Cupcake - Rapport

Cupcake gruppe D:

Lukas K. Simonsen, cph-ls491@cphbusiness.dk, Montif16

Lasse Erik Jensen, cph-lj481@cphbusiness.dk, Lorsayden

André Samuelsen, cph-as760@cphbusiness.dk, AndyTheDragon

Idris Azmi Ata Isci, cph-ii38@cphbusiness.dk, mridrisisci

Madiha Khan, Github: Madihakhan1 cph-mk871@cphbusiness.dk

Dato:

30/10/2024

Indholdsfortegnelse

Indledning

Projektet omhandler udviklingen af en webapplikation til Olsker Cupcakes, en dybdeøkologisk cupcake-virksomhed fra Bornholm. Applikationen skal hjælpe virksomheden med at administrere og præsentere deres produkter online. Vores målgruppe for denne rapport er vores fagfælle, en datamatikerstuderende på 2. semester, som ikke kender til opgaven på forhånd.

Projektet startede med en indledende kravindsamling og en halvfærdig mockup fra to københavnske hipstere, der har hjulpet virksomheden med at finde frem til nogle grundlæggende idéer til design og funktionalitet. Mockuppen er en skitse af forsiden, som angiver en vis retning for projektet, men den mangler detaljeret funktionalitet. Vores opgave er derfor at bearbejde kravene, stille uddybende spørgsmål til manglende dele, og levere en applikation, der afspejler både kundens behov og brugervenlige funktioner.

For at udvikle applikationen effektivt vil vi arbejde i etaper, med fokus på hurtigt at kunne levere en brugbar prototype. Dette indebærer at prioritere funktionaliteterne og løbende udvikle systemet, så vi kan sikre, at de mest nødvendige dele fungerer korrekt og imødekommer kundens ønsker, inden vi tilføjer yderligere funktioner.

Baggrund

Virksomhedsbeskrivelse:
Olsker Cupcakes er en dybdeøkologisk cupcake-virksomhed baseret på Bornholm. Virksomheden fokuserer på kvalitet og bæredygtighed i deres bagværk og tilbyder unikke, tilpassede cupcakes med forskellige muligheder for bund og top, så kunderne kan få en skræddersyet smagsoplevelse.

Overordnede krav til systemet:
Kunden ønsker et system, hvor deres kunder kan bestille cupcakes online med mulighed for at vælge forskellige bunde og toppe. Systemet skal også håndtere betaling og give kunderne mulighed for at oprette profiler for nemmere bestilling i fremtiden. Samtidig skal der være et administrativt overblik, så medarbejdere kan følge med i alle bestillinger og kunder i systemet.

Teknologivalg

Projektet er udviklet med følgende teknologier:

- **Java 22**: Hovedsprog for applikationens backend.
- **Javalin (version 6.3.0)**: Webframework til at håndtere HTTP-forespørgsler og responses.
- **Thymeleaf (version 3.0.15)**: Template engine til at generere dynamiske HTML-sider.
- **PostgreSQL (version 14)**: Database, der håndterer lagring af data om kunder, ordrer og produkter.
- **JDBC**: Anvendes til at etablere forbindelse mellem Java-applikationen og PostgreSQL-databasen.
- **HTML5 og CSS3**: Bruges til at designe og strukturere webapplikationens brugergrænseflade.
- **Bootstrap (version 5.3.3)**: Responsivt CSS (og JavaScript) framework, som giver en grundlæggende styling og funktionalitet der kan forventes af et moderne website.
- **IntelliJ IDEA 2024.2.4 (Ultimate)**: IDE til udvikling og debugging af projektet.

Disse teknologier understøtter udviklingen af en pålidelig og dynamisk webapplikation, som opfylder kundens behov for tilpasset bestilling og administrativt overblik.

Krav

Virksomhedens vision for systemet:

Olsker Cupcakes ønsker et digitalt system, der kan effektivisere bestillingsprocessen og forbedre kundeoplevelsen. Ved at tilbyde kunderne en enkel og tilpasset måde at bestille cupcakes online – med mulighed for at vælge specifikke bunde og toppe – håber virksomheden at øge salget og skabe en mere personlig relation til kunderne. Samtidig ønsker de et administrativt overblik over ordrer og kunder, hvilket vil gøre det nemmere at følge op på bestillinger og kundekonti. Systemet skal derfor tilføre værdi gennem øget tilgængelighed, bedre kundeoverblik og en mere effektiv intern håndtering af ordrer.

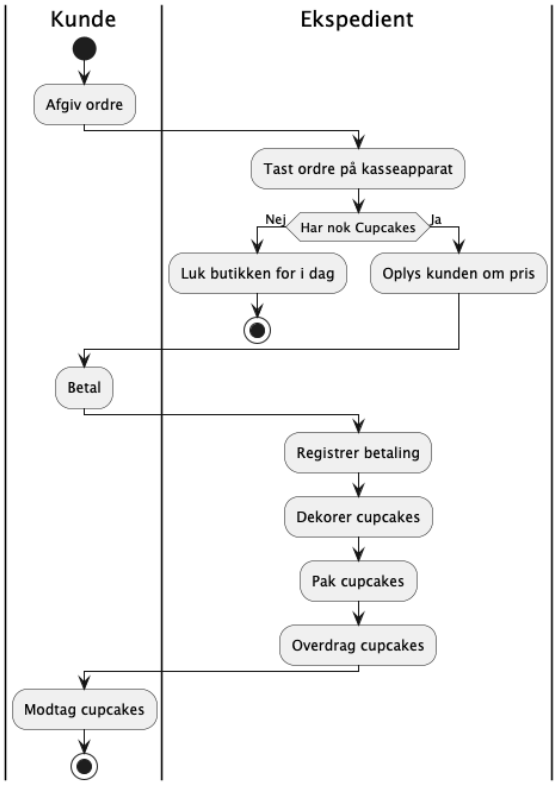
User Stories:

1. Som kunde kan jeg bestille og betale cupcakes med en valgfri bund og top, så jeg kan hente min ordre i Olsker.
2. Som kunde kan jeg oprette en konto/profil for at betale og gemme en ordre.
3. Som administrator kan jeg indsætte beløb på en kundes konto direkte i databasen, så kunden kan betale for sine ordrer.
4. Som kunde kan jeg se mine valgte ordrelinjer i en indkøbskurv og den samlede pris.
5. Som kunde eller administrator kan jeg logge ind med email og kodeord og se min email på hver side.
6. Som administrator kan jeg se alle ordrer for at få overblik over bestillingerne.

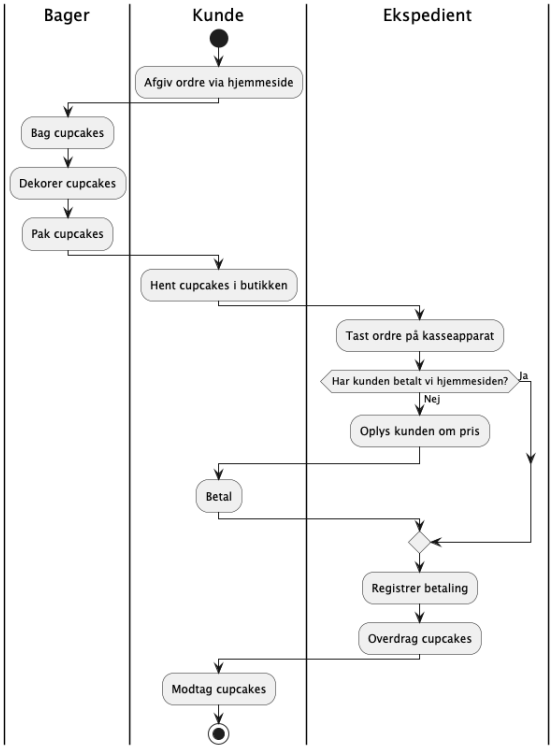
7. Som administrator kan jeg se alle kunder og deres ordrer for at kunne følge op og holde styr på kunderne.
8. Som kunde kan jeg fjerne en ordrelinje fra min indkøbskurv for at justere min ordre.
9. Som administrator kan jeg fjerne en ordre, f.eks. hvis kunden ikke har betalt.

Aktivitetsdiagram

AS-IS



TO-BE

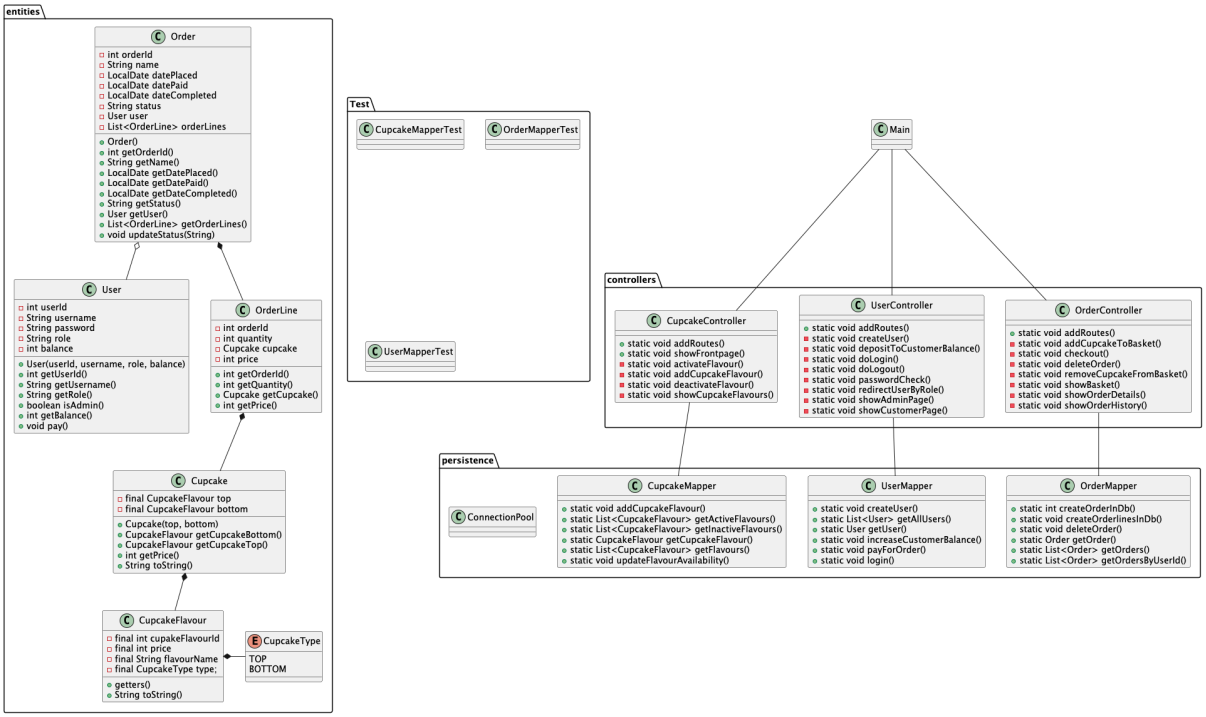


Domænemodel og ER diagram

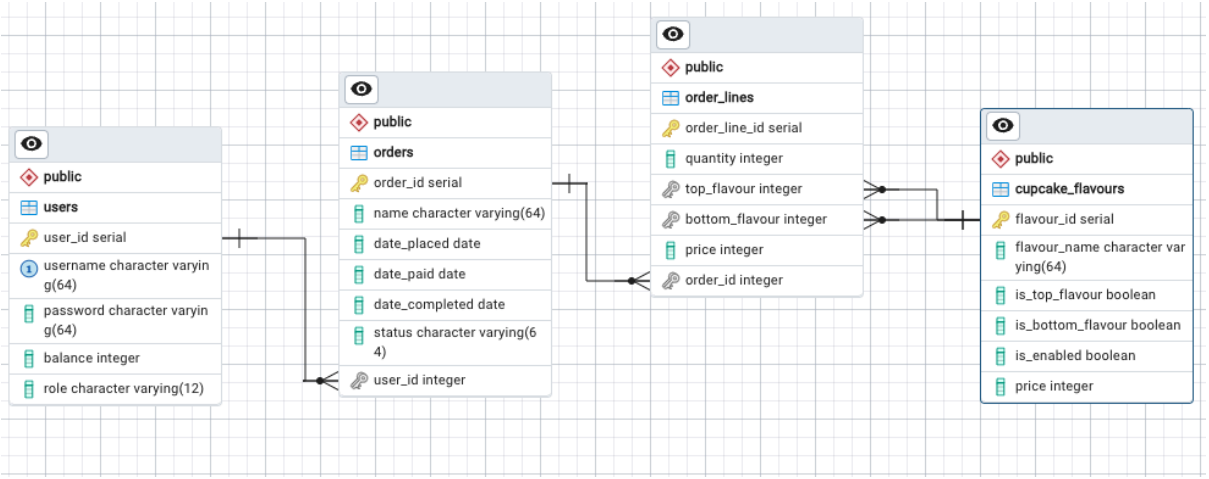
Domæne model

Vi har ikke udarbejdet en domænemodel, men i stedet udarbejdet et klassediagram og et ER-diagram ud fra de 9 User Stories.

Klassediagram



ER-Diagram



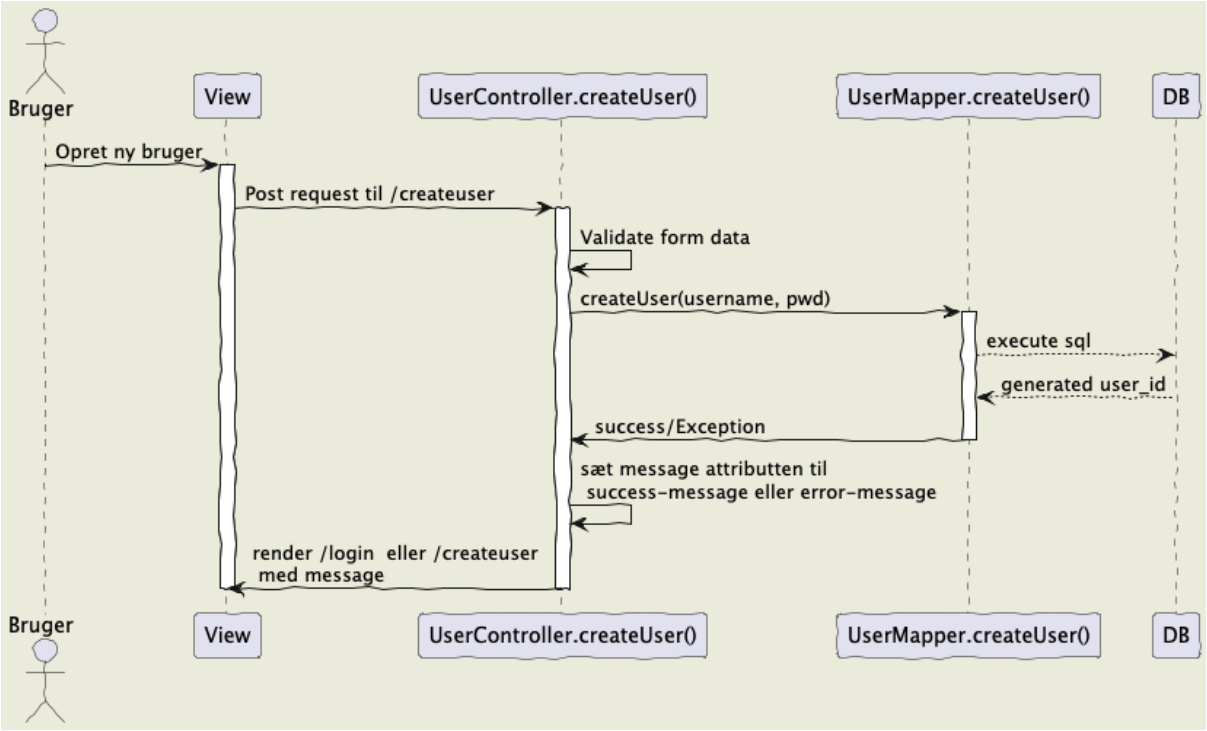
Vi har valgt at have en tabel ‘orders’ der indeholder info omkring en given ordre, en kunde har bestilt. Denne info drejer sig om datoer for bestillings-, og betalingstidspunktet og for hvornår ordren blev gennemført. Samt har vi info omkring navnet på ordren og en primærnøgle der består af et user_id og order_id.

Dernæst har vi en ‘order_lines’ tabel der indeholder de nødvendige og måske mere kritiske info (fra kundens side ad) omkring den givne ordres detaljer såsom: cupcake flavour, top_flavour, bottom_flavour, pris for de pågældende flavours antal cupcakes. Dette har vi valgt at gøre, da det gav mening for os at holde disse informationer adskilt. I denne tabel har

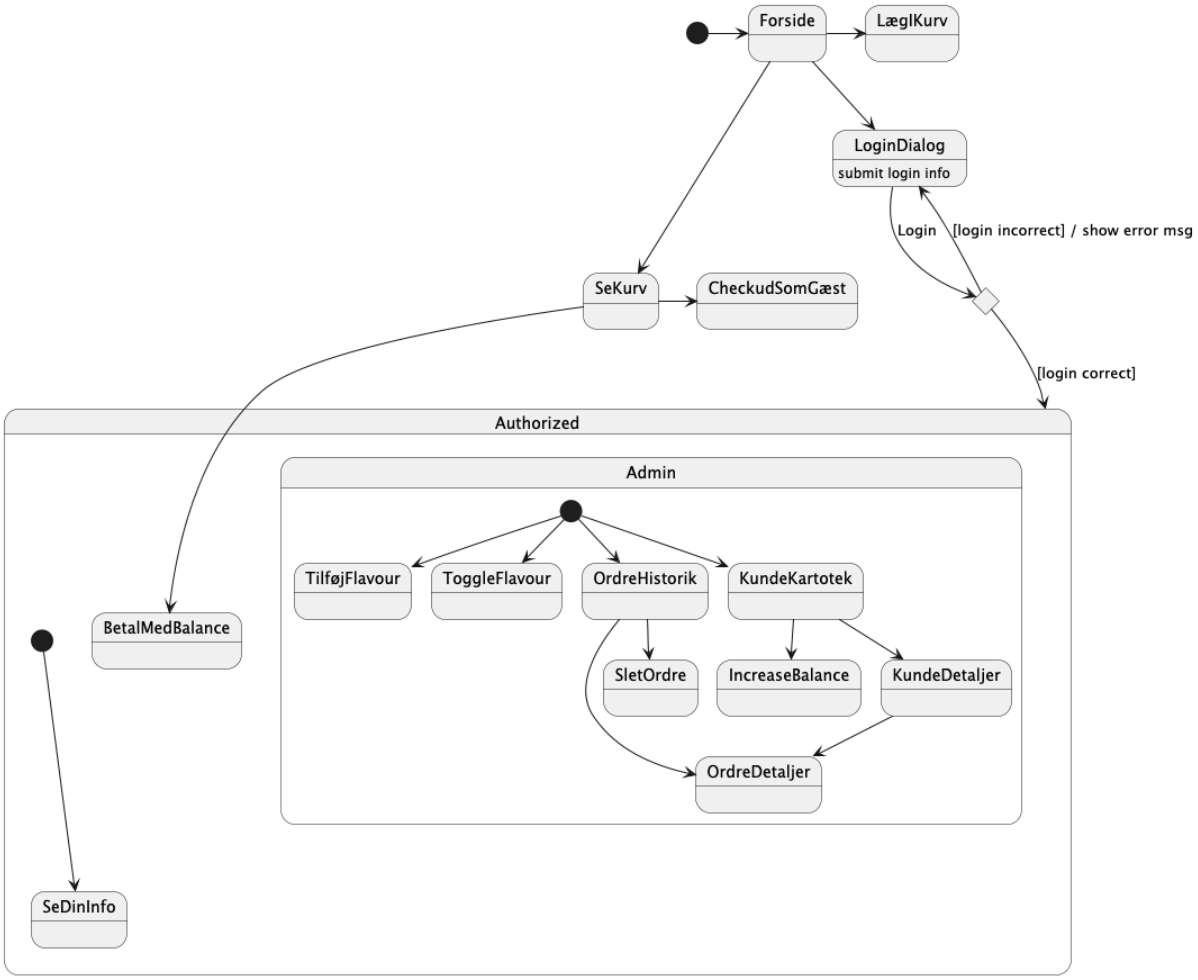
vi order_line_id og order_id som værende primærnøgle. Dette er fordi vi bruger disse felter til at identificere en ordre, som en kunde gerne vil bestille. Et order_line_id vil naturligvis tilhøre en given order_id som kan findes i 'orders'-tabellen - for så at se de yderligere detaljer omkring ordren.

Vi har også valgt at opsætte nogle indstille databasen til at kontrollere fx. unikke brugernavne og også tillade "null" brugernavne, da man i vores web-app godt kan bestille en ordre uden at have en konto hos os.

Sekvensdiagram for US-2



Navigationsdiagram



Særlige forhold

I dette segment af vores rapport, vil vi gerne snakke om de enkelte metoder vi har benyttet. Det er her vi lægger fokus på særlige segmenter af vores kode. De stykker af vores kode, som enten er nødvendige, vi er særlig stolte af eller meget brugbare i fremtiden, vil vi fremhæve her.Kurv og bruger gemmes i session
Vi har valgt at ordrelinjer som tilføjes indkøbskurven, som er en List af Orderline objekter, gemmes i en session variabel. Da vi også har valgt at indkøbskurven skal vises som en dropdown i menulinjen på hver side, så har det en fordel, at vi ikke skal lave et potentielt dyrt databasekald hver gang vi viser en side. Til gengæld har det en ulempe, at vi ikke kan spore indkøbskurve som “glemmes” eller på anden måde ikke med det samme fører til et køb. Når brugeren logger ud, så ødelægges sessionen, og indkøbskurven forsvinder.

Når man logger ind på siden, så gemmes et User objekt i en session variabel (currentUser). Vi har valgt at brugerens balance ikke synkroniseres med databasen løbende, dvs. hvis der bliver indbetalt penge på brugerens konto, så skal brugeren logge ud, før pengene bliver tilgængelige for brugeren. Her er der også mulighed for at der kan opstå en alvorlig fejl. Hvis en bruger logget ind, og en admin så registrerer en indbetaling på brugerens konto, og

brugeren så foretager et køb, så vil brugerens balance efter købet ikke være registreret korrekt. De penge som admin har registreret, vil være forsvundet. Denne fejl har vi ikke nået at rette.

Exceptions håndteres i Controlleren

Vi har arbejdet med en intention om at gribe alle de Exceptions vi kan i Controlleren, og så vise en relevant fejlbesked for brugeren. Det er primært i Mapper klasserne at vi griber SQL Exceptions og laver dem om til DatabaseExceptions. Vi har også en del steder hvor der er potentiale for NumberformatExceptions, som gribes og håndteres. Vi har primært brugt IntelliJ til at se hvilke Exceptions de metodekald vi bruger kan kaste, men har ikke været opmærksomme på at BCrypt librariet, som vi bruger, også kan kaste nogle Exceptions, som vi derfor ikke griber.

Bruger oprettelse

Vi har indført en del exception håndtering, password checks, brugernavn checks osv. når en bruger ville oprette en konto. En bruger bliver tvunget til at bruge en email, når personen vil oprette en konto. Dette bliver foreløbigt gjort med en simple if-statement:
```
if (username == null || !username.contains("@") || !username.contains("."))
```
Der kunne være der eksistere mere optimale måder at checke for valide emails. Med mere tid til projektet, så kunne vi måske håndtere det på en anden måde i fremtiden. En validering af en rigtig email kunne være brugbar. En bruger ville f.eks. kunne oprette sig med “@.@” Og det ville blive godtaget som et validt login. Selvfølgelig er det ikke en mail, som man kan sende en email til, og derfor ville en måde at kunne sortere ægte emails være en god ting at have med.

Password check

Når en bruger ville oprette et login, så bliver de både tjekket for om deres password er stærkt nok og om de har skrevet rigtigt. Vi tjekker derfor om deres password er mindst 8 tegn langt, har 1 tal og 1 speciel karakter. For at tjekke om passworded er det samme så sammenligner vi bare de 2 passwords de har skrevet

Password sikkerhed

Det mest interessante ved brugeroprettelsen, sker når deres password bliver gemt på databasen. I stedet for at gemme deres passwords i plaintext, så hasher+salter vi deres kodeord. Til dette bruger vi et import: “import org.mindrot.jbcrypt.BCrypt;”.
Bcrypt er en meget velanerkendt måde at lave krypterede kodeord på. Vi har valgt at lave denne kryptering i vores backend kode, sådan at passwords og database er helt adskilt fra hinanden. Dvs. at alle passwords der ligger på databasen er krypterede og selv en administrator eller en hacker ikke kan få et password ud af at kigge på databasen.

Login

Når en bruger vælger at logge ind, så er det deres egen kode og brugernavn de skriver. Koden finder brugernavnet i databasen og henter det hashede password. Det indtastede password hashes så med samme salt, og sammenlignes med den hash som er gemt i databasen. Hvis de stemmer overens, så bliver de logget ind.

Tjek af brugerrolle på alle admin sider

For alle endpoints (routes) som kun admins skal have adgang til (ordreoversigt, kundekartotek, sletning af ordre, opfyldning af balance osv.) laver vi i Controlleren et check for om currentUser session variablen er sat, og om den bruger der ligger i variablen har admin rettigheder.
Der ud over bruger vi Thymeleaf conditionals til kun at vise links der er relevante for den brugerrolle som er i currentUser.

Status på implementation (Idris)

Vi startede projektet med at holde fokus på de 9 User Stories. Dette gjorde vi kom godt fra start og gav os muligheden for at teste koden undervejs. Vi stødte på nogle udfordringer undervejs med et par enkelte User Stories. Vi mærkede under processen, at det påvirkede i meget lav grad, dem der sad og udviklede andre User Stories.

Dette var grundet, at vi tog en beslutning fra start af, at måden vi fordelte de individuelle opgaver op, til tilhørende User Stories, skulle både inkludere noget front- og backend samt SQL. Dette betød at nogle User Stories var afhængige af hinanden, før at man kunne fortsætte med noget nyt. Afslutningsvis var det en positiv beslutning, da man altid havde mulighed for at sparre med en anden i gruppen, med henblik på at få hjælp til der hvor man havde udfordringer.

Vi valgte at gøre web-appen mere robust og sikker ved at bl.a. kryptere brugerens kodeord med en blowfish-algoritme. Dette gjorde vi ved at importere et bibliotek i java og tilføje et dependency i vores pom-fil. Det vil sige, når en kunde opretter en konto, bliver deres kodeord hashed og lagret som et salt i databasen. Når kunden så logger ind på ny, vil appen hente dette salt fra databasen ned og sammenligne det hash med det indtastede kodeord brugeren har oplyst (som appen hasher på forhånd).

Udover dette har vi valgt også at sørge for at databasen kun tillader unikke brugernavne. Dette har vi opnået ved at oprette en unique constraint i databasen.

Vi valgte at oprette og implementere yderligere User Stories for at skabe et produkt der minder mere om et reelt produkt man vil se hos en kunde. Vi valgte at skabe:

- Mulighed for at administratoren kan deaktivere/aktivere flavour tilgængelighed.
- Mulighed for at administratoren kan tilføje nye flavours.

I tabellen nedenfor viser vi nærmere omkring de User Stories som vi også har implementeret udover de funktionelle krav

‘

User Story:	Acceptance kriterier:	Status:
1. Som kunde 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.	Givet jeg som kunde er logget ind, kan jeg betale med min store credit, hvis jeg ikke er det, kan jeg betale i butikken. Når jeg vælger at gå til kurven som bruger, bliver jeg ført videre til betalingsvinduet Så jeg kan se en succesbesked og hente min odre i butikken	Denne funktion er også implementeret. Man kan som eksisterende bruger eller som gæst bestille en cupcake med mulighed for at betale med de penge admin har indbetalt eller blot at afhente
2.Som kunde kan jeg oprette en konto/profil for at kunne betale og gemme en en ordre. 3. Som administrator kan jeg indsætte beløb på en kundes konto direkte i Postgres, så en kunde kan betale for sine ordrer.	Givet jeg ikke er logget ind og har trykket på Opret bruger menupunktet Når jeg har udfyldt formularen og trykket Opret bruger Så bliver der oprettet en bruger så jeg kan logge ind med den bruger —--------- Givet jeg er logget ind som administrator og kigger på kundedetaljer for en kunde Når jeg taster et beløb ind i inputfeltet og trykker indbetal Så opdateres kundes balance, således at det indsatte beløb lægges til den nuværende balance	Komplet implementation. Funktionen er implementeret korrekt og en bruger at oprette sig i vores system, hvis de opfylder kravene omkring brugernavn og password.
4. Som kunde kan jeg se mine valgte ordrelinier i en indkøbskurv, så jeg kan se den samlede pris.	Givet jeg som kunde har tilføjet cupcakes til kurven, Når jeg clicker på kurven, Så får jeg vist en oversigt over de valgte cupcakes, samt summen deraf.	
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).	Givet at jeg ikke er logget ind, så kan jeg logge ind på en oprettet bruger. Når jeg indtaster en bruger der er oprettet i systemet Så bliver jeg logget ind på hjemmesiden	Komplet implementation. Man kan logge ind både som bruger, men også som admin på websitet. Afhængig af om man er admin eller bruger, så giver det forskellige funktioner.
6. Som administrator kan jeg se alle ordrer i systemet, så jeg kan se hvad der er blevet bestilt.	Givet jeg er logget ind som administrator Når jeg trykker på Ordreoversigt	

	Så kan jeg se alle ordrer i systemet	
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.	Givet jeg er logget ind som administrator Når jeg trykker på Kundekartotek Så kan jeg se en oversigt over alle kunder Når jeg trykker på en kunde i oversigten Så kan jeg se en oversigt over denne kundes ordrer, samt information om kundens indestående	
8. Som kunde kan jeg fjerne en ordrelinie fra min indkøbskurv, så jeg kan justere min ordre.	Givet der er ordrerlinjer i indkøbskurven Når jeg trykker på slet ikonet ud for en ordrelinje Så forsvinder den ordrelinje fra indkøbskurven	
9. Som administrator kan jeg fjerne en ordre, så systemet ikke kommer til at indeholde udgyldige ordrer. F.eks. hvis kunden aldrig har betalt.	Givet jeg er logget ind som administrator Når jeg trykker på Ordreoversigt Så kan jeg fjerne en ordre	
10. Som en admin der er logget ind og kigger på tilføj flavour menupunktet Når jeg indtaster en flavour og en pris Så skal smagen op i databasen	Givet jeg er logget ind som administrator når jeg trykker på tilføj så kommer der en ny flavour i databasen	Man kan som admin også logge ind og oprette nye flavours, som vil blive lagret i databasen og derfter være tilgængelige for kunder
13. Som administrator kan jeg redigere i statussen på alle flavours, og derved bestemme om kunden skal have mulighed for at bestille en given flavour	givet jeg er logget ind som administrator, når jeg trykker på “rediger flavours” menupunktet kan jeg vælge hvilke flavours jeg vil aktivere/deaktivere, hvorefter den pågældende flavours status vil blive opdateret i databasen	Denne funktion er implementeret, testet og virker som det skal. Man kan som admin logge ind og tilgå en oversigt af alle flavours, der ligger i databasen og aktivere/deaktivere efter behov. Tilgængeligheden af disse, vil blive opdateret på kundens side.

Disse funktionaliteter blev implementeret relativt hurtigt og har skabt et mere fyldestgørende produkt, synes vi selv. Derudover, har vi valgt at man både kan bestille en ordre som eksisterende burger eller som gæst. Dette gør vi, ved at tillade at databasen må lagre en “Null”-værdi, da man ellers skulle være logget ind, for at kunne oprette et “brugernavn” i tabellen.

Dernæst, har vi lavet en gæsteprofil, som “gæsten” vil benytte sig af, når de bestiller en cupcake. Da alle ordrer har et orderId og alle brugere har et brugerId kan vi checke om denne ordre er tilknyttet en

Proces (Lasse)

Planen for samarbejdet lød på, at vi skulle angribe user stories en efter en, således vi alle kom ind på forskellige områder og fik kodet lidt af hvert. Vi udvalgte en techlead til at overse denne proces, samt at strukturere samarbejdet imellem os. Formålet var at have en med overblik til at tage sig af det generelle billede, og de andre i gruppen derved kunne koncentrere sig om alle detaljerne og implementering af user stories. Vi valgte primært at mødes og arbejde på skolen, så kommunikationen mellem gruppemedlemmerne var lettere. Vores plan var også at gøre heftig brug af kanban boards til at holde hinanden ajour med hvad vi hver især var i gang med, og hvor langt vi var nået. Selve user stories blev opdelt i prioriteter samt størrelse, og vi skulle derefter selv vælge en user story at arbejde på.

Overordnet virker denne arbejdsform til at have fungeret ret godt, og vi har haft nået godt i mål med de ting vi ønskede at implementere. Ud over at nå at udføre de krav vi havde, så har vi oven i købet også formået at indføre ekstra funktionalitet i programmet som fx. salted hashing af passwords.

Forløbet fungerede meget