣" + creditCardType + "\n";
  res += "Credit␣Card␣No:␣␣␣␣␣␣␣" + creditCardNo + "\n";
  res += "Credit␣Card␣Exp␣Date:␣" + creditCardExpDate + "\n";
  res += "Credit␣Card␣CW2:␣␣␣␣␣␣" + creditCardCW2 + "\n";
  res += "Reservation␣Date:␣␣␣␣␣" + reservationDate + "\n";
  res += "Purchased␣Method:␣␣␣␣␣" + purchaseMethod + "\n";
  res += "-----------------------------------------------\n";

  return res;

} // end toString()

/**
 * Compare 2 Purchase_Tickets_Req_Bean objects
 *
 * @param object a Purchase_Tickets_Req_Bean objects
 */
public boolean equals(Object object) {
  return object != null
  && (object instanceof Purchase_Tickets_Req_Bean
      && (object == this
          || (((Purchase_Tickets_Req_Bean) object).getUserName() ==
              userName)
          && (((Purchase_Tickets_Req_Bean) object).getShowLocationID()
               == showLocationID)
          && (((Purchase_Tickets_Req_Bean) object).getShowTimeID() ==
              showTimeID)
          && (((Purchase_Tickets_Req_Bean) object).getSeatsNoRows() ==
```

```
            seatsNoRows)
          && (((Purchase_Tickets_Req_Bean) object).getSeatsNoCols() ==
            seatsNoCols)));

  } // end equals()

}// end class

package model.beans.requestbeans;

import java.io.*;


/**
 * Request Java Bean sent by the MIDlet to the server side
 * to rate a movie in the DB.
 *
 * The Request Bean contains the parameters for executing the SQL Queries
 * on the server side.
 *
 * This bean is created after the request from the MIDlet is read
 * and before creating the SQL parameter list to execute the particular
 * SQL Query agains the DB
 *
 * @author Mihai balan - s031288
 *
 */
public class Rate_Movie_Req_Bean{

  // ============================================================================

  //                            PROPERTIES
  // ============================================================================


  /**
   * User name
   */
  private String userName = "";

  /**
   * User's encrypted password
   */
  private byte[] password = null;

  /**
   * Indirect Movie ID in the DB i.e. ShowLocationID
   */
```

```
private int showLocationID = 0;

/**
 * User Rating Score
 */
private int movieScore = 0;

/**
 * Constructor
 */
public Rate_Movie_Req_Bean(){}

// ============================================================================

//                            SET METHODS
// ============================================================================

/**
 * Set user's name
 *
 * @param userName
 */
public void setUserName(String userName){
  this.userName = userName;
}

/**
 * Set user's password
 *
 * @param password User's password
 */
public void setPassword(byte[] password){
  this.password = password;
}

/**
 * Set movie ID i.e. ShowLocationID in the DB
 *
 * @param showLocationID movie ID i.e. ShowLocationID in the DB
 */
public void setShowLocationID(int showLocationID){
  this.showLocationID = showLocationID;
}

/**
 * Set movie score
 *
```

```
 * @param movieScore Movie score
 */
public void setMovieScore(int movieScore){
  this.movieScore = movieScore;
}

// ============================================================================

//                            GET METHODS
// ============================================================================

/**
 * Get User's name
 * @return User name
 */
public String getUserName(){
  return userName;
}

/**
 * Get User's password
 * @return User's password
 */
public byte[] getPassword(){
  return password;
}

/**
 * Get movie ID i.e. ShowLocationID in the DB
 *
 * @return movie ID i.e. ShowLocationID in the DB
 */
public int getShowLocationID(){
  return showLocationID;
}

/**
 * Get movie score
 *
 * @return Movie score
 */
public int getMovieScore(){
  return movieScore;
}


// ============================================================================
```

```
//                  READ/WRITE METHODS
// ============================================================================


/**
 * Write bean properties to the network
 *
 * @param dataStream The stream used for writing the data
 * @throws IOException
 */
public void writeBean(DataOutputStream dataStream) throws IOException {
  dataStream.writeUTF(userName);

  // write encrypted password
  if (password != null){
    dataStream.writeInt(password.length);
    dataStream.write(password);
  }else{
    dataStream.writeInt(0);
  }

  dataStream.writeInt(showLocationID);
  dataStream.writeInt(movieScore);

} // end writeBean()

/**
 * Read the bean properties
 * from the network and cosntruct the Rate_Movie_Req_Bean
 * sent by the MIDlet
 *
 * @param dataStream The stream used for reading the data
 * @return The read Rate_Movie_Req_Bean
 * @throws IOException
 */
public static Rate_Movie_Req_Bean readBean(DataInputStream dataStream)
    throws IOException{
  Rate_Movie_Req_Bean rateMovBean = new Rate_Movie_Req_Bean();
  rateMovBean.userName    = dataStream.readUTF();

  // read encrypted password
  byte[] password = new byte[dataStream.readInt()];
  dataStream.readFully(password);
  rateMovBean.password    = password;

  rateMovBean.showLocationID = dataStream.readInt();
```

```
    rateMovBean.movieScore  = dataStream.readInt();
    return rateMovBean;

  } // end readBean()



  /**
   * Return the string representation of the Rate_Movie_Req_Bean
   *
   * @return The string representation of the Rate_Movie_Req_Bean
   */
  public String toString(){

    String res = "␣-----␣Rate_Movie_Req_Bean␣␣-----␣\n";
    res += "-----------------------------------------\n";
    res += "User␣Name:␣␣␣␣␣␣" + userName   + "\n";
    res += "Password:␣␣␣␣␣␣␣" + password   + "\n";
    res += "ShowLocationID:␣" + showLocationID + "\n";
    res += "Movie␣Score:␣␣␣␣" + movieScore + "\n";
    res += "-----------------------------------------\n";

    return res;

  } // end toString()


  /**
   * Compares two Rate_Movie_Req_Bean objects
   * @param object A Rate_Movie_Req_Bean object
   */
  public boolean equals(Object object) {
    return object != null
    && (object instanceof Rate_Movie_Req_Bean
        && (object == this
            || (((Rate_Movie_Req_Bean) object).getUserName().equals(
                userName))
            && (((Rate_Movie_Req_Bean) object).getShowLocationID() ==
                showLocationID)
            && (((Rate_Movie_Req_Bean) object).getMovieScore() ==
                movieScore)));
  } // end equals()

} // end class

package model.beans.requestbeans;

import java.io.*;
```

```
/**
 * This is a Reject_Reservation_Req_Bean that contains information about
 * reservatin user is rejecting by not accepting the payment conditions.
 * This Bean is sent by the MIDlet to the server side.
 *
 * The Request Bean contains the parameters for executing the SQL Queries
 * on the server side.
 *
 * This bean is created after the request from the MIDlet is read
 * and before creating the SQL parameter list to execute the particular
 * SQL Query agains the DB
 *
 * @author Mihai Balan - s031288
 *
 */
public class Reject_Payment_Req_Bean{

  // ============================================================================

  //                           PROPERTIES
  // ============================================================================

  //private static final long serialVersionUID = 1L;

  // ShowLocationID used to identify a show in DB
  private int   showLocationID = 0;

  // ShowTimeID used to identify a show in DB
  private int   showTimeID    = 0;

  // No of rows in the array of seats[][]
  private int   seatsNoRows   = 0;

  // No of cols in the array of seats[][]
  private int   seatsNoCols   = 0;

  // All Reserved Seats by the user that are to be canceled
  private int   seats[][]     = null;

  public Reject_Payment_Req_Bean() {}

  // ============================================================================

  //                           SET METHODS
  // ============================================================================
```

```
public void setShowLocationID(int showLocationID){
  this.showLocationID = showLocationID;
}

public void setShowTimeID(int showTimeID){
  this.showTimeID = showTimeID;
}

public void setSeatsNoRows(int seatsNoRows){
  this.seatsNoRows = seatsNoRows;
}

public void setSeatsNoCols(int seatsNoCols){
  this.seatsNoCols = seatsNoCols;
}

public void setSeats(int[][] seats){
  this.seats = seats;
}

// ============================================================================

//                           GET METHODS
// ============================================================================


public int getShowLocationID(){
  return showLocationID;
}

public int getShowTimeID(){
  return showTimeID;
}

public int getSeatsNoRows(){
  return seatsNoRows;
}

public int getSeatsNoCols(){
  return seatsNoCols;
}

public int[][] getSeats(){
  return seats;
}
```

```
// ============================================================================

//                  READ/WRITE METHODS
// ============================================================================

/**
 * Write the reject reservation bean properties to the network
 *
 * @param dataStream The DataStreamOutput to write the movie details to
 */
public void writeBean(DataOutputStream dataStream) throws IOException {

  dataStream.writeInt(showLocationID);
  dataStream.writeInt(showTimeID);
  dataStream.writeInt(seatsNoRows);
  dataStream.writeInt(seatsNoCols);

  // write all elements in the seats array
  for (int i = 0; i < seatsNoRows; i++)
    for (int j = 0; j < seatsNoCols; j++){

      dataStream.writeInt(seats[i][j]);
  }

} // end writeBean()


/**
 * Read the reject reservatio bean from the network and
 * creates the Reject_Reservation_Req_Bean bean to store all details.
 * This bean is to be used later on to extract the parameters for
 * running the Reject_Payment_Cancel_Selected_Seats stored procedure on
      the server side
 *
 * @param dataStream The DataStreamInput to read the cinema hall conf
     details
 * @return Reject_Reservation_Req_Bean that stores all cinema hall conf
      details
 * @throws IOException
 */
public static Reject_Payment_Req_Bean readBean(DataInputStream
    dataStream) throws IOException {

  Reject_Payment_Req_Bean rejReservationBean = new
      Reject_Payment_Req_Bean();
```

```
  System.out.println("---------␣In␣the␣CINEMA␣HALL␣CONFIGURATION␣
      RESPONSE␣BEAN␣-␣before␣reading");

  rejReservationBean.showLocationID = dataStream.readInt();
  rejReservationBean.showTimeID  = dataStream.readInt();

  rejReservationBean.seatsNoRows = dataStream.readInt();
  rejReservationBean.seatsNoCols = dataStream.readInt();

  // read all bookes seats values
  rejReservationBean.seats = new int[rejReservationBean.seatsNoRows][
      rejReservationBean.seatsNoCols];
  for (int i = 0; i < rejReservationBean.seatsNoRows; i++)
    for (int j = 0; j < rejReservationBean.seatsNoCols; j++){

      rejReservationBean.seats[i][j] = dataStream.readInt();
  }

  return rejReservationBean;

} // end readBean()



/**
 * Return the string representation of the Reject_Payment_Req_Bean
 *
 * @return The string representation of the Reject_Payment_Req_Bean
 */
public String toString(){

  String res = "----␣Reject_Payment_Req_Bean␣␣----\n";
  String seatsStr = "";

  for (int i = 0; i < seatsNoRows; i++){
    for (int j = 0; j < seatsNoCols; j++){
      seatsStr += seats[i][j] + "␣␣";
    }
    seatsStr += "|␣";
  }

  res += "-----------------------------------------\n";
  res += "ShowLocationID:␣" + showLocationID + "\n";
  res += "ShowTimeID:␣␣␣␣␣" + showTimeID + "\n";
  res += "No␣Of␣Rows:␣␣␣␣␣" + seatsNoRows + "\n";
  res += "No␣Of␣Cols:␣␣␣␣␣" + seatsNoCols + "\n";
  res += "Seats:␣␣␣␣␣␣␣␣␣" + seatsStr   + "\n";
```

```
    res += "-----------------------------------------\n";

    return res;

  } // end toString()


  /**
   * Compare 2 Reject_Reservation_Req_Bean objects
   *
   * @param object a Reject_Reservation_Req_Bean objects
   */
  public boolean equals(Object object) {
    return object != null
    && (object instanceof Reject_Payment_Req_Bean
        && (object == this
            || (((Reject_Payment_Req_Bean) object).getShowLocationID() ==
                showLocationID)
            && (((Reject_Payment_Req_Bean) object).getShowTimeID() ==
                showTimeID)
            && (((Reject_Payment_Req_Bean) object).getSeatsNoRows() ==
                seatsNoRows)
            && (((Reject_Payment_Req_Bean) object).seatsNoCols ==
                seatsNoCols)));

  } // end equals()

}// end class

package model.beans.requestbeans;

import java.io.*;


/**
 * This is a Select_Deselect_Seats_Req_Bean that contains information
     about
 * the selected seats by the user.
 * This Bean is sent by the MIDlet to the server side.
 *
 * The Request Bean contains the parameters for executing the SQL Queries
 * on the server side.
 *
 * This bean is created after the request from the MIDlet is read
 * and before creating the SQL parameter list to execute the particular
 * SQL Query agains the DB
 *
 * @author Mihai Balan - s031288
```

```
 *
 */
public class Select_Deselect_Seats_Req_Bean{

  // ============================================================================

  //                            PROPERTIES
  // ============================================================================

  //private static final long serialVersionUID = 1L;

  /** Command to the DB i.e. 1 = SELECT, 2 = DESELECT */
  private int command = 1;

  /** ShowLocationID used to identify a show in the DB */
  private int showLocationID = 0;

  /** ShowTimeID used to identify a show in the DB */
  private int showTimeID = 0;

  /** No of rows in the array of seats[][] */
  private int seatsNoRows = 0;

  /** No of cols in the array of seats[][] */
  private int seatsNoCols = 0;

  /** All selected Seats by the user */
  private int seats[][] = null;

  public Select_Deselect_Seats_Req_Bean() {}

  // ============================================================================

  //                            SET METHODS
  // ============================================================================


  public void setCommmand(int command){
    this.command = command;
  }

  public void setShowLocationID(int showLocationID){
    this.showLocationID = showLocationID;
  }

  public void setShowTimeID(int showTimeID){
    this.showTimeID = showTimeID;
```

```
}

public void setSeatsNoRows(int seatsNoRows){
  this.seatsNoRows = seatsNoRows;
}

public void setSeatsNoCols(int seatsNoCols){
  this.seatsNoCols = seatsNoCols;
}

public void setSeats(int[][] seats){
  this.seats = seats;
}

// ============================================================================

//                            GET METHODS
// ============================================================================


public int getCommand(){
  return command;
}

public int getShowLocationID(){
  return showLocationID;
}

public int getShowTimeID(){
  return showTimeID;
}

public int getSeatsNoRows(){
  return seatsNoRows;
}

public int getSeatsNoCols(){
  return seatsNoCols;
}

public int[][] getSeats(){
  return seats;
}

// ============================================================================

//                   READ/WRITE METHODS
```

```
// ============================================================================

/**
 * Write the select_deselect_seats bean properties to the network
 *
 * @param dataStream The DataStreamOutput to write the movie details to
 */
public void writeBean(DataOutputStream dataStream) throws IOException {
  dataStream.writeInt(command);
  dataStream.writeInt(showLocationID);
  dataStream.writeInt(showTimeID);
  dataStream.writeInt(seatsNoRows);
  dataStream.writeInt(seatsNoCols);

  // write all elements in the seats array
  for (int i = 0; i < seatsNoRows; i++)
    for (int j = 0; j < seatsNoCols; j++){

      dataStream.writeInt(seats[i][j]);
    }

} // end writeBean()


/**
 * Read the select_deselect_seats bean from the network and
 * creates the Select_Deselect_Seats_Req_Bean bean to store all details
     .
 * This bean is to be used later on to extract the parameters for
 * running the SELECT_DESELECT_MANY_SEATS stored procedure on the
     server side
 *
 * @param dataStream The DataStreamInput to read the selected /
     deselected seats
 * @return Select_Deselect_Seats_Req_Bean that stores all selected /
     deselected seats
 * @throws IOException
 */
public static Select_Deselect_Seats_Req_Bean readBean(DataInputStream
    dataStream) throws IOException {

  Select_Deselect_Seats_Req_Bean selDeselSeatsReqBean = new
      Select_Deselect_Seats_Req_Bean();
  System.out.println("---------␣In␣the␣Select_Deselect_Seats_Req_Bean
      ␣-␣before␣reading");

  selDeselSeatsReqBean.command    = dataStream.readInt();
```

```
  selDeselSeatsReqBean.showLocationID = dataStream.readInt();
  selDeselSeatsReqBean.showTimeID  = dataStream.readInt();
  selDeselSeatsReqBean.seatsNoRows = dataStream.readInt();
  selDeselSeatsReqBean.seatsNoCols = dataStream.readInt();

  // read all seats values
  selDeselSeatsReqBean.seats = new int[selDeselSeatsReqBean.seatsNoRows
      ][selDeselSeatsReqBean.seatsNoCols];
  for (int i = 0; i < selDeselSeatsReqBean.seatsNoRows; i++)
    for (int j = 0; j < selDeselSeatsReqBean.seatsNoCols; j++){
      selDeselSeatsReqBean.seats[i][j] = dataStream.readInt();
    }


  return selDeselSeatsReqBean;

} // end readBean()


/**
 * Return the string representation of the
     Select_Deselect_Seats_Req_Bean
 *
 * @return The string representation of the
     Select_Deselect_Seats_Req_Bean
 */
public String toString(){

  String res = "----␣Select_Deselect_Seats_Req_Bean␣␣----\n";
  String seatsStr = "";

  for (int i = 0; i < seatsNoRows; i++){
    for (int j = 0; j < seatsNoCols; j++){
      seatsStr += seats[i][j] + "␣␣";
    }
    seatsStr += "|␣";
  }

  res += "-----------------------------------------\n";
  res += "Command␣Code:␣␣␣" + command    + "\n";
  res += "ShowLocationID:␣" + showLocationID + "\n";
  res += "ShowTimeID:␣␣␣␣␣" + showTimeID + "\n";
  res += "No␣Of␣Rows:␣␣␣␣␣" + seatsNoRows + "\n";
  res += "No␣Of␣Cols:␣␣␣␣␣" + seatsNoCols + "\n";
  res += "Seats:␣␣␣␣␣␣␣␣␣" + seatsStr   + "\n";
  res += "-----------------------------------------\n";
```

```
    return res;

  } // end toString()


  /**
   * Compare 2 Select_Deselect_Seats_Req_Bean objects
   *
   * @param object a Select_Deselect_Seats_Req_Bean objects
   */
  public boolean equals(Object object) {

    return object != null
    && (object instanceof Select_Deselect_Seats_Req_Bean
        && (object == this
            || (((Select_Deselect_Seats_Req_Bean) object).getCommand() ==
                command)
            && (((Select_Deselect_Seats_Req_Bean) object).
                getShowLocationID() == showLocationID)
            && (((Select_Deselect_Seats_Req_Bean) object).getShowTimeID()
                 == showTimeID)
            && (((Select_Deselect_Seats_Req_Bean) object).getSeatsNoRows()
                 == seatsNoRows)
            && (((Select_Deselect_Seats_Req_Bean) object).getSeatsNoCols()
                 == seatsNoCols)));

  } // end equals()

}// end class

package model.beans.responsebeans;

import java.io.*;


/**
 * Response Java Bean sent from the servlet to the MIDlet containing the
 * result of user authentication against the DB
 *
 * @author Mihai balan - s031288
 *
 */
public class Authentication_1_Resp_Bean extends Response_Msg_Bean{

  // ============================================================================

  //                           PROPERTIES
  // ============================================================================
```

```
/**
 * User ID name
 */
private String userID = "";

/**
 * Random ID
 */
private String randomID = "";

/**
 * e-money
 */
private String eMoney = "";

/**
 * Constructor
 */
public Authentication_1_Resp_Bean(){}

/**
 * Constructor 2
 *
 * @param responseCode The response code from the SQL stored procedure
 */
public Authentication_1_Resp_Bean(int responseCode) {
  super(responseCode);
}

// ============================================================================

//                            SET METHODS
// ============================================================================

/**
 * Set userID
 *
 * @param userID
 */
public void setUserID(String userID){
  this.userID = userID;
}

/**
 * Set randomID
```

```
 *
 * @param randomID
 */
public void setRandomID(String randomID){
  this.randomID = randomID;
}

/**
 * Set eMoney
 *
 * @param eMoney
 */
public void setEMoney(String eMoney){
  this.eMoney = eMoney;
}


// ============================================================================

//                            GET METHODS
// ============================================================================

/**
 * Get userID
 * @return userID
 */
public String getUserID(){
  return userID;
}

/**
 * Get randomID
 * @return randomID
 */
public String getRandomID(){
  return randomID;
}

/**
 * Get eMoney
 *
 * @return eMoney
 */
public String getEMoney(){
  return eMoney;
}
```

```
// ============================================================================

//                  READ/WRITE METHODS
// ============================================================================


/**
 * Write the user name and password to the network
 *
 * @param dataStream The stream used for writing the data
 * @throws IOException
 */
public void writeBean(DataOutputStream dataStream) throws IOException {
  dataStream.writeInt(super.responseCode);
  dataStream.writeUTF(userID);
  dataStream.writeUTF(randomID);
  dataStream.writeUTF(eMoney);

} // end writeBean()

/**
 * Read the user name and password
 * from the network and cosntruct the Authentication_1_Req_Bean
 *
 * @param dataStream The stream used for reading the data
 * @return The read Authentication_1_Req_Bean
 * @throws IOException
 */
public static Response_Msg_Bean readBean(DataInputStream dataStream)
    throws IOException{

  Authentication_1_Resp_Bean authBean = new Authentication_1_Resp_Bean
      ();
  authBean.responseCode = dataStream.readInt();
  authBean.userID     = dataStream.readUTF();
  authBean.randomID   = dataStream.readUTF();
  authBean.eMoney     = dataStream.readUTF();

  return authBean;

} // end readBean()


/**
 * Return the string representation of the Authentication_1_Req_Bean
 *
```

```
   * @return The string representation of the Authentication_1_Req_Bean
   */
  public String toString(){

    String res = "----␣Authentication_1_Resp_Bean␣-----\n";

    res += "-----------------------------------------\n";
    res += "SQL␣Code:␣␣␣␣␣" + responseCode + "\n";
    res += "User␣Name:␣␣␣␣" + userID   + "\n";
    res += "Old␣Password:␣" + randomID + "\n";
    res += "EMoney:␣␣␣␣␣␣␣" + eMoney   + "\n";
    res += "-----------------------------------------\n";

    return res;

  } // end toString()

  /**
   * Compares two Authentication_1_Resp_Bean objects
   * @param object An Authentication_1_Resp_Bean object
   */
  public boolean equals(Object object) {
    return object != null
    && (object instanceof Authentication_1_Resp_Bean
        && (object == this
            || (((Authentication_1_Resp_Bean) object).getUserID().equals(
                userID))
            && (((Authentication_1_Resp_Bean) object).getRandomID().equals
                (randomID))
            && (((Authentication_1_Resp_Bean) object).getEMoney().equals(
                eMoney))));
  } // end equals()

} // end class

package model.beans.responsebeans;

import java.io.*;



/**
 * This is a Background Cinema Hall Conf Java Bean that contains
     information about
 * seat status for the cinema hall that display the given show.
 * The server side creates this bean after the
     Background_Hall_Update_Servlet
 * has retrieved the details about the show displayed in the given
```

```
     ShowLocalID
 * and at the given showTimeID.
 * This happens only if there was no error.
 *
 * Then, the bean is sent via HTTP to the MIDlet, and the information
     loaded in
 * the bean is extracted on the MIDlet and used to update the UI
     displaying the
 * Cinema Hall Configuration
 *
 * It inherits from the Response_Msg_Bean in order to set, serialize,
     deserialize
 * the response code value from the sql stored procedure
 *
 * @author Mihai Balan - s031288
 *
 */
public class Background_Cinema_Hall_Conf_Resp_Bean extends
    Response_Msg_Bean{

  // ============================================================================

  //                           PROPERTIES
  // ============================================================================

  //private static final long serialVersionUID = 1L;

  // No of rows in the array of AllBookedSeats[]
  private int  noRowsBookedSeats = 0;

  // No of cols in the array of AllBookedSeats[]
  private int  noColsBookedSeats = 0;

  // All Booked Seats in that Cinema Hall
  private int  allBookedSeats[][] = null;

  /**
   * Constructor 1
   *
   */
  public Background_Cinema_Hall_Conf_Resp_Bean() {}

  /**
   * Constructor 2
   *
   * @param responseCode The response code from the SQL stored procedure
   */
```

```
public Background_Cinema_Hall_Conf_Resp_Bean(int responseCode) {
  super(responseCode);
}

// ============================================================================

//                            SET METHODS
// ============================================================================


public void setNoRowsBookedSeats(int noRowBookedSeats){
  this.noRowsBookedSeats = noRowBookedSeats;
}

public void setNoColsBookedSeats(int noColsBookedSeats){
  this.noColsBookedSeats = noColsBookedSeats;
}

public void setAllBookedSeats(int[][] allBookedSeats){
  this.allBookedSeats = allBookedSeats;
}

// ============================================================================

//                            GET METHODS
// ============================================================================


public int getAllBookedSeatsRows(){
  return noRowsBookedSeats;
}

public int getAllBookedSeatsCols(){
  return noColsBookedSeats;
}

public int[][] getAllBookedSeats(){
  return allBookedSeats;
}


// ============================================================================

//                  READ/WRITE METHODS
// ============================================================================

/**
```

```
 * Write the cinema hall configuration to the network
 *
 * @param dataStream The DataStreamOutput to write the movie details to
 */
public void writeBean(DataOutputStream dataStream) throws IOException {

  dataStream.writeInt(super.responseCode);
  dataStream.writeInt(noRowsBookedSeats);
  dataStream.writeInt(noColsBookedSeats);

  // write all elements in the All Booked Seats Array
  for (int i = 0; i < noRowsBookedSeats; i++)
    for (int j = 0; j < noColsBookedSeats; j++){

      dataStream.writeInt(allBookedSeats[i][j]);
  }

} // end writeBean()


/**
 * Read the cinema hall configuration from the network and
 * creates the Background_Cinema_Hall_Conf_Resp_Bean bean to store all
     details.
 * This bean is to be used later on to construct the UI on the MIDlet
     side
 *
 * @param dataStream The DataStreamInput to read the cinema hall conf
     details
 * @return Background_Cinema_Hall_Conf_Resp_Bean that stores all cinema
      hall conf details
 * @throws IOException
 */
public static Response_Msg_Bean readBean(DataInputStream dataStream)
    throws IOException {

  Background_Cinema_Hall_Conf_Resp_Bean backCinHallConfRespBean = new
      Background_Cinema_Hall_Conf_Resp_Bean();
  System.out.println("---------␣In␣the␣BACKGROUND␣CINEMA␣HALL␣
      CONFIGURATION␣RESPONSE␣BEAN␣-␣before␣reading");

  backCinHallConfRespBean.responseCode  = dataStream.readInt();
  backCinHallConfRespBean.noRowsBookedSeats = dataStream.readInt();
  backCinHallConfRespBean.noColsBookedSeats = dataStream.readInt();

  // read all bookes seats values
  backCinHallConfRespBean.allBookedSeats = new int[
```

```
      backCinHallConfRespBean.noRowsBookedSeats][
      backCinHallConfRespBean.noColsBookedSeats];
  for (int i = 0; i < backCinHallConfRespBean.noRowsBookedSeats; i++)
    for (int j = 0; j < backCinHallConfRespBean.noColsBookedSeats; j++)
        {

      backCinHallConfRespBean.allBookedSeats[i][j] = dataStream.readInt
          ();
  }

  return backCinHallConfRespBean;

} // end readBean()


/**
 * Return the string representation of the
     Background_Cinema_Hall_Conf_Resp_Bean
 *
 * @return The string representation of the
     Background_Cinema_Hall_Conf_Resp_Bean
 */
public String toString(){

  String res = "----␣Background_Cinema_Hall_Conf_Resp_Bean␣␣----\n";
  String seatsStr = "";

  for (int i = 0; i < noRowsBookedSeats; i++){
    for (int j = 0; j < noColsBookedSeats; j++){
      seatsStr += allBookedSeats[i][j] + "␣␣";
    }
    seatsStr += "|␣";
  }

  res += "-----------------------------------------------␣--\n";
  res += "SQL␣Response␣Code:␣" + super.getResponseCode() + "\n";
  res += "No␣Of␣Rows:␣␣␣␣␣␣␣" + noRowsBookedSeats   + "\n";
  res += "No␣Of␣Cols:␣␣␣␣␣␣␣" + noColsBookedSeats   + "\n";
  res += "All␣Booked␣Seats:␣" + seatsStr            + "\n";
  res += "-----------------------------------------------\n";

  return res;

} // end toString()
```

```
  /**
   * Compare 2 Cinema_Hall_Conf_Resp_Bean objects
   *
   * @param object a Cinema_Hall_Conf_Resp_Bean objects
   */
  public boolean equals(Object object) {
    return object != null
    && (object instanceof Background_Cinema_Hall_Conf_Resp_Bean
        && (object == this
            || (((Background_Cinema_Hall_Conf_Resp_Bean) object).
                getAllBookedSeats() == allBookedSeats)));

  } // end equals()

}// end class

package model.beans.responsebeans;

import java.io.*;



/**
 * This is a Cinema Hall Conf Java Bean that contains information about
 * cinema hall that display the given show. The server side creates
 * this bean after the Select Show Servlet
 * has retrieved the details about the show displayed in the given
     ShowLocalID
 * and at the given showTimeID.
 * This happens only if there was no error.
 *
 * Then, the bean is sent via HTTP to the MIDlet, and the information
     loaded in
 * the bean is extracted on the MIDlet and used to create the UI
     displaying the
 * Cinema Hall Configuration
 *
 * It inherits from the Response_Msg_Bean in order to set, serialize,
     deserialize
 * the response code value from the sql stored procedure
 *
 * @author Mihai Balan - s031288
 *
 */
public class Cinema_Hall_Conf_Resp_Bean extends Response_Msg_Bean{

  // ============================================================================
```

```
//                            PROPERTIES
// ============================================================================

//private static final long serialVersionUID = 1L;

/** Cinema Hall Ticket basePrice */
private double basePrice = 0.0;

/** No. of discount values in the discountValues[] */
private int   noDiscValue = 0;

/** The discount Values for that Cinema Hall */
private double discountValues[] = null;

/** No. of Rows for that Cinema Hall */
private int   rows = 0;

/** No. of Cols for that Cinema Hall */
private int   cols = 0;

/** No of rows in the array of AllBookedSeats[] */
private int   noRowsBookedSeats = 0;

/** No of cols in the array of AllBookedSeats[] */
private int   noColsBookedSeats = 0;

/** All Booked Seats in that Cinema Hall */
private int   allBookedSeats[][] = null;

/**
 * Constructor 1
 *
 */
public Cinema_Hall_Conf_Resp_Bean() {
  super();
}

/**
 * Constructor 2
 *
 * @param responseCode The response code from the SQL stored procedure
 */
public Cinema_Hall_Conf_Resp_Bean(int responseCode) {
  super(responseCode);
}

// ============================================================================
```

```
//                          SET METHODS
// ============================================================================


public void setBasePrice(double basePrice){
  this.basePrice = basePrice;
}

public void setNoDiscValue(int noDiscValue){
  this.noDiscValue = noDiscValue;
}

public void setDiscValues(double[] discountValues){
  this.discountValues = discountValues;
}

public void setRows(int rows){
  this.rows = rows;
}

public void setCols(int cols){
  this.cols = cols;
}

public void setNoRowsBookedSeats(int noRowBookedSeats){
  this.noRowsBookedSeats = noRowBookedSeats;
}

public void setNoColsBookedSeats(int noColsBookedSeats){
  this.noColsBookedSeats = noColsBookedSeats;
}

public void setAllBookedSeats(int[][] allBookedSeats){
  this.allBookedSeats = allBookedSeats;
}

// ============================================================================

//                          GET METHODS
// ============================================================================


public double getBasePrice(){
  return basePrice;
}
```

```
public int getNoDiscValue(){
  return noDiscValue;
}

public double[] getDiscountValues(){
  return discountValues;
}

public int getRows(){
  return rows;
}

public int getCols(){
  return cols;
}

public int getAllBookedSeatsRows(){
  return noRowsBookedSeats;
}

public int getAllBookedSeatsCols(){
  return noColsBookedSeats;
}

public int[][] getAllBookedSeats(){
  return allBookedSeats;
}


// ============================================================================

//                  READ/WRITE METHODS
// ============================================================================

/**
 * Write the cinema hall configuration to the network
 *
 * @param dataStream The DataStreamOutput to write the movie details to
 */
public void writeBean(DataOutputStream dataStream) throws IOException {

  dataStream.writeInt(super.responseCode);
  dataStream.writeDouble(basePrice);
  dataStream.writeInt(noDiscValue);

  // write all elements in the Discount Values Array
  for (int i = 0; i < noDiscValue; i++){
```

```
    dataStream.writeDouble(discountValues[i]);
  }
  dataStream.writeInt(rows);
  dataStream.writeInt(cols);
  dataStream.writeInt(noRowsBookedSeats);
  dataStream.writeInt(noColsBookedSeats);

  // write all elements in the All Booked Seats Array
  for (int i = 0; i < noRowsBookedSeats; i++)
    for (int j = 0; j < noColsBookedSeats; j++){

      dataStream.writeInt(allBookedSeats[i][j]);
    }

} // end writeBean()


/**
 * Read the cinema hall configuration from the network and
 * creates the Cinema_Hall_Conf_Resp_Bean bean to store all details.
 * This bean is to be used later on to construct the UI on the MIDlet
     side
 *
 * @param dataStream The DataStreamInput to read the cinema hall conf
     details
 * @return Cinema_Hall_Conf_Resp_Bean that stores all cinema hall conf
     details
 * @throws IOException
 */
public static Response_Msg_Bean readBean(DataInputStream dataStream)
    throws IOException {

  Cinema_Hall_Conf_Resp_Bean cinHallConfRespBean = new
      Cinema_Hall_Conf_Resp_Bean();
  System.out.println("---------␣In␣the␣CINEMA␣HALL␣CONFIGURATION␣
      RESPONSE␣BEAN␣-␣before␣reading");

  cinHallConfRespBean.responseCode  = dataStream.readInt();
  cinHallConfRespBean.basePrice     = dataStream.readDouble();
  cinHallConfRespBean.noDiscValue   = dataStream.readInt();

  // read all Discount Values
  cinHallConfRespBean.discountValues = new double[cinHallConfRespBean.
      noDiscValue];
  for (int i = 0; i < cinHallConfRespBean.noDiscValue; i++){
```

```
    cinHallConfRespBean.discountValues[i] = dataStream.readDouble();
  }

  cinHallConfRespBean.rows          = dataStream.readInt();
  cinHallConfRespBean.cols          = dataStream.readInt();

  cinHallConfRespBean.noRowsBookedSeats = dataStream.readInt();
  cinHallConfRespBean.noColsBookedSeats = dataStream.readInt();

  // read all bookes seats values
  cinHallConfRespBean.allBookedSeats = new int[cinHallConfRespBean.
      noRowsBookedSeats][cinHallConfRespBean.noColsBookedSeats];
  for (int i = 0; i < cinHallConfRespBean.noRowsBookedSeats; i++)
    for (int j = 0; j < cinHallConfRespBean.noColsBookedSeats; j++){

      cinHallConfRespBean.allBookedSeats[i][j] = dataStream.readInt();
    }

  return cinHallConfRespBean;

} // end readBean()



/**
 * Return the string representation of the Cinema_Hall_Conf_Resp_Bean
 *
 * @return The string representation of the Cinema_Hall_Conf_Resp_Bean
 */
public String toString(){

  String res = "----␣Cinema_Hall_Conf_Resp_Bean␣␣----\n";
  String seatsStr = "";
  String discStr = "";

  for (int i = 0; i < noRowsBookedSeats; i++){
    for (int j = 0; j < noColsBookedSeats; j++){
      seatsStr += allBookedSeats[i][j] + "␣␣";
    }
    seatsStr += "|␣";
  }

  for (int i = 0; i < noDiscValue; i++){
    discStr += discountValues[i] + "␣|␣";
  }

  res += "-----------------------------------------------------\n";
```

```
    res += "SQL␣Response␣Code:␣␣␣␣␣" + super.getResponseCode() + "\n";
    res += "Base␣Price:␣␣␣␣␣␣␣␣␣␣␣" + basePrice         + "\n";
    res += "No␣of␣Discount␣Values:␣" + noDiscValue       + "\n";
    res += "Discount␣Values:␣␣␣␣␣␣␣" + discStr           + "\n";
    res += "Cinema␣Hall␣Rows:␣␣␣␣␣␣" + rows              + "\n";
    res += "Cinema␣Hall␣Cols:␣␣␣␣␣␣" + cols              + "\n";
    res += "No␣Of␣Rows:␣␣␣␣␣␣␣␣␣␣␣" + noRowsBookedSeats + "\n";
    res += "No␣Of␣Cols␣:␣␣␣␣␣␣␣␣␣␣" + noColsBookedSeats + "\n";
    res += "All␣Booked␣Seats:␣␣␣␣␣␣" + seatsStr          + "\n";
    res += "-------------------------------------------------------\n";

    return res;

  } // end toString()



  /**
   * Compare 2 Cinema_Hall_Conf_Resp_Bean objects
   *
   * @param object a Cinema_Hall_Conf_Resp_Bean objects
   */
  public boolean equals(Object object) {
    return object != null
    && (object instanceof Cinema_Hall_Conf_Resp_Bean
        && (object == this
            || (((Cinema_Hall_Conf_Resp_Bean) object).getBasePrice() ==
                basePrice)
            && (((Cinema_Hall_Conf_Resp_Bean) object).getRows() == rows)
            && (((Cinema_Hall_Conf_Resp_Bean) object).getCols() == cols)))
                ;

  } // end equals()

}// end class

package model.beans.responsebeans;

import java.io.*;



/**
 * Response Java Bean sent from the server side to the MIDlet.
 * It contains the found movies for the given location data by the user.
 * Cinema Controller populates this bean using the SQL result of the
 * SQL query for finding movies.
 *
```

```
 * Then, the bean is sent via HTTP to the MIDlet, and the information
     loaded in
 * the bean is extracted on the MIDlet and used to create the UI
     displaying the
 * Movies
 *
 * It inherits from the Response_Msg_Bean in order to set, serialize,
     deserialize
 * the response code value from the sql stored procedure
 *
 * @author Mihai balan - s031288
 *
 */
public class Find_Movies_Resp_Bean extends Response_Msg_Bean{

  // ============================================================================

  //                            PROPERTIES
  // ============================================================================


  /** No of found movies matching the given criteria */
  private int row_no = 0;

  /** Movie details i.e. Movie Name, hour, Cinema, City, Street,
      showLocationID, showTimeID */
  private String[][] movies = null;

  /**
   * Constructor 1
   */
  public Find_Movies_Resp_Bean(){}

  /**
   * Constructor 2
   *
   * @param responseCode The response code from the SQL stored procedure
   */
  public Find_Movies_Resp_Bean(int responseCode) {
    super(responseCode);
  }
  // ============================================================================

  //                            SET METHODS
  // ============================================================================
```

```
public void setRow_No(int row_no){
  this.row_no = row_no;
}

public void setMovies(String[][] movies){
  this.movies = movies;
}

// ============================================================================

//                            GET METHODS
// ============================================================================

public int getRow_No(){
  return row_no;
}

public String[][] getMovies(){
  return movies;
}


// ============================================================================

//                   READ/WRITE METHODS
// ============================================================================


/**
 * Write bean properties to the network
 *
 * @param dataStream The stream used for writing the data
 * @throws IOException
 */
public void writeBean(DataOutputStream dataStream) throws IOException {
  dataStream.writeInt(super.responseCode);
  dataStream.writeInt(row_no);

  for (int i = 0; i < row_no; i++){
    for (int j = 0; j < 7 ; j++){
      dataStream.writeUTF(movies[i][j]);
    }
  }

} // end writeBean()

/**
```

```
 * Read the bean properties
 * from the network and cosntruct the Find_Movies_Req_Bean
 * sent by the MIDlet
 *
 * @param dataStream The stream used for reading the data
 * @return The read Find_Movies_Resp_Bean
 * @throws IOException
 */
public static Response_Msg_Bean readBean(DataInputStream dataStream)
    throws IOException{

  Find_Movies_Resp_Bean findMovRespBean = new Find_Movies_Resp_Bean();
  findMovRespBean.responseCode = dataStream.readInt();
  findMovRespBean.row_no    = dataStream.readInt();

  int tmpRowNo = findMovRespBean.row_no;
  findMovRespBean.movies = new String[tmpRowNo][7];
  for (int i = 0; i < tmpRowNo; i++){
    for (int j = 0; j < 7 ; j++){
      findMovRespBean.movies[i][j] = dataStream.readUTF();
    }
  }

  return findMovRespBean;

} // end readBean()


/**
 * Return the string representation of the Find_Movies_Resp_Bean
 *
 * @return The string representation of the Find_Movies_Resp_Bean
 */
public String toString(){

  String res = "----␣Find_Movies_Resp_Bean␣␣----\n";
  String movStr = "";

  for (int i = 0; i < row_no; i++){
    for (int j = 0; j < 7; j++){
      movStr += movies[i][j] + "␣|␣";
    }
    movStr += "\n";
  }

  res += "-------------------------------------------------\n";
  res += "SQL␣Response␣Code:␣" + super.responseCode + "\n";
```

```
    res += "No␣of␣Found␣Movies:␣" + row_no        + "\n";
    res += "MOVIES␣␣␣␣:␣␣␣␣␣␣␣\n" + movStr        + "\n";
    res += "-------------------------------------------------\n";

    return res;

  } // end toString()



  /**
   * Compares two Find_Movies_Req_Bean objects
   * @param object A Find_Movies_Req_Bean object
   */
  public boolean equals(Object object) {
    return object != null
    && (object instanceof Find_Movies_Resp_Bean
        && (object == this
            || (((Find_Movies_Resp_Bean) object).getRow_No() == row_no))
            && (((Find_Movies_Resp_Bean) object).getResponseCode() ==
                super.responseCode));
  } // end equals()

} // end class

package model.beans.responsebeans;

import java.io.*;



/**
 * This is a Movie Details Java Bean that contains information about
 * movies. The server side creates this bean after the Movie Details
     Servlet
 * has retrieved the details about the movie displayed in the given
     ShowLocalID.
 * This happens only if there was no error.
 *
 * Then, the bean is sent via HTTP to the MIDlet, and the information
     loaded in
 * the bean is extracted on the MIDlet and used to create the UI
     displaying the
 * Movie Details
 *
 * It inherits from the Response_Msg_Bean in order to set, serialize,
     deserialize
 * the response code value from the sql stored procedure
```

```
 *
 * @author Mihai Balan - s031288
 *
 */
public class Movie_Details_Resp_Bean extends Response_Msg_Bean{

  // ============================================================================

  //                           PROPERTIES
  // ============================================================================

  //private static final long serialVersionUID = 1L;
  private String movieID               = "";
  private String movieName             = "";
  private String movieDuration         = "";
  private String movieGenre            = "";
  private String movieParentClassification = "";
  private String movieLanguage         = "";
  private String movieYear             = "";
  private String movieCountry          = "";
  private String userRating            = "";
  private String movieDirector         = "";
  private String movieActors           = "";
  private String movieDescription      = "";
  private byte[] moviePoster           = null;

  /**
   * Constructor 1
   */
  public Movie_Details_Resp_Bean() {}

  /**
   * Constructor 2
   */
  public Movie_Details_Resp_Bean( String movieID,
      String movieName ,
      String movieDuration,
      String movieGenre,
      String movieParentClassification,
      String movieLanguage,
      String movieYear,
      String movieCountry,
      String userRating,
      String movieDirector,
      String movieActors,
      String movieDescription )
  {
```

```
  this.movieID = movieID;
  this.movieName = movieName;
  this.movieDuration = movieDuration;
  this.movieGenre = movieGenre;
  this.movieParentClassification = movieParentClassification;
  this.movieLanguage = movieLanguage;
  this.movieYear = movieYear;
  this.movieCountry = movieCountry;
  this.userRating = userRating;
  this.movieDirector = movieDirector;
  this.movieActors = movieActors;
  this.movieDescription = movieDescription;
}

/**
 * Constructor 3
 *
 * @param responseCode The response code from the SQL stored procedure
 */
public Movie_Details_Resp_Bean(int responseCode) {
  super(responseCode);
}

// ============================================================================

//                            SET METHODS
// ============================================================================


public void setMovieID(String movieID){
  this.movieID = movieID;
}

public void setMovieName(String movieName){
  this.movieName = movieName;
}

public void setMovieDuration(String movieDuration){
  this.movieDuration = movieDuration;
}

public void setMovieGenre(String movieGenre){
  this.movieGenre = movieGenre;
}

public void setMovieParentClassification(String
    movieParentClassification){
```

```
  this.movieParentClassification = movieParentClassification;
}

public void setMovieLanguage(String movieLanguage){
  this.movieLanguage = movieLanguage;
}

public void setMovieYear(String movieYear){
  this.movieYear = movieYear;
}

public void setMovieCountry(String movieCountry){
  this.movieCountry = movieCountry;
}

public void setMovieUserRating(String userRating){
  this.userRating = userRating;
}

public void setMovieDirector(String movieDirector){
  this.movieDirector = movieDirector;
}

public void setMovieActors(String movieActors){
  this.movieActors = movieActors;
}

public void setMovieDescription(String movieDescription){
  this.movieDescription = movieDescription;
}

public void setMoviePoster(byte[] moviePoster ){
  this.moviePoster = moviePoster;
}

// ============================================================================

//                          GET METHODS
// ============================================================================


public String getMovieID(){
  return this.movieID;
}

public String getMovieName(){
  return this.movieName;
```

```
}

public String getMovieDuration(){
  return this.movieDuration;
}

public String getMovieGenre(){
  return this.movieGenre;
}

public String getMovieParentClassification(){
  return this.movieParentClassification;
}

public String getMovieLanguage(){
  return this.movieLanguage;
}

public String getMovieYear() {
  return this.movieYear;
}

public String getMovieCountry(){
  return this.movieCountry;
}

public String getMovieUserRating(){
  return this.userRating;
}

public String getMovieDirector(){
  return this.movieDirector;
}

public String getMovieActors(){
  return this.movieActors;
}

public String getMovieDescription(){
  return this.movieDescription;
}

public byte[] getMoviePoster(){
  return this.moviePoster;
}
```

```
// ============================================================================

//                  READ/WRITE METHODS
// ============================================================================

/**
 * Write the movie details to the network
 *
 * @param dataStream The DataStreamOutput to write the movie details to
 */
public void writeBean(DataOutputStream dataStream) throws IOException {

  dataStream.writeInt(super.responseCode);
  dataStream.writeUTF(movieID);
  dataStream.writeUTF(movieName);
  dataStream.writeUTF(movieDuration);
  dataStream.writeUTF(movieGenre);
  dataStream.writeUTF(movieParentClassification);
  dataStream.writeUTF(movieLanguage);
  dataStream.writeUTF(movieYear);
  dataStream.writeUTF(movieCountry);
  dataStream.writeUTF(userRating);
  dataStream.writeUTF(movieDirector);
  dataStream.writeUTF(movieActors);
  dataStream.writeUTF(movieDescription);

  if (moviePoster != null){
    System.out.println("
        ------------------------------------------------------------␣not
        ␣null");
    dataStream.writeInt(moviePoster.length);
    dataStream.write(moviePoster);
  } else {
    dataStream.writeInt(0);
    //dataStream.write("".getBytes());
  }

} // end writeBean()


/**
 * Read the movie details from the network and
 * creates the Movie_Details_Resp_Bean bean to store all details.
 * This bean is to be used later on to construct the UI on the MIDlet
     side
 *
 * @param dataStream The DataStreamInput to read the movie details from
```

```
 * @return Movie_Details_Resp_Bean that stores all movie details
 * @throws IOException
 */
public static Response_Msg_Bean readBean(DataInputStream dataStream)
    throws IOException {

  Movie_Details_Resp_Bean movDetBean = new Movie_Details_Resp_Bean();
  movDetBean.responseCode          = dataStream.readInt();
  System.out.println("---------␣In␣the␣MOVIE␣BEAN␣-␣before␣reading");
  movDetBean.movieID               = dataStream.readUTF();
  System.out.println("---------␣In␣the␣MOVIE␣BEAN␣" + movDetBean.
      getMovieID());
  movDetBean.movieName             = dataStream.readUTF();
  movDetBean.movieDuration         = dataStream.readUTF();
  movDetBean.movieGenre            = dataStream.readUTF();
  movDetBean.movieParentClassification = dataStream.readUTF();
  movDetBean.movieLanguage         = dataStream.readUTF();
  movDetBean.movieYear             = dataStream.readUTF();
  movDetBean.movieCountry          = dataStream.readUTF();
  movDetBean.userRating            = dataStream.readUTF();
  movDetBean.movieDirector         = dataStream.readUTF();
  movDetBean.movieActors           = dataStream.readUTF();
  movDetBean.movieDescription      = dataStream.readUTF();
  // used for reading the poster data
  byte[] poster = new byte[dataStream.readInt()];
  dataStream.readFully(poster);
  movDetBean.moviePoster           = poster;

  return movDetBean;

} // end readBean()



/**
 * Return the string representation of the Movie_Details_Resp_Bean
 *
 * @return The string representation of the Movie_Details_Resp_Bean
 */
public String toString(){

  String res = "----␣Movie_Details_Resp_Bean␣␣----\n";

  res += "-------------------------------------------------------------------------------\
      n";
  res += "SQL␣Response␣Code:␣␣␣␣␣␣␣␣␣␣" + super.responseCode + "\n";
  res += "Movie␣ID:␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣" + movieID            + "\n";
```

```
    res += "Movie␣Name:␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣" + movieName          + "\n";
    res += "Movie␣Duration:␣␣␣␣␣␣␣␣␣␣␣␣␣␣" + movieDuration      + "\n";
    res += "Movie␣Genre:␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣" + movieGenre         + "\n";
    res += "Movie␣Parent␣Classification:␣" + movieParentClassification +
        "\n";
    res += "Movie␣Language:␣␣␣␣␣␣␣␣␣␣␣␣␣" + movieLanguage      + "\n";
    res += "Movie␣Year:␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣" + movieYear          + "\n";
    res += "Movie␣Country:␣␣␣␣␣␣␣␣␣␣␣␣␣" + movieCountry       + "\n";
    res += "User␣Rating:␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣" + userRating         + "\n";
    res += "Movie␣Director:␣␣␣␣␣␣␣␣␣␣␣␣" + movieDirector      + "\n";
    res += "Movie␣Actors:␣␣␣␣␣␣␣␣␣␣␣␣␣" + movieActors        + "\n";
    res += "Movie␣Description:␣␣␣␣␣␣␣␣␣" + movieDescription   + "\n";
    res += "Movie␣Poster:␣␣␣␣␣␣␣␣␣␣␣␣␣" + moviePoster        + "\n";
    res += "-------------------------------------------------------------------------------\
        n";

    return res;

  } // end toString()


  /**
   * Compare 2 Movie_Details_Resp_Bean objects
   *
   * @param object a Movie_Details_Resp_Bean objects
   */
  public boolean equals(Object object) {
    return object != null
    && (object instanceof Movie_Details_Resp_Bean
        && (object == this
            || ((Movie_Details_Resp_Bean) object).getMovieID().equals(
                movieID)));

  } // end equals()

}// end class

package model.beans.responsebeans;

import java.io.*;



/**
 * This is a Purchase_Tickets_Resp_Bean that contains information about
 * the current tickets purchased by the user.
 *
 * The following info is contained in the bean
```

```
 *  - reservationID
 *  - total proce to be paid
 *  - left e-money in the user's account
 *  - ticketID[]
 *  - prices[]
 *
 * The server side creates this bean after the Purchased_Tickets_Servlet
 * has retrieved the details about the current user reservation.
 * This happens only if there was no error.
 *
 * Then, the bean is sent via HTTP to the MIDlet, and the information
     loaded in
 * the bean is extracted on the MIDlet and used to create the UI
     displaying and
 * saving the reservation and ticket information
 *
 * It inherits from the Response_Msg_Bean in order to set, serialize,
     deserialize
 * the response code value from the sql stored procedure
 *
 * @author Mihai Balan - s031288
 *
 */
public class Purchase_Tickets_Resp_Bean extends Response_Msg_Bean{

  // ============================================================================

  //                            PROPERTIES
  // ============================================================================

  //private static final long serialVersionUID = 1L;

  /** User reservation ID */
  private String reservationID = "";

  /** Total price for all purchasd tickets in the currebt reservation */
  private double totalPrice = 0.0;

  /** Left e-money in user's account */
  private double leftEmoney = 0.0;

  /** the number of tickets purchasd by the user */
  private int noOfTickets = 0;

  /** User purchased ticket IDs */
  private String ticketIDs[] = null;
```

```
/** Ticket Prices payed by the user*/
private double ticketPrices[] = null;

/**
 * Constructor 1
 */
public Purchase_Tickets_Resp_Bean() {}

/**
 * Constructor 2
 *
 * @param responseCode The response code from the SQL stored procedure
 */
public Purchase_Tickets_Resp_Bean(int responseCode) {
  super(responseCode);
}

// ============================================================================

//                           SET METHODS
// ============================================================================


public void setReservationID(String reservationID ){
  this.reservationID = reservationID;
}

public void setTotalPrice(double totalPrice){
  this.totalPrice = totalPrice;
}

public void setLeftEmoney(double leftEmoney){
  this.leftEmoney = leftEmoney;
}

public void setNoOfTickets(int noOfTickets){
  this.noOfTickets = noOfTickets;
}

public void setTicketIDs(String[] ticketIDs){
  this.ticketIDs = ticketIDs;
}

public void setTicketPrices(double[] ticketPrices){
  this.ticketPrices = ticketPrices;
}
```

```
// ============================================================================

//                            GET METHODS
// ============================================================================


public String getReservationID(){
  return reservationID;
}

public double getTotalPrice(){
  return totalPrice;
}

public double getLeftEmoney(){
  return leftEmoney;
}

public int getNoOfTickets(){
  return noOfTickets;
}

public String[] getTicketIDs(){
  return ticketIDs;
}

public double[] getTicketPrices(){
  return ticketPrices;
}


// ============================================================================

//                  READ/WRITE METHODS
// ============================================================================

/**
 * Write the reservation details to the network
 *
 * @param dataStream The DataStreamOutput to write the reservation
     details to.
 */
public void writeBean(DataOutputStream dataStream) throws IOException {

  dataStream.writeInt(super.responseCode);
  dataStream.writeUTF(reservationID);
```

```
  dataStream.writeDouble(totalPrice);
  dataStream.writeDouble(leftEmoney);
  dataStream.writeInt(noOfTickets);

  // write all tickets for the current reservation
  for (int i = 0; i < noOfTickets; i++){
    dataStream.writeUTF(ticketIDs[i]);
  }

  // write all ticket prices for the current reservation
  for (int i = 0; i < noOfTickets; i++){
    dataStream.writeDouble(ticketPrices[i]);
  }

} // end writeBean()


/**
 * Read the user reservation details from the network and
 * creates the Purchased_Tickets_Resp_Bean bean to store all details.
 * This bean is to be used later on to construct the UI on the MIDlet
     side
 * and save the reservation info into RMS
 *
 * @param dataStream The DataStreamInput to read the reservation
     details
 * @return Purchased_Tickets_Resp_Bean that stores all reservation
     details
 * @throws IOException
 */
public static Response_Msg_Bean readBean(DataInputStream dataStream)
    throws IOException {

  Purchase_Tickets_Resp_Bean purchaseTicketsRespBean = new
      Purchase_Tickets_Resp_Bean();
  System.out.println("---------␣In␣the␣CINEMA␣HALL␣CONFIGURATION␣
      RESPONSE␣BEAN␣-␣before␣reading");

  purchaseTicketsRespBean.responseCode = dataStream.readInt();
  purchaseTicketsRespBean.reservationID = dataStream.readUTF();
  purchaseTicketsRespBean.totalPrice  = dataStream.readDouble();
  purchaseTicketsRespBean.leftEmoney  = dataStream.readDouble();
  purchaseTicketsRespBean.noOfTickets = dataStream.readInt();

  // read all Ticket IDs values
  purchaseTicketsRespBean.ticketIDs   = new String[
      purchaseTicketsRespBean.noOfTickets];
```

```
  for (int i = 0; i < purchaseTicketsRespBean.noOfTickets; i++){
    purchaseTicketsRespBean.ticketIDs[i] = dataStream.readUTF();
  }

  // read all Ticket price values
  purchaseTicketsRespBean.ticketPrices = new double[
      purchaseTicketsRespBean.noOfTickets];
  for (int i = 0; i < purchaseTicketsRespBean.noOfTickets; i++){
    purchaseTicketsRespBean.ticketPrices[i] = dataStream.readDouble();
  }

  return purchaseTicketsRespBean;

} // end readBean()



/**
 * Return the string representation of the Purchase_Tickets_Resp_Bean
 *
 * @return The string representation of the Purchase_Tickets_Resp_Bean
 */
public String toString(){

  String res = "----␣Purchase_Tickets_Resp_Bean␣␣----\n";
  String tickIDStr = "";
  String tickPriceStr = "";

  for (int i = 0; i < noOfTickets; i++){
    tickIDStr += ticketIDs[i] + "␣|␣";
  }

  for (int i = 0; i < noOfTickets; i++){
    tickPriceStr += ticketPrices[i] + "␣|␣";
  }

  res += "-------------------------------------------------\n";
  res += "SQL␣Response␣Code:␣" + super.responseCode + "\n";
  res += "Reservation␣ID:␣␣␣␣" + reservationID + "\n";
  res += "Total␣Price:␣␣␣␣␣␣␣" + totalPrice    + "\n";
  res += "Left␣E-Money:␣␣␣␣␣␣" + leftEmoney    + "\n";
  res += "No␣Of␣Tickets:␣␣␣␣␣" + noOfTickets   + "\n";
  res += "Ticket␣ID's␣ID:␣␣␣␣" + tickIDStr     + "\n";
  res += "Ticket␣Prices:␣␣␣␣␣" + tickPriceStr  + "\n";
  res += "-------------------------------------------------\n";

  return res;
```

```
  } // end toString()


  /**
   * Compare 2 Purchase_Tickets_Resp_Bean objects
   *
   * @param object a Purchase_Tickets_Resp_Bean objects
   */
  public boolean equals(Object object) {
    return object != null
    && (object instanceof Purchase_Tickets_Resp_Bean
        && (object == this
            || (((Purchase_Tickets_Resp_Bean) object).getReservationID()
                == reservationID)
            && (((Purchase_Tickets_Resp_Bean) object).getTotalPrice() ==
                totalPrice)));

  } // end equals()

}// end class

package model.beans.responsebeans;

import java.io.*;


/**
 * This is a general Response Message Bean. It is used every time the
 * response from the server to the MIDlet contains only a response code
 * and maybe a message.
 * The response code corresponds to @see cinemaservice.constants.
     Error_Code_Constants
 *
 * The Bean is then used on the MIDlet side to display the coresponding
     UI
 * accordingly to the response code
 *
 * @author Mihai Balan - s031288
 *
 */
public class Response_Msg_Bean{


  // ============================================================================

  //                           PROPERTIES
  // ============================================================================
```

```
/**
 * Response code from the servlet to the midlet
 *
 * @see cinemaservice.constants.Error_Code_Constants
 *
 */
protected int responseCode;

/**
 * A message that might come with the response code
 */
protected String msg = "";


/**
 * Constructor 1
 *
 */
public Response_Msg_Bean(){}

/**
 * Constructor 2
 *
 * @param responseCode The responsecode from the sql stored procedure
 */
public Response_Msg_Bean(int responseCode){
  this.responseCode = responseCode;
}


// ============================================================================

//                            SET METHODS
// ============================================================================

/**
 * Set the response code value
 *
 * @param responseCode
 * @see cinemaservice.constants.Error_Code_Constants
 */
public void setResponseCode(int responseCode){
  this.responseCode = responseCode;

} // end setResponseCode()
```

```
/**
 * Set the message that might come with the response code
 *
 * @param msg Message that might come with the response code
 */
public void setMsg(String msg){
  this.msg = msg;

} // end setMsg()


// ============================================================================

//                            GET METHODS
// ============================================================================

/**
 * Get the response code value
 *
 * @return @see cinemaservice.constants.Error_Code_Constants
 */
public int getResponseCode(){
  return this.responseCode;

} // end getResponseCode()

/**
 * Get the message that might come with the response code
 *
 * @return Message that might come with the response code
 */
public String getMsg(){
  return msg;

} // end getMsg()



// ============================================================================

//                  READ/WRITE METHODS
// ============================================================================

/**
 * Write the Response_Msg_Bean data to the network
 *
 * @param dataStream The DataOutputStream to write the data to
```

```
 */
public void writeBean(DataOutputStream dataStream) throws IOException {
  dataStream.writeInt(responseCode);
  dataStream.writeUTF(msg);

} // end writeBean()


/**
 * Read the Response_Msg_Bean data from the network.
 * The Response_Msg_Bean is to be used to create the UI
 * on the MIDlet side
 *
 * @param dataStream The DataIntputStream to read the data from
 * @return Response_Msg_Bean to be used for generating the UI on the
     MIDlet side
 * @throws IOException
 */
public static Response_Msg_Bean readBean(DataInputStream dataStream)
    throws IOException {
  Response_Msg_Bean respMsgBean = new Response_Msg_Bean();
  respMsgBean.responseCode = dataStream.readInt();
  respMsgBean.msg = dataStream.readUTF();

  return respMsgBean;

} // end readBean()



/**
 * Return the string representation of the Response_Msg_Bean
 *
 * @return The string representation of the Response_Msg_Bean
 */
public String toString(){

  String res = "----␣Response_Msg_Bean␣␣----\n";

  res += "-----------------------------------------------\n";
  res += "Response␣Code:␣" + responseCode + "\n";
  res += "Message:␣␣␣␣␣␣␣" + msg      + "\n";
  res += "-----------------------------------------------\n";

  return res;

} // end toString()
```

```
  /**
   * Compare 2 Response_Msg_Bean objects
   *
   * @param object a Response_Msg_Bean objects
   */
  public boolean equals(Object object) {
    return object != null
    && (object instanceof Response_Msg_Bean
        && (object == this
            || ((Response_Msg_Bean) object).getResponseCode() == (
                responseCode)));

  } // end equals()

} // end class

package model.beans.responsebeans;

import java.io.*;



/**
 * This is a Select_Deselect_Seats_Resp_Bean that contains all booked
     seats
 * for a given show. It also includes the recently booked seats by the
     user.
 * The server side creates this bean after the
     Select_Deselect_Seats_Servlet
 * has selected / deselected the seats set as request by the user for
 * a given ShowLocalID and showTimeID.
 * This happens only if there was no error.
 *
 * Then, the bean is sent via HTTP to the MIDlet, and the information
     loaded in
 * the bean is extracted on the MIDlet and used to update the UI
     displaying the
 * Cinema Hall Configuration
 *
 * It inherits from the Response_Msg_Bean in order to set, serialize,
     deserialize
 * the response code value from the sql stored procedure
 *
 * @author Mihai Balan - s031288
 *
 */
```

```
public class Select_Deselect_Seats_Resp_Bean extends Response_Msg_Bean{

  // ============================================================================

  //                            PROPERTIES
  // ============================================================================

  //private static final long serialVersionUID = 1L;

  /** No of rows in the array of AllBookedSeats[] */
  private int noRowsBookedSeats = 0;

  /** No of cols in the array of AllBookedSeats[] */
  private int noColsBookedSeats = 0;

  /** All Booked Seats in that Cinema Hall */
  private int allBookedSeats[][] = null;

  /**
   * Constructor 1
   */
  public Select_Deselect_Seats_Resp_Bean() {}

  /**
   * Constructor 2
   *
   * @param responseCode The response code from the SQL stored procedure
   */
  public Select_Deselect_Seats_Resp_Bean(int responseCode) {
    super(responseCode);
  }

  // ============================================================================

  //                            SET METHODS
  // ============================================================================


  public void setNoRowsBookedSeats(int noRowBookedSeats){
    this.noRowsBookedSeats = noRowBookedSeats;
  }

  public void setNoColsBookedSeats(int noColsBookedSeats){
    this.noColsBookedSeats = noColsBookedSeats;
  }

  public void setAllBookedSeats(int[][] allBookedSeats){
```

```
  this.allBookedSeats = allBookedSeats;
}

// ============================================================================

//                            GET METHODS
// ============================================================================


public int getAllBookedSeatsRows(){
  return noRowsBookedSeats;
}

public int getAllBookedSeatsCols(){
  return noColsBookedSeats;
}

public int[][] getAllBookedSeats(){
  return allBookedSeats;
}


// ============================================================================

//                   READ/WRITE METHODS
// ============================================================================

/**
 * Write the cinema hall configuration to the network
 *
 * @param dataStream The DataStreamOutput to write all booked seats to
 */
public void writeBean(DataOutputStream dataStream) throws IOException {

  dataStream.writeInt(super.responseCode);
  dataStream.writeInt(noRowsBookedSeats);
  dataStream.writeInt(noColsBookedSeats);

  // write all elements in the All Booked Seats Array
  for (int i = 0; i < noRowsBookedSeats; i++)
    for (int j = 0; j < noColsBookedSeats; j++){

      dataStream.writeInt(allBookedSeats[i][j]);
  }

} // end writeBean()
```

```
/**
 * Read all booked seats from the network and
 * creates the Select_Deselect_Seats_Resp_Bean bean to store all seats
 * for the given show.
 * This bean is to be used later on to construct the UI on the MIDlet
     side
 *
 * @param dataStream The DataStreamInput to read the cinema hall conf
     details
 * @return Select_Deselect_Seats_Resp_Bean that stores all booked seats
 * @throws IOException
 */
public static Response_Msg_Bean readBean(DataInputStream dataStream)
    throws IOException {

  Select_Deselect_Seats_Resp_Bean selDeselSeatsRespBean = new
      Select_Deselect_Seats_Resp_Bean();
  System.out.println("---------␣In␣the␣SELECT␣DESELECT␣SEATS␣RESPONSE␣
      BEAN␣-␣before␣reading");

  selDeselSeatsRespBean.responseCode  = dataStream.readInt();
  selDeselSeatsRespBean.noRowsBookedSeats = dataStream.readInt();
  selDeselSeatsRespBean.noColsBookedSeats = dataStream.readInt();

  // read all bookes seats values
  selDeselSeatsRespBean.allBookedSeats = new int[selDeselSeatsRespBean.
      noRowsBookedSeats][selDeselSeatsRespBean.noColsBookedSeats];
  for (int i = 0; i < selDeselSeatsRespBean.noRowsBookedSeats; i++)
    for (int j = 0; j < selDeselSeatsRespBean.noColsBookedSeats; j++){

      selDeselSeatsRespBean.allBookedSeats[i][j] = dataStream.readInt()
          ;
  }

  return selDeselSeatsRespBean;

} // end readBean()


/**
 * Return the string representation of the
     Select_Deselect_Seats_Resp_Bean
 *
 * @return The string representation of the
     Select_Deselect_Seats_Resp_Bean
 */
```

```
  public String toString(){

    String res = "----␣␣Select_Deselect_Seats_Resp_Bean␣␣----\n";
    String seatsStr = "";

    for (int i = 0; i < noRowsBookedSeats; i++){
      for (int j = 0; j < noColsBookedSeats; j++){
        seatsStr += allBookedSeats[i][j] + "␣";
      }
      seatsStr += "|␣";
    }

    res += "-------------------------------------------------\n";
    res += "SQL␣Response␣Code:␣" + super.getResponseCode() + "\n";
    res += "No␣Of␣Rows:␣␣␣␣␣␣␣␣" + noRowsBookedSeats  + "\n";
    res += "No␣Of␣Cols:␣␣␣␣␣␣␣␣" + noColsBookedSeats  + "\n";
    res += "All␣Booked␣Seats:␣␣" + seatsStr           + "\n";
    res += "-------------------------------------------------\n";

    return res;

  } // end toString()

  /**
   * Compare 2 Select_Deselect_Seats_Resp_Bean objects
   *
   * @param object a Select_Deselect_Seats_Resp_Bean objects
   */
  public boolean equals(Object object) {
    return object != null
    && (object instanceof Select_Deselect_Seats_Resp_Bean
        && (object == this
            || (((Select_Deselect_Seats_Resp_Bean) object).
                getAllBookedSeatsRows() == noRowsBookedSeats)
            && (((Select_Deselect_Seats_Resp_Bean) object).
                getAllBookedSeatsCols() == noColsBookedSeats)));

  } // end equals()

}// end class

package model.init;

import gui.purchasetickets.step1moviesearch.MovieSearchHelper;

import java.util.Date;
import java.util.Vector;
```

```
import model.beans.otherbeans.TicketBean;
import model.update.UpdateModel;
import cryptography.Encryptor;
import rms.RMSOperations;
import start.Start;

/**
 * Performs initialization operation for the midlet
 * e.g. reading, saving, deleting data from RMS,
 * creating the tickets[] and make it available for the whole application
 * in order to reduce the access to RMS, etc
 *
 * @author Mihai Balan (s031288)
 *
 */
public class InitModel {

  private static Encryptor decryptor;

  /**
   * Initiate all data in the midlet such as
   * openeing the record store at the application level,
   * getting all saved tickets in the RMS, etc.
   * Each user has its own recordstore.
   *
   * @throws Exception
   */
  public static void initModelFromRMS(
      String userName,
      String password,
      String key) throws Exception{

    String url = "http://127.0.0.1:9080/Cinema_Controller/cinemaservice/
        servlets/controller/Cinema_Central_Controller_Servlet?protocol=
        AT1";

    // open the record store to make it available to the whole
        application
    // each user has its own record store
    // Thus, the application can be used by multiple users
    RMSOperations.openRecStore(userName);

    // get all tickets saved in the phone memory
    Start.tickets = RMSOperations.getAllTickets();

    // set the number of tickets saved in RMS
    Start.maxTTSaved = Start.tickets.length;
```

```
System.out.println("\n----␣INIT␣MODEL␣---␣Tickets␣initialized!\n");

// save the user name to RMS if not already the same values are in
    RMS
if(!RMSOperations.getItem("USR:").equals(userName)){
  RMSOperations.deleteItems("USR:");
  RMSOperations.writeRecord("USR:", userName);
}
Start.userName = userName;

// get emoney from RMS
Start.emoney = RMSOperations.getItem("EMN:");

// selected theme
Start.themeName = RMSOperations.getItem("THM:");

if (Start.themeName.equals("")){
  Start.themeDir = "theme_red";
  RMSOperations.writeRecord("THM:", "red");

}else if (Start.themeName.equals("blue")){
  Start.themeDir = "theme_blue";

} else if (Start.themeName.equals("yellow")){
  Start.themeDir = "theme_yellow";

}

// delete all keys and insert a new one in RMS
// if not already the same values are in RMS
if(!RMSOperations.getItem("KEY:").equals(key)){
  RMSOperations.deleteItems("KEY:");
  RMSOperations.writeRecord("KEY:", key);
}
Start.userKey = key;

decryptor = new Encryptor(Start.userKey);

//get all credit cards saved in RMS
Start.creditCards = RMSOperations.getAllCreditCards(decryptor);
// set the no. of credit cards saved in RMS
Start.maxCCSaved = Start.creditCards.length;
System.out.println("\n----␣INIT␣MODEL␣---␣Credit␣cards␣initialized!\n
    ");
```

```
  // save the user PSWD to RMS if not already the same values are in
      RMS
  if(!new String(RMSOperations.getDecryptedItem("PSW:")).equals(
      password)){
    RMSOperations.deleteItems("PSW:");
    RMSOperations.writeByteRecord("PSW:", decryptor.encryptString(
        password));
  }
  Start.userPassword = password;

  // get the wallet PIN
  Start.walletPin = new String(decryptor.decrypt(RMSOperations.
      getByteItem("PIN:")));

  removeUsedTickets(Start.tickets);


}// end initModelFromRMS()


/**
 * Remove any used or expired tickets from the client mobile
 *
 * @param tickets
 */
private static void removeUsedTickets(TicketBean[] tickets) throws
    Exception{
  int count = 0;
  String today = MovieSearchHelper.parseDate((new Date()).toString());
  //System.out.println("Expired Tickets Date: " + today);
  int[] todayAr = tokenizeDate(today);
  Vector v = new Vector(10);
  boolean expired;

  for(int i=0; i<tickets.length; i++){
    //System.out.println("Tickets " + i + " Date: " + tickets[i].
        getTKTShowDate());
    int[] ticketDateAr = tokenizeDate(tickets[i].getTKTShowDate());

    // if there are movie today an alert is to be displayed
    if((todayAr[0] == ticketDateAr[0]) && (todayAr[1] == ticketDateAr
        [1]) && (todayAr[2] == ticketDateAr[2])){
      Start.needMovieAlert = true;
    }

    expired = false;
    for(int j=0; j<3; j++){
```

```
      if(todayAr[0] > ticketDateAr[0]){
        expired = true;
      }else if((todayAr[0] == ticketDateAr[0]) && (todayAr[1] >
          ticketDateAr[1])){
        expired = true;
      }else if((todayAr[0] == ticketDateAr[0]) && (todayAr[1] ==
          ticketDateAr[1]) && (todayAr[2] > ticketDateAr[2])){
        expired = true;
      }
    }

    if (expired){
      //System.out.println("Found Tickets " + i + " Date: " + tickets[i
          ].getTKTShowDate());
      ++count;
    }else{
      v.addElement(tickets[i]);

    }// end if (expired)

  }// end for(int i=0; i<tickets.length; i++)

  //System.out.println("Expired Tickets Count: " + count);
  //System.out.println("V size: " + v.size());



  // if there are expired tickets delete them from RMS and update RMS
  if(count > 0){
    Start.tickets = new TicketBean[v.size()];

    for(int i = 0; i < v.size(); i++){
      Start.tickets[i] = (TicketBean)v.elementAt(i);
      //System.out.println("Ticket " + i +"\n" +Start.tickets[i].
          toString());
    }

    UpdateModel.deleteAndUpdateExpiredTickets();

  }// end if(count > 0)



}// end removeUsedTickets()


/**
```

```
   * Tokenize a date in year, month, day
   *
   * @param date
   * @return
   */
  private static int[] tokenizeDate(String date){

    int[] tokenizedDate = new int[3];

    tokenizedDate[0] = Integer.parseInt(date.substring(0, date.indexOf("
        .")));
    //System.out.println("Year:" + tokenizedDate[0]);

    tokenizedDate[1] = Integer.parseInt(date.substring(date.indexOf(".")
        +1, date.indexOf(".", date.indexOf(".")+1)));
    //System.out.println("Month:" + tokenizedDate[1]);

    tokenizedDate[2] = Integer.parseInt(date.substring(date.lastIndexOf('
        .')+1));
    //System.out.println("Day:" + tokenizedDate[2]);

    return tokenizedDate;
  }

}// end class

package model.update;

import java.io.IOException;

import javax.microedition.lcdui.ChoiceGroup;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;

import model.beans.otherbeans.CreditCardBean;
import model.beans.otherbeans.TicketBean;

import org.bouncycastle.crypto.CryptoException;


import gui.customdialogwindows.CanvasAlert;
import gui.mainmenu.MenuScreen;
import gui.mytickets.MyTicketsMainMenu;
import gui.mywallet.MyWalletMainMenu;
import gui.mywallet.UpdateWalletBackground;
import rms.RMSOperations;
import start.Start;
import constants.CustomAlertTypes;
```

```
import constants.SystemConstants;

/**
 * Updates the data model i.e. tickets, credit cards
 * as result of a add, cancel opearation
 *
 * @author Mihai Balan, s031288
 *
 */
public class UpdateModel {

  private static CanvasAlert alert;

  /**
   * Updates the credit cards available to the application
   * after a add new CC operation. The RMS is also updated!
   *
   */
  public static void addNewCreditCardAndUpdateAllCreditCards(
      Display display, CreditCardBean ccBean)
  throws IOException, CryptoException, Exception {

    // save the new cc in RMS and update CC number
    RMSOperations.writeEncryptedRecord("CC" + (Start.maxCCSaved + 1) + "
        :", ccBean.getBytes());
    RMSOperations.deleteItems("CCN:");
    RMSOperations.writeRecord("CCN:", String.valueOf((Start.maxCCSaved
         + 1)));


    // update the Start,creditCards bean array and the maxmaxCCSaved

    // if there are no CC saved in the memory
    if(Start.maxCCSaved == 0){

      Start.creditCards = null;
      Start.maxCCSaved += 1;
      Start.creditCards = new CreditCardBean[Start.maxCCSaved];
      Start.creditCards[Start.maxCCSaved-1] = ccBean;

    }// end if(Start.maxCCSaved == 0)


    // if there are already credit cards in the memory
    else if(Start.maxCCSaved > 0){

      CreditCardBean[] tempCards = new CreditCardBean[Start.maxCCSaved];
```

```
    for(int i = 0; i < Start.maxCCSaved; i++){
      tempCards[i] = new CreditCardBean();
      tempCards[i] = Start.creditCards[i];

    }// end for()

    Start.creditCards = null;
    Start.maxCCSaved += 1;

    Start.creditCards = new CreditCardBean[Start.maxCCSaved];

    for(int i = 0; i < tempCards.length; i++){
      Start.creditCards[i] = new CreditCardBean();
      Start.creditCards[i] = tempCards[i];
    }

    for(int i = tempCards.length; i < Start.maxCCSaved; i++){
      Start.creditCards[i] = new CreditCardBean();
      Start.creditCards[i] = ccBean;
    }

  }// end if(Start.maxCCSaved > 0)

  alert = new CanvasAlert(
      display,
      new MyWalletMainMenu().prepareScreen(),
      true,
      2000,
      "Credit␣Card␣Saved!",
      "The␣credit␣card␣has␣been␣saved␣to␣the␣phone␣memory!",
      "OK",
      CustomAlertTypes.ALERT_INFO);

}// end updateCreditCards()


/**
 * Updates the credit cards available to the application
 * after a delete CC operation. The RMS is also updated!
 *
 */
public static void deleteCreditCardAndUpdateAllCreditCards(
    ChoiceGroup cgCreditCard, Display display){

  try{
```

```
  alert = new CanvasAlert(
      display,
      new MyWalletMainMenu().prepareScreen(),
      false,
      "My␣Wallet␣update␣in␣progress...",
      "Please␣wait␣until␣My␣Secure␣Wallet␣is␣updated!",
      "info",
      CustomAlertTypes.ALERT_INFO);
  display.setCurrent(alert);

  UpdateWalletBackground uwb = new UpdateWalletBackground(cgCreditCard
       , display);
  uwb.go();
  }catch(Exception e){

    System.out.println("---------␣Excpetion␣in␣hereeeee");
    e.printStackTrace();
  }

/*  boolean[] cgSelected = new boolean[cgCreditCard.size()];
  cgCreditCard.getSelectedFlags(cgSelected);
  int selectedIndex = cgCreditCard.getSelectedIndex() + 1;

  try{
    RMSOperations.deleteItems("CC" + selectedIndex + ":");

    // save the remaining CCs
    int j = 0;
    CreditCardBean[] updatedWalletCC = new CreditCardBean[Start.
        maxCCSaved - 1];

    for (int i = 0; i < Start.maxCCSaved; i++){
      if(i != (selectedIndex-1)){
        updatedWalletCC[j] = Start.creditCards[i];
        ++j;
      }
    }

    // delete all CCs
    for (int i = 1; i <SystemConstants.MAX_NO_CREDIT_CARDS + 1 ; i++)
      RMSOperations.deleteItems("CC" + i + ":");

    // save the remaining CCs back to RMS
    for (int i = 1; i <= updatedWalletCC.length ; i++){
      RMSOperations.writeEncryptedRecord("CC" + i + ":",
          updatedWalletCC[i-1].getBytes());
    }
```

```
  RMSOperations.deleteItems("CCN:");
  RMSOperations.writeRecord("CCN:", String.valueOf((Start.maxCCSaved
       - 1)));

  // update the Start.creditCards and Start.maxCCSaved
  // to make them available to the whole application
  // and improve appl performance by reducing the access to RMS
  if (updatedWalletCC.length > 0){

    Start.creditCards = null;
    Start.maxCCSaved -= 1;
    Start.creditCards = new CreditCardBean[Start.maxCCSaved];

    for (int i = 0; i < Start.maxCCSaved; i++){
      Start.creditCards[i] = new CreditCardBean();
      Start.creditCards[i] = updatedWalletCC[i];
    }
  }

  alert = new CanvasAlert(
      display,
      new MyWalletMainMenu().prepareScreen(),
      "Credit Card Removed!",
      "The credit card has been removed from your wallet!",
      "OK",
      CustomAlertTypes.ALERT_INFO);

}catch (IOException ioe){
  System.out.println("IOException when deleting the CC " + ioe.
      getMessage());
  ioe.printStackTrace();

}catch (CryptoException ce){
  System.out.println("CryptoException when deleting the CC " + ce.
      getMessage());
  ce.printStackTrace();

}catch (Exception e){
  System.out.println("Exception when deleting the CC " + e.getMessage
      ());
  e.printStackTrace();

} // end try - catch

*/
```

```
}// end deleteCreditCardAndUpdateAllCreditCards()


/**
 * Updates the credit cards available to the application
 * after an edit CC operation. The RMS is also updated!
 *
 */
public static int editCreditCardAndUpdateAllCreditCards(
    CreditCardBean ccBean, int ccRmsIndex, Display display)
throws IOException, CryptoException, Exception{

  int index = ccRmsIndex + 1;

  RMSOperations.deleteItems("CC" + index + ":");
  RMSOperations.writeEncryptedRecord("CC" + index + ":", ccBean.
      getBytes());

  // update the Start.creditCards data
  Start.creditCards[index - 1] = ccBean;

  alert = new CanvasAlert(
      display,
      new MyWalletMainMenu().prepareScreen(),
      true,
      2000,
      "Credit␣Card␣Updated!",
      "The␣credit␣card␣has␣been␣updated!",
      "OK",
      CustomAlertTypes.ALERT_INFO);

  return index;


}// end editCreditCardAndUpdateAllCreditCards()



/**
 * Removes permanently a selected ticket from RMS
 * if the ticket has been canceled succesfully on the server side
 *
 * @param cgTickets The choice group of all tickets
 * @param tickets All tickets before deletion
 * @param display The display to show the alerts
 */
public static void deleteTicketAndUpdateAllTickets(ChoiceGroup
```

```
    cgTickets, TicketBean[] tickets, Display display, Displayable
    previous, Displayable next) throws Exception{

boolean[] cgSelected = new boolean[cgTickets.size()];
cgTickets.getSelectedFlags(cgSelected);
int selectedIndex = cgTickets.getSelectedIndex();

// cancel only tickets that have been purchased using credit card
    payment method
if(Start.tickets[selectedIndex].getTKTPurchaseMethod().equals("Credit
    ␣Card")){

  try{
    RMSOperations.deleteItems("TT" + selectedIndex + ":");

    // save the remaining TKTs
    //int noOfTickets = RMSOperations.getAllItemsLike( "TT");
    int noOfTickets = Start.maxTTSaved - 1;

    if(noOfTickets > 0){
      int j = 0;
      TicketBean[] updatedTickets = new TicketBean[noOfTickets];

      for (int i = 0; i < tickets.length; i++){
        if(i != (selectedIndex)){
          updatedTickets[j] = tickets[i];
          ++j;
        }
      }

      // delete all TKTS
      for (int i = 0; i <SystemConstants.MAX_NO_TICKETS ; i++)
        RMSOperations.deleteItems("TT" + i + ":");

      // update the no of tickets saved in RMS
      RMSOperations.deleteItems("TTN:");
      RMSOperations.writeRecord("TTN:", String.valueOf(updatedTickets.
          length));


      if (updatedTickets.length > 0){
        // update Start.tickets and maxTTSaved
        Start.tickets = null;
        Start.tickets = updatedTickets;
        Start.maxTTSaved = Start.tickets.length;

        // save the remaining TKTs back to RMS
```

```
          for (int i = 0; i < updatedTickets.length ; i++){
            RMSOperations.writeByteItem("TT" + i + ":", updatedTickets[i
                ]);
          }
        }
      }

      alert = new CanvasAlert(
          display,
          new MyTicketsMainMenu().prepareScreen(),
          "Ticket␣Canceled!",
          "The␣ticket␣has␣been␣canceled!!",
          "OK",
          CustomAlertTypes.ALERT_INFO);

    }catch (IOException ioe){
      alert = new CanvasAlert(
          display,
          previous,
          "Cancel␣Ticket␣Error!",
          "Error␣while␣trying␣to␣remove␣the␣ticket␣from␣the␣phone␣memory
              !",
          "OK",
          CustomAlertTypes.ALERT_INFO);

      System.out.println("IOException␣when␣deleting␣the␣TKT␣" + ioe.
          getMessage());
      ioe.printStackTrace();

    }catch (Exception e){
      alert = new CanvasAlert(
          display,
          previous,
          "Cancel␣Ticket␣Error!",
          "Error␣while␣trying␣to␣remove␣the␣ticket␣from␣the␣phone␣memory
              !",
          "OK",
          CustomAlertTypes.ALERT_INFO);

      System.out.println("Exception␣when␣deleting␣the␣TKT␣" + e.
          getMessage());
      e.printStackTrace();

    } // end try - catch

  }else {
    try{
```

```
      alert = new CanvasAlert(
          display,
          previous,
          "Cannot␣cancel␣unpayed␣ticket!",
          "Onyt␣tickets␣purchased␣using␣a␣CREDIT␣CARD␣can␣be␣canceled!",
          "error",
          CustomAlertTypes.ALERT_WARNING);

    }catch (Exception e){
      alert = new CanvasAlert(
          display,
          previous,
          "Cancel␣Ticket␣Error!",
          "Error␣while␣trying␣to␣remove␣the␣ticket␣from␣the␣phone␣memory
              !",
          "OK",
          CustomAlertTypes.ALERT_INFO);

      System.out.println("Exception␣when␣trying␣to␣cancel␣a␣ticket␣not␣
          bought␣using␣a␣Credit␣Card␣" + e.getMessage());
      e.printStackTrace();

    } // end try - catch

  }// end if(Start.tickets[selectedIndex].getTKTPurchaseMethod().equals
      ("Credit Card")){}

}// end cancelTicket()

public static void deleteAndUpdateExpiredTickets() throws Exception{

  // delete all TKTS
  for (int i = 0; i <SystemConstants.MAX_NO_TICKETS ; i++)
    RMSOperations.deleteItems("TT" + i + ":");

  // update the no of tickets saved in RMS
  RMSOperations.deleteItems("TTN:");
  RMSOperations.writeRecord("TTN:", String.valueOf(Start.tickets.length
      ));

  // update Start.tickets and maxTTSaved
  Start.maxTTSaved = Start.tickets.length;

  // save the remaining TKTs back to RMS
  for (int i = 0; i < Start.tickets.length ; i++){
    RMSOperations.writeByteItem("TT" + i + ":", Start.tickets[i]);
  }
```

```
  }


}// end class

package networkoperations.authentication;

import gui.authentication.AuthenticationGUI;
import gui.customdialogwindows.CanvasAlert;
import gui.mainmenu.MenuScreen;

import java.io.ByteArrayOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Random;

import javax.microedition.io.Connector;
import javax.microedition.io.HttpConnection;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;

import networkoperations.BackgroundTask;
import networkoperations.NetworkCommunicationFacade;

import cryptography.AesKey;
import cryptography.Encryptor;
import org.bouncycastle.crypto.params.ParametersWithIV;

import rms.RMSOperations;

import constants.CustomAlertTypes;
import constants.Protocol_Step_Constants;
import constants.SQL_Return_Codes;


/*****
 * In the following class the authenticaton protocol would be realized.
 * The protocol implemented here is called Needham-Schroeder protocol.
 * The class is implemented due to the singleton pattern as the
     application might
 * be in the only one state of the protocol at once.
 *
 * The class is aditionally responsible for the encryption and decription
 * with the use of AES cipher
 *
 * @author Mihai Balan (s031288), Wojciech Dobrowolski
 *
 */
```

```
public class Authenticate extends BackgroundTask{

  private Random rand    = null; // random number generator for the
      protocol
  private String userName = "";
  private String passwd  = "";
  private String authUrl = "http://127.0.0.1:9080/Cinema_Controller/
      cinemaservice/servlets/controller/Cinema_Central_Controller_Servlet
      ?protocol=AT1";
  private String targetUrl = "http://127.0.0.1:9080/Cinema_Controller/
      cinemaservice/servlets/controller/Cinema_Central_Controller_Servlet
      ?protocol=AT2";

  private Displayable previous;
  private Displayable next;

  //Key object for the communication with the 128 AES cipher
  private ParametersWithIV aesKey = null;
  private AesKey aesK          = null;
  private Encryptor encryptorDecryptor;

  CanvasAlert alert;


  /**
   * The constructor initializes the class parameters
   * @param Display device
   * @param Displayable previous Screen to return to in case of comm
       error
   * @param Displayable next  Screen to go to after succesful comm
   * @param user User Name
   * @param pass Password
   * @param key Key for the cryptographic operations using DES
   * @param url Authentication server URL
   * @param target Tarhet service URL
   * @param prevDisp String for the decission, what to display afterwards
   * @throws Exception
   */
  public Authenticate(
      Display display,
      Displayable previous,
      Displayable next,
      String user,
      String pass,
      String key) throws Exception{

    super(display);
```

```
  rand = new Random();
  aesK = AesKey.getInstance();
  encryptorDecryptor = new Encryptor(key);

  this.userName = user;
  this.passwd   = pass;
  this.previous = previous;
  this.next     = next;

  prevScreen = previous;
  localProtocolStep = Protocol_Step_Constants.PRT_STEP_AUTHENTICATION_1
      ;

  System.out.println("The␣authentication␣protocol␣has␣been␣instantiated
      ␣...");
}


/**
 * The first stage of the protocol requires the random number,
 * which is generated as long.
 */
public String generateRandomToSend(){
  long myRandom = rand.nextLong();
  return "" + myRandom;
}

/**
 * In the following method the protocol used for the authentication
 * is being handled. The method is constructed in linear way.
 * It is correct approach as it is known that they occur one by one
 * in predefined order.
 *
 * In heavy security application, the protocol should have additionall
 * assertions about its state.
 */
public void runTask() throws Exception {
  //First we communicate with the authentication server
  //Message contains of the login, password,
  //random string and the address of the server to which we
      authenticate
  String random = generateRandomToSend();

  //Values necessary for generation of the session key
  String token = "";
  String salt = "";
```

```
//All the required fields are concatenated with predefined deliminter
//All they are strings.

String message1 = userName + ";" +
        passwd + ";" +
        random + ";" +
        targetUrl;

//message is encrypted also as a string.
byte[] encrypted1 = encryptorDecryptor.encryptString(message1);

//Message is sent to the authentication server with the use of
//standard procedure
byte[] res1 = communicate(authUrl,encrypted1);

//Decryprion of the received message
String resDec1 = encryptorDecryptor.decryptString(res1);

if(!resDec1.equals(String.valueOf(SQL_Return_Codes.
    Authentication_E_MONEY_PRT_USER_NOT_AUTHENTICATED))){

  //Received message is long: EA(RA,T,St,S,EB(S,T)) or "failure:
      failure"
  //The problem is that the last part: the message that is to be
  //relayed to the robot control service, must not be transformed in
      any way.
  //Moreover, there is no tokenizer method in J2ME. Tokenized
      elements are
  //stored in the String array
  String[] resTok1 = new String[6];
  String part;
  for (int i = 0; i < 5; i++){
    int scl = resDec1.indexOf(";");
    part = resDec1.substring(0,scl);
    resDec1 = resDec1.substring(scl+1);
    resTok1[i] = part;
    //Debug
    System.out.println("Tokenized␣part:␣" + part);
  }
  //Debug
  //System.out.println("The part to send: " + resDec1);
  //System.out.println("<------------------------------------------------------------------------>")
      ;
  //System.out.println("The message decrypted: " + decryptMessage
      ("87654321", resDec1.getBytes()));

  //MEssage is once again decrypted to obtain byte[] as the result
```

```
byte[] bytesFromAuth = encryptorDecryptor.decrypt(res1);
//The message to be relayed is chopped off the original array, so
    that we not
//corrupt the padding
byte[] forBob = new byte[resDec1.length()];
System.arraycopy(bytesFromAuth,(bytesFromAuth.length - forBob.
    length),forBob,0,forBob.length);
//Debug
System.out.println("The␣byte␣array␣that␣will␣be␣sent␣fo␣Bob:␣" +
    new String(forBob));

//Checking the value of the random
if (!random.equals(resTok1[1])){
  alert = new CanvasAlert(
      display,
      new AuthenticationGUI().prepareScreen(),
      "Authentication␣Failure!",
      "Please␣check␣your␣credentials␣and␣try␣again!",
      "warn",
      CustomAlertTypes.ALERT_WARNING);
}
else if (!targetUrl.equals(resTok1[2])){
  alert = new CanvasAlert(
      display,
      new AuthenticationGUI().prepareScreen(),
      "Authentication␣Failure!",
      "Please␣check␣your␣credentials␣and␣try␣again!",
      "warn",
      CustomAlertTypes.ALERT_WARNING);
}
else {

  /**
   * At this stage it is possible to build a local session key
   */

  token = resTok1[3];
  salt = resTok1[4];

  //AES key is created with the use of the information
  //received from the authentication server
  aesKey = encryptorDecryptor.createKey(salt, token);

  //We store globally the key
  aesK.setKey(aesKey);

  //Debug
```

```
//System.out.println("AES key has been created ..." + aesKey.
    toString());

//Communication with the robot control service starts at this
    point (First stage)
String prtInd = "PRT1:";
//Byte array with the protocol indication is required
byte[] prtIndB = prtInd.getBytes();
//both parts, the protocol indication and the one for the robot
    control are joined
byte[] prt1 = new byte[prtInd.length()+forBob.length];
System.arraycopy(prtIndB,0,prt1,0,prtIndB.length);
System.arraycopy(forBob,0,prt1,prtIndB.length,forBob.length);

//results of the previous steps are communicated to the servlet
byte[] res2 = communicate(targetUrl,prt1);

//We know that at this point the message should be decrypted with
     the use of AES
byte[] res22 = encryptorDecryptor.decryptWithAES(aesKey, res2);
//Debug
//System.out.println("The random number decrypted with AES: " +
    new String(res22));
String number = new String(res22);

//number = number.concat("1");
//System.out.println("Returned number: " + number);
//At this stage we are sending the last element of the
    authentication process
byte[] number1 = number.getBytes();

//The random number is passed to the secure servlet
byte[] lastAuthStep = encryptorDecryptor.encryptWithAES(aesKey,
    number1);
//Debug
//System.out.println("LAst response encrypted. Size: " +
    lastAuthStep.length);

//Last authentication step message is formed as the previous one
String prtInd2 = "PRT2:";
byte[] prtIndB2 = prtInd2.getBytes();
byte[] prt2 = new byte[prtInd2.length()+lastAuthStep.length];
System.arraycopy(prtIndB2,0,prt2,0,prtIndB2.length);
System.arraycopy(lastAuthStep,0,prt2,prtIndB2.length,lastAuthStep
    .length);
System.out.println("Last␣protocol␣step␣formed.");
//And sent to the servlet
```

```
byte[] res3 = communicate(targetUrl,prt2);
System.out.println("Last␣protocol␣step␣sent␣...,␣response:␣" +
    res3);
String finalResp = new String(res3);

// ******************************************************
// ******************************************************
// save the e-money into rms if authentication successful
// ******************************************************
RMSOperations.deleteItems("EMN:");
RMSOperations.writeRecord("EMN:",resTok1[0]);
// ******************************************************
// ******************************************************

//Alert a = new Alert(finalResp);
//a.setTimeout(Alert.FOREVER);
//display.setCurrent(a);

//Alert informs the user about the operation result
alert = new CanvasAlert(
    display,
    next,
    "User␣" + finalResp + "!",
    "You␣are␣authorized␣to␣use␣the␣system!",
    "info",
    CustomAlertTypes.ALERT_INFO);
/*
Runtime runtime = Runtime.getRuntime();
long t = runtime.freeMemory();
System.out.println("**********************Memory before clean
    auth succ:" + t);
random = null;
token = null;
salt = null;
message1 = null;
encrypted1 = null;
res1 = null;
resDec1 = null;

resTok1 = null;
part = null;
bytesFromAuth = null;
forBob = null;
prtInd = null;
prtIndB = null;
prt1 = null;
res2 = null;
```

```
      res22 = null;
      number = null;
      number1 = null;
      lastAuthStep = null;
      prtInd2 = null;
      prtIndB2 = null;
      prt2 = null;
      res3 = null;
      finalResp = null;
      clean();

      long t1 = runtime.freeMemory();
      System.out.println("************************Memory after clean
          auth succ:" + t1);
    */
    }
  } else if (resDec1.equals(String.valueOf(SQL_Return_Codes.
      Authentication_E_MONEY_PRT_USER_NOT_AUTHENTICATED))) {
    alert = new CanvasAlert(
        display,
        new AuthenticationGUI().prepareScreen(),
        "Authentication␣Failure!",
        "Please␣check␣your␣credentials␣and␣try␣again!",
        "warn",
        CustomAlertTypes.ALERT_WARNING);

    /*
     Runtime runtime = Runtime.getRuntime();
    long t = runtime.freeMemory();
    System.out.println("************************Memory before clean
        auth error:" + t);
    random = null;
    token = null;
    salt = null;
    message1 = null;
    encrypted1 = null;
    res1 = null;
    resDec1 = null;
    clean();


    long t1 = runtime.freeMemory();
    System.out.println("************************Memory after clean auth
         error:" + t1);
    */

  } // end if(!resDec1.equals("failure:failure"))
```

```
}

/**
 * The following method is reused from the write message class.
 * It is sending the message to the tarher servlet.
 *
 * @param url Url of the service with which we communicate,
 *       either the authentication server or targeted service
 * @param msg Message to be communicated
 * @return
 * @throws Exception
 */
public byte[] communicate(String url, byte[] msg) throws Exception {
  HttpConnection c = null;
  InputStream is = null;
  OutputStream os = null;
  //ObjectOutputStream ObOs = null;
  // the binary message from the servlet
  byte[] data = null;
  // the dcrypted message from the servlet
  //String decMsg = "";

  try {
    //We are communicating with the authentication server
    //or with the target server, then the url must be passed as
        aparameter
    c = (HttpConnection)Connector.open(url);

    //Setting up all the connection properties
    c.setRequestMethod(HttpConnection.POST);
    c.setRequestProperty("User-Agent","Profile/MIDP-2.0␣Configuration/
        CLDC-1.0");
    c.setRequestProperty("Content-Language", "en-US");
    c.setRequestProperty( "Connection", "close" );
    c.setRequestProperty("Content-Length", Integer.toString(msg.length
         ));


    //Sending the message
    os = c.openOutputStream();

    os.write(msg);
    os.close();

    //int rc = c.getResponseCode();
```

```
    if( c.getResponseCode() == HttpConnection.HTTP_OK ){
      int len = (int) c.getLength();
      is = c.openInputStream();

      if( len != -1 ){
        // If length is available read the data into an array
        int total = 0;
        data = new byte[len];
        while( total < len ){
          total += is.read( data, total, len - total );
        }

      } else {
        ByteArrayOutputStream tmp = new ByteArrayOutputStream();
        int ch;
        while( ( ch = is.read() )!= -1 ){
          tmp.write( ch );
        }
        data = tmp.toByteArray();

      }

    } else {
      // in case of errors
      //decMsg = "Secure Servlet response CORRUPTED!";

    }
  } finally {
    // Takes care the connection and the stream are not oppened
    // any more after a succesfull communication or an error.
    if (is != null) {
      try { is.close(); } catch (Throwable e) { }
      is = null;
    }
    if (os != null) {
      try { os.close(); } catch (Throwable e) { }
      os = null;
    }
    if (c != null) {
      try { c.close(); } catch (Throwable e) { }
      c = null;
    }

  }
  return data;

}
```

```
/*  private void clean(){
    rand     = null;
    userName = null;
    passwd   = null;
    authUrl  = null;
    targetUrl = null;

    previous= null;
    next= null;

    //Key object for the communication with the 128 AES cipher
    aesKey = null;
    aesK  = null;
    encryptorDecryptor= null;
    alert = null;
    prevScreen = null;
    System.gc();

  }*/

}

package networkoperations;

import gui.authentication.AuthenticationGUI;
import gui.customdialogwindows.CanvasAlert;

import java.util.*;
import java.io.*;
import javax.microedition.lcdui.*;

import constants.CustomAlertTypes;
import constants.Protocol_Step_Constants;

/**
 * This class is used to perform network operations on
 * a separate thread. Each class that performs network operations,
 * (e.g. Authenticate)extends this class.
 * An animated progress gauge is displayed while the
 * network operations are performed.
 *
 * @author Mihai Balan, Wojciech Dobrowolski
 *
 */
public abstract class BackgroundTask extends TimerTask {

  protected Display   display;
```

```
protected Displayable nextScreen;
protected Displayable prevScreen;
protected String    title;

protected boolean needAlert = false;
private String alertTitle = "";
private String alertMessage = "";

private Thread workerThread;
private boolean isWrkStopped;

protected String localProtocolStep;

/**
 * Constructor - initialize the display and
 * creates the worker thread
 *
 * @param display The current display
 */
public BackgroundTask (Display display) {
  this.display = display;
  workerThread = new Thread(this);
}

/**
 * Starts the worker thread for performing the network
 * operations in the background
 */
public void go () {
  // set the flag to worker alive
  isWrkStopped = false;
  // start the worker thread
  System.out.println("-----Background␣Task␣--␣The␣worker␣thread␣for␣
      performing␣network" +
  "␣operations␣in␣backgruond␣has␣been␣started!");
  workerThread.start();
}

/**
 * Stop the worker thread by setting the pririty to MIN
 */
public void stop () {
  // set the flag to worker stopped
  isWrkStopped = true;
  System.out.println("-----Background␣Task␣--␣The␣worker␣thread␣
      priority␣set␣to␣MINIMUM");
  // stops the worker thread
```

```
  workerThread.setPriority(Thread.MIN_PRIORITY);
}

/**
 * Create the animated gauge and
 * call the template method runTask()
 * that is implemented by the derived classes.
 * In case of network communication exception
 * or other exception, the application catches
 * the exceptions and display an alert.
 */
public void run() {

  ProgressGauge pg = null;

  try {
    // Construct and start the gauge
    // The gauge is started in the init() method of the ProgressGauge
    System.out.println("-----Background␣Task␣--␣Animated␣gauge␣created"
        );
    pg = new ProgressGauge(this, title, display, prevScreen);
    System.out.println("-----1111111111111");
    System.out.println("--------" + localProtocolStep);

    if(localProtocolStep.equals(Protocol_Step_Constants.
        PRT_STEP_REJECT_PAYMENT)){
      System.out.println("-----2222222222222222");
      pg.setInfoText("Updating␣the␣application␣...");
    }
    System.out.println("-----333333333333");

    // start the task implemented by the derived classes
    runTask ();
    System.out.println("-----Background␣Task␣--␣The␣runTask()␣method␣
        implemented␣" +
    "by␣the␣classes␣that␣extend␣Background␣Task␣is␣called");
  } catch (IOException ioe) {
    // an alert need to be displayed
    needAlert   = true;
    alertTitle  = "Communication␣Error!";
    alertMessage = "Please␣check␣your␣network␣or␣server␣setup!";

    nextScreen = prevScreen;
    System.out.println("-----Background␣Task␣--␣IO␣ERROR");
    System.out.println("-----Background␣Task␣--␣Background␣task␣IO␣
        Error");
    ioe.printStackTrace();
```

```
    try{
      CanvasAlert alert = new CanvasAlert(
          display,
          new AuthenticationGUI().prepareScreen(),
          "Communication␣Error!",
          "Service␣unavailable!",
          "error",
          CustomAlertTypes.ALERT_ERROR,
          false);
      display.setCurrent(alert);

    }catch(Exception ee){
      ee.printStackTrace();
    }

  } catch (Exception e) {

    // an alert need to be displayed
    needAlert   = true;
    alertTitle  = "Unknown␣Error!";
    alertMessage = "Please␣contact␣customer␣support!";

    // return to the previous screen in case of error
    nextScreen = prevScreen;
    System.out.println("-----Background␣Task␣--␣Background␣task␣Error")
        ;
    System.out.println("-----Background␣Task␣--␣ERROR");
    e.printStackTrace();
    CanvasAlert alert = new CanvasAlert(
       display,
       prevScreen,
       "Error!",
       e.getMessage(),
       "error",
       CustomAlertTypes.ALERT_ERROR,
       false);
    display.setCurrent(alert);

  } finally {
    // Since pg could callback and reset "stopped" when its
    // Cancel key is pressed, we'd better check.
    System.out.println("-----Background␣Task␣--␣before␣if(!STOPPED)");
    if (!isWrkStopped) {
      // in case an alert was displayed
      if ( needAlert ){
        System.out.println("-----Background␣Task␣--␣Progress␣Gauge␣
```

```
            stoped");
          pg.stop();

          // create the alert but do not display it
          // let the progress gauge to display it after it stoped
          CanvasAlert alert = new CanvasAlert(
              display,
              prevScreen,
              alertTitle,
              alertMessage,
              "error",
              CustomAlertTypes.ALERT_ERROR,
              false);

        //  pg.setNextScreen(alert, nextScreen);

        } else {

          System.out.println("-----Background␣Task␣--␣Progress␣Gauge␣
              stoped");
          pg.stop();
        //  pg.setNextScreen(nextScreen);

        }
        System.out.println("-----Background␣Task␣--␣Progress␣Gauge␣stoped
            ");
        // notify the progress gauge to quit
        pg.stop();
      }
    }
  }

  /**
   *  Template method that need to be implemented in the derived classes.
   *  The actual task is implemented in this method by the derived class
   *
   *  @throws Exception
   */
  public abstract void runTask () throws Exception;
}

package networkoperations;

import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;


import networkoperations.authentication.Authenticate;
```

```
import model.beans.requestbeans.Cinema_Hall_Conf_Req_Bean;
import model.beans.requestbeans.Find_Movies_Req_Bean;
import model.beans.requestbeans.Movie_Details_Req_Bean;
import model.beans.requestbeans.Purchase_Tickets_Req_Bean;
import model.beans.requestbeans.Rate_Movie_Req_Bean;
import model.beans.responsebeans.Cinema_Hall_Conf_Resp_Bean;
import model.beans.responsebeans.Find_Movies_Resp_Bean;
import model.beans.responsebeans.Movie_Details_Resp_Bean;
import model.beans.responsebeans.Purchase_Tickets_Resp_Bean;
import model.beans.responsebeans.Response_Msg_Bean;

/**
 * Performs the network communication between
 * the mobile client and the server side
 *
 * @author Mihai Balan, s031288
 *
 */
public class NetworkCommunicationFacade {

  private static String url = "http://127.0.0.1:9080/Cinema_Controller/
      cinemaservice/servlets/controller/Cinema_Central_Controller_Servlet
      ?protocol=";


  /**
   * Send the authentication request bean over the network and get the
       response bean from the server
   *
   */
  public static void authenticate(
     Display    display,
     Displayable previous,
     Displayable next,
     String     userName,
     String     password,
     String     key) throws Exception{

    Authenticate sm = new Authenticate(display,previous, next, userName,
        password, key);
    System.out.println("-----␣Authentication␣--␣Authentication␣procedure␣
        started␣on␣a␣background␣thread!");
    sm.go();
  }

  /**
```

```
 * Send the request bean over the network and get the response bean
     from the server
 *
 * @param findMovReqBean The request bean containing data used to
     search movies for
 * @return The found movies matching the given criteria
 */
public static Find_Movies_Resp_Bean findMovies(Display display,
    Find_Movies_Req_Bean findMovReqBean) throws Exception{
  //SendMessage sm = new SendMessage(display, "".getBytes(), "", getURL
      () + "MOV", "WriteMessage");
  //sm.go();
  return null;
}


/**
 * Send the request bean over the network and get the response bean
     from the server
 *
 * @param cinemaHallConfReqBean The request bean containing data used
     to get the cinema hall configuration
 * @return Thecinema hall configuration
 */
public Cinema_Hall_Conf_Resp_Bean getCinemaHallConf(
    Cinema_Hall_Conf_Req_Bean cinemaHallConfReqBean){
  return null;
}


/**
 * Send the request bean over the network and get the response bean
     from the server
 *
 * @param movDetailsReqBean The request bean containing data used to
     get the movie details
 * @return The movie details coresponding to the requested movie
 */
public Movie_Details_Resp_Bean movieDetails(Movie_Details_Req_Bean
    movDetailsReqBean){
  return null;
}

/**
 * Send the request bean over the network and get the response bean
     from the server
 *
```

```
   * @param rateMovieReqBean The request bean containing data used to
       rate the movie
   * @return The result from rating the movie
   */
  public Response_Msg_Bean rateMovie(Rate_Movie_Req_Bean rateMovieReqBean
      ){

    return null;
  }

  /**
   * Send the request bean over the network and get the response bean
       from the server
   *
   * @param reqBean The request bean containing the data for paying for
       the reserved tickets
   * @return Purchase_Tickets_Resp_Bean
   */
  public Purchase_Tickets_Resp_Bean purchaseTickets(
      Purchase_Tickets_Req_Bean reqBean){
    return null;
  }

  private static String getURL(){
    return url;
  }

}

package networkoperations;

import java.util.Date;

import gui.customdialogwindows.CanvasAlert;
import gui.customdialogwindows.DialogWindow;
import gui.mainmenu.MenuScreen;
import gui.moviedetails.ViewMovieDetailsGUI;
import gui.purchasetickets.step1moviesearch.MovieSearchHelper;
import gui.purchasetickets.step2selectshow.SelectShowGUI;
import gui.purchasetickets.step3selectseats.SelectSeatsGUI;
import gui.settings.SettingsGUI;

import javax.microedition.lcdui.ChoiceGroup;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;

import rms.RMSOperations;
```

```
import model.beans.otherbeans.TicketBean;
import model.beans.requestbeans.Cancel_Tickets_Req_Bean;
import model.beans.requestbeans.Change_Password_Req_Bean;
import model.beans.requestbeans.Cinema_Hall_Conf_Req_Bean;
import model.beans.requestbeans.Find_Movies_Req_Bean;
import model.beans.requestbeans.Movie_Details_Req_Bean;
import model.beans.requestbeans.Purchase_Tickets_Req_Bean;
import model.beans.requestbeans.Rate_Movie_Req_Bean;
import model.beans.requestbeans.Reject_Payment_Req_Bean;
import model.beans.responsebeans.Cinema_Hall_Conf_Resp_Bean;
import model.beans.responsebeans.Find_Movies_Resp_Bean;
import model.beans.responsebeans.Movie_Details_Resp_Bean;
import model.beans.responsebeans.Purchase_Tickets_Resp_Bean;
import model.beans.responsebeans.Response_Msg_Bean;
import model.beans.responsebeans.Select_Deselect_Seats_Resp_Bean;
import model.update.UpdateModel;


import constants.*;
import cryptography.Encryptor;

/**
 * Analysis the response bean received from the server side
 * after the network communication and function of the response
 * code displays an alert or performs different operations
 *
 * @author Mihai Balan, s031288
 *
 */
public class NetworkResposeFacade{

  /**
   * Check the response code from the findMoviesRespBean
   * and displays the appropiate UI screen function of
   * the return value
   *
   * @param display          The application display
   * @param next             The next UI to display
   * @param previous         The previous UI where it was called
   * @param findMoviesRespBean The Dinf Movies Response bean
   */
  public static void displaySearchMoviesResponse(
      Display             display,
      Displayable         previous,
      Find_Movies_Req_Bean findMovReqBean,
      Find_Movies_Resp_Bean findMoviesRespBean) throws Exception{
```

```
CanvasAlert help;

switch(findMoviesRespBean.getResponseCode()){

case SQL_Return_Codes.MOVIE_LOCATION_SERVICE_ERROR:
  help = new CanvasAlert(
      display,
      previous,
      "Movie␣Search␣Error!",
      "An␣error␣was␣encounted␣while␣searching␣for␣movies!",
      "error",
      CustomAlertTypes.ALERT_ERROR);
  break;

case SQL_Return_Codes.MOVIE_LOCATION_SERVICE_NO_DATA:
  help = new CanvasAlert(
      display,
      previous,
      "No␣cinemas␣found!",
      "No␣cinemas␣could␣be␣found␣according␣to␣the␣given␣criteria!",
      "warn",
      CustomAlertTypes.ALERT_WARNING);
  break;

case SQL_Return_Codes.FIND_MOVIES_CRIT_1_PRT_MOVIES_NOT_FOUND:
case SQL_Return_Codes.FIND_MOVIES_CRIT_2_PRT_MOVIES_NOT_FOUND:
case SQL_Return_Codes.FIND_MOVIES_CRIT_3_PRT_MOVIES_NOT_FOUND:
case SQL_Return_Codes.FIND_MOVIES_CRIT_4_PRT_MOVIES_NOT_FOUND:
case SQL_Return_Codes.FIND_MOVIES_CRIT_5_PRT_MOVIES_NOT_FOUND:
  help = new CanvasAlert(
      display,
      previous,
      "No␣movies␣found!",
      "No␣movies␣could␣be␣found␣according␣to␣the␣given␣criteria!",
      "warn",
      CustomAlertTypes.ALERT_WARNING);
  break;

case SQL_Return_Codes.FIND_MOVIES_CRIT_1_PRT_MOVIES_FOUND:
case SQL_Return_Codes.FIND_MOVIES_CRIT_2_PRT_MOVIES_FOUND:
case SQL_Return_Codes.FIND_MOVIES_CRIT_3_PRT_MOVIES_FOUND:
case SQL_Return_Codes.FIND_MOVIES_CRIT_4_PRT_MOVIES_FOUND:
case SQL_Return_Codes.FIND_MOVIES_CRIT_5_PRT_MOVIES_FOUND:
  System.out.println("--------␣movies␣found!");
  display.setCurrent( new SelectShowGUI(findMovReqBean,
      findMoviesRespBean).prepareScreen());
  break;
```

```
  default:
    help = new CanvasAlert(
        display,
        previous,
        "Search␣Movies␣Error␣Default!",
        "Search␣Movies␣Error␣Default␣in␣switch-case!",
        "error",
        CustomAlertTypes.ALERT_ERROR);
  break;

  }// end switch()
  Runtime runtime = Runtime.getRuntime();
  long t = runtime.freeMemory();
  System.out.println("************************Memery␣before␣nnet␣resp␣
      facade␣find␣mov:" + t);
  help = null;
  findMovReqBean = null;
  findMoviesRespBean = null;
  previous = null;


  System.gc();
  long t1 = runtime.freeMemory();
  System.out.println("************************Memery␣after␣nnet␣resp␣
      facade␣find␣mov:" + t1);


}// end displaySearchMoviesResponse



/**
 * Check the response code from the cinemaHallConfRespBean
 * and displays the appropiate UI screen function of
 * the return value
 *
 * @param display             The application display
 * @param next                The next UI to display
 * @param previous            The previous UI where it was called
 * @param cinemaHallConfReqBean The Cinema Hall Conf Req Bean
 * @param cinemaHallConfRespBean The Cinema Hall Conf Resp Bean
 */
public static void displayCinemaHallConfResponse(
    Display                 display,
    Displayable             previous,
    String[]                showInfo,
```

```
  Cinema_Hall_Conf_Req_Bean cinemaHallConfReqBean,
  Cinema_Hall_Conf_Resp_Bean cinemaHallConfRespBean) throws Exception
      {

CanvasAlert help;
TicketBean genericTicket = new TicketBean();

switch(cinemaHallConfRespBean.getResponseCode()){

case SQL_Return_Codes.DISP_CINEMA_HALL_CONF_PRT_SHOW_NOT_FOUND:
  help = new CanvasAlert(
      display,
      previous,
      "Show␣Not␣Found!",
      "The␣cinema␣theater␣configuration␣could␣not␣be␣retrieved!",
      "warn",
      CustomAlertTypes.ALERT_WARNING);
  break;

case SQL_Return_Codes.DISP_CINEMA_HALL_CONF_PRT_SHOW_FOUND:
  // set up the info for the generic ticket that is to be used later
       on to
  // create all other tickets
  genericTicket.setTKTCinema(showInfo[1]);
  genericTicket.setTKTCinemaAddress(showInfo[4]);
  genericTicket.setTKTMovie(showInfo[0]);
  genericTicket.setTKTReservationDate(MovieSearchHelper.parseDate(new
       Date().toString()));
  genericTicket.setTKTShowDate(showInfo[2]);
  genericTicket.setTKTShowHour(showInfo[3]);

  display.setCurrent(
      new SelectSeatsGUI(
          display,
          previous,
          genericTicket,
          cinemaHallConfReqBean,
          cinemaHallConfRespBean));
  break;

default:
  help = new CanvasAlert(
      display,
      previous,
      "Select␣Seats␣Error␣Default!",
      "Select␣Seats␣Error␣Default␣in␣switch-case!",
      "error",
```

```
        CustomAlertTypes.ALERT_ERROR);
    break;

    }// end switch()

  }// end displayCinemaHallConfResponse()


  public static void displaySelectDeselectSeatsResponse(
      Display                     display,
      Displayable                 previous,
      Displayable                 next,
      Select_Deselect_Seats_Resp_Bean selDeselSeatsRespBean){

    CanvasAlert help;

    switch(selDeselSeatsRespBean.getResponseCode()){

    case SQL_Return_Codes.SELECT_DESELECT_SEATS_PRT_SEATS_SELECTED_ERROR:
      help = new CanvasAlert(
          display,
          previous,
          "Seat␣reservation␣error!",
          "Error␣reserving␣the␣selected␣seats!",
          "warn",
          CustomAlertTypes.ALERT_WARNING);
      break;

    case SQL_Return_Codes.SELECT_DESELECT_SEATS_PRT_SEATS_SELECTED_OK:
      // TO DO - to change the previous screen to the next screen after
          the seats have been saved in DB
      help = new CanvasAlert(
          display,
          next,
          true,
          2000,
          "Seat(s)␣Reserved!",
          "Your␣seat␣selection␣has␣been␣saved!",
          "info",
          CustomAlertTypes.ALERT_INFO);
      break;

    case SQL_Return_Codes.
        SELECT_DESELECT_SEATS_PRT_SEATS_DESELECTED_ERROR:
      help = new CanvasAlert(
          display,
          previous,
```

```
        "Error␣while␣canceling␣the␣reservation!",
        "Error␣while␣canceling␣the␣previous␣reserved␣seats!",
        "warn",
        CustomAlertTypes.ALERT_WARNING);
    break;

  case SQL_Return_Codes.SELECT_DESELECT_SEATS_PRT_SEATS_DESELECTED_OK:
    // TO DO - check if you have to change previous to another screen
    help = new CanvasAlert(
        display,
        previous,
        "Reserved␣Seat(s)␣Canceled!",
        "Your␣previous␣selected␣seat(s)␣has␣been␣canceled!",
        "info",
        CustomAlertTypes.ALERT_INFO);
    break;

  default:
    help = new CanvasAlert(
        display,
        previous,
        "Reserve␣Seats␣Error␣Default!",
        "Reserve␣Seats␣Error␣Default␣in␣switch-case!",
        "error",
        CustomAlertTypes.ALERT_ERROR);
  break;
  }

}// end displaySelectDeselectSeatsResponse()



/**
 * Check the response code from the cinemaHallConfRespBean
 * and displays the appropiate UI screen function of
 * the return value
 *
 * @param display             The application display
 * @param next                The next UI to display
 * @param previous            The previous UI where it was called
 * @param movDetailsReqBean   The Movie_Details_Req_Bean
 * @param movDetailsRespBean  The Movie_Details_Resp_Bean
 */
public static void displayMovieDetails(
    Display display,
    Displayable previous,
    Movie_Details_Req_Bean movDetailsReqBean,
```

```
    Movie_Details_Resp_Bean movDetailsRespBean) throws Exception{

  CanvasAlert help;

  switch(movDetailsRespBean.getResponseCode()){

  case SQL_Return_Codes.MOVIE_DETAILS_PRT_ERROR:
    help = new CanvasAlert(
        display,
        previous,
        "Movie␣Found!",
        "The␣movie␣details␣could␣not␣be␣retrieved!",
        "warn",
        CustomAlertTypes.ALERT_WARNING);
    break;

  case SQL_Return_Codes.MOVIE_DETAILS_PRT_OK:
    display.setCurrent(new ViewMovieDetailsGUI(display, previous,
         movDetailsRespBean, movDetailsReqBean.getShowLocationID()));
    break;

  default:
    help = new CanvasAlert(
        display,
        previous,
        "Movie␣Details␣Error␣Default!",
        "Movie␣Details␣Error␣Default␣in␣switch-case!",
        "error",
        CustomAlertTypes.ALERT_ERROR);
  break;

  }// end switch()

}// end displayCinemaHallConfResponse()


/**
 * Check the response code from the cinemaHallConfRespBean
 * and displays the appropiate UI screen function of
 * the return value
 *
 * @param display             The application display
 * @param next                The next UI to display
 * @param previous            The previous UI where it was called
 * @param rateMovieReqBean    The Rate_Movie_Req_Bean
 * @param rateMovieRespBean   The Response_Msg_Bean
 */
```

```
public static void displayRateMovieResponse(
    Display display,
    Displayable previous,
    Displayable next,
    Rate_Movie_Req_Bean rateMovieReqBean,
    Response_Msg_Bean  rateMovieRespBean) throws Exception{

  CanvasAlert help;

  switch(rateMovieRespBean.getResponseCode()){

  case SQL_Return_Codes.RATE_MOVIE_PRT_ERROR:
    help = new CanvasAlert(
        display,
        previous,
        "Error␣while␣during␣voting!",
        "An␣error␣occured␣during␣voting!",
        "warn",
        CustomAlertTypes.ALERT_WARNING);
    break;

  case SQL_Return_Codes.RATE_MOVIE_PRT_USER_NOT_AUTHENTICATED:
    help = new CanvasAlert(
        display,
        previous,
        "User␣Not␣Authenticated!",
        "User␣is␣not␣authenticated!",
        "warn",
        CustomAlertTypes.ALERT_WARNING);
    break;

  case SQL_Return_Codes.RATE_MOVIE_PRT_OK:
    help = new CanvasAlert(
        display,
        next,
        "Vote␣recorded!",
        "Your␣vote␣has␣been␣recorded!",
        "info",
        CustomAlertTypes.ALERT_INFO);
    break;

  default:
    help = new CanvasAlert(
        display,
        previous,
        "Rate␣Movie␣Error␣Default!",
        "Rate␣Movie␣Error␣Default␣in␣switch-case!",
```

```
        "error",
        CustomAlertTypes.ALERT_ERROR);
   break;

   }// end switch()

 }// end displayCinemaHallConfResponse()


 /**
  * Check the response code from the PurchaseTicketsResponseBean
  * and displays the appropiate UI screen function of
  * the return value
  *
  * @param display The application display
  * @param next    The next UI to display
  * @param previous The previous UI where it was called
  * @param reqBean The Purchase_Tickets_Req_Bean
  * @param respBean The Purchase_Tickets_Resp_Bean
  */
 public static void displayPurchaseTicketsResponse(
     Display display,
     Displayable previous,
     Displayable next,
     Purchase_Tickets_Req_Bean reqBean,
     Purchase_Tickets_Resp_Bean respBean) throws Exception{

   CanvasAlert help;

   switch(respBean.getResponseCode()){

   case SQL_Return_Codes.PURCHASE_TICKETS_PRT_ERROR:
     help = new CanvasAlert(
         display,
         previous,
         "Error␣while␣trying␣to␣pay␣for␣the␣tickets!",
         "An␣error␣occured␣during␣payment!␣No␣maney␣has␣been␣withdraw␣
             from␣the␣credit␣card!",
         "warn",
         CustomAlertTypes.ALERT_WARNING);
     break;

   case SQL_Return_Codes.PURCHASE_TICKETS_PRT_USER_NOT_AUTNENTICATED:
     help = new CanvasAlert(
         display,
         previous,
         "User␣Not␣Authenticated!",
```

```
        "User␣is␣not␣authenticated!",
        "warn",
        CustomAlertTypes.ALERT_WARNING);
    break;

  case SQL_Return_Codes.PURCHASE_TICKETS_PRT_INVALID_CREDIT_CARD:
    help = new CanvasAlert(
        display,
        previous,
        "Invalid␣Credit␣Card!",
        "The␣provided␣credit␣card␣is␣invalid.␣Please␣check␣the␣data␣
            again!!",
        "warn",
        CustomAlertTypes.ALERT_WARNING);
    break;

  case SQL_Return_Codes.PURCHASE_TICKETS_PRT_OK:
    help = new CanvasAlert(
        display,
        next,
        "Payment␣Done!",
        "Ticket␣payment␣successful!",
        "info",
        CustomAlertTypes.ALERT_INFO);
    break;

  default:
    help = new CanvasAlert(
        display,
        previous,
        "Purchase␣Tickets␣Error␣Default!",
        "Purchase␣Tickets␣Error␣Default␣in␣switch-case!",
        "error",
        CustomAlertTypes.ALERT_ERROR);
  break;

  }// end switch()

}// end displayCinemaHallConfResponse()


/**
 * Check the response code from the Cancel_Tickets_Resp_Bean
 * and displays the appropiate UI screen function of
 * the return value
 *
 * @param display The application display
```

```
 * @param next    The next UI to display
 * @param previous The previous UI where it was called
 * @param reqBean The Cancel_Tickets_Req_Bean
 * @param respBean The Cancel_Tickets_Resp_Bean
 */
public static void displayCancelTicketsResponse(
    Display               display,
    Displayable           previous,
    Displayable           next,
    ChoiceGroup           cgTickets,
    TicketBean[]          tickets,
    Cancel_Tickets_Req_Bean reqBean,
    Response_Msg_Bean     respBean) throws Exception{

  CanvasAlert help;

  switch(respBean.getResponseCode()){

  case SQL_Return_Codes.CANCEL_TICKETS_PRT_ERROR:
    help = new CanvasAlert(
        display,
        previous,
        "Ticket␣cannot␣be␣canceled!",
        "An␣error␣occured␣while␣trying␣to␣cancel␣the␣selected␣ticket!",
        "warn",
        CustomAlertTypes.ALERT_WARNING);
    break;

  case SQL_Return_Codes.CANCEL_TICKETS_PRT_USER_NOT_AUTHENTICATED:
    help = new CanvasAlert(
        display,
        previous,
        "User␣Not␣Authenticated!",
        "User␣is␣not␣authenticated!",
        "warn",
        CustomAlertTypes.ALERT_WARNING);
    break;

  case SQL_Return_Codes.CANCEL_TICKETS_PRT_OK:
    UpdateModel.deleteTicketAndUpdateAllTickets(cgTickets, tickets,
        display, previous, next);
    break;

  default:
    help = new CanvasAlert(
        display,
        previous,
```

```
        "Ticket␣Cancel␣Error␣Default!",
        "Ticket␣Cancel␣Error␣Default␣in␣switch-case!",
        "error",
        CustomAlertTypes.ALERT_ERROR);
  break;

  }// end switch()

}// end displayCancelTicketsResponse()


/**
 * Check the response code from the Cancel_Tickets_Resp_Bean
 * and displays the appropiate UI screen function of
 * the return value
 *
 * @param display The application display
 * @param next    The next UI to display
 * @param previous The previous UI where it was called
 * @param reqBean The Cancel_Tickets_Req_Bean
 * @param respBean The Cancel_Tickets_Resp_Bean
 */
public static void displayRejectPaymentResponse(
    Display               display,
    Displayable           previous,
    String                nextScreenName,
    Reject_Payment_Req_Bean reqBean,
    Response_Msg_Bean     respBean) throws Exception{

  CanvasAlert help;

  switch(respBean.getResponseCode()){

  case SQL_Return_Codes.REJECT_PAYMENT_PRT_ERROR:
    help = new CanvasAlert(
        display,
        previous,
        "Error␣rejecting␣the␣tickets!",
        "An␣error␣occured␣while␣trying␣to␣reject␣the␣previous␣made␣seat␣
            selection!",
        "warn",
        CustomAlertTypes.ALERT_WARNING);
    break;

  case SQL_Return_Codes.REJECT_PAYMENT_PRT_OK:
    if(nextScreenName.equals("main")){
      display.setCurrent(new MenuScreen());
```

```
    }else if(nextScreenName.equals("exit")){

      DialogWindow reallyExit = new DialogWindow(
          display,
          previous,
          "Exit␣Application?",
          "Are␣you␣sure␣that␣you␣want␣to␣exit␣the␣application?",
          "question",
          "/dialogIcons/exitTheme",
          new MenuScreen().startingpoint);

      display.setCurrent(reallyExit);

    }
    break;

  default:
    help = new CanvasAlert(
        display,
        previous,
        "Payment␣Rejected␣Error␣Default!",
        "Payment␣Rejected␣Error␣Default␣in␣switch-case!",
        "error",
        CustomAlertTypes.ALERT_ERROR);
  break;

  }// end switch()

}// end displayRejectPaymentResponse()


/**
 * Check the response code from the Resp_Msg Bean
 * returned during a change password operation
 * and displays the appropiate UI screen function of
 * the return value
 *
 * @param display The application display
 * @param next    The next UI to display
 * @param previous The previous UI where it was called
 * @param reqBean The Change_Password_Req_Bean
 * @param respBean The Resp_Msg_Bean
 */
public static void displayChangePasswordResponse(
    Display               display,
    Displayable           previous,
```

```
    Change_Password_Req_Bean reqBean,
    Response_Msg_Bean      respBean) throws Exception{

  CanvasAlert help;

  switch(respBean.getResponseCode()){

  case SQL_Return_Codes.CHANGE_PASSWORD_PRT_USER_NOT_AUTHENTICATED:
    help = new CanvasAlert(
        display,
        previous,
        "Error␣changing␣the␣password!",
        "The␣old␣password␣is␣incorrect.␣User␣is␣not␣authenticated!",
        "warn",
        CustomAlertTypes.ALERT_WARNING);
    break;

  case SQL_Return_Codes.CHANGE_PASSWORD_PRT_PASSWORD_CHANGED:
    // add new password to RMS
    RMSOperations.deleteItems("PSW:");
    RMSOperations.writeEncryptedRecord("PSW:", reqBean.getNewPassword()
        );

    help = new CanvasAlert(
        display,
        new SettingsGUI().prepareScreen(),
        "Password␣Changed!",
        "Password␣has␣been␣changed␣successfuly!",
        "info",
        CustomAlertTypes.ALERT_INFO);
    break;

  default:
    help = new CanvasAlert(
        display,
        previous,
        "Password␣Change␣Error␣Default!",
        "Password␣Change␣Error␣Default␣in␣switch-case!",
        "error",
        CustomAlertTypes.ALERT_ERROR);
  break;

  }// end switch()

}// end displayCancelPasswordResponse()
```

```
}// end class

package networkoperations;

import gui.customdialogwindows.CanvasAlert;

import javax.microedition.lcdui.*;

/**
 * This class builds an animated progress gauge to be displayed
 * during the netowrk operations performed on a background thread.
 * An  cancel button is provided in order to stop an operation.
 *
 * @author Mihai Balan, Wojciech Dobrowolski
 *
 */
public class ProgressGauge extends Form implements Runnable,
    CommandListener {

  // flag to set the gauge thread to stopped or alive
  private boolean isGdgStopped;
  private Gauge gauge;
  // The gauge thread.
  private Thread gaugeThread;
  // The worker thread.
  private BackgroundTask bckgrTask;
  // The screen to go to if the Gauge is
  private Displayable prevScreen;
  // failed or stopped mannually.
  private Displayable nextScreen = null;
  private CanvasAlert alert = null;

  private StringItem infoTextUI = new StringItem("Processing␣your␣request
      ␣...","");;

  // to draw on
  private Display display;
  // the command to cancel the gauge
  private static final Command cancelCommand =
    new Command("Cancel",Command.BACK,2);

  /**
   * The Constructor - calls the constructor of the super class (Form)
   * and sets the previous screen and the worker thread to the ones
   * specicifed as parameters.
   */
  public ProgressGauge(BackgroundTask b, String title,
      Display d, Displayable p) {
```

```
  super("Please␣wait...");
  prevScreen = p;
  bckgrTask = b; // the worker (background) thread
  init(title, d);

}


/**
 * Set the info message displayed on the progress gauge
 * or the default one if not set it.
 *
 * @param infoTextLabel
 */
public void setInfoText(String infoTextLabel){
  infoTextUI.setLabel(infoTextLabel);

}

/**
 * Creates the gauge thread and new Gauge object
 * with the given title,in non-interactive mode
 * and with maximum value = 29 and initial value =0.
 * Then add the gauge and the comnand to the form
 * and set the command listener.
 * In the end it calls the start() method
 * that starts the gauge thread.
 */
private void init(String title, Display d) {
  try {
    display = d;
    gaugeThread = new Thread(this);
    gaugeThread.setPriority(Thread.MIN_PRIORITY);

    // creates the gauge object and add it to the form
    append(infoTextUI);
    gauge = new Gauge(title, false, 20, 0);
    append(gauge);

    addCommand(cancelCommand);
    setCommandListener(this);
    start();
  } catch (Exception e) {
    System.out.println("Error␣starting␣the␣Gauge");
    display.setCurrent(prevScreen);
  }
}
```

```
/**
 * Start the gauge thread
 */
public void start() {
  // set the gauge flag to alive
  isGdgStopped = false;
  // only start the tread if not alive
  if (gaugeThread.isAlive() == false) {
    gaugeThread.start();
  }
}

/**
 * Set the gauge thread to STOPPED
 * and the next screen displayed
 * is the previous screen before the gauge.
 */
public void stop() {
  if (nextScreen == null) {
    nextScreen = prevScreen;
  }
  // set the gauge flag to stopped.
  // It is gonna be checked in while loop frm run()
  // if the gauge thread is stopped or not.
  // If yes, the thread is not running any more.
  isGdgStopped = true;
}

/**
 * The gauge thread runs in a while loop where
 * its value is updated all the time
 */
public void run() {
  // In rare cases, stop() might be called after the thread
  // start() called but run() is not yet called ...
  if ( !isGdgStopped ){
    display.setCurrent(this);
  }

  // do until notified to stop/quit
  while (isGdgStopped == false) {
    for(int i=0; i < gauge.getMaxValue(); i++) {
      gauge.setValue(i);
      // temporarily pause the thread in order
      // for the other threads to execute
      gaugeThread.yield();
```

```
    }
    gauge.setValue(0);
  }
  //*****************************************
  // ***** NOT SURE IF WE NEED NULL IN HERE OR NOT *****
  gauge = null;
  //*****************************************

  // If NO alert occured go to the next screen
/*  if (alert == null) {
    display.setCurrent(nextScreen);

  // Else, make this alert and next screen (i.e. previous screen) the
      current ones.
  } else {
    display.setCurrent(alert);
  }*/
}

/**
 * Set the next screen to be displayed after leaving the gauge
 * in case an alert was displayed
 * @param a The displayed allert
 * @param d The displayable item
 */
/*public void setNextScreen (CanvasAlert a, Displayable d) {
  System.out.println("-----ProgressGauge -- in setNextScreen(a,d)");
  alert = a;
  nextScreen = d;
  System.out.println("-----ProgressGauge -- in setNextScreen(a,d) -
      alert: " +
     alert.toString() + " " + alert.getTitle());
  System.out.println("-----ProgressGauge -- in setNextScreen(a,d) -
      nextScreen: " +
     nextScreen.toString() + " " + ((Screen) nextScreen).getTitle());
}*/

/**
 * Set the next screen to be displayed after leaving the gauge
 * in case NO alert was displayed
 * @param d The displayable item
 */
/*public void setNextScreen (Displayable d) {
  System.out.println("-----ProgressGauge -- in setNextScreen(d)");
  alert = null;
  nextScreen = d;
  System.out.println("-----ProgressGauge -- in setNextScreen(a,d) -next
```

```
        screen: " + nextScreen.toString());
  }*/

  /**
   * In case CANCEL button is pressed stop the gauge and the background
       taks
   * and sets the next screen to the previous one before the gauge
   * @param c The executed command
   * @param d The alert form
   *
   */
  public void commandAction(Command c, Displayable d) {
    // if cancel is pressed stop both the gauge and worker threads
    // and return to the previous screen
    if (c == cancelCommand) {
      // stop the gauge thread
      stop();
      // stop the working thread
      bckgrTask.stop();
      display.setCurrent(prevScreen);
    }
  }

}

package networkoperations;

import gui.customdialogwindows.CanvasAlert;
import gui.mainmenu.MenuScreen;
import gui.purchasetickets.step2selectshow.SelectShowGUI;
import gui.purchasetickets.step4discountandreservationsummary.
    TicketDiscountAndReservationSummaryGUI;
import gui.purchasetickets.step6billinginfo.BillingInfoGUI;
import gui.settings.ChangePasswordGUI;

import java.io.*;
import javax.microedition.io.Connector;
import javax.microedition.io.HttpConnection;
import javax.microedition.lcdui.*;

import model.beans.otherbeans.TicketBean;
import model.beans.requestbeans.Cancel_Tickets_Req_Bean;
import model.beans.requestbeans.Change_Password_Req_Bean;
import model.beans.requestbeans.Cinema_Hall_Conf_Req_Bean;
import model.beans.requestbeans.Find_Movies_Req_Bean;
import model.beans.requestbeans.Movie_Details_Req_Bean;
import model.beans.requestbeans.Purchase_Tickets_Req_Bean;
import model.beans.requestbeans.Rate_Movie_Req_Bean;
```

```
import model.beans.requestbeans.Reject_Payment_Req_Bean;
import model.beans.requestbeans.Select_Deselect_Seats_Req_Bean;
import model.beans.responsebeans.Background_Cinema_Hall_Conf_Resp_Bean;
import model.beans.responsebeans.Cinema_Hall_Conf_Resp_Bean;
import model.beans.responsebeans.Find_Movies_Resp_Bean;
import model.beans.responsebeans.Movie_Details_Resp_Bean;
import model.beans.responsebeans.Purchase_Tickets_Resp_Bean;
import model.beans.responsebeans.Response_Msg_Bean;
import model.beans.responsebeans.Select_Deselect_Seats_Resp_Bean;

import org.bouncycastle.crypto.CryptoException;
import org.bouncycastle.crypto.params.ParametersWithIV;

import constants.CustomAlertTypes;
import constants.Protocol_Step_Constants;

import rms.RMSOperations;
import start.Start;


import cryptography.AesKey;
import cryptography.Encryptor;

/**
 * Sends a message to a remote URL using a background thread
 * to deal with network communication and another thread
 * to display an animated gauge in order to keep the user informed
 * about the of network communication status.
 * In case of an error an alert is displayed.
 *
 * It extends the BackgroundTask super class
 *
 * @author Mihai Balan (s031288)
 *
 */
public class SendMessage extends BackgroundTask{

  // the remote URL
  private String url = "http://127.0.0.1:9080/Cinema_Controller/
      cinemaservice/servlets/controller/Cinema_Central_Controller_Servlet
      ?protocol=";

  // used for encryption - decryption operation
  private Encryptor encryptor = null;

  // AES Key
  private AesKey aesK = AesKey.getInstance();
```

```
// For the communication with the 128 AES cipher
private ParametersWithIV aesKey = aesK.getKey();

// key provided by the user for encryption
private String key = "";
private String protocolStep = "";
private String alertText = "";

private String[] showInfo;


private Find_Movies_Req_Bean        findMovieReqBean;
private Find_Movies_Resp_Bean       findMovieRespBean;
private Rate_Movie_Req_Bean         rateMovieReqBean;
private Response_Msg_Bean           respBean;
private Movie_Details_Req_Bean      movDetReqBean;
private Movie_Details_Resp_Bean     movDetRespBean;
private Cinema_Hall_Conf_Req_Bean   cineHallConfReq;
private Cinema_Hall_Conf_Resp_Bean  cineHallConfResp;
private Select_Deselect_Seats_Req_Bean selDeselSeatsReqBean;
private Select_Deselect_Seats_Resp_Bean selDeselSeatsRespBean;
private Purchase_Tickets_Req_Bean   purchaseTicketsReqBean;
private Purchase_Tickets_Resp_Bean  purchaseTicketsRespBean;
private Cancel_Tickets_Req_Bean     cancelTicketsReqBean;
private Reject_Payment_Req_Bean     rejectPaymentReqBean;
private Change_Password_Req_Bean    changePswdReqBean;

private Displayable next;
private ChoiceGroup cgTickets;

private TicketBean genericTicket;
private int[][] reservedSeats;
private TicketBean[] cinemaTickets;

private String ccType    = "";
private String ccNo      = "";
private String ccValidMonth = "";
private String ccValidYear = "";
private String ccCW2     = "";

private String nextScreenName = "";

private String oldPassword = "";
private String newPassword = "";

private CanvasAlert help;
```

```
/**
 * Constructs a background task and
 * initiate the messsage to be sent and the remote url.
 * It also sets up the screen to return to in case of exception
 *
 * @param display    The display of the MIDlet used to display
     everything on.
 * @param protocolStep The protocol step.
 * @param previous   The screen where the request came from
 * @param reqBean    The request Bean to be sent through the network
 * @throws Exception
 */
public SendMessage(Display display, String protocolStep, Displayable
    previous, Object reqBean) throws Exception{

  super(display);

  if(protocolStep.equals(Protocol_Step_Constants.PRT_STEP_FIND_MOVIES))
      {
    findMovieReqBean     = (Find_Movies_Req_Bean)reqBean;

  }else if(protocolStep.equals(Protocol_Step_Constants.
      PRT_STEP_RATE_MOVIE)){
    rateMovieReqBean     = (Rate_Movie_Req_Bean)reqBean;

  }else if(protocolStep.equals(Protocol_Step_Constants.
      PRT_STEP_MOVIE_DETAILS)){
    movDetReqBean        = (Movie_Details_Req_Bean)reqBean;

  }else if(protocolStep.equals(Protocol_Step_Constants.
      PRT_STEP_SELECT_SHOW_AND_DISPLAY_CINEMA_HALL_CONF)){
    cineHallConfReq      = (Cinema_Hall_Conf_Req_Bean)reqBean;

  }else if(protocolStep.equals(Protocol_Step_Constants.
      PRT_STEP_SELECT_DESELECT_SEATS)){
    selDeselSeatsReqBean = (Select_Deselect_Seats_Req_Bean)reqBean;

  }else if(protocolStep.equals(Protocol_Step_Constants.
      PRT_STEP_PURCHASE_TICKETS)){
    purchaseTicketsReqBean = (Purchase_Tickets_Req_Bean)reqBean;

  }else if(protocolStep.equals(Protocol_Step_Constants.
      PRT_STEP_CANCEL_TICKETS)){
    cancelTicketsReqBean = (Cancel_Tickets_Req_Bean)reqBean;
```

```
  }else if(protocolStep.equals(Protocol_Step_Constants.
      PRT_STEP_REJECT_PAYMENT)){
    rejectPaymentReqBean = (Reject_Payment_Req_Bean)reqBean;

  }else if(protocolStep.equals(Protocol_Step_Constants.
      PRT_STEP_CHANGE_PASSWORD)){
    changePswdReqBean = (Change_Password_Req_Bean)reqBean;

  }

  this.url        = url + protocolStep;
  this.protocolStep = protocolStep;
  this.key        = Start.userKey;
  prevScreen      = previous;
  localProtocolStep = protocolStep;

}// end SendMessage()


/**
 * Set the show info used to create the generic ticket bean
 *
 * @param showInfo
 */
public void setShowInfo(String[] showInfo){
  this.showInfo = showInfo;

}// setShowInfo()


/**
 * Set the next screen to be displayed after the select seats screen
 *
 * @param nextLocalScreen
 */
public void setNextScreenAfterSeatSelectionConfParams(
    int[][]                     reservedSeats,
    TicketBean                  genericTicket,
    Cinema_Hall_Conf_Req_Bean   cineHallConfReq,
    Cinema_Hall_Conf_Resp_Bean  cineHallConfResp,
    Select_Deselect_Seats_Req_Bean selDeselSeatsReqBean){

  this.reservedSeats     = reservedSeats;
  this.genericTicket     = genericTicket;
  this.cineHallConfReq   = cineHallConfReq;
  this.cineHallConfResp  = cineHallConfResp;
```

```
  this.selDeselSeatsReqBean = selDeselSeatsReqBean;

}// setNextScreenAfterSeatSelectionConfParams()


/**
 * Set the credit card data to be later used for creating the pay
     tickets request bean
 *
 * @param ccType
 * @param ccNo
 * @param ccValidMonth
 * @param ccValidYear
 * @param ccCW2
 */
public void setCreditCardData(String ccType, String ccNo, String
    ccValidMonth, String ccValidYear, String ccCW2){
  this.ccType     = ccType;
  this.ccNo       = ccNo;
  this.ccValidMonth = ccValidMonth;
  this.ccValidYear = ccValidYear;
  this.ccCW2      = ccCW2;

}// setCreditCardData


/**
 * Set the cinema tickets to be saved to RMS and displa the Billing
     Info GUI
 * @param cinemaTickets
 */
public void setCinemaTickets(TicketBean[] cinemaTickets){
  this.cinemaTickets = cinemaTickets;

}// setCinemaTickets()


/**
 * Set the cancel ticket data to be sent sent throug the network
 * for canceling the ticket
 *
 * @param cinemaTickets
 */
public void setCancelTicketsData(
    Displayable           next,
    ChoiceGroup           cgTickets,
    TicketBean[]          cinemaTickets,
```

```
    Cancel_Tickets_Req_Bean cancelTicketsReqBean){

  this.next               = next;
  this.cgTickets          = cgTickets;
  this.cinemaTickets      = cinemaTickets;
  this.cancelTicketsReqBean = cancelTicketsReqBean;

}// setCancelTicketsData()


/**
 * Set the reject payment parameters to be sent sent throug the network
 * for rejecting the payment and caceling any previous selected seats
 *
 * @param cinemaTickets
 */
public void setRejectPaymentData(
    String nextScreenName){

  this.nextScreenName = nextScreenName;

}// setCancelTicketsData()



/**
 * Set the change password parameters to be sent sent throug the
     network
 * for changing appl password
 *
 */
public void setChangePasswordData(
       String oldPassword,
       String newPassword){

  this.oldPassword = oldPassword;
  this.newPassword = newPassword;

}// setChangePasswordData()



/**
 * Encrypt the message using user's key from the record store
 * Return the encrypted string to be sent to the remote URL
 *
 * @param key The user's key used to encrypt the message.
```

```
 * @param msg The MIDlet's request that is to be encrypted.
 *
 * @return    Returns the MIDlet's encrypted request as a byte array.
 * @throws    Throws Exceprion.
 */
protected byte[] encryptMessage(String key, String msgToEncrypt) throws
     Exception{
  encryptor = new Encryptor(key);
  byte[] encrypted = encryptor.encryptWithAES(aesKey, msgToEncrypt.
      getBytes());
  System.out.println("-----␣Send␣encrypted␣message␣to␣Worker␣Servlet
      ␣--␣Encrypted␣string:␣" + encrypted.toString());
  return encrypted;
}


/**
 * Decrypt the message from the servlet using user's key
 * stored in the record store. Return the dencrypted string.
 *
 * @param key The key used to decrypt the message.
 * @param msg The servlet's encrypted answer that is to be decrypted.
 *
 * @return    Returns the decripted message as a String.
 * @throws    CryptoException in case of crypto operations.
 */
protected byte[] decryptMessage(String key, byte[] msg) throws
    CryptoException{
  encryptor = new Encryptor(key);
  byte[] decrypted = encryptor.decryptWithAES(aesKey, msg);
  System.out.println("----␣SEND␣MESSAGE␣---␣␣Decrypted␣string:␣" +
      decrypted);
  return decrypted;
}

/**
 * Try to connect to the remote URL and send the message.
 * Open the connection and set the request type to POST.
 * Set the User-Agent, Content-Language, Connection,
 * and Content-Length request properties.
 * Create a OutputStream to write the message to
 * and write the message as binary.
 * Reads the servlet answer and if OK open and InputStream
 * to read the aswer to as binary. Decrypt the servlet's answer
 * and display a form with the decrypted answer.
 * @throws Exception In caseof errors in network operations
 */
```

```
public void runTask() throws Exception{

  if (protocolStep.equals(Protocol_Step_Constants.
      PRT_STEP_MOVIE_DETAILS)){
    // Movie Details Case
    getMovieDetails();

  } else if(protocolStep.equals(Protocol_Step_Constants.
      PRT_STEP_CHANGE_PASSWORD)){
    // change user password
    changePassword();

  }else if(protocolStep.equals(Protocol_Step_Constants.
      PRT_STEP_FIND_MOVIES)){
    // find movies
    findMovies();

  } else if(protocolStep.equals(Protocol_Step_Constants.
      PRT_STEP_SELECT_SHOW_AND_DISPLAY_CINEMA_HALL_CONF)){
    // get Cinema Hall Configuration
    cinemaHallConf();

  } else if(url.endsWith("BHU")) {
    // get back Cinema Hall Configuration
    backgroundCinemaHallUpdate();

  } else if(protocolStep.equals(Protocol_Step_Constants.
      PRT_STEP_SELECT_DESELECT_SEATS)){
    // select deselect user chosen seats
    selectDeselectSeats();

  } else if(protocolStep.equals(Protocol_Step_Constants.
      PRT_STEP_REJECT_PAYMENT)){
    // reject made reservation
    rejectPayment();

  } else if(protocolStep.equals(Protocol_Step_Constants.
      PRT_STEP_PURCHASE_TICKETS)){
    // pay for the reserved tickets
    purchaseTickets();

  } else if(protocolStep.equals(Protocol_Step_Constants.
      PRT_STEP_CANCEL_TICKETS)){
    // cancel purchased tickets by credit card
    cancelTickets();

  }else if(protocolStep.equals(Protocol_Step_Constants.
```

```
      PRT_STEP_RATE_MOVIE)){
    // rate the movie
    rateMovie();

  } else {
    help = new CanvasAlert(
        display,
        prevScreen,
        "Communication␣Error!",
        "Invalid␣Protocol␣Name!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

  } // end if (url.endsWith())

} // end runTask()


/**
 * Check from which screen the message was sent from (e.g. Authenticate
     ,
 * WriteMessage). Then it schecks the servlet's answer and returns
 * an apropiate code 1, 2 or 3 corresponding to Authentication failure,
 * Authentication Success, or else. in case of Authentication Success,
 * store the authentication Token in mobile phoneRMS.
 *
 * @param response The response from the Servlet
 * @return A code i.e. 1, 2 or 3 corresponding to Authentication
     failure,
 *     Authentication Success, or else. The code is used to display
 *     a customized answer on the display
 * @throws Exception RMSException in case of RMS operations
 */
public int storeToken(String response) throws Exception{

  // if the message was sent from the Authenticaion display
  if(protocolStep.equals(Protocol_Step_Constants.
      PRT_STEP_AUTHENTICATION_1)){
    // check the servlet's answer and return the apropiate code
    if (response.equals("Authentication␣Failure")){
      return 1;
    } else{
      // if no valid token is received
      int pos = response.indexOf(":");
      if (pos == -1) {
        return 1;
      }
```

```
      else{
        // in case of Authentication Success, store the
        // authentication Token in RMS
        // delete all tokens before inserting a new token
        RMSOperations.deleteItems("TKN:");
        //add the authentication token to the record store
        RMSOperations.writeRecord("TKN:", response);
        return 2;
      }
    }
  }
  return 3;
}


private void backgroundCinemaHallUpdate(){

  // Open HTTPConnection and the corresponding data Output / Input
      Streams
  // in order to write / read operations with the servlet
  HttpConnection conn = null;
  DataInputStream dis = null;
  DataOutputStream dos = null;

  try{
    conn = openConnection(url);
    dos = openDataOutputStream(conn);

    // write the message to the servlet
    Cinema_Hall_Conf_Req_Bean cineHallConfReq = new
        Cinema_Hall_Conf_Req_Bean();
    cineHallConfReq.setShowLocationID(4);
    cineHallConfReq.setShowTimeID(15);
    cineHallConfReq.writeBean(dos);
    System.out.println(cineHallConfReq.toString());
    dos.close();

    // get the response code from the servlet
    int rc = conn.getResponseCode();

    String testMsg = "";
    alertText = "" + rc;



    System.out.println("--------------------IN␣BACKGROUND␣CINEMA␣HALL␣
        UPDATE␣----␣RESPONSE␣CODE" + rc);
```

```
    if (rc == HttpConnection.HTTP_OK) {
      dis  = openDataInputStream(conn);
      Background_Cinema_Hall_Conf_Resp_Bean bckgCinemaHallUpdateResp
           = (Background_Cinema_Hall_Conf_Resp_Bean)
          Background_Cinema_Hall_Conf_Resp_Bean.readBean(dis);

      int[][] seats = new int[bckgCinemaHallUpdateResp.
          getAllBookedSeatsRows()][bckgCinemaHallUpdateResp.
          getAllBookedSeatsCols()];
      seats = bckgCinemaHallUpdateResp.getAllBookedSeats();

      testMsg = "\n" + bckgCinemaHallUpdateResp.toString();
      System.out.println("----␣SEND␣MESSAGE1111␣---␣CINEMA␣HALL␣CONF␣
          BEAN␣FROM␣THE␣SERVLET␣=␣␣" + testMsg);
      // alertText += testMsg;

    } else {
      testMsg = "----␣SEND␣MESSAGE1111␣---␣␣cinema␣hall␣conf␣SERVLET␣
          RESPONSE␣CORRUPTED!";
      System.out.println(testMsg);
      alertText = testMsg;

    } // end if (rc == HttpConnection.HTTP_OK)

  } catch(IOException ioe){
    System.out.println("-------------------You␣got␣an␣IOException␣in␣
        BACKGROUND␣CINEMA␣HALL␣UPDATE␣details");
    ioe.printStackTrace();

  } catch (Exception e){
    System.out.println("-------------------You␣got␣an␣Exception␣in␣
        BACKGROUND␣CINEMA␣HALL␣UPDATE␣details");
    e.printStackTrace();

  } finally {
    // Close all still opened connections ans streams
    closeHTTPConnection(conn, dos, dis);

    // display in a form the decrypted message from the servlet
    //(new MyAlert("Servlet's Answer", alertText)).showScreen();
    System.out.println("----␣SEND␣MESSAGE␣---␣␣DECRIPTED␣message␣from␣
        the␣servlet␣=␣␣" + alertText);
  } // end finally

} //backgroundCinemaHallUpdate()
```

```
/**
 * Sends the Cinema Hall Configuration Request Bean over the network
 * and receives the Cinema Hall Configuration Response Bean from the
     server side.
 * It delegates the GUI rendering to the NetworkResposeFacade class
 * where the respose code is checked and the apropiate
 * UI screen is displayed.
 *
 */
private void cinemaHallConf(){

  // Open HTTPConnection and the corresponding data Output / Input
      Streams
  // in order to write / read operations with the servlet
  HttpConnection conn = null;
  DataInputStream dis = null;
  DataOutputStream dos = null;

  try{
    conn = openConnection(url);
    dos = openDataOutputStream(conn);

    System.out.println(cineHallConfReq.toString());

    cineHallConfReq.writeBean(dos);
    dos.close();

    if (conn.getResponseCode() == HttpConnection.HTTP_OK) {
      dis  = openDataInputStream(conn);
      cineHallConfResp = (Cinema_Hall_Conf_Resp_Bean)
          Cinema_Hall_Conf_Resp_Bean.readBean(dis);

    } else {
      help = new CanvasAlert(
          display,
          prevScreen,
          "Cinema␣Hall␣Conf␣Error!",
          "Response␣from␣the␣server␣corrupted!",
          "error",
          CustomAlertTypes.ALERT_ERROR);

    } // end if (rc == HttpConnection.HTTP_OK)

  } catch(IOException ioe){
    help = new CanvasAlert(
        display,
```

```
        prevScreen,
        "Cinema␣Hall␣Conf␣Error!",
        "An␣error␣was␣encounted␣while␣getting␣the␣cinema␣hall␣
            configuration!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    System.out.println("-------------------You␣got␣an␣IOException␣in␣
        CINEMA␣HALL␣CONF␣details");
    ioe.printStackTrace();

  } catch (Exception e){
    help = new CanvasAlert(
        display,
        prevScreen,
        "Cinema␣Hall␣Conf␣Error!",
        "An␣error␣was␣encounted␣while␣getting␣the␣cinema␣hall␣
            configuration!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    System.out.println("-------------------You␣got␣an␣Exception␣in␣
        CINEMA␣HALL␣CONF␣details");
    e.printStackTrace();

  } finally {
    // Close all still opened connections ans streams
    closeHTTPConnection(conn, dos, dis);

    System.out.println(cineHallConfResp.toString());

    try{
      NetworkResposeFacade.displayCinemaHallConfResponse(
          display,
          prevScreen,
          showInfo,
          cineHallConfReq,
          cineHallConfResp);

    }catch(Exception e){
      help = new CanvasAlert(
          display,
          prevScreen,
          "Cinema␣Hall␣Conf␣Error!",
          "An␣error␣was␣encounted␣while␣getting␣the␣cinema␣hall␣
              configuration!",
          "error",
```

```
          CustomAlertTypes.ALERT_ERROR);

    }

    Runtime runtime = Runtime.getRuntime();
    long t = runtime.freeMemory();
    System.out.println("************************Memery␣before␣net:" + t
        );
    help = null;
    findMovieReqBean = null;
    findMovieRespBean = null;
    rateMovieReqBean = null;
    respBean = null;
    movDetReqBean = null;
    movDetRespBean = null;
    cineHallConfReq = null;
    cineHallConfResp = null;
    aesK = null;
    aesKey = null;
    url = null;
    prevScreen = null;
    showInfo = null;

    System.gc();
    long t1 = runtime.freeMemory();
    System.out.println("************************Memery␣after␣net:" + t1
        );

  } // end finally

} //cinemaHallConf()



/**
 * Sends the Select Deselect Seats Request Bean over the network
 * and receives the Select Deselect Seats Response Bean from the server
      side.
 * It delegates the GUI rendering to the NetworkResposeFacade class
 * where the respose code is checked and the apropiate
 * UI screen is displayed.
 *
 */
private void selectDeselectSeats(){

  // Open HTTPConnection and the corresponding data Output / Input
      Streams
```

```
// in order to write / read operations with the servlet
HttpConnection conn = null;
DataInputStream dis = null;
DataOutputStream dos = null;

try{
  conn = openConnection(url);
  dos = openDataOutputStream(conn);

  System.out.println(selDeselSeatsReqBean.toString());

  selDeselSeatsReqBean.writeBean(dos);
  dos.close();

  if (conn.getResponseCode() == HttpConnection.HTTP_OK) {
    dis  = openDataInputStream(conn);
    selDeselSeatsRespBean = (Select_Deselect_Seats_Resp_Bean)
        Select_Deselect_Seats_Resp_Bean.readBean(dis);

  }else{
    help = new CanvasAlert(
        display,
        prevScreen,
        "Select␣Seats␣Error!",
        "Response␣from␣the␣server␣corrupted!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

  } // end if (rc == HttpConnection.HTTP_OK)

} catch(IOException ioe){
  help = new CanvasAlert(
      display,
      prevScreen,
      "Select␣Seats␣Error!",
      "An␣error␣was␣encounted␣while␣trying␣to␣select␣the␣seats!",
      "error",
      CustomAlertTypes.ALERT_ERROR);

  System.out.println("-------------------You␣got␣an␣IOException␣in␣
      SELECT␣DESELECT␣SEATS");
  ioe.printStackTrace();

} catch (Exception e){
  help = new CanvasAlert(
      display,
      prevScreen,
```

```
        "Select␣Seats␣Error!",
        "An␣error␣was␣encounted␣while␣trying␣to␣select␣the␣seats!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    System.out.println("-------------------You␣got␣an␣Exception␣in␣
        SELECT␣DESELECT␣SEATS");
    e.printStackTrace();

  } finally {
    // Close all still opened connections ans streams
    closeHTTPConnection(conn, dos, dis);
    System.out.println(selDeselSeatsRespBean.toString());

    try{
      NetworkResposeFacade.displaySelectDeselectSeatsResponse(
          display,
          prevScreen,
          new TicketDiscountAndReservationSummaryGUI(
              reservedSeats,
              genericTicket,
              cineHallConfReq,
              cineHallConfResp,
              selDeselSeatsReqBean,
              selDeselSeatsRespBean).prepareScreen(),
              selDeselSeatsRespBean);

    }catch(Exception e){
      help = new CanvasAlert(
          display,
          prevScreen,
          "Select␣Seats␣Error!",
          "An␣error␣was␣encounted␣while␣trying␣to␣select␣the␣seats!",
          "error",
          CustomAlertTypes.ALERT_ERROR);

    }// end inner try-catch

  } // end finally

} //selectDeselectSeats()



/**
 * Sends the Reject Payment Request Bean over the network
 * and receives the Reject Payment Response Bean from the server side.
```

```
 * It delegates the GUI rendering to the NetworkResposeFacade class
 * where the respose code is checked and the apropiate
 * UI screen is displayed.
 *
 */
private void rejectPayment(){

  HttpConnection conn = null;
  DataOutputStream dos = null;
  DataInputStream dis = null;

  try {
    conn = openConnection(url);
    dos = openDataOutputStream(conn);
    System.out.println(rejectPaymentReqBean.toString());
    rejectPaymentReqBean.writeBean(dos);
    dos.close();

    if (conn.getResponseCode() == HttpConnection.HTTP_OK) {

      dis  = openDataInputStream(conn);
      respBean = Response_Msg_Bean.readBean(dis);

    }else{
      help = new CanvasAlert(
          display,
          prevScreen,
          "Reject payment error!",
          "An error was encounted while rejecting the payment!",
          "error",
          CustomAlertTypes.ALERT_ERROR);
    }

  } catch(IOException ioe){
    help = new CanvasAlert(
        display,
        prevScreen,
        "Reject payment error!",
        "An error was encounted while rejecting the payment!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    System.out.println("-------------------You got an IOException in
        REJECT PAYMENT ");
    ioe.printStackTrace();

  } catch (Exception e){
```

```
    help = new CanvasAlert(
        display,
        prevScreen,
        "Reject␣payment␣error!",
        "An␣error␣was␣encounted␣while␣rejecting␣the␣payment!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    System.out.println("--------------------You␣got␣an␣Exception␣␣in␣
        REJECT␣PAYMENT␣");
    e.printStackTrace();

  } finally {
    // Close all still opened connections ans streams
    closeHTTPConnection(conn, dos, dis);

    System.out.println(respBean.toString());

    try{
      NetworkResposeFacade.displayRejectPaymentResponse(
          display,
          prevScreen,
          nextScreenName,
          rejectPaymentReqBean,
          respBean);

    }catch(Exception e){
      help = new CanvasAlert(
          display,
          prevScreen,
          "Reject␣payment␣error!",
          "An␣error␣was␣encounted␣while␣rejecting␣the␣payment!",
          "error",
          CustomAlertTypes.ALERT_ERROR);

    }// end try-catch

  } // end finally

} // rejectPayment()



/**
 * Sends the Purchase Tickets Request Bean over the network
 * and receives the Purchase Tickets Response Bean from the server side
     .
```

```
 * It delegates the GUI rendering to the NetworkResposeFacade class
 * where the respose code is checked and the apropiate
 * UI screen is displayed.
 *
 */
private void purchaseTickets(){

  HttpConnection conn = null;
  DataOutputStream dos = null;
  DataInputStream dis = null;

  try {
    conn = openConnection(url);
    dos = openDataOutputStream(conn);

    /********************* Encrypted data setup *****************/
    // set encrypted password
    purchaseTicketsReqBean.setPassword(encryptMessage(key, Start.
        userPassword));
    // set encrypted credit card data
    purchaseTicketsReqBean.setCreditCardType(encryptMessage(key, ccType
        ));
    purchaseTicketsReqBean.setCreditCardNo(encryptMessage(key, ccNo));

    if((ccValidMonth.equals("")) || (ccValidYear.equals("")) ||
       (ccType.equals("")) || (ccNo.equals(""))){
      purchaseTicketsReqBean.setCreditCardExpDate(encryptMessage(key, "
          "));

    }else{
      purchaseTicketsReqBean.setCreditCardExpDate(encryptMessage(key,
          ccValidMonth + "-" + ccValidYear));

    }// end if()

    purchaseTicketsReqBean.setCreditCardCW2(encryptMessage(key, ccCW2))
        ;
    /*************************************************************/

    System.out.println(purchaseTicketsReqBean.toString());

    purchaseTicketsReqBean.writeBean(dos);
    dos.close();

    if (conn.getResponseCode() == HttpConnection.HTTP_OK) {

      dis  = openDataInputStream(conn);
```

```
      purchaseTicketsRespBean = (Purchase_Tickets_Resp_Bean)
          Purchase_Tickets_Resp_Bean.readBean(dis);

      System.out.println(purchaseTicketsRespBean.toString());

    }else{
      help = new CanvasAlert(
          display,
          prevScreen,
          "Ticket␣payment␣error!",
          "An␣error␣was␣encounted␣while␣paying␣for␣the␣tickets!",
          "error",
          CustomAlertTypes.ALERT_ERROR);
    }

  } catch(IOException ioe){
    help = new CanvasAlert(
        display,
        prevScreen,
        "Ticket␣payment␣error!",
        "An␣error␣was␣encounted␣while␣paying␣for␣the␣tickets!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    System.out.println("-------------------You␣got␣an␣IOException␣in␣
        TICKET␣PAYMENT␣");
    ioe.printStackTrace();

  } catch (Exception e){
    help = new CanvasAlert(
        display,
        prevScreen,
        "Ticket␣payment␣error!",
        "An␣error␣was␣encounted␣while␣paying␣for␣the␣tickets!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    System.out.println("-------------------You␣got␣an␣Exception␣␣in␣
        TICKET␣PAYMENT␣");
    e.printStackTrace();

  } finally {
    // Close all still opened connections ans streams
    closeHTTPConnection(conn, dos, dis);

    System.out.println(purchaseTicketsRespBean.toString());
```

```
    try{
      NetworkResposeFacade.displayPurchaseTicketsResponse(
          display,
          prevScreen,
          new BillingInfoGUI(cinemaTickets, purchaseTicketsRespBean).
              prepareScreen(),
          purchaseTicketsReqBean,
          purchaseTicketsRespBean);

    }catch(Exception e){
      help = new CanvasAlert(
          display,
          prevScreen,
          "Ticket␣payment␣error!",
          "An␣error␣was␣encounted␣while␣paying␣for␣the␣tickets!",
          "error",
          CustomAlertTypes.ALERT_ERROR);

    }// end try-catch

  } // end finally

} // payTickets()


/**
 * Sends the Rate Movie Request Bean over the network
 * and receives the Response Msg Bean from the server side.
 * It delegates the GUI rendering to the NetworkResposeFacade class
 * where the respose code is checked and the apropiate
 * UI screen is displayed.
 *
 */
private void rateMovie(){

  HttpConnection conn = null;
  DataOutputStream dos = null;
  DataInputStream dis = null;

  try {
    conn = openConnection(url);
    dos = openDataOutputStream(conn);

    rateMovieReqBean.setPassword(encryptMessage(key, Start.userPassword
        ));
    rateMovieReqBean.writeBean(dos);
    dos.close();
```

```
    if (conn.getResponseCode() == HttpConnection.HTTP_OK){
      dis  = openDataInputStream(conn);
      respBean = Response_Msg_Bean.readBean(dis);
    }else{
      help = new CanvasAlert(
          display,
          prevScreen,
          "Movie␣Rating␣Error!",
          "An␣error␣was␣encounted␣while␣rating␣the␣movie!",
          "error",
          CustomAlertTypes.ALERT_ERROR);
    }

  } catch(IOException ioe){
    help = new CanvasAlert(
        display,
        prevScreen,
        "Movie␣Rating␣Error!",
        "An␣error␣was␣encounted␣while␣rating␣the␣movie!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    System.out.println("-------------------You␣got␣an␣IOException␣in␣
        RATE␣MOVIE");
    ioe.printStackTrace();

  } catch (Exception e){
    help = new CanvasAlert(
        display,
        prevScreen,
        "Movie␣Rating␣Error!",
        "An␣error␣was␣encounted␣while␣rating␣the␣movie!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    System.out.println("-------------------You␣got␣an␣Exception␣␣in␣
        RATE␣MOVIE");
    e.printStackTrace();

  } finally {
    // Close all still opened connections ans streams
    closeHTTPConnection(conn, dos, dis);

    System.out.println(rateMovieReqBean.toString());

    try{
```

```
      NetworkResposeFacade.displayRateMovieResponse(
          display,
          prevScreen,
          prevScreen,
          rateMovieReqBean,
          respBean);

    }catch(Exception e){
      help = new CanvasAlert(
          display,
          prevScreen,
          "Movie␣Rating␣Error!",
          "An␣error␣was␣encounted␣while␣rating␣the␣movie!",
          "error",
          CustomAlertTypes.ALERT_ERROR);

    }// end try-catch

  } // end finally

} // rateMovie()


private void getMovieDetails(){

  // Open HTTPConnection and the corresponding data Output / Input
      Streams
  // in order to write / read operations with the servlet
  HttpConnection conn = null;
  DataInputStream dis = null;
  DataOutputStream dos = null;

  try{
    conn = openConnection(url);
    dos = openDataOutputStream(conn);

    movDetReqBean.writeBean(dos);
    dos.close();

    if (conn.getResponseCode() == HttpConnection.HTTP_OK) {
      dis  = openDataInputStream(conn);
      movDetRespBean = (Movie_Details_Resp_Bean)Movie_Details_Resp_Bean
          .readBean(dis);

    } else {
      help = new CanvasAlert(
          display,
```

```
          prevScreen,
          "Movie␣Details␣Error!",
          "An␣error␣was␣encounted␣while␣retriving␣the␣movie␣details!",
          "error",
          CustomAlertTypes.ALERT_ERROR);

    } // end if (rc == HttpConnection.HTTP_OK)

  } catch(IOException ioe){
    help = new CanvasAlert(
        display,
        prevScreen,
        "Movie␣Details␣Error!",
        "An␣error␣was␣encounted␣while␣retriving␣the␣movie␣details!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    System.out.println("-------------------You␣got␣an␣IOException␣in␣
        get␣movie␣details");
    ioe.printStackTrace();

  } catch (Exception e){
    help = new CanvasAlert(
        display,
        prevScreen,
        "Movie␣Details␣Error!",
        "An␣error␣was␣encounted␣while␣retriving␣the␣movie␣details!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    System.out.println("-------------------You␣got␣an␣Exception␣in␣get␣
        movie␣details");
    e.printStackTrace();

  } finally {
    // Close all still opened connections ans streams
    closeHTTPConnection(conn, dos, dis);

    System.out.println(movDetRespBean.toString());

    try{
      NetworkResposeFacade.displayMovieDetails(
          display,
          new SelectShowGUI().getScreen(),
          movDetReqBean,
          movDetRespBean);
```

```
    }catch(Exception e){
      help = new CanvasAlert(
          display,
          prevScreen,
          "Movie␣Details␣Error!",
          "An␣error␣was␣encounted␣while␣retriving␣the␣movie␣details!",
          "error",
          CustomAlertTypes.ALERT_ERROR);

    }

  } // end finally

} //getMovieDetails()


/**
 * Sends the Change Password Request Bean over the network
 * and receives the Change Password Response Bean from the server side.
 * It delegates the GUI rendering to the NetworkResposeFacade class
 * where the respose code is checked and the apropiate
 * UI screen is displayed.
 *
 */
private void changePassword(){

  HttpConnection conn = null;
  DataOutputStream dos = null;
  DataInputStream dis = null;

  try {
    conn = openConnection(url);
    dos = openDataOutputStream(conn);

    changePswdReqBean.setOldPassword(encryptMessage(key, oldPassword));
    changePswdReqBean.setNewPassword(encryptMessage(key, newPassword));
    System.out.println(changePswdReqBean.toString());
    changePswdReqBean.writeBean(dos);
    dos.close();

    if (conn.getResponseCode() == HttpConnection.HTTP_OK) {

      dis  = openDataInputStream(conn);
      respBean = Response_Msg_Bean.readBean(dis);

    }else{
      help = new CanvasAlert(
```

```
          display,
          prevScreen,
          "Change␣Password␣Error!",
          "An␣error␣was␣encounted␣while␣trying␣to␣change␣the␣password!",
          "error",
          CustomAlertTypes.ALERT_ERROR);
    }

  } catch(IOException ioe){
    help = new CanvasAlert(
        display,
        prevScreen,
        "Change␣Password␣Error!",
        "An␣error␣was␣encounted␣while␣trying␣to␣change␣the␣password!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    System.out.println("-------------------You␣got␣an␣IOException␣in␣
         change␣password");
    ioe.printStackTrace();

  } catch (Exception e){
    help = new CanvasAlert(
        display,
        prevScreen,
        "Change␣Password␣Error!",
        "An␣error␣was␣encounted␣while␣trying␣to␣change␣the␣password!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    System.out.println("-------------------You␣got␣an␣Exception␣␣in␣
         change␣password");
    e.printStackTrace();

  } finally {
    // Close all still opened connections ans streams
    closeHTTPConnection(conn, dos, dis);

    System.out.println(respBean.toString());

    try{
      NetworkResposeFacade.displayChangePasswordResponse(
          display,
          prevScreen,
          changePswdReqBean,
          respBean);
```

```
    }catch(Exception e){
      help = new CanvasAlert(
          display,
          prevScreen,
          "Change␣Password␣Error!",
          "An␣error␣was␣encounted␣while␣trying␣to␣change␣the␣password!",
          "error",
          CustomAlertTypes.ALERT_ERROR);
    }

  } // end finally

} // changePassword()


/**
 * Sends the Cancel Tickets Request Bean over the network
 * and receives the Cancel Tickets Response Bean from the server side.
 * It delegates the GUI rendering to the NetworkResposeFacade class
 * where the respose code is checked and the apropiate
 * UI screen is displayed.
 *
 */
private void cancelTickets(){

  HttpConnection conn = null;
  DataOutputStream dos = null;
  DataInputStream dis = null;

  try {
    conn = openConnection(url);
    dos = openDataOutputStream(conn);

    cancelTicketsReqBean.setPassword(encryptMessage(key, Start.
        userPassword));
    System.out.println(cancelTicketsReqBean.toString());
    cancelTicketsReqBean.writeBean(dos);
    dos.close();

    if (conn.getResponseCode() == HttpConnection.HTTP_OK) {

      dis  = openDataInputStream(conn);
      respBean = Response_Msg_Bean.readBean(dis);

    }else{
      help = new CanvasAlert(
          display,
```

```
          prevScreen,
          "Ticket␣Cancel␣Error!",
          "An␣error␣was␣encounted␣while␣trying␣to␣cancel␣the␣ticket!",
          "error",
          CustomAlertTypes.ALERT_ERROR);
    }

  } catch(IOException ioe){
    help = new CanvasAlert(
        display,
        prevScreen,
        "Ticket␣Cancel␣Error!",
        "An␣error␣was␣encounted␣while␣trying␣to␣cancel␣the␣ticket!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    System.out.println("-------------------You␣got␣an␣IOException␣in␣
        CANCEL␣TICLKETS");
    ioe.printStackTrace();

  } catch (Exception e){
    help = new CanvasAlert(
        display,
        prevScreen,
        "Ticket␣Cancel␣Error!",
        "An␣error␣was␣encounted␣while␣trying␣to␣cancel␣the␣ticket!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    System.out.println("-------------------You␣got␣an␣Exception␣␣in␣
        CANCEL␣TICLKETS");
    e.printStackTrace();

  } finally {
    // Close all still opened connections ans streams
    closeHTTPConnection(conn, dos, dis);

    System.out.println(respBean.toString());

    try{
      NetworkResposeFacade.displayCancelTicketsResponse(
          display,
          prevScreen,
          next,
          cgTickets,
          cinemaTickets,
          cancelTicketsReqBean,
```

```
          respBean);

    }catch(Exception e){
      help = new CanvasAlert(
          display,
          prevScreen,
          "Ticket␣Cancel␣Error!",
          "An␣error␣was␣encounted␣while␣trying␣to␣cancel␣the␣ticket!",
          "error",
          CustomAlertTypes.ALERT_ERROR);

      System.out.println("!Exception␣in␣the␣final␣branch␣in␣searching␣
          for␣movies");
      e.printStackTrace();

    }// end try-catch

  } // end finally

} // cancelTickets()


/**
 * Sends the Find Movie Request Bean over the network
 * and receives the Find Movie Response Bean from the server side.
 * It delegates the GUI rendering to the NetworkResposeFacade class
 * where the respose code is checked and the apropiate
 * UI screen is displayed.
 *
 */
private void findMovies(){

  HttpConnection conn = null;
  DataOutputStream dos = null;
  DataInputStream dis = null;


  try {
    conn = openConnection(url);
    dos = openDataOutputStream(conn);
    System.out.println(findMovieReqBean.toString());

    findMovieReqBean.writeBean(dos);
    dos.close();

    if (conn.getResponseCode() == HttpConnection.HTTP_OK) {
      dis  = openDataInputStream(conn);
```

```
      findMovieRespBean = (Find_Movies_Resp_Bean)Find_Movies_Resp_Bean.
          readBean(dis);

    }else{
      help = new CanvasAlert(
          display,
          prevScreen,
          "Movie␣Search␣Error!",
          "An␣error␣was␣encounted␣while␣searching␣for␣movies!",
          "error",
          CustomAlertTypes.ALERT_ERROR);
    }

  } catch(IOException ioe){
    help = new CanvasAlert(
        display,
        prevScreen,
        "Movie␣Search␣Error!",
        "An␣error␣was␣encounted␣while␣searching␣for␣movies!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    System.out.println("-------------------You␣got␣an␣IOException␣in␣
        FIND␣MOVIES");
    ioe.printStackTrace();

  } catch (Exception e){
    help = new CanvasAlert(
        display,
        prevScreen,
        "Movie␣Search␣Error!",
        "An␣error␣was␣encounted␣while␣searching␣for␣movies!",
        "error",
        CustomAlertTypes.ALERT_ERROR);

    System.out.println("-------------------You␣got␣an␣Exception␣␣in␣
        FIND␣MOVIES");
    e.printStackTrace();

  } finally {
    // Close all still opened connections ans streams
    closeHTTPConnection(conn, dos, dis);

    System.out.println(findMovieRespBean.toString());
    try{
      NetworkResposeFacade.displaySearchMoviesResponse(
          display,
```

```
          prevScreen,
          findMovieReqBean,
          findMovieRespBean);

    }catch(Exception e){
      help = new CanvasAlert(
          display,
          prevScreen,
          "Movie␣Search␣Error!",
          "An␣error␣was␣encounted␣while␣searching␣for␣movies!",
          "error",
          CustomAlertTypes.ALERT_ERROR);

      System.out.println("!Exception␣in␣the␣final␣branch␣in␣searching␣
          for␣movies");
      e.printStackTrace();

    }// end try-catch

    Runtime runtime = Runtime.getRuntime();
    long t = runtime.freeMemory();
    System.out.println("**********************Memery␣before␣net:" + t
        );
    help = null;
    findMovieRespBean = null;
    findMovieReqBean = null;
    rateMovieReqBean = null;
    respBean = null;
    movDetReqBean = null;
    movDetRespBean = null;
    aesK = null;
    aesKey = null;
    url = null;
    prevScreen = null;


    System.gc();
    long t1 = runtime.freeMemory();
    System.out.println("**********************Memery␣after␣net:" + t1
        );

  } // end finally

} // findMovies()
```

```
private HttpConnection openConnection(String url) throws IOException {

  HttpConnection conn = (HttpConnection) Connector.open(url);

  // set the User-Agent, Content-Type, Request Mode, Content-Language,
  // Connection-close, Content-Length request properties
  conn.setRequestProperty("User-Agent","Profile/MIDP-2.0␣Configuration/
      CLDC-1.0");
  conn.setRequestProperty("Content-Type", "application/octet-stream");
  conn.setRequestMethod(HttpConnection.POST);
  conn.setRequestProperty("Content-Language", "en-US");
  conn.setRequestProperty( "Connection", "close" );
  conn.setRequestProperty("Content-Length", "0");

  return conn;

} // end openConnection()



private DataOutputStream openDataOutputStream(HttpConnection conn)
    throws IOException{
  return conn.openDataOutputStream();

}// end openDataOutputStream()


private DataInputStream openDataInputStream(HttpConnection conn) throws
     IOException, Exception {
  try {
    int responseCode = conn.getResponseCode();

    if (responseCode == HttpConnection.HTTP_OK) {
      DataInputStream inputStream = conn.openDataInputStream();
      return inputStream;
    }

    throw new Exception("Http_Internal␣Error␣in␣the␣SEND␣MESSAGE␣
        openDataInputStream");

  } catch (IOException ioe) {
    throw ioe;
  }

}// end openDataInputStream()

private void closeHTTPConnection(HttpConnection conn, DataOutputStream
```

```
      dos, DataInputStream din) {

    // Close all still opened connections ans streams
    if (dos != null) {
      try { dos.close(); } catch (Throwable te) {}
      dos = null;
    }

    if (din != null) {
      try { din.close(); } catch (Throwable te) {}
      din = null;
    }

    if (conn != null) {
      try { conn.close(); } catch (Throwable te) {}
      conn = null;
    }

  } // end closeHTTPConnection()


}

package rms;

import gui.mywallet.MyWalletAuthenticationGUI;

import java.io.IOException;
import java.util.Vector;
import javax.microedition.rms.*;

import model.beans.otherbeans.CreditCardBean;
import model.beans.otherbeans.TicketBean;

import org.bouncycastle.crypto.CryptoException;



import cryptography.Encryptor;


/**
 * Helper class for Record Store operations e.g.
 * adding, deleting, or searching for elements in RMS.
 * It also provides methods to open, close and display
 * all elements in RMS
 *
 * @author Mihai Balan s031288, Wojciech Dobrowolski
```

```
 *
 */
public class RMSOperations {

  private static RecordStore rs = null;

  /**
   * Open the record store if exists.
   * It allows to use different record stores for diffent users
   * i.e. each user has its own mamory share that cannot
   * be accessed by other users
   * If it doesn't exists, create a new record store
   *
   * @throws Exception In case of RMS operations
   */
  public static void openRecStore(String recName) throws Exception{
    System.out.println("----␣RMSOperations␣---␣␣Opening␣RS:␣" + recName);
    rs = RecordStore.openRecordStore(recName, true);
    System.out.println("----␣RMSOperations␣---␣␣RMS␣opened!");
  }


  /**
   * Close the record store..
   * @throws Exception In case of RMS operations
   */
  public static void closeRecStore()throws Exception{
    rs.closeRecordStore();
    System.out.println("----␣RMSOperations␣---␣␣RMS␣closed!");
  }


  /**
   * Write a string record (item) to the record store
   * The item has a special format. It starts with a key word e.g.
   *   - for a key the key word is "Key:" and it is followed by at least
        8 characters
   *   - for an URL the key word is "URL:"
   *   - for a user name the key word is "USR:"
   *   - for a token the key word is "TKN:"
   *   - for an ARL the key word is "ARL:" (URL for authentication server
       )
   * @param itemName Iem keyword
   * @param value Item value
   * @throws Exception In case of RMS operations
   */
  public static void writeRecord(String key, String value)throws
```

```
    Exception{
  String newStr = key + value;
  byte[] rec = newStr.getBytes();
  rs.addRecord(rec, 0, rec.length);
  System.out.println("----␣RMSOperations␣---␣SAVE␣␣\n" + key + "␣-->␣"
       + value);

}


/**
 * Write a byte record to the record store
 *
 * @author Mihai Balan s031288
 *
 * @param key Item identificatiion Key in RMS
 * @param value  Item value
 * @throws Exception In case of RMS operations
 *
 */
public static void writeByteRecord(String key, byte[] value)throws
    Exception{

  // creates the ticket byte[] to be stored in RMS
  // by joining the TKT key and the content
  byte[] returnTKT = new byte[key.getBytes().length + value.length];

  System.arraycopy(key.getBytes(), 0, returnTKT, 0, key.getBytes().
      length);
  System.arraycopy(value, 0, returnTKT, key.getBytes().length, value.
      length);

  rs.addRecord(returnTKT, 0, returnTKT.length);

  System.out.println("----␣RMSOperations␣---␣SAVE␣␣\n" + key + "␣-->␣"
       + value);

}

/**
 * Writes a TicketBean object to the RMS as a byte array
 * by converting the object data to a byte array
 *
 * @author Mihai Balan s031288
 *
 * @param key    The key used to store the Ticket Bean under
 * @param tktBean The ticket bean
```

```
 * @throws IOException
 * @throws RecordStoreException
 */
public static void writeByteItem(String key, TicketBean tktBean) throws
     IOException, RecordStoreException{

  byte[] data = tktBean.getBytes();
  byte[] keyByte = new byte[key.getBytes().length];
  keyByte = key.getBytes();

      byte[] rec = new byte[keyByte.length + data.length];

      System.arraycopy(keyByte, 0, rec, 0, keyByte.length);
      System.arraycopy(data, 0, rec, keyByte.length, data.length);

      rs.addRecord( rec, 0, rec.length );

}


/**
 * Encryptes the given data using the auth key stored in RMS and saves
     the
 * encrypted item into the RMS
 *
 * @author Mihai Balan s031288
 *
 * @param key Item identificatiion Key in RMS
 * @param value Item value
 * @throws Exception In case of RMS operations
 */
public static void writeEncryptedRecord(String key, byte[] value)throws
     Exception{

  String authKey = RMSOperations.getItem("KEY:");

  Encryptor encryptor = new Encryptor(authKey);

  byte[] encryptedData = encryptor.encrypt(value);

  // creates the credit card byte[] to be stored in RMS
  // by joining the CC key and the content
  byte[] returnCreditCard = new byte[key.getBytes().length +
      encryptedData.length];
  System.arraycopy(key.getBytes(), 0, returnCreditCard, 0, key.getBytes
      ().length);
  System.arraycopy(encryptedData, 0, returnCreditCard, key.getBytes().
```

```
      length, encryptedData.length);

  rs.addRecord(returnCreditCard, 0, returnCreditCard.length);
  System.out.println("----␣RMSOperations␣---␣␣SAVE␣␣\n" + new String(
      returnCreditCard));

}


/**
 * Display all records in the record store
 * It returns a vector containing string representation of
 * all records plus their ID in the record store
 *
 * @return Vector Returns a vector with all elements in RMS
 * @throws Exception In case of RMS operations
 */
public static Vector displayRecStore() throws Exception{

  RecordEnumeration re = rs.enumerateRecords(null, null, false);

  // the record store size
  int size = re.numRecords();

  // avetor to store all records in the record store
  Vector rmsRecords = new Vector(size);

  while (re.hasNextElement()) {
    int id = re.nextRecordId ();
    // Get next record
    String str = new String(rs.getRecord(id));
    rmsRecords.addElement("Record␣" + id+ ",\n␣Record␣value␣is:␣" + str
        );
  }
  return rmsRecords;
}


/**
 * Delete all record in the record store that match a certain
 * key(e.g. KEY, TKN, URL or USR, TT, CC).
 * Goes through each record, check if the record
 * matches a certain key and delete it
 * if it begins with the specified key word
 *
 * @author Mihai Balan s031288
 *
```

```
 * @param item Delete the given item
 * @throws Exception In case of RMS operations
 */
public static void deleteItems(String item) throws Exception{

  RecordEnumeration re = rs.enumerateRecords(null, null, false);

  while (re.hasNextElement()) {
    // get the record ID
    int id = re.nextRecordId ();

    // Get next record using the prevuis obtained ID
    String str = new String(rs.getRecord(id));

    // if the record is a KEY then DELETE IT
    // before addding a new key in the record store
    if (str.substring(0,4).equals(item ))
      rs.deleteRecord(id);
  }
  System.out.println("----␣RMSOperations␣---␣DELETE␣\n" + item);
}


/**
 * Search for a particular item in the
 * record store and return it as string
 *
 * @author Mihai Balan s031288
 *
 * @param item The given item to return
 * @return Returns the value of the given item
 * @throws Exception In case of RMS operations
 */
public static String getItem(String key) throws Exception{

  RecordEnumeration re = rs.enumerateRecords(null, null, false);

  while (re.hasNextElement()) {
    // get the record ID
    int id = re.nextRecordId ();

    // Get next record using the prevuis obtained ID
    String str = new String(rs.getRecord(id));

    // if the record is a KEY then return it for encryption
    if (str.substring(0,4).equals(key)){
      System.out.println("----␣RMSOperations␣---␣FOUND␣␣\n" + key + "
```

```
          ␣-->␣" + str);
      return str.substring(4);
    }
  }
  return "";
}


/**
 * Search for a particular key in the record store
 * and return it as byte[]or "".getBytes() else.
 *
 * @author Mihai Balan s031288
 *
 * @param key    The key of the given item to return
 * @return Returns the value of the given key
 * @throws Exception In case of RMS operations
 */
public static byte[] getByteItem( String item) throws Exception{

  RecordEnumeration re = rs.enumerateRecords(null, null, false);

  while (re.hasNextElement()) {
    // get the record ID
    int id = re.nextRecordId ();

    // Get next record using the prevuis obtained ID
    //  String str = new String(rs.getRecord(id));

    // if the record is a KEY then return it for encryption
    if (new String(rs.getRecord(id)).substring(0,4).equals(item)){
      System.out.println("----␣RMSOperations␣---␣RETURNED␣␣\n" + item
           + "␣-->␣" + new String(rs.getRecord(id)));

      byte[] rec = new byte[rs.getRecord(id).length-4];
      System.arraycopy(rs.getRecord(id), 4, rec, 0, rec.length);

      System.out.println("----␣RMSOperations␣---␣RETURNED␣␣\n" + item
           + "␣-->␣" + rec);

      return rec;
    }
  }
  return "".getBytes();
}
```

```
/**
 * Search for a particular encrypted item in the record store
 * and returns the decrypted item as a byte array
 *
 * @author Mihai Balan s031288
 *
 * @param item The key of the item to retrieve from RMS
 * @return Returns the value of the given item
 * @throws Exception In case of RMS operations
 */
public static byte[] getDecryptedItem(String item) throws Exception{

  RecordEnumeration re = rs.enumerateRecords(null, null, false);

  while (re.hasNextElement()) {

    // get the record ID
    int id = re.nextRecordId ();

    // Get next record using the prevuis obtained ID
    byte[] rsItem = rs.getRecord(id);

    // Get the RMS key as a String
    byte[] byteKey = new byte[4];

    System.arraycopy(rsItem, 0, byteKey, 0, 4);

    String strKey = new String(byteKey);

    // if the record is a KEY then return it for encryption
    if (strKey.equals(item)){

      byte[] encCC = new byte[rsItem.length - 4];

      System.arraycopy(rsItem, 4, encCC, 0, rsItem.length - 4);

      // decrypt the credit car data retrieved from the RMS
      String authKey = RMSOperations.getItem("KEY:");
      Encryptor decryptor = new Encryptor(authKey);
      byte[] ccDataDecrypted = decryptor.decrypt(encCC);

      System.out.println("----␣RMSOperations␣---␣FOUND␣␣\n" + item + "
          ␣-->␣" + new String(ccDataDecrypted));

      return ccDataDecrypted;
    }
```

```
  }

  return "".getBytes();
 }


 /**
  * Search how many records that have the given key are stored in RMS.
  * It returns the no of found records.
  *
  * @author Mihai Balan s031288
  *
  * @param item The part of the key to search for in RMS
  *
  * @return The number of found items
  * @throws Exception In case of RMS operations
  */
 public static int getAllItemsLike(String key) throws Exception{

  RecordEnumeration re = rs.enumerateRecords(null, null, false);
  int found = 0;

  while (re.hasNextElement()) {
    // get the record ID
    int id = re.nextRecordId ();

    // Get next record using the prevuis obtained ID
    String str = new String(rs.getRecord(id));

    // if the record is a KEY then return it for encryption
    if (str.substring(0,4).startsWith(key)){
      System.out.println("----␣RMSOperations␣---␣FOUND␣\n" + key + "
          ␣-->␣" + str);
      ++found;
    }

  }

  return found;
 }

 /**
  * Returns all saved tickets in RMS
  *
  * @author Mihai Balan s031288
  *
  * @param rs
```

```
 * @return All tickets saved in RMS
 * @throws IOException
 * @throws Exception
 */
public static TicketBean[] getAllTickets() throws IOException,
    Exception{

  TicketBean tktBean = new TicketBean();
  String noTktStr = RMSOperations.getItem("TTN:");
  int noOfTickets = 0;

  if(!noTktStr.equals("")){
    noOfTickets = Integer.parseInt(noTktStr);
  }

  TicketBean[] tickets = new TicketBean[noOfTickets];
  //RMSOperations.getAllItemsLike("TT");

  for (int i = 0; i < noOfTickets; i++){
    byte[] tktData = RMSOperations.getByteItem("TT" + i +":");

    if(tktData.length != "".getBytes().length){
      tktBean = tktBean.getTKTObject(tktData);
      tickets[i] = tktBean;
    }

  } // end for()

  return tickets;

} // end getAllTickets()


/**
 * Returns all saved credit cards
 * in My Wallet and decrypt the information
 *
 * @author Mihai Balan s031288
 *
 * @param rs
 * @return All credit cards saved in My Wallet
 * @throws IOException
 * @throws CryptoException
 * @throws Exception
 */
public static CreditCardBean[] getAllCreditCards(Encryptor decryptor)
    throws IOException, CryptoException, Exception{
```

```
  CreditCardBean ccBean   = new CreditCardBean();
  int walletCCLength      = Integer.parseInt(RMSOperations.getItem("CCN
      :"));
  CreditCardBean[] walletCC = new CreditCardBean[walletCCLength];
  //RMSOperations.getAllItemsLike("CC");

  for (int i = 1; i <= walletCCLength; i++){
    byte[] decCCData = decryptor.decrypt(RMSOperations.getByteItem("CC"
        + i +":"));

    if(decCCData.length != "".getBytes().length){
      ccBean = ccBean.getCCObject(decCCData);
      walletCC[i-1] = ccBean;
    }

  } // end for()

  return walletCC;

} // end getAllCreditCards()


/**
 * Delete all tickets in RMS (for testing purposes)
 *
 * @author Mihai Balan s031288
 *
 */
public static void resetMyTickets() throws Exception{

  for(int i = 0; i < 11; i++)
    RMSOperations.deleteItems("TT" + i + ":");

} // end resetMyWallet()

/**
 * Reset My Wallet by reseting the PIN code
 * and deleting all credit cards from the wallet.
 *
 * @author Mihai Balan s031288
 *
 */
public static void resetMyWallet() throws Exception{

  RMSOperations.deleteItems("PIN:");
```

```
    RMSOperations.deleteItems("CC1:");
    RMSOperations.deleteItems("CC2:");
    RMSOperations.deleteItems("CC3:");
    RMSOperations.deleteItems("CC4:");
    RMSOperations.deleteItems("CC5:");
    RMSOperations.deleteItems("CC6:");

    MyWalletAuthenticationGUI.pinTrials = 3;

  } // end resetMyWallet()

  /**
   * Delete the record store
   * @throws Exception In case of RMS operations
   */
  public static void deleteRecStore(String recName) throws Exception{
    System.out.println("----␣RMSOperations␣---␣␣Deleting␣RS:␣" + recName)
        ;
    if (RecordStore.listRecordStores() != null)
      RecordStore.deleteRecordStore(recName);
    System.out.println("----␣RMSOperations␣---␣␣RMS␣deleted");
  }


  /**
   * Returns the current record store
   *
   * @return
   */
  public static RecordStore getRecordStore(){
    return rs;

  }

}

package start;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

import model.beans.otherbeans.CreditCardBean;
import model.beans.otherbeans.TicketBean;


import constants.CustomAlertTypes;

import gui.GenericGUI;
```

```
import gui.authentication.AuthenticationGUI;
import gui.customdialogwindows.CanvasAlert;
import gui.mainmenu.MenuScreen;

import gui.splashscreen.SplashScreen;


import rms.RMSOperations;


/**
 * This is the main entry point of the application i.e. the MIDLET
 * A splash screen is displayed for a few seconds, folowed by the
 * authentication screen
 *
 * @author Mihai Balan, s031288
 *
 */
public class Start extends MIDlet implements CommandListener {

  public Display display;

  // stores all tickets saved in the phone memory
  public static TicketBean[] tickets = null;

  // store al credit cards saved in the phone memory
  public static CreditCardBean[] creditCards = null;

  // the no. of tickets saved in RMS till now
  public static int maxTTSaved = 0;

  // the no. of credit cards saved in RMS till now
  public static int maxCCSaved = 0;

  public static String userName  = "";
  public static String userPassword = "";
  public static String userKey   = "";
  public static String walletPin = "";
  public static String emoney    = "";
  public static String themeName = "red";
  public static String themeDir  = "theme_red";

  public static boolean needMovieAlert = false;


  public Start () throws Exception {
    display = Display.getDisplay(this);
```

```
  GenericGUI.display = display;
  MenuScreen.display = display;

}


/**
 * Starts the application by showing the splash screen
 */
public void startApp() {
  try{
    showSplashScreen(display);
    System.out.println("----␣MIDlet␣sarted!␣---␣Splash␣Screen␣Displayed
        ");

  }catch(Exception e){
    CanvasAlert splashError = new CanvasAlert(
        display,
        display.getCurrent(),
        "Error␣in␣the␣Splash␣Screen!",
        "Error␣in␣the␣Splash␣Screen!",
        "error",
        CustomAlertTypes.ALERT_ERROR);
    System.out.println("Exception␣in␣displaying␣the␣Splash␣Screen");
    e.printStackTrace();
  }
}

/**
 * Display the splash screen
 *
 */
public void showSplashScreen(Display display) throws Exception{

  // set up the application start point
  MenuScreen.startingpoint       = this;
  AuthenticationGUI.startingPoint = this;

  // create the next screen that is displayed after the SplashScreen
  AuthenticationGUI authScreen = new AuthenticationGUI();
  Displayable next = authScreen.prepareScreen();

  new SplashScreen(display, next);
}


/**
```

```
   * Pause the MIDlet
   */
  public void pauseApp() {
  }

  /**
   * When Application ends
   */
  public void destroyApp(boolean unconditional) {

    if (RMSOperations.getRecordStore() != null){

      try{
        // write the theme first
        RMSOperations.deleteItems("THM:");
        RMSOperations.writeRecord("THM:", Start.themeName);
        RMSOperations.closeRecStore();

      } catch(Exception e){
        System.out.println("Exception when Closing RMS");
        e.printStackTrace();
      }
    }// end if
  }


  public void commandAction(Command cmd, Displayable item){

  }

}// end class

package tools;

import java.io.*;

import javax.microedition.lcdui.*;


public class ImageProcessing{

  public Image getImage(String image) throws IOException{

    // convert the image to a byte array
    Class c = this.getClass();
    InputStream is = c.getResourceAsStream("/dialogIcons/" + image + ".
        png");
    ByteArrayOutputStream bos = new ByteArrayOutputStream ();
```

```
    byte [] buf = new byte [256];
    while (true) {
      int rd = is.read (buf, 0, 256);
      if (rd == -1) break;
      bos.write (buf, 0, rd);
    }
    buf = bos.toByteArray ();

    // create the image from the byte array
    Image img = Image.createImage(buf, 0, buf.length);
    bos.close();
    is.close();

    return img;
  }

public byte[] getImageAsByteArray(String image) throws IOException{

    // convert the image to a byte array
    Class c = this.getClass();
    InputStream is = c.getResourceAsStream(image + ".png");
    ByteArrayOutputStream bos = new ByteArrayOutputStream ();
    byte [] buf = new byte [256];
    while (true) {
      int rd = is.read (buf, 0, 256);
      if (rd == -1) break;
      bos.write (buf, 0, rd);
    }
    buf = bos.toByteArray ();
    bos.close();
    is.close();

    return buf;

  }
}
```

D.2 Server Side Service

```
package cinemaservice.beans.tools;


import java.io.ByteArrayOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.io.Serializable;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.log4j.*;

import cinemaservice.constants.Error_Code_Constants;
import cinemaservice.constants.Protocol_Step_Constants;
import cinemaservice.exceptions.CinemaServiceException;
import cinemaservice.model.beans.requestBeans.Cancel_Tickets_Req_Bean;
import cinemaservice.model.beans.requestBeans.Change_Password_Req_Bean;
import cinemaservice.model.beans.requestBeans.Cinema_Hall_Conf_Req_Bean;
import cinemaservice.model.beans.requestBeans.Find_Movies_Req_Bean;
import cinemaservice.model.beans.requestBeans.Movie_Details_Req_Bean;
import cinemaservice.model.beans.requestBeans.Purchase_Tickets_Req_Bean;
import cinemaservice.model.beans.requestBeans.Rate_Movie_Req_Bean;
import cinemaservice.model.beans.requestBeans.Reject_Payment_Req_Bean;
import cinemaservice.model.beans.requestBeans.
    Select_Deselect_Seats_Req_Bean;
import cinemaservice.model.beans.responseBeans.
    Background_Cinema_Hall_Conf_Resp_Bean;
import cinemaservice.model.beans.responseBeans.Cinema_Hall_Conf_Resp_Bean
    ;
import cinemaservice.model.beans.responseBeans.Find_Movies_Resp_Bean;
import cinemaservice.model.beans.responseBeans.Movie_Details_Resp_Bean;
import cinemaservice.model.beans.responseBeans.Purchase_Tickets_Resp_Bean
    ;
import cinemaservice.model.beans.responseBeans.Response_Msg_Bean;
import cinemaservice.model.beans.responseBeans.
    Select_Deselect_Seats_Resp_Bean;



/**
 * Java Bean that performs Servlet specific operations e.g.
 *  - obtain the reques bean sent by client
```

```
 *  - send the response bean and status code to the client
 *  - set the Http Resposne ans request
 *  - set the message to the client
 *
 * It follows the Java Bean definition with set...() and get...() methods
 *
 * @author Mihai Balan - s031288
 */
public class Servlet_Operations_Bean implements Serializable{

  // ============================================================================

  //                           PROPERTIES
  // ============================================================================

  private static final long serialVersionUID = 1L;

  /**
   * Initiate a custom logging category
   */
  private static Category cat = Category.getInstance(
      Servlet_Operations_Bean.class.getName());

  /**
   * Client Http request to the Servlet
   */
  private HttpServletRequest request;

  /**
   * Servlet Http Response to the client
   */
  private HttpServletResponse response;

  /**
   * Status Code to be sent to the client
   */
  int httpStatusCode;

  /**
   * Response message to the client as byte[]
   */
  private byte[] responseMsg;


  // ============================================================================

  //                           SET METHODS
```

```
// ============================================================================

/**
 * Set the Client Http Request to the servlet
 *
 * @param request Client Http Request to the servlet
 */
public void setHttpRequest(HttpServletRequest request){
  this.request = request;

} // end setHttpRequest()



/**
 * Set the Servlet Http response to the client
 *
 * @param response Servlet Http response to the client
 */
public void setHttpResponse(HttpServletResponse response){
  this.response = response;

} // end setHttpResponse()



/**
 * Set the Servlet Http status code to the client i.e.
 * 401 USER_NOT_AUTHENTICATED
 * 200 OK
 * 210 USER_AUTHENTICATED
 * 400 ERROR_IN_SQL
 * 410 DATA_NOT_FOUND
 * 406 INVALID_CREDIT_CARD
 * 501 INVALID_PROTOCOL_STEP
 * 417 UNKNOWN_ERROR
 *
 * @param httpStatusCode The Http Servlet Status Code to be sent to the
      Client
 */
public void setHttpStatusCode(int httpStatusCode){
  this.httpStatusCode = httpStatusCode;

} // end setHttpStatusCode()
```

```
/**
 * Set the message(response) to be sent to the client
 *
 * @param responseMsg The response message to the client
 */
public void setResponseMsg(byte[] responseMsg){
  this.responseMsg = responseMsg;

} // end setResponseMsg()



// ============================================================================

//                            GET METHODS
// ============================================================================


/**
 * Retrieve the Request Bean sent by the client function of the
     protocol step,
 * deserialize the retrievede request bean and return it
 */
public Object getClientRequestDataObject() throws IOException,
    CinemaServiceException{

  DataInputStream dis = new DataInputStream(request.getInputStream());

  if (request.getParameter("protocol").equals(Protocol_Step_Constants.
      PRT_STEP_MOVIE_DETAILS)){

    Movie_Details_Req_Bean movBean = null;
    cat.debug("CGP␣before␣deserialize␣movBean");
    // get the Movie_Details_Request_Bean
    movBean = Movie_Details_Req_Bean.readBean(dis);
    cat.debug("CGP␣after␣␣deserialize␣movBean");
    cat.debug(movBean.toString());
    return movBean;

  } else if (request.getParameter("protocol").equals(
      Protocol_Step_Constants.PRT_STEP_FIND_MOVIES)){

    Find_Movies_Req_Bean findMoviesReqBean = null;
    cat.debug("MOV␣before␣deserialize␣findMoviesReqBean");
    // get the findMoviesReqBean
    findMoviesReqBean = findMoviesReqBean.readBean(dis);
    cat.debug("MOV␣after␣␣deserialize␣findMoviesReqBean");
```

```
  cat.debug(findMoviesReqBean.toString());
  return findMoviesReqBean;

} else if (request.getParameter("protocol").equals(
    Protocol_Step_Constants.PRT_STEP_BACKGROUND_CINEMA_HALL_UPDATE)){

  Cinema_Hall_Conf_Req_Bean cinHallConfReqBean = null;
  cat.debug("SHW␣before␣deserialize␣cinHallConfReqBean");
  // get the Cinema_Hall_Conf_Req_Bean
  cinHallConfReqBean = Cinema_Hall_Conf_Req_Bean.readBean(dis);
  cat.debug("SHW␣after␣␣deserialize␣cinHallConfReqBean");
  cat.debug(cinHallConfReqBean.toString());
  return cinHallConfReqBean;

} else if (request.getParameter("protocol").equals(
    Protocol_Step_Constants.PRT_STEP_SELECT_DESELECT_SEATS)){

  Select_Deselect_Seats_Req_Bean selDeselSeatsReqBean = null;
  cat.debug("SHW␣before␣deserialize␣Select_Deselect_Seats_Req_Bean");
  // get the Select_Deselect_Seats_Req_Bean
  selDeselSeatsReqBean = Select_Deselect_Seats_Req_Bean.readBean(dis)
      ;
  cat.debug("SHW␣after␣␣deserialize␣Select_Deselect_Seats_Req_Bean");
  cat.debug(selDeselSeatsReqBean.toString());
  return selDeselSeatsReqBean;

}else if (request.getParameter("protocol").equals(
    Protocol_Step_Constants.
    PRT_STEP_SELECT_SHOW_AND_DISPLAY_CINEMA_HALL_CONF)){

  Cinema_Hall_Conf_Req_Bean cinHallConfReqBean = null;
  cat.debug("SHW␣before␣deserialize␣cinHallConfReqBean");
  // get the Cinema_Hall_Conf_Req_Bean
  cinHallConfReqBean = Cinema_Hall_Conf_Req_Bean.readBean(dis);
  cat.debug("SHW␣after␣␣deserialize␣cinHallConfReqBean");
  cat.debug(cinHallConfReqBean.toString());
  return cinHallConfReqBean;

} else if (request.getParameter("protocol").equals(
    Protocol_Step_Constants.PRT_STEP_REJECT_PAYMENT)){

  Reject_Payment_Req_Bean rejectReservationBean = null;
  cat.debug("SHW␣before␣deserialize␣Reject_Reservation_Req_Bean");
  // get the Reject_Reservation_Req_Bean
  rejectReservationBean = Reject_Payment_Req_Bean.readBean(dis);
  cat.debug("REJ␣after␣␣deserialize␣Reject_Reservation_Req_Bean");
  cat.debug(rejectReservationBean.toString());
```

```
    return rejectReservationBean;

  } else if (request.getParameter("protocol").equals(
      Protocol_Step_Constants.PRT_STEP_PURCHASE_TICKETS)){

    Purchase_Tickets_Req_Bean purchaseTicketReqBean = null;
    cat.debug("PTC␣before␣deserialize␣Purchase_Tickets_Req_Bean");
    // get the Reject_Reservation_Req_Bean
    purchaseTicketReqBean = Purchase_Tickets_Req_Bean.readBean(dis);
    cat.debug("PTC␣after␣␣deserialize␣Purchase_Tickets_Req_Bean");
    cat.debug(purchaseTicketReqBean.toString());

    return purchaseTicketReqBean;

  } else if (request.getParameter("protocol").equals(
      Protocol_Step_Constants.PRT_STEP_CANCEL_TICKETS)){

    Cancel_Tickets_Req_Bean cancelTicketReqBean = null;
    cat.debug("CCT␣before␣deserialize␣Cancel_Tickets_Req_Bean");
    // get the Cancel_Tickets_Req_Bean
    cancelTicketReqBean = Cancel_Tickets_Req_Bean.readBean(dis);
    cat.debug("CCT␣after␣␣deserialize␣Cancel_Tickets_Req_Bean");
    cat.debug(cancelTicketReqBean.toString());

    return cancelTicketReqBean;

  } else if (request.getParameter("protocol").equals(
      Protocol_Step_Constants.PRT_STEP_RATE_MOVIE)){

    Rate_Movie_Req_Bean rateMovieBean = null;
    cat.debug("RTM␣before␣deserialize␣rateMovieBean");
    // get the Rate_Movie_Req_Bean
    rateMovieBean = Rate_Movie_Req_Bean.readBean(dis);
    cat.debug("RTM␣after␣␣deserialize␣rateMovieBean");
    cat.debug(rateMovieBean.toString());

    return rateMovieBean;

  } else if (request.getParameter("protocol").equals(
      Protocol_Step_Constants.PRT_STEP_CHANGE_PASSWORD)){

    Change_Password_Req_Bean chgPasswd = null;
    cat.debug("CGP␣before␣deserialize␣chgPswdBean");
    // get the Change_Password_Req_Bean
    chgPasswd = Change_Password_Req_Bean.readBean(dis);
    cat.debug("CGP␣after␣␣deserialize␣chgPswdBean");
```

```
    cat.debug(chgPasswd.toString());

    return chgPasswd;

  }else {
    throw new CinemaServiceException(
        "INVALID␣PROTOCOL␣STEP␣sent␣by␣the␣MIDLET",
        request.getParameter("protocol"),
        "Servlet_Operations_Bean",
        "getClientRequestDataObject()",
        "1",
        Error_Code_Constants.INVALID_PROTOCOL_STEP);

  } // end if (protocol step)

} // end getClientRequestDataObject()



/**
 * Send the response bean to the MIDlet.
 * The bean contains the result of the SQL query ran against the DB.
 * The bean has bean built in the @see getClientRequestDataObject()
 *
 * @param bean The response bean that is to be sent to the MIDlet
 * @throws IOException
 */
public void sendResponseBeanToMidlet(Object bean) throws IOException,
    CinemaServiceException{

  if (bean == null){
    response.setStatus(Error_Code_Constants.INVALID_RESPONSE_BEAN);
  }
  else {
    response.setStatus(Error_Code_Constants.OK);
    response.setContentType("application/octet-stream");
    int contentLength = serializeResponseBean(response.getOutputStream
        (), bean);
    response.setContentLength(contentLength);
  }

} // end sendResponseToMidlet(Object object)


/**
 * Serialize the content of the response bean to the client
 * function of the protocol step
```

```
 *
 * @param in  Input  Stream obtained from the Servlet Request
 * @param out Output Stream obtained from the Servlet Response
 * @param bean The Response Bean that is to be sent to the MIDlet
 *
 * @return The number of bytes written to the network
 *         to set up the content-length parameter
 *
 * @throws IOException
 */
public int serializeResponseBean(OutputStream out, Object bean) throws
    IOException, CinemaServiceException{

  DataOutputStream dos = new DataOutputStream(out);

  ByteArrayOutputStream baos = new ByteArrayOutputStream();
  DataOutputStream dosOfBaos = new DataOutputStream(baos);

  if (request.getParameter("protocol").equals(Protocol_Step_Constants.
      PRT_STEP_MOVIE_DETAILS)){
    // serialize the Movie_Details_Resp_Bean
    ((Movie_Details_Resp_Bean)bean).writeBean(dosOfBaos);

  }else if (request.getParameter("protocol").equals(
      Protocol_Step_Constants.PRT_STEP_FIND_MOVIES)){
    // serialize the Find_Movies_Resp_Bean
    ((Find_Movies_Resp_Bean)bean).writeBean(dosOfBaos);

  }else if (request.getParameter("protocol").equals(
      Protocol_Step_Constants.
      PRT_STEP_SELECT_SHOW_AND_DISPLAY_CINEMA_HALL_CONF)){
    // serialize the Cinema_Hall_Conf_Resp_Bean
    ((Cinema_Hall_Conf_Resp_Bean)bean).writeBean(dosOfBaos);

  }else if (request.getParameter("protocol").equals(
      Protocol_Step_Constants.PRT_STEP_BACKGROUND_CINEMA_HALL_UPDATE)){
    // serialize the Background_Cinema_Hall_Conf_Resp_Bean
    ((Background_Cinema_Hall_Conf_Resp_Bean)bean).writeBean(dosOfBaos);

  }else if (request.getParameter("protocol").equals(
      Protocol_Step_Constants.PRT_STEP_SELECT_DESELECT_SEATS)){
    // serialize the Select_Deselect_Seats_Resp_Bean
    ((Select_Deselect_Seats_Resp_Bean)bean).writeBean(dosOfBaos);

  }else if (request.getParameter("protocol").equals(
      Protocol_Step_Constants.PRT_STEP_REJECT_PAYMENT)){
    // serialize the Response_Msg_Bean
```

```
      ((Response_Msg_Bean)bean).writeBean(dosOfBaos);

    }else if (request.getParameter("protocol").equals(
        Protocol_Step_Constants.PRT_STEP_PURCHASE_TICKETS)){
      // serialize the Purchase_Tickets_Resp_Bean
      ((Purchase_Tickets_Resp_Bean)bean).writeBean(dosOfBaos);

    }else if (request.getParameter("protocol").equals(
        Protocol_Step_Constants.PRT_STEP_CANCEL_TICKETS)){
      // serialize the Response_Msg_Bean
      ((Response_Msg_Bean)bean).writeBean(dosOfBaos);

    }else if (request.getParameter("protocol").equals(
        Protocol_Step_Constants.PRT_STEP_RATE_MOVIE)){
      // serialize the Response_Msg_Bean
      ((Response_Msg_Bean)bean).writeBean(dosOfBaos);

    }else if (request.getParameter("protocol").equals(
        Protocol_Step_Constants.PRT_STEP_CHANGE_PASSWORD)){
      // serialize the Response_Msg_Bean
      ((Response_Msg_Bean)bean).writeBean(dosOfBaos);

    }else {
      throw new CinemaServiceException(
          "INVALID␣PROTOCOL␣STEP␣sent␣by␣the␣MIDLET",
          request.getParameter("protocol"),
          "Servlet_Operations_Bean",
          "serializeResponseBean()",
          "4",
          Error_Code_Constants.INVALID_PROTOCOL_STEP);

    } // end if (protocol_step)

    // flush and write the beans
    dosOfBaos.flush();
    byte[] bufferedBytes = baos.toByteArray();
    dos.write(bufferedBytes, 0, bufferedBytes.length);

    return dos.size();

  }  // end serializeResponseBean()

} // end class

package cinemaservice.beans.tools;


import java.io.Serializable;
```

```
import java.text.ParseException;
import java.util.Vector;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import javax.sql.DataSource;

import org.apache.log4j.*;

import cinemaservice.exceptions.CinemaServiceException;
import cinemaservice.model.facade.ClientToFacade;


/**
 * Java Bean that performs SQL specific operations e.g.
 *
 *  - setting the pooled connection from the pool of connections,
 *  - setting the SQL statement to be executed,
 *  - setting the parameter list for the SQL statement that is to be
     executed,
 *  - setting the parameter values for the SQL statement that is to be
     executed,
 *  - getting the pooled connection,
 *  - executing an SQl statement and getting the result of the SQL
     execution
 *
 * It follows the Java Bean definition with set...() and get...() methods
 *
 * @author Mihai Balan - s031288
 */
public class SQL_Operations_Bean implements Serializable{


  // ============================================================================

  //                           PROPERTIES
  // ============================================================================

  private static final long serialVersionUID = 1L;

  /**
   * Initiate a custom logging category
   */
  private static Category cat = Category.getInstance(SQL_Operations_Bean.
```

```
    class.getName());

/**
 * The data source used to get the pooled connection from
 */
private DataSource datasource;

/**
 * The prepared statemnt to be executed
 */
private String   sqlStmt;

/**
 * The request bean containing all parameters needed
 * for executing the prepared statement
 */
private Object   reqBean ;

/**
 * The string identifier for the protocol step for the
 * client-server communication
 */
private String prtStep = "";



// ============================================================================

//                            SET METHODS
// ============================================================================

/**
 * Set the data source to the one configured in Tomcat
 * to get the connection pool
 *
 * @param datasource The datasource configured in Tomcat
 *                   for obtaining the connection pool
 */
public void setPooledSource(DataSource datasource){
  this.datasource = datasource;

} // end setPooledSource()


/**
 * Set the SQL statement text, that is to be executed
 *
```

```
 * @param stmt The SQL statemnt to be executed for each case
 */
public void setSQLStatement(String sqlStmt){
  this.sqlStmt = sqlStmt;

} // end setSQLStatement()


/**
 * Set the request bean to extract the parameter list
 * for the prepared pqSQL statement that is to be executed.
 * It also sets the protocl step used in the facade call.
 *
 * @param reqBean The request bean
 * @param prtStep The protocol step
 */
public void setSQLParameters(Object reqBean, String prtStep){
  this.reqBean = reqBean;
  this.prtStep = prtStep;

} // end setSQLParameters()


/**
 * Set the list of parameter values for the prepared pqSQL statement
 * to be executed by using the previously set parameter list
 *
 * @param pqPsqlStmt The prepared pqSQL statement to be executed
 * @param sqlParam The list of parameters for thre prepared statemnt
 * @return
 *
 * @throws SQLException
 * @throws CinemaServiceException
 * @throws ParseException
 */
public void setSQLParameterValues(PreparedStatement pqPsqlStmt, Object
    reqBean, String prtStep) throws SQLException,
    CinemaServiceException, ParseException{

  ClientToFacade client = new ClientToFacade();
  client.getReqBeanData(pqPsqlStmt, reqBean, prtStep);

} // end setSQLParameterValues()


// ============================================================================
```

```
//                          GET METHODS
// ============================================================================

/**
 * Get a connection from the pool of connections
 *
 * @return A connection from the pool
 *
 * @throws SQLException
 */
public Connection getPooledConnection() throws SQLException{
  return datasource.getConnection();

} // end getPooledConnectio()


/**
 * Execute the given SQL prepared statement with the given
 * list of parameter values and returns the result of
 * the SQL execution
 *
 * @return The result of the SQL statement execution
 *
 * @throws SQLException
 * @throws CinemaServiceException
 * @throws ParseException
 */
public Vector executeSQL() throws SQLException, CinemaServiceException
    , ParseException{

  Vector sqlResult = new Vector();

  Connection conn = null;
  PreparedStatement pgPsqlStmt = null;
  ResultSet rs = null;

  try{
    // get the connection from the pool
    conn = getPooledConnection();

    // prepare the sql statemt to be executed
    pgPsqlStmt = conn.prepareStatement(sqlStmt);

    // set the parameter values for the prepared statement
    setSQLParameterValues(pgPsqlStmt, reqBean, prtStep);

    // execute the prepared statement
```

```
    rs = pgPsqlStmt.executeQuery();

    // get the result from the result set as a String
    ResultSetMetaData dbMeta = rs.getMetaData();
    while (rs.next()) {
      for (int col=0; col < dbMeta.getColumnCount(); col++) {
        sqlResult.addElement(rs.getObject(col+1));

      }
    }


    //String showLocationID = sqlParameters[0];
    //String sqlResult = "";
    //String sqlSelect = "SELECT * FROM cinema.Movie_Details(" +
        showLocationID + ");";

    // close the record set and the prepared statment
    rs.close();
    pgPsqlStmt.close();

    // return the connection to the pool of connections
    conn.close();

  }catch (SQLException e){
    System.out.println("In␣SQL␣Operations␣Bean␣executeSQL" + e.
        getMessage());
    e.printStackTrace();
  }
  // perform any clean up in case any connection, statement remains
      opened and not used
  finally {
    try {
      if (rs != null && !conn.isClosed())
        rs.close();

    } catch (SQLException sqle1) {
      cat.warn("SQL␣exception␣" +
          "when␣trying␣to␣close␣the␣record␣set␣in␣FINALLY␣clause␣...␣"
          + sqle1.getMessage());
    }
    try {
      if (pgPsqlStmt != null && !conn.isClosed())
        pgPsqlStmt.close();

    } catch (SQLException sqle2) {
      cat.warn("SQL␣exception␣" +
```

```
            "when␣trying␣to␣close␣the␣statement␣in␣FINALLY␣clause␣...␣"
            + sqle2.getMessage());
      }
      try {
        if (!conn.isClosed())
          conn.close();

      } catch (SQLException sqle3) {
        cat.warn("SQL␣exception␣" +
            "when␣trying␣to␣close␣the␣connection␣in␣FINALLY␣clause␣...␣"
            + sqle3.getMessage());
      }
    }//  end try - catch - finally

    return sqlResult;

  } // end executeSQL()

} // end class

package cinemaservice.constants;

/**
 * Declare general constants used for parsing the
 * result of an SQL statement.
 *
 * @author Mihai Balan - s031288
 *
 */
public final class Parsing_Constants {

  /**
   * New Line separator function of the OS
   */
  public static final String NEW_LINE = System.getProperty("line.
      separator");

  /**
   * Credit Card payment method
   */
  public static final String CARD_PAYMENT = "CARD";

  /**
   * Cinema payment method
   */
  public static final String CINEMA_PAYMENT = "CINEMA";

} // end class
```

```
package cinemaservice.exceptions;

import java.util.Date;

import cinemaservice.constants.Parsing_Constants;

/**
 * Defines an user defined exception characteristics for the
 * Cineama Service. It depicts the exception msg, class/method/position
 * where the exception is thrown, and the value accordingly
 * to @see cinemaservice.constants.Error_Code_Constants
 *
 * @author Mihai Balan - s031288
 *
 */
public class CinemaServiceException extends Exception {

  private static final long serialVersionUID = 1L;

  /**
   * The Exception desription
   */
  private String description;

  /**
   * The extra field for exception description
   */
  private String msgValue;

  /**
   * The class where the exception is thrown
   */
  private String className;

  /**
   * The methos where the exception is thrown
   */
  private String methodName;

  /**
   * The position where the exception is thrown
   */
  private String position;

  /**
   * The value of the exception accordingly to
   * @see cinemaservice.constants.Error_Code_Constants
```

```
 */
private int   value;

/**
 * General exception for which a CinemaService Exception is thrown in
     the
 * try-catch block of that exception
 */
private Exception e = null;


/**
 * Constructs a CinemaServiceException using all previoulsy mentioned
     fields
 *
 * @param description Exception description
 * @param msgValue  Exception extra description
 * @param className The class where the exception is thrown
 * @param methodName The method where the exception is thrown
 * @param position  The position where the exception is thrown
 * @param value     Exception value accordingly to @see
 *                  cinemaservice.constants.Error_Code_Constants
 */
public CinemaServiceException(String description, String msgValue,
    String className, String methodName, String position,
    int value) {

  this.description = description;
  this.msgValue   = msgValue;
  this.className  = className;
  this.methodName = methodName;
  this.value      = value;
  this.position   = position;

} // end constructor

/**
 * Cinema Service Exception built from the General Exception
 *
 * @param e General exception for which a CinemaService Exception
 *         is thrown in the try-catch block of that exception
 */
public CinemaServiceException(Exception e){
  this.e = e;
}
```

```
/**
 * Returns the value of the exception accordingly to
 * @see cinemaservice.constants.Error_Code_Constants
 *
 * @return The value of the exception accordingly to
 *         @see cinemaservice.constants.Error_Code_Constants
 */
public int getValue(){
  return this.getValue();

} // end getValue()


/**
 * Returns the General Exception used to build the
 * CinemaServiceException
 *
 * @return The General Exception used to build the
 *        CinemaServiceException
 */
public Exception getException(){
  return e;
}
/**
 * Return the Exception Detailed Description
 *
 * @return Exception Detailed Description
 */
public String getMessage(){
  String exceptionText = Parsing_Constants.NEW_LINE
  + "CINEMA_SERVICE_Exception␣on␣" + new Date() + Parsing_Constants.
      NEW_LINE
  + this.description + "␣:␣" + this.msgValue + Parsing_Constants.
      NEW_LINE
  + "at␣" + this.className + "." + this.methodName
  + ",␣Position:␣" + this.position          + Parsing_Constants.NEW_LINE
      ;

  if (e == null){
    return exceptionText;}
  else {
    return e.toString() + ":␣" + e.getMessage();
  }

} // end print getMessage()
```

```
  /**
   * Print the Exception Detailed Description
   *
   */
  public void printMessage(){
    System.out.println(this.getMessage());

  } // end printMessage()

} // end class

package cinemaservice.model.facade;

import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.text.ParseException;

import cinemaservice.constants.Error_Code_Constants;
import cinemaservice.constants.Protocol_Step_Constants;
import cinemaservice.exceptions.CinemaServiceException;
import cinemaservice.model.beans.requestBeans.Authentication_1_Req_Bean;
import cinemaservice.model.beans.requestBeans.Cancel_Tickets_Req_Bean;
import cinemaservice.model.beans.requestBeans.Change_Password_Req_Bean;
import cinemaservice.model.beans.requestBeans.Cinema_Hall_Conf_Req_Bean;
import cinemaservice.model.beans.requestBeans.Find_Movies_Req_Bean;
import cinemaservice.model.beans.requestBeans.Movie_Details_Req_Bean;
import cinemaservice.model.beans.requestBeans.Purchase_Tickets_Req_Bean;
import cinemaservice.model.beans.requestBeans.Rate_Movie_Req_Bean;
import cinemaservice.model.beans.requestBeans.Reject_Payment_Req_Bean;
import cinemaservice.model.beans.requestBeans.
    Select_Deselect_Seats_Req_Bean;

import cinemaservice.model.interfaces.ClientToFacadeInterface;


/**
 * This is the First FACADE of the Cinema Service Model. It hides the
     complexity
 * of the Request - Response Model from the client and provides methods
 * for accessing the data contained in the Request Java Beans function
 * of the protocl step sent by the client to the server
 *
 * It delegates the functionality further to the Facade To Model.
 *
 * It implements the FACADE DESIGN PATTERN.
 *
 * @author Mihai Balan - s031288
 *
```

```
 */
public class ClientToFacade implements ClientToFacadeInterface{


  /**
   * Delegates the parameter setting for the preapared sql statement to
   * the Facade To Model. Functions of the reuest parameters in the
   * request data bean, the parameters of theprepared statement are set
   *
   * @param pqPsqlStmt The preapared SQL Statement that is to be executed
   * @param reqBean  The request bean
   * @param prtStep  The protocol Step
   * @throws SQLException
   * @throws CinemaServiceException
   * @throws ParseException
   */
  public void getReqBeanData(PreparedStatement pqPsqlStmt, Object reqBean
      , String prtStep) throws SQLException, CinemaServiceException,
      ParseException{

    FacadeToModel facade = new FacadeToModel();

    // beginning of the AUTHENTICATION procedure
    if (prtStep.equals(Protocol_Step_Constants.PRT_STEP_AUTHENTICATION_1)
        )
      facade.getAuth1ReqBeanData(pqPsqlStmt, (Authentication_1_Req_Bean)
          reqBean);

    //  2nd step in the AUTHENTICATION procedure
    else if (prtStep.equals(Protocol_Step_Constants.
        PRT_STEP_AUTHENTICATION_2))
      facade.getSelectDeselectSeatsReqBeanData(pqPsqlStmt, (
          Select_Deselect_Seats_Req_Bean)reqBean);

    // CHANGE PASSWORD step
    else if (prtStep.equals(Protocol_Step_Constants.
        PRT_STEP_CHANGE_PASSWORD))
      facade.getChangePasswordReqBeanData(pqPsqlStmt, (
          Change_Password_Req_Bean)reqBean);

    // FIND MOVIES step
    else if (prtStep.equals(Protocol_Step_Constants.PRT_STEP_FIND_MOVIES)
        )
      facade.getFindMoviesReqBeanData(pqPsqlStmt, (Find_Movies_Req_Bean)
          reqBean);

    // SELECT SHOW AND DISPLAY CINEMA HALL CONFIGURATION step
```

```
else if (prtStep.equals(Protocol_Step_Constants.
    PRT_STEP_SELECT_SHOW_AND_DISPLAY_CINEMA_HALL_CONF))
  facade.getCinemaHallConfReqBeanData(pqPsqlStmt, (
      Cinema_Hall_Conf_Req_Bean)reqBean);

// BACKGROUND DISPLAY CINEMA HALL UPDATE step
else if (prtStep.equals(Protocol_Step_Constants.
    PRT_STEP_BACKGROUND_CINEMA_HALL_UPDATE))
  facade.getCinemaHallConfReqBeanData(pqPsqlStmt, (
      Cinema_Hall_Conf_Req_Bean)reqBean);

// SELECT - DESELECT SEATS step
else if (prtStep.equals(Protocol_Step_Constants.
    PRT_STEP_SELECT_DESELECT_SEATS))
  facade.getSelectDeselectSeatsReqBeanData(pqPsqlStmt, (
      Select_Deselect_Seats_Req_Bean)reqBean);

// PURCHASE TICKETS step
else if (prtStep.equals(Protocol_Step_Constants.
    PRT_STEP_PURCHASE_TICKETS))
  facade.getPurchaseTicketsReqBeanData(pqPsqlStmt, (
      Purchase_Tickets_Req_Bean)reqBean);

// CANCEL RESERVATION/TICKETS step
else if (prtStep.equals(Protocol_Step_Constants.
    PRT_STEP_CANCEL_TICKETS))
  facade.getCancelTicketsReqBeanData(pqPsqlStmt, (
      Cancel_Tickets_Req_Bean)reqBean);

// REJECT PAYMENT step
else if (prtStep.equals(Protocol_Step_Constants.
    PRT_STEP_REJECT_PAYMENT))
  facade.getRejectPaymentReqBeanData(pqPsqlStmt, (
      Reject_Payment_Req_Bean)reqBean);

// RATE MOVIE step
else if (prtStep.equals(Protocol_Step_Constants.PRT_STEP_RATE_MOVIE))
  facade.getRateMovieReqBeanData(pqPsqlStmt, (Rate_Movie_Req_Bean)
      reqBean);

// MOVIE DETAILS step
else if (prtStep.equals(Protocol_Step_Constants.
    PRT_STEP_MOVIE_DETAILS))
  facade.getMovieDetailsReqBeanData(pqPsqlStmt, (
      Movie_Details_Req_Bean)reqBean);

//if unknown protocol step send the RESPONSE_CODE to the client
```

```
    // as INVALID_PROTOCOL_STEP and throw corresponding Exception on the
        server side
    else {
      throw new CinemaServiceException(
          "INVALID␣PROTOCOL␣STEP␣sent␣by␣the␣MIDLET",
          prtStep,
          "ClientToFacade",
          "getReqBeanData()",
          "1",
          Error_Code_Constants.INVALID_PROTOCOL_STEP);

    } // end if (protocol step)

  } // endgetReqBeanData()

} // end class

package cinemaservice.model.facade;

import java.sql.SQLException;
import java.text.ParseException;
import java.util.Vector;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;

import cinemaservice.beans.tools.SQL_Operations_Bean;
import cinemaservice.constants.Protocol_Step_Constants;
import cinemaservice.exceptions.CinemaServiceException;
import cinemaservice.model.beans.requestBeans.Cancel_Tickets_Req_Bean;
import cinemaservice.model.beans.requestBeans.Purchase_Tickets_Req_Bean;
import cinemaservice.model.beans.responseBeans.Response_Msg_Bean;

/**
 * This class is used perform some query against the Cinema DB
 * in special conditions e.g. Cancel tickets when an error occurs in
 * the ONLINE PAYMENT SERVICE
 *
 *
 * @author Mihai Balan - s031288
 *
 */
public class DBTools {

  /**
   * Cancel previously purchased tickets in case the
```

```
 * ONLINE PAYMENT SYSTEM fails when making the payment
 * for the tickets.
 * <pre>
 * @param reqBean       The Purchase Tickets Request Bean containing
 *                      the credit card information
 *
 * @param sqlResult      The result from PurchaseTickets stored
     procedure
 * @param noOfTickets    No of purchased tickets
 * @param reservationID_Str Reservation ID for the purchased tickets
 * @param ticketIDs      Ticket ID's for the purchased tickets
 *
 * @return A response Msg bean containing the result of cancelling the
     tickets
 *
 * @throws NamingException
 * @throws SQLException
 * @throws ParseException
 * @throws CinemaServiceException
 * </pre>
 */
public Response_Msg_Bean cancelTicketsIfPaymentFails(
    Purchase_Tickets_Req_Bean reqBean,
   Vector sqlResult, int noOfTickets, String reservationID_Str,
   String[] ticketIDs) throws NamingException, SQLException,
               ParseException, CinemaServiceException{

  // Create a datasource for pooled connections.
  // Use JNDI to retrieve the DataSource object defined
  // in the Tomcat and application configuration *.xml files
  // i.e.WEB_INF/web.xml and META_INF/context.xml in the directory
  // where the application is deployed on Tomcat
  Context initCtx = new InitialContext();
  Context envCtx = (Context) initCtx.lookup("java:comp/env");
  DataSource datasource = (DataSource) envCtx.lookup("jdbc/postgres");

  SQL_Operations_Bean sqlOpBean = new SQL_Operations_Bean();

  // create the Cancel_Tickets_Req_Bean using the date from
  // Purchase_Tickets_Req_Bean in order to cancel the previously
  // reserved tickets
  Cancel_Tickets_Req_Bean cancelTickets = new Cancel_Tickets_Req_Bean()
      ;
  cancelTickets.setUserName(reqBean.getUserName());
  cancelTickets.setPassword(reqBean.getPassword());
  //cancelTickets.setPassword("".getBytes());
  cancelTickets.setNoOfTickets(noOfTickets);
```

```
    cancelTickets.setReservationID(reservationID_Str);
    cancelTickets.setTicketID(ticketIDs);

    sqlOpBean.setPooledSource(datasource);

    sqlOpBean.setSQLStatement("SELECT␣*␣FROM␣cinema.Cancel_Tickets
        (?,␣?,␣?,␣?)");
    System.out.println("------␣CANCELLL␣SQL␣Statement␣set:\n");

    sqlOpBean.setSQLParameters(cancelTickets, Protocol_Step_Constants.
        PRT_STEP_CANCEL_TICKETS);
    System.out.println("------␣CANCELLL␣SQL␣Parameters␣Set:\n");

    // the result of cancelling the tickets
    Vector cancelSQLResult = sqlOpBean.executeSQL();

    System.out.println("------␣CANCELLL␣SQL␣Result:\n" + cancelSQLResult)
        ;
    FacadeToModel facade = new FacadeToModel();
    Response_Msg_Bean rspMsgBean = facade.setResponseCancelTicketsBean(
        cancelSQLResult);
    System.out.println("------␣CANCELLL␣Respose␣code:\n" + rspMsgBean.
        getResponseCode());
    return rspMsgBean;
  }

}

package cinemaservice.model.facade;

import org.bouncycastle.crypto.*;
import org.bouncycastle.crypto.digests.SHA1Digest;
import org.bouncycastle.crypto.engines.*;
import org.bouncycastle.crypto.generators.PKCS12ParametersGenerator;
import org.bouncycastle.crypto.modes.*;
import org.bouncycastle.crypto.paddings.PaddedBufferedBlockCipher;
import org.bouncycastle.crypto.params.*;

/**
 * This is a helper class that performs encryption decryption operations
 * using Bouncy Catle
 *
 * @author Mihai Balan, Wojciech Dobrowolski
 *
 */
public class Encryptor {

  private PaddedBufferedBlockCipher cipher;
```

```
private KeyParameter key;

/**
 * Creates an encryptor used to encrypt/decrypt a message
 * witha given key
 *
 * @param key The key to encrypt/decrypt the message
 */
public Encryptor(byte[] key){
  cipher = new PaddedBufferedBlockCipher(
      new CFBBlockCipher(
          new DESEngine(),8));
  this.key = new KeyParameter (key);
}

/**
 *  Initialize the cryptographic engine.
 *  The string should be at least 8 chars long.
 */
public Encryptor( String key ){
  this( key.getBytes() );
}


/**
 * Private routine that does the gritty work.
 */
private byte[] callCipher( byte[] data )
throws CryptoException {
  System.out.println("Cipher␣has␣been␣called␣...");
  int size = cipher.getOutputSize( data.length );
  byte[] result = new byte[ size ];
  int olen = cipher.processBytes( data, 0, data.length, result, 0 );
  olen += cipher.doFinal( result, olen );

  if( olen < size ){
    byte[] tmp = new byte[ olen ];
    System.arraycopy( result, 0, tmp, 0, olen );
    result = tmp;
  }

  return result;
}

/**
 * Encrypt arbitrary byte array, returning the
 * encrypted data in a different byte array.
```

```
 *
 * @param data Data to be encrypted as byte array
 *
 * @return Returns the encrypted data as byte array
 */
public synchronized byte[] encrypt( byte[] data )
throws CryptoException {
  if( data == null || data.length == 0 ){
    return new byte[0];
  }

  cipher.init( true, key );
  return callCipher( data );
}

/**
 *  Encrypts a string.
 *
 *  @param data Data to be encrypted as string
 *
 *  @return Returns the encrypted data as byte array
 */

public byte[] encryptString( String data )
throws CryptoException {
  if( data == null || data.length() == 0 ){
    return new byte[0];
  }

  return encrypt( data.getBytes() );
}

/**
 * Decrypts arbitrary data.
 *
 * @param data Data to be decrypted as byte array
 *
 * @return Returns the decrypted data as byte array
 */
public synchronized byte[] decrypt( byte[] data )
throws CryptoException {
  if( data == null || data.length == 0 ){
    System.out.println("Data␣turned␣out␣to␣be␣null␣...");
    return new byte[0];
  }
  System.out.println("kupa");
  cipher.init( false, key );
```

```
  System.out.println("Decrypted␣data:␣" );
  return callCipher( data );
}

/**
 * Decrypts a string that was previously encoded
 * using encryptString.
 *
 * @param data Data to be decrypted as byte array
 *
 * @return Returns the decrypted data as string
 */
public String decryptString( byte[] data )
throws CryptoException {
  if( data == null || data.length == 0 ){
    System.out.println("Something␣happened␣with␣the␣data␣...");
    return "";
  }
  System.out.println("Decrypting␣data␣..." + data.toString());
  return new String( decrypt( data ));
}

public ParametersWithIV createKey(String salt, String key){
  PBEParametersGenerator generator =
    new PKCS12ParametersGenerator(new SHA1Digest());
  generator.init(
      PBEParametersGenerator.PKCS12PasswordToBytes(key.toCharArray()),
      salt.getBytes(), 1024);
  // Generate a 128 bit key w/ 128 bit IV
  ParametersWithIV ret =
    (ParametersWithIV)generator.generateDerivedParameters(128, 128);
  return ret;
}


  public byte[] encryptWithAES(ParametersWithIV key, byte[] msg){
  BufferedBlockCipher cipher = new PaddedBufferedBlockCipher(
      new CBCBlockCipher(new AESFastEngine()));
  cipher.init(true, key);
  byte[] result = new byte[cipher.getOutputSize(msg.length)];
  int len = cipher.processBytes(msg, 0,
      msg.length, result, 0);
  try {
    cipher.doFinal(result, len);
  } catch (CryptoException ce) {
    System.out.println("Encryption␣with␣AES␣error␣...");
    ce.printStackTrace();
```

```
    }
    return result;
  }

  public byte[] decryptWithAES(ParametersWithIV key, byte[] result){
    BufferedBlockCipher cipher = new PaddedBufferedBlockCipher(
        new CBCBlockCipher(new AESFastEngine()));
    cipher.init(false, key);
    byte[] res =
      new byte[cipher.getOutputSize(result.length)];
    int leng = cipher.processBytes(result, 0,
        result.length, res, 0);
    try {
      cipher.doFinal(res, leng);
    } catch (CryptoException ce) {
      System.out.println("Decryption␣with␣AES␣error␣...");
      ce.printStackTrace();
    }
    return res;
  }

}

package cinemaservice.model.facade;

import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.text.ParseException;
import java.util.Vector;

import javax.naming.NamingException;

import cinemaservice.exceptions.CinemaServiceException;
import cinemaservice.model.beans.requestBeans.Authentication_1_Req_Bean;
import cinemaservice.model.beans.requestBeans.Cancel_Tickets_Req_Bean;
import cinemaservice.model.beans.requestBeans.Change_Password_Req_Bean;
import cinemaservice.model.beans.requestBeans.Cinema_Hall_Conf_Req_Bean;
import cinemaservice.model.beans.requestBeans.Find_Movies_Req_Bean;
import cinemaservice.model.beans.requestBeans.Movie_Details_Req_Bean;
import cinemaservice.model.beans.requestBeans.Purchase_Tickets_Req_Bean;
import cinemaservice.model.beans.requestBeans.Rate_Movie_Req_Bean;
import cinemaservice.model.beans.requestBeans.Reject_Payment_Req_Bean;
import cinemaservice.model.beans.requestBeans.
    Select_Deselect_Seats_Req_Bean;

import cinemaservice.model.beans.responseBeans.Response_Msg_Bean;
import cinemaservice.model.interfaces.RequestModelInterface;
import cinemaservice.model.interfaces.ResponseModelInterface;
```

```
/**
 * This is the FACADE of the Cinema Service Model. It hides the
     complexity
 * of the Request - Response Model from the client and provides methods
 * for accessing the data contained in the Request-Response Java Beans.
 *
 * It delegates the functionality further to the Request / Response Data
     Models.
 *
 * It implements the FACADE DESIGN PATTERN.
 *
 * @author Mihai Balan - s031288
 *
 */
public class FacadeToModel implements RequestModelInterface,
    ResponseModelInterface{

  /**
   * Request Data Model that contains the functionality for retrieving
   * the SQL parameter list from the Request Java Beans send by the
       MIDlet
   * to the Servlets
   */
  private RequestDataModel reqDataModel = new RequestDataModel();

  /**
   * Response Data Model that contains the fucntionality for creating
   * the Response Java Beans that are to be sent from the Servlets
   * to the MIDlet
   */
  private ResponseDataModel respDataModel = new ResponseDataModel();


  // ============================================================================

  //                           REQUEST DATA MODEL
  // ============================================================================


  /**
   * Delegates the request to the Request Data Model to get
   * the data stored in the Authentication_1_Req_Bean received from the
       MIDlet
   * and set the parameters of the prepared statement for changing user
       password
   *
```

```
 * @param auth1Bean  The Authentication_1_Req_Bean
 * @param pqPsqlStmt The prepared SQL statement for authenticating the
     user
 * @throws SQLException
 */
public void getAuth1ReqBeanData(PreparedStatement pqPsqlStmt,
    Authentication_1_Req_Bean auth1Bean) throws SQLException {
  reqDataModel.getAuth1ReqBeanData(pqPsqlStmt, auth1Bean);
}

/**
 * Delegates the request to the Request Data Model to get
 * the data stored in the Change_Password_Req_Bean received from the
     MIDlet
 * and set the parameters of the prepared statement for changing user
     password
 *
 * @param chgPswdBean The Change_Password_Req_Bean sent from the MIDlet
 * @param pqPsqlStmt The prepared SQL statement for changing user
     password
 * @throws SQLException
 */
public void getChangePasswordReqBeanData(PreparedStatement pqPsqlStmt,
    Change_Password_Req_Bean chgPswdBean) throws SQLException {
  reqDataModel.getChangePasswordReqBeanData(pqPsqlStmt, chgPswdBean);
}


/**
 * Delegates the request to the Request Data Model to get
 * the data stored in the Find_Movies_Req_Bean received from the MIDlet
     ,
 * talk to MLS, get all Cinemas that matche the searching criteria,
 * and create the SQL parameter list that is to be used to execute
 * the SQL querie for finding movies
 *
 * @param chgPswdBean The Find_Movies_Req_Bean sent from the MIDlet
 * @return The SQL parameter list that is to be used to executed
 *         the find_movies stored procedures
 */
public void getFindMoviesReqBeanData(PreparedStatement pqPsqlStmt,
    Find_Movies_Req_Bean findMoviesReqBean) throws SQLException,
    ParseException {
  reqDataModel.getFindMoviesReqBeanData(pqPsqlStmt, findMoviesReqBean);
}
```

```
/**
 * Delegates the request to the Request Data Model to get
 * the data stored in the Cinema_Hall_Conf_Req_Bean received
 * from the MIDlet and se the parameters of the prepared statement
 * for retrieving the cinema hall conf details
 *
 * @param cinHallConfReqBean The Cinema_Hall_Conf_Req_Bean send from
     the MIDlet
 * @param The preapared SQL statementthat is to be used to executed
 *        i.e. the SQL querie for retrieving the requested cinema hall
     conf details
 */
public void getCinemaHallConfReqBeanData(PreparedStatement pqPsqlStmt,
    Cinema_Hall_Conf_Req_Bean cinHallConfReqBean) throws SQLException{
  reqDataModel.getCinemaHallConfReqBeanData(pqPsqlStmt,
      cinHallConfReqBean);
}



/**
 * Delegates the request to the Request Data Model to get
 * the data stored in the Select_Deselect_Seats_Req_Bean received
 * from the MIDlet and set the parameters of the prepared statement
 * for selecting / deselecting user's chosen seats
 * and retrieve all booked seats for the given show
 *
 * @param selDeselSeatsReqBean The Select_Deselect_Seats_Req_Bean send
     from the MIDlet
 * @param The prepared SQL statement that is to be used to executed
 *        i.e. the SQL querie for selecting / deselecting user's chosen
     seats
 *        and retrieve all booked seats for the given show
 */
public void getSelectDeselectSeatsReqBeanData(PreparedStatement
    pqPsqlStmt, Select_Deselect_Seats_Req_Bean selDeselSeatsReqBean)
    throws SQLException {
  reqDataModel.getSelectDeselectSeatsReqBeanData(pqPsqlStmt,
      selDeselSeatsReqBean);
}


/**
 * Delegates the request to the Request Data Model to get
 * the data stored in the Reject_Payment_Req_Bean received
 * from the MIDlet and set the parameters of the prepared
 * statement for rejecting payment for the selected seats
```

```
 *
 * @param rejPaymentReqBean The Reject_Payment_Req_Bean send from the
     MIDlet
 * @param The prepared SQL statement that is to be used to executed
 *        the SQL querie for rejecting payment for the selected seats
 */
public void getRejectPaymentReqBeanData(PreparedStatement pqPsqlStmt,
    Reject_Payment_Req_Bean rejPaymentReqBean) throws SQLException{
  reqDataModel.getRejectPaymentReqBeanData(pqPsqlStmt,
      rejPaymentReqBean);
}


/**
 * Delegates the request to the Request Data Model to get
 * the data stored in the Purchase_Tickets_Req_Bean received
 * from the MIDlet and set the parameters for the prepared
 * SQL statement for purchasing user's selected seats
 *
 * @param purchaseTicketsReqBean The Purchase_Tickets_Req_Bean send
     from the MIDlet
 * @param The prepared statement for purchasing user's selected seats
 */
public void getPurchaseTicketsReqBeanData(PreparedStatement pqPsqlStmt
    , Purchase_Tickets_Req_Bean purchaseTicketsReqBean) throws
    SQLException, ParseException{
  reqDataModel.getPurchaseTicketsReqBeanData(pqPsqlStmt,
      purchaseTicketsReqBean);
}


/**
 * Delegates the request to the Request Data Model to get
 * the data stored in the Cancel_Tickets_Req_Bean received
 * from the MIDlet and create the SQL parameter list that is
 * to be used to execute the SQL querie for cancelling
 * user's purchased tickets by credit card
 *
 * @param cancelTicketsReqBean The Cancel_Tickets_Req_Bean send from
     the MIDlet
 * @param The prepared statement for canceling user's purchased tickets
      using
 *        the CREDIT CARD payment method
 */
public void getCancelTicketsReqBeanData(PreparedStatement pqPsqlStmt,
    Cancel_Tickets_Req_Bean cancelTicketsReqBean) throws SQLException{
  reqDataModel.getCancelTicketsReqBeanData(pqPsqlStmt,
```

```
      cancelTicketsReqBean);
}


/**
 * Delegates the request to the Request Data Model to get
 * the data stored in the Rate_Movie_Req_Bean received from the MIDlet
 * and set the parameter for the prepared statement for rating the
     movie
 *
 * @param rateMovBean The Rate_Movie_Req_Bean sent from the MIDlet
 * @param pqPsqlStmt The prepared statement for rating the movie
 */
public void getRateMovieReqBeanData(PreparedStatement pqPsqlStmt,
    Rate_Movie_Req_Bean rateMovBean) throws SQLException{
  reqDataModel.getRateMovieReqBeanData(pqPsqlStmt, rateMovBean);
}


/**
 * Delegates the request to the Request Data Model to get
 * the data stored in the Movie_Details_Req_Bean received
 * from the MIDlet and set the parameters for the prepared
 * statement for retrieving the requested movie details
 *
 * @param movDetailsReqBean The Movie_Details_Req_Bean send from the
     MIDlet
 * @param pqPsqlStmt The preapared SQL statement
 *                   for retrieving the requested movie details
 * @throws SQLException
 */
public void getMovieDetailsReqBeanData(PreparedStatement pqPsqlStmt,
    Movie_Details_Req_Bean movDetailsReqBean) throws SQLException{
  reqDataModel.getMovieDetailsReqBeanData(pqPsqlStmt, movDetailsReqBean
      );
}




// ============================================================================

//                           RESPONSE DATA MODEL
// ============================================================================


/**
```

```
 * Delegates the request to the Request Data Model to create
 * the Response_Msg_Bean and set up the response code
 * that is to be sent to the MIDlet as a Java Bean Object
 *
 * @param sqlResult The result of the SQL query to authenticate the
     user
 * @return The Response_Msg_Bean containing the response code of
     running the
 *        SQL statement to authenticate the user
 */
public Response_Msg_Bean setResponseAuth1Bean(Vector sqlResult){
  return respDataModel.setResponseAuth1Bean(sqlResult);
}


/**
 * Delegates the request to the Request Data Model to create
 * the Response_Msg_Bean and set up the response code
 * that is to be sent to the MIDlet as a Java Bean Object
 *
 * @param sqlResult The result of the SQL query to change user password
 * @return The Response_Msg_Bean containing the response code of
     running the
 *        SQL statement to change user's password
 */
public Response_Msg_Bean setResponseChangePasswordBean(Vector sqlResult
    ){
  return respDataModel.setResponseChangePasswordBean(sqlResult);
}


/**
 * Delegates the request to the Request Data Model to create
 * the Find_Movies_Resp_Bean and set up the response code
 * that is to be sent to the MIDlet as a Java Bean Object
 *
 * @param sqlResult The result of the SQL query to find given movies
 * @return The Find_Movies_Resp_Bean containing the details of running
 *        the find_movies stored procedures
 */
public Response_Msg_Bean setResponseFindMoviesBean(Vector sqlResult) {
  return respDataModel.setResponseFindMoviesBean(sqlResult);
}


/**
 * Delegates the request to the Request Data Model to create
```

```
 * the Cinema_Hall_Conf_Resp_Bean and set up the cinema hall conf
     details
 * that are to be sent to the MIDlet as a Java Bean Object
 *
 * @param sqlResult The result of the SQL query to get the requested
 *       cinema hall conf
 * @return The Cinema_Hall_Conf_Resp_Bean containing the requested
 *         cinema hall conf details
 */
public Response_Msg_Bean setResponseCinemaHallConfBean(Vector sqlResult
    ) {
  return respDataModel.setResponseCinemaHallConfBean(sqlResult);
}


/**
 * Delegates the request to the Request Data Model to create
 * the Background_Cinema_Hall_Conf_Resp_Bean and set up
 * the cinema hall conf details that are to be sent
 * to the MIDlet as a Java Bean Object
 *
 * @param sqlResult The result of the SQL query to get the requested
 *       cinema hall conf
 * @return The Background_Cinema_Hall_Conf_Resp_Bean containing
 *         the requested cinema hall conf details
 */
public Response_Msg_Bean setResponseBackgroundCinemaHallConfBean(Vector
     sqlResult) {
  return respDataModel.setResponseBackgroundCinemaHallConfBean(
      sqlResult);
}


/**
 * Delegates the request to the Request Data Model to create
 * the Select_Deselect_Seats_Resp_Bean that contains all booked seats
 * for the fiven show that are to be sent to the MIDlet as a Java Bean
     Object
 *
 * @param sqlResult The result of the SQL query to get all booked seats
 * @return The Select_Deselect_Seats_Resp_Bean containing all booked
     seats
 */
public Response_Msg_Bean setResponseSelectDeselectSeats(Vector
    sqlResult) {
  return respDataModel.setResponseSelectDeselectSeats(sqlResult);
}
```

```
/**
 * Delegates the request to the Request Data Model to create
 * the Response_Msg_Bean and set up the response code
 * that is to be sent to the MIDlet as a Java Bean Object
 *
 * @param sqlResult The result of the SQL query to reject the
 *                 payment for the selected seats
 * @return The Response_Msg_Bean containing the response code of
     running the
 *        SQL statement to reject the payment for the selected seats
 */
public Response_Msg_Bean setResponseRejectPaymentBean(Vector sqlResult)
     {
  return respDataModel.setResponseRejectPaymentBean(sqlResult);
}


/**
 * Delegates the request to the Request Data Model to create
 * the Purchase_Tickets_Resp_Bean and set up the response code
 * that is to be sent to the MIDlet as a Java Bean Object
 *
 * @param sqlResult The result of the SQL query to purchase user's
     reserved tickets
 * @return The Purchase_Tickets_Resp_Bean containing the response code
     of running the
 *        SQL statement to purchase user's reserved tickets
 */
public Response_Msg_Bean setResponsePurchasedTicketsBean(Vector
    sqlResult) throws SQLException, NamingException, ParseException,
    CinemaServiceException{
  return respDataModel.setResponsePurchasedTicketsBean(sqlResult);
}


/**
 * Delegates the request to the Request Data Model to create
 * the Response_Msg_Bean and set up the response code
 * that is to be sent to the MIDlet as a Java Bean Object
 *
 * @param sqlResult The result of the SQL query to cancel
 *                 user's purchased tickets by Credit Card
 * @return The Response_Msg_Bean containing the response code of
     running the
 *        SQL statement to cancel user's purchased tickets by Credit
```

```
      Card
   */
  public Response_Msg_Bean setResponseCancelTicketsBean(Vector sqlResult)
      {
    return respDataModel.setResponseCancelTicketsBean(sqlResult);
  }


  /**
   * Delegates the request to the Request Data Model to create
   * the Response_Msg_Bean and set up the response code
   * that is to be sent to the MIDlet as a Java Bean Object
   *
   * @param sqlResult The result of the SQL query to rate the movie
   * @return The Response_Msg_Bean containing the response code of
       running the
   *        SQL statement to rate the movie
   */
  public Response_Msg_Bean setResponseRateMovieBean(Vector sqlResult) {
    return respDataModel.setResponseRateMovieBean(sqlResult);
  }


  /**
   * Delegates the request to the Request Data Model to create
   * the Movie_Details_Resp_Bean and set up the movie details
   * that are to be sent to the MIDlet as a Java Bean Object
   *
   * @param sqlResult The result of the SQL query to get the requested
   *       movie details
   * @return The Movie_Details_Resp_Bean containing the requested movie
   *       details
   */
  public Response_Msg_Bean setResponseMovieDetailsBean(Vector sqlResult){
    return respDataModel.setResponseMovieDetailsBean(sqlResult);
  }

} // end class

package cinemaservice.model.facade;

import cinemaservice.model.facade.Encryptor;

import org.bouncycastle.crypto.BufferedBlockCipher;
import org.bouncycastle.crypto.CryptoException;
import org.bouncycastle.crypto.PBEParametersGenerator;
import org.bouncycastle.crypto.digests.SHA1Digest;
import org.bouncycastle.crypto.engines.AESFastEngine;
```

```
import org.bouncycastle.crypto.generators.PKCS12ParametersGenerator;
import org.bouncycastle.crypto.modes.CBCBlockCipher;
import org.bouncycastle.crypto.paddings.PaddedBufferedBlockCipher;
import org.bouncycastle.crypto.params.*;

/**
 * This class contains different helping methods
 * for encryption - decryption operations on the
 * messages between the MIDlet and Sevlets
 *
 * @author Mihai Balan - s031288
 *
 */
public class HelpingCrypto {

  /**
   * This method is producing the AES key. The object returned contains
       of
   * the 128 bit key produced from the salt and from the user serial
       number.
   * It contains also 128 bit Initialization Vector.
   *
   * @param salt The salt value generated by the authentication server
   * @param key user serial number
   * @return 128 bit AES key with 128 bit Initalization Vector
   */
  public ParametersWithIV createKey(String salt, String key){
    //Generator for the required parameters.
    PBEParametersGenerator generator =
      new PKCS12ParametersGenerator(new SHA1Digest());
    generator.init(
        PBEParametersGenerator.PKCS12PasswordToBytes(key.toCharArray()),
        salt.getBytes(), 1024);
    // Generate a 128 bit key w/ 128 bit IV
    ParametersWithIV ret =
      (ParametersWithIV)generator.generateDerivedParameters(128, 128);
    return ret;
  } // end createKey()

  /**
   * Encrypt the message using user's key from the record store
   * Return the encrypted string to be sent to the remote URL
   *
   * @param key  The user's key used to encrypt the message.
   * @param msg  The MIDlet's request that is to be encrypted.
   * @param aesKey The AES key created on the Server side
   *
```

```
 * @return    Returns the MIDlet's encrypted request as a byte array.
 * @throws    Throws Exceprion.
 */
protected byte[] encryptMessage(String key, String msgToEncrypt,
    ParametersWithIV aesKey) throws Exception{
  Encryptor encryptor = new Encryptor(key);
  byte[] encrypted = encryptor.encryptWithAES(aesKey, msgToEncrypt.
      getBytes());
  System.out.println("-----WRITE␣MESSAGE␣TO␣ROBOT␣--␣Encrypted␣string
      :␣" + encrypted.toString());
  return encrypted;

} // end encryptMessage()


/**
 * Decrypt the message from the servlet using user's key
 * stored in the record store. Return the dencrypted string.
 *
 * @param key  The key used to decrypt the message.
 * @param msg  The servlet's encrypted answer that is to be decrypted.
 * @param aesKey The AES key created on the Server side
 *
 * @return    Returns the decripted message as a String.
 * @throws    CryptoException in case of crypto operations.
 */
protected byte[] decryptMessage(String key, byte[] msg,
    ParametersWithIV aesKey) throws CryptoException{
  Encryptor encryptor = new Encryptor(key);
  byte[] decrypted = encryptor.decryptWithAES(aesKey, msg);
  System.out.println("----␣SEND␣MESSAGE␣---␣␣Decrypted␣string:␣" +
      decrypted);
  return decrypted;

} // end decryptMessage()

/**
 * The following method is responsible for decrypting the messages
 * received from the user in step PRT2 and PRT3.
 *
 * @param key The key object previously created from the salt and from
     the serial number
 * @param result Message received and intended for dectyption
 * @return plaintext encoded as a byte array
 */
public byte[] decryptWithAES(ParametersWithIV key, byte[] result){
  BufferedBlockCipher cipher = new PaddedBufferedBlockCipher(
```

```
      new CBCBlockCipher(new AESFastEngine()));
  cipher.init(false, key);
  byte[] res =
    new byte[cipher.getOutputSize(result.length)];
  int leng = cipher.processBytes(result, 0,
      result.length, res, 0);
  try {
    cipher.doFinal(res, leng);
  } catch (CryptoException ce) {
    System.out.println("Decryption␣with␣AES␣error␣...");
    ce.printStackTrace();
  }
  return res;

} // end decryptWithAES()


/**
 * This method is responsible for encryption of the data, with the use
     of
 * provided key. AES cipher in CBC mode is used.
 *
 * @param key The key object previously created from the salt and from
     the serial number
 * @param msg message as a byte array
 * @return decrypted message as a byte array
 */
public byte[] encryptWithAES(ParametersWithIV key, byte[] msg){
  //The cipher instance
  BufferedBlockCipher cipher = new PaddedBufferedBlockCipher(
      new CBCBlockCipher(new AESFastEngine()));
  //initialization of the cipher for encryption
  cipher.init(true, key);
  //the output should have the same size as input
  byte[] result = new byte[cipher.getOutputSize(msg.length)];
  //message is processe byte by byte
  int len = cipher.processBytes(msg, 0,
      msg.length, result, 0);
  try {
    //last, ususally short block must be handled in a different way
    cipher.doFinal(result, len);
  } catch (CryptoException ce) {
    System.out.println("Encryption␣with␣AES␣error␣...");
    ce.printStackTrace();
  }
  return result;
```

```
} // end encryptWithAES()

/**
 * Decrypt the message from the servlet using user's key
 * stored in the record store. Return the dencrypted string.
 *
 * @param key The key used to decrypt the message.
 * @param msg The servlet's encrypted answer that is to be decrypted.
 *
 * @return    Returns the decripted message as a String.
 * @throws    CryptoException in case of crypto operations.
 */
public String decryptMessage(String key, byte[] msg) throws
    CryptoException{
  Encryptor encryptor = new Encryptor(key);
  String decrypted = encryptor.decryptString(msg);
  System.out.println("++++++++++++++++++++Decrypted␣string:␣" +
      decrypted);
  return decrypted;

} // end decryptMessage()


/**
 * Decrypt the message from the servlet using user's key
 * stored in the record store. Return the dencrypted string.
 *
 * @param key The key used to decrypt the message.
 * @param msg The servlet's encrypted answer that is to be decrypted.
 *
 * @return    Returns the decripted message as a byte array.
 * @throws    CryptoException in case of crypto operations.
 */
public byte[] decrypt(String key, byte[] msg) throws CryptoException{
  Encryptor encryptor = new Encryptor(key);
  byte[] decrypted = encryptor.decrypt(msg);
  System.out.println("++++++++++++++++++++Decrypted␣string:␣" +
      decrypted);
  return decrypted;

} // end decrypt()

/**
 * Encrypt the servlet's response using the user's key.
 * Return the encrypted string to be sent to the MIDlet
 *
 * @param key The key used to encrypt the message.
```

```
   * @param msg The servlet's response that is to be encrypted as string.
   *
   * @return    Returns the servlet's encrypted response as a byte array.
   * @throws    CryptoException.
   */
  public byte[] encryptMessage(String key, String msg) throws
      CryptoException{
    Encryptor encryptor = new Encryptor(key);
    byte[] encrypted = encryptor.encryptString(msg);
    System.out.println("Encrypted␣string:␣" + encrypted.toString());
    return encrypted;

  } // end encryptMessage()


  /**
   * Encrypt the servlet's response using the user's key.
   * Return the encrypted string to be sent to the MIDlet
   *
   * @param key The key used to encrypt the message.
   * @param msg The servlet's response that is to be encrypted as byte
       array.
   *
   * @return    Returns the servlet's encrypted response as a byte array.
   * @throws    CryptoException.
   */
  public byte[] encryptMessage(String key, byte[] msg) throws
      CryptoException{
    Encryptor encryptor = new Encryptor(key);
    byte[] encrypted = encryptor.encrypt(msg);
    System.out.println("Encrypted␣string:␣" + encrypted.toString());
    return encrypted;

  } // end encryptMessage()

}

package cinemaservice.model.facade;

import java.lang.reflect.Array;
import java.util.UUID;
import java.util.Vector;



/**
 * Helper class for parsing different formats of Strings
 * and returning the elements in the string as int, double, arrays, etc.
```

```
 *
 * @author Mihai Balan - s031288
 *
 */
public class HelpingParser {


  /**
   * Get the number of elements in the string to be parsed
   * after splitting the string by using the given delimitator
   *
   * @param strToBeparsed String to be parsed
   * @param delimitator Delimitator as regex
   *
   * @return The no. of elements in the parsed string
   */
  public int getNoOfElements(String strToBeparsed, String delimitator){

    String parsed[] = strToBeparsed.split(delimitator);
    return parsed.length;

  } // end getNoOfElements()


  /**
   * Get the elements in the string to be parsed as int[]
   * after splitting the string by using the given delimitator
   *
   * @param strToBeparsed String to be parsed
   * @param delimitator Delimitator as regex
   *
   * @return The elements as int[] in the parsed string
   */
  public int[] getElementsAsInt(String strToBeparsed, String delimitator)
      {

    String parsed[] = strToBeparsed.split(delimitator);
    int parsedRet[] = new int[parsed.length];

    for (int i = 0; i < parsed.length; i++){
      parsedRet[i] = Integer.parseInt(parsed[i]);
    }

    return parsedRet;

  } // end getElementsAsInt()
```

```
/**
 * Get the elements in the string to be parsed as double[]
 * after splitting the string by using the given delimitator
 *
 * @param strToBeparsed String to be parsed
 * @param delimitator Delimitator as regex
 *
 * @return The elements as double[] in the parsed string
 */
public double[] getElementsAsDouble(String strToBeparsed, String
    delimitator){

  String parsed[] = strToBeparsed.split(delimitator);
  double parsedRet[] = new double[parsed.length];

  for (int i = 0; i < parsed.length; i++){
    parsedRet[i] = Double.parseDouble(parsed[i]);
  }

  return parsedRet;

} // end getElementsAsDouble()


/**
 * Get all booked seats for a given show, as int[][]
 * after splitting the string by using the given delimitator
 *
 * @param strToBeparsed String to be parsed
 * @param delimitator Delimitator as regex
 *
 * @return The seats as int[][] in the parsed string
 */
public int[][] getAllBookedSeats(String strToBeparsed, String
    delimitator){

  String parsed[] = strToBeparsed.split(delimitator);
  int seats[][] = new int[parsed.length][2];

  for (int i = 0; i < parsed.length; i++){
    seats[i][0] = Integer.parseInt(parsed[i].split(",")[0]);
    seats[i][1] = Integer.parseInt(parsed[i].split(",")[1]);
  }

  return seats;
```

```
} // end getCinemaHallSeats()


/**
 * Get the all booked seats for a given cinema hall as int[][]
 * after splitting the string by using the given delimitator
 *
 * @param strToBeparsed String to be parsed
 * @param delimitator Delimitator as regex
 *
 * @return The seats as int[][] in the parsed string
 */
public int[][] getCinemaHallSeats(String strToBeparsed, String
    delimitator){

  String parsed[] = strToBeparsed.split(delimitator);
  int seats[][] = new int[parsed.length][2];

  for (int i = 0; i < parsed.length; i++){
    parsed[i] = parsed[i].substring(1,4);
  }

  for (int i = 0; i < parsed.length; i++){
    seats[i][0] = Integer.parseInt(parsed[i].split(",")[0]);
    seats[i][1] = Integer.parseInt(parsed[i].split(",")[1]);
  }

  return seats;

} // end getCinemaHallSeats()

/**
 * Get the movies and all related information as String[][]
 * after splitting the sql result string by using the given delimitator
 * The result has the following format:
 * Movie, Hour, Cinema, City, Street, ShowLocationID, ShowTimeID \n
 * ....
 *
 * @param strToBeparsed String to be parsed
 * @param delimitator Delimitator as regex
 *
 * @return The movies as String[][] in the parsed string
 */
public String[][] getMovies(String strToBeparsed, String delimitator){

  String parsed[]    = strToBeparsed.split(delimitator);
  String movies[][]  = new String[parsed.length][7];
```

```
  // parse the sql result from the find_movies stored procedure
  for (int i = 0; i < parsed.length; i++){
    String[] tmp = parsed[i].split("\\|");
    for (int j = 0; j < 7; j++){
      movies[i][j] = tmp[j];
    }
  }

  return movies;

} // end getMovies()


/**
 * Converts a One-Dimension array to a string representation
 * of the following form: {a1, a2, ...} due to an issue in the
 * array data type conversion from java to pgSQL.
 *
 * Issue solved with driver v 7.4 and this small hack
 *
 * @param intArray The input array
 * @return String representation of the input array
 */
public String oneDimIntArrayToString(int[] intArray){

  String strArray = "{";
  for (int i = 0; i < intArray.length; i++){
    strArray += intArray[i] + ((i == intArray.length -1)?"":",");
  }
  strArray += "}";

  System.out.println("␣oneDimIntArrayToString␣----------------␣" +
      strArray);
  return strArray;

}// end oneDimIntArrayToString()


/**
 * Converts a Two-Dimension array to a string representation
 * of the following form: {{a1,b1},{a2,b2}, ...} due to an issue
 * in the array data type conversion from java to pgSQL.
 *
 * Issue solved with driver v 7.4 and this small hack
 *
 * @param intArray The input array
```

```
 * @return String representation of the input array
 */
public String twoDimIntArrayToString(int[][] intArray){

  String strArray = "{";
  for (int i = 0; i < intArray.length; i++){
    strArray += "{";
    for (int j = 0; j < intArray[0].length; j++){
      strArray += intArray[i][j] + ((j == intArray[0].length -1)?"":","
          );
    }
    strArray += ((i == intArray.length - 1)?"}":"},");
  }
  strArray += "}";

  System.out.println("␣twoDimIntArrayToString␣----------------␣" +
      strArray);
  return strArray;

} // end twoDimIntArrayToString()


/**
 * Converts a One-Dimension array to a string representation
 * of the following form: {a1, a2, ...} due to an issue in the
 * array data type conversion from java to pgSQL.
 *
 * Issue solved with driver v 7.4 and this small hack
 *
 * @param intArray The input array
 * @return String representation of the input array
 */
public String oneDimDoubleArrayToString(double[] doubleArray){

  String strArray = "{";
  for (int i = 0; i < doubleArray.length; i++){
    strArray += doubleArray[i] + ((i == doubleArray.length -1)?"":",");
  }
  strArray += "}";

  System.out.println("␣oneDimDoubleArrayToString␣----------------␣"
       + strArray);
  return strArray;

}// end oneDimDoubleArrayToString()
```

```
/**
 * Converts a Two-Dimension array to a string representation
 * of the following form: {{a1,b1},{a2,b2}, ...} due to an issue
 * in the array data type conversion from java to pgSQL.
 *
 * Issue solved with driver v 7.4 and this small hack
 *
 * @param intArray The input array
 * @return String representation of the input array
 */
public String twoDimDoubleArrayToString(double[][] doubleArray){

  String strArray = "{";
  for (int i = 0; i < doubleArray.length; i++){
    strArray += "{";
    for (int j = 0; j < doubleArray[0].length; j++){
      strArray += doubleArray[i][j] + ((j == doubleArray[0].length -1)?
          "":",");
    }
    strArray += ((i == doubleArray.length - 1)?"}":"},");
  }
  strArray += "}";

  System.out.println("␣twoDimDoubleArrayToString␣----------------␣"
       + strArray);
  return strArray;

} // end twoDimDoubleArrayToString()


/**
 * Converts a One-Dimension array to a string representation
 * of the following form: {a1, a2, ...} due to an issue in the
 * array data type conversion from java to pgSQL.
 *
 * Issue solved with driver v 7.4 and this small hack
 *
 * @param intArray The input array
 * @return String representation of the input array
 */
public String oneDimStringArrayToString(String[] stringArray){

  String strArray = "{";
  for (int i = 0; i < stringArray.length; i++){
    strArray += stringArray[i] + ((i == stringArray.length -1)?"":",");
  }
  strArray += "}";
```

```
  System.out.println("␣oneDimStringArrayToString␣------------------␣"
       + strArray);
  return strArray;

}// end oneDimIntArrayToString()


/**
 * Converts a Two-Dimension array to a string representation
 * of the following form: {{a1,b1},{a2,b2}, ...} due to an issue
 * in the array data type conversion from java to pgSQL.
 *
 * Issue solved with driver v 7.4 and this small hack
 *
 * @param intArray The input array
 * @return String representation of the input array
 */
public String twoDimStringArrayToString(String[][] stringArray){

  String strArray = "{";
  for (int i = 0; i < stringArray.length; i++){
    strArray += "{";
    for (int j = 0; j < stringArray[0].length; j++){
      strArray += stringArray[i][j] + ((j == stringArray[0].length -1)?
          "":",");
    }
    strArray += ((i == stringArray.length - 1)?"}":"},");
  }
  strArray += "}";

  System.out.println("␣twoDimStringArrayToString␣------------------␣"
       + strArray);
  return strArray;

} // end twoDimStringArrayToString()


/**
 * Retrieves the movie poster from the result of the
 * GetMovieDetails Stored procedure.
 *
 * The poster is stored as an object in a Vector.
 * It retrieves the object from the vector and
 * it uses the java.lang.reflect.Array class to
 * get it as an byte[].
 *
```

```
 * That byte[] is used in the Movie_Details_Response
 * _Bean to initialize the "poster" properties
 *
 *
 * @param sqlresult The result from the GetMovieDetails Stored
     procedure
 *                 containing all movies details and the movie poster
 * @return        The movie poster as a byte[]
 */
public byte[] getMoviePoster(Vector sqlresult){

  int l = Array.getLength(sqlresult.elementAt(1));
  byte[] image = new byte[l];

  for (int i = 0; i < l; i++){
    image[i] = Array.getByte(sqlresult.elementAt(1),i);
  }

  return image;

} // end getMoviePoster(Vector sqlresult)


/**
 * Generates a Reservation ID unique among all JVM's

 * @return A unique reservationID
 */
public String generateReservationID(){

  return generateUUID();

} // end generateReservationID()


/**
 * Generates a given no of unique TicketID's
 * among all JVM's
 *
 * @return TicketID's
 */
public String generateTicketIDs(String reservationID, int noOfTickets){

  String[] ticketIDs = new String[noOfTickets];

  for (int i = 0; i < noOfTickets; i++){
    ticketIDs[i] = reservationID + (i+1);
```

```
    }

    return oneDimStringArrayToString(ticketIDs);

  } // end generateReservationID()

  /**
   * Generates a UUID unique among all JVM's
   * for the reservationID and ticketID's values
   *
   * @return A unique UUID
   */
  public String generateUUID(){

    String uuid = UUID.randomUUID().toString();
    uuid = uuid.replaceAll("-", "").replaceAll(":", "");
    System.out.println("------------------------------------------------------------␣"
        );
    System.out.println("------------------------------------------------------------␣"
        );
    System.out.println("--------------------␣" + uuid);
    System.out.println("------------------------------------------------------------␣"
        );
    System.out.println("------------------------------------------------------------␣"
        );
    return uuid;

  } // end generateTicketID()

} // end class

package cinemaservice.model.facade;

import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Vector;

import movie_location_service.MovieLocationService;

import org.apache.log4j.*;
import org.bouncycastle.crypto.params.ParametersWithIV;

import creditcardvalidator.CardValidator;

import cinemaservice.constants.Parsing_Constants;
import cinemaservice.constants.SQL_Return_Codes;
```

```
import cinemaservice.model.beans.requestBeans.Authentication_1_Req_Bean;
import cinemaservice.model.beans.requestBeans.Cancel_Tickets_Req_Bean;
import cinemaservice.model.beans.requestBeans.Change_Password_Req_Bean;
import cinemaservice.model.beans.requestBeans.Cinema_Hall_Conf_Req_Bean;
import cinemaservice.model.beans.requestBeans.Find_Movies_Req_Bean;
import cinemaservice.model.beans.requestBeans.Movie_Details_Req_Bean;
import cinemaservice.model.beans.requestBeans.Purchase_Tickets_Req_Bean;
import cinemaservice.model.beans.requestBeans.Rate_Movie_Req_Bean;
import cinemaservice.model.beans.requestBeans.Reject_Payment_Req_Bean;
import cinemaservice.model.beans.requestBeans.
    Select_Deselect_Seats_Req_Bean;
import cinemaservice.model.interfaces.RequestModelInterface;


/**
 * Data model for the request objects from the client to the server
 *
 * Build the Request Java Beans sent by the MIDlet to the Servlets and
 * use the data stored in the request java beans to create
 * the SQL parameter lists that are to be used to executed the
 * SQL queries agains the pgSQL DB.
 *
 * @author Mihai Balan - s031288
 *
 */
public class RequestDataModel implements RequestModelInterface{

  /**
   * Initiate a custom logging category
   */
  private static Category cat = Category.getInstance(RequestDataModel.
      class.getName());

  /**
   * Constructor
   *
   */
  public RequestDataModel() {}

  /**
   * Get the data stored in the Authentication_1_Req_Bean
   * and creates the SQL parameter list that is to be used to execute
   * the SQL querie for user authentication
   *
   * @param auth1ReqBean The Authentication_1_Req_Bean send from the
       MIDlet
   * @param pqPsqlStmt The prepared statemnt for authenticating the user
```

```
 * @throws SQLException
 */
public void getAuth1ReqBeanData(PreparedStatement pqPsqlStmt,
    Authentication_1_Req_Bean auth1ReqBean) throws SQLException {

  pqPsqlStmt.setString(1,auth1ReqBean.getUserName());
  pqPsqlStmt.setString(2,auth1ReqBean.getPassword());

  // DEBUG
  cat.debug(auth1ReqBean.toString());

} //  end getAuth1ReqBeanData()



/**
 * Get the data stored in the Change_Password_Req_Bean received from
     the MIDlet
 * and creates the SQL parameter list that is to be used to execute
 * the SQL querie for changing user password
 *
 * @param chgPswdBean The Change_Password_Req_Bean send from the MIDlet
 * @param pqPsqlStmt The prepared statemnt for changing user password
 * @throws SQLException
 */
public void getChangePasswordReqBeanData(PreparedStatement pqPsqlStmt,
    Change_Password_Req_Bean chgPswdBean) throws SQLException {

  // get the SQL parameters from the Cinema_Hall_Conf_Req_Bean
  // and set the parameters for the prepared statement

  // generate the aesKey used for data decryption
  String aesSalt = System.getProperty("aesSalt"); // salt value (
      generated session key)
  String aesToken = System.getProperty("aesToken"); // token (userID)
  HelpingCrypto helpCrypto = new HelpingCrypto();
  ParametersWithIV aesKey = helpCrypto.createKey(aesSalt, aesToken);
  String oldPswd = new String(helpCrypto.decryptWithAES(aesKey,
      chgPswdBean.getOldPassword()));
  String newPswd = new String(helpCrypto.decryptWithAES(aesKey,
      chgPswdBean.getNewPassword()));
  System.out.println("----␣aesSalt␣:" + aesSalt);
  System.out.println("----␣aesToken␣:" + aesToken);
  System.out.println("----␣aes␣key:" + aesKey);
  System.out.println("----␣old␣pswd:" + oldPswd.trim());
  System.out.println("----␣new␣pswd:" + newPswd.trim());
  pqPsqlStmt.setString(1,chgPswdBean.getUserName());
```

```
  //pqPsqlStmt.setString(2,oldPswd);
  //pqPsqlStmt.setString(3,newPswd);
  pqPsqlStmt.setString(2,oldPswd.trim());
  pqPsqlStmt.setString(3,newPswd.trim());

  // DEBUG
  cat.debug(chgPswdBean.toString());

} // end getChangePasswordReqBeanData()



/**
 * Get the data stored in the Find_Movies_Req_Bean received from the
     MIDlet.
 * It uses this parameters to talk to MLS and retrieve all cinemas that
 * matche user's given search criteria.
 * Then, it construct a new SQL parameter list for running a query
 * against the Cinema DB from the Cinema Controller in order to get
 * all movies in those cinemas that fulfill user's given conditions
 *
 * @param findMoviesReqBean The Find_Movies_Req_Bean send from the
     MIDlet
 * @return The SQL parameter list that is to be used to executed
 *        the find_ movies stored procedure by the Cinema Controller
 */
public void getFindMoviesReqBeanData(PreparedStatement pqPsqlStmt,
    Find_Movies_Req_Bean findMoviesReqBean) throws SQLException,
    ParseException {

  // get the SQL parameters from the Find_Movies_Req_Bean
  String movie = findMoviesReqBean.getMovie();
  String street = findMoviesReqBean.getStreet();
  String city  = findMoviesReqBean.getCity();
  String zip   = findMoviesReqBean.getZip();
  String range = findMoviesReqBean.getRange();
  String showDate = findMoviesReqBean.getDate();
  cat.debug(findMoviesReqBean.toString());

  int errCode = -1;

  HelpingParser parser = new HelpingParser();
  String cinemaIDs[] = null;

  if ((!street.equals("")) && (!city.equals("")) && (!zip.equals(""))
       && (!range.equals(""))){
    // Contact MLS and get the cinema ID's matching
```

```
  // user's searching criteria
  MovieLocationService movieLocationService = new
      MovieLocationService();
  Vector getCinemas = movieLocationService.getCinemas(street, city,
      zip, range);

  errCode = Integer.parseInt(getCinemas.elementAt(0).toString());
  System.out.println("----------␣MLS␣code:␣" + errCode);
  if (errCode == SQL_Return_Codes.MOVIE_LOCATION_SERVICE_OK){

    String rawDBResult = getCinemas.elementAt(1).toString();
    //statusCode = Integer.parseInt(rawDBResult.substring(0, 3));
    cinemaIDs = rawDBResult.substring(3 + Parsing_Constants.NEW_LINE.
        length(),rawDBResult.length()-1).split("\\|");

  } else if(errCode == SQL_Return_Codes.MOVIE_LOCATION_SERVICE_ERROR
       || errCode == SQL_Return_Codes.MOVIE_LOCATION_SERVICE_NO_DATA)
      {

    cinemaIDs = null;
    System.out.println("+++++++++++++++␣cinema␣ID␣null␣" + errCode )
        ;

  }

} // if(req parameters)

// convert the string entry of the date to an SQL date
SimpleDateFormat formatter = new SimpleDateFormat("yyyy.MM.dd");
java.sql.Date reqDate = new java.sql.Date(formatter.parse(showDate).
    getTime());

String cinemaIDsforSQL = "";
if (cinemaIDs == null){
  cinemaIDsforSQL = "{}";
} else {
  cinemaIDsforSQL = parser.oneDimStringArrayToString(cinemaIDs);
  city = "";
}

if ((errCode == SQL_Return_Codes.MOVIE_LOCATION_SERVICE_ERROR) || (
    errCode == SQL_Return_Codes.MOVIE_LOCATION_SERVICE_NO_DATA)){
  city = "";
  cinemaIDsforSQL = "{0}";

}
```

```
  if(movie.equals("")){
    pqPsqlStmt.setString(1, movie);

  }else{
    pqPsqlStmt.setString(1, "%" + movie + "%");

  }

  if(city.equals("")){
    pqPsqlStmt.setString(3, city);

  }else{
    pqPsqlStmt.setString(3, "%" + city + "%");

  }

  pqPsqlStmt.setString(2, cinemaIDsforSQL);
  pqPsqlStmt.setDate (4, reqDate);

  // DEBUG
  cat.debug("Find␣Cinemas␣-␣Movie:␣" + movie);
  cat.debug("Find␣Cinemas␣-␣cinemaIDs:␣" + cinemaIDsforSQL);
  cat.debug("Find␣Cinemas␣-␣city:␣" + city);
  cat.debug("Find␣Cinemas␣-␣date:␣" + reqDate);

} // end getFindMoviesReqBeanData(...)



/**
 * Get the data stored in the Cinema_Hall_Conf_Req_Bean received from
     the MIDlet
 * and creates the SQL parameter list that is to be used to execute
 * the SQL querie for getting the cinema hall configuration
 *
 * @param cinHallConfReqBean The Cinema_Hall_Conf_Req_Bean send from
     the MIDlet
 * @param The preapared SQL statementthat is to be used to executed
 *        i.e. the SQL querie for retrieving the requested cinema hall
     conf details
 */
public void getCinemaHallConfReqBeanData(PreparedStatement pqPsqlStmt,
    Cinema_Hall_Conf_Req_Bean cinHallConfReqBean) throws SQLException{

  // get the SQL parameters from the Cinema_Hall_Conf_Req_Bean
  // and set the parameters for the prepared statement
  pqPsqlStmt.setInt(1,cinHallConfReqBean.getShowLocationID());
```

```
   pqPsqlStmt.setInt(2,cinHallConfReqBean.getShowTimeID());

   // DEBUG
   cat.debug(cinHallConfReqBean.toString());

 } // end getCinemaHallConfReqBeanData



 /**
  * Get the data stored in the Select_Deselect_Seats_Req_Bean received
      from the MIDlet
  * and set the prepared statement parameters based on the req bean
      property values
  * for retreving all booked seats for the given show
  *
  * @param selDeselSeatsReqBean The Select_Deselect_Seats_Req_Bean send
      from the MIDlet
  * @param pqPsqlStmt         The prepared SQL statement that is to be
      executed
  */
 public void getSelectDeselectSeatsReqBeanData(PreparedStatement
     pqPsqlStmt, Select_Deselect_Seats_Req_Bean selDeselSeatsReqBean)
     throws SQLException{

   HelpingParser parser = new HelpingParser();

   // get the SQL parameters from the Select_Deselect_Seats_Req_Bean
   // and set the parameters for the prepared statement that
   // is to be executed
   pqPsqlStmt.setInt(1,selDeselSeatsReqBean.getCommand());
   pqPsqlStmt.setInt(2,selDeselSeatsReqBean.getShowLocationID());
   pqPsqlStmt.setInt(3,selDeselSeatsReqBean.getShowTimeID());
   pqPsqlStmt.setString(4, parser.twoDimIntArrayToString(
       selDeselSeatsReqBean.getSeats()));

   // DEBUG
   cat.debug(selDeselSeatsReqBean.toString());

 } // end getSelectDeselectSeatsReqBeanData()



 /**
  * Get the data stored in the Reject_Reservation_Req_Bean received
  * from the MIDlet and set the parameters for the prepared statement
  * that is to be used to execute for rejecting
```

```
 * the user made reservation i.e. to cancel the selected seats
 *
 * @param rejReservationReqBean The Reject_Reservation_Req_Bean send
     from the MIDlet
 * @param pqPsqlStmt          The prepared SQL statement that is to be
     executed
 */
public void getRejectPaymentReqBeanData(PreparedStatement pqPsqlStmt,
    Reject_Payment_Req_Bean rejPaymentReqBean) throws SQLException{

  HelpingParser parser = new HelpingParser();

  // get the SQL parameters from the Reject_Reservation_Req_Bean
  // and set the parameters for the prepared statement
  pqPsqlStmt.setInt (1, rejPaymentReqBean.getShowLocationID());
  pqPsqlStmt.setInt (2, rejPaymentReqBean.getShowTimeID());
  pqPsqlStmt.setString(3, parser.twoDimIntArrayToString(
      rejPaymentReqBean.getSeats()));

  // DEBUG
  cat.debug(rejPaymentReqBean.toString());

} // end getRejectPaymentReqBeanData()


/**
 * Get the data stored in the Purchase_Tickets_Req_Bean received
 * from the MIDlet and sets the parameters for the prepared statement
 * for purchasing the tickets user has reserved
 *
 * @param purchaseTicketsReqBean The Purchase_Tickets_Req_Bean send
     from the MIDlet
 * @param pqPsqlStmt The prepared statement for purchasing the reserved
      tickets
 * @throws ParseException
 */
public void getPurchaseTicketsReqBeanData(PreparedStatement pqPsqlStmt
    , Purchase_Tickets_Req_Bean purchaseTicketsReqBean) throws
    SQLException, ParseException{

  boolean creditCardIsValid = false;

  // generate the aesKey used for data decryption
  HelpingCrypto helpCrypto = new HelpingCrypto();
  String aesSalt = System.getProperty("aesSalt"); // salt value (
      generated session key)
  String aesToken = System.getProperty("aesToken"); // token (userID)
```

```
ParametersWithIV aesKey = helpCrypto.createKey(aesSalt, aesToken);
String pswd = new String(helpCrypto.decryptWithAES(aesKey,
    purchaseTicketsReqBean.getPassword()));

System.out.println("----␣aesSalt␣:" + aesSalt);
System.out.println("----␣aesToken␣:" + aesToken);
System.out.println("----␣aes␣key:" + aesKey);
System.out.println("----␣pswd:" + pswd.trim());


// get and check the validity of the credit card data
// in case of CREDIT CARD PAYMENT
if (purchaseTicketsReqBean.getPurchaseMethod().toUpperCase()
    .equals(Parsing_Constants.CARD_PAYMENT)){

  CardValidator cv       = new CardValidator();
  String creditCardType  = new String(helpCrypto.decryptWithAES(aesKey
      , purchaseTicketsReqBean.getCreditCardType()));
  String creditCardNo    = new String(helpCrypto.decryptWithAES(aesKey
      , purchaseTicketsReqBean.getCreditCardNo()));
  String creditCardExpDate = new String(helpCrypto.decryptWithAES(
      aesKey, purchaseTicketsReqBean.getCreditCardExpDate()));
  String creditCardCW2   = new String(helpCrypto.decryptWithAES(aesKey
      , purchaseTicketsReqBean.getCreditCardCW2()));

  System.out.println("----␣ReqDM␣creditCardType:" + creditCardType.
      trim());
  System.out.println("----␣ReqDM␣creditCardNo:" + creditCardNo.trim()
      );
  System.out.println("----␣ReqDM␣creditCardExpDate:" +
      creditCardExpDate.trim());
  System.out.println("----␣ReqDM␣creditCardCW2:" + creditCardCW2.trim
      ());

  creditCardIsValid = cv.validateCreditCard(creditCardType.trim(),
                      creditCardNo.trim(),
                      creditCardExpDate.trim(),
                      creditCardCW2.trim());

} // end if(CREDIT CARD)

// in case of AT THE CINEMA PAYMENT
if (purchaseTicketsReqBean.getPurchaseMethod().toUpperCase()
    .equals(Parsing_Constants.CINEMA_PAYMENT)){
  creditCardIsValid = true;

}// end if(CINEMA)
```

```
HelpingParser parser = new HelpingParser();

// convert the string entry of the date to an SQL date
SimpleDateFormat formatter = new SimpleDateFormat("yyyy.MM.dd");
java.sql.Date resDate = new java.sql.Date(formatter.parse(
    purchaseTicketsReqBean.getReservationDate()).getTime());

// generate the ReservationID and TicketID's
String reservationID = parser.generateReservationID();
String ticketIDs   = parser.generateTicketIDs(reservationID,
    purchaseTicketsReqBean.getSeatsNoRows());

// if the card data is valid generate the ResID and Ticket ID's
// and update the DB (pay for the tickets)
if (creditCardIsValid){

  // get the SQL parameters from the Purchase_Tickets_Req_Bean
  // and set the parameters for the prepared statement
  pqPsqlStmt.setString(1, purchaseTicketsReqBean.getUserName());
  pqPsqlStmt.setString(2, pswd.trim());

} else {

  pqPsqlStmt.setString(1, "");
  pqPsqlStmt.setString(2, "");

} // end if (CreditCardIsValid)

// get the SQL parameters from the Purchase_Tickets_Req_Bean
// and set the parameters for the prepared statement
pqPsqlStmt.setInt (3, purchaseTicketsReqBean.getShowLocationID());
pqPsqlStmt.setInt (4, purchaseTicketsReqBean.getShowTimeID());
pqPsqlStmt.setString(5, parser.twoDimIntArrayToString(
    purchaseTicketsReqBean.getSeats()));
pqPsqlStmt.setString(6, parser.oneDimStringArrayToString(
    purchaseTicketsReqBean.getDiscounts()));
pqPsqlStmt.setString(7, reservationID);
pqPsqlStmt.setString(8, ticketIDs);
pqPsqlStmt.setDate (9, resDate);
pqPsqlStmt.setString(10, purchaseTicketsReqBean.getPurchaseMethod());
pqPsqlStmt.setString(11, (creditCardIsValid?"1":"2"));

System.out.println("-----------" + purchaseTicketsReqBean.getUserName
    ());
System.out.println("-----------" + pswd.trim());
System.out.println("-----------" + purchaseTicketsReqBean.
```

```
      getShowLocationID());
  System.out.println("-----------" + purchaseTicketsReqBean.
      getShowTimeID());
  System.out.println("-----------" + parser.twoDimIntArrayToString(
      purchaseTicketsReqBean.getSeats()));
  System.out.println("-----------" + parser.oneDimStringArrayToString(
      purchaseTicketsReqBean.getDiscounts()));
  System.out.println("-----------" + reservationID);
  System.out.println("-----------" + ticketIDs);
  System.out.println("-----------" + resDate);
  System.out.println("-----------" + purchaseTicketsReqBean.
      getPurchaseMethod());
  System.out.println("-----------" + (creditCardIsValid?"1":"2"));



  // DEBUG
  cat.debug(purchaseTicketsReqBean.toString());

} // end getPurchaseTicketsReqBeanData()


/**
 * Get the data stored in the Cancel_Tickets_Req_Bean received from the
      MIDlet
 * and creates the SQL parameter list that is to be used to execute
 * the SQL querie for cancelling the tickets
 * user has purchased using the "CARD" payment method
 *
 * @param cancelTicketsReqBean The Cancel_Tickets_Req_Bean send from
     the MIDlet
 * @return The SQL parameter list that is to be used to executed
 *         the SQL querie for cancelling the purchased tickets
 */
public void getCancelTicketsReqBeanData(PreparedStatement pqPsqlStmt,
    Cancel_Tickets_Req_Bean cancelTicketsReqBean) throws SQLException{

  HelpingParser parser = new HelpingParser();
  HelpingCrypto helpCrypto = new HelpingCrypto();

  // generate the aesKey used for data decryption
  String aesSalt = System.getProperty("aesSalt"); // salt value (
      generated session key)
  String aesToken = System.getProperty("aesToken"); // token (userID)
  ParametersWithIV aesKey = helpCrypto.createKey(aesSalt, aesToken);
  String pswd = new String(helpCrypto.decryptWithAES(aesKey,
      cancelTicketsReqBean.getPassword()));
```

```
  System.out.println("----␣aesSalt␣:" + aesSalt);
  System.out.println("----␣aesToken␣:" + aesToken);
  System.out.println("----␣aes␣key:" + aesKey);
  System.out.println("----␣pswd:" + pswd.trim());

  pqPsqlStmt.setString(1, cancelTicketsReqBean.getUserName());
  pqPsqlStmt.setString(2, pswd.trim());
  pqPsqlStmt.setString(3, cancelTicketsReqBean.getReservationID());
  pqPsqlStmt.setString(4, parser.oneDimStringArrayToString(
      cancelTicketsReqBean.getTicketID()));

  // DEBUG
  cat.debug(cancelTicketsReqBean.toString());

} // end getCancelTicketsReqBeanData()



/**
 * Get the data stored in the Rate_Movie_Req_Bean received from the
     MIDlet
 * and sets the parameters for the prepared statement for rating the
     movie
 *
 * @param rateMovBean The Rate_Movie_Req_Bean send from the MIDlet
 * @param pqPsqlStmt  The prepared SQL statement that is to be executed
 */
public void getRateMovieReqBeanData(PreparedStatement pqPsqlStmt,
    Rate_Movie_Req_Bean rateMovBean) throws SQLException{

  HelpingCrypto helpCrypto = new HelpingCrypto();

  // generate the aesKey used for data decryption
  String aesSalt = System.getProperty("aesSalt"); // salt value (
      generated session key)
  String aesToken = System.getProperty("aesToken"); // token (userID)
  ParametersWithIV aesKey = helpCrypto.createKey(aesSalt, aesToken);
  String pswd = new String(helpCrypto.decryptWithAES(aesKey,
      rateMovBean.getPassword()));

  System.out.println("----␣aesSalt␣:" + aesSalt);
  System.out.println("----␣aesToken␣:" + aesToken);
  System.out.println("----␣aes␣key:" + aesKey);
  System.out.println("----␣pswd:" + pswd.trim());

  //get the SQL parameters from the Reject_Reservation_Req_Bean
```

```
    // and set the parameters for the prepared statement
    pqPsqlStmt.setString (1, rateMovBean.getUserName());
    pqPsqlStmt.setString (2, pswd.trim());
    pqPsqlStmt.setInt    (3,rateMovBean.getShowLocationID() );
    pqPsqlStmt.setInt    (4, rateMovBean.getMovieScore());

    // DEBUG
    cat.debug(rateMovBean.toString());

  } // end getRateMovieReqBeanData()


  /**
   * Get the data stored in the Movie_Details_Req_Bean received
   * from the MIDlet and sets the parameters for the prepared
   * statement for retrieving the requested movie details
   *
   * @param movDetailsReqBean The Movie_Details_Req_Bean send from the
       MIDlet
   * @param pqPsqlStmt The prepared statement for retrieving
   *                   the requested movie details
   * @throws SQLException
   */
  public void getMovieDetailsReqBeanData(PreparedStatement pqPsqlStmt,
      Movie_Details_Req_Bean movDetailsReqBean) throws SQLException{
    // get the SQL parameters from the Reject_Reservation_Req_Bean
    // and set the parameters for the prepared statement
    pqPsqlStmt.setInt(1, movDetailsReqBean.getShowLocationID());

    // DEBUG
    cat.debug(movDetailsReqBean.toString());

  } // end getMovieDetailsReqBeanData()

}

package cinemaservice.model.facade;

import java.sql.SQLException;
import java.text.ParseException;
import java.util.Vector;

import javax.naming.NamingException;

import org.apache.log4j.*;
import org.bouncycastle.crypto.params.ParametersWithIV;

import creditcardvalidator.CardValidator;
```

```
import cinemaservice.constants.Error_Code_Constants;
import cinemaservice.constants.Parsing_Constants;
import cinemaservice.constants.SQL_Return_Codes;
import cinemaservice.exceptions.CinemaServiceException;
import cinemaservice.model.beans.requestBeans.Purchase_Tickets_Req_Bean;
import cinemaservice.model.beans.responseBeans.Authentication_1_Resp_Bean
    ;
import cinemaservice.model.beans.responseBeans.
    Background_Cinema_Hall_Conf_Resp_Bean;
import cinemaservice.model.beans.responseBeans.Cinema_Hall_Conf_Resp_Bean
    ;
import cinemaservice.model.beans.responseBeans.Find_Movies_Resp_Bean;
import cinemaservice.model.beans.responseBeans.Movie_Details_Resp_Bean;
import cinemaservice.model.beans.responseBeans.Purchase_Tickets_Resp_Bean
    ;
import cinemaservice.model.beans.responseBeans.Response_Msg_Bean;
import cinemaservice.model.beans.responseBeans.
    Select_Deselect_Seats_Resp_Bean;
import cinemaservice.model.interfaces.ResponseModelInterface;


/**
 * Data model for the response objects from the server to the client
 *
 * Build the Response Java Beans that are to be sent from the Servlets
 * to the MIDlet and return the Response Java Beans to be sent to the
     MIDlet
 *
 * @author Mihai Balan - s031288
 *
 */
public class ResponseDataModel implements ResponseModelInterface{

  /**
   * Initiate a custom logging category
   */
  private static Category cat = Category.getInstance(ResponseDataModel.
      class.getName());

  /**
   * The Constructor
   *
   */
  public ResponseDataModel() {}

  /**
```

```
 * Creates the Response_Msg_Bean and sets up the response code
 * that is to be sent to the MIDlet as a Java Bean Object
 *
 * @param sqlResult The result of the SQL query to authenticate the
     user
 * @return The Response_Msg_Bean containing the response code of
     running the
 *         SQL statement to auth the user
 */
public Response_Msg_Bean setResponseAuth1Bean(Vector sqlResult){

  // get the error code from the SQL result
  int sqlResponseCode = Integer.parseInt(sqlResult.elementAt(0).
      toString().substring(0, 3));

  Authentication_1_Resp_Bean respBean = new Authentication_1_Resp_Bean(
      sqlResponseCode);

  if (sqlResponseCode == SQL_Return_Codes.Authentication_E_MONEY_PRT_OK
      ){

    // removes the error code at the beginning of the SQL result
    // error code is separated with new line by the rest of the
        response
    String msgToBeParsed = sqlResult.elementAt(0).toString().substring
        (2 + Parsing_Constants.NEW_LINE.length());

    cat.debug("Msg␣to␣be␣parsed\n" + msgToBeParsed);

    // get all user info from the sql result
    String[] parsedMsg = msgToBeParsed.split("\n");
    String userID    = parsedMsg[0];
    String randomID  = parsedMsg[1];
    String e_money   = parsedMsg[2];

    respBean.setUserID(userID);
    respBean.setRandomID(randomID);
    respBean.setEMoney(e_money);
  }
  cat.debug("␣-----␣Auth_1_Resp_Bean:␣----␣\n" + respBean.toString());
  return respBean;

} // end setResponseChangePasswordBean()

/**
 * Creates the Response_Msg_Bean and sets up the response code
 * that is to be sent to the MIDlet as a Java Bean Object
```

```
 *
 * @param sqlResult The result of the SQL query to change user password
 * @return The Response_Msg_Bean containing the response code of
     running the
 *         SQL statement to change user's password
 */
public Response_Msg_Bean setResponseChangePasswordBean(Vector sqlResult
    ){

  // get the error code from the SQL result
  int sqlResponseCode = Integer.parseInt(sqlResult.elementAt(0).
      toString().substring(0, 3));

  Response_Msg_Bean respBean = new Response_Msg_Bean();
  respBean.setResponseCode(sqlResponseCode);
  respBean.setMsg("Chg␣Password␣Msg␣from␣response␣bean␣from␣the␣servlet
      ");
  cat.debug("␣-----␣Change␣Password:␣----␣\n" + respBean.toString());
  return respBean;



} // end setResponseChangePasswordBean()



/**
 * Creates the Find_Movies_Resp_Bean and sets up the response code
 * that is to be sent to the MIDlet as a Java Bean Object
 *
 * @param sqlResult The result of the SQL query containing all found
     movies
 *                 accordingly to user search criteria
 * @return The Find_Movies_Resp_Bean containing all found movies
     accordingly to user
 *         search criteria. The result has the following format:
 *         Movie, Hour, Cinema, City, Street, ShowLocationID, ShowTimeID
      \n
 *         ...
 */
public Response_Msg_Bean setResponseFindMoviesBean(Vector sqlResult) {

  int sqlResponseCode = Integer.parseInt(sqlResult.elementAt(0).
      toString().substring(0,3));
  Find_Movies_Resp_Bean findMoviesRespBean = new Find_Movies_Resp_Bean(
      sqlResponseCode);
```

```
  if ((sqlResponseCode == SQL_Return_Codes.
      FIND_MOVIES_CRIT_1_PRT_MOVIES_FOUND) ||
    (sqlResponseCode == SQL_Return_Codes.
        FIND_MOVIES_CRIT_2_PRT_MOVIES_FOUND) ||
    (sqlResponseCode == SQL_Return_Codes.
        FIND_MOVIES_CRIT_3_PRT_MOVIES_FOUND) ||
    (sqlResponseCode == SQL_Return_Codes.
        FIND_MOVIES_CRIT_4_PRT_MOVIES_FOUND) ||
    (sqlResponseCode == SQL_Return_Codes.
        FIND_MOVIES_CRIT_5_PRT_MOVIES_FOUND)){
    // removes the error code at the beginning of the SQL result
    // error code is separated with new line by the rest of the
        response
    String msgToBeParsed = sqlResult.elementAt(0).toString().substring
        (2 + Parsing_Constants.NEW_LINE.length());

    cat.debug("Msg␣to␣be␣parsed\n" + msgToBeParsed);

    // get all movies from the sql result
    HelpingParser parser = new HelpingParser();
    String[][] movies = parser.getMovies(msgToBeParsed, "\n");;
    /*  cat.debug("Movies length: " + movies.length);

     // DEBUG
      String debugMsg = "";
      for (int i = 0; i < movies.length; i++){
      for (int j = 0; j < 7; j++){
      debugMsg += movies[i][j] + " | ";
      }
      debugMsg += "\n";
      }
      cat.debug(debugMsg);
      */

    findMoviesRespBean.setRow_No(movies.length);
    findMoviesRespBean.setMovies(movies);

  } // end if (sqlResponseCode == 205 or 206 or 207 or 208 or 209)

  // DEBUG
  cat.debug(findMoviesRespBean.toString());
  return findMoviesRespBean;

} // end setResponseFindMoviesBean()
```

```
/**
 * Creates the Cinema_Hall_Conf_Resp_Bean and sets up the cinema hall
     conf
 * that are to be sent to the MIDlet as a Java Bean Object
 *
 * @param sqlResult The result of the SQL query to get the requested
 *       cinema hall conf details
 * @return The Cinema_Hall_Conf_Resp_Bean containing the requested
 *         cinema hall conf details
 */
public Response_Msg_Bean setResponseCinemaHallConfBean(Vector sqlResult
    ) {

  int sqlResponseCode = Integer.parseInt(sqlResult.elementAt(0).
      toString().substring(0,3));
  Cinema_Hall_Conf_Resp_Bean cinHallConfRespBean = new
      Cinema_Hall_Conf_Resp_Bean(sqlResponseCode);

  if(sqlResponseCode == SQL_Return_Codes.
      DISP_CINEMA_HALL_CONF_PRT_SHOW_FOUND){
    // removes the error code at the beginning of the SQL result
    // error code is separated with new line by the rest of the
        response
    String msgToBeParsed = sqlResult.elementAt(0).toString().substring
        (2 + Parsing_Constants.NEW_LINE.length());

    // parse the result and get the movie details
    String[] parsedMsg = msgToBeParsed.split("\n");

    cat.debug("Msg␣to␣be␣parsed\n" + msgToBeParsed);
    /*
     String debugMsg = "";
     for (int i = 0; i<parsedMsg.length; i++)
     debugMsg += i + " = " + parsedMsg[i] + "\n";
     cat.debug(debugMsg);
     */
    String basePrice_Str    = "";
    String disValues_Str    = "";
    String rows_cols_Str    = "";
    String allBookesSeats_Str = "";

    if(parsedMsg.length == 4){
      basePrice_Str    = parsedMsg[0];
      disValues_Str    = parsedMsg[1];
      rows_cols_Str    = parsedMsg[2];
      allBookesSeats_Str = parsedMsg[3];
```

```
} else{
  basePrice_Str    = parsedMsg[0];
  disValues_Str    = parsedMsg[1];
  rows_cols_Str    = parsedMsg[2];
  allBookesSeats_Str = "";
}
if (allBookesSeats_Str.equals(""))
  System.out.println("----------␣All␣Booked␣Seats␣is␣empty:␣" +
      allBookesSeats_Str);
else
  System.out.println("----------␣All␣Booked␣Seats␣is␣not␣empty:␣"
       + allBookesSeats_Str);

// helper class to parse different string formats
// and return the elements in the string
HelpingParser parser = new HelpingParser();

double basePrice = Double.parseDouble(basePrice_Str);
// get the number of discount values by parsing the string that
    contains
// all disoucnt values
int noDiscValue = parser.getNoOfElements(disValues_Str, "\\|");

// get all discount values as double[] by parsing the string
// that contains all discount values
double[] discountValues = new double[noDiscValue];
discountValues = parser.getElementsAsDouble(disValues_Str, "\\|");

// get the no of rows and cols for that cinema hall
int[] row_col = parser.getElementsAsInt(rows_cols_Str, "\\|");
int rows = row_col[0];
int cols = row_col[1];

int noRowBookedSeats = 0;
int noColsBookedSeats = 0;
int[][] allBookedSeats = null;
if (!allBookesSeats_Str.equals("")){
  // get all booked seats as int[] by parsing the string
  // that contains all bookes seats
  noRowBookedSeats = parser.getNoOfElements(allBookesSeats_Str, "
      \\|");;
  noColsBookedSeats = 2;
  allBookedSeats = new int[noRowBookedSeats][noColsBookedSeats];
  allBookedSeats = parser.getCinemaHallSeats(allBookesSeats_Str, "
      \\|");
}
```

```
    cinHallConfRespBean.setBasePrice(basePrice);
    cinHallConfRespBean.setNoDiscValue(noDiscValue);
    cinHallConfRespBean.setDiscValues(discountValues);
    cinHallConfRespBean.setRows(rows);
    cinHallConfRespBean.setCols(cols);
    cinHallConfRespBean.setNoRowsBookedSeats(noRowBookedSeats);
    cinHallConfRespBean.setNoColsBookedSeats(noColsBookedSeats);
    cinHallConfRespBean.setAllBookedSeats(allBookedSeats);

  }// end if (sqlResponseCode == ...)

  // DEBUG
  cat.debug(cinHallConfRespBean.toString());

  return cinHallConfRespBean;

} // end setResponseCinemaHallConfBean()



/**
 * Creates the Background_Cinema_Hall_Conf_Resp_Bean and sets up the
     cinema hall conf
 * that are to be sent to the MIDlet as a Java Bean Object
 *
 * @param sqlResult The result of the SQL query to get the requested
 *       cinema hall conf details
 * @return The Background_Cinema_Hall_Conf_Resp_Bean containing the
     requested
 *         cinema hall conf details
 */
public Response_Msg_Bean setResponseBackgroundCinemaHallConfBean(Vector
     sqlResult) {

  int sqlResponseCode = Integer.parseInt(sqlResult.elementAt(0).
      toString().substring(0,3));
  Background_Cinema_Hall_Conf_Resp_Bean backCinHallConfRespBean = new
      Background_Cinema_Hall_Conf_Resp_Bean(sqlResponseCode);

  if(sqlResponseCode == SQL_Return_Codes.
      BCKG_CINEMA_HALL_UPDATE_PRT_SHOW_FOUND){
    // removes the error code at the beginning of the SQL result
    // error code is separated with new line by the rest of the
        response
    String msgToBeParsed = sqlResult.elementAt(0).toString().substring
        (2 + Parsing_Constants.NEW_LINE.length());
```

```
// parse the result and get the movie details
String[] parsedMsg = msgToBeParsed.split("\n");

cat.debug("Msg␣to␣be␣parsed\n" + msgToBeParsed);
/*
 String debugMsg = "";
 for (int i = 0; i<parsedMsg.length; i++)
 debugMsg += i + " = " + parsedMsg[i] + "\n";
 cat.debug(debugMsg);
 */
String allBookesSeats_Str = "";

if(parsedMsg.length == 1)
  allBookesSeats_Str = parsedMsg[0];
else
  allBookesSeats_Str = "";

if (allBookesSeats_Str.equals(""))
  System.out.println("----------␣All␣Booked␣Seats␣is␣empty:␣" +
      allBookesSeats_Str);
else
  System.out.println("----------␣All␣Booked␣Seats␣is␣not␣empty:␣"
       + allBookesSeats_Str);

// helper class to parse different string formats
// and return the elements in the string
HelpingParser parser = new HelpingParser();

// get all booked seats as int[] by parsing the string
// that contains all bookes seats
int noRowBookedSeats = 0;
int noColsBookedSeats = 0;
int[][] allBookedSeats = null;
if (!allBookesSeats_Str.equals("")){
  // get all booked seats as int[] by parsing the string
  // that contains all bookes seats
  noRowBookedSeats = parser.getNoOfElements(allBookesSeats_Str, "
      \\|");;
  noColsBookedSeats = 2;
  allBookedSeats = new int[noRowBookedSeats][noColsBookedSeats];
  allBookedSeats = parser.getCinemaHallSeats(allBookesSeats_Str, "
      \\|");
}

backCinHallConfRespBean.setNoRowsBookedSeats(noRowBookedSeats);
backCinHallConfRespBean.setNoColsBookedSeats(noColsBookedSeats);
backCinHallConfRespBean.setAllBookedSeats(allBookedSeats);
```

```
  }// end if (sqlResponseCode == ...)

  // DEBUG
  cat.debug(backCinHallConfRespBean.toString());

  return backCinHallConfRespBean;

} // end setResponseBackgroundCinemaHallConfBean()



/**
 * Creates the Select_Deselect_Seats_Resp_Bean and sets up all booked
     seat info
 * that is to be sent to the MIDlet as a Java Bean Object
 *
 * @param sqlResult The result of the SQL query to get all booked seats
 *                  for the given show
 * @return The Select_Deselect_Seats_Resp_Bean containing all booked
     seats
 */
public Response_Msg_Bean setResponseSelectDeselectSeats(Vector
    sqlResult) {

  int sqlResponseCode = Integer.parseInt(sqlResult.elementAt(0).
      toString().substring(0,3));
  Select_Deselect_Seats_Resp_Bean selDeselSeatsRespBean = new
      Select_Deselect_Seats_Resp_Bean(sqlResponseCode);

  if ((sqlResponseCode == SQL_Return_Codes.
      SELECT_DESELECT_SEATS_PRT_SEATS_SELECTED_OK) ||
    (sqlResponseCode == SQL_Return_Codes.
        SELECT_DESELECT_SEATS_PRT_SEATS_DESELECTED_OK) ||
    (sqlResponseCode == SQL_Return_Codes.
        SELECT_DESELECT_SEATS_PRT_SEATS_SELECTED_ERROR) ||
    (sqlResponseCode == SQL_Return_Codes.
        SELECT_DESELECT_SEATS_PRT_SEATS_DESELECTED_ERROR)){

    // removes the error code at the beginning of the SQL result
    // error code is separated with new line by the rest of the
        response
    String msgToBeParsed = sqlResult.elementAt(0).toString();
    //String msgToBeParsed = sqlResult.elementAt(0).toString().
        substring(2 + Parsing_Constants.NEW_LINE.length());

    // parse the result and get the movie details
```

```
    String[] parsedMsg = msgToBeParsed.split("\n");

    cat.debug("Msg␣to␣be␣parsed\n" + msgToBeParsed);
    cat.debug("Msg␣to␣be␣parsed␣length:" + parsedMsg.length);
    /*
     String debugMsg = "";
     for (int i = 0; i<parsedMsg.length; i++)
     debugMsg += i + " = " + parsedMsg[i] + "\n";
     cat.debug(debugMsg);
     */
    int noRowBookedSeats = 0;
    int noColsBookedSeats = 0;
    int[][] allBookedSeats = null;

    if (parsedMsg.length == 2){
      cat.debug("All␣Booked␣seats␣raw:" + parsedMsg[1]);
      String allBookesSeats_Str = parsedMsg[1].substring(1);
      cat.debug("All␣Booked␣seats␣:" + allBookesSeats_Str);

      // helper class to parse different string formats
      // and return the elements in the string
      HelpingParser parser = new HelpingParser();

      // get all booked seats as int[] by parsing the string
      // that contains all bookes seats
      noRowBookedSeats = parser.getNoOfElements(allBookesSeats_Str, "
          \\|");;
      noColsBookedSeats = 2;
      allBookedSeats = new int[noRowBookedSeats][noColsBookedSeats];
      allBookedSeats = parser.getAllBookedSeats(allBookesSeats_Str, "
          \\|");

    }

    selDeselSeatsRespBean.setNoRowsBookedSeats(noRowBookedSeats);
    selDeselSeatsRespBean.setNoColsBookedSeats(noColsBookedSeats);
    selDeselSeatsRespBean.setAllBookedSeats(allBookedSeats);

  } // end if (sqlResponseCode == ...)

  cat.debug(selDeselSeatsRespBean.toString());
  return selDeselSeatsRespBean;

} // end setResponseSelectDeselectSeats()
```

```
/**
 * Creates the Response_Msg_Bean and sets up the response code
 * that is to be sent to the MIDlet as a Java Bean Object
 *
 * @param sqlResult The result of the SQL query to cancel
 *                  the reserved seats by the user
 * @return The Response_Msg_Bean containing the response code of
     running the
 *         SQL statement to cancel the reserved seats
 */
public Response_Msg_Bean setResponseRejectPaymentBean(Vector sqlResult)
     {

  // get the error code from the SQL result
  int sqlResponseCode = Integer.parseInt(sqlResult.elementAt(0).
      toString().substring(0, 3));

  Response_Msg_Bean respBean = new Response_Msg_Bean(sqlResponseCode);
  respBean.setMsg("Reject␣Payment␣Msg␣from␣response␣bean␣from␣the␣
      servlet");

  // DEBUG
  cat.debug("␣----␣Reject␣Payment␣-----\n" + respBean.toString());

  return respBean;

} // end setResponseRejectPaymentBean()



/**
 * Creates the Purcahsed_Tickets_Resp_Bean and sets up the reservation
     details
 * that are to be sent to the MIDlet as a Java Bean Object
 *
 * It reserves the tickets in the DB abd then pays for the tickets via
     the
 * ONLINE payment service. If the payment is successful, send the
     reservation ID
 * and ticketID's further to the MIDlet. Else, cancel the reserved
     tickets and
 * send and PAYMENT SERVICE ERROR to the MIDlet.
 *
 * @param sqlResult The result of the SQL query to get the purchase the
      tickets
```

```
 * @return The Purchased_Tickets_Resp_Bean containing the reservation
     details
 */
public Purchase_Tickets_Resp_Bean setResponsePurchasedTicketsBean(
    Vector sqlResult) throws SQLException, NamingException,
    ParseException, CinemaServiceException{

  // get the error code from the SQL result
  int sqlResponseCode = Integer.parseInt(sqlResult.elementAt(0).
      toString().substring(0,3));
  Purchase_Tickets_Resp_Bean purchaseTicketsRespBean = new
      Purchase_Tickets_Resp_Bean(sqlResponseCode);
  System.out.println("------------------------␣SQL␣resp␣Code␣Purchased
      ␣tickets:␣" + sqlResponseCode);

  if(sqlResponseCode == SQL_Return_Codes.PURCHASE_TICKETS_PRT_OK){

    // removes the error code at the beginning of the SQL result
    // error code is separated with new line by the rest of the
        response
    String msgToBeParsed = sqlResult.elementAt(0).toString().substring
        (2 + Parsing_Constants.NEW_LINE.length());

    // parse the result and get the reservation details
    String[] parsedMsg = msgToBeParsed.split("\n");

    cat.debug("Msg␣to␣be␣parsed\n" + msgToBeParsed);
    /*
     String debugMsg = "";
     for (int i = 0; i<parsedMsg.length; i++)
     debugMsg += i + " = " + parsedMsg[i] + "\n";
     cat.debug(debugMsg);
     */
    String reservationID_Str = parsedMsg[0];
    String price_money_Str = parsedMsg[1];

    // helper class to parse different string formats
    // and return the elements in the string
    HelpingParser parser = new HelpingParser();

    // get the ticket IDs
    double price_money_Double[] = (parser.getElementsAsDouble(
        price_money_Str, "\\|"));
    double totalPrice = price_money_Double[0];
    double leftEmoney = price_money_Double[1];

    int noOfTickets = parsedMsg.length - 2;
```

```
String ticketIDs[] = new String[noOfTickets];
double ticketPrices[] = new double[noOfTickets];

// get the ticket IDs and prices for each ticket
for (int i = 0; i < noOfTickets; i++){
  String[] ticket_price = parsedMsg[i+2].split("\\|");
  ticketIDs[i] = ticket_price[0];
  ticketPrices[i] = Double.parseDouble(ticket_price[1]);
}

// create and set the response bean properties
purchaseTicketsRespBean.setReservationID(reservationID_Str);
purchaseTicketsRespBean.setTotalPrice(totalPrice);
purchaseTicketsRespBean.setLeftEmoney(leftEmoney);
purchaseTicketsRespBean.setNoOfTickets(noOfTickets);
purchaseTicketsRespBean.setTicketIDs(ticketIDs);
purchaseTicketsRespBean.setTicketPrices(ticketPrices);

Purchase_Tickets_Req_Bean reqBean = (Purchase_Tickets_Req_Bean)
    sqlResult.elementAt(1);
// DEBUG
cat.debug(reqBean.toString());

// call the external payment service and pay for the
// purchasd tickets in case totalPrice > 0 and the payment method
    is CARD
// DO NOT DO THIS FOR TICKETS THAT ARE TO BE PAID AT THE CINEMA
boolean payed = false;
if (totalPrice > 0 && reqBean.getPurchaseMethod().toUpperCase().
    equals("CARD")){
  CardValidator cv = new CardValidator();

  // generate the aesKey used for data decryption
  HelpingCrypto helpCrypto = new HelpingCrypto();
  String aesSalt = System.getProperty("aesSalt"); // salt value (
      generated session key)
  String aesToken = System.getProperty("aesToken"); // token (
      userID)
  ParametersWithIV aesKey = helpCrypto.createKey(aesSalt, aesToken)
      ;
  System.out.println("----␣aesSalt␣:" + aesSalt);
  System.out.println("----␣aesToken␣:" + aesToken);
  System.out.println("----␣aes␣key:" + aesKey);

  // get and the decrypt the Credit Card information
  String creditCardType  = new String(helpCrypto.decryptWithAES(
```

```
        aesKey, reqBean.getCreditCardType()));
    String creditCardNo    = new String(helpCrypto.decryptWithAES(
        aesKey, reqBean.getCreditCardNo()));
    String creditCardExpDate = new String(helpCrypto.decryptWithAES(
        aesKey, reqBean.getCreditCardExpDate()));
    String creditCardCW2   = new String(helpCrypto.decryptWithAES(
        aesKey, reqBean.getCreditCardCW2()));

    System.out.println("----␣RespDM␣creditCardType:" + creditCardType
        .trim());
    System.out.println("----␣RespDM␣creditCardNo:" + creditCardNo.
        trim());
    System.out.println("----␣RespDM␣creditCardExpDate:" +
        creditCardExpDate.trim());
    System.out.println("----␣RespDM␣creditCardCW2:" + creditCardCW2.
        trim());

    payed = cv.pay( creditCardType.trim(),
        creditCardNo.trim(),
        creditCardExpDate.trim(),
        creditCardCW2.trim(),
        totalPrice);

    // if an error occured during the payment
    // cancel the reservation and send an erorr
    // message to the user
    if (!payed){
      // 420 = cannot make the payment due to network communication
          problems
      // between the payment service and the cinema controller
      Purchase_Tickets_Resp_Bean respBean =
        new Purchase_Tickets_Resp_Bean(
            Error_Code_Constants.ERROR_WHILE_PAYING);
      cat.debug(respBean.toString());

      // cancel the purchased tickets
      DBTools dbTools = new DBTools();
      Response_Msg_Bean rspMsgBean = dbTools.
          cancelTicketsIfPaymentFails(reqBean, sqlResult, noOfTickets
          , reservationID_Str, ticketIDs);
      System.out.println("------␣CANCELLL␣Respose␣code:\n" +
          rspMsgBean.getResponseCode());

      return respBean;
    }
  }
```

```
  } // end if(sqlResponseCode == SQL_Return_Codes.
      PURCHASE_TICKETS_PRT_OK)

  // DEBUG
  cat.debug(purchaseTicketsRespBean.toString());
  return purchaseTicketsRespBean;

} // end setResponsePurchasedTicketsBean()


/**
 * Creates the Canceled_Tickets_Resp_Bean and sets up the canceled
     tickets details
 * that are to be sent to the MIDlet as a Java Bean Object
 *
 * @param sqlResult The result of the SQL query to get the canceled
     ticket details
 * @return The Response_Msg_Bean containing the canceled ticket details
 */
public Response_Msg_Bean setResponseCancelTicketsBean(Vector sqlResult)
     {

  // get the error code from the SQL result
  int sqlErrorCode = Integer.parseInt(sqlResult.elementAt(0).toString()
      .substring(0, 3));

  Response_Msg_Bean respBean = new Response_Msg_Bean(sqlErrorCode);
  respBean.setMsg("Canceled_Tickets␣Msg␣from␣response␣bean␣from␣the␣
      servlet");

  // DEBUG
  cat.debug("-----␣Canceled_Tickets_Resp_Bean␣-----\n" + respBean.
      toString());
  return respBean;

} // end setResponseCancelTicketsBean()


/**
 * Creates the Response_Msg_Bean and sets up the response code
 * that is to be sent to the MIDlet as a Java Bean Object
 *
 * @param sqlResult The result of the SQL query to get the requested
 *      movie details
 * @return The Response_Msg_Bean containing the response code of
     running the
 *        SQL statement to rate the movie
```

```
 */
public Response_Msg_Bean setResponseRateMovieBean(Vector sqlResult) {

  // get the error code from the SQL result
  int statusCode = Integer.parseInt(sqlResult.elementAt(0).toString().
      substring(0, 3));
  Response_Msg_Bean respBean = new Response_Msg_Bean();
  respBean.setResponseCode(statusCode);
  respBean.setMsg("Rate␣Movie␣Msg␣from␣response␣bean␣from␣the␣servlet")
      ;

  // DEBUG
  cat.debug( "-----␣Rate␣Movie␣----" + respBean.toString());

  return respBean;

} // end setResponseRateMovieBean()


/**
 * Creates the Movie_Details_Resp_Bean and sets up the movie details
 * that are to be sent to the MIDlet as a Java Bean Object
 *
 * @param sqlResult The result of the SQL query to get the requested
 *       movie details
 * @return The Movie_Details_Resp_Bean containing the requested movie
 *       details
 */
public Response_Msg_Bean setResponseMovieDetailsBean(Vector sqlResult){

  // get the error code from the SQL result
  int sqlResponseCode = Integer.parseInt(sqlResult.elementAt(0).
      toString().substring(0,3));
  Movie_Details_Resp_Bean movDetRespBean = new Movie_Details_Resp_Bean(
      sqlResponseCode);

  // only in case that the movie si found get the details
  if (sqlResponseCode == SQL_Return_Codes.MOVIE_DETAILS_PRT_OK){

    // removes the error code at the beginning of the SQL result
    // error code is separated with new line by the rest of the
        response
    String msgToBeParsed = sqlResult.elementAt(0).toString().substring
        (2 + Parsing_Constants.NEW_LINE.length());
    HelpingParser parser = new HelpingParser();

    // parse the result and get the movie details
```

```
      String[] parsedMsg = msgToBeParsed.split("\n");

      cat.debug("Msg␣to␣be␣parsed\n" + msgToBeParsed);
      /*
       String debugMsg = "";
       for (int i = 0; i<parsedMsg.length; i++)
       debugMsg += i + " = " + parsedMsg[i] + "\n";
       cat.debug(debugMsg);
       */

      movDetRespBean.setMovieID               (parsedMsg[0]);
      movDetRespBean.setMovieName             (parsedMsg[1]);
      movDetRespBean.setMovieDuration         (parsedMsg[2]);
      movDetRespBean.setMovieGenre            (parsedMsg[3]);
      movDetRespBean.setMovieParentClassification (parsedMsg[4]);
      movDetRespBean.setMovieLanguage         (parsedMsg[5]);
      movDetRespBean.setMovieYear             (parsedMsg[6]);
      movDetRespBean.setMovieCountry          (parsedMsg[7]);
      movDetRespBean.setMovieUserRating       (parsedMsg[8]);
      movDetRespBean.setMovieDirector         (parsedMsg[9]);
      movDetRespBean.setMovieActors           (parsedMsg[10]);
      movDetRespBean.setMovieDescription      (parsedMsg[11]);
      movDetRespBean.setMoviePoster           (parser.getMoviePoster(
          sqlResult));

    } // end if (sqlResponseCode == SQL_Return_Codes.MOVIE_DETAILS_PRT_OK
        )

    // DEBUG
    cat.debug(movDetRespBean.toString());
    return movDetRespBean;

  } // end setResponseMovieDetailsBean()


} // end class

package cinemaservice.model.interfaces;

import java.sql.PreparedStatement;
import java.sql.SQLException;

import cinemaservice.exceptions.CinemaServiceException;


/**
 * Interface for the Client To Facade functionality
 *
```

```
 * @author Mihai Balan - s031288
 *
 */
public interface ClientToFacadeInterface {

  public void getReqBeanData(PreparedStatement pqPsqlStmt, Object reqBean
      , String prtStep) throws SQLException, CinemaServiceException, java
      .text.ParseException;

}

package cinemaservice.model.interfaces;

import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.text.ParseException;

import cinemaservice.model.beans.requestBeans.Authentication_1_Req_Bean;
import cinemaservice.model.beans.requestBeans.Cancel_Tickets_Req_Bean;
import cinemaservice.model.beans.requestBeans.Change_Password_Req_Bean;
import cinemaservice.model.beans.requestBeans.Cinema_Hall_Conf_Req_Bean;
import cinemaservice.model.beans.requestBeans.Find_Movies_Req_Bean;
import cinemaservice.model.beans.requestBeans.Purchase_Tickets_Req_Bean;
import cinemaservice.model.beans.requestBeans.Rate_Movie_Req_Bean;
import cinemaservice.model.beans.requestBeans.Movie_Details_Req_Bean;
import cinemaservice.model.beans.requestBeans.Reject_Payment_Req_Bean;
import cinemaservice.model.beans.requestBeans.
    Select_Deselect_Seats_Req_Bean;


/**
 * Interface for the Request Data Model fucntionality
 *
 * @author Mihai Balan - s031288
 *
 */
public interface RequestModelInterface {

  public void getAuth1ReqBeanData(PreparedStatement pqPsqlStmt,
      Authentication_1_Req_Bean auth1Bean) throws SQLException;

  public void getChangePasswordReqBeanData(PreparedStatement pqPsqlStmt,
      Change_Password_Req_Bean chgPswdBean) throws SQLException;

  public void getFindMoviesReqBeanData(PreparedStatement pqPsqlStmt,
      Find_Movies_Req_Bean findMoviesReqBean) throws SQLException,
      ParseException;
```

```
  public void getCinemaHallConfReqBeanData(PreparedStatement pqPsqlStmt,
      Cinema_Hall_Conf_Req_Bean cinHallConfReqBean) throws SQLException;

  public void getSelectDeselectSeatsReqBeanData(PreparedStatement
      pqPsqlStmt, Select_Deselect_Seats_Req_Bean selDeselSeatsReqBean)
      throws SQLException;

  public void getRejectPaymentReqBeanData(PreparedStatement pqPsqlStmt,
      Reject_Payment_Req_Bean rejReservationReqBean) throws SQLException;

  public void getPurchaseTicketsReqBeanData(PreparedStatement pqPsqlStmt
      , Purchase_Tickets_Req_Bean rejReservationReqBean) throws
      SQLException, ParseException;

  public void getCancelTicketsReqBeanData(PreparedStatement pqPsqlStmt,
      Cancel_Tickets_Req_Bean cancelTicketsReqBean) throws SQLException;

  public void getRateMovieReqBeanData(PreparedStatement pqPsqlStmt,
      Rate_Movie_Req_Bean rateMovBean) throws SQLException;

  public void getMovieDetailsReqBeanData(PreparedStatement pqPsqlStmt,
      Movie_Details_Req_Bean movDetailsReqBean) throws SQLException;

}

package cinemaservice.model.interfaces;

import java.sql.SQLException;
import java.text.ParseException;
import java.util.Vector;

import javax.naming.NamingException;

import cinemaservice.exceptions.CinemaServiceException;
import cinemaservice.model.beans.responseBeans.Response_Msg_Bean;


/**
 * Interface for the Response Data Model fucntionality
 *
 * @author Mihai Balan - s031288
 *
 */
public interface ResponseModelInterface {

  public Response_Msg_Bean setResponseAuth1Bean(Vector sqlResult);

  public Response_Msg_Bean setResponseChangePasswordBean(Vector sqlResult
```

```
      );

  public Response_Msg_Bean setResponseFindMoviesBean(Vector sqlResult);

  public Response_Msg_Bean setResponseBackgroundCinemaHallConfBean(Vector
       sqlResult);

  public Response_Msg_Bean setResponseSelectDeselectSeats(Vector
      sqlResult);

  public Response_Msg_Bean setResponseCinemaHallConfBean(Vector sqlResult
      );

  public Response_Msg_Bean setResponseRejectPaymentBean(Vector sqlResult)
      ;

  public Response_Msg_Bean setResponsePurchasedTicketsBean(Vector
      sqlResult) throws SQLException, NamingException, ParseException,
      CinemaServiceException;

  public Response_Msg_Bean setResponseCancelTicketsBean(Vector sqlResult)
      ;

  public Response_Msg_Bean setResponseRateMovieBean(Vector sqlResult);

  public Response_Msg_Bean setResponseMovieDetailsBean(Vector sqlResult);

}

package cinemaservice.movieposter;

import java.io.*;
import java.sql.*;

public class DBConnTools
{
  private Connection con;

  public DBConnTools() throws ClassNotFoundException, SQLException{

    con = connect();

  } // end DBConnTools()

  private static Connection connect() throws ClassNotFoundException,
      SQLException{

    String url = "jdbc:postgresql://localhost:5432/postgres";
```

```
  String user = "zeratul";
  String passwd = "ericsson";

  Class.forName("org.postgresql.Driver");
  return DriverManager.getConnection(url,user,passwd);

}// end connect()


public int setMoviePoster(File file, FileInputStream poster, int
    movieID) throws SQLException {


  //Connection conn = null;
  PreparedStatement pgPsqlStmt = null;
  int res = 0;
  String sqlStmt = "UPDATE␣cinema.movies␣SET␣poster␣=␣?␣WHERE␣movieid
      ␣=␣?;";

  try{

    // preapare the SQL statement
    pgPsqlStmt = con.prepareStatement(sqlStmt);
    System.out.println("1");
    System.out.println(file.length());
    // set the parameters to execute the query
    pgPsqlStmt.setBinaryStream(1, poster, (int)file.length());
    System.out.println("1a");
    pgPsqlStmt.setInt(2, movieID);
    System.out.println("2");
    // execute the prepared statement
    res = pgPsqlStmt.executeUpdate();
    System.out.println("3");
    // close the prepared statment
    pgPsqlStmt.close();
    System.out.println("4");
    // return the connection to the pool of connections
    con.close();
    System.out.println("5");
    return res;

  }
  // perform any clean up in case any connection, statement remains
      opened and not used
  finally {
    try {
      if (pgPsqlStmt != null && !con.isClosed())
```

```
        pgPsqlStmt.close();

    } catch (SQLException sqle1) {
      System.out.println("SQL␣exception␣" +
          "when␣trying␣to␣close␣the␣statement␣in␣FINALLY␣clause␣...␣"
          + sqle1.getMessage());
    }
    try {
      if (!con.isClosed())
        con.close();

    } catch (SQLException sqle2) {
      System.out.println("SQL␣exception␣" +
          "when␣trying␣to␣close␣the␣connection␣in␣FINALLY␣clause␣...␣"
          + sqle2.getMessage());
    }
  }//  end try - catch - finally

} // end setMoviePoster()


public byte[] getMoviePoster(int movieID) throws SQLException {


  //Connection conn = null;
  PreparedStatement pgPsqlStmt = null;
  ResultSet res = null;
  String sqlStmt = "SELECT␣poster␣FROM␣cinema.movies␣WHERE␣movieid
      ␣=␣?;";
  byte[] imgBytes= null;

  try{

    // preapare the SQL statement
    pgPsqlStmt = con.prepareStatement(sqlStmt);
    System.out.println("1");
    // set the parameters to execute the query
    pgPsqlStmt.setInt(1, movieID);
    System.out.println("2");
    // execute the prepared statement
    res = pgPsqlStmt.executeQuery();
    System.out.println("3");

    if (res != null) {
        while(res.next()) {
            imgBytes = res.getBytes(1);
            // use the stream in some way here
```

```
          }
          res.close();
       }

      // close the prepared statment
      pgPsqlStmt.close();
      System.out.println("4");
      // return the connection to the pool of connections
      con.close();
      System.out.println("5");
      return imgBytes;

    }
    // perform any clean up in case any connection, statement remains
        opened and not used
    finally {
      try {
        if (pgPsqlStmt != null && !con.isClosed())
          pgPsqlStmt.close();

      } catch (SQLException sqle1) {
        System.out.println("SQL␣exception␣" +
            "when␣trying␣to␣close␣the␣statement␣in␣FINALLY␣clause␣...␣"
            + sqle1.getMessage());
      }
      try {
        if (!con.isClosed())
          con.close();

      } catch (SQLException sqle2) {
        System.out.println("SQL␣exception␣" +
            "when␣trying␣to␣close␣the␣connection␣in␣FINALLY␣clause␣...␣"
            + sqle2.getMessage());
      }
    }//  end try - catch - finally

  } // end setMoviePoster()


} // end class

package cinemaservice.movieposter;

import java.awt.GridLayout;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.*;
import java.lang.reflect.Array;
```

```
import java.sql.SQLException;
import java.util.Vector;

import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;

public class MoviePosterHelper {


  public static void main(String[] args) {

    int movID = 1;

    //C:/Documents and Settings/Zeratul/workspace/J2ME/Cinema_Controller/
        src/cinemaservice/movieposter/pics/mov.jpg
    File file = new File("C:/Documents␣and␣Settings/Zeratul/workspace/
        J2ME/Cinema_Controller/src/cinemaservice/movieposter/pics/mov" +
        movID + ".jpg");
    int result = 0;
    byte[] image = null;

    try{

      // insert the movie poster in the DB
      FileInputStream fis = new FileInputStream(file);
      DBConnTools posterTools = new DBConnTools();
      result = posterTools.setMoviePoster(file, fis ,movID);

      // get the movie poster from the DB
      posterTools = new DBConnTools();
      image = posterTools.getMoviePoster(movID);

      Vector v = new Vector();
      v.addElement(image);
      System.out.println("-----␣" + v.elementAt(0));

      int l = Array.getLength(v.elementAt(0));
      System.out.println("-------␣length␣" + l);
      byte[] temp = new byte[l];
      //Object x = Array.get(Array.get(v.elementAt(0),0));
      //System.arraycopy(x,0,temp,0,l-1);
      for (int i = 0; i < l; i++){

        temp[i] = Array.getByte(v.elementAt(0),i);

      }
```

```
      System.out.println("-------␣Test␣");
      System.out.println(temp.toString());

      ImageIcon img = new ImageIcon(temp);
      JLabel photo = new JLabel(img);
      JFrame frame = new JFrame("picture");

      frame.addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent e){
          System.exit(0);
        }
      });

      frame.setLayout(new GridLayout(1,1));
      frame.setContentPane(photo);
      frame.pack();
      frame.setVisible(true);


    } catch(FileNotFoundException fne){
      System.out.println("--␣" + fne.getMessage());
    } catch(ClassNotFoundException cne){
      System.out.println("--␣" + cne.getMessage());
    } catch(SQLException sql){
      System.out.println("----␣" + sql.getMessage());
    //} catch(FileNotFoundException fnf){
    //  System.out.println("------ " + fnf.getMessage());
    }

    //System.out.println("Result code from updating movie poster: " +
        result);
  }
}

package cinemaservice.servlets.controller;

import cinemaservice.constants.Error_Code_Constants;
import cinemaservice.constants.Protocol_Step_Constants;
import cinemaservice.exceptions.CinemaServiceException;

import java.io.IOException;
import java.util.Date;

import javax.servlet.*;
import javax.servlet.http.*;

import org.apache.log4j.*;
```

```
/**
 * This is the FACADE and Central CONTROLLER of the Cinema Controller
     System (CCS).
 * It hides the CCS complexity from the client side and
 * provides simple methods for the client to interact with the CCS.
 *
 * It receives the request from the client i.e. PROTOCOL_STEP + a REQUEST
      BEAN.
 * The Request Bean contains the parameters for executing the SQL Queries
 * on the server side. There are several request beans such as:
     Movie_Details_Resp_Bean,
 * Change_Password_Req_Bean, etc.
 * Function of the PROTOCOL_STEP, dispatch the requerst further to the
 * particular worker servlet that implements that functionality.
 *
 * The implementation is using the FACADE, MODEL-VIEW-CONTROLLER, and
     REFACTORING Design Patterns.
 *
 * @author Mihai Balan - s031288
 *
 */
public class Cinema_Central_Controller_Servlet extends HttpServlet
    implements Servlet {


  // ============================================================================

  //                            DECLARATIONS
  // ============================================================================


  /**
   * Serial Version of the Servlet
   */
  private static final long serialVersionUID = 1L;

  /**
   * Initiate a custom logging category
   */
  private static Category cat = Category.getInstance(
      Cinema_Central_Controller_Servlet.class.getName());



  // ============================================================================
```

```
//                              METHODS
// ============================================================================


/**
 * Initialize the Cinema_Central_Controller_Servlet
 * If the initialization fails, the servlet is not started
 *
 * @throws ServletException in case initialization fails
 */
public void init(ServletConfig config) throws ServletException {
  // Store the ServletConfig object and log the initialization
  super.init(config);

} // end init()


/**
 * Calls doPost() to deal with GET() requests from the client side
 *
 * @param request Client Request to the servlet
 * @param response Servlet Response to the client
 */
protected void doGet(HttpServletRequest request, HttpServletResponse
    response)
throws ServletException, IOException {
  doPost(request, response);

} // end doGet()


/**
 * Deals with POST requests from the client side and
 * calls the processRequest method to process the client request
 *
 * @param request Client Request to the servlet
 * @param response Servlet Response to the client
 */
protected void doPost(HttpServletRequest request, HttpServletResponse
    response)
throws ServletException, IOException {

  try {
    processRequest(request, response);

  }catch (ServletException se){
    cat.warn("ServletException: " + se.toString() + ": " + se.
```

```
        getMessage());
    response.setStatus(Error_Code_Constants.INTERNAL_SERVER_ERROR);
    throw se;

  }catch (IOException ioe){

    cat.warn("IOException:␣" + ioe.toString() + ":␣" + ioe.getMessage()
        );
    response.setStatus(Error_Code_Constants.INTERNAL_SERVER_ERROR);
    throw ioe;

    // catch user defined exception
  } catch (CinemaServiceException cse){
    cat.warn("CinemaServiceException:␣" + cse.getMessage());
    response.setStatus(cse.getValue());

  } // end try-catch

} // end doPost()



/**
 * Retrieves the PROTOCO_STEP from the client request.
 * Function of the PROTOCO_STEP dispatchs the client request to the
 * worker servlet that deals with that particular PROTOCO_STEP.
 *
 * It also forwards the response from the targeted servlet to the
     MIDlet.
 *
 * @param request Client Request to the servlet
 * @param response Servlet Response to the client
 *
 * @throws IOException
 * @throws ServletException
 * @throws
 */
private void processRequest(HttpServletRequest request,
    HttpServletResponse response)
throws ServletException, IOException, CinemaServiceException {

  // the protocol step used to identify the actions from the client
      side
  String protocolStep = request.getParameter("protocol");

  // push the client ID to be logged
  NDC.push(request.getRemoteAddr());
```

```
cat.info("------------------------------------------------------------------------"
    );
cat.info("------------------------------------------------------------------------"
    );
cat.info("Cinema␣Central␣Controller␣Servlet␣requested␣by␣(" + request
    .getRemoteAddr()+ ").");
cat.debug("Protocol␣Step:␣" + protocolStep);

// construct the path to the worker servlet i.e.
// the servlet that deals with the particular client request
// e.g. Find_Movies protocol step targets FindMovieServlet
String targetServlet = "/cinemaservice/servlets/workers/";

// beginning of the AUTHENTICATION procedure
if (protocolStep.equals(Protocol_Step_Constants.
    PRT_STEP_AUTHENTICATION_1)) /* 1 */
  targetServlet += "Authentication_Servlet_1";

//  1st and 2nd steps in the AUTHENTICATION procedure
else if (protocolStep.equals(Protocol_Step_Constants.
    PRT_STEP_AUTHENTICATION_2)) /* 2 */
  targetServlet += "Authentication_Servlet_2";

// CHANGE PASSWORD step
else if (protocolStep.equals(Protocol_Step_Constants.
    PRT_STEP_CHANGE_PASSWORD)){ /* 3 */
  targetServlet += "Change_Password_Servlet";}

// FIND MOVIES step
else if (protocolStep.equals(Protocol_Step_Constants.
    PRT_STEP_FIND_MOVIES)) /* 4 */
  targetServlet += "Find_Movies_Servlet";

// SELECT SHOW AND DISPLAY CINEMA HALL CONFIGURATION step
else if (protocolStep.equals(Protocol_Step_Constants.
    PRT_STEP_SELECT_SHOW_AND_DISPLAY_CINEMA_HALL_CONF)) /* 5 */
  targetServlet += "Select_Show_Servlet";

// BACKGROUND DISPLAY CINEMA HALL UPDATE step
else if (protocolStep.equals(Protocol_Step_Constants.
    PRT_STEP_BACKGROUND_CINEMA_HALL_UPDATE)) /* 6 */
  targetServlet += "Background_Hall_Update_Servlet";

// SELECT - DESELECT SEATS step
else if (protocolStep.equals(Protocol_Step_Constants.
    PRT_STEP_SELECT_DESELECT_SEATS)) /* 7 */
  targetServlet += "Select_Deselect_Seats_Servlet";
```

```
// PURCHASE TICKETS step
else if (protocolStep.equals(Protocol_Step_Constants.
    PRT_STEP_PURCHASE_TICKETS)) /* 8 */
  targetServlet += "Purchase_Tickets_Servlet";

// CANCEL RESERVATION/TICKETS step
else if (protocolStep.equals(Protocol_Step_Constants.
    PRT_STEP_CANCEL_TICKETS)) /* 9 */
  targetServlet += "Cancel_Tickets_Servlet";

// REJECT PAYMENT step
else if (protocolStep.equals(Protocol_Step_Constants.
    PRT_STEP_REJECT_PAYMENT)) /* 10 */
  targetServlet += "Reject_Payment_Servlet";

// RATE MOVIE step
else if (protocolStep.equals(Protocol_Step_Constants.
    PRT_STEP_RATE_MOVIE)) /* 11 */
  targetServlet += "Rate_Movie_Servlet";

// MOVIE DETAILS step
else if (protocolStep.equals(Protocol_Step_Constants.
    PRT_STEP_MOVIE_DETAILS)) /* 12 */
  targetServlet += "Movie_Details_Servlet";

//if unknown protocol step send the RESPONSE_CODE to the client
// as INVALID_PROTOCOL_STEP and throw corresponding Exception on the
    server side
else {
    /* 13 */
  response.setStatus(Error_Code_Constants.INVALID_PROTOCOL_STEP);

  throw new CinemaServiceException(
      "INVALID␣PROTOCOL␣STEP␣sent␣by␣the␣MIDLET",
      protocolStep,
      "Cinema_Central_Controller_Servlet",
      "processRequest()",
      "1",
      Error_Code_Constants.INVALID_PROTOCOL_STEP);

} // end if (protocol_step)


// if the protocol step is correct dispatch the request to
// the worker servlet that is to deal with the particular user
    request
```

```
    RequestDispatcher target = request.getRequestDispatcher(targetServlet
        );

    // forward the client request to the worker servlet
    target.forward(request, response);

  } // end processrequest()



  /**
   * Returns information about the Cinema Central Controller Servlet
   *
   * @return Information about the Cinema Central Controller Servlet
   */
  public String getServletInfo() {
    return "Hello␣from␣the␣Cinema␣Central␣Controller␣Servlet␣on␣" + new
        Date();

  } // end getServletInfo()

} // end class

package cinemaservice.servlets.workers;

import java.io.*;
import java.sql.SQLException;
import java.text.ParseException;
import java.util.Random;
import java.util.Vector;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.sql.DataSource;

import org.apache.log4j.Category;

import org.bouncycastle.crypto.CryptoException;

import cinemaservice.beans.tools.SQL_Operations_Bean;
import cinemaservice.constants.Error_Code_Constants;
import cinemaservice.constants.SQL_Return_Codes;
import cinemaservice.exceptions.CinemaServiceException;
import cinemaservice.model.beans.requestBeans.Authentication_1_Req_Bean;
import cinemaservice.model.beans.responseBeans.Authentication_1_Resp_Bean
```

```
    ;
import cinemaservice.model.facade.FacadeToModel;
import cinemaservice.model.facade.HelpingCrypto;


/**
 * This servlet performs encrypted communication with a MIDlet.
 * The data is transfered between them as binary.
 * It decrypts the MIDlet's request and sends back
 * an encrypted answer to the MIDlet.
 *
 * It performs the first step of the authentication protocol i.e.
 * it receives an encrypted chalange consisting of
 * credentials (user name + password), address of the targeted system
 * (Authentication_Servlet_2), and a random number.
 *
 * It decrypts the chalange, check the credentials against the DB, and
 * if user is authnticated against the DB retrieves userID
 * and an OTP (token) from the DB.
 *
 * It checks if the random number sent from the mobile device has not
 * been used before and generate a salt value that is to be used
 * by the mobile device to create the session key.
 *
 * Then, it encrypts the salt value, and the token with the key preshered
 * with Authentication_Servlet_2. Then, it encrypts the random no.
     received from the mobile,
 * the address of the Authentication_Servlet_2, the serial no. of mobile
     phone,
 * and the message encrypted with Authentication_Servlet_2's key, with
     the key
 * preshered with the MIDlet on the mobile phone.
 *
 * The encrypted message is then sent back to the mobile device.
 *
 * If the user is not authenticated against the DB, or any other error
 * occurs during the first step in the authentication process, an erorr
 * msg is sent back to the mobile device.
 *
 * @author Mihai Balan - s031288
 * @version 3.0
 */
public class Authentication_Servlet_1 extends HttpServlet {

  private static final long serialVersionUID = 2L;

  /**
```

```
 * Initiate a custom logging category
 */
private static Category cat = Category.getInstance(
    Authentication_Servlet_1.class.getName());

// Key and ServKey can be preshered by using real keys stored on the
    server side,
// DB, or binary or xml files. This s only for demonstration purposses.
//The key for the communication with the mobile phone.
String key = "12345678";
//The new key has been introduced as the private key of
    Authentication_Servlet_2
String servKey = "87654321";

//It is aditionally necessary to introduce the Authentication_Servlet_2
     url
private String servUrl = "http://127.0.0.1:9080/Cinema_Controller/
    cinemaservice/servlets/controller/Cinema_Central_Controller_Servlet
    ?protocol=AT2";

/**
 * The datasource object for obtainig the connection from the pool
 */
private DataSource datasource = null;

/**
 * Initialize the Authentication_Servlet_1 by initializing the
 * pool of PostgreSQL connections usiong JNDI.
 * The configuration files for the conection pool are:
 * META-INF/context.xml and WEB_INF/web.xml
 * If the initialization fails, the servlet is not started
 *
 * @throws ServletException in case initialization fails
 */
public void init(ServletConfig config) throws ServletException {

  try {
    // Create a datasource for pooled connections.
    // Use JNDI to retrieve the DataSource object defined
    // in the Tomcat and application configuration *.xml files
    // i.e.WEB_INF/web.xml and META_INF/context.xml in the directory
    // where the application is deployed on Tomcat
    Context initCtx = new InitialContext();
    Context envCtx = (Context) initCtx.lookup("java:comp/env");
    datasource = (DataSource) envCtx.lookup("jdbc/postgres");
  } catch (NamingException ne) {
    cat.fatal("----␣Authentication_Servlet_1␣-␣JNDI␣Naming␣Exception␣
```

```
        when␣trying␣to␣acquire␣the␣DataSource␣object\n" +
       ne.getMessage() + "\n" + ne.getStackTrace());
  }

} // end init()


public void doGet(HttpServletRequest request,HttpServletResponse
    response)
throws IOException, ServletException {
  doPost(request, response);

} // end doGet()

public void doPost(HttpServletRequest request, HttpServletResponse
    response)
throws IOException, ServletException {

  // for encryption - decryption opperations
  HelpingCrypto helpCrypto = new HelpingCrypto();

  // the byte array that holds the request
  byte[] reqData = null;

  // the decrypted MIDlet's request
  String decReq = "";

  // the servlet response before encryption
  byte[] res = null;

  // the resonse to send back to the MIDlet
  byte[] resData = null;

  // get the length of the request
  int len = request.getContentLength();

  // open an input stram to process the request
  ServletInputStream sis = request.getInputStream();
  if (len != -1){
    // if length is available read the data into an array
    reqData = new byte[len];
    sis.read(reqData);
  }
  else{
    // If length is not available read data from the input stream
    // one character at a time and store the answer in a byte array.
    ByteArrayOutputStream tmp = new ByteArrayOutputStream();
```

```
  int ch;
  // read char by char from the ServletInputStream until EOF
  while( ( ch = sis.read() )!= -1 ){
    // and write to the ByteArrayOutputStream
    tmp.write( ch );
  }
  // move the data to the reqData
  reqData = tmp.toByteArray();
}

try{
  decReq = helpCrypto.decryptMessage(key, reqData);

  // set the response code to OK in case the decryption is OK
  response.setStatus( HttpServletResponse.SC_OK );

  // set up the authentication token and response to be sent to the
      MIDlet
  byte[] authentication = returnAuthenticationResponse(helpCrypto,
      decReq, request.getParameter("protocol"));

  // if an invalid user name and password are submited
  // an "Authentication Failure" msg is sent back to the midlet
  // Else, "Authentication Success" msg and token are sent back to
      the midlet
  if (authentication.equals((String.valueOf(SQL_Return_Codes.
      Authentication_E_MONEY_PRT_USER_NOT_AUTHENTICATED)).getBytes()))
          {
    System.out.println("-------==============␣␣not␣authenticated␣");

    res = (String.valueOf(SQL_Return_Codes.
        Authentication_E_MONEY_PRT_USER_NOT_AUTHENTICATED)).getBytes()
            ;
  }
  else {
    System.out.println("-------==============␣␣Authenticated␣");
    res = authentication;
  }
}catch(CryptoException cre){
  //set the response code to ERROR in case of decryption errors
  response.setStatus( Error_Code_Constants.ERROR_IN_DECRYPTING);
  // set up the response to the MIDlet
  res = ("Decryption␣error␣on␣the␣server␣side:␣" + cre.getMessage()).
      getBytes();

} catch (CinemaServiceException cse){
  cat.warn("CinemaServiceException:␣" + cse.getMessage());
```

```
  response.setStatus(cse.getValue());

} catch (SQLException sqle) {
  cat.warn("SQLException:␣"+ sqle.getSQLState()
      + "␣-␣" + sqle.getErrorCode() + "␣-␣" + sqle.toString());
  response.setStatus(Error_Code_Constants.ERROR_IN_SQL);

} catch (ParseException pe) {
  cat.warn("ParseException:␣"+ pe.getMessage());
  response.setStatus(Error_Code_Constants.ERROR_IN_SQL);
}

// set the content type
response.setContentType("application/octet-stream");

try{
  // encrypt the response to be send back to the MIDlet
  resData = helpCrypto.encryptMessage(key, res);

  System.out.println("<----------------Decryption␣test
      ---------------------->");
  // Decryprion of thereceived message
  byte[] resDec1 = helpCrypto.decrypt(key,resData);

  //System.out.println("The part to send: " + resTok1[4].getBytes());

  System.out.println("<------------------------------------------------------------------------>"
      );
  System.out.println("The␣message␣decrypted:␣" + new String(resDec1))
      ;
  System.out.println("The␣length␣of␣the␣String:␣" + (new String(
      resDec1)).length() + ",␣Size␣of␣byte:␣" + resDec1.length);

} catch(CryptoException ce){
  String resErr ="Encryption␣error␣on␣the␣server␣side:␣" + ce.
      getMessage();
  resData = resErr.getBytes();
}

// set the Content length property
response.setContentLength(resData.length);

// get a ServletOutputStream to send the response to the MIDlet as
    binary data
ServletOutputStream sos = response.getOutputStream();
sos.write(resData);
sos.flush();
```

```
}// end doPost()


/**
 * Checkif the user is authenticated against the DB
 * and creates the rest of the message that is to be sent to
 * the mobile device
 *
 * @param request The decrypted request
 * @param prtStep The protocl Step of the application
 * @return The encrypted message to be sent to the mobile device or
     error
 *
 * @throws SQLException
 * @throws ParseException
 * @throws CinemaServiceException
 */
protected byte[] returnAuthenticationResponse(HelpingCrypto helpCrypto
    , String request, String prtStep) throws SQLException,
    ParseException, CinemaServiceException{

  //First we have to check the authenticity of the person
  String[] tokenized = null;
  tokenized = request.split(";");
  byte[] conc = null;
  //According to the protocol, the authentication credentials
  //should be in the first two cells of the array
  String credentials = "" + tokenized[0] + ":" +tokenized[1];
  System.out.println("Creadentials:␣" + credentials);

  // Perform the authentication against the DB
  Vector sqlResult = new Vector();
  SQL_Operations_Bean sqlOpBean = new SQL_Operations_Bean();
  Authentication_1_Req_Bean requestBean = new Authentication_1_Req_Bean
      ();
  requestBean.setUserName(tokenized[0]);
  requestBean.setPassword(tokenized[1]);
  sqlOpBean.setPooledSource(datasource);
  sqlOpBean.setSQLStatement("SELECT␣*␣FROM␣cinema.Authenticate_E_Money
      (?,␣?)");
  sqlOpBean.setSQLParameters(requestBean, prtStep);
  sqlResult = sqlOpBean.executeSQL();
  String sqlErrorCode = sqlResult.elementAt(0).toString().substring
      (0,3);
  cat.debug("Error␣code:␣" + sqlErrorCode);
  String authentication = returnToken(sqlResult);
```

```
String eMoney = returnEmoney(sqlResult);
System.out.println("----------␣User␣authentication␣1:␣" +
    sqlErrorCode);

System.out.println("Authentication␣token:␣" + authentication);
if (!authentication.equals(
    String.valueOf(
        SQL_Return_Codes.
        Authentication_E_MONEY_PRT_USER_NOT_AUTHENTICATED))){
  //We have the authentication token in the returned string
  //We shall store the random and the target address
  String random = tokenized[2];
  String target = tokenized[3];

  // Check if the request had the desired form
  if (target.equals(servUrl)){

    //Generate the session key
    String sessionKey = generateKey();
    System.out.println("Generated␣Session␣key:␣" + sessionKey);
    //The response should have the following form:
    //  EA(RA,B,K, EB(K,A))
    String forBob = sessionKey + ";" + authentication;
    System.out.println("Info␣for␣BOB:␣" + forBob);
    String replayMsg = "";
    try {
      byte[] bytesFB = helpCrypto.encryptMessage(servKey,forBob);

      System.out.println("Encrypted␣Info␣for␣Bob:␣" + bytesFB.toString
          ());
      System.out.println("␣<<<=========␣For␣Bobby:␣" + helpCrypto.
          decryptMessage(servKey, bytesFB));
      String[] uid = authentication.split(":");
      replayMsg = eMoney + ";" + random + ";" +
      target + ";" + uid[0] + ";" +
      sessionKey + ";";
      byte[] prefix = replayMsg.getBytes();
      conc = new byte[bytesFB.length + replayMsg.length()];
      System.arraycopy(prefix,0,conc,0,prefix.length);
      System.arraycopy(bytesFB,0,conc,prefix.length,bytesFB.length);
      System.out.println("The␣response␣for␣the␣mobile␣app:␣" + new
          String(conc));

    } catch (CryptoException e) {
      System.out.println("Error␣in␣encryption␣of␣the␣message␣for␣BOB
          ␣..");
      e.printStackTrace();
```

```
      }
      return conc;
    }
    else {
      return (String.valueOf(SQL_Return_Codes.
          Authentication_E_MONEY_PRT_USER_NOT_AUTHENTICATED)).getBytes()
              ;
    } // end if (target.equals(servUrl))

  }else {
    return authentication.getBytes();

  } // end if (!authentication.equals(...))

} // end returnAuthenticationResponse()

/**
 * Generate the session key
 *
 * @return Session key
 */
public String generateKey(){
  Random rand = new Random();
  long myRandom = rand.nextLong();
  String theRandom = "" + myRandom;
  return theRandom;

}// end generateKey()


/**
 * Creates the token that is a part of the message to be sent back to
     the MIDlet
 *
 * @param authRequest The Authentication Request sent by the MIDlet to
     the AuthServlet
 * @return Returns an authentication answer i.e. "failure:failure" or
     an auth
 *     token in case of sucessful authentication
 */
protected String returnToken(Vector sqlResult){

  FacadeToModel facade = new FacadeToModel();
  Authentication_1_Resp_Bean respBean = (Authentication_1_Resp_Bean)
      facade.setResponseAuth1Bean(sqlResult);

  if(respBean.getResponseCode() == SQL_Return_Codes.
```

```
       Authentication_E_MONEY_PRT_OK){

    String token = respBean.getUserID() + ":" + respBean.getRandomID();
    System.getProperties().put("token", token);
    System.out.println("--------␣Authentication␣Token␣from␣the␣DB:␣" +
        token);
    return token;

  } else{
    System.out.println("--------␣User␣not␣authenticated␣agasint␣the␣DB
        :␣failure:failure␣401");
    return String.valueOf(SQL_Return_Codes.
        Authentication_E_MONEY_PRT_USER_NOT_AUTHENTICATED);
  }

} // end returnToken()

/**
 * Returns the amount of e-money for the given user
 *
 * @param sqlResult The result of executing the authentication against
     the DB
 *
 * @return The amount of e-money of the given user
 */
protected String returnEmoney(Vector sqlResult){

  FacadeToModel facade = new FacadeToModel();
  Authentication_1_Resp_Bean respBean = (Authentication_1_Resp_Bean)
      facade.setResponseAuth1Bean(sqlResult);

  if(respBean.getResponseCode() == SQL_Return_Codes.
      Authentication_E_MONEY_PRT_OK){

    String eMoney = respBean.getEMoney();
    System.out.println("--------␣EMoney␣from␣the␣DB:␣" + eMoney);
    return eMoney;

  } else{
    System.out.println("--------␣User␣not␣authenticated␣agasint␣the␣DB
        :␣failure:failure␣401");
    return String.valueOf(SQL_Return_Codes.
        Authentication_E_MONEY_PRT_USER_NOT_AUTHENTICATED);
  }

} // end returnEmoney()
```

```
  /** Returns a short description of the servlet.
   *
   * @return Returns a String description of the servlet
   */
  public String getServletInfo() {
    return "This␣is␣a␣servlet␣that␣performs␣encrypted␣communication\n" +
    "␣with␣a␣midlet.␣It␣decrypts/encrypts␣the␣requests/answers.\n";
  } // end getServletInfo()

} // end class

package cinemaservice.servlets.workers;

import java.io.*;
import java.util.Date;
import java.util.Random;

import javax.servlet.*;
import javax.servlet.http.*;

import org.bouncycastle.crypto.CryptoException;
import org.bouncycastle.crypto.params.ParametersWithIV;

import cinemaservice.model.facade.HelpingCrypto;

/**
 * This servlet performs encrypted communication with a MIDlet.
 * The data is transfered between them as binary. It decrypts the MIDlet'
     s request
 * and sends back an encrypted answer to the MIDlet.
 *
 * The communication with this servlet is working on the basis of 2
     protocol states:
 *  - PRT1: The servlet is challenged with the message relayed from the
     authentication
 *      server. It extracts salt value and a a token from the message.
     TOken is
 *      check against the database and in case of success, random number
     sent back
 *      previously encrypted with a session key.
 *  - PRT2: The servlet checks the random number it previously generated
     . In case it is
 *      one smaller than the one previously produced, then the
     authentication is finished.
 * @author Mihai Balan - s031288
 * @version 3.0
 */
```

```
public class Authentication_Servlet_2 extends HttpServlet {

  private static final long serialVersionUID = 3L;

  // The key object for the communication with the 128 AES cipher
  ParametersWithIV aesKey = null;

  /**
   * Initialize the Authentication_Servlet_2
   *
   * @throws ServletException in case initialization fails
   */
  public void init(ServletConfig config) throws ServletException {}


  /** Processes requests for the HTTP GET method.
   *  Print a message followed by the curret date and time.
   *
   * @param request Servlet request
   * @param response Servlet response
   *
   * @throws IOException, ServletException
   */
  public void doGet(HttpServletRequest request,HttpServletResponse
      response)
  throws IOException, ServletException {
    //doPost(request, response);
    response.setContentType("text/plain");
    PrintWriter out = response.getWriter();
    out.println("Secure␣Servlet␣invoked!");
    out.println(new Date());

  } // end doGet()

  /** Processes requests for the HTTP POST method.
   *  It gets the ContentLength of the request and
   *  checks if it is valid or not. It writes
   *  the MIDlet'sencrypted request to a byte array.
   *  Then it decrypts and processes the decrypted request
   *  and sends an encrypted answer back to the servlet.
   *
   * @param request Servlet request
   * @param response Servlet response
   * @throws IOException, ServletException
   */
  public void doPost(HttpServletRequest request, HttpServletResponse
      response)
```

```
throws IOException, ServletException {

  // for encryption - decryption opperations
  HelpingCrypto helpCrypto = new HelpingCrypto();

  // the byte array that holds the request
  byte[] reqData = null;

  // get the length of the request
  int len = request.getContentLength();

  // open an input stram to process the request
  ServletInputStream sis = request.getInputStream();
  if (len != -1){
    // if length is available read the data into an array
    reqData = new byte[len];
    sis.read(reqData);
  }
  else{
    // If length is not available read data from the input stream
    // one character at a time and store the answer in a byte array.
    ByteArrayOutputStream tmp = new ByteArrayOutputStream();
    int ch;
    // read char by char from the ServletInputStream until EOF
    while( ( ch = sis.read() )!= -1 ){
      // and write to the ByteArrayOutputStream
      tmp.write( ch );
    }
    // move the data to the reqData
    reqData = tmp.toByteArray();
  }

  //every request is checked against the protocol
  //appropriate answer is returned to the user
  byte[] resp = handleTheProtocol(helpCrypto, reqData);

  //The status of the response should be set for propper
  //functioning of the underlying http communication
  if ((resp == null) || (resp.length == 0)){
    response.setStatus( HttpServletResponse.SC_INTERNAL_SERVER_ERROR );
    resp = ("Server␣Error").getBytes();
  }
  else {
    response.setStatus( HttpServletResponse.SC_OK );
  }

  // set the content type
```

```
  response.setContentType("application/octet-stream");

  // set the Content length property
  response.setContentLength(resp.length);

  // get a ServletOutputStream to send the response to the MIDlet as
      binary data
  ServletOutputStream sos = response.getOutputStream();
  sos.write(resp);
  sos.flush();

} // end doPost()


/**
 * In this method, the protocol is handled according to the protocol
 * description specified in the description header of this class.
 *
 * @param reqData The message sent by the user
 * @return The response that should be sent to the user
 */
public byte[] handleTheProtocol(HelpingCrypto helpCrypto, byte[]
    reqData){

  //The random number the checking of the authentication correctness
  //and preventing from the replay attacks
  String randomNumber = "";

  //global variable, to which the response message would be joined
  byte[] res = null;

  //the key used for decrypting/encrypting messages
  String key = "87654321";

  //The request must be transformed into string
  //The protocol indicator is not encrypted in this implementation
  //Therefore it is just chopped of the request
  // ... like that
  String request = new String(reqData);
  String protItem = request.substring(0,4);

  //The length of the string should be the same
  //as the length of an array containing the encrypted data
  request = request.substring(5);

  /**
   * At this point the first step of the protocol is handled
```

```
 * i.e. Decrypt the message from the mobile device and extract
 * the salt value and the token. It checks the credentials received
 * from the mobile device with Authentication_Servlet_1.
 *
 * If the credentials are correct it generates a new session key
 * with the salt value and the userID from the received token.
 * It generates another random no, and encryptes this no. with the
 * previously generated session key. Then, the response is sent to
 * the mobile device
 */
if (protItem.equals("PRT1")){
  //First we decrypt the message containing the session key
  try {
    //byte array containing the message sent by the user is
    //coppied direcly from therequest data structure, in
    //order not to corrupt the padding
    byte[] payload = new byte[request.length()];
    System.arraycopy(reqData,5,payload,0,request.length());

    //properly extracted message part may be decrypted here
    String decReq = helpCrypto.decryptMessage(key, payload);

    //Message contains two elements, they are delimited
    String[] decReqAr = decReq.split(";");

    //User credentials are checked against the database
    String tkn = System.getProperty("token");

    //Debug
    //System.out.println("Token ... " + tkn);
    if(tkn.equals(decReqAr[1])){

      //it is necessary also to grab the serial number of the user
      //for the purpose of creating the session key
      String[] uid = decReqAr[1].split(":");

      //The key is phisically created
      aesKey = helpCrypto.createKey(decReqAr[0], uid[0]);
      System.out.println("--------------------aes␣key:" + aesKey.
          toString());
      // share the aesKEY parts in order to generate the key for PRT3
          when needed
      System.out.println("-------------The␣AESKEY␣is␣made␣of:␣" +
          decReqAr[0] + "+++++" + uid[0]);
      System.setProperty("aesSalt",decReqAr[0]); // salt value (
          generated session key)
      System.setProperty("aesToken",uid[0]); // token (userID)
```

```
      //Debug
      //System.out.println("AES key has been created ..." + aesKey.
          toString());

      //According to the protocol we generate a random long
      randomNumber = generateRandom();

      //We encrypt the random number with our session key
      res = helpCrypto.encryptWithAES(aesKey, randomNumber.getBytes())
          ;
      System.out.println("The␣random␣number␣encrypted␣with␣AES␣is:␣"
          + new String(res));
      System.out.println("test");
    } else{
      //In case of the inappropriate authentication token
      System.out.println("There␣is␣no␣valid␣token");
      res = null;
    }
  } catch (CryptoException e) {
    e.printStackTrace();
  }

} // end if (protItem.equals("PRT1"))

/**
 * At this point the second step of the protocol is handled i.e.
 * Cheks if the random no . sent from the mobile device is the same
 * with the one on the server side
 */
if (protItem.equals("PRT2")){
  try {
    //Similarly as in PRT1, the relevant message part is extracted
    byte[] payload = new byte[request.length()];
    System.arraycopy(reqData,5,payload,0,request.length());

    //decrypting
    byte[] decReq = helpCrypto.decryptWithAES(aesKey, payload);
    String decReq1 = new String(decReq);

    //Debug
    System.out.println("The␣random␣number␣decrypted:␣" + decReq1);
    String rnum = randomNumber;

    //Debug
    System.out.println("Our␣random␣number,␣modified:␣" + rnum);
```

```
        //Obtained random number is schecked
        if (!decReq1.equals(rnum)){
          res = ("Authenticated").getBytes();
        }
        else {
          res = ("Incorrect␣Random␣Number").getBytes();
        }
      } catch (Exception e) {
        e.printStackTrace();
      }

    }// end if (protItem.equals("PRT2"))

    return res;

  } // end handleTheProtocol()

  /**
   * In tis method, long random number is generated and returned as a
       string.
   * The sthing may be directlu encrypted and sent to the user.
   *
   * @return Random number used for the verification of the protocol
   */
  public String generateRandom(){
    Random rand = new Random();
    long myRandom = rand.nextInt();
    String theRandom = "" + myRandom;
    return theRandom;

  }// end generateRandom()


  /** Returns a short description of the servlet.
   *
   * @return Returns a String description of the servlet
   */
  public String getServletInfo() {
    return "This␣is␣a␣servlet␣that␣performs␣encrypted␣communication\n" +
    "␣with␣a␣midlet.␣It␣decrypts/encrypts␣the␣requests/answers.\n";
  }

}

package cinemaservice.servlets.workers;


import cinemaservice.model.beans.requestBeans.Cinema_Hall_Conf_Req_Bean;
```

```
import cinemaservice.model.beans.responseBeans.Response_Msg_Bean;
import cinemaservice.model.facade.FacadeToModel;

import java.util.Date;
import java.util.Vector;

import java.io.IOException;

import javax.servlet.*;
import javax.servlet.http.*;

import org.apache.log4j.*;


/**
 * Background_Hall_Update_Servlet implments the Generic_Worker_Servlet.
 *
 * The TEMPLATE METHODS in the Generic_Worker_Servlet are implemented
 * as HOOK METHODS in here.
 *
 * It retrieves and casts the Cinema_Hall_Conf_Req_Bean from the client.
 * Constructs the parameter list to execute the
     Background_Cinema_Hall_Update stored procedure.
 * The parameter list contains: showLocationID, showTimeID.
 *
 * Set the SQL Statemet to be executed to the SQL_Operations_Bean against
 * the pgSQL DB.
 *
 * Execute the SQL statement, parse the result of the SQL, and send
 * a Cinema_Hall_Conf_Res_Bean Object back to the client.
 *
 * The implementation is using the TEMPLATE METHOD, REFACTORING, FACADE
 * and GENERALIZING Design Patterns.
 *
 * @author Mihai Balan - s031288
 */
public class Background_Hall_Update_Servlet extends
    Generic_Worker_Servlet implements Servlet {

  // ======================================================================

  //                           DECLARATIONS
  // ======================================================================

  private static final long serialVersionUID = 3L;

  /**
```

```
 * Initiate a custom logging category
 */
private static Category cat = Category.getInstance(
    Background_Hall_Update_Servlet.class.getName());

// ============================================================================

//                              METHODS
// ============================================================================

/**
 * Initialize the Background_Hall_Update_Servlet and calls
 * init() in the Generic_Worker_Servlet class.
 *
 * If the initialization fails, the servlet is not started
 *
 * @throws ServletException in case initialization fails
 */
public void init(ServletConfig config) throws ServletException {
  // Store the ServletConfig object and log the initialization
  // Performs also the connection pooling initialization
  super.init(config);

} // end init()


/**
 * Deal with GET requests from the client side and calls
 * doGet() in the Generic_Worker_Servlet class.
 *
 * @param request The HTTP request from the client to the servlet
 * @param response The HTTP response from the servlet to the client
 */
protected void doGet(HttpServletRequest request, HttpServletResponse
    response)
throws ServletException, IOException {

  super.doGet(request, response);

} // end doGet()


/**
 * Deal with POST requests from the client side and calls
 * doPost() in the Generic_Worker_Servlet class.
 *
 * @param request The HTTP request from the client to the servlet
```

```
 * @param response The HTTP response from the servlet to the client
 */
protected void doPost(HttpServletRequest request, HttpServletResponse
    response)
throws ServletException, IOException {

  super.doPost(request, response);

} // end do Post()


/**
 * Returns information about the Background_Hall_Update_Servlet
 *
 * @return Information about the Background_Hall_Update_Servlet
 */
public String getServletInfo() {
  return "Hello␣from␣the␣Background_Hall_Update_Servlet␣on␣" + new Date
      ();

} // end getServletInfo()


// ============================================================================

//                            HOOK METHODS
// ============================================================================


/**
 * HOOK METHOD for casting the Request Bean to the corresponding type
 * e.g. Cinema_Hall_Conf_Req_Bean.
 *
 * @param requestBean The Cinema_Hall_Conf_Req_Bean sent by the MIDlet
     to the server side
 *                  containing the client request data as an object
     with
 *                  get and set methods
 * @return The Cinema_Hall_Conf_Req_Bean
 */
protected Cinema_Hall_Conf_Req_Bean getRequestBean(Object requestBean){

  cat.debug("Before␣the␣Cinema_Hall_Conf_Req_Bean");
  Cinema_Hall_Conf_Req_Bean cinHallConfBean = (
      Cinema_Hall_Conf_Req_Bean)requestBean;
  cat.debug("After␣the␣Cinema_Hall_Conf_Req_Bean");
  return cinHallConfBean;
```

```
  } // end getRequestBean()



  /**
   * HOOK METHOD for setting the SQL statement that is to be executed
   * i.e. to get the cinema hall configuration for the given show.
   *
   * @return The SQL statement to be executed
   */
  protected String setSQLStatement() {
    return "SELECT␣*␣FROM␣cinema.Background_Cinema_Hall_Update(?,␣?);";

  } // end setSQLStatement()


  /**
   * HOOK METHOD for parsing the result of the SQL statement
   * (the response from retrieving the cinema hall configuration)
   * and defining the response to be sent to the client
   * (the response code to the user).
   *
   * It delegates the control to the ResponseDataModel
   * responsable for parsing the SQLResult and creating
   * the Background_Cinema_Hall_Conf_Resp_Bean.
   *
   * @param sqlResult The result of the SQL statement execution
   * @return        The response to be sent back to the client.
   */
  protected Response_Msg_Bean parseSQLResponse(Vector sqlResult) {

    FacadeToModel facade = new FacadeToModel();
    return facade.setResponseBackgroundCinemaHallConfBean(sqlResult);

  } // end parseSQLResponse()

} // end class

package cinemaservice.servlets.workers;


import cinemaservice.model.beans.requestBeans.Cancel_Tickets_Req_Bean;
import cinemaservice.model.beans.responseBeans.Response_Msg_Bean;
import cinemaservice.model.facade.FacadeToModel;

import java.util.Date;
import java.util.Vector;
```

```
import java.io.IOException;

import javax.servlet.*;
import javax.servlet.http.*;

import org.apache.log4j.*;


/**
 * Cancel_Tickets_Servlet implements the Generic_Worker_Servlet.
 *
 * The TEMPLATE METHODS in the Generic_Worker_Servlet are implemented
 * as HOOK METHODS in here.
 *
 * Construct the parameter list to execute the Cancel_Tickets stored
     procedure.
 *
 * The input parameter list contains:
 *   - UserName
 *   - Password
 *   - ReservationID
 *   - TicketIDs[]
 *
 * The output parameter list contains:
 *   - StatusCode
 *
 * Set the SQL Statemet to be executed to the Cancel_Tickets stored
     procedure.
 *
 * Execute the SQL statement, then parse the result of the SQL and send
 * a response back to the client.
 *
 * The implementation is using the TEMPLATE METHOD, REFACTORING, FACADE,
 * and GENERALIZING Design Patterns.
 *
 * @author Mihai Balan - s031288
 */
public class Cancel_Tickets_Servlet extends Generic_Worker_Servlet
    implements Servlet {

  // ========================================================================

  //                          DECLARATIONS
  // ========================================================================

  private static final long serialVersionUID = 3L;
```

```
/**
 * Initiate a custom logging category
 */
private static Category cat = Category.getInstance(
    Cancel_Tickets_Servlet.class.getName());



// ============================================================================

//                              METHODS
// ============================================================================

/**
 * Initialize the Cancel_Tickets_Servlet and calls
 * init() in the Generic_Worker_Servlet class.
 *
 * If the initialization fails, the servlet is not started
 *
 * @throws ServletException in case initialization fails
 */
public void init(ServletConfig config) throws ServletException {
  // Store the ServletConfig object and log the initialization
  // Performs also the connection pooling initialization
  super.init(config);

} // end init()


/**
 * Deal with GET requests from the client side and calls
 * doGet() in the Generic_Worker_Servlet class.
 *
 * @param request The HTTP request from the client to the servlet
 * @param response The HTTP response from the servlet to the client
 */
protected void doGet(HttpServletRequest request, HttpServletResponse
    response)
  throws ServletException, IOException {
    super.doGet(request, response);

} // end doGet()


/**
 * Deal with POST requests from the client side and calls
```

```
 * doPost() in the Generic_Worker_Servlet class.
 *
 * @param request The HTTP request from the client to the servlet
 * @param response The HTTP response from the servlet to the client
 */
protected void doPost(HttpServletRequest request, HttpServletResponse
    response)
  throws ServletException, IOException {
    super.doPost(request, response);

} // end do Post()


/**
 * Returns information about the Cancel_Tickets_Servlet
 *
 * @return Information about the Cancel_Tickets_Servlet
 */
public String getServletInfo() {
  return "Hello␣from␣the␣Cancel_Tickets_Servlet␣on␣" + new Date();

} // end getServletInfo()


// ============================================================================

//                            HOOK METHODS
// ============================================================================


/**
 * HOOK METHOD for casting the Request Bean to the corresponding type
 * i.e. Cancel_Tickets_Req_Bean.
 *
 * @param requestBean The Cancel_Tickets_Req_Bean sent by the MIDlet to
     the
 *                  server side containing the client request data as
 *                  an object with get and set methods
 * @return The Cancel_Tickets_Req_Bean
 */
protected Cancel_Tickets_Req_Bean getRequestBean(Object requestBean){

  cat.debug("Before␣the␣req␣bean");
  Cancel_Tickets_Req_Bean cancelTicketsReqBean = (
      Cancel_Tickets_Req_Bean)requestBean;
  cat.debug("After␣the␣req␣bean");
  return cancelTicketsReqBean;
```

```
  } // end getRequestBean()



  /**
   * HOOK METHOD for setting the SQL statement that is to be executed
   * i.e. to cancel the payed tickets using the "CARD" payment method
   *
   * @return The SQL statement to be executed
   */
  protected String setSQLStatement() {
    return "SELECT␣*␣FROM␣cinema.Cancel_Tickets(?,␣?,␣?,␣?)";

  } // end setSQLStatement()



  /**
   * HOOK METHOD for parsing the result of the SQL statement
   * (the result for cancelling the tickets i.e. response code, etc)
   * and constructing the Response_Msg_Bean to be sent
   * to the client.
   *
   * It delegates the control to the ResponseDataModel
   * responsable for parsing the SQLResult and creating
   * the Response_Msg_Bean.
   *
   *
   * @param sqlResult The result of the SQL statement execution
   * @return        The Response_Msg_Bean to be sent back to the client.
   */
  protected Response_Msg_Bean parseSQLResponse(Vector sqlResult) {
    FacadeToModel facade = new FacadeToModel();
    return facade.setResponseCancelTicketsBean(sqlResult);

  } // end parseResponse()

} // end class

package cinemaservice.servlets.workers;


import cinemaservice.model.beans.requestBeans.Change_Password_Req_Bean;
import cinemaservice.model.beans.responseBeans.Response_Msg_Bean;
import cinemaservice.model.facade.FacadeToModel;

import java.util.Date;
```

```
import java.util.Vector;

import java.io.IOException;

import javax.servlet.*;
import javax.servlet.http.*;

import org.apache.log4j.*;


/**
 * Change_Password_Servlet implments the Generic_Worker_Servlet.
 *
 * The input parameter list contains:
 *   - UserName
 *   - Old Password
 *   - New Password
 *
 * The TEMPLATE METHODS in the Generic_Worker_Servlet are implemented
 * as HOOK METHODS in here.
 *
 * It retrieves and casts the Change_Password_Req_Bean from the client.
 * Constructs the parameter list to execute the Change_Password stored
     procedure.
 * The parameter list contains: user_name, old_password, new_password.
 *
 * Set the SQL Statemet to be executed to the SQL_Operations_Bean against
 * the pgSQL DB.
 *
 * Execute the SQL statement, parse the result of the SQL, and send
 * a Response Bean Object back to the client.
 *
 * The implementation is using the TEMPLATE METHOD, REFACTORING, FACADE
 * and GENERALIZING Design Patterns.
 *
 * @author Mihai Balan - s031288
 */
public class Change_Password_Servlet extends Generic_Worker_Servlet
    implements Servlet {

  // ============================================================================

  //                           DECLARATIONS
  // ============================================================================

  private static final long serialVersionUID = 3L;
```

```
/**
 * Initiate a custom logging category
 */
private static Category cat = Category.getInstance(
    Change_Password_Servlet.class.getName());

// ============================================================================

//                              METHODS
// ============================================================================

/**
 * Initialize the Change_Password_Servlet and calls
 * init() in the Generic_Worker_Servlet class.
 *
 * If the initialization fails, the servlet is not started
 *
 * @throws ServletException in case initialization fails
 */
public void init(ServletConfig config) throws ServletException {
  // Store the ServletConfig object and log the initialization
  // Performs also the connection pooling initialization
  super.init(config);

} // end init()


/**
 * Deal with GET requests from the client side and calls
 * doGet() in the Generic_Worker_Servlet class.
 *
 * @param request The HTTP request from the client to the servlet
 * @param response The HTTP response from the servlet to the client
 */
protected void doGet(HttpServletRequest request, HttpServletResponse
    response)
throws ServletException, IOException {

  super.doGet(request, response);

} // end doGet()


/**
 * Deal with POST requests from the client side and calls
 * doPost() in the Generic_Worker_Servlet class.
 *
```

```
 * @param request The HTTP request from the client to the servlet
 * @param response The HTTP response from the servlet to the client
 */
protected void doPost(HttpServletRequest request, HttpServletResponse
    response)
throws ServletException, IOException {

  super.doPost(request, response);

} // end do Post()


/**
 * Returns information about the Change_Password_Servlet
 *
 * @return Information about the Change_Password_Servlet
 */
public String getServletInfo() {
  return "Hello␣from␣the␣Change_Password_Servlet␣on␣" + new Date();

} // end getServletInfo()


// ============================================================================

//                            HOOK METHODS
// ============================================================================


/**
 * HOOK METHOD for casting the Request Bean to the corresponding type
 * e.g. Movie_Details_Resp_Bean.
 *
 * @param requestBean The Change_Password_Req_Bean sent by the MIDlet
     to the server side
 *                  containing the client request data as an object
     with
 *                  get and set methods
 * @return The Change_Password_Req_Bean
 */
protected Change_Password_Req_Bean getRequestBean(Object requestBean){

  cat.debug("Before␣the␣req␣bean");
  Change_Password_Req_Bean chgPswdBean = (Change_Password_Req_Bean)
      requestBean;
  cat.debug("After␣the␣req␣bean");
  return chgPswdBean;
```

```
  } // end getRequestBean()



  /**
   * HOOK METHOD for setting the SQL statement that is to be executed
   * i.e. to get details about the movie that is played in a given show.
   *
   * @return The SQL statement to be executed
   */
  protected String setSQLStatement() {
    return "SELECT␣*␣FROM␣cinema.Change_Password(?,␣?,␣?)";

  } // end setSQLStatement()


  /**
   * HOOK METHOD for parsing the result of the SQL statement
   * (the response from changing user's password)
   * and defining the response to be sent to the client
   * (the response code to the user).
   *
   * It delegates the control to the ResponseDataModel
   * responsable for parsing the SQLResult and creating
   * the Response_Msg_Bean.
   *
   * @param sqlResult The result of the SQL statement execution
   * @return        The response to be sent back to the client.
   */
  protected Response_Msg_Bean parseSQLResponse(Vector sqlResult) {

    FacadeToModel facade = new FacadeToModel();
    return facade.setResponseChangePasswordBean(sqlResult);

  } // end parseSQLResponse()

} // end class

package cinemaservice.servlets.workers;


import cinemaservice.model.beans.requestBeans.Find_Movies_Req_Bean;
import cinemaservice.model.beans.responseBeans.Response_Msg_Bean;
import cinemaservice.model.facade.FacadeToModel;

import java.util.Date;
import java.util.Vector;
```

```
import java.io.IOException;

import javax.servlet.*;
import javax.servlet.http.*;

import org.apache.log4j.*;


/**
 * Find_Movies_Servlet implements the Generic_Worker_Servlet.
 *
 * The TEMPLATE METHODS in the Generic_Worker_Servlet are implemented
 * as HOOK METHODS in here.
 *
 * Construct the parameter list to execute the Find_Movies stored
     procedures.
 * It retrieves the searching criteria send by the user and makes a
     request to the
 * Movie Location Service to find all cinema in the given range from the
     given user
 * location. The cinemas returned by MLS are then further used in the SQL
      queries
 * by the Cinema Controller to get all movies that match the given
     searching criteria
 *
 * The input parameter list contains:
 *   - Movie
 *   - User given location i.e. street, city, zip
 *   - Range: Finding movies in the given range from the user given
     position
 *   - Show Date
 *
 * The output parameter list contains:
 *   - List of all found movies containing:
 *        Movie, showHour, showLocationID, showTimeID, Cinema, City,
     Street
 *
 * Set the SQL Statemet to be executed to the Find_Movies stored
     procedures.
 *
 * Execute the SQL statement, then parse the result of the SQL and send
 * a response back to the client.
 *
 * The implementation is using the TEMPLATE METHOD, REFACTORING, FACADE,
 * and GENERALIZING Design Patterns.
 *
```

```
 * @author Mihai Balan - s031288
 */
public class Find_Movies_Servlet extends Generic_Worker_Servlet
    implements Servlet {

  // ============================================================================

  //                            DECLARATIONS
  // ============================================================================

  private static final long serialVersionUID = 3L;

  /**
   * Initiate a custom logging category
   */
  private static Category cat = Category.getInstance(Find_Movies_Servlet.
      class.getName());



  // ============================================================================

  //                              METHODS
  // ============================================================================

  /**
   * Initialize the Find_Movies_Servlet and calls
   * init() in the Generic_Worker_Servlet class.
   *
   * If the initialization fails, the servlet is not started
   *
   * @throws ServletException in case initialization fails
   */
  public void init(ServletConfig config) throws ServletException {
    // Store the ServletConfig object and log the initialization
    // Performs also the connection pooling initialization
    super.init(config);

  } // end init()


  /**
   * Deal with GET requests from the client side and calls
   * doGet() in the Generic_Worker_Servlet class.
   *
   * @param request The HTTP request from the client to the servlet
   * @param response The HTTP response from the servlet to the client
```

```
 */
protected void doGet(HttpServletRequest request, HttpServletResponse
    response)
  throws ServletException, IOException {
    super.doGet(request, response);

} // end doGet()


/**
 * Deal with POST requests from the client side and calls
 * doPost() in the Generic_Worker_Servlet class.
 *
 * @param request The HTTP request from the client to the servlet
 * @param response The HTTP response from the servlet to the client
 */
protected void doPost(HttpServletRequest request, HttpServletResponse
    response)
  throws ServletException, IOException {
    super.doPost(request, response);

} // end do Post()


/**
 * Returns information about the Find_Movies_Servlet
 *
 * @return Information about the Find_Movies_Servlet
 */
public String getServletInfo() {
  return "Hello␣from␣the␣Find_Movies_Servlet␣on␣" + new Date();

} // end getServletInfo()


// ============================================================================

//                             HOOK METHODS
// ============================================================================


/**
 * HOOK METHOD for casting the Request Bean to the corresponding type
 * i.e. Find_Movies_Req_Bean.
 *
 * @param requestBean The Find_Movies_Req_Bean sent by the MIDlet to
     the
```

```
 *                  server side containing the client request data as
 *                  an object with get and set methods
 * @return The Find_Movies_Req_Bean
 */
protected Find_Movies_Req_Bean getRequestBean(Object requestBean){

  cat.debug("Before␣the␣req␣bean");
  Find_Movies_Req_Bean findMoviesReqBean = (Find_Movies_Req_Bean)
      requestBean;
  cat.debug("After␣the␣req␣bean");
  return findMoviesReqBean;

} // end getRequestBean()



/**
 * HOOK METHOD for setting the SQL statement that is to be executed
 * i.e. find all movies matching the given criteria
 *
 * @return The SQL statement to be executed
 */
protected String setSQLStatement() {

  return "SELECT␣*␣FROM␣cinema.Find_Movies_Criteria(?,␣?,␣?,␣?)";

} // end setSQLStatement()



/**
 * HOOK METHOD for parsing the result of the SQL statement
 * (the movie list containing all movies that matched
 * user's searching criteria) and constructing
 * the Find_Movies_Resp_Bean to be sent to the client.
 *
 * It delegates the control to the ResponseDataModel
 * responsable for parsing the SQLResult and creating
 * the Find_Movies_Resp_Bean.
 *
 *
 * @param sqlResult The result of the SQL statement execution
 * @return        The Find_Movies_Resp_Bean to be sent back to the
     client.
 */
protected Response_Msg_Bean parseSQLResponse(Vector sqlResult) {
```

```
    FacadeToModel facade = new FacadeToModel();
    return facade.setResponseFindMoviesBean(sqlResult);

  } // end parseResponse()

} // end class

package cinemaservice.servlets.workers;

import cinemaservice.beans.tools.SQL_Operations_Bean;
import cinemaservice.beans.tools.Servlet_Operations_Bean;

import cinemaservice.constants.Error_Code_Constants;
import cinemaservice.constants.Protocol_Step_Constants;
import cinemaservice.exceptions.CinemaServiceException;
import cinemaservice.model.beans.requestBeans.Purchase_Tickets_Req_Bean;

import java.io.IOException;

import java.text.ParseException;
import java.util.Date;
import java.util.Vector;

import java.sql.SQLException;
import javax.sql.DataSource;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import javax.servlet.*;
import javax.servlet.http.*;

import org.apache.log4j.*;


/**
 * Defines the generic worker servlet for all worker servlets called from
      the
 * Cinema_Central_Control_Center. All servlets that perform the given
     protocol step
 * extend this class and implement the template methods provided in here
     i.e.
 * parseRequestAndGetSQLParamList(...), setSQLStatement(...), and
     parseResponse(...)
 * as hook methods in the concrete classes.
 *
 * Initialize the Connection Pool sing JNDI, adn process the requst from
```

```
     the client
 * by extracting the request content, the parameter list needed to
     execute the SQL query,
 * setting the SQL statement to be eecuted, executing adn retrieving the
     result
 * of the SQL statement, and sending the resilt to the client.
 *
 * Servlet_Operations_Bean Java Bean is used for general Servet
     Operations i.e.
 * reading the request content, and
 * sending the response to the client.
 *
 * SQL_Operations_Bean Java Bean is used for SQL purposes e.g. getting
     the
 * connection from the pool, setting the SQl statement,
 * parameter list and values, and executing the SQL statement.
 *
 *
 * The implementation is using the TEMPLATE METHOD, SINGLETON,
     REFACTORING,
 * FACADE, GENERALIZING Design Patterns.
 *
 * @author Mihai Balan - s031288
 */
public abstract class Generic_Worker_Servlet extends HttpServlet
    implements Servlet {


  // ============================================================================

  //                           DECLARATIONS
  // ============================================================================




  private static final long serialVersionUID = 2L;

  /**
   * Initiate a custom logging category
   */
  private static Category cat = Category.getInstance(
      Generic_Worker_Servlet.class.getName());

  /**
   * The datasource object for obtainig the connection from the pool
   */
```

```
private DataSource datasource = null;



// ============================================================================

//                              METHODS
// ============================================================================


/**
 * Initialize the Generic_Worker_Servlet by initializing the
 * pool of PostgreSQL connections usiong JNDI.
 * The configuration files for the conection pool are:
 * META-INF/context.xml and WEB_INF/web.xml
 * If the initialization fails, the servlet is not started
 *
 * @throws ServletException in case initialization fails
 */
public void init(ServletConfig config) throws ServletException {

   try {
    // Create a datasource for pooled connections.
    // Use JNDI to retrieve the DataSource object defined
    // in the Tomcat and application configuration *.xml files
    // i.e.WEB_INF/web.xml and META_INF/context.xml in the directory
    // where the application is deployed on Tomcat
    Context initCtx = new InitialContext();
    Context envCtx = (Context) initCtx.lookup("java:comp/env");
    datasource = (DataSource) envCtx.lookup("jdbc/postgres");
  } catch (NamingException ne) {
    cat.fatal("----␣Generic_Worker_Servlet␣-␣JNDI␣Naming␣Exception␣when
        ␣trying␣to␣acquire␣the␣DataSource␣object\n" +
       ne.getMessage() + "\n" + ne.getStackTrace());
  }

} // end init()


/**
 * Deal with GET requests from the client side.
 *
 * Any GET request to the target servlet e.g. Movie_Details_Servlet
 * end up in here.
 *
 * @param request The HTTP request from the client to the servlet
 * @param response The HTTP response from the servlet to the client
```

```
 * @throws CinemaServiceException
 */
protected void doGet(HttpServletRequest request, HttpServletResponse
    response)
throws ServletException, IOException{

  //doPost(request, response);

} // end doGet()


/**
 * Deal with POST requests from the client side.
 *
 * Any POST request to the target servlet e.g. Movie_Details_Servlet
 * end up in here.
 *
 * If a POST request is received, call the processRequest()
 * to deal with the request.
 *
 * @param request The HTTP request from the client to the servlet
 * @param response The HTTP response from the servlet to the client
 */
protected void doPost(HttpServletRequest request, HttpServletResponse
    response)
throws ServletException, IOException{

  try{
    processRequest(request, response);

  } catch (CinemaServiceException cse){
    cat.warn("CinemaServiceException:␣" + cse.getMessage());
    response.setStatus(cse.getValue());

  } catch (SQLException sqle) {
    cat.warn("SQLException:␣"+ sqle.getSQLState()
        + "␣-␣" + sqle.getErrorCode() + "␣-␣" + sqle.toString());
    response.setStatus(Error_Code_Constants.ERROR_IN_SQL);

  } catch (ParseException pe) {
    cat.warn("ParseException:␣"+ pe.getMessage());
    response.setStatus(Error_Code_Constants.ERROR_IN_SQL);

  } catch (IOException ioe){
    cat.warn("IOException:␣"
        + ioe.toString() + ":␣" + ioe.getMessage());
    response.setStatus(Error_Code_Constants.INTERNAL_SERVER_ERROR);
```

```
    throw ioe;
  } // end try-catch

  /* }catch (ClassNotFoundException cne){
    cat.warn("ClassNotFoundException: " + cne.getMessage());
   */

} // end doPost()


/**
 * Retrieve the request java bean from the MIDlet,
 * extracts the parameter list and values needed in the SQL statement,
 * set the SQL statement that is to be run, execute and
 * get the result from the SQL statement, and send a response back to
     the MIDlet
 *
 * @param request The HTTP request from the client to the servlet
 * @param response The HTTP response from the servlet to the client
 * @throws ParseException
 */
private void processRequest(HttpServletRequest request,
    HttpServletResponse response)
throws IOException, SQLException, CinemaServiceException,
    ParseException{


  /**
   * Java Bean for for general Servet Operations i.e.
   * reading the request content, and sending the response to the
       client.
   */
  Servlet_Operations_Bean servOpBean = new Servlet_Operations_Bean();

  /**
   * Request Java Bean that contains the data sent from the client side
        to
   * the server side
   */
  Object requestBean = null;

  /**
   * Response Java Bean that contains the response to be sent
   * from the server side to the MIDlet
   */
  Object responseBean = null;
```

```
    /**
     * Java Bean for SQL purposes e.g. getting the connection from the
         pool,
     * setting the SQl statement,parameter list and values,
     * and executing the SQL statement
     */
    SQL_Operations_Bean sqlOpBean = new SQL_Operations_Bean();

    // the result of executing the SQL query
    Vector sqlResult = new Vector();

    // the error code value from the result of the SQL execution
    String sqlErrorCode = "";

    // using the Servlet_Operations_Bean
    servOpBean.setHttpRequest(request);

    // get the Request Bean sent by the client
    requestBean = getRequestBean(servOpBean.getClientRequestDataObject())
        ;

    // extract the parameters need to execute the SQL query from
    // the Request Bean sent by the MIDlet
    //sqlParameters = deserializeRequestBeanAndGetSQLParamList(
        requestBean);
    //System.out.println("------ SQL Parameter: " + sqlParameters.
        elementAt(0).toString());
    String prtStep = request.getParameter("protocol");

    if (prtStep.equals(Protocol_Step_Constants.PRT_STEP_PURCHASE_TICKETS)
        ){
      sqlResult.add(0, null);
      sqlResult.add(1, (Purchase_Tickets_Req_Bean)requestBean);
    }

    // set the SQL statement that is to be executed
    String sqlStmt = setSQLStatement();

    // get the dataresource used to obtained the pooled conn
    // set the sql statement to be executed
    // set the parameter list for the sql statement
    // and execute the sql statement
//    ------
    sqlOpBean.setPooledSource(datasource);
//    ------
    sqlOpBean.setSQLStatement(sqlStmt);
    System.out.println("------␣SQL␣Statement␣set:\n");
```

```
//   ------
    sqlOpBean.setSQLParameters(requestBean, prtStep);
    System.out.println("------␣SQL␣Parameters␣Set:\n");
//   ------

    if (prtStep.equals(Protocol_Step_Constants.PRT_STEP_PURCHASE_TICKETS)
        ){
      sqlResult.set(0,sqlOpBean.executeSQL().elementAt(0));
    } else {
      sqlResult = sqlOpBean.executeSQL();
    }
    System.out.println("------␣SQL␣Result:\n" + sqlResult);


    // get the Error Code from the SQL execution i.e. the int value
    // located on the first position of the response message
    sqlErrorCode = sqlResult.elementAt(0).toString().substring(0,3);
    cat.debug("Error␣code:␣" + sqlErrorCode);
    //setResponseHTTPStatusCode(sqlErrorCode,servOpBean);
    servOpBean.setHttpStatusCode(Error_Code_Constants.OK);

    cat.debug("Before␣creating␣response␣Bean");
    // create the Response Bean to be sent to the MIDlet
    responseBean = parseSQLResponse(sqlResult);
    cat.debug("After␣creating␣response␣Bean");

    // send the Response Bean to the the MIDlet
    servOpBean.setHttpResponse(response);
    cat.debug("Before␣sending␣the␣Rating␣Response!");
    servOpBean.sendResponseBeanToMidlet(responseBean);
    cat.debug("After␣sending␣the␣Rating␣Response!");

  } // end processRequest()



  /**
   * Returns information about the Generic_Worker_Servlet
   *
   * @return Information about the Generic_Worker_Servlet
   */
  public String getServletInfo() {
    return "Hello␣from␣the␣Generic_Worker_Servlet␣on␣" + new Date();
  } // end getServletInfo()
```

```
// ============================================================================

//                            TEMPLATE METHODS
// ============================================================================


/**
 * TEMPLATE METHOD for casting the Request Bean to the corresponding
     type
 * e.g. Movie_Details_Resp_Bean.
 *
 * This method must be override in the particular worker servlets that
 * extend the Generic_Worker_Servlet
 *
 * @param reqBean The Request Bean sent by the MIDlet to the server
     side
 *               containing the client request data as an object with
 *               get and set methods
 * @return A Request Bean Object casted to the appropiate type
 *        e.g. Movie_Details_Resp_Bean
 */
protected abstract Object getRequestBean(Object reqBean);


/**
 * TEMPLATE METHOD for setting the SQL statement that is to be executed
     .
 *
 * This method must be override in the particular worker servlets that
     extend the
 * Generic_Worker_Servlet in order to define the particular SQL needed
 * for each case.
 *
 * @return The SQL statement to be executed
 */
protected abstract String setSQLStatement();


/**
 * TEMPLATE METHOD for parsing the result of the SQL statement and
     defining
 * the response to be sent to the client.
 *
 * This method must be override in the particular worker servlets that
     extend the
 * Generic_Worker_Servlet to parse different formats of results and
```

```
       define
   * the particular responses to be sent to the client function of each
       case.
   *
   * @param sqlResult The result of the SQL statemnt execution
   * @return        The response to be sent back to the client.
   */
  protected abstract Object parseSQLResponse(Vector sqlResult);

} // end class

package cinemaservice.servlets.workers;


import cinemaservice.model.beans.requestBeans.Movie_Details_Req_Bean;
import cinemaservice.model.beans.responseBeans.Response_Msg_Bean;
import cinemaservice.model.facade.FacadeToModel;

import java.util.Date;
import java.util.Vector;

import java.io.IOException;

import javax.servlet.*;
import javax.servlet.http.*;

import org.apache.log4j.*;


/**
 * Movie_Details_Servlet implements the Generic_Worker_Servlet.
 *
 * The TEMPLATE METHODS in the Generic_Worker_Servlet are implemented
 * as HOOK METHODS in here.
 *
 * Construct the parameter list to execute the Movie_Details stored
     procedure.
 * The parameter list is made of Show_Location_ID.
 *
 * Set the SQL Statemet to be executed to the Movie_Details stored
     procedure.
 *
 * Execute the SQL statement, then parse the result of the SQL and send
 * a response back to the client.
 *
 * The implementation is using the TEMPLATE METHOD, REFACTORING, FACADE,
 * and GENERALIZING Design Patterns.
 *
```

```
 * @author Mihai Balan - s031288
 */
public class Movie_Details_Servlet extends Generic_Worker_Servlet
    implements Servlet {

  // ============================================================================

  //                            DECLARATIONS
  // ============================================================================

  private static final long serialVersionUID = 3L;

  /**
   * Initiate a custom logging category
   */
  private static Category cat = Category.getInstance(
      Movie_Details_Servlet.class.getName());



  // ============================================================================

  //                              METHODS
  // ============================================================================

  /**
   * Initialize the Movie_Details_Servlet and calls
   * init() in the Generic_Worker_Servlet class.
   *
   * If the initialization fails, the servlet is not started
   *
   * @throws ServletException in case initialization fails
   */
  public void init(ServletConfig config) throws ServletException {
    // Store the ServletConfig object and log the initialization
    // Performs also the connection pooling initialization
    super.init(config);

  } // end init()


  /**
   * Deal with GET requests from the client side and calls
   * doGet() in the Generic_Worker_Servlet class.
   *
   * @param request The HTTP request from the client to the servlet
   * @param response The HTTP response from the servlet to the client
```

```
 */
protected void doGet(HttpServletRequest request, HttpServletResponse
    response)
  throws ServletException, IOException {
    super.doGet(request, response);

} // end doGet()


/**
 * Deal with POST requests from the client side and calls
 * doPost() in the Generic_Worker_Servlet class.
 *
 * @param request The HTTP request from the client to the servlet
 * @param response The HTTP response from the servlet to the client
 */
protected void doPost(HttpServletRequest request, HttpServletResponse
    response)
  throws ServletException, IOException {
    super.doPost(request, response);

} // end do Post()


/**
 * Returns information about the Movie_Details_Servlet
 *
 * @return Information about the Movie_Details_Servlet
 */
public String getServletInfo() {
  return "Hello␣from␣the␣Movie_Details_Servlet␣on␣" + new Date();

} // end getServletInfo()


// ============================================================================

//                             HOOK METHODS
// ============================================================================


/**
 * HOOK METHOD for casting the Request Bean to the corresponding type
 * e.g. Movie_Details_Req_Bean.
 *
 * @param requestBean The Movie_Details_Req_Bean sent by the MIDlet to
     the
```

```
 *                  server side containing the client request data as
 *                  an object with get and set methods
 * @return The Movie_Details_Req_Bean
 */
protected Movie_Details_Req_Bean getRequestBean(Object requestBean){

  cat.debug("Before␣the␣req␣bean");
  Movie_Details_Req_Bean movDetReqBean = (Movie_Details_Req_Bean)
      requestBean;
  cat.debug("After␣the␣req␣bean");
  return movDetReqBean;

} // end getRequestBean()



/**
 * HOOK METHOD for setting the SQL statement that is to be executed
 * i.e. to get details about the movie that is played in a given show.
 *
 * @return The SQL statement to be executed
 */
protected String setSQLStatement() {
  return "SELECT␣*␣FROM␣cinema.Movie_Details(?);";

} // end setSQLStatement()



/**
 * HOOK METHOD for parsing the result of the SQL statement
 * (the details about the requested movie)
 * and constructing the Movie_Details_Resp_Bean to be sent
 * to the client (the details about the requested movie).
 *
 * It delegates the control to the ResponseDataModel
 * responsable for parsing the SQLResult and creating
 * the Movie_Details_Resp_Bean.
 *
 *
 * @param sqlResult The result of the SQL statement execution
 * @return        The Movie_Details_Resp_Bean to be sent back to the
     client.
 */
protected Response_Msg_Bean parseSQLResponse(Vector sqlResult) {
  FacadeToModel facade = new FacadeToModel();
```

```
    return facade.setResponseMovieDetailsBean(sqlResult);

  } // end parseResponse()

} // end class

package cinemaservice.servlets.workers;


import cinemaservice.constants.Error_Code_Constants;
import cinemaservice.exceptions.CinemaServiceException;
import cinemaservice.model.beans.requestBeans.Purchase_Tickets_Req_Bean;
import cinemaservice.model.beans.responseBeans.Response_Msg_Bean;
import cinemaservice.model.facade.FacadeToModel;

import java.sql.SQLException;
import java.text.ParseException;
import java.util.Date;
import java.util.Vector;

import java.io.IOException;

import javax.naming.NamingException;
import javax.servlet.*;
import javax.servlet.http.*;

import org.apache.log4j.*;


/**
 * Purchase_Tickets_Servlet implements the Generic_Worker_Servlet.
 *
 * The TEMPLATE METHODS in the Generic_Worker_Servlet are implemented
 * as HOOK METHODS in here.
 *
 * Construct the parameter list to execute the
     Compute_Price_And_Maybe_Pay stored procedure.
 *
 * The input parameter list contains:
 *   - UserName
 *   - Password (OTP)
 *   - ShowLocationID.
 *   - ShowTimeID
 *   - SelectedSeats[][]
 *   - DiscountTypes[]
 *   - ReservationID
 *   - TicketIDs[]
 *   - ReservationDate
```

```
 *   - PaymentMethod
 *   - IsCreditCardValid
 *
 * The output parameter list contains:
 *   - StatusCode
 *   - ReservationID
 *   - TotalPriceToBePayd
 *   - LeftE-money
 *   - TicketID's
 *   - TicketPrices for each ticket
 *
 * Set the SQL Statemet to be executed to the Compute_Price_And_Maybe_Pay
      stored procedure.
 *
 * Execute the SQL statement, then parse the result of the SQL and send
 * a response back to the client.
 *
 * The implementation is using the TEMPLATE METHOD, REFACTORING, FACADE,
 * and GENERALIZING Design Patterns.
 *
 * @author Mihai Balan - s031288
 */
public class Purchase_Tickets_Servlet extends Generic_Worker_Servlet
    implements Servlet {

  // ============================================================================

  //                            DECLARATIONS
  // ============================================================================

  private static final long serialVersionUID = 3L;

  /**
   * Initiate a custom logging category
   */
  private static Category cat = Category.getInstance(
      Purchase_Tickets_Servlet.class.getName());



  // ============================================================================

  //                              METHODS
  // ============================================================================

  /**
   * Initialize the Purchase_Tickets_Servlet and calls
```

```
 * init() in the Generic_Worker_Servlet class.
 *
 * If the initialization fails, the servlet is not started
 *
 * @throws ServletException in case initialization fails
 */
public void init(ServletConfig config) throws ServletException {
  // Store the ServletConfig object and log the initialization
  // Performs also the connection pooling initialization
  super.init(config);

} // end init()


/**
 * Deal with GET requests from the client side and calls
 * doGet() in the Generic_Worker_Servlet class.
 *
 * @param request The HTTP request from the client to the servlet
 * @param response The HTTP response from the servlet to the client
 */
protected void doGet(HttpServletRequest request, HttpServletResponse
    response)
throws ServletException, IOException {
  super.doGet(request, response);

} // end doGet()


/**
 * Deal with POST requests from the client side and calls
 * doPost() in the Generic_Worker_Servlet class.
 *
 * @param request The HTTP request from the client to the servlet
 * @param response The HTTP response from the servlet to the client
 */
protected void doPost(HttpServletRequest request, HttpServletResponse
    response)
throws ServletException, IOException {
  super.doPost(request, response);

} // end do Post()


/**
 * Returns information about the Purchase_Tickets_Servlet
 *
```

```
 * @return Information about the Purchase_Tickets_Servlet
 */
public String getServletInfo() {
  return "Hello␣from␣the␣Purchase_Tickets_Servlet␣on␣" + new Date();

} // end getServletInfo()


// ============================================================================

//                             HOOK METHODS
// ============================================================================


/**
 * HOOK METHOD for casting the Request Bean to the corresponding type
 * i.e. Purchase_Tickets_Req_Bean.
 *
 * @param requestBean The Purchase_Tickets_Req_Bean sent by the MIDlet
     to the
 *                  server side containing the client request data as
 *                  an object with get and set methods
 * @return The Purchase_Tickets_Req_Bean
 */
protected Purchase_Tickets_Req_Bean getRequestBean(Object requestBean){

  cat.debug("Before␣the␣req␣bean");
  Purchase_Tickets_Req_Bean purchaseTicketsReqBean = (
      Purchase_Tickets_Req_Bean)requestBean;
  cat.debug("After␣the␣req␣bean");
  return purchaseTicketsReqBean;

} // end getRequestBean()



/**
 * HOOK METHOD for setting the SQL statement that is to be executed
 * i.e. to get details about the movie that is played in a given show.
 *
 * @return The SQL statement to be executed
 */
protected String setSQLStatement() {

  return "SELECT␣*␣FROM␣cinema.Compute_Price_And_Maybe_Pay
      (?,␣?,␣?,␣?,␣?,␣?,␣?,␣?,␣?,␣?,␣?);";
```

```
  } // end setSQLStatement()



  /**
   * HOOK METHOD for parsing the result of the SQL statement
   * (the result from purchasing the tickets i.e. reservation info, etc)
   * and constructing the Purchase_Tickets_Resp_Bean to be sent
   * to the client.
   *
   * It delegates the control to the ResponseDataModel
   * responsable for parsing the SQLResult and creating
   * the Purchase_Tickets_Resp_Bean.
   *
   *
   * @param sqlResult The result of the SQL statement execution
   * @return        The Purchase_Tickets_Resp_Bean to be sent back to the
        client.
   */
  protected Response_Msg_Bean parseSQLResponse(Vector sqlResult) {

    FacadeToModel facade = new FacadeToModel();
    Response_Msg_Bean rspBean = new Response_Msg_Bean(
        Error_Code_Constants.ERROR_WHILE_PAYING);

    try {
      return facade.setResponsePurchasedTicketsBean(sqlResult);

    } catch (SQLException sqle) {
      cat.fatal(sqle.getMessage());
      return rspBean;
    } catch (NamingException ne) {
      cat.fatal("----␣Payment␣System␣Error␣-␣JNDI␣Naming␣Exception␣" +
          "when␣trying␣to␣acquire␣the␣DataSource␣object\n" +
          ne.getMessage() + "\n" + ne.getStackTrace());
      return rspBean;
    } catch (ParseException pe) {
      cat.fatal(pe.getMessage());
      return rspBean;
    } catch (CinemaServiceException cse) {
      cat.fatal(cse.getMessage());
      return rspBean;
    }

  } // end parseResponse()

} // end class
```

```
package cinemaservice.servlets.workers;


import cinemaservice.model.beans.requestBeans.Rate_Movie_Req_Bean;
import cinemaservice.model.beans.responseBeans.Response_Msg_Bean;
import cinemaservice.model.facade.FacadeToModel;

import java.util.Date;
import java.util.Vector;

import java.io.IOException;

import javax.servlet.*;
import javax.servlet.http.*;

import org.apache.log4j.*;


/**
 * Rate_Movie_Servlet implments the Generic_Worker_Servlet.
 *
 * The TEMPLATE METHODS in the Generic_Worker_Servlet are implemented
 * as HOOK METHODS in here.
 *
 * It retrieves and casts the Rate_Movie_Req_Bean from the client.
 * Constructs the parameter list to execute the Rate_Movie stored
     procedure.
 *
 * The parameter list contains:
 *   - user_name
 *   - password
 *   - movie ID
 *   - movie rating score.
 *
 * Set the SQL Statemet to be executed to the SQL_Operations_Bean against
 * the pgSQL DB.
 *
 * Execute the SQL statement, parse the result of the SQL, and send
 * a Response Bean Object back to the client.
 *
 * The implementation is using the TEMPLATE METHOD, REFACTORING, FACADE
 * and GENERALIZING Design Patterns.
 *
 * @author Mihai Balan - s031288
 */
public class Rate_Movie_Servlet extends Generic_Worker_Servlet implements
     Servlet {
```

```
// ============================================================================

//                            DECLARATIONS
// ============================================================================

private static final long serialVersionUID = 3L;

/**
 * Initiate a custom logging category
 */
private static Category cat = Category.getInstance(Rate_Movie_Servlet.
    class.getName());

// ============================================================================

//                              METHODS
// ============================================================================

/**
 * Initialize the Rate_Movie_Servlet and calls
 * init() in the Generic_Worker_Servlet class.
 *
 * If the initialization fails, the servlet is not started
 *
 * @throws ServletException in case initialization fails
 */
public void init(ServletConfig config) throws ServletException {
  // Store the ServletConfig object and log the initialization
  // Performs also the connection pooling initialization
  super.init(config);

} // end init()


/**
 * Deal with GET requests from the client side and calls
 * doGet() in the Generic_Worker_Servlet class.
 *
 * @param request The HTTP request from the client to the servlet
 * @param response The HTTP response from the servlet to the client
 */
protected void doGet(HttpServletRequest request, HttpServletResponse
    response)
throws ServletException, IOException {

  super.doGet(request, response);
```

```
} // end doGet()


/**
 * Deal with POST requests from the client side and calls
 * doPost() in the Generic_Worker_Servlet class.
 *
 * @param request The HTTP request from the client to the servlet
 * @param response The HTTP response from the servlet to the client
 */
protected void doPost(HttpServletRequest request, HttpServletResponse
    response)
throws ServletException, IOException {

  super.doPost(request, response);

} // end do Post()


/**
 * Returns information about the Rate_Movie_Servlet
 *
 * @return Information about the Rate_Movie_Servlet
 */
public String getServletInfo() {
  return "Hello␣from␣the␣Rate_Movie_Servlet␣on␣" + new Date();

} // end getServletInfo()


// ============================================================================

//                            HOOK METHODS
// ============================================================================


/**
 * HOOK METHOD for casting the Request Bean to the corresponding type
 * e.g. Rate_Movie_Req_Bean.
 *
 * @param requestBean The Rate_Movie_Req_Bean sent by the MIDlet to the
      server side
 *                  containing the client request data as an object
     with
 *                  get and set methods
 * @return The Rate_Movie_Req_Bean
```

```
   */
  protected Rate_Movie_Req_Bean getRequestBean(Object requestBean){

    cat.debug("Before␣the␣Rate␣Movie␣req␣bean");
    Rate_Movie_Req_Bean rateMovBean = (Rate_Movie_Req_Bean)requestBean;
    cat.debug("After␣the␣Rate␣Movie␣req␣bean");
    return rateMovBean;

  } // end getRequestBean()



  /**
   * HOOK METHOD for setting the SQL statement that is to be executed
   * i.e. to get details about the movie that is played in a given show.
   *
   * @return The SQL statement to be executed
   */
  protected String setSQLStatement() {
    return "SELECT␣*␣FROM␣cinema.Rate_Movie(?,␣?,␣?,␣?);␣";

  } // end setSQLStatement()


  /**
   * HOOK METHOD for parsing the result of the SQL statement
   * (the response from rating the movie)
   * and defining the response to be sent to the client
   * (the response code to the user).
   *
   * It delegates the control to the ResponseDataModel
   * responsable for parsing the SQLResult and creating
   * the Response_Msg_Bean.
   *
   * @param sqlResult The result of the SQL statement execution
   * @return        The response to be sent back to the client.
   */
  protected Response_Msg_Bean parseSQLResponse(Vector sqlResult) {

    FacadeToModel facade = new FacadeToModel();
    return facade.setResponseRateMovieBean(sqlResult);

  } // end parseSQLResponse()

} // end class

package cinemaservice.servlets.workers;
```

```
import cinemaservice.model.beans.requestBeans.Reject_Payment_Req_Bean;
import cinemaservice.model.beans.responseBeans.Response_Msg_Bean;
import cinemaservice.model.facade.FacadeToModel;

import java.util.Date;
import java.util.Vector;

import java.io.IOException;

import javax.servlet.*;
import javax.servlet.http.*;

import org.apache.log4j.*;


/**
 * Reject_Payment_Servlet implments the Generic_Worker_Servlet.
 *
 * The TEMPLATE METHODS in the Generic_Worker_Servlet are implemented
 * as HOOK METHODS in here.
 *
 * It retrieves and casts the Reject_Payment_Req_Bean from the client.
 * Constructs the parameter list to execute the
 * Reject_Payment_Cancel_Selected_Seats stored procedure.
 *
 * The parameter list contains:
 *  - showLocationID
 *  - showTimeID
 *  - seats[][].
 *
 * Set the SQL Statemet to be executed to the SQL_Operations_Bean against
 * the pgSQL DB.
 *
 * Execute the SQL statement, parse the result of the SQL, and send
 * a Response_Msg_Bean Object back to the client.
 *
 * The implementation is using the TEMPLATE METHOD, REFACTORING, FACADE
 * and GENERALIZING Design Patterns.
 *
 * @author Mihai Balan - s031288
 */
public class Reject_Payment_Servlet extends Generic_Worker_Servlet
    implements Servlet {

  // ============================================================================
```

```
//                            DECLARATIONS
// ================================================================================

private static final long serialVersionUID = 3L;

/**
 * Initiate a custom logging category
 */
private static Category cat = Category.getInstance(
    Reject_Payment_Servlet.class.getName());

// ================================================================================

//                               METHODS
// ================================================================================

/**
 * Initialize the Reject_Payment_Servlet and calls
 * init() in the Generic_Worker_Servlet class.
 *
 * If the initialization fails, the servlet is not started
 *
 * @throws ServletException in case initialization fails
 */
public void init(ServletConfig config) throws ServletException {
  // Store the ServletConfig object and log the initialization
  // Performs also the connection pooling initialization
  super.init(config);

} // end init()


/**
 * Deal with GET requests from the client side and calls
 * doGet() in the Generic_Worker_Servlet class.
 *
 * @param request The HTTP request from the client to the servlet
 * @param response The HTTP response from the servlet to the client
 */
protected void doGet(HttpServletRequest request, HttpServletResponse
    response)
throws ServletException, IOException {

  super.doGet(request, response);

} // end doGet()
```

```
/**
 * Deal with POST requests from the client side and calls
 * doPost() in the Generic_Worker_Servlet class.
 *
 * @param request The HTTP request from the client to the servlet
 * @param response The HTTP response from the servlet to the client
 */
protected void doPost(HttpServletRequest request, HttpServletResponse
    response)
throws ServletException, IOException {

  super.doPost(request, response);

} // end do Post()


/**
 * Returns information about the Reject_Payment_Servlet
 *
 * @return Information about the Reject_Payment_Servlet
 */
public String getServletInfo() {
  return "Hello␣from␣the␣Reject_Payment_Servlet␣on␣" + new Date();

} // end getServletInfo()


// ============================================================================

//                            HOOK METHODS
// ============================================================================


/**
 * HOOK METHOD for casting the Request Bean to the corresponding type
 * e.g. Reject_Payment_Req_Bean.
 *
 * @param requestBean The Reject_Payment_Req_Bean sent by the MIDlet to
      the server side
 *                   containing the client request data as an object
     with
 *                   get and set methods
 * @return The Reject_Payment_Req_Bean
 */
protected Reject_Payment_Req_Bean getRequestBean(Object requestBean){
```

```
    cat.debug("Before␣the␣Reject_Payment_Req_Bean");
    Reject_Payment_Req_Bean rejectPayBean = (Reject_Payment_Req_Bean)
        requestBean;
    cat.debug("After␣the␣Reject_Payment_Req_Bean");
    return rejectPayBean;

  } // end getRequestBean()


  /**
   * HOOK METHOD for setting the SQL statement that is to be executed
   * i.e. to get the cinema hall configuration for the given show.
   *
   * @return The SQL statement to be executed
   */
  protected String setSQLStatement() {
    return "SELECT␣*␣FROM␣cinema.Reject_Payment_Cancel_Selected_Seats
        (?,␣?,␣?);";

  } // end setSQLStatement()


  /**
   * HOOK METHOD for parsing the result of the SQL statement
   * (the response from rejecting the ticket payment)
   * and defining the response to be sent to the client
   * (the response code to the user).
   *
   * It delegates the control to the ResponseDataModel
   * responsable for parsing the SQLResult and creating
   * the Response_Msg_Bean.
   *
   * @param sqlResult The result of the SQL statement execution
   * @return        The response to be sent back to the client.
   */
  protected Response_Msg_Bean parseSQLResponse(Vector sqlResult) {

    FacadeToModel facade = new FacadeToModel();
    return facade.setResponseRejectPaymentBean(sqlResult);

  } // end parseSQLResponse()

} // end class

package cinemaservice.servlets.workers;


import cinemaservice.model.beans.requestBeans.
```

```
    Select_Deselect_Seats_Req_Bean;
import cinemaservice.model.beans.responseBeans.Response_Msg_Bean;
import cinemaservice.model.facade.FacadeToModel;

import java.util.Date;
import java.util.Vector;

import java.io.IOException;

import javax.servlet.*;
import javax.servlet.http.*;

import org.apache.log4j.*;


/**
 * Select_Deselect_Seats_Servlet implments the Generic_Worker_Servlet.
 *
 * The TEMPLATE METHODS in the Generic_Worker_Servlet are implemented
 * as HOOK METHODS in here.
 *
 * It retrieves and casts the Select_Deselect_Seats_Req_Bean from the
     client.
 * Constructs the parameter list to execute the
     Select_Deselect_Many_Seats stored procedure.
 *
 * The parameter list contains:
 *  - command_code
 *  - showLocationID
 *  - showTimeID
 *  - selecredSeats[]
 *
 * Set the SQL Statemet to be executed to the SQL_Operations_Bean against
 * the pgSQL DB.
 *
 * Execute the SQL statement, parse the result of the SQL, and send
 * a Select_Deselect_Seats_Resp_Bean Object back to the client.
 *
 * The implementation is using the TEMPLATE METHOD, REFACTORING, FACADE
 * and GENERALIZING Design Patterns.
 *
 * @author Mihai Balan - s031288
 */
public class Select_Deselect_Seats_Servlet extends Generic_Worker_Servlet
     implements Servlet {

  // ============================================================================
```

```
//                            DECLARATIONS
// ============================================================================

private static final long serialVersionUID = 3L;

/**
 * Initiate a custom logging category
 */
private static Category cat = Category.getInstance(
    Select_Deselect_Seats_Servlet.class.getName());

// ============================================================================

//                               METHODS
// ============================================================================

/**
 * Initialize the Select_Deselect_Seats_Servlet and calls
 * init() in the Generic_Worker_Servlet class.
 *
 * If the initialization fails, the servlet is not started
 *
 * @throws ServletException in case initialization fails
 */
public void init(ServletConfig config) throws ServletException {
  // Store the ServletConfig object and log the initialization
  // Performs also the connection pooling initialization
  super.init(config);

} // end init()


/**
 * Deal with GET requests from the client side and calls
 * doGet() in the Generic_Worker_Servlet class.
 *
 * @param request The HTTP request from the client to the servlet
 * @param response The HTTP response from the servlet to the client
 */
protected void doGet(HttpServletRequest request, HttpServletResponse
    response)
throws ServletException, IOException {

  super.doGet(request, response);

} // end doGet()
```

```
/**
 * Deal with POST requests from the client side and calls
 * doPost() in the Generic_Worker_Servlet class.
 *
 * @param request The HTTP request from the client to the servlet
 * @param response The HTTP response from the servlet to the client
 */
protected void doPost(HttpServletRequest request, HttpServletResponse
    response)
throws ServletException, IOException {

  super.doPost(request, response);

} // end do Post()


/**
 * Returns information about the Select_Deselect_Seats_Servlet
 *
 * @return Information about the Select_Deselect_Seats_Servlet
 */
public String getServletInfo() {
  return "Hello␣from␣the␣Select_Deselect_Seats_Servlet␣on␣" + new Date
      ();

} // end getServletInfo()


// ============================================================================

//                            HOOK METHODS
// ============================================================================


/**
 * HOOK METHOD for casting the Request Bean to the corresponding type
 * i.e. Select_Deselect_Seats_Req_Bean.
 *
 * @param requestBean The Select_Deselect_Seats_Req_Bean sent by the
     MIDlet to the server side
 *                  containing the client request data as an object
     with
 *                  get and set methods
 * @return The Select_Deselect_Seats_Req_Bean
 */
```

```
protected Select_Deselect_Seats_Req_Bean getRequestBean(Object
    requestBean){

  cat.debug("Before␣the␣Select_Deselect_Seats_Req_Bean");
  Select_Deselect_Seats_Req_Bean selDeselSeatsBean = (
      Select_Deselect_Seats_Req_Bean)requestBean;
  cat.debug("After␣the␣Select_Deselect_Seats_Req_Bean");
  return selDeselSeatsBean;

} // end getRequestBean()



/**
 * HOOK METHOD for setting the SQL statement that is to be executed
 * i.e. to select / deselect user chosen seats and return all booked
 * seats for the given show
 *
 * @return The SQL statement to be executed
 */
protected String setSQLStatement() {
  return "SELECT␣*␣FROM␣cinema.Select_Deselect_Many_Seats␣(?,␣?,␣?,␣?)"
      ;

} // end setSQLStatement()


/**
 * HOOK METHOD for parsing the result of the SQL statement
 * (all booked seats for the given show)
 * and defining the response to be sent to the client
 * (the response code to the user).
 *
 * It delegates the control to the ResponseDataModel
 * responsable for parsing the SQLResult and creating
 * the Select_Deselect_Seats_Resp_Bean.
 *
 * @param sqlResult The result of the SQL statement execution
 * @return        The response to be sent back to the client.
 */
protected Response_Msg_Bean parseSQLResponse(Vector sqlResult) {

  FacadeToModel facade = new FacadeToModel();
  return facade.setResponseSelectDeselectSeats(sqlResult);

} // end parseSQLResponse()
```

```
} // end class

package cinemaservice.servlets.workers;


import cinemaservice.model.beans.requestBeans.Cinema_Hall_Conf_Req_Bean;
import cinemaservice.model.beans.responseBeans.Response_Msg_Bean;
import cinemaservice.model.facade.FacadeToModel;

import java.util.Date;
import java.util.Vector;

import java.io.IOException;

import javax.servlet.*;
import javax.servlet.http.*;

import org.apache.log4j.*;


/**
 * Select_Show_Servlet implments the Generic_Worker_Servlet.
 *
 * The TEMPLATE METHODS in the Generic_Worker_Servlet are implemented
 * as HOOK METHODS in here.
 *
 * It retrieves and casts the Cinema_Hall_Conf_Req_Bean from the client.
 * Constructs the parameter list to execute the Display_Cinema_Hall_Conf
     stored procedure.
 *
 * The parameter list contains:
 *  - showLocationID
 *  - showTimeID.
 *
 * Set the SQL Statemet to be executed to the SQL_Operations_Bean against
 * the pgSQL DB.
 *
 * Execute the SQL statement, parse the result of the SQL, and send
 * a Cinema_Hall_Conf_Res_Bean Object back to the client.
 *
 * The implementation is using the TEMPLATE METHOD, REFACTORING, FACADE
 * and GENERALIZING Design Patterns.
 *
 * @author Mihai Balan - s031288
 */
public class Select_Show_Servlet extends Generic_Worker_Servlet
    implements Servlet {
```

```
// ============================================================================

//                            DECLARATIONS
// ============================================================================

private static final long serialVersionUID = 3L;

/**
 * Initiate a custom logging category
 */
private static Category cat = Category.getInstance(Select_Show_Servlet.
    class.getName());

// ============================================================================

//                              METHODS
// ============================================================================

/**
 * Initialize the Select_Show_Servlet and calls
 * init() in the Generic_Worker_Servlet class.
 *
 * If the initialization fails, the servlet is not started
 *
 * @throws ServletException in case initialization fails
 */
public void init(ServletConfig config) throws ServletException {
  // Store the ServletConfig object and log the initialization
  // Performs also the connection pooling initialization
  super.init(config);

} // end init()


/**
 * Deal with GET requests from the client side and calls
 * doGet() in the Generic_Worker_Servlet class.
 *
 * @param request The HTTP request from the client to the servlet
 * @param response The HTTP response from the servlet to the client
 */
protected void doGet(HttpServletRequest request, HttpServletResponse
    response)
throws ServletException, IOException {

  super.doGet(request, response);
```

```
} // end doGet()


/**
 * Deal with POST requests from the client side and calls
 * doPost() in the Generic_Worker_Servlet class.
 *
 * @param request The HTTP request from the client to the servlet
 * @param response The HTTP response from the servlet to the client
 */
protected void doPost(HttpServletRequest request, HttpServletResponse
    response)
throws ServletException, IOException {

  super.doPost(request, response);

} // end do Post()


/**
 * Returns information about the Select_Show_Servlet
 *
 * @return Information about the Select_Show_Servlet
 */
public String getServletInfo() {
  return "Hello␣from␣the␣Select_Show_Servlet␣on␣" + new Date();

} // end getServletInfo()


// ============================================================================

//                            HOOK METHODS
// ============================================================================


/**
 * HOOK METHOD for casting the Request Bean to the corresponding type
 * e.g. Cinema_Hall_Conf_Req_Bean.
 *
 * @param requestBean The Cinema_Hall_Conf_Req_Bean sent by the MIDlet
     to the server side
 *                  containing the client request data as an object
     with
 *                  get and set methods
 * @return The Cinema_Hall_Conf_Req_Bean
 */
```

```
  protected Cinema_Hall_Conf_Req_Bean getRequestBean(Object requestBean){

    cat.debug("Before␣the␣Cinema_Hall_Conf_Req_Bean");
    Cinema_Hall_Conf_Req_Bean cinHallConfBean = (
        Cinema_Hall_Conf_Req_Bean)requestBean;
    cat.debug("After␣the␣Cinema_Hall_Conf_Req_Bean");
    return cinHallConfBean;

  } // end getRequestBean()



  /**
   * HOOK METHOD for setting the SQL statement that is to be executed
   * i.e. to get the cinema hall configuration for the given show.
   *
   * @return The SQL statement to be executed
   */
  protected String setSQLStatement() {
    return "SELECT␣*␣FROM␣cinema.Display_Cinema_Hall_Conf(?,␣?);";

  } // end setSQLStatement()


  /**
   * HOOK METHOD for parsing the result of the SQL statement
   * (the response from retrieving the cinema hall configuration)
   * and defining the response to be sent to the client
   * (the response code to the user).
   *
   * It delegates the control to the ResponseDataModel
   * responsable for parsing the SQLResult and creating
   * the Cinema_Hall_Conf_Res_Bean.
   *
   * @param sqlResult The result of the SQL statement execution
   * @return        The response to be sent back to the client.
   */
  protected Response_Msg_Bean parseSQLResponse(Vector sqlResult) {

    FacadeToModel facade = new FacadeToModel();
    return facade.setResponseCinemaHallConfBean(sqlResult);

  } // end parseSQLResponse()

} // end class

package creditcardvalidator;
```

```
/**
 * This is an emulator for a Credit Card Validation & Payment Service
 *
 * @author Mihai Balan - s031288
 *
 */
public class CardValidator {

  /**
   * Check if the credit card is valid
   *
   * @param creditCardType
   * @param creditCardNo
   * @param creditCardExpDate
   * @param creditCardCW2
   * @return Credit card is valid or not
   */
  public boolean validateCreditCard(String creditCardType, String
      creditCardNo, String creditCardExpDate, String creditCardCW2){

    if (creditCardType.toUpperCase().equals("VISA"))
      return true;
    else
      return false;

  } // end validateCreditCard()


  /**
   * Makes the payment using the given credit card data
   *
   * @param creditCardType
   * @param creditCardNo
   * @param creditCardExpDate
   * @param creditCardCW2
   * @param amount
   *
   * @return Successful payment or not
   */
  public boolean pay(String creditCardType, String creditCardNo, String
      creditCardExpDate, String creditCardCW2, double amount){

    if (creditCardType.toUpperCase().equals("VISA"))
      return true;
    else
      return false;
```

```
  }// end pay()

} // end class

package movie_location_service;

import java.sql.*;
import java.util.Vector;

/**
 * @author Mihai Balan - s031288
 */
public class DBConnTools
{
  private Connection con;

  /**
   * Constructor
   */
  public DBConnTools() throws ClassNotFoundException, SQLException{

    con = connect();

  } // end DBConnTools()


  /**
   * Establish a connection to the DB
   */
  private static Connection connect() throws ClassNotFoundException,
      SQLException{

    String url = "jdbc:postgresql://localhost:5432/postgres";
    String user = "zeratul";
    String passwd = "ericsson";

    Class.forName("org.postgresql.Driver");
    return DriverManager.getConnection(url,user,passwd);

  }// end connect()


  /**
   * Extract a particular cinema from the DB based on a given address
   *
   * @param street Street where the cinema is located
   * @param city  City where the cinema is located
   * @param zip   Zip code of the city where the cinema is located
```

```
 * @param range The max distance where the system is searching for a
     cinema
 *              from the current user's position
 *
 * @return      Cinemas' ID
 * @throws SQLException
 */
public Vector getCinemaIDs(String street, String city, String zip,
    String range) throws SQLException {

  Vector sqlResult = new Vector();
  Vector sqlParam = new Vector(5);

  //Connection conn = null;
  PreparedStatement pgPsqlStmt = null;
  ResultSet rs = null;

  String sqlStmt = "SELECT␣*␣FROM␣cinema.Movie_Location_Service
      (?,␣?,␣?,␣?,␣?);";

  int rangeInt = Integer.MAX_VALUE;

  try {
    rangeInt = Integer.parseInt(range);

  } catch (NumberFormatException nfe){
    System.out.println("Range␣set␣to␣max:␣" + nfe.getMessage());

  } // end try - catch

  sqlParam.add(rangeInt);        // range
  sqlParam.add(new String(""));  // movie
  sqlParam.add(new String("%" + street + "%")); // street
  sqlParam.add(new String("%" + city + "%")); // city
  sqlParam.add(new String("%" + zip + "%")); // zip

  try{

    // preapare the SQL statement
    pgPsqlStmt = con.prepareStatement(sqlStmt);

    // set the parameters to execute the query
    for (int i = 0; i < 5; i++){
      pgPsqlStmt.setObject(i + 1, sqlParam.elementAt(i));
    }

    // execute the prepared statement
```

```
  rs = pgPsqlStmt.executeQuery();

  // get the result from the result set as a String
  ResultSetMetaData dbMeta = rs.getMetaData();
  while (rs.next()) {
    for (int col=0; col < dbMeta.getColumnCount(); col++) {
      sqlResult.addElement(rs.getObject(col+1));

    } // end for

  } // end while()

  // close the record set and the prepared statment
  rs.close();
  pgPsqlStmt.close();

  // return the connection to the pool of connections
  con.close();

}
// perform any clean up in case any connection, statement remains
    opened and not used
finally {
  try {
    if (rs != null && !con.isClosed())
      rs.close();

  } catch (SQLException sqle1) {
    System.out.println("SQL␣exception␣" +
        "when␣trying␣to␣close␣the␣record␣set␣in␣FINALLY␣clause␣...␣"
        + sqle1.getMessage());
  }
  try {
    if (pgPsqlStmt != null && !con.isClosed())
      pgPsqlStmt.close();

  } catch (SQLException sqle2) {
    System.out.println("SQL␣exception␣" +
        "when␣trying␣to␣close␣the␣statement␣in␣FINALLY␣clause␣...␣"
        + sqle2.getMessage());
  }
  try {
    if (!con.isClosed())
      con.close();

  } catch (SQLException sqle3) {
    System.out.println("SQL␣exception␣" +
```

```
          "when␣trying␣to␣close␣the␣connection␣in␣FINALLY␣clause␣...␣"
          + sqle3.getMessage());
     }
    }// end try - catch - finally

    return sqlResult;

  } // end getCinemaIDs()

} // end class

package movie_location_service;

import java.sql.SQLException;
import java.util.Vector;

import cinemaservice.constants.SQL_Return_Codes;

/**
 * Emulates a real Cinema Location Service that locates
 * cinemas in a given area
 *
 * @author Mihai Balan - s031288
 *
 */
public class MovieLocationService {

  public MovieLocationService(){}

  /** Returns the list of Cinema IDs corresposnding to the user searching
      criteria
   *
   * @param street The street of the user's current location
   * @param city  The city of the user's current location
   * @param zip   The zip of the user's current location
   * @param range The range within user's current position the MLS has to
       find cinemas
   *
   * @return A list of all cinema IDs matching user's given searching
      criteria
   */
  public Vector getCinemas(String street, String city, String zip, String
      range){

    // connect to the DB and get all cinema ID's
    // of all cinemas matching user's searching criteria
    // i.e. find all cinemas in the given range from the user's current
       position
```

```
    Vector res = new Vector(2);

    try {

      DBConnTools dbConn = new DBConnTools();
      Vector cin = new Vector(1);
      cin = dbConn.getCinemaIDs(street, city, zip, range);
      System.out.println("=======MLS␣length␣␣" + cin.get(0).toString() +
          "\n" + cin.get(0).toString().length());

      if (cin == null){
        res.add(SQL_Return_Codes.MOVIE_LOCATION_SERVICE_ERROR);
        res.add(null);
        return res;
      }

      if (cin.get(0).toString().length() > 3){
        res.add(SQL_Return_Codes.MOVIE_LOCATION_SERVICE_OK);
        res.add(cin);
        return res;

      } else{
        res.add(SQL_Return_Codes.MOVIE_LOCATION_SERVICE_NO_DATA);
        res.add(null);
        return res;
      }

    } catch (ClassNotFoundException cne){
      System.out.println("Exception␣in␣MLS:␣" + cne.getMessage());
      res.add(SQL_Return_Codes.MOVIE_LOCATION_SERVICE_ERROR);
      res.add(null);
      return res;

    } catch (SQLException sqle){
      System.out.println("Exception␣in␣MLS:␣" + sqle.getMessage());
      res.add(SQL_Return_Codes.MOVIE_LOCATION_SERVICE_ERROR);
      res.add(null);
      return res;
    }

  }// end getCinemas();

} // end class
```

D.3 Database

```
-- # ========================================================
-- #          STEP 1 --- AUTHENTICATION PROTOCOL
-- # --------------------------------------------------------
-- #            User Authentication Procedure
-- #
-- # Authenticated the user and returns an error code i.e:
-- #     401 = user not authenticated
-- #     201 = user authenticated
-- # --------------------------------------------------------
-- # The followIng format is to be used to call the function:
-- # SELECT * FROM Authenticate(User Name, Password);
-- # e.g.
-- # SET search_path TO cinema, public;
-- # SET DATESTYLE TO ISO;
-- # SELECT * FROM Authenticate('adm', '12345678');
-- # ========================================================

SET search_path TO cinema, public;
SET DATESTYLE TO ISO;

CREATE FUNCTION Authenticate (TEXT, TEXT) RETURNS TEXT AS '

  DECLARE
    -- Declare aliases for user input.
    user_name ALIAS FOR $1;
    password ALIAS FOR $2;

    -- Declare a variable to hold the return error.
    return_error TEXT;

  BEGIN
    SET search_path TO cinema, public;
    SET DATESTYLE TO ISO;

    SELECT INTO return_error username
    FROM cinema.Users
    WHERE cinema.Users.UserName = user_name
    AND   cinema.Users.Password = password;

    IF NOT FOUND THEN
      RETURN 401;
    ELSE
      RETURN 201;
    END IF;
```

```
  END;
' LANGUAGE 'plpgsql';
-- ======================================================



SET search_path TO cinema, public;
SET DATESTYLE TO ISO;

CREATE TYPE UsersEMoney AS
   (
  e_money TEXT,
  userID TEXT,
  randomID TEXT
   );


-- # ======================================================
-- #         STEP 1.a --- AUTHENTICATION PROTOCOL
-- # ------------------------------------------------------
-- #            User Authentication Procedure
-- #
-- # Authenticated the user and returns the amount of e-money
-- # in user's account
-- #     401 = user not authenticated
-- #     221 = user authenticated & OK
-- # ------------------------------------------------------
-- # The followIng format is to be used to call the function:
-- # SELECT * FROM Authenticate_E_Money(User Name, Password);
-- # e.g.
-- # SET search_path TO cinema, public;
-- # SET DATESTYLE TO ISO;
-- # SELECT * FROM Authenticate_E_Money('adm', '12345678');
-- # ======================================================

SET search_path TO cinema, public;
SET DATESTYLE TO ISO;

CREATE OR REPLACE FUNCTION Authenticate_E_Money (TEXT, TEXT) RETURNS TEXT
     AS '

  DECLARE
    -- Declare aliases for user input.
    user_name ALIAS FOR $1;
    password ALIAS FOR $2;

    -- Declare a variable to hold the authentication result
```

```
    -- i.e. 201 for user authenticated, 401 else
    error_code INTEGER;

    -- Declare a variable to hold the return error.
    output TEXT;

    row_data_user  cinema.UsersEMoney%ROWTYPE;

  BEGIN
    SET search_path TO cinema, public;
    SET DATESTYLE TO ISO;

    -- verify if user credentials are correct
    error_code := Authenticate(user_name, password);

    -- if user is authenticated
    IF error_code = 201 THEN

      FOR row_data_user IN
        SELECT e_money, userID, randomID
        FROM  cinema.Users
        WHERE cinema.Users.UserName = user_name
        AND   cinema.Users.Password = password
      LOOP
             -- create the output
                  output := ''221'' || ''\n'' || row_data_user.userID ||
                      ''\n'' || row_data_user.randomID|| ''\n'' ||
                      row_data_user.e_money || ''\n'';
      END LOOP;

      RETURN output;

     ELSE

  RETURN 401;

     END IF;

  END;
'␣LANGUAGE␣'plpgsql';
-- ========================================================


-- # ============================================================================

-- #             STEP 4.1 --- BACKGROUND CINEMA HALL UPDATE
-- # ----------------------------------------------------------------------------
```

```
-- #               Background_Cinema_Hall_Update Procedure
-- #
-- # Find all booked seats for the given show
-- # base price for that show, discoutn values, and all boooked seats
-- #  - It accepts 2 input arguments i.e. ShowLocationID and ShowTimeID
-- #  - It  returns an error code followed by a list of all booked seats
-- #       - error code values:
-- #           204 - Show found according to the given criteria
-- #           404 - Show NOT found according to the given criteria
-- # ------------------------------------------------------------------------

-- # The followong format is to be used to call the function:
-- # SELECT * FROM Background_Cinema_Hall_Update(ShowLocationID,
    ShowTimeID);
-- # e.g.
-- # SET search_path TO cinema, public;
-- # SET DATESTYLE TO ISO;
-- # SELECT * FROM Background_Cinema_Hall_Update(4, 15);
-- # ========================================================================


SET search_path TO cinema, public;
SET DATESTYLE TO ISO;

CREATE OR REPLACE FUNCTION Background_Cinema_Hall_Update(INTEGER, INTEGER
    ) returns text AS '
DECLARE

    -- Declare aliases for user input.
    show_location_id ALIAS FOR $1;
    show_time_id     ALIAS FOR $2;

   -- Declare a variable to hold the booked seats for that show
    text_output_seats TEXT = '''';
    error_code       TEXT = ''204'';

    myRec Record;

    -- Declare a variable to hold rows of BookedSeats type
    row_data_booked  cinema.BookedSeats%ROWTYPE;


BEGIN
  SET search_path TO cinema, public;
  SET DATESTYLE TO German;

  -- check if the given show_location_id and show_time_id exists
```

```
  -- in the shows table and create the error_code based on that
  SELECT INTO myrec * FROM cinema.shows
  WHERE cinema.shows.showlocationid = show_location_id
  AND cinema.shows.showtimeid = show_time_id;

  IF NOT FOUND THEN
    error_code = ''404'';
  END IF;

  -- find all booked seats for that show and add them to the output
      string
  FOR row_data_booked IN
    SELECT *
    FROM  cinema.BookedSeats
    WHERE cinema.BookedSeats.ShowLocationID = show_location_id
    AND   cinema.BookedSeats.ShowTimeID  = show_time_id
    ORDER BY cinema.BookedSeats.RowNo, cinema.BookedSeats.SeatNo
  LOOP
        -- Insert the result of the select into the text_output variable.
        IF text_output_seats = '''' THEN
    text_output_seats =   text_output_seats    || ''('' ||
          row_data_booked.RowNo || '','' ||
          row_data_booked.SeatNo || '')'' ;
        ELSE
    text_output_seats =   text_output_seats    || ''|(''||
          row_data_booked.RowNo || '','' ||
          row_data_booked.SeatNo || '')'' ;
        END IF;

  END LOOP;

  RETURN error_code || ''\n'' ||text_output_seats;
END;
'language␣plpgsql;
-- ============================================================================


-- # ============================================================================

-- #
-- # STEP 6 -- CANCEL A PREVIOUS MADE RESERVATION OR ONLY SOME TICKETS
     FOR
-- #          A PREVIOUS MADE RESERVATION. REFUND THOSE TICKETS BY USING
      E-MONEY
-- #
-- # ----------------------------------------------------------------------------

-- #                          Cancel_Tickets Procedure
```

```
-- #
-- # Check if the user is authenticated and in that case cancel the given
-- # reservation or ticket(s) in that reservation that have been purchase
     by using
-- # the CARD payment method. Refund those tickets by e-money
-- # Save user's␣Ticket␣ID's and reservation ID in the DB
-- #   - It accepts as input 4 arguments i.e. UserName, OTP,
-- #         ReservationID, and an array of [TicketIDs]
-- #   - It returns an erro_code i.e.
-- #         214 = tickets canceled
-- #         201 = user authenticated,
-- #         401 = user not authenticated,
-- #         414 = error,
-- # ============================================================================


SET search_path TO cinema, public;
SET DATESTYLE TO ISO;

CREATE OR REPLACE FUNCTION Cancel_Tickets(TEXT, TEXT, TEXT, TEXT[])
    returns text AS '
DECLARE

    -- Declare aliases for user input.
    user_name       ALIAS FOR $1;
    otp             ALIAS FOR $2;
    reservation_id  ALIAS FOR $3;
    ticket_ids      ALIAS FOR $4;

    -- Declare a variable to hold the authentication result
    -- i.e. 201 for user authenticated, 401 else
    error_code INTEGER;

    -- Dimension of the array of seat_row array
    dimension_tickets     INTEGER;

    -- number of left tickeets for the given reservation ID
    -- after deleting the given tickets
    left_tickets          INTEGER;

    -- the amount of e_money to be refunded
    emoney                 float;

    -- Declare a variable to hold the output result
    text_output TEXT = '''';

    -- Declare a variable to hold rows from of Prices,
```

```
    -- Discount types, respectively
    row_data_price      cinema.Prices%ROWTYPE;
    row_data_discount   cinema.ShowDiscountValue%ROWTYPE;
    row_data_discount_id cinema.ShowDiscountID%ROWTYPE;
    row_data_booked_seat cinema.BookedSeats%ROWTYPE;

BEGIN
  SET search_path TO cinema, public;
  SET DATESTYLE TO ISO;

  -- get the number of tickets in the input array
  dimension_tickets := ARRAY_UPPER(ticket_ids, 1);

  -- verify if user credentials are correct
  error_code := Authenticate(user_name, otp);

       -- if user is authenticated
  IF error_code = 201 THEN

    -- get the base price for the given show
    SELECT INTO emoney SUM(Tickets.FinalPrice)
    FROM cinema.Tickets
    WHERE cinema.Tickets.ResID = (
      SELECT Reservations.ResID
      FROM  cinema.Reservations, cinema.PaymentMethod
      WHERE cinema.Reservations.ResID          = reservation_id
      AND   cinema.Reservations.UserName       = user_name
      AND   cinema.PaymentMethod.PaymentMethodID = cinema.Reservations.
          PaymentMethodID
      AND   cinema.PaymentMethod.PaymentMethodType = UPPER(''CARD''))
    AND cinema.Tickets.TicketID = ANY(ticket_ids);

    -- if the given tickets or reservation IDs could not be found return
        an error
    IF emoney IS NULL THEN
      error_code := 414;

    ELSE
      -- begin TRANSACTION mode
      BEGIN
        -- if the tickets were found delete them and the coreponding
            booked seats
        FOR i IN 1..dimension_tickets LOOP
          DELETE
          FROM cinema.BookedSeats
          WHERE cinema.BookedSeats.BookedSeatID =
            (SELECT Tickets.BookedSeatID
```

```
          FROM cinema.Tickets
          WHERE cinema.Tickets.TicketID = ticket_ids[i]);
      END LOOP;


      -- check if the no. of tickets for the given reservation is 0
      -- and delete the reservation only in this case

      SELECT INTO left_tickets Count(*)
      FROM cinema.Tickets
      WHERE cinema.Tickets.ResID = (
        SELECT Reservations.ResID
        FROM  cinema.Reservations
        WHERE cinema.Reservations.ResID = reservation_id);


      IF left_tickets = 0 THEN
        DELETE
        FROM  cinema.Reservations
        WHERE cinema.Reservations.ResID = reservation_id;

      END IF;

      -- and update the e-money amount for that user
      UPDATE cinema.Users SET E_Money = emoney +
             (SELECT Users.E_Money FROM cinema.Users
              WHERE cinema.Users.UserName = user_name)
      WHERE cinema.Users.UserName = user_name;

    -- cach an exception that might occur during the transaction
    EXCEPTION
      WHEN OTHERS THEN
        error_code := 414;
    END;
    -- end TRANSACTION mode

  END IF;

END IF;

-- build the output result
IF error_code = 414 or error_code = 401 THEN
  text_output = error_code;
ELSE
  -- tickets canceled sucessfully
  error_code := 214;
  text_output = text_output || error_code ;
```

```
  END IF;

  RETURN text_output;
END;
'language␣plpgsql;
-- ============================================================


-- # ============================================================================

-- #
-- # STEP 9 -- CANCEL ALL UNPAID TICKETS RESERVED BY USING THE PAY AT
    THE CINEMA
-- #          PAYMENT METHO, TICKETS THAT HAVE NOT BEEN PURCHASED YET
-- #                           (BACKGROUND THREAD)
-- #
-- # ----------------------------------------------------------------------------

-- #                  Cancel_Unpaid_Tickets_Before_Show Procedure
-- #
-- # Cancel all reserved tickets by using the Pay at the Cinema method
-- # that have not been purchased.
-- # This is to be done by a background thread on the server side within
     30 minutes
-- # before the show
-- #   - It returns an error_code i.e.
-- #         415 = error,
-- #         215 = all unpaid tickets for the corresponding shows are
    canceled
-- # ============================================================================


SET search_path TO cinema, public;
SET DATESTYLE TO ISO;

CREATE OR REPLACE FUNCTION Cancel_Unpaid_Tickets_Before_Show() returns
    TEXT AS '
DECLARE

   -- Declare a variable to the error code
   error_code INTEGER;

   -- Declare the current timestamp value
   date_value TIMESTAMP;

BEGIN
  SET search_path TO cinema, public;
  SET DATESTYLE TO ISO;
```

```
  date_value = ''now'';

  -- begin TRANSACTION mode
  BEGIN
    -- delete all tickets with the CINEMA payment method
    -- that have not been purchased within 30 minutes before the shows
    DELETE
    FROM cinema.BookedSeats
    WHERE cinema.BookedSeats.BookedSeatID IN
       (SELECT Tickets.BookedSeatID
        FROM cinema.Tickets
        WHERE cinema.Tickets.TicketID IN
        (SELECT DISTINCT tickets.ticketid
         FROM cinema.tickets, cinema.reservations, cinema.paymentmethod,
             cinema.showdiscount,
                   cinema.showtime, cinema.DateOfShow , cinema.HourOfShow
         WHERE cinema.tickets.paid                      = ''N''
         AND cinema.Tickets.ResID                       = cinema.
             Reservations.ResID
         AND cinema.PaymentMethod.PaymentMethodID       = cinema.
             Reservations.PaymentMethodID
         AND UPPER(cinema.PaymentMethod.PaymentMethodType) = ''CINEMA''
         AND cinema.tickets.showlocationid              = cinema.
             showdiscount.showlocationid
         AND cinema.tickets.showtimeid                  = cinema.
             showdiscount.showtimeid
         AND cinema.showdiscount.showtimeid             = cinema.showtime.
             showtimeid
         AND cinema.showtime.showtimeid                 = cinema.hourofshow
             .hourshowid
         AND cinema.showtime.dateshowid                 = cinema.DateOfShow
             .dateshowid
         AND cinema.DateOfShow.DateOfShow               = to_date(to_char(
             date_value + interval ''30 minutes'', ''YYYY.MM.DD''), ''
             YYYY.MM.DD'')
         AND to_char((select hourofshow.hourofshow from cinema.hourofshow
              where cinema.hourofshow.hourshowid = cinema.showdiscount.
             showtimeid ), ''HH24:MI:SS'') <= to_char(date_value +
             interval ''30 minutes'', ''HH24:MI:SS'')
)
      );

    -- Check if there is a reservation with 0 tickets and in the same
        time
    --   the hour and date of the show for that reservation is within 30
        min
```

```
    --   before curent hour and data, and the payment method is CINEMA.
    -- Delete all reservations that meet those criterion

    DELETE
    FROM cinema.Reservations
    WHERE cinema. Reservations.ResID NOT IN
      (SELECT DISTINCT Tickets.ResID
       FROM  cinema.Tickets)
    AND  cinema.Reservations.ShowTimeID IN
      (SELECT distinct ShowTime.ShowTimeID
       FROM  cinema.ShowTime, cinema.DateOfShow, cinema.HourOfShow
       WHERE cinema.showtime.showtimeid  = cinema.hourofshow.hourshowid
       AND   cinema.showtime.dateshowid  = cinema.DateOfShow.dateshowid
       AND   cinema.DateOfShow.DateOfShow = to_date(to_char(date_value +
            interval ''30 minutes'', ''YYYY.MM.DD''), ''YYYY.MM.DD'')
        AND  to_char((select hourofshow.hourofshow from cinema.
             hourofshow where cinema.hourofshow.hourshowid = cinema.
             ShowTime.showtimeid ), ''HH24:MI:SS'') <= to_char(
             date_value + interval ''30 minutes'', ''HH24:MI:SS'')
      );

    error_code := 215;
  -- catch an exception that might occur during the transaction
  EXCEPTION
    WHEN OTHERS THEN
      error_code := 415;
    WHEN
  END;
  -- end TRANSACTION mode

  RETURN error_code;
END;
'language␣plpgsql;
-- ============================================================================================================



-- # ===================================================================

-- #                    STEP 2 ---- CHANGE PASSWORD
-- # -------------------------------------------------------------------

-- #                     Change Password Procedure
-- #
-- # It accepts 3 input arguments: Username, Old_Password, New_password
-- # It returns an error code i.e.
-- #    402 = user not authenticated
-- #    202 = password changed
```

```
-- # ------------------------------------------------------------------

-- # The followong format is to be used to call the function:
-- # SELECT * FROM Change_Password(User Name, Old Password, New Password)
    ;
-- # e.g.
-- # SET search_path TO cinema, public;
-- # SET DATESTYLE TO ISO;
-- # SELECT * FROM Change_Password('adm', '12345678', 'aaa');
-- # ==================================================================


SET search_path TO cinema, public;
SET DATESTYLE TO ISO;

CREATE FUNCTION Change_Password (TEXT, TEXT, TEXT) RETURNS TEXT AS '

  DECLARE
    -- Declare aliases for user input.
    user_name   ALIAS FOR $1;
    old_password ALIAS FOR $2;
    new_password ALIAS FOR $3;

    -- Declare a variable to hold the return error.
    return_error TEXT;

  BEGIN

    SET search_path TO cinema, public;
    SET DATESTYLE TO ISO;

    SELECT INTO return_error username
    FROM  cinema.Users
    WHERE cinema.Users.UserName = user_name
    AND   cinema.Users.Password = old_password;

    IF NOT FOUND THEN
      RETURN 402;
    ELSE
      UPDATE cinema.Users
      SET   Password = new_password
      WHERE UserName = user_name
      AND   Password = old_password;
      RETURN 202;
    END IF;

  END;
```

```
'␣LANGUAGE␣'plpgsql';
-- ================================================


-- # ============================================================================

-- #
-- #  STEP 5 --- ACCEPT THE PRICE FOR THE CURRENT RESERVATION
-- #             AND PAY FOR THE TICKET ONLY IF PAYMENT METHOD IS CREDIT
    CARD.
-- #             NO MATTER OF THE PAYMENT METHOD MAKE THE RESERVATION AND
-- #             TICKET INFORMATION PERSISTENT TO THE DB
-- #
-- # (User presses the PURCHASE button on the TICKET PAYMENT or SECURE
    WALLET forms)
-- # ----------------------------------------------------------------------------

-- #                           Date Type Creation
-- #
-- # Create the data type that will store the discount id and discount
    value
-- # ============================================================================


SET search_path TO cinema, public;
SET DATESTYLE TO ISO;

CREATE TYPE ShowDiscountID AS
   (DiscountSchemaID INTEGER,
    DiscountValue  NUMERIC(2,1)
   );
-- ============================================================================================



-- # ============================================================================

-- #
-- #  STEP 5 --- ACCEPT THE PRICE FOR THE CURRENT RESERVATION
-- #             AND PAY FOR THE TICKET ONLY IF PAYMENT METHOD IS CREDIT
    CARD.
-- #             NO MATTER OF THE PAYMENT METHOD MAKE THE RESERVATION AND
-- #             TICKET INFORMATION PERSISTENT TO THE DB
-- #
-- # (User presses the PURCHASE button on the TICKET PAYMENT or SECURE
    WALLET forms)
-- # ----------------------------------------------------------------------------
```

```
-- #                  Compute_Price_And_Maybe_Pay Procedure
-- #
-- # Check if the user is authenticated and in that case computes the
    price
-- # for each ticket and the final price to be paid.
-- # In case the payment method is CARD then pay the tickets. Else, do
    not pay!
-- # Save user's␣Ticket␣ID's and reservation ID in the DB
-- #   - It accepts as input 7 arguments i.e. UserName, OTP,
    ShowLocationID,
-- #         ShowTimeID, an array of [SelectedRowNo, SelectSeatNo],
-- #         an array of [DiscountTypeS], the reservatioID,
-- #         an array of [TicketIDs], the reservation_date,
-- #         the payment_method, and isCreditCardValid
-- #
-- #         isCreditCardValid = 1 - credit card IS valid
-- #         isCreditCardValid = 2 - credit card IS NOT valid
-- #
-- #   - It returns a list containing: RESERVATION_ID,
    TOTAL_PRICE_TO_BE_PAID,
-- #         LEFT_E_MONEY, TICKET_ID',␣and␣PRICE␣for␣each␣TICKET_ID.
-- #   - It also returns an error_code i.e.
-- #          212= price computed successfully and reservation saved
-- #          201= user authenticated
-- #          401= user not authenticated,
-- #          412= transaction error,
-- #          413= invalid credit card
-- # ============================================================================


SET search_path TO cinema, public;
SET DATESTYLE TO ISO;

CREATE OR REPLACE FUNCTION Compute_Price_And_Maybe_Pay(TEXT, TEXT,
    INTEGER, INTEGER, INTEGER[], TEXT[], TEXT, TEXT[], DATE, TEXT, TEXT)
     returns text AS '
DECLARE

    -- Declare aliases for user input.
    user_name       ALIAS FOR $1;
    otp             ALIAS FOR $2;
    show_location_id ALIAS FOR $3;
    show_time_id    ALIAS FOR $4;
    row_seat        ALIAS FOR $5;
    discount_types  ALIAS FOR $6;
    reservation_id  ALIAS FOR $7;
    ticket_ids      ALIAS FOR $8;
```

```
res_date        ALIAS FOR $9;
payment_method  ALIAS FOR $10;
valid_credit_card ALIAS FOR $11;

-- Declare a variable to hold the authentication result
-- i.e. 201 for user authenticated, 401 else
error_code            INTEGER;

-- Dimension of the array of seat_row array
dimension_row_seat    INTEGER;

-- Dimension of the array of discount types array
dimension_discount_types INTEGER;

-- base ticket price for the given show
base_price            float;

-- discount values for each ticket for the given show
discount_value        float[];

-- calculated prices for each ticket
calculated_price      float[];

-- the amount of e_money for the given user
emoney                float;

-- the total price to be paid for all tickets
price_to_be_paid      float;

-- discount IDs for each ticket for the given show
discount_ids          INTEGER[];

-- the payment method_id coresponding to the
-- given payment method given by the user
payment_method_id     INTEGER;

-- current booked seat ids
booked_seat_ids       INTEGER[];

-- Declare a variable to hold the output result
text_output TEXT = '''';

-- declare variable to hold a boolean value
-- indicating if a ticket is paid or not
paid_bool boolean = ''Y'';

-- Declare a variable to hold rows of the Prices,
```

```
    -- Discount types, respectively
    row_data_price      cinema.Prices%ROWTYPE;
    row_data_discount   cinema.ShowDiscountValue%ROWTYPE;
    row_data_discount_id cinema.ShowDiscountID%ROWTYPE;
    row_data_booked_seat cinema.BookedSeats%ROWTYPE;

BEGIN
  SET search_path TO cinema, public;
  SET DATESTYLE TO ISO;

  --  if credit card is not valid return an error message
  IF valid_credit_card = ''2'' THEN
    RETURN ''413'';
  END IF;

  -- get the number of elements in each of the input arrays
  dimension_row_seat     := ARRAY_UPPER(row_seat, 1);
  dimension_discount_types := ARRAY_UPPER(discount_types, 1);

  -- if the payment mehtod is credit card then use the
  -- ticket_paid = true when inserting the PAID value
  -- in the ticket table. Else, use false.
  IF UPPER(payment_method) = ''CARD'' THEN
    paid_bool := ''Y'';
  ELSE
    paid_bool := ''N'';
  END IF;

  -- verify if user credentials are correct
  error_code := Authenticate(user_name, otp);

       -- if user is authenticated
  IF error_code = 201 THEN

    -- get the base price for the given show
    FOR row_data_price IN
      SELECT Prices.PriceID, Prices.BasePrice
      FROM   cinema.Shows, cinema.Prices
      WHERE  cinema.Shows.ShowLocationID = show_location_id
      AND    cinema.Shows.ShowTimeID  = show_time_id
      AND    cinema.Shows.PriceID     = cinema.Prices.PriceID
    LOOP
      -- set the base_price variable
      base_price := row_data_price.BasePrice;
    END LOOP;
```

```
-- get the discount values for each of the tickets
FOR i IN 1..dimension_discount_types LOOP
  FOR row_data_discount IN
    SELECT DiscountSchema.DiscountValue
    FROM  cinema.ShowDiscount, cinema.DiscountSchema
    WHERE cinema.ShowDiscount.ShowLocationID   =  show_location_id
    AND   cinema.ShowDiscount.ShowTimeID       =  show_time_id
    AND   cinema.ShowDiscount.DiscountSchemaID =  cinema.
        DiscountSchema.DiscountSchemaID
    AND   cinema.DiscountSchema.DiscountType LIKE UPPER (
        discount_types[i])
  LOOP
    -- set the discount value for each ticket
    discount_value[i] = row_data_discount.DiscountValue;
    -- text_output = text_output || ''|DiscValue: '' ||
        discount_value[i] ||
    --       '', DiscType: '' || discount_types[i];
  END LOOP;
END LOOP;


-- get the discount ids for each of the tickets
FOR i IN 1..dimension_discount_types LOOP
  FOR row_data_discount_id IN
    SELECT DiscountSchema.DiscountSchemaID
    FROM  cinema.ShowDiscount, cinema.DiscountSchema
    WHERE cinema.ShowDiscount.ShowLocationID = show_location_id
    AND   cinema.ShowDiscount.ShowTimeID    = show_time_id
    AND   cinema.ShowDiscount.DiscountSchemaID = cinema.
        DiscountSchema.DiscountSchemaID
    AND   cinema.DiscountSchema.DiscountType LIKE UPPER (
        discount_types[i])
  LOOP
    -- set the discount value for each ticket
    discount_ids[i] = row_data_discount_id.DiscountSchemaID;
    -- text_output = text_output || ''|DiscID: '' || discount_ids[i
        ] ||
    --       '', DiscType: '' ||discount_types[i];
  END LOOP;
END LOOP;


-- get the payment method id for the gievn payment method type
SELECT INTO payment_method_id PaymentMethodID
FROM  cinema.PaymentMethod
WHERE cinema.PaymentMethod.PaymentMethodType LIKE UPPER(
    payment_method);
```

```
 --text_output = text_output || ''|PaymentMethod: ''||
     payment_method_id;



-- get the booked_seat_ids for the current booked tickets
--  saved as temp in the booked seats table;
-- conected to the current show ids for each of the tickets
-- and having the given row and seat values
FOR i IN 1..dimension_row_seat LOOP
  FOR row_data_booked_seat IN
    SELECT BookedSeats.BookedSeatID
    FROM   cinema.BookedSeats
    WHERE cinema.BookedSeats.ShowLocationID = show_location_id
    AND   cinema.BookedSeats.ShowTimeID  = show_time_id
    AND   cinema.BookedSeats.TEMP        = ''Y''
    AND   cinema.BookedSeats.RowNo       = row_seat [i][1]
    AND   cinema.BookedSeats.SeatNo      = row_seat [i][2]

  LOOP
    -- set the discount value for each ticket
    booked_seat_ids[i] = row_data_booked_seat.BookedSeatID;
    -- text_output = text_output || ''|SeatID: '' ||booked_seat_ids[i
        ];
  END LOOP;
END LOOP;


-- calculate the price for each ticket
FOR i IN 1..dimension_discount_types LOOP
  calculated_price[i] = base_price - base_price * discount_value[i];
  -- text_output = text_output || ''|TicketPrice: ''||
      calculated_price[i];
END LOOP;


-- Only in case user selected the CREDIT CARD payment method
-- finds the amount of e-money for the authenticated user
-- IF UPPER(payment_method) = ''CARD'' THEN
  SELECT INTO emoney Users.E_money
  FROM   cinema.Users
  WHERE cinema.Users.UserName = user_name
  AND   cinema.Users.OTP     = otp;
  -- text_output = text_output || ''|E_MONEY: '' ||emoney;
-- END IF;

-- calculates the price to be paid for all tickets
```

```
price_to_be_paid := 0;
FOR i IN 1..dimension_discount_types LOOP
  price_to_be_paid = price_to_be_paid + calculated_price[i];
END LOOP;


-- Only in case user selected the CREDIT CARD payment method
-- use the amount of e-money for the payment
IF UPPER(payment_method) = ''CARD'' THEN
  IF emoney >= price_to_be_paid THEN
    emoney = emoney - price_to_be_paid;
    price_to_be_paid := 0;
  ELSE
    price_to_be_paid = price_to_be_paid - emoney;
    emoney := 0;

  END IF;
END IF;
-- text_output = text_output || ''|PRICE_TO_BE_PAID '' ||
    price_to_be_paid;
-- text_output = text_output || ''|left_emoney '' ||emoney;


-- begin TRANSACTION mode
BEGIN

  -- Only in case user selected the CREDIT CARD payment method
  -- update the value of e_money in the users table
  IF UPPER(payment_method) = ''CARD'' THEN
    UPDATE cinema.Users
    SET   e_money  = emoney
    WHERE cinema.Users.UserName = user_name
    AND   cinema.Users.OTP     = otp;
    -- text_output = text_output || ''|Total price: ''||
        price_to_be_paid || ''|Emoney: ''||emoney;
  END IF;

  -- insert the ticket info and reservation info into the db for the
      given user and show
  INSERT INTO cinema.Reservations(ResID, ResDate, TotalPriceToBePaid
      , PaymentMethodID, UserName, ShowLocationID, ShowTimeID)
  VALUES (reservation_id, res_date, price_to_be_paid,
      payment_method_id, user_name, show_location_id, show_time_id);

  FOR i IN 1..dimension_discount_types LOOP
    INSERT INTO cinema.Tickets(TicketID, FinalPrice, Paid,
        ShowLocationID, ShowTimeID, DiscountSchemaID, ResID,
```

```
          BookedSeatID)
      VALUES (ticket_ids[i], calculated_price[i], paid_bool,
          show_location_id, show_time_id, discount_ids[i],
          reservation_id, booked_seat_ids[i]);
    END LOOP;

    FOR i IN 1..dimension_row_seat LOOP
        UPDATE cinema.BookedSeats SET Temp = ''N'', ExpDate = NULL
        WHERE cinema.BookedSeats.ShowLocationID = show_location_id
        AND   cinema.BookedSeats.ShowTimeID  = show_time_id
        AND   cinema.BookedSeats.TEMP        = ''Y''
        AND   cinema.BookedSeats.RowNo       = row_seat [i][1]
        AND   cinema.BookedSeats.SeatNo      = row_seat [i][2];
    END LOOP;

  -- catch any exception might occur
  EXCEPTION
    WHEN OTHERS THEN
      error_code := 412;
  END;
  -- end TRANSACTION mode

END IF;

     -- build the output result
IF error_code = 412 OR error_code = 401 THEN
  text_output := error_code || text_output;

ELSE
  -- everything went OK
  error_code = 212;
  -- add the emoney ammount only in case of CARD payment method
  IF UPPER(payment_method) = ''CARD'' THEN
    text_output = text_output || error_code  || ''\n''||
              reservation_id || ''\n''||
              price_to_be_paid || ''|''||
              emoney;
  ELSE
    text_output = text_output || error_code  || ''\n''||
              reservation_id || ''\n''||
              price_to_be_paid || ''|''||
              emoney;
  END IF;
  FOR i IN 1..dimension_row_seat LOOP
    text_output = text_output ||''\n'' || ticket_ids[i]
           || ''|'' || calculated_price[i] ;
  END LOOP;
```

```
  END IF;

  RETURN text_output;
END;
'language␣plpgsql;
-- ================================================================================


-- # ============================================================================================================

-- #   CREATE THE TABLES IN THE CINEMA SCHEMA, THE CONNECTIONS AMONG THE
     TABLES, CONSTRAINTS AND INDECES
-- # ============================================================================================================



-- # ========================================================================

-- #  Drop the CINEMA schema if exists. Create the CINEMA schema
    afterwards
-- #  Set the current searching path to the CINEMA and PUBLIC schemata
-- #  and the date format to DDMMYYYY
-- # ========================================================================

--DROP  SCHEMA cinema CASCADE;
CREATE SCHEMA cinema AUTHORIZATION zeratul;

SET search_path TO cinema, public;
SET DATESTYLE TO German;




-- # =========================
-- #      Cinemas table
-- # =========================
CREATE TABLE Cinemas (
  CinemaID   SERIAL,
  CinemaName VARCHAR(30),
  Street     VARCHAR(30),
  City       VARCHAR(25),
  Zip        VARCHAR(8),

   -- KEY     CONSTRAINT DEFINITIONS
   CONSTRAINT Cinemas_PK_CinemaID PRIMARY KEY(CinemaID),

   -- NOT NULL CONSTRAINT DEFINITIONS
```

```
    CONSTRAINT Cinemas_NOT_NULL_CinemaName CHECK (CinemaName IS NOT NULL)
        ,
    CONSTRAINT Cinemas_NOT_NULL_Street  CHECK (Street    IS NOT NULL),
    CONSTRAINT Cinemas_NOT_NULL_City    CHECK (City      IS NOT NULL),
    CONSTRAINT Cinemas_NOT_NULL_Zip     CHECK (Zip       IS NOT NULL)
);

    -- INDEX DEFINITIONS
    CREATE INDEX Cinemas_IDX_CinameName      ON Cinemas (upper(CinemaName
        ));
    CREATE INDEX Cinemas_IDX_Street          ON Cinemas (upper(Street));
    CREATE INDEX Cinemas_IDX_City            ON Cinemas (upper(City));
    CREATE INDEX Cinemas_IDX_CinNam_Str_City_Zip ON Cinemas (upper(
        CinemaName), upper(Street), upper(City), upper(Zip));
    CREATE INDEX Cinemas_IDX_Str_City_Zip    ON Cinemas (upper(Street),
        upper(City), upper(Zip));
    CREATE INDEX Cinemas_IDX_City_Zip        ON Cinemas (upper(City),
        upper(Zip));
    CREATE INDEX Cinemas_IDX_CinNam_City_Zip ON Cinemas (upper(CinemaName
        ), upper(City), upper(Zip));
-- ============================




-- # =========================
-- #      CinemaHalls table
-- # =========================
CREATE TABLE CinemaHalls (
  CinemaID SERIAL,
  HallID   VARCHAR(20),
  Rows     NUMERIC(2, 0),
  Cols     NUMERIC(2, 0),

    -- KEY     CONSTRAINT DEFINITIONS
    CONSTRAINT CinemaHalls_PK_CinemaID_HallID PRIMARY KEY (CinemaID,
        HallID),

    CONSTRAINT CinemaHalls_FK_HallID       FOREIGN KEY (CinemaID)
        REFERENCES Cinemas (CinemaID)
                ON DELETE CASCADE
                ON UPDATE CASCADE,

    -- CHECK   CONSTRAINT DEFINITIONS
    CONSTRAINT CinemaHalls_CHECK_Rows  CHECK (Rows > 0),
    CONSTRAINT CinemaHalls_CHECK_Cols  CHECK (Cols > 0),
```

```
    --  NOT NULL CONSTRAINT DEFINITIONS
    CONSTRAINT CinemaHalls_NOT_NULL_Rows CHECK (Rows IS NOT NULL),
    CONSTRAINT CinemaHalls_NOT_NULL_Cols CHECK (Cols IS NOT NULL)
);

    --  INDEX DEFINITIONS
    CREATE INDEX CinemaHalls_IDX_CinID_HallID ON CinemaHalls (CinemaID,
        HallID);
-- ============================




-- # =========================
-- #     Movies table
-- # =========================
CREATE TABLE Movies (
  MovieID             SERIAL,
  MovieName           VARCHAR(30),
  Duration            NUMERIC(3, 0),
  Genre               VARCHAR(20),
  ParentClassification VARCHAR(6),
  Language            VARCHAR(15)  DEFAULT 'ENGLISH',
  Year                NUMERIC(4, 0) DEFAULT to_number(to_char(now(), '
      YYYY'), '9999'),
  MovieCountry        VARCHAR(30)  DEFAULT 'USA',
  Poster              BYTEA,
  Description         TEXT,
  ProducingStudios    VARCHAR(200),
  Director            VARCHAR(30),
  Actors              VARCHAR(200),

    --  KEY     CONSTRAINT DEFINITIONS
    CONSTRAINT Movies_PK_MovieID PRIMARY KEY (MovieID),

    --  CHECK   CONSTRAINT DEFINITIONS
    CONSTRAINT Movies_CHECK_Duration         CHECK (Duration > 0),
    CONSTRAINT Movies_CHECK_ParentClassification CHECK (
        ParentClassification IN ('G', 'PG', 'M', 'MA␣15+', 'R␣18+', 'X
        ␣18+')),
    CONSTRAINT Movies_CHECK_Year             CHECK ((Year > 1900) AND (
        Year <= to_number(to_char(now(), 'YYYY'), '9999'))),

    --  NOT NULL CONSTRAINT DEFINITIONS
    CONSTRAINT Movies_NOT_NULL_MovieName         CHECK (MovieName
                IS NOT NULL),
    CONSTRAINT Movies_NOT_NULL_Duration          CHECK (Duration
```

```
                IS NOT NULL),
    CONSTRAINT Movies_NOT_NULL_Genre             CHECK (Genre
                    IS NOT NULL),
    CONSTRAINT Movies_NOT_NULL_ParentClassification CHECK (
        ParentClassification IS NOT NULL),
    CONSTRAINT Movies_NOT_NULL_Language          CHECK (Language
                IS NOT NULL),
    CONSTRAINT Movies_NOT_NULL_Year              CHECK (Year
                     IS NOT NULL),
    CONSTRAINT Movies_NOT_NULL_Contry            CHECK (MovieCountry
              IS NOT NULL),
    CONSTRAINT Movies_NOT_NULL_Description       CHECK (Description
               IS NOT NULL),
    CONSTRAINT Movies_NOT_NULL_ProducingStudios  CHECK (ProducingStudios
         IS NOT NULL),
    CONSTRAINT Movies_NOT_NULL_Director          CHECK (Director
                IS NOT NULL),
    CONSTRAINT Movies_NOT_NULL_Actors            CHECK (Actors
                   IS NOT NULL)
);

    --  INDEX DEFINITIONS
    CREATE INDEX Movies_IDX_MovieName ON Movies (upper(MovieName));
-- ============================


-- # ================================
-- #       ShowLocation table
-- # ================================
CREATE TABLE ShowLocation (
  ShowLocationID SERIAL,
  CinemaID      SERIAL,
  HallID        VARCHAR(20),
  MovieID       SERIAL,

    --  KEY CONSTRAINT DEFINITIONS
    CONSTRAINT ShowLocation_PK_ShowID       PRIMARY KEY (ShowLocationID),

    CONSTRAINT ShowLocation_FK_CinemaID_HallID FOREIGN KEY (CinemaID,
        HallID) REFERENCES CinemaHalls (CinemaID, HallID)
                              ON DELETE CASCADE
                                ON UPDATE CASCADE,

    CONSTRAINT ShowLocation_FK_MovieID      FOREIGN KEY (MovieID)
        REFERENCES Movies     (MovieID)
```

```
                              ON DELETE CASCADE
                              ON UPDATE CASCADE
);

    --  INDEX DEFINITIONS
    CREATE INDEX ShowLocation_IDX_CinemaID    ON ShowLocation (CinemaID);
    CREATE INDEX ShowLocation_IDX_HallID      ON ShowLocation (HallID);
    CREATE INDEX ShowLocation_IDX_CinemaID_HallID ON ShowLocation (
        CinemaID, HallID);
    CREATE INDEX ShowLocation_IDX_MovieID     ON ShowLocation (MovieID);
    CREATE INDEX ShowLocation_IDX_All_FKs     ON ShowLocation (CinemaID,
        HallID, MovieID);
-- ============================




-- # =========================
-- #     DateOfShow table
-- # =========================
CREATE TABLE DateOfShow (
  DateShowID SERIAL,
  DateOfShow DATE DEFAULT current_date,

    --  KEY     CONSTRAINT DEFINITIONS
    CONSTRAINT DateOfShow_PK_DateShowID PRIMARY KEY (DateShowID),

    --  CHECK   CONSTRAINT DEFINITIONS
    CONSTRAINT DateOfShow_CHECK_DateOfShow CHECK (DateOfShow >= now()),

    --  NOT NULL CONSTRAINT DEFINITIONS
    CONSTRAINT DateOfShow_NOT_NULL_DateOfShow CHECK (DateOfShow IS NOT
        NULL)
);

    --  INDEX DEFINITIONS
    CREATE INDEX DateOfShow_IDX_DateOfShow ON DateOfShow (DateOfShow);
-- ============================



-- # =========================
-- #     HourOfShow table
-- # =========================
CREATE TABLE HourOfShow (
  HourShowID SERIAL,
```

```
  HourOfShow TIME,

    -- KEY     CONSTRAINT DEFINITIONS
    CONSTRAINT HourOfShow_PK_HourShowID PRIMARY KEY (HourShowID),

    -- NOT NULL CONSTRAINT DEFINITIONS
    CONSTRAINT HourOfShow_NOT_NULL_HourOfShow CHECK (HourOfShow IS NOT
        NULL)
);

    -- INDEX DEFINITIONS
    CREATE INDEX HourOfShow_IDX_HourOfShow ON HourOfShow (HourOfShow);
-- =============================




-- # =========================
-- #      ShowTime table
-- # =========================
CREATE TABLE ShowTime (
  ShowTimeID  SERIAL,
  DateShowID  SERIAL,
  HourShowID  SERIAL,

    -- KEY CONSTRAINT DEFINITIONS
    CONSTRAINT ShowTime_PK_ShowID   PRIMARY KEY (ShowTimeID),

    CONSTRAINT ShowTime_FK_DateShowID FOREIGN KEY (DateShowID) REFERENCES
         DateOfShow (DateShowID)
                    ON DELETE SET NULL
                  ON UPDATE CASCADE,

    CONSTRAINT ShowTime_FK_HourShowID FOREIGN KEY (HourShowID) REFERENCES
         HourOfShow (HourShowID)
                  ON DELETE SET NULL
                    ON UPDATE CASCADE
);

    -- INDEX DEFINITIONS
    CREATE INDEX ShowTime_IDX_DateShowID   ON ShowTime (DateShowID);
    CREATE INDEX ShowTime_IDX_HourShowID   ON ShowTime (HourShowID);
    CREATE INDEX ShowTime_IDX_All_FKs      ON ShowTime (DateShowID,
        HourShowID);
-- =============================
```

```
-- # ==========================
-- #      Prices table
-- # ==========================
CREATE TABLE Prices(
  PriceID   SERIAL,
  BasePrice NUMERIC(10,2),

    -- KEY  CONSTRAINT DEFINITIONS
    CONSTRAINT Prices_PK_PriceID PRIMARY KEY (PriceID),

    -- CHECK CONSTRAINT DEFINITIONS
    CONSTRAINT Prices_CHECK_BasePrice CHECK (BasePrice >= 0.0),

    -- NOT NULL CONSTRAINT DEFINITIONS
    CONSTRAINT Prices_NOT_NULL_BasePrice CHECK (BasePrice IS NOT NULL)
);
-- ============================
```

```
-- # ==========================
-- #       Shows table
-- # ==========================
CREATE TABLE Shows (
  ShowLocationID  SERIAL,
  ShowTimeID      SERIAL,
  PriceID         SERIAL,

    -- KEY CONSTRAINT DEFINITIONS
    CONSTRAINT Shows_PK_ShowID      PRIMARY KEY (ShowLocationID,
        ShowTimeID),

    CONSTRAINT Shows_FK_ShowLocationID FOREIGN KEY (ShowLocationID)
        REFERENCES ShowLocation (ShowLocationID)
                       ON DELETE CASCADE
                         ON UPDATE CASCADE,

    CONSTRAINT Shows_FK_ShowTimeID  FOREIGN KEY (ShowTimeID)
        REFERENCES ShowTime    (ShowTimeID)
                       ON DELETE CASCADE
                       ON UPDATE CASCADE,
```

```
    CONSTRAINT Shows_FK_PriceID     FOREIGN KEY (PriceID)
        REFERENCES Prices       (PriceID)
                        ON DELETE SET NULL
                       ON UPDATE CASCADE
);

    -- INDEX DEFINITIONS
    CREATE INDEX Shows_IDX_PK    ON Shows (ShowLocationID, ShowTimeID);
    CREATE INDEX Shows_IDX_PriceID ON Shows (PriceID);
    CREATE INDEX Shows_IDX_All   ON Shows (ShowLocationID, ShowTimeID,
        PriceID);
-- ============================




-- # =========================
-- #    DiscountSchema table
-- # =========================
CREATE TABLE DiscountSchema(
  DiscountSchemaID SERIAL,
  DiscountType    VARCHAR(10)   DEFAULT 'NONE',
  DiscountValue   NUMERIC(2, 1) DEFAULT 0.0,

    -- KEY     CONSTRAINT DEFINITIONS
    CONSTRAINT DiscountSchema_PK_DiscountSchemaID PRIMARY KEY (
        DiscountSchemaID),

    -- NOT NULL CONSTRAINT DEFINITIONS
    CONSTRAINT DiscountSchemaID_NOT_NULL_DiscountType CHECK (DiscountType
         IS NOT NULL),
    CONSTRAINT DiscountSchemaID_NOT_NULL_DiscountValue CHECK (
        DiscountValue IS NOT NULL),

    -- CHECK   CONSTRAINT DEFINITIONS
    CONSTRAINT DiscountSchemaID_CHECK_DiscountType CHECK (DiscountType IN
         ('CHILD', 'STUDENT', 'PENSIONER', 'VOUCHER', 'NONE')),
    CONSTRAINT DiscountSchemaID_CHECK_DiscountValue CHECK ((DiscountValue
         >= 0.0) AND (DiscountValue <= 1.0))
);

    -- INDEX DEFINITIONS
    CREATE INDEX DiscountSchema_IDX_DiscountType ON DiscountSchema (upper
        (DiscountType));
-- ============================
```

```
-- # =========================
-- #    ShowDiscount table
-- # =========================
CREATE TABLE ShowDiscount(
  ShowLocationID  SERIAL,
  ShowTimeID      SERIAL,
  DiscountSchemaID SERIAL,

    -- KEY CONSTRAINT DEFINITIONS
    CONSTRAINT ShowDiscount_PK_ShowLoc_ShowTime_DiscID PRIMARY KEY (
        ShowLocationID, ShowTimeID, DiscountSchemaID),

    CONSTRAINT ShowDiscount_FK_Shows              FOREIGN KEY (
        ShowLocationID, ShowTimeID) REFERENCES Shows   (ShowLocationID,
        ShowTimeID)
                              ON DELETE CASCADE
                              ON UPDATE CASCADE,

    CONSTRAINT ShowDiscount_FK_DiscountSchemaID   FOREIGN KEY (
        DiscountSchemaID)       REFERENCES DiscountSchema (
        DiscountSchemaID)
                              ON DELETE CASCADE
                                ON UPDATE CASCADE
);

    -- INDEX DEFINITIONS
    CREATE INDEX ShowDiscount_IDX_PK2 ON Shows   (ShowLocationID,
        ShowTimeID);
    CREATE INDEX ShowDiscount_IDX_PK3 ON ShowDiscount (ShowLocationID,
        ShowTimeID, DiscountSchemaID);
-- ============================
```

```
-- # =========================
-- #       Users table
-- # =========================
CREATE TABLE Users (
  UserName   VARCHAR(30),
  Name       VARCHAR(30),
```

```
  Password   TEXT,
  UserID     TEXT,
  RandomID   TEXT,
  OTP        TEXT,
  OTPExpDate TIMESTAMP,
  E_Money    NUMERIC(10, 2) DEFAULT 0.0,

    -- KEY     CONSTRAINT DEFINITIONS
    CONSTRAINT Users_PK_UserName PRIMARY KEY (UserName),

    -- NOT NULL CONSTRAINT DEFINITIONS
    CONSTRAINT Users_NOT_NULL_Password CHECK (Password IS NOT NULL),
    CONSTRAINT Users_NOT_NULL_E_Money CHECK (E_Money IS NOT NULL),

    -- CHECK   CONSTRAINT DEFINITIONS
    CONSTRAINT Users_CHECK_OTPExpDate CHECK (OTPExpDate >= now()),
    CONSTRAINT Users_CHECK_E_Money  CHECK (E_Money  >= 0.0)
);

    -- INDEX DEFINITIONS
    CREATE INDEX Users_IDX_Password ON Users (Password);
    CREATE INDEX Users_IDX_OTP      ON Users (OTP);
    CREATE INDEX Users_IDX_OTPExpDate ON Users (OTPExpDate);
    CREATE INDEX Users_IDX_Credentials ON Users (UserName, Password);
    CREATE INDEX Users_IDX_OTP_OTPExp ON Users (OTP, OTPExpDate);

-- ============================




-- # =========================
-- #       Rating table
-- # =========================
CREATE TABLE Rating (
  UserName   VARCHAR(30),
  MovieID    SERIAL,
  UserRating NUMERIC(3, 0),

    -- KEY     CONSTRAINT DEFINITIONS
    CONSTRAINT Rating_PK_UserName_MovieID PRIMARY KEY (UserName, MovieID)
        ,

    CONSTRAINT Rating_FK_UserName       FOREIGN KEY (UserName) REFERENCES
         Users (UserName)
                    ON DELETE CASCADE
                    ON UPDATE CASCADE,
```

```
    CONSTRAINT Rating_FK_MovieID        FOREIGN KEY (MovieID) REFERENCES
        Movies (MovieID)
                    ON DELETE CASCADE
                    ON UPDATE CASCADE,

    -- CHECK   CONSTRAINT DEFINITIONS
    CONSTRAINT Rating_CHECK_UserRating CHECK (UserRating >= 0.0),

    -- NOT NULL CONSTRAINT DEFINITIONS
    CONSTRAINT Rating_NOT_NULL_UserRating CHECK (UserRating IS NOT NULL)
);

    -- INDEX DEFINITIONS
    CREATE INDEX Rating_IDX_PKs ON Rating (UserName, MovieID);
-- ============================



-- # =========================
-- #    PaymentMethod table
-- # =========================
CREATE TABLE PaymentMethod(
  PaymentMethodID  SERIAL,
  PaymentMethodType VARCHAR(7),

    -- KEY     CONSTRAINT DEFINITIONS
    CONSTRAINT PaymentMethod_PK_PaymentMethodID PRIMARY KEY (
        PaymentMethodID),

    -- NOT NULL CONSTRAINT DEFINITIONS
    CONSTRAINT PaymentMethodD_NOT_NULL_PaymentMethodType CHECK (
        PaymentMethodType IS NOT NULL),

    -- CHECK   CONSTRAINT DEFINITIONS
    CONSTRAINT PaymentMethod_CHECK_PaymentMethodType CHECK (
        PaymentMethodType IN ('CINEMA', 'CARD', 'NONE'))
);

    -- INDEX DEFINITIONS
    CREATE INDEX PaymentMethod_IDX_PaymentMethodType ON PaymentMethod (
        upper(PaymentMethodType));

-- !!!!!!!!! insert the values in the table in the following order
-- 1 CINEMA
-- 2 CARD
```

```
-- 3 NONE
-- ============================



-- # =========================
-- #     Reservations table
-- # =========================
CREATE TABLE Reservations (
  ResID             VARCHAR(44),
  ResDate           DATE,
  TotalPriceToBePaid NUMERIC(10, 2),
  PaymentMethodID   SERIAL,
  UserName          VARCHAR(30),
  ShowLocationID    SERIAL,
  ShowTimeID        SERIAL,

    -- KEY  CONSTRAINT DEFINITIONS
    CONSTRAINT Reservations_PK_ResID    PRIMARY KEY (ResID),

    CONSTRAINT Reservations_FK_Payment  FOREIGN KEY (PaymentMethodID)
               REFERENCES PaymentMethod (PaymentMethodID)
                                  ON DELETE SET DEFAULT
                                  ON UPDATE CASCADE,

    CONSTRAINT Reservations_FK_UserName FOREIGN KEY (UserName)
                     REFERENCES Users       (UserName)
                                  ON DELETE CASCADE
                                  ON UPDATE CASCADE,

    CONSTRAINT Reservations_FK_ShowLocTime FOREIGN KEY (ShowLocationID,
        ShowTimeID) REFERENCES Shows (ShowLocationID, ShowTimeID)
                                  ON DELETE CASCADE
                                  ON UPDATE CASCADE,

    CONSTRAINT Reservations_FK_ShowTimeID FOREIGN KEY (ShowTimeID)
                  REFERENCES ShowTime    (ShowTimeID)
                                  ON DELETE CASCADE
                                  ON UPDATE CASCADE,

    -- CHECK CONSTRAINT DEFINITIONS
    CONSTRAINT Reservations_CHECK_TotalPrice     CHECK (
        TotalPriceToBePaid >= 0.0),

    -- NOT NULL CONSTRAINT DEFINITIONS
    CONSTRAINT Reservations_NOT_NULL_ResDate     CHECK (ResDate
```

```
        IS NOT NULL),
    CONSTRAINT Reservations_NOT_NULL_TotalPrice  CHECK (
        TotalPriceToBePaid IS NOT NULL)
);

    -- INDEX DEFINITIONS
    CREATE INDEX Reservations_IDX_PaymentMethodID           ON
        Reservations (PaymentMethodID);
    CREATE INDEX Reservations_IDX_UserName                  ON
        Reservations (UserName);
    CREATE INDEX Reservations_IDX_ShowsID                   ON
        Reservations (ShowLocationID, ShowTimeID);
    CREATE INDEX Reservations_IDX_PaymentMethodID_UserName_Shows ON
        Reservations (PaymentMethodID, UserName, ShowLocationID,
        ShowTimeID);
    CREATE INDEX Reservations_IDX_PaymentMethodID_Shows      ON
        Reservations (PaymentMethodID, ShowLocationID, ShowTimeID);

-- ============================



-- # ==========================
-- #     BookedSeats table
-- # ==========================
CREATE TABLE BookedSeats (
  BookedSeatID   SERIAL,
  RowNo          NUMERIC(2, 0),
  SeatNo         NUMERIC(2, 0),
  Temp    BOOLEAN      DEFAULT 'NO',
  ExpDate        TIMESTAMP    DEFAULT NULL,
  ShowLocationID SERIAL,
  ShowTimeID     SERIAL,

    -- KEY     CONSTRAINT DEFINITIONS
    CONSTRAINT BookedSeats_PK_BookedSeatID PRIMARY KEY (BookedSeatID),

    CONSTRAINT BookedSeats_FK_ShowLocTime FOREIGN KEY (ShowLocationID,
        ShowTimeID) REFERENCES Shows (ShowLocationID, ShowTimeID)
                                  ON DELETE CASCADE
                                  ON UPDATE CASCADE,

    -- CHECK   CONSTRAINT DEFINITIONS
    CONSTRAINT BookedSeats_CHECK_Rows            CHECK (RowNo
        > 0),
    CONSTRAINT BookedSeats_CHECK_Cols            CHECK (SeatNo
```

```
        > 0),

    --  NOT NULL CONSTRAINT DEFINITIONS
    CONSTRAINT BookedSeats_NOT_NULL_Row          CHECK (RowNo        IS
        NOT NULL),
    CONSTRAINT BookedSeats_NOT_NULL_Seat         CHECK (SeatNo       IS
        NOT NULL)
);

    --  INDEX DEFINITIONS
    CREATE INDEX BookedSeats_IDX_Shows      ON BookedSeats (
        ShowLocationID, ShowTimeID);
    CREATE INDEX BookedSeats_IDX_Shows_Row_Seat ON BookedSeats (
        ShowLocationID, ShowTimeID, RowNo, SeatNo);
-- ============================




-- # =========================
-- #       Tickets table
-- # =========================
CREATE TABLE Tickets (
  TicketID        VARCHAR(44),
  FinalPrice      NUMERIC(10, 2),
  Paid     Boolean DEFAULT 'No',
  ShowLocationID  SERIAL,
  ShowTimeID      SERIAL,
  DiscountSchemaID SERIAL,
  ResID           VARCHAR(44),
  BookedSeatID    SERIAL,

    --  KEY     CONSTRAINT DEFINITIONS
    CONSTRAINT Tickets_PK_TicketID  PRIMARY KEY(TicketID),


    CONSTRAINT Tickets_FK_Show_Disc  FOREIGN KEY (ShowLocationID,
        ShowTimeID, DiscountSchemaID) REFERENCES ShowDiscount (
        ShowLocationID, ShowTimeID, DiscountSchemaID)
                                ON DELETE SET NULL
                               ON UPDATE CASCADE,

    CONSTRAINT Tickets_FK_ResID     FOREIGN KEY (ResID)
        REFERENCES Reservations (ResID)
                                ON DELETE CASCADE
                               ON UPDATE CASCADE,
```

```
    CONSTRAINT Tickets_FK_BookedSeatID FOREIGN KEY (BookedSeatID)
        REFERENCES BookedSeats (BookedSeatID)
                            ON DELETE CASCADE
                            ON UPDATE CASCADE,

    -- CHECK   CONSTRAINT DEFINITIONS
    CONSTRAINT Tickets_CHECK_FinalPrice   CHECK (FinalPrice >= 0.0),


    -- NOT NULL CONSTRAINT DEFINITIONS
    CONSTRAINT Tickets_NOT_NULL_FinalPrice CHECK (FinalPrice IS NOT NULL)
);

    -- INDEX DEFINITIONS
    CREATE INDEX Tickets_IDX_Shows               ON Tickets (
        ShowLocationID, ShowTimeID);
    CREATE INDEX Tickets_IDX_DiscountSchemaID    ON Tickets (
        DiscountSchemaID);
    CREATE INDEX Tickets_IDX_Paid                ON Tickets (Paid);
    CREATE INDEX Tickets_IDX_Shows_DiscountSchemaID ON Tickets (
        ShowLocationID, ShowTimeID, DiscountSchemaID);
    CREATE INDEX Tickets_IDX_ResID               ON Tickets (ResID);
    CREATE INDEX Tickets_IDX_BookedSeatID        ON Tickets (BookedSeatID
        );
    CREATE INDEX Tickets_IDX_Shows_BookedSeatID  ON Tickets (
        ShowLocationID, ShowTimeID, BookedSeatID);

-- ============================


-- # ========================================================================

-- #          STEP 4 --- SELECT A SHOW AND DISPLAY THE CINEMA HALL
-- #                     CONFIGURATION FOR THAT SHOW
-- # ------------------------------------------------------------------------

-- #                        Data Type Creation
-- #
-- # Create the data types that will store all information about a
-- # cinema halls configuration for a given show
-- # ------------------------------------------------------------------------


SET search_path TO cinema, public;
SET DATESTYLE TO ISO;

CREATE TYPE ShowCinemaHallsConf AS
   (Rows           NUMERIC(2,0),
```

```
   Cols           NUMERIC(2,0)
  );

CREATE TYPE ShowDiscountValue AS
  (DiscountValue  NUMERIC(2,1)
  );
-- ================================



-- # ========================================================================

-- #          STEP 4 --- SELECT A SHOW AND DISPLAY THE CINEMA HALL
-- #                     CONFIGURATION FOR THAT SHOW
-- # ------------------------------------------------------------------------

-- #                 Display_Cinema_Hall_Conf Procedure
-- #
-- # Find the cinema hall configuration for the given show i.e.
-- # base price for that show, discoutn values, and all boooked seats
-- #  - It accepts 2 input arguments i.e. ShowLocationID and ShowTimeID
-- #  - It returns an error code followed by a list made of:
-- #    base price,
-- #    discount values[],
-- #    no. of rows for the cinema hall, no. of columns for the cinema
    hall
-- #    and a list of all booked seats
-- #       - error code values:
-- #           203 - Show found according to the given criteria
-- #           403 - Show NOT found according to the given criteria
-- # ========================================================================


SET search_path TO cinema, public;
SET DATESTYLE TO ISO;

CREATE OR REPLACE FUNCTION Display_Cinema_Hall_Conf(INTEGER, INTEGER)
    returns text AS '
DECLARE
    -- Declare aliases for user input.
    show_location_id ALIAS FOR $1;
    show_time_id     ALIAS FOR $2;

    -- Declare a variable to hold the final result that contains:
    -- the discount values, the price value,
    -- the cinema hall configuration i.e. no. of cols and rows,
    -- and the booked seats for that show, respectevly.
```

```
    text_output             TEXT = '''';
    text_output_price       TEXT = '''';
    text_output_discount    TEXT = '''';
    text_output_configuration TEXT = '''';
    text_output_seats       TEXT = '''';
    error_code              TEXT = ''203'';

    myRec Record;

    -- Declare a variable to hold rows from the Prices, CinemaHalls,
    -- DiscountSchema, and BookedSeats table, respectivly
    row_data_price     cinema.Prices%ROWTYPE;
    row_data_cinema    cinema.ShowCinemaHallsConf%ROWTYPE;
    row_data_discount  cinema.ShowDiscountValue%ROWTYPE;
    row_data_booked    cinema.BookedSeats%ROWTYPE;


BEGIN
  SET search_path TO cinema, public;
  SET DATESTYLE TO ISO;

  -- check if the given show_location_id and show_time_id exists
  -- in the shows table and create the error_code based on that
  SELECT INTO myrec * FROM cinema.shows
  WHERE cinema.shows.showlocationid = show_location_id
  AND cinema.shows.showtimeid = show_time_id;

  IF NOT FOUND THEN
    error_code = ''403'';
  END IF;

  -- find the price for that show and add it to the output string
  FOR row_data_price IN
    SELECT Prices.PriceID, Prices.BasePrice
    FROM  cinema.Shows, cinema.Prices
    WHERE cinema.Shows.ShowLocationID = show_location_id
    AND   cinema.Shows.ShowTimeID   = show_time_id
    AND   cinema.Shows.PriceID      = cinema.Prices.PriceID
  LOOP
       -- Insert the result of the select into the text_output variable.
       text_output_price = text_output_price || row_data_price.BasePrice
            || '''';
  END LOOP;

  -- find the cinemahall rows and cols for that show and add them to the
      output string
  FOR row_data_cinema IN
```

```
  SELECT CinemaHalls.Rows, CinemaHalls.Cols
  FROM  cinema.Shows, cinema.ShowLocation, cinema.CinemaHalls
  WHERE cinema.Shows.ShowLocationID = show_location_id
  AND   cinema.Shows.ShowTimeID    = show_time_id
  AND   cinema.Shows.ShowLocationID = cinema.ShowLocation.
      ShowLocationID
  AND   cinema.CinemaHalls.CinemaID = cinema.ShowLocation.CinemaID
  AND   cinema.CinemaHalls.HallID  = cinema.ShowLocation.HallID
LOOP
     -- Insert the result of the select into the text_output variable.
     text_output_configuration = text_output_configuration ||
         row_data_cinema.Rows    || ''|'' ||
         row_data_cinema.Cols;
END LOOP;

-- find the dicount values for that show and add them to the output
    string
FOR row_data_discount IN
  SELECT DiscountSchema.DiscountValue
  FROM  cinema.ShowDiscount, cinema.DiscountSchema
  WHERE cinema.ShowDiscount.ShowLocationID =    show_location_id
  AND   cinema.ShowDiscount.ShowTimeID     =    show_time_id
  AND   cinema.ShowDiscount.DiscountSchemaID =  cinema.DiscountSchema.
      DiscountSchemaID
  AND   (  cinema.DiscountSchema.DiscountType LIKE ''CHILD''
    OR cinema.DiscountSchema.DiscountType LIKE ''STUDENT''
    OR cinema.DiscountSchema.DiscountType LIKE ''PENSIONER''
    OR cinema.DiscountSchema.DiscountType LIKE ''VOUCHER'')
LOOP
  -- Insert the result of the select into the text_output variable.
  IF text_output_discount = '''' THEN
    text_output_discount = text_output_discount || row_data_discount.
        DiscountValue;
  ELSE
    text_output_discount = text_output_discount || ''|'' ||
        row_data_discount.DiscountValue;
  END IF;

END LOOP;

-- find all booked seats for that show and add them to the output
    string
FOR row_data_booked IN
  SELECT *
  FROM  cinema.BookedSeats
  WHERE cinema.BookedSeats.ShowLocationID = show_location_id
  AND   cinema.BookedSeats.ShowTimeID  = show_time_id
```

```
    ORDER BY cinema.BookedSeats.RowNo, cinema.BookedSeats.SeatNo
  LOOP
       -- Insert the result of the select into the text_output variable.
       IF text_output_seats = '''' THEN
    text_output_seats = text_output_seats  || ''(''||
           row_data_booked.RowNo || '','' ||
           row_data_booked.SeatNo || '')'' ;
       ELSE
    text_output_seats = text_output_seats  || ''|(''||
           row_data_booked.RowNo || '','' ||
           row_data_booked.SeatNo || '')'' ;
       END IF;

  END LOOP;

  IF error_code = ''403'' THEN

    RETURN error_code;

  ELSE
    RETURN error_code      || ''\n''||
    text_output_price      || ''\n'' ||
    text_output_discount   || ''\n'' ||
    text_output_configuration || ''\n'' ||
    text_output_seats      || '''';

  END IF;

END;
'language plpgsql;
-- ================================================================================================================



-- # ========================================================================

-- #                      STEP 3 ---- FIND MOVIES
-- # ------------------------------------------------------------------------

-- #                       Data Type Creation
-- #
-- # Create the data type that will store all information about
-- # a movie displayed in a particular city on a given date
-- # ========================================================================


SET search_path TO cinema, public;
SET DATESTYLE TO ISO;
```

```
CREATE TYPE allmoviestype AS
   (cinemaname    VARCHAR(30),
    street        VARCHAR(30),
    city          VARCHAR(25),
    movie         VARCHAR(30),
    hourofshow    TIME,
    showlocationid INTEGER,
    showtimeid    INTEGER
   );
-- ============================================================================================



-- # ============================================================================================

-- #                       STEP 3 ---- FIND MOVIES
-- # --------------------------------------------------------------------------------------------

-- #                 Find_Movies_Criteria Procedure
-- #
-- # Finds a particular movie based on user given searching criteria
-- #  - It accepts 4 input arguments i.e.
-- #     - MovieName followed by the % sign e.g. 'Xm%' to look for 'Xmen
    ␣3'
-- #     - array of CinemaID's
-- #     - city name
-- #     - show date
-- #
-- #  - It returns an error code followed by a list of all Movies
-- #    that are displayed by the given CinemaID's␣on␣the␣given␣Date
-- #      - error code values:
-- #           205 - Movies found according to the   criteria
-- #           405 - Movies NOT found according to the given criteria
-- # ============================================================================================ #



SET search_path TO cinema, public;
SET DATESTYLE TO ISO;

CREATE OR REPLACE FUNCTION Find_Movies_Criteria(text, integer[], text,
    date) returns text AS '
DECLARE
    -- Declare aliases for user input.
    movie_name  ALIAS FOR $1;
```

```
    cinema_array ALIAS FOR $2;
    cinema_city ALIAS FOR $3;
    show_date   ALIAS FOR $4;

    -- Declare a variable to hold the result
    text_output TEXT = ’’’’;

    -- Dimension of the cinemaIDs array
    dimension integer;

    -- Declare a variable to hold rows of AllMoviesType data type
    row_data cinema.AllMoviesType%ROWTYPE;

BEGIN
  SET search_path TO cinema, public;
  SET DATESTYLE TO ISO;

  -- SEARCHING CRITERIA NO.1 i.e.
  -- find a particular movie in a certain range from the given position
      and on a given date
  IF ((movie_name != ’’’’) AND (cinema_array != ’’{}’’) AND (cinema_city
       = ’’’’) AND (show_date IS NOT NULL)) THEN

    dimension := ARRAY_UPPER(cinema_array, 1);

    FOR i IN 1..dimension LOOP
      FOR row_data IN SELECT Cinemas.CinemaName, Cinemas.Street, Cinemas.
          City,
                MovieName, HourOfShow, Shows.ShowLocationID, Shows.
                    ShowTimeID
        FROM   cinema.Movies, cinema.ShowLocation, cinema.ShowTime,
            cinema.HourOfShow, cinema.Shows, cinema.Cinemas
        WHERE  cinema.ShowLocation.CinemaID =  cinema_array[i]
        AND cinema.ShowLocation.CinemaID =  cinema.Cinemas.CinemaID
        AND cinema.ShowLocation.MovieID =   cinema.Movies.MovieID
        AND UPPER(cinema.Movies.MovieName) LIKE UPPER(movie_name)
        AND cinema.Shows.ShowLocationID =   cinema.ShowLocation.
            ShowLocationID
        AND cinema.Shows.ShowTimeID     =   cinema.ShowTime.ShowTimeID
        AND cinema.ShowTime.HourShowID  =   cinema.HourOfShow.HourShowID
        AND cinema.ShowTime.ShowTimeID  IN
            (SELECT ShowTime.ShowTimeID
             FROM cinema.ShowTime NATURAL INNER JOIN cinema.DateOfShow
             WHERE cinema.DateOfShow.DateOfShow = show_date
            )
        ORDER BY MovieName, HourOfShow, Cinemas.CinemaName, Cinemas.City
             , Cinemas.Street
```

```
    LOOP

          -- Insert the result of the select into the text_output
              variable
          text_output = text_output || ''\n'' || row_data.Movie || ''|''
               || row_data.HourOfShow
            || ''|'' || row_data.CinemaName || ''|'' || row_data.City
            || ''|'' || row_data.Street   || ''|'' || row_data.
                ShowLocationID
            || ''|'' || row_data.ShowTimeID;

    END LOOP;

  END LOOP;


  IF text_output = '''' THEN
    text_output = ''405'' || text_output;
  ELSE
    text_output = ''205'' || text_output;
  END IF;

  RETURN text_output;
END IF;



-- SEARCHING CRITERIA NO.2 i.e.
-- find a particular movie in a given city and on a given date
IF ((movie_name != '''') AND (cinema_array = ''{}'') AND (cinema_city
     != '''') AND (show_date IS NOT NULL)) THEN

  FOR row_data IN SELECT Cinemas.CinemaName, Cinemas.Street, Cinemas.
      City,
       MovieName, HourOfShow, Shows.ShowLocationID, Shows.ShowTimeID
    FROM   cinema.Movies, cinema.ShowLocation, cinema.ShowTime, cinema.
        HourOfShow, cinema.Shows, cinema.Cinemas
    WHERE  cinema.ShowLocation.CinemaID =   cinema.Cinemas.CinemaID
    AND UPPER(cinema.Cinemas.City)  LIKE  UPPER(cinema_city)
    AND cinema.ShowLocation.MovieID =    cinema.Movies.MovieID
    AND UPPER(cinema.Movies.MovieName) LIKE UPPER(movie_name)
    AND cinema.Shows.ShowLocationID =    cinema.ShowLocation.
        ShowLocationID
    AND cinema.Shows.ShowTimeID     =    cinema.ShowTime.ShowTimeID
    AND cinema.ShowTime.HourShowID  =    cinema.HourOfShow.HourShowID
    AND cinema.ShowTime.ShowTimeID  IN
       (SELECT ShowTime.ShowTimeID
```

```
        FROM cinema.ShowTime NATURAL INNER JOIN cinema.DateOfShow
        WHERE cinema.DateOfShow.DateOfShow = show_date
       )
    ORDER BY MovieName, HourOfShow, Cinemas.CinemaName, Cinemas.City,
        Cinemas.Street
  LOOP

       -- Insert the result of the select into the text_output variable
           .
    text_output = text_output || ''\n'' || row_data.Movie || ''|'' ||
        row_data.HourOfShow
           || ''|'' || row_data.CinemaName || ''|'' || row_data.City
           || ''|'' || row_data.Street   || ''|'' || row_data.
               ShowLocationID
           || ''|'' || row_data.ShowTimeID;


  END LOOP;

  IF text_output = '''' THEN
    text_output = ''406'' || text_output;
  ELSE
    text_output = ''206'' || text_output;
  END IF;

  RETURN text_output;

END IF;



-- SEARCHING CRITERIA NO.3 i.e.
-- find all movies in a given city and on a given date
IF ((movie_name = '''') AND (cinema_array = ''{}'') AND (cinema_city
     != '''') AND (show_date IS NOT NULL)) THEN

  FOR row_data IN SELECT Cinemas.CinemaName, Cinemas.Street, Cinemas.
      City,
       MovieName, HourOfShow, Shows.ShowLocationID, Shows.ShowTimeID
    FROM   cinema.Movies, cinema.ShowLocation, cinema.ShowTime, cinema.
        HourOfShow, cinema.Shows, cinema.Cinemas
    WHERE  cinema.ShowLocation.CinemaID =  cinema.Cinemas.CinemaID
    AND UPPER(cinema.Cinemas.City) LIKE UPPER(cinema_city)
    AND cinema.ShowLocation.MovieID =  cinema.Movies.MovieID
    AND cinema.Shows.ShowLocationID =  cinema.ShowLocation.
        ShowLocationID
    AND cinema.Shows.ShowTimeID   =    cinema.ShowTime.ShowTimeID
```

```
    AND cinema.ShowTime.HourShowID =   cinema.HourOfShow.HourShowID
    AND cinema.ShowTime.ShowTimeID IN
        (SELECT ShowTime.ShowTimeID
         FROM cinema.ShowTime NATURAL INNER JOIN cinema.DateOfShow
         WHERE cinema.DateOfShow.DateOfShow = show_date
        )
    ORDER BY MovieName, HourOfShow, Cinemas.CinemaName, Cinemas.City,
         Cinemas.Street
  LOOP

       -- Insert the result of the select into the text_output variable
       text_output = text_output || ''\n'' || row_data.Movie || ''|''
            || row_data.HourOfShow
         || ''|'' || row_data.CinemaName || ''|'' || row_data.City
         || ''|'' || row_data.Street   || ''|'' || row_data.
             ShowLocationID
         || ''|'' || row_data.ShowTimeID;

  END LOOP;

  IF text_output = '''' THEN
    text_output = ''407'' || text_output;
  ELSE
    text_output = ''207'' || text_output;
  END IF;

  RETURN text_output;

END IF;



-- SEARCHING CRITERIA NO.4 i.e.
-- find all movies in a certain range from the given position and on a
    given date
IF ((movie_name = '''') AND (cinema_array != ''{}'') AND (cinema_city
     = '''') AND (show_date IS NOT NULL)) THEN

  dimension := ARRAY_UPPER(cinema_array, 1);

  FOR i IN 1..dimension LOOP
    FOR row_data IN SELECT Cinemas.CinemaName, Cinemas.Street, Cinemas.
         City,
          MovieName, HourOfShow, Shows.ShowLocationID, Shows.ShowTimeID
      FROM   cinema.Movies, cinema.ShowLocation, cinema.ShowTime,
           cinema.HourOfShow, cinema.Shows, cinema.Cinemas
      WHERE  cinema.ShowLocation.CinemaID = cinema_array[i]
```

```
      AND cinema.ShowLocation.CinemaID = cinema.Cinemas.CinemaID
      AND cinema.ShowLocation.MovieID = cinema.Movies.MovieID
      AND cinema.Shows.ShowLocationID = cinema.ShowLocation.
          ShowLocationID
      AND cinema.Shows.ShowTimeID   = cinema.ShowTime.ShowTimeID
      AND cinema.ShowTime.HourShowID = cinema.HourOfShow.HourShowID
      AND cinema.ShowTime.ShowTimeID IN
          (SELECT ShowTime.ShowTimeID
           FROM cinema.ShowTime NATURAL INNER JOIN cinema.DateOfShow
           WHERE cinema.DateOfShow.DateOfShow = show_date
          )
      ORDER BY MovieName, HourOfShow, Cinemas.CinemaName, Cinemas.City
          , Cinemas.Street
    LOOP

          -- Insert the result of the select into the text_output
              variable.
          text_output = text_output || ''\n'' || row_data.Movie || ''|''
               || row_data.HourOfShow
            || ''|'' || row_data.CinemaName || ''|'' || row_data.City
            || ''|'' || row_data.Street   || ''|'' || row_data.
                ShowLocationID
            || ''|'' || row_data.ShowTimeID;


    END LOOP;

  END LOOP;

  IF text_output = '''' THEN
    text_output = ''408'' || text_output;
  ELSE
    text_output = ''208'' || text_output;
  END IF;

  RETURN text_output;

END IF;



-- SEARCHING CRITERIA NO.5 i.e.
-- find all cinemas where the given move is played on the given date
IF ((movie_name != '''') AND (cinema_array = ''{}'') AND (cinema_city
     = '''') AND (show_date IS NOT NULL)) THEN

  FOR row_data IN
```

```
    SELECT   Cinemas.CinemaName, Cinemas.Street, Cinemas.City,
        MovieName,
       HourOfShow, Shows.ShowLocationID, Shows.ShowTimeID
      FROM   cinema.Movies, cinema.ShowLocation, cinema.ShowTime,
          cinema.HourOfShow, cinema.Shows, cinema.Cinemas
      WHERE  cinema.ShowLocation.CinemaID =   cinema.Cinemas.CinemaID
      AND UPPER(cinema.Movies.MovieName) LIKE UPPER(movie_name)
      AND cinema.ShowLocation.MovieID =    cinema.Movies.MovieID
      AND cinema.Shows.ShowLocationID =    cinema.ShowLocation.
          ShowLocationID
      AND cinema.Shows.ShowTimeID    =     cinema.ShowTime.ShowTimeID
      AND cinema.ShowTime.HourShowID  =    cinema.HourOfShow.HourShowID
      AND cinema.ShowTime.ShowTimeID  IN
          (SELECT ShowTime.ShowTimeID
           FROM cinema.ShowTime NATURAL INNER JOIN cinema.DateOfShow
           WHERE cinema.DateOfShow.DateOfShow = show_date
          )
      ORDER BY MovieName, HourOfShow, Cinemas.CinemaName, Cinemas.City
          , Cinemas.Street
  LOOP

          -- Insert the result of the select into the text_output
              variable.
          text_output = text_output || ''\n'' || row_data.Movie || ''|''
               || row_data.HourOfShow
            || ''|'' || row_data.CinemaName || ''|'' || row_data.City
            || ''|'' || row_data.Street   || ''|'' || row_data.
                ShowLocationID
            || ''|'' || row_data.ShowTimeID;


  END LOOP;

  IF text_output = '''' THEN
    text_output = ''409'' || text_output;
  ELSE
    text_output = ''209'' || text_output;
  END IF;

  RETURN text_output;

 END IF;

END;
'language␣plpgsql;
-- ============================================================================================
```

```
-- # ========================================================================

-- #             Get All Cinemas Procedure - Data Type Creation
-- # ------------------------------------------------------------------------

-- # Creates a type to hold a cinema result set.
-- # ------------------------------------------------------------------------


SET search_path TO cinema, public;
SET DATESTYLE TO ISO;

CREATE TYPE AllCinemasType AS
  (CinemaID  INTEGER,
   CinemaName VARCHAR(30),
   Street    VARCHAR(30),
   City      VARCHAR(25),
   Zip       VARCHAR(8)
  );

-- # ========================================================================

-- #                     Get All Cinemas Procedure
-- #
-- # Returns a set of Cinemas i,e, (CinemaID, CinemaName, Street, City,
    Zip)
-- # ------------------------------------------------------------------------


SET search_path TO cinema, public;
SET DATESTYLE TO ISO;

CREATE OR REPLACE FUNCTION Get_All_Cinemas () RETURNS SETOF
    AllCinemasType AS '

  SET search_path TO cinema, public;
  SET DATESTYLE TO ISO;

  SELECT *
  FROM  cinema.Cinemas;

'␣LANGUAGE␣'sql';
-- ============================================================================



-- # ============================================================
```

```
-- #
-- #                  STEP 8. GET MOVIE DETAILS
-- #
-- # ------------------------------------------------------------
-- #                      Data Type Creation
-- #
-- # Creates a data type to hold the movie description as TEXT and
-- # the movie poster as BYTEA
-- # ============================================================

SET search_path TO cinema, public;
SET DATESTYLE TO ISO;

CREATE TYPE movie_details AS (
   MovieDetails TEXT,
   Poster     BYTEA
);


-- # ============================================================
-- #
-- #                  STEP 8. GET MOVIE DETAILS
-- #
-- # ------------------------------------------------------------
-- #                   Movie_Details Procedure
-- #
-- # Get all info about a movie and send it back as response.
-- # If theere is an entry fot that user and movie in the rating
-- # table, update the user rating score. Else create a new entry.
-- #    - It accepts as input 1 arguments i.e. ShowLocationID
-- #   - It returns an erro_code followed by the movie info
-- #       - error codes:
-- #            417 = error;
-- #            217 = OK
-- # ============================================================

SET search_path TO cinema, public;
SET DATESTYLE TO ISO;

CREATE OR REPLACE FUNCTION Movie_Details (INTEGER) RETURNS movie_details
    AS '
DECLARE

   -- Declare aliases for user input.
   show_location_id ALIAS FOR $1;

   -- Declare a variable to hold the error
```

```
    error_code      INTEGER;

    -- hold the movie ID for the gven show
    movie_ID        INTEGER;

    -- hold the user average rating score for this movie
    user_rating     FLOAT;

    -- Declare a variable to hold themovie details
    movie_output    TEXT = '''';

    -- Declare a variable to hold the output result
    text_output     TEXT = '''';

    -- variable to hold the TEXTUAL description of the movie
    MovieDetails    TEXT = '''';

    -- Declare a variable to hold rows of the Movies type,
    -- and the Movie description and BYTEA complex return type
    row_data_movies cinema.Movies%ROWTYPE;
    row_data_details cinema.movie_details%ROWTYPE;

BEGIN
  SET search_path TO cinema, public;
  SET DATESTYLE TO ISO;


  -- get the movie_ID for the given show_location_ID
  SELECT INTO movie_ID ShowLocation.MovieID
  FROM  cinema.ShowLocation
  WHERE cinema.ShowLocation.ShowLocationID = show_location_id;

  -- if there is no such movie return an error
  IF movie_ID IS NULL THEN
    error_code :=417;

  ELSE
    -- get the average user rating score for the given movie
    SELECT INTO user_rating AVG(Rating.UserRating)
    FROM  cinema.Rating
    WHERE cinema.Rating.MovieID = movie_ID;

    -- get movie info
    FOR row_data_movies IN
      SELECT *
      FROM   cinema.Movies
      WHERE  cinema.Movies.MovieID = movie_ID
```

```
  LOOP
    -- create the output movie details
    movie_output := movie_output ||
        row_data_movies.MovieID           || ''\n'' ||
        row_data_movies.MovieName         || ''\n'' ||
        row_data_movies.Duration          || ''\n'' ||
        row_data_movies.Genre             || ''\n'' ||
        row_data_movies.ParentClassification || ''\n'' ||
        row_data_movies.Language          || ''\n'' ||
        row_data_movies.Year              || ''\n'' ||
        row_data_movies.MovieCountry      || ''\n'' ||
        user_rating                       || ''\n'' ||
        row_data_movies.Director          || ''\n'' ||
        row_data_movies.Actors            || ''\n'' ||
        row_data_movies.Description       || ''\n'';
  END LOOP;

  error_code := 217;
END IF;


-- build the output result
IF error_code = 417 THEN

  MovieDetails := error_code;

  FOR row_data_details IN
    SELECT MovieDetails, NULL
  LOOP
       return row_data_details;
  END LOOP;

ELSE
  text_output = text_output || error_code || ''\n'' || movie_output;
  MovieDetails := text_output;

  FOR row_data_details IN
    SELECT MovieDetails, Poster
    FROM  cinema.Movies
    WHERE cinema.Movies.MovieID = movie_ID
  LOOP
       return row_data_details;

  END LOOP;

END IF;
```

```
END;
'language plpgsql;
-- ================================================================================


-- # ============================================================================================

-- #                STEP 11 ---- MOVIE LOCATION SERVICE
-- # --------------------------------------------------------------------------------------------

-- #                 Movie_Location_Service Procedure
-- #
-- # Finds all cinemas ia certain range from user's given position.
-- # It retrieves the list of CinemaId's as a string separated by "|"
-- #  - It accepts 5 input arguments i.e.
-- #       range(>=0), movie name, street, city, zip.
-- #       The street, city, zip can be preceded,
-- #       replaced or followed by the % sign
-- #  - It returns an error code followed by a list of all found CinemaId
     's
-- #   separated by "|"
-- #      - error code values:
-- #           219 - Cinemas found according to the given criteria
-- #           419 - Cinemas NOT found according to the given criteria
-- # ============================================================================================


SET search_path TO cinema, public;
SET DATESTYLE TO ISO;

CREATE OR REPLACE FUNCTION Movie_Location_Service(integer, text, text,
    text, text) returns text AS '
DECLARE
    -- Declare aliases for user input.
    range  ALIAS FOR $1;
    movie  ALIAS FOR $2;
    street ALIAS FOR $3;
    city   ALIAS FOR $4;
    zip    ALIAS FOR $5;

    -- Declare a variable to hold the result
    text_output TEXT = '''';

    -- Declare the row_type from the selection
    row_data cinema.Cinemas%ROWTYPE;


BEGIN
```

```
SET search_path TO cinema, public;
SET DATESTYLE TO ISO;

-- case 1 i.e. Protocol step 3.1
     -- user submits: movie, stret, city, zip, range, date
IF movie <> '''' THEN

  IF range = 0 THEN

    FOR row_data IN    SELECT Cinemas.CinemaID
        FROM   cinema.Cinemas
        WHERE  UPPER(cinema.Cinemas.Street) LIKE UPPER(street)
        AND UPPER(cinema.Cinemas.City) LIKE UPPER(city)
        AND UPPER(cinema.Cinemas.Zip) LIKE UPPER(zip)
    LOOP
      -- Insert the result of the select into the text_output variable
      text_output = text_output || ''|'' || row_data.CinemaID;
    END LOOP;

  ELSIF range > 0 THEN

    FOR row_data IN    SELECT Cinemas.CinemaID
        FROM   cinema.Cinemas
        WHERE  UPPER(cinema.Cinemas.City) LIKE UPPER(city)
        AND UPPER(cinema.Cinemas.Zip) LIKE UPPER(zip)
    LOOP
      -- Insert the result of the select into the text_output variable
      text_output = text_output || ''|'' || row_data.CinemaID;
    END LOOP;

  END IF;

-- case 4 i.e. Protocol step 3.4
-- user submits: stret, city, zip, range, date
ELSE

  IF range = 0 THEN

    FOR row_data IN    SELECT Cinemas.CinemaID
        FROM   cinema.Cinemas
        WHERE  UPPER(cinema.Cinemas.Street) LIKE UPPER(street)
        AND UPPER(cinema.Cinemas.City) LIKE UPPER(city)
        AND UPPER(cinema.Cinemas.Zip) LIKE UPPER(zip)
    LOOP
      -- Insert the result of the select into the text_output variable
      text_output = text_output || ''|'' || row_data.CinemaID;
    END LOOP;
```

```
    ELSIF range > 0 THEN

      FOR row_data IN    SELECT Cinemas.CinemaID
          FROM   cinema.Cinemas
          WHERE  UPPER(cinema.Cinemas.City) LIKE UPPER(city)
          AND UPPER(cinema.Cinemas.Zip) LIKE UPPER(zip)
      LOOP
        -- Insert the result of the select into the text_output variable
        text_output = text_output || ''|'' || row_data.CinemaID;
      END LOOP;

    END IF;

  END IF;



  IF text_output = '''' THEN
    text_output = ''419'' || text_output;
  ELSE
    text_output = ''219'' || text_output;
  END IF;

  RETURN text_output;
END;
'language␣plpgsql;
-- ================================================================================================



-- # ============================================================
-- #
-- #                      STEP 7. RATE A MOVIE
-- #
-- # ------------------------------------------------------------
-- #                      Rate_Movie Procedure
-- #
-- # Check if the user is authenticated and if so, rate the movie.
-- # If theere is an entry fot that user and movie in the rating
-- # table, update the user rating score. Else create a new entry.
-- #    - It accepts as input 4 arguments i.e. UserName, OTP,
-- #      ShowLocationID, and UserRatingScore
-- #   - It returns an erro_code i.e.
-- #         201 = user authenticated,
-- #         401 = user not authenticated,
-- #         418 = error,
-- #         218 = rating done
```

```
-- # ============================================================

SET search_path TO cinema, public;
SET DATESTYLE TO ISO;

CREATE OR REPLACE FUNCTION Rate_Movie(TEXT, TEXT, INTEGER, INTEGER)
    returns text AS '
DECLARE

    -- Declare aliases for user input.
    user_name       ALIAS FOR $1;
    otp             ALIAS FOR $2;
    show_location_id ALIAS FOR $3;
    rating_score    ALIAS FOR $4;

    -- Declare a variable to hold the authentication result
    -- i.e. 201 for user authenticated, 401 else
    error_code    INTEGER;

    -- hold the movie ID for the gven show
    movie_ID      INTEGER;

    -- Dimension of the array of seat_row array
    previous_rating INTEGER;

    -- Declare a variable to hold the output result
    text_output   TEXT = '''';

BEGIN
  SET search_path TO cinema, public;
  SET DATESTYLE TO ISO;

  -- verify if user credentials are correct
  error_code := Authenticate(user_name, otp);

       -- if user is authenticated
  IF error_code = 201 THEN

    -- get the movie_ID for the given show_location_ID
    SELECT INTO movie_ID ShowLocation.MovieID
    FROM cinema.ShowLocation
    WHERE cinema.ShowLocation.ShowLocationID = show_location_id;

    -- if there is no such movie return an error
    IF movie_ID IS NULL THEN
      error_code = 418;
```

```
    ELSE
      -- check if the user has rated this movie before
      SELECT INTO previous_rating Count(*)
      FROM cinema.Rating
      WHERE cinema.Rating.UserName = user_name
      AND   cinema.Rating.MovieID = movie_ID;

      -- if the user has NOT rated this movie before
      --  insert the data into the rating table
      IF previous_rating = 0 THEN

        INSERT INTO cinema.Rating
        VALUES (user_name, movie_ID, rating_score);
        error_code := 218;

      ELSE
        -- if user has rated this movie before,
        -- update the rating score value
        UPDATE cinema.Rating SET UserRating = rating_score
        WHERE cinema.Rating.UserName = user_name
        AND   cinema.Rating.MovieID = movie_ID;
        error_code := 218;

      END IF;

    END IF;
  END IF;

  -- build the output result
  IF error_code = 418 OR error_code = 401 THEN
    text_output = error_code;
  ELSE
    text_output = text_output || error_code;
  END IF;

  RETURN text_output;
END;
'language␣plpgsql;
-- ============================================================


-- # ================================================================
-- #
-- #   STEP 10. --- CANCEL ALL SELECTED SEATS BY USER IN CASE HE/SHE
-- #               DOES NOT WANT TO ACCEPT THE PAYMENT CONDITIONS
-- #
-- # ----------------------------------------------------------------
-- #        Reject_Payment_Cancel_Selected_Seats Procedure
```

```
-- #
-- #    Cancel all selected seats by user in case he/she
-- #     does not want to accept the payment conditions
-- #      - It accepts as input 3 arguments i.e.
-- #        ShowLocationID, ShowTimeID, and an array of [row, seat]
-- #    - It returns an error code i.e
-- #          416 = error;
-- #          216 = all seats are cancelled
-- # ================================================================

SET search_path TO cinema, public;
SET DATESTYLE TO ISO;

CREATE OR REPLACE FUNCTION Reject_Payment_Cancel_Selected_Seats (INTEGER
    , INTEGER, INTEGER[]) returns text AS '
DECLARE

   -- Declare aliases for user input.
   show_location_id ALIAS FOR $1;
   show_time_id     ALIAS FOR $2;
   row_seat         ALIAS FOR $3;

   -- Declare a variable to hold an error_code
   error_code       INTEGER;

   -- Declare a variable to hold the return result
   text_out         TEXT;

   -- declare an array to hold the number of seats reserved by user
   dimension_row_seat INTEGER;

   -- Declare a variable to hold the BookedSeatID
   booked_seat_id   INTEGER;

BEGIN
  SET search_path TO cinema, public;
  SET DATESTYLE TO ISO;

  dimension_row_seat := ARRAY_UPPER(row_seat, 1);

  text_out := '''';

  -- begin TRANSACTION mode
  BEGIN
    -- check if the selected seat exists in the booked seats table
    FOR i IN 1..dimension_row_seat LOOP
```

```
      SELECT INTO booked_seat_id BookedSeatID
      FROM  cinema.BookedSeats
      WHERE cinema.BookedSeats.ShowLocationID = show_location_id
      AND   cinema.BookedSeats.ShowTimeID  = show_time_id
      AND   cinema.BookedSeats.RowNo       = row_seat [i][1]
      AND   cinema.BookedSeats.SeatNo      = row_seat [i][2];

      -- seat cannot be found
      IF NOT FOUND THEN
        RAISE EXCEPTION ''Nonexistent ID --> %'', booked_seat_id;

      ELSE
        -- delete the seat
        DELETE
        FROM  cinema.BookedSeats
        WHERE cinema.BookedSeats.ShowLocationID = show_location_id
        AND   cinema.BookedSeats.ShowTimeID  = show_time_id
        AND   cinema.BookedSeats.RowNo       = row_seat [i][1]
        AND   cinema.BookedSeats.SeatNo      = row_seat [i][2];

      END IF;

    END LOOP;

    text_out := text_out || ''216'';


  -- catch any exception might occur
  EXCEPTION
    WHEN OTHERS THEN
      text_out := text_out || ''416'';
  END;
  -- end TRANSACTION mode

  RETURN text_out;
END;
'language␣plpgsql;
-- ============================================================


-- # ========================================================================

-- #       STEP 4.2 --- CHECK ID USER'S␣SELECTED␣SEATS␣ARE␣STILL␣FREE
-- #                   AND SAVE THEM TO DB IF YES
-- # ------------------------------------------------------------------------

-- #              Select_Deselect_Many_Seats Procedure
-- #
```

```
-- # Check if the user's selected seats are still free
-- # when user presses (DE)SELECT in the Seat Selection form.
-- # Make user's seat selection persistent in the DB.
-- # If user presses (DE)SELECT for the seats that he/she has just
    selected,
-- # remove the previous selected seat from the DB
-- #  - It accepts as input 5 arguments i.e.
-- #       CommandCode e.g "SELECT = 1 / DESELECT = 2",
-- #       ShowLocationID, ShowTimeID, and an array of [row, seat]
-- #  - It returns an error code followed by a list of all booked seats
-- #    including the latest ones
-- #      - erorr code
-- #           210 Seat Selected
-- #           211 Seat Deselected
-- #           410 Error OR Seat Already Selected
-- #           411 Error OR Seat Already Deselected
-- # ============================================================================


SET search_path TO cinema, public;
SET DATESTYLE TO ISO;

CREATE OR REPLACE FUNCTION Select_Deselect_Many_Seats(INTEGER, INTEGER,
    INTEGER, INTEGER[]) returns text AS '
DECLARE

    -- Declare aliases for user input.
    command_code     ALIAS FOR $1; --SELECT = 1, DESELECT = 2
    show_location_id ALIAS FOR $2;
    show_time_id     ALIAS FOR $3;
    selectedSeats    ALIAS FOR $4;

    -- Declare a variable to hold the booked seats for that show
    text_output_seats TEXT = '''';

    error_code TEXT = '''';

    -- declare an array to hold the number of seats reserved by user
    dimension_row_seat INTEGER;

    -- Declare a variable to hold rows of BookedSeats type
    row_data_booked cinema.BookedSeats%ROWTYPE;

    -- Declare a variable to hold the BookedSeatID
    booked_seat_id  INTEGER;

    -- Declare a variable to hold the BookedSeatID for deletion
```

```
    booked_seat_id_del INTEGER;

    -- Declare expiration date for the booked seat
    seat_exp_date   TIMESTAMP;

BEGIN
  SET search_path TO cinema, public;
  SET DATESTYLE TO ISO;

  dimension_row_seat := ARRAY_UPPER(selectedSeats, 1);

  -- SELECT SEAT
  IF command_code = 1 THEN

    -- begin TRANSACTION mode
    BEGIN
      -- check if the selected seats are free.
      -- If the seats are free make the selection persistent to the DB
      FOR i IN 1..dimension_row_seat LOOP

        SELECT INTO booked_seat_id BookedSeatID
        FROM  cinema.BookedSeats
        WHERE cinema.BookedSeats.ShowLocationID = show_location_id
        AND   cinema.BookedSeats.ShowTimeID  = show_time_id
        AND   cinema.BookedSeats.RowNo       = selectedSeats[i][1]
        AND   cinema.BookedSeats.SeatNo      = selectedSeats[i][2];

        -- seat is not free
        IF FOUND THEN
          RAISE EXCEPTION ''Seat already reserved ID --> %'',
              booked_seat_id;

        -- seat is free
        ELSE
          -- save the seat and row in the booked seats table
          seat_exp_date := ''now'';
          seat_exp_date := seat_exp_date + interval ''10 minutes'';
          INSERT INTO cinema.BookedSeats(RowNo, SeatNo, Temp, ExpDate,
              ShowLocationID, ShowTimeID)
          VALUES (selectedSeats[i][1], selectedSeats[i][2], ''Y'',
              seat_exp_date, show_location_id, show_time_id);
          error_code := ''210'';
        END IF;

      END LOOP;

    -- catch any exception might occur
```

```
  EXCEPTION
    WHEN OTHERS THEN
      error_code := ''410'';
  END;
  -- end TRANSACTION mode


END IF;


-- DESELECT SEAT
IF command_code = 2 THEN

  -- begin TRANSACTION mode
  BEGIN
    -- check if the selected seat exists in the booked seats table as
        temp seat
    FOR i IN 1..dimension_row_seat LOOP

      SELECT INTO booked_seat_id BookedSeatID
      FROM  cinema.BookedSeats
      WHERE cinema.BookedSeats.ShowLocationID = show_location_id
      AND   cinema.BookedSeats.ShowTimeID  = show_time_id
      AND   cinema.BookedSeats.temp        = true
      AND   cinema.BookedSeats.RowNo       = selectedSeats[i][1]
      AND   cinema.BookedSeats.SeatNo      = selectedSeats[i][2];

      IF NOT FOUND THEN
        RAISE EXCEPTION ''Seat already reserved ID --> %'',
            booked_seat_id;

      ELSE
        DELETE
        FROM  cinema.BookedSeats
        WHERE cinema.BookedSeats.ShowLocationID = show_location_id
        AND   cinema.BookedSeats.ShowTimeID  = show_time_id
        AND   cinema.BookedSeats.temp        = true
        AND   cinema.BookedSeats.RowNo       = selectedSeats[i][1]
        AND   cinema.BookedSeats.SeatNo      = selectedSeats[i][2];

        error_code := ''211'';

      END IF;

    END LOOP;

  -- catch any exception might occur
```

```
    EXCEPTION
      WHEN OTHERS THEN
        error_code = ''411'';
    END;
    -- end TRANSACTION mode

  END IF;


  text_output_seats = text_output_seats || error_code || ''\n'' ;

  -- find all booked seats for that show and add them to the output
      string
  FOR row_data_booked IN
    SELECT *
    FROM  cinema.BookedSeats
    WHERE cinema.BookedSeats.ShowLocationID = show_location_id
    AND   cinema.BookedSeats.ShowTimeID  = show_time_id
    ORDER BY cinema.BookedSeats.RowNo, cinema.BookedSeats.SeatNo
  LOOP

    text_output_seats = text_output_seats  || ''|''||
                row_data_booked.RowNo || '','' ||
           row_data_booked.SeatNo ;

  END LOOP;

  RETURN text_output_seats;
END;
'language␣plpgsql;
-- ================================================================================



-- # ============================================================================

-- #       STEP 4.2 --- CHECK ID USER'S␣SELECTED␣SEAT␣IS␣STILL␣FREE
-- #                    AND SAVE THEM TO DB IF THEY IF SO
-- # ----------------------------------------------------------------------------

-- #                    Select_Deselect_Seat Procedure
-- #
-- # Check if the user's␣selected␣seat␣is␣still␣free
-- # when user presses (DE)SELECT in the Seat Selection form.
-- # Make user's␣seat␣selection␣persistent␣in␣the␣DB.
-- # If user presses (DE)SELECT for a seat that he/she has just selected,
-- # remove the previous selected seat from the DB
-- #  - It accepts as input 5 arguments i.e.
```

```
-- #       CommandCode e.g "SELECT␣=␣1␣/␣DESELECT␣=␣2",
-- #       ShowLocationID, ShowTimeID,
-- #       SelectedRowNo, and SelectedSeatNo.
-- #  - It returns an error code followed by a list of all booked seats
-- #    including the latest ones
-- #      - erorr code
-- #           200 Select / Deselect Done
-- #           400 Error
-- #           410 Seat Already Selected
-- # ========================================================================


SET search_path TO cinema, public;
SET DATESTYLE TO ISO;

CREATE OR REPLACE FUNCTION Select_Deselect_Seat(INTEGER, INTEGER, INTEGER
    , INTEGER, INTEGER) returns text AS '
DECLARE

    -- Declare aliases for user input.
    command_code     ALIAS FOR $1; --SELECT = 1, DESELECT = 2
    show_location_id ALIAS FOR $2;
    show_time_id     ALIAS FOR $3;
    row_no           ALIAS FOR $4;
    seat_no          ALIAS FOR $5;

    -- Declare a variable to hold the booked seats for that show
    text_output_seats TEXT = '''';

    error_code TEXT = '''';

    -- Declare a variable to hold the seat free/booked code
    -- if the seat is free or not i.e.
    -- seat free = 1, seat already booked = 2, NONE = 0
    return_seat_free_taken_code TEXT = '''';

    -- Declare a variable to hold the return code
    -- if the seat is free or not i.e. seat has been booked = 1,
    -- seat has been canceled = 2; error = 3; NONE = 0;
    return_booked_canceled_code TEXT = '''';

    -- Declare a variable to hold rows of BookedSeats type
    row_data_booked cinema.BookedSeats%ROWTYPE;

    -- Declare a variable to hold the BookedSeatID
    booked_seat_id  INTEGER;
```

```
    -- Declare expirataion date for the booked seat
    seat_exp_date   TIMESTAMP;

BEGIN
  SET search_path TO cinema, public;
  SET DATESTYLE TO ISO;

  -- SELECT SEAT
  IF command_code = 1 THEN

    -- check if the selected seat is free.
    -- If the seat is free make the selection persistent to the DB
    SELECT INTO booked_seat_id BookedSeatID
    FROM  cinema.BookedSeats
    WHERE cinema.BookedSeats.ShowLocationID = show_location_id
    AND   cinema.BookedSeats.ShowTimeID  = show_time_id
    AND   cinema.BookedSeats.SeatNo      = seat_no
    AND   cinema.BookedSeats.RowNo       = row_no;

    -- seat is free
    IF NOT FOUND THEN
      -- return a code i.e. seat free = 1
      return_seat_free_taken_code := 1;

      -- save the seat and row in the booked seats table
      seat_exp_date := ''now'';
      seat_exp_date := seat_exp_date + interval ''10 minutes'';
      INSERT INTO cinema.BookedSeats(RowNo, SeatNo, Temp, ExpDate,
          ShowLocationID, ShowTimeID)
      VALUES (row_no, seat_no, ''Y'', seat_exp_date, show_location_id,
          show_time_id);

      -- seat has been booked
      return_booked_canceled_code := 1;
      error_code := ''200'';

    -- seat is booked
    ELSE
      -- return a code i.e. seat already booked by another user = 2
      return_seat_free_taken_code := 2;
      return_booked_canceled_code := 0;

      error_code := ''410'';
    END IF;

    IF error_code = '''' THEN
      error_code := ''400'';
```

```
  END IF;

END IF;


-- DESELECT SEAT
IF command_code = 2 THEN

  -- check if the selected seat exists in the booked seats table
  SELECT INTO booked_seat_id BookedSeatID
  FROM  cinema.BookedSeats
  WHERE cinema.BookedSeats.ShowLocationID = show_location_id
  AND   cinema.BookedSeats.ShowTimeID  = show_time_id
  AND   cinema.BookedSeats.SeatNo      = seat_no
  AND   cinema.BookedSeats.RowNo       = row_no;

  -- seat exists (data is correct)
  IF FOUND THEN
    -- cancel the previosly selected seat
    DELETE
    FROM  cinema.BookedSeats
    WHERE cinema.BookedSeats.ShowLocationID = show_location_id
    AND   cinema.BookedSeats.ShowTimeID  = show_time_id
    AND   cinema.BookedSeats.SeatNo      = seat_no
    AND   cinema.BookedSeats.RowNo       = row_no;

    return_seat_free_taken_code := 0;
    return_booked_canceled_code := 2;
    error_code := ''200'';
  ELSE
    -- return a code i.e. seat does not exist - ERROR
    return_seat_free_taken_code := 0;
    return_booked_canceled_code := 3;
    error_code := ''400'';
  END IF;

  IF error_code = '''' THEN
    error_code := ''400'';
  END IF;

END IF;


-- find all booked seats for that show and add them to the output
    string
FOR row_data_booked IN
  SELECT *
```

```
    FROM  cinema.BookedSeats
    WHERE cinema.BookedSeats.ShowLocationID = show_location_id
    AND   cinema.BookedSeats.ShowTimeID  = show_time_id
  LOOP
       -- Insert the result of the select into the text_output variable.
       IF text_output_seats = '''' THEN

    text_output_seats =  text_output_seats    ||
         error_code           || ''\n(''||
         row_data_booked.RowNo || '','' ||
         row_data_booked.SeatNo || '')'' ;
       ELSE

    text_output_seats =  text_output_seats    || ''|(''||
         row_data_booked.RowNo || '','' ||
         row_data_booked.SeatNo || '')'' ;
       END IF;

  END LOOP;

  RETURN text_output_seats;
END;
'language␣plpgsql;
-- ============================================================================
```

Appendix E

Server Side Configuration

E.1 Log4j Configuration File

```
#
# ------------------------------------------------------------------------
#              Log4j configuration file for the Cinema Service
# ------------------------------------------------------------------------
#
#
# ------------------------------------------------------------------------
#
# DEBUG to write debugging messages which should not be printed
# when the application is in production.
#
# INFO for messages similar to the "verbose" mode of many applications.
#
# WARN for warning messages which are logged to some log but
# the application is able to carry on without a problem.
#
# ERROR for application error messages which are also logged to some
# log but, still, the application can hobble along.
#
# FATAL for critical messages, after logging of which the application
# quits abnormally.
#
# ------------------------------------------------------------------------
#
#
# ------------------------------------------------------------------------
# Configure the logger to output info level messages into a rolling log file.
# ------------------------------------------------------------------------
log4j.rootLogger=DEBUG, C
#log4j.rootLogger=INFO, C
#log4j.rootLogger=WARN, C
#log4j.rootLogger=ERROR, C
#log4j.rootLogger=FATAL, C
#
#
# ------------------------------------------------------------------------
# Configuration for a rolling log file ("cinema.log")
#
# Control the maximum log file size
# Archive 2 log files
```

```
#
#   %n - newline
#   %m - log message
#   %p - message priority (FATAL, ERROR, WARN, INFO, DEBUG or custom)
#   %c - name of the category (logger)
#   %t - name of current thread
# ------------------------------------------------------------------------
log4j.appender.C=org.apache.log4j.RollingFileAppender
log4j.appender.C.MaxFileSize=100KB
log4j.appender.C.MaxBackupIndex=2
log4j.appender.C.ImmediateFlush=true
log4j.appender.C.Append=false
log4j.appender.C.File=${catalina.base}/logs/cinema.log
log4j.appender.C.layout=org.apache.log4j.PatternLayout
log4j.appender.C.layout.ConversionPattern=%-6p[%x] %c.%M, line %L:%n%m%n
#
#log4j.appender.C.layout.ConversionPattern=%-6p-%x [%t] %c.%M, line %L:%n %m%n
```

E.2 Tomcat Context.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>

  <WatchedResource>WEB-INF/web.xml</WatchedResource>

<Resource
name="jdbc/postgres"
auth="Container"
  type="javax.sql.DataSource"
username="u1"
password="u1"
defaultAutoCommit="true"
driverClassName="org.postgresql.Driver"
url="jdbc:postgresql://localhost:5432/postgres"
  maxActive="20"
maxIdle="10"
maxWait="-1"
removeAbandoned="true"
removeAbandonedTimeout="60"
logAbandoned="true"
/>

</Context>
```

E.3 Tomcat web.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

<resource-ref>
  <description>
Resource reference to a factory for java.sql.Connection
instances that may be used for talking to a particular
database that is configured in the server.xml file.
</description>

<res-ref-name> jdbc/postgres </res-ref-name>
<res-type> javax.sql.DataSource </res-type>
<res-auth> Container </res-auth>
</resource-ref>


<servlet>
<description></description>
<display-name>PoolTest</display-name>
<servlet-name>PoolTest</servlet-name>
<servlet-class>PoolTest</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>PoolTest</servlet-name>
<url-pattern>/PoolTest</url-pattern>
</servlet-mapping>

  ...


<session-config>
    <session-timeout>30</session-timeout>
</session-config>

<welcome-file-list>
<welcome-file>index.html</welcome-file>
```

```
</welcome-file-list>

</web-app>
```

Bibliography

[1] J. Borchers, O. Deussen, A. Klingert, and C. Knorzer. Layout rules for graphical web documents, computer graphics and the www. *IEEE*, 20 - 3:415–426, 1996.

[2] Jnetdirect Experts Software Components. http://www.jnetdirect.com.

[3] H. Garcia-Molina, J. Ullman, and J. Widom. *Database Systems - The Complete Book*. Prentice Hall, 2002.

[4] E Giguere. Databases and midp, part 1: Understanding the record management system. *Sun Microsystems - Technical Article and Tips*, 2004.

[5] Shay Horovitz. Location based services for mobile devices. *Embedded Computing Seminar*, 2006.

[6] http://forum.nokia.com. Location acquisition api specification, 2004.

[7] http://www.bouncycastle.org. The legion of the bouncy castle.

[8] http://www.commandprompt.com/home/. The postgresql company.

[9] http://www.devx.com. Devx.com.

[10] http://www.dotnetjunkies.com. The dotnetjunkies.

[11] http://www.javaworld.com. Java world.

[12] http://www.phptr.com/articles/. Wireless j2me platform programming.

[13] http://www.wikipedia.org. Wikipedia - the free encyclopedia.

[14] J. Hunter and W. Crawford. *Java Servlet Programming.* O'Reilly, 1997.

[15] Sun Java. www.java.sun.com.

[16] C. John and T. John. Metaphor and the cognitive representation of computing systems. *IEEE*, 1982.

[17] J. Muchow. *Core J2ME Technology.* Prentice Hall, 2001.

[18] J. Preece, Y. Rogers, and H. Sharp. *Interaction Design: Beyond Human - Computer Interaction.* Wiley Publisher, 2002.

[19] B. Schneier. *Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C.* Wiley Computer Publishing, John Wiley and Sons, Inc., 1996.

[20] P. Sestoft. *Systematic Software Test.* Department of Mathematics and Physics Royal Veterinary and Agricultural University, 1998.

[21] www.postgresql.org. Postgresql relational database system.

[22] J. Xiaoping. *Object-Oriented Software Development Using Java, Second Edition.* Addison Wesley, 2003.

[23] M. Yuan J. *Enterprise J2ME: Developing Mobile Java Applications.* Prentice Hall, 2003.