2. SEMESTER CUPCAKE PROJEKT

RESUME

Cupcakeprojektet er en skræddersyet løsning til bageriet Olsker Cupcakes på Bornholm, der ønsker en brugervenlig platform til at tilbyde byg-selv cupcakes.

Maria Nørmølle - cph-mn760@cphbusiness.dk
Github: mariaNoermoelle

Rodney Muyanga - cph-rm135@cphbusiness.dk
Github: RodneyMuyanga

Michella Stuhr Bech - cph-mb606@cphbusiness.dk
Github: MichellaBech

DAT 2. SEM F24 HOLD B
11-04-24

Indholdsfortegnelse

Indledning

Dette projekt er en hjemmeside til salg af cupcakes. Vi har lavet en hjemmeside, hvor man kan lave en bruger, logge på og lægge byg-selv cupcakes i en kurv. Denne rapport henvender sig til andre på 2. semester på datamatikeruddannelsen.

Baggrund

Cupcakeprojektet henvender sig til bageriet Olsker Cupcakes på Bornholm, som gerne vil have, at man skal kunne hente byg-selv cupcakes fra dem. Hjemmesiden her, er med til at gøre valg af cupcakes nemmere, så man kan bygge sin egen cupcake. Byg-selv cupcake betyder, at man selv kan vælge topping, bund og hvor mange man vil have af en bestemt cupcake. Herefter kan man lægge dem i sin kurv. På hjemmesiden skal man kunne oprette en profil og logge ind. Når man er logget ind, skal man kunne se sin kurv. Som administrator skal man kunne se alle kundernes ordrer, og slette ordrer, når der er betalt.

Teknologi valg

Vi har brugt disse teknologier:

IntelliJ IDEA 2023.2.

PgAdmin 8.3

Docker Engine v25.0.2

Javalin 6.1.3

Java 17

HTML 5

CSS 4.15

Thymeleaf template engine 3.1.2

Postgressql 42.7.2

Krav

Virksomheden håber, at de med deres nye system kan tilbyde kunderne byg-selv cupcakes, som kan afhentes i butikken. De håber, at deres kundekreds kan blive større, når kunder helt selv kan designe en cupcake med lige præcis de smage de kan lide, så kunderne ikke er afhængige af det faste udvalg i butikken.

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 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.

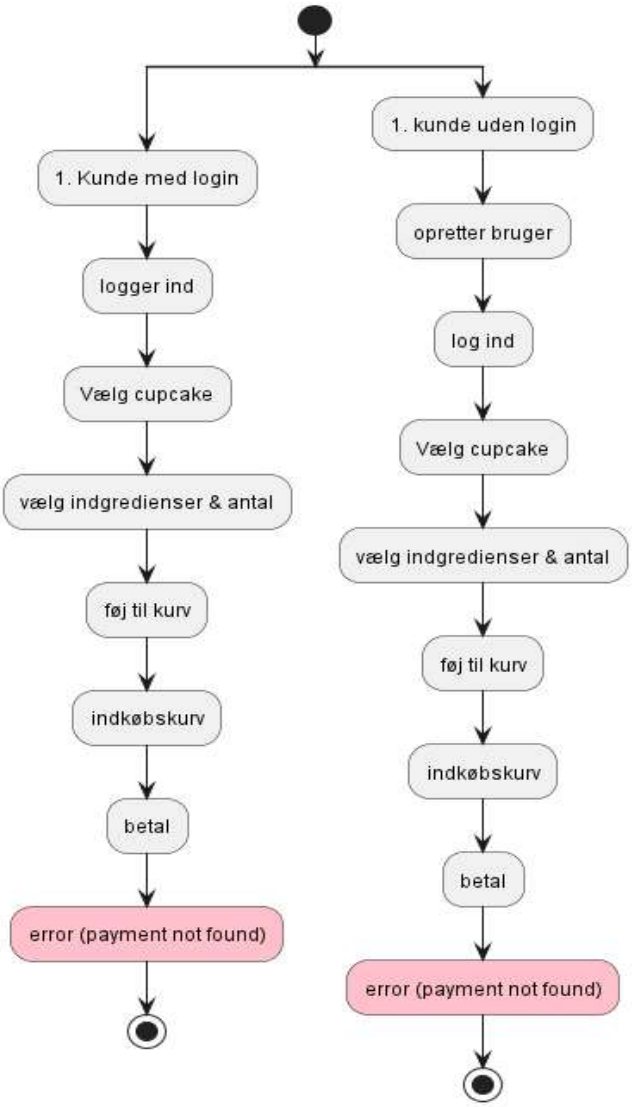
US-7: Som administrator kan jeg se alle kunder i systemet og deres ordrer, sådan at jeg kan følge op på ordrer og holde styr på mine kunder.

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.

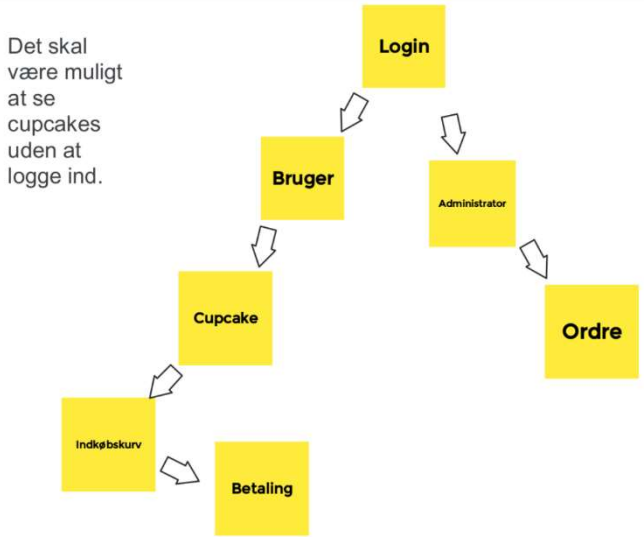
Aktivitetsdiagram

Dette er vores nuværende aktivitetsdiagram. Vi havde oprindeligt ikke planlagt at bruge aktivitetsdiagrammet som retningslinje, så vi har ikke udarbejdet en "TO-BE" version. Førstegangsbesøgende skal oprette en bruger ved at angive deres e-mail, oprette et brugernavn og et password. Når dette er gjort, kan man logge ind med de angivne oplysninger og begynde at sammensætte sin egen cupcake ved at vælge ønskede ingredienser og antal cupcakes, som derefter tilføjes til indkøbskurven. Vores brugeroplevelse slutter her, da vi ikke har nået at aktivere betalingsfunktionaliteten på grund af problemer med databasen.

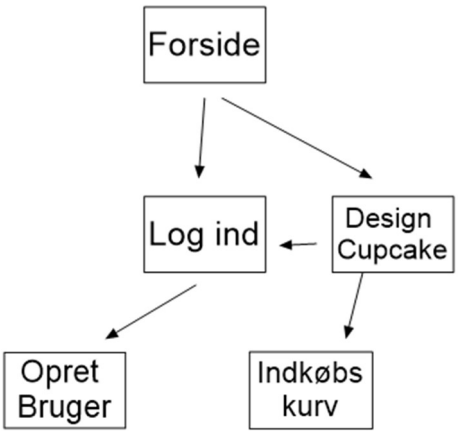


Domæne model og ER Diagram

Domæne model



Her er den første domænemodel, vi lavede, inden vi begyndte på projektet. Vi vurderede dog, at man gerne måtte se cupcakes mulighederne inden man loggede ind. Administratoren skulle have mulighed for at se alle ordrer.

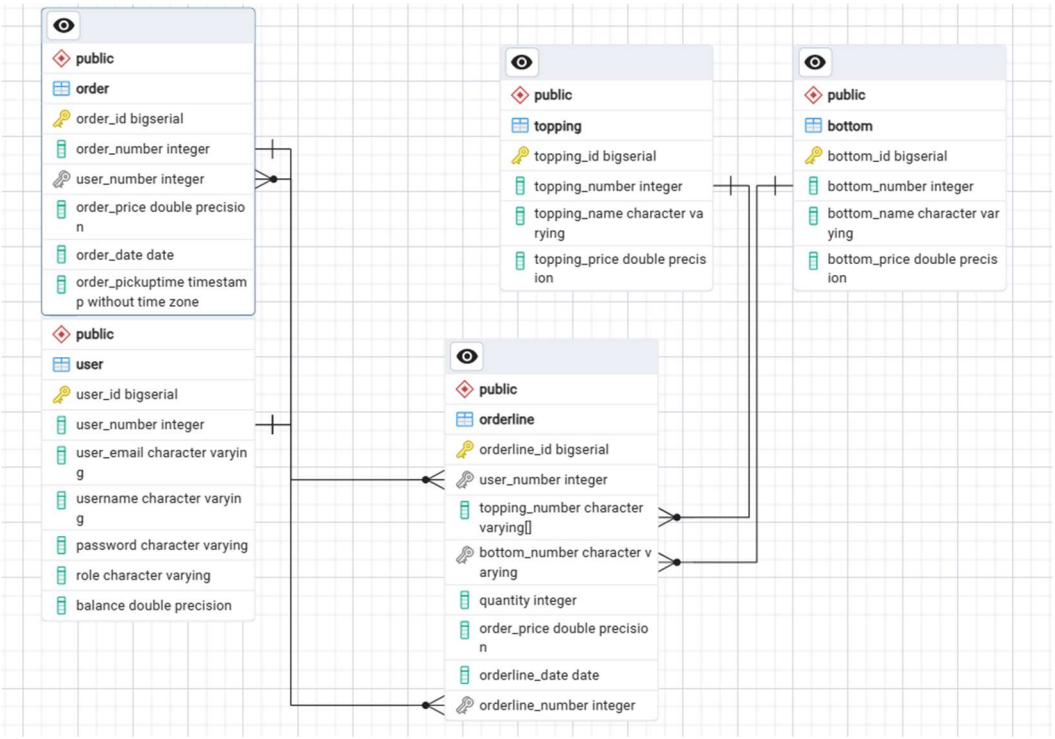


Den endelige domænemodel, vi endte med. Fra vores forside kan man logge ind eller trykke ind på den side, hvor man selv kan designe sin cupcake. Hvis man går ind på log ind siden, kan man derfra oprette en bruger. Efter man har oprettet en bruger, kommer man tilbage til forsiden, hvor man bliver bedt om at logge ind. Man kan ikke tilføje til indkøbskurven, hvis man ikke er logget ind. Inde på siden, hvor man kan designe sin egen cupcake, kan man vælge topping, bund og antal. Herefter kan man putte det i indkøbskurven. Hvis man ikke er logget ind, kommer der en besked med det. Øverst på siden, hvor man selv designer sin cupcake, er der en log ind knap, så man nemt kan logge ind for at bestille. Når man er logget ind, kommer man tilbage til forsiden. Efter man har valgt sine

cupcakes, kan man trykke på indkøbskurven og se hvilke slags cupcakes man har bestilt og hvor mange man har bestil.

ER diagram

Vi bruger dette ER diagram til at have overblik over databasen. Nedenfor er ER diagrammet over vores database.



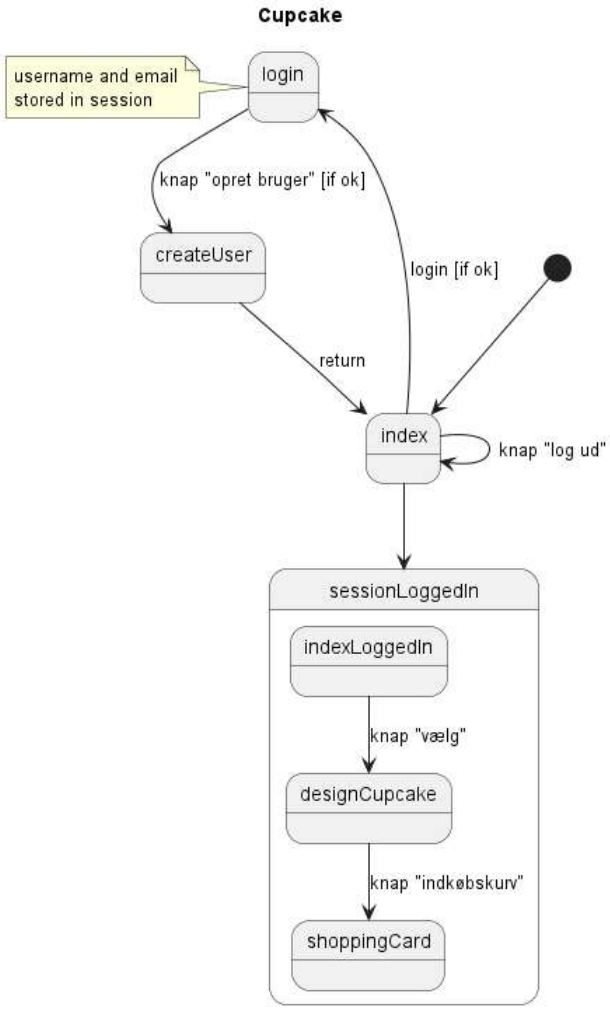
Denne database overholder ikke 3. normalform. Det ses i topping og bottom. De to tabeller er næsten ens, hvor topping_number i topping giver nummeret på en bestemt topping, gør bottom_number det samme i bottom. I topping tabellen ses det, at topping_number er afhængig af topping_name, og topping_price er også afhængig af topping_name. For at få databasen her til at overholde 3. normalform kunne vi lave en ny tabel med priser i, og koble de to tabeller på pris-tabellen. Det har vi ikke gjort pga. tidspres.

Orderline tabellen skulle forbinde tabellerne user og order. Dette er dog ikke tilfældet i det program vi har lavet. På vores hjemmeside bruger vi tabellen user, til at logge ind og gemme de cupcakes man vælger, på en bestemt session. Når man logger ud, gemmes dataen fra sessionen ikke. Vi ville gerne have lavet det sådan, at når man køber cupcakes ville de blive gemt i databasen, det havde vi dog ikke tid til. Hvis vi havde haft tid til at implementere den funktion, ville vi skulle bruge orderline, så de to tabeller blev forbundet.

Det leder os videre til at de to tabeller order og orderline ikke bliver brugt i vores program. Det gør de ikke, netop fordi vi har valgt at gemme topping og bottom direkte på user. Dette valg er taget pga. tidspres.

Navigationsdiagram

Nedenfor ses navigationsdiagrammet for vores cupcake projekt.



Det her navigationsdiagram beskriver hvordan flowet er vores program. Vi starter på den sorte prik, og kommer ind på index siden. Her kan man bevæge sig til login. Hvis man ikke har en bruger, kan man komme videre til createUser. Når man har oprettet en bruger, kommer man tilbage til index siden. Fra index skal man logge ind med den nyligt lavede bruger. Så kommer man ind i sessionen, og kan nu vælge og se cupcakes. Den side man kommer ind på, når man trykker på "vælg" knappen, er designCupcake. I designCupcake vælger man hvilke cupcakes man gerne vil have. Efter man har valgt cupcakes, trykker man på "indkøbskurv" knappen, der kan man se hvilke cupcakes man har valgt. Her ender flowet, da man ikke kan gå tilbage til forsiden.

Særlige forhold

I dette afsnit kommer vi ind på vores Figma og Kanban, som viser vores arbejdsform og vores vision for hjemmesiden. Desuden vil vi beskrive dele af koden, som vil give en dybere forståelse for vores projekt. Vi vil komme ind på vores createuser metode i klassen UserController, som laver en ny bruger. Og vores html i designcupcake, hvor vi henter dataen fra databasen, som bliver vist på hjemmesiden.

Designcupcake.html

I denne del beskriver vi designcupcake.html. Html siden vil blive beskrevet fra toppen og ned. Der lægges vægt på de vigtigste dele af denne side.

Herunder ses toppen af designcupcake.html

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>designcupcake</title>
6      <link rel="stylesheet" th:href="@{css/forside.css}" href="../public/css/forside.css">
7  </head>
8  <div class="showingemail" th:if="${session.currentUser != null}">
9      <span th:text="${session.currentUser.userMail}"></span> |
10 </div>
11 <body>
12 <div class="loginbutton" th:if="${session.currentUser == null}">
13     <form method="get" action="loginpage">
14         <button type="submit">Log ind</button>
15     </form>
16 </div>
17 <h1>Byg Selv Drømmecupcake</h1>
18 <br><br>
19
```

Det første vi vil komme ind på, er i linje 8-10, her har lavet en div med en klasse, “showingemail”, som tjekker på om brugeren er logget ind. Hvis brugeren er logget ind, vil man kunne se e-mailen oppe i hjørnet af hjemmesiden. På linje 12-16 er der en loginknap, “Log ind”, hvis man ikke er logget ind, som henviser til vores loginpage, hvor man kan lave en ny bruger. Man skal lave en ny bruger, for at kunne bestille cupcakes. Man kan se de cupcakes, der er at vælge imellem, uden man er logget ind.

Herunder ses starten på tabellen, som gør at man kan vælge cupcakes i designcupcake.html.

```
17  <h1>Byg Selv Drømmecupcake</h1>
18  <br><br>
19
20  <form method="post">
21      <table>
22          <tr>
23              <td><label for="bottom">Vælg din yndlings smag: </label></td>
24              <td>
25                  <select name="bottom" id="bottom" required>
26                      <option value="">Vælg bund</option>
27                      <!-- Iterate over bottoms and populate options -->
28                      <option th:each="entry : ${bottoms.entrySet()}" th:value="${entry.key}"
29                              th:text="${entry.key + ' - ' + entry.value + ' DKK'}"></option>
30                  </select>
31                  </select>
32                  <input type="hidden" name="bottom_price" th:value="${bottoms[bottom]}">
33
34
35              </td>
36          </tr>
37          <br/>
38          <tr>
39              <td><label for="topping">Vælg din yndlings smag: </label></td>
40              <td>
```

Denne del af html'en viser starten på den tabel, som gør at man kan vælge de forskellige dele af cupcakes. Vi har valgt at lave en dropdown-menu. Denne del viser bundenes menu. Vi henter data fra databasen direkte ind i på hjemmesiden. Det gør vi i linje 28-29.

I linje 28 starter vi med at lave en iteration over bottoms fra cupcakeController klassen. Vi bruger bottoms her, fordi det er navnet på det linkedhashmap, som kommer fra getAllBottoms metoden i cupcakeMapper klassen. I cupcakeController klassen kalder vi det linkedhashmap, som kommer fra getAllBottoms metoden. I cupcakeControlleren hedder linkedhashmappet "bottoms". Og det er det bottoms navn, vi kalder i linje 28 under "entry : ${bottoms.entrySet()}".

I næste stykke kode, "th:value :"${entry.key}", tilføjer vi en værdi til html elementet, som bliver koblet op på den nøgle, vi har fra linkedhashmappet.

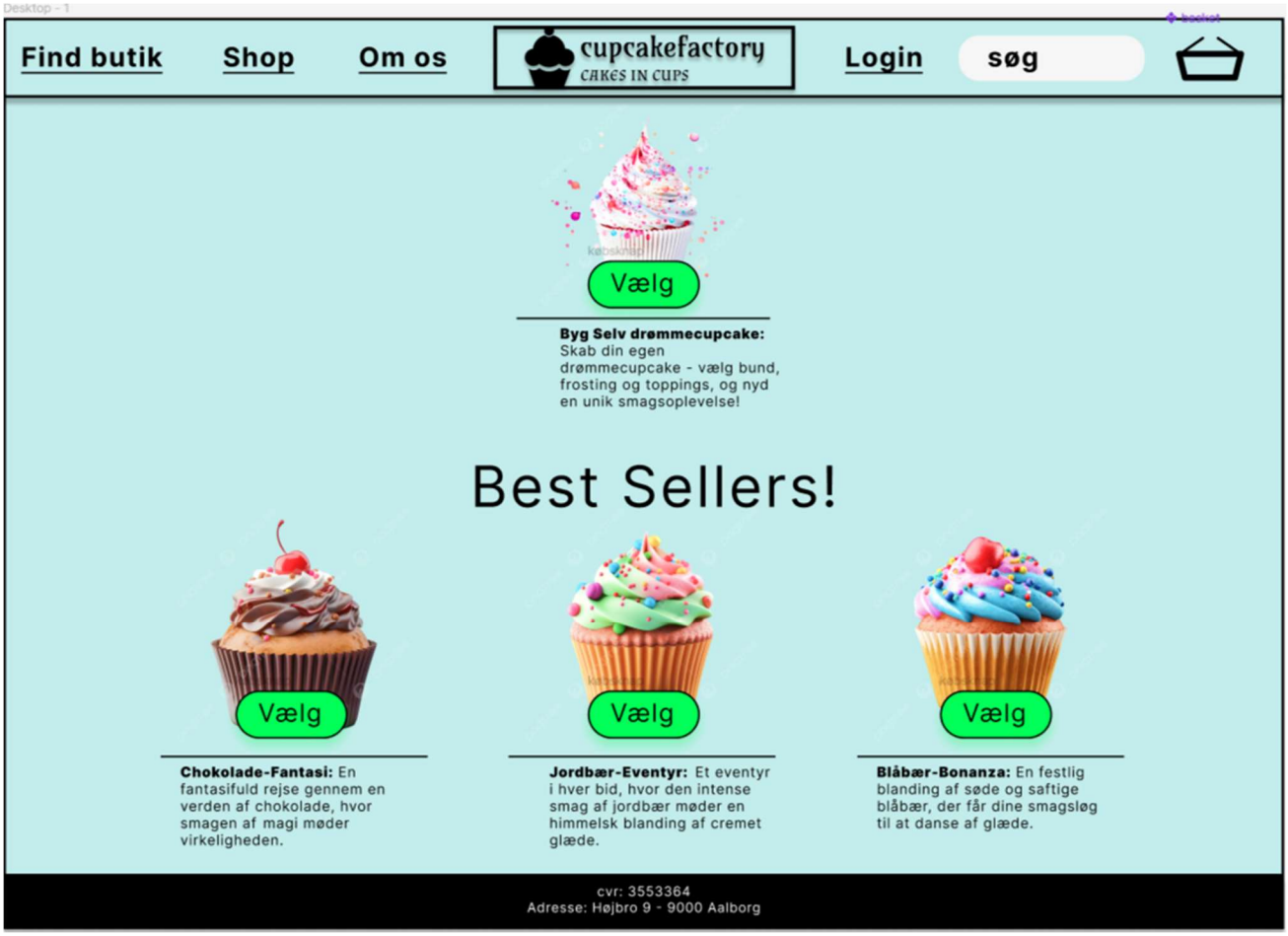
I linje 29 vises teksten i dropdown-menuen med nøglen, som er chokolade, vanilje mv. hentet fra databasen. Og værdien, altså prisen, som er koblet op på den specifikke nøgle.

I topping delen af koden, fra linje 39-49, sker det samme som det, der er beskrevet ovenfor. Dog med den forskel, at dataene bliver hentet ned fra topping tabellen i stedet for bottom tabellen.

Designet i Figma & planen i Kanban

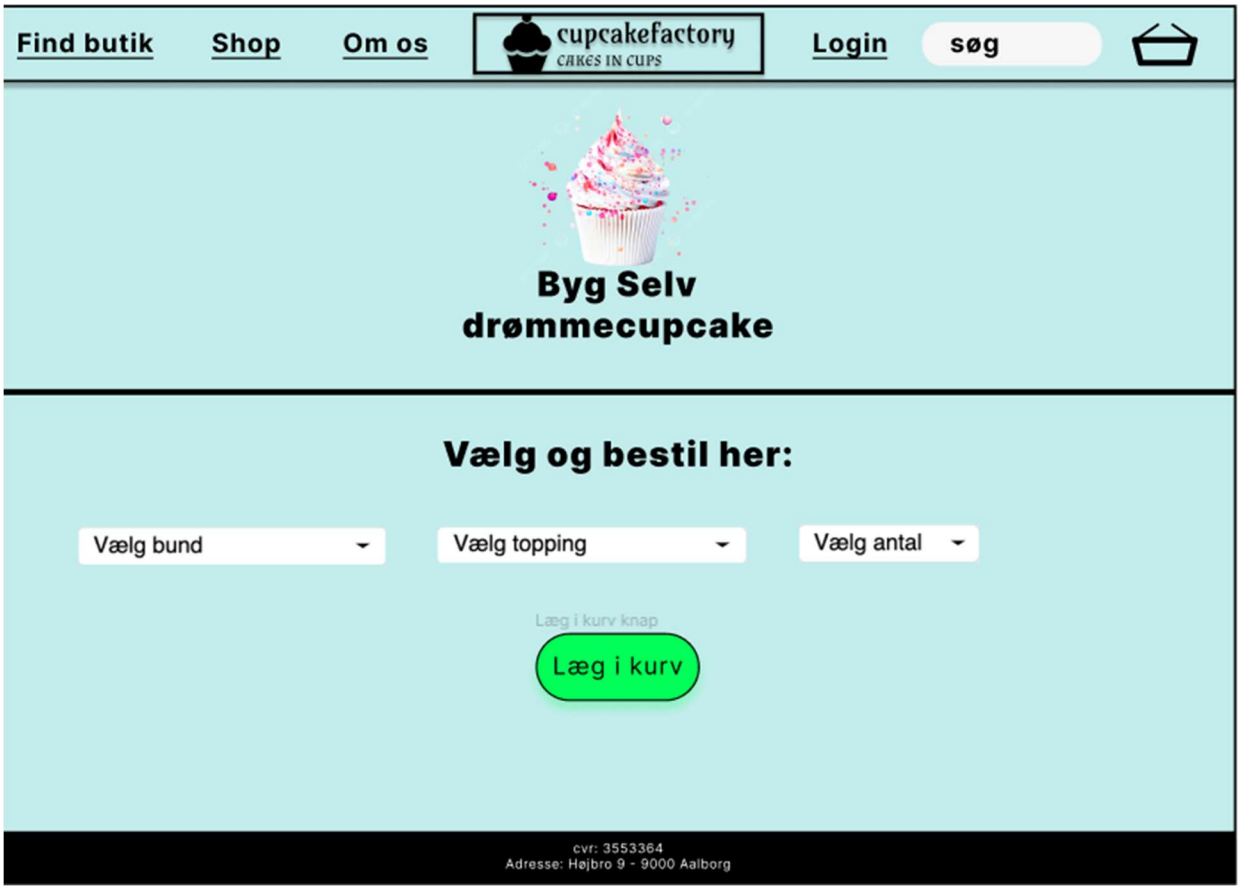
Figma (mockup):

Vi lavede en Figma, som vi brugte til lave vores mockup. Her brugte vi rigtig meget tid på at designe, så vi havde nogle klare retningslinjer, og en fælles forestilling af hvordan vores endelige webpages skulle se ud.



Forside:

Forsiden viser forskellige valgmuligheder for besøgende. Øverst kan man vælge "Byg din egen drømmecupcake" hos Cupcakefactory med en indbydende tekst og billede af en lækker cupcake. Nedenfor præsenteres de bedst sælgende cupcakes med fængende navne og beskrivelser, så besøgende hurtigt kan få et overblik og blive fristet.

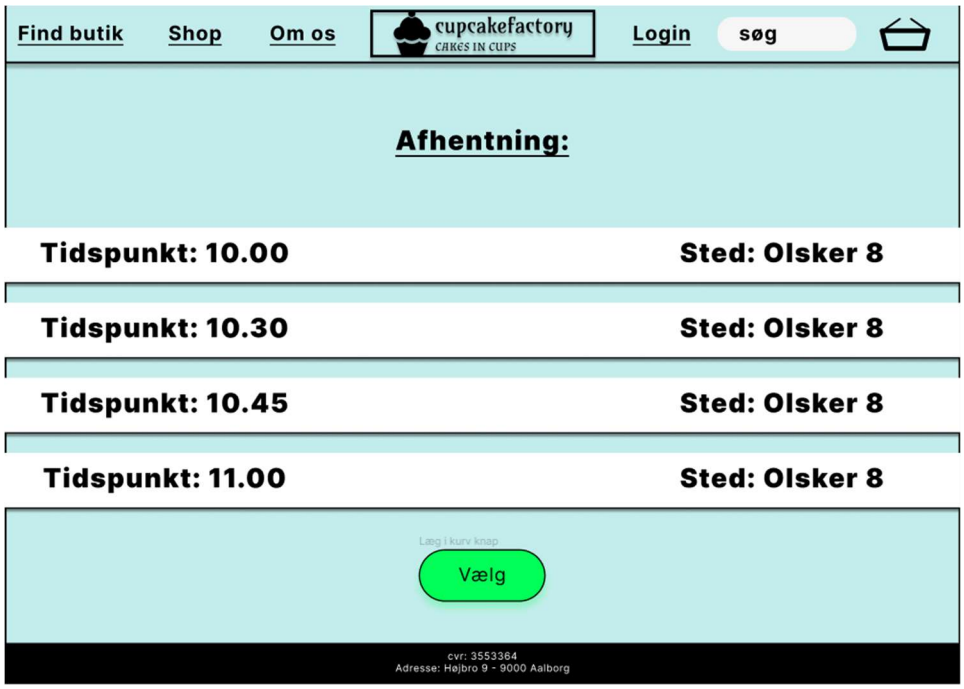


Byg selv drømmecupcake:

Her havde vi forstillet os, at man som besøgende havde valgt at klikke på “byg selv cupcake”, hvor man ville blive mødt med denne webpage. Det muligt at vælge bund, topping og antal, hvorefter man kan læg sit valg i kurven.

Indkøbskurven:

Her skal det forestille den webpage, som man kommer ind på, når man har sammensat sin "byg selv cupcake". En lille oversigt over de tilvalgte ingredienser og en samlet total sum. Hvis oversigten stemmer overens med de ønskede valg, trykker man køb og ryger videre til afhentningsplatformen.



Afhentning:

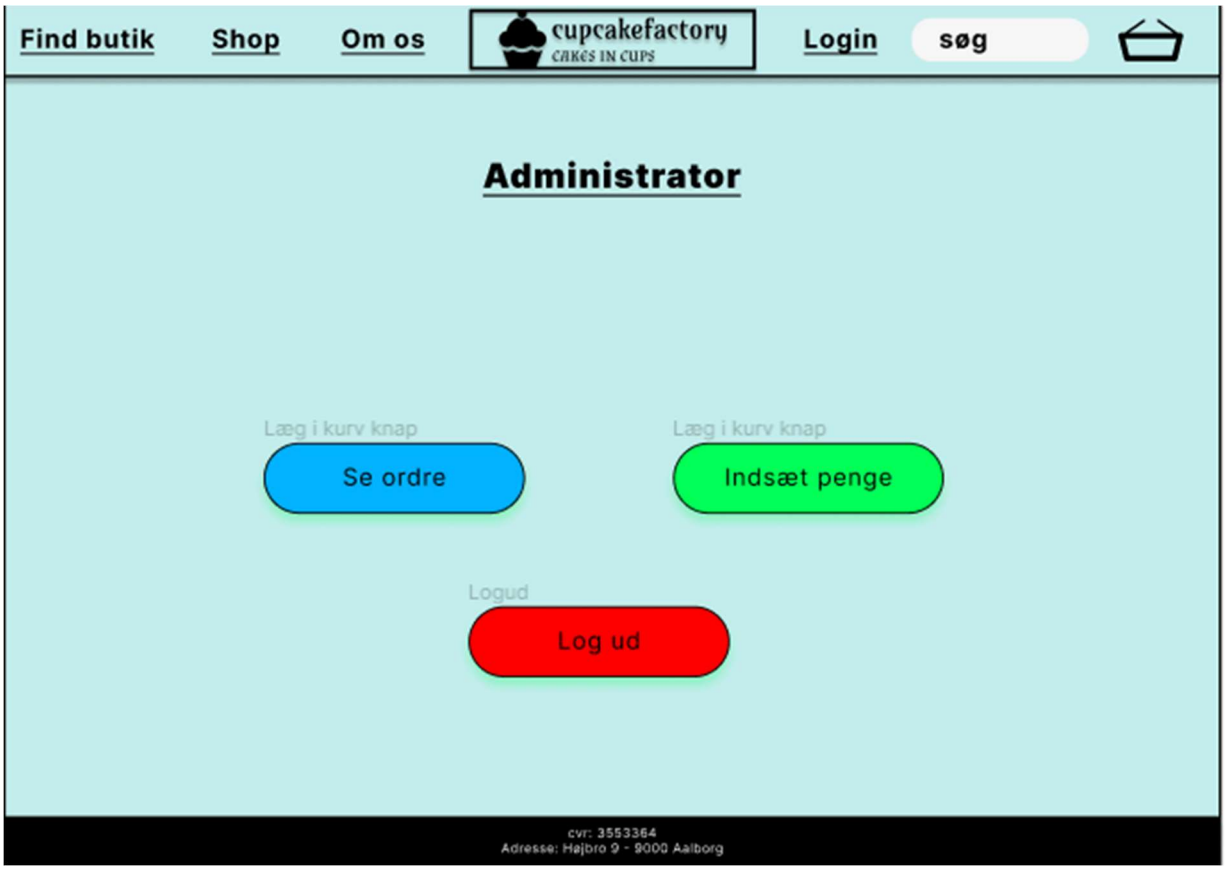
Når man har accepteret sin ordre, kan man vælge et afhentningstidspunkt og et afhentningssted, i tilfælde af virksomheden befinder sig i flere forskellige byer.

Afslutning på køb:

Her ses ordrens afhentningssted og tidspunkt, samt den totale pris.

Vælge profil:

Sådan havde vi forestillet os profilvalget skulle se ud, når man trykker på login. Login på venstre side skulle være forbeholdt eksisterende kunder og administrator, imens den højre side skal være forbeholdt nye kunder, som endnu ikke har en oprettet konto. Her har vi bare brugt billeder fra nettet til at illustrere inputformularerne.

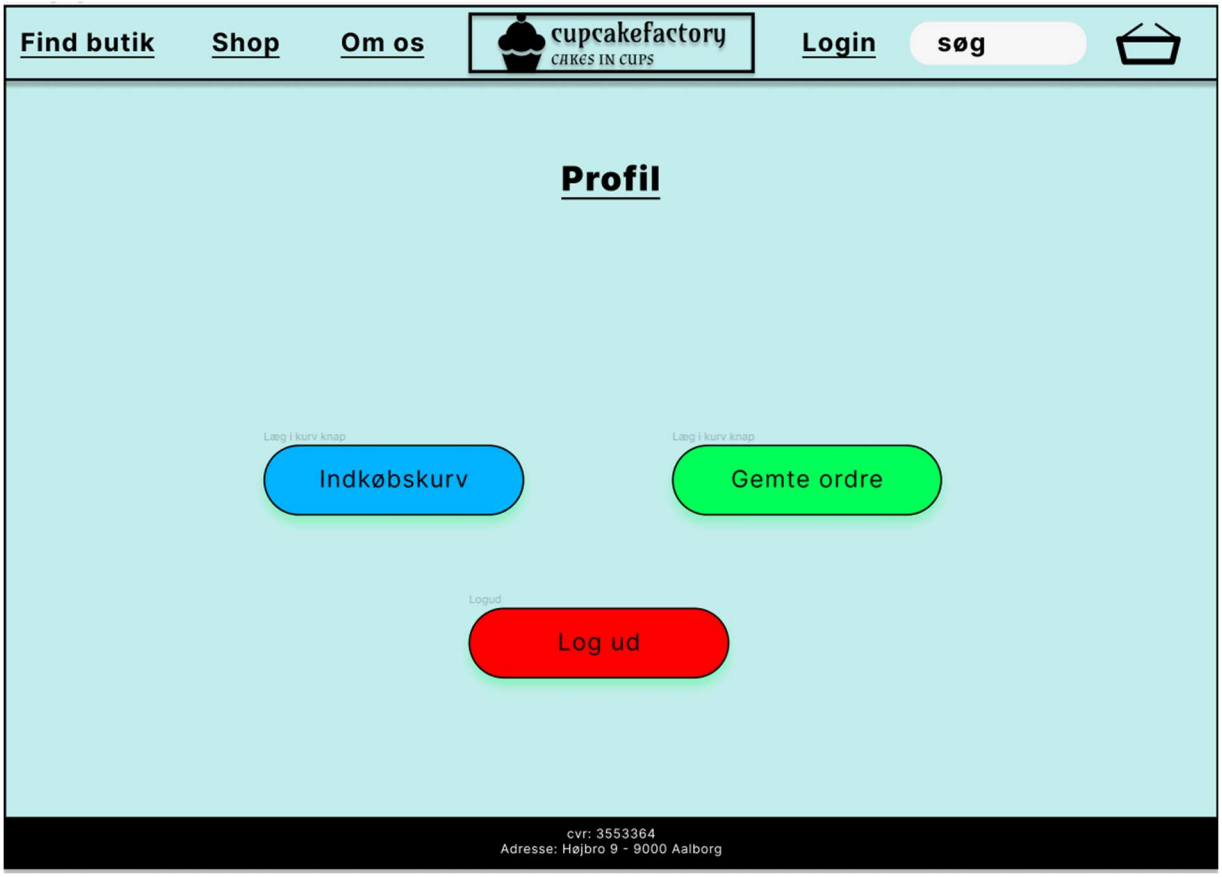


Administrator:

Hvis man er logget ind som administrator, så vil ens profilside se således ud.

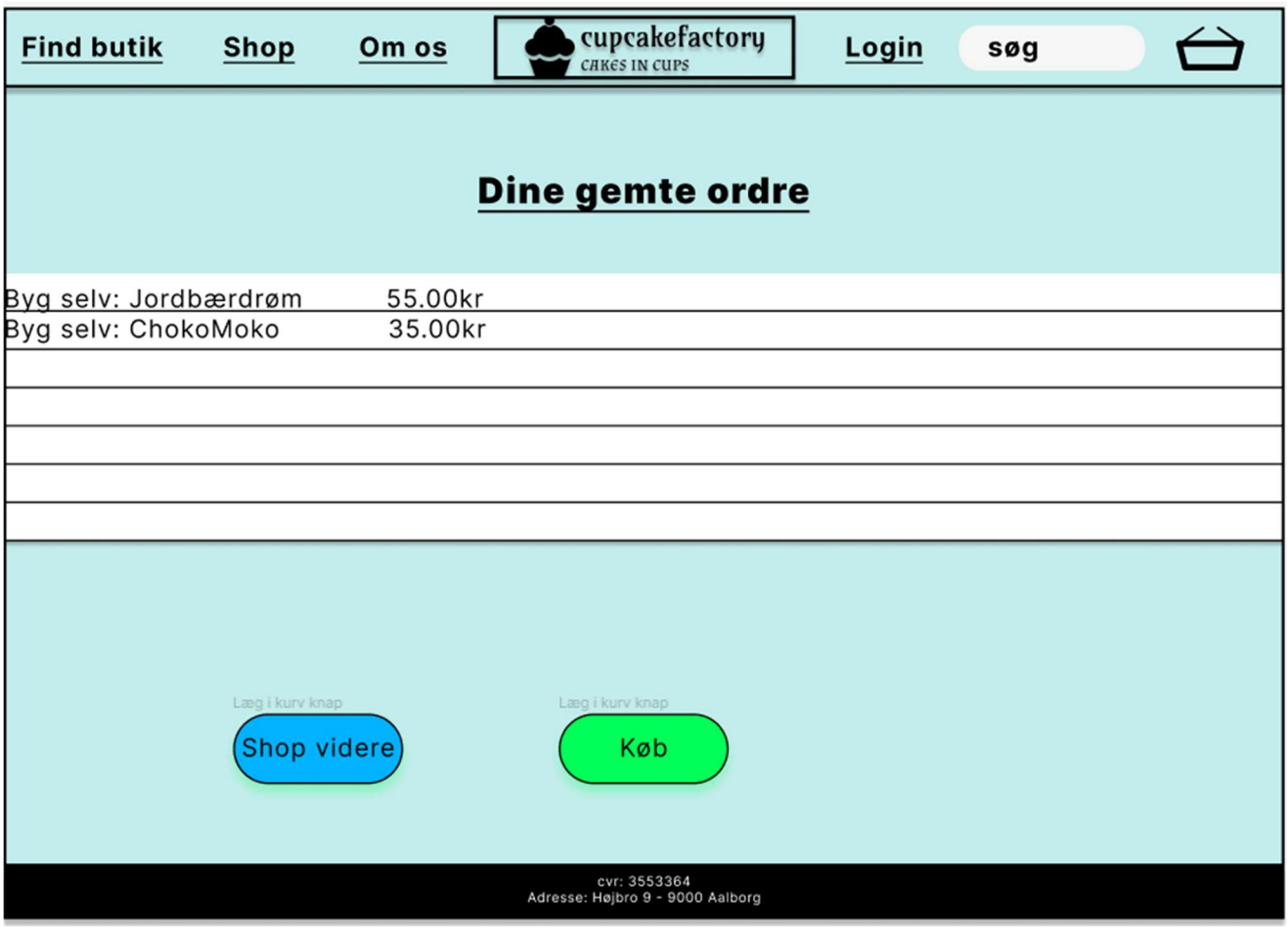
Administrator indsætter penge:

Vælger administratoren "indsæt penge", ville man blive mødt med denne side. Dette er en oversigt over kunderne og deres saldo. Den grønne knap skal trykkes, også vil man kunne indsætte et vilkårligt beløb på kundens konto.



Kundeprofil:

Sådan vil det se ud for en kunde, som logger ind. På næste billede har kunden trykket på “Gemte ordre”.

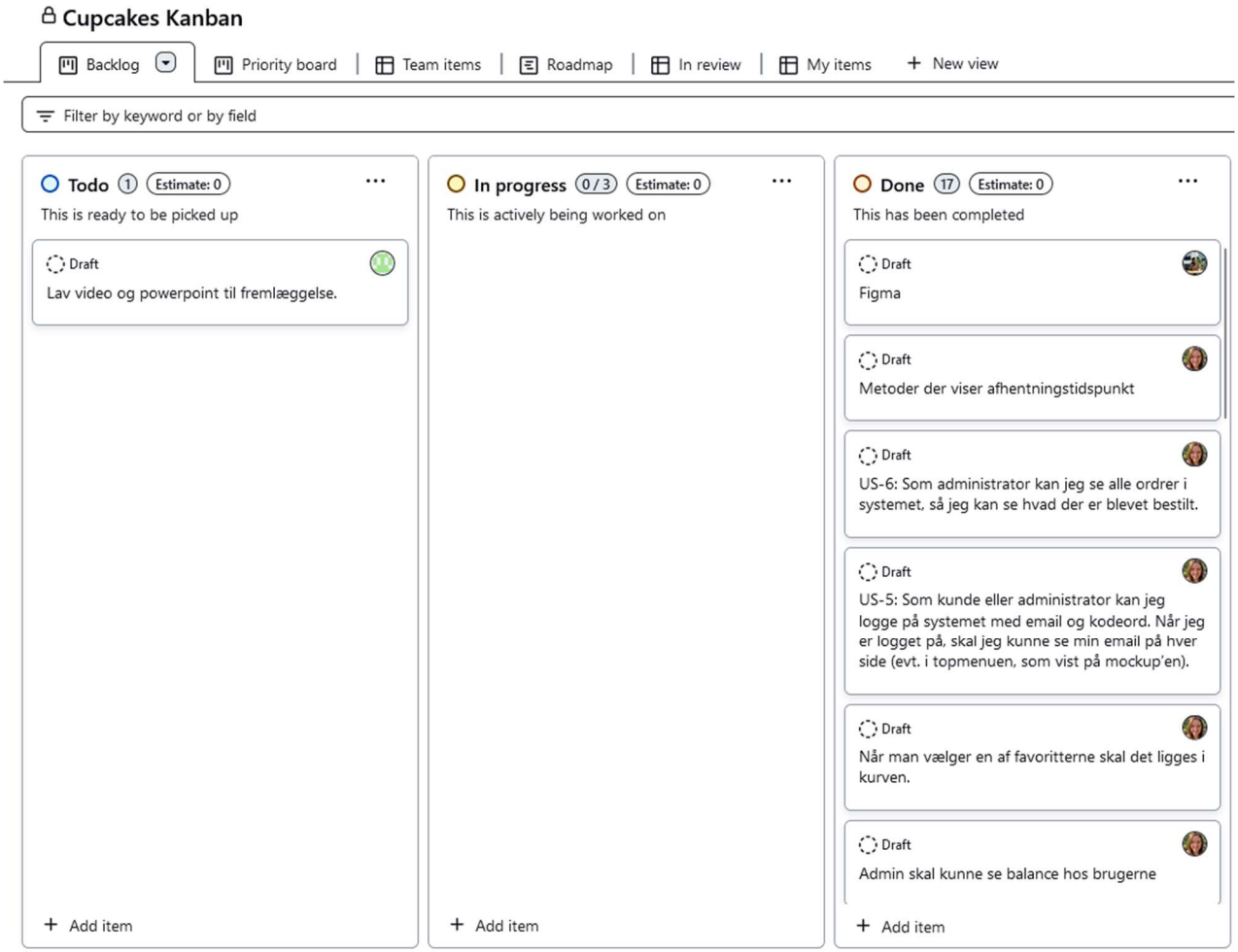


Gemte ordre:

Herinde vil det være muligt at se dine gemte ordre. Man ville kunne klikke på en ordre, hvorefter den ville blive markeret og så kan man trykke "køb" for at købe ordren.
Man kan også vælge at "Shoppe videre", hvis man bare skulle ind og kigge, hvad man havde gemt.

Kanban

Kanban har vi brugt flittigt. Det er noget af det første, vi har brugt tid på inden vi startede med vores projekt. Vi bruger kanban via github til at skrive alle vores mål ned, om nogle af målene er igangværende eller om de er færdige, simpelthen for at få et overblik over hvem der laver hvad og hvornår. Vi synes, at det er værd at bruge tid på, da man i denne proces får tænkt projektet igennem og vendt alle aspekter. Overordnet set synes vi, Kanban er et godt redskab.



UserController, createuser method

I vores UserController har vi en metode, der hedder private static void createuser(Context ctx, Connectionpool connectionpool).

```
private static void createuser(Context ctx, ConnectionPool connectionPool)
{
    //hent form parameter
    String email = ctx.formParam("email");
    String username = ctx.formParam("username");
    String password1 = ctx.formParam("password1");
    String password2 = ctx.formParam("password2");

    if (password1.equals(password2)) {
        try {
            UserMapper.createuser(email, username, password1, connectionPool);
            ctx.attribute("message", "Du er oprettet som bruger, med brugernavn: " +
username + ". Nu kan du logge ind.");
            ctx.render("index.html");

        } catch (DatabaseException e) {
            ctx.attribute("message", "Dit brugernavn eller email findes allerede, prøv
igen eller log ind");
            ctx.render("createuser.html");
        }

    } else {
        ctx.attribute("message", "Dine to passwords matcher ikke, prøv igen");
        ctx.render("createuser.html");
    }


}
```

Vi starter med at hente form parameterne fra createuser.html siden, som er de værdier brugeren indtaster. herefter bliver password 1 og password 2 sammenlignet. Hvis de er ens, bliver der kaldt på UserMapper.createuser(), hvor man bliver oprettet som bruger i databasen. Creatuser() metoden i UserMapper, håndterer SQL delen af koden, hvor der bliver brugt CRUD. I dette tilfælde bruger vi create med et INSERT statement. Når man er oprettet, bliver man sendt til forsiden med beskeden "Du er oprettet som bruger, med brugernavn "brugernavn". Nu kan du logge ind. Hvis man indtaster

et brugernavn eller e-mail der findes i databasen i forvejen, vil man få printet denne besked: “Dit brugernavn eller email findes allerede, prøv igen eller log ind”, mens man bliver på createuser.html siden. Der bliver også printet en besked til brugeren, hvis adgangskoderne ikke er ens, “Dine to passwords matcher ikke, prøv igen”, hvor man også bliver på createuser.html siden.

Vi valgte at sætte unique constraint i databasen, hvor der er sat én på username og én på user_email. Det er i databasen at både brugernavn og e-mail bliver tjekket om de er ens med de allerede oprettede brugere i databasen.

Status på implementation

User stories	Status på implementation
US-1: Kunde kan bestille og betale, vælge valgfri bund og top, hente ordre.	Vi har gjort det muligt at opfylde dele af US-1. Man kan sammensætte med en valgfri bund og top. Vi nåede ikke at gøre det muligt at lave afhentningstid, eller at betale for den valgte cupcake.
US-2: Kunde kan oprette konto og gemme ordre.	Det er muligt at oprette en bruger, og logge ind med den oprettet bruger. Dog kan man ikke vælge profil, man kan ikke være administrator eller gemme ordre.
US-3: Administrator kan indsætte beløb på en kundes konto direkte i Postgres.	Vi nåede ikke at gøre det muligt at vælge administrator. Hver kunde i vores database har 400kr på sin konto, når man bliver oprettet i vores system. Disse 400kr kan dog ikke tages i brug.
US-4: Kunde kan se ordrelinier i indkøbskurv, og se den samlede pris.	Kunden kan ikke se ordrelinjer. Vi havde sat vores database op til at håndtere ordrelinjer, men fik ikke implementeret det i vores backend. Vi gjorde det muligt at komme ind i indkøbskurven, men den samlede pris kan kunden ikke se, selvom vores backend har en metode der håndterer den matematiske del og lægger tallene sammen.
US-5: Kunde/administrator kan logge på med email og kodeord. Se email på hver side.	Kunden kan logge på. Kundens login fremgår også oppe i hjørnet.
US-6: Administrator kan se alle ordrer.	Dette er ikke muligt, da vi ikke har en administrator del implementeret i vores system.
US-7: Administrator kan se alle kunder og deres ordrer.	Ingen oversigt over kunderne. Vores mockup i figma havde en side designet til denne del, men vi nåede ikke så langt med koden.

US-8: Kunde kan fjerne en ordrelinie fra indkøbskurv.	Dette nåede vi ikke.
US-9: Administrator kan fjerne ordre, så systemet ikke kommer til at indeholde ugyldige ordrer.	Dette nåede vi ikke.

Proces

Hvad var vores planer for teamets arbejdsform og projektforløbet?

Kanban: Vi planlagde via Kanban hvem der skulle gøre hvad. Vi lavede mange underopgaver, som vi delte ud imellem os. Rodney satte sig på frontend-delen. Maria og Michella aftalte at lave backend-delen.

Standup-meetings: Vi talte om at lave standup-meetings, så vi kunne følge hinandens proces og forventningsafstemme.

Arbejdsform: Vi aftalte at mødes på skolen og arbejde på projektet i dagtimerne. Både så vi nemmere kunne aftale med hinanden hvad der skulle ske, og så vi løbene kunne tale med hinanden. Men også så vi havde fri, når vi gik hjem.

Hvordan kom det til at forløbe i praksis?

Kanban: Det gik godt med at dele opgaverne ud. Det var et godt hjælpemiddel at kunne se, hvad vi hver især arbejdede på. Det begyndte dog at gå lidt i stå med at bruge det, i slutningen af kodeprocessen fordi vi ikke fik forventningsafstemt i løbet af ugen. Michella, kom til at sidde meget med kode delen selv, hvilket betød at Kanban blev overflødigt/nedprioriteret.

Standup-meeting: Vi holdt møder hver dag, oftest om formiddagen, men vi fik ikke opdateret hinanden i løbet af dagen. Hvilket gjorde, at der blev skabt en flaskehals-tendens, fordi vi sad og ventede på hinanden. Denne ventetid skabte frustrationer, fordi vi nogle gange sad og ventede unødvendigt på hinanden. Vi kom til at lave et uheldigt system, med at skrive til hinanden for sent, så f.eks. Michella sad til over midnat, hvilket ikke var en del af vores plan.

Arbejdsform: Vi kunne ikke overholde vores aftale, om at mødes på skolen pga. sygdom og pres på hjemmefronten. Derfor blev det svært at tale med hinanden løbende og holde hinanden opdateret. Det gjorde, at vi kom til at sidde sent nogle aftener, og ikke kunne holde fri, når vi gerne ville.

Hvad gik godt og hvad kunne have været bedre?

Vores dynamik i gruppen er rigtig god. De udfordringer vi løb ind i var fordi der var udefrakommende faktorer, som gjorde, at vi ikke fik arbejdet så meget som vi gerne ville. I de dage vi kodede, gik arbejdet ikke som det skulle i forhold til vores forventninger. De dage vi har skrevet rapport, er gået mere efter vores forventninger, fordi vi har kunne overholde de aftaler, vi lavede i starten med standup-meetings mv.

Hvad har I lært af processen og hvad vil I evt. gøre anderledes næste gang?

Vi har lært, at vi skal melde klart ud, når vi ikke magter en opgave. Vi har også lært, at vi arbejder godt som gruppe, når alle er friske og frie.

Vi har lært, at vi skal lave en gruppekontrakt. Desuden skal vi også prioritere hvad der er "need to have / nice to have". I vores tilfælde blev der brugt for meget tid på Figma og css delen, hvilket kunne være nedprioriteret.

Til næste gang, vil vi være bedre til at lave deadlines og få holdt standup-meetings to gange om dagen, både om formiddagen og om eftermiddagen.

Vi vil være mere ærlige overfor hinanden, så vi bedre kan fange hvis en eller flere i gruppen er pressede.