Forside

Gruppe: 2 (JDJE).

Gruppemedlemmer: Emil Grønlund, Jimmy Pham, Jannich Højmose og Daniel Bengtsen.

Dato: 16/03 - 30/03, 2020.

GitHub-repository: https://github.com/VeraUtil/Cupcakes

Tegn: 11506 svarende til 4,79 normalsider.

Indholdsfortegnelse

Indledning

Projektets kunde er Olsker Cupcakes, der - som navnet hentyder - laver cupcakes. Virksomheden er et nystartet iværksætterfirma, der har fokus på dybdeøkologi og som er i rivende udvikling. Derfor er virksomheden i gang med at ekspandere i form af et nyt system, der håndterer bestillingen af deres produkter på et website. Dette system har vi haft til opgave at programmere i samarbejde med kunden.

- Kundens krav er som følgende (user stories):

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.*

Ekstra:

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.*

- Vi har brugt følgende teknologier til udførelse af projektet:

- JDBC.
- MySQL, ver.: 8.0.
- MySQL-Connector-Java, ver.: 8.0.19.
- IntelliJ, ver.: 2019.3.3.
- Digital Ocean Droplet.
- Tomcat, ver.: 8.5 & 9.
- PlantUML integration (plugin til IntelliJ), ver.: 2.23.0.
- PlantUML Syntax Check (plugin til IntelliJ), vers.: 0.2.0.
- Graphviz, ver.: 2.38.
- AdobeXD, ver.: 27.2.12.4
- JSTL, ver.: 1.2.
- Maven, ver.: 1.8.
- Maven-Compiler-Plugin, ver.: 3.1.
- Maven-War-Plugin, ver.: 2.3.
- Maven-Dependency-Plugin, ver.: 2.6.
- GIT, ver.: 2.21 & 2.23.0.
- FileZilla, ver.: 3.47.2.1.
- PuTTY, ver.: 0.73.
- Bootstrap, ver.: 4.3.1.
- Hamcrest-core, ver.: 1.3.
- Javaee-web-api, ver.: 7.0.
- Javaee-endorsed-api, ver.: 7.0.

Interessentanalyse

	Stor indflydelse	**Lille indflydelse**
Bliver påvirket af projektet	Kunden (Ressourceperson)	Kundens kunder og kundens medarbejdere (Gidslerne)
Bliver ikke påvirket af projektet	Os (Grå eminencer)	Kundens bank (Eksterne interessenter)

Ressourceperson:

Kunden er ressourcepersonen, da kunden netop har indflydelse på projektet og i største grad bliver påvirket af det færdige produkt. Det er derfor også den vigtigste interessent i vores projekt.

Grå eminencer:

I dette tilfælde er det os, da vi har stor indflydelse på projektet og er i tæt dialog med ressourcepersonen (kunden). Kundens produkt - altså salget af cupcakes - har dog lille til ingen indflydelse på os.

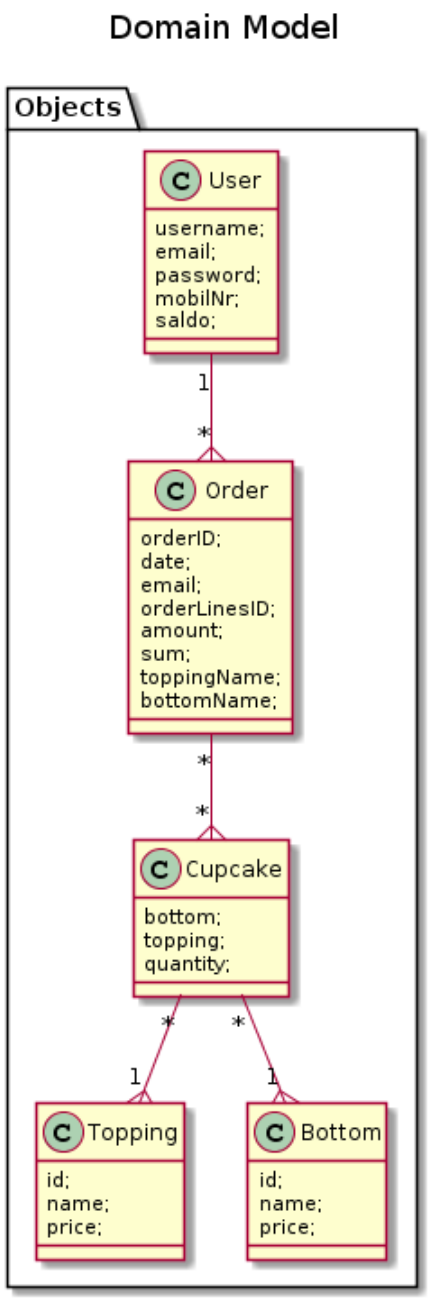
Gidslerne:

I dette tilfælde har vi vurderet, at det er kundens kunder og kundens medarbejdere. Kundens kunder bliver påvirket i form af en ny måde at handle hos Olsker Cupcakes på, mens medarbejderne får en ny arbejdsrutine, men fælles har begge parter lille til ingen indflydelse på projektets gang, trods den store påvirkning.

Eksterne interessenter:

De eksterne interessenter er banken, der låner Olsker Cupcakes penge. Dette skyldes, at banken giver projektet startkapital, men projektets forløb og resultat har den lille til ingen indflydelse på. Det færdige produkt kommer heller ikke til at have indflydelse for banken.

Domænemodel



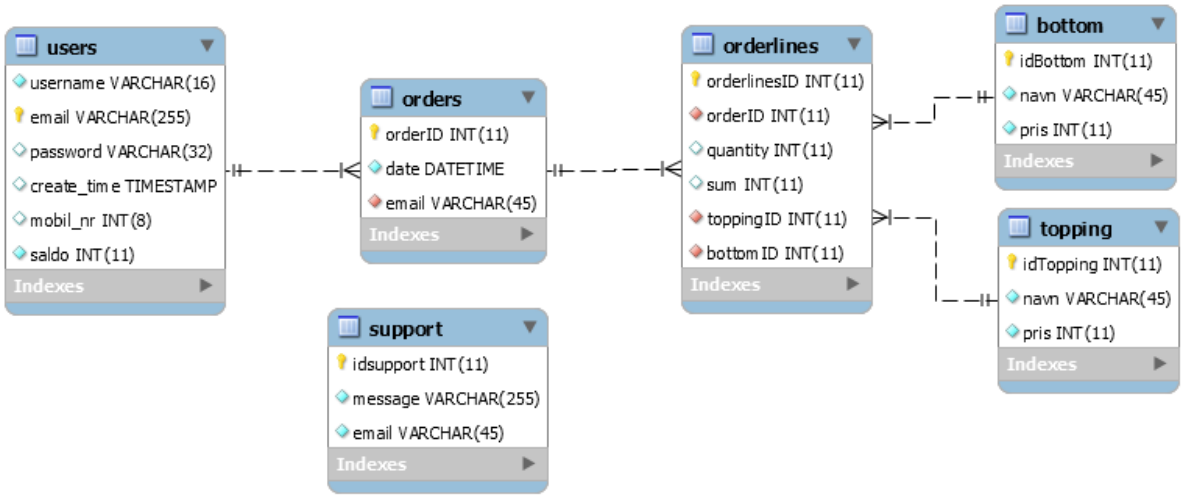
På vores domæne model fremgår forskellige objekt-klasser. Et User-objekt består af et username, en email (der er unik), et kodeord, mobilNr og en saldo. Vores User-klasse har en én til mange relation til Order, da en User kan lave flere ordrer, men den enkelte ordre kan ikke have flere users - altså ordren er unik.

Vores Order-klasse har en mange til mange relation til Cupcake, da et Order-objekt kan have mange cupcakes og et Cupcake-objekt kan være på mange forskellige ordrer. Et Ordrer-objekt består af en række variabler, bl.a. orderID som er primærnøgle og orderLinesID, som er fremmednøgle, der kobler forskellige cupcakes sammen - dvs. en ordre med flere underordrer.

På Cupcake-klassen er der en mange til én relation til Topping- og Bottom-klasserne, da et Cupcake-objekt kun kan bestå af én type bund og én type topping. Den enkelte type bund og topping kan dog godt være på flere forskellige cupcakes og har derfor en mange relation. Cupcake-objektet består af et Topping- og Bottom-objekt samt et antal (kvantitet) af cupcakes. Henholdsvis består Topping og Bottom af et unikt ID, et navn og en pris.

Diagrammer

Er-diagram:



Er-diagrammet beskriver relationerne mellem tabellerne i databasen. Users-tabellen har e-mail som primærnøgle. Vi har valgt ikke at anvende et autogenereret ID, da e-mailen er unik; man kan ikke oprette den samme mailadresse mere end én gang.
Orders-tabellen har email fra users-tabellen som fremmednøgle og orderID som primærnøgle. Orderlines-tabellen har et autogenereret ID som primærnøgle og 3 forskellige fremmednøgler. Disse 3 fremmednøgler er hver især et autogenereret ID, som henholdsvis kommer fra orders-, bottom- og topping tabellerne.
Support-tabellen har en autogenereret primærnøgle, men har ikke nogen relation til de andre tabeller i databasen, da de andre tabeller ikke er afhængige af data fra support-tabellen og vice versa.

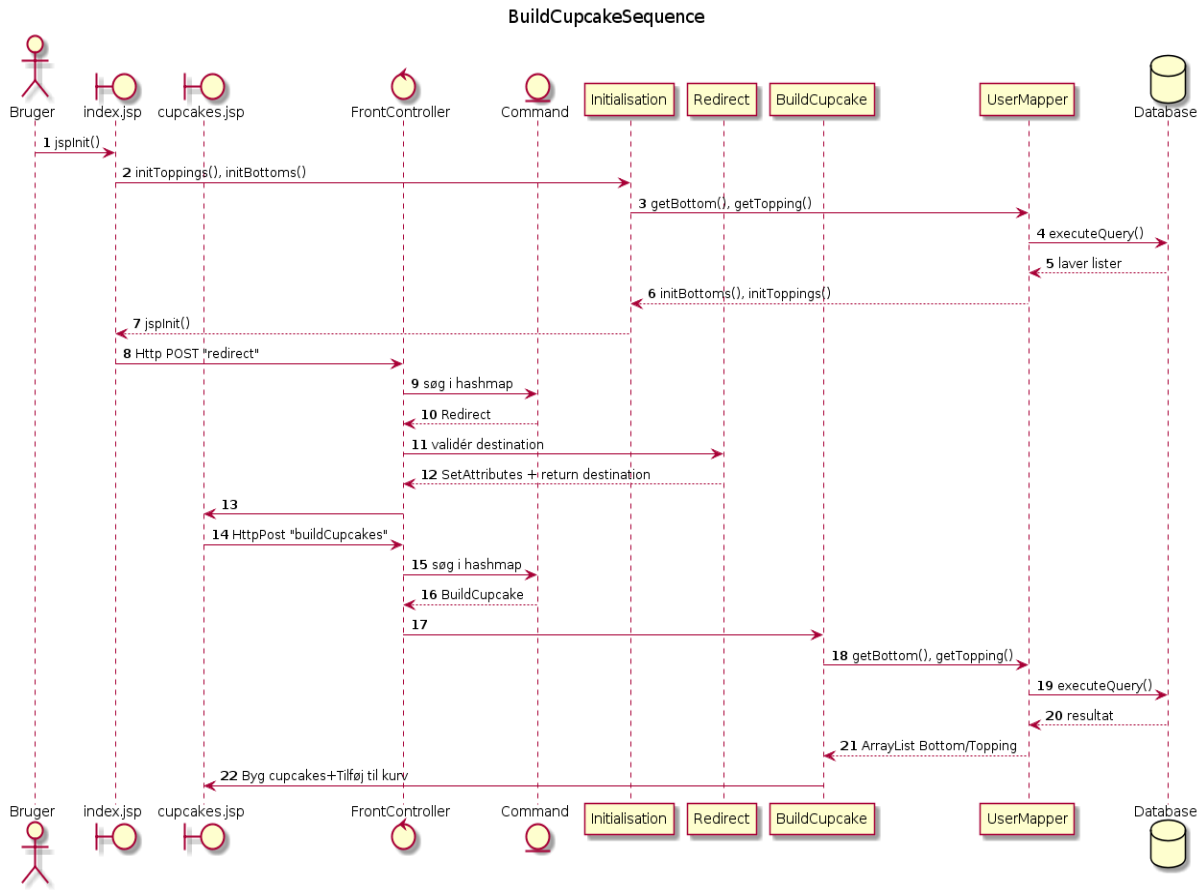
Databasen er på anden normalform, idet attributterne - antal og sum - i orderlines-tabellen ikke er nøgler, men samtidig er afhængige af hinanden. Dette kunne være løst ved at sætte den samlede sum ind i orders-tabellen og fjerne den fra orderlines. På denne måde ville databasen opnå tredje normalform.

På trods af en manglende tredje normalform, håndterer vi konflikten mellem antal og sum i orderlines-tabellen. I artiklen om normalisering - i afsnittet om kravene for tredje normalform - står følgende:

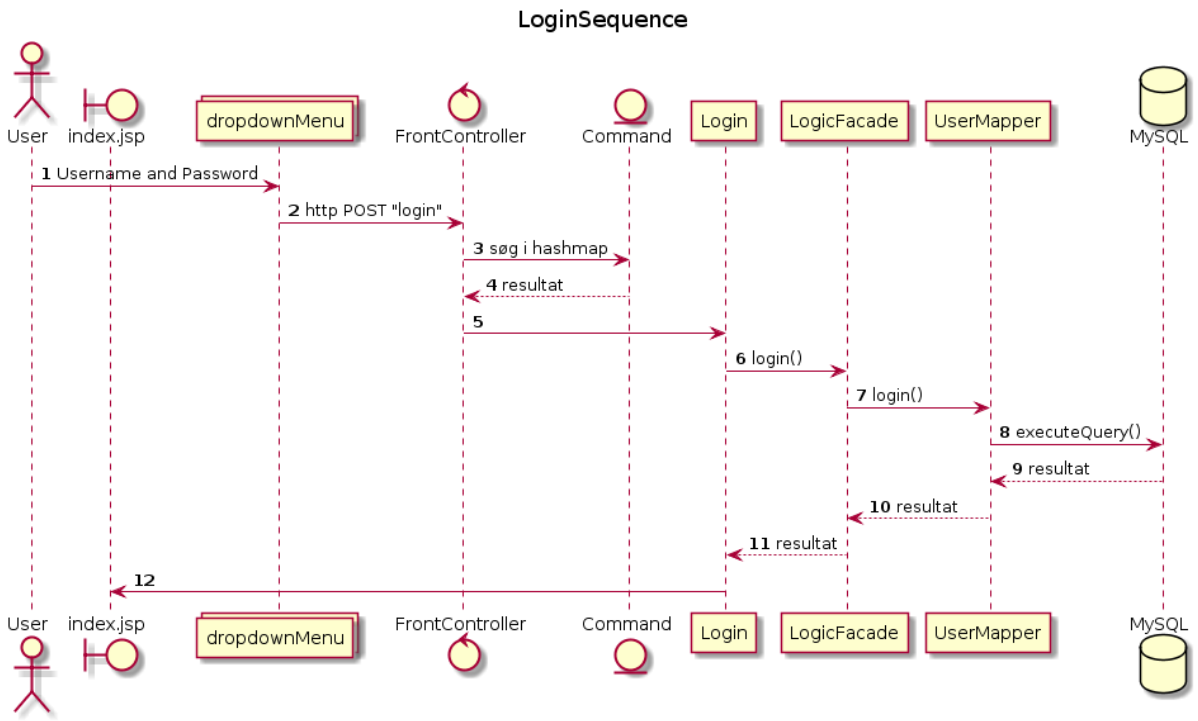
> *"Hvis vi vil opfylde kravene til tredje normalform, skal vi lave en separat tabel til at håndtere relationen mellem postnumre og bynavne."* [1]

Da der står "skal vi lave en separat tabel til at **håndtere relationen** mellem postnumre og bynavne", kan vi argumentere for, at vi håndterer relationen mellem antal (i deres eksempel: bynavne) og sum (i deres eksempel: postnumre) i Java ved, at den enkelte orderline beregnes løbende i en session før den lægges i tabellen.

Sekvensdiagram:

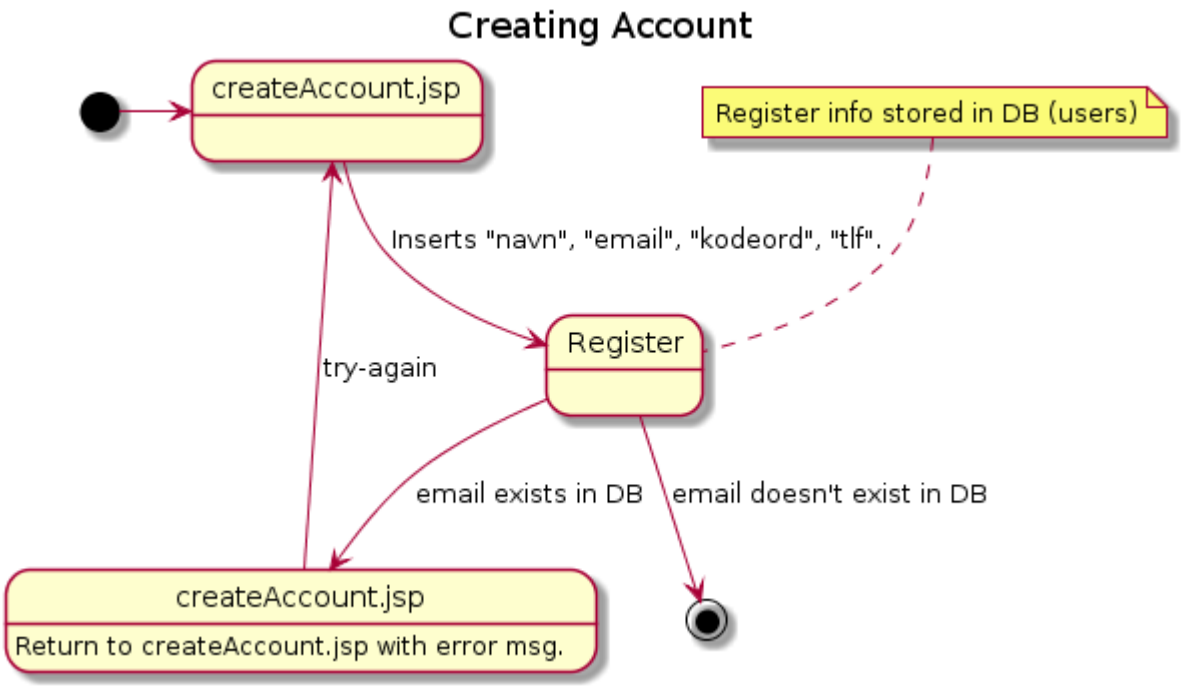


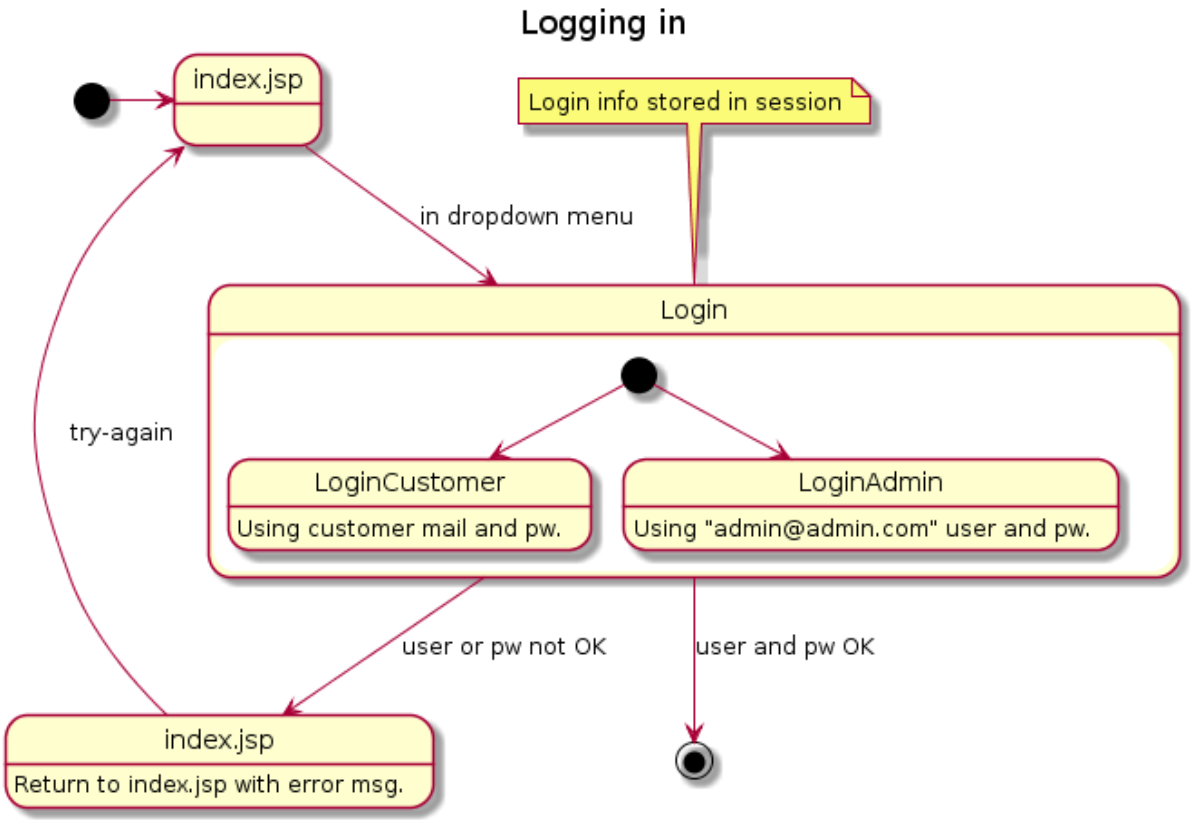
[1] https://datsoftlyngby.github.io/dat2sem2020Spring/uge09/resources/DB_normalisering.pdf
Relationelle Databaser



Som det kan ses, benytter vi en Command Pattern skabelon i vores program. Dvs. at fra jsp siderne sender vi en POST med et specifikt navn til FrontControlleren, som derefter søger i et HashMap, som vi har i den abstrakte Command-klasse. Efter den har fundet det specifikke navn - der fungerer som ID - kan det kobles op på en Java-klasse, der nedarver fra Command-klassen og den Java-klasse sendes så tilbage til FrontControlleren og bliver derefter kørt. Alt efter hvad den Java-klasse har af funktioner, vil der være tilgang til andre Java-klasser, som f.eks. tilgår vores database eller har andre funktioner såsom at oprette en kurv, tilføje/fjerne fra kurven og beregne totalprisen for hver vare i/hele ordren. Derefter bliver resultatet sendt tilbage igennem Java-klassens execute()-metode, som er nedarvet fra Command-klassen, og derfra kan man sende resultatet igennem et Scope, så det kan ses på vores jsp-sider.

Tilstand-/navigationsdiagram:





Fælles for de to diagrammer er, at der skrives noget input, som bliver valideret. Hvis valideringen fejler, sendes brugeren tilbage til samme jsp-side med en fejlbesked. Hvis valideringen er succesfuld, er brugeren oprettet eller logget ind.

- Creating Account diagrammet:

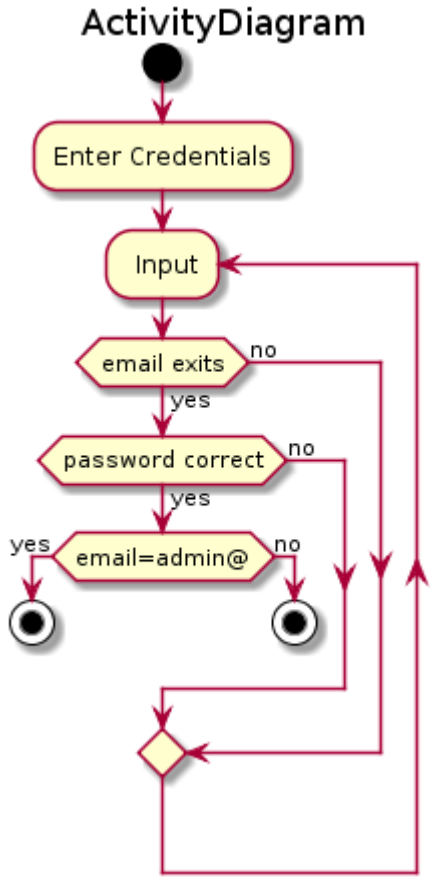
Brugeren står på createAccount.jsp-siden og indtaster værdierne i formularen til oprettelse af en konto. Valideringen sker på email-adressen, da den er primærnøgle i users-tabellen (som består af alle konti). Findes e-mailen allerede i systemet, eller har brugeren udfyldt formularen forkert, modtager brugeren en fejlbesked på siden og må udfylde oprettelsesformularen igen. Er valideringen succesfuld, er brugeren blevet oprettet og kan efterfølgende logge sig ind.

- Logging In diagrammet:

Brugeren står på index.jsp-siden, klikker på dropdown-menuen i øverste højre hjørne og indtaster e-mail samt password. Hvis e-mail og/eller kodeord ikke matcher værdierne, der er gemt i databasen, modtager brugeren en fejlbesked på siden og må udfylde login-formularen igen. Er valideringen succesfuld, bliver brugeren sendt til samme side og er så logget ind. Hvis e-mailen undtagelsesvis er "admin@admin.com"[2] samt kodeordet dertil, er brugeren logget ind som administrator og sendes til admin.jsp-siden.

[2] Admin er oprettet på forhånd i databasen med denne mail. Matcher kodeordet ikke, bliver brugeren ikke sendt til admin.jsp selvom den angivne mail er indtastet i loginfeltet. Brugeren kan med andre ord, hverken oprette sig som eller logge sig på som admin.

Aktivitetsdiagram:



Vores aktivitetsdiagram viser flowet for login-processen. Det fungerer ved, at brugeren indtaster sit login, hvilket er startpunktet. Såfremt den angivne e-mail og kodeord stemmer overens med værdierne i databasen, fortsætter processen og tjekker derefter, hvorvidt e-mailen er lig med admin-mailen, hvilket er defineret i backend-delen. Såfremt e-mailen er tilsvarende til admin-mailen, bliver brugeren returneret til dette scenaries slutpunkt (admin-siden). Hvis brugeren derimod er logget ind uden admin-mail og kodeord, returneres brugeren til dette scenaries slutpunkt (index-siden). Hvis ikke mailen og kodeordet stemmer overens, fremkommer en fejlmeddelelse om forkert login.

Særlige forhold

- Hvad gemmes i session:

Vi gemmer forskellige informationer i sessionen. Login-informationen gemmes i en session, så brugeren kan navigere rundt på de forskellige sider uden at blive logget ud. Her gemmes e-mailen i en session string således, at vi kan vise brugerens e-mail på alle vores jsp sider via sessionScope.

```
HttpSession session = request.getSession();

session.setAttribute( s: "email", email);
```

Kurven gemmes også i en session, da man skal kunne bevæge sig rundt på de forskellige sider uden, at kurven bliver slettet.

- Hvordan valideres brugerinput:

Vi har lavet en række valideringer på brugerinput, hvor der bl.a. tages højde for om felterne på siden er blevet udfyldt. Dette eksempel er fra cupcake.jsp-siden:

```
String tmpAntal = request.getParameter( s: "antal");
String bottom = request.getParameter( s: "bund");
String topping = request.getParameter( s: "top");


if(tmpAntal.equals("") | bottom.equals("") | topping.equals("")) {
    request.setAttribute( s: "BuildCupcakeBesked1",  o: "Du udfyldte ikke alle felter");
    request.setAttribute( s: "toppings", Initialisation.getToppingList());
    request.setAttribute( s: "bottoms", Initialisation.getBottomsList());
    return "cupcakes";
}
```

Her hentes de værdier brugeren har indtastet på cupcakes.jsp-siden. Hvis et enkelt felt ikke er udfyldt, sendes der en fejlbesked tilbage til brugeren. Efterfølgende hentes listerne fra databasen med top og bund, og brugeren sendes tilbage til cupcakes-siden med fejlbeskeden: “Du udfyldte ikke alle felter” med rød skrift. Hvis brugeren har udfyldt det hele, vil beskeden i stedet være: “Din cupcake er tilføjet til kurven” med grøn skrift.
Når brugeren skriver antal cupcakes ind, valideres inputtet ved at sætte typen i input-feltet til “number”:

```
<input name="antal" type="number" id="antal" class="form-control-smaller" placeholder="Skriv antal...">
```

Status på implementation

- Vores mangler:

- Dropdown-menuen mangler dynamik, da den ikke tager højde for om brugeren er logget ind; der står altid “Opret Profil” og “Log Ind”.
- Vores UserMapper-klasse indeholder alle “Mapper”-funktioner. Man kunne med fordel have fordelt metoderne i forskellige Mapper-klasser afhængigt af deres funktion.
- I Presentationlayer-pakken har vi klasser, der kun består af én metode-funktion, hvilket kunne have været fordelt ud. F.eks. har vi EmptyCart-klassen, som fordelagtigt kunne være blevet placeret i en Cart-klasse.