Cupcake Projekt – Olsker Cupcakes

Gruppe 4, Skrevet af:

GitHub:

https://github.com/cph-jm390/CupCake

Video Link:

Video Rapport

Johan Møbjerg

cph-jm390@cphbusiness.dk

Github.com/cph-jm390

William Foss Nielsen

cph-wn11@cphbusiness.dk

Github.com/Misfosster

Emilio Castro Lagunas

cph-ec100@cphbusiness.dk

Github.com/EmmsDK

Indholdsfortegnelse:

Indledning & Baggrund:

Olsker Cupcakes er et fiktivt projekt dannet på en historie om et par hipstere fra København der har været på et dybdeøkologisk iværksættereventyr på Bornholm. Der har de været forbi en bager ved navn Olsker Cupcakes som har ramt den helt rigtige opskrift. Efter at havde været på besøg hos Olsker Cupcakes har de indsamlet nogle krav og lavet en halvfærdig mockup af en tænkt forside til en bestillingshjemmeside. En mockup er en meget løs udkast af hvordan den færdige hjemmeside skal se ud. Det er nu vores opgave at stille spørgsmål til det manglende funktionalitet og komme med et forslag til en færdiggjort hjemmeside. Denne opgave er er skrevet med fokus på en medstuderende, der ikke har den store viden inde for Cupcake projektet, som fagfælle.

Teknologiske Valg:

Dette projekt er bygget op på baggrund af en SQL-database der holder alt information der skal bruges på hjemmesiden. Front-end af hjemmesiden er bygget med JDBC for at kunne samarbejdede med vores database ved hjælp af Java-Servlets. Back-end af vores hjemmeside er bygget på IntelliJ version 2021.3, hvilket er en ældre version af programmet. Grunden til at vi har brugt en ældre version er fordi der er flere metodekald mellem JDBC og Java der kun virker på den ældre version.

User-Stories:

Målet med dette projekt har været at udfylde de seks første *User-Stories* hvilket var funktionelle krav. Efterfølgende var det yderligere 3 optionelle *User-Stories*.
I første *User-Story* er kravet at som kunde, skal man kunne bestille og betale cupcakes med en valgfri bund og top, sådan at kunden senere kan køre forbi butikken i Olsker og hente sine ordrer. Da vi startede med denne opgave, blev vi hurtigt enige om at den bedste metode, både for Olsker Cupcakes, og for de brugere/kunder der bruger hjemmesiden, vil være at kunden vælger hvilken topping, bund og kvantitet på hjemmesiden. Efterfølgende tjekker kunden deres bestilling ud hvor prisen automatisk bliver trukket fra deres balance der er knyttet til deres bruger.

I anden *User-Story*, skal kunden kunne oprette en konto/profil på hjemmesiden for at kunne betale og gemme sine ordrer. Den nemmeste måde at takle dette ville være at

lave en loginside som kunden rammer som det første når de prøver at komme ind på hjemmesiden. På loginsiden har kunden muligheden for enten at oprette en bruger eller logge ind med en allerede eksisterende bruger. For at dette skulle virke blev vi nødt til at sikre os at de brugere der blev oprette, blev gemt inde på databasen både så kunden kan gemme de ordre der blev lavet, og så kunden kan benytte brugeren til fremtidige ordre.

I *User-Story* tre er karvende at der skulle dannes en administrator som kan indsætte beløb på kundernes kontoer direkte i MySQL, så kunderne kan betale for sine ordrer. Denne *User-Story* hænger meget sammen med US1, hvor kunden skulle kunne bestille og betale for sine cupcakes. Denne *User-Story* er ikke rigtig baseret på kode men mere på at have en funktionel SQL-database. Det vigtigste i denne *User-Story* er at når kunden betaler for sine ordrer bliver beløbet trukket fra den balance der står inde på SQL-databasen.

User-Story fire bestod af at kunden skulle kunne se de valgte ordrelinjer i en indkøbskurv, og at de kunne se den samlede pris på ordren. Denne *User-Story* fik vi koblet sammen med den første *User-Story*. Således at når kunden er inde og vælge hvilken cupcake de vil bestille og kvantiteten af ordren, bliver det sammensat i en ordrelinje der bliver printet med den samlede pris når kunden trykker ”Tilføj til Kurven”.

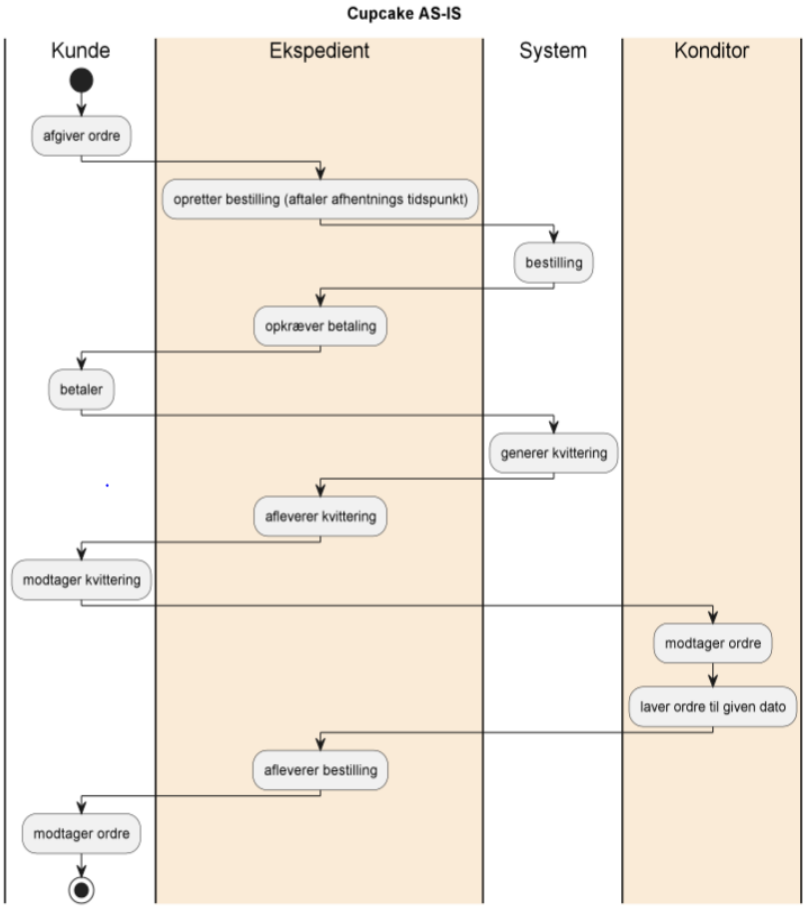
I femte *User-Story* skal både kunden og administratoren kunne logge på systemet med e-mail/brugernavn og kodeord. Når kunden eller administratoren er logget på, skal han, hun, eller andet-køn kunne se sin e-mail på hver side. Det vi gjorde, var at vi brugte en af fanerne i topmenuen til at fremvise den e-mail/brugernavn der er logget ind med. Hvis man trykker på sit brugernavn som en ordinær kunde bliver man henvist til en brugerprofil side. Her kan man se ens brugernavn/e-mail og ens personlige balance, der er koblet op på balancen på SQL-databasen. Hvis man trykker på sit brugernavn som administrator, bliver man henvist til vores administrator side som bliver brugt i *User-Story* seks.

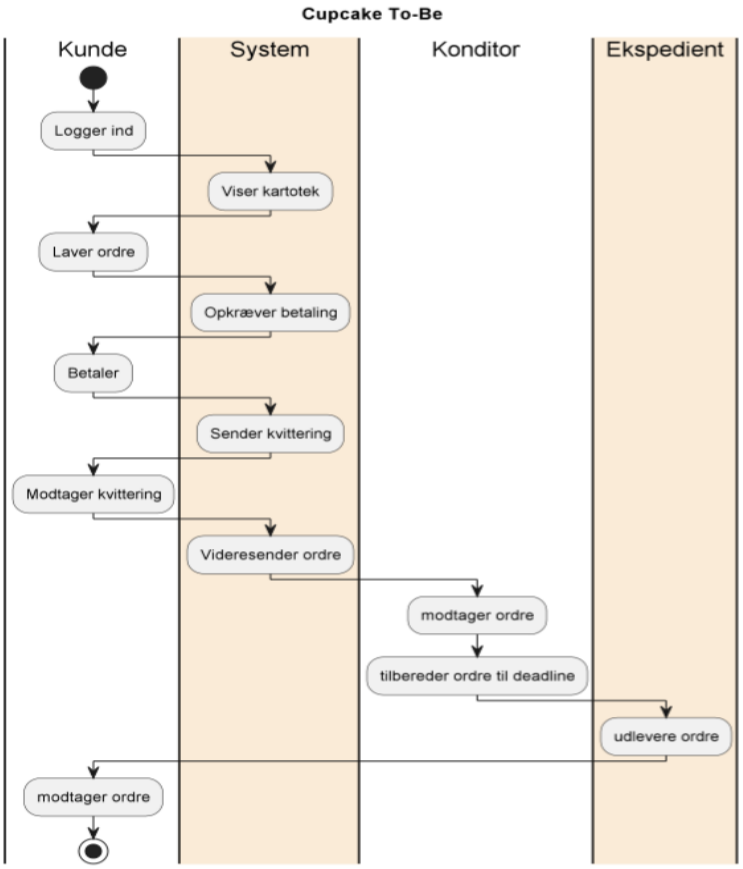
I *User-Story* seks og syv er der fokus på administrator funktionalitet. Når man er logget ind som administrator skal man kunne se alle ordrer i systemet, så man kan se

hvad der er blevet bestilt. Yderligere skal administratoren også kunne se alle kunder i systemet, så man kan følge op på ordrer og holde styr på kunderne. Alt dette forgår inde på administrator siden, hvilket vi fik dannet igennem *User-Story* fem. Når administratoren kommer ind på administrator siden, kan han, hun, eller andet-køn nemlig se alle ordrer og alle kunderne i systemet.

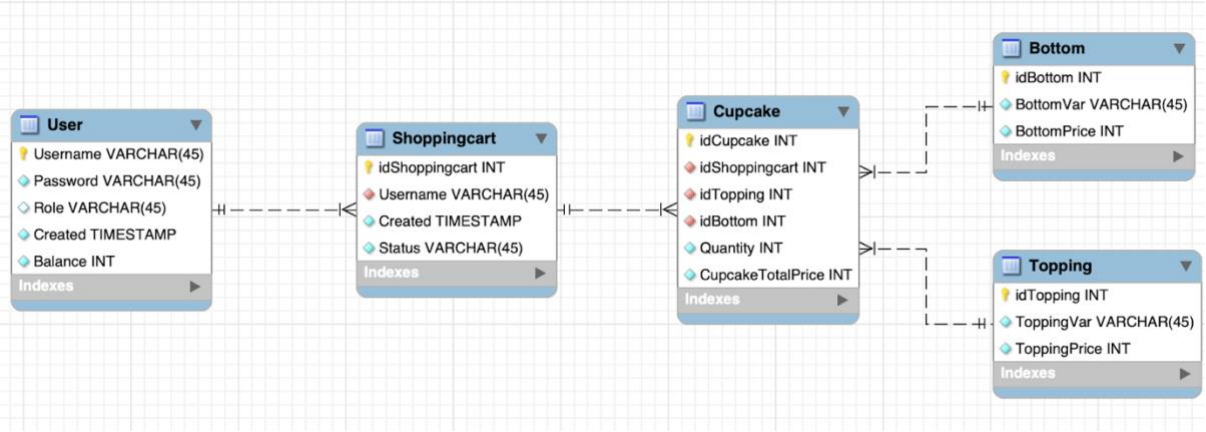
Aktivitetsdiagram

Nedenfor ses to aktivitetsdiagrammer som er blevet konstrueret i forbindelse med Projektet. Der er et AS-IS og et TO-BE aktivitetsdiagram, som beskriver henholdsvis nuværende og kommende forventet arbejdsproces for Olsker Cupcakes. De fleste trin i hvert diagram er relativt selvforklarende. Dog skal det nævnes, at idéen ved at konditoren først får ordren efter kvitteringen er givet til kunden, er fordi Olsker Cupcakes skal være sikre på kunden har fået købsbeviset så det kan fremvises når ordren skal afhentes igen. Dette gøres for at undgå snyd. Når man sammenligner AS-IS og To Be, kan man se langt det meste af arbejdet der kan automatiseres bliver rykket hen til vores system og website. Det vil evt. Resultere i konditoren ikke behøves en ekspedient, da ekspedientens eneste funktion vil være at udlevere bestillingen.





EER-Diagram

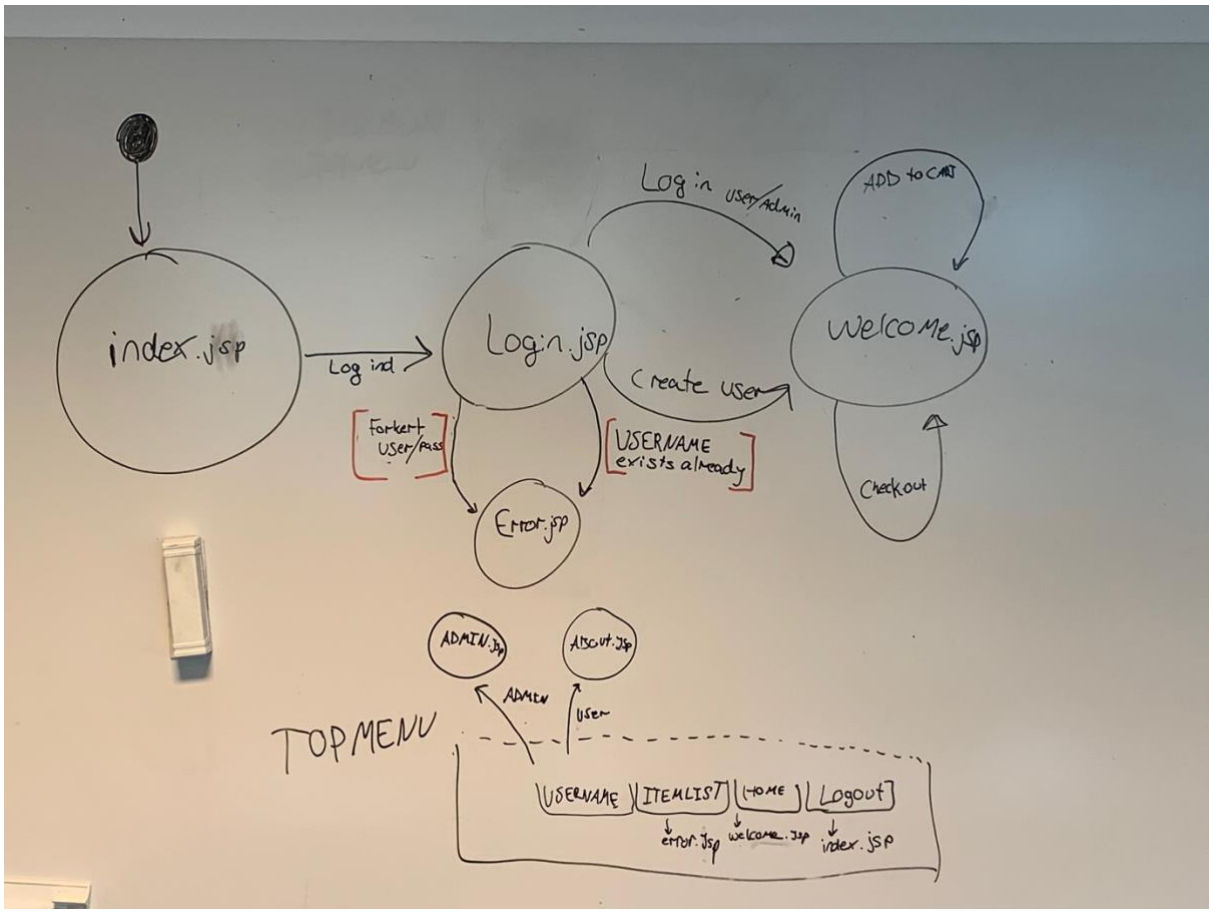


Tanken bag EER-diagrammet har i stor grad været objektorientering. På samme måde som man i Java ville have en bruger, som har en Shoppingcart, osv. har vi baseret databasen på den fysiske relation mellem objekter, som en kunde ville have i, eksempelvis et supermarked.

Alle primære nøgler, med undtagelse af Users, er 'auto-incremented', så vi er sikre på at de har et unikt id. Derudover har vores tanker hele tiden været, at der skulle være en tydelig forbindelse mellem tabellerne, sådan at man, eksempelvis kan få fat i topping, hvis man har en kundes brugernavn. Den måde databasen er sat op på, betyder også, at man som administrator kan se hvilke ordrer der tilhører hvem, samt hvilke kvantiteter og forskellige cupcakes de har bestilt.

Som man kan se på ovenstående diagram, er det cupcake der forbinder User til Shoppingcart, sammen med Topping og Bottom. Dette fordi cupcake tager id'erne fra Topping/Bottom, samt den tilsvarende data, får bestemt en kvantitet, hvorefter de, ved hjælp af, IdShoppingcart har en reference til Shoppingcart, som pakker det ind som en ordre tilhørende brugeren. Endvidere er det også CupcakeTotalPrice, der bliver brugt når kunden skal betale, da brugerens saldo skal være over den gældende pris.

Navigationsdiagram



På ovenstående billede ses vores navigationsdiagram over cupcake-projektet. Når man åbner siden, kommer man som det første ind på index.jsp siden, hvor man skal trykke på log ind for at bliver ført til næste side, login.jsp. Her kan man lave en ny bruger eller logge ind som eksisterende bruger. Når man logger ind, tjekker siden selv givende bruge har admin rolle. Når man er logget ind, bliver brugeren ført til welcome.jsp, hvor man kan bestille cupcakes, og her kommer top-menuen frem for at brugeren nemmere kan navigere. Den eneste reelle forskel på en admin bruger og normal bruger ift. Display, er at admin ikke har en about.jsp (min profil) men i stedet har en admin.jsp, hvor der kan indlæses alle brugere og ordre samt deres informationer.

Særlige Forhold

Session har vi brugt når vi skulle trække noget information fra vores database ind til en JDBC-klasse, ved hjælp af Servlets, for at printe det på hjemmesiden. Vi har blandt andet gemt indkøbskurven i en session for at kunden kan se hvilken cupcakes de har liggende i indkøbskurven sammen med den pris der passer med cupcakesne. Vi har også gemt en liste af alle brugere fra databasen så at administratoren kan trække listen på administrator siden.

Igennem vores projekt har vi brugt exceptions i flere forskellige forhold. Et eksempel vil være når en kunde skal oprette en bruger, der har vi brugt en database exception for at sikre os at kunden ikke kan oprette en bruger med det samme brugernavn der allerede befinder sig i databasen. Vi har brugt den samme exception når en kunde prøvet at logge ind med en kode der ikke passer til noget brugernavn i vores database. Den måde vi har valgt at lave sikkerhed i forbindelse med login har været igennem vores SQL-Database. Når man opretter en bruger på hjemmesiden, bliver den gemt ind i databasen. En bruger er baseret på brugernavnet hvilket altid skal være unikt, hvilket betyder der ikke kan være flere brugere med det samme brugernavn. Hvis en kunde prøver at oprette en bruger med et utilgængeligt brugernavn, får han derfor en fejlbesked. Kundens kodeord bliver derefter gemt på brugernavnet så den bruger altid skal bruge det præcise kodeord for at kunne logge ind på hjemmesiden.

De brugertyper vi har brugt i vores database, er ’user’ for den ordinære kunder, og ’admin’ for administratoren. Når en kunde opretter en bruger, bliver deres brugertype automatisk sat til at være ’user’. Man kan ikke oprette en profil på hjemmesiden med brugertypen ’admin’, den brugertype skal skrives ind i SQL-Databasen direkte. I JDBC har vi brugt brugertyper til at differentiere de forskellige brugere for at kunne sende en ’admin’ bruger ind på administratorsiden på samme knap vi sender en ’user’ bruger ind på deres profilside.

Status på Implementation

På nuværende tidspunkt er der intet der stopper brugeren af websiden fra at trykke på "Home"-knappen inden de er logget ind, hvilket kunne give problemer i form af bestillinger, som tager en bruger som parameter. Dette kun eventuelt løses ved hjælp af et If-statement som tjekker om sessionens user er 'null', og hvis den er, undlader at instantiere "Home"-knappen. Der er stadig mangler i forhold til administrator-rettigheder, da administratoren kun kan se ordrerne, samt ordrenumre – ikke redigere i dem.

Derudover mangler vi en Checkout.jsp, som viser hvad brugeren har købt på hjemmesiden indtil videre. Yderliger, mangler vi muligheden for at slette ordre, da vi ikke har en mellemside der viser indkøbskurven. Indkøbskurven bliver vist idet man opdaterer den igennem "addToCart", men bliver bare vist som tekst. Endvidere, har vi ikke brugt sønderligt meget tid på interfacet - så det har ikke fået så finpudset et 'look' som vi havde ønsket. Til sidst kunne vi også godt have tænkt os at rydde lidt mere op i vores kode, så længere snører af kode eksempelvis blev udregnet i en metode i den passende klasse.

Proces

Processen igennem projektet har haft sine udfordringer, dog mest til at starte med.
Vores plan med projektet i forløbet var at vi ville danne os et overblik over den udleverede startkode. Årsagen til dette er, at vi ud fra det kunne danne os et overblik over hvilke ting som skulle udbygges og evt. ikke var lavet, og herefter lave en strategi for kodningsprocessen ud fra diverse diagrammer og mock-up. I kodningsprocessen var tanken at vi ville arbejde på hver vores ting, men snakke sammen undervejs så vi ikke kom til at ændre klasser et andet gruppemedlem sad i. På denne måde ville vi prøve at undgå merge-konflikter i Git.

Da vi fik startkoden, kunne vi ikke få den til at virke, og da den kom til at virke og vi begyndte at kode, fik vi errors når vi begyndte at navigere rundt på websitet. Vi fandt ud af en uge før aflevering problemet lå i vores Java-sql kode, som har givet os et klart indtryk af hvor vigtigt stavning, stor og lille skrift egentlig er. Disse to ting gjorde at vi

som gruppe havde fire ud af ti dage mindre at arbejde i, hvilket resulterede i, at vi blev nødt til at prioritere, hvad der skulle laves og hvor dybdegående det skulle være. Vi fik derfor som gruppe aldrig lavet et ordentligt mock-up ved hjælp af Figma, hvilket naturligt resulterede i vores displays er uraffineret og grov, da der ikke var en plan at arbejde ud fra.

På trods af vores tidsmangel grundet sql og starkode problemer, gik det stadigvæk overraskende godt. Koden blev lavet relativt hurtigt og vi gik fra at have lavet nul funktionalitet til at have lavet funktionalitet til alle kravmæssige *User Stories* i løbet af et par dage. Dog på grund af manglende modeller er koden ikke struktureret men der i mod til tider rodet. Vi undgik Git merge-konflikter næsten hele vejen igennem projektet, hvilket også har bidraget til at vi kunne kode så meget funktionalitet på så kort tid. Vores måde at undgå konflikterne, var at lave fire branches, ud over Main, én til hvert gruppemedlem, samt én kaldet Development vi brugte til at flette vores arbejde sammen.

Som førhen nævnt er det gået op for os som gruppe hvor vigtigt der er at have en direkte kobling i forbindelse med stavning, lille og stor skrift når det gælder sql, da det i virkeligheden har været det som har gjort os tidspresset. Indirekte har det også givet os et indblik i hvor hjælpsomme diagrammer, modeller og et program som figma til mock-ups kan være. Blandt andet ville en gennemtænkt domæne-model i starten af projektet have gavnet os meget, da det til en vis grad ville hjælpe os med en overordnet klasse struktur.