Cupcake

Deltagere:

Navn	E-mail	Github	Klasse
Oliver Juul Reder	cph-or42@cphbusiness.dk	Kvark96	B
Peter Andersen	cph-pt124@cphbusiness.dk	atombomben	B
Hans-Henrik Madsen	cph-hm163@cphbusiness.dk	hans-henrik	B
Sebastian Hejlesen	cph-sh528@cphbusiness.dk	HejlesenCPH	B

Dato:
28.04.21

Link til GitHub repository:

https://github.com/Kvark96/vigilant-umbrella

Link til hjemmeside:

http://64.227.117.30:8080/sem2-startcode-1.0-SNAPSHOT/

Indledning 3

Baggrund 3

Teknologivalg 3

Krav 4

Domæne model 5

EER-diagram 6

Aktivitetsdiagram 7

Navigationsdiagram 8

Særlige forhold 9

Status på implementation 9

Proces 10

Indledning

Vi har igennem et skoleprojekt hos Copenhagen business academy skulle udvikle en bestillings hjemmeside. Hjemmesiden der gør brug af en integration med en SQL-database, der giver produktet store ændringsmuligheder på forskellige genstande, uden slut-brugeren skal have java programmerings færdigheder.

Baggrund

Hjemmesiden er for en fiktiv cupcake forretning, hvorpå brugere blandt andet skal kunne oprette konti, bestille cupcakes til afhentning, og betale.

Samtidig skal det for administratorer af hjemmesiden være muligt at se alle ordrer, og hvem der har bestilt disse, samt alle brugere i systemet. Systemet skal køres med TomCat, og desuden implementere en SQL-database til persistens.

Disse krav er specificeret i krav-sektionen.

Teknologivalg

- IntelliJ IDEA Ultimate v2021.1
- SQL Server v8.0
- Tomcat v9.0.45.1
- JDBC v8.0.19
- Bootstrap v4.3.1
- Corretto v1.8.0_292

Krav

Cupcake-forretningen har længe ønsket sig en hjemmeside der gør det lettere for brugerne at bestille deres varer. Det vil desuden være lettere for dem at administrere, i forhold til hvis folk hele tiden ringer til dem.

Vi er derfor kommet frem til følgende use-cases:

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 ordrelinjer i en indkøbskurv, så jeg kan se den samlede pris.

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

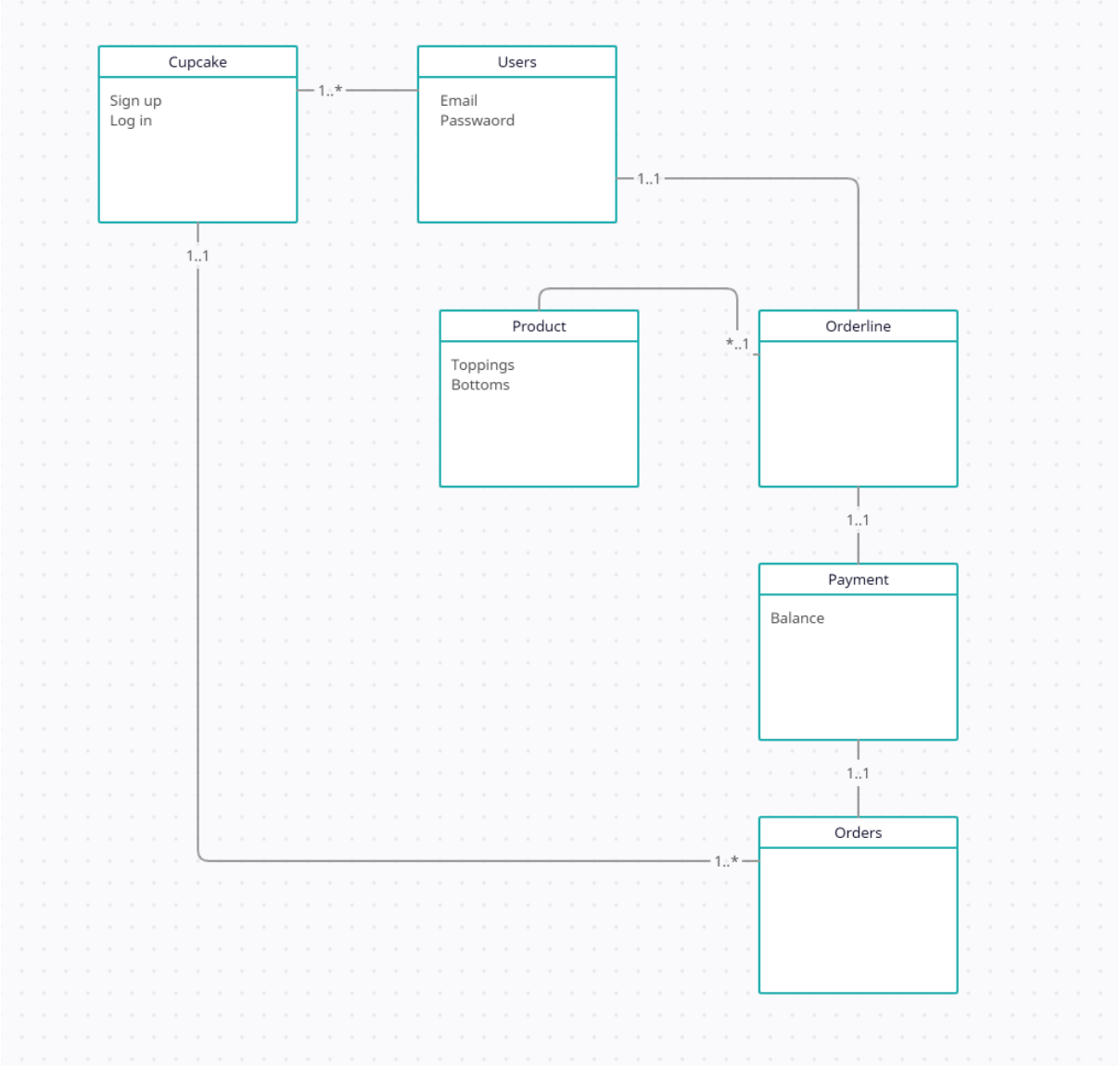
US-6: Som administrator kan jeg se alle ordrer i systemet, så jeg kan se hvad der er blevet bestilt.

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 ugyldige ordrer. F.eks. hvis kunden aldrig har betalt.

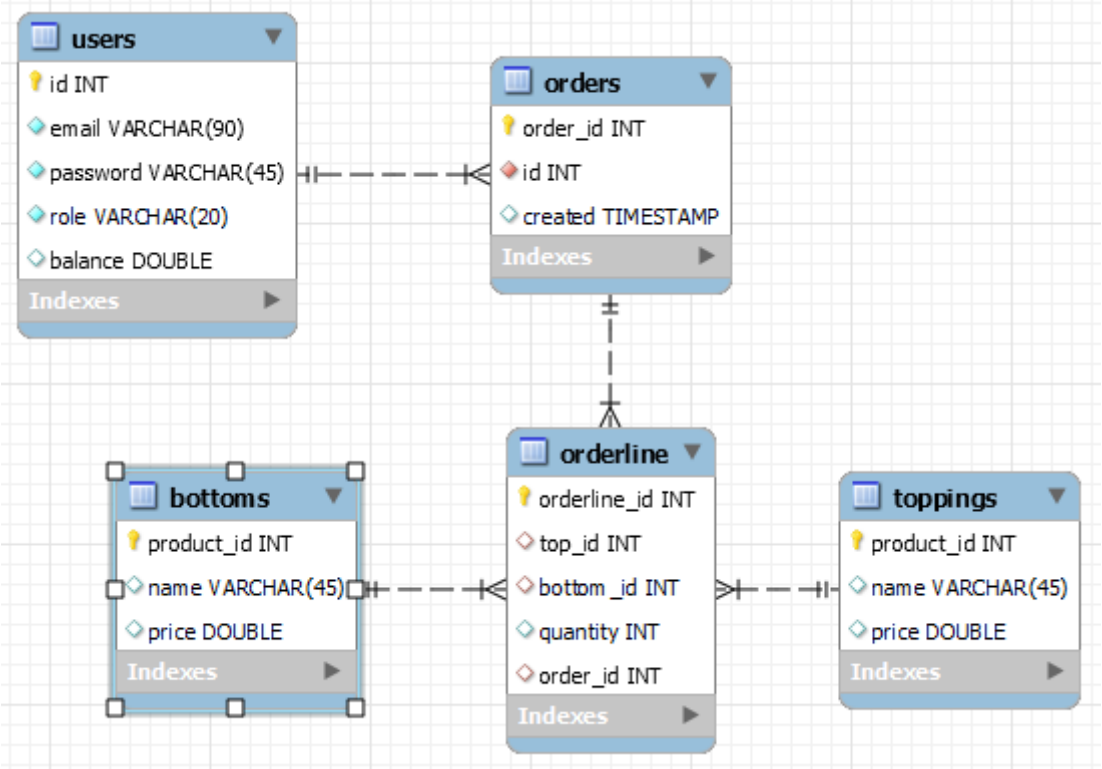
Domæne model



- Der er en Cupcake hjemmeside som modtager users via enten *Sign up* eller *Log in.* Cupcake siden har en til mange relation med *Users* da der kan være mange brugere på en hjemmeside
- En bruger har en *orderline*
- En *orderline* kan have mange forskellige variationer af cupcakes bestående af *Toppings* og *Bottoms*

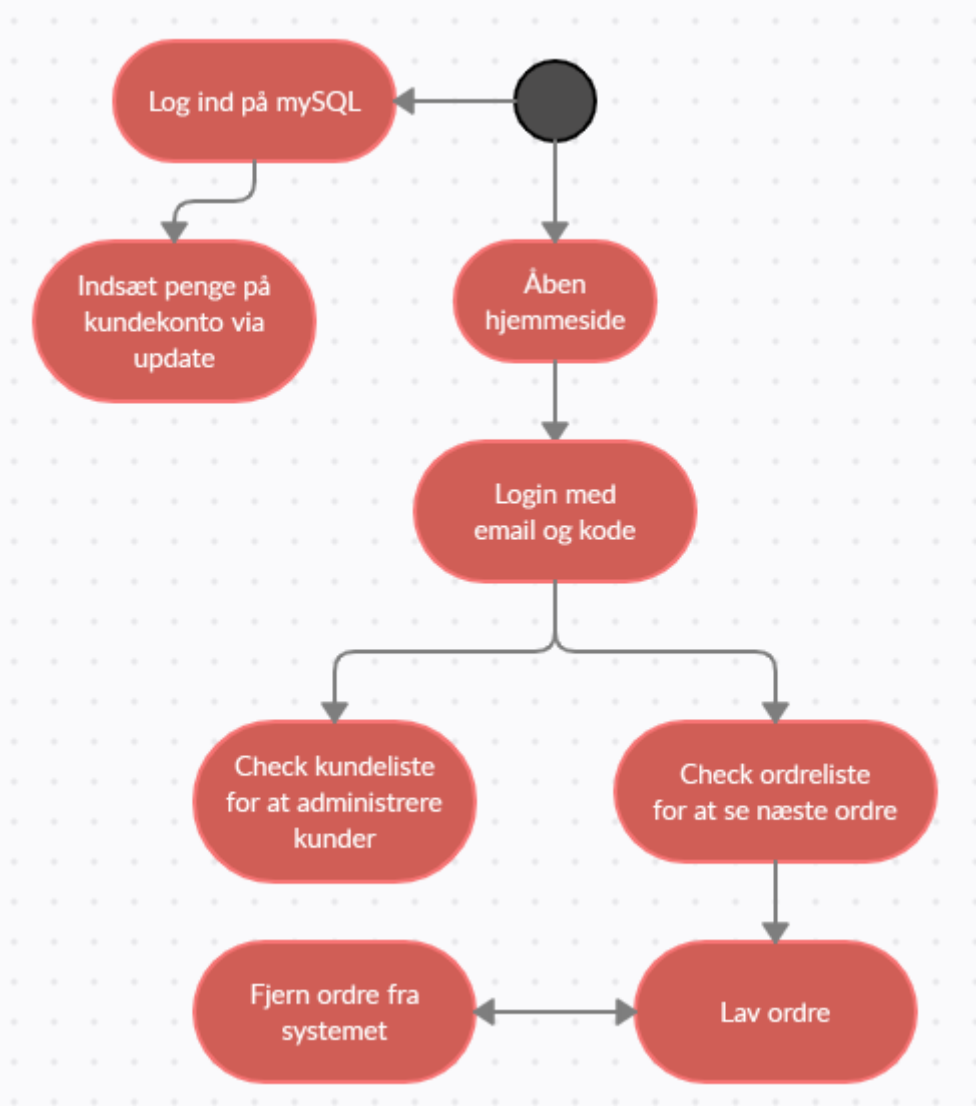
- En *orderline* har en til en relation med *payment*, du kan kun betale for en *orderline* ad gangen.
- En *payment* bliver betalt med *balance*, og når *orderline* er betalt har du en ordre, derfor en til en relation mellem *payment* og *order*.
- Der kan være mange forskellige *orders* på hjemmesiden tilhørende mange forskellige bruger, derfor har order en til mange relation med hjemmesiden Cupcakes.

EER-diagram



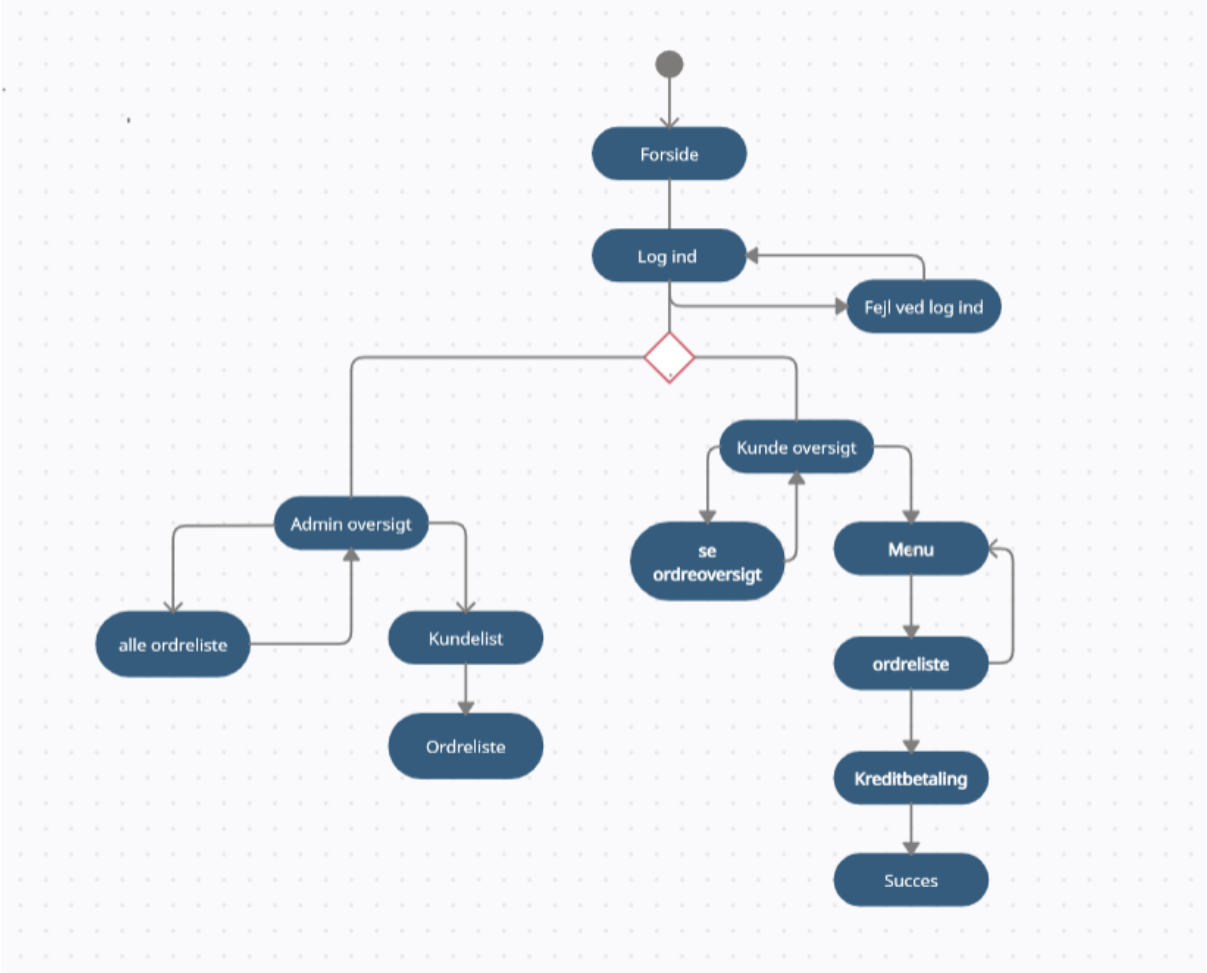
Da kunden selv skal kunne vælge både top og bund, er der lavet en *orderline* tabel hvor disse kan samles med antallet af denne type muffins. Man kan dermed bestille tre af én slags, to af en anden, og så videre, og disse '*orderlines*' vil dertil blive koblet sammen med orders på *order_id*.

Aktivitetsdiagram



Det er værd at notere at man kun kan lave en *employee* profil gennem mySQL. Diagrammet viser *workflowet* for virksomheden efter implementeringen af hjemmesiden. Det har ikke været muligt at lave en *AS-IS* model, da vi ikke har noget information omkring hvordan bestillingerne laves nu. Af denne årsag er det heller ikke muligt at bedømme hvor meget virksomheden vil få gavn af hjemmesiden. Det er dog rimeligt at antage, at det vil gøre arbejdsprocessen lettere for virksomheden, da de vil kunne modtage flere ordrer end hvis det, for eksempel, var telefonisk bestilling.

Navigationsdiagram



Navigationen igennem systemet er delt op til 2 spor efter log ind. Log ind siden omdirigere brugeren, tilbage til log ind siden ved fejl, i enten brugernavn, eller kodeord.

Kunde sporet: Efter log ind rammer man kunde oversigten, fra kunde oversigten har man mulighed for at vælge ordreoversigt for at se ordren der ligger i det nuværende session scope. Man kan også vælge at gå til cupcake menuen, derfra kan man komme til ordre listen, hvor man kan se ens valgte cupcakes. I ordrelisten kan man fjerne cupcakes, se den samlede pris for valgte, eller vælge at gå tilbage til menuen for at vælge flere cupcakes. Til sidst kommer man til kredit betalingssiden, hvor man kan se brugerens indestående kredit, samt det samlede beløb for de valgte cupcakes. Derfra kan man trykke betal, og man vil komme til succes siden.

Administrator sporet: Efter log ind rammer administrator oversigten, derfra kan man vælge at se alle ordre der er gået igennem systemet. Man kan også vælge at se hvilke brugere der er tilknyttet systemet, samt hvilke ordre hver enkelte brugere har valgt afgive.

Særlige forhold

Som udgangspunkt har Java og SQL entiteterne fulgt hinanden -- alle SQL tabellerne har en klasse i Java. På denne måde forenes attributterne, hvilket gør programmet mere effektivt, og mere læseligt. SQL bruges dermed også udelukkende til persistens. Med SQL kommer *exceptions*, og disse håndteres i metoderne hvori de bliver kastet. Her bliver en besked printet til konsollen, hvilket primært bliver brugt til *trouble-shooting*.

Det eneste sted der er validering af brugerinput er ved login, da alle andre inputs er gennem *dropdown-menuer* eller knapper. Log ind systemet fungerer desuden ved at brugernavn og kode bliver tjekket op mod databasen. Der er ingen reel kryptering på koden, hvilket naturligvis er et sikkerhedsproblem -- dog ikke et vi har valgt at arbejde videre på, da vi i stedet fokuserede på use-cases. Der er dog udelukkende gjort brug af SQLs *PreparedStatements*, hvilket burde forhindre SQL-injektioner.

Session-objektet er i høj grad blevet brugt til at håndtere indkøbskurven (*i.e. orderline*), da denne skal persistere hvis man skifter side. Det er desuden brugt til bruger-objektet ved login, af samme årsag.

Status på implementation

Vi har nået at implementere alle Jsp-siderne fra navigations diagrammet, dog mangler der funktionaliteten ved fjernelse af en bestemt valgt cupcake.

Use-case 1 endte som udgangspunkt med at være færdig, der er dog ikke nogen håndtering af hvad der skal ske hvis brugeren ikke har nok penge på sin konto. I dette tilfælde vil *balancen* blive sat til -1.0, hvilket senere vil kunne udbygges til at håndtere problemet.

Sproget på hjemmesiden er ændret til dansk, med undtagelse af fejlbeskeder.

Vi stødte ind i et problem, da vi ville lave *backend* koden til fjern-knappen. Meningen var at, vi skulle kunne fjerne en ordre fra indkøbskurven, men når vi ville genindlæse siden med det opdaterede *sessionScope* fik vi fejlen "*java.lang.NumberFormatException: null*". Vi havde ikke meget tid til at kigge videre på det, da det var vigtigere at prioritere vores rapport og *deployment* af websitet.

Proces

Vi startede projektet med at planlægge hvordan databasen skulle se ud. Dette fik vi gjort på første dag. Vi gik derefter i gang med at løse *use-casene*, med det primære fokus på de første seks, da disse var essentielle for programmet. Vi delte arbejdet nogenlunde ligeligt ud, men endte med at løbe ind i ufatteligt mange problemer med *use-case* 1. Vi endte også

med at bruge lang tid på *use-case* 4, da denne også drillede. Undervejs fik vi også lavet *mock-ups*, men disse blev dog tilsidesat frem for funktionaliteten, fordi vi i sidste ende vurderede det som værende vigtigst.

Af samme årsag er der heller ikke lavet tests til programmet.

Grundet uddelegeringen af arbejdet, endte vi også med at have problemer med sammensætningen af delene, da løsningerne ikke altid var ens. Vi måtte i sidste ende sidde sammen og rette til, så programmet blev mere sammenhængende. Dette kunne muligvis være undgået ved bedre kommunikation, og eventuelt flere gruppemøder.

Under nogle af de sidste udfordringer spurgte vi Tutors om hjælp, som ikke selv helt kunne forstå hvorfor nogle af vores fejl under *deployment* opstod, men fik guidet os i den rigtige vej mod løsningen.

Vi er som udgangspunkt tilfredse med projektet, men havde gerne set at al funktionaliteten blev implementeret. Vi har efterfølgende overvejet om man skulle have fokuseret mere på at få grundelementerne implementeret, specielt ved use-case 1, da denne inddrog de fleste af tabellerne. Det kunne derfor godt tænkes, at dette ville have givet et bedre grundlag for resten af projektet.