Cupcake Projekt

- [ ] *Tobias, cph-tr197@cphbusiness.dk, TRossen89*
- [ ] *David, cph-dh223@cphbusiness.dk, cph-dh223*
- [ ] *Lauritz, cph-lh502@cphbusiness.dk, Laumh-exe*
- [ ] *Valdemar, cph-vc61@cphbusiness.dk, Valde1998*
- [ ] *Alexander, cph-ar365@cphbusiness.dk, DoctorHeyde*

Projekt - 23/10/2023, Rapport - 30/10/2023

Indholdsfortegnelse

Indledning

Vi vil i denne rapport præsentere vores webapplikation “Olsker Cupcakes”. Hjemmesiden indeholder funktioner til at kunne bestille og købe cupcakes samt et simpelt login system. Dette projekt er designet til at demonstrere vores evne til at arbejde med en virksomhed og udvikle en prototype der lever op til vores egne og virksomhedens krav og kan arbejdes videre med.
Denne dokumentation giver en dybdegående indsigt i vores projekt og dets tekniske aspekter. Vi benytter os af værktøjer som vi har fået gennem vores undervisning på andet semester.
Rapporten er skrevet til og skal kunne forstås af en studerende på andet semester af datamatikeruddannelsen.

Baggrund

Webshoppen Olsker Cupcakes skal designes som en hjemmeside der kan tilgås på diverse enheder, men først og fremmest skal hjemmesiden være brugervenlig i en pc browser, dernæst på en smartphone. Brugeren skal på hjemmesiden have mulighed for at designe en cupcake med top og bund, og tilføje den til en indkøbskurv. Senere skal brugeren se totalbeløbet og bestille ordren. I et senere afsnit går vi dybden af virksomhedens krav til hjemmesiden.

Teknologi valg

Vi har brugt følgende teknologier til at designe hjemmesiden:

- Figma - til at designe visuelle mockups af hjemmesiden. De repræsenterer en færdig udgave af siden.
- Postgresql Database - til at gemme data om ordrer, kunder osv. Database er normaliseret på 3 normalform.
- Github - til kildekode. Her har vi brugt Github projekter til at lave et Kanban board.
- Udviklet i Java 17, Javalin, Thymeleaf template engine, HTML, JDBC og CSS.
- Kildekode er skrevet i IntelliJ IDEA 2022.3.2 (Ultimate Edition).

Krav

MVP (Minimum Viable Product) Website. Produktet skal kunne opret en konto eller login. Brugeren skal kunne lave en kurv med cupcakes og bestil dem. Brugeren skal kunne se deres saldo og sine ordrer.

User Stories (funktionelle krav)

- [ ] 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 Postgres, 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 email og kodeord. Når jeg er logget på, skal jeg kunne se min 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 ordrelinie 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 udgyldige ordrer. F.eks. hvis kunden aldrig har betalt.

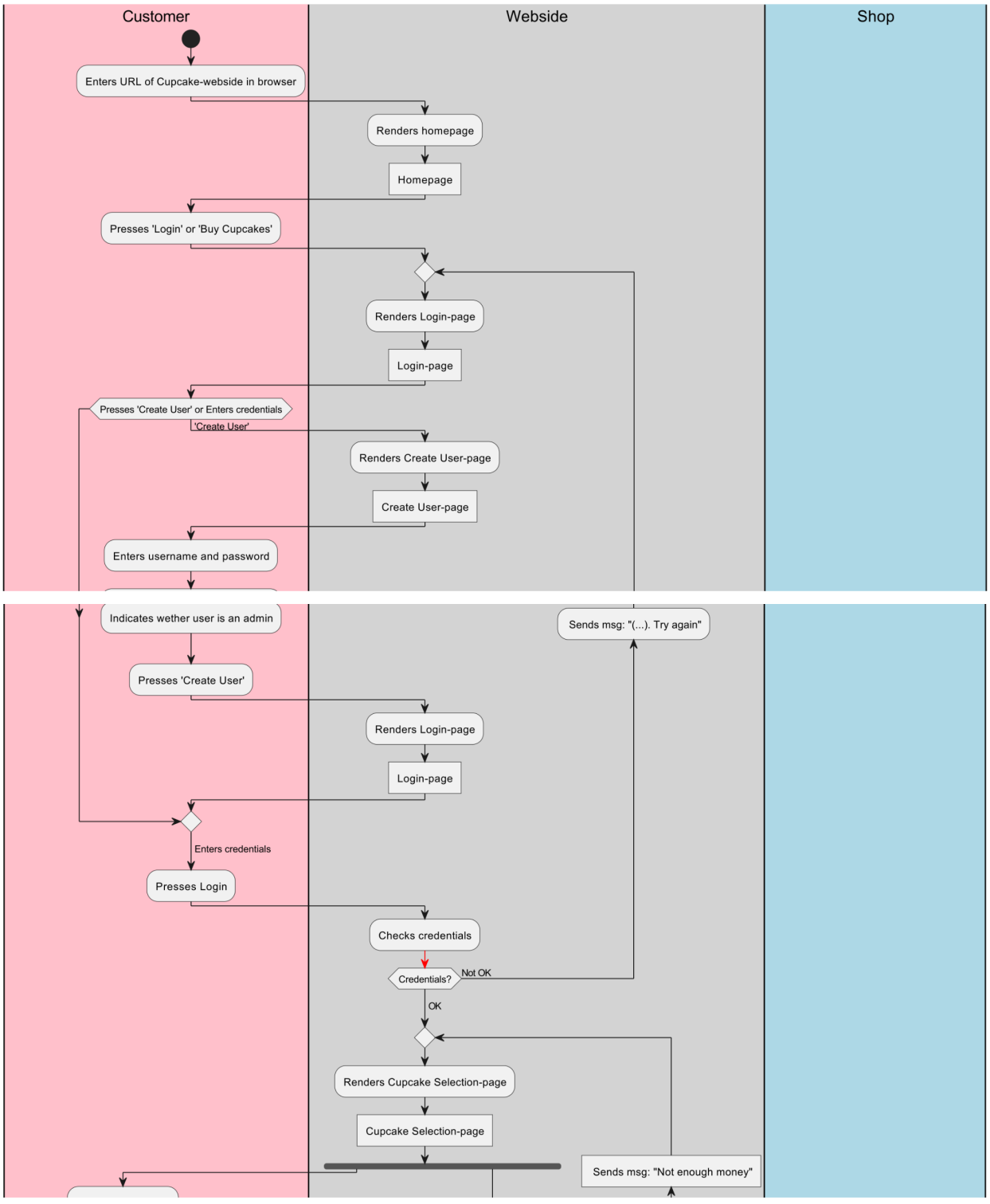
Aktivitetsdiagram

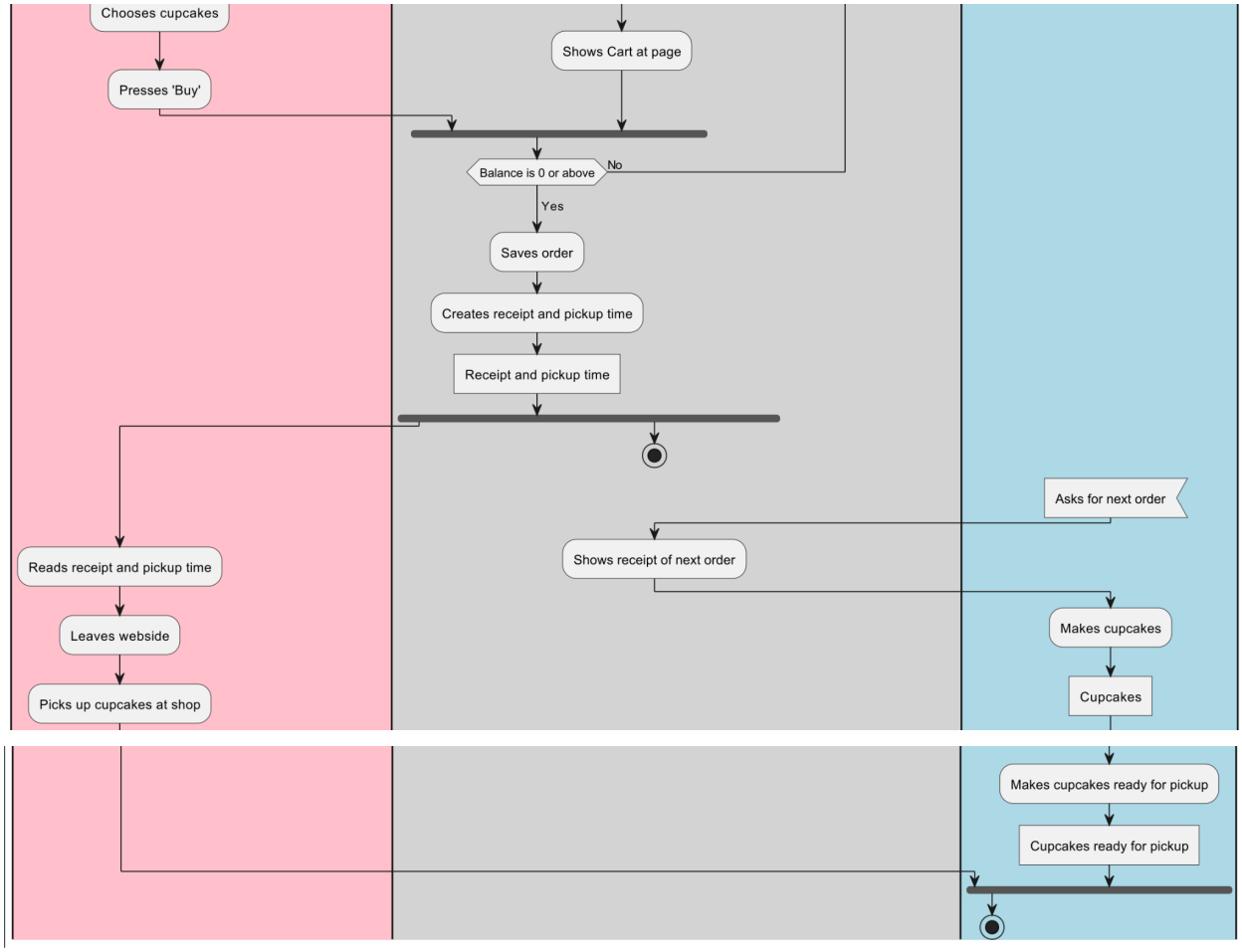
Nedenstående aktivitetsprogram viser et (meget) detaljeret diagram over de aktiviteter vi forestillede os at en bruger, cupcake-websiden og cupcake-butikken ville foretage og de "objekter" de ville producere, hvis en bruger bestilte cupcakes gennem vores cupcake-webside. Aktivitetsdiagrammet er ikke et diagram over en administrators brug af websiden eller over en brugers navigation på hjemmesiden, men altså kun et diagram over de handlinger, der ville blive foretaget og de objekter der ville blive produceret, hvis en bruger gik ind på websiden og bestilte cupcakes.

De tre "svømmebaner" i diagrammet (lyserød, grå og lyseblå) indikerer de tre aktører: 1) bruger (lyserød), 2) webside (grå), 3) butik (lyseblå).

Bokse med runde hjørner indikerer aktiviteter/handlinger. Bokse med skarpe hjørner indikerer "objekter". 6-kantede bokse indikerer to-eller flere mulige handlinger eller systemtilstande. Bokse mellem to fede, sorte streger indikerer at to aktører foretager aktiviteter/handlinger på samme tid. En femkantet boks med et "hak" ind i højre side (se den første boks i Shop-svømmebanen), indikerer en aktivitet, som foretages "af sig selv" med et vis interval.

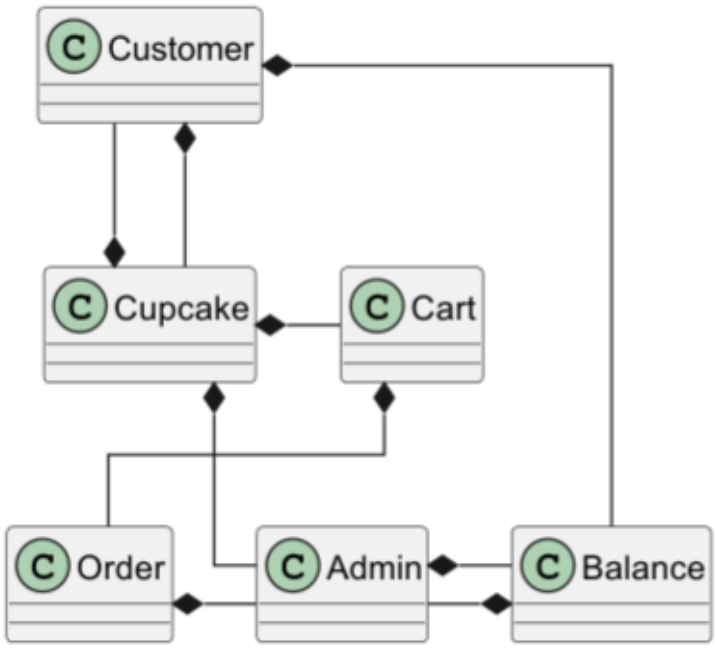
Den røde pil i diagrammet indikerer at den handling pilen peger på, ikke er implementeret i det færdige produkt.


Customer
Webside
Shop
Enters URL of Cupcake-webside in browser
Renders homepage
Homepage
Presses 'Login' or 'Buy Cupcakes'
Renders Login-page
Login-page
Presses 'Create User' or Enters credentials
'Create User'
Renders Create User-page
Create User-page
Enters username and password
Indicates wether user is an admin
Presses 'Create User'
Sends msg: "(...). Try again"
Renders Login-page
Login-page
Enters credentials
Presses Login
Checks credentials
Credentials?
Not OK
OK
Renders Cupcake Selection-page
Cupcake Selection-page
Sends msg: "Not enough money"



Domæne model og ER-diagram

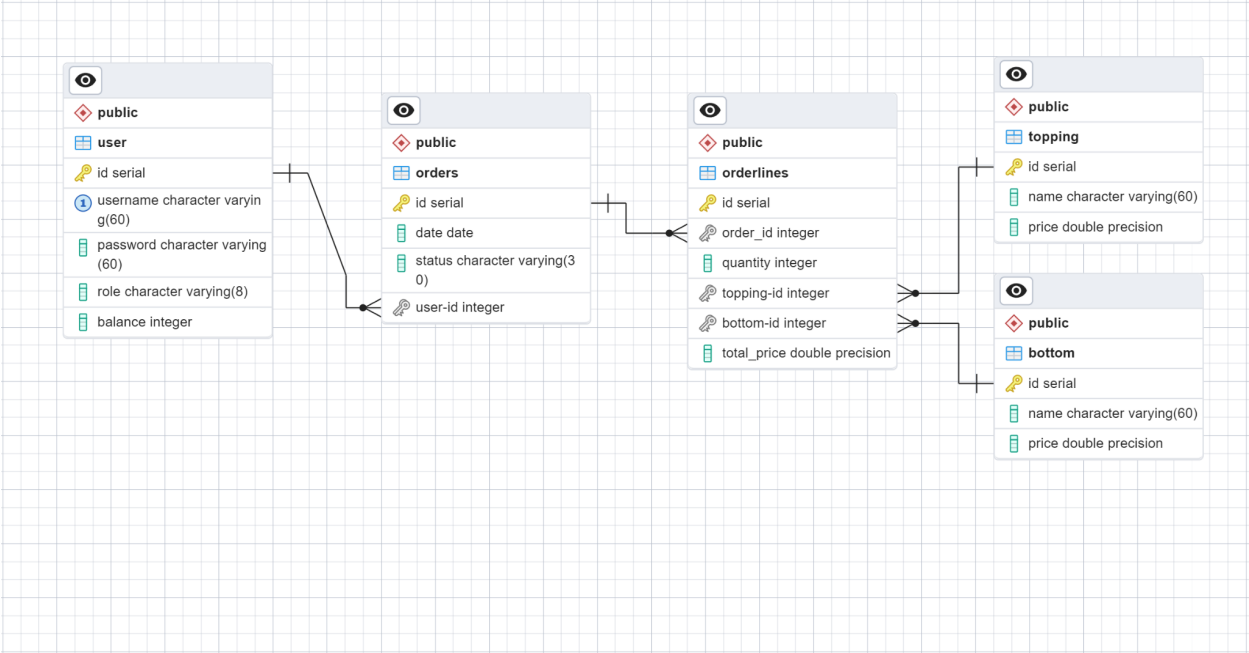
Domænemodel:



- Customer og Cupcake skal kunne tale sammen.
- Cart skal holde en Order af Cupcakes.
- Admin skal kunne se all Orders og Balance af Customers.
- Balance snakker med Admin og Customer.

ERD:

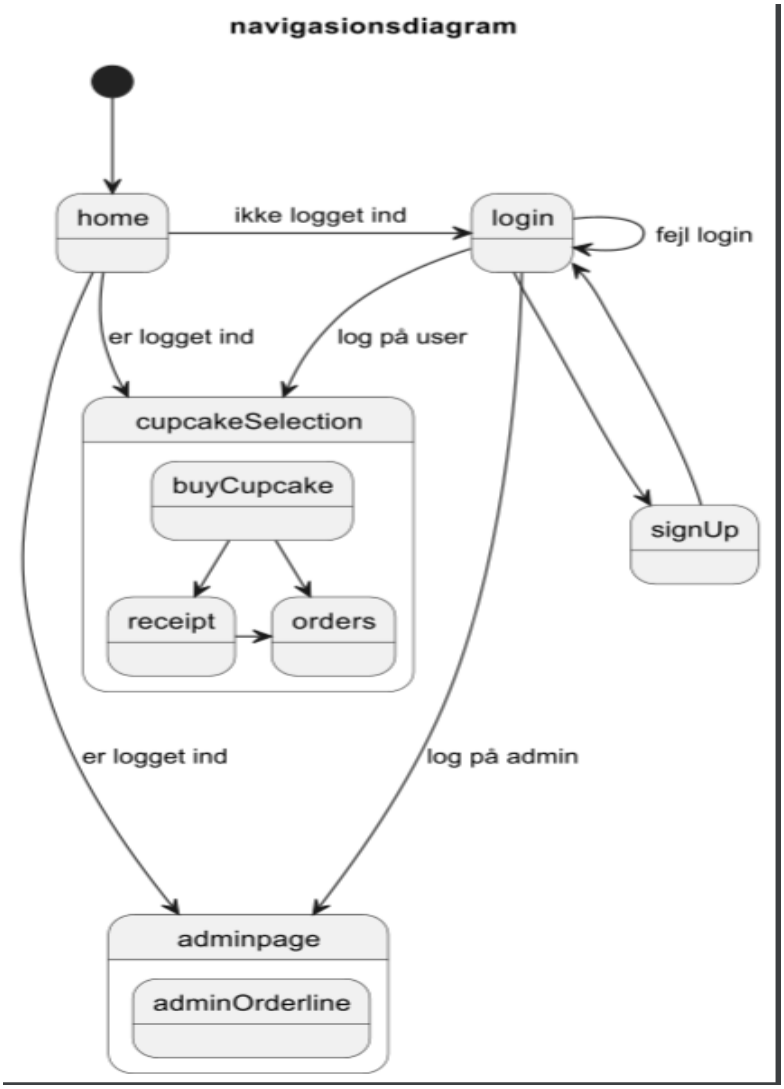
Vores Entity Relation Diagram ser således ud:



Alle tabeller har en PK, der er autogenereret af postgres. Vi har en FK i orders-tabellen, der relaterer ordrerne til brugerne (user-tabellen) og vi har en FK i orderlines-tabellen, der relaterer de forskellige ordrelinjer til ordrerne (orders-tabellen). Der er altså en mange-til-en-relation mellem orders-tabellen og user-tabellen (én specifik bruger kan være relateret til mange forskellige ordrer, mens én specifik ordre kun kan være relateret til én bruger). Derudover har vi to yderligere FK's i orderlines-tabellen, hvor den ene relaterer topping-tabellen til orderlines-tabellen og den anden relaterer bottom-tabellen til orderlines-tabellen. Her er der en mange-til-en-relation mellem topping- og bottom-tabellerne og orderlines-tabellen (én topping/bottom kan være relateret til mange forskellige ordrelinjer, mens én ordrelinje kun kan være relateret til én topping og én bottom).

Vi har gjort username i user-tabellen unik, så det ikke er muligt at indsætte to ens brugernavne i tabellen. Eftersom man i vores program (udelukkende) skriver sit brugernavn og password for at logge ind, er enten brugernavn eller password nødt til at være unikt i vores database. Vi har valgt at brugernavnet skal være unikt, da vi har vurderet det til at være det mest simple og sikre.

Navigationsdiagramemar



Vores cupcake program benytter en fælles navigationsbar, der sidder oppe i headeren på alle siderne, som hjælper en med at navigere rundt i programmet på en hurtig måde. Den har forskellige funktioner alt efter om man er logget ind som bruger eller som admin. Desuden hvis man ikke er logget ind har den også en standardindstilling, med home og login. Er man logget ind som bruger, har man adgang til cupcakeselection, orders, home og logout. Er man derimod logget ind som admin har man adgang til adminorderline, home og logout.

Udover navigationsbaren, er der knapper, input-felter og dropdowns på de forskellige websider til at bevæge sig rundt i programmet og foretage forskellige valg (vælge cupcakes, købe cupcakes, se ordrer etc.).

Vi har lavet nogle figma mockups til nogle af siderne, som man kan se her (nogle af disse er mockups er ufærdige):

https://www.figma.com/file/tSkSUe8K00dn37RwGXHaEe/Untitled?type=design&node-id=0%3A1&mode=design&t=DQrTERRIX9YFzqCi-1

Særlige forhold

Vi vil i dette afsnit beskrive nogle af de kodevalg vi har taget i programmeringen af websiden, så det er nemmere og mere overskueligt at forstå vores kode.

Websidens brugertyper

Man skal som sagt være logget ind for at kunne se andet end forsiden, login-siden og create user-siden, og man kan være logget ind som enten Customer eller Admin. For at adskille mellem en Customer og en Admin i renderingen af de forskellige sider og knapper i navigationsbaren, som disse brugertyper har adgang til, har vi i databasen under tabellen 'user', valgt at have en kolonne, der hedder "role" (Character Varying/String). Her kan man enten være "customer"/"user"* eller "admin". Der er altså ved hjælp af det, der er angivet i "role", at vi viser de korrekte sider og det, der skal vises i navigationsbaren.

*Lige nu, er det sådan, at når vi indsætter en ny Customer i vores tabel i databasen, får de i nogle tilfælde "customer" og i andre tilfælde "user", som deres "role". Dette resulterer i nogle unødvendigt lange if-statements i navigationsbaren (if ((...).equals("customer") || (...).equals("user"))).

Order som en record, uden alle ordre-informationer

Vi har en record*, der hedder 'Order', som ikke indeholder alle ordreinformationer, der er tilgængelige i databasen. Den indeholder kun *(int id, Date date, String status, String userName)*.

*Til den datamatikerstuderende, der ikke ved hvad en record er: En record er en klasse, man bruger til kun at gemme data i. Det er kort sagt en klasse, hvor der ikke kan manipuleres med de data, der indsættes i constructoren, når et objekt oprettes fra klassen. Der er for eksempel ikke nogen Setters. Det er derudover en klasse, der kræver få linjer kode at implementere. Man behøver kun at skrive 1) en accessmodifier, 2) klassenavnet og, i en parentes lige efter klassenavnet, 3) de attributter, som klassen skal have og som sættes, når et objekt oprettes. Hos os ser Order record-klassen således ud:

```
public record Order(int id, Date date, String status, String
userName){}
```

Session attributes

Vi har 8 forskellige session attributes i vores program. Ingen af disse oprettes før en bruger er logget ind, og det er aldrig alle der oprettes ved hver session. Hvilke session attributes, der oprettes, er afhængigt af hvilken slags bruger, der logger ind: Admin eller Customer.

Customer session attributes

- Når en **Customer logger ind**, oprettes følgende session attributes:

    1) 1.1) Navn: ***"currentUser"***. Objekt der gemmes i dette session attribute: ***User*** *(int id, String name, String password, String role (customer), double balance)*.

       Dette session attribute oprettes for at have styr på for det første om den bruger, der logger ind, er en Customer eller en Admin. De to forskellige brugere har adgang til forskellige sider: Kun Customer har adgang til cupcake selection-siden og kun Admin har adgang til admin-siden, hvor alle ordrer i databasen kan ses. Dernæst, hvis det er en Customer, der logger ind, bruges dette session attribute til at holde styr på hvilken bruger, der er på siden, så bestillinger af cupcakes gemmes korrekt i databasen og det kun er denne Customers ordrer, der fremvises når, der trykkes på "Orders"-knappen.

    2) Navn: ***"allBottoms"***. Objekt der gemmes: ***List<Bottoms>*** *allBottoms*.

       Dette session attribute bruges til at vise de forskellige Bottoms-muligheder i dropdown-menuer på cupcake selection-siden

    3) Navn: ***"allToppings"***. Objekt der gemmes: ***List<Toppings>*** *allToppings*

       Dette session attribute bruges til at vise de forskellige Toppings-muligheder i dropdown-menuer på cupcake selection-siden

- Når en **Customer tilgår cupcake selection-siden**, oprettes følgende session attribute:

    4) Navn: **"cart"**. Objekt der gemmes: **Cart**(List<Orderline> orderlines)

Dette session attribute oprettes for at holde styr på og vise brugeren hvad brugeren har tilføjet til kurven. Cart-klassen indeholder metoder, til at

- tilføje ordrelinjer til kurven
- *
- beregne den samlede pris på det der er i kurven
- få fat i alle ordrelinjer i kurven
- tømme kurven for orderlines

Når en bruger trykker "Buy" på cupcake selection-siden og dermed bestiller, det der er i kurven, bliver der oprettet en kopi a Cart-objektet i vores session attribute (denne kopi tilføjes til et ("almindeligt") attribute, som senere vises på receipt-siden). Derefter tømmes det originale Cart-objekt for ordrelinjer og der oprettes et nyt session attribute med dette Cart-objekt (der ikke har nogle ordrelinjer), således at hvis brugeren tilgår cupcake selection-siden igen (efter afgivelse af en bestilling) er kurven tømt.

*Lige nu er der ikke en metode i Cart-klassen, der fjerner ordrelinjer fra kurven, men det ville måske være smartere og mere overskueligt. Som det er nu, fjernes der ordrelinjer fra kurven gennem CartController-klassen, ved at cart-objektet fra vores "cart" session attribute bliver gemt i en variable 'cart', og listen af ordrelinjer tilgås gennem denne variable (cart.getOrderlines()) og gemmes i en List<Orderlines> variable 'orderlines'. Herfra bliver der så slettet en ordrelinje ved hjælp af orderlines.remove(indexOf(orderlineIndex)). Derefter oprettes Cart-objektet igen som et sessionAttribute. Med en metode i Cart-klassen, der fjerner ordrelinjer, ville man bare kunne kalde denne metode i CartController klassen, når en ordrelinje skulle slettes fra kurven.

- Når en **Customer tilgår orders-siden**, oprettes følgende session attribute:

5) Navn: ***"userOrders"***. Objekt der gemmes: ***List<Order****(int id, Date date, String status, String userName)***> orders***

Dette session attributes oprettes for at vise listen af bestillinger, den pågældende bruger har foretaget

6) Navn: ***"orderlines"***. Objekt der gemmes: ***List<Orderlines****(int id, Bottom bottom, Topping topping, int quantity, double totalPrice)***> *orderlines*

   Dette session attributes oprettes for at vise ordrelinjer, der hører til den pågældende ordre, som en bruger har foretaget

Admin session attributes

- Når en **Admin logger ind**, oprettes følgende session attributes:

  - 1.2)
    Navn: ***"currentUser"***. Objekt der gemmes: ***User*** *(int id, String name, String password, String role (admin), double balance)*

    Dette session attribute oprettes for at have styr på for det første om den bruger, der logger ind, er en Customer eller en Admin. De to forskellige brugere har adgang til forskellige sider.

7) Navn: ***"orders"***. Objekt der gemmes: ***Order*** *(int id, Date date, String status, String userName)* [Order-klassen er en record]

   Dette session attribute bruges til at vise en liste af alle ordrer i databasen.

8) Navn: **"orderlines"**. Objekt der gemmes: **List<Orderlines**(int id, Bottom bottom, Topping topping, int quantity, double totalPrice)**>** orderlines

   Dette session attribute bruges til at vise de ordrelinjer, der hører til de forskellige ordrer på listen over alle ordrer i databasen, som en administrator kan se på sin side.

CSS styling

Størrelsen på elementerne

Vi har i den styling, som vi har lavet indtil videre, valgt at bruge enheden 'vw', til at bestemme størrelserne af de forskellige elementer på vores side.

1 vw = 1 % af bredden af den viewport (det "vindue" man kan se på den skærm), der viser html-elementet (https://www.w3schools.com/cssref/css_units.php)

Navigationsbar

På alle andre sider end frontpage.html er naviagationsbaren implementeret med

```
<div th:replace="~{frontpage :: header}"></div>
```

Det er således en kopi af navigationsbaren, som den er defineret på frontpage.html, der implementeres. Hvis man vil ændre i navigationsbaren, skal man ændre i den på frontpage.html.

Status på implementation

Generelle mangler:

- Exceptions-håndtering i store dele af programmet
- CSS-styling af store dele af programmet
- CSS-media queries, der gør hjemmesiden responsiv

I forhold til de forskellige user stories ser status således ud:

- [x] ~~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.~~

  Små mangler:

  1) **Hvis man skriver bogstaver i input-feltet hvor man angiver antal cupcakes man ønsker at bestille, crasher programmet. Der mangler en NumberFormatException-håndtering**
  2) **Receipt-siden, der vises når man har bestilt nogle cupcakes på cupcake selection-siden, mangler en angivelse af hvornår man kan hente sine cupcakes**

- [ ] US-2 Som kunde kan jeg oprette en konto/profil for at kunne betale og gemme en en ordre. **Login mangler; hvis man skriver et username/ password der ikke findes, så crasher hjemmesiden. Derudover er der ikke kodet nogle sikkerhedsforanstaltninger omkring login, udover at der bruges PreparedStatement.**
- [x] ~~US-3: Som administrator kan jeg indsætte beløb på en kundes konto direkte i Postgres, så en kunde kan betale for sine ordrer.~~ **~~DONE~~**
- [x] ~~US-4: Som kunde kan jeg se mine valgte ordrelinier i en indkøbskurv, så jeg kan se den samlede pris.~~ **~~DONE~~**
- [x] ~~US-5: Som kunde eller administrator kan jeg logge på systemet med email og kodeord. Når jeg er logget på, skal jeg kunne se min email på hver side (evt. i topmenuen, som vist på mockup'en).~~ **~~DONE~~**

- [x] ~~US-6: Som administrator kan jeg se alle ordrer i systemet, så jeg kan se hvad der er blevet bestilt.~~ ~~**DONE**~~
- [ ] 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. **Ikke implementeret .**
- [x] ~~US-8: Som kunde kan jeg fjerne en ordrelinie fra min indkøbskurv, så jeg kan justere min ordre.~~ ~~**DONE**~~
- [ ] US-9: Som administrator kan jeg fjerne en ordre, så systemet ikke kommer til at indeholde udgyldige ordrer. F.eks. hvis kunden aldrig har betalt. **Ikke implementert.**

Process

Vi startede med at lave domæne-diagram og et entity-relation-diagram. Herefter lavede vi et git repository for projektet med et tilknyttet Github-projekt. Med de værktøjer kunne vi begynde på at issues som var designet ud fra vores user-stories. Vi fortsatte med at lave issues i løbet af ugen så vi kunne dække mere af vores user-stories.
I Github-projektet organiserede vi vores issues på et Kanban-board. Kanban-boardet blev desværre ikke altid holdt up-to-date med hvilke issues vi var i gang med at udvikle på, og hvilke issues der havde en pull-request. I løbet af ugen lavede vi også et kort daily standup, hvor vi diskuterede om der var nogen der var stødt ind i problemer og om der var nogen der var færdige med de issues de var i gang med dagen inden. Efter den første uge blev der kaldt code freeze mandag eftermiddag, hvorefter vi begyndte at skrive denne rapport.

Vi har ved færdiggørelse af opgaven evalueret og kommet frem til følgende. Vi manglede et klassediagram. Med et klassediagram kunne vi have gjort koden lettere at forstå. Da der var et par gruppemedlemmer, der var tvunget til at arbejde hjemme i noget af første uge, ville det have gjort koden tydeligere for dem. Der var også et par misforståelser i specifikke designvalg af koden, der resulterede i andre løsninger end aftalt. Dette var på ingen måde kritisk for virkningen af applikationen, men var måske et tegn på at mere forberedende arbejde kunne have været en fordel.
Vi er også enig om at Kanban-boardet i Githubet-projektet skulle have været mere opretholdt og issues skulle have været løbende blevet opdateret.

Link til videovisning af webside

https://youtu.be/XOQG7br5sMo