CUPCAKE-projekt

April 2024

Bornholm Gruppe 4.

Rapport udarbejdet af:

Patrick Alexander Kempf

cph-pk217@cphbusiness.dk

Github - obigonemad

Bornholm

Simone Riis Skovgaard

cph-sr138@cphbusiness.dk

Github - simoneskovgaard

Bornholm

Jaqueline Schwencke Overgaard

cph-jo215@cphbusiness.dk

Github - FruSchwencke

Bornholm

Indholdsfortegnelse

Indledning

Olsker Cupcakes er en virksomhed som specialiserer sig i cupcakes. Deres cupcakes kan afhentes ved deres bageri i Olsker. De ønsker et system, hvor deres kunder kan bestille cupcakes via deres hjemmeside. For at bestille cupcakes, skal man være oprettet som kunde i deres system. Kunden skal både kunne vælge bund og topping på deres cupcakes, og se hvad den samlede pris for deres bestilling er. Derudover skal administratoren af virksomheden have mulighed for at logge ind og tilgå forskellige oplysninger om ordrerne i systemet.

Teknologivalg

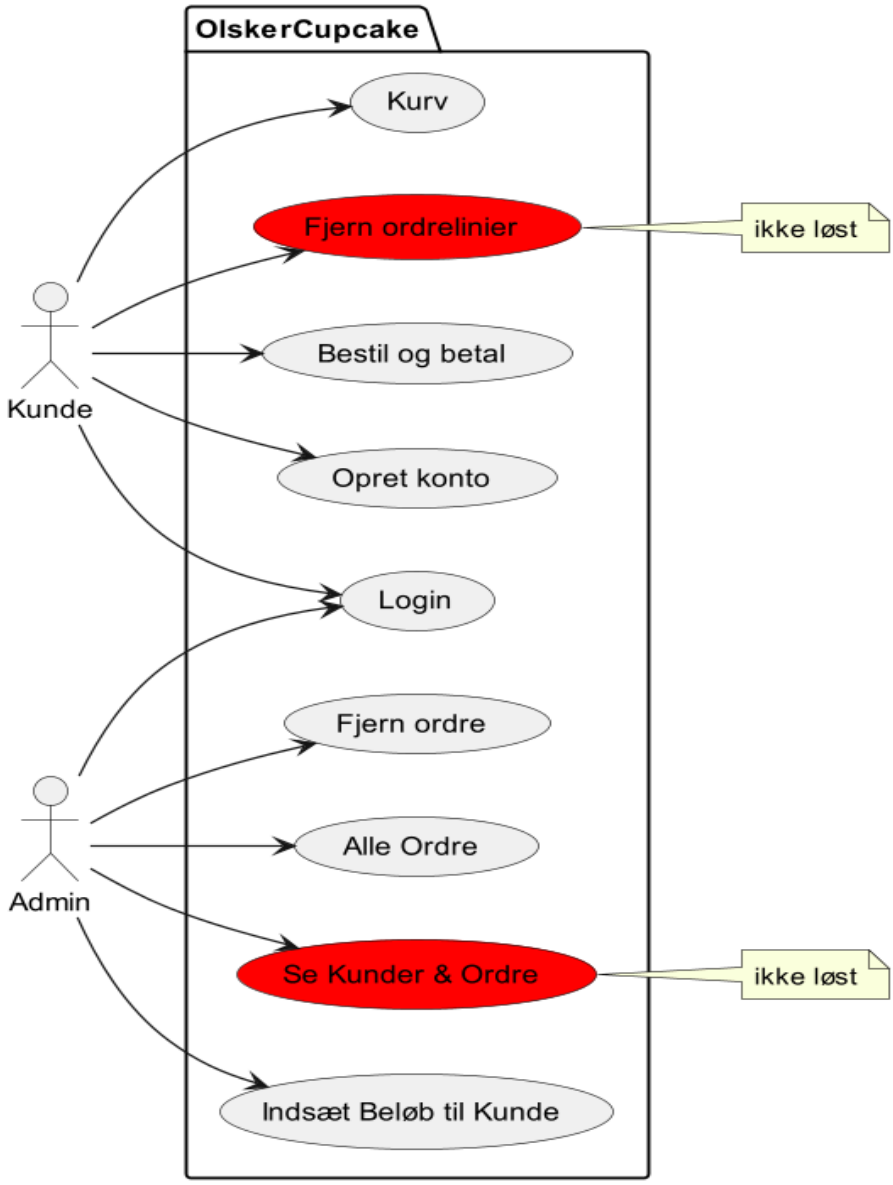
Olsker Cupcake er bygget til at køre på en Docker Desktop (4.28.0) og er udviklet i Java , Javalin, Thymeleaf template engine, Postgres Database, HTML og CSS.

Navn	Version
intellij	2023.3.4
maven	4.0.0
SDK	17
javalin.version	6.1.3
thymeleaf.version	3.1.2.RELEASE
hikariCP.version	5.1.0
junit.version	5.10.2
postgresql.version	42.7.2

Krav

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. Endvidere har vi udarbejdet et Usecase diagram, med det formål at demonstrere interaktionen med hjemmesiden.

UseCase Diagram



User-Stories

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 e-mail 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.

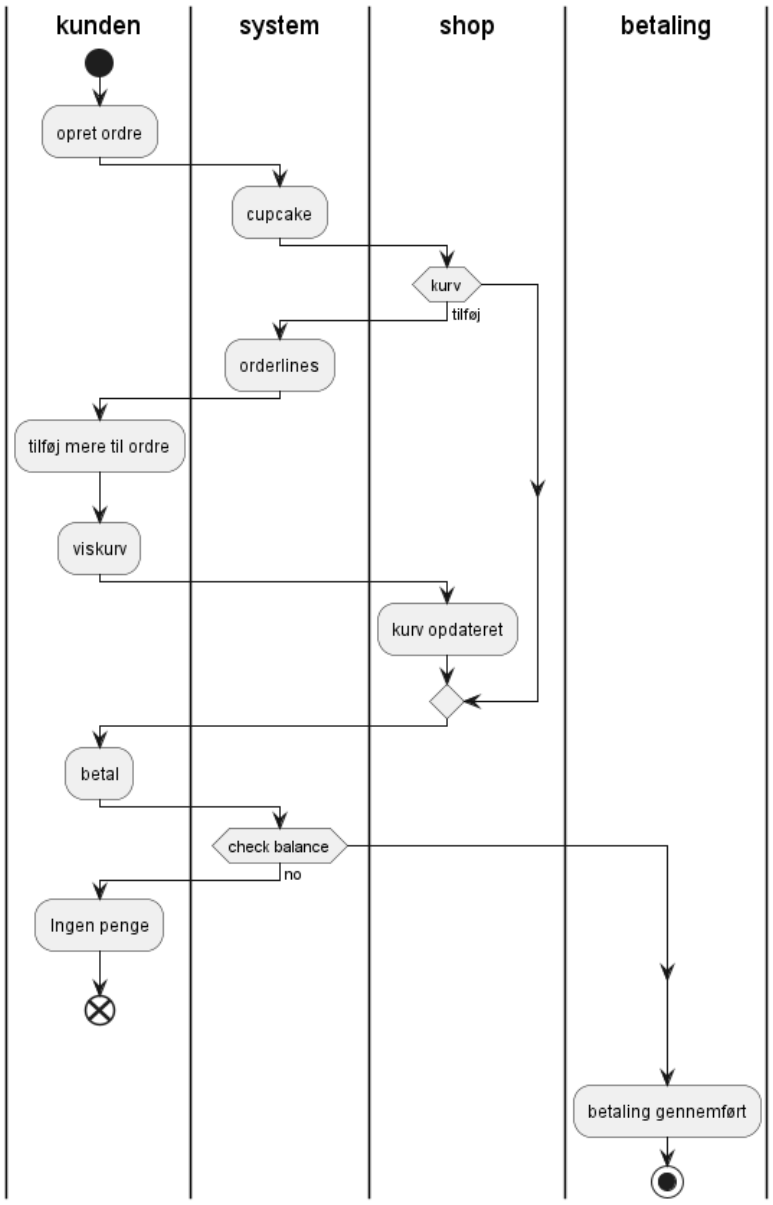
US-7: Som administrator kan jeg se alle kunder i systemet og deres ordrer, så jeg kan følge op på ordrer og holde styr på mine kunder.

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

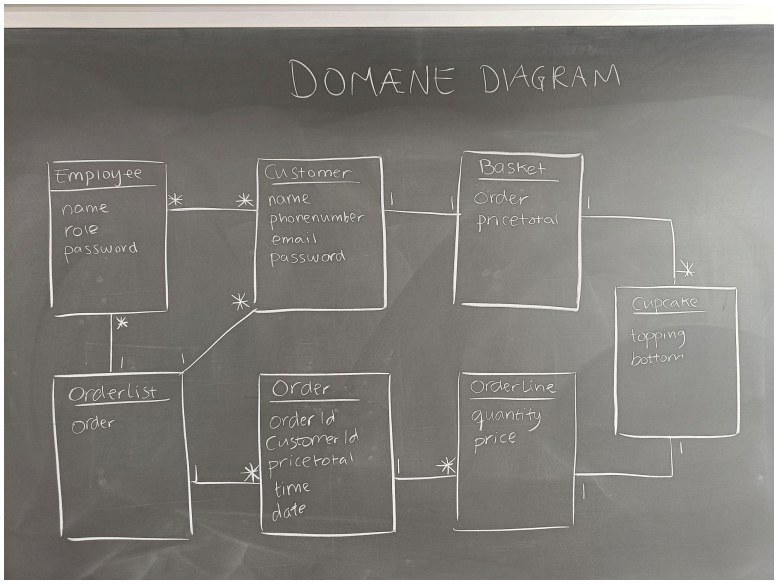
US-9: Som administrator kan jeg fjerne en ordre, så systemet ikke kommer til at indeholde ugyldige ordrer. F.eks. hvis kunden aldrig har betalt.

Aktivitetsdiagram

Gruppen har udarbejdet et aktivitetsdiagram, ved hjælp af plantUML i Intellij. Dette diagram har til formål, at visualisere flowet på hjemmesiden, når man ønsker at købe en cupcake hos virksomheden.



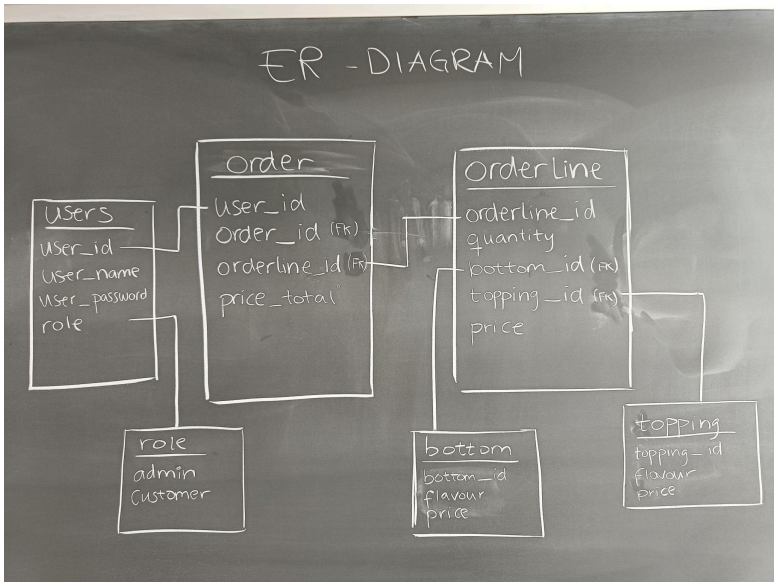
Domæne model og ER diagram



Dette domæne diagram repræsenterer det første arbejde med Olsker Cupcakes. Her ses de indledende tanker om hvilke klasser vi potentielt skulle arbejde med og de relationer vi så nødvendige.

De relationer vi havde flest overvejelser omkring var "Order", og hvordan relationen til "Basket" skulle være. Vi besluttede, at "Basket" skulle have relation til "Customer", da det er en kunder der har en kurv til at holde sit indkøb, samt en relation til "Cupcake", for det er de entiteter Olsker Cupcakes udelukkende specialiserer sig i, og her har vi sat relationen 1-M, da vi må forvente, at der skal kunne være mere end én cupcake i kurven .
Vi endte ud med, at "Basket" derfor kunne skulle indeholde attributterne "Order" og "Pricetotal", hvor "Cupcake", "Orderline" og "Order" skulle holde de attributter, vi så som specifikke detaljer for en samlet ordre.

ER diagram



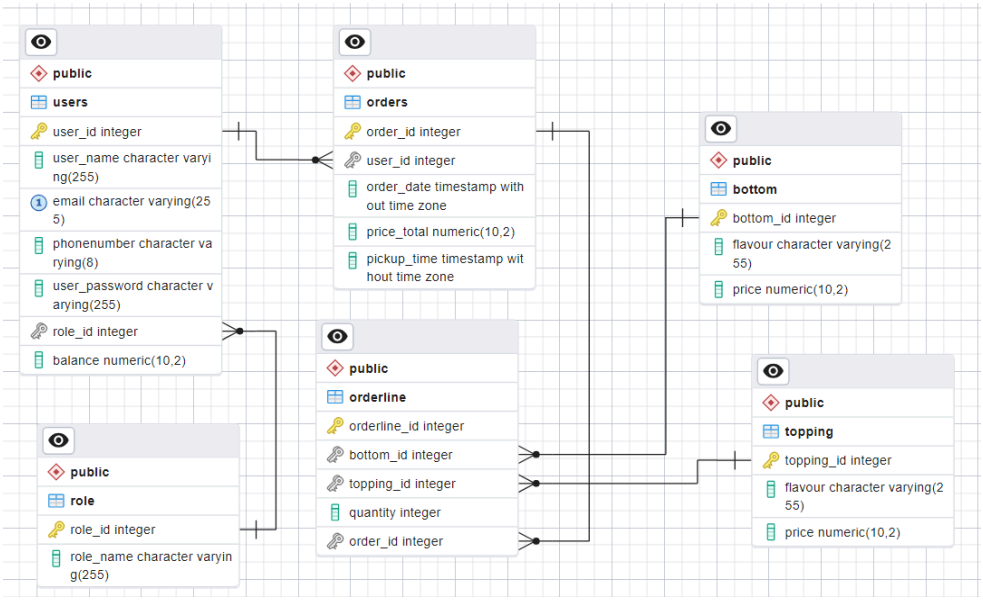
Dette ER-diagram viser det videre arbejde ved ovenstående domæne diagram. Her er attributterne blevet udvidet ved begyndende overvejelser omkring navngivning i databasen og hvor vi så potentielle foreign keys. I første omgang satte vi foreign keys på "order" og "orderline".

Herunder ses ER-diagrammet for programmets nuværende udviklingstrin, og en gennemgang af diagrammet ud fra de tre første normalformer.

Første normalform

ER-diagrammet er i første normalform, da hver tabel indeholder hver deres nøgle der identificerer dem. Ligeledes er der ikke flere værdier, der definerer én attribut, hvilket betyder, at felterne er atomare værdier. Til sidst opfyldes også, at der er ingen gentagne kolonner i hver tabel.

Anden normalform



Da tabellerne allerede lever op til første normalform, er der, for anden normalform, også at der ikke må være attributter, der ikke selv tilhører nøglen, som afhænger af dele af nøglen. Hvilket vi mener, der ikke er. Alle klasser indeholder foreign keys, der hvor vi mener at attributter skal være en del af en anden klasse. Dermed er netop disse attributter ikke afhængige af klassen primærnøgle, men har sit eget referencepunkt.

Tredje normalform

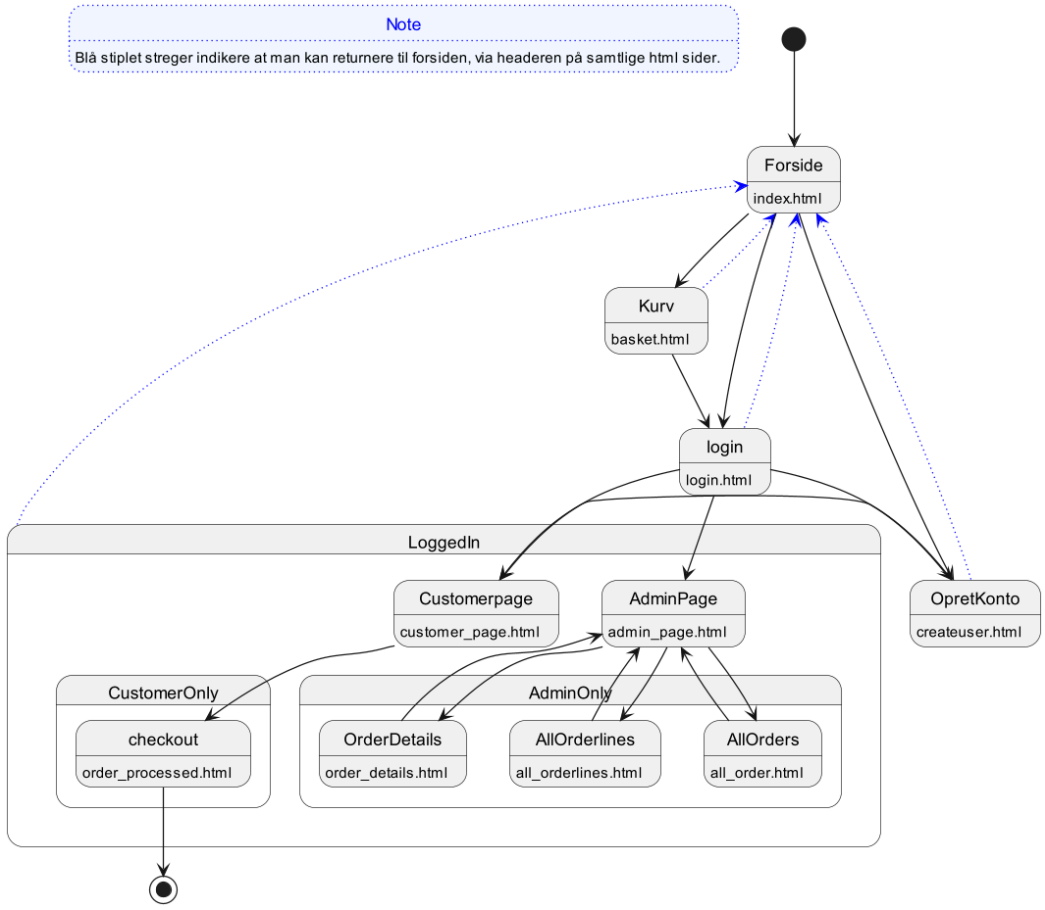
Første og anden normalform er opfyldt og herefter vurderes om tabellerne er fuldt i tredje normalform, ved at kigge efter om der er data, der ikke afhænger af primærnøglen, men er i en form for afhængighedsforhold til andet data i tabellen, som ikke er nøgle data. Det mener vi ikke er tilfældet for vores tabeller, og dermed er tredje normalform opfyldt.

Refleksioner

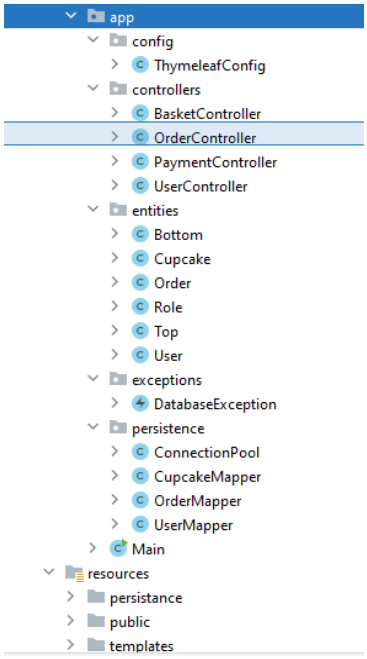
"Users" har relationen 1-M til "orders", denne relation baserer sig på, at vi forventer at en user potentielt kan have mere end én ordre. Fra "orders" ses relationen 1-M "orderline", fordi vi forventer, at en bestilling skal kunne indeholde mere end én ordrelinjer med udvalgte cupcakes. En ordrelinje er et samlingspunkt for det antal cupcakes, som kunden ønsker med en valgt bund og topping. Klassen "orderline" indeholder derfor flere foreign keys med attributter fra andre klasser.

Det der kunne gøre, at vores tabeller ikke er i tredje normalform, ville fx være hvis vi også ville have kundens adresse, hvilket ville udløse et postnummer, som er tæt forbundet med en bestemt by, hvilket danner et afhængighedsforhold udover den primærnøgle klassen har.

Navigationsdiagram



Særlige forhold



Opbygning af system

Her ses et overblik over, hvordan vores system er bygget op. Vi har valgt at benytte os af mappers og controllers - MVC.

Model - view - controller.
På den måde har vi skrevet vores metoder, der skal hente fra databasen i vores persistence package, vi har brugt controllers til at kontrollere hvilke attributer der bliver sendt videre, og kan ses i vores HTML. Vi har benyttet os af Thymeleaf.

Vi er opmærksomme på, at vores Controller-klasser er skrevet på forskellige måder, og at det dermed kan fremstå mere uoverskueligt at forstå vores kode.

I dette program vil vi fremhæve et par ting; det ene er brugen er "ctx.sessionAttribute" og "ctx.attribute" i vores controllere, og vores "checkBalance"-metode for, at demonstrere validering ved et brugerinput.

I programmets controllere er der forskellige brug af Javalins context-metoder "sessionAttribute" og "attribute".

```
public static void addRoutes(Javalin app, ConnectionPool connectionPool) {
    app.get( path: "basket", ctx -> {

        //initializing a current basket-list
        List<Cupcake> basketList = ctx.sessionAttribute( key: "basketList");

        // getting the sum of all orderlines by using totalPrice
        double sum = basketList.stream().mapToDouble(Cupcake::getPrice).sum();


        ctx.sessionAttribute("basketList", basketList);
        ctx.attribute("basketList", basketList);
        ctx.sessionAttribute("sum", sum );
        ctx.attribute("sum", sum );
        ctx.render( filePath: "basket.html");

    });

    app.get( path: "order_details.html", ctx -> showOrderLines(ctx, connectionPool));
    app.get( path: "/allOrderlines", ctx -> showAllOrderlines(ctx, connectionPool));
    app.get( path: "allOrders", ctx -> getAllOrders(ctx, connectionPool));
    app.get( path: "/adminback", ctx -> ctx.render( filePath: "admin_page.html"));
    app.post( path: "/deleteOrder", ctx -> deleteOrder(ctx, connectionPool));
}
```

Dette er sket på baggrund af manglende research herom grundet tidspres.
Det vi ved nu er, at "ctx.attribute" håndtere data der er forbundet med et enkelt HTTP-requests. Når HTTP-forespørgslen er besvaret, forsvinder dataen.

Og "ctx.sessionAttribute" håndterer data i forbindelse med en specifik bruger session. Hvor denne data persisterer hen over mere end én HTTP-request.

Derfor skal vi overveje hvilken data der skaber brugervenlighed, men ikke behøver at persistere over flere request, dette kunne fx være udregninger på cupcake priser.

Og hvilke data der er behov for at persistere hen over hele brugerens session, såsom login-informationer og ordrehistorik.

Den anden ting vi vil fremhæve er "checkBalance"-metoder og et udklip af "PaymentController".

Herunder ses udklippet af "PaymentController", hvor der er to if-statements, det første validerer om brugeren eksisterer i databasen, det næste validerer om useren har en balance tilknyttet sin konto.

```
if(user != null ){
    // user.getBalance()
    if (UserMapper.checkBalance(user.getUserId(), connectionPool) >= sum){
        //sending order to DB if user exists
        OrderMapper.createOrder(user, basketList, sum, connectionPool);
        ctx.render( filePath: "order_processed.html");
    }else {
        System.out.println("Du har ikke penge nok");
        //TODO: the user should be notified by the pay button of insufficient funds and not be directed to index page
        ctx.render( filePath: "index.html");
    }
}else ctx.render( filePath: "login.html");
```

"checkBalance"-metoden, er den vi kalder fra "PaymentController", metoden ses her til højre, og er en metode, der er nødvendig i applikationer som denne, da disse tjeks sikre, at der ikke bliver bestilt under forudsætning, at kunden har tilstrækkelig midler, og at der ikke er flere sessioner åbne med den samme bruger, og der dermed kan ske flere bestillinger på midler der allerede er brugt.

```java
public static double checkBalance(int userId, ConnectionPool connectionPool) throws DatabaseException {
    String sql = "SELECT balance from users WHERE user_id = ?";

    try (Connection connection = connectionPool.getConnection();

         PreparedStatement ps = connection.prepareStatement(sql)) {
        ps.setInt( parameterIndex: 1, userId);

        ResultSet rs = ps.executeQuery();
        if (rs.next())
        {
            double balance = rs.getInt( columnLabel: "balance");

            return balance;
        } else
        {
            throw new DatabaseException("fejl");
        }

    } catch (SQLException e) {
        String msg = "Der er sket en fejl.";
        throw new DatabaseException(msg, e.getMessage());
    }
}
```

US-2: Opret profil.

- Email i login, har et "@", som simpel validation. Gruppen er dog opmærksom på, at man ved denne simple validering vil kunne gøre brug af uhensigtsmæssige specialtegn m.m, ved login eller oprettelse af konto. Email er endvidere gjort unikt i databasen, såfremt at der ikke kan oprettes andre konti med en eksisterende email.
- Der er ikke nogle sikkerhedskrav til kodeord når man opretter en bruger, andet end at det skal være identisk med det første. Dog kan det være 255 tegn langt.
- Telefonnummer har ikke nogle krav, dog lavet til Unique i databasen og har max 8 tegn.
- Når man logger ind, vil man blive dirigeret til en tilhørende side, alt efter om du er Admin eller kunde.
- Når man hvis man fejler under login, vil man blive mødt af en "fejl i login" - dette gjort, for at man ikke ved om det er email eller password, hvori fejlen ligger.
- Logout funktionen var ikke en del af user stories, men vi har valgt at implementere efter et kort opkald med virksomheden, som fandt det særdeles brugbart på deres hjemmeside.

US-3: Indsæt beløb i postgres.

Vi har valgt at lave en metode addBalance som indsætter penge på kundes konto, da vi tænker det vil gøre det nemmere for admin. Denne metode kan kun tilgås fra admins side, og der kræver at admin kender user_id fra databasen.

Tid og dato:

Da vi ikke har nået at få med i vores metoder, at ordrerne skal have en afhentningsdato, har vi valgt at sige, at alle ordrer blot bliver bestilt uden nogen specifik dato og tid. Det er hverken nævnt i user stories eller som krav, at administrator skal kunne se, hvornår at ordrerne er bestilt til. Havde vi haft tid til det, ville det selvfølgelig være ideelt, at hver ordre havde en afhentningsdato. I stedet har vi valgt at tilføje et notat til kunden på vores index.html side, som indikerer, hvornår en ordre skal være foretaget, for at få den dagen efter. På den måde kan admin hente alle ordrer på det tidspunkt, for at se hvilke cupcakes der er i systemet.

Da vi har opfyldt US-9 og givet admin mulighed for at slette ordrer, kan vedkommende derfor sørge for, at alle ordrer der er lavet/afhentet kan slettes, inden vedkommende henter de nye ordrer dagen efter.

Status på implementation

Style:

Vores fokus i denne opgave har ikke været at få stylet vores sider. Vi har lavet et stylesheet, hvor en smule css er anvendt, men vi er opmærksomme på, at stylingen hverken er optimal i forhold til udseende eller brugervenlighed.

Vi har i vores forløb ikke benyttet os af at lave en Mock up i Figma. Det undlod vi fra start af, da vi ikke havde et klart overblik over hvordan vi ville bygge vores html sider op. Havde vi haft en Mock up i Figma, havde det muligvis gjort det nemmere at style løbende og efterfølgende.

Exceptions:

Der er flere af vores metoder, hvor vi ved at Exceptions ikke bliver håndteret korrekt. Vi er opmærksomme på, at dette kan skabe problemer, men vi har fokuseret på at få systemet til at virke. Planen var at håndtere exceptions senere, men det er ikke alle metoder vi har haft tid til at gennemgå.

Hvis man forsøger at gennemføre en betaling, men ikke har nok penge på kontoen, bliver man på nuværende tidspunkt ført tilbage til index.html. For at løse dette problem kunne vi have håndteret det i exceptions og givet brugeren besked om at balancen ikke er høj nok.

Fejl i metoder:

Der er fejl i vores betalingsdel, det har vi kun opdaget efter kodestop. Der er blevet lavet en metode som tjekker om brugeren har penge på sin konto, men når betalingen går igennem, bliver der ikke trukket penge fra kontoens balance, dette skulle selvfølgelig være rettet.

Generelle mangler:

- Vi har i denne opgave ikke benyttet os af Unittest. Vi har fokuseret på at få opfyldt flest mulige krav, frem for at benytte tiden på unittest. I vores tilfælde har det virket hurtigere at teste manuelt om systemet virker. Vi er opmærksomme på, at vores system muligvis kan have fejl og mangler, som kunne være opdaget ved at teste løbende.
- Vi mangler at få løst US-7 og US-8.
- Vi har i denne opgave ikke lavet en integrationstest af vores database.
- Der er kun lavet et aktivitetsdiagram, vi mangler at lave for admin-siderne, login/create-user m.m.
- Når man er logget ind, bliver Email kun vist i navigationenbaren på: index.html admin_page.html og customer_page.html

Proces

Vi har været udfordret ift at mødes, da et medlem skulle på ferie og grundet job ved siden af studiet. Derfor har vi brugt githubs kanban-board til at uddele opgaverne, og vi har skrevet rapporten i et google docs dokument, som man løbende har kunnet tilføje.

2 ud af 3 medlemmer har arbejdet ud fra branches, 3 medlem havde problemer når der blev opdateret, så auto merged den ind til main.

I forhold til det indledende forarbejde med diagrammer og opbygning af vores database har vi kunnet mærke, at vi ikke har haft nok tid sammen til at få det hele på plads. Det har blandt andet gjort, at vores database ikke var korrekt fra start. Dette har skabt små problemer, når vi har skrevet vores metoder. Vi har derfor været nødsaget til at ændre i vores database løbende. Ved næste projekt vil vi bruge mere tid på dette.

Vi har fra starten af haft en klar aftale om, hvordan vi navngiver. Både i java og i databasen. Dette har gjort det nemt for os alle, at få metoderne til at hente fra databasen uden misforståelser. En ting vi ikke fik aftalt på forhånd, var hvordan/om vi ville kommentere vores kode. Derfor er der både metoder med danske og engelske kommentarer og metoder uden kommentarer, og som nævnt tidligere er vores Mappers og Controllers ikke opbygget ens.

Videolink

https://youtu.be/hlHL5D_ePaw