CUP
CAKE

17. marts 2019

Cupcake

Jacob Folke Hildebrandt, cph-jh439@cphbusiness.dk, Leafmight
Simon Kristopher Kruse Bentzen, cph-sb365@cphbusiness.dk, Simonkruse2
Renz Oliver de Chavez, cph-rd78@cphbusiness.dk, dechavez4
Vincent Tran, cph-vt39@cphbusiness.dk, Vincent-CT

Copenhagen Business

`https://github.com/Simonkruse2/cupcakeShop.git`

B-Klassen, gruppe 5 - 17. marts 2019

CPH Business Lyngby

Indholdsfortegnelse

CPH Business Lyngby

Indledning

I cupcake projektet har vi arbejdet med programmering af en cupcake webshop. Projektet har indeholdt både opsætning af database i mySQL og frontend/backend webprogrammering. Vi har fulgt MVC modellen, for at gøre det hele mere overskueligt og organiseret, ved at anvende en DAO og controller til forbindelsen mellem databasen og webshop. Databasen er forsøgt normaliseret for at undgå redundans. Derudover har det været et krav at koden til projektet skulle oprettes som git repository, og at hvert medlem har deployed det færdige program på hver deres linux server.

Baggrund

Firmaet bag Cupcake-siden har tidligere sendt kager via mail, men dette fungerede ikke, da kagerne enten ikke blev modtaget eller der var taget bidder af kagen, så de er nu gået over til et online bestillingssystem med ”pick-up”.

De har en meget hurtig kage maskine så når ordren er lavet er kagen klar til afhentning. Kunden skal være bruger for at lave en bestilling, i toppen står kundens kredit, som kan tilføjes efter behov. Når kunden er logget ind, kan man vælge de forskellige kombinationer af ”toppings” og ”bottom” som man vil have og den mængde af kombinationen man vil have. Når kunden er færdig, kan han/hun gå til betaling og ordren er derefter klar til afhentning.

Teknologivalg

- Netbeans 8.2
- Tomcat 8.0.27.0
- MySQLWorkbench 8.0.12.CE
- Ubuntu Server som er hosted på Digital Ocean

Krav

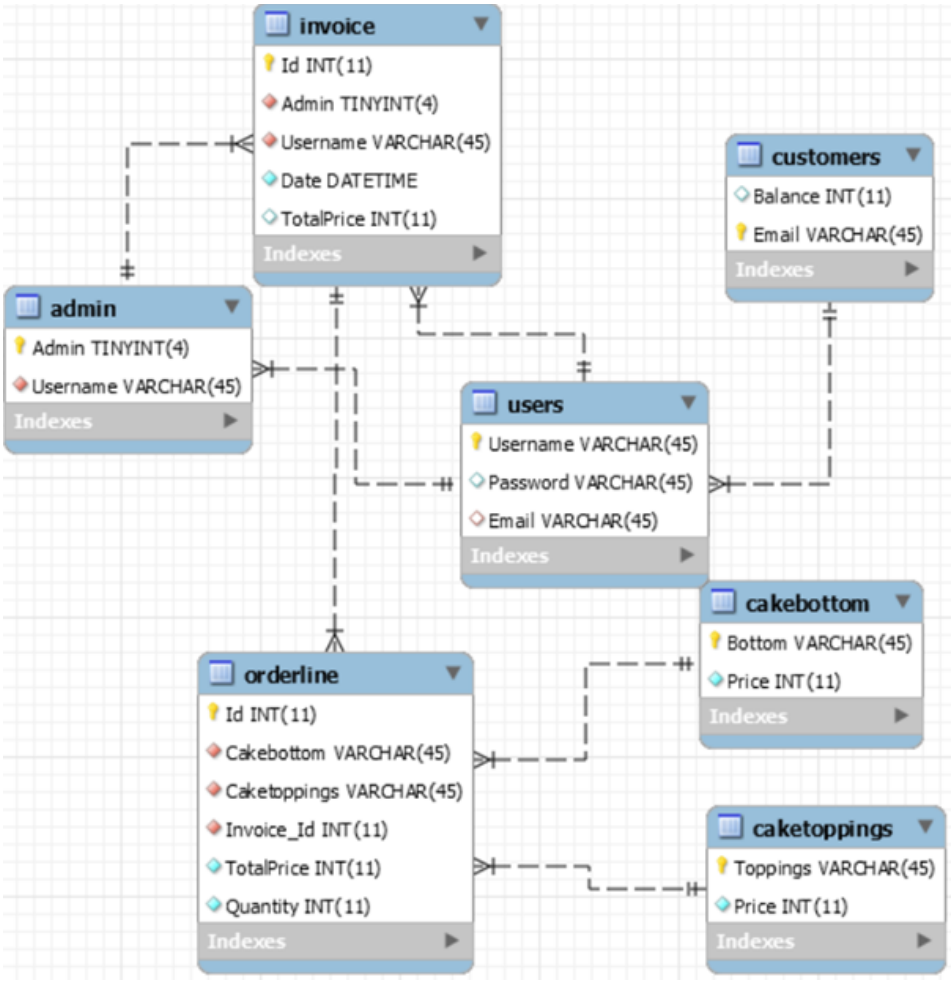
Firmaet forventer at der er et login system, som er forbundet til en database, der holder på disse brugere. Hvis kunden ikke allerede er bruger, skal de kunne oprette en. Brugere kan derefter gå ind på shoppen og se de forskellige varianter af toppings og bunde samt priser. Når de har valgt kombinationen, skal kunden kunne vælge den mængde af kagerne, der ønskes. Der skal være en ”balance”, som kunden kan se og tilføje til efter behov. Hvis der er penge nok, kan de gå til betaling og færdiggøre deres bestilling, hvorefter Admin (ejeren/ansatte) kan gå ind og se ordrerne så kagerne kan blive lavet klar til kunden med det samme.

ER diagram

Vi har til vores program valgt at anvende mySQL som database. Vi har af flere omgange været nødt til at lave nogle enkelte tabeller om, da programmets vidde var blevet undervurderet. De forskellige iterationer har været lavet ved både at skrive scriptet i hånden, men også vha. mySQLs modelgenerator. Det endelige ER diagram er genereret vha. reverse-engineering. Vi valgte at tabellen ’users’ kun skulle indeholde et username, som primary key, et password og en email. E-mailen er sat til at være en foreign key, oprettet i customers. ’customers’ indeholder en balance, som er en integer (int), og en e-mail, som er primary key. Vi valgte den her fordeling, da vi, ved projektets start, ikke mente at en admin havde brug for hverken en e-mail eller en balance og det var derfor nærliggende at dele de to op. Efterfølgende har vi diskuteret om det ikke ville give mening, at admin reelt set bare havde et kæmpe beløb på sin konto til at teste systemet og en e-mail, hvis nu man på et senere tidspunkt gerne ville sende nyhedsbreve eller lignende ud. Balance er sat som en int, da alle vores priser er heltal, men efter at have tænkt det igennem, ville det give god mening at det var en double i stedet, da priserne, f.eks. ved et udsalg, sagtens kan ende med at indeholde decimaler.
’admin’ tabellen indeholder en admin af typen boolean, der egentlig bare tjekker om vedkommende er admin eller ej. Primary key i tabellen er admin, hvilket set i bakspejlet er en dårlig idé, da primary keys jo skal være unikke og med det setup vi har valgt, kan man ikke have flere admins. Ydermere indeholder tabellen også et username, der er foreign key til ’users’ tabellen.
’invoice’ tabellen indeholder et unikt id, der er sat til at auto-incremente. Tabellen har to foreign keys, username og admin, der er sat i henholdsvis ’users’ og ’admin’ tabellen. Ligeledes bliver der automatisk sat et tidspunkt for hvornår invoicen er blevet oprettet, ud fra Ubuntu serverens egen systemtid. Tabellen har også en totalPrice, af typen int, som blev udregnet ud fra antallet af

cupcakes og den individuelle cupcakes pris. Ligesom ovenfor, ville det også her give mening at det var af typen double i stedet for int.

'orderline' tabellen indeholder også et unikt id, der er sat til at auto-incremente. I 'orderline' er der tre foreign keys i tabellen, cakebottom, caketoppings og invoice_id, de stammer fra henholdsvis cakebottom, caketoppings og invoice. TotalPrice, af typen int, er udregnet ved at plusse prisen fra cakebottom med prisen fra caketoppings og quantity, af typen int, er antallet af den givne kombination af cakebottom og caketoppings. Vores overordnede idé med denne tabel var at hver enkelt orderline skulle tilføjes til den samme invoice, så en kunde havde mulighed for at købe op til flere forskellige cupcakes på samme tidspunkt, men på nuværende tidspunkt har det ikke været en mulighed. Både 'cakebottom' og 'caketoppings' består blot af et navn på enten en bund eller en top og en pris på selvsamme.



Navigationsdiagram

Create Customer

Det første kunden vil se er login-siden ”index.jsp”, hvor der er mulighed for at indtaste sit brugernavn og sin dertilhørende kode og trykke på ”Sign in”. Yderligere er der mulighed for at klikke på ”click here if you’re not already a user”. Dette vil tage kunden til en ny side ”createCustomer.jsp”, hvor man kan indtaste hhv. E-mail (E-mail), Brugernavn (Username) og Adgangskode (Password). Når kunden har udfyldt disse 3 felter, kan der klikkes på ”CREATE”, hvor brugeren derefter vil blive oprettet/registreret. Kunden burde kun oprettes, hvis e-mail er korrekt (skal indeholde ”@” og ”.”), hvilket ikke er blevet implementeret endnu, og hvis hverken e-mail eller brugernavn ikke allerede er i brug. Hvis dette er tilfældet, vil kunden få en fejlbesked (evt. i form af en ny side, hvor kunden får at vide om det er brugernavnet eller e-mailen som er taget).

Index

Yderligere kan kunden også have trykket forkert til at starte med og allerede var registreret med en bruger, kan han/hun klikke på ”login page”, hvor man bliver taget tilbage til login-siden ”index.jsp”. Når der trykkes på ”Sign in” på ”index.jsp”-siden vil der være et tjek for om brugernavnet og kodeordet passer med den der er registreret i databasen. Hvis det ikke passer, vil der komme en fejlbesked (evt. i form af en ny side, hvor der vil stå at kodeord eller brugernavn er forkert). Hvis kodeord og brugernavn passer med det der ligger i databasen, vil brugeren tages videre til ”shop.jsp”-siden.

Shoppen

Her har kunden mulighed for at vælge de kombinationer af kager der ønskes (toppings og bottom), hvor der er vist en dertilhørende pris til hver smag. Der kan indtastes et antal af den kage der ønskes, hvorefter der kan klikkes på ”Select”, og ordren vil blive vist i tabellen nedenfor. Når kunden er færdig med at tilføje kager til sin liste, kan her vælges om ordren, der er vist i tabellen skal gennemføres. På toppen af siden er der vist, hvor meget kunden har på sin konto (”Balance”), her kan der tilføjes efter behov/løst. Hvis kunden ikke har nok penge på sin konto burde kunden en fejlbesked om dette og købet bliver dermed ikke registreret (evt. i form af ny jsp-side). Der kan derefter tilføjes penge til kontoen og købet kan nu gennemføres.

Admin

Når der logges ind, tjekkes der for om brugeren er ”Admin”, hvis dette er tilfældet, vil brugeren i stedet blive vist til siden ”admin.jsp”. Her kan admin se både Invoices og OrderLines. Invoices er en liste af brugere, der linker listen af ordrerne til en bruger via et id.

Servlet/controller:

Alle disse sideskift sker via en servlet ”controller.java”, der indeholder en switch, denne er vist nedenunder.

```
String origin = request.getParameter("origin");
switch (origin) {
    case "index":
        index(request, response);
        break;
    case "login":
        login(request, response);
        break;
    case "createCustomer":
        createCustomer(request, response);
        break;
    case "makeCustomer":
        makeCustomer(request, response);
        break;
    case "checkLogin":
        checkLogin(request, response);
        break;
    case "shop":
        shop(request, response);
        break;
    case "admin":
        admin(request, response);
        break;
    case "Invoice":
        Invoice(request, response);
        break;
    case "balance":
        updateBalance(request, response);
    default:
        throw new AssertionError();
}
```

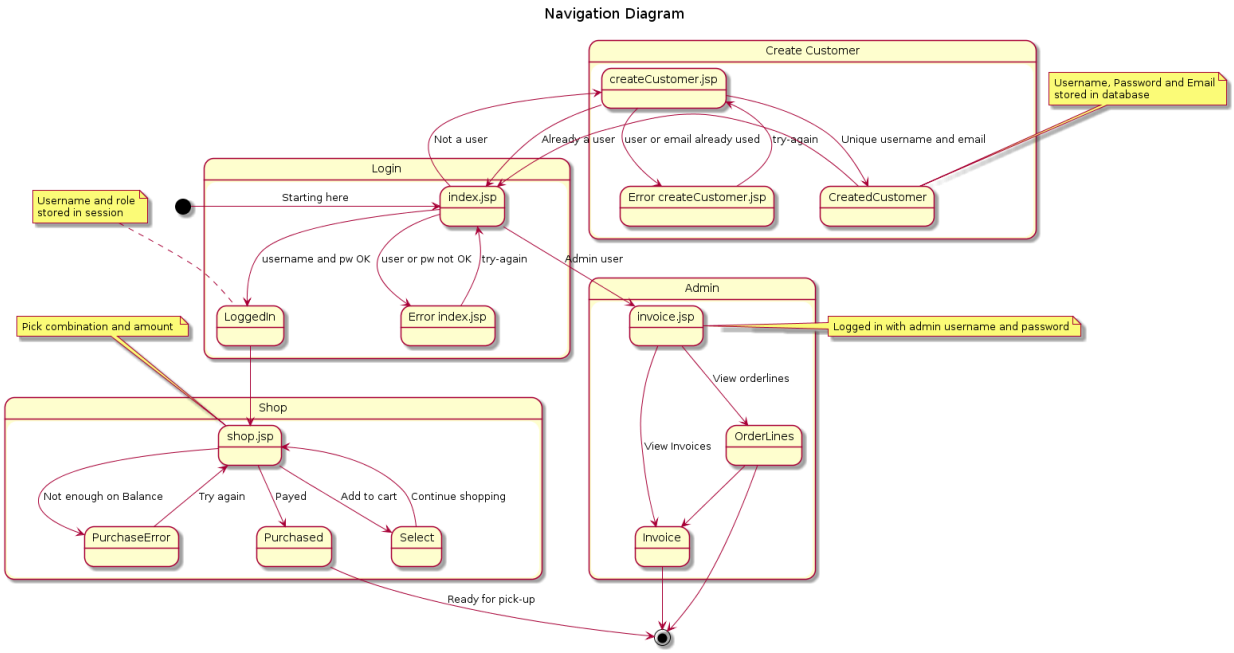
Her ses at når ”origin” får sat parameter til f.eks. ”createCustomer”, vil createCustomer-metoden blive kaldt. Denne metode henter de 3 input som kunden har givet (hhv. e-mail, brugernavn og kodeord). Hvorefter der laves en customer og en user med de givne værdier. Til sidst bliver der kaldt på index-siden igen, hvor kunden vil blive vist til startsiden. Dette er vist på nedenstående billede.

```
private void makeCustomer(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    String email = request.getParameter("email");
    String username = request.getParameter("username");
    String password = request.getParameter("password");

    d.createCustomer(email);
    d.createUser(username, password, email);

    request.getRequestDispatcher("index.jsp").forward(request, response);
}
```

Selve origin bliver sat i jsp-siderne, hvilket er vist nedenunder:

```
<a href="Controller?origin=createCustomer" >click here if you're not already a user</a>
```

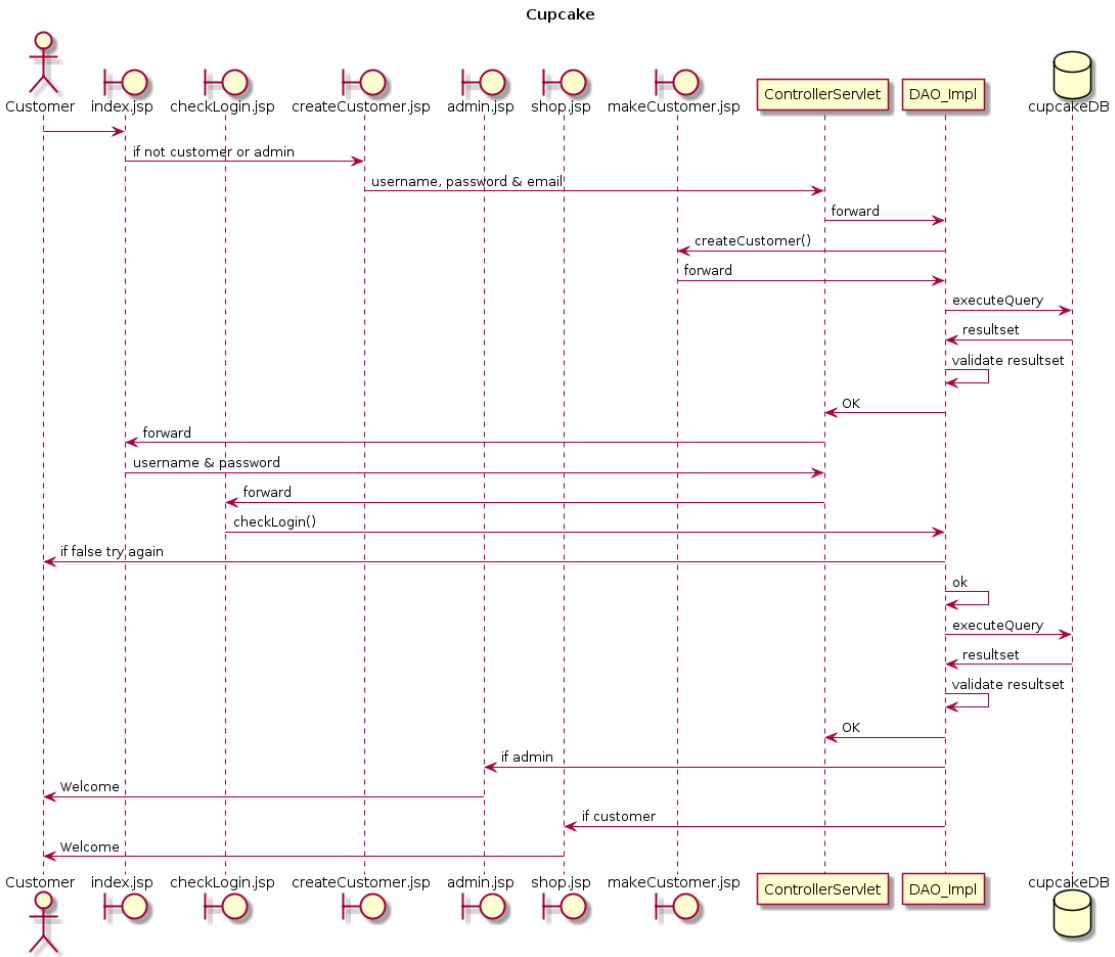
Dette er det samme for alle tilfælde for sideskift, selve eksemplet ovenover viser sideskiftet til når brugeren skal lave en bruger, altså skifter fra index.jsp til createCustomer.jsp.
Hele Navigationsdiagrammet er vist på nedenstående billede, hvor der ikke er inkluderet servlet, da dette gør diagrammet meget uoverskueligt.



Sekvens diagrammer

På figur 1 ses et sekvensdiagram over hvordan programmet overordnet virker. Vi tager udgangspunkt i startsiden index.jsp, der er den første side som man rammer ved opstart. Herfra er der mulighed for at logge ind med username og password, hvis man allerede er oprettet som kunde eller admin. Er dette ikke er tilfældet kan man komme videre til createCustomer.jsp, ved at vælge opret ny bruger, for at oprette sig som customer. Vi har valgt at anvende en frontcontroller som blot indeholder en switch-case til at oprette servlet og omdirigere brugeren til den valgte jsp-side. Som det fremgår af sekvensdiagrammet på nedenstående figue starter customer ved index.jsp og vi antager at brugeren ikke allerede er en customer eller admin. Der vælges derfor opret ny bruger. Dette link sætter vores switchcase parameter i controlleren til at være lig med createCustomer, som kalder metoden createCustomer(). Denne metode laver et HttpServletrequest/response, og forwarder request/response til createCustomer.jsp. Herfra har vi

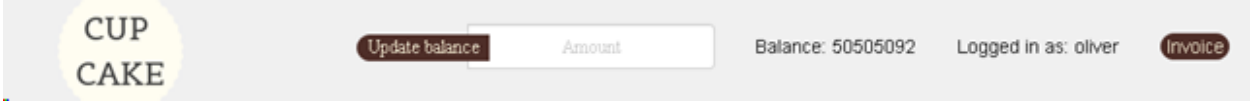
2 input felter som prompter brugeren om username, password og email. Disse parametre forwardes til DataAccessObject_Impl, som indeholder metoder til at hente eller sende query's til databasen cupcake. Metoden createCustomer tjekker med databasen om brugeren allerede eksisterer, hvis dette ikke er tilfældet oprettes brugeren med de tre parametre i databasen og omdirigere brugeren tilbage til Controlleren, som sætter parametrene til index. Brugeren er nu tilbage på index-siden og kan nu logge ind med username og password.
Halvvejs nede i sekvensdiagrammet på figur 1 indtaster brugeren nu username og password, og parametrene sættes til checkLogin som kalder metoden checkLogin(). Denne metode tager imod de 2 parametre username og password, og sender query til databasen for at se om username og password matcher/eksisterer, og returnerer true hvis de matcher. Vi laver et check med en if/else statement inde i checkLogin.jsp og forwarder brugeren til shop.jsp, hvis checkLogin() returnerer true eller tilbage til index.jsp, hvis den returnerer false. Hvis alt er vel kan brugeren nu tilgå shop.jsp og begynde at handle.



Særlige forhold

Der er blevet sat meget fokus på hvordan vi skulle håndtere informationer, som brugeren/administratoren indtaster på siden for at logge ind, hvilket vi har grebet an via HttpSession.

Vi benytter os af sessions til at gemme og hente informationer om den pågældende bruger. Session er derfor oplagt at bruge til login systemer, da den både kan opbevare, men samtidige også tilgå de informationer, der opbevares til de forskellige sider. Vi anvendte session til at udskrive navnet på den pågældende bruger for at tydeliggøre, at systemet havde registreret login korrekt og samtidig forbedre brugervenligheden. Så det vi gemmer ned i vores session er faktisk en bruger, hvor vi så derfra kan tilgå brugerens informationer.



Som det illustreres på ovenstående billedet, ses der balancen og hvem der er logget ind. Disse data tilgår vi via vores gemte session.

Når der arbejdes med hjemmesider, spiller brugervenlighed en kæmpe rolle. Derfor implementerer man nogle fejlmeddelelser, der informerer os at de indtastet information er ukorrekte. Så en af de fremgangsmåder til at håndtere disse meddelelser, løste vi ved hjælp af required. Required er en attribut, som vi har tilføjet til vores formular, der giver en fejlmeddelelse hvis inputfeltet står tomt eller har en ukorrekt teksttype. Dette er dog ikke den bedste løsning i form af validering, men vælger vi at udvikle yderligere på projektet havde vi tænkt os at programmere en metode, der kan verificere at brugeren, som logger ind, ikke er en anden ved at sende en mail med en verifikationskode.

Som udgangspunkt skal man altid optimere sikkerheden for en hjemmeside, for at undgå hacking. En af de mere eftertragtede sikkerhedsoptimering er hash & salt i forbindelse med login, men eftersom vi ikke helt er blevet introduceret til hashing, har vi af gode grunde ikke kunne implementere dette.

Vores login tjek, sikrer at de rette informationer bliver tjekket via en metode. Metoden har til formål at tjekke ud fra vores formular om parametrene, som brugeren indtaster, er ens med de informationer, som allerede er gemt i databasen. Dette eksekveres via en metode, som indeholder en Query:*" SELECT Password FROM 'users' WHERE Username = username;"*.

Vores anden fremgangsmåde til at håndtere sikkerhed, var at fjerne muligheden for brugeren til at tilgå sider end index.jsp medmindre de allerede var logget ind. Vi havde tænkt os at lave et tjek på de enkelte sider, for om brugeren var logget ind. Ligeledes mangler vi at tilføje en egentlig logud metode, men det ville kunne løses vha. en metode der ser omkring sådan her ud:

```
private void logOut(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    if (request.getParameter("logud") != null) {
        session.invalidate();
        request.getRequestDispatcher("index.jsp").forward(request, response);
    }
}
```

Hvis parameteret log ud ikke er lige med null, så session.invalidate(dræb session) og send brugeren tilbage til index-(log ind)siden. Dette gør, at det ikke er muligt for brugeren at tilgå shop-siden igen uden at logge ind igen.

Det næste som vi havde lagt af planer, for at øge sikkerheden på vores kode der kommunikere med vores database er SQL-prepared statement.
Et SQL-prepared statement er en opstilling af de enkelte elementer, som kendes fra en simpel SQL-queries der sendes videre til serveren. Dette gøre det muligt for serveren at modtag en Query/forespørgsel på en handling der skal udføres og dernæst sender den de enkelte værdier til serveren. Når det sagt, sender den ikke længere en direkte forespørgsel til serveren, men kun den halve forespørgsel da værdierne først kommer efterfølgende.
Når det er sagt, er vores program meget sårbart, da det er muligt at angribe vores database via SQL-injection. Hvis der er mangel på håndtering af brugerinput og databasekald, kan dette resultere i, at data bliver manipuleret, opdateret eller ligefrem slettet.

Status på implementation

Vi har overordnet set en del mangler i vores system, men det er opgaver som vi allerede har taget hul på. Vi har bl.a. oprettet metoder til vores shoppingCart, invoices og orderlines og klargjort oprettelsen af de forskellige elementer i databasen. Fremadrettet vil vi arbejde med at forbedre

sikkerheden for brugerne, da vi stadig blot gemmer passwords i plaintext og tilføje en email verifikation når man bliver oprettet som bruger. Ydermere vil vi arbejde på at give admin overblik, ved at tilføje muligheden for at se alle ordrer, både gennemført af den enkelte bruger, men også siden systemet blev implementeret.
Vi mangler også at få balance til at opdatere når man tilføjer midler til sin konto og så har vi heller ikke implementeret en logud-knap endnu.
Vi er ikke kommet i mål med at få selve forretningslogikken til at fungere, hvilket selvfølgelig er den største prioritet. Det indebærer bl.a. at priserne bliver udregnet korrekt, at de individuelle cupcakes bliver tilføjet til kurven i korrekt antal og at salget kan gennemføres.
Generelt har vi nedprioriteret exceptionhandling, hvilket selvfølgelig også er en stor prioritet inden kunden får programmet udleveret. Helt overordnet set, så mangler der fejlbeskeder når kunden indtaster forkert data i inputfelter.
Disse mangler ses f.eks. på;

- index-siden hvor et forkert login ikke giver nogen fejl, men blot smider kunden tilbage til login-siden igen.
- createCustomer-siden hvor forkert input i brugernavn eller e-mail ikke giver en fejlsbeked.
- shop-siden hvor balance og quantity forsøges at opdateres med et tal giver en fejl, men ingen besked.

Test
Vi har ikke haft stort fokus på at lave egentlige unit-tests til vores program undervejs, men vi har naturligvis gennemtestet programmet ved at tilgå det som brugere imens vi har udviklet.