Cupcake

Gruppe: Cupcake Bornholm 3

Deltagere

Navn	CphB email	GitHub bruger
Emil Vang	cph-ev58@cphbusiness.dk	vaangis93
Oliver Gudbergsen	cph-og92@cphbusiness.dk	ShushNhush
Peter Torgersen	cph-pt121@cphbusiness.dk	Hunkstaban
Tobias Ipsen	cph-hi88@cphbusiness.dk	TobiasIpsen

Tidspunkt

Projekt og rapport udført April 2024

Link til videodemo af projektet

https://cphbusiness.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=96604095-bba4-4744-87ea-b14e01222d38

Indholdsfortegnelse

Indledning

Målgruppen for denne rapport er studerende på 2. Semester af datamatikeruddannelsen. Formålet med projektet er at udvikle et system til en online butik som sælger cupcakes. Systemets backend er udviklet i Java og forbundet til en postgresql database for opbevaring af data. Frontend er skrevet i html og stylet med css. Der bliver brugt frameworket Javalin til at håndtere indgående HTTP requests, samt håndtere flere threads så systemet kan have flere brugere aktive på samme tid. Der er benyttet frameworket Thymeleaf for dynamisk at vise data på frontend. For at data kan fremvises, benyttes Javalin context attributter der videresendes fra backend - disse attributter defineres vha. relevante data fra databasen, som hentes igennem DataMapper-klasser.

Baggrund

Systemet er udviklet til en lokal butik som sælger cupcakes i Olsker på Bornholm.
De vil udvide deres butik med en online shop, hvor kunder kan bestille cupcakes, som de kan hente i butikken på et senere tidspunkt.

Kunden har givet nogle forskellige krav til hvordan deres webshop skal fungere. Kunden skal kunne oprette en konto på webshoppen, og kunne vælge forskellige kombinationer af bunde og toppe til deres cupcakes. Kunden skal kunne se og justere deres indkøbskurv inden de færdiggøre deres bestilling.
Som administrator skal de kunne se alle bestillinger der er blevet placeret i systemet.

Teknologi valg

IntelliJ IDEA (IDE): v2023.2.3

Java: Amazon Coretto v17.0.8

Figma (UI Mockup): v116.16.13

JDBC

Postgresql (Database): v42.7.2

Thymeleaf (Frontend Framework): v.3.1.2

Javalin (Backend Framework): v6.1.3

Hikari: v5.1.0

Maven: v17

Docker (Server software): v4.27.2

PGAdmin (Database manager): v8.3

Krav

Det udviklede system skal give en let tilgængelig webshop for Olsker Cupcakes' kunder. Olsker Cupcakes skal kunne se en liste over alle bestillinger, så de kan se hvad der skal laves/hentes.

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 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.

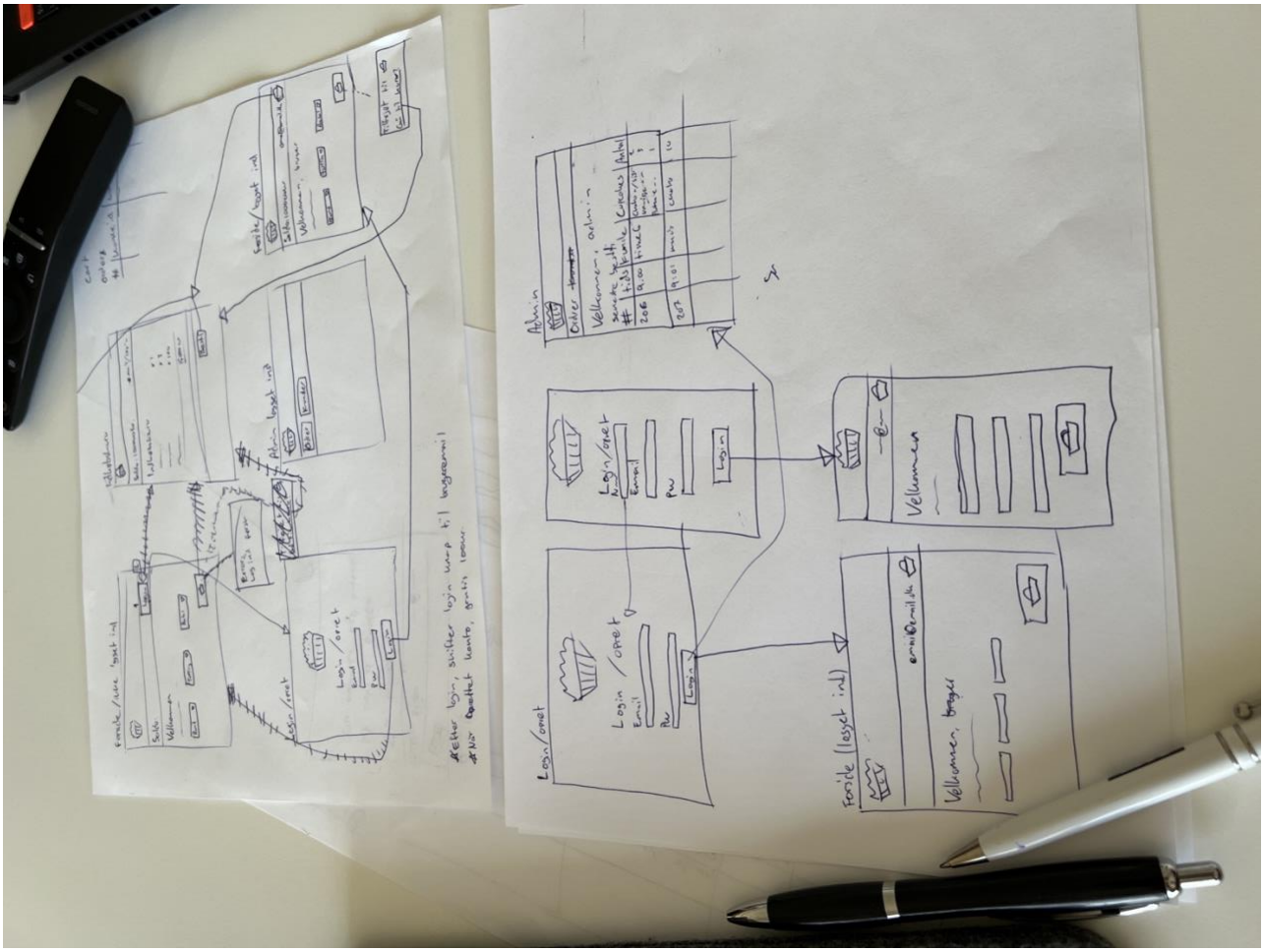
US-8: Som kunde kan jeg fjerne en ordrelinie fra min indkøbskurv, så jeg kan justere min ordre.

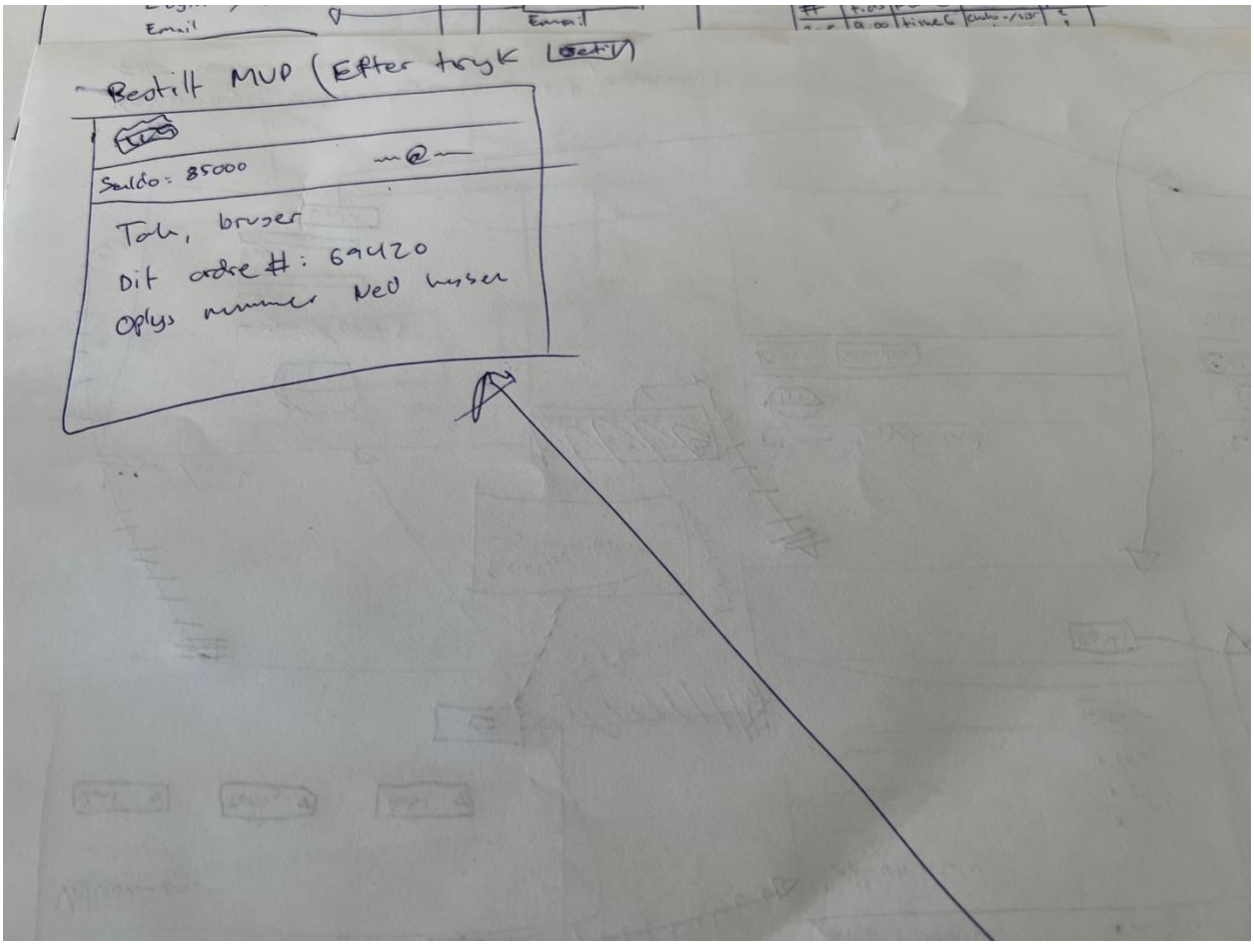
US-9: Som administrator kan jeg fjerne en ordre, så systemet ikke kommer til at indeholde udgyldige ordrer. F.eks. hvis kunden aldrig har betalt.

UI

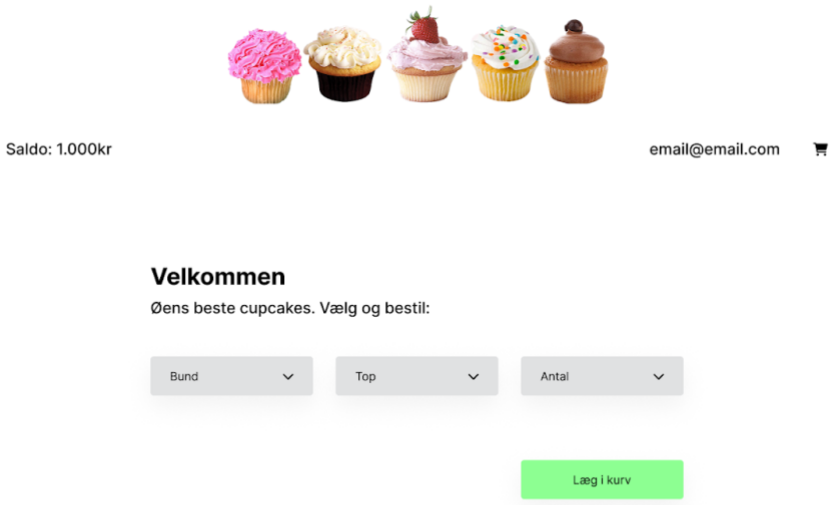
De initielle tanker omkring opsætningen af projekts UI startede med skitser på papir, som blev efterfulgt af en overførsel til en digital mockup vha. Figma.

Skitser




Email
Email
Bestil MUP (Efter tryk
Saldo: 85000
Tak, bruger
Dit ordre #: 69420
Opløs

Figma mockup

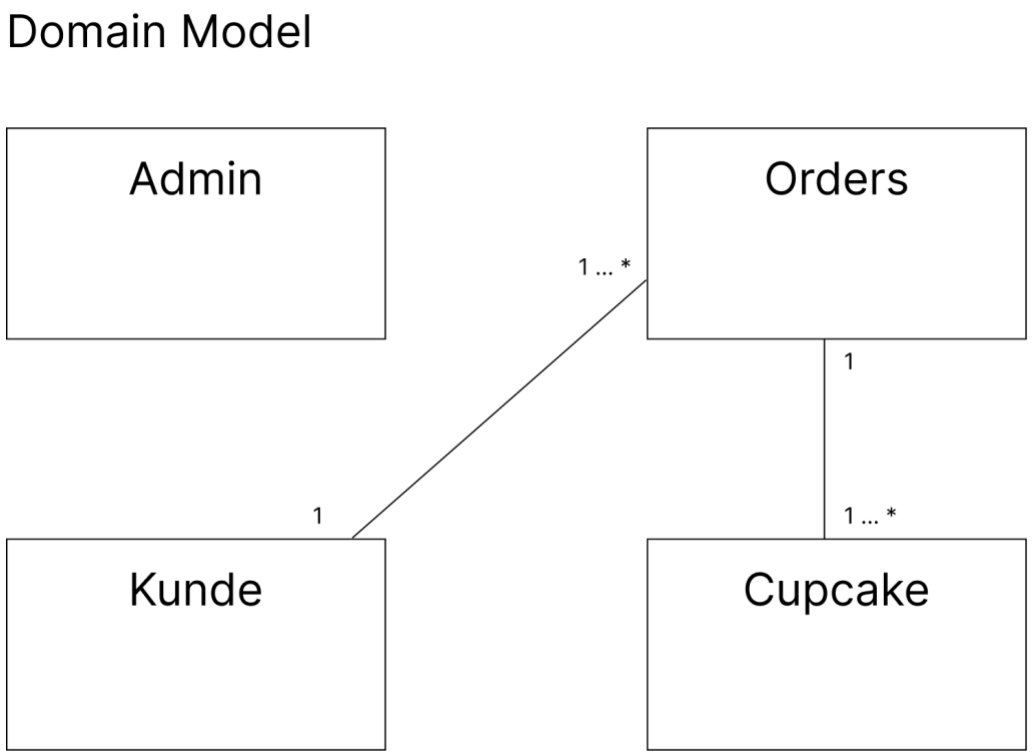


Her er vist en enkelt frame fra vores Figma mockups af forsiden, efter brugeren har logget ind. De andre frames kan ses i Bilag 1.

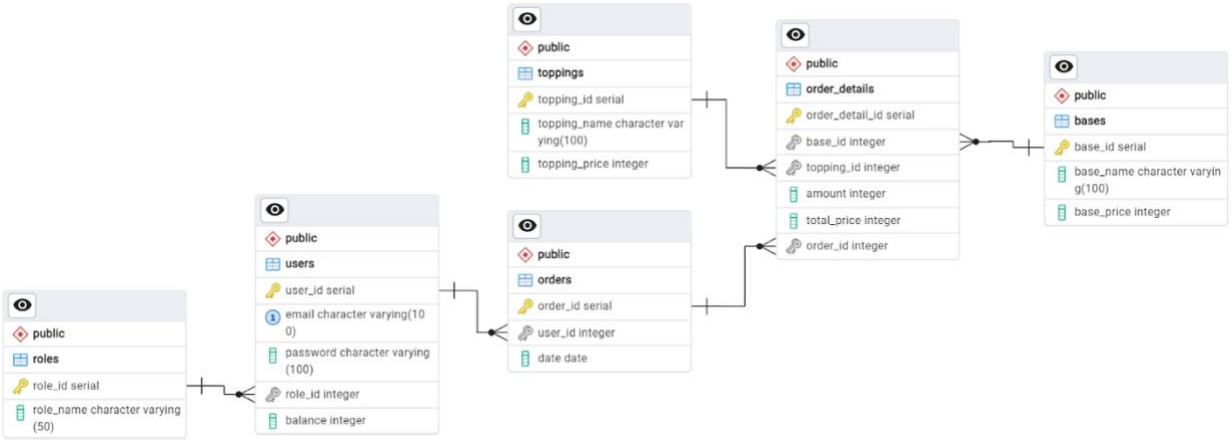
Domæne model og ER diagram

Domænemodel

Én kunde kan foretage én eller mange ordre. Én ordre kan indeholde én eller mange cupcakes. Admin kan ikke foretage sig noget i modellen, og bliver kun vist alle ordrer.



ER Diagram



Order_details og orders:
Vi har valgt at lave en order_details tabel som har alle informationerner om en ordre. Derpå har vi en order_id FK som linker til order_id på orders tabellen. På orders har vi også en user_id, som viser hvilken user der har bestilt.

Tabellen order_details går imod 3NF. Det gør den fordi total_price ikke er direkte afhængig af primærnøglen, men afhænger af de to fremmednøgler base_id og topping_id. Skulle databasen være på 3NF, ville en ny tabel, f.eks. kaldet ‘cupcakes’, oprettes, hvor order_details kun vil få et cupcake id, som indeholder bund, top og pris – dette er dog fravalgt, da det vil skabe data reduncancy, da alle kombinationer af cupcakes skal oprettes.

Navigationsdiagram

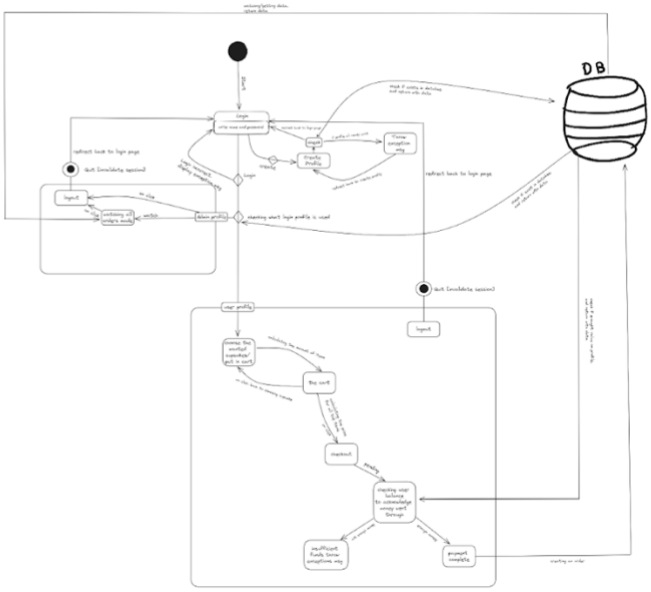
Der er blevet lavet 2 modeller af et navigationsdiagram (kan ses større på bilag 2).

Model 1: Denne model repræsenterer en mere simplificeret navigationsdiagram, der fremhæver de mest essentielle elementer. Dette diagram er designet med fokus på systemforståelse, overblik, brugervenlighed og intuitivitet, hvilket gør det let for brugerne at navigere gennem systemet. Formålet med Model 1 er at give brugerne et klart og overskueligt overblik over navigationsstrukturen uden unødvendig kompleksitet.

Model 2: Denne model repræsenterer et mere komplekst navigationsdiagram, der er mere ”suited” til programmeringsdelen end forståelsesdelen og overblikket. Dette diagram kan indeholde mere tekniske detaljer, database henvisninger, og andre tekniske komponenter, der er relevante for udviklere og programmører. Formålet med Model 2 er at give en dybere indsigt i systemets tekniske arkitektur.

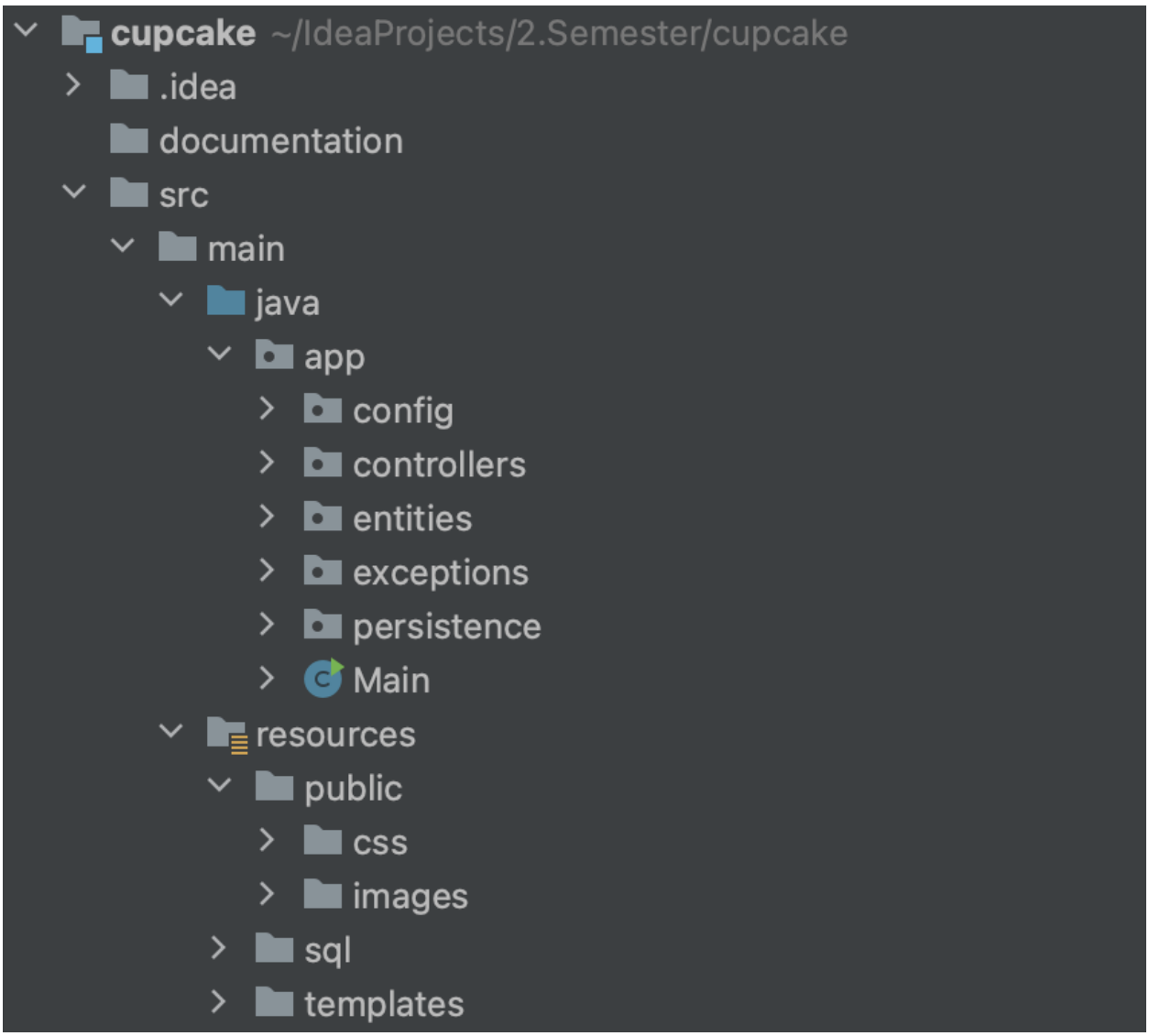
model 1

model 2



Særlige forhold

Projektstruktur



Projektet er struktureret efter Model-View-Control princippet - det er gjort på denne måde for at simplificere kode- og klassestruktur, samt separere frontend og backend.

Loginhåndtering

For at sørge for at forskellige brugeres login kan håndteres jf. kundens User Stories, er der oprettet en roles database tabel der kan bruges som fremmednøgle på users

tabellen - det betyder, at der kan laves en kontrol på, hvilken type bruger der logger ind på systemet, vha. et if-statement i Java-koden:

```
59 @    private static void login(Context ctx, ConnectionPool connectionPool) {
60
61          String email = ctx.formParam( key: "email");
62          String password = ctx.formParam( key: "password");
63
64          try {
65              User user = UserMapper.login(email, password, connectionPool);
66              if (user.getRoleID() == 2) {
67                  adminLogin(ctx, connectionPool, user);
68              } else {
69                  userLogin(ctx, connectionPool, user);
70              }
71
72          } catch (DatabaseException e) {
73              String msg = "Forket email eller kodeord. Prøv igen";
74              ctx.attribute("loginError", msg);
75              ctx.render( filePath: "login.html");
76          }
77
78      }
```

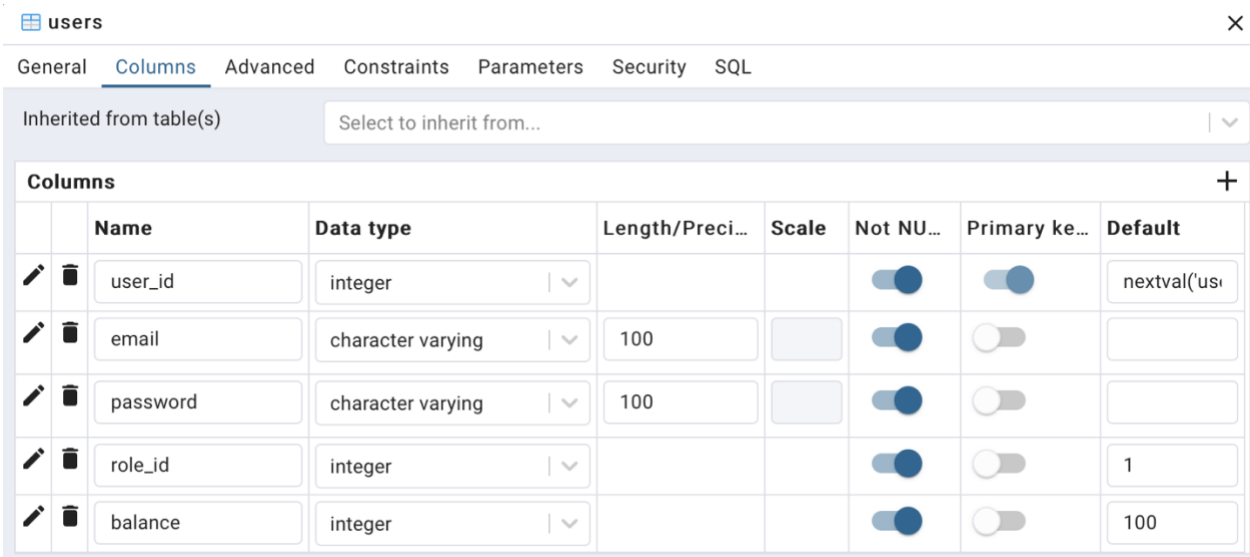
login() i controllers.UserController

Der ligger to roller på roles tabellen, hvor role_id 1 er en almindelig kunde-bruger og role_id 2 er en admin:

	role_id [PK] integer	role_name character varying (50)
1	1	customer
2	2	admin

Oprettes en bruger igennem systemet, bliver role_id default sat til at være 1 (customer) - admins kan derfor kun defineres gennem databasen.

Derudover er der oprettet en balance kolonne på users, som default får indsat 100 kr., når en ny bruger bliver oprettet. Ønsker brugeren flere penge, skal det indsættes igennem databasen:



Sessionsdata

Ved log ind bliver der kun oprettet én attribut til sessionen kaldet “currentUser”:

```
80 @    private static void userLogin(Context ctx, ConnectionPool connectionPool, User user) {
81          ctx.sessionAttribute("currentUser", user);
82          ctx = CupcakeController.baseToppingAttributes(ctx, connectionPool);
83          ctx.render( filePath: "index.html");
84      }
85
            1 usage   petert +1
86 @      private static void adminLogin(Context ctx, ConnectionPool connectionPool, User user) {
87          ctx.sessionAttribute("currentUser", user);
88          List<Order> orderList =  OrderMapper.viewAllOrders(connectionPool);
89          ctx.attribute("orderList", orderList);
90          ctx.render( filePath: "orders.html");
91      }
```

controller.UserController

Denne sessionsattribut indeholder alt info, som er defineret på User entitet klassen - i denne klasse findes også en klassevariabel kaldet “cartList”:

```
6   public class User {
7
        3 usages
8       private int userID;
        4 usages
9       private String email;
        4 usages
10      private String password;
        3 usages
11      private int roleID;
        3 usages
12      private int balance;
        4 usages
13      private List<OrderDetail> cartList = new ArrayList<>();
```

entities.User

Ved at bruge en klassevariabel til indkøbskurven betyder det, at de cupcakes der tilføjes til kurven vil eksistere på den bruger, der er logget ind, så længe sessionen kører.

Det er samtidig håndteret, at en besked vises til klienten, hvis deres indkøbskurv er tom. Ved navigation til indkøbskurven (cart.html) sendes en attribut ‘cartList’, der er hentet fra brugerens sessionsindkøbskurv, med - er denne tom fjernes ordretabellen, samt muligheden for at trykke på bestil og oprette en ordre. Dette er opnået ved at bruge Thymeleafs if- og unless-conditionals, der kontrollerer hvorvidt listen har data, og afhængig af det viser/skjuler html elementerne:

```java
@    private static void goToCart(Context ctx, ConnectionPool connectionPool) {

        User user = ctx.sessionAttribute( key: "currentUser");
        List<OrderDetail> cartList = user.getCartList();

        //Figure out the total price of the order so it can be sent as an attribute
        int orderTotalPrice = 0;
        for (OrderDetail orderDetail : cartList) {
            orderTotalPrice += orderDetail.getTotalPrice();
        }

        ctx.attribute("cartList", cartList);
        ctx.attribute("orderTotalPrice", orderTotalPrice);
        ctx.render( filePath: "cart.html");

    }
```

goToCart() i controllers.OrderController

```html
        <div th:if="${#lists.isEmpty(cartList)}">
            <p class="p">Din indkøbskurv er tom! Gå tilbage og tilføj nogle lækre cupcakes :-)</p>
        </div>
        <div id="cart-table-container" th:unless="${#lists.isEmpty(cartList)}">

            <table class="cart-table">
                <thead>
                <tr>
                    <th>Bund</th>
                    <th>Top</th>
                    <th>Antal</th>
                    <th>Pris</th>
                    <th></th>
                </tr>
                </thead>

                <tbody>
                <tr th:each="cart, index : ${cartList}">
                    <td th:text="${cart.baseName}"></td>
                    <td th:text="${cart.toppingName}"></td>
                    <td th:text="${cart.amount}"></td>
                    <td th:text="${cart.totalPrice} + ' kr.'"></td>
                    <td>
                        <form th:action="@{/removeFromCart}" method="post">
                            <input type="hidden" name="cartIndex" th:value="${index.index}">
                            <button class="remove-item-button" type="submit">Fjern</button>
                        </form>
                    </td>
                </tr>
                </tbody>
            </table>

        </div>
```

```
94          <button class="defbutton green" th:unless="${#lists.isEmpty(cartList)}" type="submit"
95                  formaction="newOrder">Bestil
96          </button>
```

templates/cart.html

Exceptions og validering af brugerinput

Der er oprettet en klasse DatabaseException i exceptionspakken, der giver udviklerne mulighed for at brugerdefinere fejlbeskeder.
Derudover er der defineret context attributter med tilhørende beskeder til de mere almindelige fejl der kan forekomme for klienten, så de kan vises på browser-vinduet - ved at bruge thymeleaf if-conditional sikres der, at disse fejlbeskeder kun bliver vist, hvis attributten har indhold.

Der er oprettet fejlbeskedsattributter til:

- Login - hvis brugeren skriver forkert email/kodeord. Attribut: loginError:

```
72          } catch (DatabaseException e) {
73              String msg = "Forket email eller kodeord. Prøv igen";
74              ctx.attribute("loginError", msg);
75              ctx.render( filePath: "login.html");
76          }
```

login() i controllers.UserController

```
30              <div class="user-not-found p" th:if="${loginError}">
31                  <p th:text="${loginError}"></p>
32              </div>
```

login.html

- Opret - hvis der forsøges at oprette en bruger med en email, der allerede eksisterer på databasen. Attribut: alreadyExist:

```
44          } catch (DatabaseException e) {
45              String msg = "Bruger med denne email eksisterer allerede.";
46              ctx.attribute("alreadyExist", msg);
47              ctx.render( filePath: "create.html");
48          }
```

create() i controllers.UserController

```
26              <div class="user-not-found" th:if="${alreadyExist}">
27                  <p th:text="${alreadyExist}"></p>
28              </div>
```

create.html

- Utilstrækkelig saldo - hvis en bruger forsøger at bestille en ordre, hvis totale pris overstiger brugerens saldo. Attribut: balanceError:

```
97          } catch (DatabaseException e) {
98              String msg = "Din saldo dækker desværre ikke denne bestilling."
99                      + "Fjern noget fra din ordre, eller kontakt en admin.";
100             ctx.attribute("balanceError", msg);
101             ctx.attribute("cartList", user.getCartList());
102             ctx.attribute("orderTotalPrice", orderTotalPrice);
103             ctx.render( filePath: "cart.html");
104         }
```

newOrder() i controllers.OrderController

```
103             <div class="insufficient-funds p" th:if="${balanceError}">
104                 <p th:text="${balanceError}"></p>
105             </div>
```

cart.html

Database 'view_all_orders' view

Vi oprettede et view i pgAdmin til at vise alle ordre, samt hver individuelle ordres tilhørende detaljer, så vi undgår at have vores lange SQL sætning i Java. Det gør det også nemmere at opdatere viewet/SQL sætningen, hvis den bliver brugt på tværs af metoder. Dette forenkler forespørgsler og gør koden mere læsbar. En anden fordel ved views er at de er read-only, hvilket kan have en security fordel.

```java
public static List<Order> viewAllOrders(ConnectionPool connectionPool) {
    String sql = "SELECT * FROM public.view_all_orders";
```

view_all_orders

General Definition Code Security SQL

```sql
SELECT orders.order_id,
   users.email,
   bases.base_name,
   toppings.topping_name,
   order_details.amount,
   order_details.total_price,
   orders.date
  FROM order_details
    JOIN orders ON order_details.order_id = orders.order_id
    JOIN users ON orders.user_id = users.user_id
    JOIN bases ON order_details.base_id = bases.base_id
    JOIN toppings ON order_details.topping_id = toppings.topping_id
 GROUP BY orders.order_id, users.email, bases.base_name, toppings.topping_name,
 order_details.amount, order_details.total_price, orders.date
 ORDER BY orders.order_id DESC;
```

Status på implementation

Fejl og mangler

- Ifølge projektets figma mockups er der vist foreslag til visning af hjemmesiden på mobil skærmstørrelser - implementeringen af dette mangler. Vi ved det skal laves ved brug af css media queries samt flexbox.
- Styling af de forskellige sider kræver mere arbejde, samt generelle kodestandarder er ikke overholdt (f.eks. navgivning af .css filerne, hvor der er brugt camelCase istedet for kebab-case).
- User Story 7 og 9 (der dog ikke er et leveringskrav) er ikke blevet implementeret.
- Fejl ved håndtering af oprettelse af ny bruger. Ikke defineret at det *skal* være en email som “brugernavn”. Brugeren kan på nuværende tidspunkt skrive og oprette en bruger med hvad som helst– og endda lade email feltet stå tomt.

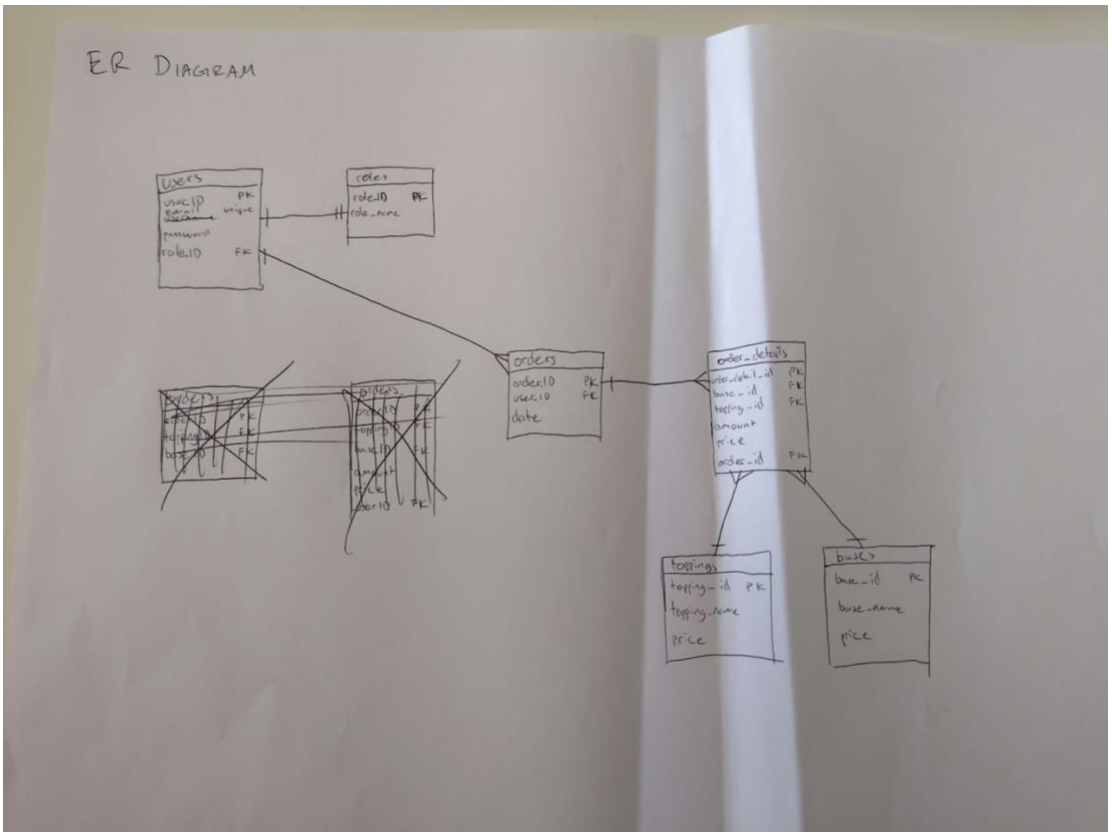
Proces

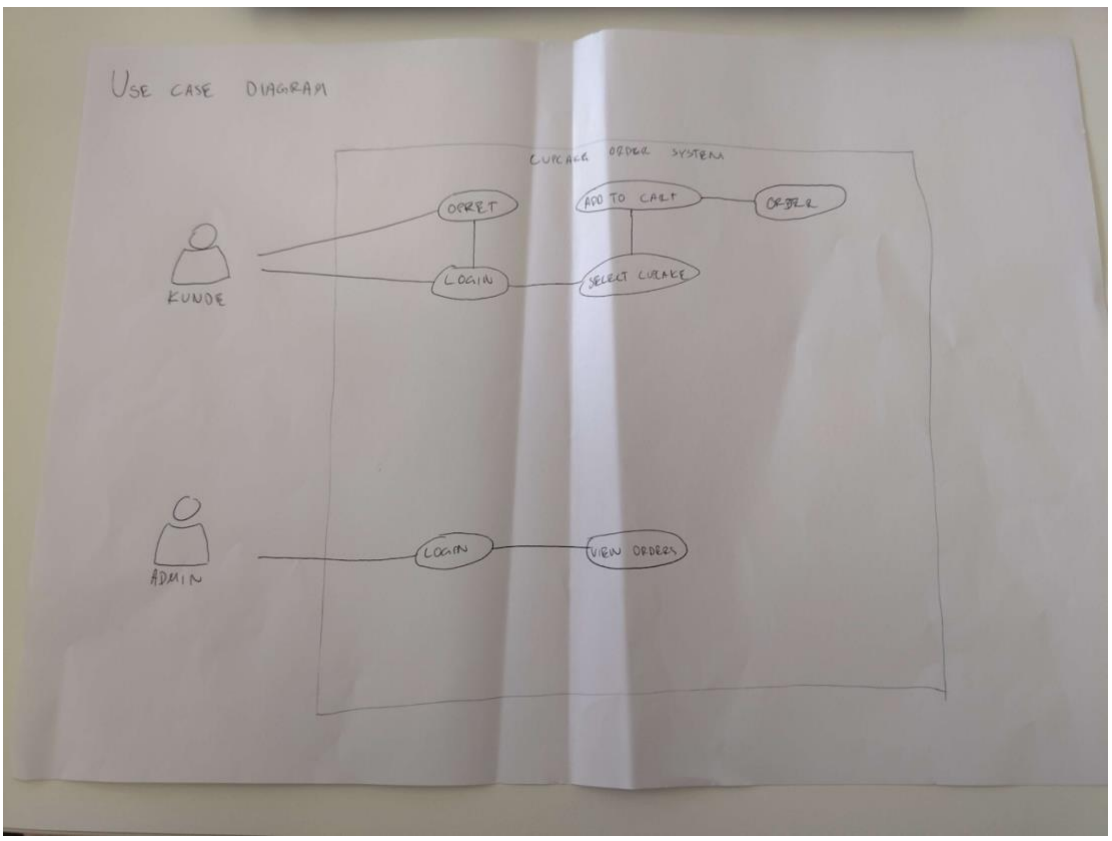
Gruppekontrakt

I starten af projektet udformede gruppen en gruppekontrakt (kan findes under ‘documentation/groupcontract.pdf’ i GitHub repo) for at finde de enkelte gruppemedlemmers styrker og svagheder, så vi havde mulighed for at bruge dem til gruppens samlede fordel, samt være opmærksom på eventuelle udfordringer der kunne opstå under forløbet.
At starte projektet på denne måde har vi alle i gruppen oplevet som værende fordelagtig, da det gav os en god fornemmelse af hvordan projektet vil komme til at forløbe, samt kender hinandens kompetencer, ambition og niveau. Det betød at projektet kunne startes relativt hurtigt, da vi havde en fornemmelse af gruppens, samt vores individuelle, arbejdsproces.

Diagrammer

Vi sørgede for at definere et Use Case-diagram og ER-diagram i begyndelsen af projektet, for at få en fornemmelse af hvad vores system og database skulle indeholde. Det hjalp gruppen til hurtigt at opsætte databasen, samt hvilke klasser der var nødvendige.


ER DIAGRAM
users
user_ID PK
unique
password
role_ID FK
roles
role_ID PK
role_name
orders
order_ID PK
user_ID FK
date
order_details
toppings
topping_id PK
topping_name
price
bases
base_id PK
base_name
price


USE CASE DIAGRAM
CUPCAKE ORDER SYSTEM
OPRET
ADD TO CART
ORDER
LOGIN
SELECT CUPCAKE
KUNDE
LOGIN
VIEW ORDERS
ADMIN

Vi har overvejet at et navigationsdiagram ville være en fordel at lave i begyndelsen af projektet, så vi hurtigt kan få et overblik, hvordan siderne skal routes og interagere med hinanden.

Kanban

For at holde styr på de enkelte opgaver i processen, oprettede vi et Kanban Board igennem vores GitHub repository (dette er public og ligger under “Projects”). Dog blev de opgaver vi oprettede på boardet for små, hvilket gjorde at brugen af det hurtigt blev meget rodet. Som et eksempel oprettede vi opgaver til hver enkelte Controller og Mapper klasser - disse opgaver er svære at definere som done, da de tit har relationer til andre klasser og hinanden. Det har derfor været svært at få en fornemmelse af hvor langt i processen projektet var nået, da “Done” feltet blev fyldt med opgaver, der i realiteten ikke var helt færdige.

Vi har diskuteret, at der ved næste projekt oprettes et overordnet Kanban board der indeholder de User Stories, som bliver defineret i starten af projektet - og for hver påbegyndt (“Doing”) User Story, bliver der oprettet et sub-board, som indholder de underopgaver, der skal til, for at nå i mål.

Git branches og GitHub Pull Requests

For at strukturere vores arbejde brugte vi Git branches ud fra vores Main branch til hver ny funktion, html side/styling, Kanban opgave osv - og til forlængelse af dette benyttede vi os af GitHubs Pull Request feature. Derudover satte vi en restriktion, at der ikke kunne merges til Main, før at et Pull Request var blevet efterset og godkendt. Først definerede vi at alle andre gruppemedlemmer skulle godkende et request, før det kunne merges - det fandt vi hurtigt ud af ikke gav nogen mening, da flere gruppemedlemmer godt kan arbejde på den samme branch. Det endte med, at mindst ét gruppemedlem skulle godkende - men at vi stadig samlet gennemgik hvert request - enten fysisk eller online.

Dette resulterede i at gruppen fik en fornemmelse af, hvad der var blevet lavet, selv på opgaver de ikke selv havde siddet med, samt give input og rette fejl, inden koden blev merget ind på Main. Pull Requests har en funktionalitet til at kommentere direkte på koden på GitHub, så vi alle kunne indsætte rettelser eller forslag på specifikke linjer. Generelt har Pull Requests været en positiv tilføjelse til vores arbejdsproces, da det hjalp med at holde overblik over kodeprocessen, og sørgede for at holde vores Main branch fri for fejl.

Kodestandarder

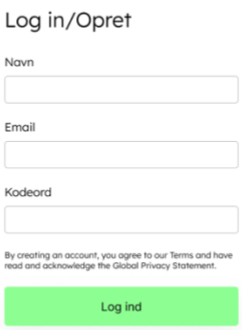
Gruppen diskuterede kun mundtligt vores kodestandarder inden kodestart, som betød vi hurtigt ramte ind i forskelle i navngivning, hvordan camel-, snake-, kebabcase blev benyttet samt fejl i oversættelse mellem dansk og engelsk, og på hvilke dele vi benyttede hvilket sprog.
Det er vigtigt, at der bliver formuleret et kodestandardsdokument til næste projekt, der kontinuerligt kan refereres til under kodeforløbet, så vi sørger for at holde standarden under hele projektet.

Proceskonklusion

Forløbet har generelt fungeret godt, men der er fundet bedre fremtidige arbejdsprocesser, samt arbejdsprocesser der beholdes, men ændres - Kanban board vil blive brugt igen, men måden den implementeres i processen bliver ændret, så opgaverne ikke bliver for små. Kodestandardisering får et større fokus til næste projekt, samt flere diagrammer tegnes og diskuteres ved projektopstart. Vi bibeholder måden vi har arbejdet med branches og Pull Requests vha. Git og GitHub.

Bilag

Bilag 1 – Figma Mockups



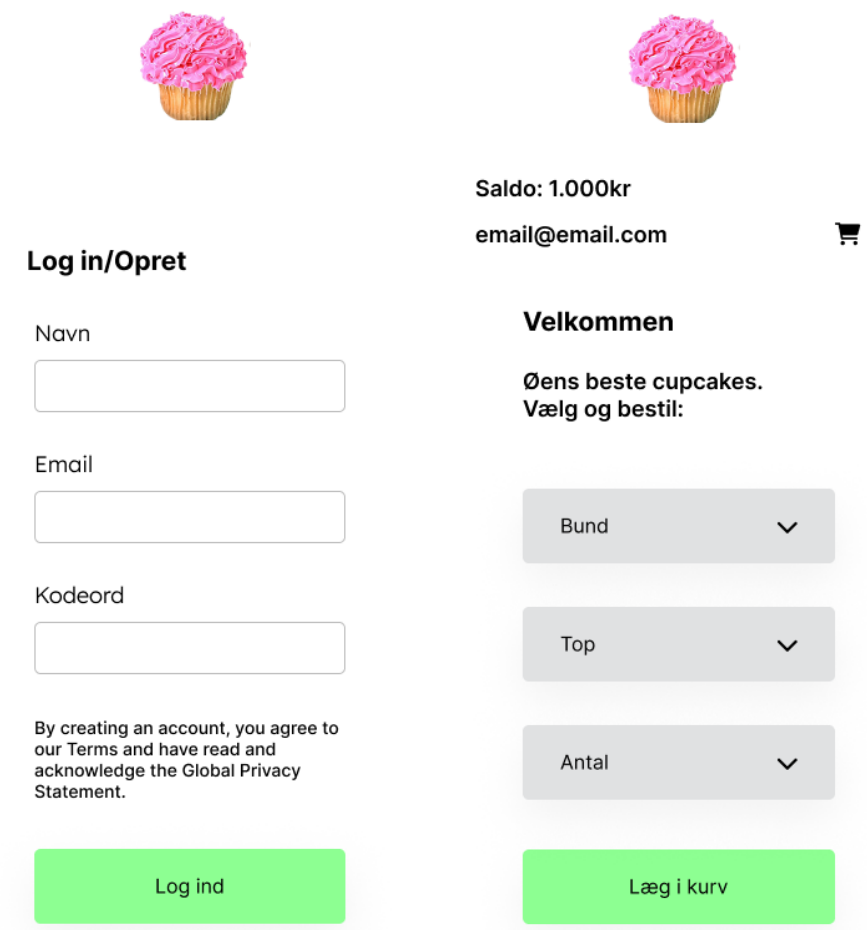
Login, desktop

Ordre Kunder email@email.com

Seneste bestillinger

Ordre Nr	Tid	Kunde	Bestilling	Antal
1113	10:30	Oliver	Choko med banan	1
			Choko med M&M	3
1112	10:25	Emil	Vanilje med Nutella	5
1111	9:30	Tobias	Choko med Flødeskum	1
1110	8:49	Peter	Vanlije med Kakao	2

Seneste ordre, desktop



Login og forside, mobil

Saldo: 1.000kr

email@email.com

Indkøbskurv

Bund	Top	1x	pris
Bund	Top	1x	pris
Bund	Top	1x	pris
Bund	Top	1x	pris
Bund	Top	1x	pris
Total			pris

Bestil

Indkøbskurv, desktop

Saldo: 1.000kr

email@email.com

Indkøbskurv

Choko Vanlije	1x 200kr
Choko Vanlije	1x 200kr
Choko Vanlije	1x 200kr
Choko Vanlije	1x 200kr

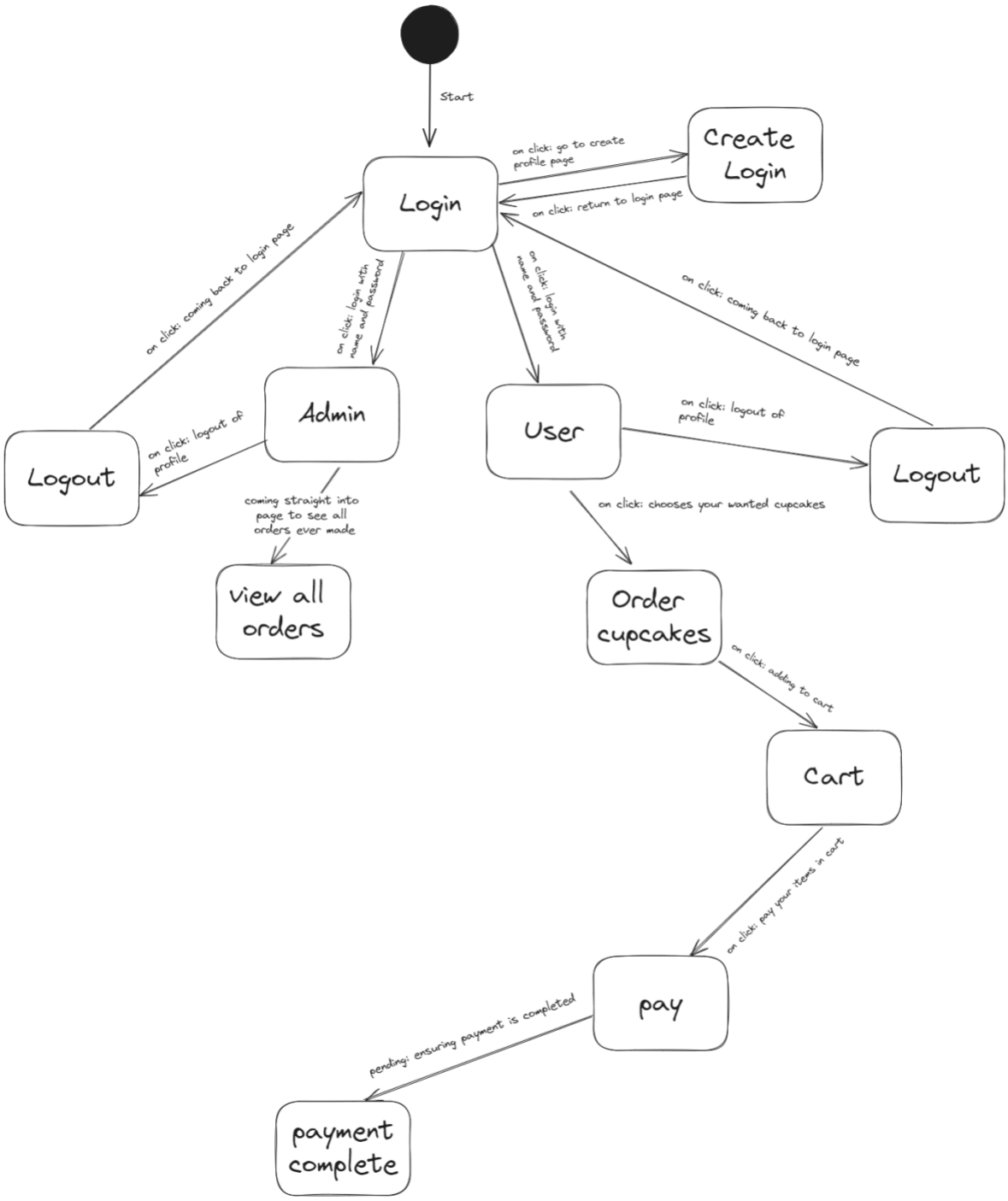
Total 800kr

Bestil

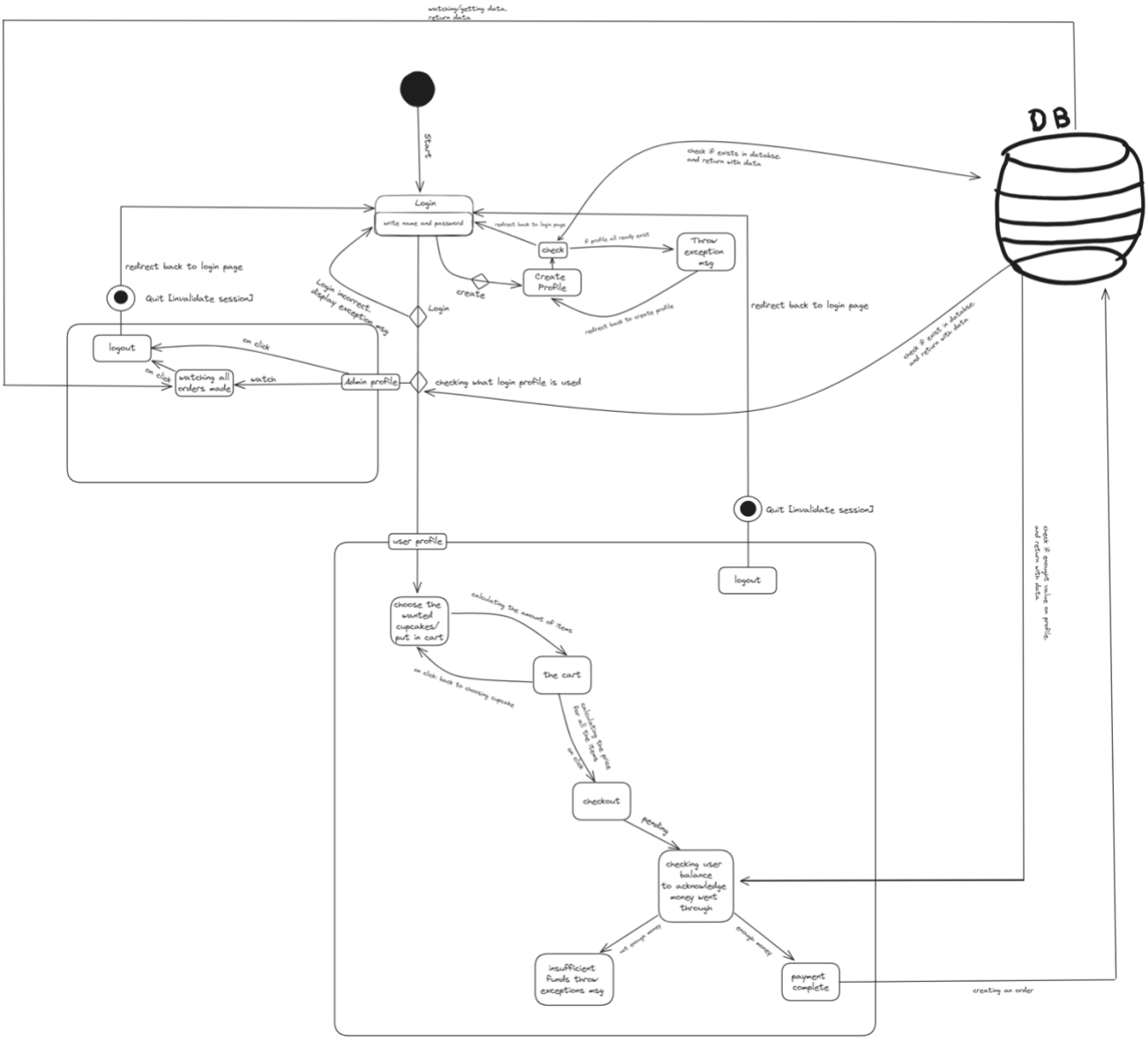
Indkøbskurv, mobil

Bilag 2 – Navigationsdiagrammer

Model 1:



Model 2:


watching/getting data, return data
DB
Start
Login
write name and password
redirect back to login page
check
check if exists in database and return with data
if profile all ready exist
Throw exception msg
Create Profile
create
redirect back to create profile
redirect back to login page
Login incorrect, display exception msg
Login
Quit [invalidate session]
logout
on click
watching all orders made
watch
admin profile
checking what login profile is used
check if exist in database and return with data
Quit [invalidate session]
user profile
logout
choose the wanted cupcakes/ put in cart
calculating the amount of items
the cart
on click: back to choosing cupcake
calculating the price for all the items
on click
checkout
pending
checking user balance to acknowledge money went through
not enough money
enough money
insufficient funds throw exceptions msg
payment complete
creating an order
check if enough value on profile and return with data