Hold B Gruppe1 - April 2020

Projekt Cupcake

Elever	Email	Github
Mathias Drejer	Cph-md266@cphbusiness.dk	https://github.com/Drejse
Mikkel Dahl Jacobsen	cph-mj838@cphbusiness.dk	https://github.com/mdjac
Christian Schaadt Rosenbæk	cph-cr272@cphbusiness.dk	https://github.com/rosenbaek/sem2-cupcake

Indholdsfortegnelse

Indledning

Opgaven går ud på at lave en webshop til Olsker Cupcakes. Kunden ønsker en webshop, hvor det er muligt at bestille samt betale for cupcakes for derefter at afhente cupcakes i Olskers butik. I rapporten finder du materiale og overvejelser fra vores møde med kunden samt en beskrivelse af vores løsningsforslag.

Baggrund

Olsker Cupcakes er en virksomhed bosat på Bornholm. Det er en lille butik, som ønsker at forbedre deres forretning ved tilføjelse af en webshop der kan øge deres online salg. Vi har modtaget en løs skitse fra kunden omkring noget tænkt design og funktioner heriblandt:

- Kunden skal kunne sammensætte en cupcake ud fra et udvalg af bunde og toppe.
- Kunden skal kunne oprette en profil for at kunne betale samt se tidligere ordredetaljer.
- Der skal være en admin side med mulighed for at danne et samlet overblik.
- Olsker ønsker en kredit ordning så kunden kan betale for sin ordre.

Teknologi valg

Webshoppen er udviklet som en multipage application ved hjælp af følgende teknologier:

IDE: IntelliJ 2020.3
Sprog:
- Java 1.8
- JSTL
- HTML 5
- CSS

Database: MySQL v.8
Database Driver: JDBC
Framework: Maven
Server: Tomcat 9.0.44

Krav

Olsker Cupcakes håber på at øge deres omsætning ved at tilføje muligheden for online salg ved hjælp af en webshop. Vores første møde med Olsker Cupcakes mundede ud i følgende userstories:

Userstories:

US-1: Som kunde kan jeg bestille og betale cupcakes med en valgfri bund og top, sådan at jeg senere kan køre forbi butikken i Olsker og hente min ordre.

US-2 Som kunde kan jeg oprette en konto/profil for at kunne betale og gemme en ordre.

US-3: Som administrator kan jeg indsætte beløb på en kundes konto direkte i MySql, så en kunde kan betale for sine ordrer.

US-4: Som kunde kan jeg se mine valgte ordrelinjer i en indkøbskurv, så jeg kan se den samlede pris.

US-5: Som kunde eller administrator kan jeg logge på systemet med e-mail og kodeord. Når jeg er logget på, skal jeg kunne min e-mail på hver side (evt. i topmenuen, som vist på mockup'en).

US-6: Som administrator kan jeg se alle ordrer i systemet, så jeg kan se hvad der er blevet bestilt.

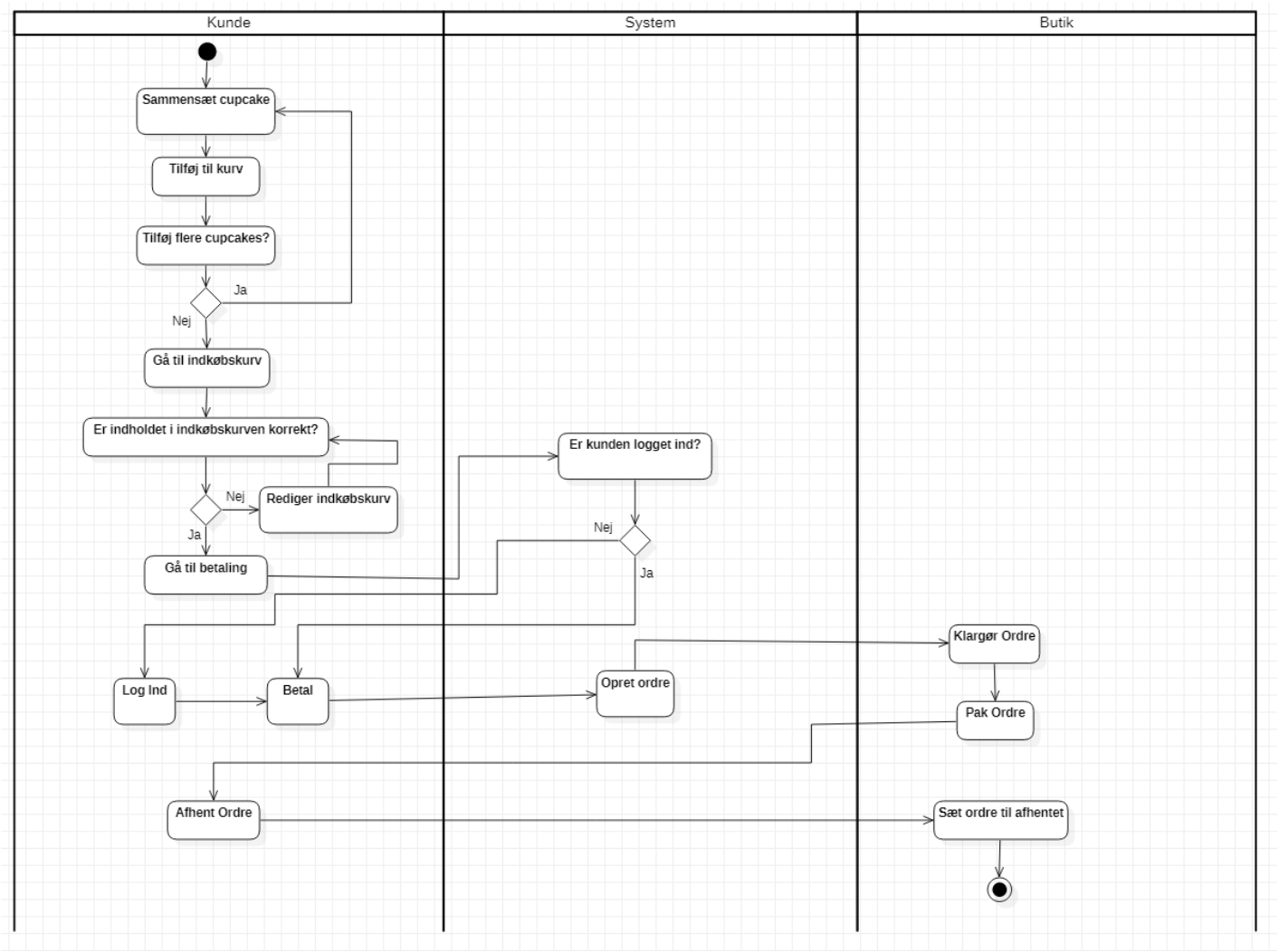
US-7: Som administrator kan jeg se alle kunder i systemet og deres ordrer, sådan at jeg kan følge op på ordrer og holde styr på mine kunder.

US-8: Som kunde kan jeg fjerne en ordre fra min indkøbskurv, så jeg kan justere min ordre.

US-9: Som administrator kan jeg fjerne en ordre, så systemet ikke kommer til at indeholde ugyldige ordrer. F.eks. hvis kunden aldrig har betalt.

Aktivitetsdiagram

Vi har valgt at lave et aktivitetsdiagram der beskriver købs flowet som kunder vil gennemgå når de skal bestille cupcakes hos Olsker Cupcakes.
Diagrammet er lavet ud fra den antagelse at en kunde typisk vil logge ind når de er klar til at betale og oprette ordren, det er dog også muligt i vores løsning at logge ind undervejs eller inden man starter det beskrevne flow.

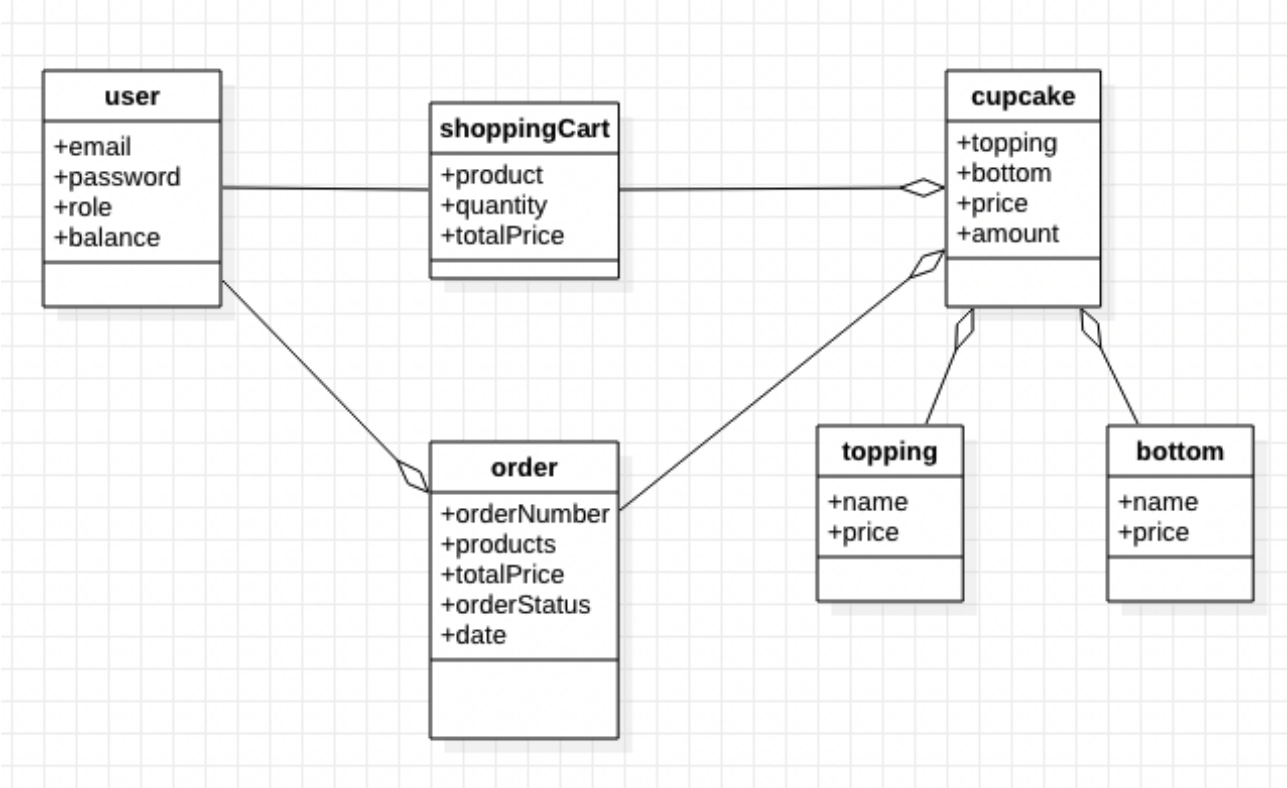


Domæne model og ER diagram

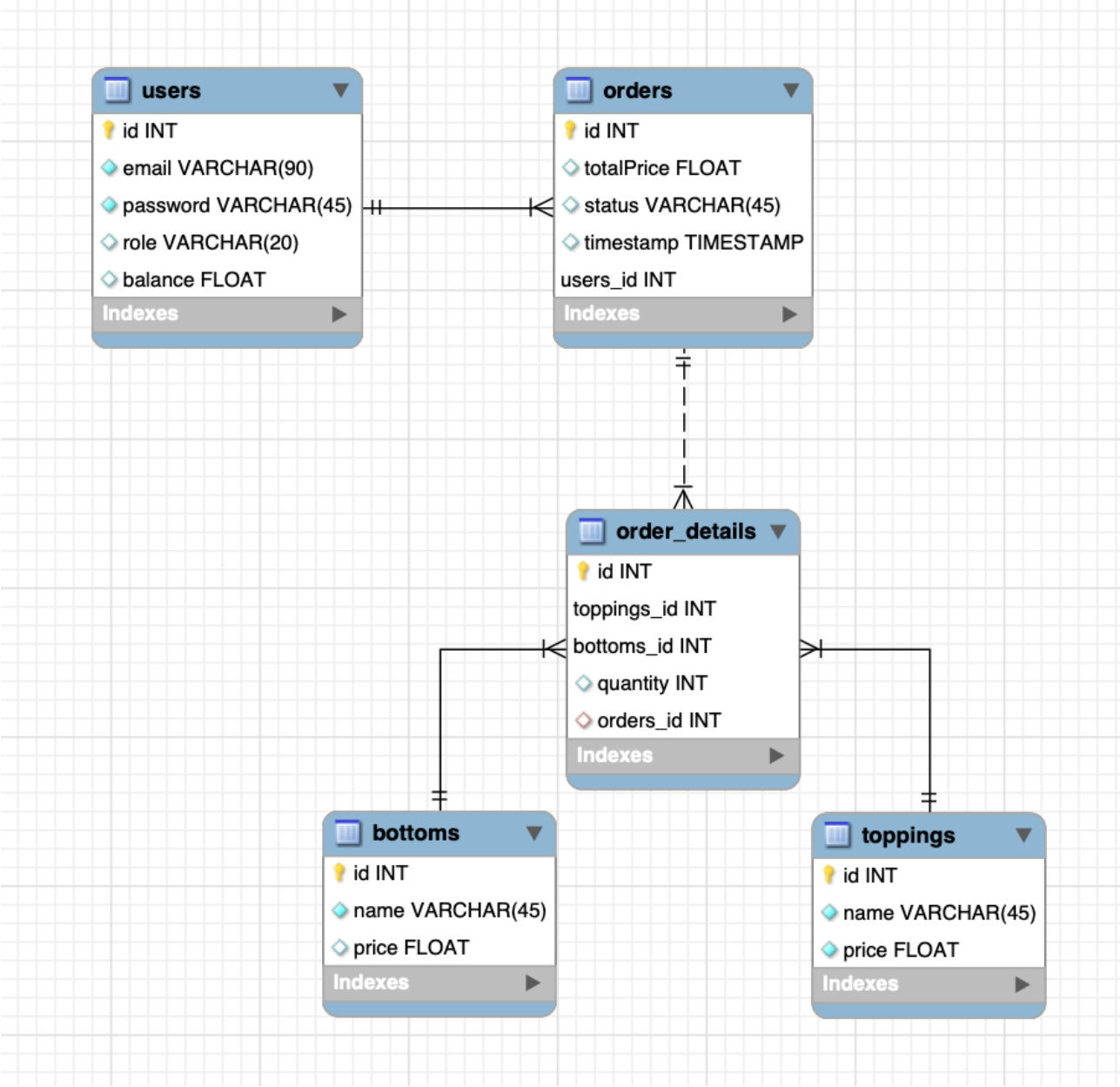
Domænemodel:

Når du kommer inde på webshoppen som bruger, kan du med det samme begynde at tilføje cupcakes til indkøbskurven. Du behøver ikke at være logget ind fra starten.

- Én bruger kan have mange ordre.
- Én ordre består af én eller flere cupcakes.
- Én cupcake består af én topping og én Bottom.
- Én Topping/Bottom kan være en del af mange cupcakes.



ER diagram


users
id INT
email VARCHAR(90)
password VARCHAR(45)
role VARCHAR(20)
balance FLOAT
Indexes
orders
id INT
totalPrice FLOAT
status VARCHAR(45)
timestamp TIMESTAMP
users_id INT
Indexes
order_details
id INT
toppings_id INT
bottoms_id INT
quantity INT
orders_id INT
Indexes
bottoms
id INT
name VARCHAR(45)
price FLOAT
Indexes
toppings
id INT
name VARCHAR(45)
price FLOAT
Indexes

Vores database er som udgangspunkt bygget efter 3. normalform. Der er dog 3 undtagelser:

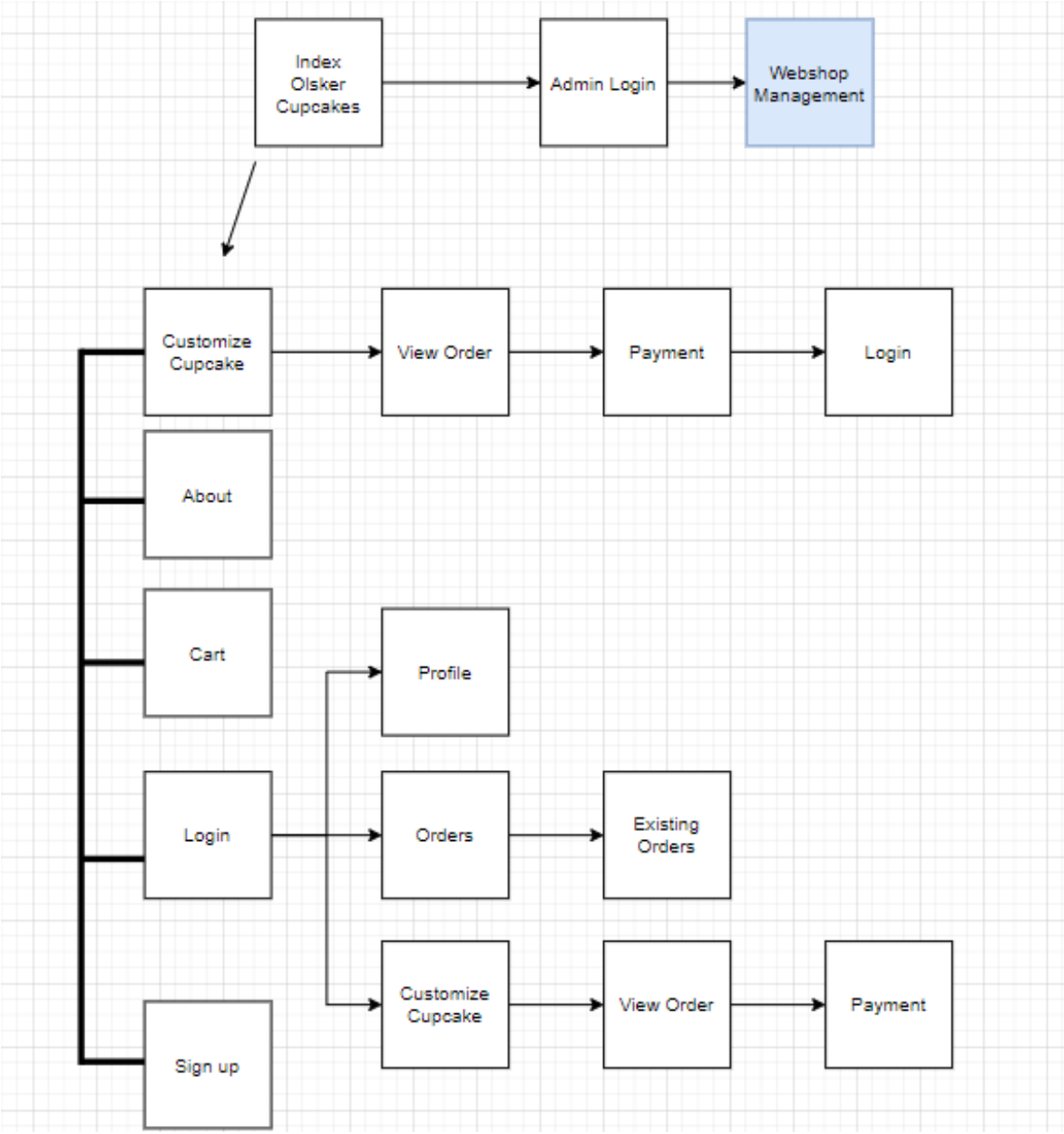
- *'totalPrice' i orders tabellen:*
  Årsagen til, at vi har valgt ikke at følge 3. normalform i dette tilfælde er, at hvis Olsker Cupcakes i fremtiden vil lave priserne om på deres toppings/Bottoms, så vil totalpriserne på tidligere ordre være forkert, i forhold til hvad kunden faktisk har betalt. Havde vi haft mere tid, så kunne dette løses ved, at vi lavede endnu en tabel som indeholdt transaktioner. Den tabel skulle for hver transaktion have en unik nøgle, som skal kunne referere til *users_id* og orders_id. Til slut mangler vi bare at skulle fjerne '*totalPrice*' fra *'orders'* også vil vores database overholde 3. normalform

- 'role' *i users tabellen:*
  Vi har valgt ikke at benytte 3. normalform her, da vi mener at det er lidt voldsomt at lave endnu en tabel som indeholder rollerne til dette lille projekt. Ved denne løsning vil det være et problem hvis man skriver rollen forkert. Man kunne løse dette på en anden måde ved hjælp af Enum i backend.

- *'status' i orders tabellen:*
  Når man ser på vores ER diagram ovenfor, så følger vi ikke 3. normalform, da 'status' ikke har et unikt id i egen tabel. Vi har dog løst dette ved at inkorporere Enum til at styre den værdi vi skyder ind i 'status'.

Denne database er bygget meget specifikt op til cupcakes. Hvis Olsker Cupcakes i fremtiden vil udvide deres sortiment til andet end cupcakes, så bør databasen laves om, således at '*order_details*' ikke indeholder topping og Bottoms. Disse 2 henvender sig kun til cupcakes.

Navigationsdiagram



Gennemgående på webshoppen har vi en navigationsbar i toppen. Det varier fra om det er en kunde der benytter siden, eller om det er en employee eller admin, da en admin har andre muligheder end kunden som benytter webshoppen.
Som ny bruger på websitet, er det første der forekommer en forside. Allerede fra start har du som kunde mulighed for at påbegynde køb af cupcake(s), når dette er gjort, kan vedkommende dernæst vælge at se ordrerne indkøbskurven, og derefter gå til betaling. Er brugeren ikke logget ind, vil du blive dirigeret til loginsiden først og derefter bliver kunden dirigeret til betalingssiden.

Når kunden på webshoppen har oprettet en bruger, vil vedkommende kunne se sin købshistorik under Orders. Dette kræver at være logget ind.

Admin kan tilgå en admin side, hvor vedkommende bla. Har en oversigt over ordrer, samt kunder i systemet og kan ligeledes tilføje kredit til kunder.

Når man rammer vores hjemmeside, så rammer man som det første vores frontcontroller. Frontcontrolleren har et filter som tjekker på om brugeren er logget ind, det hjælper os med at kunne gemme sider, som ikke er tilegnet en bruger der ikke er logget ind. Frontcontrolleren er altså den som dirigerer alle forespørgsler. Vi har defineret de kendte kommandoer og dertilhørende servlets i et HashMap, som bruges af frontcontrolleren til at dirigere de indkomne forespørgsler. Hvis man frontcontrolleren støder på en kommando som ikke er til stede i dette hashmap, så bliver brugeren sendt til en fejlside.

JSP-sider:

- Index.jsp
- Loginpage.jsp
- Aboutpage.jsp
- AdminAllOrderspage.jsp
- Customerpage.jsp
- Employeepage.jsp
- Paymentpage.jsp
- Profilepage.jsp
- Registerpage.jsp
- Shoppingcart.jsp
- UserOrderHistory.jsp
- Errorpage.jsp

Servlets:

```
commands = new HashMap<>();
commands.put("index", new CommandUnprotectedPage( pageToShow: "index"));
commands.put("loginpage", new CommandUnprotectedPage( pageToShow: "loginpage"));
commands.put("logincommand", new LoginCommand( pageToShow: ""));
commands.put("logoutcommand", new LogoutCommand( pageToShow: ""));
commands.put("registerpage", new CommandUnprotectedPage( pageToShow: "registerpage"));
commands.put("registercommand", new RegisterCommand( pageToShow: ""));
commands.put("customerpage", new CommandProtectedPage( pageToShow: "customerpage",  role: "customer"));
commands.put("employeepage", new EmployeePageCommand( pageToShow: "employeepage",  role: "employee"));
commands.put("addtoshoppingcart", new ManageShoppingCart( pageToShow: "index"));
commands.put("shoppingcart", new CommandUnprotectedPage( pageToShow: "shoppingcart"));
commands.put("gotopayment", new PaymentCommand( pageToShow: "paymentpage",  role: "customer"));
commands.put("paynow", new PayNowCommand( pageToShow: "userorderhistory",  role: "customer"));
commands.put("removefromshoppingcart", new RemoveFromShoppingCart( pageToShow: "shoppingcart"));
commands.put("addcredittouser", new AddCreditToUserCommand( pageToShow: "employeepage",  role: "employee"));
commands.put("showorders", new ShowOrdersCommand( pageToShow: "userorderhistory"));
commands.put("removefromorders", new RemoveFromOrders( pageToShow: "adminallorderspage",  role: "employee"));
commands.put("changeorderstatus", new ChangeOrderStatusCommand( pageToShow: "employeepage",  role: "employee"));
commands.put("aboutpage", new CommandUnprotectedPage( pageToShow: "aboutpage"));
commands.put("profilepage", new CommandUnprotectedPage( pageToShow: "profilepage"));
commands.put("editprofile", new EditProfileCommand( pageToShow: "profilepage"));
```

Særlige forhold

Application Scope:
I vores Applications scope har vi valgt at gemme data som ikke ændres regelmæssigt og som ikke er følsomt data. Bottom, Toppings & status (*Paid* eller *PickedUp*) hentes fra databasen og bliver sat ind i Application scope gennem Front Controlleren.

Session Scope:
Vi gemmer vores *shoppingcart* samt user I session scope da de kun skal være i live imens sessionen er i gang. På den måde kan vi tilgå de relevante informationer på tværs af applicationen.

Exceptions:
Generelt nå vi griber en exception, så gemmer vi til slut en fejlbesked i request scope som ”error”. På den måde kan vi i de relevante jsp sider tjekke på om ”error” er tom. Hvis den IKKE er tom, så viser vi fejlbeskeden til brugeren. Desuden har vi lavet det sådan, at hvis der sker en fejl under oprettelse af en ordrelinje i database, så sletter vi den resterende del af den ordre i *’orders’*. På den måde har vi ikke halvfærdige og ugyldige ordre i vores database.

Validering af brugerinput.
Vi håndterer brugerinput i forskellige scenarier:
- Ved oprettelse af ny bruger, tjekker vi i databasen om der forekommer duplikering, findes der allerede en bruger med samme e-mail, forekommer en fejlbesked. Ligeledes, hvis man prøver at logge ind med ugyldig e-mail eller password, så forekommer en fejlbesked.
- Hvis en admin forsøger at manipulere med data i databasen som ikke findes f.eks. et bruger ID, forekommer en fejlbesked.

Sikkerhed:
Vi har ikke tilføjet yderligere sikkerhed i forbindelse med login. Vi havde planlagt at kryptere passwords i database, men nåede ikke dette på grund af sygdom.

Status på implementation

Eftersom, at vi kom godt fra start ift. At have et solidt overblik over projektet gjorde vi det nemmere for os selv at sørge for, vi fik implementeret det som vi ønskede at have med i projektet. Ift. Vores navigationsdiagram har vi alle siderne med i projektet og selve funktionaliteten fungerer, som ønsket.

Vi har nogle få ting vi ønskede at ændre lidt på, men er generelt set tilfredse med resultatet af vores webshop. Det er ting, som f.eks at kunne ændre sin ordre når man har lavet en bestilling. På nuværende tidspunkt er det kun muligt at fjerne bestillingen helt – hvilket måske ikke er helt så hensigtsmæssigt.

Derudover, ønskede vi at udfordre os selv lidt i form af en form for transaktions mulighed i stedet for, at man som admin skal ind og tilføje kredit til brugeren. Men dette er noget der ligger ude for opgaven, og som egentlig bare var ekstra til os selv.

Vi er meget tilfredse med udseendet af vores webshop, dog med lidt mere tid, kunne det være sjovt at sørge for at selve shoppen fik et lidt mere professionelt touch.

Proces

Som det første i vores proces oprettede vi et kanban board inde på vores github repositatory. Her fik vi et generelt overblik over de forskellige userstories og ønskede funktionalitet. Vores plan var at dele opgaven op, så vi undgik at sidde foran én skærm og dermed øge vores produktion.

Da vi begyndte at kode, delte vi de lidt mindre arbejdsopgaver op, men da funktionaliteten blev mere kompleks, samlede vi os og livekodede sammen, for at undgå problemer med blandt andet git og sammensætning af hele applikationen.

Til næste gang vil vi prøve at dele endnu mere op, så vi kan få lavet mere på kortere tid samt så den enkelte kan få lov til at tænke mere over problemløsningen selv. Dette vil gøre git og sammenkoblingen mellem de forskellige dele af applicationen mere uforudsigelig, så det vil kræve god kommunikation og struktur.