Cupcake

Gruppe 3 hold B:

Nicolai Rosendahl - cph-nr135@cphbusiness.dk - git: MrJustMeDahl
Pelle Hald Vedsmand - cph-pv73@cphbusiness.dk - git: pelle112112
Carsten Juhl - cph-cj505@cphbusiness.dk - git: CarstenJuhl
Danyal Kitir - cph-dk174@cphbusiness.dk - git: DanyLoyal

Demo af applikation: https://youtu.be/Xzhirt19HpI
github: https://github.com/MrJustMeDahl/cupcake
28/3-2023

Indholdsfortegnelse

Indledning

Denne rapport præsenterer arbejdet udført af Gruppe 3, Hold B i forbindelse med udviklingen af en webapplikation for en cupcake butik. Projektets formål var at skabe en platform, hvor kunder kunne bestille og betale for cupcakes, samt hvor administratorerne kunne administrere ordrer og kunder. Rapporten beskriver vores projekt, forventninger, teknologivalg, krav og modeller, samt den overordnede proces og vores erfaringer gennem projektets forløb.
Gennem projektet har vi arbejdet med forskellige teknologier og værktøjer, og vi har lært at navigere mellem dem for at skabe en fungerende løsning. Vi har haft fokus på funktionalitet, og vi har taget højde for både kundens og administratorens perspektiver. Rapporten reflekterer vores succeser og udfordringer, samt de områder, hvor vi kunne have forbedret vores arbejde.
Vi håber, at denne rapport vil give et indblik i vores arbejdsproces og de valg, vi har truffet undervejs i projektet.

Teknologivalg

- Intellij v2021.3
    - Næsten alle benyttede os af 2021.3, men der var en af os som kørte 2023 versionen, som også virker.
- Mysql connector v8.0.30 (JDBC)
    - Databasen er sat op i Mysql. Queries er kørt i workbench før programstart.
    - Workbench vi har benyttet er v8.0.
- JSTL v1.2
- Maven project
- Java
    - Vi har kørt forskellige JDKs
- PlantUML v8059
- Javax servlets v4.0.1
- JUNIT v5.8.2
    - Vi har ikke benyttet os af Junit siden starten af projektet.
- Tomcat (local) v9.073

Krav

Funktionelle krav:

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.
2. Som kunde kan jeg oprette en konto/profil for at kunne betale og gemme en ordre.
3. Som administrator kan jeg indsætte beløb på en kundes konto direkte i MySql, så en kunde kan betale for sine ordrer.
4. Som kunde kan jeg se mine valgte ordrelinjer i en indkøbskurv, så jeg kan se den samlede pris.
5. Som kunde eller administrator kan jeg logge på systemet med email og kodeord. Når jeg er logget på, skal jeg kunne se min email på hver side (evt. i topmenuen, som vist på mockup'en).
6. Som administrator kan jeg se alle ordrer i systemet, så jeg kan se hvad der er blevet bestilt.
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.
8. Som kunde kan jeg fjerne en ordre fra min indkøbskurv, så jeg kan justere min ordre.
9. Som administrator kan jeg 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 Figma eller lignende, 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 med mindre andet giver bedre mening.
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. Løsningen skal udvikles med udgangspunkt i vores startkode.

Modeller

Use-case diagram:

Use-case diagrammet, som vist i bilag 1, giver et overblik over de sider man kan besøge på applikationen, samt en kort beskrivelse af hvilken funktion siden har.
Applikationen er lavet med henblik på at give brugeren en simpel og intuitiv måde at navigere gennem websitet på. Vi navigerer hovedsageligt til de forskellige sider via links indsat i navigationsbaren.
Det var vigtigt for os at give administratoren et arbejdssted som ikke var alt for blandet sammen med kundens oplevelse, derfor har administratoren fået en side, som det kun er adminen der har adgang til. Derigennem er det givet at adminen har fuld adgang til hele applikationen, mens brugeren har begrænset adgang.
Diagrammet er udarbejdet således at vi med den viste opsætning kan løse alle de funktionelle krav der er til projektet.

Flowdiagram:

I vores flow diagram (bilag 2), giver vi et simpelt og enkelt overblik over brugerens vej gennem websiden, fra login til logout, og alt der imellem. Hvilke jsp sider der dirigeres gennem hvilke servlets, samt hvilke servlets der forwarder deres data til andre servlets. Dette kan give en bedre forståelse for tilhørsforhold ved en evt overlevering.

Klassediagram:

Model-delen af applikationen er bygget op omkring de entitetsklasser som er vist i bilag 3, samt de persistence klasser som er vist i bilag 4.
Her er det værd at bide mærke i at der er en tilhørende facade-klasse til hver persistence-klasse, som for overskuelighedens skyld ikke fremgår af klassediagrammet, da metodesignaturene for disse klasser stemmer 1-til-1 overens med de metoder som fremgår af Mapper-klasserne. Vi har valgt at inkludere mapperen frem for facaden, da det er disse klasser som holder al funktionaliteten.
Derudover har vi valgt at lade ShoppingBasket nedarve fra Order, da en indkøbskurv i vores system bare er en ordre, som ikke er blevet afgivet endnu og hvor kunden stadig har mulighed for at tilføje til eller fjerne fra.

EER-diagram:

Databasen er opsat som vist i bilag 5.
Alle vores tabeller er lavet med henblik på at overholde 3. normalform.
Vi har benyttet foreign keys i vores order og cupcake tabeller til at sikre at tilstanden i vores database altid er valid. Altså at vi ikke har ordre som ikke tilhører en bruger eller cupcakes, som ikke ligger på en ordre eller ikke består af både en top og en bund.
Vores cupcakeview viser et join mellem cupcaketabellen, cupcaketoppingtabellen og cupcakebasetabellen, som vi bruger når vi skal instantiere vores cupcake elementer i Java.

Særlige forhold

- Efter en bruger er logget ind, bliver bruger-objektet gemt på session-scopet. Dette gøres for at vi på tværs af siderne altid kan fremvise relevant brugerinformation, og så vi i vores servlets altid har mulighed for at foretage ændringer på brugeren, uden at skulle tage hensyn til at finde frem til hvem der er logget ind. Brugeren bliver via vores servlets opdateret på session-scopet hver gang man går gennem en servlet, som har brug for at vise detaljer om brugeren, som hurtigt ændres, f.eks. hvis der er føjet noget til brugerens indkøbskurv.
- Som admin, har man fuld adgang til applikationen. Benytter adminen sig administrationsiden har vedkommende mulighed for at filtrere ordrerne efter brugere. Gøres dette gemmes bruger-id'et for den valgte bruger på session-scopet, sådan at vi på siden, kun viser data der tilhører den valgte bruger. Bruger-id'et på session-scopet bliver sat hver gang man filtrere eller nulstillet, hvis man vælger "alle brugere".
- Sessionen bliver invalideret når en bruger logger ud.
- I java er vores shoppingbasket en ordre, som ikke er blevet afgivet endnu. Vi har derfor via vores servlets sikret at hver bruger kun kan oprette en ny ordre, hvis de ikke allerede har en aktiv shoppingbasket, sådan at kunder ikke kan have mere end en indkøbskurv af gangen.
- Alle persistenceklasser gemmer data i vores database, alle metoder i disse klasser har derfor mulighed for at kaste en SQL-Exception. Dette håndteres ved at kaste en Database-Exception, som bliver kastet videre fra mapperen, til facaden og op til servletten. I servletten håndteres exceptionen ved at forwarde brugeren til error-siden, hvor fejlmeddelelsen bliver vist.

- Vores system har kun 2 roller som kan være tilknyttet brugeren - admin og user. Det er ikke muligt at ændre en brugers rolle og alle nye brugere bliver oprettet som user. Der er derfor kun mulighed for at have en admin bruger.

Status på implementation

Generelt er vi nået godt i mål med implementationen af koden. Alle JSP sider er oprettet og man navigerer korrekt igennem servlets uden at der opstår store fejl.
User-stories er blevet løst, og vores hjemmeside kan løse kravene stillet af kunden.

I sidste øjeblik gik vi i gang med at oprette en side, hvor kunden ville have mulighed for at se tidligere ordrer. Dette så vi nødvendigt, da "Bestil" og "Betal" knapperne under shoppingcart.jsp fjerner alt indholdet i indkøbskurven og sender det videre til admin (Bageriet). Det kunne derfor godt være svært for kunderne at holde styr på deres ordrer, og om de er gået igennem.
I en mere virkelig situation ville kunden som regel have modtaget en email eller SMS, hvor ordren og ordrelinjerne ville have stået på. Dette kunne også løses med en "Din ordre er gået igennem" besked. Nu har vi dog løst problemet, ved at indføre en "ordre-side" for kunderne, så de kan holde styr på deres ordrer.

Der er stadig et par fejl, som vi ikke har haft tid til at rette, som opstår når man opretter en bruger. Her kan man godt oprette en bruger uden et password eller et brugernavn. Udover dette har man mulighed for at oprette flere brugere med samme mail, som en allerede eksisterende bruger.

Det generelle design er heller ikke det kønneste, men vi synes det er godkendt ift tidsplanen, da funktionaliteten skulle komme i første række. Designet er dog meget overskueligt, og det er nemt at navigere hjemmesidens funktioner.
Hvis man sammenligner hjemmesiden med vores figma modeller (Se bilag 6 og 7), så er vi nået godt i mål med det forventede design. Vi benytter os af de samme billeder, samt samme webside struktur. Vores navbar er dog en smule anderledes i design, men funktionaliteten er næsten det samme.

Udgangspunktet for projektet har været, at alle sider skulle være responsive og fungere på alle skærmstørrelser. Den eneste side dette ikke er lykkedes på er welcome.jsp, hvor nogle af funktionerne forsvinder ud af skærmen når man ser siden i mobilstørrelse.

Havde vi haft mere tid, er der nogle ting vi gerne ville have implementeret.

- At man på diverse ordrer (på alle sider) skulle kunne se antallet af cupcakes ud for hver linje. I stedet for at hver linje kun repræsenterer en cupcake.
- Gjort designet på alle siderne ensartet.
- Vist en samlet pris på kundens valg af cupcake og antal, før de lægger dem i kurven.
- Ændre så alt er på dansk - både siderne og den data der ligger i databasen, samt at danske tegn kan fremvises korrekt gennem hele applikationen.

Proces

Gennem vores projekt har vi været meget "large" med vores forventninger og arbejdsmål. Da vi skrev vores gruppekontrakt, satte vi nogle få rammer:
Vi planlagde at mødes hver dag på Discord kl. 10, for at samle op på arbejdet fra dagen før, samt planlægge arbejdet for næste dag. Her kunne vi også komme med input ift projektet, og om der var nogle elementer vi muligvis havde glemt. Dette fungerede umiddelbart fint, da alle selv havde mulighed for at distribuere deres egen arbejdstid i løbet af dagen, da mange af os har et arbejde udover vores studie.
Vi arbejdede primært hjemmefra, hvor vi har benyttet os af Discord til at komme med input og spørgsmål til de andre medlemmer af gruppen.
Det kunne til tider være svært at vide, hvad de forskellige personer i gruppen sad og arbejdede på, da vi kun holdte møde én gang om dagen.
Vi benyttede os af Github Projects til at holde øje med vores tidslinje.

Vores forventning for projektet var ikke defineret så præcist, hvilket nok skal ændres til fremtidige projekter. Projektlængden for cupcake projektet gjorde os usikre på, hvor meget kode vi ville være i stand til at skrive, hvilket også gjorde at vi ikke fik ordentligt forventningsafstemt.
Vi ville gerne have et velfungerende projekt, som løste alle de givne user-stories stillet til os af kunden. Her skulle funktionaliteten være i højsædet, og designet ville vi løse til sidst.

Derudover skulle alle prøve at arbejde med de forskellige værktøjer vi har lært i løbet af semesteret, så man i sidste ende er i stand til at stå inde for alle dele af koden.

Vi har arbejdet ud fra et princip, hvor vi har bygget tingene op fra bunden. Vi er altså startet med at få vores database på plads, og er derefter begyndt at lave backenden af projektet. Vi er nået frem til at denne fremgangsmåde både har fordele og ulemper.
Det fungerede f.eks. rigtigt fint at vi ikke hele tiden skulle skifte mellem at arbejde på forskellige dele af projektet, men derimod kunne fokusere på at få en funktion op at køre af gangen. Udover det skabte denne fremgangsmåde et godt fundament for at systemet blev abstrakt og objektorienteret, da vi i vores arbejde med modellen ikke vidste præcis hvad vi skulle bruge derfra, men kun havde en idé om hvad den skulle kunne.
Det fungerede derimod ikke så godt, at vi ind imellem fandt os selv i situationer hvor vi skulle sidde og gætte os frem til hvilken funktionalitet vi skulle bruge senere hen, og derved har fået lavet noget funktionalitet som ikke endte med at blive brugt. Eksempler på dette er at vi havde regnet med der skulle være en paymentMapper, som tog sig af overførsler. Udover dette giver vores cupcakeMappere mulighed for at tilføje nye smage, ændre priser eller fjerne smage fra menukortet, alt sammen funktioner som ikke endte med at blive implementeret i det endelige produkt.
Alt i alt har vi vurderet vores fremgangsmåde til at være fornuftig nok, men må nok også erkende at vi måske ikke har den fornødne erfaring til at få maksimalt udbytte ud af at arbejde på den facon.

I forhold til Figma modellerne, så skulle vi have brugt mere tid fra start på at lave figma modeller til samtlige sider. Figma modellerne er gode til at holde et overblik over hjemmeside strukturen, og det kan være vigtigt i større projektgrupper, hvor man altid vil have det samme udgangspunkt.
Design kan for mange personer være et abstrakt emne at begive sig ud i, og Figma kan derfor være en hjælpende hånd både strukturelt, men det kan også bruges som inspiration til designstilen.

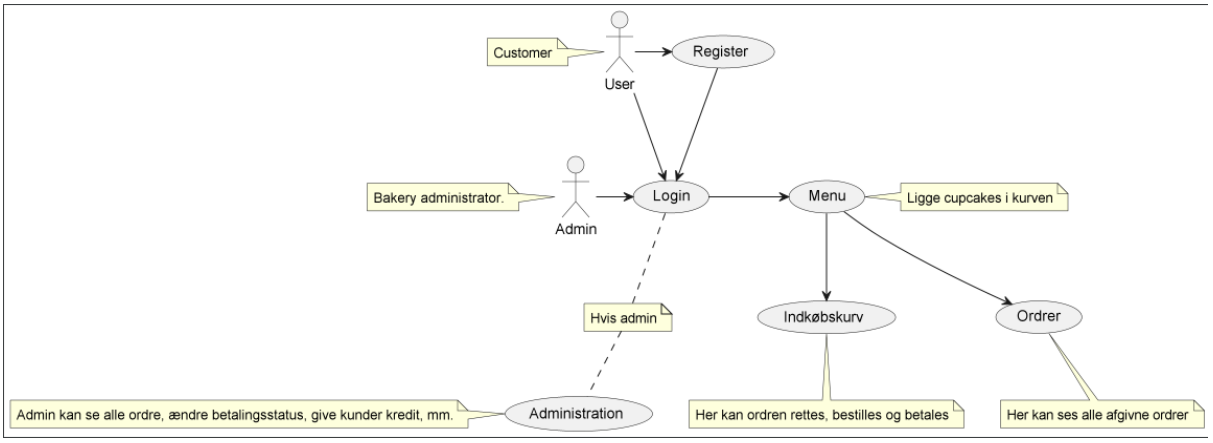
Forståelsen for hvordan servlets bruges har ændret sig fra projektets start til projektets afslutning, hvilket også afspejler sig i vores source kode. I begyndelsen skrev vi relativt komplekse servlets, hvor det i sidste ende blev svært at overskue hvad der egentlig sker. Skulle vi udføre projektet igen, ville vi fra starten strukturere vores servlets sådan at der er en

servlet der hører til hver .jsp-side, som tager sig af at hente hvad der skal bruges for at vise siden fra databasen og ville ellers lave meget specifikke servlets til at håndtere andre opgaver. På den måde kan man når der bliver sendt en request på applikationen blive directed til de servlets som udfører handlingerne, og til sidst blive directed til den servlet der forwarder dig til den korrekte side. Således undgår vi meget komplekse servlets, samt for meget genanvendt kode, som gør det svært at refaktorere vores arbejde. Udover det skal vi også have en bedre folder struktur, da det til sidst blev svært at finde rundt i vores servlets. De bør altså fordeles i relevante packages afhængigt af den funktion de udfører, dette princip gør sig gældende med alle dele af applikationen, da den ikke skulle have været meget større før vi havde mistet overblikket, med det setup vi har nu.

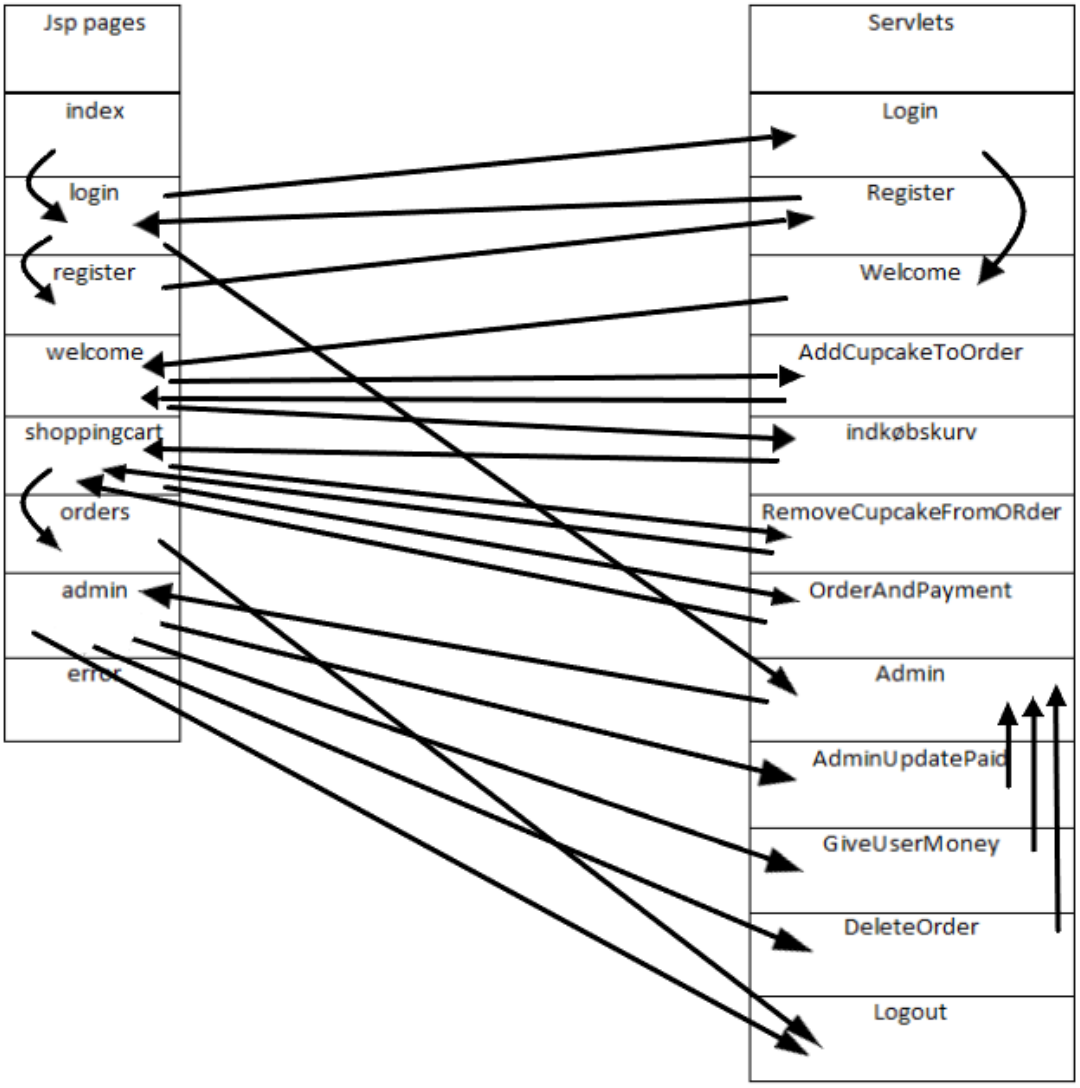
Overordnet set er vi tilfredse med vores produkt, men til næste gang skal vi være bedre til at forventningsafstemme og planlægge projektet.

Bilag:

Bilag 1.



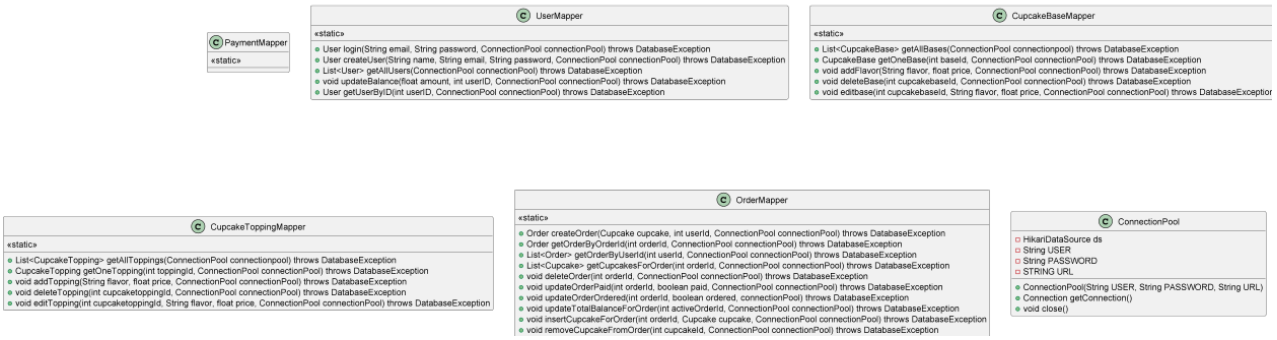
Bilag 2.


Jsp pages
index
login
register
welcome
shoppingcart
orders
admin
error
Servlets
Login
Register
Welcome
AddCupcakeToOrder
indkøbskurv
RemoveCupcakeFromORder
OrderAndPayment
Admin
AdminUpdatePaid
GiveUserMoney
DeleteOrder
Logout

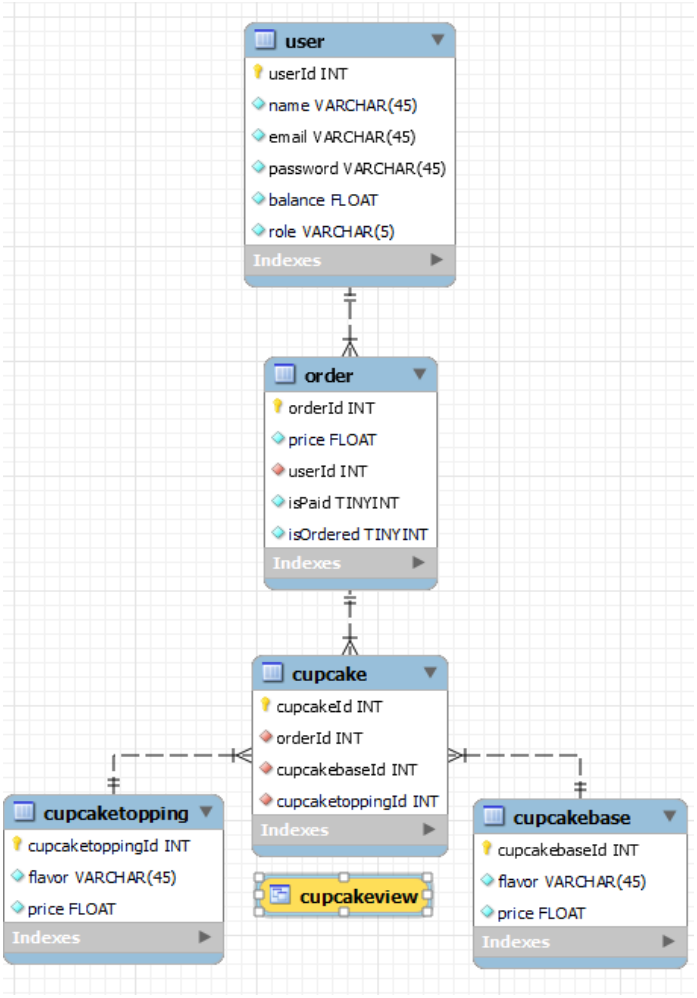
Bilag 3.


User
int userID
String name
String email
String password
Role role
float balance
List<Order> allOrders
User(int userID, String name, String email, String password, float balance, Role role, List<Order> allOrders)
User(String name, String email, String password)
int getUserID()
String getName()
String getEMail()
Role getRole()
float getBalance()
boolean setBalance()
List<Order> getAllOrders()
Order
int orderID
List<Cupcake> cupcakes
float totalPrice
boolean ordered
boolean paid
Order(int orderID, int userID, List<Cupcake> cupcakes, boolean ordered, boolean paid)
List<Cupcake> getCupcakes()
float getTotalPrice()
boolean getOrdered()
boolean getPaid()
ShoppingBasket
ShoppingBasket(Super())
boolean addCupcake()
boolean removeCupcake()
Cupcake
CupcakeTopping topping
CupcakeBase base
float fullPrice
int cupcakeId
Cupcake(CupcakeTopping topping, CupcakeBase base, int cupcakeId)
Cupcake(CupcakeTopping topping, CupcakeBase base)
float getFullPrice()
CupcakeTopping getCupcakeTopping()
CupcakeBase getCupcakeBase()
int getCupcakeId()
CupcakePart
String flavor
float price
CupcakePart(String flavor, float price)
float getPrice()
String getFlavor()
CupcakeTopping
int cupcaketoppingId
CupcakeTopping(int cupcaketoppingeId, String flavor, float price)
CupcakeBase
int cupcakebaseId
CupcakeBase(int cupcakebaseId, String flavor, float price)

Bilag 4.



Bilag 5.



Bilag 6: Figma model 1

Bilag 7: Figma model 2