Olsker Cupcakes

Så e dejn klarad

Cupcake Projekt
Datamatiker 2. Semester
CPHBusiness Lyngby
7. April 2024

Patrick, Nadia, Sandra, Victor
Patrick Please, Shiihon, cph-sm522, Scrabble30
cph-nh354@cphbusiness.dk, cph-vd52@cpbusiness.dk, cph-sm522@cphbusiness.dk

Indholdsfortegnelse

Indledning

Vi har fået til opgave at lave en hjemmeside til "Olsker cupcakes", med en række funktionelle krav, samt et delvis færdigt mockup. Formålet med hjemmesiden er at gøre det muligt for firmaet at kunne tage imod bestillinger, samt kommunikere med kunden. Rapporten er skrevet med udgangspunkt i at en anden datamatiker studerende skal kunne læse og forstå denne. En video til demonstration af det endelige produkt findes i bilags dokumentet suppleret med rapporten.

Baggrund

Dette projekt er for Olsker Cupcakes, som er et dybdeøkologisk bageri fra Bornholm.
Et par hipsters fra København indsamlede nogle krav efter de besøgte bageriet og efterlod en halvfærdig mockup af hvordan de forestillede sig, at den færdige forside for hjemmesiden skulle se ud. Den mockup har fungeret som en foreløbig skitse og har været udgangspunkt for det videre arbejde, hvori det har været vores opgave at udfylde de funktionelle mangler og foreslå forbedringer.

I takt med projektets progression, har vores umiddelbare mål været at udvikle en prototypisk version af hjemmesiden, hvor vi skulle koncentrere os om de essentielle funktioner. De funktionelle krav inkluderer blandt andet muligheden for at kunderne kan bestille og betale for cupcakes, hvor de selv kan vælge både bund og top, samt antal.

Vores primære intention med implementeringen af disse krav er at opfylde behovet for et dynamisk og brugervenligt bestillingssystem. Et system, der ikke kun understøtter kundernes behov om personliggjorte cupcake-bestillinger, men også imødekommer behovet hos administratorer for effektivt at kunne overvåge og administrere bestillinger. Derved ville systemet udgøre Olsker Cupcakes' online tilstedeværelse, gøre bestillingsprocessen mere effektiv og forbedre kundeoplevelsen.

Teknologivalg

Projektet er bygget på Java 17, Corretto SDK fra Amazon, version 17.0.10 og udviklet i IntelliJ IDEA 2024.1. Til backend bruges Javalin 6.1.3 og til frontend Thymeleaf template version 3.1.2.
Når det kommer til persistens af data i projektet, bliver der gjort brug af en database heraf PostgresSQL version 16.2. Selve kommunikationen med databasen sker ved hjælp af PostgresSQL JDBC Driver 42.7.2 med hikariCP version 5.1.0 til at oprette en connection pool, som kan tilgå databasen. I projektet er der yderligere foretaget integrationstest, der gør brug af junit 5.10.2.

Krav

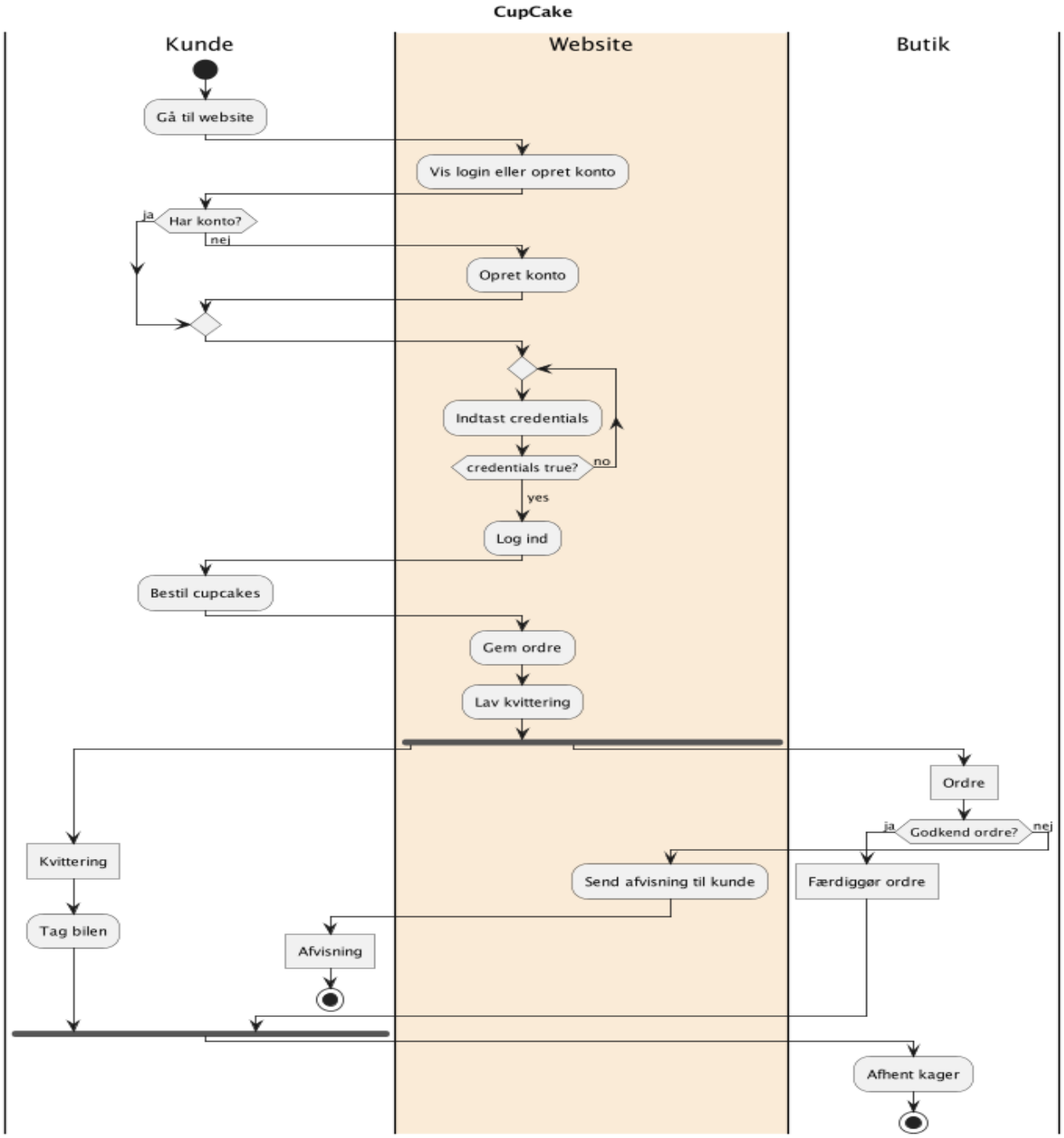
Virksomheden Olsker Cupcakes ønsker dermed et system der tillader deres kunder at bestille cupcakes med en given bund, top og antal. Efterfølgende kan kunder betale for deres ordre og køre forbi butikken hvor de kan hente deres ordrer. Det skal gøre det nemt for virksomheden at modtage og se bestillinger fra kunder samt at kunne administrere ordrer såsom at slette ugyldige ordrer og lign. Derudover skal det være muligt at kunne se, hvilke kunder der ligger i systemet.

På den måde ville systemet give virksomhedens kunder en ny måde at bestille og købe varer på samt hjælpe virksomheden til at få en overskuelig måde at holde overblik over bestillinger fra deres hjemmeside. De endelige user stories for hjemmesiden kan ses forneden som projektet er blevet udarbejdet på baggrund af.

User stories (Funktionelle krav)

Nr.	User Story	Status
US-1	Som kunde kan jeg bestille og betale cupcakes med en valgfri bund og top, så jeg senere kan køre forbi butikken i Olsker og hente min ordre.	Færdig
US-2	Som kunde kan jeg oprette en konto/profil for at kunne betale og gemme en ordre.	Færdig
US-3	Som administrator kan jeg indsætte beløb på en kundes konto direkte i Postgres, så en kunde kan betale for sine ordrer.	Færdig
US-4	Som kunde kan jeg se mine valgte ordrelinjer i en indkøbskurv, så jeg kan se den samlede pris.	Færdig
US-5	Som kunde eller administrator kan jeg logge på systemet med e-mail og kodeord. Når jeg er logget på, skal jeg kunne se min email på hver side (evt. i topmenuen, som vist på mockup'en).	Færdig
US-6	Som administrator kan jeg se alle ordrer i systemet, så jeg kan se hvad der er blevet bestilt.	Færdig
US-7	Som administrator kan jeg se alle kunder i systemet og deres ordrer, så jeg kan følge op på ordrer og holde styr på mine kunder.	Færdig
US-8	Som kunde kan jeg fjerne en ordrelinje fra min indkøbskurv, så jeg kan justere min ordre.	Færdig
US-9	Som administrator kan jeg fjerne en ordre, så systemet ikke kommer til at indeholde ugyldige ordrer. F.eks. hvis kunden aldrig har betalt.	Færdig

Aktivitetsdiagram



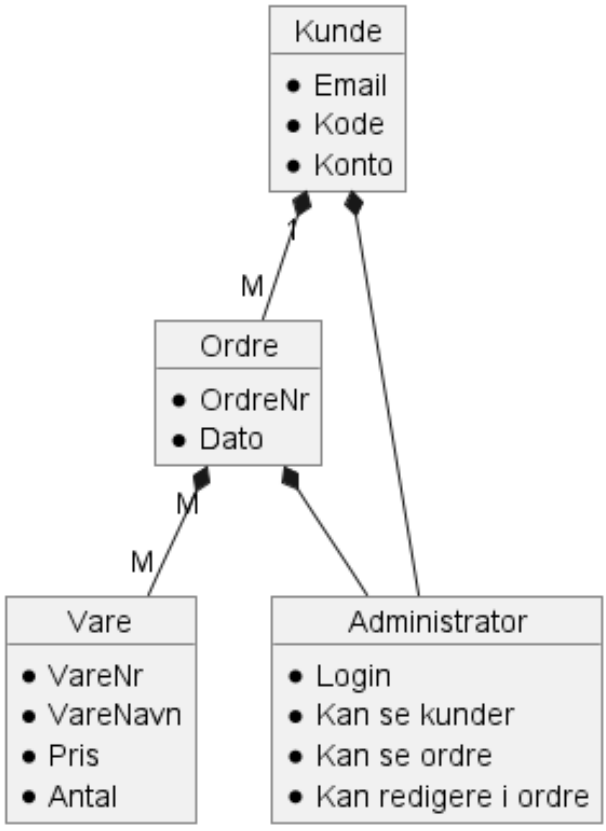
Aktivitetsdiagram over hjemmesiden

Dette aktivitetsdiagram beskriver processen for en kunde, der bestiller cupcakes gennem en hjemmeside. Processen starter når kunden ankommer til hjemmesiden, hvor de står overfor valget mellem at logge ind eller oprette en ny konto. Uafhængigt af kundens valg, enten at fortsætte med en eksisterende konto eller at oprette en ny, så fører næste trin dem til at indtaste de nødvendige legitimationsoplysninger for at kunne fortsætte.

Denne proces kan gentages, indtil korrekt information er indgivet, hvorefter kunden officielt er logget ind. Med adgang til hjemmesiden kan kunden derefter bestille deres ønskede cupcakes. Hjemmesiden håndterer bestilling ved først at gemme den og derefter generere en kvittering for kunden efter den er blevet betalt.
Fra dette punkt deler processen sig i to. På den ene side modtager kunden deres kvittering og gør klar til at hente deres bestilling. På den anden side evaluerer butikken ordren for at afgøre, om den kan godkendes.

Hvis ordren godkendes, så færdiggøres forberedelserne og kunden kan hente deres cupcakes. Skulle ordren derimod afvises, ville kunden blive informeret om dette, og processen ender.

Domæne model



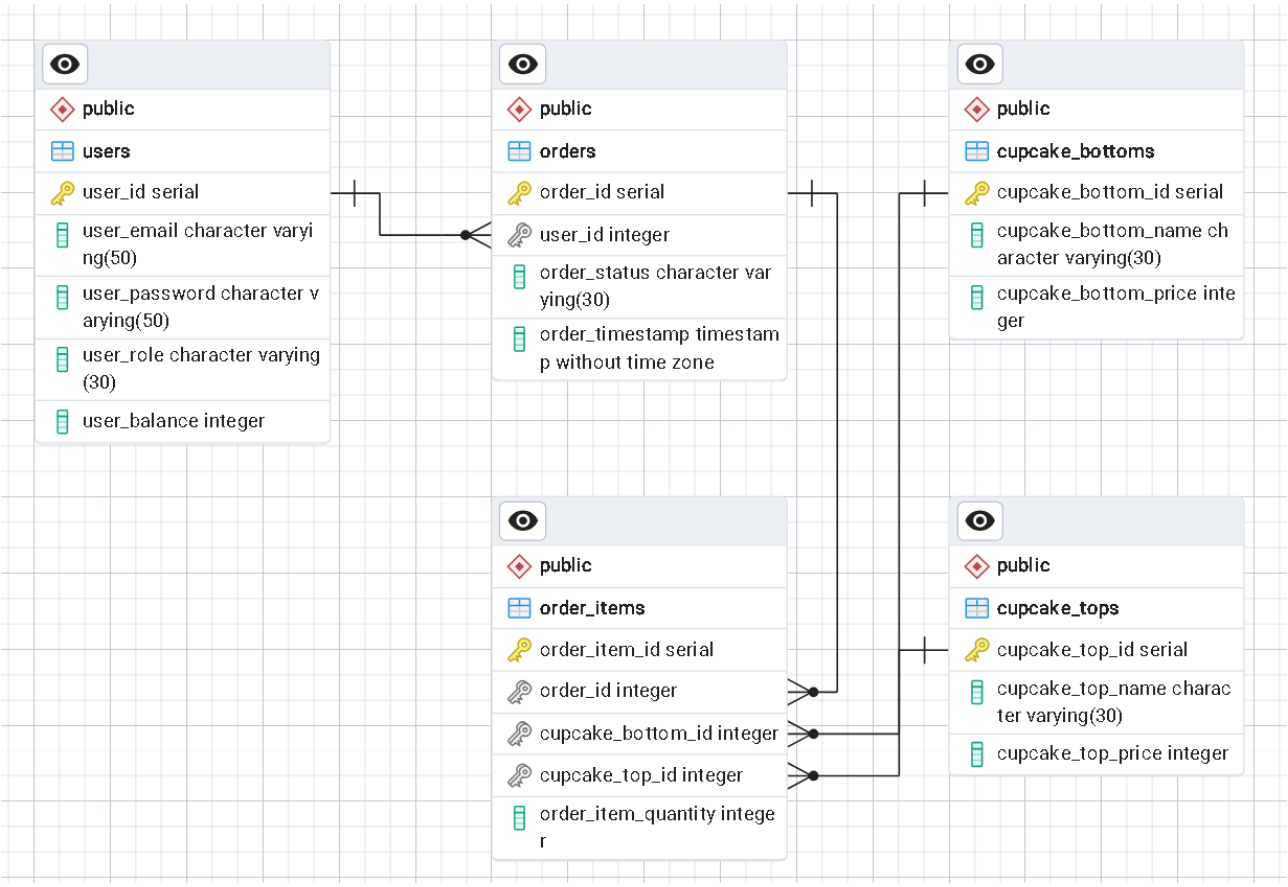
Domænemodel

- *En kunde kan have mange ordrer.*
- *En ordre tilhører en enkelt kunde, men kan indeholde mange varer.*
- *En vare kan være i flere ordrer.*
- *En administrator har adgang til at se ordrer og kunder, samt kan redigere i ordre.*

En kunde kan oprette en konto og derved kunne lave bestillinger (ordrer), kunden har også en konto - saldo, som de bruger til at kunne købe cupcakes med på Olsker's hjemmeside. Hver ordre tilhører en enkelt kunde
(1-M relation). Vi har valgt en M-M relation mellem varer og ordrer da en ordre kan indeholde mange varer, og en vare kan optræde i flere ordrer. Til sidst har vi administratoren som har adgang til at se alle de ordrer der er, og redigere i disse (complete eller in process), admin har også adgang til at se hvilke kunder ordrerne tilhører.

ER Diagram

Databasen har vi valgt at strukturere med fem tabeller. De følgende tabeller er users, orders, order_items, cupcake_bottoms, cupcake_tops. Alle tabeller gør brug af en unikt autogenereret id som primærnøgle, hvor dataen gemmes på. Tabellerne er lavet med udgangspunkt i at være på tredje normalform.



ER diagram over databasen

Brugere

Brugerdata opbevares i users tabellen hvor brugerens e-mail bliver gemt samt brugerens kodeord, rolle og balance. Brugerens e-mail gør brug af en unique constrain der gør at man ikke kan oprette en ny bruger med en mail der allerede findes i systemet. Rollen beskriver, om brugeren er kunde eller administrator, for eksempel ejeren af virksomheden. Balance beskriver hvor mange penge brugeren har i systemet.

Til at opbevare brugerens penge i systemet, deres balance, har vi valgt at bruge integer for at reducere systemets kompleksitet. Dette betyder dog at det ikke er muligt at opbevare komma tal for priser eller lign. Grunden til at vi har valgt at opbevare penge i systemet som integer frem for andre datatyper såsom float og double er for at undgå afrundingsfejl der kan opstå med de datatyper.

Datatypen double har større præcision end float, men afrundingsfejl kan stadig forekomme. PostgresSQL har også en datatypen numeric der har større præcision end double. Denne datatype bliver også anbefalet af Postgres til at repræsentere penge i databasen. Men penge skal også kunne repræsenteres i Java programmet i en entity-klasse, der også kræver en datatype.

I Java programmet møder vi samme problemer som i databasen. Her kan vi også bruge float og double til at repræsentere penge i systemet. Men de er igen uønskede grundet risikoen for afrundingsfejl. Der findes BigDecimal som har bedre præcision end double, men kan til gengæld være besværlig at arbejde med når det kommer til matematiske operationer. Så for at minimere kompleksiteten af systemet valgte vi at repræsentere penge som en integer.

Bestillinger

For bestillinger har vi orders tabellen der gemmer bruger id på den bruger som oprettede bestillingen samt en ordre status og et ordre timestamp. Bruger id'et er en foreign key og ordre timestamp gemmer hvornår ordren blev oprettet.

Ordrelinjer

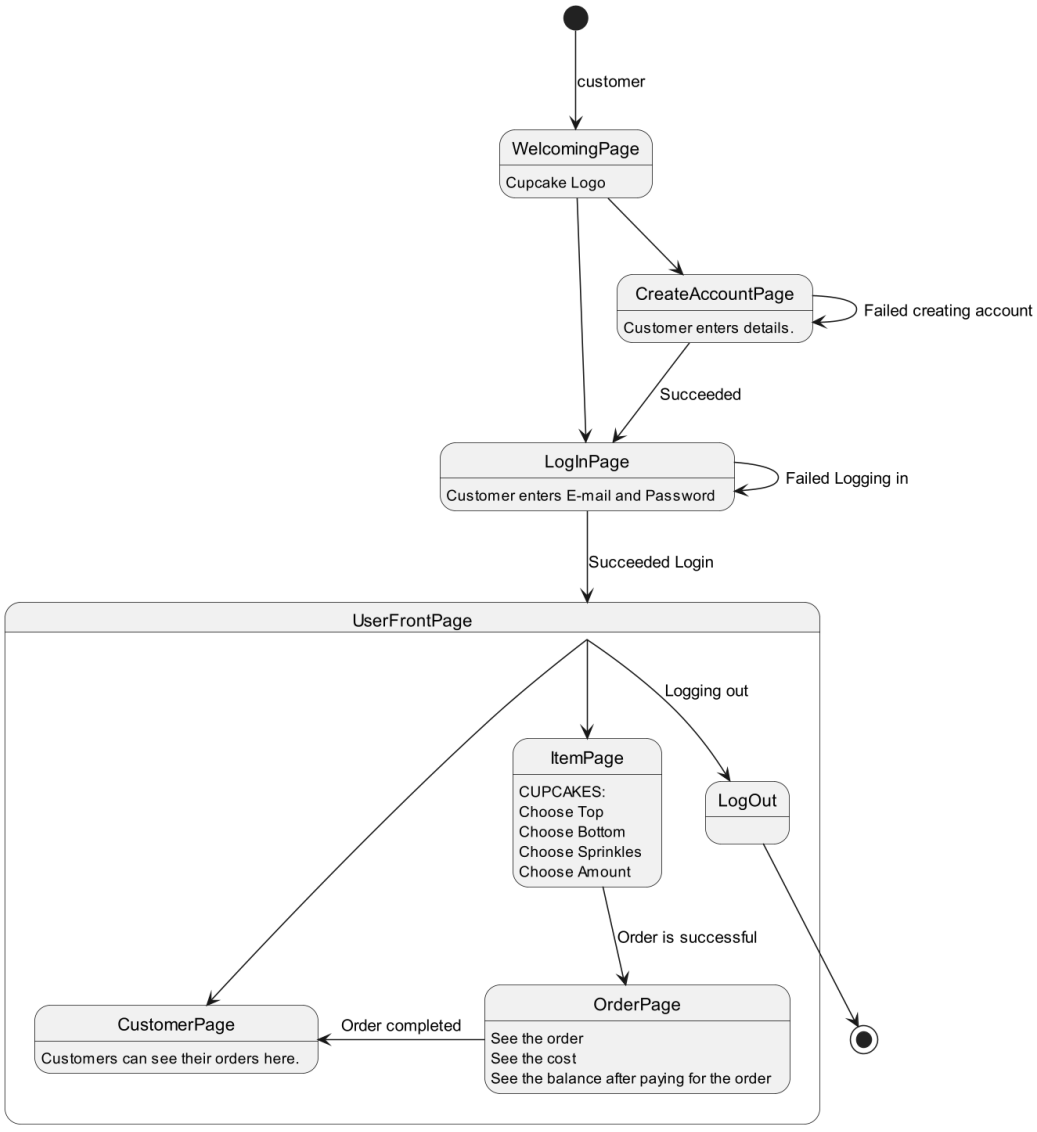
Alle ordrelinjer gemmes i order_items tabellen med et ordre id, en cupcake bund id, en cupcake top id og et antal for hvor mange cupcakes af den valgte bund og top der ønskes. Ordre id, cupcake bund id og cupcake top id er foreign keys hvor ordre id henviser til hvilken ordre denne ordrelinje tilhører, cupcake bund id for hvilken bund der er blevet valgt og cupcake top id for hvilken cupcake top der er blev valgt.

Cupcake bunde & toppe

Til sidst har vi, cupcake_bottoms, til at gemme vores cupcake bunde med navn og dens pris. Til vores cupcake toppe har vi cupcake_tops som gemmer en tops navn og pris. Det kan ikke undgås at gennemskue at de to tabeller ligner utrolig meget hinanden. På daværende tidspunkt af diagrammets oprettelse var det ikke klart hvordan de to evt. kunne sættes sammen til en enkelt tabel. Men muligvis en cupcake_part tabel med en type kolonne eller en mere generel produkt tabel kunne have løst denne situation.

Navigationsdiagram

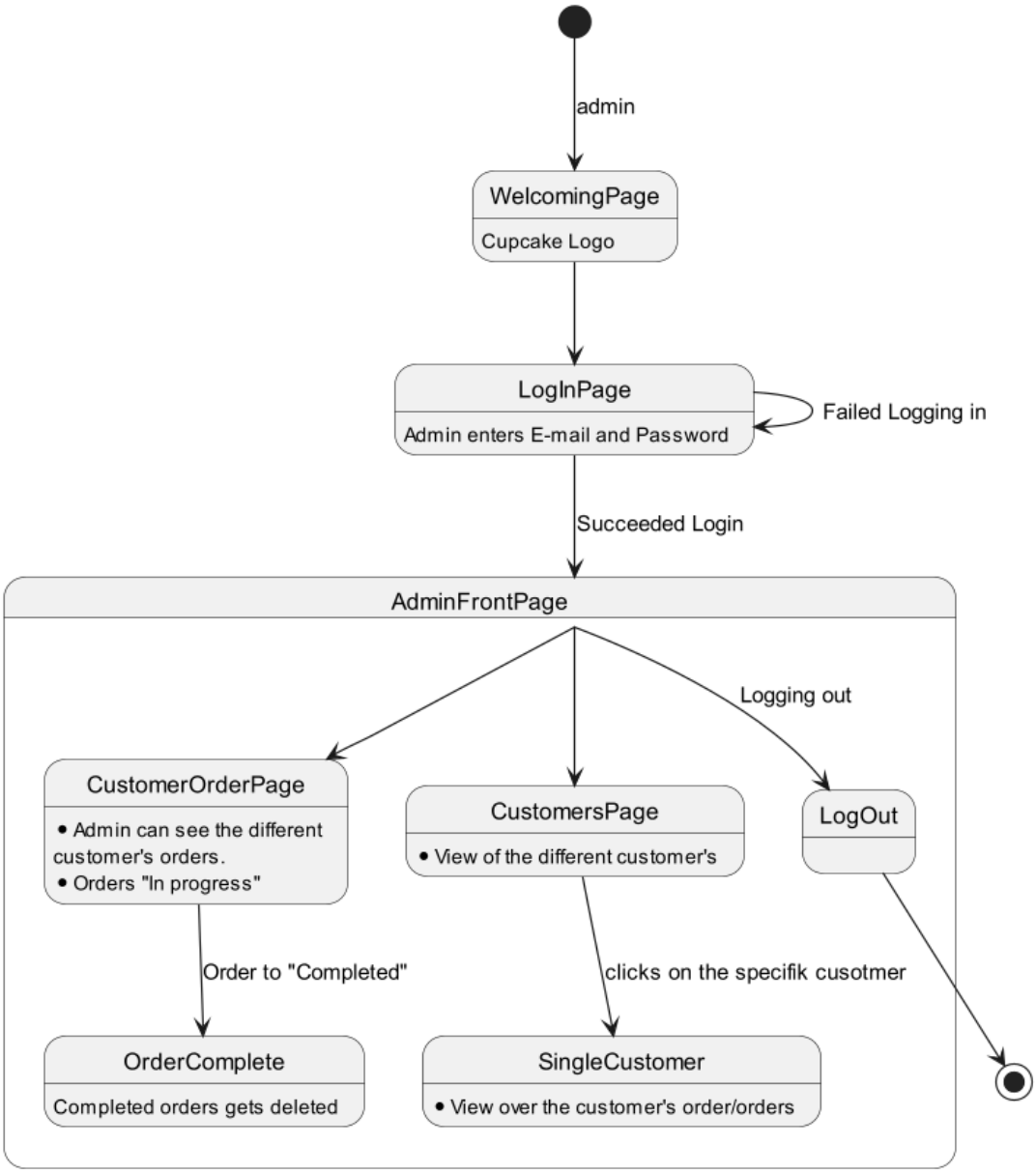
Kunde:



Navigationsdiagram for kunder

Kunden kommer ind til velkomstsiden som har firmaets logo på, er kunden allerede en del af systemet skal de logge på, ellers skal de oprette en ny konto, som herefter viser tilbage til log ind siden. Når kunden har logget ind kommer de ind til en ny side - forsiden, hvor de kan vælge om de vil se deres ordrer, eller om de vil oprette en ny ordre. Opretter de en ny, skal der vælges, top, bund, og drys samt antal og klikke på bestil ordre. På ordre siden kan man se hvad man skal til at bestille, hvor meget det kommer til at koste samt hvor meget kunden har tilbage i sin "kunde konto", efter købet, hvor ordren vil blive smidt tilbage til kundens egen ordreside.

Admin:



Navigationsdiagram for administrator

Admin kommer ind på samme måde som en kunde, forskellen her er at admin ikke kan oprette en ny konto, men kommer ind på sin egen forside, hvor der er en side til kunder, samt den enkelte kundes side hvor man kan se de specifikke ordrer der tilhører den valgte kunde. Admin kan også vælge at gå ind og se alle de ordrer der er i gang ("In Process"), og sætte dem til "completed" når ordren er klar til at blive afhentet. Ordren bliver derefter slettet.

Særlige forhold

Bruger input validering

For at sørge for at fremtidige kunder og nuværende kunder har hver deres ordre, og hver deres kontaktoplysninger (mindsker risiko for fejl ved ordrerne fra firmaets side), er der blevet sat en unique constrain på brugerens E-mail i vores database - dvs. at disse skal være unikke. På denne måde kan der let differentieres mellem kunder og deres bestillinger.

Prøver brugeren at oprette en ny konto med samme mail, vil der komme en besked der fortæller at e-mailen allerede findes, og at man enten skal logge ind eller prøve igen, med en ny e-mail. Inde i vores UserMapper klasse sørger vi også for at få en besked i vores terminal, hvor vi kan se, hvis brugeren ikke er sat ind i databasen, kan vi konkludere at der er sket en fejl.

```
try (
        Connection connection = connectionPool.getConnection();
        PreparedStatement ps = connection.prepareStatement(sql)
) {
    ps.setString( parameterIndex: 1, email);
    ps.setString( parameterIndex: 2, password);

    int rowsAffected = ps.executeUpdate();
    if (rowsAffected != 1) {
        throw new DatabaseException("Fejl ved oprettelse af ny bruger");
    }
```

Inde i UserMapper, createUser metoden

Her er et lille eksempel hvor der bliver kigget efter om hvorvidt der er en duplicate value - altså en gentagelse af en allerede eksisterende mail. Sker dette, bliver beskeden "msg" sendt ud til brugeren.

```
} catch (SQLException e) {
    String msg = "Der er sket en fejl. Prøv igen";
    if (e.getMessage().startsWith("ERROR: duplicate key value ")) {
        msg = "E-mailen findes allerede. Vælg et andet, eller log ind";
    }
```

Inde i UserMapper, createUser metoden nede i catch delen af try/catch

Separation of Concern

```
private static void placeOrder(Context ctx, ConnectionPool connectionPool) {
    try {

        User user = ctx.sessionAttribute( key: "currentUser");
        List<OrderItem> basket = ctx.sessionAttribute( key: "basket");

        if (user != null && basket != null && !basket.isEmpty()) {

            Order newOrder = new Order(user.getUserId(), basket,  status: "In Progress", LocalDateTime.now());

            int totalPrice = newOrder.getTotalPrice();
            int currentBalance = user.getBalance();

            if (currentBalance >= totalPrice) {
                int newBalance = currentBalance - totalPrice;
                UserMapper.setUserBalance(user.getUserId(), newBalance, connectionPool);
                OrderMapper.createOrder(newOrder, connectionPool);

                ctx.sessionAttribute("basket", new ArrayList<OrderItem>());
                ctx.redirect( location: "/pop-up");
```

placeOrder metoden inde i OrderController

Denne kode viser vores tilgang til at placere en ordre på hjemmesiden, ved at anvende principperne om separation of concerns (SoC).

Vi har session management som er håndteret ved at hente den aktuelle bruger og indkøbskurven fra sessionen. Det adskiller håndtering af brugerdata fra selve ordre placeringens logik, hvilket bidrager til en renere og mere overskuelig kodebase.

Forretningslogikken tjekker, om brugerens saldo er tilstrækkelig til at dække indkøbskurvens samlede pris. Hvis dette er tilfældet, opdateres brugerens saldo, og en ny ordre oprettes i databasen.

Vi har database logikken i UserMapper og OrderMapper klasserne, som håndterer al interaktion og kommunikation med databasen. Dette sikrer, at selve den måde som hjemmesiden fungerer på, er adskilt fra de specifikke detaljer om datalagring, hvilket gør koden mere fleksibel og lettere at opdatere eller udvide i fremtiden.

Sammenkædet demonstrerer disse elementer en effektiv anvendelse af SoC-princippet, hvilket resulterer i en kode, der er lettere at forstå, teste, og vedligeholde.

Metode til at genindlæse den nuværende side

Nogle steder i programmet er det nødvendigt blot at lave en ændring til den nuværende html side og derefter opdatere siden. Det kan for eksempel være hvis brugeren sletter en ordrelinje fra sin indkøbskurv eller en administrator sletter en ugyldig ordre under bestillinger. I sådanne situationer skal siden vises som den samme med den samme url men blot opdatere indholdet. Til at genindlæse den samme side med den samme url, lavede vi en metode refreshCurrentPage(). Denne metode tager ind Context, der indeholder det seneste request. I denne request header gemmes et referer link der henviser til hvilken side brugeren var på før de gik videre.

Med alt dette er det muligt at rekonstruere URL af den side brugeren stod på samt evt. hvilke query parameters der var en del af den URL. Dermed kan vi redirecte til dette URL, hvorved det ville ramme vores routing og loade den nødvendige data. Yderligere kan en fallbackpage suppleres i metoden hvis det ikke lykkedes at rekonstruere den originale URL. Denne fallbackPage kan for eksempel være vores user-frontpage eller vores admin-frontpage. Så hvis det ikke var muligt at loade den originale side ville den falde tilbage på en af forsiderne, an på om brugeren er en kunde eller administrator.

```
207
208     private static void refreshCurrentPage(Context ctx, String fallbackPage) {  3 usages
209         try {
210             URI previousURI = new URI(ctx.req().getHeader( s: "referer"));
211             previousURI = new URI( scheme: null,  authority: null, previousURI.getPath(), previousURI.getQuery(),  fragment: null);
212
213             ctx.redirect(previousURI.toString());
214         } catch (URISyntaxException | NullPointerException e) {
215             ctx.redirect(fallbackPage);
216         }
217     }
218
```

refreshCurrentPage metode i OrderController klassen

Status på implementation

Vi kan konkluderer at vi har nået at implementere alle 9 funktionelle krav, samt de ikke funktionelle krav. Derudover er alle siderne på hjemmesiden blevet stylet. Dog blev der ikke lagt stor vægt på hjemmesidens responsivitet. Det virker til nød men kan forbedres. Dette ses bedre på mindre enheder såsom tablets eller telefon. Derudover er hjemmesiden implementeret primært ud fra den minimalistiske mockup. Dette betyder at der er mangler på brugervenlig respons forskellige steder på hjemmesiden såsom når en kunde tilføjer en ordrelinje til sin indkøbskurv. Dette ville være et område at arbejde videre på.

De forskellige klasser, controllere, mappere, entity klasser og lign er blevet opdelt i packages for overskuelighedens skyld. Dog givet den dårlige planlægning, har der ikke været meget tid eller overskud til refaktorering af koden. Så dette ville yderligere være et fokusområde, hvis man skulle arbejde videre på projektet.

Proces

I arbejdsprocessen for projektet stod vi overfor både udfordringer og lærerige øjeblikke. Vores tilgang til projektet var ikke planlagt fornuftigt i forhold til arbejdsformer eller arbejdsforløb, da vi startede projektet. Det gjorde, at der heller ikke var en klar fordeling af roller inden for gruppen. Dette førte til at vi gik direkte i gang med kodningen og hjemmesider uden først at have færdiggjort vigtige planlægningsværktøjer som klasse og domæne diagram. Arbejdsopgaver blev fordelt uden en struktureret tilgang, og vores kommunikation omkring mødetider var ikke klar, hvilket resulterede i ineffektivitet.

Vi valgte at implementere Kanban som en metode til at strukturere og organisere vores opgaver bedre, hvilket hjalp os til at få et bedre overblik over opgaver og gav en mere effektiv arbejdsfordeling. For at gøre selve kodningen mere effektiv, begyndte vi at anvende pull requests og branch protection for at mindske konflikter og forbedre kvaliteten af den kode vi producerede.

Selvom vi havde forskellige synspunkter, som til tider kunne skabe frustration, så ved vi at de diskussioner vi kunne have, var lærerige og bidrog til vores faglige udvikling. Udover det, så forblev vores arbejdsmoral høj på trods af de komplikationer vi stødte på undervejs.

Hele processen har lært os vigtigheden af god planlægning og god kommunikation i projektarbejde, og vi vil i fremtidige projekter, lægge større vægt på planlægning i begyndelsen af arbejdsprocessen for at opnå en mere effektiv proces.

Bilag

Link til video demonstration af vores projekt:
https://youtu.be/CWXv2S8z3w0