Project Cupcake rapport

- **Deltagere:**
  Navn: Rasmus Barfod Prætorius
  cph-mail: cph-rp134@cphbusiness.dk
  github-navn: Rasm-P
  Klasse: B

  Navn: Rasmus Hemmingsen
  cph-mail: cph-rh178@cphbusiness.dk
  github-navn: Razz
  Klasse: B

  Navn: Ditlev Andersen
  cph-mail: cph-di22@cphbusiness.dk
  github-navn:DitlevR
  Klasse: B

  Navn: Ludvig Bramsen
  cph-mail: cph-lb270@cphbusiness.dk
  github-navn: BenDaimio
  Klasse: B

- **Dato:**
  Projekt: 28-02-2019
  Rapport: 11-03-2019
  Projektaflevering: 17-03-2019

- **Links**
  Github project:
  https://github.com/razz7/cupcake

  Cupcake webshop:
  http://178.62.228.96/Cupcake/
  Admin user: Ole, 1234
  Customer user: Ditlev, 12345

Indledning

Dette projekt tager udgangspunkt i udviklingen af backend og frontend delene til en funktionel demo af en cupcake webshop. Udviklingen af denne simple webshop gør brug af en MySql database, java servlets og JSP sidder som backend delen, og HTML samt CSS og lidt javascript bootstrapping som frontend delen. Selve funktionaliteten af webshoppen fungere som en pick-up salgsside til cupcakes, hvor en bruger kunne bruge webshoppen til at lave en ordre af cupcakes, lægge dem i en indkøbskurv, betale og senere derefter selv hente de bestilte cupcakes. Hver cupcake skulle bestå af en top og en bund, med forskellige prædefinerede priser, som internt i systemet skulle kunne instantieres og tilknyttes en kundes indkøbskurv i form af en række forskellige java objekter, for den enkelte brugeres registrerede profil. Hver profil skulle derfor have noget brugerinformation, såvel som en balance der aktivt kunne ændres, når der lægges penge til eller trækkes fra kontoen.

Baggrund

Virksomheden der skal bruge systemet er en online cupcake shop, der bruger en pick-up funktion med deres nyeste cupcake maskine, hvor kunder kan forudbestille deres cupcakes gennem nogle registrerede profiler, og derefter komme og hente deres cupcakes i selve butikken.

Et krav til systemet indebærer hvordan deres kunder gennem denne online shop skal kunne oprette en bruger, eller logge ind på en eksisterende bruger, og derfra kunne specificere en række cupcakes der bliver lagt i en indkøbskurv. Prisen bliver heraf udregnet ud fra hvilken variation af cupcake bunde eller toppe der er valgt. Hver cupcake skal bestå af henholdsvis en top og en bund valgt af brugeren. Sortimentet kommer til at være bredt, og hver variation har en forskellig pris. Cupcakes vælges fra en menu og markeres med et ønsket antal. Brugeren kan efter at have valgt sin variation af cupcakes se en indkøbskurv med de valgte cupcakes, pris og mængde.

Teknologivalg

Nedenfor ses de programmer og versioner som er blevet brugt i udarbejdelsen af produktet:

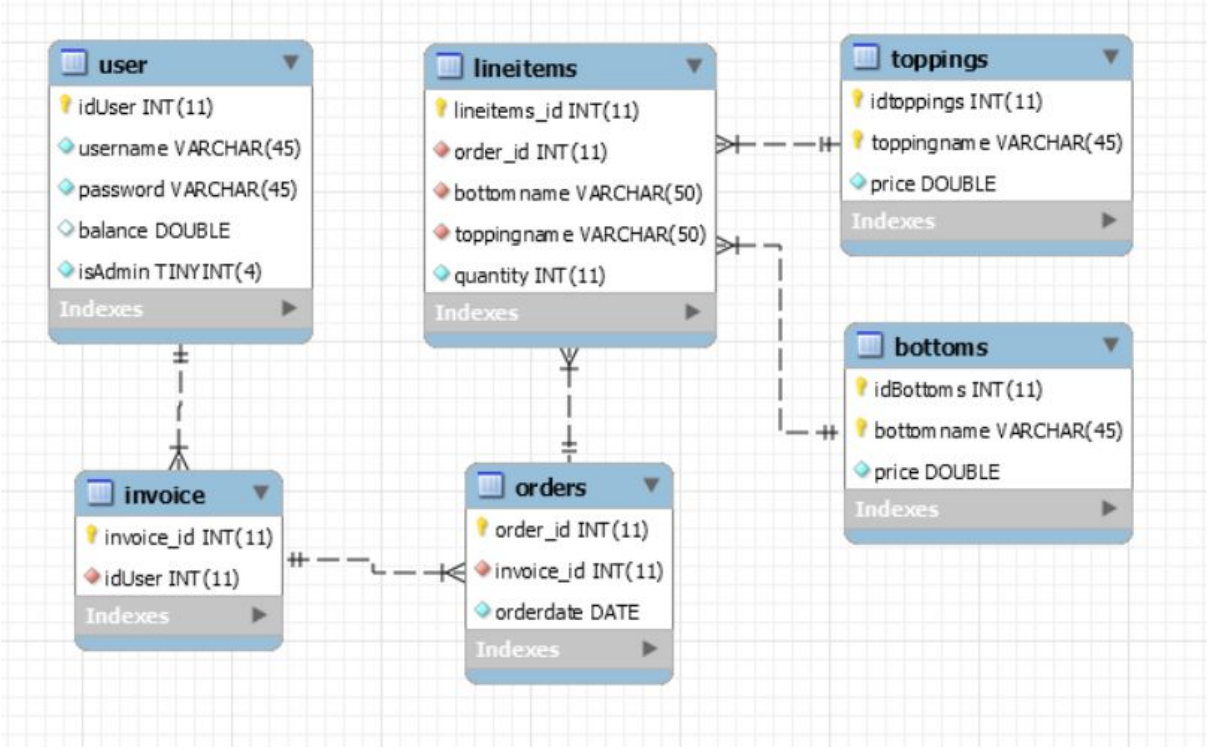
Netbeans 8.2 Java EE version, med Apache Tomcat 8.0.27 plugin.
MySql Workbench version-8.0.12 build -13312926 CE.
Digital Ocean Droplet med Ubuntu 18.10 x64 og installation af Java jdk-11.0.1,
-Mysql-server version-14.14 distribution-5.7.25, og apache-tomcat-9.0.16.
Git, Git-Bash med git-version 2.20.1.

Krav

De formelle krav og firmaets håb for dette system indebærer at systemet kan oprette og håndtere brugere, som bliver oprettet af kunder i forbindelse med bestillingen af cupcakes. Cupcakes skal laves fra deres nye maskine som en del af deres salg af pick-up cupcakes. Systemet skal kunne håndtere flere bestillinger fra enkelte kunder, hvoraf produkterne har forskellige specifikationer af bunde, toppe og mængde. Systemet skal kunne tilknytte brugeren en indkøbskurv hvor håndteringen af prisen sker, efter en ordrer bliver bestilt. Dette system skal være med til at optimere både forretningens og kundens involvering i bestillingen af cupcakes.

Domæne model og ER diagram

De følgende modeller og diagrammer beskriver grundlaget for relationer og den centrale data for det meste af systemet. Disse tabeller og relationer giver en ide om hvad systemet generelt håndterer og arbejder med:



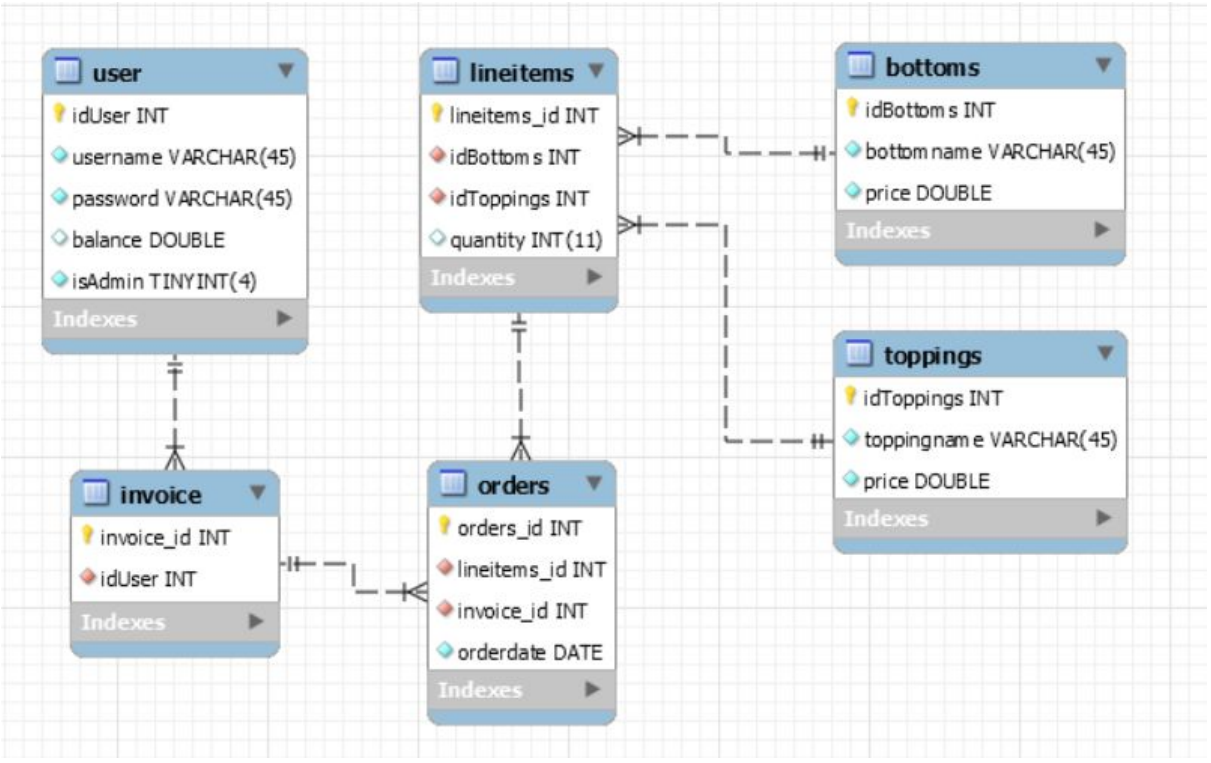
Det ovenstående ER-diagram beskriver databasen med dets tabeller og relationerne mellem disse tabeller, som de ser ud i databasen, samt de enkelte attributter, primær- og fremmednøgler.

Databasen opfylder både første, anden og tredje normalform, ved at have unikke rækker, samme type data i kolonner, ingen multivalues, ingen partielle afhængigheder, og ingen direkte transitive afhængigheder. Dog har vi lidt redundans i vores lineitems table for bottomname og toppingname, som vi valgte at beholde for at gøre vores kode en smule simplere at håndtere i det nuværende system. Dette ville dog være en mulig rettelse ved yderligere tidsrum.

Forholdene mellem de fleste af tabellerne i databasen er en til mange relationer, på nær forholdet mellem user og orders tabellen, hvor mange brugere kan have mange ordre. Dette forhold løses ved at bruge invoice tabellen til at beskrive unikke forhold mellem user og orders på user_id og invoice_id som foreign keys.

Når det kommer til databasens gennemgående konsistens, er der i lineitems table for bottomname og toppingname en mindre gentagelse af en varchar værdi der allerede ville kunne findes gennem henholdsvis en bottoms_idbottoms foreign key relation til idBottoms primary key i bottoms tabellen, og toppins_idtoppings foreign keys relation til idToppings primary key i toppings tabellen.

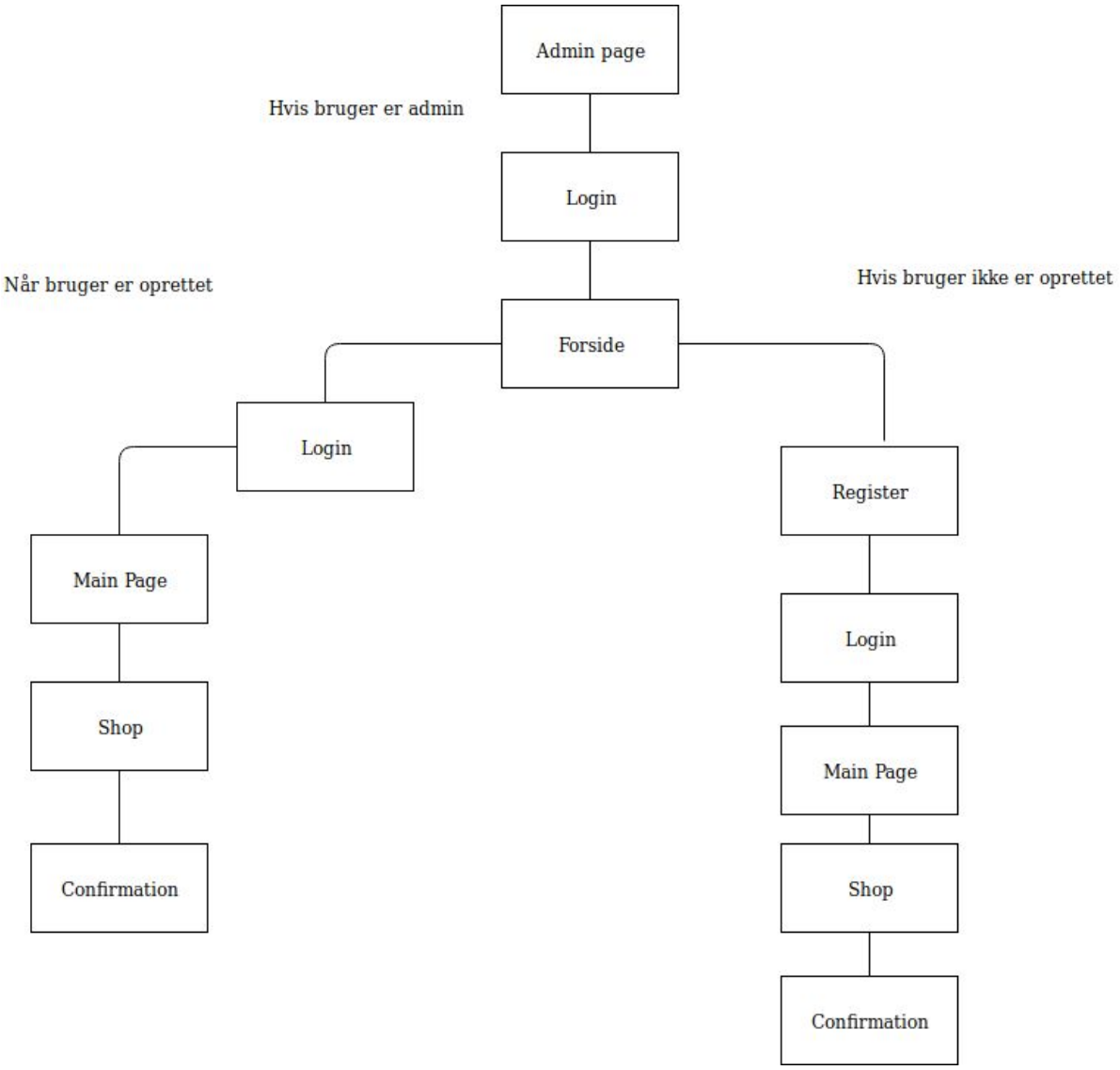
Med henblik på hvad vi godt kunne tænke os at lave om eller delvist ændre på databasen vedrørende dets nuværende stadie, er der en del elementer på listen som vi med udgangspunkt i funktionalitet og den tredje normalform, godt kunne tænke os at implementere. Blandt andet kunne vi godt tænke os at databasen var navngivet noget bedre, og at der var autoinkrimenterende id'er for absolut alle tabellerne, samt en ordentlig fordeling og håndtering af bottoms og toppings elementerne, gennem deres unikke id, for den topping eller bottom som kunden nu måtte vælge. Vi kunne også tænke os at give databasen noget mere struktur så alle elementerne, specielt med fokus på orders og invoice, kunne komme til deres intentionelle relationelle brug som mange til mange rellationer mellem kunder og ordre. Det kan derfor med sikkerhed siges at der i forhold til den nuværende database, er en del ting som vi på nuværende tidspunkt, godt kunne have tænkt os at implementere anderledes. Nedenfor ses et ER-diagram, der nærmere illustrerer den sammensætning som vi i enden godt kunne have tænkt os at databasen tog form efter.



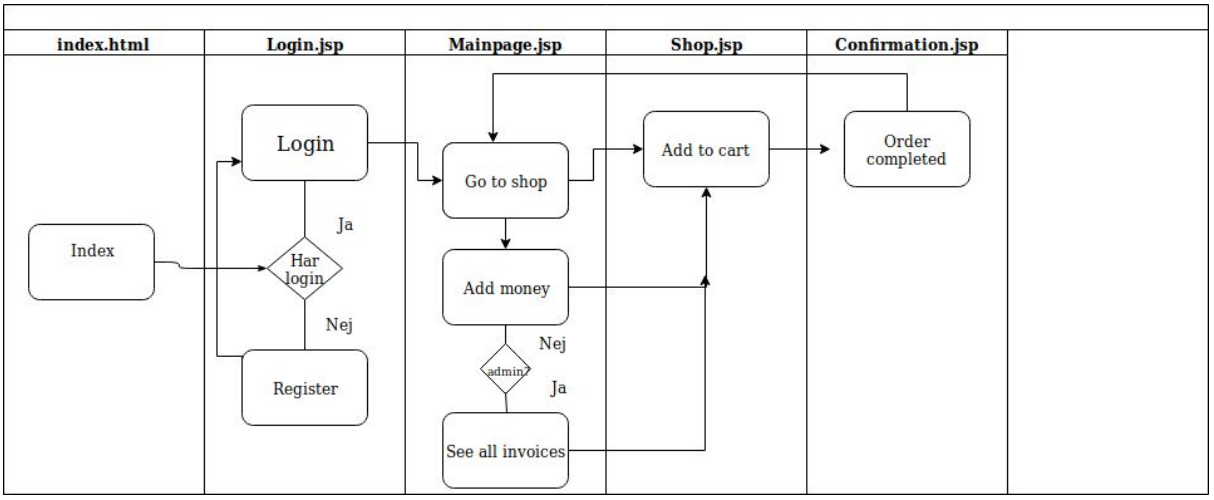
Navigationsdiagram

Navigationsdiagrammerne nedenfor viser i et mere overskueligt format, hvordan en bruger kan navigere fra og til de forskellige sider i systemet, samt hvilke brugere, der kan få adgang til hvilke sider. Da grundlaget for navigationsdiagrammerne hovedsageligt tager udgangspunkt i bedst muligt at beskrive navigationen mellem vores jsp sider, har vi valgt følgende illustrationer, hvorpå vi kan tolke vores system i diagramformat.

Nedenfor ses en forsimplet version af navigationerne. Denne lidt forsimplede version er måske nemmere at forstå for nogen, og giver et mere fuldstændigt indtryk, hvilket grunden til at vi har valgt at inkludere den. Her har vi vores forside som startpunkt, og alt efter om brugeren har et login eller ej, og om du er admin, kan de komme rundt til forskellige sider.



Det følgende diagram ses et mere konkret navigationsdiagram i såkaldte swimlanes, der hjælper med at forklare hvordan genvejene mellem de enkelte jsp sider interagere med hinanden samt under hvilke forhold gennem systemet.

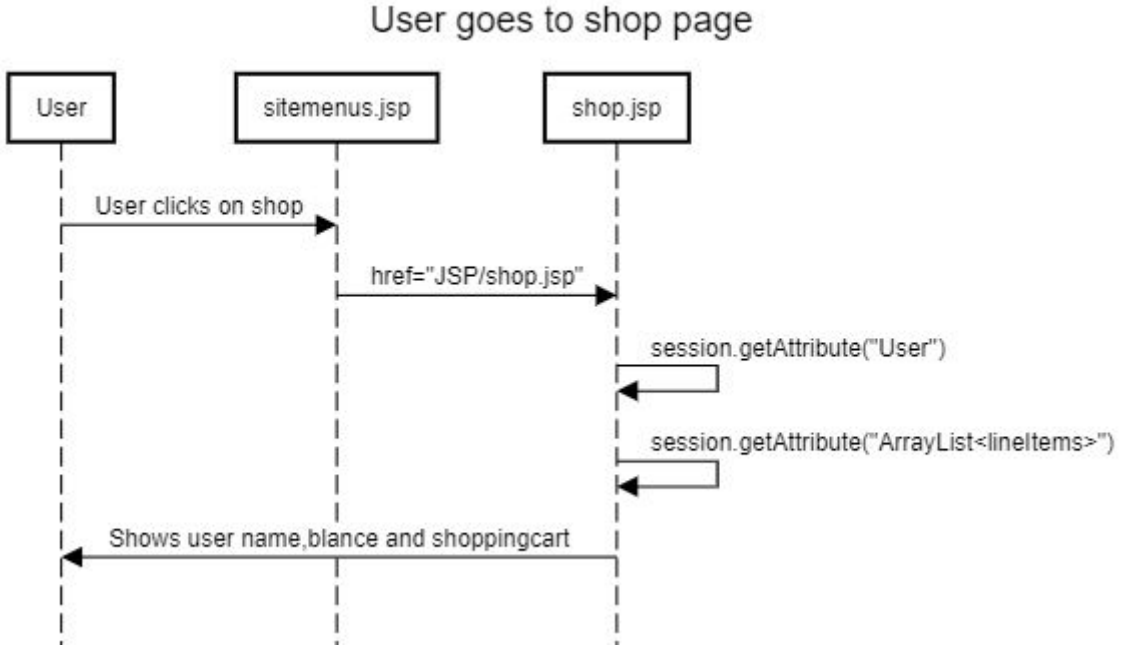


Vores navigationsdiagram er bygget op om de 3 forskellige cases af en bruger, der kan kommer ind på vores side. En bruger der allerede har et login, kan logge ind og blive sendt til loggedIn versionen af vores main page. På main page kan brugere tilføje penge til deres konto, gå til shoppen eller logge ud. Hvis de går til shoppen kan de tilføje cupcakes til deres bestilling. Herefter kan de købe deres ordrer, hvis de har penge nok så kommer de til confirmation siden. Hvis de ikke har nok penge på deres konto kommer de til error siden. Hvis en bruger er admin kan de efter at de er logget ind komme til admin siden og se alle ordrer. Admin kan også kommer til shop tilføje penge ligesome de ordinære kunder.

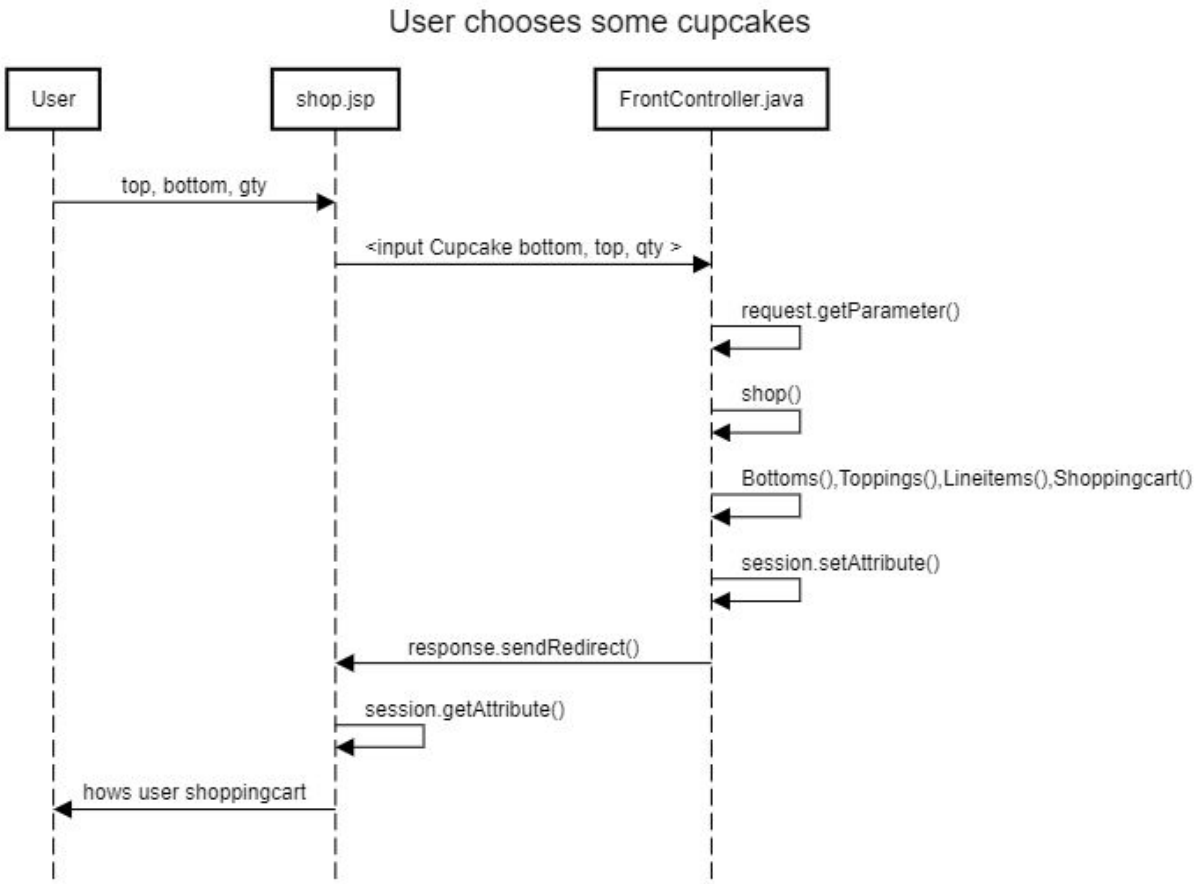
Hvis brugerne ikke har et login, kan de gå til register siden og registere sig selv med et username og password, herefter kan de komme tilbage til login siden og logge ind. Når de er logget ind, kan de gå til shop, tilføje penge og komme til confirmation siden. Alle vores sider bruger en gennemgående header som navigationsværktøj, hvorpå genvejene kan findes. I toppen af navigations diagrammet kan vi se navne på siderne, som vi kan komme ind til (på nær confirmation da den kommer efter et køb). Headeren og dens genveje ser anderledes ud hvis man er admin. I tilfældet for en admin vil der komme en knap i headeren, hvorpå der står admin page. På denne side kan man se alle invoices for alle brugere, samt brugerdata om balance og brugerens køb. Det er vores front controller der styrer slagets gang i forhold til at navigere rundt mellem siderne, hvilket foregår ved et switch case system ud fra http request parametre.

Sekvensdiagrammer

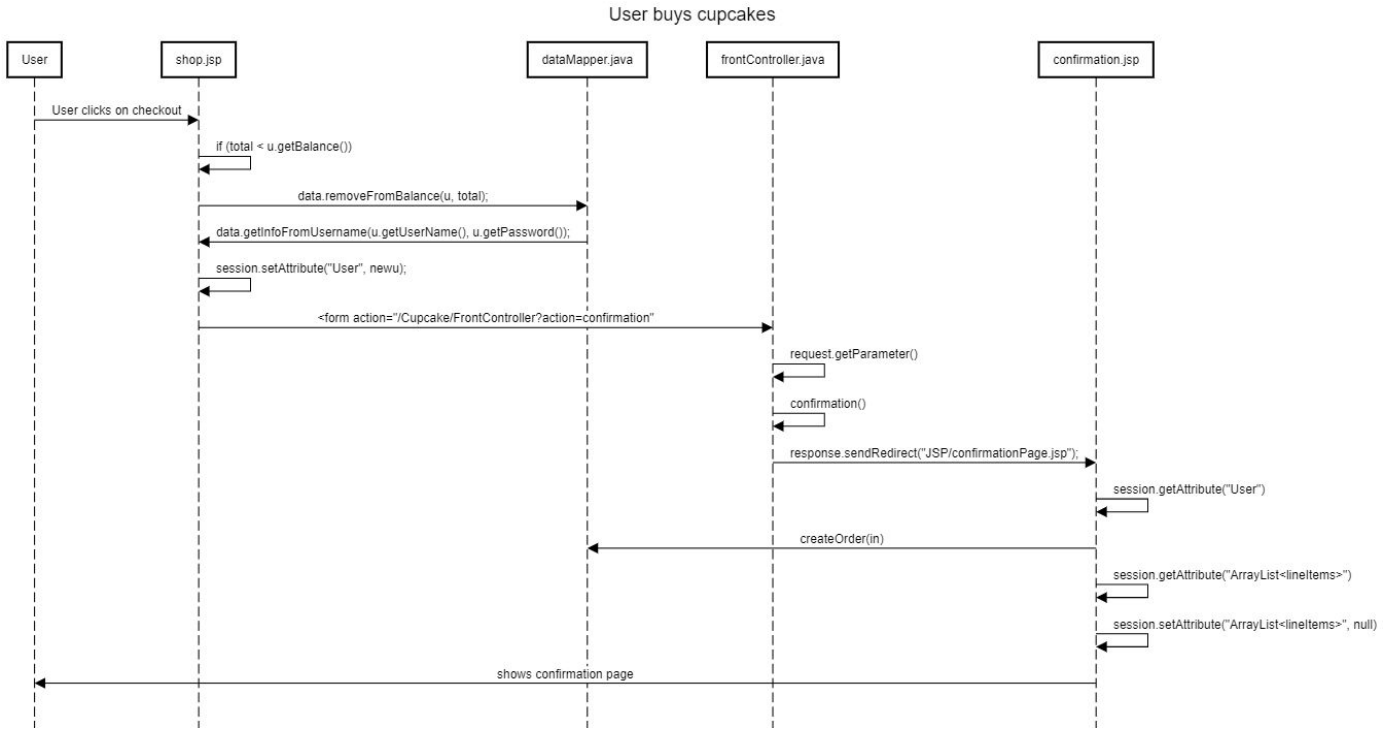
Sekvensdiagrammerne nedenfor viser hvordan et typisk forløb i programmet foregår mellem jsp siderne, servietten, og de enkelte metoder der bliver kaldt på. De følgende diagrammer tager udgangspunkt i hvordan en bruger henholdsvis kommer ind på shop siden, vælger bottoms og toppings, og til sidst så køber de valgte cupcakes gennem shop siden.



I dette sekvensdiagram klikker brugerne først på sitemenus knap til shoppen, hvorefter en herf reference dirigerer brugernes browser hen på shop.jsp siden. shop.jsp kalder derefter på session.getAttribute() for henholdsvis sessionens user og den pågældende users tilhørende arraylist af lineitems. Begge disse methode kaldes af session.getAttribute() i hvert sit if statement, der tjekker om de pågældende attributter ikke er null. Hvis attributterne er null og dermed ikke findes, bliver de ikke vist til brugeren. Brugeren bliver derefter præsenteret for shop.jsp siden, med de tilhørende layoutindstillinger for henholdsvist brgueringofrmation og den nuværedne shoppingcart tilhørende sessionens bruger.



Brugeren starter med at vælge top, bund og antallet af cupcakes i en eller flere ordre. shop.jsp sender inputtet fra brugeren videre til frontcontrolleren. Metoden request.getParameter() i frontcontrolleren tager parametrene fra shop.jsp. Så kaldes metoden shop() der med sine parametre og metoderne getOneBottom(), getOneToppings(), laver bottom og top objekter, der bruges til at lave et cupcake objekt. Dette cupcake objekt sættes ind i et lineItems() objekt, som så gives videre til sessionen med setAttribute(). Derefter sender metoden response.sendRedirect() brugeren tilbage til shop.jsp siden, så brugeren kan fortsætte købet.



I dette sekvensdiagram tager vi udgangspunkt i, at brugeren har valgt en række cupcakes til sin shoppingcart, og nu er klar til at købe dem. Bruger starter med at trykke på checkout knappen på shop.jsp siden. Derefter checker shopping siden om den nuværende sessions brugers balance, er over den total som købet udgør gennem et if statement. Hvis brugerens balance er mindre end købet, ændre checkout knappens parametre sig til action=error gennem et if statement, og brugeren føres hen til error.jsp. Hvis dette ikke er tilfældet og brugeren har nok penge, blive removeFromBalance() fra dataMapperen kaldt, hvorved der trækkes den efterspurgte pris fra brugerens balance i databasen. Herefter bliver getInfoFromUsername() fra dataMapperen kaldt for at opdatere brugen i den nuværende session med session.setAtttribute(), så balancen også bliver opdateret i browseren. Derefter dirigeres der til frontControlleren med parameteren action=confirmation, hvilket i front bliver lavet til en string med request.getParameter(), og så kørt gennem et switch statement, med action parametrene som cases. I denne action case for confirmation kaldes metoden confirmation(), der gennem et if statement checker at shoppingcarten ikke er tom, hvorefter response.sendRedirect() enten sender brugeren tilbage til shoppen, hvis den er tom, eller sender brugeren til en confirmation.jsp side, hvis shoppingcarten ikke var tom og købet gik igennem. confirmation.jsp siden kalder session.getAttribute() på user, for at vise brugerinformationer som balance og brugernavn, hvorefter createOrder() fra dataMapperen bliver kaldt for at tilføje indkøbslisten til databasen. Herefter loopes shoppingcarten fra session.getAttributes() igennem for at vise brugeren alle elementerne han lige har købet. Derefter kaldes session.setAttributes("ArrayList<lineItems>", null) for shoppingcarten til igen at være null og dermed tom igen. Herefter vises confirmation.jsp til brugeren med det købte indhold, for at vise brugeren at købet gik igennem.

Særlige forhold

Når det kommer til de informationer som vi gemmer i den enkelte forbindelsens session, har vi henholdsvist gemt informationer om brugeren, brugerens login-status samt brugerens nuværende indkøbslister og invoices gennem session.setAttribute(). Grunden til at vi har knyttet disse informationer til brugerens session, er at vi nemmere kan håndtere og behandle data i forbindelse med den nuværende bruger. Når det kommer til selve brugeren, gemmer vi et user object i den nuværende session, der indeholder information om både brugerens navn, password og balance, så vi altid har adgang til dem, for at kunne vise relevant brugerindhold på siderne. Brugerens login status bliver også gemt i sessionen som true indtil brugeren vælger at logge ud, hvilket altid validere om brugeren er logget ind når de prøver at tilgå systemets funktioner. Når brugeren så beslutter sig for at lave et køb af cupcakes og lægge sine valg i indkøbskurven, bliver objekterne i denne indkøbskurv også gemt i sessionen, så brugeren altid kan gå ud og vende tilbage til sin indkøbskurv og sit køb. Indkøbskurv og invoices bliver gemt i sessionen indtil brugeren vælger at forlade sessionen ved at logge ud eller færdiggøre købet.

Vi håndterer exceptions i de forskellige metoder, der har en try catch blok. De bliver håndteret primært i DataMapper klassen og i CupcakeMapper. Disse metoder er dem der skriver til og finder data fra databasen, hvis metoderne fejler kommer der en printet fejl og en custom fejl bliver printede til java konsollen. Hvis vi havde mere tid ville vi nok lave en log funktion, som kunne logge fejl så man kunne følge med i hvor programmet fejler. Vi bruger e.printskacktrace til at få den konkrete java fejl. Ved vores custom fejl står der noget om hvilken metoder der fejler, f.eks. “Error in getting information for toppings”. Disse fejlbeskeder kunne i fremtiden også blive printet til brugerens jsp sider, så brugeren får en ide om hvad der gik galt.

Validering af brugerinput behandles i systemet, ved at vi først og fremmest har en del if statements der checker for om objekter != null og dermed eksisterer, samt om enkelte parametre ikke er tomme og dermed kan indgå i vores metoder og antages om valid brugerinput. Ud over dette har vi lavet en transaction class som vi bruger til at wrappe vores invoices i, inden vi inserter dem i vores database, da vi derved checker om alle parameter og involverede objekter eksisterer og er etableret korrekt. Dermed indsættes enten den fulde invoice, eller ingen invoice.

Ved yderligere udvikling af webapplikationen ville vi implementerer en mere sikker måde at kryptere brugernes adgangsinformationer på. Det kunne for eksempel laves ved at hashe brugerens password, og derved gøre det væsentligt mere kompliceret at få adgang for uvedkommende. På nuværende tidspunkt bliver brugerens selvvalgte password gemt i databasen som en String, uden at være krypteret. Dette ville nok være en væsentlig ting at implementere ved videreudvikling.

Når det kommer til brugen af forskellige brugertyper, skelner systemet mellem to brugergrupper, henholdsvis kunderne, og admin brugerne som en administrativ rolle. Disse bruger har selvfølgelig adgang til forskellige funktioner i cupcake shoppen. Kundernes brugere har adgang til selve basisprogrammet, hvor de kan lægge cupcakes i deres indkøbskurv, og derefter købe dem, samt navigere mellem shop, confirmation, invoice, loggedIn og mainPage siderne. Når det kommer til admin brugerne har de adgang til en række yderliger funktioner, hvilket involverer selve adminPage og seeAllInvoices siden som ikke kan findes på kundernes headermenu. adminPage har indtil videre adgang til at se all invoices for alle brugere, og kan dermed overvåge hver enkelte transaktion som kunderne laver. Ligesom at kunderne kan klikke ind på deres egen invoice for at se yderligere detaljer, kan admin brugerne også klikke ind på alle de forskellige bruges invoices, og derved se enhver detalje vedrørende både købet og selve brugeren der gjorde købet.

Et andet særligt forhold som vi har gjort brug af i systemet, er det at vi kun har en servlet der fungere som frontController. Denne frontController kan køre i flere instanser på den samme session, og fungere som en slags hovedcentral for alle http requests og http responses, hvorfra håndteringen af både de viste sider og deres backend indhold håndteres. Denne frontController, gør brug af en switch case der reagere på hvilke parametre der indgår i en http request, når der navigeres til frontController som path. Hver case i dette switch statement, indeholder hver sin metode hvori, der kaldes på en masse relevante underliggende metoder og til sidst et response.sendRedirect(), som kan dirigere brugeren videre mellem systemets jsp sider.

Status på implementation

Når det kommer til statussen på implementationen af vores system, er der en del ting som vi stadig godt kunne have tænkt os at implementer, ændre eller overvejende genopbygge. Nogle af disse elementer involvere yderliger ønskede funktioner, andre har at gøre med fejl der gør systemet mere ustabilt end det burde være. Størstedelen kommer af at vi gennem forløbet i at arbejde med systemet, er blevet en del klogere på hvordan en sådan webservice etableres og struktureres.

En af de umiddelbare større punkter er stabiliteten af programmet, der har en række huller og bugs, som vi i enden af dette forløb ikke har tid til at fejlfinde. Da dette er en demo er det dog ikke så ødelæggende, så længe at systemet i grundlaget køre, men det er dog noget som vi gerne fortsat ville have arbejdet på.

Selv om vi har nået at lave størstedelen af jsp siderne, kunne vi også her godt havde tænkt os at lave nogle flere individuelle sider, samt tilføje yderligere elementer til både kunde

brugeren og admin brugerne. En af de elementer som vi mangler er blandt andet at have jsp sider, hvor henholdsvis kunder og admin kan tilgå, og måske endda i varieret grad redigere i bruger og ordreinformationer. Der er også brug for noget mere struktur og flere elementer i fordelingen af invoice informationer i invoice.jsp, i stedet for bare at vise en lang toString.

Et mere grundlæggende element som på mange måder ligger lidt i vejen for systemets status, er databasen som vi på mange måder godt kunne tænke os at udbygge og rette op på yderligere. Da databasen er en fundamental del for meget af systemet, og indgår i stort set alle metoderne for både cupcakeMapper og dataMapper, er de funktionelle og strukturelle afvigelser forholdsvis kritiske for vores system, og meget svært at rette på nuværende tidspunkt. Den nuværende version af vores database var opbygget efter behov, og virker umiddelbart også efter de behov, men strukturen for yderliger udvidelser og forholdet til overholdelsen af normalformerne, gør den særdeles svær at arbejde med på nuværende tidspunkt, og bestemt på længere sigt.

Andre elementer som har effekt på systemets nuværende status, involverer de metoder vi bruger, hvor vi kalder dem fra og de enkelte sql queries, der også godt kunne bruge lidt opmærksomhed. Mange af disse elementer har ikke direkte effekt for systemets brugere, men kunne gøre systemet en del nemmere at arbejde med og senere hen udvide.

Med dette sagt har vi nået langt de fleste af de krævede elementer for systemet, og har lavet en fungerende platform for en online cupcake shop der opfylder kundens essentielle krav til systemet, samt den opgavevejledning som vi fik udleveret. Fejlfinding og stabilisering af systemets mange elementer er dog stadig igangværende på afleveringstidspunktet.

Vores admin side loader ret langsomt, og vi får en sjælden gang imellem en NullPointerException, som vi desværre ikke har haft tid til at løse. Dog kunne det tænkes at grunden til vi har den lange loade tid, kunne have noget at gøre med den måde som vi henter alle invoices på.

Test

Vi ville have testet de metoder, som har mest betydning for at programmet virker. De funktioner som har med databasen at gøre, er ret afgørende for om siden fungere optimalt, og at tingene bliver lagret på den rigtige måde.

Det primære formål ved at lave disse test, er at se hvad metoderne gør i tilfælde af at en forkert parameter indtræder, og at kontrollere at der ikke er nogle systemfejl. Her vil det være væsentligt at finde ud af om en metode lagrer nogle forkerte parameter, eller den er bygget til at fejhandle.

Nedenstående er de test som vi ville have lavet af datamappen- og cupcakemapper klassen.

CreateOrder	Her ville vi lave en ordrer med en bruger som vi laver. Da det er en af de mest essentielle metoder er det vigtigt at både teste med nogle rigtige parameter men også teste hvor man går ud fra den skal fejle.
getInvoiceForCustomer	Her vil vi kalde på den ordrer som blev lavet i testen af CreateOrder. Her ville man kunne se om testen overhovedet returnerer noget eller den ikke kan finde ordren.
AddToBalance	Da det også er en temmelig essentiel metode kunne man kalde denne metode på kendt user og derefter kalde getBalanceFromDB metoden for at tjekke om den har virket.
RemoveFromBalance	Samme som den øvrige men hvor man fjerne penge fra en bruger.

Hvis vi havde tid til at implementere det ordentligt, ville vi have testet alle vores metoder, der skulle skriv eller hente data fra databasen. For dem som skulle hente specifik data, ville vi køre test på en bruger som vi ved fungere og få metoden til at hente noget fra databasen for at se om vi får noget korrekt igen.