



Olsker Cupcakes Bestillingssystem

Copenhagen Business Academy
Gruppe 12

Alissa Andrea Storm

`cph-as784@cphbusiness.dk`

Github: A-St0rM

Asger Storgaard Høffner

`cph-ah323@cphbusiness.dk`

Github: AsgerSH

Jonathan Rentoft Larsen

`cph-jl563@cphbusiness.dk`

Github: JonathanRentoft

4. april 2025

Indhold

1	Indledning	1
1.1	Om virksomheden	1
1.2	Teknologier og værktøjer	1
2	Krav	3
2.1	Firmaets mål med systemet	3
2.2	User stories	3
3	Systemmodellering	5
3.1	Aktivitets diagram	5
3.2	Domæne model	7
3.3	ER diagram	8
3.4	Navigations diagram	10
4	Styling	12
4.1	Laws of UX	12
4.2	Figma Mockups	15
5	Særlig forhold	16
5.1	Session Data	16
5.2	Håndtering af exceptions	18
5.3	Validering af brugerinput	18
5.4	Login sikkerhed	19
5.5	Database brugertyper	19
6	Implementering	20
6.1	Status på implementering	20
7	Proces	21
7.1	Github Projects	21

7.2	Roller i teamet	22
7.3	Kodestandard og Definition of Done (DoD)	23
8	Links	24

1 Indledning

1.1 Om virksomheden

Denne rapport præsenterer udviklingen af en webshopløsning til Olsker Cupcakes, et dybdeøkologisk bageri på Bornholm. Projektet blev igangsat på baggrund af en dialog med bageriets ejere, hvor behovet for en online platform blev identificeret og dokumenteret i form af user stories.

Rapporten henvender sig til en fagfælle på samme uddannelsesniveau, som ikke er bekendt med Cupcake-opgaven, og har til formål at beskrive projektets omfang, mål og leverancer. Webshoppen skal give kunder mulighed for at oprette en konto, logge ind, bestille og betale for cupcakes online. Derudover skal administratorer kunne håndtere ordrer effektivt via en brugervenlig grænseflade.

Rapporten indeholder desuden teknisk dokumentation for løsningen og er struktureret således, at den følger udviklingsprocessen fra planlægning og kravspecifikation, over implementering af user stories, til den endelige tekniske dokumentation.

1.2 Teknologier og værktøjer

Dette projekt er udviklet ved hjælp af følgende teknologier og værktøjer:

- **Java (version 17.0.14):** Det primære programmeringssprog, anvendt til udvikling af backend-logik og forretningsfunktionalitet.
- **JDBC (Java Database Connectivity, version 4.2):** Et Java API til kommunikation med relationelle databaser. Bruges her til at oprette forbindelse til og sende forespørgsler til PostgreSQL.
- **PostgreSQL (version 16.2):** Open source, relationsdatabase anvendt til lagring og håndtering af applikationens data.
- **IntelliJ IDEA (version 2024.3.5):** Et integreret udviklingsmiljø (IDE) brugt til udvikling, debugging og test af Java-applikationen.

- **Maven (version 3.8.5)**: Et build management værktøj, som bruges til at håndtere afhængigheder og bygge projektet.
- **Git (version 2.39.1)**: Et versionsstyringssystem anvendt til kildekodehåndtering og samarbejde i projektet.

Disse værktøjer og teknologier udgør fundamentet for udviklingen og vedligeholdelsen af projektet, og versionerne er specificeret for at sikre konsistens og kompatibilitet under videre udvikling.

2 Krav

2.1 Firmaets mål med systemet

Olsker Cupcakes ønsker at digitalisere deres salgsproces ved at få udviklet et simpelt, men funktionelt bestillingssystem til deres cupcakes. Firmaets primære mål er at gøre det lettere for kunder at bestille og betale cupcakes online, så de blot kan afhente dem fysisk i bageriet i Olsker. Det skal skabe en mere fleksibel og moderne kundeoplevelse, samtidig med at det frigør tid for personalet i butikken, som slipper for at tage imod ordrer manuelt.

Systemet skal ikke kun hjælpe kunder, men også give administratorer overblik over ordrer og kundedata, så de lettere kan følge op, holde styr på økonomien og rydde op i ugyldige bestillinger.

2.2 User stories

Det første kundemøde mundede ud i en række såkaldte user-stories. De beskriver på kort form hvilke brugere, som har hvilke behov og hvad de ønsker at opnå.

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.

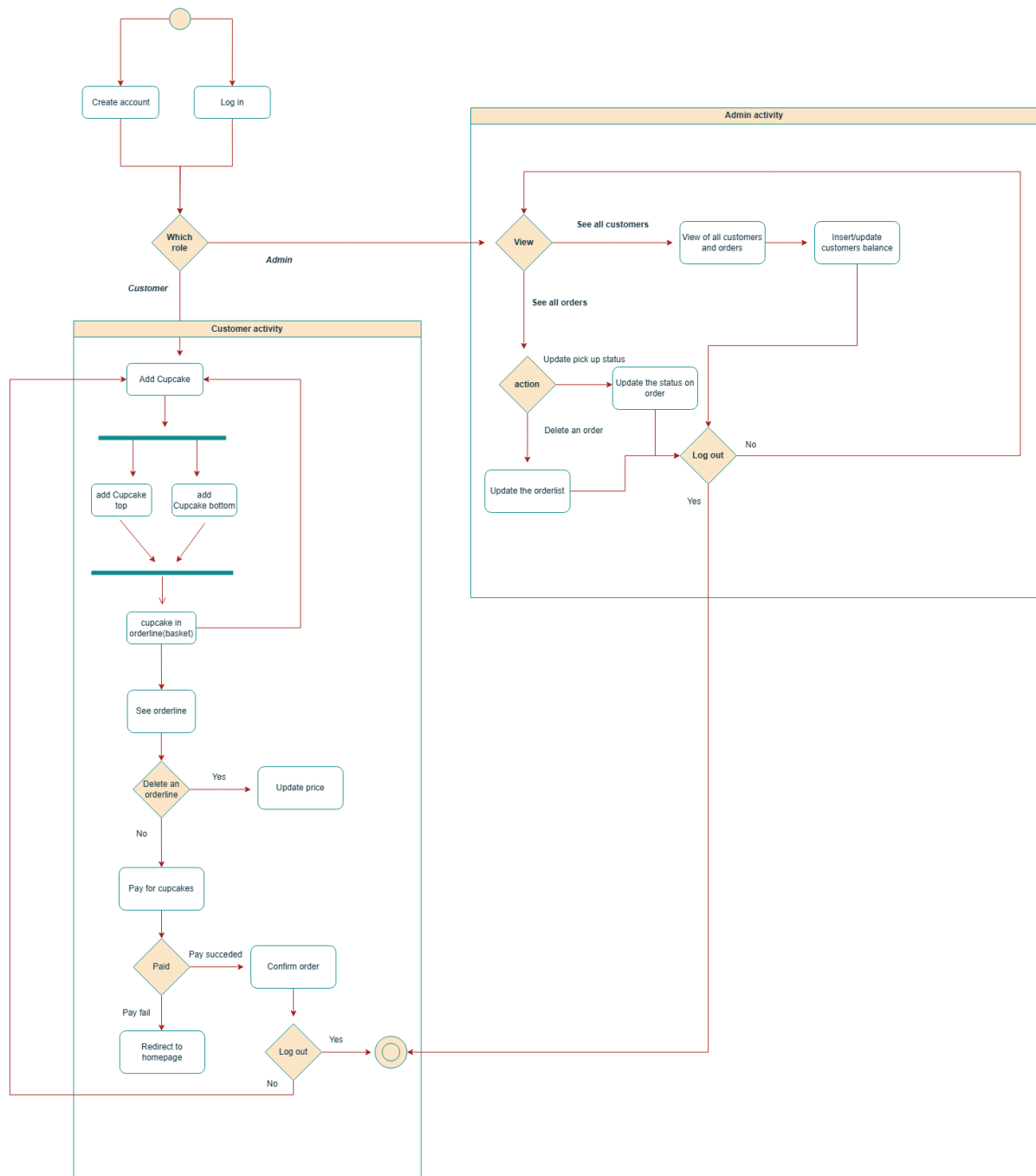
US-7: Som administrator kan jeg se alle kunder i systemet og deres ordrer, sådan at jeg kan følge op på ordrer og holde styr på mine kunder.

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.

3 Systemmodelling

3.1 Aktivitets diagram



Aktivitetsdiagrammet herunder giver et overblik over de vigtigste arbejdsgange i cupcake-systemet. Det er udarbejdet tidligt i projektforløbet for at skabe en fælles

forståelse af, hvordan brugere og administratorer interagerer med systemet.

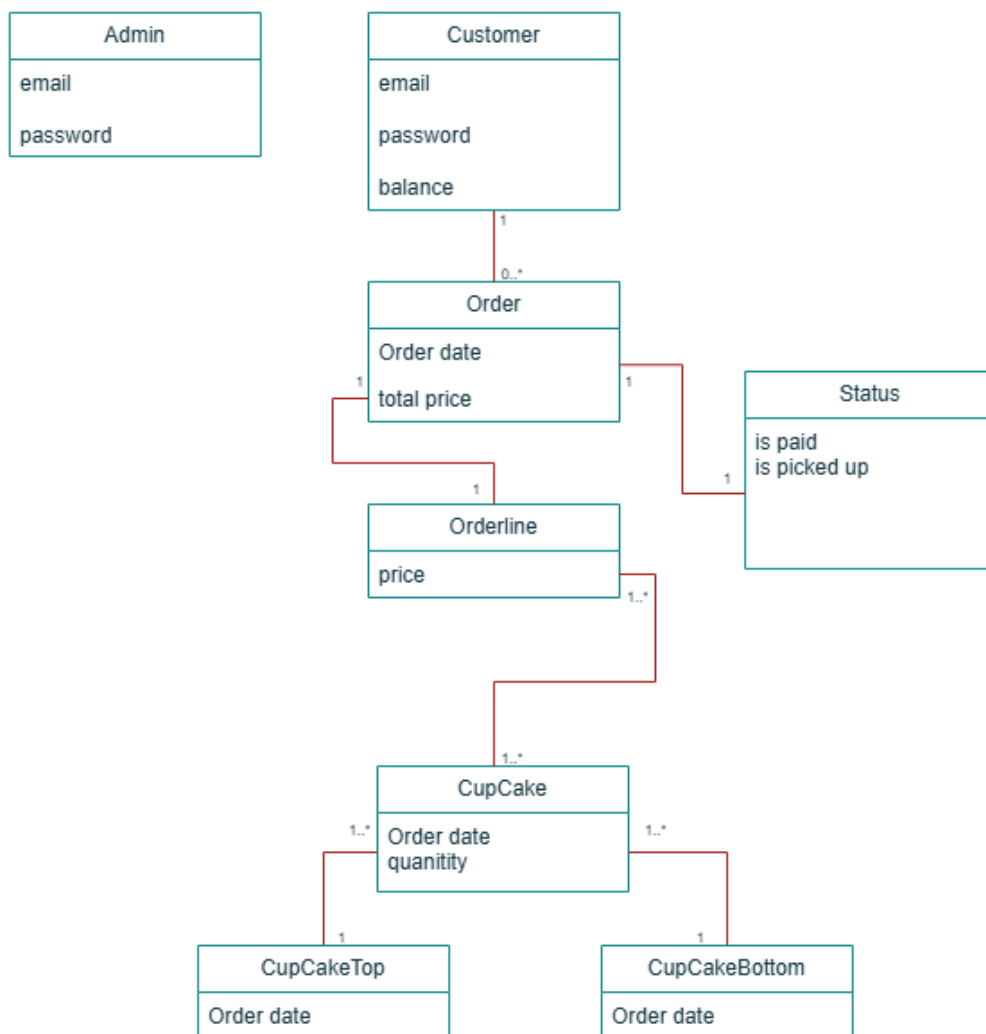
Diagrammet er opdelt i to “svømmebaner” – én for kunden og én for administratoren – som tydeligt adskiller deres roller og handlinger. Kunder kan oprette en bruger, logge ind, bygge cupcakes med valgfri top og bund, og herefter lægge dem i kurven (orderline). De har mulighed for at slette cupcakes fra kurven, inden de betaler. Når betalingen er gennemført, bliver ordren bekræftet og gemt.

På admin-siden vises funktioner som at se alle kunder og ordrer, opdatere betalingsstatus eller afhentning, samt muligheden for at slette ugyldige ordrer. Dette hænger direkte sammen med vores user stories, hvor en administrator eksempelvis skal kunne håndtere ubetalte ordrer.

Diagrammet har især været en hjælp i planlægningsfasen og fungerede som en visuel guide til både udviklingen af domænemodellen og senere funktionalitet i systemet. Det har givet os mulighed for at forstå hvordan forskellige brugere bevæger sig gennem systemet.

3.2 Domæne model

Domænenmodellen for Cupcake-systemet giver et overblik over de centrale begreber og deres relationer i systemet. Modellen repræsenterer de vigtigste entiteter, som kunder og administratorer interagerer med under bestillingsprocessen.



Systemet er centreret omkring kunder, som kan oprette sig med en e-mail, adgangskode og en saldo. Hver kunde kan foretage flere ordrer, som gemmer totalpris og bestillingsdato. Hver ordre er tilknyttet en status, der angiver, om den er betalt og/eller afhentet.

Hver ordre er koblet til præcis én orderline. En orderline repræsenterer brugerens

indkøbskurv og gør det muligt at gruppere flere cupcakes under samme bestilling. Dette designvalg blev truffet for at opfylde krav i User Story 8 og 9: henholdsvis at kunden skal kunne redigere sin kurv, og at en administrator skal kunne slette ordrer, der ikke er betalt.

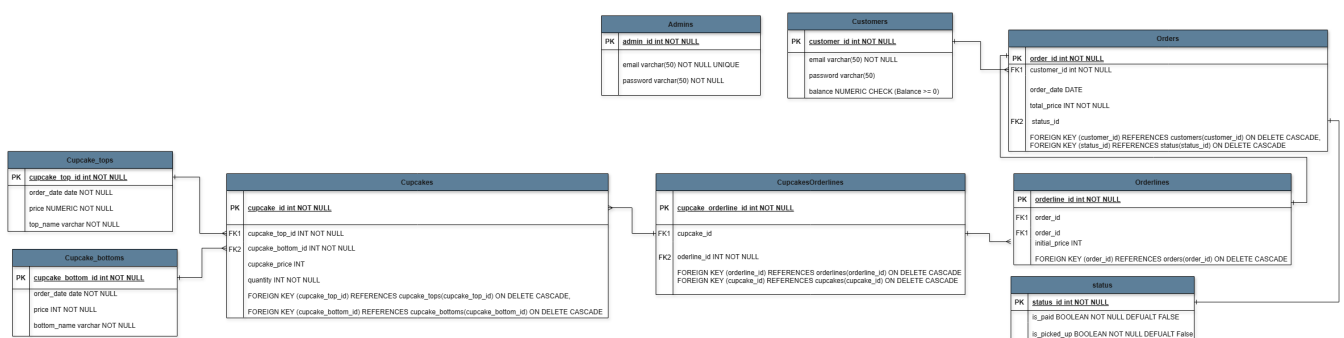
Orderline er koblet til én eller flere cupcakes. Hver cupcake består af én bund og én top, som vælges individuelt og begge har egen pris. Ved at modellere toppene og bundene som separate entiteter muliggør det dynamisk sortimentsstyring, hvor administratorer kan ændre i udvalget uden at påvirke de øvrige systemdele.

Administratoren er inkluderet i modellen med sine loginoplysninger, men har ingen direkte relation til andre entiteter, da rollen primært bruges til vedligeholdelse og kontrolfunktioner i systemet.

En central overvejelse i udviklingen af domænemodellen har været introduktionen af Orderline som en separat entitet. Orderline repræsenterer en indkøbskurv og fungerer som bindeled mellem en ordre og de enkelte cupcakes, kunden ønsker at bestille.

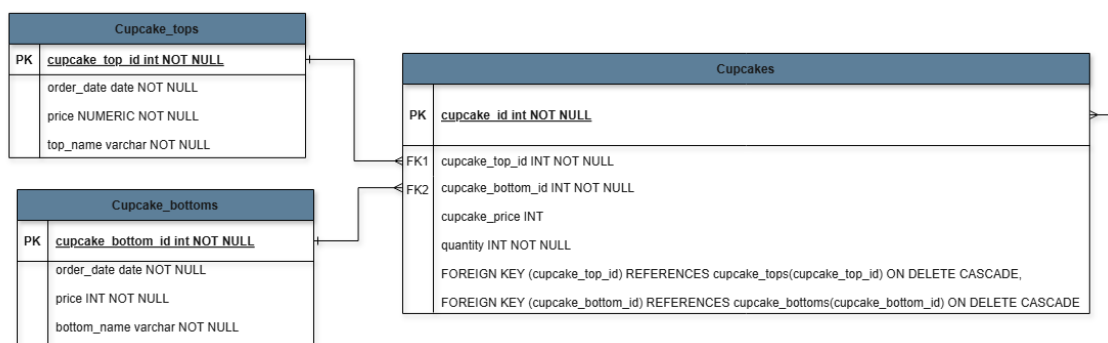
En vigtig beslutning i vores design var at oprette Orderline som sin egen del i modellen, i stedet for bare at tilknytte cupcakes direkte til en ordre. Det gør det nemmere at ændre i en bestilling, f.eks. hvis man vil fjerne en cupcake, og det gør systemet mere klart og fleksibelt, hvis man senere vil tilføje flere funktioner.

3.3 ER diagram



ER-diagrammet viser strukturen bag den relationelle database, der bruges i Cupcake-systemet. Diagrammet indeholder alle de centrale tabeller og relationer, der skal til for at håndtere både kunder, ordrer, cupcakes og administratorer.

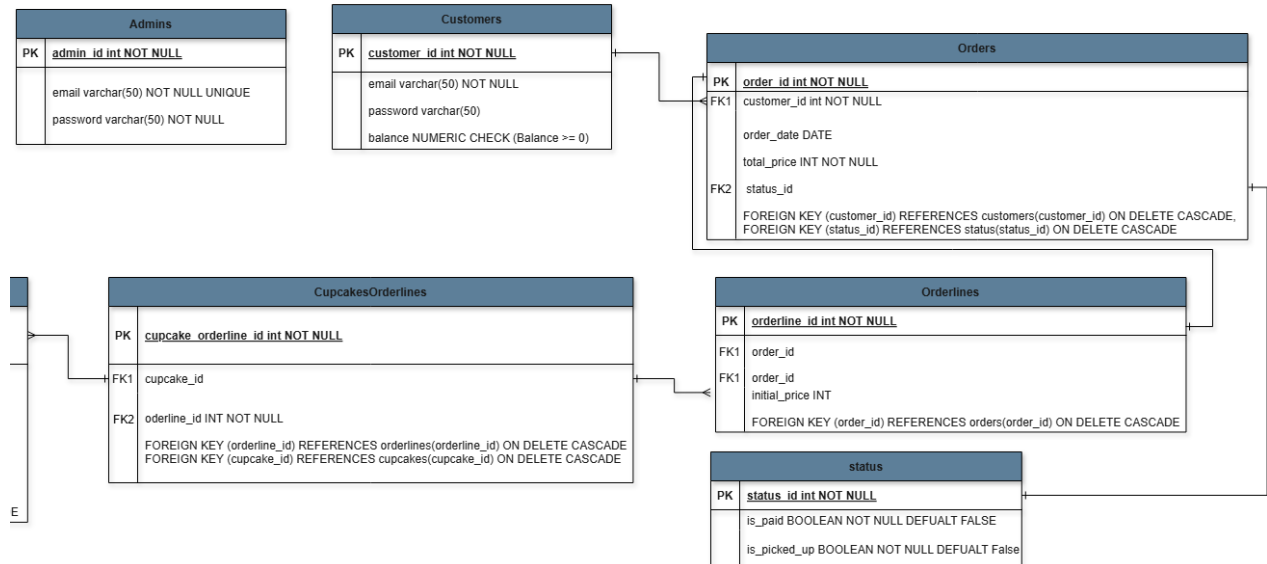
Vores database er bygget op efter 3. normalform, hvilket betyder, at vi har opdelt data i separate tabeller, så vi undgår gentagelser og holder oplysningerne rene og overskuelige. For eksempel har vi lagt cupcake-tops og cupcake-bottoms i deres egne tabeller. På den måde gemmer vi ikke den samme topping eller bund flere steder – vi henviser bare til dem med en nøgle. Det gør det nemmere at vedligeholde systemet og sikrer, at ændringer kun skal laves ét sted.



Et centralt designvalg er at adskille en ordre og de cupcakes, den indeholder, via en orderline. Dette er en 1-1 relation mellem orders og orderlines, som kan virke unødvendig, men er valgt af hensyn til forretningslogikken. Orderline fungerer som ”indkøbskurv” og gør det muligt at tilføje eller fjerne cupcakes, før ordren betales. Dette valg er tæt koblet til User story 8 og 9, hvor brugeren skal kunne justere sin kurv, og administratoren skal kunne fjerne ordrer, der aldrig er gennemført. Det har gjort systemet mere fleksibelt og realistisk.

Relationen mellem orderlines og cupcakes håndteres som en mange til mange relation, hvilket er løst med en junction tabel kaldet `cupcakesorderlines`. Det var nødvendigt, da en cupcake kan tilføjes til forskellige ordrer (f.eks. samme kombination af bund og

topping), og en orderline kan indeholde mange forskellige cupcakes.

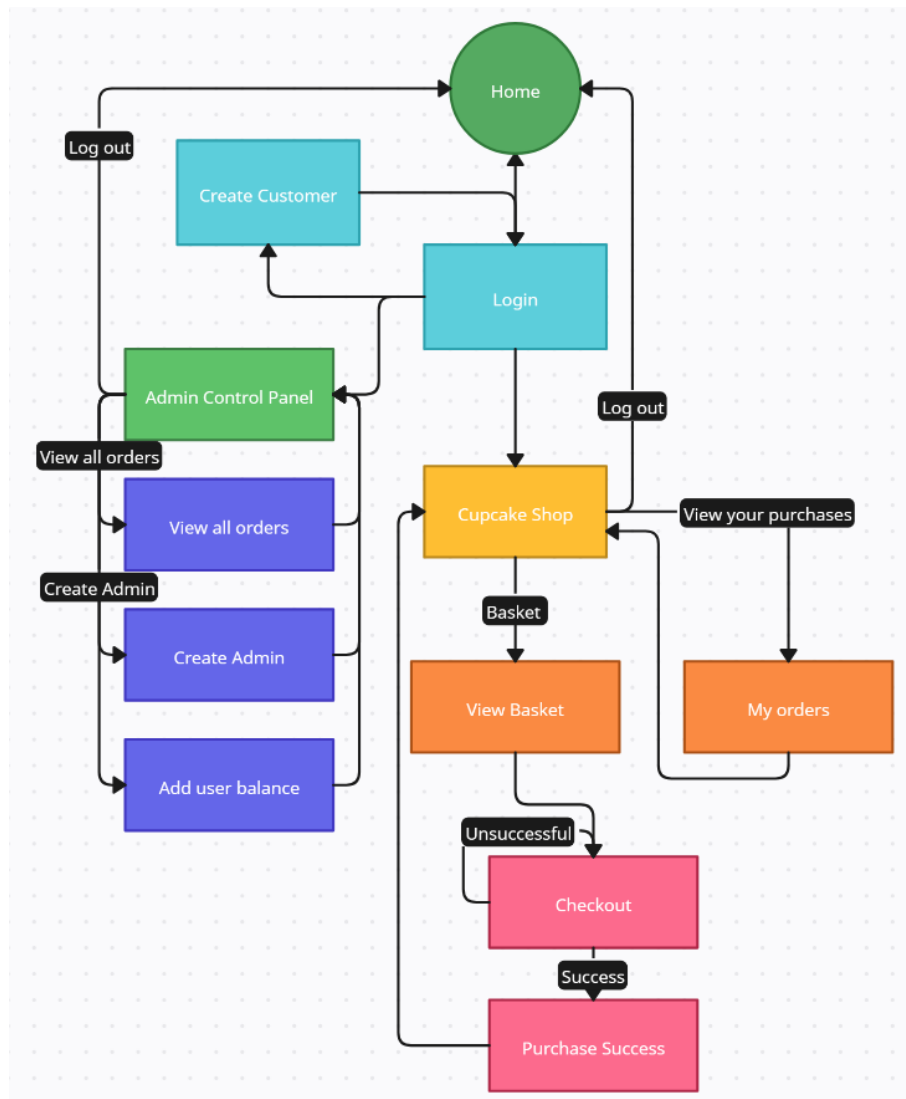


For at sikre datakonsistens er der anvendt fremmednøgler og ON DELETE CASCADE på relevante relationer. Dette betyder, at når f.eks. en kunde slettes, vil alle relaterede ordrer og tilhørende orderlines og cupcakes også blive fjernet automatisk.

Der er ingen steder i modellen, hvor alternative nøgler eller naturlige nøgler er blevet brugt i stedet for automatiske ID'er. Dette valg er truffet for at gøre integration og vedligeholdelse lettere.

3.4 Navigations diagram

For at få dannet et overblik over vores hjemmeside, og al dens funktionalitet, så har vi lavet et navigationsdiagram. Navigationsdiagrammet giver en fortælling om, hvordan systemet hænger sammen, og hvilke krav der skal opfyldes for at en kunde kan købe cupcakes.



Home er den første side en kunde/administrator møder når de går ind på hjemmesiden. Det er vores "index.html" side, og den har kun en reference til "login.html" siden. Der er en knap nederst på siden også med teksten "Make your first order!" som sender en til "login.html" - som ændres til "Make your order!" når man er tilbage på forsiden som logget ind, og som sender en til "cupcakeshop.html".

Login er en html-side som ligesåvel som resten af projektet bruger css-styling. Denne side giver brugeren muligheden for at indtaste e-mail og password til at logge ind. Hvis du logger ind med en bruger som findes i databasen, så sendes du videre til "cupcakeshop.html". Hvis man har fejl i oplysningerne, så bliver man oplyst på selve login siden

at der er gået noget galt i login. Hvis man er en administrator der gerne vil logge ind, så kan man trykke på *"Admin Login"* knappen og indtaste ens login-oplysninger for en administrator bruger. Hvis man ikke har en bruger endnu, så kan man herindefra trykke på knappen *"Create Account"*, som navigerer brugeren til vores *"createcustomer.html"* side, hvor man kan oprette en ny bruger.

Cupcake Shop er siden hvor du kan købe dine cupcakes. Her kan man vælge hvilken brødtype og topping man vil have, samt hvor mange af den sammensatte cupcake man vil bestille. Når man har besluttet sig for hvad man vil bestille, så trykker man *"add to basket"*, og så kan man enten lave flere cupcakes der skal ned i indkøbskurven, eller man kan trykke på knappen *"Basket"*, som sender dig til *"cart.html"*, hvor du kan se din ordre inden du trykker køb. Her kan man også nå at fjerne ting fra sin kurv. Når man trykker køb, sendes man til *"confirmation.html"* hvor ordren bekræftes, og så navigeres man til cupcake shoppen. Fra *Cupcake Shop* kan man også trykke på *"View your purchases"*, som navigerer til *"purchase-history.html"* som giver brugeren et overblik over tidligere ordre.

Når en administrator logger ind, så føres man hen til *Admin Control Panel*, hvor administratorer kan se alle ordre, lave nye administrator accounts og tilføje penge til kundernes konti.

4 Styling

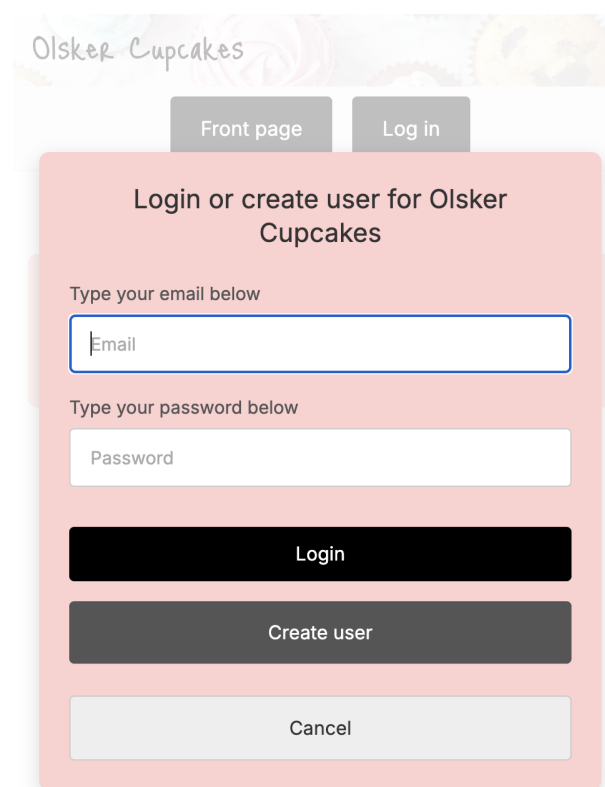
4.1 Laws of UX

Vi har brugt lidt tid på at få designet hjemmesiden, så den passer til de ønsker og krav, kunden har haft. På grund af den begrænsede tid har vi valgt at fokusere mest på funktionalitet frem for at gøre siden ekstra flot eller salgsorienteret. Hvis vi havde haft lidt mere tid, ville vi have arbejdet mere med det visuelle udtryk og gjort siden mere indbydende og "lækkert" at bruge.

Under udviklingen har vi forsøgt at implementere flere principper fra "Laws of UX" for at forbedre brugeroplevelsen og gøre interaktionen mere intuitiv.

Law of Common Region: Law of Common Region: På sider som login, opret bruger og opret admin har vi samlet relaterede elementer i visuelle bokse med skyggeeffekter. Dette giver en tydelig adskillelse fra baggrunden og skaber et klart fokusområde for brugeren.

Miller's Law: For at undgå at overbelaste brugerens arbejdshukommelse har vi struktureret indholdet vertikalt, især på sider der også skal fungere på mindre skærme. Ved hjælp af flex-direction: column vises informationer én ad gangen, så brugeren kan fokusere og navigere nemt.



The image shows a login form for 'Olsker Cupcakes'. The form is a light red box with rounded corners and a drop shadow, centered on a background that includes a header with the site name and two navigation buttons. The form itself has a title, two input fields with labels, and three buttons at the bottom.

Olsker Cupcakes

Front page Log in

Login or create user for Olsker Cupcakes

Type your email below

Email

Type your password below

Password

Login

Create user

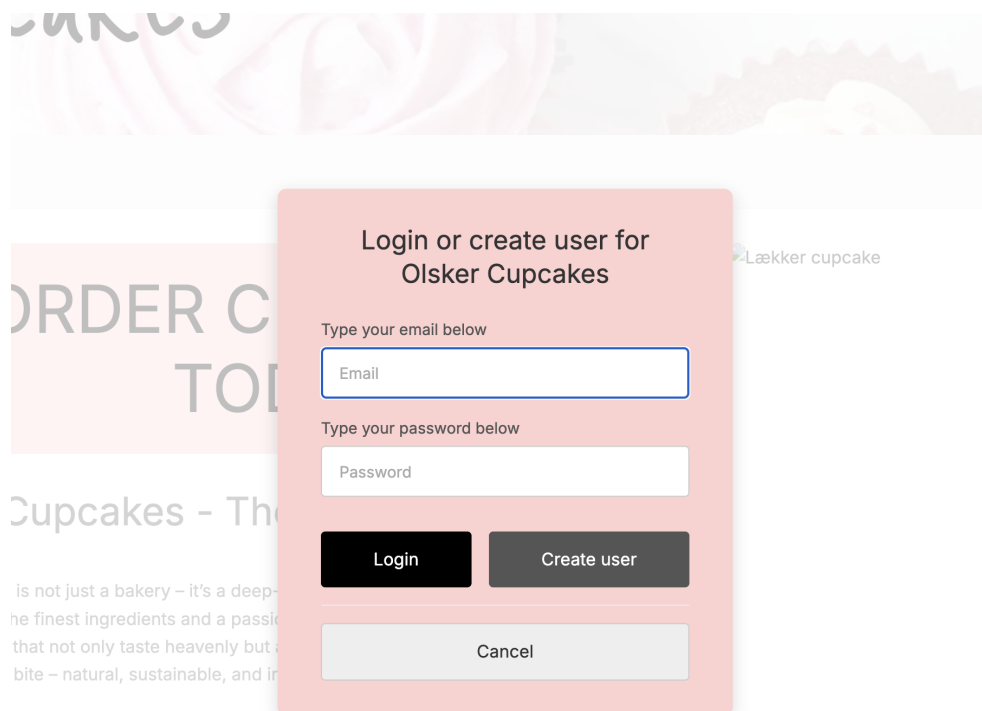
Cancel

Admin Login

Figur 1: Login form

Fitts' Law og Jakob's Lov:Knappernes størrelse og placering er gennemtænkt – de er store nok til nemt at kunne klikkes på og har tydelige farver og labels. Vi har brugt gennemgående styling til knapperne, som gør dem let genkendelige. Navigation og interaktive elementer er placeret, som brugeren forventer det – f.eks. navigationen øverst – hvilket skaber genkendelighed og mindsker kognitiv belastning.

Overlay-effekten:Når login- eller opret-bruger-boksen vises, reduceres baggrundens gennemsigtighed. Det gør det nemmere for brugeren at fokusere på den aktuelle opgave. Vi har valgt at have en “Annullér”-knap frem for at lukke boksen ved klik udenfor, da dette ville kræve JavaScript, hvilket ikke var en prioritet i dette projekt.



Figur 2: Login form

Typografi: Vi har anvendt skrifttyperne Inter og Montserrat fra Google Fonts for at skabe et moderne og læsevenligt udtryk. Derudover har vi brugt en anden skrifttype til

logoet for at få det til at skille sig visuelt ud. Der har været fokus på konsistens i brugen af font-størrelser, vægt og linjeafstand, hvilket skaber en rolig og overskuelig læseoplevelse.

Alt i alt har vi haft en intention om at gøre designet både funktionelt og brugervenligt, samtidig med at vi så vidt muligt har indarbejdet grundlæggende UX-principper.

4.2 Figma Mockups

I starten af projektet udarbejdede vi mockups i Figma for at visualisere systemets brugerflade og skabe et fælles udgangspunkt for designbeslutningerne. Ved at bruge Figma som værktøj kunne vi hurtigt få et overblik over, hvordan siden skulle struktureres, og hvordan brugerens interaktion med systemet skulle foregå.

Formålet med mockups var ikke at skabe et endeligt design, men snarere at fungere som en visuel guide for hele gruppen. Det gjorde det lettere at fordele opgaver, diskutere layout og sikre en nogenlunde ensartet stil på tværs af de forskellige sider.

Figma-mockuppet fungerede derfor som en prototype, som kunne justeres løbende, og som i sidste ende har haft stor betydning for det færdige systems udtryk og brugeroplevelse. For at se mockup i Figma, henvises der til **Links**-sektionen bagerst i rapporten.

5 Særlig forhold

5.1 Session Data

Session Attribute

I dette projekt håndterer vi sessiondata således at når en bruger er logget ind (kunde eller admin), så bliver der vist på alle sider hvilken e-mail adresse der er logget ind med.

```
public void login(@NotNull Context ctx) {  ➤ AsgerSH +1
    // Henter form parametre til login
    String email = ctx.formParam(key: "email");
    String password = ctx.formParam(key: "password");

    // Tjek om brugeren findes i databasen
    try {
        CustomerDTO customerDTO = customerMapper.login(email, password);
        ctx.sessionAttribute("currentCustomer", customerDTO);
    }
}
```

Figur 3: Login snippit

På Figur 1, kan man se, at når vi i vores *CustomerController* logger ind, så sender vi en `ctx.sessionAttribute` med. Denne session attribut har en *"inactivity timeout"* på 30 minutter, hvilket vil sige at når brugeren har været inaktiv i 30 minutter, så annulleres sessionen. Eftersom det er en `customerDTO` vi opretter, så gør vi således også at man ikke kan tjekke sessiondataen (ved at undersøge hjemmesiden), og se kodeordet for brugeren nogen steder.

På Figur 2 kan vi samtidig se, at når vi router `/logout`, hvilket vores *"Log out"*-knapper gør, så af sikkerhedsmæssige årsager sætter vi attributten til at være *"null"*, og derefter invaliderer sessionen. Dette gør vi, i tilfælde af at `ctx.req().getSession` af en eller anden årsag ikke kan finde sessionen, så i stedet for at gøre ingenting, så bliver attributten altid sat til `null` først.

```
// General Routing
app.get(path: "/", Context ctx -> ctx.render(filePath: "index.html"));
app.get(path: "/logout", Context ctx -> {
    ctx.sessionAttribute("currentCustomer", null);
    ctx.sessionAttribute("currentAdmin", null);
    ctx.req().getSession().invalidate();
    ctx.redirect(location: "/");
});
```

Figur 4: Logout snippit

For at brugeren altid kan se hvad email de har logget ind med, så har vi i navigationsbarren på alle sider lavet et thymeleaf if-statement, der henter session.currentCustomer / Admin, og hvis den ikke er null, så vises emailen for brugeren. Dette kan ses i nedenstående Figur 3.

```
<li id="activeuser" th:if="${session.currentCustomer != null}">
    <span th:text="${session.currentCustomer.email}"></span>
```

Figur 5: Email snippit

Ydermere, så bruger vi også sessiondata til at gøre brugeroplevelsen lidt bedre, i forhold til at man ikke skal kunne se en login-knap selvom man er logget ind. Her bruger vi igen et thymeleaf if-statement der tjekker om attributten som kommer med i log in findes eller ej. Hvis den er null, så vises "login" knappen, hvis der er data i, så vises "log out" knappen. Dette ses på Figur 4.

```
<li id="loginbutton" th:if="${session.currentCustomer == null}">
    <a href="login.html" th:href="@{/login}">Log in</a>
</li>
<li id="logoutbutton" th:if="${session.currentCustomer != null}">
    <a href="index.html" th:href="@{/logout}">Logout</a>
```

Figur 6: Button snippit

5.2 Håndtering af exceptions

Eftersom vores system har bl.a. kontakt til en SQL-database, så indeholder systemet mulighed for Exceptions. I vores system bruger alle vores Mapper-metoder en try/catch. Vi kaster og griber SQLException og DatabaseException, hvori vores håndtering er ved at de sender en besked ud, som vi selv vælger hvad skal være. I vores egen oprettede DatabaseException har vi gjort således at den sender en besked retur enten som *userMessage* eller *systemMessage*.

5.3 Validering af brugerinput

I vores system har vi taget forbehold for, at når en kunde (eller admin) opretter en konto, så kan de ikke få lov til at lave kontoen uden at opfylde de krav vi har sat for kontooprettelsen. Vi har først og fremmest en front-end løsning der kræver at når en konto bliver oprettet, så skal koden opfylde 3 krav: *Koden skal indeholde 1 tal, have en minimum længde på 4 og mindst 1 stort bogstav*. Udover dét, så skal man også gentage kodeordet før man får lov at oprette brugeren, så man sikrer sig at det er den korrekte kode der bliver oprettet.

Når kunder eller admin vil logge ind, så tager vi brug af et SQL statement (ses i Figur 5), som sammenligner email i database-tabellen i lowercase, med vores prepared statement som også konverteres til lowercase. Dette gør at når brugeren eksempelvis indtaster en email som har en blanding af store og små bogstaver, så kan vi stadig sammenligne dem korrekt.

```
String sql = "SELECT * FROM customers WHERE LOWER(email) = LOWER(?) AND password = ?";
```

Figur 7: SQL snippit

5.4 Login sikkerhed

Login-systemet benytter session-baseret autentificering, hvor brugeren bliver identificeret gennem en aktiv session efter korrekt login. Brugere (både kunder og admins) logger ind med email og kodeord, og deres session lagres for at kunne give adgang til relevante funktioner i systemet.

Dog er sikkerheden begrænset. Kodeord gemmes som klartekst i databasen, hvilket ikke er sikkert. I et produktionssystem burde kodeord hashes for at beskytte mod datalæk.

Systemet har basal login-funktionalitet og sessionhåndtering, men ingen egentlig sikkerhedsforanstaltninger som man ville forvente i en professionel applikation. Det er dog tilstrækkeligt for et læringsprojekt og et lukket miljø.

5.5 Database brugertyper

Systemet benytter to forskellige brugertyper: kunde (customer) og administrator (admin). Disse er implementeret som separate tabeller i databasen og håndteres forskelligt i systemets logik.

Ved login gemmes den aktive bruger i en session, afhængigt af typen:

- En kunde gemmes i `sessionAttribute("currentCustomer")`
- En administrator gemmes i `sessionAttribute("currentAdmin")`

Dette gør det muligt at skelne mellem brugerroller og sikre, at kun administratorer får adgang til administrative funktioner såsom visning og sletning af ordrer samt indsættelse af saldo på kundekonti.

Formålet med brugertyperne er at sikre adgangskontrol og adskille funktionalitet afhængigt af rollen. Dette giver en mere sikker og struktureret applikation, hvor rettigheder er tydeligt opdelt.

6 Implementering

6.1 Status på implementering

Projektet er i sin helhed blevet implementeret med de vigtigste funktionelle krav på plads. Systemet understøtter login, bestilling af cupcakes, køb, visning af tidligere ordrer og administration af kunders saldo.

Der er dog enkelte områder, vi ikke har nået at færdiggøre:

Vi har ikke lavet JUnit-tests eller integrationstests. Det betyder, at funktionaliteten kun er testet manuelt, og vi har derfor ikke automatisk validering af korrekthed og robusthed.

Vi har ikke nået at finpudse design og styling af hjemmesiden. Selvom siderne fungerer og har grundlæggende layout, kunne det visuelle udtryk være mere ensartet og professionelt.

Vi har ikke implementeret ON DELETE CASCADE på status-tabellen, hvilket betyder, at man manuelt skal slette statusrækker, hvis man fjerner en tilhørende ordre.

Hjemmesiden er ikke blevet tilpasset til mobilvisning (fx iPhone). Det betyder, at layoutet ikke nødvendigvis fungerer optimalt på mindre skærme.

Disse punkter er nogle vi ville arbejde videre med, hvis projektet skulle videreudvikles eller i en fremtidig iteration.

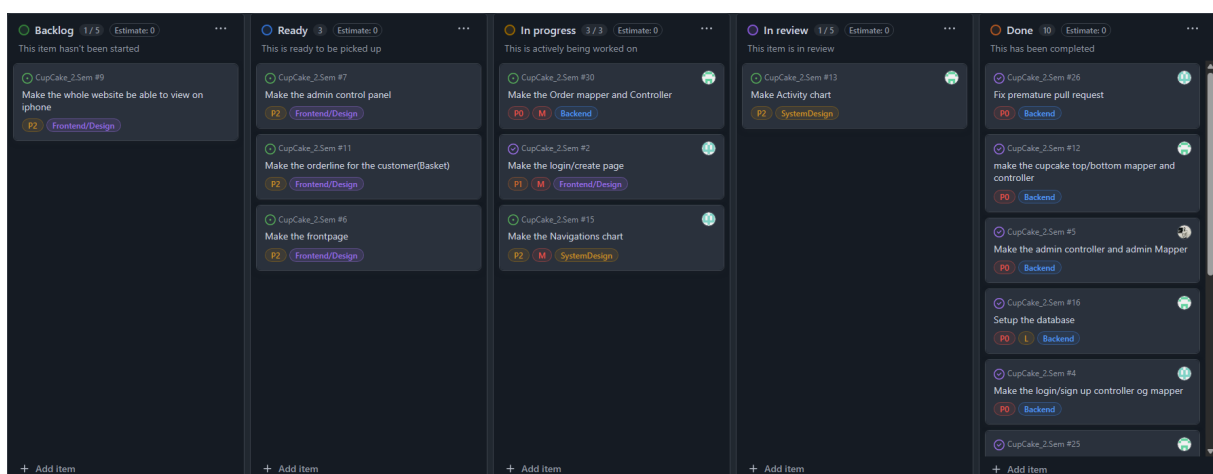
7 Proces

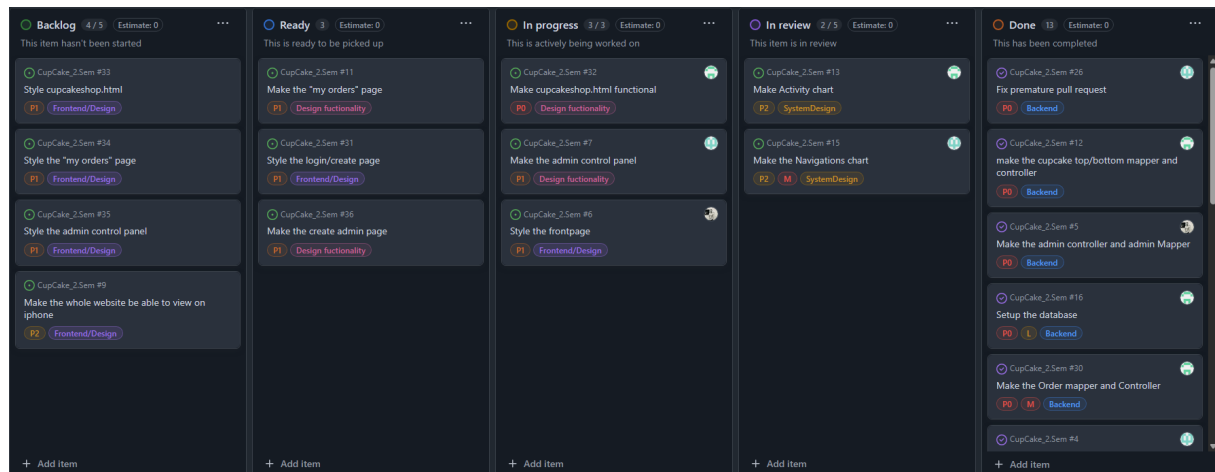
Projektet på de her 2 uger har forløbet sig ret dynamisk. Vi har skullet vænne os til hinandens arbejdsstrukturer og håndtere hvordan vi arbejdede på trods af opstået sygdom. Vi har fået det til at fungere ud fra de værktøjer vi havde til rådighed og formåede at lave aftaler for hvor og hvornår vi mødtes til gennemgang af de issues vi havde udarbejdet. Til en anden gang ville det helt klart være fordelagtigt at lave en møde-struktur fra starten af, så man planlagde helt konkret hvilke dage vi skulle møde og hvornår. En af de ting vi gjorde godt, var at vi regnede baglæns i forhold til hvornår visse dele af projektet skulle være færdigt. Eksempelvis lavede vi kodeløb, så vi havde 2 dage til at skrive rapport i.

7.1 Github Projects

I dette projekt har vi taget brug af et af Githubs værktøjer, *Github Projects*. Dette er en kanban board, som har givet os mulighed for at få et let, håndgribeligt billede af hvordan vi ville arbejde med udførelsen af projektet. Vi startede med at oprette en bunke issues, som vi kategoriserede ud fra hvor vigtige de var, og hvad for en type de er. P0 betød højest prioritet, og så kunne type være eksempelvis *"Backend"*.

Herfra udarbejdede vi små regler for dette kanban board, eksempelvis aftalte vi en *WIP (Work In Progress) limit*, hvor vi blev enige om at man maksimalt må have 2 personlige issues ad gangen i progress.





7.2 Roller i teamet

En af opgaverne i dette projekt var, at dele ansvarsområder ud. Eftersom dette har været et relativt kort projekt, så besluttede vi at vi ikke havde nogen faste roller. Vi valgte derimod at skiftevis være Product Owner, Tech Lead og Scrum Master.

Primært da vi brainstormede opgaven, med hvilke issues vi skulle tilføje til kanban-boardet, der tog vi skiftevis product owner hatten på, i forhold til *"har Olsker Cupcakes virkelig brug for denne funktion"*?. Denne brug har været god til at mindske splid mellem vores gruppemedlemmer, eftersom man hurtigt kan blive meget ambitiøs med funktionalitet, og det er en god måde at skabe en dialog om relevans.

Ud fra alle vores issues i Github Projects har vi oprettet branches til at arbejde i. Når disse branches er lavet færdigt, så har vi arbejdet med at lave Pull Requests. For hver branch der skulle merges, har der været et pull request som vi skiftevis har gennemgået. Vi har alle ageret som reviewer på disse pull requests, og sørget for at kodestandarder er overholdt, så vi har skiftevis alle haft Tech lead rollen.

Scrum master rollen har vi tilgængæld ikke gjort meget brug af, eftersom vi ikke har afholdt stand-up møder. Vi har løbende når vi har aftalt at mødes (enten virtuelt eller

fysisk) kommet med en status osv. Det vil altså sige at de ting scrum masteren har til opgave at udføre, har vi mere eller mindre i fællesskab udført sammen. Det vil sige at alle er blevet hørt ifht. løsningsforslag, fælles klargøring til vejledning, fælles plan for ugen og sygdom meldt til gruppen samlet.

7.3 Kodestandard og Definition of Done (DoD)

I teamet har vi arbejdet ud fra nogle fælles kodestandarder for at sikre, at koden er ensartet og nem at arbejde videre med – både for os selv og hinanden. Vi har valgt at strukturere vores projekt efter MVC-designmønstret (Model, View, Controller), hvor vi har adskilt vores forretningslogik, datatilgang og præsentation.

Vi bruger Java 17 som SDK, og al javakode er skrevet i camelCase. Routing er samlet ét sted i RoutingController-klassen, og må ikke ligge direkte i Main. HTML-sider har deres egen tilknyttede CSS-fil, så design og struktur er overskueligt og nemt at tilpasse.

Kommentarer til metoder er tilføjet, hvor det giver mening – især hvis metoden er mere avanceret end et simpelt getter/setter-kald. Vi har desuden organiseret statiske filer som CSS, billeder og (hvis nødvendigt) JavaScript i public/-mappen.

Definition of Done (DoD) Vi har i gruppen defineret en fælles forståelse af, hvornår en opgave er ”færdig”, for at sikre kvalitet og undgå misforståelser. En feature er først ”done”, når følgende er opfyldt:

- Kodekvalitet: Koden er færdig og følger vores kodestandarder.
- Review: En anden har gennemgået koden, og evt. kommentarer er blevet behandlet.
- Testing: Unit tests og integrationstests er skrevet og passerer (vi har dog ikke nået at implementere disse).
- Dokumentation: Relevant dokumentation er opdateret.

- Deployment: Koden er merged til main og fungerer med resten af systemet.
- Godkendelse: Featuren er testet og godkendt af Product Owner (os selv).

8 Links

[Klik her for at se video-demo på YouTube](#)

[Klik her for at se Github-repo](#)

[Klik her for at se Figma-Mockup](#)