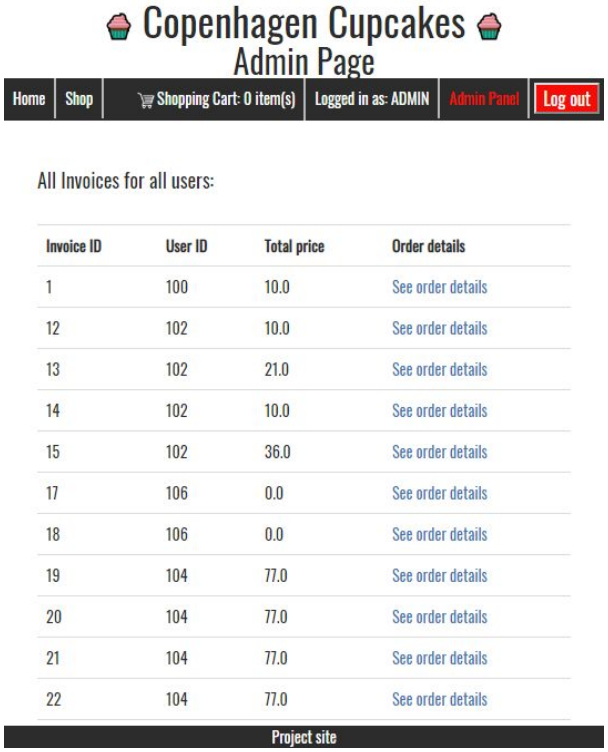
CupCake Webprojekt



Deltagere

Navn	**Skolemail**	**Githubnavn**
Rúni Vedel Niclasen	cph-rn118@cphbusiness.dk	Runi-VN
Camilla Jenny Valerius Staunstrup	cph-cs340@cphbusiness.dk	Castau
Asger Koch Bjarup	cph-ab363@cphbusiness.dk	HrBjarup
Steingrimur Kolbeinn Jonsson	cph-sj406@cphbusiness.dk	Steingrimurjonsson

Projektet blev påbegyndt d. 27/02-2019 og afleveret d. 17/03-2019.

Datamatiker 2. Semester, klasse: A
Link til projektet på github
Link til javaDocs (github pages)
Link til projektets ip (online på DigitalOcean Droplet)

Indledning

En virksomhed har brug for en webshop til at håndtere deres kunders ordrer. Al information er lagret i en database og behandles i backenden før det vises eller ændres i frontenden.
Frontend: CSS, HTML, jsp.
Backend: MySQL, Java, Java servlets.

Baggrund

Virksomheden, som skal bruge systemet
Virksomheden sælger cupcakes, men kun som afhentning. De vil gerne have bygget en webshop hvor deres kunder kan bestille og betale for Cupcakes til straks-afhentning. Cupcakes skal altid defineres i toppe og bunde. Virksomheden vil gerne have adgang til et administratorpanel som viser alle ordrer i systemet.

Specifikke krav

- Det skal være muligt at oprette en bruger på applikationen og kunne logge ind på denne
- Der skal være en webshop som viser de forskellige toppe og bunde som kunden kan vælge mellem
- Kunden skal efter valg af top og bund kunne lægge en Cupcake i en Shopping cart
- Når Handlen gennemføres skal der laves en Invoice og ordrens pris skal trækkes fra kundens Balance
- Der skal være en Customer Page som viser alle kundens tidligere ordrer. Det skal være muligt at trykke på en specifik ordre og se detaljerne for denne ordre
- Der skal være en Admin Page, som indeholder en liste over alle kunders ordrer. Det skal være muligt at trykke på en specifik ordre og se detaljerne for denne ordre

Kundens muligheder i systemet
En kunde har mulighed for at oprette en bruger i systemet og at logge ind med denne. Kundens brugerkonto har mulighed for at placere ordrer til virksomheden, givet at denne har tilstrækkelig valuta. Hvis dette ikke gælder, har kunden mulighed for at tilføje valuta til sin konto på Customer Page. Ved korrekt afgivet ordre modregnes beløbet fra Customer Page saldo. Kunden har via sin Customer Page mulighed for at se sine afgivne ordrer samt sin brugerdata.

Teknologi valg

Udviklingsværktøjer

- Netbeans IDE (8.2)

Teknologier

- Linux Ubuntu 18.10 x64
- MySql Version 8.0.15 for Linux on x86_64 (MySQL Community Server - GPL)
- Java: 1.8.0_181
- JavaEE Web API (7.0)
- Apache Tomcat 8.0.27.0
- Maven 4.0.0

Dependencies

- Mysql JDBC (8.0.12)
- Maven Javadoc Plugin 2.9

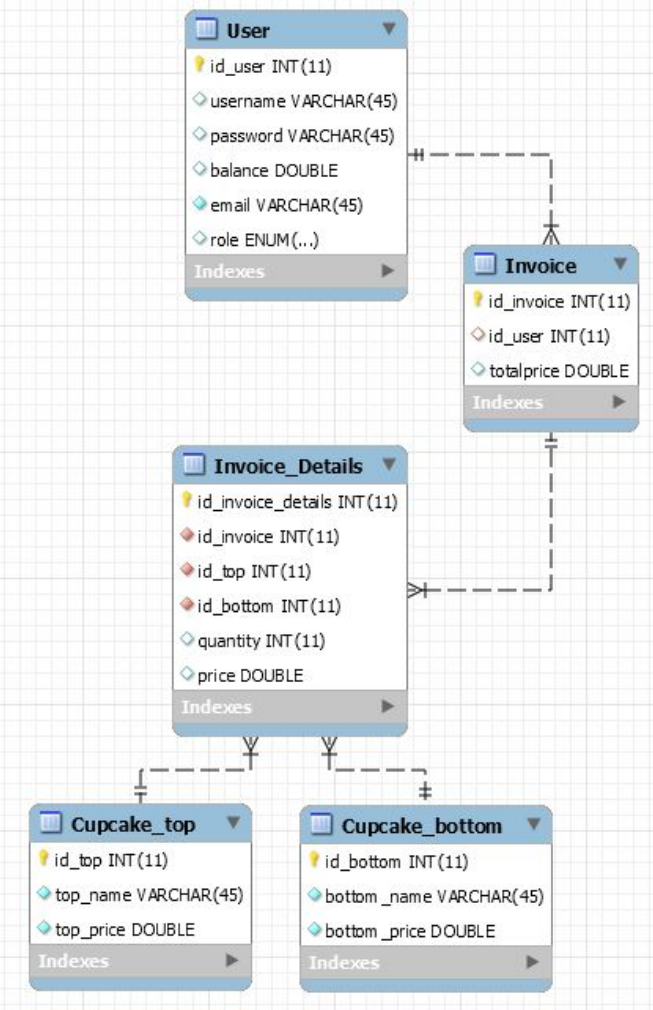
Frameworks

- Bootstrap 3.4.0

Modeller og diagrammer

Vi byggede databasen på denne måde, især med hensyn på at opnå 3. Normalform. Herudover har vi forsøgt at strukturere vores data så vi nemt kunne lave objekter der repræsenterede dataen i koden.

ER Diagram over Cupcake-databasen



Beskrivelse af ER Diagram

User:

- Kan have mange Invoice(s)
- id_user (Primary Key) anvender AUTO_INCREMENT
- username & email er unique for at undgå dobbelt-registreringer.
- Balance har default 0.00
- role har enum-typerne *admin* & *user*, med default user.

En bruger skal jf. kunden oprettes med username, password & email - resten sættes pr. *default* af databasen.

Invoice: (“Regning” / “Faktura”)

- Kan have en User
- Kan have mange Invoice_Details
- id_invoice (Primary Key) anvender AUTO_INCREMENT
- id_user (Foreign Key -> User.id_user)
    - For at binde en specifik invoice til dennes user.

Invoice_Detials: (“Ordrelinjer”)

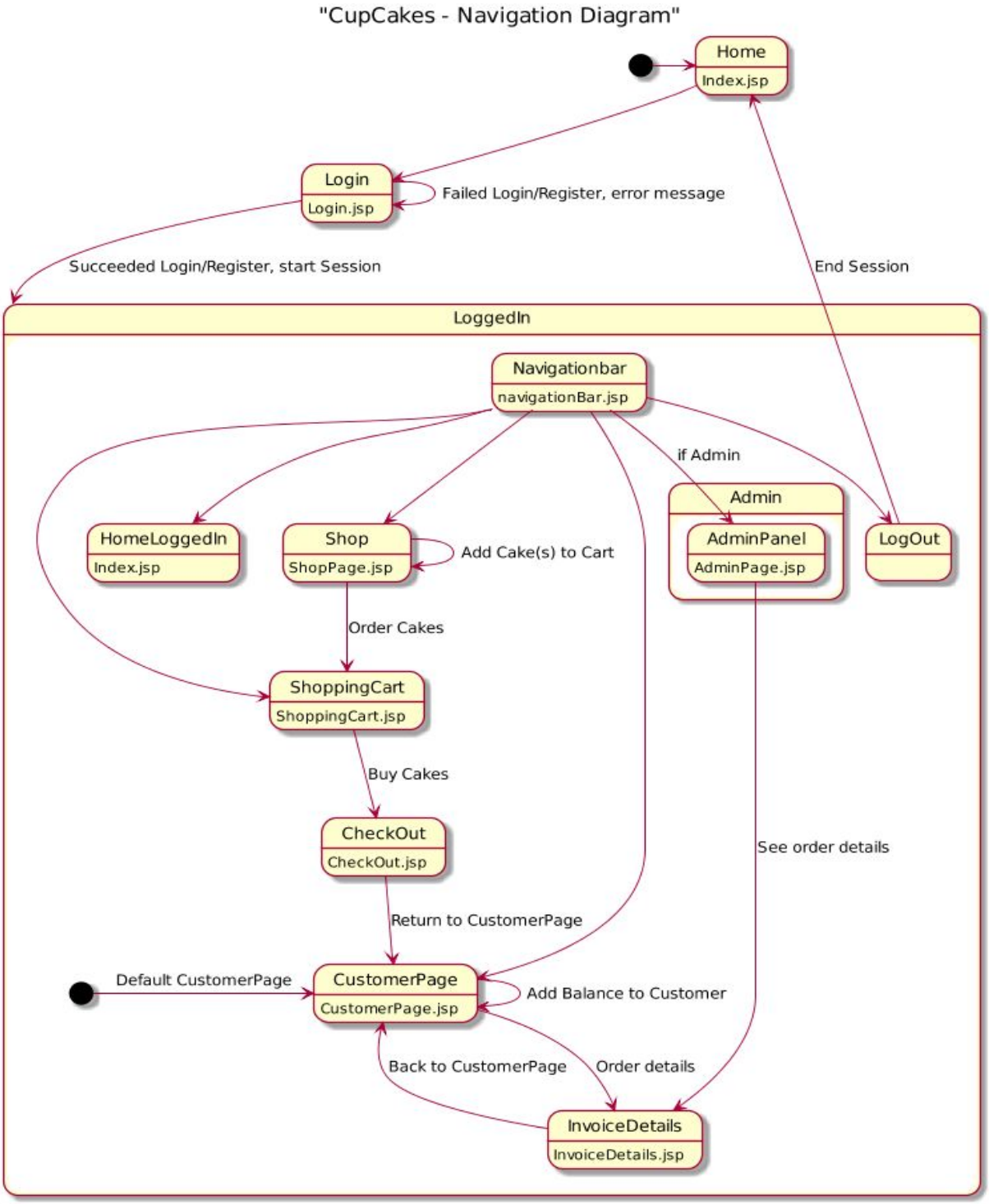
- Kan have en slags cupcake_top og en cupcake_bottom (flere mulige hvis der tages brug af *Quantity)*
- Id_invoice_details (Primary Key) anvender AUTO_INCREMENT
- Id_top, id_bottom & Id_invoice er alle foreign keys
    - For at binde en specifik cupcake_top og cupcake_bottom til denne invoice_detail, og samtidig binde hele denne Invoice_Detail til den specifikke Invoice
    -

Cupcake_top & Cupcape_bottom

- Kan have flere Invoice_Details
- id_top og id_bottom (Cupcake_top/Cupcake_bottom) har **ikke** AUTO_INCREMENT da der ikke er planlagt nye cupcakes.

Navigationsdiagram

Navigationsdiagrammet viser de muligheder en bruger har i vores applikation, fra start til slut. Den sorte prik indikere start når ip'en bliver tilgået. Den sorte prik vises igen inden i state "LoggedIn" og peger på den default side brugeren starter på når session er påbegyndt.



Beskrivelse af Navigationsdiagram

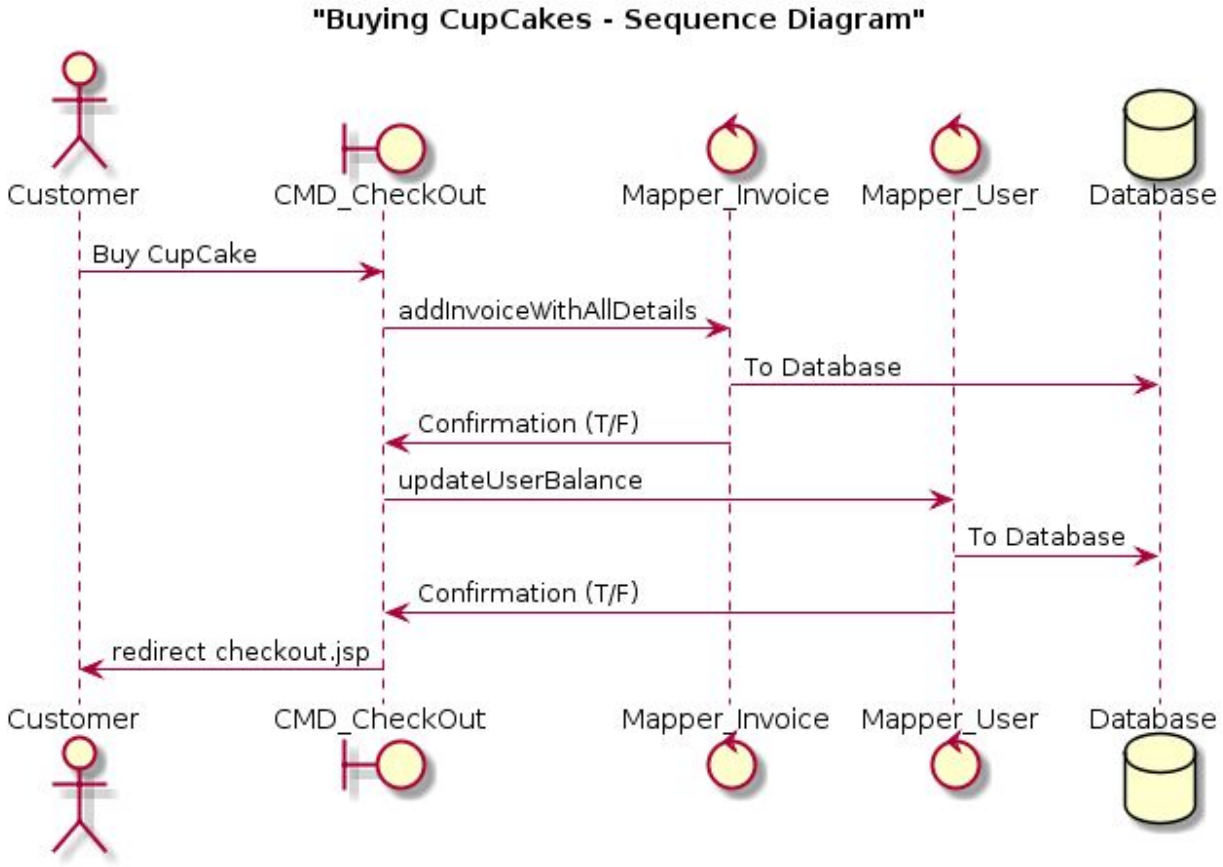
http://207.154.233.238/ leder til index.jsp som viser indholdet af applikationens **Home**. Når man ikke er logget ind kan man tilgå:

- **Login/Registrer** hvorfra der kan logges ind eller oprettes en bruger
    - ved fejlindtastning sendes man tilbage til loginsiden, samtidig med at der vises en fejlbesked
    - ved login bliver der tjekket på rettigheder (admin/user)
    - ved login bliver session sat

Efter login kan de andre sider i **Navigationsbaren** tilgås:

- **Home** fører til index.jsp som også vises som startside
- **Shop** fører til ShopPage.jsp hvorfra:
    - der kan vælges Cupcake top og bund kombination
    - der kan vælges antal af kombinationen
    - Add cake(s) to cart tilføjer den valgte kombination til Shopping Cart
    - Empty the cart tømmer Shopping Cart for items
    - Remove fjerner et enkelt item fra Shopping Cart
    - Order cake(s) fører til Shopping Cart (kan også tilgås fra navigationsbaren)
- **Shopping Cart** viser en oversigt over alle de Cupcakes man er ved at købe
    - Buy cupcake(s) gennemfører købet og fører til Checkout
    - fra Checkout kan der returneres til Customer Page (kan også tilgås fra navigationsbaren).
- **Customer Page** (vises som “Logged in as: NAME” i navigationsbaren) viser brugerinformation
    - her kan der “købes” valuta (Balance) til køb af Cupcakes
    - tidligere køb fremgår af afsnittet Invoices
    - Order details fører til Invoice DetailsPage som indeholder informationer om det specifikke køb. Back to Customer Page retunerer til Customer Page (kan også tilgås fra navigationsbaren)
- **Admin Panel** (kun synlig hvis admin-rettigheder er tildelt) viser alle invoices for alle brugere
    - See order details fører til Invoice DetailsPage som indeholder informationer om det specifikke køb. Back to Customer Page retunerer til Customer Page (kan også tilgås fra navigationsbaren)
- **Log out** slutter Session og fører tilbage til Home

Sekvensdiagram
Sekvensdiagrammet viser en oversigt over metodekald i applikationen når en kunde køber indholdet af shoppingcarten.



Beskrivelse af Sekvensdiagram
Diagrammet starter i et scenarie hvor brugeren har valgt en Cupcake top og en Cupcake bund i Shop, lagt dem i Shopping Cart og har trykket Order cake(s) og nu er inde på Shopping Cart siden og er klar til at trykke på Buy CupCake(s).

Frontcontrollerens ansvar
Når der trykkes på Buy CupCake(s) sendes en request til `FrontController()`, som opretter et `Command()` objekt, der kigges på pathinfo som i switchcasen bruges til at bestemme hvilken `CMD_` der skal oprettes. I dette tilfælde er de `CMD_Checkout()`. Herefter kaldes `execute()` i den valgte `CMD_`.

Commandens ansvar
`CMD_Checkout()` opretter en `Mapper_User()` og en `Mapper_Invoice()` for at kunne tilgå de relevante metoder til databasekald. Der oprettes flere relevante objekter (`Model_User()` og

`Cart()`) som alle trækkes ud af `Session()` objektet. Ligeledes trækkes en boolean `buyPermission` ud af `Session()`. Hvis `buyPermission` er `true` og Arraylisten i `Cart()` objektet ikke er tom udføres if-sætningen.
If-sætningen opretter `Model_Invoice()` og `Model_InvoiceDetails()` objekterne og sætter variablerne i disse objekter ud fra informationerne der tidligere blev sat `Model_User()` og `Cart()`. For-loopet sorterer ArrayListen `cakes`, dette gøres for at holde styr på hvor mange ens `Model_CupCake()` listen indeholder. Dette bruges til at sætte variablen `Quantity` i ArrayListen med `Model_InvoiceDetails()`. Herefter kaldes `addInvoiceWithAllDetails()` i `Mapper_Invoice()` og `updateUserBalance()` i `Mapper_User()`.

Mapperenes ansvar
I `Mapper_Invoice()` sørger `addInvoiceWithAllDetails()` for at skrive dataen der blev genereret i `CMD_Checkout()` til databasen. Dette sker ad to omgange med `PreparedStatement()`, en for tabellen `cupcake.Invoice` og en for tabellen `cupcake.Invoice_Details`, ved at sætte `setAutoCommit()` til `false` og selv kalde `commit()`. Dette gøres for at være sikker på at der ikke sættes data ind i `cupcake.Invoice_Details` uden at der er korresponderende data i `cupcake.Invoice` først, da kolonnen `Id_Invoice` er `Primary Key` i `cupcake.Invoice` og `Foreign Key` i `cupcake.Invoice_Details`.

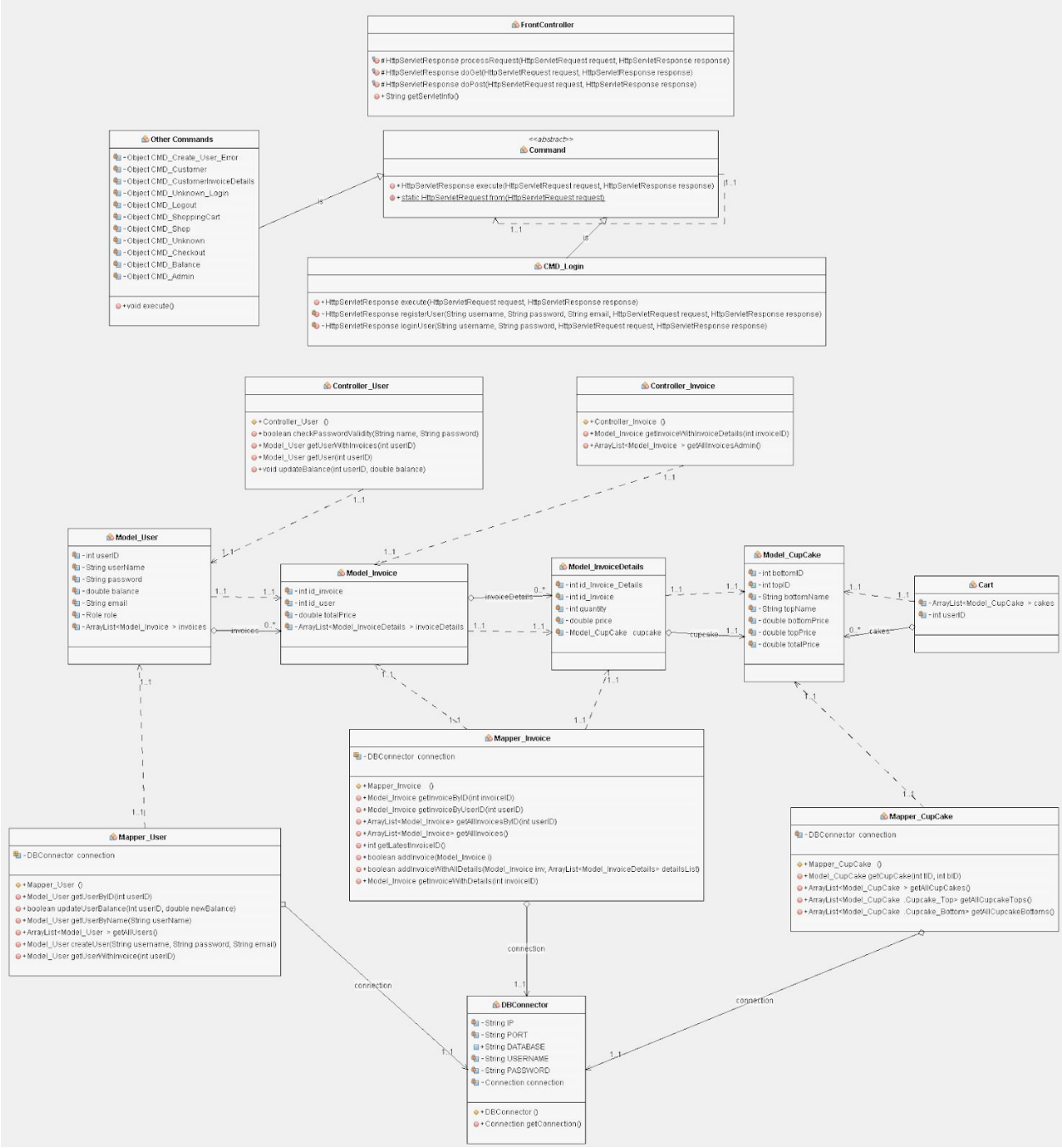
I `Mapper_User()` sørger `updateUserBalance()` for at skrive kundens Balance efter købet til databasen, igen via `PreparedStatement()`.

Retur til Commanden
Hvis disse kald til databasen går godt returnerer metoderne til `CMD_Checkout()` som opdater `Model_User()` objektet med den korrekte Balance og overskriver det `Model_User()` objekt der ligger i `Session()` med det nye.

Til sidst `forwardes` der via `getRequestDispatcher()` til `ShopCheckOut.jsp` som viser en besked til kunden om at ordren og betalingen er gået igennem og med mulighed for at returnere til Customer Page.

Klassediagram

[Link til fuld størrelse]



Beskrivelse af klassediagram

Dette klassediagram viser opbygning af applikationen. Der er blevet taget visse friheder i opbygningen af klassediagrammet for overskuelighedens skyld. Eksempelvis er alle get- og set-metoder fjernet fra klassediagrammet. De klasser, der derimod ikke har nogen fields, men som kun har nogle få metoder er der ikke rørt ved, og de indeholder alle deres metoder og constructors.

En yderligere undtagelse er vores mange implementationer af `Command()` klassen. `Command()` klassen er en abstract klasse, der fungerer som superklasse for alle de klasser, som vi har valgt at kalde CMD_navn. Alle CMD-klasserne arver fra Command-klassen og har dermed adgang til metoden `execute()`. Indholdet af metoden er forskelligt i CMD-klasserne, men strukturelt set er de alle ens. I klassediagrammet vurderede vi at det blev for uoverskueligt hvis alle CMD-klasserne skulle vises, så de er alle (på nær `CMD_Login()`) blevet flyttet ind i én klasse i diagrammet, og denne klasse har så metoden `execute()`, hvilket symboliserer at alle CMD-klasserne der står her, har adgang til denne metode.

`CMD_Login()` er brugt som eksempel i klassediagrammet da den er forskellig fra de andre CMD_klasser idét den har flere metoder. `CMD_Login()` vises derfor separat.
Hver CMD står for at behandle det givne `HttpRequest()` og bruge en `Dispatch()` til at `forward()` til den rigtige jsp-side. CMD_klasserne har mulighed for at benytte `Controller()` klasser i logiklaget og `Mapper_navn()` i datalaget.
`FrontController()` opretter et `Command()` objekt, baseret på pathinfo og kalder `execute()`.

Særlige forhold

Session

Når en bruger er blevet valideret ved login oprettes et User-objekt i programmet. Dette objekt gemmes i session. Når brugeren logger ind, opretter systemet også et cart-objekt, som bruges når brugeren tilgår shoppen for at købe cupcakes. Cart-objektet indeholder de cupcakes, som brugeren vælger i shoppen. Cart-objektet bliver tømt eller nulstillet, når brugeren har placeret en ordre. Både User-objektet og cart-objektet bliver fjernet fra session, når brugeren logger ud.

Håndtering af exceptions

Exceptions bliver håndteret på normal vis med try-catches. Metoderne i datalaget kaster exceptions og disse exceptions bliver håndteret længere ‘oppe’ i systemet, hvor det giver mening at catch’e exceptions så man f.eks. kan give en fejlbesked til brugeren og forklare at noget er gået galt. Programmets håndtering af exceptions er dog ikke perfekt og kunne forbedres.

Validering af brugerinput
Det er ikke meget input brugeren kan give til systemet. De få ting, brugeren kan angive er:

- Brugernavn og kodeord**:** Valideres af javakoden ved at tjekke for match i databasen
- Antal af kager, brugeren vil købe samt hvilke slags**:** Dette håndteres af servletten og JSP-siderne
- Antal £ der skal tilføjes til brugerens balance**:** håndteres i JSP

Validering ifbm Login
Når en bruger er logget ind, bliver brugerens oplysninger gemt i session. Det er ikke muligt at tilgå shoppen eller andre dele af websiden (andet end forside og login-side), hvis ikke man er logget ind (session og brugerens rettigheder tjekkes hele tiden).

Case sensitivity
Brugernavn er case insensitive
Kodeord er case sensitive
Email er case insensitive

Brugertyper
Der findes kun to typer brugere: Admin og Bruger. En admin kan se alle regninger og alle detaljer for alle regninger, hvor en bruger kun kan se sine egne regninger og detaljer. En admin har samme funktioner som en bruger, men har bare rettigheder til at se alle informationer. Dvs. en admin kan fx også bestille cupcakes.

Brugen af data fra databasen
Databasen er bygget op med tanke på at dataen skulle ligne deres respektive Java-objekter. Dog er det anderledes i tilfældet for `Model_Cupcake()`, hvor at `Cupcake_Top` og `Cupcake_Bottom` bliver sammensat til ét Cupcake-objekt.
Senere i projektet fandt vi en nødvendighed for at vise toppe og bunde hver for sig, hvilket vi løste ved at oprette en `Model_Cupcake()` inner (nested) class for begge tilfælde.

Status på implementation

Den overordnede funktionalitet, inklusiv de fleste krav og styling af applikationen er nået. Det er muligt at oprette en bruger/logge ind, der er en oversigt over tidligere køb og herunder detaljer af de specifikke køb. Det er muligt at "købe" valuta på siden til køb af Cupcakes. Det er muligt at vælge mellem de forskellige toppe og bunde og sammensætte en kage, vælge antal af den specifikke kage, lægge dem i kurven, fjerne kager fra kurven igen, bestille og betale for ordren. Ordrer, kunder og kager er gemt i databasen. Der er Admin-funktionalitet, hvor en bruger med adminrettigheder kan se alle ordrer og deres detaljer.

Stylingen er foretaget med bootstrap/css og der er indsat repræsentative billeder af nogle af de Cupcakes en kunde har mulighed for at købe.

Mangler og forbedringer
I forhold til krav til applikationen har vi nået de fleste krav og kan præsentere et færdigt produkt. Vi har gennem udarbejdelsen indset flere forhold som vi ikke selv synes er optimale, samt at der er enkelte krav vi ikke opfylder fuldt ud.

Mangler:

- Vi håndterer ikke Exceptions korrekt i det meste af applikationen. For det meste thrower vi Exceptions videre til frontend, som så i nogle tilfælde her er sat op til at catche dem i CMD_klasserne. Vi kunne have lavet vores egne specifikke exceptions som ville kunne bruges gennem programmet og i visse tilfælde sendes ud til frontenden, hvor vi så kunne have brugt dem til at skræddersy error-sider med mere.

- Vi har ikke lavet nogle Junit-tests i applikationen, men har udført manuelle test af applikationen og udbedret de fejl vi har fundet.

- Vi mangler at lave en pænere styling af nogle få af vores Error-sider i applikationen.

- Hele vores frontend er statisk html med css-styling. En del funktioner i frontenden kunne have været lavet bedre med Javascript, som vi slet ikke har berørt i denne applikation. Eksempler på noget vi gerne ville have brugt Javascript til, er håndtering af forkert indtastet brugerdata i Login/Registrer. Ligeledes ville vi i Shop gerne have brugt Javascript til at opdatere antal af den valgte kage, så det ikke var nødvendigt med en genindlæsning af siden og opdatering af items og pris i navigationsbaren.

Forbedringer:

- Vores overordnede struktur og arkitektur i applikationen er rodet og bærer præg af en manglende forståelse af kommunikation gennem en applikation fra frontend, ned i backend og med databasekald, ved projektets begyndelse. Vi har undervejs indset nogle af de fejlvalg vi har truffet i denne forbindelse, men har valgt ikke at ændre for meget i arkitekturen i dette projekt. Et eksempel på dette er vores inddeling af data, logic og presentation lagene. Vores CMD_klasser indeholder egentlig en del logik og burde ligge i logic-laget. Ligeledes bruger nogle af CMD_klasserne vores controllere i logic-laget mens andre ikke gør, hvilket gør vores controllere i logic-laget lidt irrelevante.

- Vores frontend kunne have været sat bedre op til at repræsentere den data vi har i vores tables. F.eks. fremgår quantity ikke på alle tables.

- Bootstrap styling af visse sider (Index.jsp) og elementer (navigationsbaren) er ikke optimeret for lavere opløsninger (mindre viewports).

- Vores opbygning med en enkelt Servlet som FrontController har gjort det svært for os have en korrekt brug af GET/POST i backenden, da vi ikke differentierer mellem de requests der bliver modtaget. Vi indså på et sent tidspunkt i projektet at en løsning kunne være at have to forskellige proccesRequest-metoder med hver deres execute-metoder i FrontControlleren, som så blev kaldt af henholdsvis doGet og doPost og at Command blev udvidet tilsvarende, ved at have ekstra Commands til at håndtere POST requests de steder hvor det er relevant, f.eks. CMD_Login.

Bilag

Bilag 1 - Navigationsdiagram UML

```
@startuml
scale max 800 width

title "CupCakes - Navigation Diagram"
[*] -> Home
Home : Index.jsp
Login : Login.jsp
Home -down-> Login
Login -down-> LoggedIn : Succeeded Login/Register, start Session
Login --> Login: Failed Login/Register, error message

state LoggedIn {
    state Admin {
        AdminPanel : AdminPage.jsp
    }
    [*] -> CustomerPage : Default CustomerPage
    Navigationbar : navigationBar.jsp
    Navigationbar --> HomeLoggedIn
    Navigationbar --> Shop
    Navigationbar --> ShoppingCart
    Navigationbar --> CustomerPage
    Navigationbar --> AdminPanel : if Admin
    Navigationbar --> LogOut
    HomeLoggedIn : Index.jsp
    Shop : ShopPage.jsp
    ShoppingCart : ShoppingCart.jsp
    CustomerPage : CustomerPage.jsp
    InvoiceDetails : InvoiceDetails.jsp
    CheckOut : CheckOut.jsp
    LogOut :
    CustomerPage --> CustomerPage : Add Balance to Customer
    CustomerPage --> InvoiceDetails : Order details
    InvoiceDetails --> CustomerPage : Back to CustomerPage
    Shop --> Shop : Add Cake(s) to Cart
    Shop --> ShoppingCart : Order Cakes
    ShoppingCart --> CheckOut : Buy Cakes
    CheckOut --> CustomerPage : Return to CustomerPage
    AdminPanel --> InvoiceDetails : See order details
    LogOut -up-> Home : End Session
}
@enduml
```

Bilag 2 - Sekvensdiagram UML

```
@startuml
title "Buying CupCakes - Sequence Diagram"
actor Customer
boundary "CMD_CheckOut" as GUI
control "Mapper_Invoice" as mapI
control "Mapper_User" as mapU
database Database

Customer -> GUI : Buy CupCake
GUI -> mapI : addInvoiceWithAllDetails
mapI -> Database : To Database
mapI -> GUI : Confirmation (T/F)
GUI -> mapU : updateUserBalance
mapU -> Database : To Database
mapU -> GUI : Confirmation (T/F)
GUI -> Customer : redirect checkout.jsp
@enduml
```