



Olsker Cupcake

28. april 2021

Link to GitHub project: <https://github.com/Mathias2860DK/Cupcake-Project>

Mathias

Mail:

cph-mp557@cphbusiness.dk

GitHub:

<https://github.com/Mathias2860DK>

Mustafa

Mail:

cph-mt357@cphbusiness.dk

GitHub:

<https://github.com/MustafaTokmakci>

Markus

Mail:

cph-ma587@cphbusiness.dk

GitHub:

<https://github.com/TheAgns>

Indledning	1
Baggrund og krav	2
Teknologivalg	4
Aktivitetsdiagram	4
Domæne model og ER-diagram	5
Særlige forhold	10
Status på implementering	11
Tests	11
Proces/konklusion	12

Indledning

Vi har i dette projekt fået stillet til opgave, at lave et website til Olsker Cupcakes, der skal være med til at nå ud til flere kunder, end de ellers ville have kunne. Folkene bag Olsker Cupcake ønsker optimering af deres forretning og ved at komme på digitale marked, kan firmaet nå ud til flere kunder, reducere ventetiden i butikken og have et bedre overblik over forretningen. Alle ordre skal nu gemmes i en database, som medarbejderne skal have adgang til.

I dette projekt har vi allerede fra start fået lidt startkode at gøre godt med. Det kan ses her: <https://github.com/jonbertelsen/sem2-startcode> . Denne startkode tager udgangspunkt i forskellige design patterns:

- Model View Controller (MVC pattern)
- Frontcontroller pattern
- Command pattern

- Singleton pattern
- Facade pattern
- Dependency injection

På websitet ønsker Olsker Cupcake en "employee" rolle og "customer" rolle til at styre websitet. En "employee" skal have adgang til alle brugere, ordre og "orderlines" via websitet (En orderline fungerer som produkter i kurven). Her skal være mulighed for at slette ordre(r) i tilfælde af fejlbestilling eller lign. "Employee" kan derefter tilføje penge tilbage på brugerens konto.

Som "customer" skal man kunne oprette en profil, designe sin egen cupcake og lægge den i sin kurv. Kunden skal kunne slette orderlinjer fra sin kurv og se den totale pris. I og med at der ikke er ønsket konkret betalingsløsning, har vi lavet et kreditsystem, som kunden kan betale med, og "employee" kan give kredit ved efterspørgsel.

Baggrund og krav

Cupcake shoppen Olsker Cupcakes har i en længere periode overvejet at skalere deres butik. Derfor har de kontaktet os. Det skal nu være muligt for medarbejderne at se ordrene på hjemmesiden, så cupcakesne kan være klar, allerede inden kunden er trådt ind i butikken. Det vil skabe bedre overblik hos medarbejderne, og de kan via hjemmesiden se hvornår cupcakesene er bestilt, og dermed også hvornår de skal være klar. Nu skal det være slut med ventetid og tid til forandring, der skaber overblik.

Vi har fået et par user-stories med, så vi bedre kan forstå hvor kunden vil hen. Vi har selv videre formidlet disse user-stories til Java sprog og implementeret dem. Hvor langt vi er kommet med user-stories og status på dem kan ses under "Status på implementation".

User stories:

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 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 ordrelinier 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 se email 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.

Teknologivalg

Her er et overblik over de teknologier der er blevet anvendt til projektet.

IntelliJ Ultimate Idea - Version 2020.3

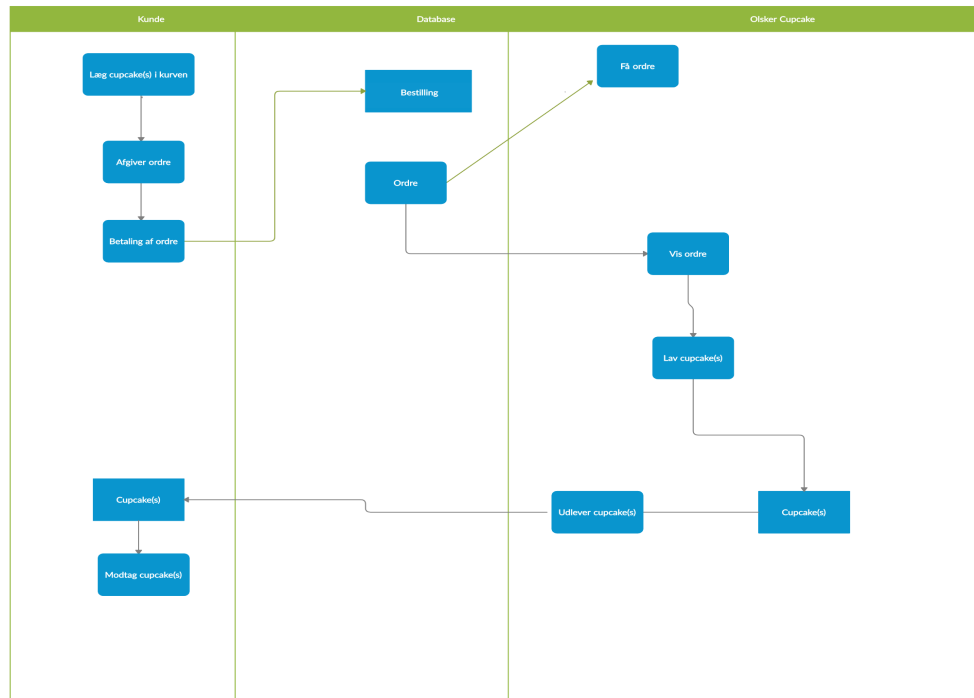
Java 8 - Version 1.8

MySQL Connector - Version 8.0.19

Ubuntu - Version 20.04

Apache Tomcat - Version 9.0.22

Aktivitetsdiagram

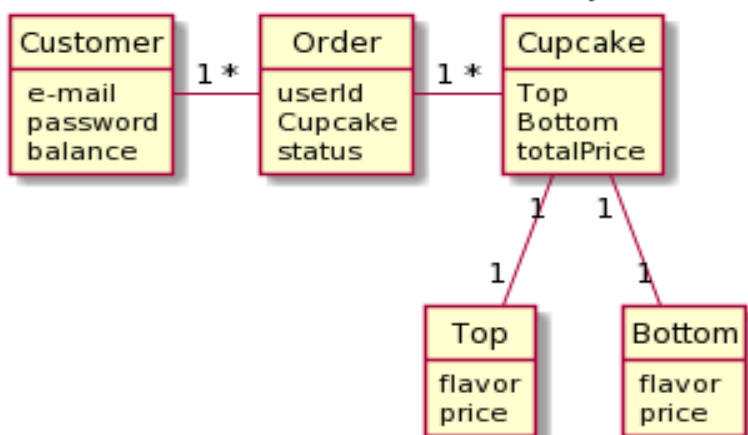


Vi antager at kunden allerede er logget ind på vores webside, da der ellers ikke ville være særlig meget funktionalitet. Kunden kan selv sammensætte sine cupcakes med top og bund, lægge den i kurven for derefter at kunne se den samlede pris. Hvis kunden ikke er færdig med at shoppe cupcakes, kan kunden vende tilbage og lægge flere cupcakes i sin kurv, inden kunden trykker "pay". Olsker Cupcake er også vores administrator bruger som kan se de ordre, som er blevet betalt for derefter at kunne gå i gang med at lave dem.

Domæne model og ER-diagram

Domænenemodel:

Domænenemodel Olsker Cupcakes

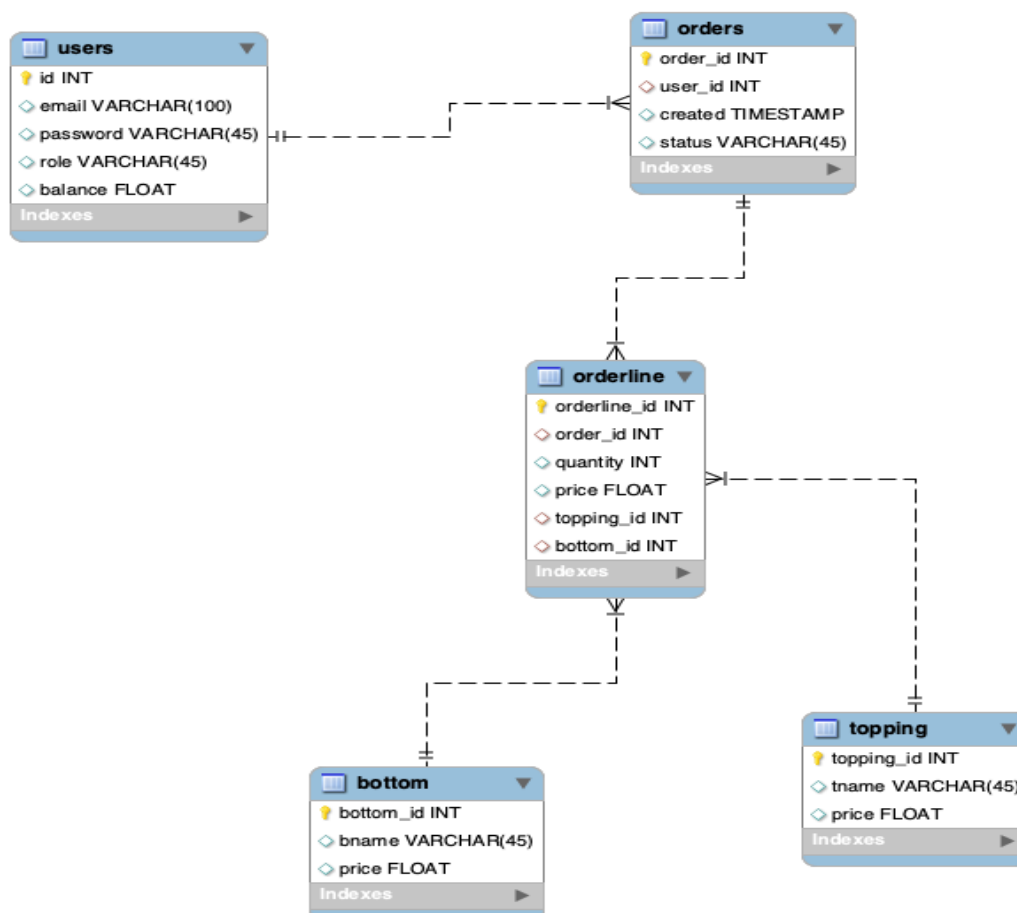


Forklaring af model

- En *customer* kan have flere *ordre*, men en *ordre* kan kun være tilknyttet en *customer*.
- En *ordre* kan indeholde flere cupcakes, det kan være flere af den samme, flere forskellige eller en kombination af disse. Dog kan en *cupcake* kun være knyttet til en *ordre*.
- En *cupcake* kan have en *top*, og en *top* er knyttet til en *cupcake*. Da en *cupcake* består af en *top* og en *bottom*, gælder dette også for *bottom*.
- Noget af det vi her har overvejet mest, er hvordan vi skal have *Ordre* og *Cupcake* til at give mest mening og deres forhold til hinanden.

Tanken bag af det viste domæne er, at en kunde kan have en ordre, som består af et antal cupcakes, som kunden selv vælger ud fra *top* og *bottom*. Cupcakesene skal desuden fungere som *orderlines* i vores kode, som skal hjælpe os med at holde styr på de forskellige sammensætninger af *top* og *bottom*, og hvilke *ordre* de tilhører, altså hvilken *customer* de tilhører, da en *ordre* er tilknyttet en *customer*.

ER diagram



Primær nøgle/PRIMARY KEY: En primær nøgle definerer et bestemt felt i et table. Ét table må kun have en primær nøgle og nøglen må ikke være NULL, NULL betyder at den SKAL indeholde noget. Alle de andre felter i et table er afhængige af primærnøglen.

Fremmed nøgle/foreign key: En fremmed nøgle er et eller flere felter i et table som identificerer en række i et andet table. Det table som fremmednøglen er defineret i, kalder man for child table og refererer til en primær nøgle i et andet table.

UNIQUE: er et eller flere sæt af kolonner i et table, som identificerer en registrering i et table. En unik nøgle gør at værdien i en kolonne er unik i hele det table.

Users: Indeholder, user_id (PRIMARY) som samtidig er Auto Increment, email (UNIQUE), password, balance, role. Der er en til mange relation, imellem users table og order table.

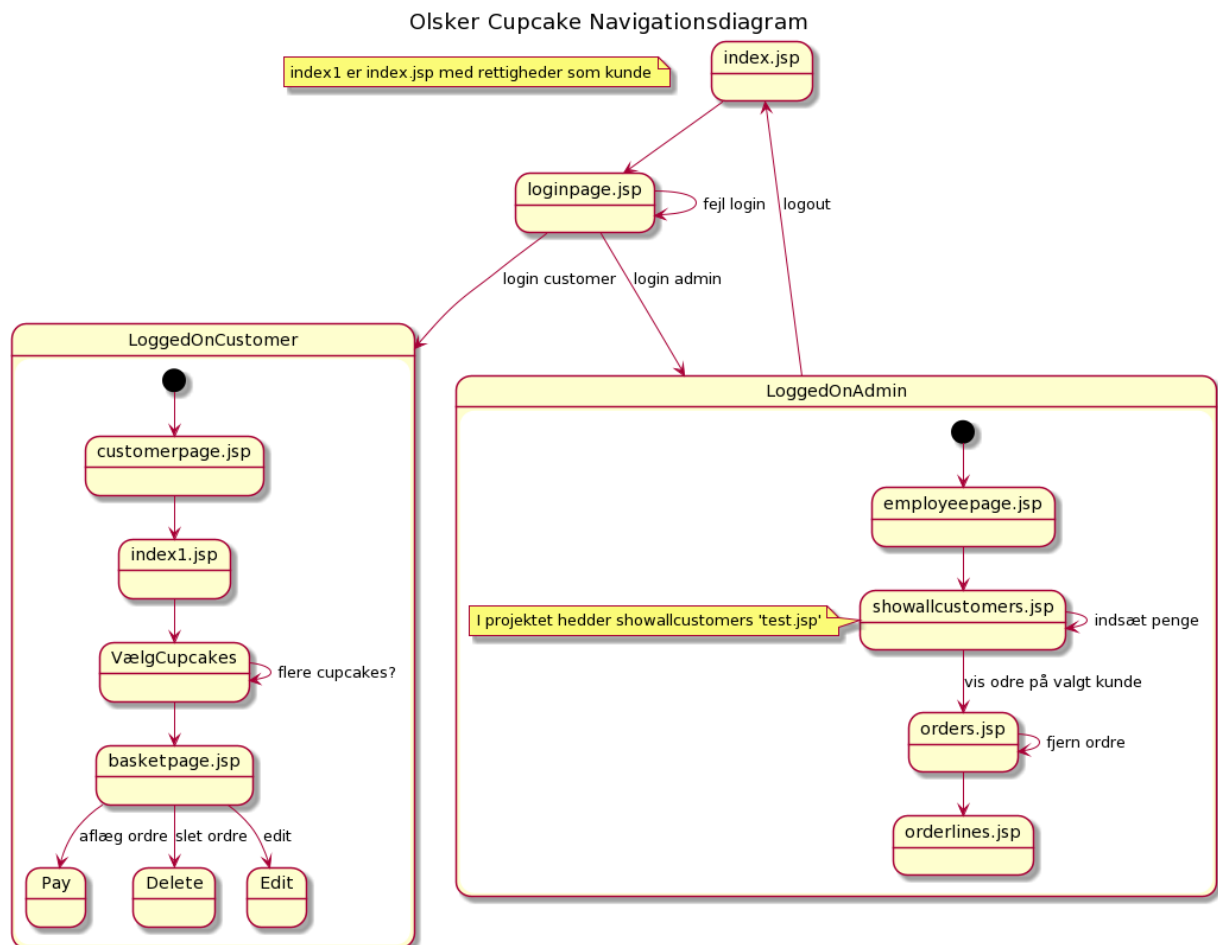
Order: Inderholder, order_id (PRIMARY) Auto Increment, users_id (fk_orders_users_idx) og created og status. Der er en til mange relation, imellem order table og orderline table.

Orderline: Inderholder, orderline_id (PRIMARY) Auto Increment, order_id (fk_orderline_orders1_idx), quantity, price, topping_id (FK_orderline_topping1_idx), bottom_id(FK_orderline_bottom1_idx) .

Topping: indeholder, topping_id(PRIMARY), tname og price

Bottom: Indeholder, bottom_id(PRIMARY), bname og price.

Navigationsdiagram



Her ses et navigationsdiagram over hjemmesiden. OBS: Index1.jsp er index.jsp med kunde rettigheder. Med kunde rettigheder kan der tilføjes ting til kurven og betale.

Alle boxe med ".jsp" stemmer overens med vores jsp sider i IntelliJ. OBS: showallcustomers.jsp hedder test.jsp som også fremgår af diagrammet. Dette er selvfølgelig ikke hensigtsmæssigt og det skyldes en underlig fejl, som vi havde svært ved at opspore. Alle boxe med stort forbogstav, såsom "Pay" er forskellige kommandoer fra "commands" hashmappet.

Typisk flow af programmet:

1. Bruger starter på index.jsp
2. Logger ind via loginpage.jsp
3. Bliver sendt videre til customerpage eller employeepage(forkert login medføre en fejl meddelelse og holder en på loginsiden).

Employee flow:

1. Admin kan tilgå showallcustomers.jsp via employeepage.jsp
2. Tilføje penge til kunde, hvis kunden ønsker det
3. Se ordrer som er tilknyttet til den valgte bruger eller slette en ordre

Kunde flow:

1. Bruger kan tilgå index1.jsp (index.jsp med kunde rettigheder).
2. Kunde kan nu tilføje valgfrie cupcakes til kurven
3. Når kunden ikke ønsker flere cupcakes, kan de fortsætte til "basketpage.jsp"
4. Der har de mulighed for at slette ordrelinier og betale ordren.

Vi har dog gjort det sådan, at nogle af kommandoerne er under samme kommando klasse. Dette har vi valgt at gøre for nemhedens skyld. Hvor vi henter parametrene via request scopet og tjekker om eksempelvis "pay" er forskellig fra null. Det samme gælder "delete". Edit er desværre ikke implementeret.

```
String delete = request.getParameter(s: "delete");
String pay = request.getParameter(s: "pay");
String edit = request.getParameter(s: "edit");
if (pay != null) {
    //Laver en order og orderliner, hvis kunden har nok penge
}
if(delete != null){
    //sletter en orderline
}
if(edit != null){
    //Ikke implementeret
}
```

Udsnit fra ManageOrderlineCommand (koden fyldte for meget til at have den rigtige kode med her). Dvs. alt på basketpage.jsp er omringet af en form der sender ud til ManageOrderlineCommand, som så håndtere dem.

Særlige forhold

Websitet har 2 typer af brugere. En admin(employee) og en kunde(customer). På loginsiden bliver disse roller defineret. Når disse roller bliver sat på session scopet (når man logger ind), kan vi bruge dem til at sikres os, at brugeren ikke kan tilgå sider på websitet, som kun er tilegnet admin-brugeren og omvendt. Dette sker i "AuthorizationFilter", som tjekker hvilken rolle man skal være, for at få adgang til de forskellige sider. Har man ikke den rolle der er påkrævet, får man en 401 error.

Vores kurv (List<Orderline> orderlineList) bliver også gemt i session scopet. Vores første tanke var at gemme alle cupcakes direkte i databasen, så man senere kunne lave noget statistik på dette og måske undersøge om der er et mønster i hvad folk lægger i kurven, men som ikke bliver betalt. Det kunne potentielt være brugbart, men der er også mulighed for massere af ubrugelig data. Men i og med det er 'billigere' at gemme "orderlines"(det fungere som vores kurv) på session scope end i databasen, blev vi enige om, at først være

helt sikre på, at kunden betalte, inden vi sendte dataen afsted til databasen. På den måde er vi sikre på, at dataen er pålidelig i og med den er betalt og ikke bare lagt i kurven.

Sikkerheden på websitet er heller ikke i top. Har man adgang til databasen, har man adgang til alles passwords, eftersom de ikke er blevet hashed. Dette ville vi har gjort, hvis tiden var der. Til gengæld bliver alle SQL-queries fyret af via et "PreparedStatement", som sikre os imod SQL-injections.

Validering af brugerinput sker heller ikke alle steder. Bootstrap hjælper os med validere email, når man opretter en bruger. Derudover bliver der også tjekket for, om de to passwords i registreringsformularen er ens. Når en "employee" indsætter penge på kundernes konto, bliver der heller ikke valideret, hvorvidt man smider tal eller tekst ind.

Status på implementering

Alle userstories er opfyldt. Vi har udover user-storiesene selv udtænkt nogle ekstra funktioner og forbedringer, her er nogle af dem vi ikke nåede eller som ikke virker optimalt.

Kunde kan ikke se sin saldo i kurv-siden.


Hvis der i kurven bliver tilføjet over tre varer og man prøver at slette disse, så vil beløbet for den sidste cupcake stadig stå som at skulle betales.

Når en kunde betaler for en vare, kommer kunden ikke ind på en kvitteringsside. Dette er noget vi godt kunne tænke os at få implementeret, selvom det ikke står i user-storiesene. Hvor vi så også ville have sat et ekstra led på i aktivitetsdiagrammet, hvor ordren bliver hentet fra databasen og bliver vist på skærmen som kunde, som en kvittering.

Når en administrator sletter en ordrer, kommer man ind på ordre-siden, selvom ordren slettes. Vi kunne godt tænke os at man kom tilbage på kundesiden.

Tests

Tests er desværre heller ikke noget vi har rigtigt har haft tiden til. Der er dog blevet lavet test af "OrderMapper"-klassen. Her kunne vi benytte os af design mønsteret dependency



injection og teste på en test database. Alle andre Mapper-klasser får også en database med som konstruktør, hvilket gør de andre Mapper-klasser nemme at teste.

Proces/konklusion

Startkoden har gjort det muligt for os at kunne arbejde som et team. Der er blevet opstillet forskellige design mønstre, som har gjort det nemt og overskueligt at uddelegere opgaverne. Trods dette har vi alligevel kodet det meste af koden samlet, så alle er med på hvordan vi har gjort. Til fremtiden vil vi gerne lave en mere struktureret plan, for hvordan arbejdsgangen skal forløbe, og at vi har mulighed og overblik til at kunne uddele forskellige arbejdsopgaver i gruppen. Vi kommer til at benytte os mere af github funktionen "Issues", da vi tænker dette vil give et godt overblik. Vores arbejdsgang fungerede godt i dette projekt, hvor vi alle i gruppen fik en god forståelse for hvordan opgaven skulle udvikles. Men vi har en fornemmelse af, at man bliver nødt til at uddelegere opgaverne når ens projekt bliver større.