CUPCAKE-SHOP

Mads mikael Keinicke – cph-mk731@cphbusiness.dk - https://github.com/stoxhorn/ - klasse B

Table of Contents

Indledning

For dig der studere på 2. semester, vil den her rapport kunne fortælle lidt om, hvordan man planlægger og hvad man skal bruge, til at lave en webservice med tilhørende hjemmeside. Mere specifikt vil der blive lavet en webshop. Dette vil gøres ved hjæ.lp af JSP, Java og SQL. Der vil ikke blive fortalt om hvordan man hoster dem I clouden, men kun lokalt.

Baggrund

Olsker Cupcakes fra bornholm har lavet en mockup til en webshop de vil lancere. Det er vores opgave at hjælpe med at tilføje manglende funktionaliteter og levere produktet.

De er kommet med nogle forskellige krav til hvad de som minimum forventer vi leverer. Her indgår det f.eks. At en bruger skal kunne lave, gemme og købe en ordre og at en administrator kan læse denne, og indsætte penge på en brugers konto, så de kan betale den.

Teknologi valg

IntiiliJ IDEA 2022.1 Ultimate – Build #IU-221.5080.210

Runtime version: 11.0.14.1+1-b2043.24 amd64

VM: OpenJDK 64.bit Server VM vt JetBrains s.r.o.

Java

Openjdk 11.0.14.1 2022-02-08

OpenJDK Runtime Environment (build 11.0.14.1+1-Ubuntu-0ubuntu1.20.04)

OpenJDK 64-Bit Server VM (build 11.0.14.1+1-Ubuntu-0ubuntu1.20.04, mixed mode, sharing)

MYSQL

mysql Ver 8.0.28 for Linux on x86_64 (MySQL Community Server - GPL)

TomCat

apache-tomcat-9.0.62

Krav

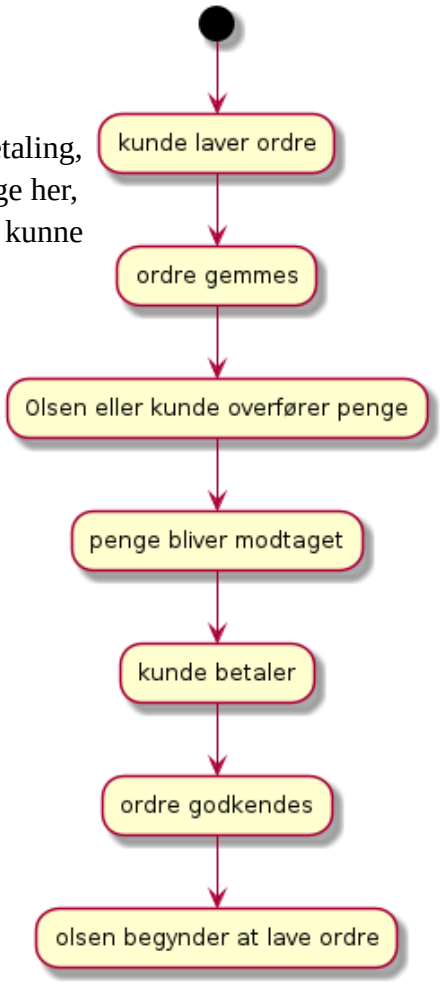
Vi er blevet givet 9 userstories som funktionelle krav.

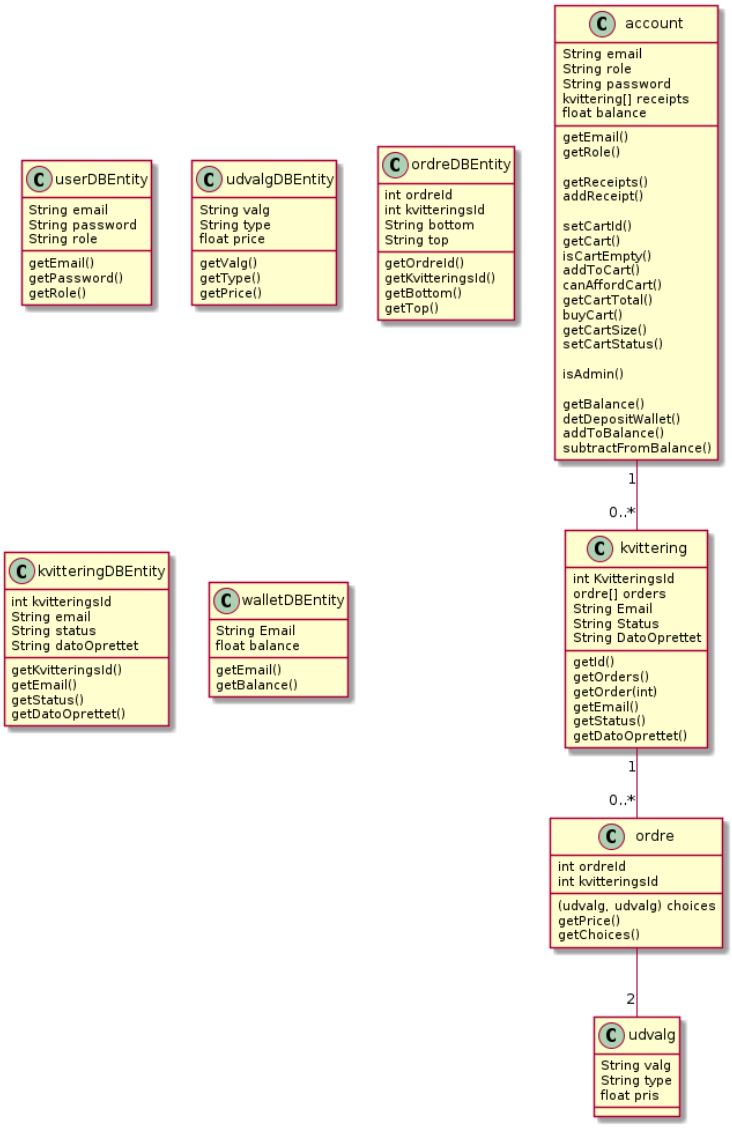
US-1	Som kunde kan jeg bestille og betale cupcakes med en valgfri bund og top, sådan at jeg senere kan køre forbi butikken i Olsker og hente min ordre.
US-2	Som kunde kan jeg oprette en konto/profil for at kunne betale og gemme en en ordre.
US-3	Som administrator kan jeg indsætte beløb på en kundes konto 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 se 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.

Af disse Userstories er de første 6 blevet implementeret.

Aktivitetsdiagram

Olsen har et simpelt workflow. Kunden laver en ordre. Så tager Olsen imod betaling, hvorved ordren godkendes, og Olsen starter med at lave deres ordre. Det vigtige her, er at notere en ordre først godkendes ved betaling. F.eks. Vil Olsen gerne selv kunne styre overførsler til kunderne, sådan at han kan styre når de kan betale eller ej.





Domæne model

Entiteter

Til højre ses de entiteter jeg bruger. Alle DBEntity klasserne, er en direkte kopi af tabeller I min database.

Account, Kvittering, Ordre og Udvalg er alle de objekter som min server arbejder med. DBEntity objekterne har kun constructors og getters, sådan at jeg skal lave et nyt objekt, hvis jeg skal tilføje til “walletDBEntity’s” balance, for eksempel.

På den måde kan jeg separere min læsning af databasen, fra håndteringen af den data jeg har læst mere flydende.

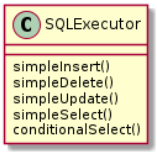
Og jeg kan samtidigt også skrive til databasen mere flydene. Det kan gøre linjerne lidt lange, men jeg kan nemmere scale op og tilføje til koden. Med andre ord giver det meget low coupling, mellem min kode og læsningen af databasen.

Det gør også at jeg kan lave objekter der slet ikke skal tænke på at værre nemme at skrive og læse fra databasen, men I stedet kan lave objekter der passer nemt og præcist til det jeg har brug for.

Services

Her er et diagram over de services jeg har. Convertersne er de static klasser jeg bruger til at konvertere mellem DBEntity objekter og de målrettede objekter.

Builder klasserne eksisterer fordi userDBEntity og kvitteringDBEntity ikke har ordre og udvalg objekter inde I sig, når de læses fra databasen. Så I stedet for at læse dem alle I en lang constructer, kan jeg sætte lidt nemmere op, og evt. Tilføje kvitteringer til Account løbende, inden objektet er færdiglavet.



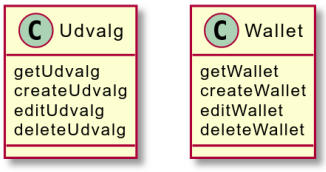
SQLExecutor er en static klasse der eksekvere generelle SQL queries eller statements, men den koordinere ikke for specifikke objekter eller kontrollere persistence, den hjælper kun med det.

Persistence

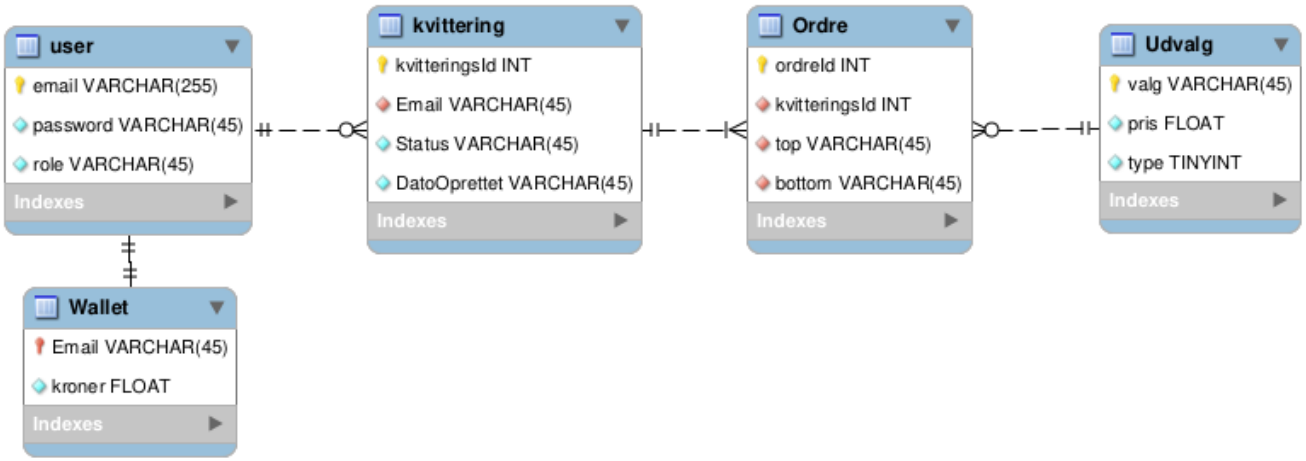
For persistence har jeg en mapper klasse til hver objekt. De sørge alle sammen for kalde SQLExecutor funktioner med try/catch, og har ekstra funktioner, som f.eks:

Overloadede funktioner til edit/update/delete funktioner, der kalder konverteringsServicesne, sådan at de kan kaldes med “Kvittering” objekter, og ikke kun kvitteringDBEntity objekter.

Funktioner der kan finde det højeste ID eller med specifik værdi I en kolonne.



EER diagram



1NF

1.Der skal være en nøgle, der entydigt identificerer den enkelte række i tabellen.

Dette er gældene for alle tabellerne.

2.De enkelte felter må kun indeholde en værdi(atomare værdier).

Dette er også gældene, dato er gemt som en hel string og ikke flere, f.eks.

3.Der må ikke være kolonner, der gentages.

Indenfor de enkelte tabeller er der ikke, men wallet kunne godt have ligget under user, og dermed gentager "email" sig, og optræder lige så mange gange, som den gør I user tabellen. Men der referreres her til gentagelser I samme tabel.

2NF

1.Den opfylder alle kravene til første normalform.

2.Ingen attributter/ egenskaber, der ikke selv tilhører nøglen, må afhænge af en del af nøglen.
I alle tabellerne er de forskellige værdier kun bundet til nøglen som en identifikation, og er ikke afhængige af nøglen. Det er systemet der er afhængige af at værdien sidder sammen med den rigtige nøgle. En kvittering ser end ud, lige meget hvor meget ID'et ændrer sig.

3NF

1.Den opfylder alle kravene til anden normalform.

2.Ingen attributter/egenskaber må afhænge af andre attributter, der ikke er selv er nøgler.
I wallet er dents ID afhængig af email I User, som er en nøgle. Det samme for email I kvittering.

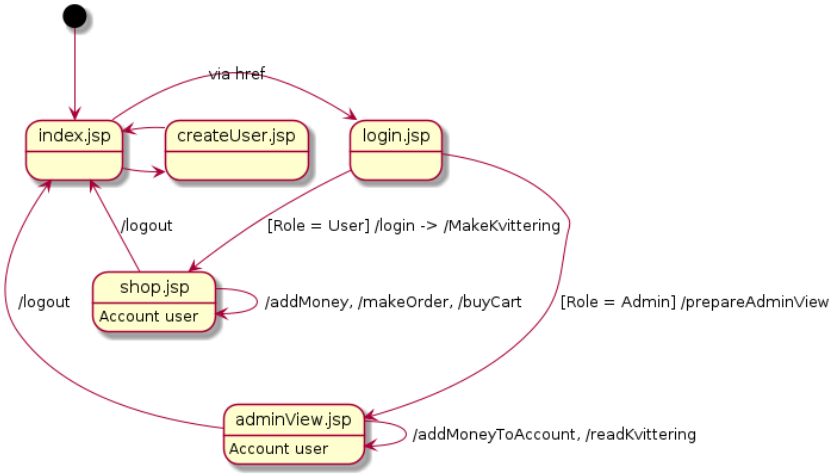
“kvitteringsId” I Ordre, er afhængig af nøglen I Kvitterings tabellen.

“top” og “bottom” I Ordre er afhængig af “valg” nøglen I udvalg.

Navigationsdiagram

NavigationsDiagrammet er meget simpelt.

Den er bygget med en side til hver service. Siden vil så løbende opdateres med hvad brugeren nu er I gang med.



Index.jsp er hovedsiden der byder en velkommen. Her kan man vælge at lave en user, eller logge ind.

Login.jsp nås uden en servlet. Login.jsp henfører til /login servletten, som sender brugeren videre til /makeKvittering servletten.

Den opretter en indkøbsvogn, som kan holde ordre I sig, og fører videre til shop.jsp.

Shop.jsp er så den eneste side lavet til brugeren. Siden opdateres som brugeren tilføjer varer.

Betaling og tilføjelse af penge sker på samme side, og fører tilbage til shop.jsp.

/addMoney, /makeOrder, /buyCart servletsne fører alle til shop.jsp, og er ikke beregnet til at blive kaldt fra andre steder. Skulle man lande på shop.jsp uden at være logget ind, vil siden kun have et link til index.jsp

adminView er så den eneste side lavet til admin. Den nås hvis ens rolle er “admin” I databasen når man logger ind. Login.jsp vil så kalde servletten /prepareAdminView, som fører til adminView.jsp.

AdminView kan tilføje penge til en bruger eller læse en kvittering, via servletsne /addMoneyToAccount og /readKvittering.

Øverst I banneret kan man logge ud fra både adminView.jsp og shop.jsp, og man vil lande på index.jsp.

Særlige forhold

Account objektet bliver brugt til at håndtere og gemme data. JSP sider vil hendte al data fra dette objekt, og sende data videre I request scope. Account objektet gemmes som “user” I sessionScope.

Der er ikke meget sikkerhed I forhold til login. Til gengæld vil JSP sider kun vise deres indhold, hvis user objektet I sessionScop har rollen “admin”.

Der opstod noget uheldig navngivning. Det man normalt kalder en ordre, er I virkeligheden en “vare”. I det her projekt vil det svare til en enkelt cupcake kombination. Der er blevet blandet uheldigt rundt på dansk og engelsk. Så nogen gange er det receipt, og andre gange kvittering. Receipt foregår mest I web-laget, og ting der relaterer til web-laget, så som account-klassen. Kvittering foregår mest I bussiness laget.

Samtidig kan man altid tilføje nye kombinationer. Det kræver så en manuel SQL query, da det ikke er programmeret muligheden nogen steder. Men, en ordre kan kun laves hvis kombinationsdelene eksistere som en række I “udvalg” tabellen, men så snart den er der, så kan den fungere frit I resten af systemet uden problemer. Så man kan altid komme på nye opskrifter.

Status på implementation

Alt der fremgår I navigationsdiagrammet er nået. De første 6 usecases er nået, og grundet tidspres var der ikke ideen at nå mere.

Der er ikke blevet stylet noget. Kun “template” billedet fra CPHBusiness er ændret. Der var ikke tid til mere.

Mange servlets kan ryddes mere op, ved at lave service klasser for disse. Men grundet manglene erfaring, var jeg usikker på præcis hvordan servletsne skulle struktureres for navigationsdiagrammet.

F.eks. Er der noget roddet kode for at finde det rigtige ID til den næste ordre eller kvittering, som sagtens kan indgå I en service, eller som I en del af mappersne.

Da jeg lavede alle mapper klasserne fandt jeg også ud af der var en del jeg kunne generalisere. Det gav anledning til SQLExtractor. Man kunne nok slå dem alle sammen til en mapperKlasse, men det er også rart at kunne få nogle mere målrettede funktioner. Plus, jo mere genereliseret man gør det, jo større er faren for et SQL injection. Jeg er usikker på hvor stor faren er I min kode.

Proces

Arbejdsformen var ved at planlægge I trello, og dele det op I mindre bider selvfølgelig. Det vigtigste var først at få bygget persistence laget. Hvis man har nogle roddede entiteter og klasser, så bliver resten af projektet altid noget møj.

Det var derfor jeg lavede seperate SQL-entitets-klasser og klasser jeg rent faktisk havde tænkt mig at bruge. Det var godt nok en masse ekstra arbejde, men det gjorde at jeg også I koden kunne nemt læse hvornår jeg roddede med objekter der hørte til web-laget, eller hørte til persistence. De og konvteringsklasserne gjorde linjerne lidt længere og koden kunne se lidt roddet ud. Men med et bare linjeskift ser det hele finere ud, og det skabte et simpelt flow, der blev gentaget ofte, og dermed skabte et nemt forståeligt mønster. De tog dog ret lang tid at lave, selvom de var simple.

Til næste gang, skal der laves sekvensdiagrammer af servlets. Jeg havde ikke så super meget erfaring med dem, da jeg startede på det, men sekvensdiagrammer kunne have gjort dem så meget nememere at overskue.