Cupcake Shop

- **Gruppe π medlemmer:**
  - **Frederik Dinsen**
    - Email: cph-fd77@cphbusiness.dk
    - Github: fdinsen
    - Klasse: E
  - **Oliver Vang**
    - Email: cph-ov111@cphbusiness.dk
    - Github: olvang
    - Klasse: E
  - **Simon Kjems Jørgensen**
    - Email: cph-sj414@cphbusiness.dk
    - Github: Rodedrengen
    - Klasse: E

- *TIdspunkt: 30-03-2020*

Indledning

Dette projekt havde som mål at bygge en online webshop til en lille cupcake forretning. Vi skulle gøre brug af en MySQL database til at håndtere ordrer og kunder som opretter sig. Selve siden skulle bygges som en Java Web Applikation, for at give mulighed for dynamisk opbyggede sider, som henter information ind fra databasen i runtime.

Baggrund

"Olsker Cupcakes er en dybdeøkologisk iværksættereventyr fra Bornholm, som har ramt den helt rigtige opskrift." De har nu brug for at komme ind i den digitale verden, ved at udvide deres forretning med en Webshop.

En kunde skal kunne logge ind, samt registrerer sig hvis de ikke har en konto.
Kunden skal kunne sammensætte og bestille CupCakes med en top og en bund, via deres egen personlige profil.
På deres profil skal de have adgang til deres tidligere ordre, samt kunne se deres 'balance'.
Indkøbskurven skal give et overblik over hele ordren, ved at vise ordrelinjer, order linjers priser samt en total pris. Man skal også kunne fjerne tilføjet ordrelinjer i en indkøbskurv, hvis man har lavet en fejl.

Som administrator, skal man kunne se og ændre i og kunder. Her er det centralt at man kan ændre en kundes 'balance'. Man skal også kunne se alle kunders ordre samt kunne slette ordre/ordrelinjer.

Teknologivalg

Vi gjorde brug af en MySQL (V. 8.0.15) database til at opbevare ordrer og kundeinformationer. Den er forbundet til Java applikationen med JDBC (Java Database Connectivity) (V. 8.0.19). Systemet er skrevet i IntelliJ IDEA (V. Ultimate 2019.3) , og køres på localhost på en Tomcat 8.5.511 server. Unit Tests er skrevet ved hjælp af Junit (V. 4.13), og Hamcrest (1.3).

Siden bruger en række Bootstrap 4.4.1 elementer og der er brugt Chart.js (https://www.chartjs.org/ V. 2.8.0) til at lave en oversigt over salget i forskellige måneder. Vi har brugt Font Awesome (https://fontawesome.com/ V. 5.13.0) til forskellige ikoner. UML diagrammer er skrevet i plantUML og design mockups er lavet i Adobe XD (V. 28.2.12).

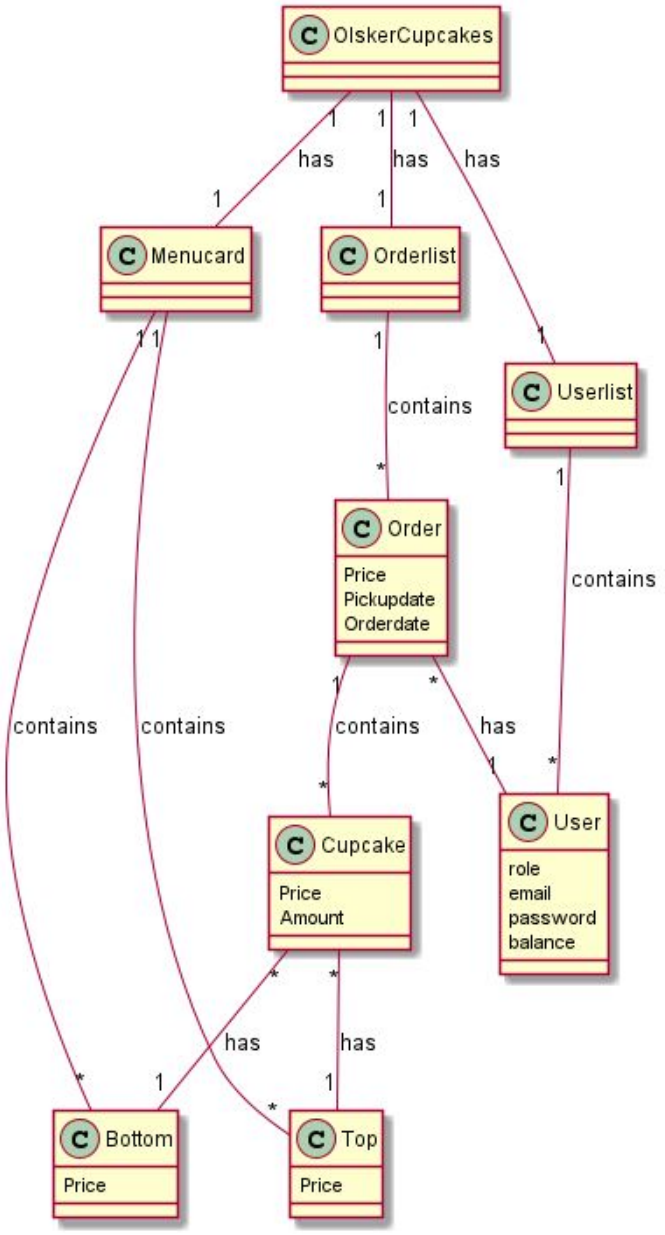
Krav

- **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 MySql, 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 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 ordre 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

Domæne Model

Systemet (OlskerCupcakes) skal kunne gemme brugere, så de kan lave bestillinger, og placerede ordrer, så Olsker kan gøre bestillingerne klar. Derudover skal der gemmes et menukort over de cupcake toppe og bunde som man kan købe. Alle disse har en én til én relation, da listerne kun skal eksistere én gang.



Menukortet indeholder mange toppe og bunde, men hver bund og top fremstår kun en gang på menukortet.

Ordrelisten indeholder alle ordrer som er blevet placeret. Det er kun ordre som der er blevet klikket 'betal' på, da det ikke giver mening at administratoren kan se ordre som kunderne ikke er færdige med at lave. På hver ordre gemmes en pris, en afhentningsdato og en ordredato.

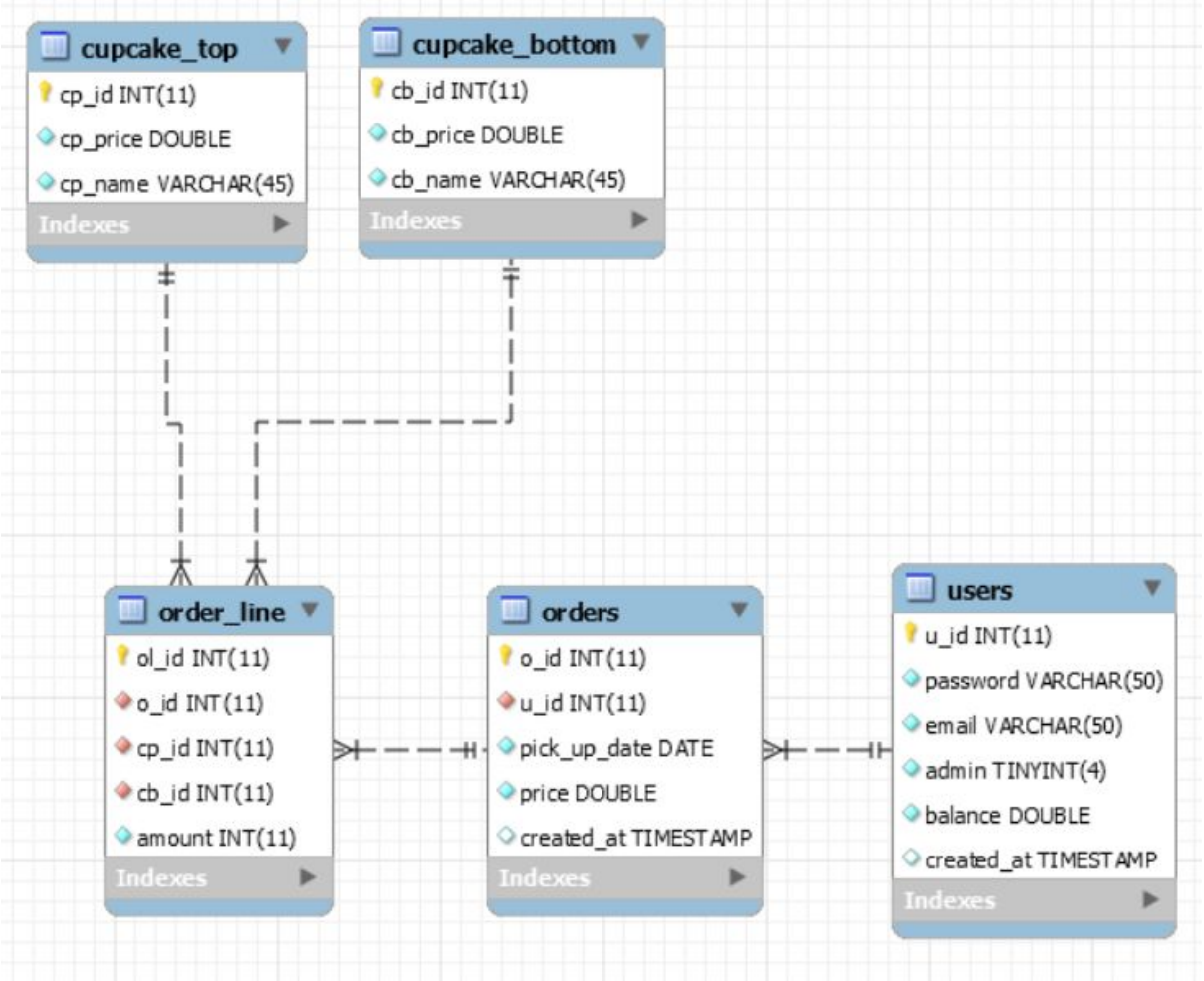
Hver ordre indeholder så mange cupcakes, hvorpå en samlet pris for den top og bund som cupcaken er bygget op af, gemmes såvel som et antal af den valgte cupcake som vil bestilles.

Hver ordre har også en bruger forbundet til sig, så Olsker ved hvem der har bestilt den givne ordre.

En bruger er bygget op af en email adresse, et kodeord, en saldo og en rolle, altså om de har administratorrettigheder på siden. Saldoen sættes af administratoren, når kunden har været nede i butikken og lagt penge på sin konto.

Brugerlisten indeholder alle brugere.

ER Diagram



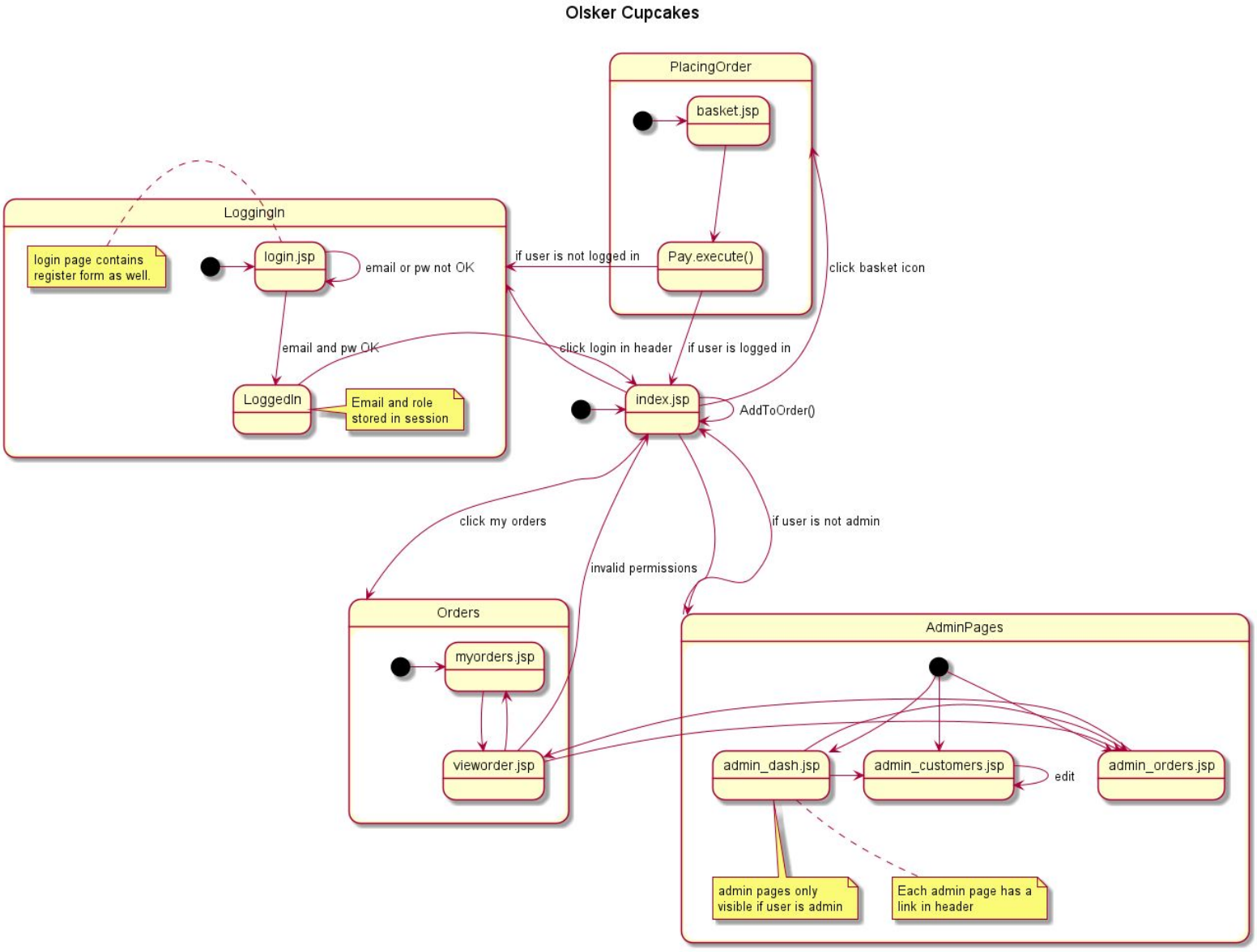
Databasen består af 5 tabeller, users, orders, order_line, cupcake_top og cupcake_bottom. Orders er forbundet til users med en foreign key som også har on cascade delete på. Det er gjort fordi man gerne vil undgå en ordre bliver oprettet på en bruger som ikke eksisterer. Desuden hvis man sletter en bruger sletter databasen automatisk alle ordre der hører til den bruger.

order_line er forbundet til orders med en foreign key og en on cascade delete. Igen for at undgå en order_line kan blive blive oprettet uden en kunde og så når en order bliver slettet, bliver alle dertilhørende order_lines og slettet.

Desuden er order_line forbundet til cupcake_top og cupcake_bottom med foreign keys. Så sikre vi os at der ikke kan eksisterer et cp_id eller et cb_id i order_line som ikke også eksisterer i cupcake_top eller cupcake_bottom. Dette giver dog det problem, at hvis vi opdaterer vores cupcake_top eller cupcake_bottom tabeller vil order_line ikke længere passe.

Løsningen kunne evt være man havde en fjerde atribut på cupcake_top / cupcake_bottom som hed inUse. Denne kunne så afgøre om man skulle bruge denne top / bund, eller om den var udgået af sortimentet.

Navigationsdiagram



Vi har fire hoved-dele af systemet: At logge ind, at placere en ordre, at se sine ordrer og admin-funktionerne. Øverst på alle disse sider er der en fælles header, som lader brugeren navigere til alle andre afdelinger.

På index siden kan man kontinuerligt tilføje cupcakes til sin ordre. For at placere sin ordre navigerer man til basket.jsp. Hvis man ikke er logget ind, tages man til login-siden. Hvis man er logget ind, kan man klikke betal, og så sendes man tilbage til index.

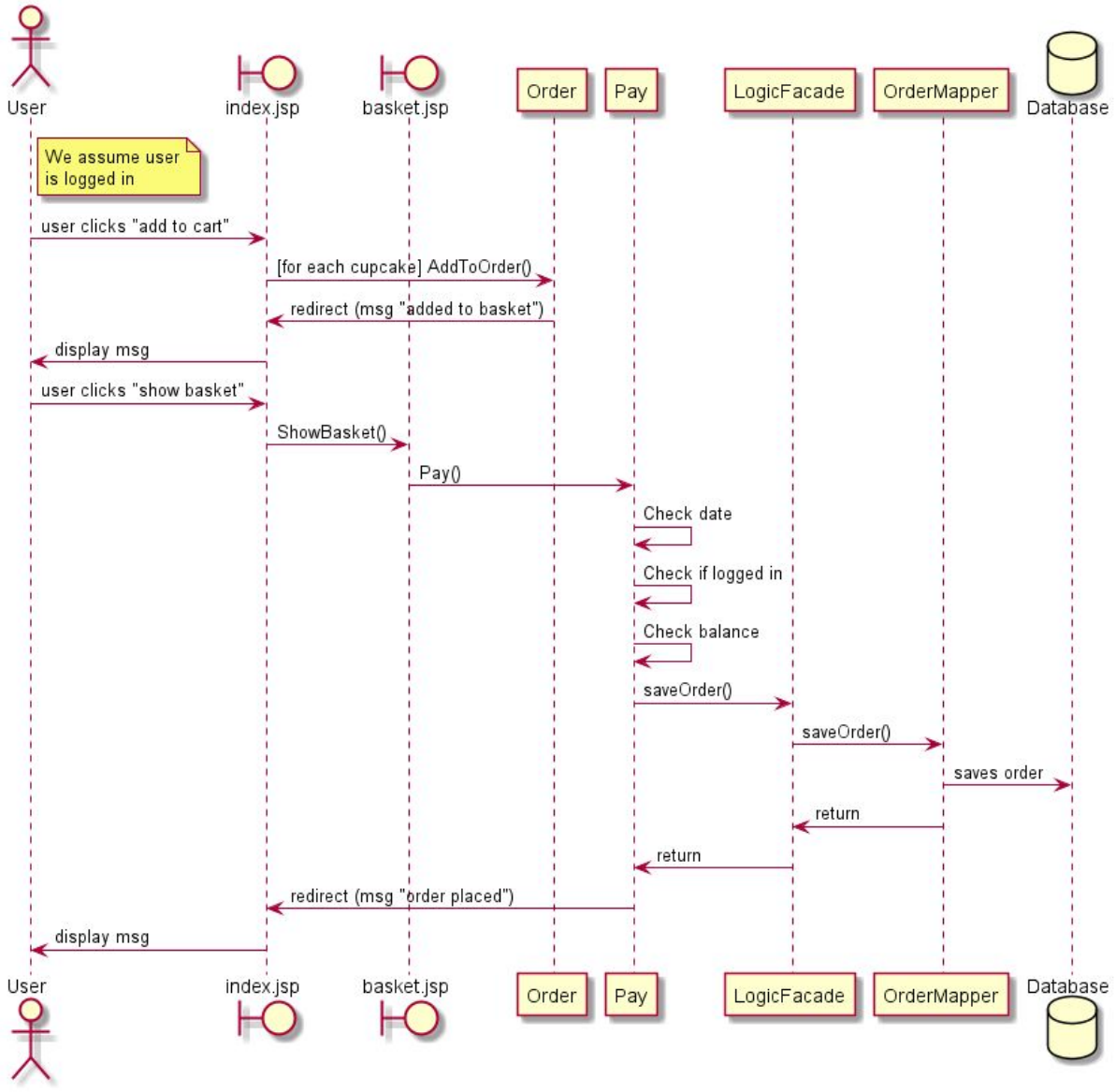
Hvis man vil se sine ordrer, kan man klikke “Mine Ordrer” i headeren, og få en komplet liste over ens placerede ordrer. Man kan her klikke ind og se indholdet af hver ordre på vieworder.jsp. Hvis man prøver at se en ordre man ikke ejer, sendes man tilbage til index.

Hvis man er logget ind som en administrator kan man se links til admin siderne. Disse er skjult for almindelige brugere. Hvis man finder et link til en adminside, men man ikke er logget ind som en admin, smides man tilbage til index.

Her kan man enten se administrator dashboardet, som har links til de andre adminsider, samt lidt ekstra information, eller man kan gå direkte til at se en oversigt over kunderne og en oversigt over alle ordrer. På oversigten over brugere kan man redigere de enkelte brugere. På oversigten over ordrer kan man klikke ind på individuelle ordrer og se detaljerne, også på vieworder.jsp

Sekvensdiagrammer

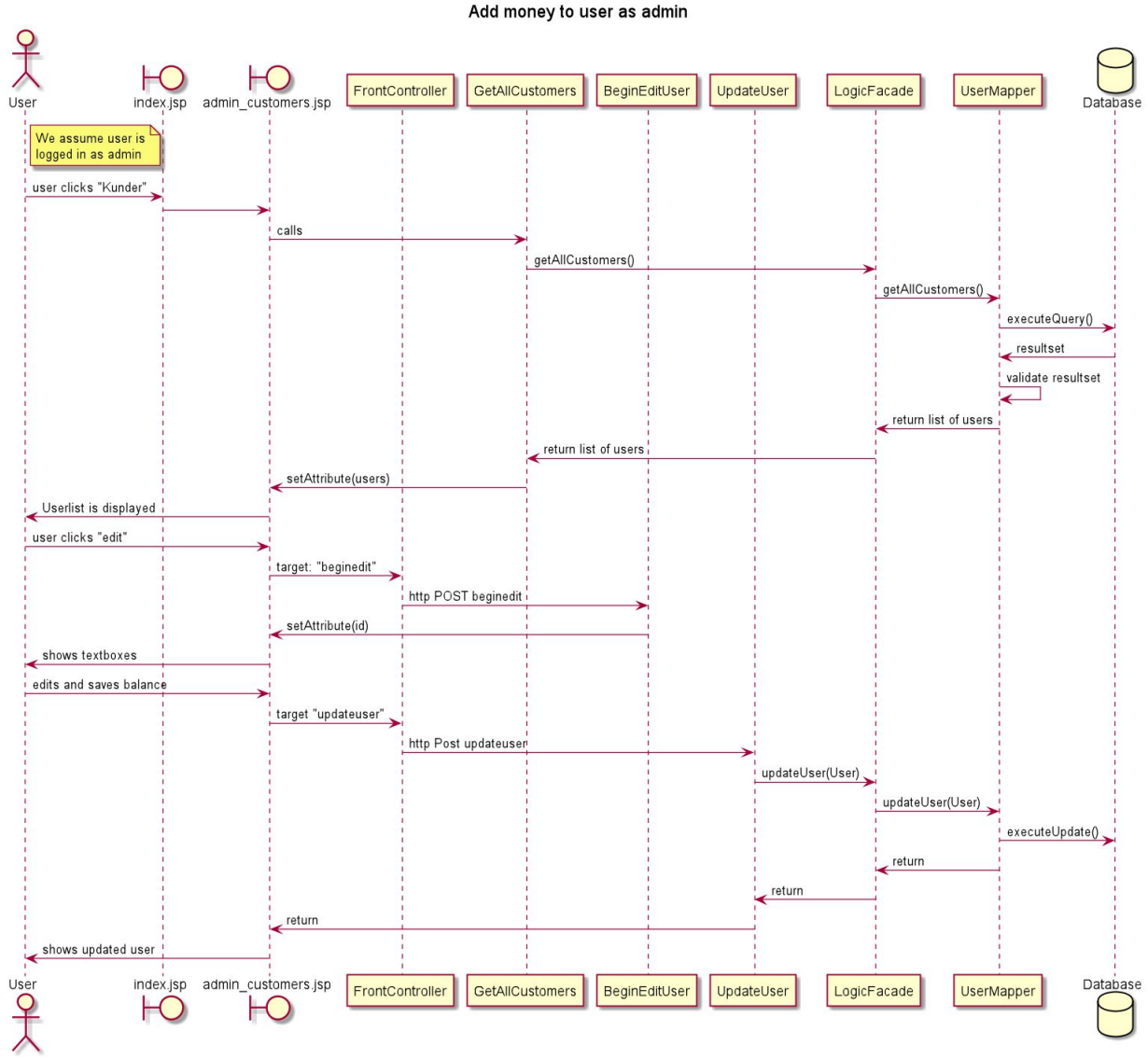
Bestilling og betaling af en Ordre



Vi går ud fra at, der ingen steder bliver tastet forkert eller ugyldige tegn.
På index.jsp har man mulighed for at tilføje CupCakes til sin ordre, hver gang man tilføjer noget til kurven, vil 'AddToOrder' blive kaldt, hvor ordrelinjen vil blive langt ned i et order objekt på session niveau. Når kurven vises kaldes der på order objektet, og der udskrives et overblik over ordren. Kunden vil kunne se en betal knap, som kun vises når man er logget ind.
Når man klikker 'Betal', vil der i Pay blive valideret dato, om de logget ind, samt om deres balance er høj nok til at kunne betale og til sidst vil den gemme ordren i databasen.
Når ordren er gemt, vil alle attributter på session niveau blive opdateret eller sat til 'null' og

kunden vil herefter blive omdirigeret til index.jsp, med en besked om at ordren er fuldført.

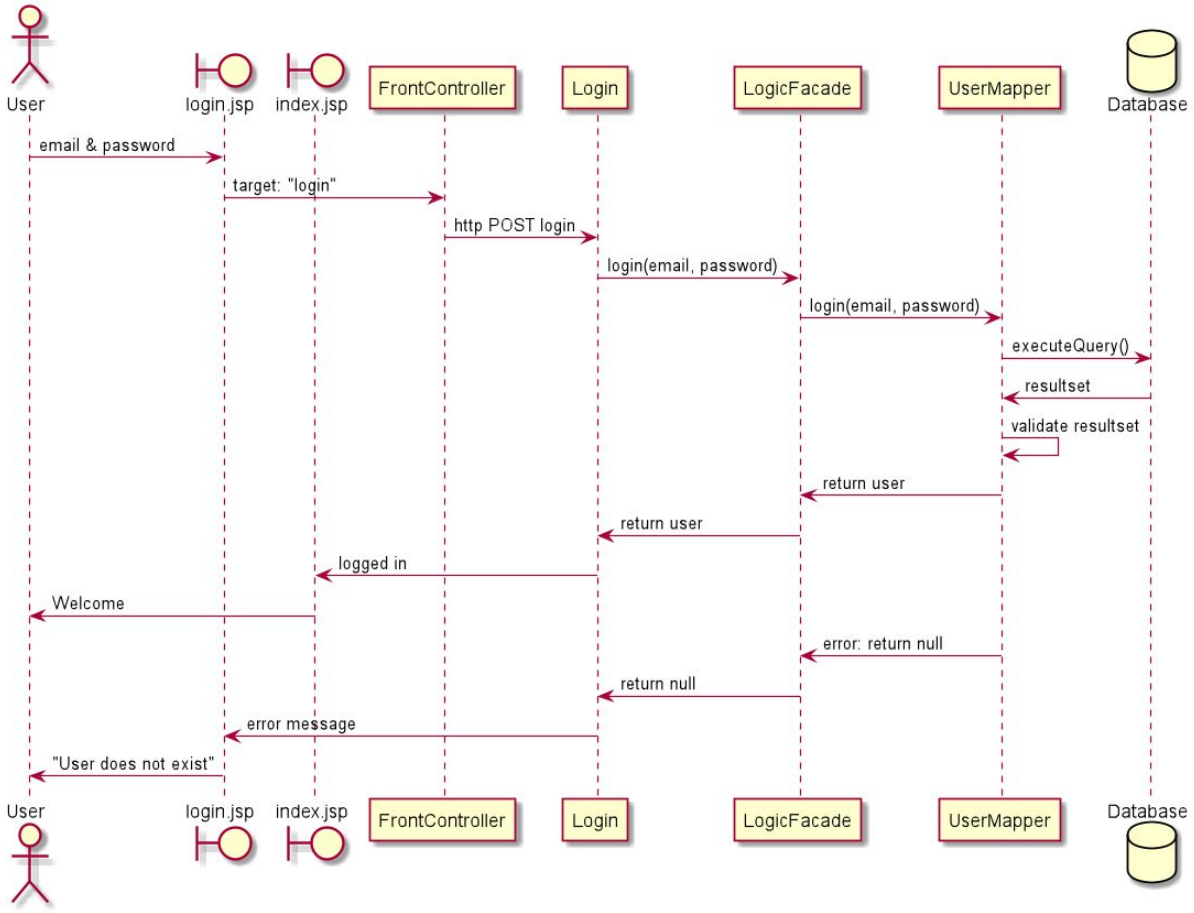
Tilføj penge til en bruger som Administrator



Vi går ud fra at, der ingen steder bliver tastet forkert eller ugyldige tegn.
Fra index.jsp siden vil der oppe i Navigationsbaren, være et link til en 'Kunde Oversigt' når man er logget ind på Admin. Når der bliver klikket, vil der inden siden bliver vist, kaldt ned i databasen for at hive alle kunderne ud. Her bliver de gemt i en Liste, der returneres tilbage ud til admin_customers.jsp, som der loopes igennem inden siden vises til brugeren.
Admin vælger den bruger han ønsker at redigere, som gør at BeginEditUser vil blive kaldt med et id på den bruger der skal redigeres. Her bliver både email og id'et sat som en request attribut og der returneres tilbage til admin_customers.jsp. På admin_customers.jsp, ligger der en IF sætning der tjekker for om IDToEdit er sat, hvilke gør at den nu udskriver den brugers informationer som inputfelter. Her ændre admin balancen og klikker på gem, der gør at UpdateUser bliver kaldt. Her vil der blive oprettet et User objekt, der bliver givet med til updateUser i userMapper. Usermapper "udpakker" objektet og inden sætter det i

databasen.
Herefter bliver der retuneret til UpdateUser, hvor at den ændret brugers oplysninger bliver opdateret i listen over alle kunder. Til sidst retuneres der til admin_customers.jsp, der igen kan udskrive listen over alle kunder med den opdateret data.

Login sekvensdiagram



Sekvensdiagrammet starter med brugeren indtaster sine oplysninger, email og password, på login.jsp. Her går vi ud fra der ikke er nogen ugyldige tegn. Disse oplysninger bliver så sendt til frontcontroller servletten hvor de sendes videre til login.java. Login.java sender så forespørgelsen ned igennem logicfacede og videre ned i usermapper.

Usermapper eksekverer så en query der prøver at finde de oplysninger brugeren har tastet ind. I dette tilfælde går vi ud fra brugeren har indtastet de rette oplysninger og usermapperen retunerer den aktuelle user op igennem logicfacede til login som så sætter brugeren på session. Index.jsp viser så den aktuelle bruger der er logget ind og byder dem velkommen.

Hvis brugeren derimod har tastet forkert, returneres null fra mapperen, hvilket Login command-klassen håndterer og sender en fejlmeddelelse op til brugeren.

Særlige forhold

I sessionen gemmer vi bruger objektet, samt deres rolle som en boolean, email som en string og saldo som en double, så de informationer er lette at tilgå. Derudover er selve ordren gemt på sessionen, da den skal gemmes på tværs af sider.

Vi omdanner alle exceptions til vores egen LoginSampleException, som så håndteres i command-klasserne ved at smide brugeren til index siden, med en fejlmeddelelse. Havde vi haft mere tid, ville vi have delt den ud i flere forskellige exceptions som er mere specifikke, og derfor mere sigende.

Bootstrap sørger for at der ikke tastes en ukomplet email ind i email-felterne, og vi tjekker om password felterne matcher. Vi burde nok validere dem selv, men det havde vi ikke tid til. Derudover kan man kun indtaste en afhentningsdato gennem en lille kalender-popup, så der kan kun indtastes datoer. Antal er et number-field, så der kan kun tastes tal op til 30, hvilket er en arbitrær begrænsning vi valgte, for at undgå sindssyge bestillinger. Hvis der tastes over 30, lægges der bare 30 i kurven. Der benyttes selvfølgelig Prepared Statements, så der er beskyttet mod SQL injections derigennem. Søgefelterne på oversigten over ens ordre bruger javascript, så der sendes ingen information til serveren. Derudover tjekkes ejerskab af en ordre, når man prøver at se den. Så hvis brugeren ændrede ordre-id'et i URL'en til en ordre de ikke ejer, bliver de smidt tilbage til forsiden.

Havde vi haft mere tid, ville vi have brugt hash-and-salt på vores kodeord, da det er skidt at de gemmes i plaintext. Vi burde nok også fjerne passwordet fra bruger-objektet, før det gemmes i sessionen, da vi ikke er sikre på om det kan tilgås fra developer-tools på en eller anden måde.

I den nuværende version af systemet gemmes saldoen som en double, hvilket er et problem når vi begynder at give procentvise tilbud, da floating-point tal mangler præcision, som er skidt når der tales om penge. Fremadrettet vil vi gemme saldoer og priser som integers, i deres ører-pris.

Status på implementation

- Vi ville gerne have haft vores .jsp sider til at automatisk udfylde den tilgængelige skærm-plads, så vi undgik en masse white-space under footeren.
- Saldo skulle have været integers.
- Vi skulle have haft flere exceptions.
- Mere inputvalidering.
- Nogle tests er ikke blevet opdateret til den nye test-data, så de fejler selvom metoderne virker.
- Siden virker lidt for lille på nogle smartphones.

Test

- Vi har testet på mapper-klasserne.