CupCakes

Ordering Cupcake webshop

Af Malte Hviid-Magnussen, Benjamin Kongshaug, Mikkel Kornval Christoffersen og
Nikolaj Tost Søgaard,

Deltagere - Klasse A - Uge 9-10-11 - 2019
Malte Hviid-Magnussen, cph-mh748@cphbusiness.dk, MalteMagnussen
Benjamin Kongshaug, cph-bk131@cphbusiness.dk, kongshaug
Mikkel Kornval Christoffersen, cph-mc335@cphbusiness.dk, Kornvalles
Nikolaj Tost Søgaard, cph-ns169@cphbusiness.dk, Hiaku

Indholdsfortegnelse

Introduktion

I dette projekt har vi lavet en hjemmeside til en butik der sælger cupcakes. Kunderne bestiller online igennem vores hjemmeside, hvorefter de henter deres bestilte cupcakes i butikken.
En kunde kan på hjemmesiden oprette en bruger.
En oprettet bruger har en konto hvor man betaler sine cupcakes, ser sine invoices og bestiller cupcakes.
Hvis man er logget ind som administrator har man mulighed for at se alle invoices.

Baggrund

Virksomheden sælger cupcakes fra deres butik, cupcakes består af en top og en bund. Kunderne har mulighed for selv at sammensætte deres cupcake gennem hjemmesiden og henter dem derefter selv i butikken.

Vi har udviklet en simpel webshop som er skrevet i java og kodet vha. programmet Netbeans 8.2, ydermere har vi anvendt en MySQL database, java servlets og jsp pages i backend og html, css og javascript i frontend.

- Kunden skal via hjemmesiden kunne:
    - Oprette en bruger
    - Logge ind på sin bruger.
    - Sætte penge ind på sin konto.
    - Sammensætte en cupcake ved frit at vælge imellem top og bund.
    - Tilføje flere cupcakes til en kurv og bestille dem samlet.
    - Se en liste af faktura over sine bestilte cupcakes.

Alle krav er opfyldt.

Teknologivalg

Den software der er blevet anvendt til at udvikle projektet:

- NetBeans IDE 8.2 - Anvendt til at skrive JAVA, HTML, CSS og JavaScript.
- MySQL Workbench 8.0.15 - som database til vores brugere, cupcakes og invoices.
- JDBC 8.0.15 til at connecte til vores database.
- DigitalOcean Droplet til at hoste vores hjemmeside.

- Apache Tomcat 8.0.27.0 - Server
- JDK 1.8
- Git Bash version 2.20.1.1

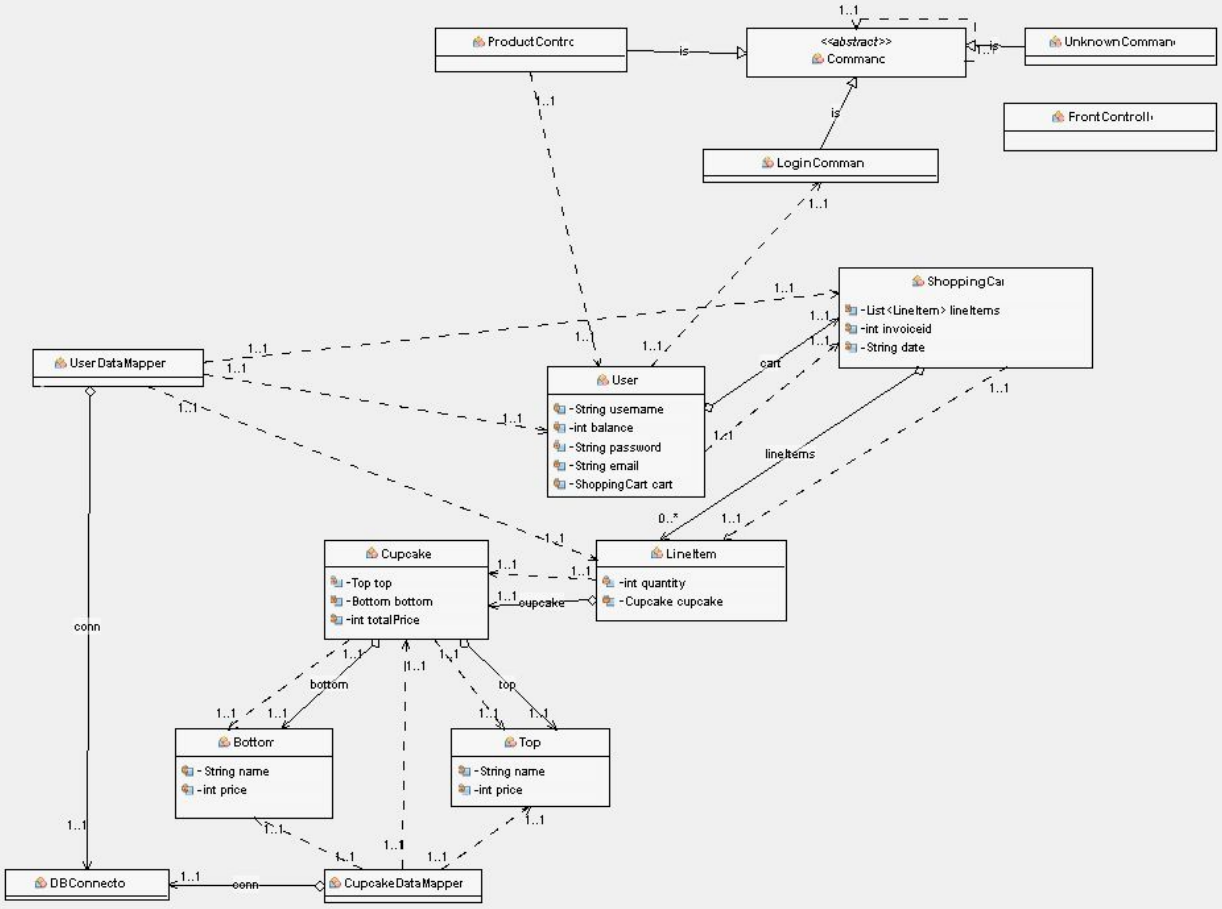
Krav:

Hvad er firmaets håb med dette system (hvad er deres vision for systemet eller hvilken værdi er det jeres system skal tilføre deres virksomhed):

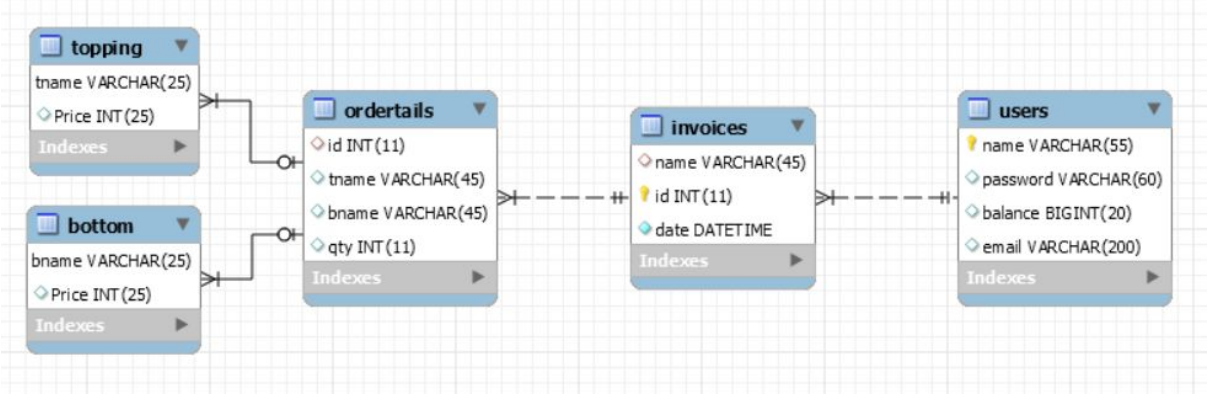
Firmaet kan nu udelukkende fokusere på at forberede Cupcakes, og kan spare lønnings timer ved kassen i butikken.
Kunderne køber og bestiller nu cupcakes via hjemmesiden og ikke længere i butikken. Dvs at kunderne ikke længere skal stå i kø for at bestille i butikken, og ikke skal vente på at deres custom cupcakes bliver forberedt. De er bare klar til afhentning i butikken.
Derudover har firmaet nu et sted de kan se alle faktura på en nem og overskuelig måde. Det vil klart spare administratoren noget tid og dermed penge. Derudover er der nu større muligheder for marketing, som de også kunne bruge til at starte en udleverings service.

Diagrammer:

Domain Model:

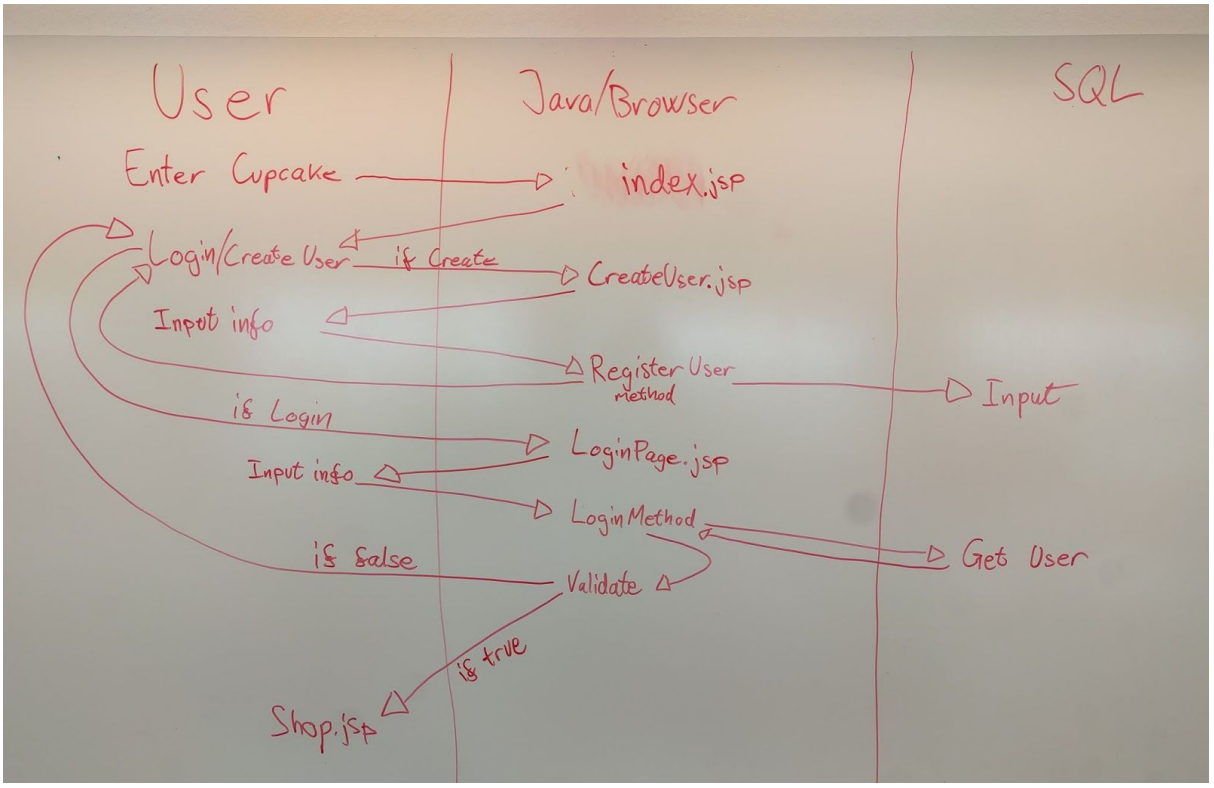


ER diagram:

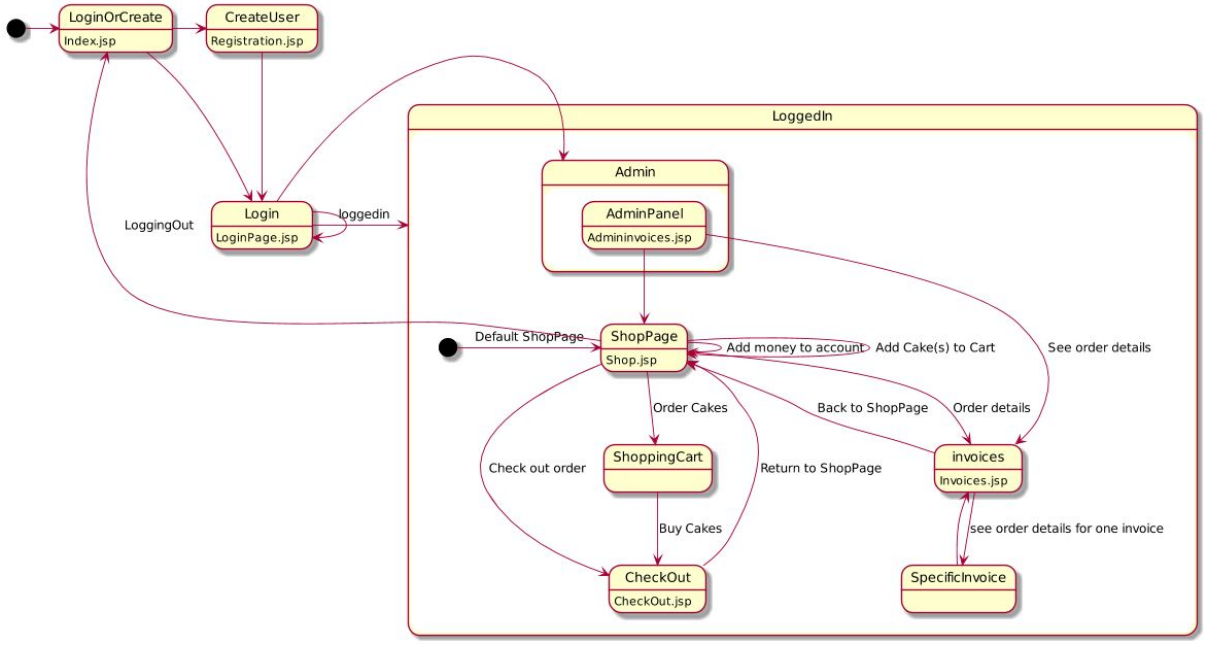


User tabellen har en 1...* relation til invoices.
Der er fremmednøgle på brugers navn.
Invoices har en 1...* relation til orderdetails.
Der er fremmednøgle mellem invoices ID og orderdetails ID.
En bruger kan altså have bestilt mange ordrer.
Og en ordre kan indeholde mange cupcakes.
På invoices har vi ID der ikke er autogenereret. Det kunne dog lige så godt have været auto-genereret.
På User tabellen er `name` en primary key. Den skal være unik. Det behandles som et username.
I invoices er `name` en fremmednøgle, og refererer til den User der har bestilt ordren. Date sættes automatisk på invoices.
I orderdetails tabellen er `id` en fremmednøgle der refererer til den invoice den tilhører.
OrderDetails repræsenterer en enkelt cupcake.
Der er 1 - 1 relation mellem toppings og orderdetails, samt mellem bottoms og orderdetails.
Invoice repræsenterer en ShoppingCart der indeholder mange cupcakes.

Aktivitets Diagram:

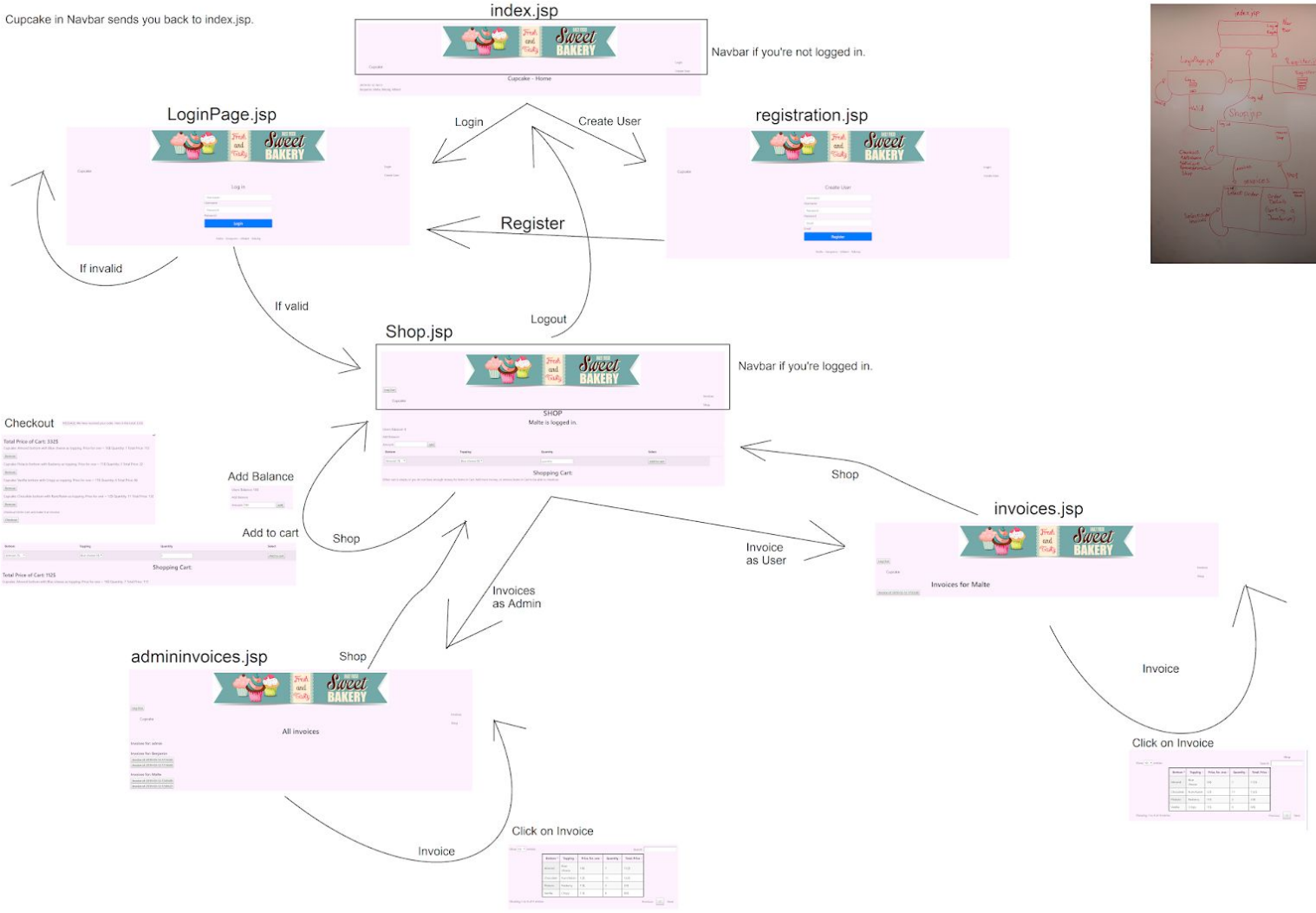


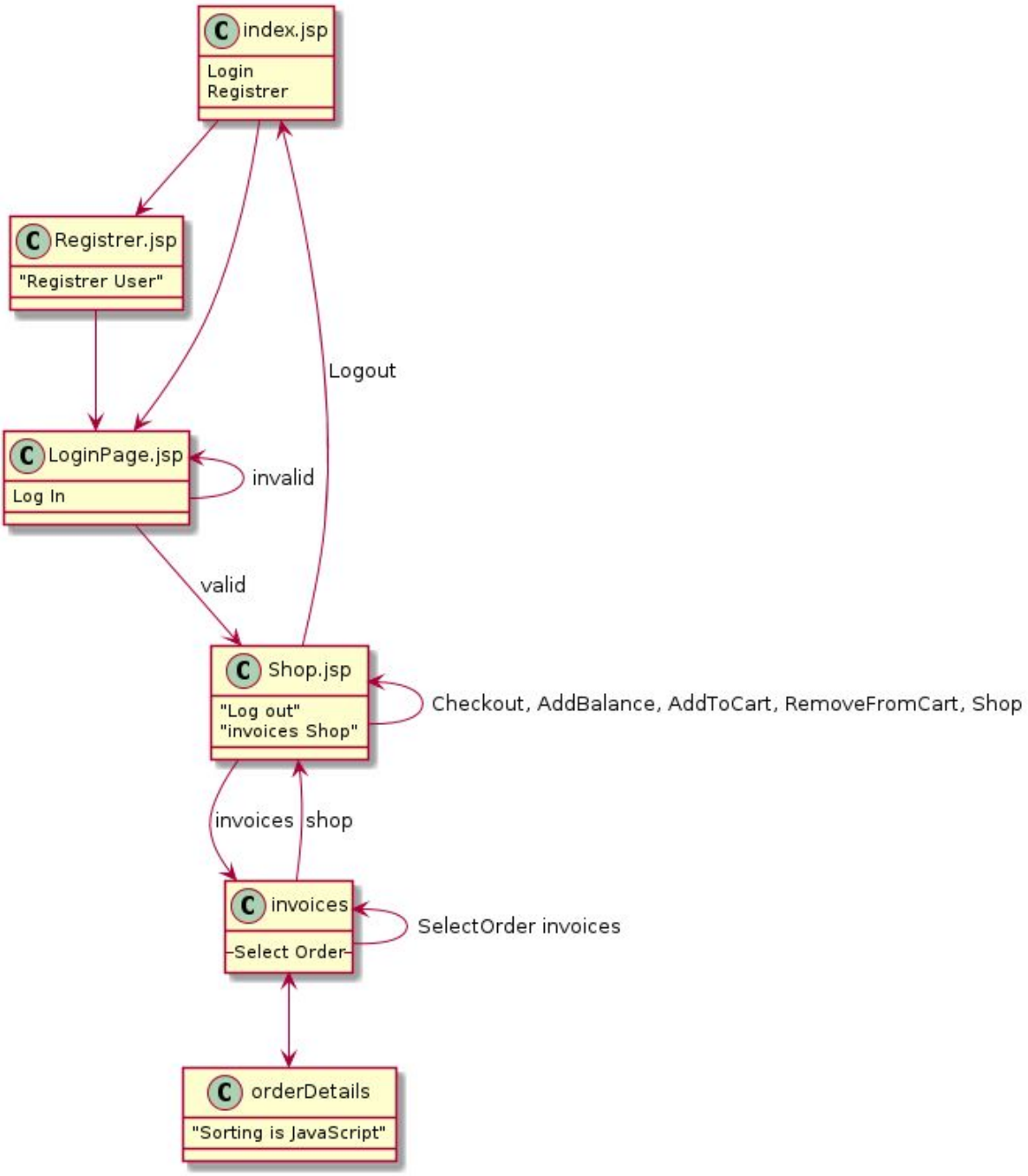
State Diagram:



[Link til Github]

Navigationsdiagram:

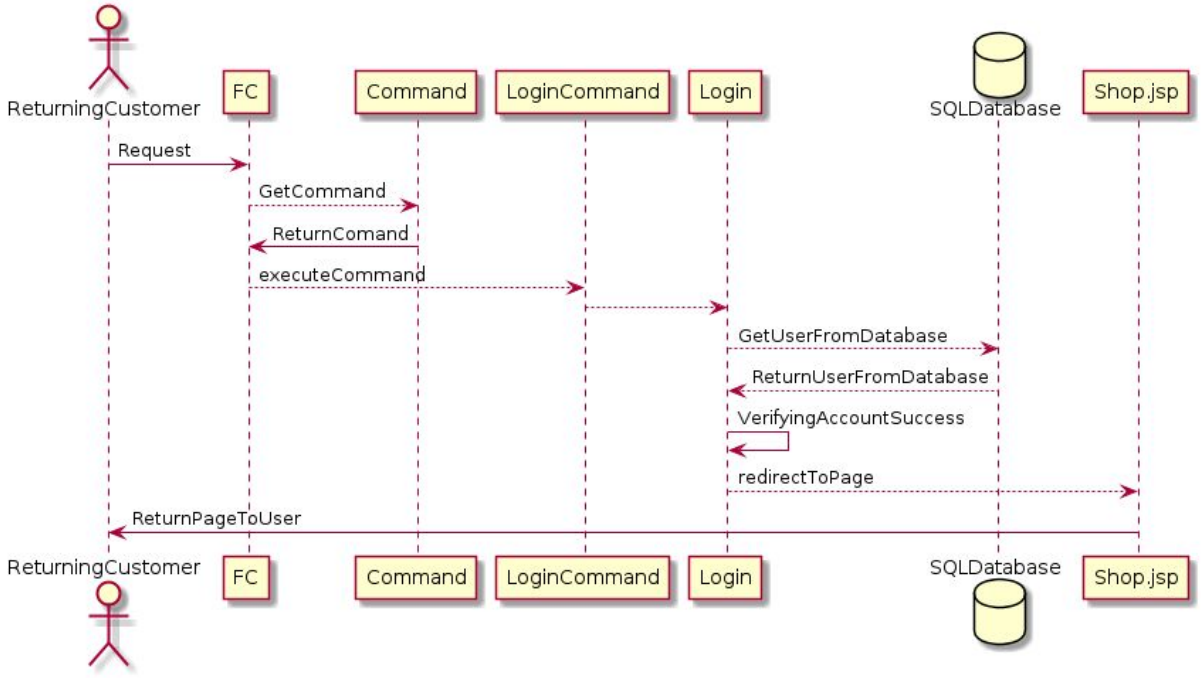




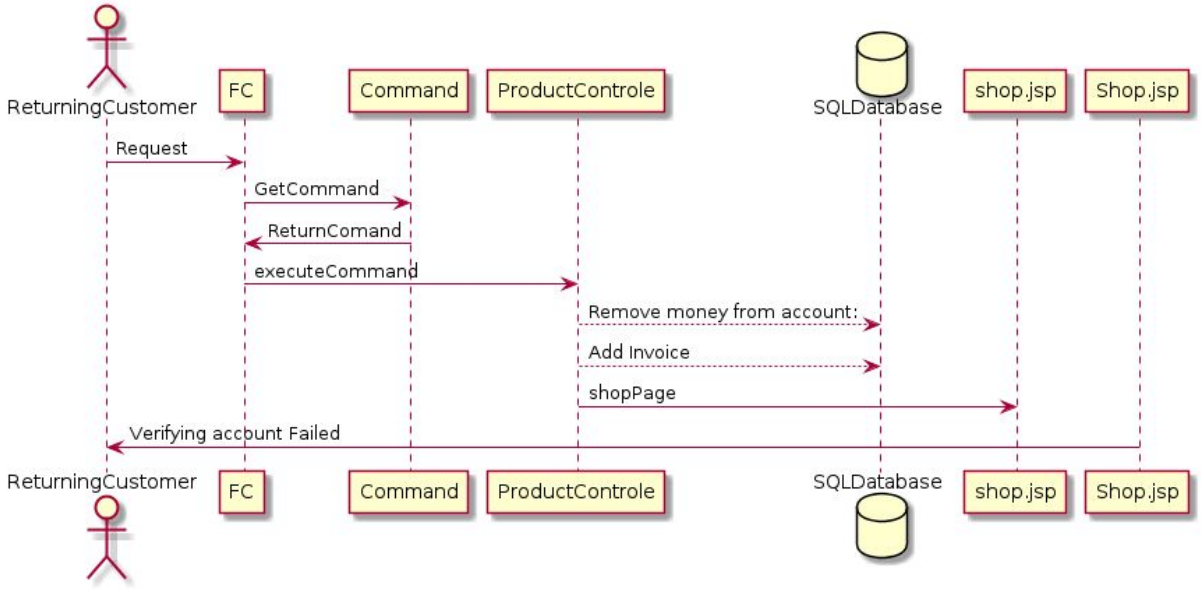
[Link til Github]

Sekvensdiagrammer:

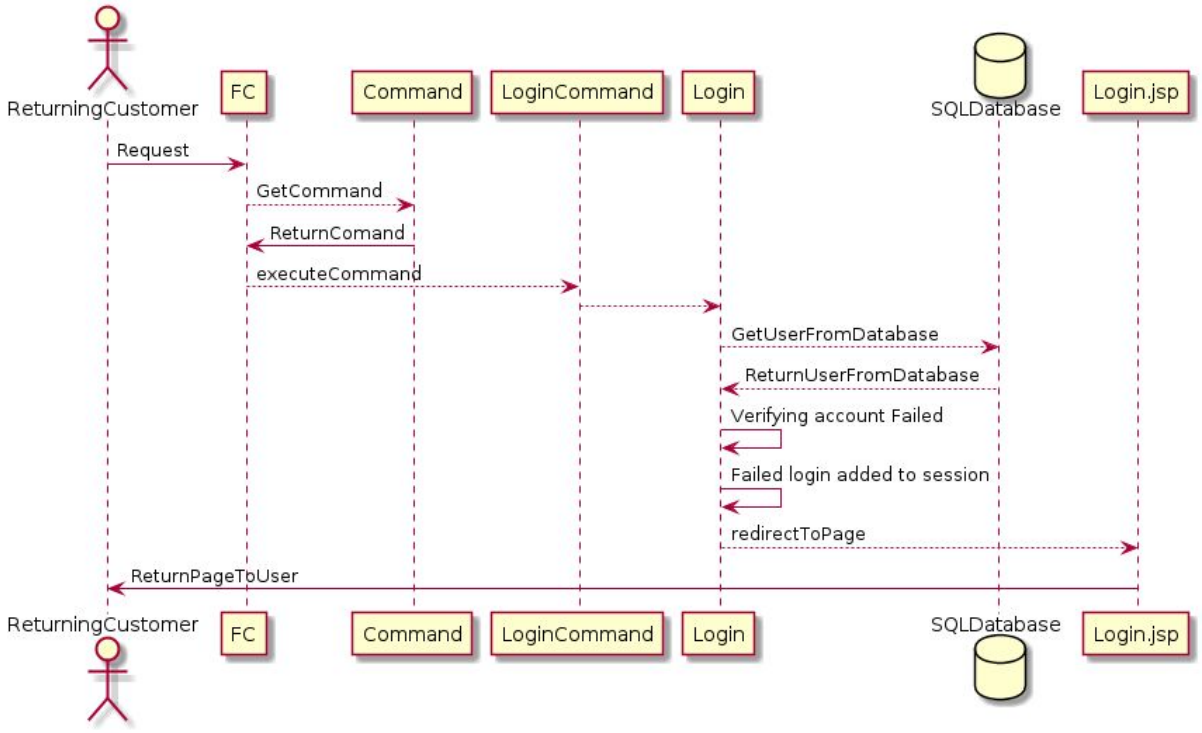
Sekvensdiagram for Login use case



Sekvensdiagram for Checkout af Shopping Cart



Sekvensdiagram for Login Failed Usecase



[Link til Github]

Særlige forhold

Desto mere der kan gemmes i Session og ikke i SQL, desto bedre. Det er meget dyrere at gemme noget i SQL, end det er at gemme det i session.

Vi gemmer brugerens data i sessionen når vedkommende logger ind. Hvis brugeren ikke findes i vores database gemmer vi en fejlbesked i sessionen. Brugerens data indeholder brugernavn, password, balance og email. Sessionen holder også på brugerens indkøbskurv.

Validering af brugerinput i Shop:
Når brugeren står på shop siden og har valgt en top og en bund til sin cupcake skal personen taste en quantity ind, hvis brugeren taster andet end et tal ind opstår en NumberFormatException i programmet når brugeren trykker "add to cart", håndteringen af denne error ses nedenfor:

```
try {
    int qty = (int) Integer.parseInt(
            (String) request.getParameter("qty")
    );
    lineitem.addQuantity(qty);
} catch (NumberFormatException e) {
    String errormessage = "Wrong input in Quantity field. Try again.";
    HttpSession session = request.getSession();
    session.setAttribute("errormessage", errormessage);
    return;
}
```

I vores catch sætter vi en errormessage på session. Brugeren returneres til shop siden hvor en error message står under topbanneret på siden med teksten: "Wrong input in quantity field. Try again."
Den sammensatte cupcake med forkert indtastet quantity er fjernet og brugeren kan nu prøve igen.

Validering af brugerinput i Login:

Når en bruger står på login siden og taster oplysninger ind kan det ske at brugeren ikke taster noget ind, kun udfylder et af de 2 felter eller taster forkert oplysninger ind.
Alle disse Error cases bliver håndteret på følgende måde:

```
private void login(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
    /* Get Parameters from the URL. (From the HTTP request) */
    String username = (String) request.getParameter("username");
    String password = (String) request.getParameter("password");

    /* Check if User exists in the SQL database */
    if (!StringUtils.isNullOrEmpty(password)
            && !StringUtils.isNullOrEmpty(username)) {

        try {
            /* check if user is valid */
            User user = new UserDataMapper().getUser(username);
            if (password.equals(user.getPassword())) {
                HttpSession session = request.getSession();
                /* Put user on session */
                session.setAttribute("user", user);
                /* Forward to Shop */
                RequestDispatcher rd = request.getRequestDispatcher("jsp/Shop.jsp");
                rd.forward(request, response);

            }
        } catch (SQLException ex) {
            Logger.getLogger(LoginCommand.class.getName()).log(Level.SEVERE, null, ex);
        }
```

I vores første if statement sikrer vi os at der er blevet indtastet data i formen. Det sikrer vores næste if statement for nullpointer exceptions. Hvis den indtastede data er forkert vil vores SQL kald returnere NULL værdier og forsøge at lave en User med NULL som name og password, for at undgå NullPointerException har vi fjernet constructoren fra User og derigennem sikret os at vores DataMapper ikke kommer med NullPointerExceptions.

Hvis brugerens input er korrekt og stemmer overens med Databasen, så bliver User sat på session og ført til Shop siden.

Brugertyper:
Der findes kunder og en admin. Normale brugere kan købe og se deres egne invoices, samt sætte flere penge på deres bruger. Admins kan alt hvad en normal bruger kan, og har derudover tilladelse til at se alle invoices over alle brugere.
I JDBC fungerer admin og en normal bruger ens. De behandles begge som User. Når en bruger skal have fat i alle sine egne faktura kaldes en metode der henter en liste af ShoppingCarts for 1 bruger.
Når admin skal have fat i alle brugeres faktura, kaldes først en metode der henter en liste af alle bruger og så loopes denne metode igennem for listen af Users `public List<ShoppingCart> getInvoices(User user)`
Læs videre i "Status på implementationen" hvordan vi ville have forbedret det.

Status på implementationen

Ting vi mangler fra opgavebeskrivelsen:
https://docs.google.com/document/d/1DH8Apv6kSGZJc_hFKnIRcaw96WFEwf7YJZNpnPjV_E8/edit#:

- In you database tables apply indices columns that are likely to be searched more often
- Make an automatic backup plan for your database that backup data every night
- Make an AJAX (fetch()) call to the the server to remove line item from the ShoppingCart object on the session.

Disse er ikke blevet implementeret pga tidsmangel og pga det er 3. semester pensum.

Andet vi kunne tænke os at implementere hvis der var mere tid:
Bedre validering af brugerinput.
Fx bedre forhindre SQL injections og bedre sørge for at der fx blev indtastet en email i email feltet. Lige nu er det fx fint at indtaste noget uden et @.

En bedre brugeroplevelse for admin.
Lige nu fungerer admin bare som en normal bruger, med den ene undtagelse at hans "invoices" link fører ham til admininvoices.jsp kontra invoices.jsp som en normal bruger ville se.
Derudover er admin desværre kun hardcoded ind i programmet. I User klassen bliver en bruger til admin hvis hans username er admin. Dvs der kan kun være en admin.
Sådan som vi ville implementere det hvis vi havde mere tid, ville være en ekstra række på User tabellen i SQL, der så bestemmer om vedkommende er admin eller bruger.
Så ville ansatte i butikken fx kunne have deres egen login der fungerede som admin, i stedet for at alle ansatte er nødt til at logge ind som admin for at kunne se faktura.
I vores side-header er der dette check:

```
} else if ("admin".equals(user.getRole())) {
```

for hvilken navbar der skal vises til brugeren.

Vi kunne godt have tænkt os at implementere en backup af databasen der ville ske automatisk ved slutningen af dagen.

Vi kunne måske også have tænkt os at lave en nogle færdiglavede cupcakes man kunne vælge, samt at se de mest populære cupcakes.

Det kunne også være smart hvis admin kunne slette brugere, og måske se ud for bruger hvornår de sidst har været logget på.

Test

- Klasser der er testet
  UserDataMapper og CupCakeDataMapper.
- Metoder der er testet
  UserDataMapper, GetUser, CupcakeDataMapper, GetTotalPrice, GetBottoms, GetTops, GetBottomPrice, GetTopPrice.

Vi burde nok have testet bredere, men der har i dette projekt ikke været fokus på tests, så vi besluttede kun at teste de vigtige DataMapper metoder, der bliver brugt ofte i resten af koden og er essentielle for at programmet virker.