Design og implementering af et lagersystem

Martin Skytte Sørensen

Kongen Lyngby 2013
IMM-B.Eng-2013-32

Technical University of Denmark
Informatics and Mathematical Modeling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk IMM-B.Eng-2013-08

Summary (English)

The goal of the thesis is to analyze and develop an inventory management system for an online bookstore, who only haves a transit warehouse. To solve the problem an application have been made, which can pair an order with a book. The thesis talks about how to optimize the daily routines of packing orders. Because it is a transit warehouse nothing is in stock, all items is ordered on a daily basis. So the problem revolves around how to optimize the pairing, by going from manual pairing to an application doing the pairing.

The problem in hand is resolved by, analyzing the normal packing routine of the company. Furthermore it is resolved by analyzing, which things are at hand and how they can be use to optimize the routine and also by looking at what future solutions could be.

Summary (Danish)

Målet for denne afhandling er at udvikle et lagersystem for en online boghandel, som kun har et transitlager. For at løse dette problem er et stykke software lavet, softwaret kan parre en ordre med en bog. Afhandlingen kigger på hvordan daglige rutiner for pakning af bøger kan optimeres, da der er tale om et transitlager, er der intet på lager, for alle vare bliver bestilt på daglig basis. Så problemet drejer sig om, hvordan man kan optimere parringen af bøger ved, at gå fra manuel parring til en automatisk parring.

Problemet er løst ved at analysere den nuværende pakke-rutine i virksomheden. Derudover er der kigget på, hvilke ting som allerede er tilgængelige og hvordan disse ting kan bruges til at optimere rutinen. Samtidig kigges der på, hvad der vil være af fremtidige muligheder for systemet.

Preface

This thesis was prepared at the department of Informatics and Mathematical Modeling at the Technical University of Denmark in fulfillment of the requirements for acquiring a B.Eng. in Computer Science.

The thesis deals with an inventory management system for an online bookstore. The aim is to optimize how the packing of orders is done, by going from manual pairing a book to an order, to a software system doing the pairing. The pairing process is done by calling a web service which contains information about the orders and the inventory.

The thesis consists of an analysis of the problem, which focuses on what is at hand and how it can be used. Then there is the design of the system, which focuses on how the system is to be build. There is also an implementation part, which focuses on how the system is implemented. Finally there is a conclusion that looks at what have been learned throughout the thesis.

Lyngby, 26-08-2013

Martin Skytte Sørensen

Acknowledgements

Jeg vil gerne have sige stor tak til Hubert Baumeister for at have taget imod min forespørgsel om at lave denne bachelor. Samtidig vil jeg også sige tak til Hubert for at have hjulpet med at have stille nogle gode spørgsmål, som har gjort at store dele af bacheloren er blevet udpenslet og lavet om så tingene er nemmere at forstå samt giver bedre sammenhæng.

Yderligere vil jeg sige tak til Jens Sjelhøj og Mianne Heyn for, at have læst min opgave igennem flere gange og rettet flere sproglige fejl, som havde gjort opgaven ulæselig

Contents

CHAPTER 1

Introduktion

Dette projekt omhandler et lagerstyrings system som snakker sammen med en webshop. Webshoppen sælger bøger og kræver derfor en speciel omgang med ordre, da det eneste boghandlen har, er et transitlager. Det vil sige, at der ikke er et lager som i traditionel forståelse, hvor man får varer hjem og så bliver de sat på hylder, så man nemmere kan finde dem. Måden dette lager er bygget op på er ved, at der modtages varer og disse varer bliver parret direkte sammen med en ordre. Hvis der mangler en bog i en ordre bliver denne liggende på lageret, indtil alle bøger tilhørende en ordre er kommet til boghandlen. Denne proces mellem parring af bøger og ordre bliver i dag gjort manuelt ved at logge ind på administrationssiden, hvorefter hver ordre tjekkes manuelt for, hvilke bøger der skal bruges og bøgerne bliver så fundet en efter en ud fra leverancen.

Denne proces ønskes optimeret ved, at få et scanningssystem, som kan snakke sammen med webshoppen og samtidig køre på en bred række af styresystemer. Måden processen ønskes lavet om på er ved, at parringen sker ved at scanne en bog, hvorefter systemet selv finder den ordre som bogen tilhøre.

Teknisk introduktion

Webshoppen køre i et open scource PHP system kaldet Prestashop [1]. Når Prestashop køre på en server, er der allerede implementeret en REST XML webservice [2] som indeholder de fleste funktioner som Prestashop har.

Det er som tidligere nævnt et krav at det skal kunne køre på en bred række styresystemer. Et andet krav er, at programmet skal ligge på computeren, så der ikke skal logge ind på hjemmesiden for at komme i gang med den ønskede aktivitet. Grundet dette er kodesproget Java valgt som hovedsprog.

Om Boghandlen

Boghandlen er en lille internet boghandel med 5 ansatte. På nuværende tidspunkt er boghandlen ved at flytte til større lokaler på ca 110m^2. På grund af måden bogmarked opbygget på i Danmark, er det som tidligere nævnt kun nødvendigt med et transitlager. Det vil sige at selvom boghandlen har 76.000+ bøger på webshoppen, er der intet på lager før en bog er bestilt. Det betyder at der kun skal være pakke plads, samt kontor plads til kundeservice mv.

CHAPTER 2

Analyse

For at finde ud af hvordan systemet skal bygge op og hvad der er brug for er der lavet forskellige analyser. Det følgende vil komme ind på afgrænsning af behov for produktet, samtidig vil det analysere, hvad der er brug for at komponenter.

Hvad er tilgængeligt

For at få en forståelse af mulighederne for denne opgave kigges der på, hvad der er tilgængeligt af udstyr og ressourcer.

Hos selve firmaet er der et par gamle computere af ukendt specifikationer; en med ældre Mac os og en med Windows XP. Til hver af disse computere er der en mus og et tastatur, samt en USB infrarød stregkodescanner. Computerne er ydermere tilkoblet internettet.

Hos hosting udbyderen er der en Intel Core i7-2600 Quard-Core med Hyper-Threading, 16 GB ram, 2 x 3TB Harddisk i raid 1 og ubegrænset trafik. På serveren køre der fuld LAMP [3] setup.

I selve lokalet er der stillet kasser op som fungerer som rum. Hvert rum har en størrelse af 1m x 0.5m x 0.4m.

Use stories

For at få et overblik over, hvad det er der egentlig ønskes af systemet, er der i samarbejde med boghandlen lavet en afgrænsning for, hvad systemet skal kunne.

- Som pakkeriansvarlig, vil jeg kunne scanne en bog og matche den til en ordre for ikke at skulle sortere bøgerne efter ordre.

- Som pakkeriansvarlig, vil jeg vide, hvilken hylde en ordre står på, så den nemmere kan pakkes.

- Som pakkeansvarlig, vil jeg kunne printe adresser på alle ordre der er færdige

Ud fra de ovennævnte use cases kan det ses, at der som minimum ønskes et system som kan scanne bøger, printe adresser og parre ordre med bøger.

User case

Ud fra de forskellige use cases, og i samarbejde med den pakkeansvarlige, er der lavet et flow, som er ønsket at få i det nye system. Et lignende flow har den pakkeansvarlige tidligere haft erfaring med fra en tidligere boghandel.

Pakke modtaget fra NBC eller DBK

En pakke med bøger kommer ind fra NBC eller DBK, pakken bliver åbnet og er fyldt med bøger. Der bliver taget en bog fra pakken, den bliver scannet og systemet parrer bogen med en ordre, hvis der er en. Hvis bogen ikke tilhøre en ordre skal den pakkeansvarlige informeres via en fejlmeddelelse. Hvis bogen parres med en ordre skal systemet fortælle, hvor bogen skal stå, samt fortælle om ordren er færdig. Hvis ordren er færdig skal der printes en adresse og en følgeseddel.

Tekniske muligheder

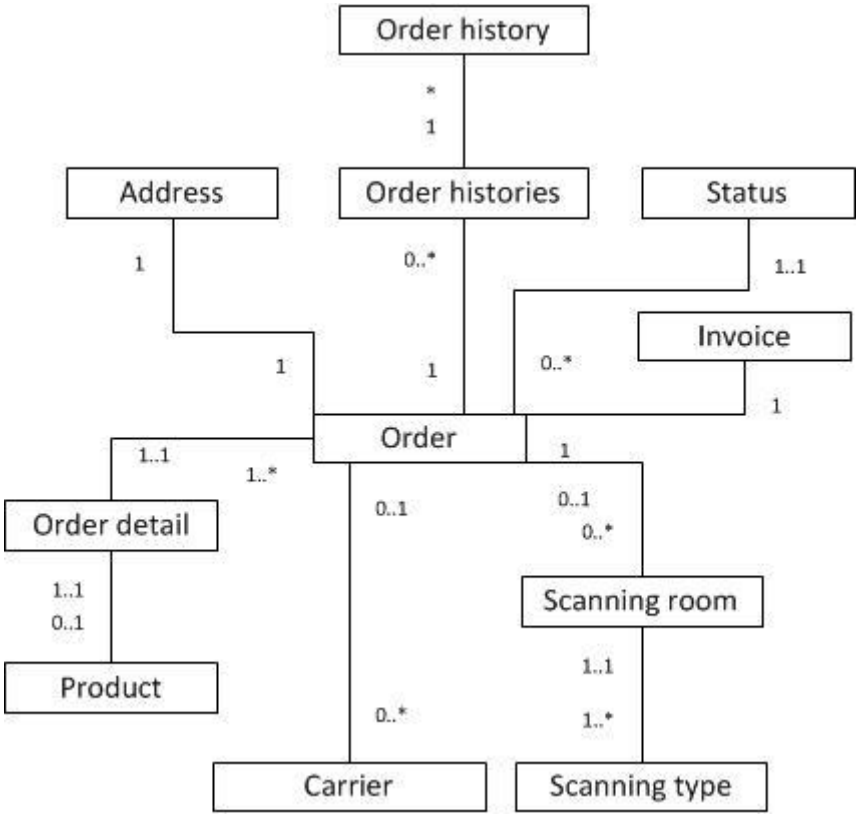
For at få et større overblik over, hvordan applikationen skal hænge sammen, er der her taget et kig på, hvordan det efterspurgte flow kan implementeres direkte i en klient samt hvad der skal udvides på serversiden af boghandlen.

Prestashops muligheder

Med udgang i Prestashop er de fleste funktioner allerede implementeret. Der er et lagersystem i Prestashop. Dette lagersystem er dog som i traditionel version, altså et produkt og dens placering på lageret [2]. Der kan oprettes flere lagersteder med mere, men da der skal bruges et ordre lager (transitlager) og ikke et almindeligt lager, skal Prestashops funktionalitet udvides med dette.

Domænet

Efter alle de oplysninger der er indhentet, kan der laves en simpel domænemodel af, hvordan applikationen er opbygget. Domænemodellen kan ses på Figur 1 Domænemodel af applikationen.



Figur 1 Domænemodel af applikationen

På domænemodellen kan det ses at det hele drejer sig om ordrer. Det er ordrer, som er hovedområdet for applikationen, hvilket giver god mening. Det kan yderligere ses, hvordan de forskellige elementer i domænet har forskellige multiplicitet overfor hinanden. Det kan også ses, at området for domænet er afgrænset til de elementer, der indgår i de forskellige use cases. På den måde kommer der ikke alle de elementer indover, som Prestashop har at tilbyde, men kun de elementer der skal bruges og er centrale for denne opgave.

Efter et kig på domænemodellen kommer det også til syne, at det hele kan deles op i to dele. Delene er som følgende:

1. Java applikation (indeholder hele domænet)
2. Prestashop webservice (indeholder store dele af domænet)

Det er naturligt for disse ting at overlappe hinanden, da der er tale om en applikation, som bygger på to forskellige webservicer.

Input validering

For at sikre at webservicerne får de ting som de beder om, skal der være input validering på de forskellige felter, der bliver repræsenteret i GUIen. Ydermere skal der også være noget input validering på nederste niveau. Så det sikres at webservicen får det den skal have.

Hvordan parrer man en bog og ordre

Hvordan forgår det i dag

Hele flowet fra start til slut foregår ved at en kunde bestiller en bog hos boghandlen. Når dagen er omme bliver der lavet en CSV liste over alle ordre der er kommet i løbet af dagen, denne CSV liste indeholder følgende

- Ordrenummer
- EAN eller ISBN13 nummer
- Antal
- Titel
- Boghandlernummer

CSV filen bliver herefter oploadet til Bogportalens bestillingssystem. De sørger så for at bøger bliver sendt fra DBK og NBC, ved hjælp af egen transport som levere ordren dagen efter.

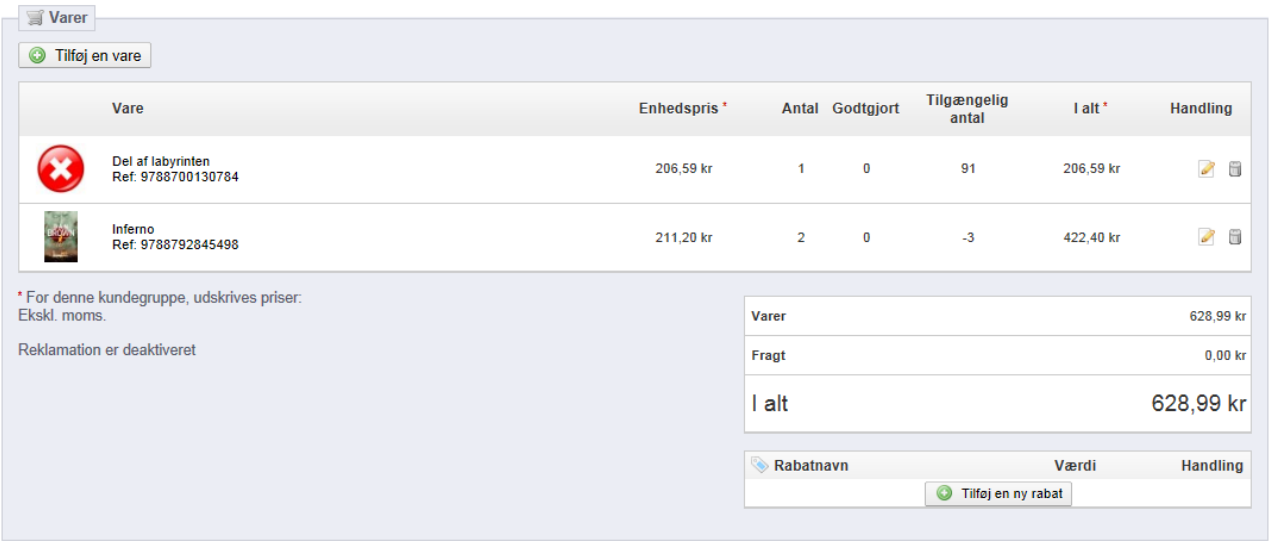
Parringen forgår i dag ved at Prestashops administrations panel bliver tilgået. Her bliver der så gået ind under ordre for at se, hvilke ordre der er kommet. Prestashop indeholder status på alle ordrer i systemet, på den måde kan hver ordre ses om den er sendt, pakket eller ikke påbegyndt. Hver ordre, som ikke står til at være afsendt eller på anden måde afvist/annulleret, bliver tilgået enkeltvis.

Når en ordre bliver tilgået, kigges der på, hvilke bøger der er i en ordre. Herefter findes disse bøger i kassen af modtagende bøger fra NBC/DBK. Her tjekkes om de EAN numre der er på bøgerne passer over ens med de EAN numre, der skulle være på bøgerne fra ordren. Dette kan have stor betydning, da der kan være forskellige udgaver af samme bog, men at bøgerne også har forskelligt EAN nummer. På Figur 2 Sådan ser en ordre linje ud” kan der ses et udsnit af, hvordan ordre linjerne står i Prestashop.

Efter de rigtige bøger er fundet og blevet dobbelt tjekket, pakkes de ind sammen med en følgeseddel og en adresse bliver skrevet på pakken, hvorefter den sendes.

Følgeseddel og adresse til en ordre bliver begge printet i fra Prestashops administrations for hver enkelt ordre.

Når en ordre er pakket sendes den ved hjælp af Post Danmark eller GLS, herefter trækkes der penge for de afsendte ordre den pågældende dag.



Figur 2 Sådan ser en ordre linje ud

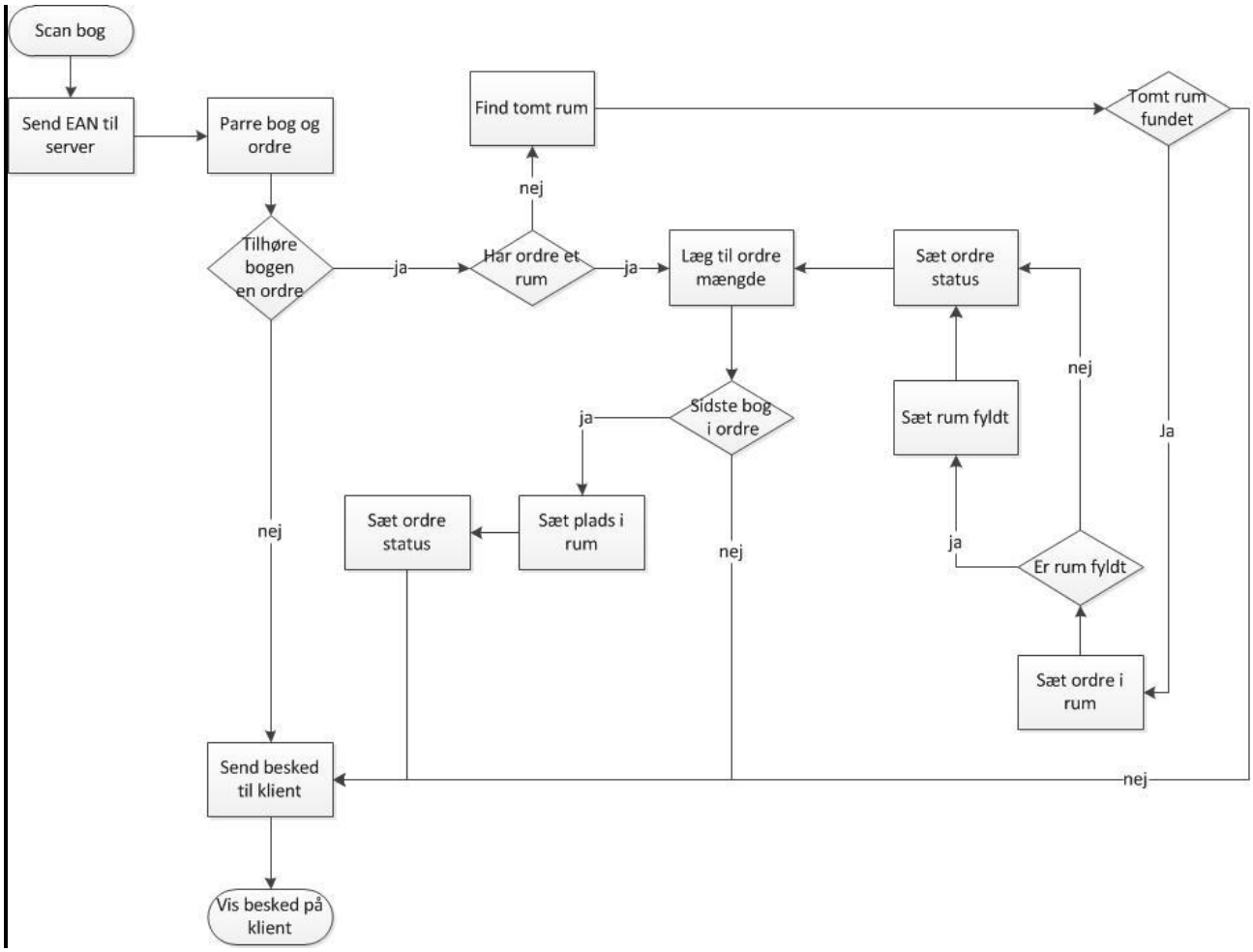
Hvordan kunne det virke

For at gå i detaljer med hvordan parringen kunne virke, sættes der først nogle generelle regler op. Reglerne er sat op efter forskellige forudsætninger, som er baseret på fysiske begrænsninger eller rimelige antagelser.

Den første antagelse er, at hvert rum kan indeholde mindst 3 ordre grundet pladsoptimering, er det en vigtig ting fordi rummene er så store og en ordrer ikke fylder et helt rum. Det vides fra pakkeriansvarliges tidligere erfaring med boghandel, at hver ordre i gennemsnit indeholder 3,2 bøger pr ordre. Så hvis der sættes en grænse på 3 ordre pr. rum, vil det sige, at der kan være, 9,6 bøger i et rum.

Den anden antagelse er, at der ikke løbes tør for pap til at pakke ordre ind. Denne antagelse er lavet ud fra, at det kan vælges, at købe pap med til bøgerne, når man bestiller fra Bogportalen. Derved sender NBC eller DBK pap med.

På baggrund af antagelserne er Figur 3 Ny måde at parre lavet. Følgende er en forklaring af, hvad der forgår på figuren.



Figur 3 Ny måde at parre

Først scanner en medarbejder en bog. Klienten tager herefter EAN nummeret og sender det til serveren

På serveren gælder det først om, at finde en ordre som indeholder bogen, men hvor ordren ikke er pakket, eller at den mængde af bogen ikke er scannet til en ordre. Hvis der ikke findes en ordre, så skal der meldes en fejl tilbage. Hvis bogen findes i en eller flere ordrer, vil et FIFO pattern være det mest fair. Forstået på den måde at først til mølle får, først malet mel. Udover dette så sikre det også at ældre ordre ikke bliver sprunget over i pakningsprocessen.

Hvis en ordre allerede har et rum, så skal der lægges en til den scannede mængde af bøger af den bestemte bog. Fordi der nu er scannet en bog mere til den ordre af den type bog. Hvis denne bog var den sidste i en ordre, skal der skiftes ordrestatus til at den er sendt samt at det rum som ordren er i skal gøres frit, altså sættes til at der er plads til en ordre i rummet. Såfremt bogen ikke er den sidste i ordren, så skal navnet på rummet findes og en samlet status for ordren skal sendes.

Hvis en ordre ikke allerede har et rum, så skal det første ledige rum findes. Et ledigt rum, er en kasse som har under 3 ordre i sig, altså et rum er ledigt hvis det har 0, 1 eller 2 ordre i sig. Hvis der ikke er nogle ledige rum skal der meldes en fejl om at pakke nogle ordre. Hvis et ledigt rum findes, så skal ordren lægges i det rum, samt rummet skal tjekkes om det nu har det antal ordre der max må være. Såfremt at rummet har det antal ordre, som det må have, skal rummet sættes til fyldt. Ordrens status sættes så til påbegyndt. Herefter tjekkes det om det var den sidste bog i ordren, det afgøre om rummet skal sættes til ikke at være fuldt på samme måde som tidligere beskrevet.

Herefter sendes resultatet til klienten som viser resultatet til medarbejderen.

Tynd mod tyk

Der flere måder at selve applikationen kan laves på, men de to måder, som er mest oplagt er enten en tyk eller en tynd klient.

Med tyk klient menes der, at det er der regnekraften ligger. Derved kan den hurtigere beregne forskellige ting end serveren ville kunne. Her beregner klienten de forskellige ting og giver herefter svar til serveren. Der menes samtidig at en tyk klient, er der hvor hoved applikationerne køre men samtidig gør brug af servere til at holde styr på data. Der kan nævnes for eksempel nævnes databaser. Den tykke klient kan samtidig udføre forretningsmæssige funktioner og som tidligere nævnt have kontakt til decentrale servere.

Med tynd klient menes der, at den ikke har så meget regnekraft. Derfor vil det tage lang tid at beregne forskellige ting. Her vil det være serveren der beregner tingene og sender svar tilbage til klienten. Altså sagt på en anden måde har den tynde klient en applikation som påtager sig input output, samtidig med at den kommunikerer med en hoved applikation som holder styr på data. Den tynde klient søger at sende input fra og til decentrale servere, dette gøres fra klientens model som kan være mere eller mindre kompleks.

For og imod at være tyk

Hvis en tyk klient blev valgt ville dette have sine fordele. Hvis nu at internet går ned imens der scannes bøger, kan der stadigvæk scannes bøger og pakkes bøger, da det er selve klienten der står for at lave denne scanning. Dog kan der opstå problemer med synkronisering, hvis der er mere end en som scanner bøger. Her skal de klienterne internt snakke sammen for at fortælle hinanden, hvem der har gjort hvad eller også skal de snakke med serveren. Hvis de kører på lokalt netværk, kan de snakke sammen selv, hvis internettet er gået ned. Når internettet køre igen, kan de hver især fortælle, hvad de har lavet af ændringer.

Udover at det vil give mulighed for at kunne scanne uden, at skulle være tilkoblet til internettet, så vil det også spare på server ressourcer. Det vil spare på den måde, at serveren ikke skal svare på de forespørgsler som der kommer på hjemmesiden, samt at parre bøger på samme tid. På den måde vil serveren blive mindre belastet.

For og imod at være tynd

Hvis en tynd klient er valgt behøver klienterne ikke længere tænke over, hvordan de skal synkronisere, så alle klienterne ved, hvad de er nået til hver især. Dette behøver de ikke, da det er serveren der står for, at skulle holde disse opdateret. Det vil sige, at kompleksiteten af klienterne vil falde drastisk. Grunden til, at det kan siges, at kompleksiteten falder, er, at Prestashop ikke benytter sig af JSON eller anden form for JavaScript til at udføre kald til webservicen. Dog hvis internettet går ned kan scanningen ikke fortsætte, fordi der ingen måde er at kontakt serveren på.

Samtidig vil denne metode også kræve flere ressourcer af serveren, da serveren skal holde styr på scanningen samtidig med den almindelige webshop.

En fordel ved at vælge en tynd klient, vil være udvikling til mobile enheder. Her tænkes der på tablets og smartphones, fordi der ikke skal bruges mange ressourcer til disse enheder så de nemmere kan håndtere et sådan klientprogram.

Mobileenheder, fremtid nu

Mobileenheder er et emne der er værd, at kigge på i forhold til fremtidige muligheder, eftersom smartphones og tablets bliver hurtigere og mere kraftige, kunne disse være en fremtidens scanner.

Langt de fleste smartphones og tablets, har et kamera i sig. Disse kameraer kan i fremtiden bruges til selve scanningen af bøger ved, at få kameraet til, at læse EAN på bogen. Hvorefter den sender EAN nummeret til serveren og serveren giver et svar tilbage.

Ydermere kan det laves sådan, at en smartphone eller tablet kan snakke sammen med en printer over netværk og herefter printe en følgeseddel ud til en ordre.

Det kan også nævnes, at eftersom Java er valgt som sprog for klienten, kan klienten forholdsvist hurtigt omskrives til Android enheder. Dette kan hurtigt ske, da Android programmering er baseret på Java.

Der kunne også argumenteres for at scanningen, kunne laves som en udvidelse af Prestashops administrationsside, som på den måde vil give adgang til at scanne fra nettet på alle enheder som har internet adgang. Her skal der dog bruges noget scannings funktionalitet, hvilket betyder at enten skal et kamera tilgås eller også skal den mobileenhed have en scanner tilsluttet.

Hvorfor udvidelse af Prestashop

Der er valgt at lave en udvidelse til Prestashop frem for, at lave en selvstændig webservice. Grunden til en udvidelse af Prestashop er valgt skyldes, at Prestashop tilbyder en del funktionalitet til hurtigt, at kunne tilføje nye enheder til webservicen. Det betyder samtidig, at der ikke behøves, at tænke over, hvordan REST udformningen skal være, da den allerede er givet. På den måde er det også allerede givet, hvordan den XML der bliver sendt frem og tilbage skal se ud. Her kan det dog argumenteres for at, der kunne laves en enkelt webservice for sig selv og så bare efterligne den XML som Prestashop giver. Men i det tilfælde, hvor der så skal bruges andre elementer fra Prestashops webservice såsom en adresse på en kunde, giver det mening at holde det samlet i stedet for, at skulle implementere disse ting i en selvstændig webservice. Det betyder også at man ikke behøver, at skulle holde styr på to forskellige webservicer eller få dem til at snakke sammen. Herved kan man holde klienten mere simpel.

Udover at holde klienten ensartet og simpel, har det det også den fordel at vedligeholdelse bliver lettere. For hvis Prestashop vælger at lave om i datamodellen, er det i selve udvidelsen af Prestashop der skal ændre. Da denne udvidelse bliver lavet ved hjælp af interfaces og udvidelser af modellen, vil det være Prestashops kerne der styre eventuelle ændringer. Hvorimod ved en standalone webservice vil der være risiko for at ændringer til modellen kan skabe store udfordringer.

Når der kommer nye udgaver af Prestashop udkommer de altid i flere udgaver inden ændringer bliver udgivet. Det vil sige der er altid en alpha udgave, en release candidate og så en produktions udgave. Udover dette så kan man køre to former for opdaterings strategi med Prestashop. Man kan få den til selv at opdatere når der er nye udgaver, eller man kan gøre det manuelt.

Sikkerhed i Prestashop

Der er også sikkerhedshensyn at tage, hvis der vælges at lave en standalone webservice. Af sikkerhedshensyn er det største, hvordan sikres der at et kald til webservicen kommer fra en pålidelig kilde. Det er der allerede sikret i Prestashops webservice, dette er gjort ved hjælp af HTTP autentificering. Men autentificering gives der som i større systemer en API nøgle, som er 32 karakterer lang. Ydermere kan der selektivt vælges hvilke dele af Prestashop en API nøgle skal have adgang til. Det vil sige at hver nøgle kan have adgang til noget forskelligt.

CHAPTER 3

Design

Design delen kan deles op i to dele. Den første del er Prestashop webservice udvidelse, som udgøre server siden af en parring mellem en ordre og en bog. Det er også denne udvidelse der udgøre server delen ved den tynde klient, som bliver nævnt i 'Hvad skal så vælges - tyk eller tynd'. Den anden del er selve Java programmet. Så for at hoved delen af Java programmet kan køre, er det nødvendigt at lave Prestashop delen først.

Hvad skal så vælges - tyk eller tynd

Serveren er en rimelig ressourcestærk server til en webshop, samtidig med at der findes gratis ydelser såsom cloudflare.com [4]. Cloudflare er med til at sprede serverload for den almindelige webside, ved at levere et CDN framework. Det resulterer i at serveren i store træk ikke mærker mange forespørgsler på hjemmesiden. Det gør Cloudflare ved at automatisk at cache statiske filer til en hjemmeside, herefter placerer de filerne i deres datacentre, så filerne hurtigere kan blive serveret til en bruger af hjemmesiden. Cloudflare vil derfor være med til at aflaste serveren ved udgivelser af nye bøger som er populære, derved forebygges der imod et pres på serveren.

Hvis internettet gik ned ville det i forvejen gå udover webshoppen, da det ikke ville være muligt at bestille bøger fra Bogportalen. Ved at dette ikke ville være muligt, ville der heller ikke kunne pakkes bøger. Det er kun hvis man kigger isoleret set på at internettet er nede i mindre end en dag, at en tyk klient vil have sine fordele. Internetnedbrud opleves dog sjældent og kan formindskes med en redundant internetforbindelse.

På baggrund af lav kompleksitet og rigeligt med ressourcer er der valgt en tynd klient. Dette er også valgt grundet sjældne nedbrud af internettet.

Yderligere er der også en resterektion på de ældre computeres regnekræft. Det er derfor nødvendigt at tage højde for computerne. Det betyder også at en tynd klient vil være at fortrække i forhold til de allerede anskaffede computere.

Det kan også nævnes at da Prestashop er valgt som shop, giver det også den fordel at der allerede er hoved applikation, som har mulighed for at kommunikere med klienter via en REST service.

Grunden til der ikke er valgt at gøre brug af mobileenheder, er ønsket om genbrug af tidligere brugte computere og scanner. Ydermere vil boghandlen gerne kunne udvide applikationen med at kunne gemme scannet data, hvis internettet går ned.

Samtidig har boghandlen ikke mobileenheder til rådighed på nuværende tidspunkt, det vil sige at medarbejdere skulle bruge egne enheder, som for eksempel mobile telefoner. Her er der større chance for at en utilfreds medarbejder bruger applikationen utilsigtet.

Prestashop udvidelse

Udvidelsen af Prestashops REST webservice skal i al sin enkelthed kun modtage et EAN nummer som input, hvorefter den selv skal stå for at parre en ordre med en bog, finde et rum til ordren og sætte status for ordren. Når de tre ting er gjort skal webservicen komme med et output som indeholder

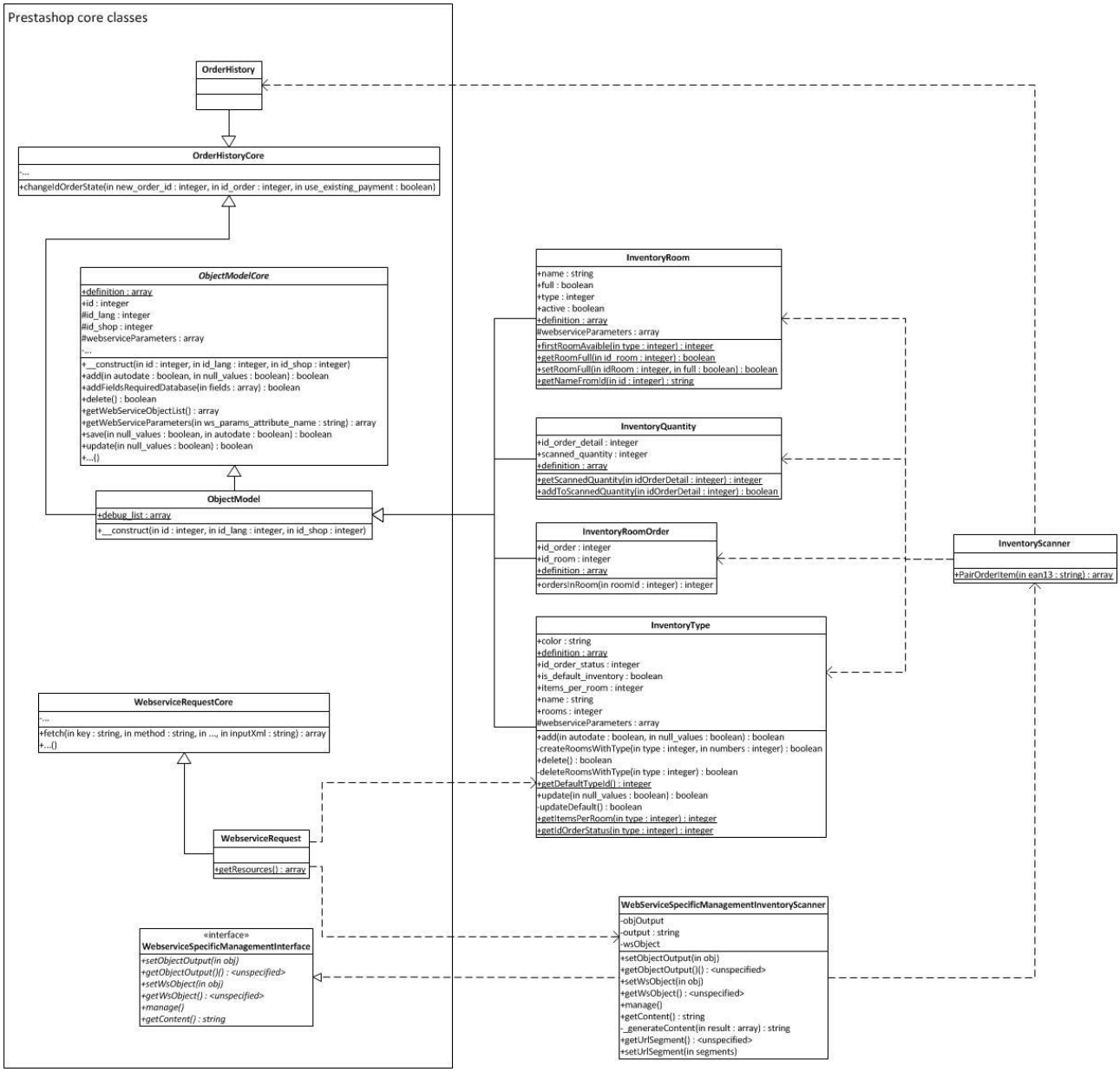
- Status for ordre
- Rum til ordre

Da der ikke er noget dokumentation omkring, hvordan man skal udvide webservicen, er der kigget i kildekoden for at finde ud af sammenhængen mellem diverse klasser. Ydermere er der ved hjælp af søgning på internettet fundet ud af, hvordan andre har tilpasset webservicen [2].

Ud fra de ting som der er fundet ud af omkring webservicen, er denne blevet udvidet via flere metoder. Der er ikke brugt et specielt design pattern andet end tingene er lagt i den mappestruktur som Prestashop forventer for udefrakommende udvidelser.

Der bliver på intet tidspunkt ændre i nogen af hoved filerne for Prestashop, da det er opbygget på den måde, at alle objekter har det som Prestashop kalder et core objekt. Hvis man ønsker at ændre i et core objekt skal dette gøres i et override objekt som nedarver core objektet. På Figur 4 Klasse diagram for PHP delen af scanner kan et eksempel på dette være; OrderHistoryCore som bliver udvidet af OrderHistory. I dette tilfælde vil alle override ting blive indskrevet i OrderHistory, hvis der er nogle.

På Figur 4 Klasse diagram for PHP delen af scanner ses et klassediagram som beskriver opbygningen af udvidelsen af webservicen (nogle af Prestashops klasser er undladt). På diagrammet kan det ses, at de normale REST kald alle udvider Prestashops Object model. Denne model søger for nemt at lave Create, Read, Update og Delete (CRUD) på enhver klasse der udvider modellen.



Figur 4 Klasse diagram for PHP delen af scanner

Ydermere kan det ses, at der er et interface i Prestashop til, at lave et mere specifikt webservice kald. Interfacet kaldet WebserviceSpecificManagementInterface implementeres. Interfacet holder styr på de funktioner som ikke har fuld CRUD men blot en af funktionerne. Her kan der som eksempel nævnes søgning i Prestashop, hvor der vil blive returneret både kategorier og produkter, som individuelle lister af objekter af de respektive typer.

Herefter implementeres interfacet WebserviceSpecificManagementInterface i klassen WebserviceSpecificManagementInventoryScanner. Den klasse bliver så tilføjet til metoden getResources i WebserviceRequest klassen, som er en override metode, hvori nye kald til webservicen skal implementeres. Herved styre Prestashop selv, hvilken output der skal returnere ved et webservice kald.

For at få webservicen til at integrer diverse objekt modeller, såsom InventoryRoom, skal disse også implementeres i getResources i WebserviceRequest klassen.

De forskellige navne på klasser, som extender Prestashops objekt model, er taget ud fra hvad de repræsenterer i databasen. For eksempel kan det ses at InventoryRoomOrder viser, at der er en sammenhæng mellem ps_inventory_room og ps_order i databasen. På den måde afspejler modellen og databasen hinanden.

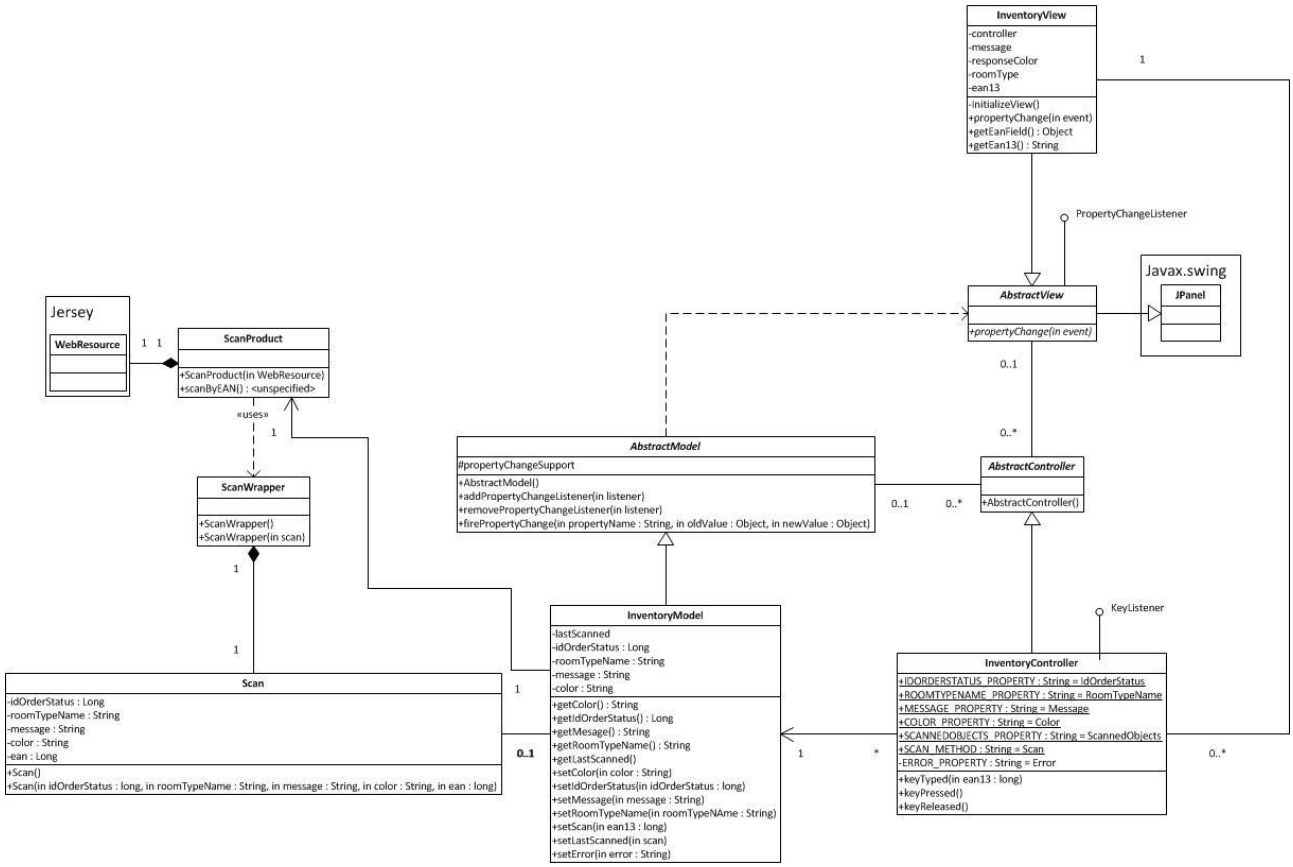
Java applikation

I Java delen er der brugt et MVC pattern til, at separere model laget fra interfacet. På den måde kan modellen testes for sigt selv, og ligeså kan guien. Samtidig giver det en god struktur i selve programmet.

Det har yderligere været nødvendigt at implementere wrapper klasser på grund af måden Prestashop sender sin REST XML, samtidig med måden JAXB forventer, at pakke XML ind og ud på. Det vil sige at hvert objekt som kommer fra Prestashop er indkapslet i en wrapper, som kun indeholder det egentlige objekt.

Ved at bruge dette, kan det de forskellige lag i applikationen adskilles. Herved kan lagene testes individuelt eller på anden måde bruges i forskellige sammenhæng. For eksempel, hvis man ønsker en anden type GUI end den som er implementeret.

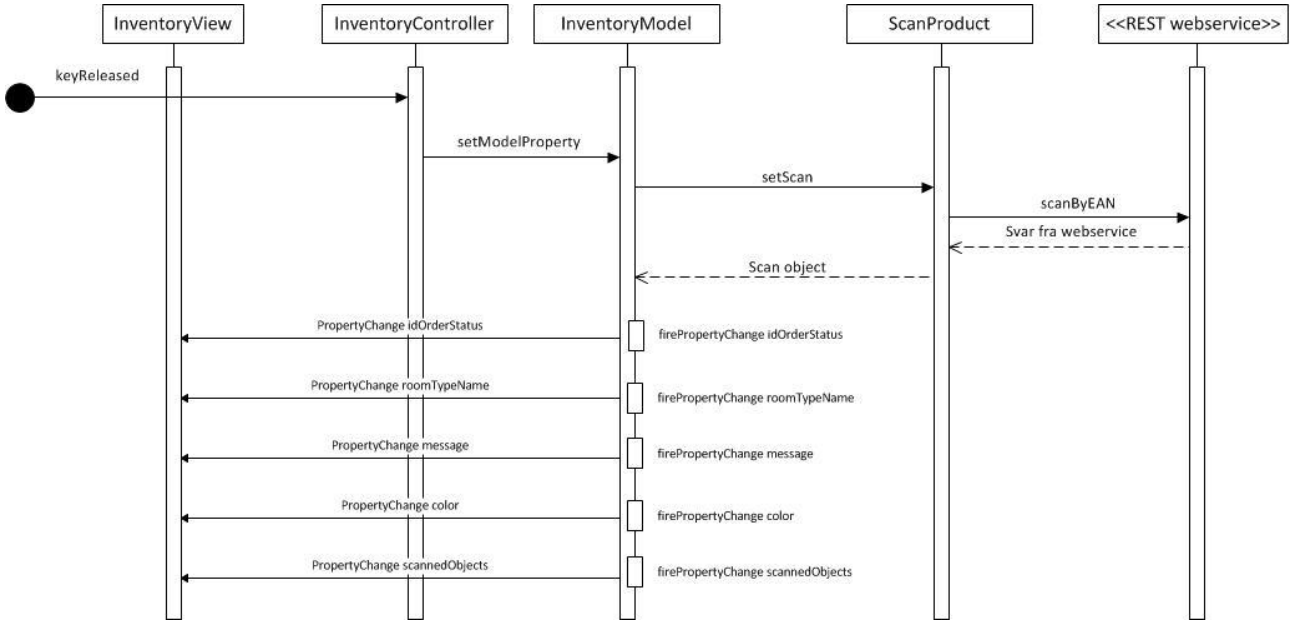
På Figur 5 Klasse diagram over scanning kan der ses et eksempel på, hvordan scanningen virker i et detaljeret klassediagram. Her kan de forskellige MVC klasser ses, men yderligere kan det ses at Model laget er splittet op i flere ting. Altså Model laget består i praksis af selve modellen og et data lag som varetager al kommunikation med den gældende webservice.



Figur 5 Klasse diagram over scanning

Hvordan bliver data behandlet

Måden hvorpå denne kommunikation forgår på er ved, at en aktivitet bliver påbegyndt på GUI laget. Herefter bliver den relevante information sendt til Controller laget, som igen sender dette videre til Model laget. Herfra bliver de forskellige dele af modellen aktiveret og webservicen bliver kaldt med de pågældende parametre. Et svar kommer tilbage til Model laget hvorefter de berørte ting bliver opdateret i modellen. Disse opdateringer bliver annonceret til de views, som er lytter til den pågældende model. Derefter opdateres de relevante emner i GUIen. Et sekvensdiagram over aktiviteten scan bog kan ses på Figur 6 Sekvens diagram over scanning'.



Figur 6 Sekvens diagram over scanning

CHAPTER 4

Implementering

I Teknisk introduktion blev det nævnt, at Java er valgt som kodesprog. Det kan herefter nævnes, at følgende pakker til at udvide Java er valgt Jersey og JAXB. Disse pakker er valgt for at kalde Prestashops webservice.

Jersey er en HTTP klient, som kan kalde en webservice, pakke kaldet og responset fra serveren ind i XML. Jersey kan bruge JAXB til at gøre dette, derfor var JAXB et naturligt valg til bruge.

JAXB er et værktøj til Java til, at pakke et objekt ind i XML eller pakke en XML fil ud til et objekt. Dette kunne også være gjort ved at serilizere et objekt, men JAXB blev fundet mere egnet, i samarbejde med Jersey, til at kalde en webservice og få et respons tilbage.

Parring af ordre og bog

Som det blev besluttet i 'Hvad skal så vælges - tyk eller tynd' er parringen mellem bog og ordre lavet på server siden. Det vil sige, at denne er kodet i PHP. Der er taget udgangspunkt i de funktioner som Prestashop allerede har stillet til rådighed. Der kan nævnes active records til at kommunikere med databasen, samt Prestashops generelle regler beskrevet i deres dokumentation [6].

På figur Kode 1 Parring mellem ordre og bog, kan det ses, at pseudo koden følger det der blev opnået i Figur 3 Ny måde at parre for, at finde frem til parringen. Der bruges i implementering et FIFO mønster. Forstået på den måde, at den ordre der er ældst, og indeholder bogen, vil være den ordre der bliver fundet og behandlet først.

```
PairOrderItem(ean){

    orderDetails = firstOrderDetailsNotPackedWithProduct(ean);

    if(orderDetails == null) {
        inventoryTypeID = InventoryType->getDefaultTypeID();

        if(!orderDetails->room == 0){
            emptyRoom = InventoryRoom->firstRoomAvaible (inventoryTypeID);
            if(!emptyRoom){
                return “no empty rooms, pack some orders”;
            }

            InventoryRoomOrder = new InventoryRoomOrder();
            InventoryRoomOrder->id_order = orderDetails->id_order;
            InventoryRoomOrder->id_inventory_room = emptyRoom;
            InventoryRoomORder->save();

            if(InventoryRoomOrder->getItemsPerRoom($type)){
                InventoryRoom->setRoomFull(emptyRoom, true);
            }

            OrderHistory->changeOrderState(“packing”, orderDetails->id_order);
            orderDetails->room = emptyRoom;
        }

        InventoryQuantity->addToScannedQuantity(orderDetails->id_order_detail);

        if(orderComplete(id_order)){
            orderHistory->changeIdOrderState(“packed”, orderDetails->id_order);
            InventoryRoom->setRoomFull(orderDetails->room, false);
            InventoryRoomOrder->deleteByIdOrderIdRoom(orderDetails->room, orderDetails->id_order);

            return array(id_order, orderDetails->room, “packing”, Green, ean);
        }

        return array(orderDetails->id_order, orderDetails->room, “complete”, Green, ean);

    }

    return array(0, 0, “Error no order with that book”, Red, ean);
}
```

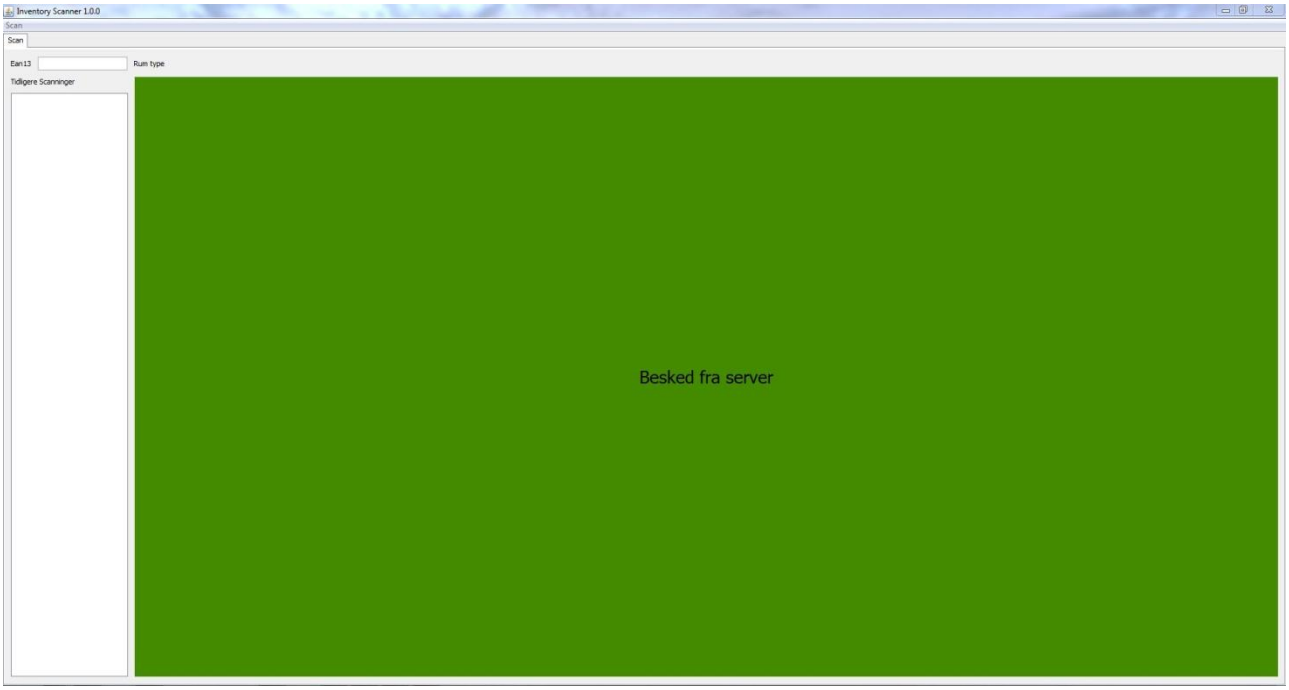
Kode 1 Parring mellem ordre og bog

Hvad ser brugeren

Når denne parring skal præsenteres for en bruger af klienten, er der implementeret en GUI, som har til formål at være så simpel som mulig, men også giver relevant information til brugeren. Ved en parring er relevant information til en bruger følgende

- Et overblik over hvad der er scannet tidligere
- Hvilken rum en ordre ligger i
- Information omkring ordre status

For at opnå de tre ting, er der på GUI en tekst boks i toppen, hvor EAN nummeret indskrives eller scannes. Så er der en liste med tidligere scanninger og til sidst en stor boks, hvor statussen bliver indikeret med en farve, samt beskeden fra webservicen. Dette kan ses på Figur 7 Scanning’s gui



Figur 7 Scanning's gui

Hvordan fungere det

GUIen fungere ved, at der bliver tastet et EAN nummer ind i tekstboksen og der trykkes herefter ENTER. De scannere der bliver brugt, trykker automatisk ENTER efter hvert nummer de har indlæst, eller sagt på en anden måde USB scanneren emulere input fra et tastatur, så den returnere et EAN nummer efterfulgt af ENTER. Herefter valideres EAN nummeret og hvis nummeret ikke kan valideres, vil brugeren blive præsenteret med en fejlbesked. Hvis EAN nummeret godtages sendes nummeret videre gennem controlleren og flowet fra Figur 6 Sekvens diagram over scanning begynder.

CHAPTER 5

Test

For at tjekke om de forskellige funktioner i programmet virker som de skal, er der brugt forskellige metoder til at teste. Måden der er testet på, er ved hjælp af JUnit test, samt ved, at tjekke om de ting der forventes sker.

JUnit test af data lag

For at teste model laget er der blevet brugt JUnit tests, men selvom JUnit er til fuldautomatisk test, har der været brugt nogle specielle ordrer for, at udføre disse test. Disse ordrer er oprettet direkte i Prestashop, hvor der er blevet bestilt nogle specielle produkter til hver ordre. Dette skal sikre, at de automatiske test kan køre og får testet et hvert scenarie af scanningen og hævningen af penge.

Ydermere er alle wrapper objekter og selve metoderne i objekterne blevet testet for Exceptions og ganske normale getters og setters.

GUI validering, what you see is what you get

GUI er der ikke blevet lavet fuldautomatisk test af, herimod er det blevet testet ved, at hver enkelt knap egentlig har gjort det, som der har været ønsket. For eksempel er det testet, om fejlmeddelelser er blevet vist ved forkert input, samt om de forskellige knapper har kaldt de rigtige funktioner og efterfølgende har fået svar. Altså er dette testet ved fysisk at kigge på input og output.

For at teste GUI er der taget udgangspunkt i samme test som datalaget, det vil sige at de fejl der bliver testet for ved scanning af en ordre også bliver valideret i GUI. Så hvis der forventes en fejl i datalaget, så forventes der at ses en fejl i GUI. Hvis en operation går godt forventes der at ses et positivt respons.

CHAPTER 6

Konklusion

Det er med succes lykkedes, at implementere og teste en klient, som kan parre en bog med en ordre. Dette betyder i praksis, at der er sparet en masse tid, da processen er gået fra, at være 100 % manuel til nu, at være delvist automatiseret.

Det er samtidig lykkedes, at vise, hvorfor det er en tynd klient, der er valgt frem for en tyk. Dette er gjort på baggrund af de tilgængelige ressourcer, samt at dette også sikre en udvikling hen imod mobile enheder, såsom tablets og smartphones.

Hvis der kigges lidt ekstra på systemet og laves små justeringer kan systemet også bruges til shops, som ikke har transitlager. Her vil systemet kunne bruges til at pakke efter og sikre, at de rigtige ting bliver sendt til en kunde.

Som systemet er lige nu bliver tidligere scanninger kun gemt så længe programmet ikke bliver lukket. Dette bør der laves om på, så der for eksempel kan laves rapportering af scanninger eller at programmet gemmer den tilstand som det er i. På den måde kan der også skabes mulighed for at scanne og gemme scanningerne, i tilfælde af at internettet får ned. Herefter vil den så kunne sende scanningerne og få svar tilbage fra webservicen.

Der kan også gøres noget ved listen af tidligere scanninger. Der bør tilføjes funktionalitet til at tage imod klik, hvorefter den så skal vise information for den pågældende scanning.

Referencer

[1] Prestashop, »PrestaShop - Start an online store today with PrestaShop's free e-commerce software.,« [Online]. Available: http://www.prestashop.com/. [Senest hentet eller vist den 14 August 2013].

[2] Prestashop, »Web service reference - PrestaShop 1.5 - PrestaShop documentation,« [Online]. Available: http://doc.prestashop.com/display/PS15/Web+service+reference. [Senest hentet eller vist den 14 August 2013].

[3] Wikipedia, »LAMP (software bundle) - Wikipedia, the free encyclopedia,« [Online]. Available: http://en.wikipedia.org/wiki/LAMP_(software_bundle). [Senest hentet eller vist den 19 August 2013].

[4] CloudFlare, »Home | CloudFlare | The web performance & security company,« [Online]. Available: https://www.cloudflare.com/. [Senest hentet eller vist den 14 August 2013].

[5] R. Eckstein, »Java SE Application Design With MVC,« [Online]. Available: http://www.oracle.com/technetwork/articles/javase/index-142890.html. [Senest hentet eller vist den 14 August 2013].

[6] Prestashop, »Coding Standards - Prestashop,« [Online]. Available: http://doc.prestashop.com/display/PS15/Coding+Standards. [Senest hentet eller vist den 20 August 2013].

Figur Liste