Development of Cross-Platform Mobile Application for Pernexus Systems

Morten Zobbe
s132436

Kongens Lyngby 2015

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary

The sales of smartphones and tablets have increased significantly over the past few years. As more and more people use their mobile devices in their everyday life, has the need for apps grown bigger and today hundreds of thousands of different apps are available in the app stores. Until recently, mobile developers had to use platform specific tools when developing for the different mobile operating systems. This meant that the developing process was costly if one wanted to develop an app for more than one platform. But with the use of new technologies, it is now possible to develop mobile apps for multiple mobile operating systems at a time.

This thesis investigates the different cross-platform technologies that makes it possible to develop for several mobile platforms at the same time. With the different technologies it is now possible to reuse the same source code for multiple platforms, which means that the development process is now faster and easier. However, there are advantages and disadvantages of the different technologies and these will be described in this thesis.

The thesis is prepared in collaboration with Pernexus Systems and the result is an app developed with the cross-platform technology PhoneGap. The development process with PhoneGap and the user experience of the app will be evaluated and compared to the other cross-platform technologies. Which app-development technology to use, shows to be dependent on whether you are willing to sacrifice some of the user experience for the benefit of having a fast and easy development process.

Resumé

De seneste år er salget af smartphones og tablets eksploderet. I takt med, at flere og flere folk bruger deres mobile enheder i hverdagen, er behovet for flere apps blevet større og i dag findes der hundredetusindevis forskellige apps. Indtil for få år siden, skulle udviklere bruge platformspecifikke værktøjer til at udvikle til de forskellige mobile operativsystemer, hvilket betød en meget ressourcekrævende proces, hvis man gerne ville udvikle til mere end én mobil platform. Men ved hjælp af nye teknologier, er det i dag blevet muligt, at udvikle mobile apps til flere mobile operativsystemer ad gangen.

Denne afhandling undersøger de forskellige cross-platform-teknologier, der gør det muligt, at udvikle til flere mobile platforme på én gang. Med de forskellige teknologier, kan man genbruge den samme kildekode til flere platforme, hvilket betyder at udviklingsprocessen både bliver hurtigere og lettere. Dog er der fordele og ulemper ved de forskellige teknologier, og disse vil blive beskrevet nærmere i denne afhandling.

Afhandlingen er udarbejdet i samarbejde med Pernexus Systems og har resulteret i en app, som er udviklet med cross-platform-teknologien, PhoneGap. Udviklingsprocessen med PhoneGap samt brugeroplevelsen af app'en bliver evalueret og sammenlignet med de andre cross-platform teknologier. Hvilken appudviklingsteknologi man skal bruge, viser sig at afhænge af, hvorvidt man er villig til at ofre en del af brugeroplevelsen, til fordel for en hurtigere og lettere udviklingsproces.

Preface

This thesis was prepared at DTU Compute in fulfillment of the requirements for acquiring an M.Sc. in Digital Media Engineering.

The thesis investigates the different cross-platform technologies used to make mobile applications. It looks into the advantages and disadvantages of each technology, to provide an answer to when a specific technology should be used.

During the thesis, a mobile application for three different platforms was developed, which is also documented in this report.

Lyngby, 15-June-2015

Morten Zobbe

Acknowledgements

I would like to thank the people at Pernexus Systems without whom it would not be possible to complete this thesis. Especially, I would like to thank Karsten Torp for invaluable advice and guidance during this project. Furthermore, I would like to thank Pernexus Systems for providing work laptop, software licenses, office space and a good work environment.

Additionally, I would like to thank my supervisor at Technical University of Denmark (DTU), Stig Høgh for guidance and supervision towards completing this thesis.

I would also like to thank my friends and family for continuous support and encouragement during the project period.

Contents

Abbreviations

API ..... Application programming interface

UI ..... User interface

SDK ..... Software development kit

RFID ..... Radio frequency identification

IDE ..... Integrated development environment

MVC ..... Model-view-controller

App ..... Application

Chapter 1

Introduction

Mobile applications have become a great part of our every day life. According to Gartner[18] mobile app downloads will have exceeded 268 billion by 2017. With an increasing global shipment of smartphones [17], it is fair to say that mobile apps have become very popular. Different mobile operating systems forces developers to develop apps for each platform they want their app to support. The most common mobile platforms are Android (78%), iOS (18.3%) and Windows Phone (2.7%) [14]. To support these different platforms a developer typically create native applications, which each is implemented in different programming languages:

Mobile OS	**Programming Language**
Android	Java
iOS	Objective-C / Swift
Windows Phone	C#

What this means is, that it is not possible to reuse the source code when developing the same app for different platforms. A native app provides the best user experience but requires a lot of experience and time to develop. User experience is important because the user expects an app to behave a certain way that is easy to use. If the app fails to do so, the user's experience of the app will be bad and he or she will unlikely use the app.

Over the past few years a new technology to develop apps has emerged to compete with native app development. The new technology is cross-platform app development, which main goal is to develop apps with native performance that works on as many platforms as possible. This thesis will focus on cross-platform app development for an IT company called Pernexus Systems.

1.1 Pernexus Systems

Pernexus Systems was started in 2008. The people behind Pernexus have more than 20 years of experience in working with utilities for construction and especially with construction projects with an infrastructure of pipes and cables. Today Pernexus Systems has about 7 employees and is a relatively small IT company, which main product is the development and maintenance of the web portal, `Entrepriseportalen.dk`.

1.1.1 Entrepriseportalen

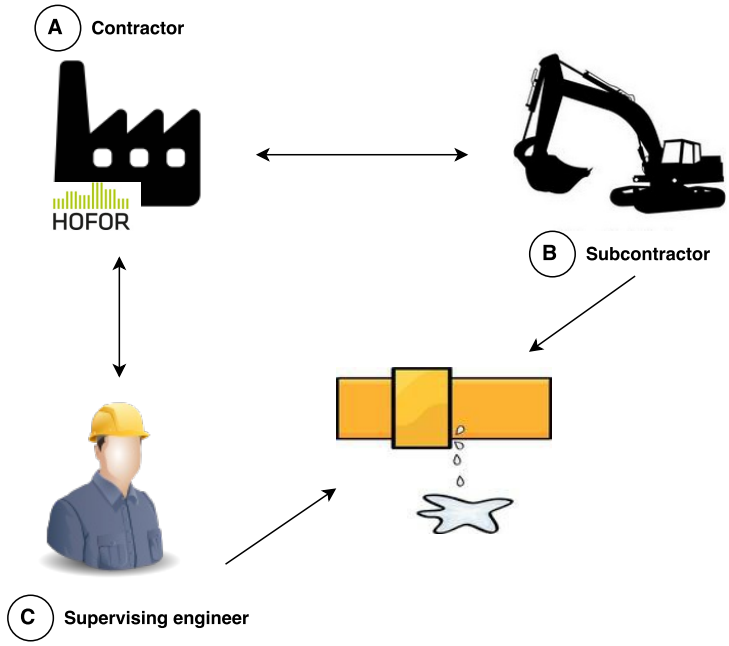
`Entrepriseportalen.dk` is a web portal for construction companies like HOFOR A/S and DONG Energy, who uses the web portal to manage their projects, which often have many subcontractors. Entrepriseportalen is used as a link between the construction company and their subcontractors to ensure that all parties of the project have the same updated blueprints, budgets and project schedules. The different parties of the project can also send invoices to each other through the web portal, while it makes sure that the government rules on document management are met. To get a better idea of how companies use Entrepriseportalen, an example will described in the following.

1.1.1.1 Example Usage of Entrepriseportalen

Figure 1.1 shows the following example:
Let us say that a water pipe has begun leaking. HOFOR (A) is the company in charge of maintaining the water pipes in Copenhagen and they create a new project in Entrepriseportalen for the leakage to be fixed. HOFOR then hire a subcontractor (B) to fix the leakage. The subcontractor does the excavation work to reach the broken pipe. The subcontractor has to fill out a form to document the excavation work. Then the subcontractor fixes the pipe and another form has to be filled out to document the work. A supervising engineer (C) is

hired by HOFOR to supervise and verify the work done by the subcontractor. The supervising engineer has to complete a form to document that the work is executed correctly by the subcontractor. The subcontractor then closes the hole and fills out a form that the work is done. When HOFOR has checked that the work has been done correctly they close the project.

Figure 1.1: Example workflow between parties in construction project.



By handling the forms and documentation in the Entrepriseportalen website makes sure that each party of the project has access to the relevant documents. HOFOR can follow each step of the project and make sure that the work is done correctly. Furthermore by having the documentation in Entrepriseportalen makes it possible for the parties to bill each other directly through the web portal.

1.1.2 EP Mobile

Pernexus Systems also offers a mobile application for Android to their customers. This app is called *EP Mobile* and it is designed to ease the process of filling out documents and forms when the workers are in the field. Before EP Mobile the supervisors had to fill out the different documents and forms by hand, which

they would later have to type into Entrepriseportalen. They would also have to attach photos from the construction site manually. An advantage of the mobile app is that they can complete the forms on the spot and attach the photos directly. Another major advantage of filling out the documents in a mobile application is that you can have the exact GPS location, which is very useful for these projects.

In reference to the example above, the app is very useful when the supervising is doing an inspection report of the work done. In EP Mobile the supervisor can choose the form he wants to fill out from a list of forms related to the project. When he chooses that he wants to fill out the inspection report, he is presented with questions about the work that have been done on the construction site. He can add pictures and GPS coordinates from the construction site to the document directly through the app. The questions on the report are defined by HOFOR themselves, to make sure they have the correct information, whether it is needed for their own archives or in order to pay the subcontractors for their work. When the supervising engineer has completed the inspection report, the report is automatically send to the project in Entrepriseportalen, where it can be viewed by the different parties involved in the project. The app also allows the user to see project details and already filled out documents on the go.

1.2 Thesis Problem

With an increasing customer base, more and more customers are requesting the EP Mobile app for iOS and Windows Phone. Pernexus Systems want to fulfill the wishes of the customers and develop EP Mobile for iOS and Windows Phone. The problem is that with only 2 full-time developers it would take almost all of Pernexus' time and resources to develop native apps for iOS, Android and Windows. Therefore, it was decided by Pernexus to make a cross-platform app that supports Windows Phone, iOS and Android. This app should eventually have the same functionalities as EP Mobile so it will not be necessary to spend time on maintaining two different apps.

1.2.1 Thesis Definition

The motivation for this thesis is to find what cross-platform mobile app technology that best fit Pernexus Systems' needs. A cross-platform app will be implemented using the chosen technology and will have to work on Windows Phone and iOS. The technology used will be evaluated and compared to other

cross-platform technologies. The knowledge gathered from this thesis can be taken into considerations for other companies that want to develop a mobile application for several platforms.

1.3 Report Structure

The structure of the report and the contents of the different chapters will be as follows:

1. **Introduction** - An introduction to the thesis problem.
2. **Analysis** - Description of prior work, definition of requirements for the cross-platform app, use cases and an analysis of current cross-platform app technologies.
3. **Design** - Description of the general architecture of the app and design of the user interface.
4. **Implementation** - Description of the implementation of the cross-platform app.
5. **Evaluation & Discussion** - An evaluation of the use of PhoneGap as cross-platform technology, discussion of what cross-platform technology to use and whether it is better to use cross-platform approach instead of a native approach.
6. **Conclusion & Future Work** - Findings and a conclusion on the project including future work.

Chapter 2

Analysis

This chapter will analyze the prior work done at Pernexus Systems leading up to this project. It will also conduct a domain analysis to introduce the key concepts that are relevant for developing a cross-platform app at Pernexus Systems. Functional and non-functional requirements will be defined and the scope of the project will be delimited to that set of requirements.

This chapter will also introduce and analyze the different mobile cross-platform solutions currently on the market. Pros and cons of these solutions will be presented and will be compared to the requirements to find the best matching cross-platform solution for Pernexus Systems.

2.1 Prior work

As introduced in Chapter 1, Pernexus Systems already offer an Android app for the customers, called 'EP Mobile'. This section will analyze the functionalities of EP Mobile and describe them in detail to give an understanding what the users can accomplish with the existing app.

2.1.1 EP Mobile

The EP Mobile app has been on the market since January 2011 and has over time and with constant feedback from the users, developed into including all the functionalities that meet the users' current needs and requirements. Hence, the cross-platform app will eventually need to have the same functionalities. Below are listed some of the main features of EP Mobile that the cross-platform app needs to be able to replicate. Note that this is not what the cross-platform app is required to be able to do within the scope of this project, due to the limited time frame. The scope of this project will be explained further in the requirements, Section 2.3.

EP Mobile allows the user to:

1. Login to Entrepriseportalen's server
2. Choosing nearby projects from the map based on current location - (Figure A.2)
3. Get a list of forms that can be completed, sorted by: - (Figure A.3)
   (a) Last accessed project
   (b) Name of project
   (c) Distance to project
   (d) Date the project was created
4. An overview of different projects where the user can access: - (Figure A.4)
   (a) General information about the project
   (b) Documents and images affiliated with the project
   (c) Already completed forms
5. Fill out forms - (Figure A.5)
6. Uploading images to forms
7. Connect to a RFID scanner via Bluetooth and scan RFID tags[1]

As mentioned, these features should eventually be implemented in the cross-platform app as well. Thus, to ease the process of development the cross-platform app can use the same principles for communicating with the Entrepriseportalen server.

[1]RFID (Radio-frequency identification) tags are used by some companies to identify different components in the ground.

As mentioned, the EP Mobile app has been under continuous development and Pernexus has listened to the feedback from the users, which has led to the user interface that can be seen in Appendix A. Since this user interface has been developed based on the feedback from the users, it will be used as a reference for the user interface of the cross-platform app.

2.2 Domain Analysis

In this section a domain analysis will be conducted to give the reader a better understanding of the processes and glossary used in the development of the cross-platform app. A domain analysis is "the process of identifying, collecting, organizing, and representing the relevant information in a domain based on the study of existing systems and their development histories, knowledge captured from domain experts, underlying theory, and emerging technology within the domain" [31].

The domain analysis will be based on the knowledge of the EP Mobile app, since the new cross-platform app will aim to have the same features as EP Mobile and follow the same design. The following describes the background information that has been gathered from the EP Mobile system. This information will be used to ease the development process of the cross-platform app.

2.2.1 Glossary

Company
A company can either be the main contractor who launches a project or a subcontractor who does work on a certain project. These companies all have employees that uses the Entrepriseportalen system.

User
A user is an employee in a company. The user has an account in the Entrepriseportal and can use the account in the app as well.

Project
A construction project is called a project. This could for example be to patch a leakage in a water pipe (See Example 1.1.1.1).

Form
A project can have many different forms depending on the type of project. To follow Example 1.1.1.1 the forms for this project could be: Begin excavation work, Work on water pipe and Completed excavation work.

Question
A question is a question on a form that the user can answer on the mobile device.

Answer
An answer is an answer to a question on the form.

2.2.2 General Knowledge of the Domain

User

- A user is one user of the mobile app with an account
- A user can have access to multiple projects
- A user can have access to multiple forms in a project depending on his rights

Project

- A project can have one or more affiliated users
- A project can have multiple forms
- A project can have multiple documents

Form

- A form can have one or more questions
- A form can have multiple images
- A form can have one or more sections
- A form can only have one header section

Question

- A question can only have one answer
- A question can be required, meaning the user must answer it

Answer

- An answer has a GPS position
- An answer has an answer time

- An answer can have a comment
- An answer can have an image
- An answer must be of one of the following types depending on the question:
  1. Yes or no (boolean) answer
  2. An integer number
  3. A decimal number
  4. Short text
  5. Long text
  6. A date
  7. A date and time
  8. A time
  9. A list of different options
- An answer can have a default value
- An answer can have a minimum and/or maximum value

2.2.3 Clients and users

The users of the app can either be supervisors or construction workers employed by a company. The supervisors have access to other forms than construction workers and vice versa. Both parties might want to access the project information pages in the app to get a quick overview of the project. However, the main use of the mobile client is to fill out forms, which they can do in the app while they are on the construction site. Other parties impacted by the system will be managers in the companies, who have an interest in the project process being as easy and quick as possible to optimize the work.

2.2.4 Environment

The users of EP Mobile must have an Android mobile device, since EP Mobile is only developed for Android. `Entrepriseportalen.dk` runs in all modern browsers such as Google Chrome, Safari, Firefox and Internet Explorer, which runs on modern desktop computers.

For the cross-platform app, Pernexus Systems wants it to run on iOS, Windows Phone 8 and Android allowing the customers to bring their own mobile device, whether it is a tablet or a smartphone. This enables the users to bring it to the field and fill out the needed forms.

2.2.5 Tasks and Procedures

When a user has completed a form, the supervisor will go through the form and approve it if its correct. Otherwise the user will be notified by the supervisor and will have to make changes to the form. As soon as a form is approved the project can move on to the next stage and the subcontractor(s) and/or supervisor(s) will be notified. When all of the forms has been approved in a project, the project supervisor can set the project as completed.

2.3 Requirements

The main problem of this project as described in Section 1.2, is to find the mobile cross-platform solution that fits the requirements of Pernexus Systems and apply this to make a cross-platform app. This will be described by the non-functional requirements. The scope of this project will be reflected by the functional requirements to what deemed fit to be able to implement during the time frame of this project.

2.3.1 Functional Requirements

Functional requirements for the mobile cross-platform app as well as the server will be defined, as the app will have to communicate with the Entrepriseportalen server.

2.3.1.1 Mobile app

MFR1
User must be able to login using the same username and password he/she uses for Entrepriseportalen.

MFR2
User must be able to fill out a form and submit it to the Entrepriseportalen server.

MFR3
The user must only be able to fill out the forms that he/she has the rights to complete.

MFR4
The mobile app must be able to present a list of forms, that the user has rights to complete, grouped by project.

MFR5
The user must be able to sort the list of forms from MFR4 either by distance to the project, name of the project, last accessed project or date of creation.

MFR6
Answers to questions on a form must be able to handle different data types as explained in Section 2.2.2.

MFR7
User must be able to attach a picture to a answer.

MFR8
User must be able to attach a picture to a form.

MFR9
An answer must have a GPS position.

MFR10
An answer must have an answering time.

MFR11
The mobile app should be able to save files to the local filesystem on the mobile device.

2.3.1.2 Server

SFR1
The server must be able to allow the mobile app to login using an access token.

SFR2
The server must be able to provide a collection of forms that the user has access to.

SFR3
The server must be able to save a form sent from the mobile app.

SFR4
The server must be able to handle image upload from forms.

2.3.2 Non-functional Requirements

NFR1
The app must be able to run on Windows Phone 8 or above, iOS 8 or above and Android API Level 15 or above.

NFR2
Maximum code for the app needs to be reused for all three platforms, expediting the developing process and maintenance of the app.

NFR3
The app should strive towards a native look and feel.

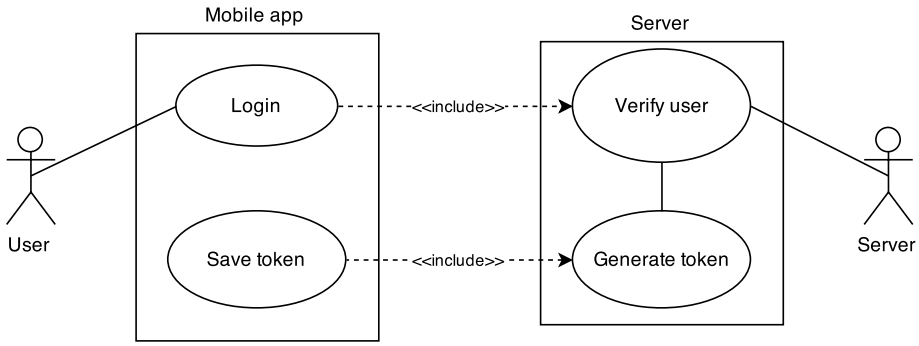
NFR4
The priority of implementing the app should be for Windows Phone, iOS and Android in that specific order.

NFR5
All developers at Pernexus Systems should be able to build the app from their laptop, ensuring that if one developer is absent the app can be built anyways.

2.3.3 Use Cases

The following use cases will give an overview of the features of mobile app. They describe how the users can interact with the app to send forms and describe the communication between the server and the mobile app.

Figure 2.1: Use Case Diagram 1: Authentication



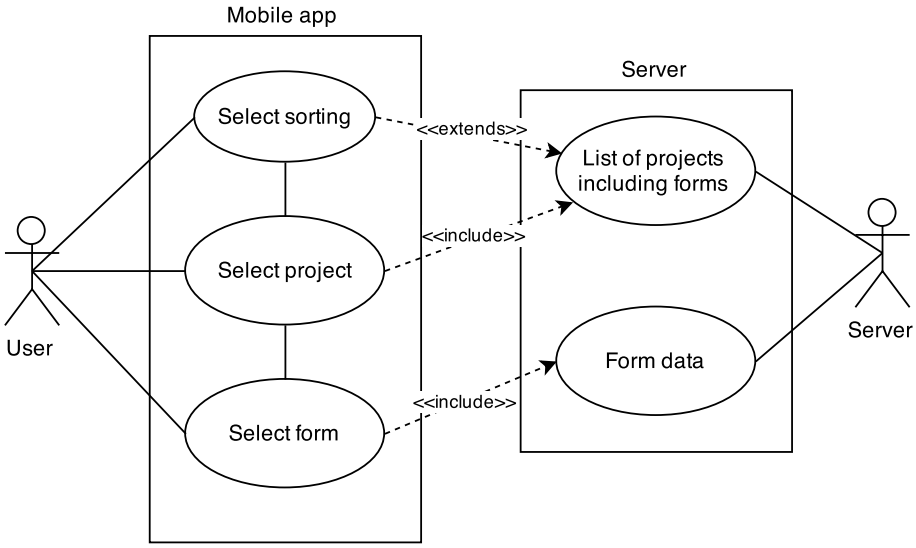
Use Case 1	**Authentication**
Primary Actors:	• User • Server
Brief:	All requests to the Pernexus server requires an authentication token. The token will be obtained by the mobile client when the user is authenticated through a login.
Postconditions:	The user is authenticated.
Preconditions:	None.
Triggers:	The user submits his/her username and password on the login screen.

Basic Flow:

1. The user is presented with a login page.
2. The user submit his/her username and password.
3. The credentials are verified by the server.
4. The server generates a authentication token.
5. The token is returned to the mobile app and saved.
6. The user is now authenticated.

Extensions:	3.a Invalid credentials: 1. Mobile client shows an error message

Figure 2.2: Use Case Diagram 2: Retrieve Form

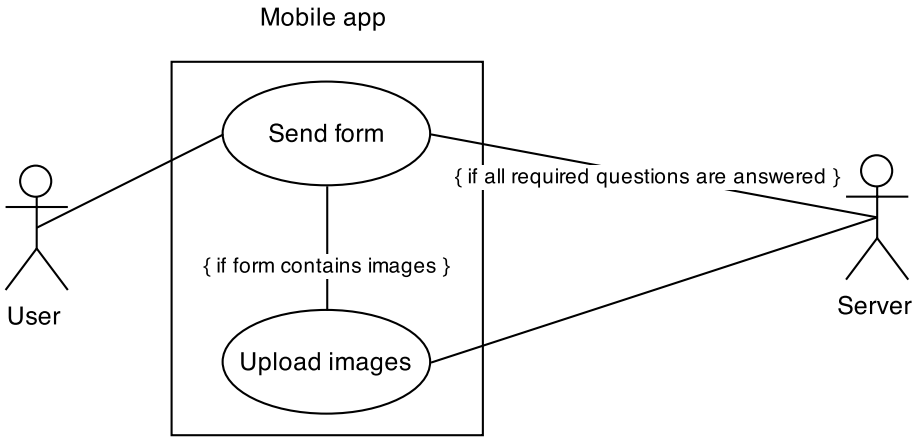


Use Case 2	**Retrieve Form**
Primary Actors:	• User • Server
Brief:	When the user wants to complete a form he will need to select the form from a list of available forms. These forms are grouped by projects. To get a list of projects the user selects how he want the projects sorted, either by distance, name, date of creation or when it was last accessed.
Postconditions:	The form with questions is presented in the mobile client.
Preconditions:	The user is authenticated.
Triggers:	The user selects a form from the list of forms.
Basic Flow:	

1. The user selects how the projects should be sorted.
2. The server generates a list of projects that has available forms, sorted by the sorting criteria.
3. From the list of projects the user selects the project that contains the form he wishes to complete.
4. The user is presented with a list of available forms on the project.
5. The user selects the form he wishes to complete.
6. Form data including questions are loaded from the server and presented to the user.
7. The user is now able to fill out the form.

Extensions:	None.

Figure 2.3: Use Case Diagram 3: Send Form



Use Case 3	**Send form**
Primary Actors:	• User • Server
Brief:	When the user has completed the form he/she can send it to the server by pressing the send button.
Postconditions:	The completed form is uploaded to the server.
Preconditions:	The user is authenticated.
Triggers:	The user presses the 'Send' button.

Basic Flow:

1. The user has filled out the form.
2. The user presses the 'Send' button.
3. The mobile app will send the answers to the server.
4. The mobile app will upload the images attached to the form.
5. The form is now sent.

Extensions:	2.a Not all required fields are filled out: 1. Mobile app shows an error dialog message. 2. Send the user back to step 1. 4.a Upload of images gone wrong: 1. Mobile app shows an error message.

2.4 Mobile Cross-platform Technologies

To find the mobile cross-platform solution that fits Pernexus Systems' requirements best, we need to look at what solutions are available on the market. This section will describe the three major categories of cross-platform app technologies, web apps, hybrid apps and compiled-to-native apps and the difference between these. Further detail about specific solutions within the three categories will be presented.

2.4.1 Web Apps

Web apps are applications that run in the Internet browser and are typically based on technologies such as HTML5, CSS and JavaScript. Web apps capitalize on the wide-spread browser support and standardization of the web technologies mentioned above. This standardization is also used in the web browsers on the different mobile devices, giving the opportunity for web apps to run on mobile devices as well as desktop computers. Through the use of different frameworks, web apps can simulate a native application look and through the HTML5 API it is possible to access different hardware functionalities, such as the camera and GPS [1]. Different user interface frameworks such as jQuery Mobile [21] and Sencha Touch [2] will optimize the user interface to fit the different screen sizes of the different mobile devices to give the app a native look.

The main advantage of web apps is that only one application needs to be coded and maintained and then it works on every platform. This means that the development of an app for multiple platforms is fast, since only code for one app is needed instead of three. Furthermore, web apps do not need to be installed since the app runs directly on the web, hence the user will always use the latest version. On the other hand, since the app only exists on the web it can only be used when there is an internet connection available.

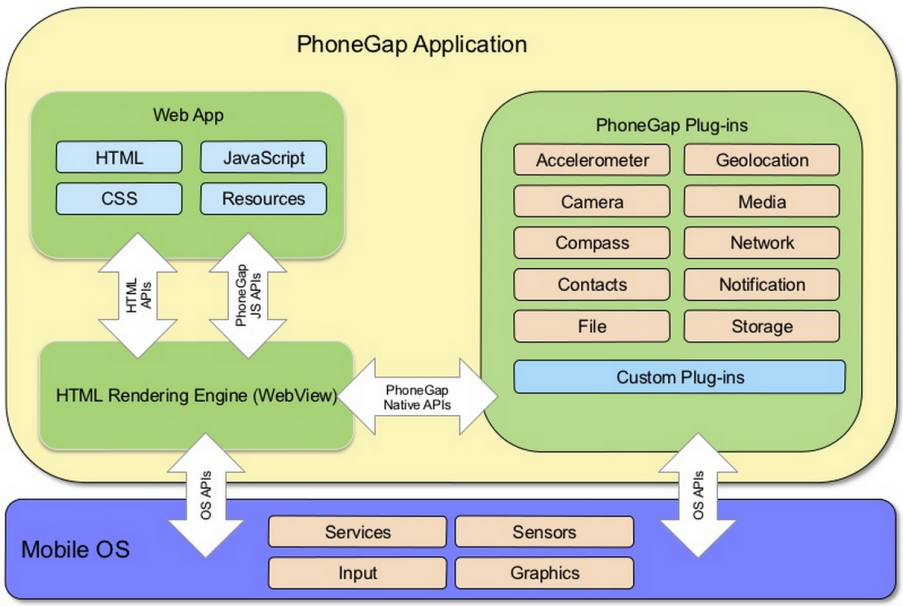
Another disadvantage of web apps is the extra time needed to render and download the web pages from the Internet, which will affect the user experience. Even though it is possible to use frameworks to optimize the look and feel of the web app and mimic native apps, studies show that the use of these frameworks will affect the application's performance and hereby also the user experience [30].

2.4.2 Hybrid Apps

Hybrid apps takes over where web apps left off. They combine features from web apps and native apps by embedding a web app inside a native web container (UIWebView in iOS and WebView in Android) that gives the web app access to the underlying platform's hardware features through different APIs. The web app inside this native container is developed like a normal web app but can make use of the mobile device hardware, which enhances performance. Therefore hybrid apps are developed almost the same as web apps and a detailed knowledge of the target platform is not required. Seeing that hybrid apps are capitalizing on the same wide-spread technology as web apps, they will run on almost any mobile platform. Just as for web apps, different frameworks for hybrid apps exist, to give the app a native look and feel.

Hybrid apps have all the main advantages as web apps. The same code works for every platform, which results in fast development and quick maintenance across all platforms. Another advantage of hybrid apps is that the contents of the app is loaded locally, which increases performance and makes it possible to use the app without an internet connection. The typical and most popular technology used to make hybrid apps is PhoneGap [4].

Figure 2.4: PhoneGap app architecture [5]



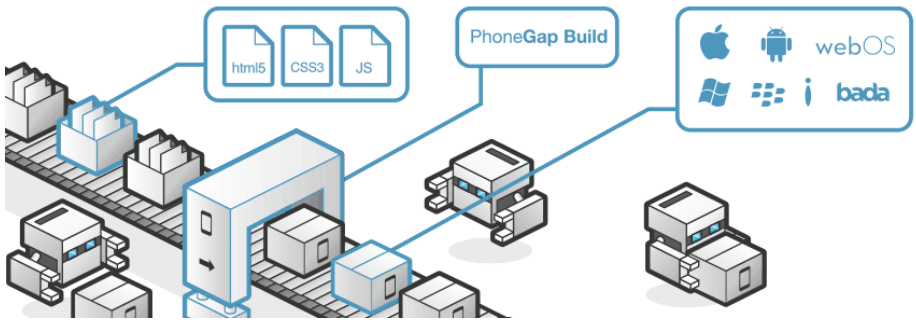
2.4.2.1 PhoneGap

PhoneGap was originally created by Nitobi, which was purchased by Adobe Systems in 2011. In 2012 the PhoneGap code was donated to the Apache Software Foundation under the name Apache Cordova. The Apache Software Foundation ensures that PhoneGap is free and open-source under the Apache License. Other software companies such as IBM and Microsoft contributes actively to the project and PhoneGap has been downloaded over 1 million times and is used by more than 400,000 developers, hence making the PhoneGap community very large and thousands of apps has been released in the different mobile app stores [3].

In Figure 2.4 is the main architecture of a PhoneGap application shown. As mentioned, the hybrid app consists of a web app inside a native web container. The web app can use different plugins provided by PhoneGap to access the mobile operating system and hardware. PhoneGap even lets you develop and use your own custom plugins to access the hardware of the mobile device.

PhoneGap offers a tool to build your hybrid app via a cloud service. This service is called PhoneGap Build and can from a single web app created in HTML5, CSS and JavaScript create apps for multiple platforms as shown in Figure 2.5. These apps can then be distributed to the different mobile app stores and installed like native ones.

Figure 2.5: Build apps for different platforms via PhoneGap Build [3]



A table of the supported platforms and their supported features are listed in Figure 2.6.

Figure 2.6: Table of the supported platforms and their supported features [4]

	iPhone / iPhone 3G	iPhone 3GS and newer	Android	Blackberry OS 6.0+	Blackberry 10	Windows Phone 8	Ubuntu	Firefox OS
Accelerometer	✓	✓	✓	✓	✓	✓	✓	✓
Camera	✓	✓	✓	✓	✓	✓	✓	✓
Compass	X	✓	✓	X	✓	✓	✓	✓
Contacts	✓	✓	✓	✓	✓	✓	✓	✓
File	✓	✓	✓	✓	✓	✓	✓	X
Geolocation	✓	✓	✓	✓	✓	✓	✓	✓
Media	✓	✓	✓	X	✓	✓	✓	X
Network	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Alert)	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Sound)	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Vibration)	✓	✓	✓	✓	✓	✓	✓	✓
Storage	✓	✓	✓	✓	✓	✓	✓	✓

2.4.2.2 Telerik Platform

Telerik Platform [6] is a commercial tool that can be used to make cross-platform apps. It is based on the PhoneGap technology but offers a wide range of extra features that will ease the development process and a mobile user interface powered by Kendo UI [7]. Telerik offers the tools to do:

- Rapid prototyping of the app through an app designer
- Build the app by drag and drop elements through an IDE
- Connect the app to your existing database
- Setup a test suite for the app
- Deploy the app to the different mobile platforms

Seeing that Telerik is build on the PhoneGap platform, it supports iOS, Android and Windows Phone as well. Though, one major problem with the Telerik Platform solution is that you buy into using their platform. You are restricted to use their tools and use only what they offer. For example, you will not be able to choose your own favorite UI framework and if new platform specific technologies emerges, you will have to wait for Telerik to support them.

2.4.3 Compiled to Native

Another technology used to create cross-platform apps is to make apps run natively on the mobile platform when they are compiled. These apps will be referred to as *'compiled to native'* apps in this report. Two of the most popular technologies that uses this approach is Appcelerator's Titanium and Xamarin, which both will be described in the following.

2.4.3.1 Appcelerator's Titanium

Appcelerator's Titanium [8] is a framework for writing native apps in JavaScript. The main philosophy of Titanium is that there are several mobile development APIs that can be normalized across platforms and be reused between platforms, but some UI conventions, platform-specific APIs and features should be implemented platform specific to provide the best user experience. It does not try to achieve "write once - run everywhere" like web and hybrid apps. Instead Titanium provides a JavaScript API with platform specific features, so the developer can take advantage of the platform specific user interface.

In order to develop apps with Titanium the developer has to install the tools and SDKs for the desired platforms just like when developing native apps. When these are installed the developer can use Titanium Studio, an Eclipse based IDE, to develop apps. The whole application is written in JavaScript and the native user interfaces are accessed through the Titanium API. Developing with Titanium Studio gives the developer the ability to debug the app directly through the IDE and to test the app in the different platform simulators. Titanium Studio will also build the final apps, which can be uploaded to the different app stores.

In a Titanium application, the JavaScript source code will be embedded as a string into a file in the native programming language (Java for Android and Objective-C for iOS) and will be evaluated at runtime by a JavaScript interpreter. The interpreter creates a bridge between the JavaScript code and the native code. For example, a JavaScript object from the source code will be created in the parallel native code.

One of the great strengths of Titanium is the look and feel of the app. The bridge between the platform specific UI components from the Titanium API and the native UI controllers, will give the app the same animations and behavior as a native app. Though, giving that the code for the user interface has to be platform specific means that only some of the code can be reused across

platforms, which will affect the time of development.

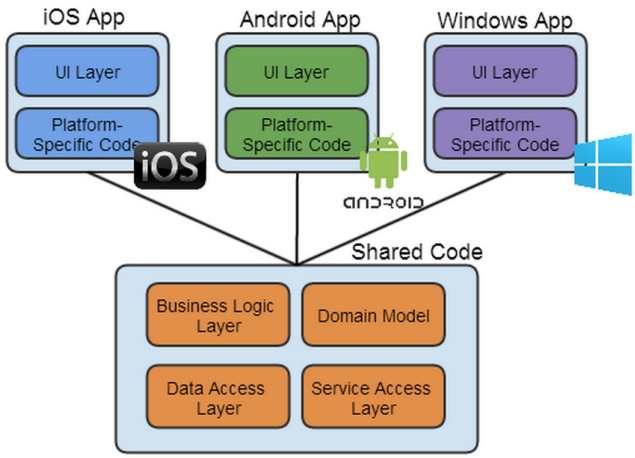
2.4.3.2 Xamarin

Xamarin[11] is another tool to build cross-platform apps that runs natively when compiled. Xamarin was created in 2011 by the same people behind the Mono project, an open-source implementation of Microsoft's .Net framework [10].

Xamarin offers mobile cross-platform solutions built entirely in C#. Just like for Titanium, the developer needs to install all the tools and SDKs needed to create native apps for the different platforms, in order to compile apps for these with Xamarin. When compiling apps for iOS in Xamarin they are compiled directly to ARM assembly code, which runs natively. On Android the code is compiled to native assembly code when the app is launched. This means that the cross-platform apps developed in Xamarin runs natively and the idea behind Xamarin is that anything you can do in a native developed app, you can do in a Xamarin app.

Just like Titanium, is Xamarin not a cross-platform developing tool used to develop "write once, run everywhere"-apps. As shown in Figure 2.7 only some of the code can be shared between platforms, while the rest is platform specific code and code for the user interface. According to Xamarin, on average 75% of the code can be shared between platforms [9].

Figure 2.7: Code sharing between platforms in Xamarin



Xamarin offers an IDE called Xamarin Studios that makes it easier to keep an overview of the source code for different platforms. It also allows the developer to debug the cross-platform app directly through the IDE and to design the app UI through platform specific tools. When the Xamarin app is compiled from Xamarin Studios is the result an application package, which can be deployed to the different mobile app stores.

Xamarin has several great advantages, seeing that it can access all of the underlying platform's API through native bindings. The native compilation will give Xamarin apps a high performance as well as the possibility to give the app a native look and feel. Another advantage is of course that it is possible to reuse a big part of the code across platforms. Furthermore developers can build on their existing skills in C# and do not have to know and learn Java and Objective-C. [13] [12]

2.5 Choosing the Right Cross-platform Technology

We have now learned about the mobile cross-platform solutions that are currently available on the market and the concepts of web, hybrid and compiled to native apps. To find the cross-platform technology that best meet the requirements stated in Section 2.3, we need to take a closer look at the features of each technology and their advantages and disadvantages.

In [30] and [28], comparisons of cross-platform solutions for mobile applications were conducted, though they did not include an evaluation of Telerik Platform and Xamarin. Findings from these papers and the knowledge from the previous sections will help us form a list of pros and cons for each technology, listed in the following.

2.5.1 Pros and cons

To get a quick overview of the advantages and disadvantages of each of the mobile cross-platform technologies a list of pros and cons has been made. These lists will also later assist to give an overview of which platforms meet the requirements for Pernexus' cross-platform app.

Mobile web app

Pros	Cons
• Supported by all platforms • Write once, run everywhere • Users always have newest version • Fast development • Easy to maintain • Debugging through web browser	• No access to all hardware features • Bad look and feel • Bad performance with many features

Mobile web apps are supported by almost every platform because each platform has its own native browser. This also means that code only has to be written once and it will run on every platform, which will speed up the developing time and ease the maintenance, since only one source code has to be maintained. The mobile web app is accessible through the web, which will ensure that the user will always run the latest app but will also require the user to have an Internet connection. The HTML5 API makes it possible to access most hardware

features such as camera, GPS and accelerometer, but not all hardware features are accessible with web apps. The authors' experiments with web apps in [30] showed that with a large amount of contents and animations, the performance of the app is bad, which will affect the user experience. As mentioned in Section 2.4.1 the web app can simulate a native look with use of different UI frameworks. But these will not give the same feel as the UI of a native app, because of slow animations and the lack of native UI elements.

PhoneGap

Pros	Cons
• Supports all major platforms • Write once, run everywhere • Access hardware features through plugins • Fast development • Easy to maintain • Apps built via cloud service	• No native feel • In some cases difficult to debug

PhoneGap supports all major platforms and since the app is built around a web app it has the same advantages of fast developing time, easy maintenance and only one source code that fits all platforms as a web app. Furthermore, it is possible to access the hardware of the underlying platform through different plugins. Via PhoneGap Build it is possible to build the apps for all platforms with one click. Even though the app is build around a web app, the authors in [30] found that the performance of a PhoneGap app is comparable to a native app. Though, the user will not be able to achieve the same feel as in a native app since the UI does not consist of native UI elements.

Telerik Platform

Pros	Cons
• Supports all major platforms • Write once, run everywhere • Access hardware features through plugins • Fast development • Easy to maintain • Apps built via IDE • Easy to test	• No native feel • Limited by the platform

Since Telerik Platform is using the same technology as PhoneGap it has a lot of the same advantages and disadvantages. Though, Telerik Platform provides an IDE that might make development even faster with its design tools and test suite. Just like PhoneGap and web apps the user will not have a native feel. Another downside is that the developer is limited by the platform. For example, when a new version of iOS or Android is launched, the developer will have to wait for Telerik Platform to support the newest version.

Titanium

Pros	Cons
• Native look and feel • Access hardware features through plugins • Easy to design through IDE • Easy to debug	• Does not support Windows Phone • Bad performance • Platform specific code for UI • Requires a lot of knowledge about the framework • Need all SDKs installed locally • Limited by platform

As previously described, the great strength of an Appcelerator Titanium app is that it can achieve a native look and feel by using native UI elements. Furthermore it can also access hardware features through different plugins. The IDE, Appcelerator Studio, makes it easy to design and debug the app. Though one major problem is that Titanium does not support Windows Phone. Compared to PhoneGap and web apps you will need to write platform specific code for the UI, which will affect the development time and the maintenance of the app. In [30] the authors experienced bad app performance when a large amount of content had to be handled. Furthermore. they experienced that Titanium requires a lot of framework specific knowledge in order to develop the app, which will affect developing time until a lot of experience has been gained. Just like Telerik Platform, the developer is limited by the platform features.

Xamarin

Pros	Cons
• Native look and feel • Access to all native features • Near native performance • Supports all major platforms • Easy to debug	• Platform specific code • Development time • Need all SDKs installed locally

Through native bindings, Xamarin can access the underlying platform, thus giving Xamarin apps the ability to utilize all native features and provide a near native performance. These native bindings also provides access to native UI elements, which will result in a native look and feel. Xamarin supports iOS, Android and Windows Phone and through the IDE, Xamarin Studios or Visual Studio, it is possible to debug the app. Just like Titanium the developer will have to write platform specific code to utilize the native features and the native UI. This will result in a increased development time and maintenance time, since different pieces of source code has to be maintained. Furthermore, a Mac OSX computer is needed to build apps for iOS and a Windows 8 PC to build apps for Windows Phone.

2.5.2 Compared to the Requirements

We now have an overview of the advantages and disadvantages of the different cross-platform technologies. To find what the best solution for Pernexus Systems is, we will now compare the pros and cons and the features of each cross-platform technology to Pernexus' requirements (see Section 2.3).

Table 2.4 shows which platforms meet the different requirements from Section 2.3. NFR1 states that the app must support Windows Phone, iOS and Android. Appcelerator Titanium does not support Windows Phone, thus we can exclude it from our list of potential candidates.
With Xamarin we can obtain a native look and feel, whereas we are only able to achieve a native look with use of UI frameworks for web, PhoneGap and Telerik apps. NFR3 describes that the cross-platform app must strive towards a native look and feel, giving Xamarin the advantage.
Because of the ability to achieve a native look and feel with Xamarin, means that some code has to be platform specific. Therefore will the same code not be used for all the platforms in Xamarin. Apps made with PhoneGap, Telerik and web apps uses the same code for all platforms. NFR2 claims that maximum

Requirements	Web app	PhoneGap	Telerik Platform	Appcelerator Titanium	Xamarin
	Technology				
Supports iOS, Windows Phone & Android	✓	✓	✓		✓
Native look and feel				✓	✓
Same code for all platforms	✓	✓	✓		
Access to GPS	✓	✓	✓	✓	✓
Access to camera	✓	✓	✓	✓	✓
Access to filesystem		✓	✓	✓	✓
Connect to Pernexus' server	✓	✓	✓	✓	✓
Upload images	✓	✓	✓	✓	✓
Locally build app to all platforms	✓	✓	✓		

Table 2.4: Technology features compared to the requirements.

code should be reused for all platforms, giving PhoneGap, Telerik and web apps the advantage in this matter.
All technologies are able to access the GPS, camera and filesystem, except web apps that does not have access to the filesystem. All technologies are also able to connect to Pernexus's server and upload images.
NFR5 states that all developers should be able to build the app from their laptop. From the PhoneGap Build cloud service it is possible to build the app for all platforms. Through the Telerik Platform IDE it is also possible to build the app for all platforms. But since Xamarin is binding code to native code the app must be built on a Mac OSX computer for iOS and on Windows 8 for Windows Phone, meaning the developer would need both operating systems to build the app for iOS and Windows Phone.

2.5.3 The Choice

We have now compared the requirements of the cross-platform app with the functionalities of the different cross-platform technologies. Based on the findings

we are ready to make a choice of what technology to use, to develop the cross-platform app.

Mobile web app
The mobile web app has its advantages when it comes to distribution and fast development time. But the user experience of a web app is very bad compared to the other technologies and cannot be ignored. Therefore will a web app not be the best choice of technology.

PhoneGap
PhoneGap apps uses the same source code for all platforms, which will result in a fast development and maintenance time. Furthermore, it is possible for all developers to build the app via PhoneGap Build, making it the prime candidate for technology to use. Though, it is not possible to achieve a native feel with PhoneGap, which might give the advantage to Xamarin.

Telerik Platform
Telerik Platform has all the same advantages and disadvantages as PhoneGap. With the use of Telerik's IDE it can even speed up the development process and make it easier to debug the app, making Telerik Platform another good candidate for the technology to use.

Appcelerator Titanium
The lack of support for Windows Phone has automatically excluded Appcelerator's Titanium from the considerations since it is a must.

Xamarin
Xamarin's ability to talk to the underlying platform's API is the greatest strength of Xamarin, which also will result in a native look and feel. But this is also a weakness of Xamarin, meaning a part of the code needs to be platform specific, resulting in longer development and maintenance time. Though, because of its ability to give a great native look and feel it will be a good candidate for the technology to use.

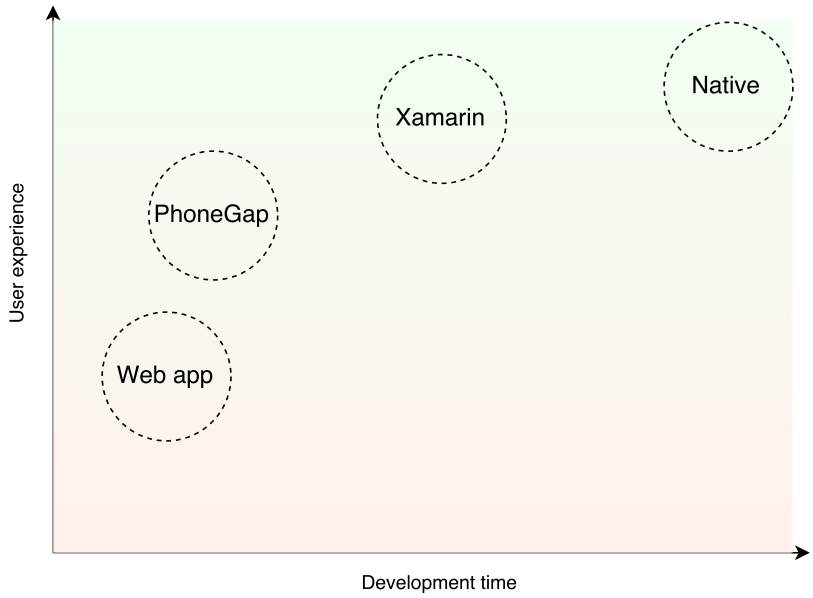
2.5.3.1 PhoneGap vs Telerik

As explained earlier, can the use of Telerik Platform's IDE speed up the developing process of the app compared to PhoneGap. Though, the same result can be achieved by using PhoneGap and at Pernexus they believe if the same thing can be achieved by using an open-source project (like PhoneGap) without any major cost, it should be used. Furthermore, Pernexus does not want to be limited by Telerik Platform's restriction, thus making PhoneGap the choice over Telerik Platform.

2.5.3.2 PhoneGap vs Xamarin

A big difference between PhoneGap apps and Xamarin apps is Xamarin's ability to achieve a high user experience with the use of native UI elements in the app. Though, the development time will automatically be longer for Xamarin apps, since some of the code has to be platform specific. Figure 2.8 shows an example of how the developing time for different technologies versus the user experience it can achieve. A native app can of course achieve the highest user experience, but with native bindings Xamarin apps are able to achieve almost the same. The lack of native UI elements will result in a lower user experience in a PhoneGap app. But with a good response time and a UI framework to make it look native, the user experience of a PhoneGap app will be acceptable. The low performance of a web app will automatically lead to a lower user experience. Note that Figure 2.8 is not based on any actual data, but considerations based on the knowledge from the previous sections.

Figure 2.8: Developing time vs. user experience for cross-platform technologies.



From this it seems that you are sacrificing user experience when lowering the development time. This might have a huge effect on the decision of which cross-platform technology to use depending on the kind of app you want to make. At Pernexus they are developing an app for business use and user interface is not

a top priority. Of course they want the app to be intuitive and easy to use, but most important is that the app works. Development time is therefore weighted higher at Pernexus than the extra user experience that Xamarin can achieve, making PhoneGap the choice of technology used to develop the cross-platform app for Pernexus Systems.

2.6 Chapter Summary

This chapter described the prior work done leading up to this project by describing the functionalities of the already existing app from Pernexus Systems, 'EP Mobile'. It then analyzed the domain to give the user a better understanding of the terms and processes used when developing the app. Requirements for the cross-platform app was specified and three use cases of the app was described. Existing mobile cross-platform technologies were analyzed including definitions of web, hybrid and 'compiled-to-native' apps. A comparison of the different cross-platform technologies with Pernexus' app requirements was done in order to find the best cross-platform solution for Pernexus Systems. PhoneGap was chosen as the solution that fit best.

Chapter 3

Design

This chapter will look into the design of the cross-platform mobile application developed during this thesis. We will look into the design of the existing mobile app from Pernexus Systems, 'EP Mobile' for Android. The general design of the cross-platform app will take basis in the design of the 'EP Mobile' app using the descriptions from the Domain analysis (Section 2.2).

As described in Section 2.4.2.1 a PhoneGap app consists of a single web application that works across several platforms. This web application is built using the *AngularJS* framework thus the structural design of the cross-platform app will have to follow the design of a AngularJS web application.

3.1 Structure of pages

As mentioned above the content of a cross-platform PhoneGap app will be developed as a web application. This web application will have different pages, which will form the different views in the final app. To fulfill the requirements from Section 2.3 the cross-platform app will need to have the following pages:

Login
The login page where the user is able to login. This page will have to communicate with the server for authentication.

List of forms
A page with a list of the forms the user has access to fill out. This page will have to communicate with the server in order to retrieve the list of forms.

Form
This page is the form with questions. It will have to communicate with the server to retrieve the questions on the form and to upload the answered form. The user should also be able to take pictures and upload these along with the form. Furthermore, this page will also have to access the GPS in order to attach the current location to the answers.

The already existing Android app 'EP Mobile' does communication with the server via REST calls, which we can reuse in the cross-platform app. Furthermore, the user interface design of 'EP Mobile' can also be reused for the cross-platform app. How both the server communication and the user interface can be reused, will be described in further detail in the following section.

3.2 EP Mobile design

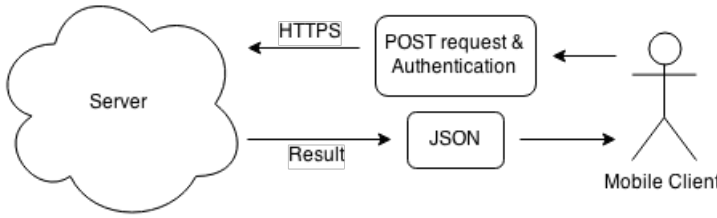
Seeing that Pernexus already has a fully functional Android app, has the design for communicating with the backend already been implemented. Since the app has been around for a few years, it has gone through an iterative process to meet the users' needs. The continuing feedback from the users of 'EP Mobile' has given Pernexus a great knowledge of how the general application design and user interface should be in order to give the best user experience.

3.2.1 Server communication

As mentioned earlier the current 'EP Mobile' app does communication with the backend server following a RESTful architecture design approach [26]. Figure 3.1 shows the overall architecture of the REST calls from the mobile client.

For example, to retrieve the list of forms, the mobile client does a POST call through HTTPS to the server that contains user credentials and the actual

Figure 3.1: The design of REST calls from the mobile client to the server.



request. The reason that Pernexus uses the POST method and not GET, is that there is no length restrictions on the POST method. There is in most cases a need to send a lot parameters with the request and these will be lost using the GET method, because of data length restrictions. Furthermore, we are sending sensitive data such as user credentials along with the request and this will not be cached when using the POST method as opposite to GET.
The server needs the user credentials in order to retrieve the list of forms that this specific user has access to. When the user has been authenticated on the backend, the result will be send back in JSON format, which will then be handled by the mobile client.

Using the REST calls will allow the user to:

- Login
- Retrieve the list of forms
- Retrieve the questions on the form
- Send the answers on a form

This approach will also be used in the cross-platform app and therefore no new implementation will have to be done on the server side to handle this.

3.2.2 User interface

As earlier mentioned, the 'EP Mobile' app has been used by the customers over the last few years. During this time the user interface (UI) of 'EP Mobile' has been under continuous development. At Pernexus they have listened to feedback from the users and changed the user interface accordingly. This knowledge of

what gives the best user experience when using the app, will be used for the UI design for the cross-platform app. Thus, the cross-platform app should aim to have the same UI as 'EP Mobile'. Appendix A contains some screenshots of the current UI in 'EP Mobile' that the cross-platform app will try to aim for in terms of UI design.

3.2.2.1 Home screen

Figure A.2 shows the screen that the user will see once he is logged in. From this view he can quickly access the list of forms by pressing pressing the Forms button. He can then choose how he wants the list to be sorted; either by distance, name, last accessed or by date.

3.2.2.2 Form list

Figure A.4 shows the list of forms. Forms are grouped by projects and the user is able to collapse each project to see the forms that he has access to fill out. The user can fill out the form by tapping the name of the form.

3.2.2.3 Form

Figure A.5 is the form view where the user can fill out the form. A form has a header section with some general questions and it can have sections of questions.

Figure A.6 shows the answer dialog that pops up when the user clicks to answer a question. From here the user can also add pictures to the answer.

3.3 AngularJS

AngularJS is a JavaScript framework, which main goal is to assist with creating single page application that only requires HTML, CSS and JavaScript. It is an open source project that is maintained by Google [32]. A great advantage of building a single page application is performance. An AngularJS application only has one HTML page and the content of this page is changed with the use of JavaScript. Therefore, the content is only loaded when needed and there is no need to load a new HTML page when navigating to a new page, which

optimizes performance. AngularJS utilizes different concepts to separate the different parts of the application to easy development. The main concept that AngularJS is the Model View Controller pattern.

3.3.1 MVC Design Pattern

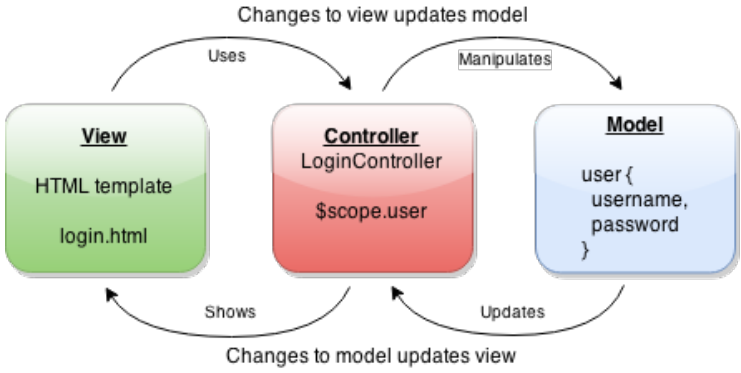
AngularJS uses the Model View Controller (MVC) design pattern to structure and organize applications and address the problems of reusability and scalability [29]. The MVC pattern consists of three parts:

Model
Represents the logical structure of data in an application. This could for example be a user as seen in figure 3.2. Here the model is a user that has a username and a password.

View
Represents the user interface - what the user sees and can interact with. To stay with the example above, it could be a login page with input fields where the user can enter his username and password.

Controller
Provides the link between the view and the model and handles the business logic. It ensures that changes to the view update the model and changes to the model updates the view; also known as two-way data binding. This means that when the user enter his username, the model is updated with the username from the view. This process is shown in figure 3.2.

Figure 3.2: How the model, view and controller updates in AngularJS.



As we can see from figure 3.2 the controller is keeping the view and the model separated. This means that no model is exclusively tied to a view, which proves useful when having an application with many different views, which all can use the same model. Furthermore, when developing for different platforms we might want different views depending on the platform (screen sizes, placement of buttons etc.) and since the model is not directly tied to the view we can use the same model for the different views.

3.3.2 Templates, Controllers & Services

As mentioned earlier, AngularJS is a framework for building single page applications. With use of JavaScript, the content of the page is changed programmatically by using templates. The application itself only consist of a single index.html page and the templates are loaded into this page as they are needed. The data that the user inputs in the templates needs to be handled accordingly to the business logic. This is done in the controller.

3.3.2.1 Templates

A template can be categorized as a view. This is what the user can see and interact with. The template is programmatically loaded into the application view when it is needed. For the cross-platform app in this project we will need to have a template for the user login, the list of forms and the form page itself. The templates are shown as green in figure 3.3.

3.3.2.2 Controllers

As earlier described controllers are the link between the view and the model. Furthermore it handles the manipulation of the data accordingly to the business logic. For example it could handle that the user input of a username from the view is converted to lowercase before updating it to the model. The controller also keeps an eye on the model and when the model is updated it makes sure the view is updated (as shown in figure 3.2).

3.3.2.3 Services

While controllers communicates and manipulates directly with the view, we might want some code that is abstract and we can re-use. This is what services in AngularJS can be used for. A service can be used in controllers through dependency injection meaning that several controllers can use one or many different services. For our cross-platform app this approach is a good idea to utilize, since we want to communicate with the server in many different views.

Figure 3.3: The architecture of the AngularJS application.

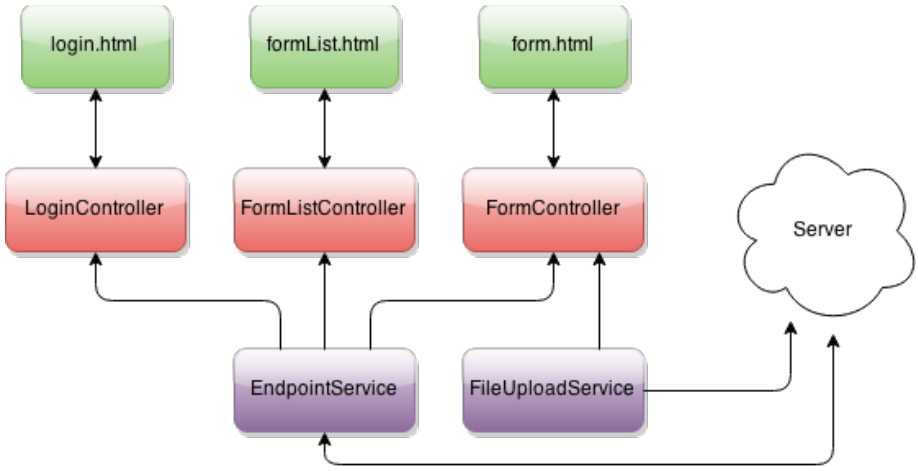


Figure 3.3 shows the overall architecture of the AngularJS web application for a PhoneGap app. It consists of three templates (green): a login page, a page to show the list of forms the user has access to and a page where the user can fill out the form. Each template has a controller (red) to handle the business logic. The application also has two services (purple). An EndpointService to communicate with the server and a FileUploadService that allows the user to upload pictures from a form to the server. The LoginController uses the EndpointService to authenticate the user through a login. The EndpointService does a REST call to the server with parametres defined in the LoginController. Also the FormListController uses the EndpointService to retrieve the list of forms. Again, the EndpointService does a REST call with parameters from the FormListController. The FormController uses both the EndpointService and the FileUploadService. The EndpointService is used to retrieve the questions on the specific form and to send the answers to the server when the form is

completed by the user. The FileUploadService is used to upload the pictures the user has taken while he was completing the form.

3.4 Plugins

As earlier described in Section 2.4.2.1, the PhoneGap app needs different plugins in order to access the platform hardware. In accordance with the requirements (Section 2.3) for this cross-platform app, the user should be able to take pictures and upload them along with the completed form. To take the pictures we need access to the camera using the Camera plugin that comes along with PhoneGap. The requirements state that there should also be a GPS location on the answers on the form and therefore we need to access to the phone's GPS. We can access the GPS by using the Geolocation plugin from PhoneGap. To upload the pictures to the server we can use the FileTransfer plugin from PhoneGap, which makes it easy to upload files. All these plugins are provided and maintained by PhoneGap to work cross-platform, which means that the code used in the PhoneGap app to access the plugin will be the same, independently of platform.

3.5 Chapter summary

This chapter described the overall design of the cross-platform app. It described how the app will be developed as a web application that works across iOS, Android and Windows Phone. The app will be developed using the AngularJS framework and using the MVC design pattern, which was described. The overall architecture of the AngularJS app that will run in the PhoneGap web container was described. In basis of the current Pernexus app 'EP Mobile', the communication with the server through REST calls was described and will be reused in the cross-platform app. Furthermore, the cross-platform app will aim for having the same UI as 'EP Mobile'. Lastly, the different plugins needed for the app was described.

Chapter 4

Implementation

This chapter will describe how the cross-platform application built with PhoneGap was implemented on the basis of the design in Chapter 3. It describes the setup of the application project that is needed in order to build the application for the different platforms via PhoneGap Build. It describes the user interface framework that was used to give the app a native look. Furthermore, this chapter will describe the process of how the app is handling the communication with the server, both in terms of REST calls and file upload. It will also describe how the app is using hardware functionalities such as camera and GPS via plugins.

4.1 PhoneGap setup

In Chapter 3 we learned that a PhoneGap app is built as an AngularJS single page web application. As described in Section 3.3.2, we make use of templates, controllers and services to ease the process of development. This means that during development we will have many different HTML pages (templates) and many different JavaScript files (controllers and services). But the final app will only consist of a single web page - index.html. In the following the process of how we go from having many HTML pages to a single page application for each platform.

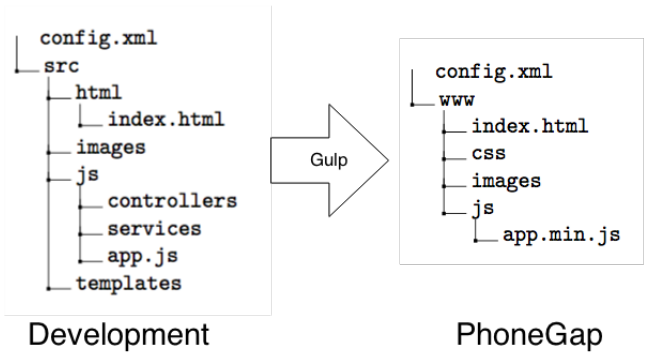
4.1.1 Project structure

Figure 4.1 shows the overall structure of the app project. On the left is the project structure while the app is in development. The `config.xml` file handles the configuration of the project. It tells PhoneGap which plugins to use, the version number and define the image files for the app logos and splashscreens. We can also define different configurations such as app orientation and HTTP access control, which is very useful since we need the app to access Pernexus' servers.

The `src` folder contains the files that we modify to build the cross-platform app. In the `html` folder is the `index.html` file that is the single page that loads the content from the templates as needed. The `index.html` is set up to use the `app.js` script from the `js` folder and use the CSS from the UI framework (will be explained further in Section 4.2). From the `app.js` script the `index.html` knows what template to use when a link is pressed. For example, a link to `/login` will make the `index.html` use the `login.html` template.

Each template uses a controller. These are located in the `controller` folder. A controller can make use of a service and these services are located inside the `services` folder.

Figure 4.1: Project structure from development to PhoneGap.



This structure eases the development since we have a file for each of our templates, controllers and services. Using designated folders for these files gives a great overview of the different files that is in the project and makes it easy and fast to do the implementation.

4.1.2 Gulp

Gulp [19] is a build tool that is used to develop larger pieces of software out of many small pieces. What we can use this for in this project, is to put together a single JavaScript file that contains all of our controllers, services and templates.

In figure 4.1 on the right we can see the structure of the project after is has been built with gulp. What is unchanged is the `config.xml` that stays in the same directory. The images and the `index.html` has been moved to the `www` folder, which is now using the newly built `app.min.js` script instead of `app.js`. As mentioned the `app.min.js` script contains the templates, controllers and services, but it also contains the `cordova.js` script, which makes it possible for the web application to talk to the native web container on the mobile device.

Furthermore, there is a `css` folder inside the `www` folder. Gulp also makes sure that the CSS from the UI framework, which will be explained in the next section (Section 4.2), is copied to the built project to give the app a native look.

4.1.3 PhoneGap Build

One of the main reasons we chose to use PhoneGap was that they offer the feature to build the app for the different platforms in the cloud. For this, PhoneGap offers a solution named PhoneGap Build [23]. The tool is very simple to use. You simply upload a zip-file containing the project files built with gulp (the files on the right side in figure 4.1) and PhoneGap will build the app for each platform. PhoneGap uses different definitions from the `config.xml` to configure the built app, such as which plugins to include. It only takes a few seconds for PhoneGap Build to process and build the apps for the three different platforms. As soon as they are built they can easily be downloaded from the website and be deployed to the different app stores.

4.2 UI Framework

As earlier mentioned in Section 2.4.2, it is possible to obtain a native look in the app with the use of a mobile user interface framework. A mobile UI framework consists of HTML5, CSS and JavaScript and gives the application the ability to use animations and gestures to give the app a native feel, and also give the app a native look. The UI framework detects the device and screen size and

optimizes the look for the given device. This is extremely useful when developing cross-platform since mobile devices for all platforms have different screen sizes.

4.2.1 MobileAngularUI

For this app the MobileAngularUI framework [22] was used. While other mobile UI frameworks such as ionic [20] and jQuery Mobile [21] are more well-documented and further developed, they do not support Windows Phone, thus they cannot be used for this project as by Pernexus' requirement the app must support Windows Phone. MobileAngularUI combines elements from AngularJS and Twitter Bootstrap [24] and makes it possible to use native looking elements such as:

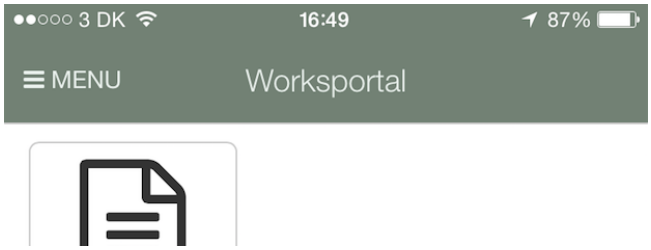
- Navigation bars
- Side menus
- Buttons
- Tab navigation
- Input fields
- Dropdown menus
- Modals

We use these elements by simply using a pre-defined class in the HTML. Listing 4.1 shows how we use the classes `navbar` and `navbar-app` to make a navigation bar in the app. The `navbar-absolute-top` places the navigation bar at the top. Within the `div` with the `navbar-brand` class is the navigation bar title. `navbar-brand-center` places the title in the center of the navigation bar. The result of this can be seen in Figure 4.2.

Listing 4.1: How the navigation bar is implemented with MobileAngularUI.

```
1 <div class="navbar navbar-app navbar-absolute-top">
2   <div class="navbar-brand navbar-brand-center">
3     Worksportal
4   </div>
5 </div>
```

Figure 4.2: The navigation bar from MobileAngularUI.



4.3 Templates

As mentioned, the `index.html` loads the content of the app as it is needed. Listing 4.2 shows the body of the `index.html` file. The `ng-app` tag tells AngularJS to use the PernexusMobile module and the `ng-controller` points AngularJS to use the MainController. These are both defined in the `app.js`. Inside the `app-content div` on line 6 is where the template is loaded and as mentioned the app knows what template to use from the `app.js`.

Listing 4.2: `index.html` loads templates through `ng-view`

```
1  <body ng-app="PernexusMobile" ng-controller="
       MainController">
2    <div class="app">
3    <!-- App Body -->
4        <div class="app-body">
5          <div class="app-content">
6            <ng-view></ng-view>
7          </div>
8        </div>
9      </div>
10 </body>
```

Listing 4.3 shows an example of the Login template - the page where the user login. In line 1 we can see that the template is using the controller “LoginCtrl”. The different classes “scrollable”, “scrollable-content section” and “form-group” are defined by our UI framework to give the page a native look. Inside the input on lines 7-10 we are using `ng-model` to bind the data in the template to the controller. This means that when the user inputs his username it will get bound to the data model as explained in Section 3.3.1. This way the controller can handle the input data from the user.

Listing 4.3: From the `login.html` template

```
1  <div class="scrollable" ng-controller="LoginCtrl">
2      <div class="scrollable-content section">
3          <form role="form" ng-submit='login()'>
4              <fieldset>
5                  <div class="form-group">
6                      <label>Brugernavn</label>
7                      <input type="text"
8                             ng-model="user.username"
9                             class="form-control"
10                            placeholder="Brugernavn">
11                 </div>
12                 ...
```

4.4 Controllers & Services

Listing 4.4 is an example of the controller used for the login process. It is using the module "PernexusMobile.controllers.Login", which is included in the `app.js` so the app knows how to reference the controllers inside this module. `['PernexusMobile.services']` means that the "Pernexus.services" module has been injected and we can use the different services from this module. In line 2 we define the controller with a name, "LoginCtrl" and to use `$scope` and the "EndpointService" from the "Pernexus.services" module. `$scope` is what bind the data between the controller and the template. As we can see in line 3 the `$scope.user` is defined and in Listing 4.3 line 8 we reference to this user with `ng-model`.

In line 9 of Listing 4.4 we define the login function that is used in the template when the user presses the login button. The login process is handled by a REST call in the EndpointService, which is given the parameters with username and password and a URL to where to do the REST call. This will be explained in further detail in the following section.

Listing 4.4: From the `LoginController.js` controller

```
1  angular.module('PernexusMobile.controllers.Login', ['
       PernexusMobile.services'])
2      .controller('LoginCtrl', function($scope,
           EndpointService) {
3        $scope.user = {
4                    username: '',
5                    password: ''
6              };
7
8          //"Login function"
9          $scope.login = function() {
10           var param =
11             {
12                 loginName: $scope.user.username.trim(),
13                 password: $scope.user.password.trim()
14             };
15
16             EndpointService.doPost( param, "usercheck/
                   login", function(data) {
17               //"User has successfully logged in"
18               ...
```

4.5 REST calls

In order for the user to login, retrieve forms and send forms, the app needs to communicate with the Pernexus server. This is done, as earlier mentioned through REST POST calls. These POST calls are called from the `EndpointService.doPost()` function in the "EndpointService", which is shown in Listing 4.5.

The function takes the arguments: `data`, `domain`, `successCallback` and `errorCallback`. `data` is the data that will be send with the POST request. The `domain` is the domain on the server where we will do the request. `successCallback` and `errorCallback` are functions that will be called whether the request is a success or gives an error. To do the POST call we need a URL to the server. In line 2 we get this from the "EndpointConfigService". The `domain` is then attached to the server URL to get the full URL of where to do the request. We then use the built-in `$http` service from AngularJS. If the call succeeds and there is a result we call the `successCallback` function on the returned data. If the call does not return a result the `errorCallback` function will be called.

Listing 4.5: The doPost function in `EndpointService` to call POST methods to the Pernexus server.

```
1  service.doPost = function(data, domain, successCallback,
       errorCallback) {
2    var url = EndpointConfigService.getUrl();
3    url = url +"/"+domain;
4
5    $http.post(url, data)
6    .success(function(data) {
7      if( data.result )
8        {
9          successCallback(data);
10       }
11     else{
12         errorCallback(data);
13     }
14   })
15   .error(function(data, statusText) {
16     ...
```

4.5.1 Login

Listing 4.6 shows how the `LoginController` uses the `doPost` function to login the user. The `param` contains the username and password. The domain where we do the login request is described in the second argument. The third argument is our success callback function. When the POST call succeeds the returned data from the request, will contain a token that will be saved locally on the mobile device along with the username. The login is saved using the `UtilsService.store()` function so the app automatically will login the next time the user launches the app. The `window.location` in line 10 will then load the `home` template that the link is referring. The fourth argument that `doPost` takes is the error callback function, which will call the logout function to clear the configuration file that was saved locally on the device.

Listing 4.6: How the `doPost` function is used to login the user.

```
1  EndpointService.doPost(param, "usercheck/login", function
       (data) {
2    var token = data.token;
3    var config = {
4      "userName": param.username,
5      "userToken": token
6    };
7
8    EndpointService.setCredentials(param.username, token);
9    UtilsService.store(config);
10   window.location = "#/home";
11
12   }, function(data, status){
13     //"Invalid credentials - signing out to clear config"
14     service.logout();
15   }
16 });
```

4.5.2 Forms

The `doPost` function is also used to retrieve and send forms. To get a list of the forms we simply set parameters of how the returned forms should be sorted, either by name, date or distance to the project, and how many forms we want returned. These parameters are sent with the POST request and the device will be returned a list of forms.

When the user wants to fill out a specific form, the app has to load it from the server. Again we use the `doPost` function along with parameters to identify the form, such as form ID. These forms all have questions that all have an ID. This ID is used when the app sends the completed form back to the server. The completed form is sent to the server including parameters that identify the form (form ID) and an array of answers that includes the ID of the question in order to pair the answer with a question.

4.6 Using plugins

As mentioned in Section 3.4 we need to use different PhoneGap plugins in order to access the hardware functionalities on the mobile device. To meet the requirements of the app we needed to use plugins for the camera, the GPS and

to transfer files to upload images to the server. As mentioned earlier in this chapter, we simply define in the `config.xml` which plugins we want to use and they will be available.

4.6.1 Camera

Listing 4.7 shows the code used to add an image to a question on the forms. We use the `navigator.camera.getPicture` function to open the camera (line 11). The function takes a success function, an error function and options. If the camera succeeds taking a picture it will be added to the array of photos for that question (line 12). We set the options in a variable to control different camera options such as encoding type and image quality.

Listing 4.7: Add image with camera plugin.

```
1  service.addImage = function addImage(callbackFunction) {
2    var options = {
3      quality: 75,
4      sourceType: Camera.PictureSourceType.CAMERA,
5      targetWidth: 1920,
6      targetHeight: 1920,
7      encodingType: Camera.EncodingType.JPEG,
8      correctOrientation: true
9    };
10
11   navigator.camera.getPicture( function(imageData){
12       question.photos.push(imageData);
13     }, function(errorMessage){
14     //"Handle camera error"
15     }, options );
16 }
```

4.6.2 Geolocation

To get the GPS location we use the geolocation plugin. As seen in Listing 4.8 the plugin has a function called `watchPosition` which is called every time the GPS location changes. It has a success function and an error function just like the camera plugin. When a new GPS location is detected the latitude and longitude is saved in a variable and we will always have the latest location of the user.

Listing 4.8: Get GPS position with geolocation plugin.

```
1 navigator.geolocation.watchPosition(function(pos) {
2                 position.x = pos.coords.latitude;
3                 position.y = pos.coords.longitude;
4             }, function (posError) {
5                 //"Handle Geolocation error"
6             });
```

4.6.3 FileTransfer

To upload the images to the server we can use the FileTansfer plugin provided by PhoneGap. In Listings 4.9 is the code used to upload an image to the server. In line 1 we simple make a new `FileTransfer`. From our `EndpointConfigService` we get the URL of where the images will be uploaded to and save it in the `endpoint` variable. We initiate the upload using the `upload` function with the arguments of the image URI, the endpoint URL and a success function.

Listing 4.9: Upload image to

```
1 var ft = new FileTransfer();
2 var endpoint = EndpointConfigService.
      getFormPhotoUploadUrl();
3 ft.upload(fileURI, encodeURI(endpoint), function(metadata
      ) {
4   $log.info("Image successfully uploaded.");
5 })
```

4.7 Platform specific problems

As described in Chapter 2, the major advantage of PhoneGap is, write once - run everywhere; the same code works on every platform. But during the implementation of the PhoneGap app this showed not to be completely true.

4.7.1 Windows Phone

As the main focus at first was on developing the app for Windows Phone a few platform specific problems arose during the implementation. Testing the app during development showed us that some things that worked in the iOS and Android app, did not work when the app was deployed to Windows Phone.

4.7.1.1 Debugging

Especially live debugging of the app was a major problem on Windows Phone. There is no way of doing live debugging of a PhoneGap app on Windows Phone and when a error arose there was no way of knowing where it was in the code. Android Developing Studio comes with a tool called LogCat [25], which let you monitor the apps running on the device and in the iOS simulator you can do the same. But as mentioned, some code would run on iOS and Android but would fail to run on Windows Phone, which led to many hours of sifting through the code to find the errors.

4.7.1.2 UI

Some UI elements did not work properly on Windows Phone and had to be optimized for fitting Windows Phone. This was mainly due to the UI framework not being optimized for Windows Phone and some of the CSS had to be changed in order for it to work. The PhoneGap app is running in a web container, meaning that the app runs in a Internet Explorer container on Windows Phone. Some of the CSS in AngularMobileUI is not optimized for Internet Explorer and some of it needed to be changed in order to work properly.

Furthermore, AngularMobileUI is using Font Awesome to draw icons. On Windows Phone 8 these icons showed up as squares and again a fix in the CSS had to be made in order to get the icons to show. MobileAngularUI is not optimized for Windows Phone, which resulted in the different UI elements being small and difficult to see and press.

4.7.2 iOS

Deploying for iOS also led to unforeseen obstacles. To build the app for iOS in PhoneGap Build some developer certificates are needed and have to be uploaded to PhoneGap Build. These certificates can only be obtained from XCode on Mac, meaning that you will need a Mac to be able to build the app for iOS.

When you want to submit your finished app to the AppStore this must be done with an application called ‘Application Loader’, which can only be found on a Mac running OS X. Again, to submit an app built with PhoneGap Build requires a Mac.

4.8 Chapter summary

This chapter described the process of implementing and building the cross-platform app with PhoneGap. It described how the project should be structured in order for the app to be built in the cloud with PhoneGap Build. It also described how MobileAngularUI was used to give the app a native look. The use of AngularJS and its use of templates, controllers and services for this app was explained. How the communication via REST calls was done is also described in this chapter. It covers how we make use of different PhoneGap plugins to access the camera and GPS. Lastly, some of the platform specific problems that arose during the implementation was described. Screenshots from the final app can be seen in Appendix B.

Chapter 5

Evaluation & Discussion

This chapter will discuss the process of developing cross-platform apps with PhoneGap. Furthermore, it will be discussed whether PhoneGap was the right choice of technology to use compared to other cross-platform technologies. The question of it is worthwhile to develop cross-platform apps compared to native apps will also be discussed, as well as the future of cross-platform technologies.

5.1 Evaluation of PhoneGap

This section will focus on an evaluation of using the PhoneGap technology to make cross-platform mobile applications. The evaluation is based on the experience gained during the implementation of the PhoneGap app described in the previous chapter.

The evaluation will focus on the ease of development including the whole setup needed to develop apps with PhoneGap. It will focus on the features that are available when developing with PhoneGap including the user interface. Furthermore, the performance of the app will be evaluated along with development and maintenance time. The distribution of the PhoneGap app to the different app stores will also be evaluated as well as the opportunities for further development of the app.

Development environment

To set up the PhoneGap development environment is quite easy. You only need to follow a few steps in order to be ready to develop. To setup PhoneGap with Gulp along with MobileAngularUI (see Section 4.1) requires only a few more steps. Most IDEs offers plugins for code completion for PhoneGap's API. Building the app for all platforms was extremely easy with PhoneGap Build and no need to install the different platform SDKs was necessary.

Ease of development

To develop in PhoneGap does not require advanced knowledge, seeing that the app is developed in HTML and JavaScript. Compared to Java, Objective-C and C# that is used to develop native apps, is HTML and JavaScript fast to learn and implementation can start almost right away. However, some knowledge about how an AngularJS application is implemented and how to use the UI framework properly is still needed. PhoneGap offers a starter project, which is a great starting point for development.

In the Analysis (Chapter 2) it was described that a PhoneGap app was "write once, run everywhere", meaning that when the app was implemented it would run on each of the desired platforms. Though, we learned from Section 4.7 that this is not the whole truth. During implementation a lot of Windows Phone-specific problems arose, which led to many hours of debugging and optimizing the CSS for the app to be able to run on Windows Phone. However, when these problems were fixed the same code would run on every platform.

Speed and cost of development

The fact that it is easy to start implementing a PhoneGap app will of course increase the speed of development. That the same code runs on each platform is also a huge benefit, if not the biggest, to the speed of development. Though, as explained above and in Section 4.7 during implementation some platform specific problem arose. These problems, and the fact that it is difficult to debug a PhoneGap app on Windows Phone, had a large negative effect on the development time for this project.

Maintainability

Only one code has to be maintained for all three platforms, which means that code for all three platforms is easy to maintain. Consequently, the time spend on maintaining all three platforms is much less compared to having three different source codes.

User interface

Even with the use of a UI framework that gave a native look, it was not possible to achieve a native feel of the app. Gestures can be enabled

using the UI framework but they do not feel native. MobileAngularUI was chosen as the UI framework because it supports Windows Phone. More evolved UI frameworks such as jQuery Mobile and ionic was the first choice, but since they were not compatible with Windows Phone, MobileAngularUI was used.
As explained in Section 4.7 there were some problems getting the UI to work properly on Windows Phone, which has to be fixed by overwriting the CSS from the UI framework. Furthermore, when the same code is used for all three platforms they will have almost the same look. That is one major problem when going for a native look. A native look on Windows Phone is much different from a native look on iOS. This means that the user will not experience a native feel of the app.

Access to device-specific features
PhoneGap provides an easy to use API to access device-specific features. All major hardware features are available in PhoneGap and it is even possible to create your own plugins to talk to the underlying platform.

Application speed
There is no noticeable difference between the performance of the PhoneGap app and a native app. The app opens fast and responds smooth to user interactions like a native app.

Distribution
Another unforeseen problem encountered during the implementation was that a Mac is actually needed to build PhoneGap apps for iOS. PhoneGap claims that the app can be built from the cloud service PhoneGap Build and installing the SDKs for the different platforms is not necessary. But as explained in Section 4.7, if the app has to be built for iOS some certificates has to be uploaded to PhoneGap Build. These certificates has to be obtained from a Mac OSX computer, meaning a Mac is needed for developing iOS apps with PhoneGap. However, when these certificates are uploaded, all developers with access to PhoneGap Build can build apps for all platforms. Though, when the iOS app has to be deployed to the App Store it has to be done through a tool for Mac OSX called Application Loader, again making a Mac computer a necessity. This problem was solved by purchasing a Mac Mini, which was used for the deployment of the app to the iOS App Store.

Opportunities for further development
Seeing that the PhoneGap app is developed like a web app makes it easy to expand further. New pages just needs to be added to the project and link to these on the already existing pages.

To sum up, once you get over the hurdle of fixing the platform specific problems,

the development with PhoneGap is fast and it is easy to start implementing the app right away. A drawback with PhoneGap is that it not possible to achieve a native feel, though as explained earlier this was not a problem for this project.

5.2 PhoneGap vs. Other Cross-Platform Technologies

In this section the PhoneGap approach is compared to other mobile cross-platform technologies. PhoneGap might have seemed to be the best choice at first but with the extra knowledge gathered during implementation this might have changed and using another cross-platform technology might have been a better choice.

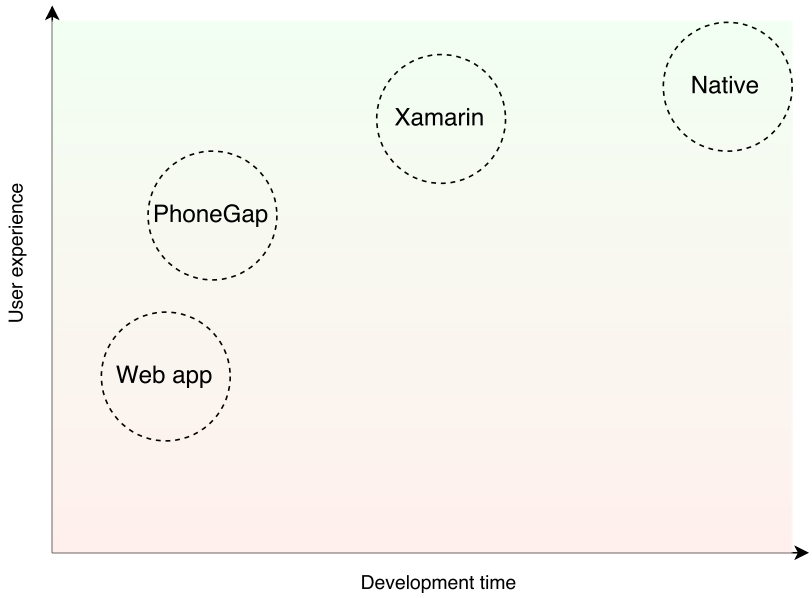
5.2.1 PhoneGap vs. Mobile Web Apps

To skip the step of distributing the app to the different app stores with PhoneGap and waiting for Apple's app review process, mobile web apps might have been a better solution. The users still need an internet connection in order to retrieve and send forms, therefore the web app will always be available to them. Furthermore, it would make it easier to release an update across all platforms at once and make sure that every user is running the latest version.
But as explained in Section 2.5, a problem is that web apps have limited access to the underlying platform. As it is now, we need access to the platform's filesystem to save a configuration file with login details and details about the server. Furthermore, we need to cache images and in web apps it is only possible to cache 5MB data. This problem could be solved by using HTML5's *WebStorage* to locally store the configuration and having the user download the images each time they request them, though it will decrease the user experience. However, a future feature of the app is that it should be able to connect to devices via Bluetooth, which will not be possible with a web app. The limited access to the hardware along with the bad user experience in a web app, makes it clear that PhoneGap was a better choice for this project. In a project where user experience is not a priority or you need to reach a lot of people a mobile web app could be a solution.

5.2.2 PhoneGap vs. Xamarin

The fact that Xamarin has access to all native functionalities and can achieve almost native performance as well as a native look and feel, makes it a very attractive cross-platform solution. Though, the native bindings comes with a price - development time. As mentioned in [9], Xamarin claims that on average 75% of the code can be reused across platforms in Xamarin projects. Which means that 25% of the code is platform specific. In another DTU thesis project by Tomasz Cielecki [27], he found that 69% of the code could be shared in his Xamarin project. The platform specific code means that not only the development time will be longer but time used maintaining the app will also be longer, seeing that specific code for three different platforms, instead of one, has to be updated. But this might be a sacrifice that you are willing to take if native use and feel is important. As earlier explained and shown in Figure 5.1, more time used on development can achieve a better user experience.

Figure 5.1: Developing time vs. user experience for cross-platform technologies.



To explore the difference in user experience further, a test could be conducted to see how much difference in native look and feel there is between a PhoneGap app and Xamarin app. The test could answer answer whether the difference is noticeable and how much this means to the user. The results could be used to

weigh in when making the choice of whether to use PhoneGap or Xamarin.

Another problem with Xamarin compared to PhoneGap is that a Mac OSX computer is needed for iOS development and a Windows 8 computer is needed for Windows Phone development. This means that the developer at Pernexus would have to switch between computers when developing for the different platforms. However, the use of Xamarin Studios with access to the platforms' SDKs makes it possible to debug the app. A lot of time was spent on debugging the PhoneGap app for Windows Phone and could have been avoided using Xamarin.

In [28] findings show that when extra features are added to a PhoneGap app, the memory usage increases. This indicates that a PhoneGap app with a lot of functionalities might have a decrease in performance. With Xamarin's nearly native performance, an app developed in Xamarin will in most cases be able to run as fast as possible.

The choice of whether to use Xamarin or PhoneGap depends on the ambitions of app. If a native look and feel is important or the app has complex functionalities, Xamarin might be the best choice. If you are able to compromise on the feel and look in favor of a quick and easy development process, PhoneGap will be a better choice.

5.3 Cross-Platform vs. Native

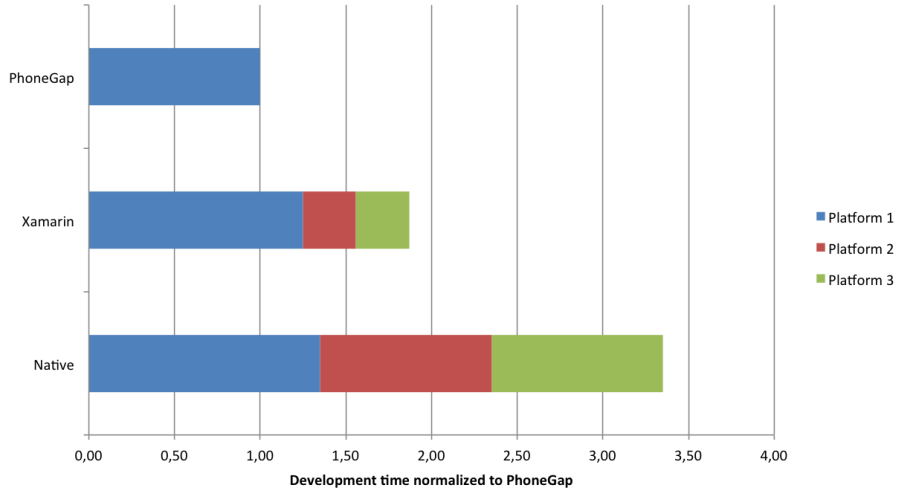
In this section it will be discussed if it is worth it to develop cross-platform apps rather than native apps.

Development time

The biggest advantage of developing cross-platform apps compared to native is the major shortening of time used on development. To develop with PhoneGap does not require any advanced knowledge and the developer can start implementing right away. Figure 5.2 shows an estimation of the development time when developing apps for three different platforms using the PhoneGap, Xamarin and native approach. Developing with PhoneGap is fastest since you do not have to deal with platform specific code. Furthermore once you have developed the code for PhoneGap it will work for all platforms and no extra time is needed for the other platforms. With Xamarin you need to develop platform specific, which increases implementation time. Consequently extra time is added per extra platform, since approximately 25% of the code will be platform specific. For native

apps you do not have any code sharing between the projects. However, there is a learning curve from when you have developed the first app and reuse the logic from the first platform. Therefore the development time is shorter for the next two platforms.

Figure 5.2: Estimated development time for 3 platforms.



User interface

As mentioned earlier, Xamarin creates native bindings to the underlying platform thus achieving a native look and feel consisting of native UI elements. This is not possible to achieve with PhoneGap and the feel and look of the app will be bad compared to Xamarin and native apps, but the implementation will be faster. Appcelerator Titanium and Xamarin are examples on cross-platform technologies that provides a native look and feel.

Performance

One would imagine that when Xamarin is only compiled to native and not running directly native, this would affect the performance. Actually experiments done by Harry Cheung [16] suggests otherwise. The tests show that an app written in Xamarin is faster than the same app written in Objective-C and Java respectively. Even though this might be unique for his test case, it is hard not to ignore that a Xamarin app has as good, if not better, performance than a native app. But there still might be some cases where native development might be needed. When developing high graphics consuming games for example, it is important to get the

best performance as possible. With a native implementation it is easier to control every aspect of your app and a native approach is therefore applicable.

If a cross-platform app is just as good as a native, how come there is not more cross-platform apps on the market? A reason might be that companies already have invested in native apps, with teams making native apps and a native code base. A restructure to develop cross-platform apps would therefore require a major effort. Another reason might be that Xamarin has only existed for a few years and there might be more experienced native developers than experienced Xamarin developers.

Figure 5.3 illustrates a suggestion of when to use PhoneGap, Xamarin or a native development approach. If the app has many users, user experience is most likely important. An app with high complexity requires the best performance and a native approach is there for most likely applicable. Though, if the app has low complexity it is easy to develop in PhoneGap without having to compromise the user experience including performance. As the complexity of the app increases the developer might have to use Xamarin instead of PhoneGap to ensure a good user experience.

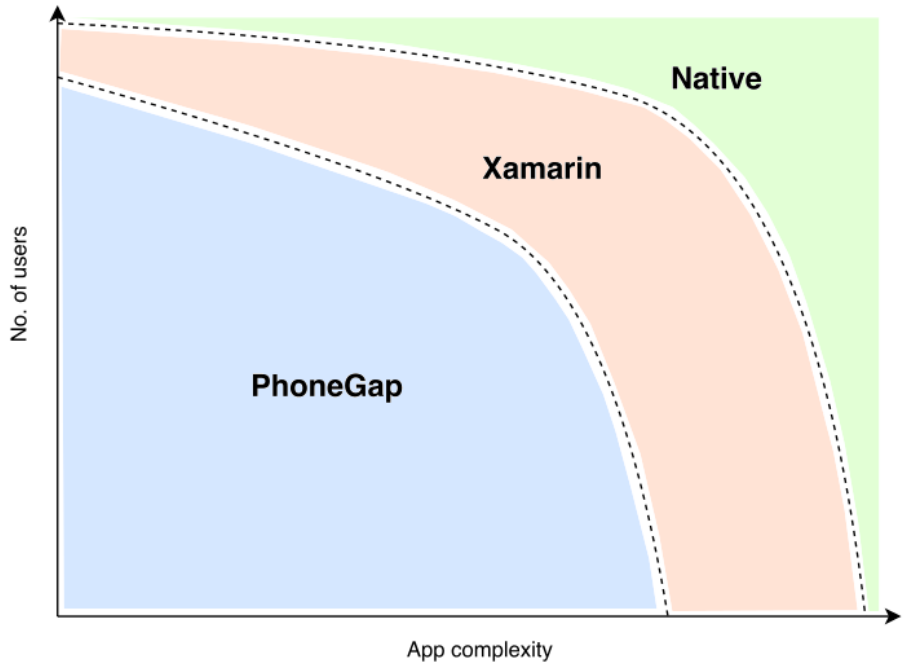
5.4 The Future of Cross-Platform Apps

It is difficult to predict what the future of cross-platform apps holds, but with the major reduction in time and cost of development a cross-platform app can provide, it is very likely that we will see more and more cross-platform apps in the app stores in the future.

5.4.1 Hybrid apps

PhoneGap, as it is now, will never be able to beat the user experience of a native app. To close the gap between PhoneGap and native apps, the PhoneGap web container needs to provide more features. It should provide native functionalities to improve the feel of the app and increase the performance. Luckily, the Cordova project is open-source and third-parties are able to build on top of the project to provide developers with possibilities to achieve a native look and feel.

Figure 5.3: When you should use PhoneGap, Xamarin or native development approach.



5.4.2 Compiled-to-native apps

With its nearly native performance and native look and feel, there is no doubt that Xamarin will grow bigger and bigger over the next few years. Additionally, at their Build conference in April 2015 Microsoft announced that with the upcoming Windows 10 it is now possible to use existing Android and iOS app code for Windows Phone 10 apps. This is done using a tool called 'Universal Platform Bridges' [15] that allows the developer to import the existing app project into Visual Studio and with only few changes getting the app to run on Windows 10. Whether this approach will create more apps available for Windows Phone will be interesting to follow.

5.5 Chapter summary

This chapter included an evaluation of developing a cross-platform app using PhoneGap. It discussed whether it would have been better to use another cross-platform approach to optimize the implementation. Furthermore it discussed if it is worthwhile to develop cross-platform apps instead of native apps. Lastly, the chapter discussed the future of cross-platform development.

Chapter 6

Conclusion & Future Work

This chapter will present the findings gathered during this thesis. The project as a whole will be concluded and future work needed for the app to be as good as 'EP Mobile' will be described.

6.1 Findings

In Chapter 1 (Introduction) the findings were that to develop a native app for Windows Phone, iOS and Android it must be done in three different programming languages, which consequently will prolong the development process when developing for all three platforms. Pernexus Systems want to offer their app for all three platforms while still having a fast development process and an app that is easy to maintain. For this, the solution was to develop an cross-platform application.

Chapter 2 (Analysis) explored and analyzed the different cross-platform solutions currently available on the market. Furthermore, the scope of the project was described from Pernexus' requirements for the cross-platform app. A comparison of the different mobile cross-platform technologies was made to find what solution that best fit Pernexus' needs. The pros and cons of each technol-

ogy was presented and compared to the requirements. *PhoneGap* was chosen as the technology to use for implementation of a cross-platform mobile app.

Chapter 3 and 4 (Design and Implementation) described the overall design and implementation of the project. The PhoneGap app is mostly implemented as a web app and the *AngularJS* framework was used to develop the app in HTML5, JavaScript and CSS. Unforeseen platform specific problems for Windows Phone arose during implementation, which led to a longer development time than first anticipated.

In Chapter 5 (Discussion) the development process with PhoneGap and the final application was evaluated. It is easy to start developing with PhoneGap, which makes the development of an app really fast. The fact that the source code only has to be written once for it to work on all platforms, is another huge advantage of PhoneGap. However, the user experience of a PhoneGap app is reflected by the user experience of the web app that the PhoneGap app consists of and is behind other technologies such as *Xamarin* in terms of feel and look. While this was not a problem for the users of Pernexus' app, it might be relevant for other developers and companies when choosing a cross-platform technology. Though, is the development time considerably less when developing PhoneGap apps than developing with Xamarin. And a small loss in user experience might be acceptable compared to the cost and benefits of a fast development process.

6.2 Conclusion

The overall conclusion of this thesis is that for Pernexus Systems, PhoneGap seems to be the best choice to develop a cross-platform mobile application. Even though PhoneGap cannot achieve the same look and feel as a native application, the user experience with a PhoneGap app is good enough for Pernexus' customers. Other cross-platform development technologies such as Xamarin can achieve the same native look and feel as a native app, but platform specific code has to be written in order to achieve this, which consequently increases development time. It is possible to achieve the same performance as a native app with Xamarin, while having the ability to share 75% of the source code between platforms. This makes the time of development of a cross-platform app for three platforms significantly shorter than developing native apps for three different platforms.

In the end, the choice of which development approach to use depends on the project. If you are willing to sacrifice some of the native look and feel of your app to achieve a fast development process, PhoneGap might be the choice over

Xamarin. If native user experience is important Xamarin will be a better choice.

6.3 Future Work

The app implemented during this project is already in the Apple App Store and in use by one of Pernexus Systems' customers. As mentioned earlier, the future releases of the cross-platform app should eventually have the same features as the current Android app, EP Mobile. Features will gradually be added to the cross-platform app until it reaches the same functionality as EP Mobile. When the cross-platform app has the same features as EP Mobile will EP Mobile be phased out and the focus will only be on the cross-platform app.

Right now the PhoneGap app has an uniform look across all platforms since the same source code is used for each platform. This affects the user's experience of the app, since it is clear that the app is not built from native UI elements. A way to avoid this perception is to create a custom UI that differs from any typical UI element. It would make sense to create a user interface that is customized for the customers needs and do not necessarily follow the UI guidelines for each platform. This way Pernexus Systems can put their own touch on the app.

For PhoneGap to provide the same native user experience as for example Xamarin apps can achieve, it is necessary to give the PhoneGap container that is connected to the underlying platform more functionalities. This could for example include the ability to use native UI elements to give the app a native look and feel and not a uniform look across all platforms.

Appendix A

Screenshots from EP Mobile

This appendix contains screenshots from the already existing Android app from Pernexus Systems, EP Mobile.

Figure A.1: Login screen in EP Mobile.

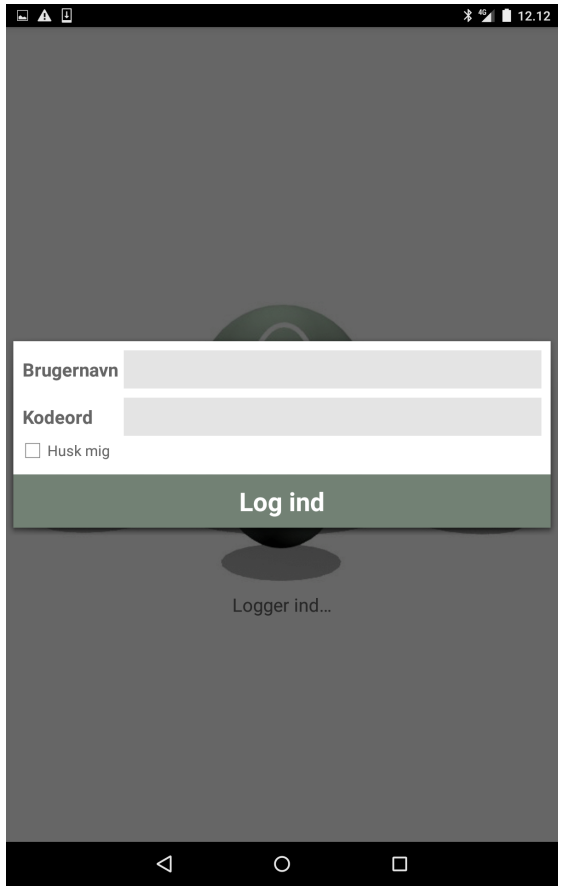


Figure A.2: Home screen in EP Mobile.

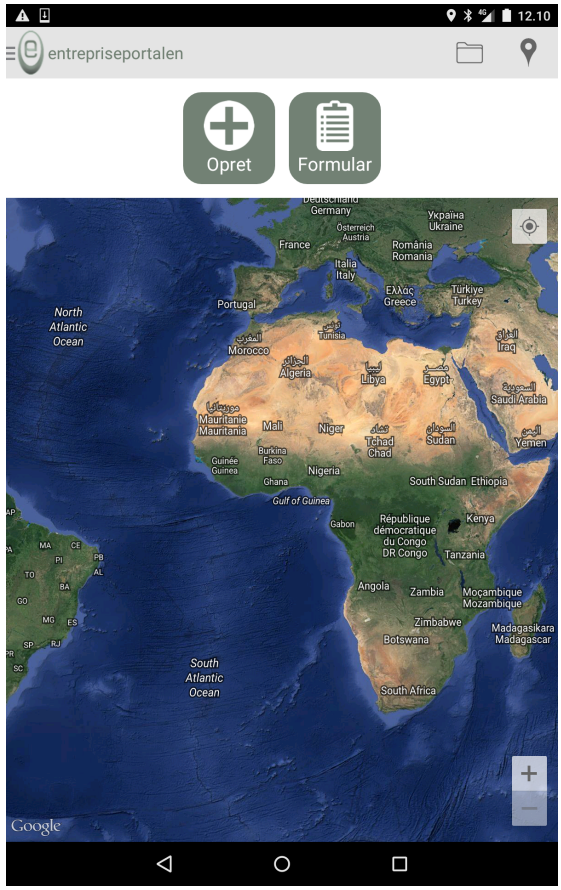


Figure A.3: Sorting projects in EP Mobile.

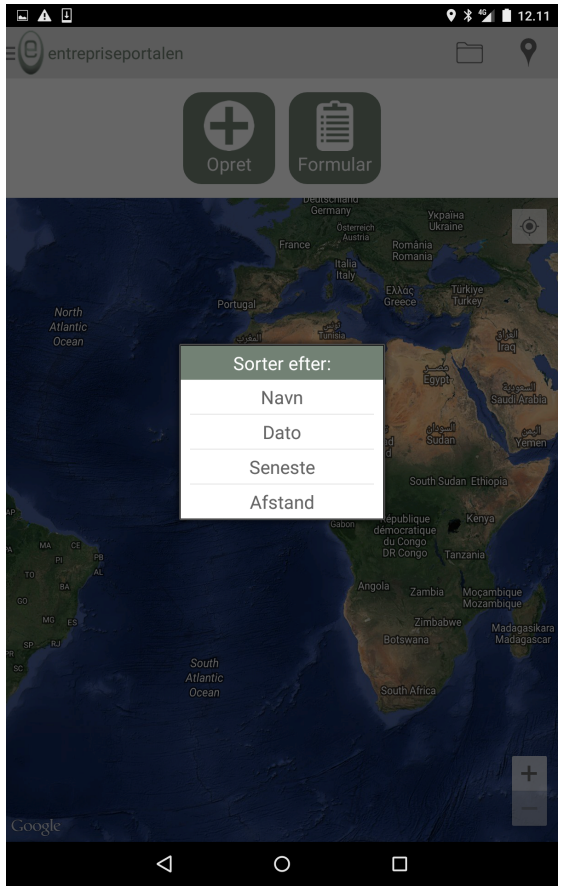


Figure A.4: Project list in EP Mobile.

Figure A.5: Form view in EP Mobile.

Figure A.6: Question view in EP Mobile.

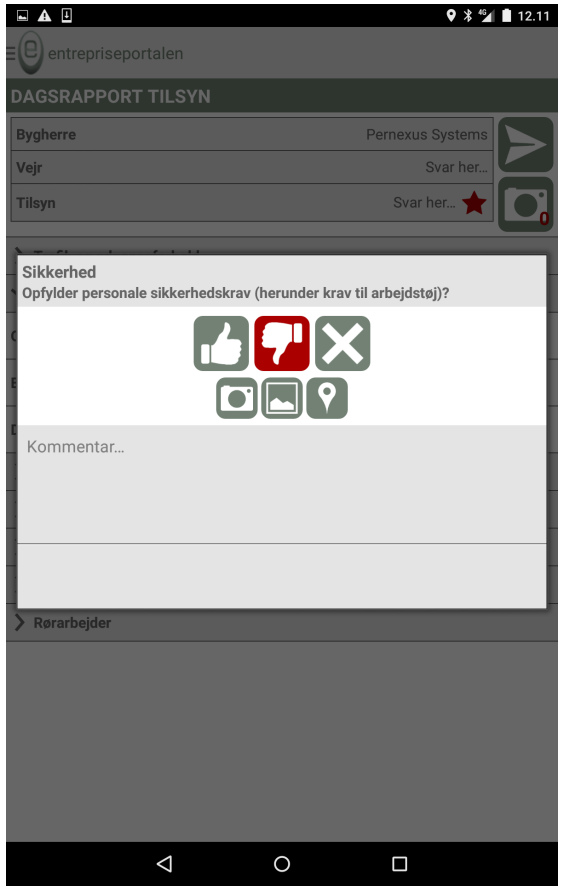
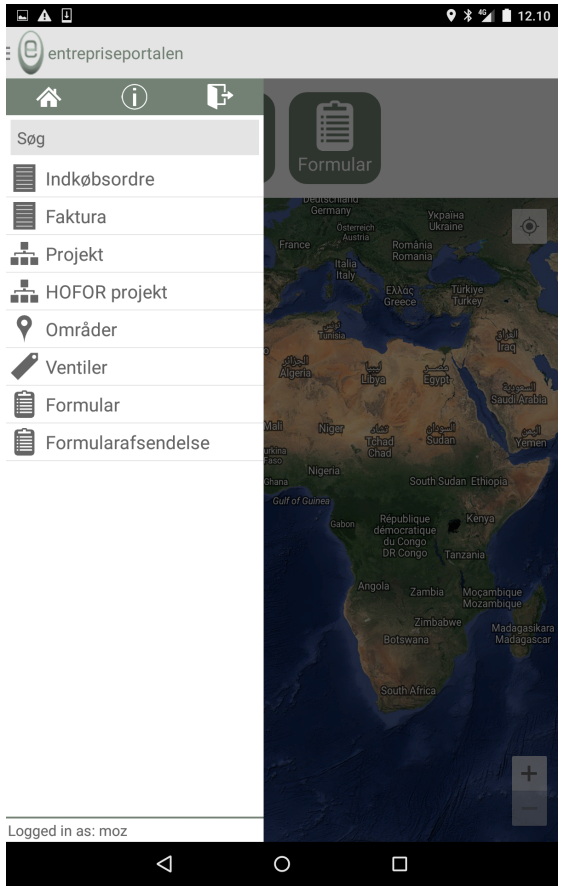


Figure A.7: Menu in EP Mobile.



Appendix B

Screenshots from the PhoneGap app

This appendix contains screenshots from the PhoneGap app developed during this project.

Figure B.1: Login screen in PhoneGap app.

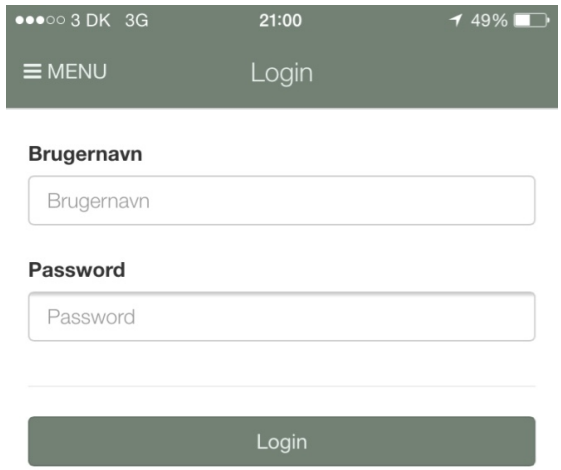


Figure B.2: Home screen in PhoneGap app.

Figure B.3: Sorting projects in PhoneGap app.

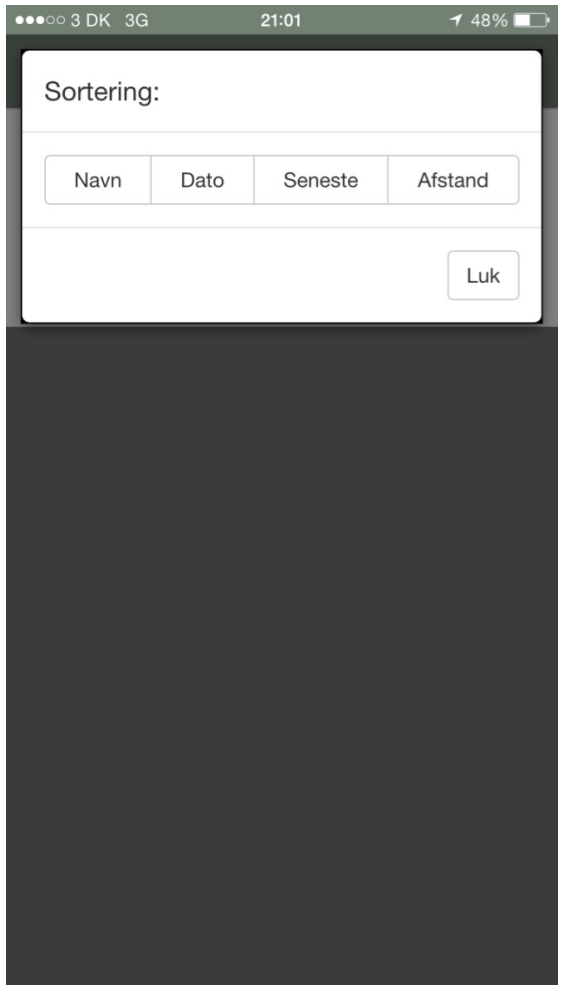


Figure B.4: Project list in PhoneGap app.

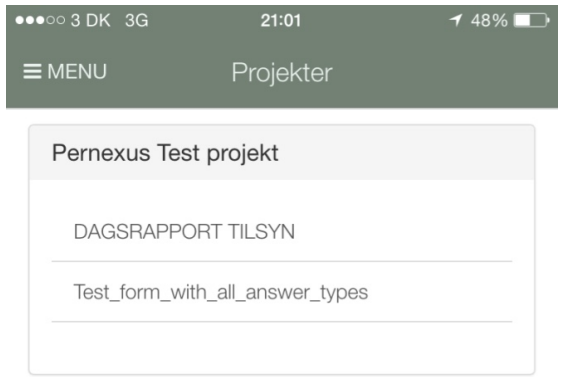


Figure B.5: Form view in PhoneGap app.

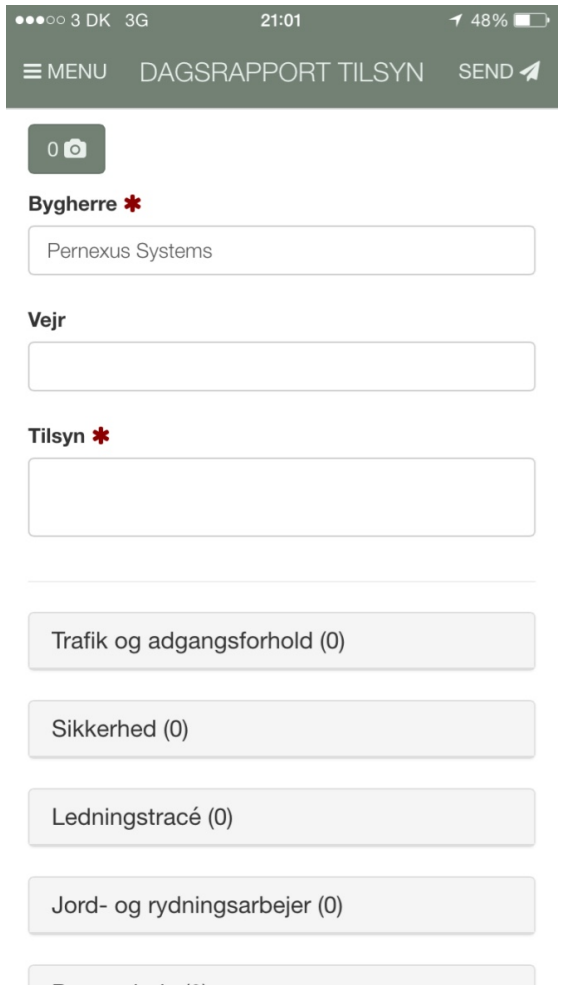


Figure B.6: Question view in PhoneGap app.

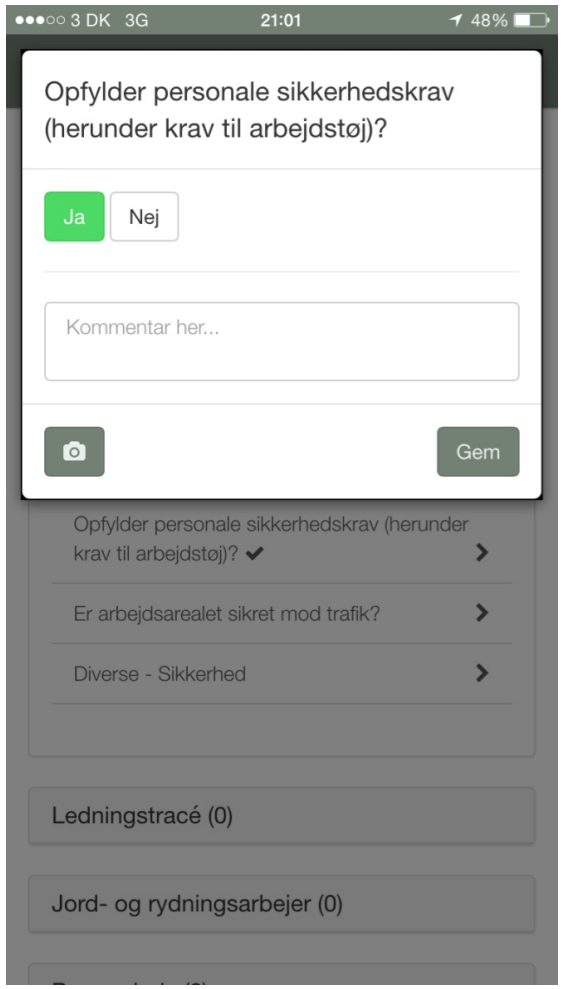
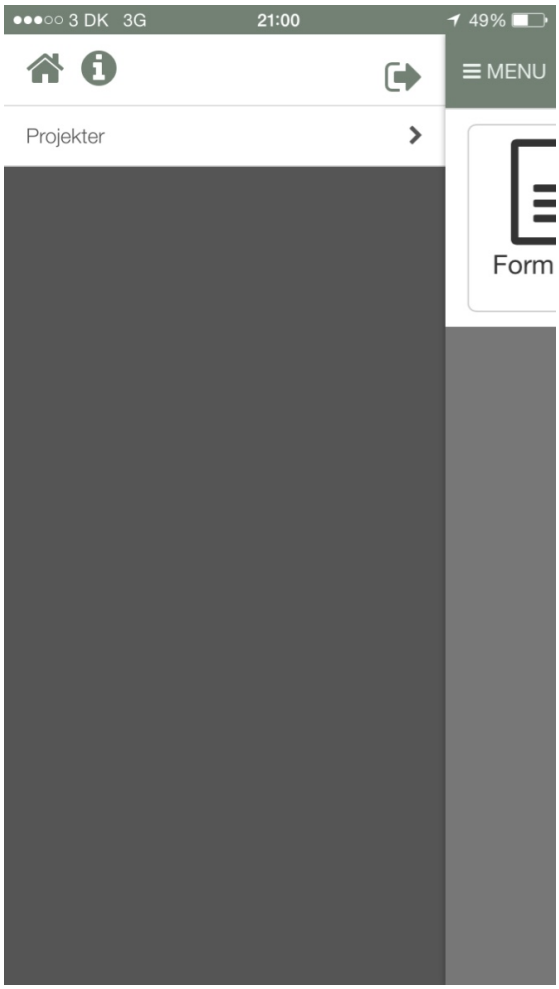


Figure B.7: Menu in PhoneGap app.



Bibliography

[1] Html5 mobile compatibility. http://mobilehtml5.org/, Last accessed: 02/06/2015.

[2] Sencha touch. http://www.sencha.com/products/touch/, Last accessed: 02/06/2015.

[3] About phonegap. http://phonegap.com/about/, Last accessed: 03/06/2015.

[4] Phonegap. http://phonegap.com/, Last accessed: 03/06/2015.

[5] Phonegap day - ibm, phonegap and the entreprise slideshow. http://www.slideshare.net/drbac/phonegap-day-ibm-phonegap-and-the-enterprise, Last accessed: 03/06/2015.

[6] Telerik platform. http://www.telerik.com/platform, Last accessed: 03/06/2015.

[7] Telerik ui for phonegap. http://www.telerik.com/phonegap-ui, Last accessed: 03/06/2015.

[8] Appcelerator titanium. http://www.appcelerator.com/product/, Last accessed: 06/06/2015.

[9] Microsoft and xamarin partner globally to enable microsoft developers to develop native ios and android apps with c# and visual studio. http://xamarin.com/pr/xamarin-microsoft-partner, Last accessed: 07/06/2015.

[10] Mono project. http://www.mono-project.com/, Last accessed: 07/06/2015.

[11] Xamarin. http://xamarin.com/, Last accessed: 07/06/2015.

[12] Xamarin - faq. http://xamarin.com/faq, Last accessed: 08/06/2015.

[13] Xamarin - introduction to mobile development. http://developer.xamarin.com/guides/cross-platform/getting_started/introduction_to_mobile_development/, Last accessed: 08/06/2015.

[14] Idc - smartphone vendor market share. http://www.idc.com/prodserv/smartphone-market-share.jsp, Last accessed: 10/06/2015.

[15] Introducing the universal windows platform bridges. https://dev.windows.com/en-us/uwp-bridges, Last accessed: 10/06/2015.

[16] Mobile app performance. https://medium.com/@harrycheung/cross-platform-mobile-performance-testing-d0454f5cd4e9, Last accessed: 10/06/2015.

[17] Android and ios squeeze the competition, swelling to 96.3% of the smartphone operating system market for both 4q14 and cy14, according to idc. http://www.idc.com/getdoc.jsp?containerId=prUS25450615, Last accessed: 11/06/2015.

[18] Gartner says by 2017, mobile users will provide personalized data streams to more than 100 apps and services every day. http://www.gartner.com/newsroom/id/2654115, 2014 Last accessed: 11/06/2015.

[19] gulpjs. http://gulpjs.com/, Last accessed 28/05/2015.

[20] ionic framework. http://ionicframework.com/, Last accessed: 28/05/2015.

[21] jquery mobile. https://jquerymobile.com/, Last accessed: 28/05/2015.

[22] Mobileangularui. http://mobileangularui.com/, Last accessed: 28/05/2015.

[23] Phonegap build. https://build.phonegap.com/, Last accessed: 28/05/2015.

[24] Twitter bootstrap. http://getbootstrap.com/, Last accessed: 28/05/2015.

[25] Android logcat. http://developer.android.com/tools/help/logcat.html, Last accessed: 31/05/2015.

[26] Allamaraju, S. *RESTful Web Services Cookbook*. O'REILLY, 2010.

[27] Cielecki, T. Environmental sensor monitoring tablet application designed using cross-platform design patterns and frameworks. Masters thesis, Technical University of Denmark, January 2015.

[28] Dalmasso, I., Datta, S. K., Bonnet, C., and Nikaein, N. Survey, comparison and evaluation of cross platform mobile application development tools. *Wireless Communications and Mobile Computing Conference (IWCMC)* (2013).

[29] Grant, A. *Beginning Angularjs*. Apress, 2014.

[30] Heitkötter, H., Hanschke, S., and Majchrzak, T. A. Comparing cross-platform development approaches for mobile applications. *Web Information Systems and Technologies Volume 140* (2013), pp 120–138.

[31] Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, A. S. Feature-oriented domain analysis (foda) feasibility study. Tech. rep., Carnegie Mellon University, 1990.

[32] Liang, Y. E. *PhoneGap and AngularJS for Cross-platform Development*. Packt Publishing, 2014.