Link til repository:

https://github.com/mrPrimeBeef/cupcake

Link til video demo

https://youtu.be/DaZ4HdDT5a8

Navn	cph-email	Github-navn
Agnethe Christensen	cph-ac331@cphbusiness.dk	Agnethechr
Rolf Lundsgaard Josephsen	cph-rj17@cphbusiness.dk	MrPrimeBeef
Peter Bollhorn	cph-pb274@cphbusiness.dk	pbollhorn
Malou Vich Lavrentiew	cph-ml786@cphbusiness.dk	malouVich

Indholdsfortegnelse

Indledning

Denne rapport dokumenterer udviklingen af en webapp, der er designet til at imødekomme behovene hos en virksomhed inden for salg af cupcakes. Formålet med projektet er at skabe et brugervenligt system, hvor kunderne får mulighed for at bestille personligt tilpassede cupcakes. Dette system gør det muligt for kunderne at vælge mellem forskellige bundtyper og toppings samt at angive det ønskede antal af hver specifik cupcake. Derudover inkluderer systemet en login-funktion, der giver både almindelige brugere mulighed for at købe cupcakes og administratorer adgang til at se og administrere alle ordrer. Denne tilpasningsevne er ikke blot en funktion, men en central del af brugeroplevelsen, der sigter mod at gøre bestillingsprocessen enkel.

Systemet er designet som en full stack-webapplikation med adskilte frontend- og backend-komponenter. Frontenden er bygget med HTML, CSS og Thymeleaf for at skabe en dynamisk og brugervenlig grænseflade, mens backend-logikken er udviklet i Java. Databasen er baseret på PostgreSQL, hvilket muliggør sikker og effektiv lagring af kunde- og ordredata. For at sikre fleksibilitet og konsistens i udviklingsmiljøet anvendes Docker Desktop, som kører hele applikationen, i containere, hvilket gør systemet nemt at teste og udrulle på tværs af forskellige platforme.

Web Appen er udviklet med en sådan tilgang, at der kan være flere brugere, sådan at systemet kan tilgås af både almindelige kunder og administratorer. Almindelige brugere kan logge ind, se deres tidligere ordrer og oprette nye bestillinger, mens administratorer har adgang til en særlig administrationsside. Her kan man se alle kunders bestillinger samt administrere ordrer

Rapporten er målrettet datamatikerstuderende på 2. semester, som har grundlæggende kendskab til de anvendte teknologier, men som ikke nødvendigvis har indsigt i den specifikke cupcake-opgave. Rapporten sigter mod at præsentere projektet på en måde, som gør det let for fagfæller at forstå både det tekniske design og forretningsstrategien bag systemet.

Baggrund

Virksomheden, Olsker Cupcakes, som dette projekt udvikles for, er en lille, specialiseret bager beliggende på Bornholm. Bageriet har etableret sig som en aktør med en dybdeøkologisk tilgang og er kendt for deres unikke opskrifter og kvalitets ingredienser. Deres mål er at kombinere håndværk og bæredygtighed i hver cupcake, hvilket afspejles i deres passion for at skabe disse produkter.

Olsker Cupcakes ønsker at udvide deres forretning ved at introducere en fleksibel bestillingsmulighed, som imødekommer den stigende efterspørgsel fra kunderne efter personlige og skræddersyede produkter.

Denne fleksibilitet vil ikke blot differentiere deres tilbud fra mere traditionelle bestillingssystemer, men også give kunderne en følelse af ejerskab og engagement i deres køb. Systemets design og implementering vil derfor fokusere på at skabe en intuitiv brugergrænseflade, der understøtter både kunder og administratorer i deres respektive roller. Dette projekt er derfor ikke blot en teknisk udfordring, men også en mulighed for at bidrage til Olsker Cupcakes' forretningsstrategi ved at forbedre kundetilfredsheden og styrke deres position på markedet.

Teknologivalg

Dette projekt er et full stack-webapplikation, som er udviklet med både backend- og frontend-teknologier for at sikre en komplet og sammenhængende brugeroplevelse.

Backend-delen er bygget i Java (version 17) og bruger PostgreSQL (version 42.7.2) til lagring af data. PostgreSQL kører i en container i Docker Desktop (version 4.34.1). Databaseforbindelsen håndteres effektivt med HikariCP (version 5.1.0), og applikationens webserver er baseret på Javalin (version 6.1.3), som leverer en letvægtsstruktur til API'er og routing.

Frontenden er designet med HTML og CSS, og Thymeleaf (version 3.1.2) bruges til server-side rendering, som genererer dynamiske sider på serveren

Til at teste systemet anvendes JUnit (version 5.10.2). Projektet er udviklet i IntelliJ IDEA (version 2023.3.5), hvilket gør det nemt at videreudvikle og vedligeholde systemet.

Krav

Firmaets krav til systemet er at det skal kunne håndtere komplekse bestillinger, hvor brugerne har frihed til at vælge både bund og topping samt det ønskede antal af hver cupcake. Systemet skal også tilbyde en administrativ funktionalitet, der giver administratorer mulighed for at se og administrere alle ordrer, hvilket sikrer en effektiv drift og kundeservice.

User Stories

For at opnå denne vision er der identificeret en række user stories, som fungerer som vejledende krav til systemet. Disse user stories er formuleret således, at de adresserer centrale funktioner og brugerbehov.

Der er følgende 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.

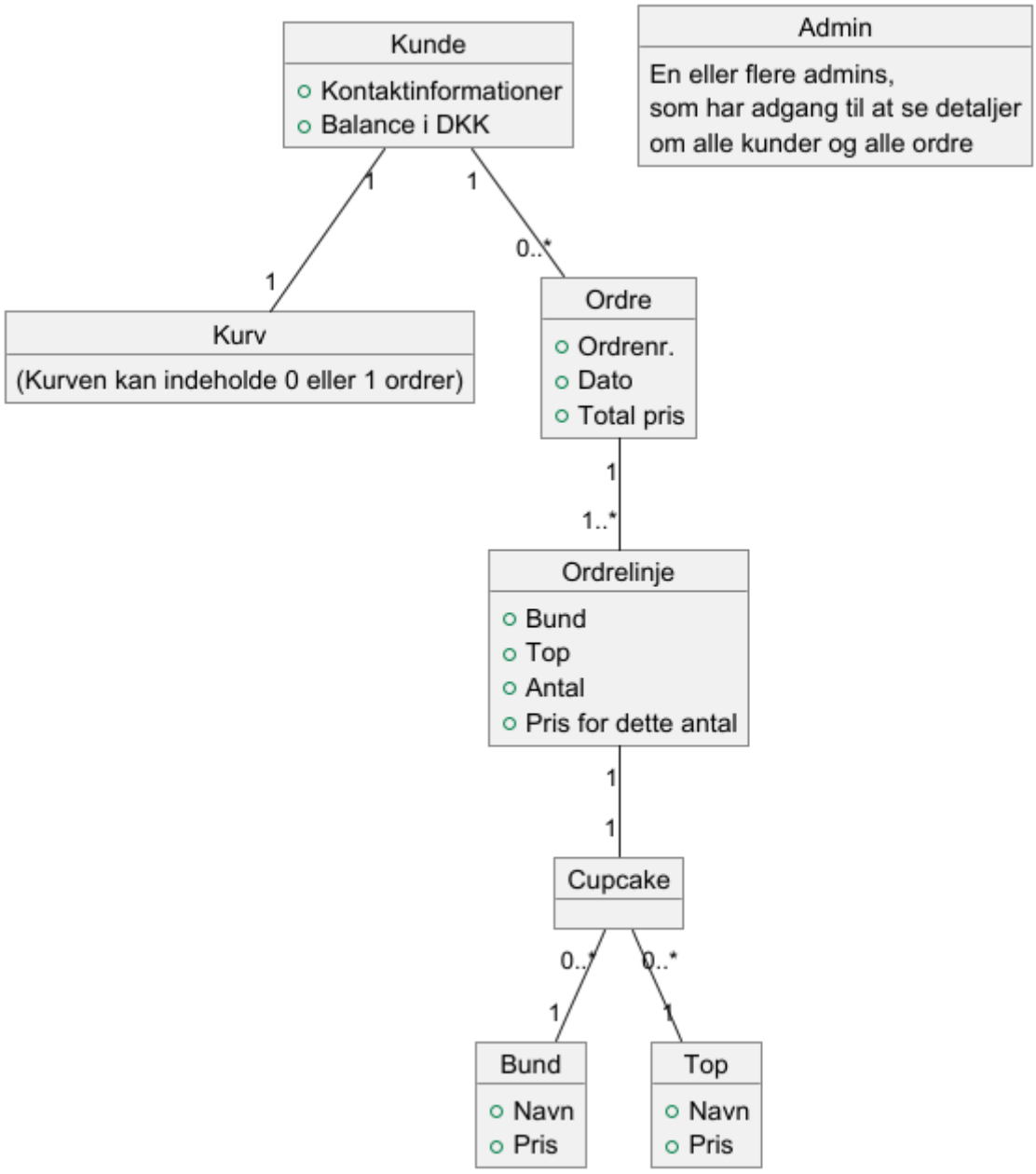
US-8: Som kunde kan jeg fjerne en ordrelinie fra min indkøbskurv, så jeg kan justere min ordre.

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.

Domæne model og ER diagram

Domænemodel

For at være sikker på at vi har forstået Olsker Cupcakes behov og ønsker til webappen, har vi holdt et møde med dem, hvor vi i samarbejde har udviklet en domænemodel. For det er vigtigt at vi som softwareudviklere udvikler en webapp, som rent faktisk løser kundens behov, og ikke blot hvad vi tror behovet er. Vores domænemodel er vist på figur 1.



Figur 1 – Domæne model.

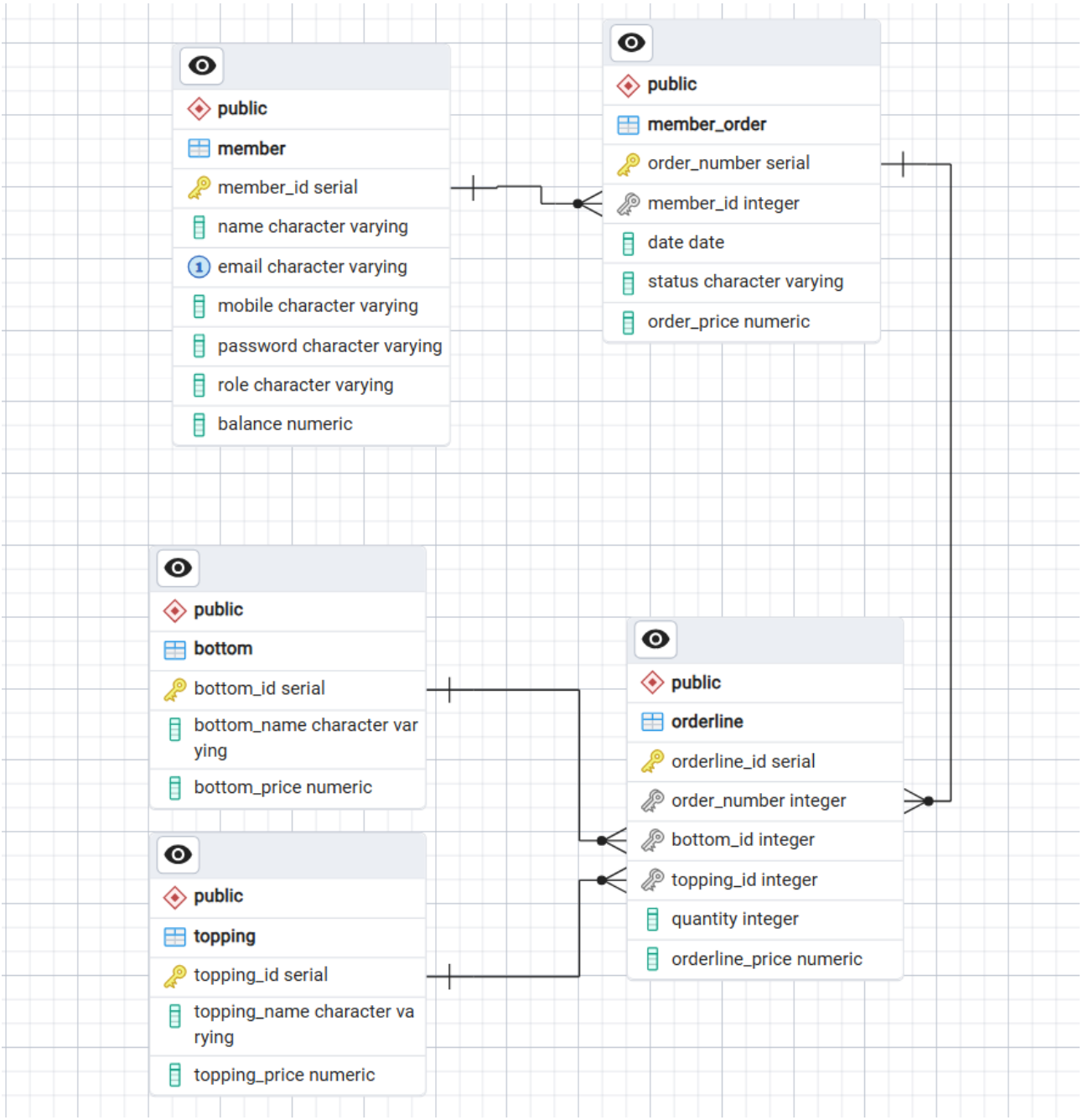
Domænemodellen er udarbejdet så den afspejler Olsker Cupcakes forretning:

- Hos Olsker Cupcakes kan én kunde have nul til mange ordrer. Grunden til at en kunde kan have nul ordre, er at det er blevet valgt at kunden skal oprette sig som bruger, før de kan få lov til at bestille deres første ordre.
- Én ordre kan have én til mange ordrelinjer. Det vil ikke give mening at have en ordre med nul ordrelinjer.
- Én ordrelinje indeholder én slags cupcake (som kunden selv har sammensat ved frit at vælge den slags bund og top de ønsker).
- De forskellige slags bunde og toppe har hver deres navn og hver deres pris. Prisen for en cupcake er summen af pris for bund og top. Én slags bund kan indgå i nul til mange cupcakes. Og tilsvarende kan én slags top indgå i nul til mange cupcakes.

Her skal bemærkes at hver enkelt kunde, ordre og ordrelinje er helt unik. Hvad bund og top angår, er der derimod tale om, at hver slags bund og top er unik. Den enkelte fysiske bund og top er ikke unik, og man kan derfor holde styr på dem ved at tælle deres antal.

ER diagram

Efter mødet med Olsker Cupcakes, gik vi i gang med at udvikle et database design med udgangspunkt i domænemodellen. Vores database design er vist på figur 2.



Figur 2 – ER Diagram.

Navngivning af tabeller

"user" og "order" er reserverede ord i PostgreSQL. For at undgå konflikt har vi derfor navngivet vores brugertabel som **member** og vores ordretabel som **member_order**.

Sammenligning med domænemodellen

Entiteterne i domænemodellen er oprettet som tabeller i vores database. Dog er der tre undtagelser:

- Vores database har ikke nogen "cupcake" tabel. Istedet har vi modelleret cupcakes i **orderline** tabellen, hvor de bliver sammensat af bund og top.
- Vores database har heller ikke nogen "kurv" tabel. I vores database er kurven modelleret ved at **member_order** har attributten "status" som kan være "Completed" eller "In progress". Hvis status er "In progress" betyder det at ordren er i kurven. Der kan derfor kun være én ordre som har status "In progress".
- Endeligt har vores database ikke nogen "admin" tabel. Istedet har vores **member** tabel attributten "role", som kan være "customer" eller "admin"

Afvigelser fra 3. normal form

Vores database overholder 3. normal form bortset fra hvad angår pris. Vi har pris med i både **bottom**, **topping**, **orderline** og **member_order**. Hvis vi skulle have overholdt 3. normal form fuldt ud, skulle vi kun have haft pris med i **bottom** og **topping**.

Der er en god grund til at vi også har valgt at have pris med i **orderline**: Olsker Cupcakes ønsker løbende at kunne justere deres priser på bunde og toppe pga. inflation og andre forretningsmæssige årsager. Fordi vi har pris med i **orderline** forbliver priserne i ordrehistorikken til den pris de er solgt til, uanset hvordan priserne på bunde og toppe fremadrettet bliver justeret.

Desværre er der ikke nogen god grund til at vi også har taget pris med i **member_order**: Fordi vi har valgt at gøre det, kræver det nemlig at vores pris i **member_order** altid er synkroniseret med summen af de tilhørende priser i **orderline**. Vi har formået at få denne synkronisering til at fungere. Men i bagklogskabens lys havde det været et bedre design ikke at have pris med i **member_order**, og istedet beregne prisen ud fra **orderline** hver gang det er nødvendigt. For denne metode er simplere at implementere, og mindre skrøbelig, for der er ikke nogen synkronisering, som potentielt kan gå galt.

Primærnøgler

Vi bruger et automatisk genereret ID som primærnøgle i alle vores tabeller. Navnet på vores primærnøgler er lavet ved at tage tabelnavnet og så sætte "_id" bag på. En undtagelse herfra er **member_order** tabellen hvor vi har valgt at kalde primærnøglen for "order_number". Det skyldes at vi fra starten planlagde at tage denne primærnøgle med frem i frontend og vise den til brugeren som et "ordrenr." Imidlertid besluttede vi senere hen også at tage "member_id" med frem i frontend og vise det som et "kundenr.". Noget af systematikken i vores navngivning er dermed brudt.

Fremmednøgler

Relationerne imellem vores tabeller har vi etableret vha. fremmednøgler. F.eks. har **orderline** fremmednøgler til både **bottom**, **topping** og **member_order**. Alle vores fremmednøgler er deklareret eksplicit i PostgreSQL således at de får foreign key constraints på. Foreign key constraints er smart, for det betyder, at når vores Java backend forsøger at indsætte en række i f.eks. **orderline** tabellen, så giver PostgreSQL kun lov til det hvis nøglerne til de fremmede tabeller rent faktisk eksisterer som primærnøgler i disse tabeller.

Andre constraints

Vi har brugt en *unique* constraint på email attributten i **member** tabellen. For vi har valgt at man i vores webapp skal logge på med email og password. Og dermed er det vigtigt at email unikt definerer en bruger.

Navigationsdiagram

I vores Cupcake webapp har vi gjort brug af tre forskellige header fragments og en fælles footer fragment for alle html sider. Herunder har vi to forskellige navigations bar, en i hver header fragtment, en til brugeren som i dette projekt er en kunde, som får vist en "bestil cupcakes", "mine ordre", "log ud", kundens email og et ikon af en indkøbskurv. Kundens navigations bar kan ses på alle html sider lige så snart kunden er logget ind.
Vores anden navigation bar er til vores admin bruger, som får vist "ordre", "kunder", "log ud" og admins email. Admins navigations bar kan ses på alle html sider lige så snart admin er logget ind.

Grunden til at vi har valgt at have to forskellige navigations bar, er da der er forskel på hvad vores kunde og admin har brug for at se.
Hertil har vi valgt at have en fælles footer på alle html siderne, da vi ikke har ment at det er nødvendigt at den viser forskellige elementer om, man er logget ind som kunde eller admin. I footeren viser vi Olsker Cupcakes adresse, CVR nummer, telefonnummer og åbningstider. På den har vi lavet en fake adresse, CVR nummer, telefonnummer og åbningstider.

Den sidste header fragment vi har valgt at lave i projektet er en header uden navigations bar til de html sider hvor brugeren endnu ikke er logget ind eller har oprettet sig som bruger endnu. Hertil har vi to error html sider hvor vi ikke har valgt at have nogle navigations bar, da de bliver henvist til på forskellige tidspunkter i programmet.

Figur 3 - Kundes navigations bar.

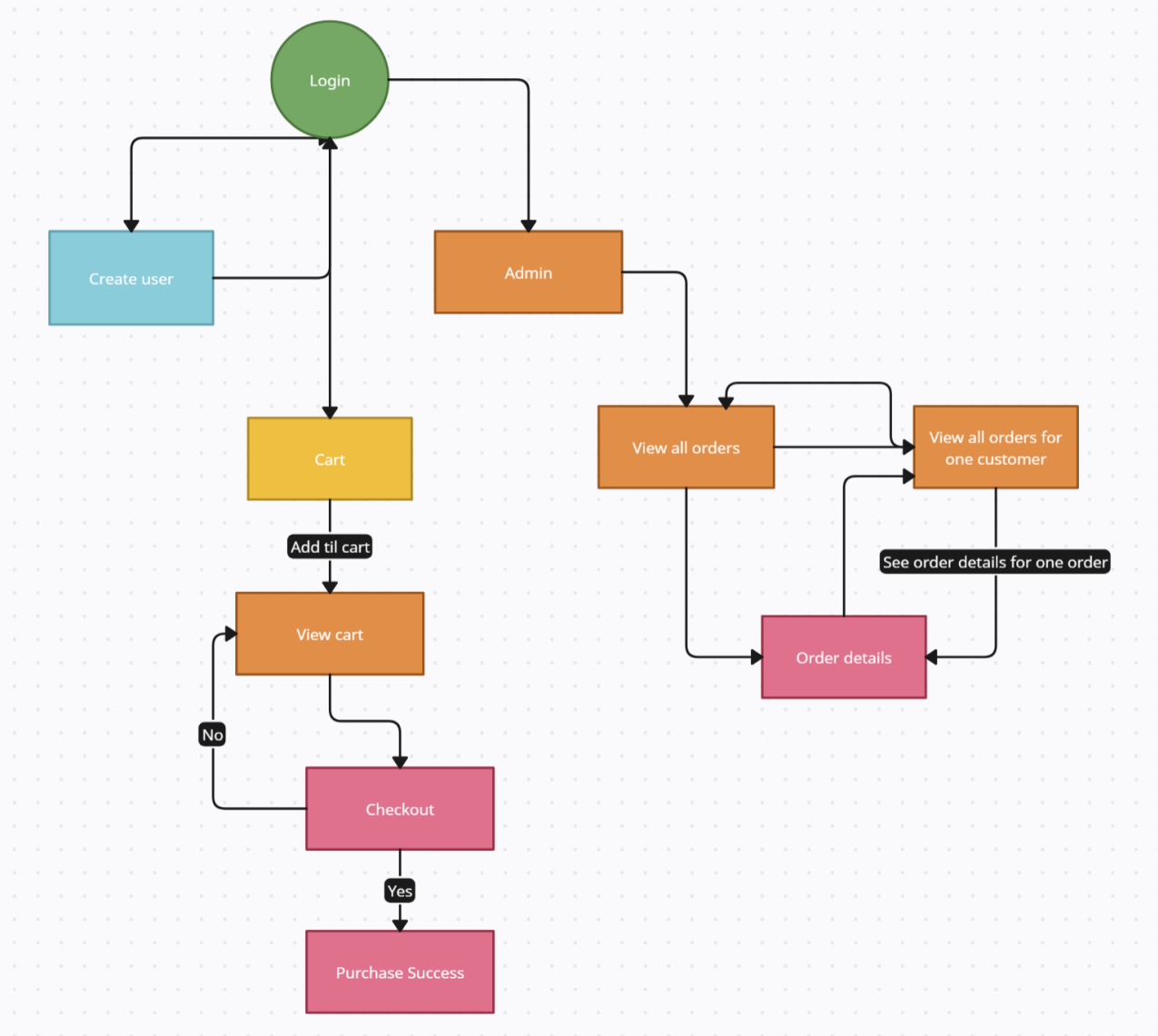
Figur 4 – Admins navigations bar.

Navigationsdiagram

Nedenstående figur 5 viser vores navigationsdiagram, som følger hvordan man som bruger, herunder både som kunde og som admin, benytter sig af vores webapp. Fra at kunden har mulighed for enten at oprette sig som ny bruger eller logge ind som eksisterende kunde.

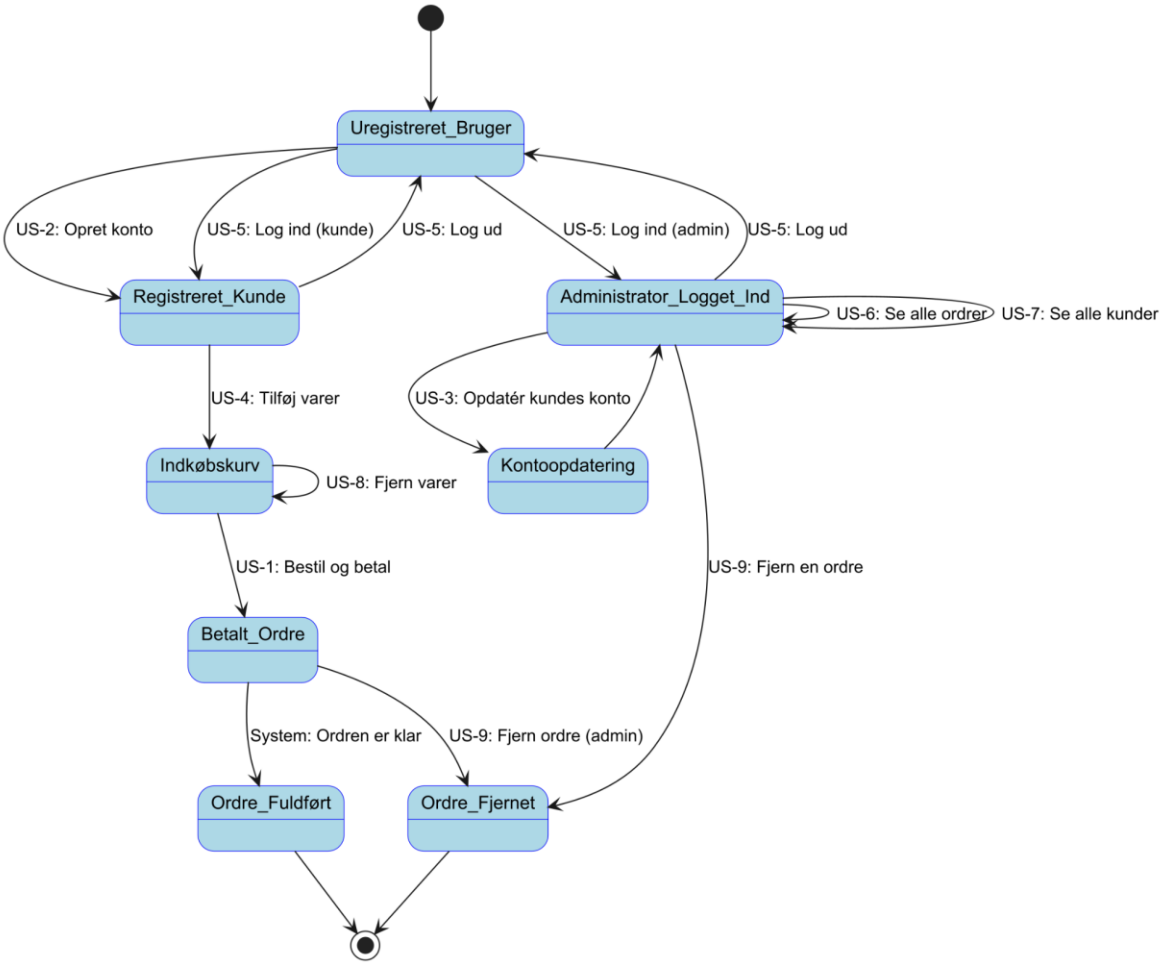
Hvorefter brugeren bliver sendt tilbage til log ind html siden og kan komme videre i programmet.
Efter kunden er logget ind bliver kunden sendt videre til indkøbskurv, hvor kunden har mulighed for at tilføje de nødvendige dele. I vores program består det af både en bund, topping og antallet af cupcakes. Efter at valgene er registreret, kan kunden komme videre i programmet og oprette samt betale for sin bestilling, for at færdiggøre en ordre i systemet.

Figur 5 viser også hvordan man som admin bruger i vores program, har mulighed for at se kundernes ordrer og navigerer rundt mellem de forskellige ordrer og kunder i systemet. For at kunne danne sig et overblik over alle ordrer og hver enkelt kundes ordrehistorik, som hjælper til at holde styr på at alle ordrer bliver bearbejdet rigtigt.



Figur 5 – Navigationsdiagram.

Figur 6 viser et state diagram, hvor vi har sat de givne user stories ind og det viser programmets flow. Ligesom i vores navigationsdiagram viser brugerens oplevelse af programmet fra start til slut.



Figur 6 – State Diagram.

Særlige forhold

Guard condition

Vi har valgt at det eneste der gemmes i sessionen er **currentMember**, fordi denne er vigtig i forbindelse med login og fungerer som en guard condition for alle siderne. Det gør man skal være logget ind som kunde/admin for at se henholdsvis kunde eller admin siderne.

Exceptions

Vi har bruger vores egen database exception, defineret i DatabaseException.java, som vi bruger når vi henter data fra databasen. Vores data mappers kaster database exceptions og de bliver grebet af vores controllers. Vores controllers kaster ikke disse exceptions videre, men skriver i stedet en fejlbesked ud til brugeren i frontenden.

Implementering af kurv

Vi har valgt at kundens kurv bliver gemt i databasen, i stedet for session scope. Dette betyder at kunden kan fylde cupcakes i sin kurv, og selvom session bliver budt, kan kunden fortsætte sin ordre, når personen logger ind igen.

Status på implementation

Vi har implementeret User Stories 1-8, samt en kundeside, hvor kunder kan se alle deres ordrer. Hvilket resulterer i en fuldt funktionel webapplikation, som kunne bruges online, selvom den i øjeblikket kun kører lokalt med server og klient på samme computer.
Der er lavet styling til desktop versionen, men vi har ikke haft tid til at optimere designet til mobil størrelser.
Næsten alle mappers har integrationstests, undtagen **OrderMapper**, som vi ikke nåede at teste grundet code freeze. I **OrderlineMapper** mangler der også to tests, da enkelte metoder blev ændret under en omskrivning kodning.

Overordnet design

Vi har valgt at kræve login, før brugerne kan se og købe cupcakes. Et alternativ kunne have været, at brugerne kunne se produkterne uden login og først skulle logge ind ved køb. Men da opgaven kræver en tjek af, om brugeren har penge på kontoen, valgte vi login fra starten. Dette sikrer, at hvis en administrator skal tilføje penge til en bruger, sker det inden købsprocessen og ikke i sidste øjeblik.

Gemme data

For at sikre et mere robust system gemmer vi kundens kurv direkte i databasen i stedet for i session scope. På denne måde bevares data, selv hvis en kunde afbrydes midt i købsprocessen, hvilket sikrer, at kurven altid er opdateret.

Exceptions

Vi håndterer exceptions på engelsk for udviklere og på dansk for brugerne. På grund af tidsbegrænsninger og code freeze har vi ikke været konsekvente i implementeringen. Mapperne smider som regel engelske exceptions, mens controllerne håndterer danske. Vi kunne med fordel logge de engelske fejlbeskeder for at lette debugging. (Se figur # for et eksempel på exceptions fra en mapper).

Her på figur 7 ses et eksempel på en exception, som er tilegnet software delevopers.

```
public static ArrayList<Bottom> getAllBottoms(ConnectionPool connectionPool) throws DatabaseException {  2 usages
    ArrayList<Bottom> bottoms = new ArrayList<>();

    String sql = "SELECT * FROM bottom";

    try (Connection connection = connectionPool.getConnection();
         PreparedStatement ps = connection.prepareStatement(sql)) {

        ResultSet rs = ps.executeQuery();

        while (rs.next()) {
            int buttomId = rs.getInt( columnLabel: "bottom_id");
            String bottomName = rs.getString( columnLabel: "bottom_name");
            double bottomPrice = rs.getDouble( columnLabel: "bottom_price");
            bottoms.add(new Bottom(buttomId, bottomName, bottomPrice));
        }

    } catch (SQLException e) {
        throw new DatabaseException("Error in getting bottoms from database", e.getMessage());
    }
    return bottoms;
}
```

Figur 7 - Eksempel på exception som vises til udvikleren.

Her på figur 8 ses et eksempel på exceptions som bliver vist til brugerne

```
private static void showCart(Context ctx, ConnectionPool connectionPool) {  1 usage  Peter Bollhorn +2
    Member currentMember = ctx.sessionAttribute( key: "currentMember");
    if (currentMember == null) {
        ctx.attribute("errorMessage", "Log ind for at bestille.");
        ctx.render( filePath: "error.html");
        return;
    }

    try {
        int activeOrderNumber = OrderMapper.getActiveOrderNumber(currentMember.getMemberId(), connectionPool);
        ArrayList<Orderline> orderlines = OrderlineMapper.getOrderlinesByOrderNumber(activeOrderNumber, connectionPool);
        if (orderlines.isEmpty() ) {
            ctx.attribute("tomKurv", "Kurven er tom.");
            ctx.render( filePath: "kurv.html");
            return;
        }
```

Figur 8 – Eksempel på exception som vises til brugeren.

Validering af brugerinput

Brugerinput håndteres hovedsageligt via HTML, hvor alt input gemmes som en *String*. Der er simpel validering, såsom at e-mails skal indeholde "@". Derudover sikrer en *unique constraint* i SQL, at der ikke kan oprettes to brugere med samme e-mail.

Sikkerhed

1) Vi har ikke implementeret yderligere sikkerhed udover den SQL-validering, hvor input behandles som *String* og sammenlignes med eksisterende data i databasen. Altså vi bruger ikke password hashing, så applikationen har en begrænset sikkerhed.
2) Vi har guard conditions på siderne, så brugere ikke får adgang til sider, de ikke har tilladelse til. Grundet code freeze nået vi ikke at implementere guard condition på at slette ordrelinjer. Så hvis en ondsindet person ved, hvad der skal skrives i browseren URL bar, kan personen slette ordrelinjer for alle brugere.
3) Vi har ikke implementeret andet validering af brugerinput end det som html tilbyder.
4) Vi bruger *PreparedStatement* i vores SQL queries, for at beskytte os mod SQL injections. *PrepareStatement* er også industri standarden.

Proces

Planlægning af arbejdsform og projektforløb

Vi startede med at definere en arbejdsform med fokus på daglige morgenmøder og en tydelig opdeling af opgaver. Vores plan var at bruge mange af elementer fra Scrum, såsom Daily Standup, Sprint Review, Increment, Definition of Done og Scrum Board, hvor vi løbende kunne justere på eventuelle udfordringer, der krævede hele teamets opmærksomhed. Hvert teammedlem tog en opgave fra vores Kanban-board og arbejdede selvstændigt med mulighed for at bede om hjælp ved behov. Til versionsstyring anvendte vi git teknologien og platformen GitHub og en struktureret gren-struktur med Main -> dev -> feature branch. For at minimere merge-konflikter aftalte vi, at hver feature-branch skulle merges med dev lokalt, før den blev merged ind i dev igen. Alt, der skulle tilføjes til Main, blev overført via pull requests, og vi gennemgik dette sammen for at sikre fælles indsigt i koden og maksimere læringen for alle.

Forløb i praksis

I praksis blev vores planlagte arbejdsform kun delvist fulgt. I stedet for ét dagligt møde om morgenen, holdt vi mange gange op til tre møder om dagen. Kommunikation foregik også ofte ad hoc, hvilket førte til mange løbende afklaringer uden for de planlagte møder.

Hvad gik godt, og hvad kunne være bedre

En af de mest positive faktorer var vores effektive brug af GitHub til versionsstyring, som sikrede, at teamet kunne arbejde parallelt uden større konflikter i koden. Dog kunne vi have strammet op på vores mødestruktur og kommunikationsform, da de mange spontane afklaringer til tider skabte forvirring.
Vi lavede Integrationstestne på bagkant, hvor det ville have givet bedre mening at lave dem først eller lige efter, så man kunne sikre sig at metoderne virkede og ikke behøvede at manuelt test på dette.
Vi lavede også domæne modellen på bagkant, hvilket også ville have været smartere som noget af det første.

Læring og forbedringsmuligheder

Vi har under forløbet lært en række kodemæssige principper, som ville have resulteret i forbedrede metoder, hvis vi havde kendt dem fra starten. Dette gælder især i forhold til strukturen af vores CSS, exception handling og formulering af fejlbeskeder samt at lave

opgaver mere i den rigtige rækkefølge. Til fremtidige projekter vil vi være mere opmærksomme på at optimere disse elementer for at sikre en bedre kodekvalitet og mere effektiv samarbejdsproces.