Cupcake hjemmeside projekt.

August Manniche Møller, cph-am382@cphbuiness.dk, manniche1234, C-klassen.

Jens-Olof Pingel Vogel, cph-jv124@cphbusiness.dk, frooshJP, C-klassen.

Younes Karim, cph-yk31@cphbusiness.dk, younesk31, C-Klassen.

Kasper Overgaard Dahl, cph-kd131@cphbusiness.dk, KD131, C-klassen.

Søren Aagaard Bendtsen, cph-sb442@cphbusiness.dk,saabendtsen, C-Klassen.

Rapporten er blevet færdig d. 28/04/2021.

Indholdsfortegnelse

Indledning

Vi fik af opgave, at vi skulle lave en webshop, som sælger cupcakes fra Olsker Cupcakes. Kunden har bedt om en webshop til at hjælpe med at modtage og håndtere ordre og kunder. Projektet er lavet i Java og HTML, hostet på en TomCat server hos digitalOcean.
Data blive håndteret og opbevaret via MySQL.

Baggrund

Virksomheden Olsker Cupcakes har bedt om udvikling af en webshop, der kan håndtere kunder og ordre.
Kunder skal kunne oprette en profil på hjemmesiden, og derefter lægge ordre. Det skal være muligt for kunden at se sine egne tidligere ordre. Når en kunde lægger en ordre, trækkes de i deres balance på siden.

En ordre består af en cupcakes, hvor kunden selv kan bestemme hvilke smage henholdsvis toppen og bunden skal have. Kunden skal præsenteres for en oversigt over hvilke bunde og toppe, der er mulighed for at købe, samt hvilket antal kunden ønsker at købe.

Som administrator skal det være muligt at se, redigere og slette ordre, samt indsætte penge på kunders balance. Administrator skal også kunne slette og redigere en kunde.

Teknologivalg

Der er gjort brug af IntelliJ v.2020.3.3, MySQL v.8.0.22, samt apache tomcat server v.9.0.44

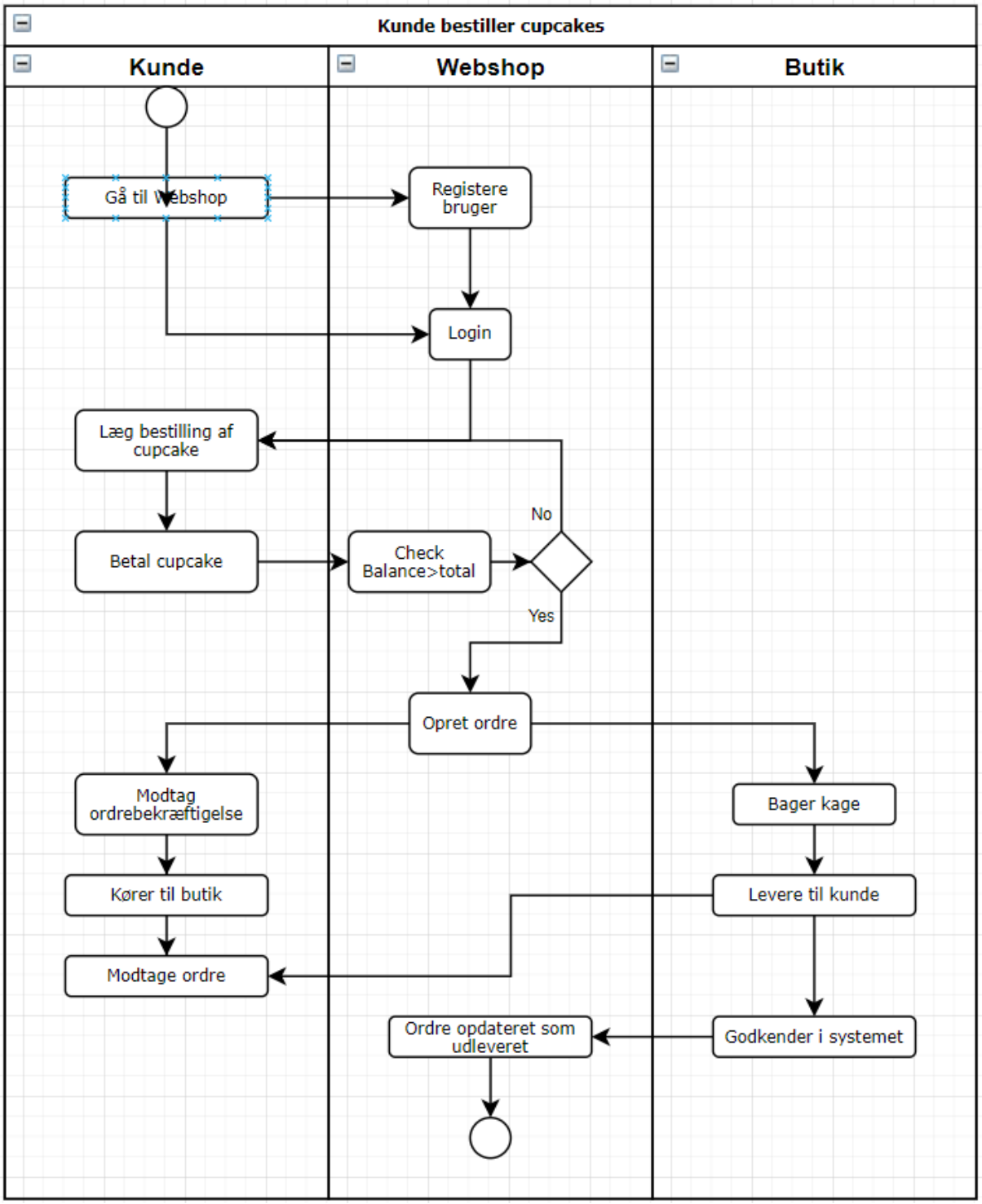
JDBC er brugt til at oprette forbindelse mellem java og vores MySQL database, som bliver etableret under vores frontController.java fil, som så har diverse instanser af datamapper(er), som hver har sin funktionalitet/metoder til at hive relevant data ud af databasen og indsætte dette i Html via servlet attributter i tomcat.

User stories

- ✓ User Story-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.
- ✓ User Story-2 Som kunde kan jeg oprette en konto/profil for at kunne betale og gemme en en ordre.
- ✓ User Story-3: Som administrator kan jeg indsætte beløb på en kundes konto direkte i MySql, så en kunde kan betale for sine ordrer.
- ✓ User Story-4: Som kunde kan jeg se mine valgte ordrelinjer i en indkøbskurv, så jeg kan se den samlede pris.
- ✓ User Story-5: Som kunde eller administrator kan jeg logge på systemet med e-mail og kodeord. Når jeg er logget på, skal jeg kunne min email på hver side
- ✓ User Story-6: Som administrator kan jeg se alle ordrer i systemet, så jeg kan se hvad der er blevet bestilt.
- ✓ User Story-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.
- ✓ User Story-8: Som kunde kan jeg fjerne en ordre fra min indkøbskurv, så jeg kan justere min ordre.
- ✓ User Story-9: Som administrator kan jeg fjerne en ordre, så systemet ikke kommer til at indeholde ugyldige ordrer. F.eks. hvis kunden aldrig har betalt.

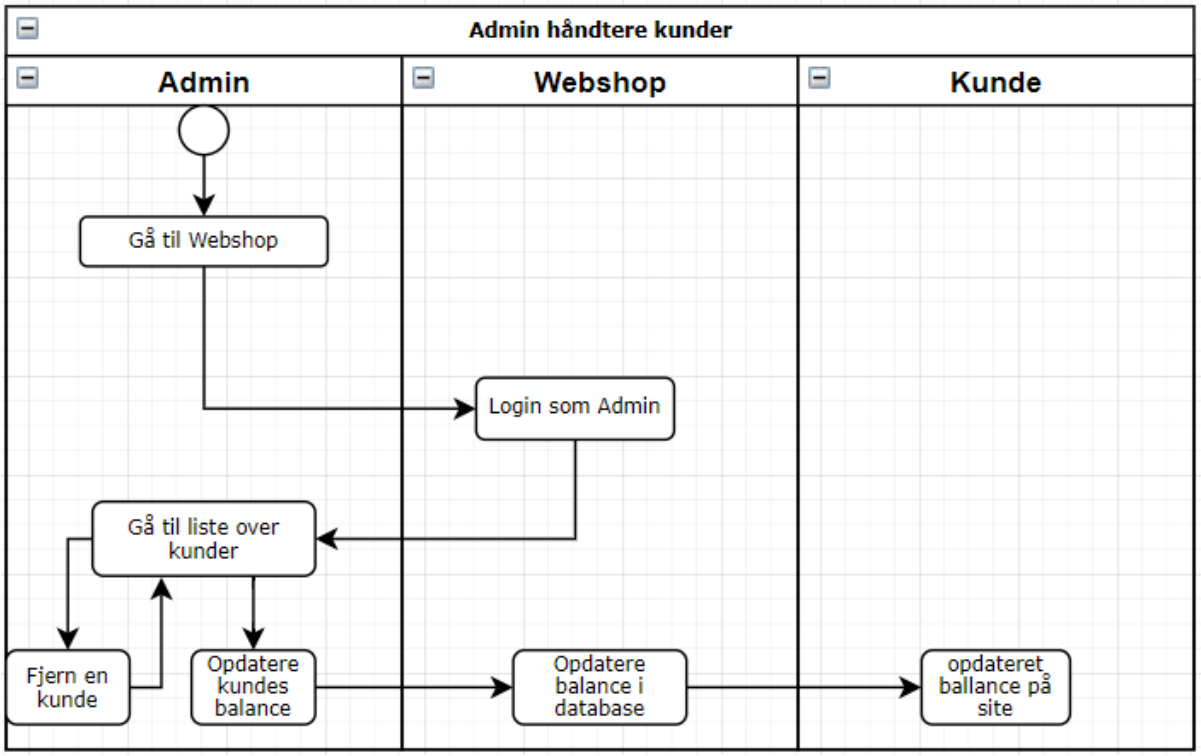
Aktivitetsdiagrammer

Kunde bestiller cupcakes



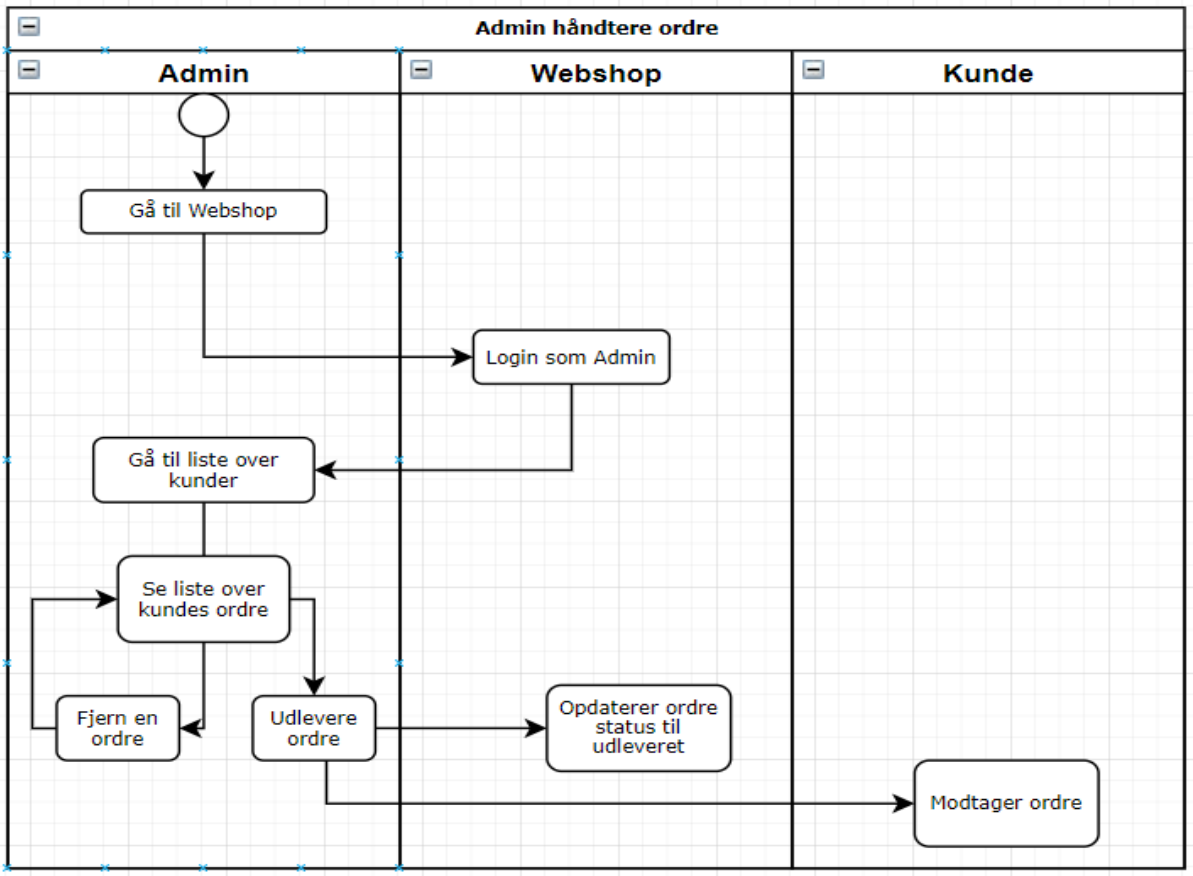
Diagrammet beskriver processen for en kunde, der logger ind på webshoppen, lægger en bestilling og får den udleveret af personalet. Webshoppen sørger for at aflæse og opdatere alle data i databasen.

Admin håndterer kunder



Diagrammet beskriver processen for en ansat, der logger ind på webshoppen for at slette en kunde, eller opdatere deres balance på deres kundekonto.

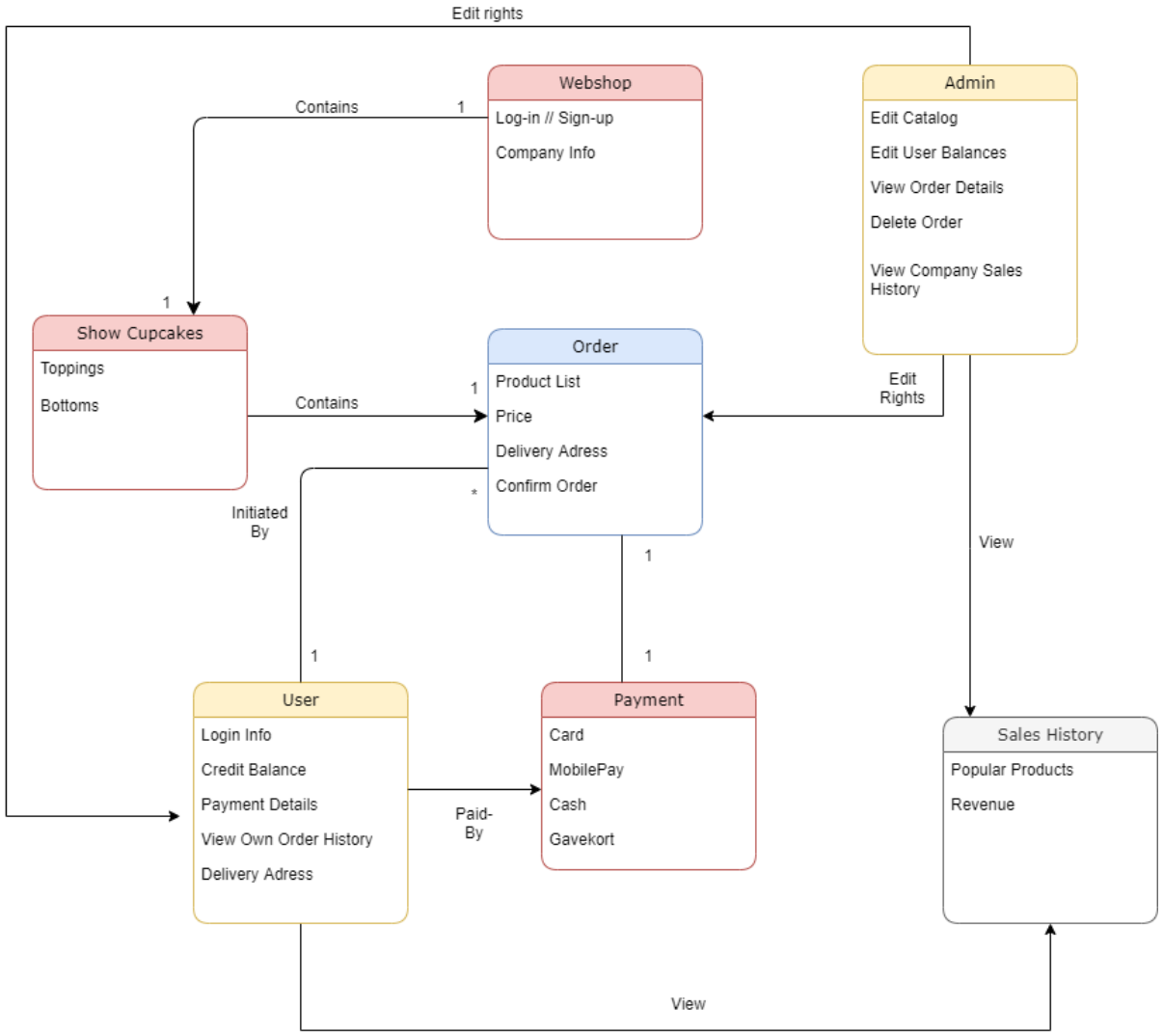
Admin håndterer ordre



Diagrammet beskriver processen for en ansat, der logger på webshoppen for at udlevere eller slette en kundes ordre.

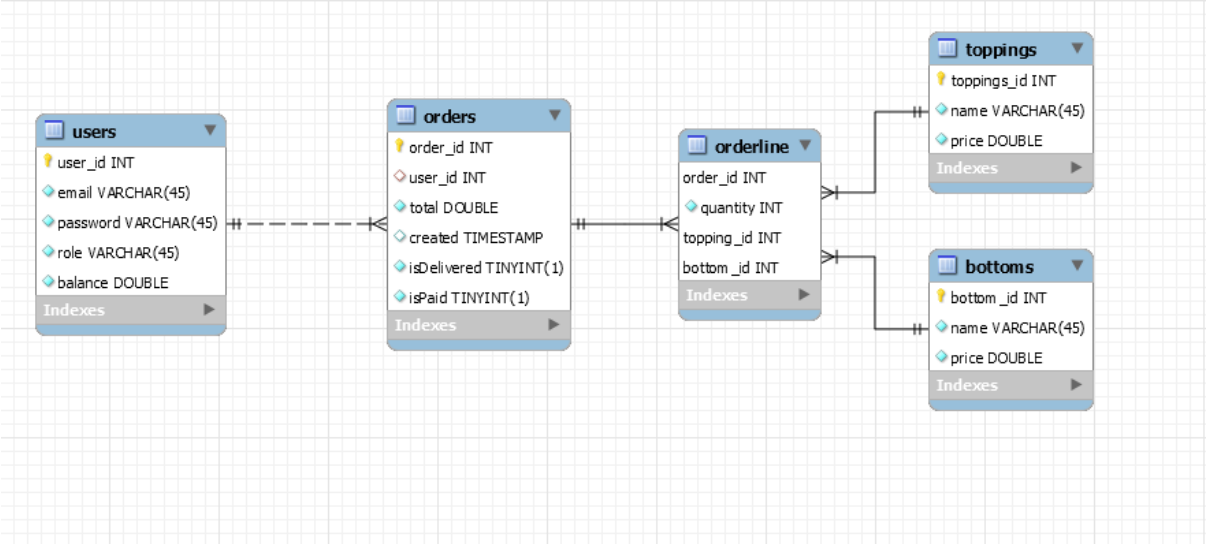
Domæne model og EER-diagram

Domæne Model:



Vi har givet vores admin-klasse en “edit rights”, da de skal kunne sørge for at hjemmesiden samt databasen hele tiden er opdateret. Dvs. hvis der nu skiftes ud i toppings og bunde, så skal han kunne ændre det. Ligeledes med user, så skal de kunne slette og opdatere dem, så der er korrekt information og data på dem. Vi har også tænkt at admin ikke selv kan lave en bestilling, men kan ændre i de bestillinger der nu skal laves, og se alle de ordre der nu har været igennem tiden.

EER-diagram:



Vi har opdelt vores *toppings* og *bottoms* i to forskellige tabeller, som vi så connecter via en tabel ved navn *Orderline.* Den giver vi en collum ved navn *quantity*, så at der godt kan være flere af den sammen slags cupcake i samme ordre.

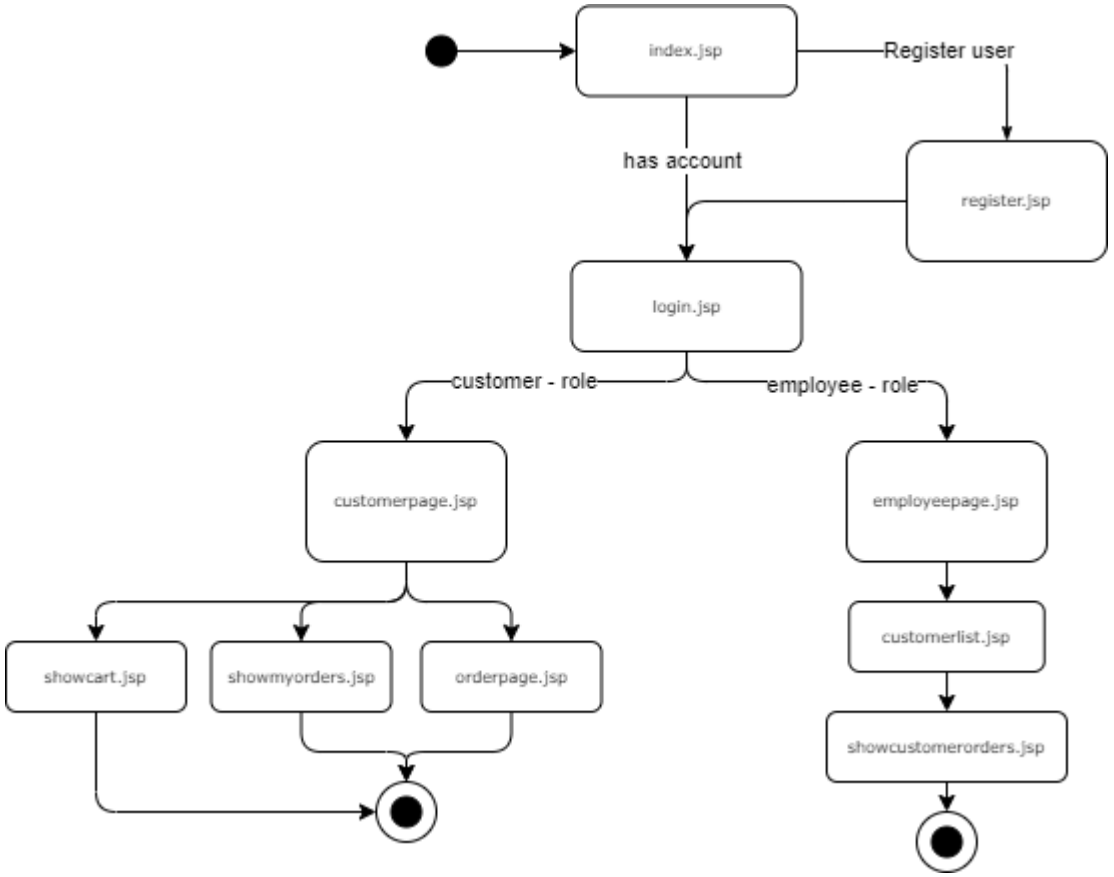
Så en *user* kan bestille en topping og en bottoms → sætter dem sammen til en cupcake i *orderline* → som bliver koblet på en ordre.

Vores *Orderline* tabel er den eneste, der ikke har sin "egen" automatisk genereret ID, men tager order_id, topping_id og bottom_id som primary keys.

Vi har lavet fremmednøgler om til cascade når man slettet noget. Så at det bliver slettet i alle tabeller. Det bruges, så en administrator kan slette ordrer, og de tilhørende *orderlines* bliver slettet automatisk. Man kunne også have slettet fra *orderlines* først, men vi valgte at benytte et cascading delete.

I *orders* bliver user_id sat til null, når man sletter en *user*, så *useren* kan blive slettet, men dens ordre stadig kan ses.

Navigationsdiagram



Alt efter om du logger ind som *customer* eller *employee*, så har man en forskellig navigation bar, der hjælper en til at navigere rundt på hjemmesiden. *Index* siden har vi lavet som en “show cupcakes” for gæster. Det er også her, man logger ind. For at foretage sig noget så skal du være logget ind. Så snart du er logget ind, så kan du gøre forskellige ting, alt efter hvad du er logget ind som.

Liste over vores commands:

index	Sender dig til forsiden.
aboutus	Sender dig til *aboutus*-siden
loginpage	Sender dig ud til forsiden til den given rolle du nu har, dvs. hvis du er customer bliver du sendt til *customerpage*, og employee til *employeepage*.
logincommand	Tjekker login-information, henter brugerens information fra databasen, og sender dig til en side baseret på din *role*.
logoutcommand	Invaliderer sessionen, som logger dig ud.
registerpage	Sender dig til *register*-siden, hvor du kan oprette en ny bruger.
registercommand	Validerer input, opretter en ny bruger i databasen, logger dig ind automatisk, og sender dig til bestillingssiden.
customerpage	Sender dig til siden for brugerfunktionalitet, *customerpage*.
orderpage	Sender dig til bestillingssiden, *orderpage*.
employeepage	Sender dig til siden for adminfunktionalitet, *employeepage*.
updateBasket	Lægger en vare i kurven.
showcart	Udregner totalprisen og sender dig til kurven.
updatequantity	Opdaterer mængden af cupcakes bestilt i kurven, samt sletter cupcakes fra ordren.
customerlist	Funktionalitet som kan benyttes som admin. Laver en liste over alle kunder og deres individuelle balances.
insertorder	Tjekker om kunden har råd til ordren, færdiggør ordren, og putter den i databasen.
clistfunctions	Funktionalitet fra *customerlist*-siden. Rediger saldo, slet bruger, se brugers ordrer.
showmyorders	Henter brugerens egen ordrehistorik fra databasen og sender dig hen til *showmyorders*-siden.
showcustomerorders	Henter ordrehistorik fra databasen baseret på et specifikt id og sender dig til *showcustomerorders*-siden.

deleteOrder	Sletter en ordre i databasen ud fra et specifikt ordre id.
deliverOrder	Sætter feltet *isDelivered* i databasen til *true* og henter en opdateret ordreliste fra databasen.

Særlige forhold

Vi gemmer et *User* objekt i sessionScope, som er den bruger, der logget ind. Ud fra den *user*, kan vi hive information ud som f.eks. saldo, id, email, osv. Bemærk at noget af den information også gemmes separat i sessionScope, selvom der ikke er behov for det.

Vi gemmer også en brugers kurv i sessionScope, kaldet *cupcakeList* i koden, som er en *List* af *Cupcakes*. Et *Cupcake* objekt fungerer som en *orderline* i vores kode. Det indeholder en top og en bund, samt mængden, der er bestilt.

Mange exceptions bliver håndteret således, at der bliver sat en fejlbesked i requestScope kaldet *error*, som bliver vist i rødt på den side, man lavede fejl.

Yderligere er HTML inputs også sat til den type data, de skal indeholde, såsom *number* for tal, hvilket umiddelbart stopper en bruger fra at indtaste forkert data. Man kan dog redigere HTML-siden og komme forbi den sikring, så det bliver også valideret i Java.

Status på implementation

Hvis man lægger en ny cupcake ordre i kurven, der indeholder en sammensætning af top og bund allerede i kurven, vil ordren fejle. *OrderLine* tabellen i databasen har bund, top, og ordre-id som sammensat primary key, og den nye bestilling vil derfor ikke være unik. Det var meningen, at hvis man lagde to af de samme cupcakes i kurven, at mængden skulle opdateres, men det blev ikke implementeret.

Vi havde originalt lavet en tabel til at indeholde yderligere information om brugere, men vi har ikke implementeret noget af den funktionalitet, da det ikke var centralt for opgaven.

Det er ikke alle inputs, der er valideret 100%. På *registerpage.jsp* f.eks. er input-typen til email *type="email"*, men det bliver ikke yderligere valideret i Java ved brug af *regex*, hvilket vil sige, at man kunne redigere HTML-siden og indtaste hvad man vil.

Alle exceptions er *UserException*, hvilket kunne specificeres. F.eks. hvis der ikke kan oprettes forbindelse til databasen, giver det ikke mening at få en *UserException*, men en *DatabaseConnectionException* eller lign.

Der er ingen tests udover de UserMapperTests, der fulgte med startkoden.

Proces

Planer

Siden projektets begyndelse, var vi interesserede i at have en transparent arbejdsprocess og implementerede derfor en kanban-style projektstyringstavle på github.
På denne tavle var opgaver, som inkluderede de udleverede user-stories og andre opgaver, som kunne identificeres allerede inden kode arbejdet startede. I takt med at projektet gik i gang blev flere opgaver tilføjet til tavlen.

Praksis

På den første dag sad vi sammen og implementerede start-koden og fik projektet sat op, så det var til at gå i gang med.
Vi aftalte at mødes hver morgen for at gennemgå opgaver, der var blevet arbejdet på dagen før, så alle havde set koden.
Efterfølgende fortsatte vi i et tæt samarbejde med en daglig arbejdsdag, hvor opgaver blev fordelt. Dagene forløb således, at man tog fat i en opgave fra tavlen, enten alene eller i samarbejde. Samtidigt var der en høj grad af flydende arbejdsstruktur, hvor enkelte personer kunne gå i mellem opgaver og hjælpe til, hvor der var brug for ekstra hjerneceller eller en sparringspartner.

Refleksion omkring forløb

Projektet gik godt, og vi gjorde brug af de projektstyringsværktøjer vi havde sat os for at benytte. Der var nogle specifikke opgaver, som viste sig at være mere komplicerede end først antaget - hvilket resulterede i mere arbejdstid end forventet.
For at undgå lignende problematikker kunne man undersøge opgaverne mere grundigt inden projektets begyndelse og tildele dem nogle estimater på hvor mange arbejdstimer de kræver for at blive løst.
Set i bakspejlet har vi kunne se at user story nr. 1 var langt mere krævende end de andre userstories. I fremtiden, vil vi gerne bruge mere tid på at inddele store userstories i mindre bidder, således at de nemmere kan deles ud på en kanban tavle. Vi stoppede med at have briefing møder, hvilket betød at folk manglede en rød tråd i koden, med de nye implementationer. Alle vidste godt selv hvad de havde lavet, men ikke rigtig nogen der havde det store overblik over det hele.