Cupcake Projekt 2020 Forår

Gruppemedlemmer:

Anne Cathrine Høyer Christensen cph-ac221@cphbusiness.dk Github navn: ACHChristensen

Mathias Frank Parking cph-mp525@cphbusiness.dk, Github navn: mp525

Matti Ben Serup Hansen cph-mh829@cphbusiness.dk, Github navn: MattiBSH

Nikolaj Trankjær cph-nt105@cphbusiness.dk, Github navn: Sjuber

Vibeke Nordestgaard, cph-vn41@cphbusiness.dk, Github: LouYourLife

Lærernes navne: Nikolaj Brandt Hemmeshøj, Jon Bertelsen, Tue Hellstern

Cupcake Projekt 2020 Forår.

30. marts 2020

Indhold

Indledning

Olsker Cupcakes er et nyopstartet iværksætteri, som gerne vil implementere et IT-system, som kan hjælpe dem med at sælge deres produkt, cupcakes, med valgfri bund og topping. Kunden skal kunne oprette en bruger, bestille samt betale for diverse cupcakes og afhente ordren fysisk i butikken.

Baggrund

En virksomhed ved navn Olsker Cupcakes ønsker en hjemmeside, hvorfra de kan sælge deres cupcakes. Kravene, som er stillet, vises følgende:

Teknologiske valg

Til dette IT-system har vi valgt at bruge IntelliJ IDEA 2019.3.3, Digital Ocean (Linux server), Tomcat 9.0.31, jdbc, JSP, Filezilla, Adobe XD, Workbench 8.0, samt sprogene: MySQL, JavaScript, HTML, CSS og Java 8.

Krav

Virksomheden Olsker Cupcakes ønsker en hjemmeside, som deres kunder kan logge ind på eller registrere sig. Herefter vil det så være muligt for kunden at bestille ordrer på cupcakes, som så lægges ind på en liste, som de ansatte kan se fra administrator siden. De ansatte skal kunne slette ordrer, og kunder skal kunne slette deres egne ordrer, hvis de fortryder deres køb.

Funktionelle krav

Der er to slags brugere, nemlig kunde eller administrator, som har forskellige funktionelle krav:

1. Kunden skal kunne bestille top og bund til cupcakes og senere kunne hente ordren i butikken efter ordren er bestilt og betalt.
2. Kunden skal kunne oprette en konto/profil for at kunne betale og gemme en ordre.
3. Kunden skal kunne se vedkommendes valgte ordrer i en indkøbskurv, så vedkommende kan se den samlede pris.
4. Kunden skal kunne fjerne en ordre fra sin indkøbskurv, så kunden kan justere sin ordre
5. Administrator skal kunne indsætte beløb på en kundes konto direkte i MySql, så en kunde kan betale for sine ordrer.
6. Kunden eller administratoren skal kunne logge på systemet med e-mail og kodeord. Når brugeren er logget på, skal brugeren kunne se sin e-mail på hver side.
7. Administratoren skal kunne se alle ordrer i systemet, så vedkommende kan se, hvad der er blevet bestilt.
8. Administratoren skal kunne se alle kunder i systemet og deres ordrer, således at vedkommende kan følge op på ordrer og holde styr på sine kunder.
9. Administratoren skal kunne fjerne en ordre, så systemet ikke kommer til at indeholde ugyldige ordrer, f.eks. hvis kunden aldrig har betalt.

Ikke-funktionelle krav

1. Der laves en mockup i Adobe XD, som viser de websider, den færdige løsning kommer til at bestå af.
2. Ordrer, kunder og øvrige data skal gemmes i en database.
3. Databasen skal normaliseres på 3. normalform.

4. Kildekoden skal deles på GitHub.
5. Det færdige produkt skal udvikles i Java, MySql, HTML, CSS, Twitter Bootstrap og køre på en Tomcat webcontainer.
6. Det færdige produkt skal i sidste ende køre på en Droplet hos Digital Ocean
7. Løsningen skal udvikles med udgangspunkt i "Kaspers Command-Pattern skabelon"

Domænemodel og ER-diagram

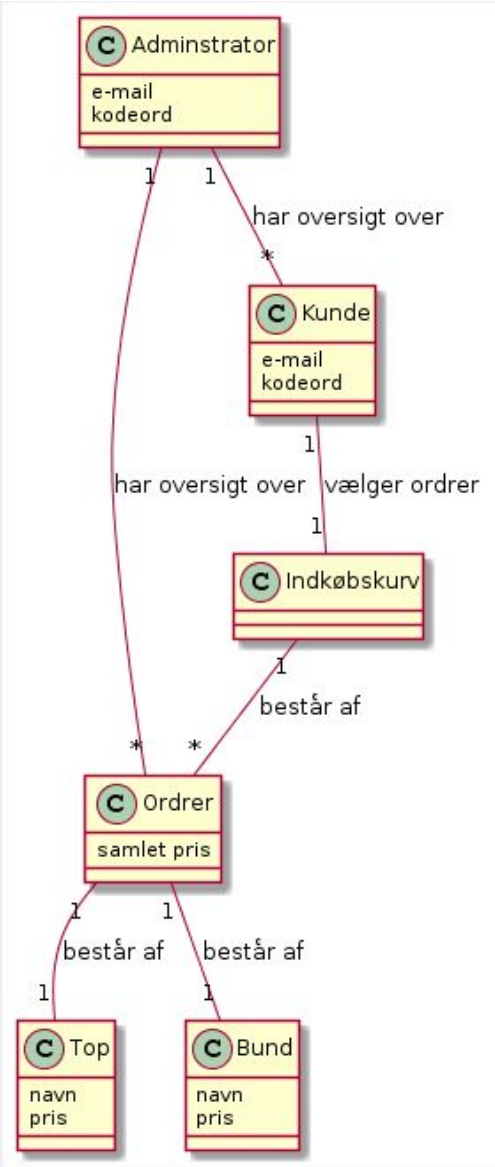
Domænemodel

Som kerneelementer i domænet har vi enten administrator eller kunde. Vi har en indkøbskurv, som består af en ordre, der igen består af en top og bund.

En administrator kan tilgå kunderne og se, hvilke ordrer de har, og kan slette disse ordrer, hvis f.eks. kunden ikke har betalt.

Kunden har adgang til en indkøbskurv, som er en oversigt over bestilte ordrer. Kunden kan også slette en ordre fra indkøbskurven.

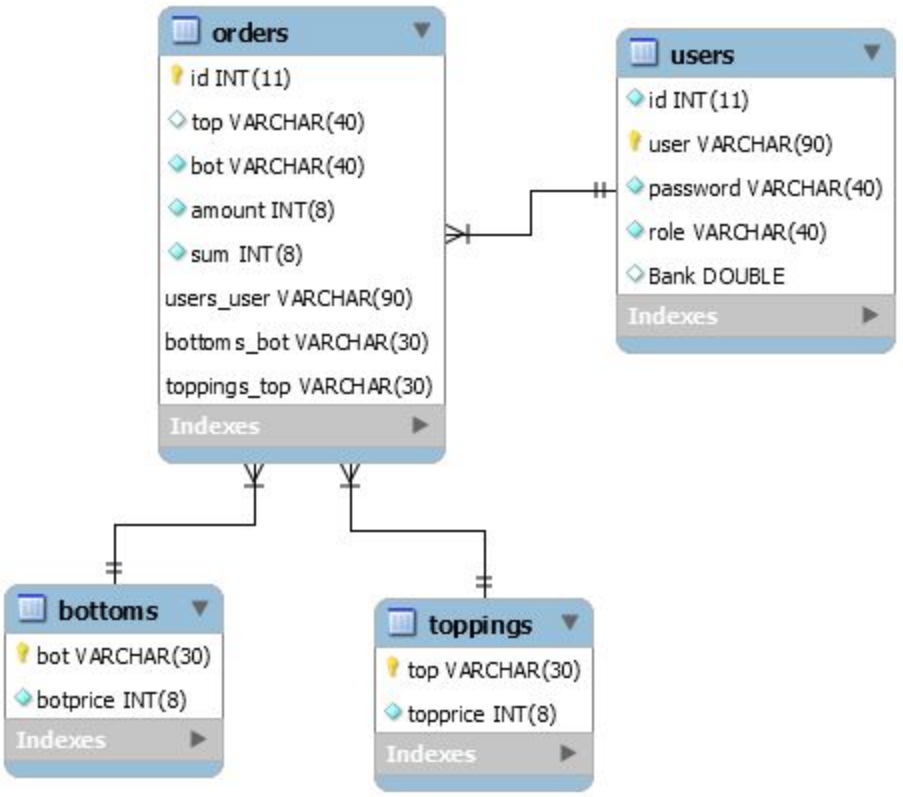
Ordre har en samlet pris, der bliver udgjort af både den valgte tops og bunds pris.



Toppene har, ud over pris, også et navn, og det samme gælder også for bundene.

ER-diagram:

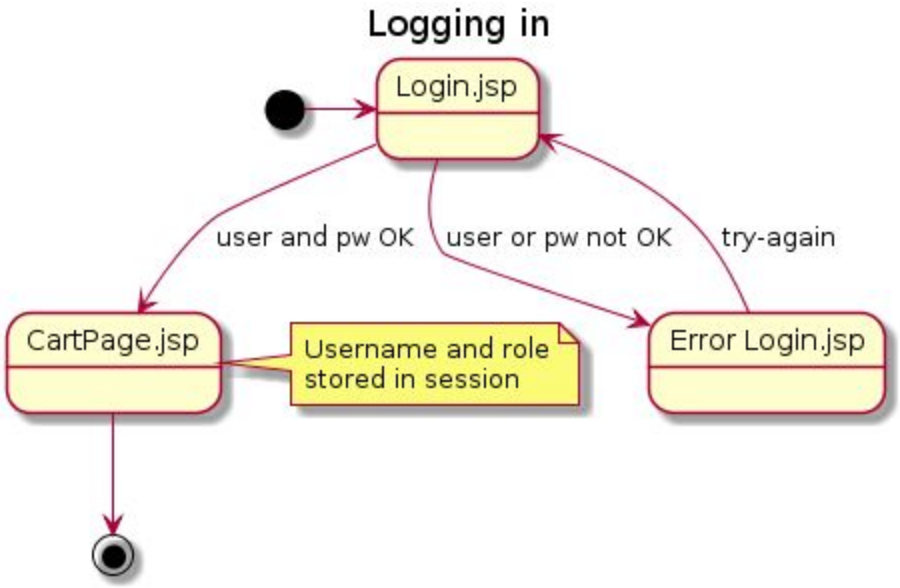
ER-diagrammet er en beskrivelse af databasestrukturen.



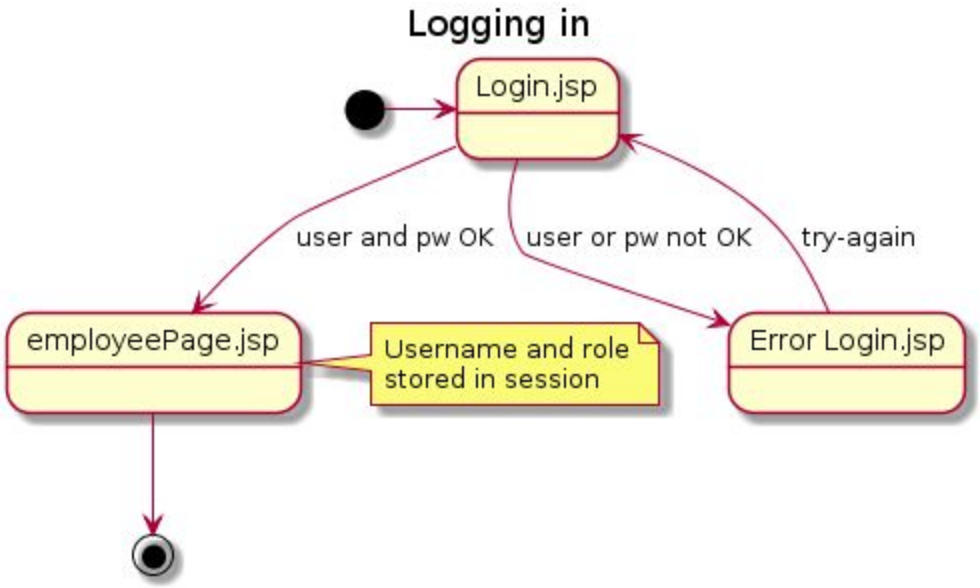
Vores database består af fire tabeller. Vi har en tabel, som opbevarer information om de brugere, der bliver lavet i systemet, kaldet users. Vi har vores ordretabel, som indeholder alle ordrer. Og så har vi to tabeller med vores toppings og vores bunde, vi fik givet af kunden. Dette diagrams tilstand forklares nærmere i afsnittet om status på implementationen.

Tilstandsdiagrammer

Tilstandsdiagrammer er diagrammer, der viser, hvordan man kommer fra den ene side til den næste.



I dette tilstandsdiagram, som viser kundens mulige interaktioner på login-siden, viser vi, hvor man bliver ført hen, når man står ved login-siden og logger ind. Vi viser, at man derfra kommer to steder hen. Den første mulighed er, at man kommer til CartPage.jsp siden med ens user gemt i sessionen. Dette sker, hvis man har tastet en korrekt e-mail og adgangskode ind. Hvis man ikke har det, bliver man sendt tilbage til login-siden, hvor den nu er opdateret med en fejlbesked.

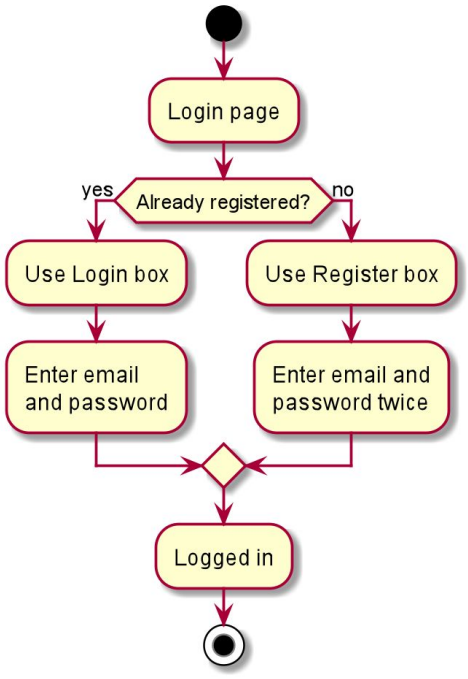


I dette tilstandsdiagram viser vi de forskellige bevægelser, man kan tage fra login-siden som administrator. Hvis man taster en e-mail og adgangskode, som er forbundet til en user med rollen

administrator, så bliver man sendt til employeepage.jsp, og ens user bliver gemt i sessionen. Hvis man ikke taster en gyldig e-mail og adgangskode, så kommer man til en opdateret version af login-siden, hvor der er en fejlbesked.

Aktivitetsdiagrammer

Aktivitetsdiagrammet for login-siden viser de to muligheder kunder har, når de vil ind i systemet. Er kunden allerede registreret i systemet, skal de indtaste deres oplysninger i boksen med login skrevet over sig. Er kunden derimod ny til firmaet og hjemmesiden, hvilket betyder de ikke allerede er registreret i systemet, vil de blive bedt om at indtaste deres e-mail og et password for at blive registreret og kunne give deres bestilling. I begge tilfælde vil kunden ende op med at være logget ind og kan afgive deres bestilling.



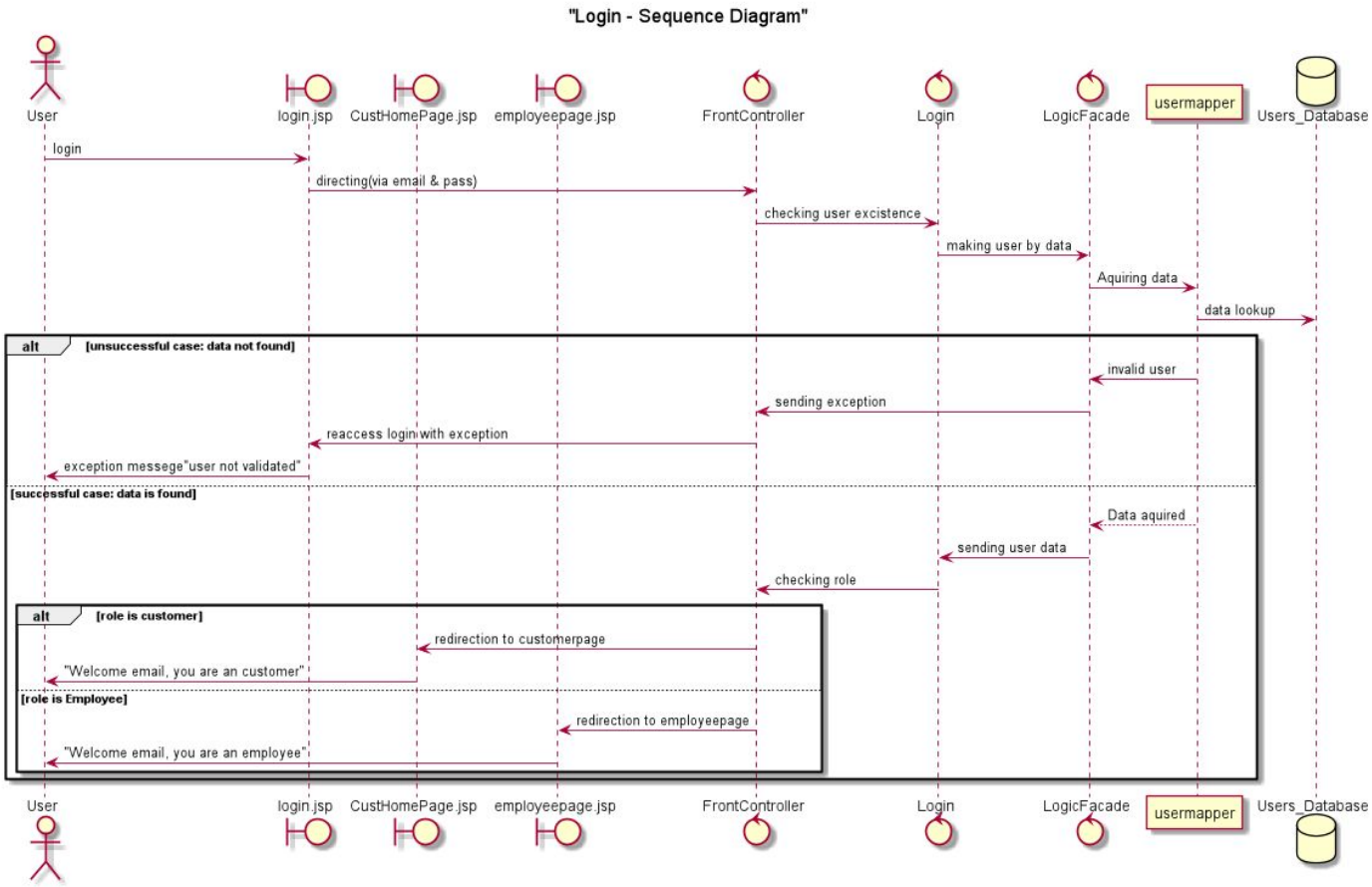
Det andet aktivitetsdiagram viser hvordan kunden går igennem en bestilling på hjemmesiden. Kunden har to valg. De kan logge ind først, hvilket tager dem over til indkøbskurven, hvor de kan tilføje ordrer til kurven og betale til sidst. Alle ordrer kan ses nederst på siden.

Kunden kan også vælge at lave en ordre først, hvorefter de vil blive taget til login-siden og skal logge ind eller registrere sig i systemet. Derefter kommer de til indkøbskurven og kan tilføje flere ordrer skulle de ønske det.

Sidst i diagrammet står der “pay and logout”. Det er afslutningen på bestillingen, og en fiktionel funktion vi ikke fik implementeret. Det er muligt at logge ud, men ikke muligt at betale fysisk med en kreditkort, da det ikke var en funktion, der var nødvendig i programmet.

Sekvensdiagrammer

Sekvensdiagrammer viser de handlinger, der foregår inde i programmet, når en kunde aktiverer en knap på siden. I dette tilfælde er det en kunde, der logger ind:



Når en kunde har klikket på en knap og har indtastet information på siden, så henvender login sig til 'FrontController' java koden, som fungerer som en kontrollør. Den kører nogle kommandoer, som tjekker, om kunden er i databasen, og derefter gør den én af to muligheder.

case 1: Data er ikke fundet.

Den sender kunden tilbage til login-siden med en error message.

case 2: Data er fundet.

Programmet tjekker ens rolle og sender én videre til den korresponderende side, CustomerPage.jsp, hvis useren er kunde, og employeepage.jsp, hvis useren er administrator.

Interessentanalyse

Projektet er ejet af Olsker Cupcakes og udført af gruppe 240 fra Copenhagen Business Academy Datamatiker hold E. De eksterne interessenter er dem, der ikke køber fra Olsker Cupcakes, fordi de evt. ikke er interesseret i færdiglavede cupcakes. De har derfor lille indflydelse og er ikke påvirket. Gidsler af projektet er Olsker Cupcakes ansatte. De bliver påvirket, eftersom de har mulighed for at kunne udføre ordrerne og strukturere derefter. De ansatte har også en administratorrolle, men rollen som ansat i sig selv har kun en lille indflydelse på projektet. Grå eminence består af administrator rollen, som administrerer ordrer og indsætter, hvor mange penge kunderne skal have til rådighed. Kunderne har i projektet mulighed for at bestille deres ordrer før afhentning, så de kan afhente deres ordrer hurtigere, tilgå ordrer, slette uønsket ordrer fra indkøbskurven samt at oprette sig som brugere. De har derfor stor indflydelse og bliver i høj grad påvirket, så kunderne af Olsker Cupcakes er ressourcepersonerne.

Særlige forhold

I vores program bruger vi HTTP request, response og session objekter. Request objektet bruger vi primært til at gemme indtastede data fra brugeren fra en form, som bliver udfyldt på en af .jsp siderne af brugeren, og til at navigere .jsp siderne. Session objektet gemmer vi også informationer på, som f.eks brugerspecifikke data. Efter at have logget ind, gemmer session objektet brugerens e-mail, password og rolle, og laver et “user” objekt. Hver gang brugeren navigerer et nyt sted hen, bliver der tjekket, om der er et session objekt, så brugeren bliver vist den passende .jsp side.

‘FrontController’-klassen er vores eneste Servlet og bliver brugt hovedsageligt til navigering af websitet. Når ‘FrontController’en nås, instantierer den tre ArrayLists: én af alle systemets kunder, én af alle toppings og én af alle bundene. Alle disse data bliver hentet fra databasen, og man kan godt mærke når man kommer til ‘FrontController’en rent performancemæssigt, idet at det tager ekstra lang tid at loade siden. Vi gør dette for at opdatere listerne regelmæssigt, f.eks når en administrator sletter en ordre, så bliver listerne opdateret, inden siden man stod på vises igen, så man kan få et opdateret overblik over ordrerne. Det samme gælder for kunden, når han/hun fjerner en ordre eller laver en ny. Når kunden sletter en ordre, tester programmet også, om ordren tilhører kunden, der er logget ind, dvs. den tjekker om ordre ejeren også er session ejeren.

Den måde vi har håndteret exceptions på, er ved at fange dem med enten en “try-catch” metode, eller også “kaster” vi dem fra command-klasserne. F.eks fanger vi de fleste exceptions i ‘FrontController’-klassen, hvor fejlbeskeden så bliver printet på login.jsp siden. Brugerinput har vi valgt at validere ved at definere i de forme, som brugeren udfylder, hvilke data, der skal indtastes, f.eks har vi skrevet i html (login.jsp) :

```
<input type="email" name="email" class="form-control" value="email">
```

og ved at skrive “type=email” kan man sikre, at det, der bliver tastet, har et standard e-mail format. Vi har også sikret, at når man taster antal cupcakes, kan man ikke taste et negativt tal eller lade feltet være tomt. Alt dette bliver der tjekket for, inden ordren bliver lavet, så den netop ikke bliver lavet i tilfælde med et fejlinput.

Sikkerheden i forbindelse med login, består af en hjemmelavet exception, der “kastes” efter et tjek med databasen, om den kunde, der forsøger at logge ind, overhovedet findes. Den exception

er blevet kaldt “LoginSampleException”, som skriver beskeden “Could not validate user”, i tilfælde af et ikke-gyldigt login.

```
public static User login( String email, String password ) throws LoginSampleException {
    return UserMapper.login( email, password );
}
```

Så ovenstående metode, login(), kalder UserMapper.login(), som håndterer database-kommunikation i forbindelse med brugere. Denne metode ligger i LogicFacade klassen.

Status på implementation

Vi er nået langt med de fleste mål hvad angår implementationen af kravene og jsp siderne.Vi har dog nogle mangler vi har opdaget hen mod slutningen af processen, som ikke er nået at blive ordnet.

Opgaven er beskrevet, sådan at databasen skulle være på 3. normalform. Dette har vi alle haft et indtryk af var gjort, men da vi kiggede tilbage på databaseopsætningen, kunne vi se, at det ikke var tilfældet. Det har også noget at gøre med den fortolkning af opgaven vi havde, da ingen af os havde set, at et forslag på en opsætningen blev forklaret og “lavet” for os i videoform. Dette er også grunden til, at ER-diagrammet har den form det har, men systemet virker, som vi har sat det op alligevel. Vi havde ikke tænkt over, at en ordre kunne indeholde mere end én slags cupcake i et vist antal. Vi har lavet det sådan, at en ordre indeholder en topping, bund, antal, brugernavn og en samlet pris. Vi kunne have gjort det at lave cupcake objekter og kommet dem i ordren i stedet. Og det, at vi ikke har gjort det, har måske resulteret i, at vores databaseopbygning er blevet for simpel.

Vi har også opbygget det sådan, at alt der kommer i "indkøbskurven", bliver sendt ned i databasen som forskellige ordrer. Vi har senere fundet ud af en bedre måde nemlig at have en midlertidig array liste med cupcake objekter, som kunden bestemmer opbygningen af og hvor mange, der skal bestilles. Når kunden så var tilfreds, kunne de data sendes til databasen.
En anden ting, vi aldrig nåede at få implementeret, var betaling af en ordre. Som administrator kan man give en kunde en sum penge på kontoen, men vi har ikke nogen funktion til at bruge de penge til noget.