Navn	Email	Github
Benjamin Skovgaard	cph-bs190@cphbusiness.dk	benskov95
Frederik Janum	cph-fj108@cphbusiness.dk	FreddyBoi95

Cupcake webshop

Navn	Email	Github
Benjamin Skovgaard	cph-bs190@cphbusiness.dk	benskov95
Frederik Janum	cph-fj108@cphbusiness.dk	FreddyBoi95

Contents

Navn	Email	Github
Benjamin Skovgaard	cph-bs190@cphbusiness.dk	benskov95
Frederik Janum	cph-fj108@cphbusiness.dk	FreddyBoi95

Indledning

Vi har brugt de sidste 2 uger på at lave en cupcake webshop, som skal leveres til butikken Olsker Cupcakes, der ønsker at give deres kunder bedre og bredere mulighed for at købe deres cupcakes. Opgaven er blevet løst med IntelliJ som IDE, og vi har brugt MySQL Workbench til at gemme vores databaseinformationer. Selve websiden er blevet kodet i HTML (senere lavet om til jsp sider), og vi har anvendt en given skabelon som fundament til java-delen af programmet.

Denne skabelon kører med en servlet (FrontController) og klassen Command, der styrer navigationsdelen af websiden. Det er gennem Command klassen, at diverse jsp sider får instantieret deres respektive java-kodede funktioner, og gennem FrontControlleren, at man sørger for at alle requests bliver håndteret korrekt – dvs. Sørger for, at brugeren kommer til den rette side for eksempel.

Vi har kørt og testet websiden gennem en lokal TomCat server i IntelliJ, og har senere smidt projektet op på vores DigitalOcean droplet, der også har en TomCat server installeret. På den måde har vi rykket projektet fra et blive kørt lokalt til at køre online gennem vores droplets IP adresse.

Navn	Email	Github
Benjamin Skovgaard	cph-bs190@cphbusiness.dk	benskov95
Frederik Janum	cph-fj108@cphbusiness.dk	FreddyBoi95

Baggrund

Vi har fået opgaven fra butikken Olsker Cupcakes, som er en lille cupcakebutik på Bornholm, der ønsker at få lavet en hjemmeside til deres butik. Olsker Cupcakes er et dybdeøkologisk iværksættereventyr, som eftersigende har ramt den helt rigtige opskrift, og det er gennem et møde mellem nogen københavnere, at vi har fået kravene til det færdige produkt.

I dette møde blev kravene konkretiseret i form af nogle userstories, og de kan ses nedenfor:

US-1: Som kunde kan jeg bestille og betale cupcakes med en valgfri bund og top, sådan at jeg senere kan køre forbi butikken i Olsker og hente min ordre.

US-2 Som kunde kan jeg oprette en konto/profil for at kunne betale og gemme en en ordre.

US-3: Som administrator kan jeg indsætte beløb på en kundes konto direkte i MySql, så en kunde kan betale for sine ordrer.

US-4: Som kunde kan jeg se mine valgte ordrelinier i en indkøbskurv, så jeg kan se den samlede pris.

US-5: Som kunde eller administrator kan jeg logge på systemet med email og kodeord. Når jeg er logget på, skal jeg kunne min email på hver side (evt. i topmenuen, som vist på mockup'en).

US-6: Som administrator kan jeg se alle ordrer i systemet, så jeg kan se hvad der er blevet bestilt.

US-7: Som administrator kan jeg se alle kunder i systemet og deres ordrer, sådan at jeg kan følge op på ordrer og holde styr på mine kunder.

US-8: Som kunde kan jeg fjerne en ordre fra min indkøbskurv, så jeg kan justere min ordre.

US-9: Som administrator kan jeg fjerne en ordre, så systemet ikke kommer til at indeholde udgyldige ordrer. F.eks. hvis kunden aldrig har betalt.

Navn	Email	Github
Benjamin Skovgaard	cph-bs190@cphbusiness.dk	benskov95
Frederik Janum	cph-fj108@cphbusiness.dk	FreddyBoi95

Teknologi valg

- IntelliJ 2019 3.3
- MySQL Workbench 8.0.18 CE
- TomCat 8.5.511
- JDBC Driver (mysql-connector-java 8.0.18)
- JDK: Java 11.0.6
- Git version 2.24.1.windows.2
- DigitalOcean droplet (Linux server – Ubuntu 18.04.4)
- Graphviz (så PlantUML pluginnet virkede)

Krav

Firmaets vision

Olsker Cupcakes håber at de med denne webside potentielt kan tiltrække flere kunder, og give deres eksisterende kunder bedre mulighed for at bestille cupcakes. De er i forvejen udfordret af beliggenhed, da Olsker ligger lidt afsides fra de større bornholmske byer. Derfor er det muligt , at websiden kan øge deres omsætning.

Desuden kan det hjælpe Olsker Cupcakes med reklamation gennem de sociale medier, for eksempel igennem links til hjemmesiden på en Facebook gruppe, Twitter etc.

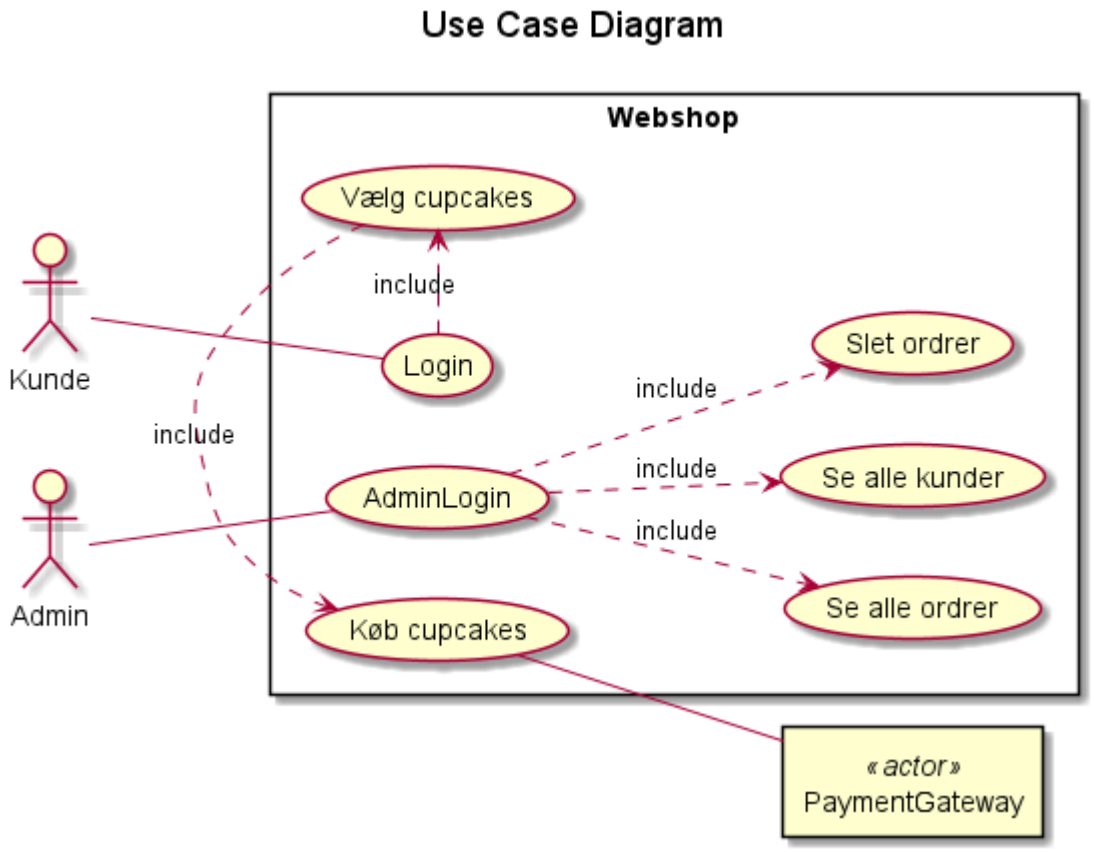
Scrum userstories

Ifølge rapportskabelonen er dette noget, som vi vender tilbage til efter påskeferien. Userstories bliver allerede brugt i baggrundssektionen.

Navn	Email	Github
Benjamin Skovgaard	cph-bs190@cphbusiness.dk	benskov95
Frederik Janum	cph-fj108@cphbusiness.dk	FreddyBoi95

Use-case diagram

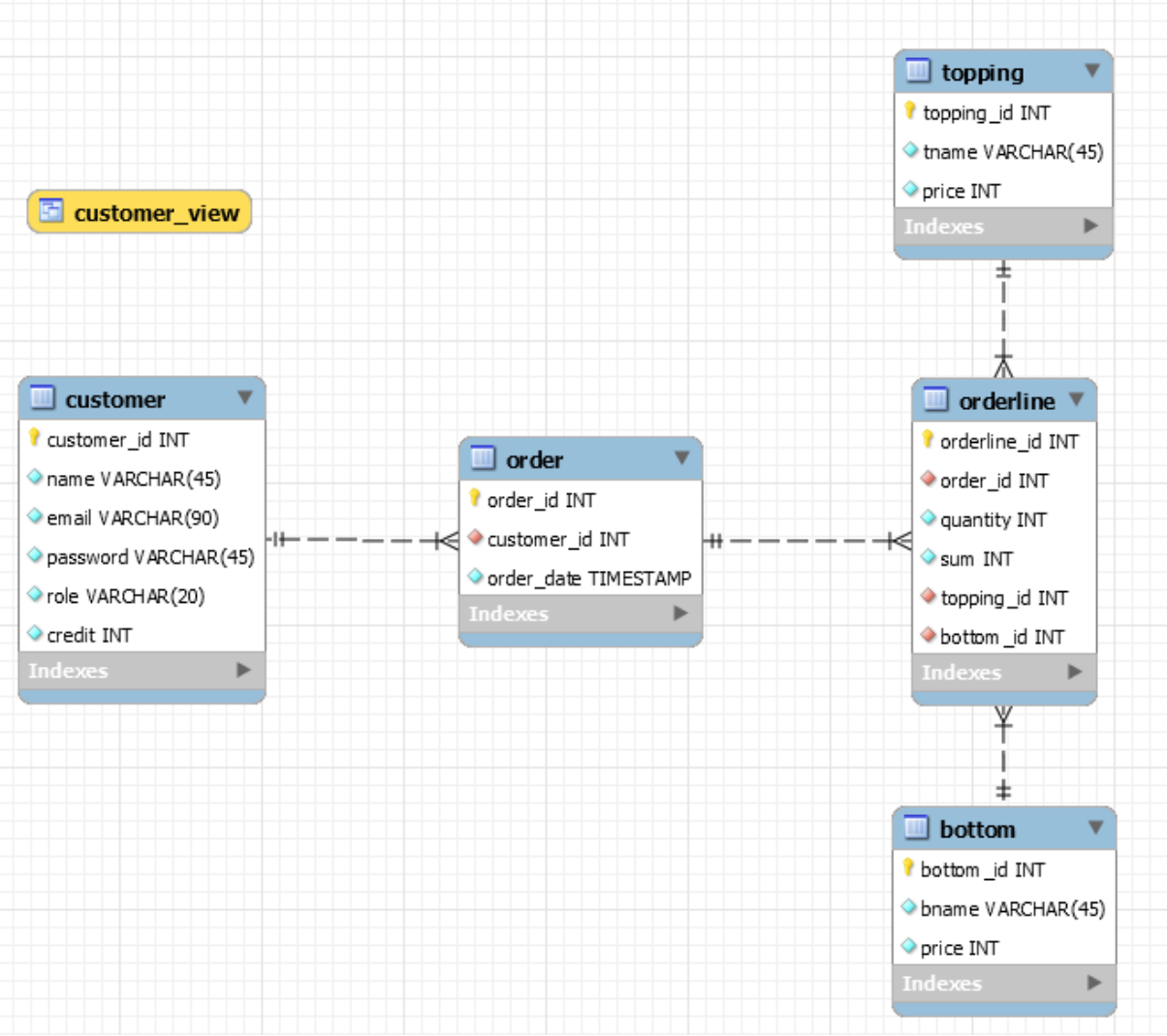
Efter mødet med kunden kom vi frem til dette use-case diagram, som i grove træk viser hvilke funktioner websiden skal have.



Som man kan se, er der to aktører, dvs. To forskellige typer brugere der skal bruge websiden. Kunden kan vælge og købe/bestille cupcakes, mens adminen kan se diverse informationer om kunder, ordrer og modificere informationen i form af sletning af ordrer. Selvom at der er en PaymentGateway, benyttes den ikke på vores webside – men det ville den naturligvis, hvis denne webside skulle rigtigt op og køre for en virkelig virksomhed. Der er dog lavet kode, der sikrer, at man ikke kan købe for flere penge end man har, og også at man kan tanke op, hvis man er ved at løbe tør.

Navn	Email	Github
Benjamin Skovgaard	cph-bs190@cphbusiness.dk	benskov95
Frederik Janum	cph-fj108@cphbusiness.dk	FreddyBoi95

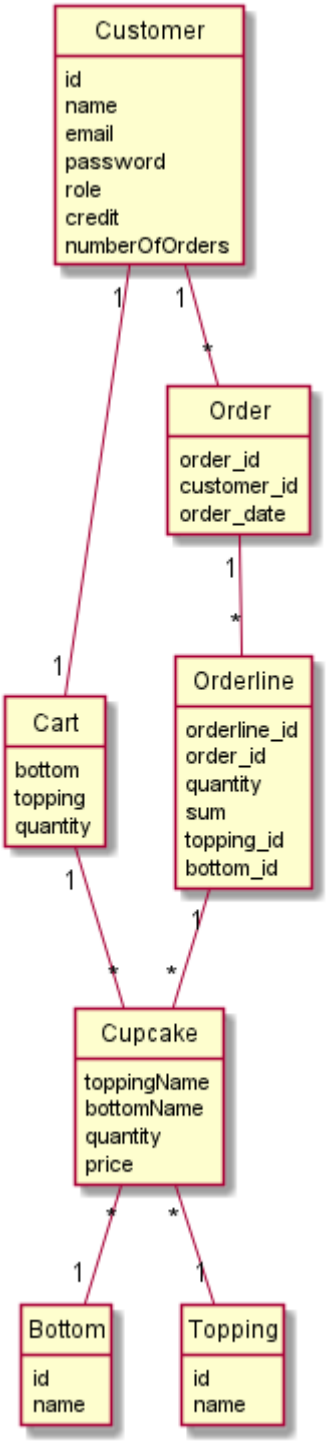
EER-Diagram



Ovenfor kan man se EER-diagrammet fra vores cupcakeshop database. Hvis man starter fra customer-tabellen, kan man se at der er en 1 til mange relation mellem customer og order. Customer id'et bliver så brugt som fremmednøgle i order, og order id bliver brugt som fremmednøgle i orderline. På denne måde sikres det, at der er samhørighed mellem tabellerne (altså at en bestemt kundes ordre altid hænger direkte sammen med selve kunden gennem kundens id).

Yderligere er der så bottom og topping tabellerne, hvis id'er brugeres i orderline som fremmednøgler som del af den færdige ordre. Alle primærnøgler i tabellerne er autogenererede. Vi har også sat en UQ (unique) constraint på email kolonnen i customer tabellen, for at sikre at alle kunde emails er unikke og at man ikke kan bruge den samme til at lave flere kunder.

Domæne-diagram

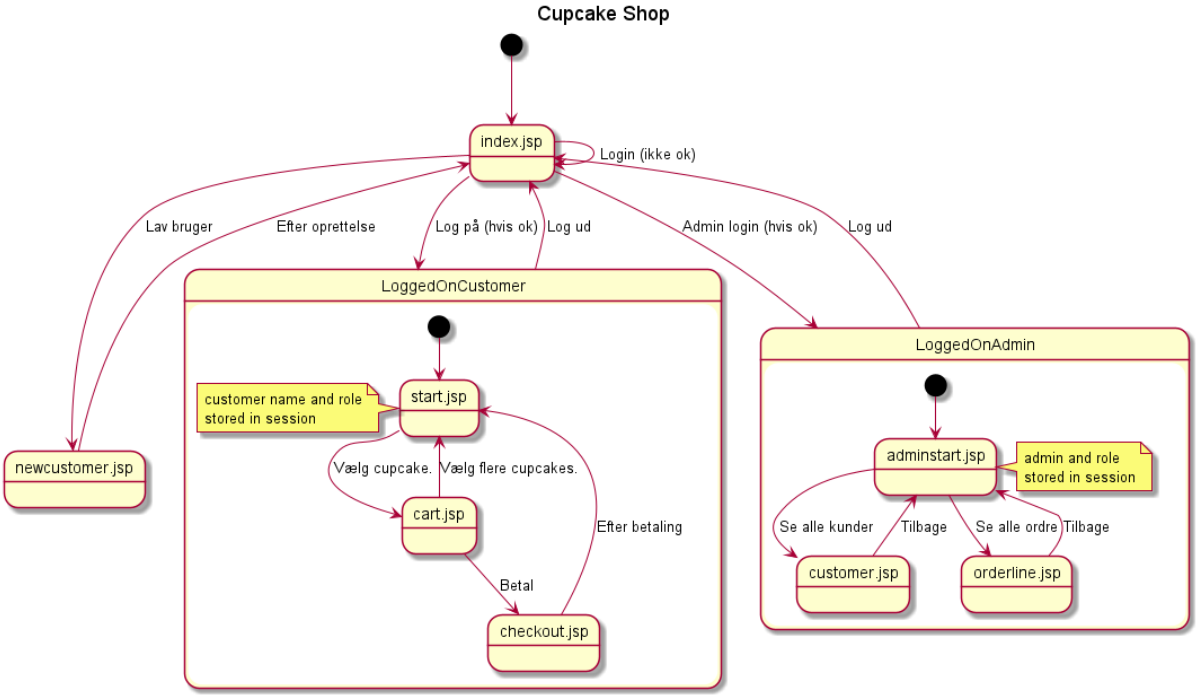


Her ses vores domæne diagram, som er lavet i PlantUML i IntelliJ. Som man kan se, minder det lidt om EER-diagrammet, men der er nogle tydelige forskelle. Der er en Cart klasse, som bliver brugt til at holde alle kundens valgte cupcakes, og en Cupcake klasse, som fungerer som skabelon for de informationer, der senere bliver bliver brugt til at lave ordrene, der sendes til databasen.

Der er også en 1 til 1 relation mellem Customer og Cart. Det skyldes, at kunden rent logisk ikke har flere kurve, men en enkelt kurv som de så kan fylde et antal varer i. Derudover er der Cupcake, som består af nogle informationer, der hentes fra Bottom og Topping (gennem mappers).

I Cupcake bruger vi ikke id'er men navne til bottom og topping, og vi henter så id'erne fra bottom og topping mappers ved, at læse dem ind i to arraylists og hente de id'er der passer med hhv. Bottom og topping navnene gennem sammenligning i for-loops. På den måde har vi id'erne som skal bruges til at lave en Orderline.

Navigationsdiagram



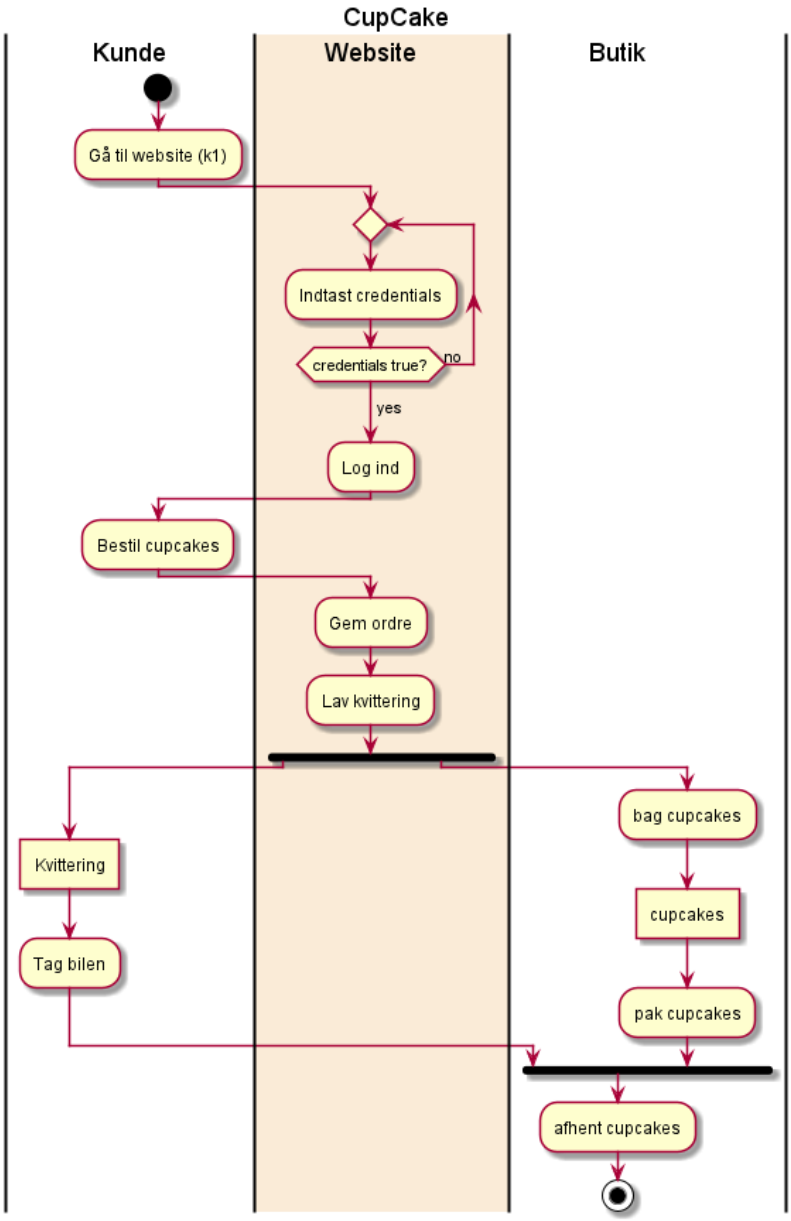
Her kan man se navigationsdiagrammet til vores webside, som er lavet i PlantUML i IntelliJ. Selvom det ikke fremgår her, har vi en fælles navigationsbar på hver side – dog er den en lille smule anderledes for admin. Den har et par simple funktioner:

1. **Log ud:** Logger kunden ud og returnerer dem til index siden.
2. **Mine ordrer:** Sender kunden til myorders.jsp (som ikke fremgår på diagrammet), som er en simpel side, der henter alle kundens ordrer ned fra databasen.
3. **Saldo:** Viser kundens nuværende saldo, som opdateres løbende i databasen og på navigationsbaren, når køb foretages.
4. **Email:** Viser emailen på den kunde, som er logget ind på tidspunktet.
5. **Kurv symbol:** Sender kunden til deres kurv.

Al navigation sker gennem FrontControlleren, som er vores eneste servlet i programmet. FrontControlleren arbejder sammen med Command klassen, for at sende kunden til de rigtige sider, og for at sørge for, at de forskellige sider har de rigtige funktioner med sig (i deres tilhørende javaklasser).

Navn	Email	Github
Benjamin Skovgaard	cph-bs190@cphbusiness.dk	benskov95
Frederik Janum	cph-fj108@cphbusiness.dk	FreddyBoi95

Aktivitetsdiagram



Her kan man se vores aktivitetsdiagram, hvis formål er at vise det overordnede systems adfærd. Ved at starte i det øvre venstre hjørne kan man se, hvordan kunden bruger websiden til at bestille sine cupcakes, og senere afhente dem.

Vi har et loginsystem, som kunden først skal igennem, før de kan lave en ordre. Som man kan se, tester man i systemet om kundens email og kode (credentials) passer med et eksisterende par i databasen.

Hvis dette ikke er tilfældet, så looper man tilbage til indtastning. Der er naturligvis også mulighed for at lave en ny konto, hvis man ikke har en, men det er ikke illustreret her.

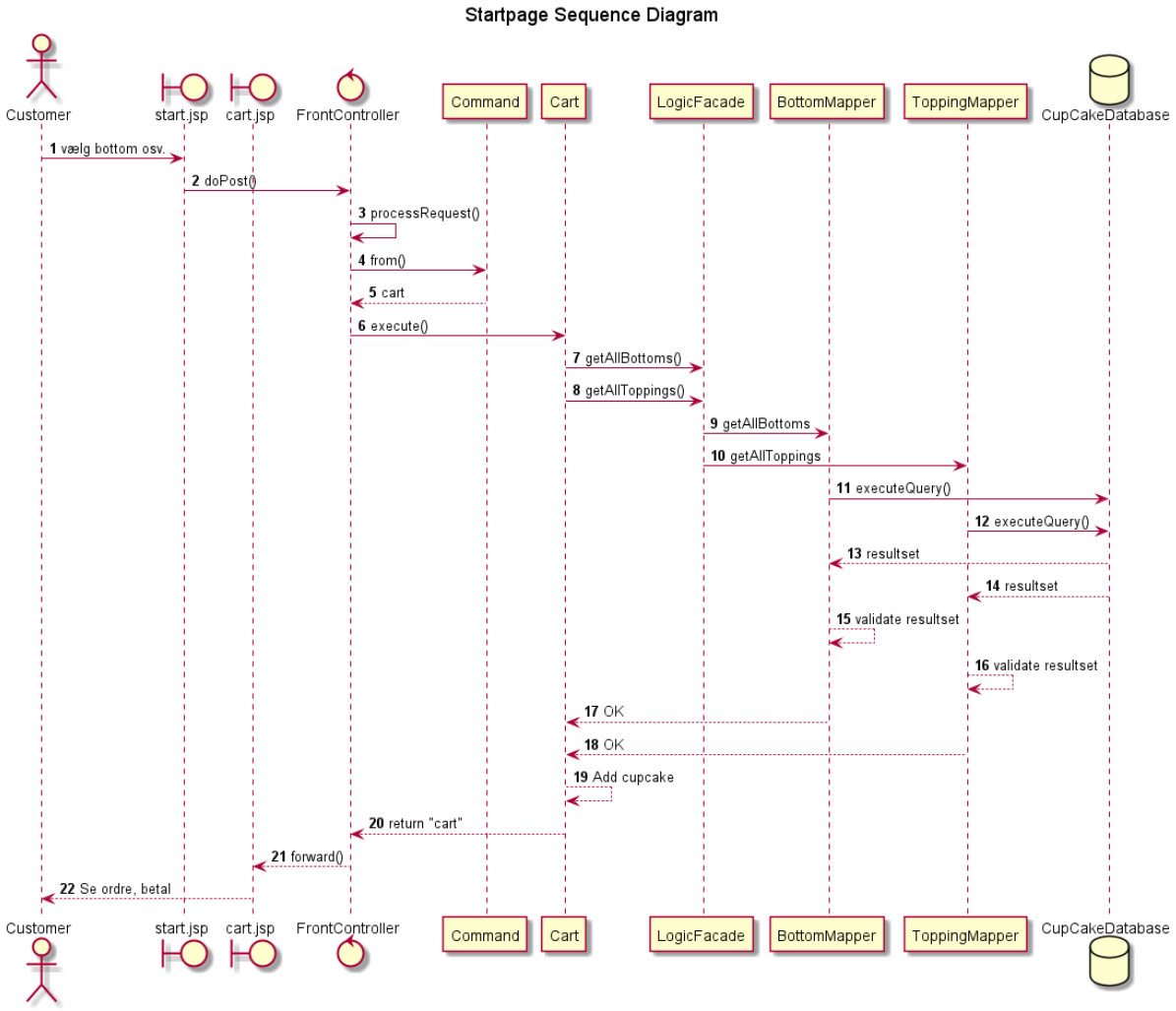
Når kunden har valgt og betalt for sine cupcakes, gemmes ordren. Så bliver kunden videreført til checkout siden, hvor de kan se deres kvittering. Herefter sker resten af processen udenfor systemet.

Hele processen foregår altså gennem tre elementer: kunde, website og butik. Kunden interagerer med websiden, og butikken modtager kundens ordre, som så forberedes og afhentes af kunden senere.

Navn	Email	Github
Benjamin Skovgaard	cph-bs190@cphbusiness.dk	benskov95
Frederik Janum	cph-fj108@cphbusiness.dk	FreddyBoi95

Sekvensdiagrammer

Sekvensdiagram for startside



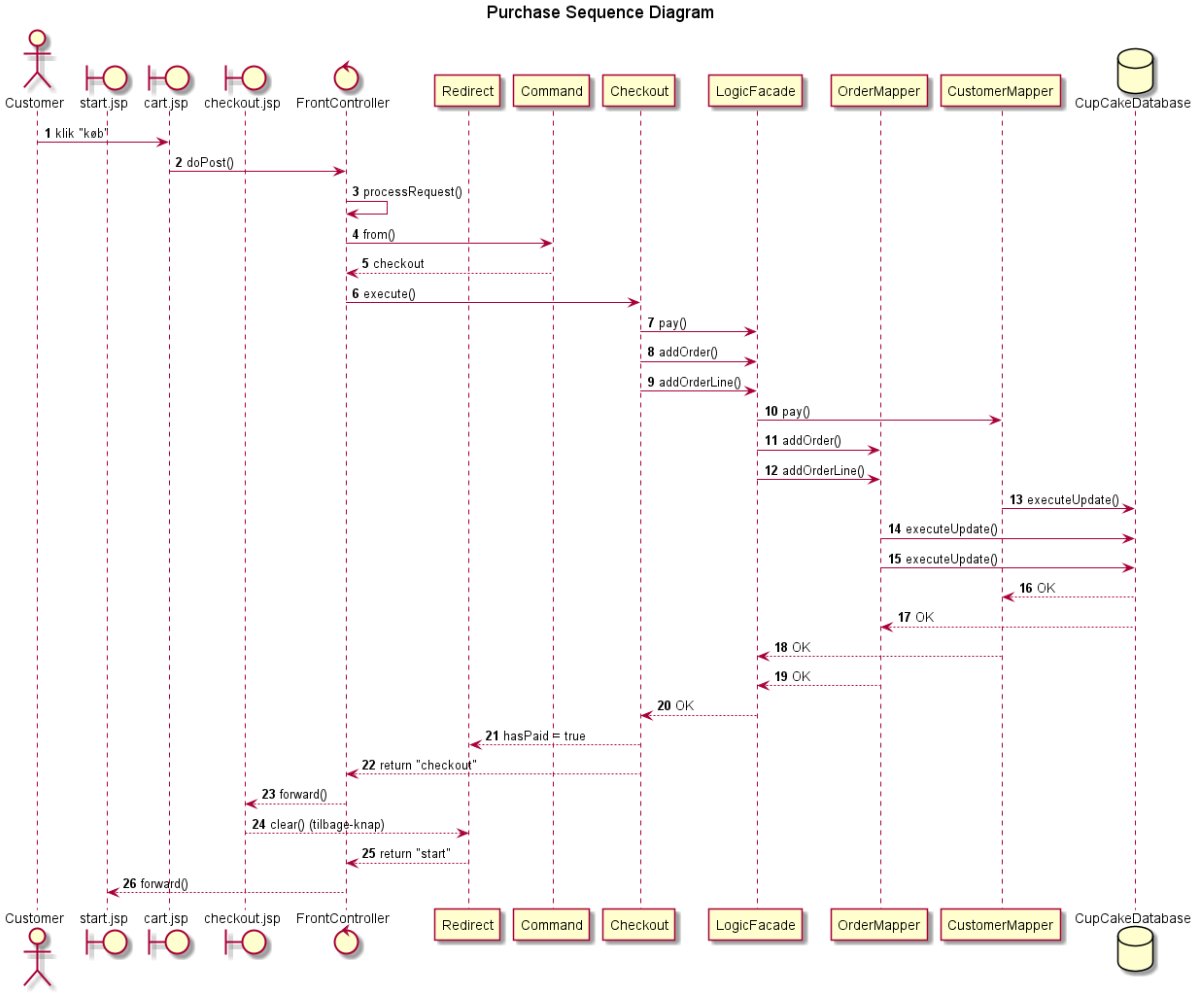
Her er vores sekvensdiagram for den første del af købsprocessen, som består i at lægge et antal cupcakes i sin kurv. Som man kan se, starter kunden på start.jsp siden med at vælge bund, topping og antal af cupcakes – når det er gjort, og kunden har trykket på ”læg i kurv”, sendes valgene og destination videre gennem en HTML form til frontcontrolleren, som kører processRequest() metoden, som gennem Command afgør hvor kunden nu skal sendes hen, og hvilken klasses execute() metode der skal køres.

Fra Cart klassen køres der så to mappermetoder fra BottomMapper og ToppingMapper gennem LogicFacade. LogicFacade er bare en klasse, der holder alle mappermetoder og maskerer deres underliggende kode. Når disse metoder køres, sender de hver en SQL forespørgsel op til databasen gennem executeQuery(), som er en metode fra PreparedStatement klassen. Hvis alt går godt, returneres alle bunde og toppings i databasen, som så biver brugt til at sammenligne med kundens valg for at finde pris, id osv.

Navn	Email	Github
Benjamin Skovgaard	cph-bs190@cphbusiness.dk	benskov95
Frederik Janum	cph-fj108@cphbusiness.dk	FreddyBoi95

Endelig sendes kunden til cart.jsp siden gennem FrontControlleren, som kalder forward() metoden med ”cart” stringen der er blevet returneret af Cart klassen. Herfra kan kunden vælge at gå tilbage til startsiden for at lægge flere cupcakes i sin kurv, ved at trykke på ”jeg er ikke færdig” knappen.

Sekvensdiagram for køb



Her er sekvensdiagrammet for den anden del af købsprocessen. Denne del af processen sker efter kunden har lagt cupcakes i sin kurv, og er klar til at betale. Når der trykkes ”køb” på cart.jsp siden, sendes der en ArrayList med Cupcake objekter og destination videre gennem en HTML form til FrontController servletten. Så køres processRequest() metoden, som så gennem Command klassen afgør, at Checkout klassens execute() metode skal bruges.

I Checkout klassens execute() bliver der kaldt tre metoder, to fra OrderMapper og en fra CustomerMapper, gennem LogicFacade. Først kaldes pay(), som sætter kundens nuværende credit/saldo til en ny værdi, der er beregnet ved at trække prisen af det færdige køb fra kundens saldo. Dette opdateres så i databasen gennem executeUpdate().

Derefter kaldes addOrder(), som tilføjer en ny ordre til databasen med kundens ID med executeUpdate() og returnerer det genererede order ID. Til sidst bruges addOrderLine() metoden inde i et for-loop, som tilføjer en orderline i databasen for hver cupcake i kundens kurv hver gang der loopes. Hver orderline der skabes får det returnerede order ID med, så man kan se, at alle orderlines tilhører den specifikke ordre.

Hvad man ikke kan se på diagrammet er, at vores Cupcake objekt faktisk kun indeholder bund og toppingnavne. Vi skal bruge bund og topping id'er til orderlines, så vi sørger også for, at køre loops på getAllBottoms() og getAllToppings(), for at sammenligne med navnene og finde de tilsvarende id'er, så vi kan bruge dem til at lave orderlines.

Når det så alt sammen er gjort, og alt er gået fint, sættes en boolean ved navn hasPaid til true. Den er vigtig, fordi når vi senere bruger Redirect klassen til at komme tilbage til start.jsp, vil vi gerne sikre at kundens kurv er blevet tømt, men kun hvis der er betalt. Endelig returnerer Checkout klassen "checkout" stringen, som bliver brugt i FrontController servletten. Her kaldes forward() metoden, som videresender kunden til checkout.jsp siden.

Herfra kan kunden vælge at trykke på en tilbage knap, som sender dem til start.jsp siden igen med Redirect klassen, som kører ligeledes gennem FrontControlleren, og med det tryk på knappen bruges clear() metoden på cupcakes ArrayListen hvis den førnævnte boolean er sat til true.

Særlige forhold

Session

Følgende variable og objekter gemmes i sessionScope i Login klassen:

1) hasPaid: En boolean der bruges senere til at afgøre, om cupcakes ArrayListen (kurven) skal tømmes.
2) Cupcakes: Instantieres som en tom ArrayList, der senere skal fyldes med cupcakes af kunden.
3) Customer: Ved login hentes kundens oplysninger fra databasen og bliver lagt i et Customer objekt. Bruges bl.a. til at vise kundens email på hver side på websiden.

Der er naturligvis flere variable og objekter der bliver gemt, men disse er hyppigt brugt og bare et par eksempler på, at man bevarer information der bliver/kan blive benyttet senere.

Vi har ikke brugt applicationScopet på noget tidspunkt, da vi havde problemer med at få fat i servletcontexten i starten, og da vi fandt ud af hvordan vi kunne gøre det, var vi allerede godt i gang, så vi ville ikke ændre på for meget.

Requestscopet er blevet brugt enkelte steder, mest til fejlbeskeder på enkelte sider hvor der kan opstå fejl med input.

Exception håndtering

Der står vel at mærke, at vi kommer tilbage til det senere i semesteret, men der kan stadig snakkes kort om hvordan exceptions håndteres. I langt størstedelen af tilfælde håndteres de enten gennem if-statements eller gennem try-catch blocks. Derudover er der også nogle exceptions, som ligger helt i roden af programmet og kaldes i stort set alle metoder. Det er:

1) LoginSampleException
2) SQLException
3) ClassNotFoundException

Navnene på dem er lidt sigende om, hvad de bruges til. LoginSampleException bliver primært anvendt til loginprocessen, hvor den kastes hvis login informationen ikke stemmer overens med noget i databasen.

SQLException kastes, hvis den SQL sætning der er sendt med i en af mappermetoderne har syntaksfejl eller lignende, eller hvis der af anden årsag ikke kan skabes forbindelse til databasen. ClassNotFoundException bruges i Connector klassen, og kastes hvis JDBC driveren vi bruger ikke kan findes når Class.forName() metoden kaldes i connection() metoden.

Validering af brugerinput

VI har sørget for at håndtere diverse fejl, hvis brugeren skulle taste/vælge noget forkert eller ikke vælge noget overhovedet med try-catch blokke og if-statements. Eksempelvis på startsiden - hvis man ikke vælger antal, bund eller top, kommer man ikke videre til kurven. Man skal vælge en af mulighederne i alle tre felter – det virker ikke med kun et eller to.

Et andet eksempel er når admin skal søge efter ordrer eller kunder. Hvis man nu taster ”hej” i stedet for et ID, ville der normalt opstå en exception (formentlig NumberFormatException eller lignende), men i stedet håndteres den og printer ”Du skal skrive et tal” som fejl besked. Hvis man indtaster et ID, som ikke findes i databasen, printes der også en fejlbesked.

Sikkerhed i forbindelse med login

Vi har sørget for, at alle emails er unikke i databasen, så man er sikker på, at der ikke kommer tre forskellige bruger og laver kontoer med samme email. Det forhindrer også problemer med, at en kunde bliver logget ind som den forkerte. Udover det, har vi jo arbejdet ud fra en skabelon, så størstedelen af loginsystemet er ikke lavet af os, men af Kasper, som har lavet skabelonen.

Navn	Email	Github
Benjamin Skovgaard	cph-bs190@cphbusiness.dk	benskov95
Frederik Janum	cph-fj108@cphbusiness.dk	FreddyBoi95

Brugertyper

Der er brugertyper på i databasen i form af roles, hvor der er ”customer” rollen og ”admin” rollen. De bruges ikke meget i vores program – det bedste eksempel må nok være i Login klassen, hvor der til slut i execute() metoden returneres enten ”start” eller ”adminstart”, afhængigt af brugerens rolle.

Status på implementation

Umiddelbart anser vi projektet som færdigt. Det er blevet testet talrige gange på både loginsystem, købsprocess og alt derimellem, og vi kan med relativ sikkerhed sige, at 90% af alle fejl der kunne opstå er håndterede. Det er selvfølgelig muligt, at vi har overset noget undervejs.

Der var dog en lille fejl der blev opdaget rent sent, som var, at man kunne gå videre til checkout med en tom kurv. Den blev nemt håndteret med et if-statement, som sikrer man ikke kommer videre hvis enten kurvens (dvs. Cupcakes ArrayListen) størrelse er 0 eller hvis prisen er 0.

Konklusion

Alt i alt har projektet været lærerigt, da vi har fået brugt den viden om HTML og servlets, som vi har tilegnet os i de sidste par uger. Der har dog også været ret mange forskellige ting siden vores opstart på 2. semester i februar, og det har gjort, at nogle ting har siddet bedre fast end andre. Vi har også arbejdet med en skabelon som fundament for projektet, så det var selvfølgelig også nødvendigt at bruge lidt tid på, at sætte sig ind i hvordan den hænger sammen, og hvordan man får dannet alle de nødvendige forbindelser i koden.

Vi er umiddelbart tilfredse med slutproduktet, da vi mener, at det er en fin og velovervejet webside som lever op til alle krav, og som kunne fungere fint i den virkelige verden – dog ville den nok have godt af lidt finpudsning.