02.04. – 11.04 2024

Cupcake

Udarbejdet af datamatiker studerende på 2. semester. Hold B, gruppe 5.

Teodora Milijic, email: cph-tm291@cphbusiness.dk, github: TeodoraM24

Rasmus Kristoffer Knudsen, email: cph-rk181@cphbusiness.dk, github: RasmusKnudsen

Tobias Nielsen, email: cph-tn293@cphbusiness.dk, gitbub: Tobitastik

Contents

Indledning

Dette projekt er udført af datamatikerstuderende på 2. semester. Projektet er en fiktiv case, der skal repræsentere virkeligheden i en arbejdsplads. Denne rapport vil fokusere på vores analyse, design og implementering af de teknologiske valg, der er blevet foretaget. Projektet vil primært fokusere på vores arbejdsproces og de beslutninger, vi traf undervejs.

Baggrund

Olsker Cupcakes er et økologisk bageri, der har fundet den perfekte opskrift. De præsenterede os med en mockup og forskellige funktionelle krav.

Kunderne ønskede at kunne oprette en profil på siden, gemme deres ordrer, bestille og betale for cupcakes med ønskede toppings og bunde, og senere kunne afhente dem fra bageriet. Kunden kunne se alle deres ordrelinjer i kurven for at tjekke den endelige pris og også fjerne ordrelinjen fra kurven. Både kunder og administrator kunne logge ind med en e-mail og en adgangskode, og gerne se deres e-mail vist et sted på hver side.

Som administrator skal man kunne indsætte penge på kunders konti (direkte ind i Postgres), så kunderne kan betale for deres ordrer. Administrator skal også kunne se alle kunder og ordrer, så de kan følge op på ordrerne og holde styr på deres kunder. Som administrator kan man fjerne en ordre, for eksempel hvis kunden aldrig har betalt, så systemet ikke indeholder ugyldige ordrer.

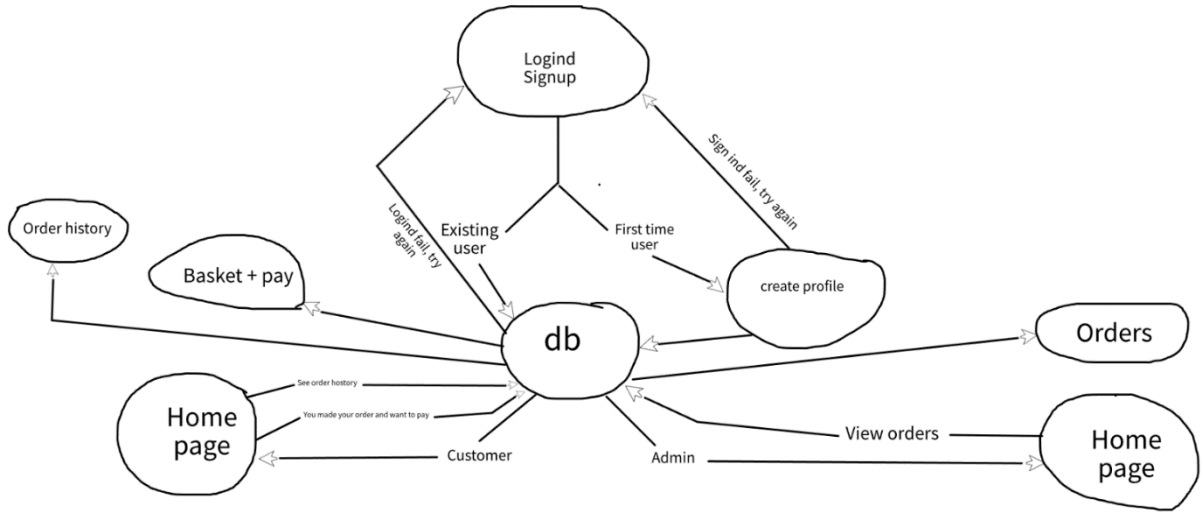
Teknologi valg

Teknologierne, der blev brugt til at skabe dette projekt, er JDBC, PostgreSQL, Java, og IntelliJ version 2023.3.5 og 2024.1

Arbejdsprocessen

UML-aktivitetsdiagram

Projektet blev startet ved at oprette en UML-aktivitetsdiagram, der beskriver forretningsprocessens arbejdsgang.



Dette diagram viser, hvordan hjemmesiden fungerer. Det starter med en velkomstside, hvor brugerne bliver bedt om enten at tilmelde sig eller logge ind. Hvis en bruger er en førstegangsbruger og derfor vælger at tilmelde sig, vil det føre dem til en side, hvor de skal oprette en profil. Hvis der opstår en fejl under oprettelsen af profilen, vil brugeren blive omdirigeret tilbage til velkomstsiden. Hvis ikke, gemmes oplysningerne i databasen, og nu skal brugeren logge ind igen (den del vises ikke på diagrammet).

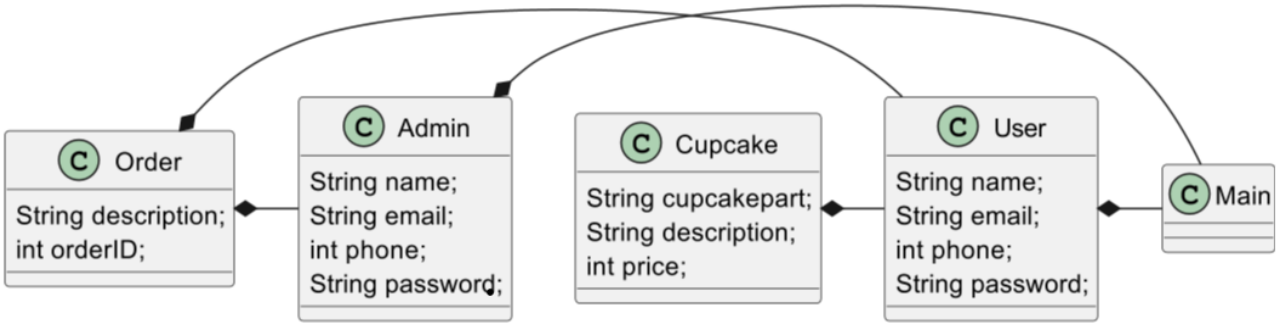
Når en bruger logger ind, går oplysningerne ind i databasen for at bekræfte brugernavn og adgangskode, men også for at tjekke profiltypen, som brugeren har. Der er to profiltyper, administrator og kunde. Baseret på valget ved tilmelding, omdirigeres brugerne til to forskellige startsider.

På kundens startside kan brugeren vælge deres cupcakes. Efter at have hentet oplysninger fra databasen kan kunderne se deres kurv og betale. Kunden kan også vælge at se deres ordrehistorik, som også henter oplysninger fra databasen.

På administratorens startside præsenteres de med en mulighed for at se ordrer fra alle kunder. Når de beslutter sig for at se ordren, kommunikerer programmet med databasen og henter aktuelle oplysninger.

Domænemodellen

Efter at have etableret arbejdsgangen på siden, begyndte vi at undersøge den visuelle repræsentation af enheder og deres relationer inden for problemområdet. Domænemodellen blev oprettet, og relationen mellem klasserne blev etableret.



Vi har fem klasser: "Main", "Cupcake", "User", "Admin" og "Order". I vores tilfælde er der en stærk livscyklusafhængighed mellem klasserne, hvilket gør det umuligt at fungere, hvis en bliver slettet. Dette kaldes sammensætning (composition).

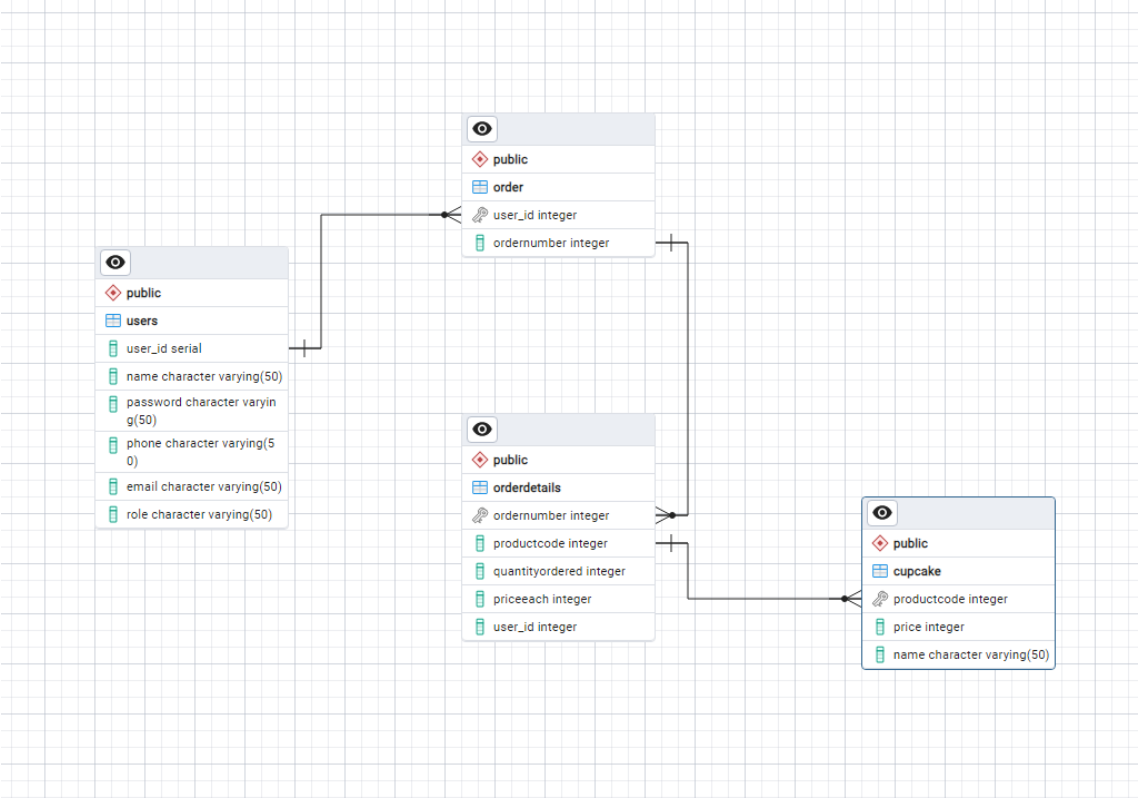
I dette tilfælde ville både klasserne "Cupcake" og "Order" ikke fungere uden klasserne "User" og "Admin".

Sammenfattende er klasserne afhængige af hinanden.

ER-diagram

Efter at have etableret hjemmesidens arbejdsgang og sammenhængen mellem klasserne, begyndte vi at arbejde på databasen i Postgres.

Første udkast til ER-diagram:



I users tabellen valgt vi at lave ”user_id” til serial så vi var sikker på hver tilmeldte bruger ville få et unikt bruger id. Id’en skal bruges til at holde styr på de forskellige brugers ordre.
”name”, ”password” og ”email” kolonnerne bliver gemt som character varying, op til 50 karakterer så der er plads til folk med meget lange navne eller meget sikre kodeord.
”phone” bliver også gemt som character varying, dette er gjort i håb om cupcakes fra Olsker bliver så populære at folk kommer fra Sverige og derfor bliver et krav til at gemme landekoden med i telefonnummeret.

Tanken bagved ”order” tabellen var at have en nem til gang til brugernes ordre, ved at holde styr på flere forskellige kombination af cupcakes fra en bestilling.

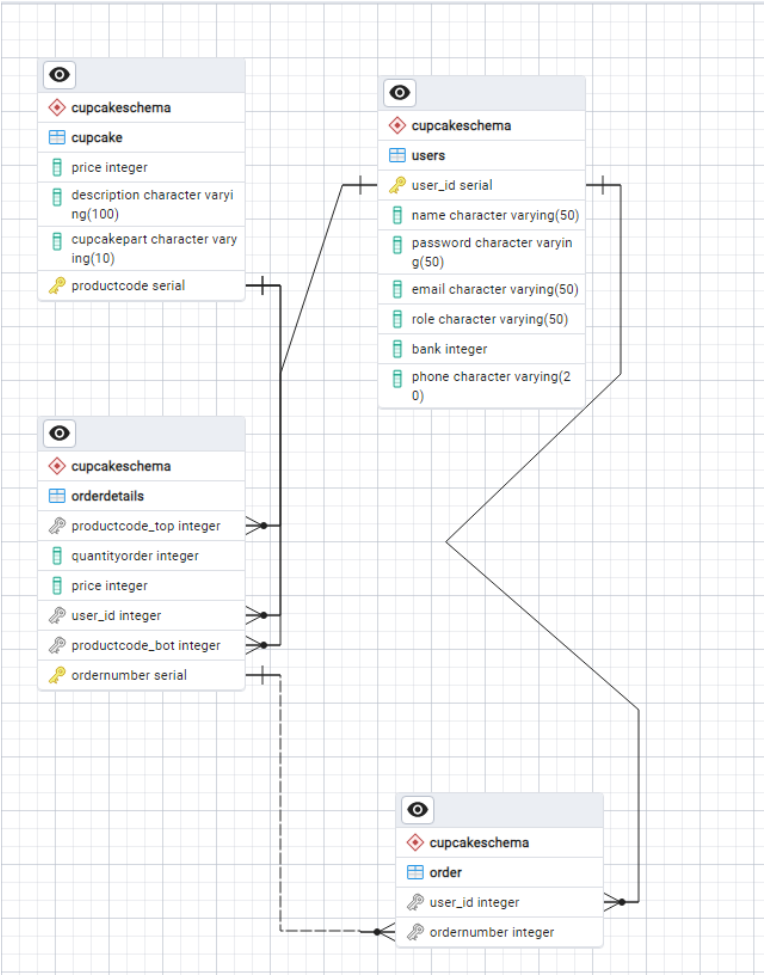
I ”orderdetails” taballen er tanken at databasen selv generere ordrenummeret så det er sikkert at ordre får sit eget id.
”productcode” er til at styr på hvilke kombination af cupcake der bliver bestilt.
”quantityordred” er til at holde styr på hvor mange af cupcakes der er bestilt.

”priceeach” er for den total af de bestilte cupcakes.
Til sidste er ”user_id” -kolonnen til at pege på hvilken kunde der har bestilt ordren.

Den sidste tabel er ”cupcake” hvor vi ville have tre kolonner, en ”produktkode”, en pris og et navn på bunden/topping til cupcaken.

Det var tankerne på vores første udkast af databasen, men i takt med at vi begyndte og kode viste det sig at vi havde glemt og overset nogle punkter til databasen.

Vores nuværende udgave af databasen:



Hvis vi begynder med ”cupcake” tabellen så er den fået navneskift på to tabeller samt en ekstra kolonne. Hvis vi begynder med den nye kolonne, så holder den styr på om det er bund eller

topping af cupcaken. ”name” kolonnen har skiftet navn til ”description”, men har samme funktion som da vi planlagde databasen.

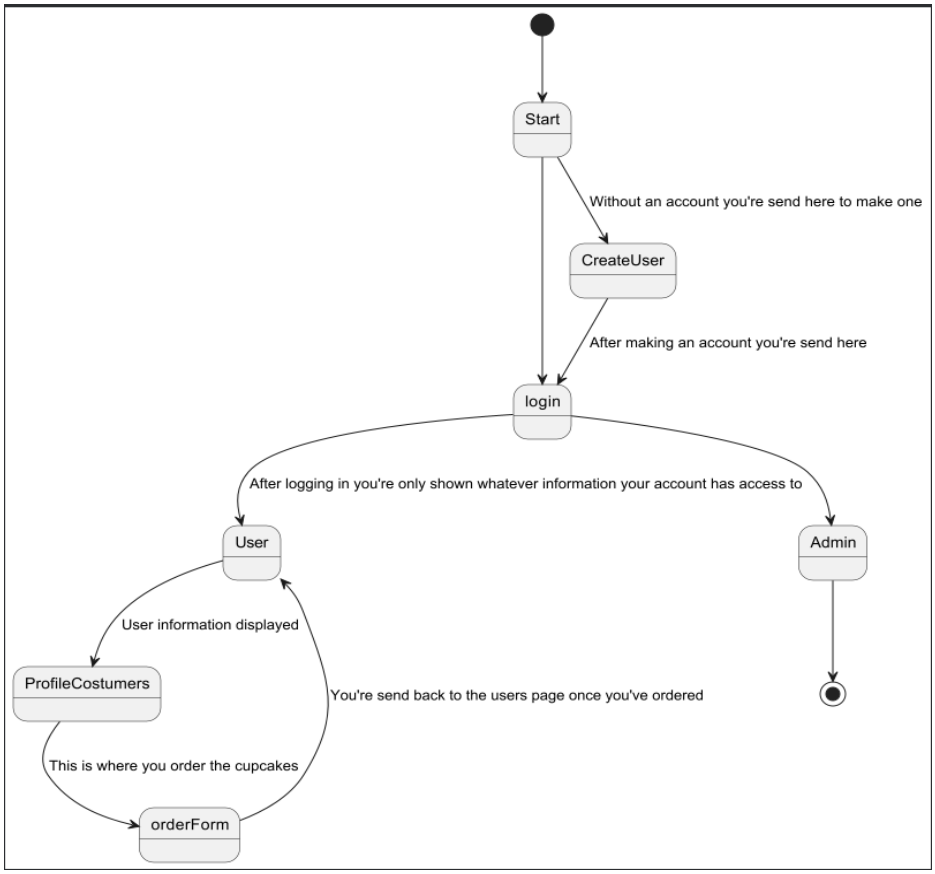
”Users” tabellen har også fået en ekstra kolonne som hedder ”bank”. Den er til at have styr på hvor mange penge den givende kunde har.
Når en kunde bliver oprettet, vil person automatisk blive sat som en “customer” i database. På forhånd er der oprettet en administrator i databasen. Udover de to ændringer er ”user” tabellen som den blev planlagt i begyndelsen af opgaven.

”Orderdetail” tabellen har også fået ændringer i løbet af projektet.
De mindste ændringer er kolonnen ”priceeach”, som har skiftet navn til ”price”.
Det mere spændende er ”productcode” som er blevet til to kolonner, som er til at holde styr på produktkoden til bunden af cupcaken ”productcode_bot” og topping ”productcode_top”. Det vil sige, at databasen er blevet en form for lappeløsning, da vi fandt ud af, at vores Cupcake klasses struktur ikke var optimal. Ideelt skal der kigges på Cupcake klassen i fremtiden, men programmet kører med lappeløsningen som den er nu.

Order tabellen har ikke set nogen ændring og dette skyldes prioritering og der desværre ikke var til tid og arbejde med order klassen.

Navigationsdiagram

Efterfølgende lavede vi et navigationsdiagram for at hjælpe os med at få et mere præcist overblik over alle sider i projektet.
Formålet med navigationsdiagrammet er at præsentere dette på en mere overskuelig måde.



Diagrammet ovenfor prøver at beskrive en tilstandsovergang for en bruger i et system, hvor brugeren kan logge ind, oprette en bruger. Er dette en administrator bruger kan de ikke komme videre, da projektet har en ufærdigt administrator-del. Derefter, hvis brugeren er en almindelig bruger, går de til en side, der viser deres profiloplysninger, og derfra kan de gå videre til ordreformularen for at bestille cupcakes.

Særlige forhold

For at forstå, hvordan oplysninger behandles og gemmes i databasen, skal vi se nærmere på vores UserMapper og UserController og deres createuser-funktioner.

UserMapper funktionen ansvarlig for at indsætte en ny bruger i en database tabel kaldet "users".

Funktionen tager fem parametre, "navn", "password", "email", "telefon" og "connectionPool", der bruges som et objekt af klassen "ConnectionPool", der administrerer forbindelser til databasen.

Funktionen konstruerer en SQL INSERT-forespørgsel for at indsætte en ny række med de angivne oplysninger i "users". Den får en databaseforbindelse fra forbindelsespuljen ved hjælp af metoden "getConnection()". Derefter opretter den et "PreparedStatement"-objekt til at udføre SQL-forespørgslen.

Funktionen angiver parametrene for det forberedte udsagn med de værdier, der er givet til funktionen. Derefter udfører den SQL INSERT ved hjælp af "executeUpdate()", der returnerer antallet af berørte rækker af forespørgslen.

Hvis antallet af rækker ikke er lig med et, kaster den en "DatabaseException" med en fejlmeddelelse. Hvis der opstår en SQL-undtagelse under forespørgselskørslen, fanger den undtagelsen. Hvis meddelelsen indikerer en duplikatnøgleovertrædelse (fx brugernavn), giver den en brugerdefineret fejlmeddelelse.

For at opsummere giver funktionen en sikker måde at indsætte nye brugere i databasen, samtidig med at den håndterer potentielle fejl, der kan opstå.

I UserController er funktionen ansvarlig for at håndtere brugeroprettelsesforespørgsler. Den tager to parametre: "ctx," et "Context" objekt, der indeholder information om den aktuelle webanmodning og -respons, og "connectionPool."

Funktionen henter oplysninger fra webanmodningen ved hjælp af "formParam" metoden fra "Context" objektet, som inkluderer brugernavn, to adgangskoder (som skal matche), e-mail og telefonnummer.

Funktionen validerer adgangskoderne for at sikre, at de matcher. Hvis adgangskoderne matcher, forsøger funktionen at oprette en ny bruger ved at kalde "UserMapper.createuser" funktionen, som er ansvarlig for at indsætte brugeroplysninger i databasen, som tidligere forklaret.

Hvis der opstår en "DatabaseException" under oprettelse af brugeren, håndterer funktionen den ved at angive en fejlmeddelelse i "ctx" objektet og vise en relevant side til brugeren.

I løbet af funktionen får brugeren feedback om resultatet af oprettelsen eller eventuelle fejl. Dette inkluderer meddelelser om succesfuld oprettelse, database-relaterede fejl og adgangskodefejl.

Alle funktioner i programmet er oprettet baseret på den samme koncept, hvilket gør dem så sikre som muligt, men også informative for alle brugere.

Arbejdsgang

Vi organiserede arbejdet i kanban. Hele processen var ny for os, og vi kunne have gjort det bedre, når det kommer til kommunikation. Vi brugte fire dage på login og tilmelding, hvilket tog tid, der kunne have været brugt til andre vigtige dele af projektet. Fordi vi ikke kunne finde en løsning, brugte vi login og tilmelding fra vores tidligere projekter og tilpassede det til Cupcake-projektet. Når databasen var oprettet, og login fungerede, fokuserede vi primært på brugersiden af programmet, hvilket resulterede i et ufuldendt produkt på admin-siden. Vi kunne ikke opfylde alle user-stories.

US-1: "Som kunde kan jeg bestille og betale cupcakes med en valgfri bund og top, så jeg senere kan køre forbi butikken i Olsker og hente min ordre.", er delvist udført, brugere kan vælge top og bund af cupcake, uden mulighed for at betale og afhente senere.

US-2: "Som kunde kan jeg oprette en konto/profil for at kunne betale og gemme en en ordre.", er også delvist færdig, kunder kan oprette en profil, og der er funktioner i projektet, der gemmer ordrer, men desværre kan det ikke vises på grund af den ufuldendte betalingsproces.

US-3: "Som administrator kan jeg indsætte beløb på en kundes konto direkte i Postgres, så en kunde kan betale for sine ordrer.", som tidligere nævnt er administrator-delen af hjemmesiden ufuldendt.

US-4: "Som kunde kan jeg se mine valgte ordrelinjer i en indkøbskurv, så jeg kan se den samlede pris." er ikke færdig.

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).", brugere kan logge ind med et brugernavn i stedet for en email, men deres email vises ikke.

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

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." er ufuldendt.

US-8: "Som kunde kan jeg fjerne en ordrelinie fra min indkøbskurv, så jeg kan justere min ordre." er ufuldendt.

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." er ufuldendt.

Det, der kunne have hjulpet os med at færdiggøre dette projekt til tiden, var bedre kommunikation og koordination af arbejdet, men også mere tid.

Kilder

https://github.com/dat2Cph/content/blob/main/cupcake/rapportskabelon.md

https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-aggregation-vs-composition/