Rapport udarbejdet i perioden 10/10/19 - 31/10/19
Cupcake-projekt udarbejdet i perioden 10/10/19 - 31/10/19

Cupcake

Gruppe 3:

Andreas: cph-ap294@cphbusiness.dk, PetersenAndreas
Cahit: cph-cb342@cphbusiness.dk, Cabak06
Kristine: cph-kd109@cphbusiness.dk, KDalby
Marcus: cph-mj734@cphbusiness.dk, KumaSudosa
Michael: cph-mk548@cphbusiness.dk, Micniks

Links:

Github: https://github.com/KumaSudosa/Cupcake
Kode på server: http://68.183.211.81:8080/CupcakeProject/

Indledning

Projektet handler om at lave en hjemmeside til køb af cupcakes for en cupcake-virksomhed. I de næste afsnit vil vi komme ind på baggrunden for projektet, dvs. hvad firmaet forventer af programmet. Dernæst vil vi liste alle de teknologier vi har brugt til at udvikle programmet, og beskrive firmaets håb for projektet. Derefter har vi et afsnit med kommentarer til vores senere diagrammer mm. I afsnittet efter kommer vi ind på særlige forhold i opgaven, herunder validering af brugerinput, fejlhåndtering og sikkerhed. Dernæst et afsnit om mulige implementationer vi kunne tilføje programmet, og hvilke mangler det har. Til sidst vises alle diagrammerne og modellerne: Tilstandsdiagrammer, Domænemodel, ER-diagram, sekvensdiagrammer og aktivitetsdiagrammer

Baggrund

Systemet er udviklet til brug hos et firma der producerer og sælger cupcakes. Firmaet ønsker et system hvor kunden, efter at have oprettet sig i firmaets system, kan sammensætte to dele, bottoms og toppings, som har forskellige priser afhængigt af indhold, og på den måde kan lave deres egen cupcake.
Firmaet ønsker ikke at have nogen former for udbringning tilknyttet og har en cupcake-maskine der laver den pågældende sammensætning øjeblikkeligt, så kunden kan afhente sin ordre omgående.
Kunden betaler gennem sin balance på hjemmesiden, som ifølge opgavebeskrivelsen skal indbetales fysisk i butikken. Hvis der er utilstrækkelige midler på kundens konto, kan ordren ikke gennemføres og kunden afvises, indtil dennes ordre og midler, stemmer overens.
I systemet er der også lavet 2 administratorkonti som har tilladelse til at ændre brugeres balance samt se alle tidligere ordrer (invoices) for en eller alle brugere.
Slutteligt er det også muligt for kunden, når denne er logget ind, at se egne tidligere ordrer, som bliver gemt i en database.

Teknologi valg

Til at udvikle programmet har vi, som udviklere, benyttet følgende teknologier:

- Netbeans, version 8.2
- Maven, maven-compiler-plugin 3.1
- Junit, v. 4.12
- Org.hamcrest, hamcrest-core 1.3
- Javax, javaee-web-api 7.0
- Mysql, mysql-connector-java 8.0.17
- JDBC
- JSTL 1.2

Programmer uden fra projektet:

- MySQL Workbench, CE 8.0
- Git, Gitbash, v. 2.23.0 for windows

Krav

Firmaet håber på at systemet kan tilføre dem værdi ved at give kunden mulighed for at bestille online, og så blot oplyse ordrenummer (eller email) i butikken. Programmet giver ligeså firmaet mulighed for at holde styr på tidligere ordrer. Derudover kan det også bruges til firmaets revision da alle ordrerne, bliver gemt og kan tilgås gennem en administratorkonto. Til dette formål kan man også se datoen på hver ordre der er lagt.

Diagrammer

Billeder til alle diagrammer er gemt under bilag, og navngivet på billedet.

Domæne model og ER diagram

Domænemodellen viser en oversigt over alle pages brugeren kan opholde sig på, og hvordan brugeren navigerer rundt i programmet. Der er to muligheder i modellen fra login, som afgør om man er admin eller customer, hvilket så igen fører til hver sin navigationsvej.

I ER-diagrammet er det relevant at lægge mærke til “Test”-table, som udelukkende bruges til at teste på, og ikke er med i selve programmet.
Alle relationer i databasen er én-til-mange-relationer:
Flere Line-Items kan holde den samme bottom og topping (f.eks. 2 line-items kan begge holde topping nr. 1). Én invoice kan på samme måde holde flere Line-Items (Invoice nr.1 kan indeholde både lineitem 1 og lineitem 2). Slutteligt kan én user have mange invoices (lave mange ordrer). Disse overvejelser ligger til grund for relationerne i databasen.

Navigations Diagrammer

Som udgangspunkt er customer og admin to “forskellige” brugertyper, men de kører på de samme JSP sider, som ud fra brugerrollen viser deres forskellige funktioner. De benytter begge den samme login side, som er den eneste lighed, ellers er viewet forskelligt. For eksempel kan en customer kun se sine egne invoices hvorimod admin vil kunne se alle invoices for alle customer. Der benyttes en menu-bar som bruges til at navigere rundt mellem de forskellige JSP sider.

Sekvens Diagrammer

Grundet rapport-størrelse-begrænsning, vil vi kun forklare om main sekvensdiagrammet, som fundet under bilag **Buy Cupcake**.

Diagrammet er et succes sekvens diagram som fokuserer på processen af at købe en cupcake som customer aktør, fra product.jsp siden til færdig invoice på invoice.jsp siden.

Step 5: Check input validation
Checker om noget input er null eller om amount ikke fungerer som en int, og smider en IllegalArgumentException hvis der bliver fundet et problem.

Step 16: Check Customer Balance and ShoppingCart
Hvis shopping cart er tom, eller shopping cart total price er større end balance, går man stadig til confirmation.jsp, men man kan ikke vælge at confirme, og der bliver i stedet for vist en fejlbesked.

Særlige forhold

Vi gemmer brugeren i en session så programmet kan holde fast i hvem der er logget ind og vedkommendes oplysninger.

Exceptions håndteres primært gennem LoginExceptions samt IllegalArgumentExceptions.

Vores validering af brugerinput går på at kaste exceptions, at definere data-type og længden af denne data, brugeren må indtaste i de forskellige felter. F.eks. når brugeren tilføjer cupcakes til sin kurv, skal der minimum være en “amount” på 1 og højst en amount på 999, ellers bliver der kastet en exception.
Ligeledes bruger vi samme princip i passwordvalidering hvor vi også har defineret grænser for hvad brugeren må vælge. Vi har i programmet skrevet at passwordet skal indeholde minimum ét stort tegn, ét lille tegn og ét tal, samt mellem 6-20 karakterer før det accepteres. Hvis dette ikke opfyldes kastes der en exception og brugeren kastes tilbage til samme login/registration-page, med tomme felter, og en fejlmeddelelse der beder brugeren om at passwordet skal indeholde de fastsatte værdier.
Ved login tjekker vi brugerens indtastede data ved at sammenligne denne med værdierne i databasen. Hvis den indtastede data ikke er lig med hvad der står i databasen, sendes brugeren tilbage og får beskeden “No Matching Email/Wrong Password” og brugeren kan prøve igen.

En anden sikkerhedsforanstaltning vi benytter er at bruge post-metoder i stedet for get-metoder. Dette gør programmet mere sikkert eftersom man ikke kan se følsom information i URL’en og er tvunget til at følge flowet gennem programmet.
Det er ligeså en sikkerhed at vi ikke er i stand til at oprette nye administratorer gennem programmet - kun via databasen.

I databasen har vi oprettet alle brugere med en rolle - som er enten “c” eller “a” (som repræsentation for “customer” og “admin”), og er gemt som en varchar. Når programmet henter database-informationen ned, tjekker den alle brugerne igennem, derunder deres rolle, og hvis brugeren er customer så gennemgår han ét specifikt flow i programmet med bestemte rettigheder, og hvis brugeren er admin gennemgår han et andet flow, hvor han kun har specielle admin-rettigheder, hvor han ikke har mulighed for at købe cupcakes.

Status på implementation

Vi kunne godt have tænkt os at implementere en mulighed for at søge blandt invoices - både som bruger (hvis man har en stor mængde tidligere ordrer), og som admin hvor man får en stor datamængde. Her kunne det være rart f.eks. at kunne søge på dato eller navn/email.

På samme måde kunne det være rart at kunne sortere invoices efter samme princip. F.eks. trykke på en knap så listen sorteres efter “latest purchase” eller “highest amount”, hvilket ville gøre det mere overskueligt.

En anden mulig implementation kunne være at give admins tilladelse til at redigere og slette brugere, og evt. at kunne tilføje en yderligere admin-account gennem programmet (i stedet for kun at kunne oprette dem gennem databasen). Dette ville dog også medføre flere overvejelser ift. sikkerheden i vores program.
Vi kan også implementere en metode for admins at kunne ændre, eller fjerne, det nuværende bruger-startbeløb på 50 DKK. Dette er på nuværende tidspunkt kun muligt gennem koden, da værdien er hardcoded. Det kunne være ret for firmaet ikke at give 50 DKK væk gratis i resten af sin eksistens.

Som bruger kunne det være rart at have en metode til selv at indsætte penge på sin konto. For nuværende er det nødvendigt fysisk at møde op i butikken for at få en admin til at indsætte det. Dette fungerer, men er ikke ligefrem brugervenligt.
Ligeså kunne det være rart for en bruger at kunne tømme sin indkøbskurv med ét tryk på én knap, i stedet for at fjerne alle ordrerne individuelt. Dette er ikke en nødvendighed overhovedet, men en eventuel service til brugeren hvis denne f.eks. har 50 ordrer.

I forbindelse med opgave 9 har vi truffet nogle frie valg, blandt andet i forbindelse med valg af farver. Vi er i stand til at fremvise alle de ønskede principper, men har ikke opfyldt dem 100% designmæssigt.

Visse commands kan navngives anderledes for bedre at repræsentere flowet i programmet. På samme måde kan der være flere commands, da nogle af dem har flere funktioner, og dette ikke er optimalt. Derudover bør “shoppage”-JSP’en omnavngives til “customerpage”.

Tests

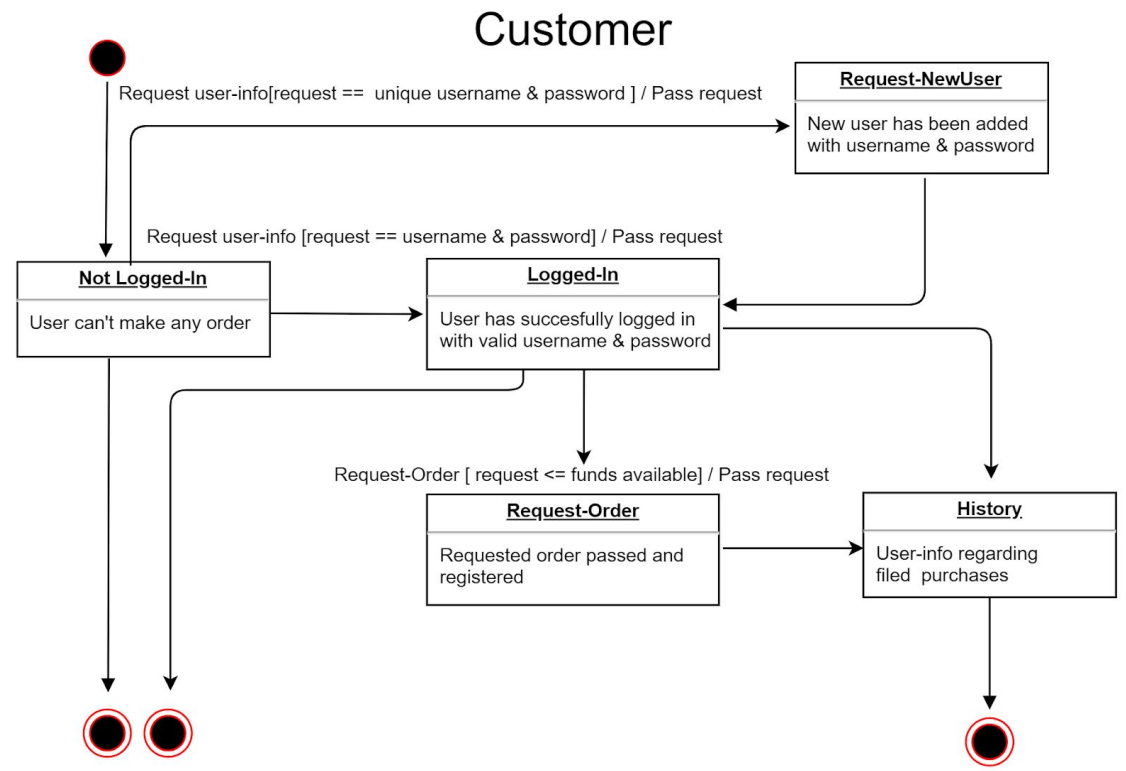
Projektet har ikke haft et stærkt fokus på tests, og har kun brugt Junit-tests. Junit-tests er blevet lavet med mock-objekter af mapper, som kommunikerer med SQL-databasen. Der er kun blevet lavet test til logic klasser.

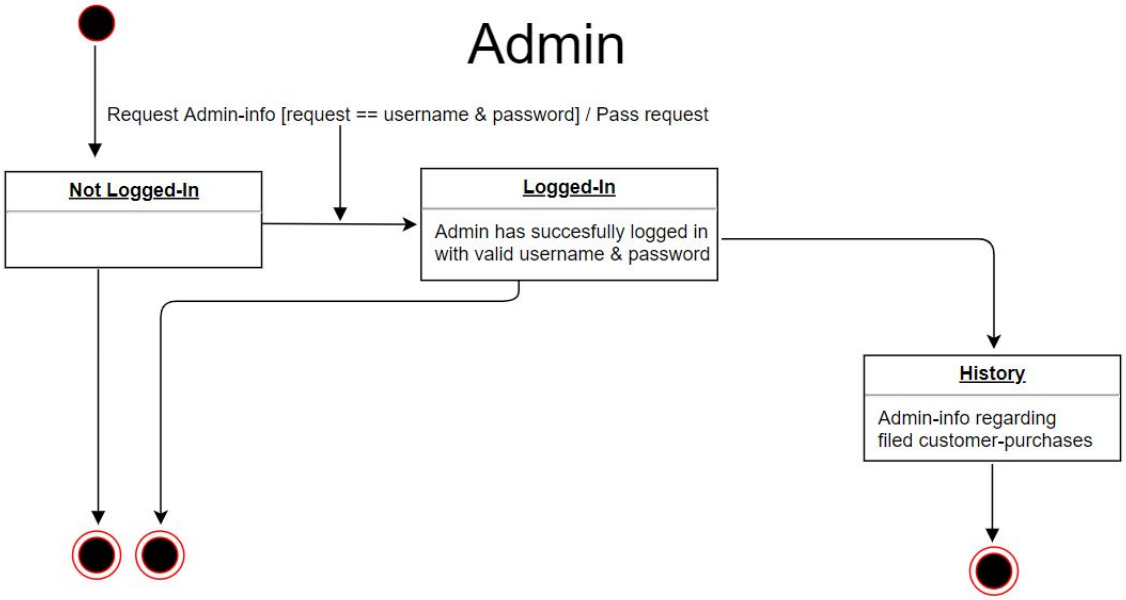
Følgende mock-klasser er lavet som bygger på interface for tilhørende klasser:

- FakeCupcakeMapper
- FakeInvoiceMapper
- FakeUserMapper

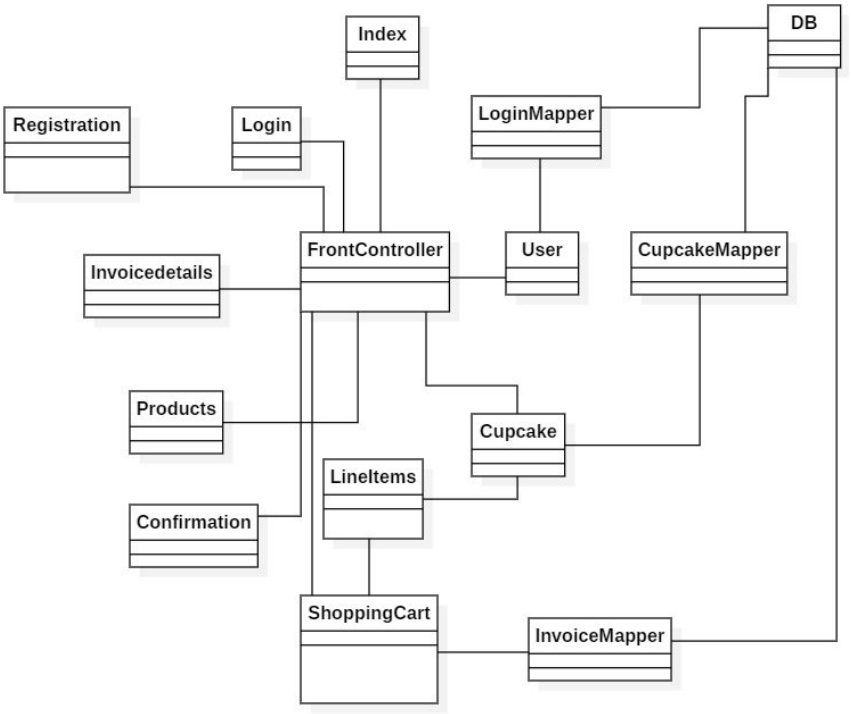
BILAG

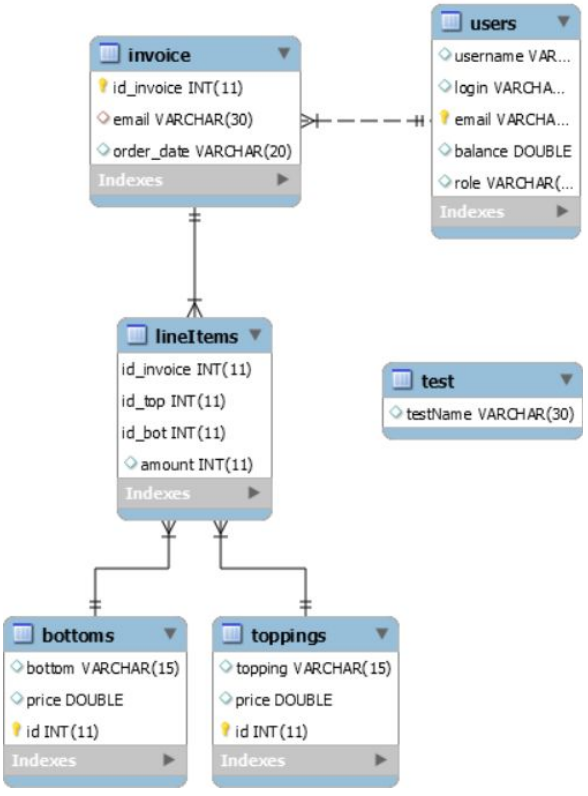
Tilstandsdiagrammer



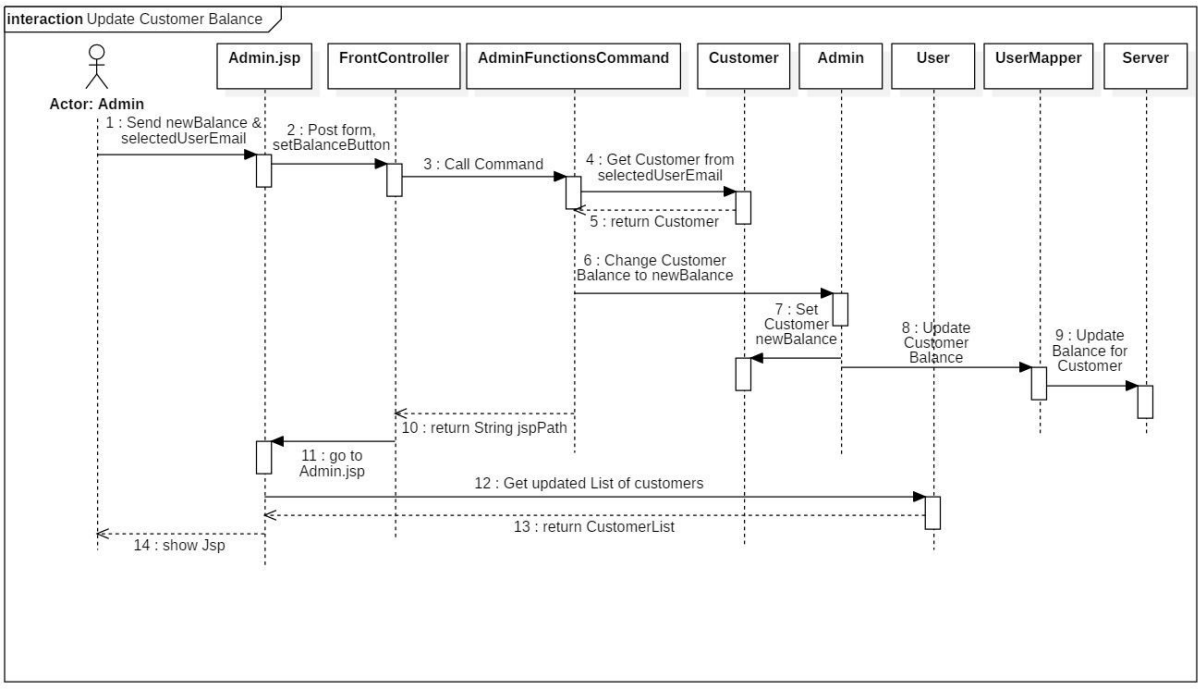


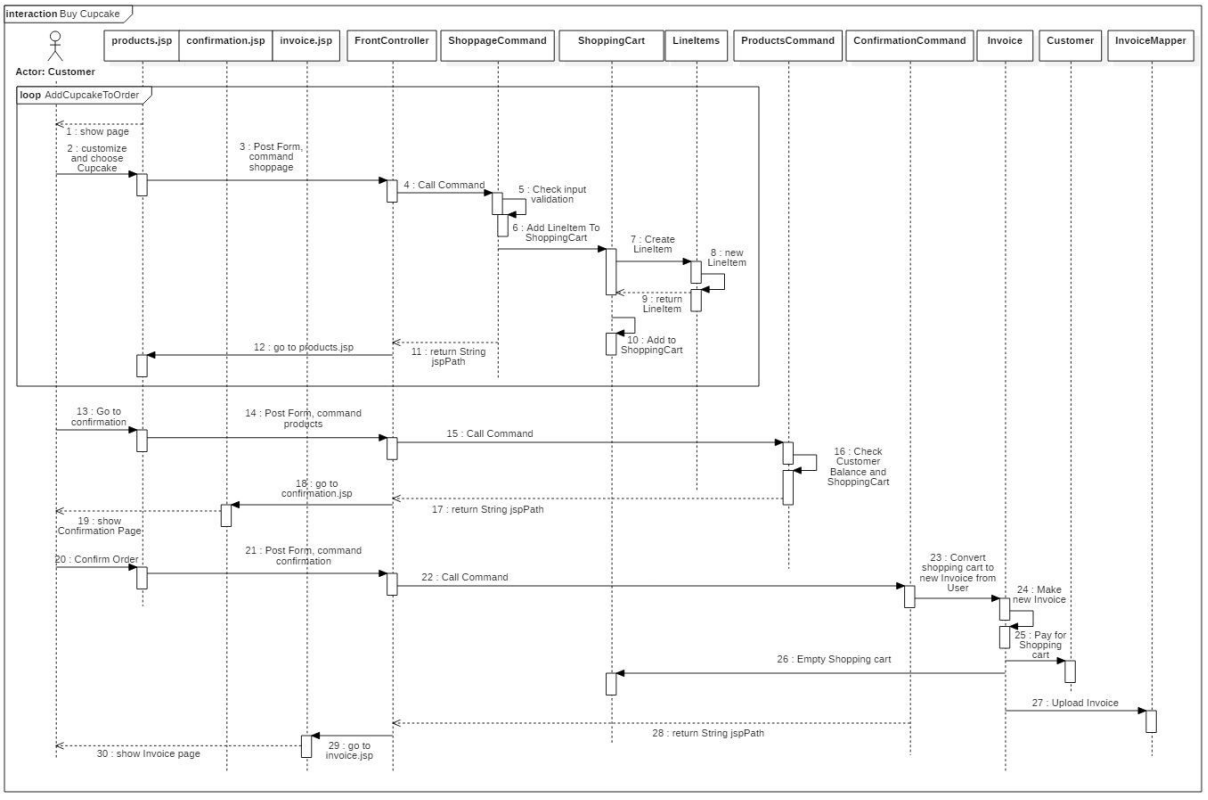
Domæne model og ER diagram

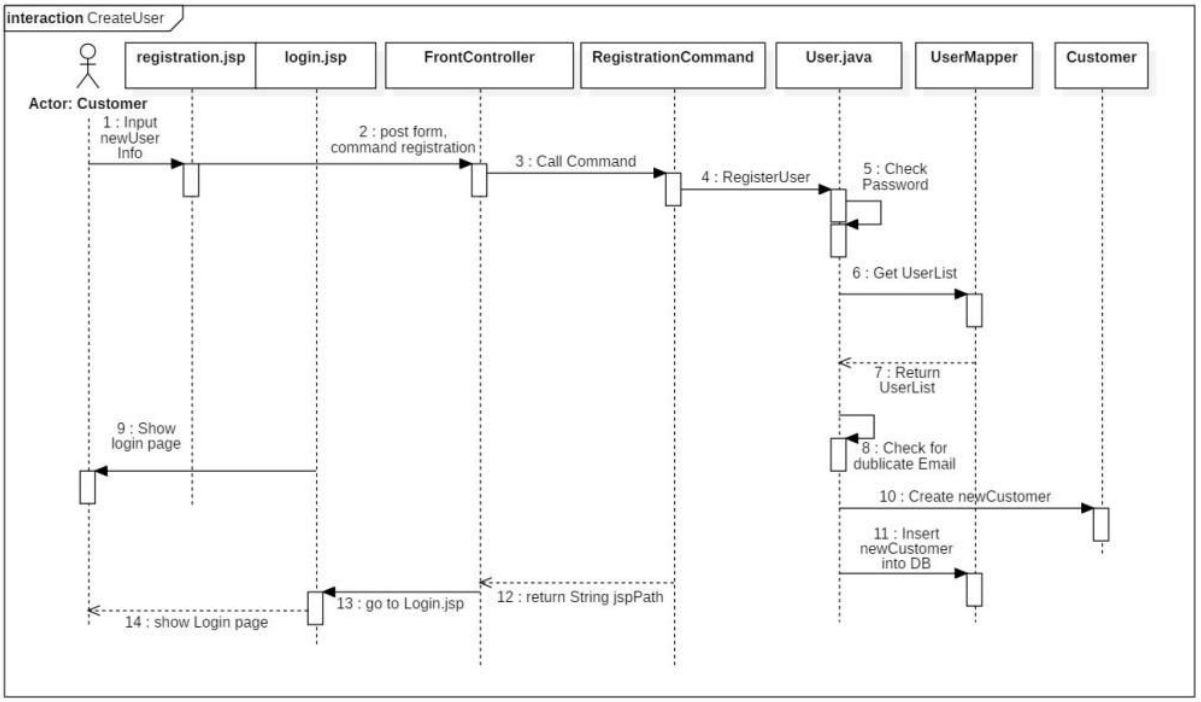


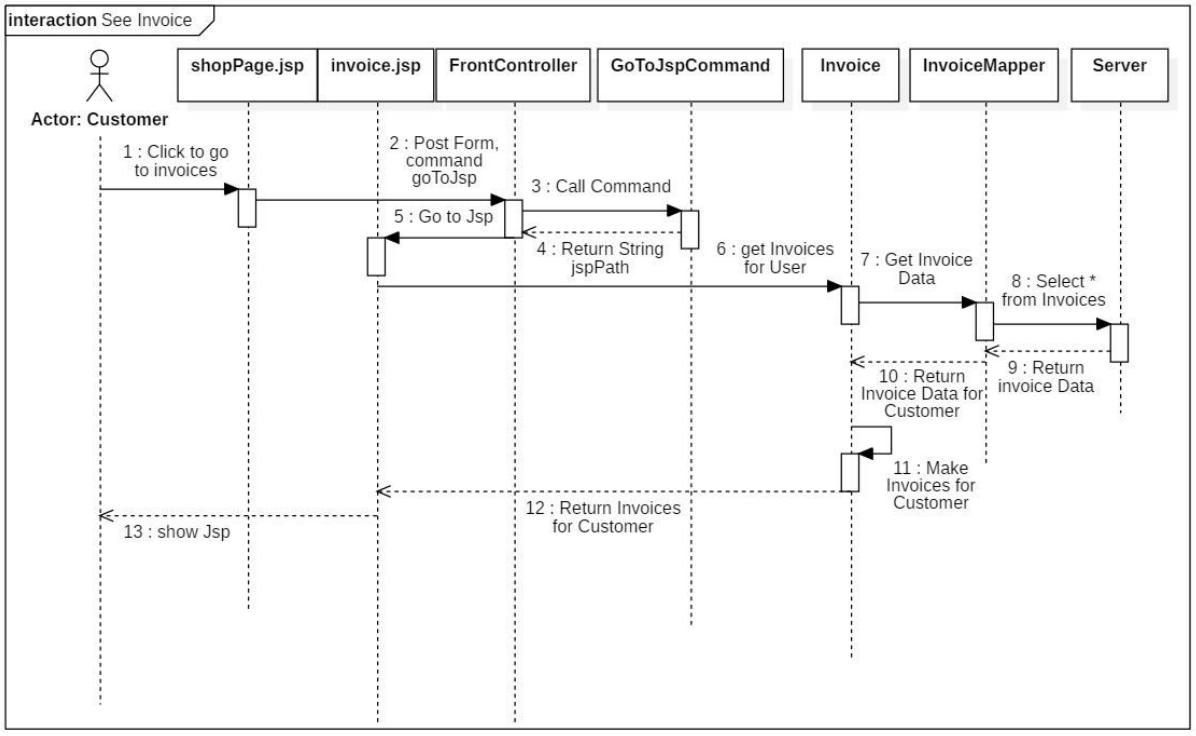


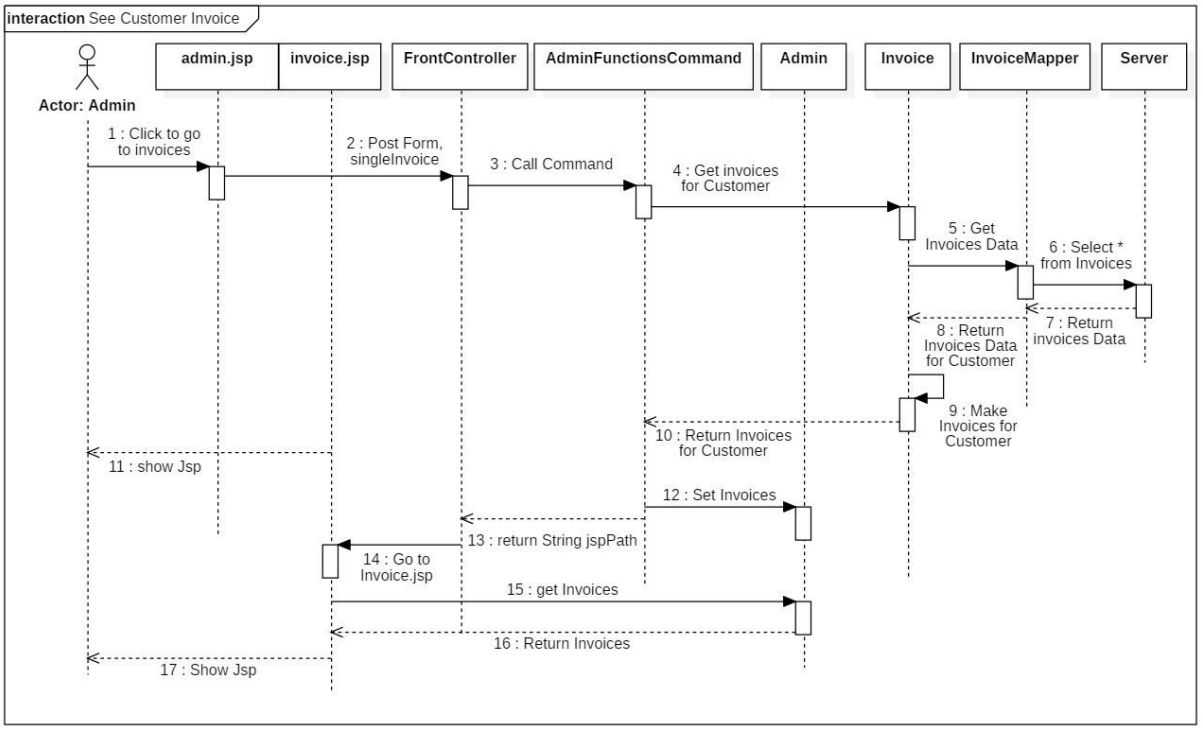
Sekvensdiagrammer

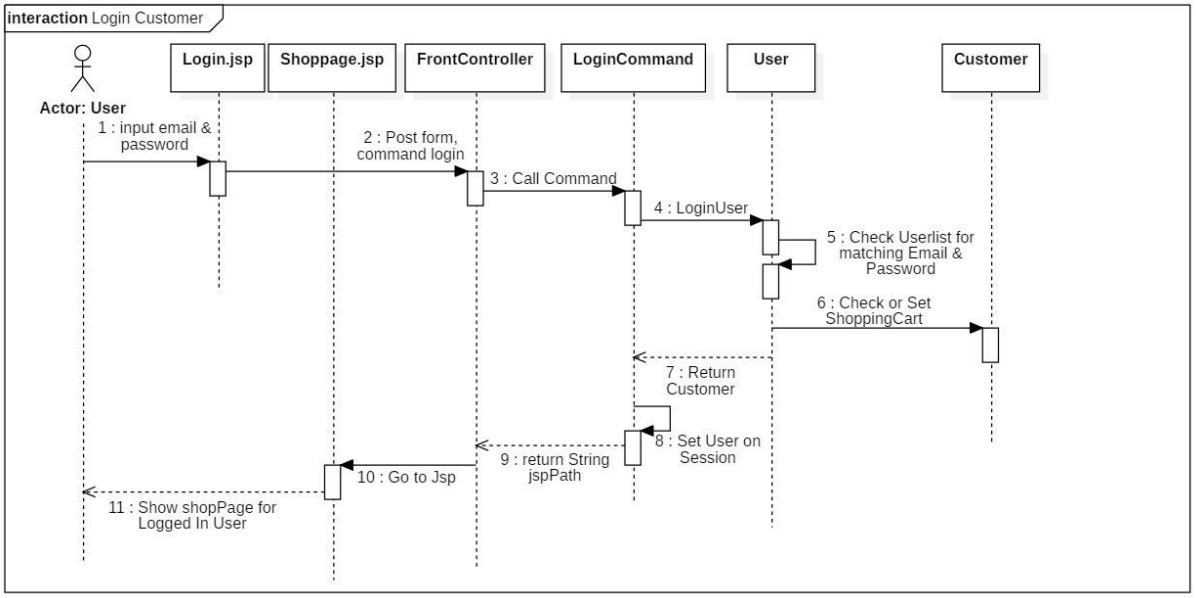


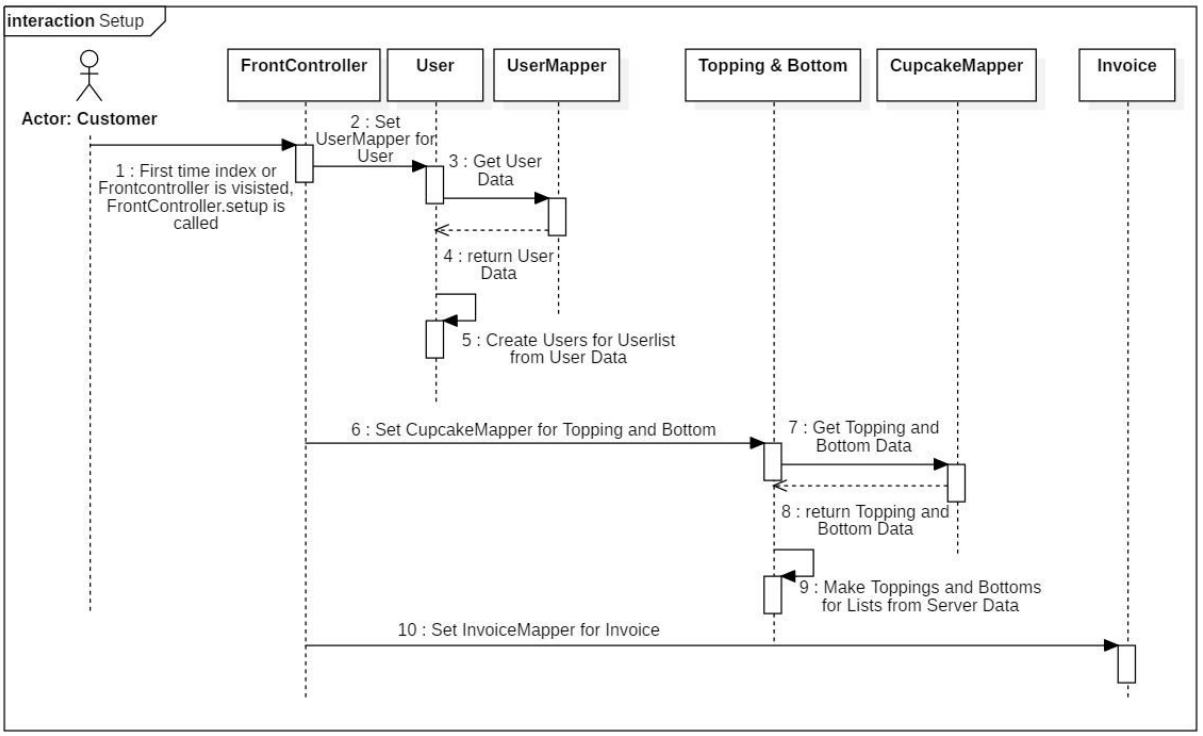



interaction CreateUser
registration.jsp
login.jsp
FrontController
RegistrationCommand
User.java
UserMapper
Customer
Actor: Customer
1 : Input newUser Info
2 : post form, command registration
3 : Call Command
4 : RegisterUser
5 : Check Password
6 : Get UserList
7 : Return UserList
8 : Check for dublicate Email
9 : Show login page
10 : Create newCustomer
11 : Insert newCustomer into DB
12 : return String jspPath
13 : go to Login.jsp
14 : show Login page


interaction See Invoice
shopPage.jsp
invoice.jsp
FrontController
GoToJspCommand
Invoice
InvoiceMapper
Server
Actor: Customer
1 : Click to go to invoices
2 : Post Form, command goToJsp
3 : Call Command
4 : Return String jspPath
5 : Go to Jsp
6 : get Invoices for User
7 : Get Invoice Data
8 : Select * from Invoices
9 : Return invoice Data
10 : Return Invoice Data for Customer
11 : Make Invoices for Customer
12 : Return Invoices for Customer
13 : show Jsp

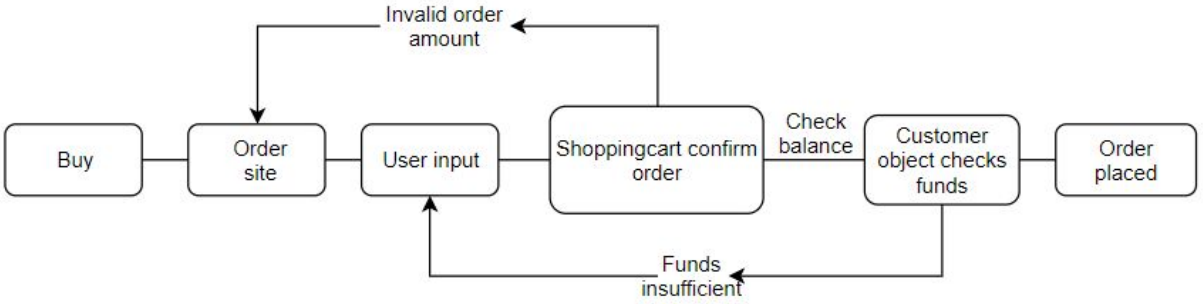

interaction See Customer Invoice
Actor: Admin
admin.jsp
invoice.jsp
FrontController
AdminFunctionsCommand
Admin
Invoice
InvoiceMapper
Server
1 : Click to go to invoices
2 : Post Form, singleInvoice
3 : Call Command
4 : Get invoices for Customer
5 : Get Invoices Data
6 : Select * from Invoices
7 : Return invoices Data
8 : Return Invoices Data for Customer
9 : Make Invoices for Customer
10 : Return Invoices for Customer
11 : show Jsp
12 : Set Invoices
13 : return String jspPath
14 : Go to Invoice.jsp
15 : get Invoices
16 : Return Invoices
17 : Show Jsp


interaction Login Customer
Actor: User
Login.jsp
Shoppage.jsp
FrontController
LoginCommand
User
Customer
1 : input email & password
2 : Post form, command login
3 : Call Command
4 : LoginUser
5 : Check Userlist for matching Email & Password
6 : Check or Set ShoppingCart
7 : Return Customer
8 : Set User on Session
9 : return String jspPath
10 : Go to Jsp
11 : Show shopPage for Logged In User

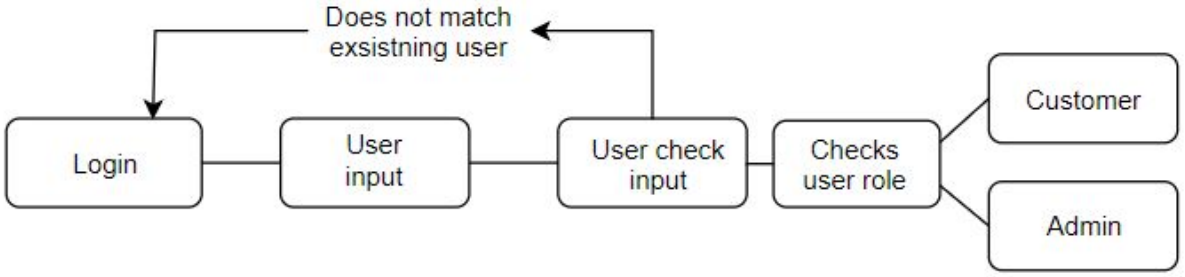

interaction Setup
Actor: Customer
FrontController
User
UserMapper
Topping & Bottom
CupcakeMapper
Invoice
1 : First time index or Frontcontroller is visisted, FrontController.setup is called
2 : Set UserMapper for User
3 : Get User Data
4 : return User Data
5 : Create Users for Userlist from User Data
6 : Set CupcakeMapper for Topping and Bottom
7 : Get Topping and Bottom Data
8 : return Topping and Bottom Data
9 : Make Toppings and Bottoms for Lists from Server Data
10 : Set InvoiceMapper for Invoice

Navigations Diagrammer

Køb:



Login:



Opret bruger: