CUPCAKES

Et projekt af:
Martin Laurvig Brandstrup / cph-mb606@cphbusiness.dk / github: MartinBrandstrup
Martin Bøgh Sander-Thomsen / cph-ms782@cphbusiness.dk / github: cph-ms782
Martin Wulff Nielsen / cph-mn521@cphbusiness.dk / github: cph-mn521
Niels Friis Møller Bang / cph-nb168@cphbusiness.dk / github: nfmbang

Klasse A

Udarbejdet i ugerne 9-11, rapport afleveret 15/marts-2019

Hjemmesiden: 128.199.46.149/CupCake
Java-docs: cph-mn521.github.io/Cup_Cake

Indledning

I dette projekt har vi udviklet en webshop der kan håndtere og behandle køb og fakturering af bestillinger samt præsentere brugeren med en personlig side, der indeholder deres bestillinger, historik, samt se deres instore balance. Siden har både registrering og login funktioner samt en "kurv" der indeholder en ikke betalt bestilling.

Vi gemmer relevant data i en database, og har derfor udviklet både en front end og en back end. Det færdige projekt har vi deployet på en tomcat server som fungere som vores daemon til at køre vores servlet og code.

Vi har så vidt muligt anvendt 3 lags arkitektur, Front/Presentations lag, Logik lag og Data Lag. Vi har implementeret en controller class der fungere som bindeled mellem disse lag, så vi undgår for mange kommunikationsveje mellem vores forskellige klasser.

Baggrund:

En butik ønsker at få lavet en webshop til deres cupcake bageri, de har en imponerende cupcake bage maskine der kan lave cupcakes på sekunder. De ønsker en webshop hvor man kan bestille cupcakes til afhentning, for at undgå den lange lange kø der opstår når kunderne ikke kan bestemme sig om hvad de vil ha når de står i butikken.

De har eksperimenteret med at sende cupcakes via pakke post, men det var meget problematisk.

Krav:

Cupcake maskinen har nogle begrænsninger der skal afspejles i den webshop der skal udvikles. En cupcake består altid af en top og en bund, prisen beregnes herefter. Der er mange kombinationer, og de skal kunne vælges imellem. Der skal dog altid være både top og bund.

Kunderne ønskes at ha en kredit hos butikken, som bliver fratrukket den specifikke bruger når en bestilling bliver placeret. Tilføjelse af kredit sker eksternt, så det er ikke en funktionalitet brugeren skal kunne tilgå. Dette skal derimod ske i databasen.

Desuden skal siden kunne håndtere både login og registrering af kunder. Således at en ny kunde kan få en bruger og en kredit balance.

Kunden har også efterspurgt en oversigts funktion som deres medarbejdere kan bruge til at få et overblik over butikkens nuværende status samt gamle ordre. Det har vi tolket som en administrator funktion.

Teknologi valg:

OBS. Se alle teknologier og versionsnumre nedenunder denne tekst

Produktet af dette projekt er (udover denne rapport) at implementere en hjemmeside, der kan bruges til at sælge kager. For at kunne vise denne hjemmeside er hosting firmaet Digital Ocean (www.digitalocean.com) blevet valgt. De giver én muligheden for at have en computer i skyen, så man kan benytte forskellige teknologier, der gør hjemmesiden mulig. Styresystemet på denne computer er linux, som har alle programmerne let tilgængelige.

Backend og frontend er programmeret i Netbeans, hvor Maven sørger for at alle afhængigheder er downloadet og ligger på rette plads. Til versionsstyrings system bruges Git med filerne hosted på github.com. Alle tests er udført med Junit.

For at håndtere og gemme datamængder er MySQL databasen valgt, og til selve Java Serlvet Containeren er benyttet Apache Tomcat. Backenden er programmeret i Java og frontenden består af JSP sider. Siderne benytter HTML og javascript og er stylet med CSS (via bla. Bootstrap).

Til forbindelsen mellem Java og MySQL databasen er den seneste JDBC driver benyttet. OBS. forbindelse til databasen lukker ned efter otte timer og skal herefter genåbnes.

Backup af databasen foretages dagligt af Ubuntu Linux's cronjob system, hvor en SQL-fil gemmes i folderen /backups/mysql. SQL-filen kan bruges til at genoprette databasen og fylde den med indhold.

Server:

Hosting: Digital Ocean 1GB hukommelse 25 GB disk
Operativ system: Ubuntu Linux 18.04.1
Backup: cron 3.0

Backend:

Apache Tomcat 9.0.15
MySQL database 8.0.15
Oracle Java 8
JDBC 8.0.15
Servlet 3.1

Frontend:

HTML 5
JSP 2.3
CSS 3
Javascript ECMAScript 6
Bootstrap 4.3

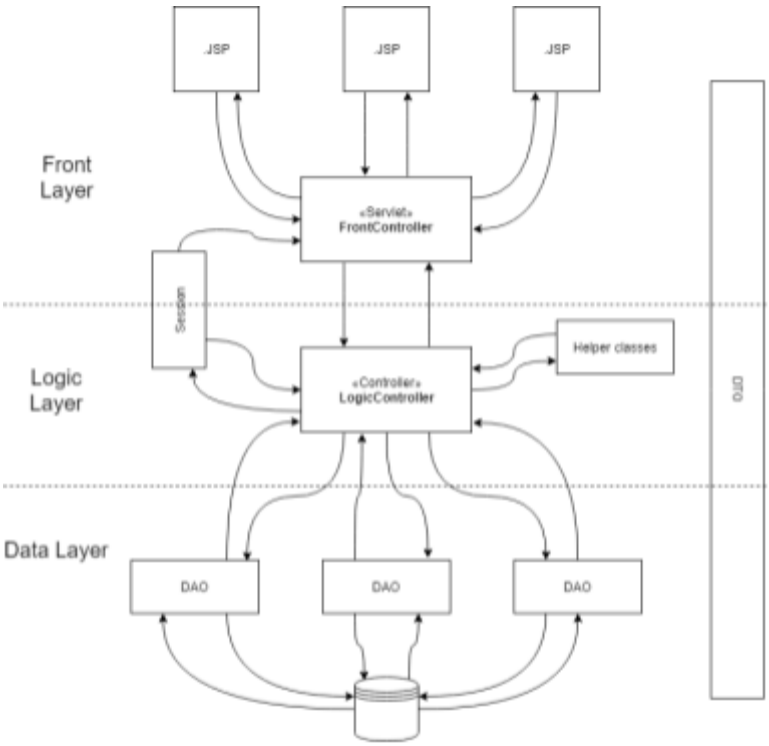
Udviklings-/hjælpe-programmer:

Netbeans 8.2
Java platform JDK 1.8
Maven 4.0.0
Filezilla 3.28.0
JUnit 4.2
Github
Git 2.17.1

Domænemodel og ER-diagram:

Domæne model:

Vi har så vidt muligt prøvet at bruge følgende domæne model (se figur 1. Diagrammet er abstract).



Figur 1. Domæne model

Modellen arbejder i 3 lag. Front, Logic og data. Al kommunikation mellem lagene sker gennem en Controller klasse. Vores DAO (Data Access Objects) fungere som en data mapper for et specifikt objekt, der har en tilhørende DTO (Data Transfer Object).

En DTO er en simpel klasse uden logic, og bliver flittigt anvendt i alle lag, derfor har den ikke nogen egentlig placering i diagrammet.

Sessionen bruges som et mellem objekt i lagene *front* og *logic*, da den indeholder information der skal vises til brugeren, men også data der skal behandles af controlleren. Front controlleren kan ikke skrive til sessionen, da den kun bør håndtere præsentation og behandling af requests.

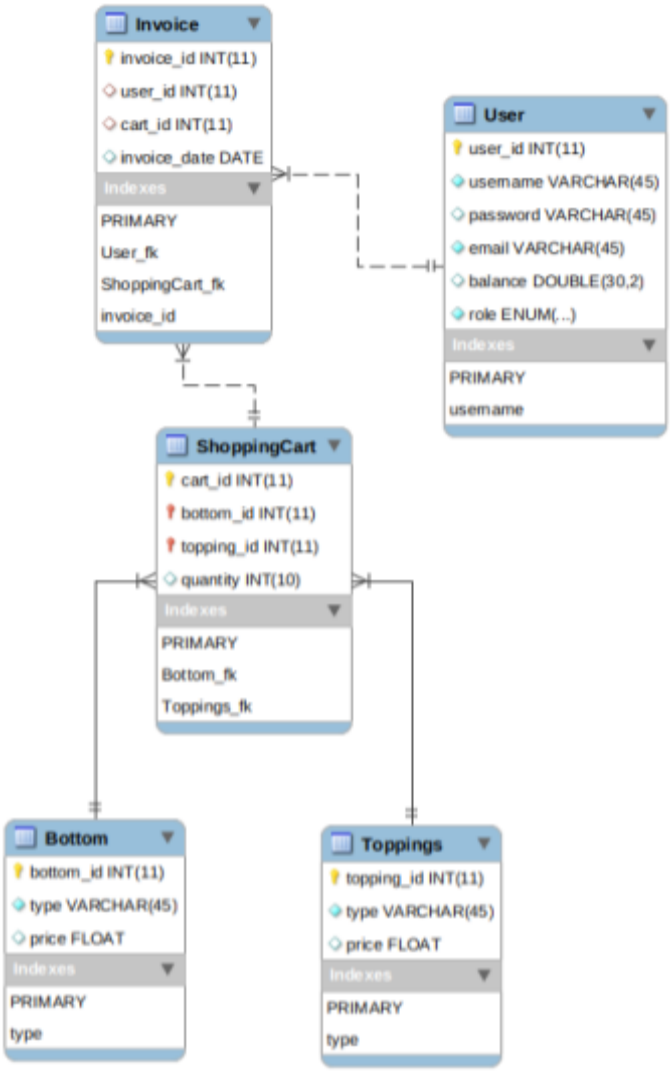
Aktuel Implementation

Grundet at opgaven introducerede os til nye teknologier og metoder (hovedsageligt JSP og Servlets) er det ikke lykkedes at holde os til den ideelle domæne model. Vi har derfor tit JSP og JSPF, der har direkte kontakt med controlleren og Sessionen. Dette ville være et oplagt emne til fremtidig udvikling. Den aktuelle domæne model kan ses i appendix.

ER-diagram:

Et diagram over databasens tabeller og sammenhæng derimellem er vist i figur 2. Alle tabeller er på 3. Normalform.

Alle primærnøgler (på nær den i ShoppingCart) er autogenerede og derved unikke. Eneste sted, hvor det har været nødvendigt at sætte en kolonne til at være unik er i User tabellens Username. Det er gjort for at undgå uheldige situationer med flere brugere med samme navn.



Figur 2. ER diagram

Navigationsdiagram:

Hjemmesiden består af en fælles hoved/menu del der deles af alle frontend sider (se figur 3.). Det er muligt at logge ind (#3 på figuren), hvorefter man kan blive præsenteret for forskellige knapper alt efter hvem der logger ind. Når man er logget ind vises navn (#1) og balance (#2) for brugeren.

Figur 3. Hjemmeside front efter login

Som ny (ikke logget ind-) bruger på siden, præsenteres følgende knapper (den relevante siden navn står i parentes):

Login (login.jsp**)**: Her kan man logge ind, hvis man er en registreret bruger.

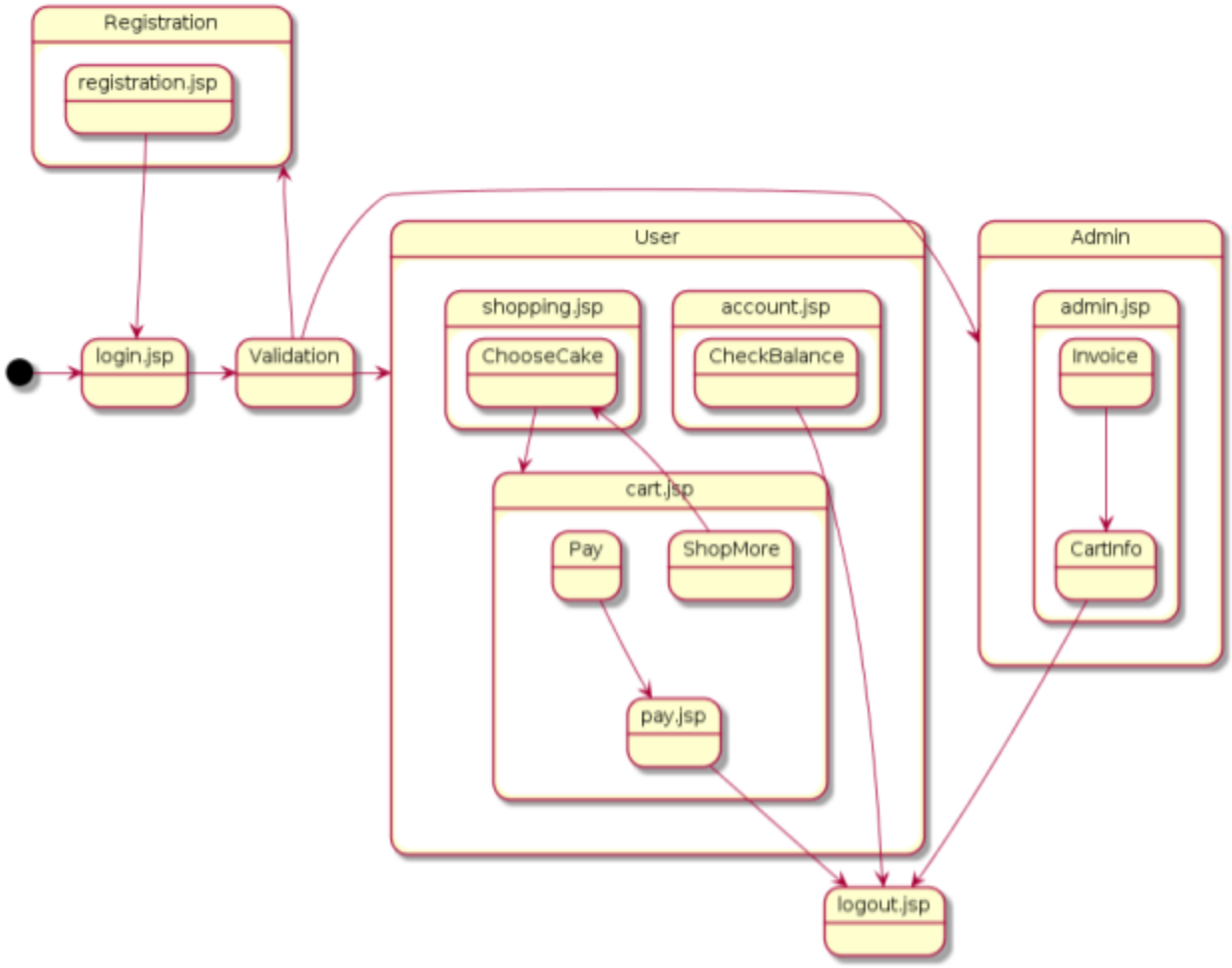
Registrering (registration.jsp**)**: Hvis man ikke er registreret bruger, kan man blive det her.

Som registreret bruger/køber præsenteres følgende knapper:
Logout (logout.jsp**)**: Her kan man logge ud.
Shop (shopping.jsp**)**: På denne side udvælger man sine kager.
Cart (cart.jsp**)**: Her kan man vælge at fortsætte med at shoppe eller fortsætte til betaling.
Account (Account.jsp**)**: Overblik over sin konto får man her.

Som administrator præsenteres følgende knapper:
Samme knapper som den normale bruger, og derudover følgende

Admin (admin.jsp**)**: Admin siden viser alle fakturaer med dato og hvis man klikke på en enkelt faktura dukker indholdet op i en ny tabel.

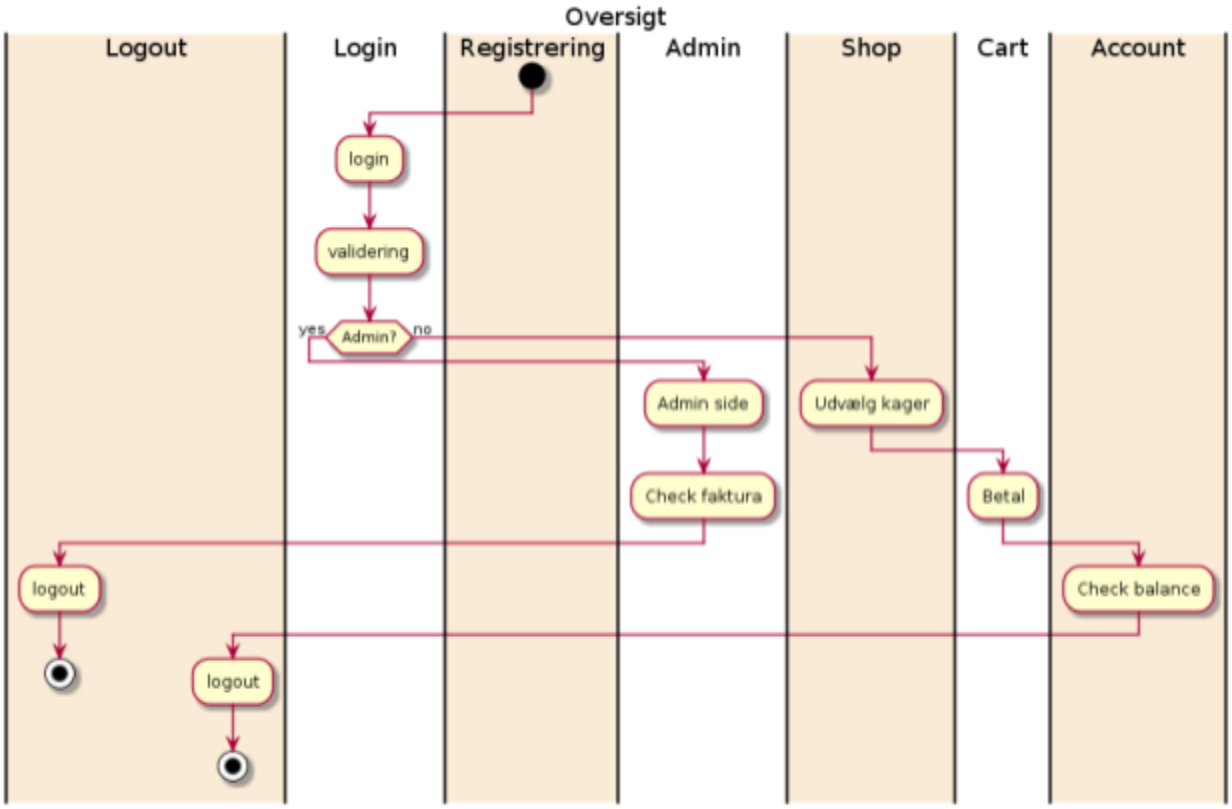
Når man navigerer igennem siden kan flowet være som nedenstående (figur 4). **Validation** vælger hvilken boks man ender i. Enten er man ikke-registeret bruger (hvorefter man ryger til **Registration)**, registreret bruger/køber (**User**) eller administrator (**Admin**).



Figur 4. Oversigt

Man kan også følge flowet i oversigt-billedet nedenunder (figur 5), hvor to typiske brug af hjemmesiden er illustreret (en for admin og en for køber).

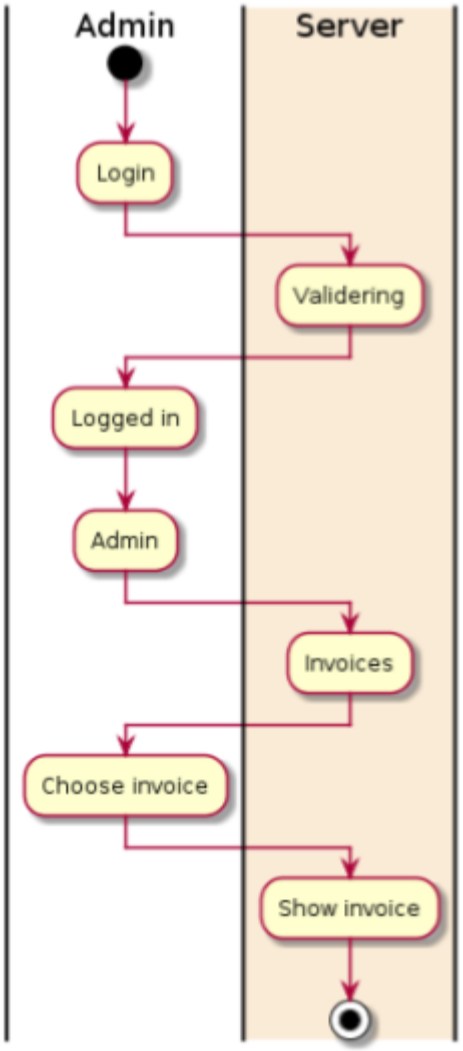
Swim Lane oversigt



Figur 5. Bevægelses mønstre

Navigering fra frontend til backend og return er vist i figur 6. og 7. (igen via et forventet typisk admin flow og et typisk køber flow). **Admin/Client** er browseren (frontend) mens **Server** er backenden (en servlet på en Tomcat server). Der er kun een servlet på serveren, så alle server processer kommer fra denne.

Admin og Client Server Kommunikation



Figur 6. Admin flow

Figur 7. Klient flow

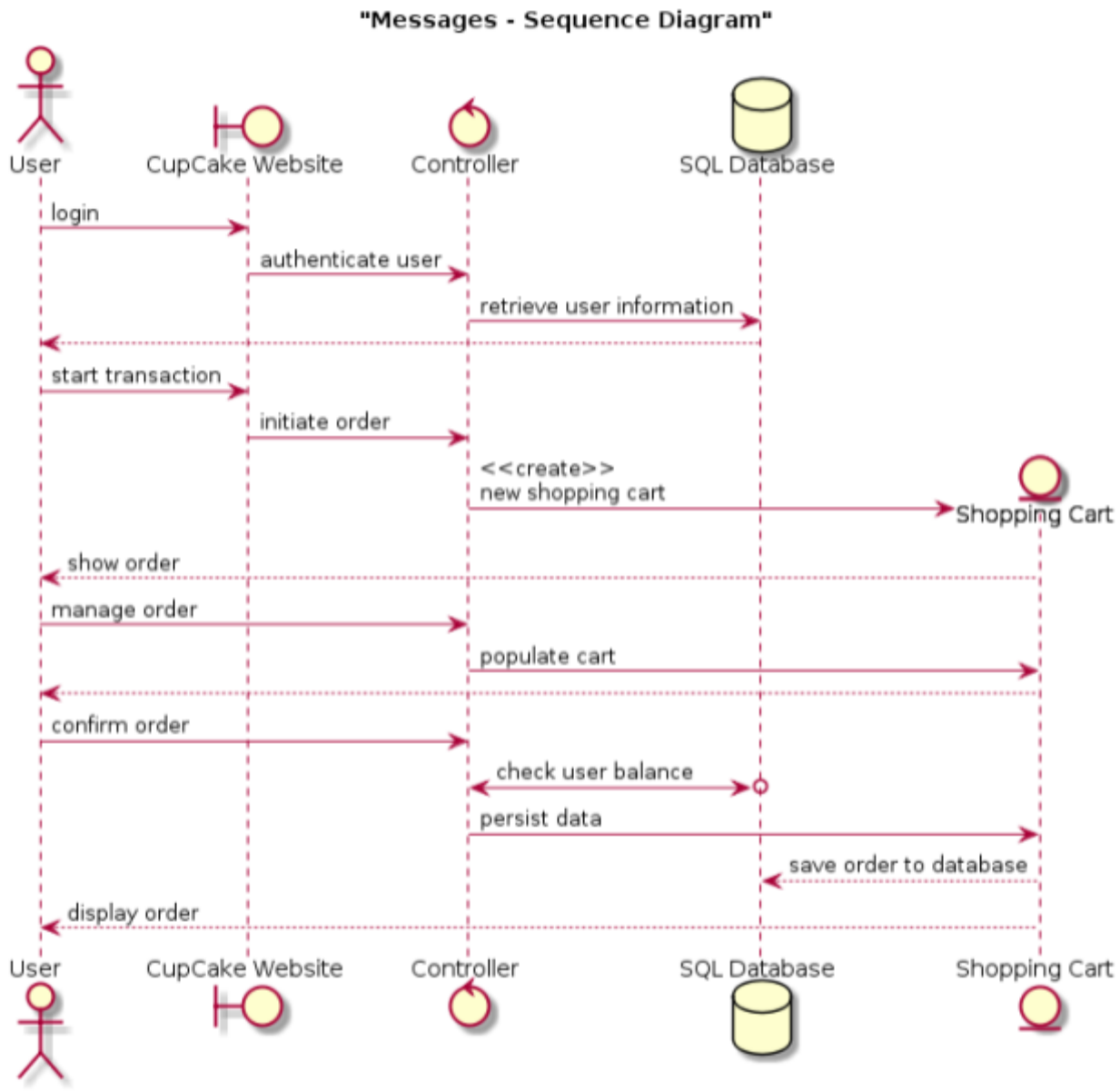
Sekvens diagrammer:

I figur 8. ses eksempler på logikken bag nogle brugervalg. Specifikt et login, en transaktion, en ordre håndtering og ordrebekræftelse:

Login:
Brugeren præsenteres for en login side. Login siden sender login detaljer videre til controlleren, der henter information fra databasen. Resultatet af denne søgning og validering sendes tilbage til brugeren.

Transaktion:
Brugeren præsenteres for en side, hvor vedkommende kan foretage en tranaktion. Når denne transaktion starter oprettes en indkøbskurv, hvor evt. f.ere køb kan opsamles i. Indkøbskurven vises herefter til kunden.

Ordre håndtering:
Når brugeren accepterer at betale for en indkøbsvogns indhold, foretages et check for at se om der er nok indhold på hans konto. I bekræftende tilfælde, gemmes vognen i databasen og en ordrebekræftelse sendes tilbage til kunden. Hvis der ikke er nok penge, kommer der en besked med dette.



Figur 8. Sekvensdiagram

Særlige forhold:

Sessionen: Vi bruger sessionen til at gemme informationer om brugeren, som fx login-status, så vi med simple tjek på hver side kan vise brugeren den information der er relevant for brugeren i det øjeblik. Derudover gemmer vi information om hvilke varer brugeren har i sin indkøbskurv på det givne tidspunkt. Så brugeren kan bruge resten af siden uden at afslutte deres bestilling (med fx at tilføje flere cupcakes til indkøbskurven)

Exceptions: Exceptions der opstår i databasen bliver sendt videre op igennem lagene indtil de rammer et punkt hvor de skal håndteres. For eksempel at hvis vi får en SQL-Exception i databasen, kan vi sende den videre op til controlleren som kan kommunikere til fronten at passwordet er forkert eller brugernavnet allerede eksistere i databasen, afhængigt af hvor fejlen opstår i brugerens session.

Brugervalidering: Validering af brugerinput bliver håndteret af HTML koden i form af et required-tag der sikrer os at der ikke bliver sendt tomme strenge videre. Yderligere håndteres brugernavne som unikke, så der ikke kan oprettes flere brugere med samme brugernavn. Skulle man taste sine informationer ind forkert under login, bliver får man at vide at fx sit password er forkert. Hvis ens brugernavn er indtastet forkert så får man at vide at brugeren ikke eksisterer i databasen, og man bliver bedt om at oprette en bruger.

Under oprettelse af brugere bliver email-formatet håndteret som en email gennem vores HTML.

Fremtidigt vil der nemt kunne implementeres tjek af email med fx Regular Expressions.

Database: I databasen har vi primært brugt floats og varchars. Da vores database skal kunne håndtere Strings og kommatal fra jdbc.
I forhold til sikkerhed er det ikke godt. Da alle personer med adgang til vores database kan læse al information om de brugere der er oprettet i databasen. For at løse dette skal der implementeres en bedre løsning, for eksempel gennem kryptering.

Status på implementation:

shopping.jsp: Siden er i (næsten) ren html, så den kunne bruge lidt styling. Overholder indtil videre ikke konventionen om ikke at skrive til controlleren.
cart.jsp: Det er ikke muligt at foretage køb selvom der er en knap der beskriver man kan købe. Cart.jsp er ikke ordnet, så den er let at vedligeholde og der er noget redundant kode efter hjemmesiden har forandret sig. Desuden holder den sig ikke til design konventionen om ikke at kommunikere med andre lag.

admin.jsp: Der er en bug, der gør, at hvis man vælger en invoice (#2 eller #7) i den samlede invoice tabel, kommer der noget volapyk i visningen af brugerens indkøbsliste (#2) eller intet (#7).
Account.jsp: Der mangler styling på balance mm. Der er en placeholder for muligheden for at se invoices (kan benyttes af alle brugere, men det burde være sådan at brugeren kun kan se sin egne invoices). Der er en placeholder knap for at kunne tilføje kredit til sin konto. Dette er dog en funktionalitet der ikke er ønsket, men som senere kunne være god at bruge. Især fordi det er upraktisk at skulle rette det direkte i databasen.

Add balance knappen virker ikke og fremprovokerer et crash af hjemmesiden.

Test

For at fjerne belastning ift. videre udvikling, ville implementering af unit test være nødvendigt. Det ville beskytte systemet mod at få pushet updates der ville ødelægge nødvendige funktionaliteter. I vores nuværende projekt har vi ikke implementeret tests.

Vi vil ikke implementere test af getters og setters, da de ikke indeholder logik eller anden kode, af samme årsag vil vi ikke teste vores DTO'er.

Klasse	Metoder	Status	Dæknings-grad
logic.Controller	fetchBottoms fetchToppings createUser loginCheck fetchUser fetchCart fetchInvoiceList fetchInvoice fetchTotalPrice isCupCakeDuplicate putCartInDB cancelOrder getInvoiceID	Ikke implementeret	
data.DAO.CupcakeDAO	getBottoms getBottom getToppings getTopping	Ikke implementeret	

	addBottom addToppings addToDB		
data.DAO.InvoiceOrderDAO	InvoiceOrderDAO saveShoppingCartToDB calculateTransactionCost payForOrder createPlaceholderInvoiceInDB saveOrderToDB cancelOrder	Ikke implementeret	
data.DAO.UserDAO	UserDAO getUser getUser createUser	Ikke implementeret	

Da vi i mange af vores metoder bruger HttpServletRequests til at kommunikere med brugeren skal der i flere af funktionerne, i sær i vores Controller implementeres en mock-klasse af HttpServletRequest for at kunne imitere kommunikation med en server. Der ville for eksempel kunne importeres mock-klasser fra Spring eller Mockito, der kunne også skrives sin egen mock-HttpServletRequest.

Med vores nuværende implementation ville det være nødvendigt at køre en del test af vores jsp og vores jsp fragmenter. Da disse udfører en del (front)control opgaver. Især vores cartController.jspf kalder en del funktionalitet som ved videreudvikling ville kunne returnere en del fejl under anvendelse.

Bilag:

Aktuelt klassediagram