Cupcake projekt

Marts 2019

Klasse B -		
cph-bb159@cphbusiness.dk	*Benjamin*	*Aeydin24*
cph-cw97@cphbusiness.dk	*Christian*	*cwulfftorn*
cph-ia62@cphbusiness.dk	*Iben*	*IbenKAndersen*
cph-nd76@cphbusiness.dk	*Nicklas*	*TheDanishWonder*

Indledning

Cupcake projekt handler om at udforme en webshop, hvor en bruger kan købe cupcakes til afhentning i en fysisk butik. For at købe cupcakes skal brugeren logge ind på webshoppen og tilføje penge til sin brugers balance. Brugeren har mulighed for at vælge, hvilken bund og topping han ønsker at bestille samt antal af hver cupcake. Webshoppen viser en samlet pris for hver bestilling af cupcakes samt en totalpris for hele bestillingen. Når brugeren er færdig med at bestille cupcakes, kan han trykke på checkout, hvorefter der bliver trukket penge fra brugerens balance. Hans cupcakes er herefter klar til afhentning i butikken.

Baggrund

Beskrivelse af den virksomhed, der skal bruge systemet

Virksomheden kan producere cupcakes efter en bestilling på en webshop af bottoms, toppings og antal.

Virksomheden producerer cupcakes i en butik, hvor kunden kan afhente sin bestilling.

Hvilke krav har kunden til systemet

Kunden skal kunne vælge en bottom og en topping til en cupcake og vælge et antal, som kunden ønsker at købe.

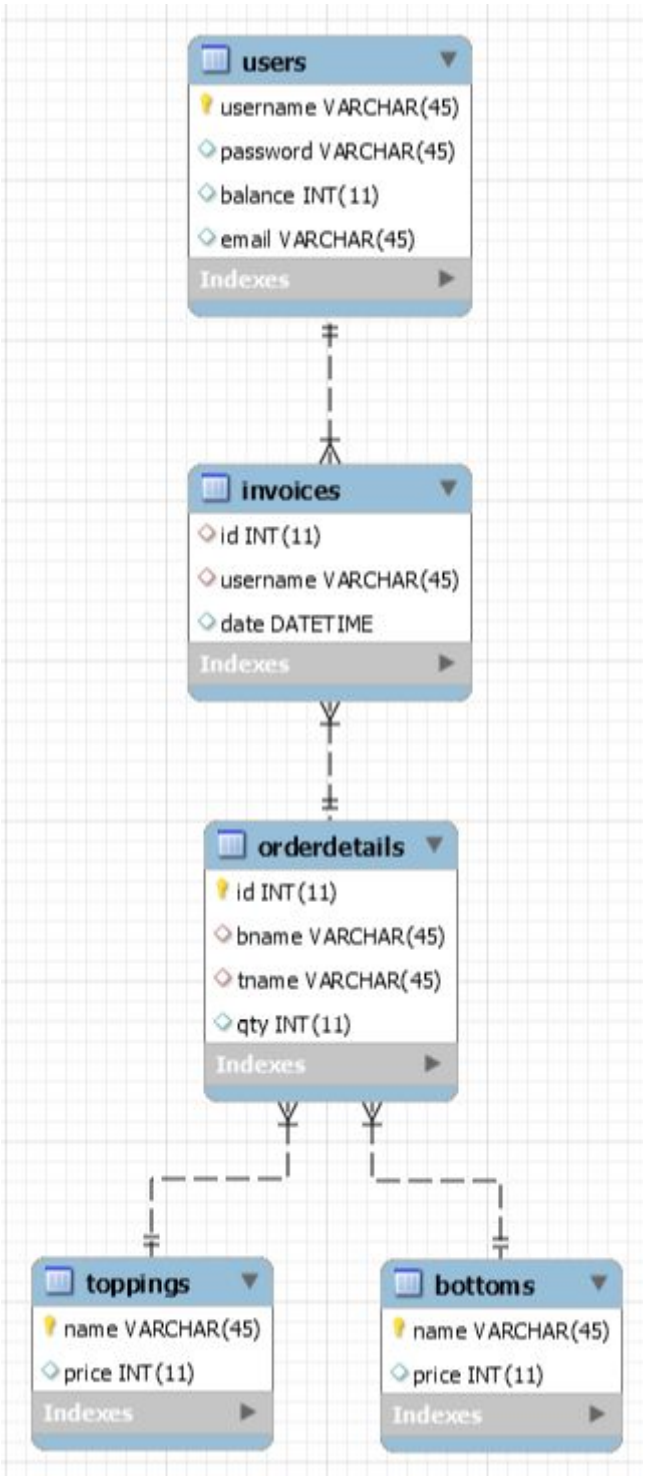
Kunden skal kunne tilføje flere bestillinger af cupcakes, som bliver lagt sammen i en kurv.

Kunden skal kunne tilføje penge til sin balance for at købe cupcakes.

Teknologivalg

- NetBeans 8.2 (Til at skrive HTML, JAVA og Javascript)
- MySQL Workbench 8.0 (Til databasen (Users, cupcakes, orderdetails og invoices)
- JDBC 8.0.15 (Til at connecte til databasen)
- Tomcat 9.0.8 (Som server)
- Digital Ocean (Til at hoste vores webpage)
- Github (Til at gemme projektet globalt)
- JDK 1.8

ER-diagram


users
username VARCHAR(45)
password VARCHAR(45)
balance INT(11)
email VARCHAR(45)
Indexes
invoices
id INT(11)
username VARCHAR(45)
date DATETIME
Indexes
orderdetails
id INT(11)
bname VARCHAR(45)
tname VARCHAR(45)
qty INT(11)
Indexes
toppings
name VARCHAR(45)
price INT(11)
Indexes
bottoms
name VARCHAR(45)
price INT(11)
Indexes

Overvejelser bag ER-diagram

Tabeller med mange-mange relation

Toppings- og bottoms-tabellerne har en mange til mange relation. Vi har lavet en tabel med orderdetails, der kombinerer disse. Orderdetails har to foreign keys til hhv. toppings name og bottoms name.

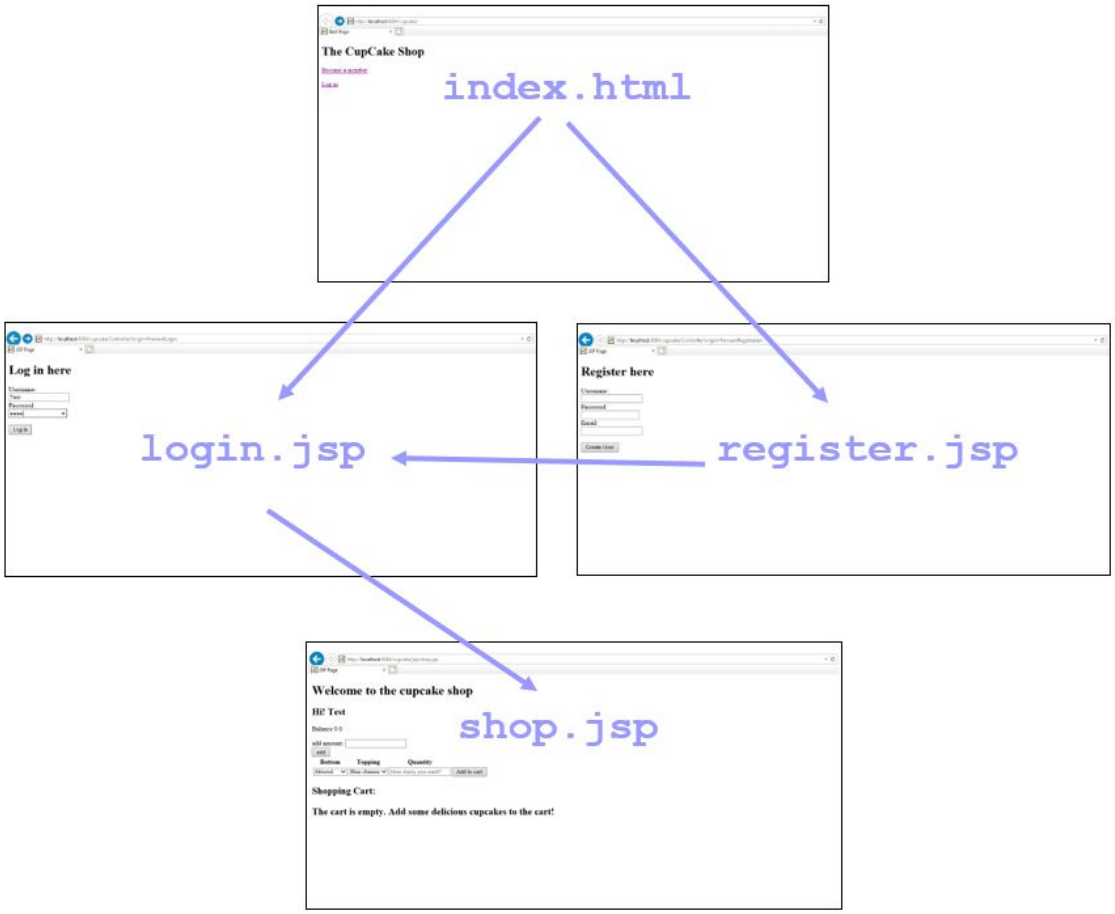
Users- og orderdetails-tabellerne har også en mange til mange relation. Vi har her lavet en tabel med invoices, som kombinerer disse to. Invoices har to foreign keys. Den ene foreign key går til username i users-tabellen, og den anden går til id i orderdetails.

Tabeller der ikke benytter automatisk genereret ID

Vores users-tabel benytter sig ikke af automatisk genereret ID. Vores tanker bag dette er, at der ikke er to brugere, der kan have samme brugernavn. Derfor har vi valgt kolonnen 'username' som private key i denne tabel.

Både bottoms- og toppings-tabellerne har heller ikke automatisk genereret ID. På samme måde som ved users, kan man ikke have flere smage med samme navn. Derfor har vi valgt name hos hhv. bottoms og toppings som private key.

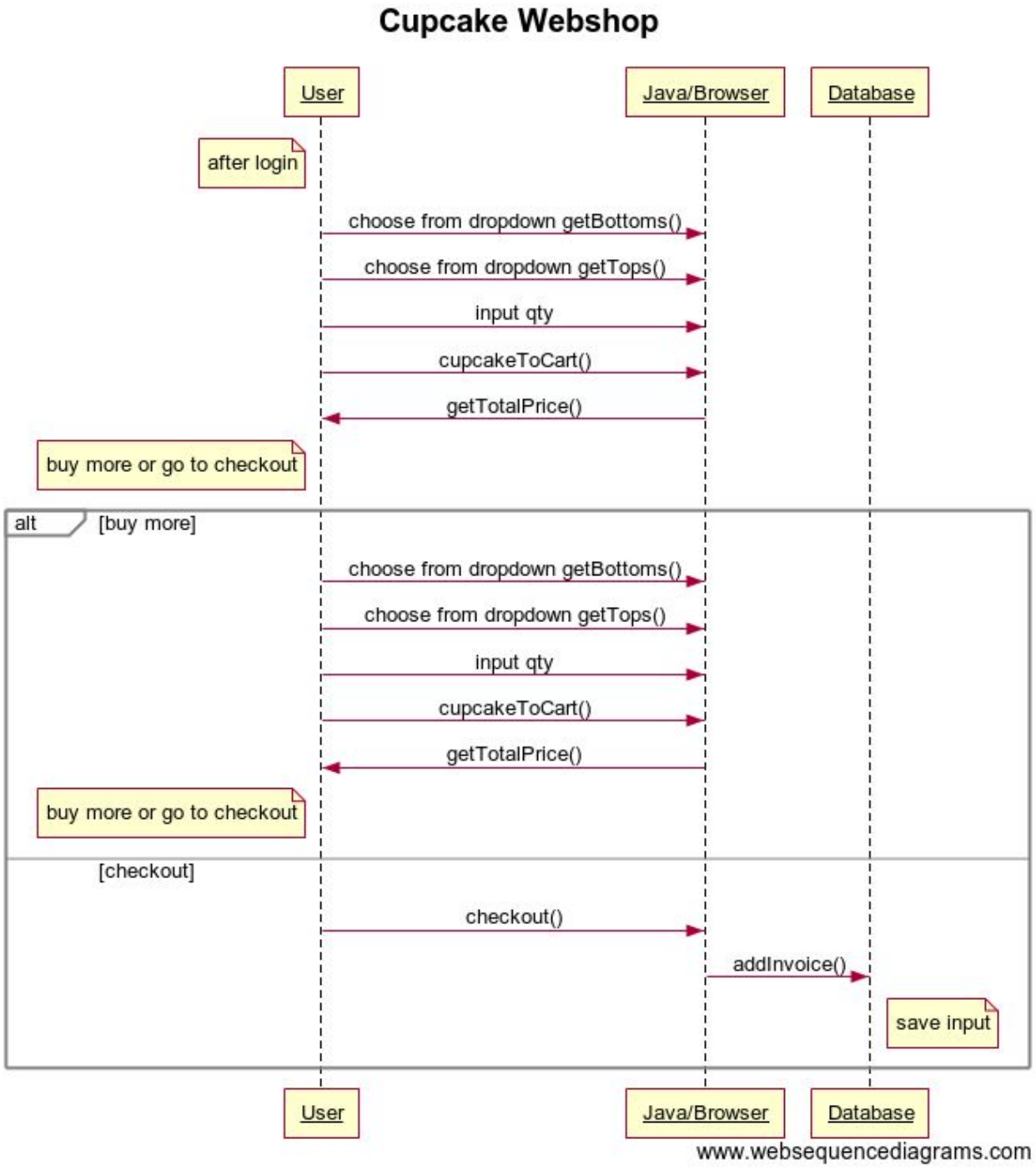
Navigationsdiagram



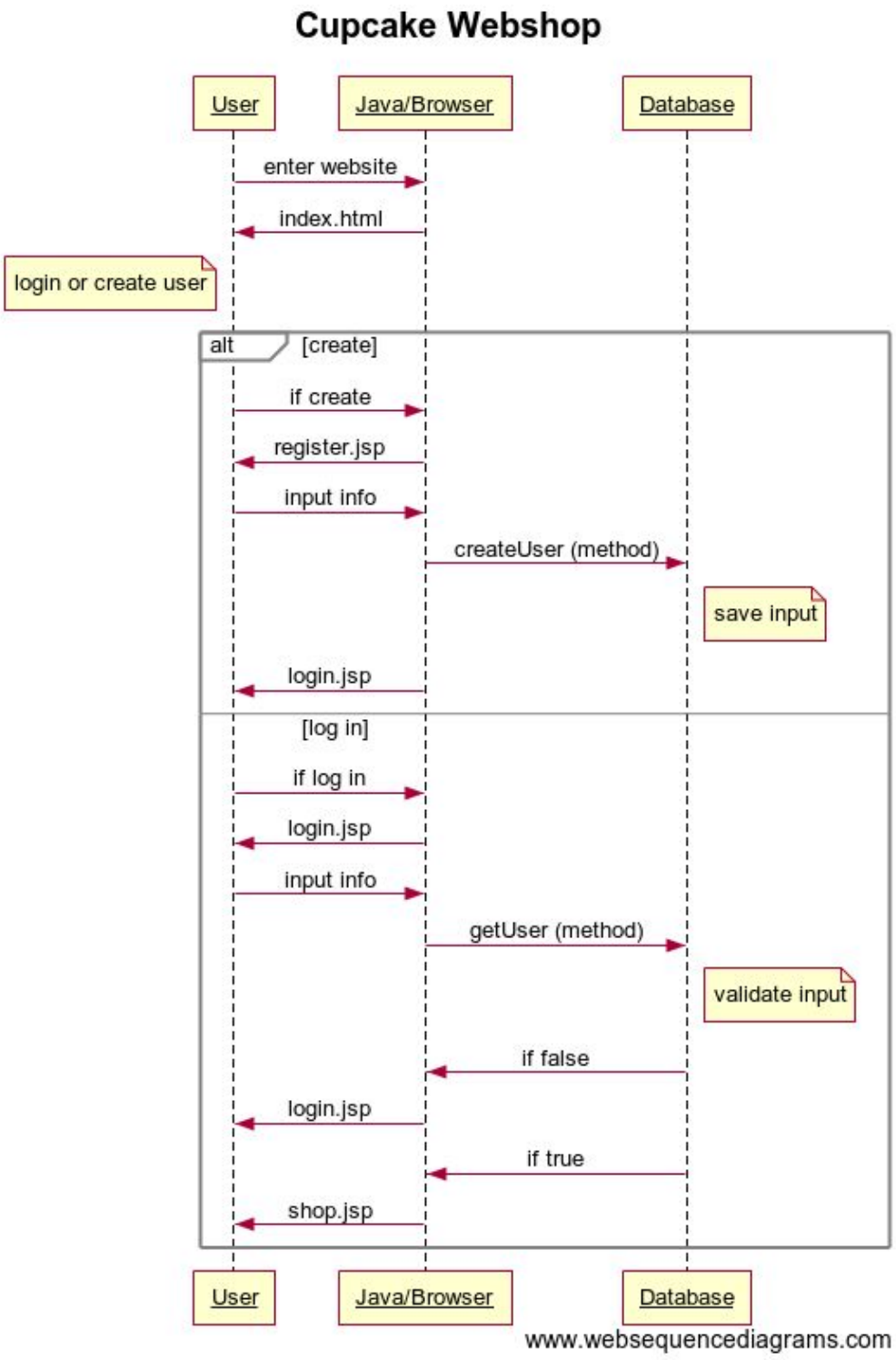
Vores servlet WebController sørger for at brugeren kan navigere rundt mellem siderne med valideret data. I WebController-servletten har vi en *switch*, der har alle vores metoder til de forskellige requests.

Sekvensdiagram

Fra shop.jsp til køb af cupcake



Fra index.html til shop.jsp



Særlige forhold

Hvilke informationer gemmes i session

Brugerens input på login-siden gemmes i session. Ud fra disse input, kan vi kalde de øvrige oplysninger på en bruger inde fra databasen. Dette gør vi ud fra en række hjælpemetoder i vores kode, hvor vi bruger brugerens username som input til selve metoden. Dette gør det muligt at kalde brugerens balance, så vi kan vise den på webshoppen, og det gør det muligt for brugeren at tilføje penge til sin balance, der bliver gemt i databasen med det samme.

Når vi gemmer en brugers input i session er vi derved sikre på, at det er den rigtige bruger, vi har med at gøre. Det er den bruger, der er logget ind på webshoppen, vi gemmer, og det er derfor kun oplysninger på denne bruger, der er relevant for os.

Hvordan håndterer vi exceptions

SQLException

Vi kaster SQLExceptions, da vi får vores data fra og gemmer vores data i en sql-database. Når man kaster en SQLException får man den relevante fejlkode, hvis noget går galt i programmet, når vi forsøger at tilgå databasen.

Vi bruger SQLException, når vi connector til vores database i vores DBConnector-klasse, og vi bruger den, når vi vil hente data fra databasen til bl.a. at validere brugerinput, og når vi vil gemme ny data i databasen fx en ny bruger, flere penge i balancen eller en invoice på et køb på webshoppen.

ServletException

Vi kaster ServletException i vores servlet WebController. Den viser, når serveren ikke er tilgængelig for brugeren ved at sende et brugbart HTML-svar med fejlmeddelelse til brugeren på hjemmesiden.

IOException

Vi kaster IOException, når en input- eller en output-operation er gået i stå. Vi kaster også denne exception i vores servlet WebController, da det er her, vi registrerer brugerens input og laver metoder til at give et output ud fra brugerens valg og indtastninger. Der kommer en fejlkode, hvis brugeren fx indtaster et input, som ikke kan bruges, eller hvis internetforbindelsen ryger, mens man prøver at køre en hjemmeside.

NumberFormatException

Vi kaster NumberFormatException i addBalance-metode i vores servlet WebController, fordi vi i denne metode konverterer en streng, fra brugerens input, til en integer. Denne exception bliver kastet for at indikere at applikationen har prøvet at konvertere en streng til en tal-type, men at strengen ikke har den rigtige format.

Hvordan validerer vi brugerinput

Alle input fra brugeren bliver gemt i vores sql-database. Det gælder bl.a. i det tilfælde, når brugeren laver en ny user. Så bliver inputtet tilføjet i databasen som en ny user i tabellen users. Når en bruger prøver at logge ind på vores webshop, valideres hans input ved at tjekke at brugeren eksistere i vores sql-database, med hhv. det brugernavn og password, han taster ind på login-siden.

Hvis der ikke findes en bruger med det brugernavn og password, der er indtastet, får brugeren at vide, at der ikke findes en bruger med disse inputs, og bliver sendt tilbage til registrering.

Hvis der findes en bruger med de indtastede inputs, bliver brugeren ført videre til shoppen, hvor han kan købe cupcakes og tilføje penge til sin balance.

Hvordan er vores sikkerhed i forbindelse med login

På login siden, har vi brugt html “password” type, så brugerens password ikke bliver vist når det indtastes, og når brugeren logger ind bliver username og password sendt via html “POST” Method, som gør at der ingen af værdierne bliver sendt med over i url’en, som ville ske hvis vi brugte “GET” Method.

Hvilke brugertyper har vores database

Der findes pt kun brugere, da vi ikke har haft tid til at kigge på admin.

Brugere kan tilføje cupcakes til en cart og købe dem, brugeren kan også tilføje penge til deres account.

Status på implementation

Hvad mangler vi at lave

Vi mangler at style vores webshop.

Vi mangler at kunne køre vores projekt på en Digital Ocean-server. Projektet er lagt op, men det kan ikke køre, fordi databasen ikke ligger på serveren.

Vi mangler at lave Customer-page og Admin-page (røde opgaver).

Test

Forskellige metoder er oprettet for at være sikker på at vores cupcake shop henter den korrekte data fra vores SQL database. Der er derudover også tilføjet tests som tilføjer users til databasen.

Vores tests er skrevet med en try-catch blok som, i tilfælde af at testen skulle fejle på database-siden, ville udskrive den relevante fejlkode. Der er *system.out.println();* hele vejen igennem de fleste metoder så man kan se hvornår hvad bliver udført og hvis det går galt se hvor det er.