Cupcake

31. Oktober 2024

Jonas Guldborg Kestenholz
cph-jk491@cphbusiness.dk
Jonas-Kestenholz
f24 vinterstart

Rasmus Kristoffer Knudsen
cph-rk181@cphbusiness.dk
RasmusKnudsen
f24 vinterstart

Micke Joe Dengaard
cph-md405@cphbusiness.dk
MickeJDengaard
f24 vinterstart

Valdemar Arthur Larsen
cph-vl91@cphbusiness.dk
ValdemarLarsen02 & ValdemarLarsen
f24 vinterstart

Indholdsfortegnelse

Indholdsfortegnelse	**1**
Indledning	**2**
Teknologivalg	**2**
Krav	**4**
Aktivitetsdiagram	**5**
Domæne model	**7**
Navigationsdiagram	**8**
Særlige forhold	**10**
ERD Diagram	**14**
Implementation	**15**
Proces	**15**

Indledning

Denne rapport omhandler en overdragelse af et program til virksomheden “Olsker cupcakes”. Programmet her skal gøre det lettere for firmaets kunder at bestille cupcakes digitalt. Vi har brugt diverse redskaber til at komme i mål med “Olsker cupcakes” krav til programmet, alt dette vil vi komme nærmere i rapporten. Skulle projektet forklares til en studerende på 2. sem datamatiker, så ville dette være et simpelt program kørt fra IntelliJ, som generer en lokal hjemmeside. Hele programmet er sat op, så vi har kunne komme tættere på brugen af både Java, HTML, CSS og databaser.

Baggrund

“Olsker Cupcakes” er et iværksættereventyr fra Bornholm. Der menes her at man har ramt den helt rigtige opskrift. Der mangler dog en nem løsning for kunderne at kunne bestille og betale for cupcakes.

Teknologivalg

DataGrip 2024.2.2

DataGrip er en database-IDE udviklet af JetBrains, som giver avancerede værktøjer til at administrere og arbejde med forskellige databasesystemer. Den tilbyder funktioner som SQL-redigering, query eksekvering, visualisering af database struktur, og understøttelse af flere databaser, herunder PostgreSQL

Neon.tech (AWS Server, PostgreSQL 16)

Neon.tech er en moderne PostgreSQL platform på AWS, designet til at være skalerbar og nem at integrere med cloud-baserede applikationer.

intellij openjdk-23 23.0.1

IntelliJ IDEA er en populær IDE til Java-udvikling, som giver omfattende værktøjer til kodning, debugging og integration med forskellige teknologier og frameworks.

Figma version 116.14.4

Figma er et digitalt design- og samarbejdsværktøj, der bruges til at lave brugergrænseflader, prototyper og grafiske designs.

Javalin (io.javalin:javalin)
Javalin er et letvægts web framework baseret på Java og Kotlin, designet til at skabe RESTful web APIs. Det er simpelt og hurtigt, hvilket gør det velegnet til mindre applikationer eller mikroservices.

JBCrypt (org.mindrot:jbcrypt)
JBCrypt bruges til at hashe og verificere adgangskoder med crypt-algoritmen. Dette tilføjer et ekstra sikkerhedslag, når man arbejder med brugeroplysninger, ved at sikre, at adgangskoder lagres i et krypteret format.

SLF4J (org.slf4j:slf4j-simple)
SLF4J (Simple Logging Facade for Java) bruges til logging i Java-applikationer. slf4j-simple er en minimal implementering af SLF4J, som gør det muligt at logge til konsollen uden konfiguration kompleksitet.

Javelin Rendering (io.javelin:javelin-rendering)
Dette modul udvider Javelin til at understøtte rendering af views, såsom HTML-sider. Det integreres ofte med skabelon systemer som Thymeleaf for at kunne vise dynamisk genereret HTML-indhold i browseren.

Thymeleaf (org.thymeleaf:thymeleaf)
Thymeleaf er en skabelon motor til Java, der bruges til server-side rendering af HTML. Det gør det muligt at oprette dynamisk genererede websider baseret på data fra serveren.

Thymeleaf Extra - Java 8 Time (org.thymeleaf.extras:thymeleaf-extras-java8time)
Denne afhængighed udvider Thymeleaf til at understøtte Java 8's java.time API'er (f.eks. LocalDate og LocalDateTime), så datoer og tidspunkter kan håndteres og formateres korrekt i Thymeleaf-skabeloner.

Jackson Databind (com.fasterxml.jackson.core:jackson-databind)
Jackson bruges til at serialisere Java-objekter til JSON og deserialisere JSON til Java-objekter. jackson-databind er en central del af Jackson-biblioteket og understøtter konvertering af objekter til JSON-format, som ofte bruges til at bygge RESTful APIs.

PostgreSQL JDBC Driver (org.postgresql:postgresql)
PostgreSQL JDBC driver gør det muligt for Java-applikationer at kommunikere med en PostgreSQL database. Denne afhængighed giver adgang til SQL-databasefunktionalitet.

Dotenv for Java (io.github.cdimascio:dotenv-java)
Dotenv lader applikationen indlæse miljøvariabler fra en .env-fil, hvilket gør det nemt at konfigurere følsomme oplysninger, såsom adgangskoder og API-nøgler, uden at indarbejde dem direkte i koden.

Krav

Formålet med systemet til Olsker Cupcakes er at udvikle en æstetisk og brugervenlig webshop, hvor virksomhedens kunder let og hurtigt kan bestille skræddersyet cupcakes. Vores system optimerer bestillingsprocessen, og gør det let for kunderne at blande Olsker's forskellige smagsvarianter, og derefter hente deres bestilling. Derudover har Olsker også nu fået en digital tilstedeværelse, hvor håndteringen af deres ordrer og kundeadministration er let og effektiv.

Visionen med vores system er, at Olsker Cupcakes oplever en styrket konkurrenceevne, ved at være tilstede i den digitale verden. Systemet vil skabe en værdi hos Olsker, da kunderne vil opleve en større kundetilfredshed når de på nem vis er i stand til at se, bestille og redigere deres ordrer på Olsker's nye hjemmeside. Samtidig har vi forenklet Olsker's administrative arbejde, og derved frigjort tid de kan bruge på det der betyder noget, nemlig at lave cupcakes.

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.

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 ordrelinjer 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 mock up'en).

US-6: Som administrator kan jeg se alle ordrer i systemet, så jeg kan se hvad der er blevet bestilt.

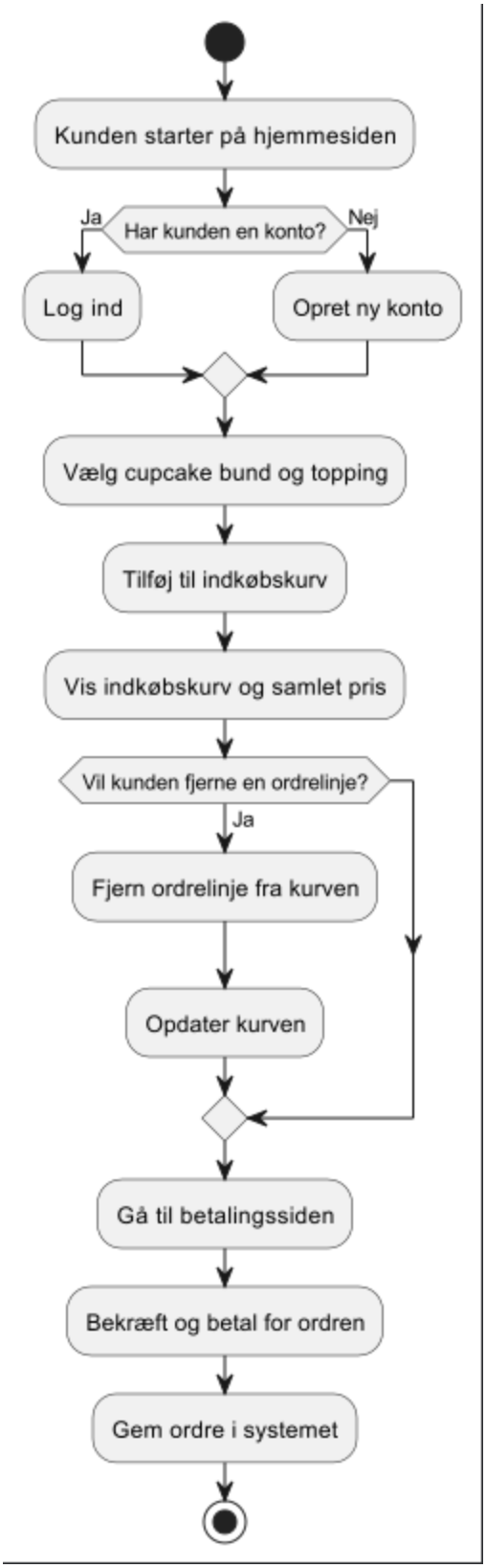
US-7: Som administrator kan jeg se alle kunder i systemet og deres ordrer, så jeg kan følge op på ordrer og holde styr på mine kunder.

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

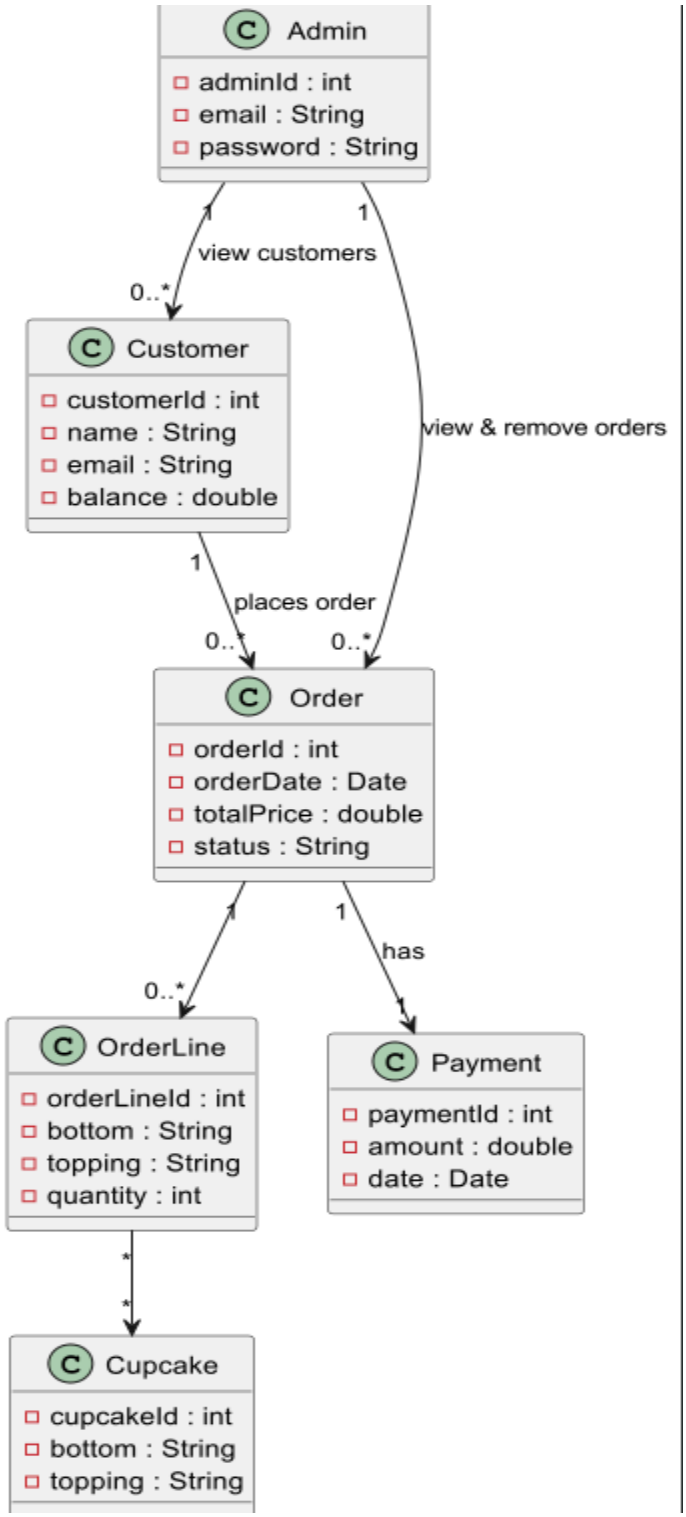
US-9: Som administrator kan jeg fjerne en ordre, så systemet ikke kommer til at indeholde ugyldige ordrer. F.eks. hvis kunden aldrig har betalt.

Aktivitetsdiagram

På næste side vises et diagram over aktiviteten i systemet.

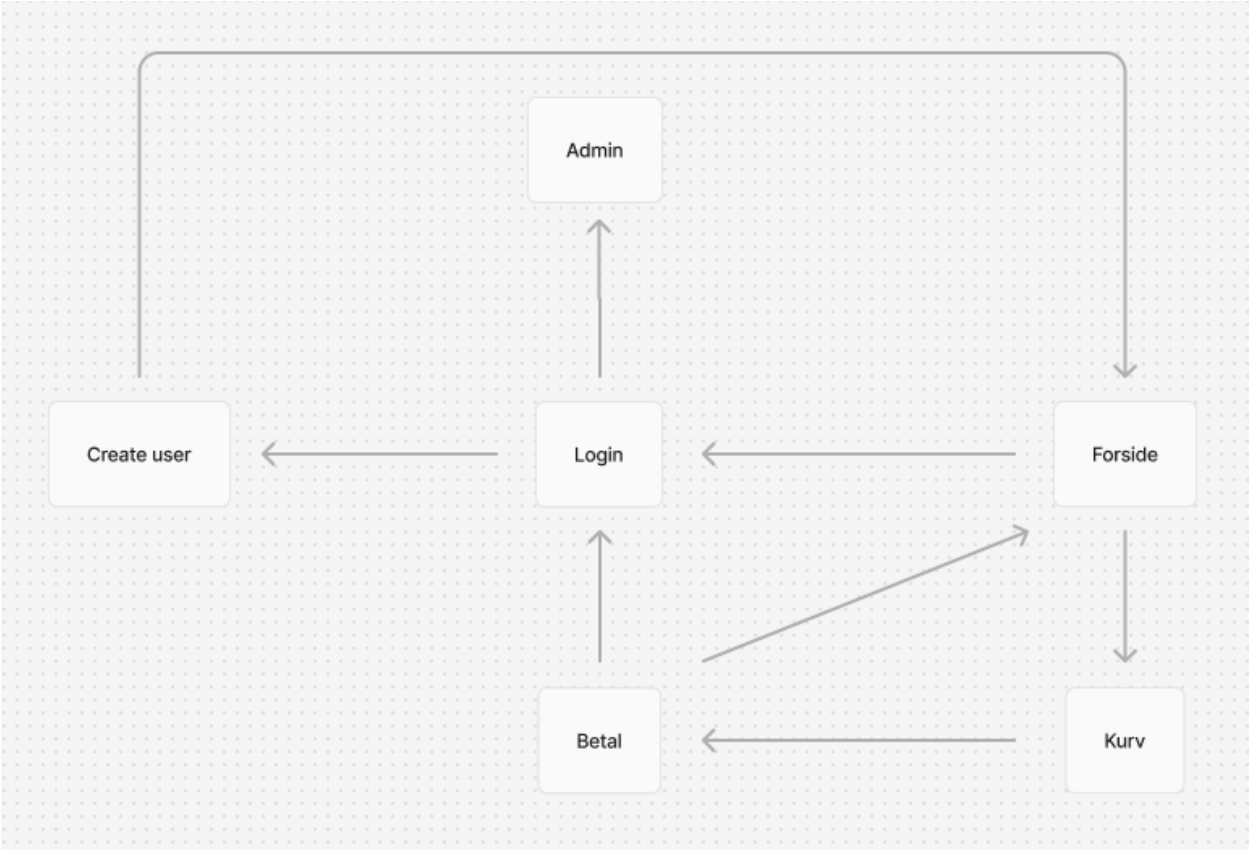

Kunden starter på hjemmesiden
Ja
Har kunden en konto?
Nej
Log ind
Opret ny konto
Vælg cupcake bund og topping
Tilføj til indkøbskurv
Vis indkøbskurv og samlet pris
Vil kunden fjerne en ordrelinje?
Ja
Fjern ordrelinje fra kurven
Opdater kurven
Gå til betalingssiden
Bekræft og betal for ordren
Gem ordre i systemet

Domæne model

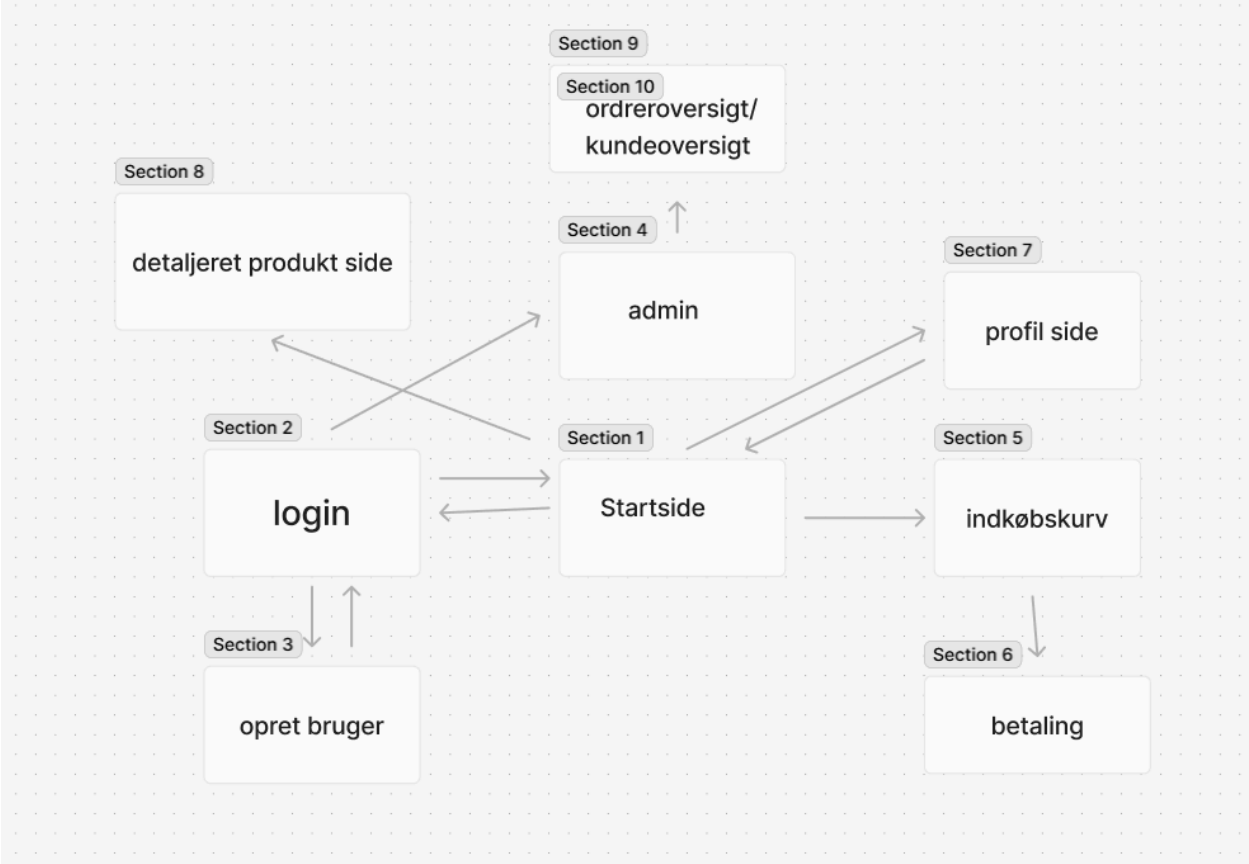


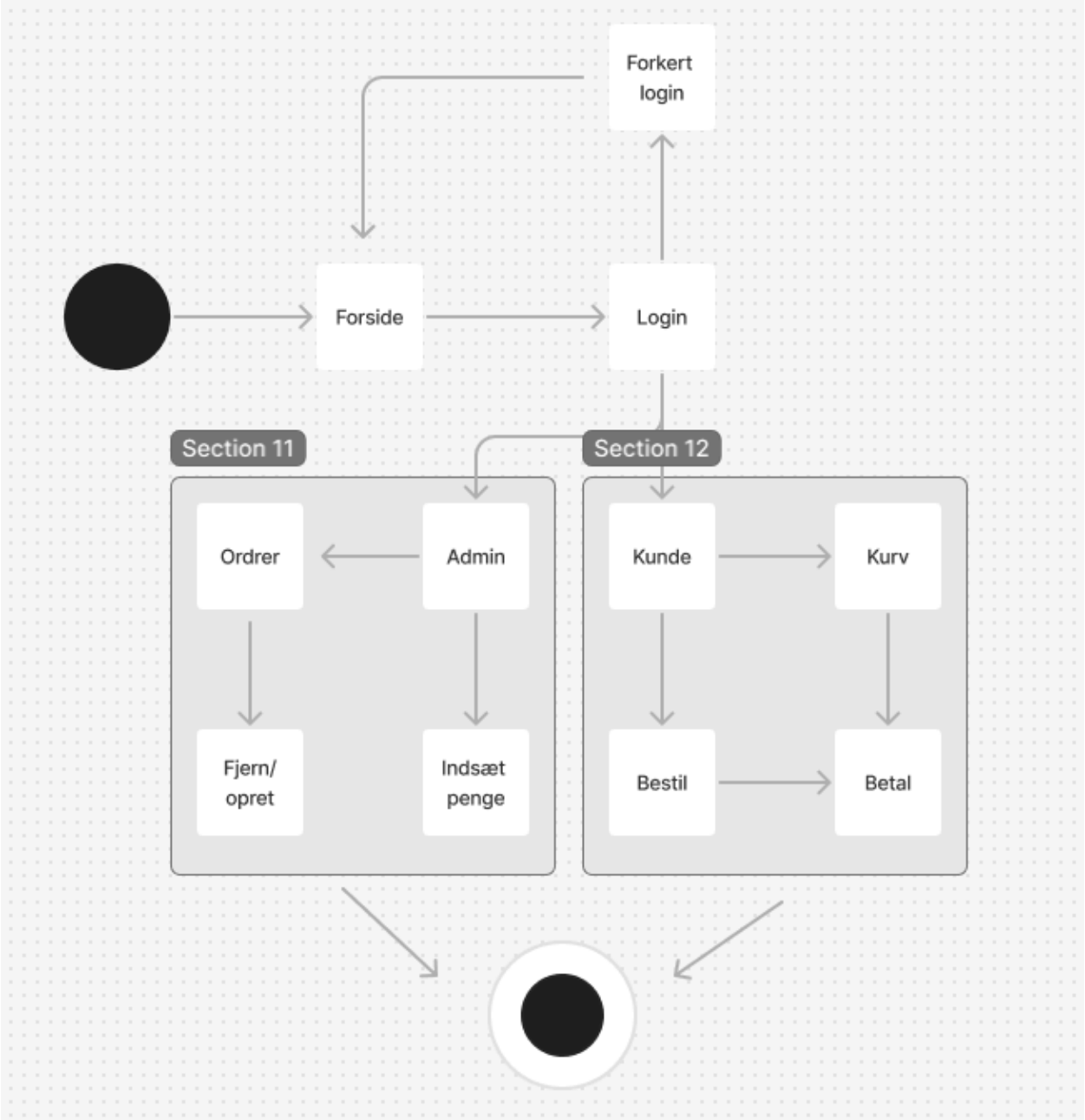
Navigationsdiagram

Her er et simpelt navigationsdiagram over siderne på vores system.



Tidligt i opgaven lavede vi også et simpelt navigationsdiagram, dette ser dog anderledes ud, end hvad vi er sluttet med.


Section 9
Section 10
ordreroversigt/
kundeoversigt
Section 8
detaljeret produkt side
Section 4
admin
Section 7
profil side
Section 2
login
Section 1
Startside
Section 5
indkøbskurv
Section 3
opret bruger
Section 6
betaling



Særlige forhold

Den modtager brugernavn og to adgangskoder fra HTML og sikrer først, at de to adgangskoder stemmer overens. Hvis adgangskoderne matcher, hashes adgangskoden ved hjælp af BCrypt for at øge sikkerheden, hvorefter metoden forsøger at oprette brugeren i databasen gennem createUser metoden i UserMapper klassen. Ved en succesfuld oprettelse informeres brugeren via en besked i ctx og omdirigeres til forsiden.

Hvis brugernavnet allerede findes, håndteres dette ved en undtagelse (Database Exception), som viser en fejlbesked, der beder brugeren prøve igen. Hvis adgangskoderne ikke stemmer overens, gives feedback til brugeren om fejlen, og vedkommende bliver bedt om at indtaste oplysningerne igen.

```
private static void createUser(Context ctx, DatabaseController databaseController) {
    String username = ctx.formParam( key: "username");
    String password1 = ctx.formParam( key: "password1");
    String password2 = ctx.formParam( key: "password2");

    if (password1 != null && password1.equals(password2)) {
        String hashedPassword = BCrypt.hashpw(password1, BCrypt.gensalt());

        try {
            UserMapper.createUser(username, hashedPassword, databaseController);
            ctx.attribute("message", "Du er nu registreret med brugernavnet: " + username + ". Du kan nu logge på.");
            ctx.redirect( location: "/");
        } catch (DatabaseException e) {
            ctx.attribute("message", "Brugernavn findes allerede. Prøv igen eller log in.");
            ctx.render( filePath: "createuser.html");
        }
    } else {
        ctx.attribute("message", "Kodeord stemmer ikke overens, prøv igen.");
        ctx.render( filePath: "createuser.html");
    }
}
```

Metoden login håndtere bruger login-processen ved at modtage brugernavn og adgangskode fra HTML Først forsøger metoden at hente brugeroplysningerne ved hjælp af login metoden i usermapper klassen , som returnerer en bruger, hvis brugernavnet findes i databasen. Derefter sammenlignes den angivne adgangskode med den lagrede adgangskode ved hjælp af BCrypt. Hvis adgangskoderne matcher, lagres brugerens information i en session attribut (current User), og brugeren omdirigeres til forsiden med en besked.

```
private static void login(Context ctx, DatabaseController databaseController) {
    String username = ctx.formParam( key: "username");
    String password = ctx.formParam( key: "password");

    try {
        User user = UserMapper.login(username, databaseController);


        if (BCrypt.checkpw(password, user.getPassword())) {
            ctx.sessionAttribute("currentUser", user);
            ctx.attribute("message", "Du er nu logget ind.");
            ctx.redirect( location: "/");
        } else {
            ctx.attribute("message", "Forkert adgangskode. Prøv igen.");
            ctx.render( filePath: "login.html");
        }
    } catch (DatabaseException e) {
        ctx.attribute("message", e.getMessage().contains("Bruger ikke fundet")
                ? "Brugernavn ikke fundet. Prøv igen."
                : "Fejl under login. Prøv igen.");
        ctx.render( filePath: "login.html");
    }
}
```

```
String orderDetailsJson = objectMapper.writeValueAsString(
        Map.of(
                k1: "bottom", bottom,
                k2: "topping", topping,
                k3: "quantity", quantity
        )
);
```

Her bliver Map brugt til at oprette ordre detaljerne som nøgle-værdi par, da det gør det nemt at tilføje nye detaljer. Object Mapper er fra Jackson biblioteket, og konverterer vores Map object til en JSON streng. Denne JSON streng repræsenterer ordre detaljerne som JSON, hvilket kan gemmes direkte i databasen. Herefter bliver orderDetailsJson indsat i databasen.

```
statement.setObject( parameterIndex: 3, orderDetailsJson);
```

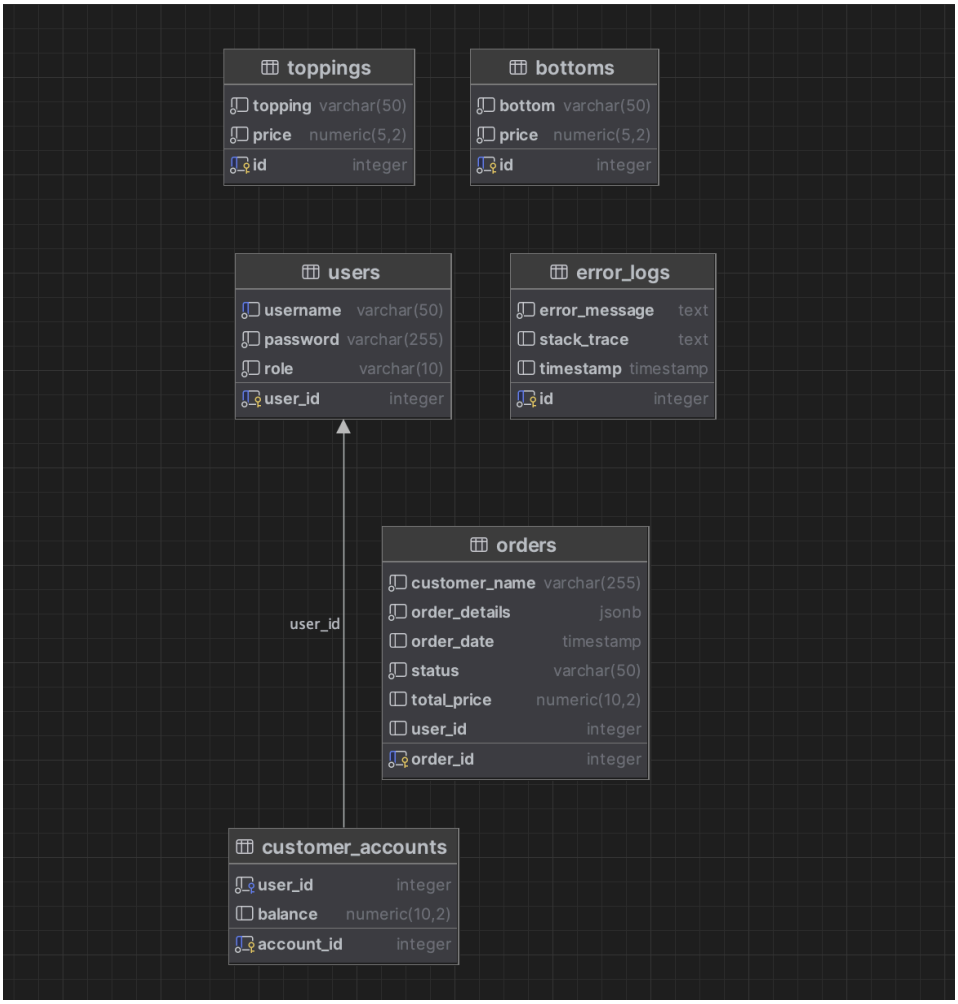
order_details kolonnen i databasen modtager JSON-strengen som konverteres til JSONb. Denne fremgangsmåde gør det muligt at gemme ordredetaljer i en enkelt kolonne, som derved kan udvides eller tilpasses uden ændringer i selve databasen.

Håndtering af vores exceptions

Vi har valgt, at vi gerne vil have et bedre overblik over alle mulige exceptions, som programmet vil kunne give os.

For at kunne sikre dette, har vi valgt at lave en klasse, der håndterer dette for os. Så i stedet for blot at printe vores exceptions ud i vores lokale klient terminal, bliver de nu gemt i vores fælles database. Dette valgte vi at gøre for, at vi senere vil kunne trække disse data ud og lave et reelt diagram over typiske fejl i vores program. Samtidig vil det også sikre, at vi ikke misser eventuelle fejl, der sker meget sjældent eller lignende. I og med at vores program i forvejen gør brug af “live” database, kunne vi samtidigt også nemt hente fejl fra alle lokale maskiner.

ERD Diagram



Her ses et ERD diagram over vores database, inde i vores *users* har vi *user_id* der automatisk bliver lavet af vores database, så vi sikre det altid er unikt per bruger der bliver oprettet. Vores table “customer_accounts” bruger også dette user_id for at skabe en relation mellem de to, det sikre at vi kan nogle bedre og langt mere effektive SQL-forespørgsler.

Implementation

Vi har implementeret alle user stories.

Man kan dog i vores system "betale" for en tom kurv, hvorefter du vil blive mødt af en bekræftelse på dit køb. Vi har opdaget det for sent, og dermed ikke kunne nå at løse problemet. Derudover har vi en unit test i PaymentService der fejler, det har vi heller ikke nået at få kigget på og løst.

Proces

Vores planer var at dele programmets User stories op mellem gruppemedlemmerne hvor vi individuelt arbejdede på dem. Dette gjorde vi ved at lave et kanban-board. Derudover havde vi også et mock-up, som var lavet i Figma, så vi kunne få et visuelt overblik over det kommende arbejde. Det forløb sig også nogenlunde sådan i praksis, da vi har været gode til at hjælpe hinanden undervejs, når der blev spurgt efter hjælp.
Vi har lært, at selvom vi har lavet en plan og skabt os et overblik over programmet, så sker der stadigvæk uforudsete ændringer, da det færdige produkt blev anderledes i forhold til vores vision. Derfor har det været utrolig positivt for vores samarbejde, at vi har kunne tale sammen og hjælpe hinanden med de udfordringer vi har mødt. Vi har også lært, at vi har brug for at sætte realistiske deadlines og delmål til næste gang. Dette kunne have været med til at skabe en bedre oversigt over projektet. Dette kunne måske løses ved at lave stand-up meetings eller gøre endnu bedre brug af vores kanban-board, så vi har en klar kommunikation om hvor langt vi er med vores forskellige dele af projektet. Derudover kunne vi også undgå misforståelser eller ekstra arbejde, hvis vi havde haft en aftale om hvilke metoder og metodenavne vi skulle bruge i vores klasser.

Kanban, Figma-mock-up, git/github,

Bedre til næste gang: sætte realistiske deadlines, delmål. Klar kommunikation/planlægning
Link til kanban: https://github.com/users/ValdemarLarsen02/projects/2

Link til gennemgang af hjemmesiden: https://youtu.be/0it2QrKvmgM