Cupcake

Hold: Lyngby A-6
Copenhagen Business School - Datamatiker 2. sem. 2024

Oskar Bahner Hansen & Nicolai Jahncke Strand
Mail: obh82@cphbusiness.dk & <email>
Github: oskar123456 & <git>

10. april 2024

Indhold

1 Indledning

1.1 Baggrund

Denne rapport omhandler **Cupcake**-projektet på 2. semester af datamatiker uddannelsen.

Projektet gik ud på at forundersøge og implementere en hjemmeside til et bageri, der sælger *cupcakes*. En række funktionelle krav blev udleveret i form a *user-stories*, på baggrund af hvilke vi kunne gå i gang med en forundersøgelse og senere implementation af en simpel, men forhåbentlig nogenlunde funktionel butikshjemmeside.

Som nævnt var vores opgave, at implementere en butikshjemmeside til et *cupcake*-bageri. Dvs. der skulle være mulighed for at bestille, samt for en *administrator* at kunne undersøge og ændre på bestillinger (ofte referet til som *ordrer* i koden). Der skulle altså strikkes et software-produkt sammen, der både kunne servicere brugerforespørgsler gennem en browser, og som kunne gemme nødvendige oplysninger permanent i en **database**.

1.2 Prototype

Projektet kan findes på github, og kan f.eks. bygges med **JetBrains IntelliJ Idea**.

Videodemonstration af den færdige prototype kan findes på dette link.

1.3 Teknologivalg

Implementationen afhænger af:

- `Java` **17**
- `Javalin`-*web-framework*
- `Thymeleaf`-*html-template-framework*
- `Postgres`, `HTML` og `CSS`

Derudover brugte vi `JetBrains IntelliJ Idea` **2024.1**

1.4 Krav

Håbet med projektet var, at få stablet en hjemmeside på benene til salg af *cupcakes*. Kunder skulle kunne bestille og derefter hente deres bestillinger i butikken fysisk, og i den forbindelse ville en *administrator*-bruger (ansat/medejer) kunne ordne betaling fysisk, for derefter at nedfælde betalingen i systemet. Dvs. vi ikke havde behov for nogen særlig grundig behandling af betalinger i systemet, men blot en simpel brugerflade til at holde styr på betalingshistorik.

1.4 User-stories

Kravene til hjemmesiden/systemet blev udmundet i en række *user-stories* (i alt ni), som dannede grundlag for design og implementation:

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 Postgres, 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.

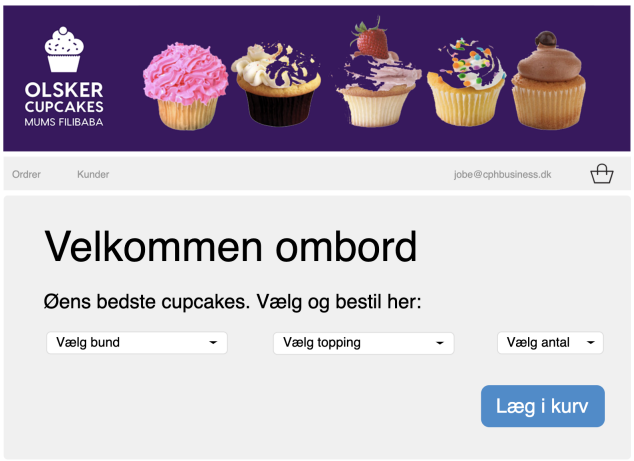
US-7: Som administrator kan jeg se alle kunder i systemet og deres ordrer, sådan at jeg kan følge op på ordrer og holde styr på mine kunder.

US-8: Som kunde kan jeg fjerne en ordrelinie 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.

1.5 Udseende

Der var lavet et *mockup* af hvordan hjemmesiden skulle se ud:



(1.1)

Der vil blive taget udgangspunkt i dette til alle eventuelle undersider. Dvs. de skal alle have en navigationsbjælke og en titelbjælke med logoet.

2 Design

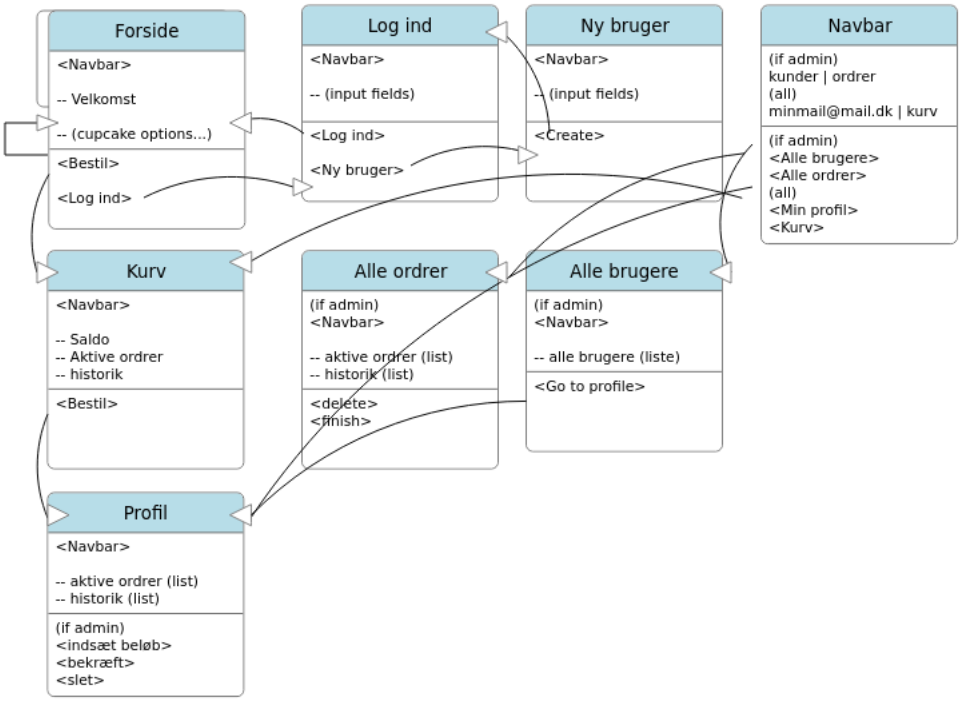
2.1 Udseende

Der er taget udgangspunkt i den udleverede *mockup* til alle sider og undersiders design, dvs. de vil få et udseende som ligger så tæt som muligt op af forsiden.

2.2 Domæne, navigation & database

2.2 Domæne

På baggrund af de ovenstående funktionelle krav blev der udarbejdet en domænemodel, i et forsøg på at sikre implementationens opretholdelse af de funktionelle krav allerede i design-fasen:



(2.1)

De første modelskitser blev skrevet i hånden, men er i rapporten blevet erstattet med mere læsbare digitale versioner.

US-1 dækkes af `forside` og `kurv`, hvor der kan vælges mellem aktuelle tilbud, og derefter lægges i en kurv, samt bekræftes en liste over bestillinger. Derudover er det muligt for en *administrator* at bekræfte en bestilling, dvs. når den er betalt og hentet fysisk i butikken.

US-2 dækkes selvsagt af `ny bruger`.

US-3 dækkes af `profil`, hvor en *administrator* udelukkende vil have mulighed for at ændre eller slette en kundes bestilling.

US-4 dækkes selvsagt af `kurv`.

US-5 dækkes selvsagt af `log ind`.

US-6 dækkes selvsagt af `alle ordrer`.

US-7 dækkes af `alle brugere`, hvorfra der er et link fra hver bruger til deres profil. På selve profilen vil der udover ordrelister, være særlige administrator-muligheder.

US-8 dækkes af `kurv`, hvor der vil være mulighed for at bekræfte en bestilling. Dvs. intet skal gemmes i systemet, førend bestillingerne er bekræftede.

US-9 dækkes af særlige administrator-muligheder på alle kunders profiler.

2.3 Database

For at opfylde krav, og få systemet til at kunne gemme kundeinformation samt bestillinger permanent, er vi nødt til at have en database. Udfra systembeskrivelsen har vi følgende datatyper, der skal indgå i databasen:

Tabel 2.1: users

navn	email	role	password	ordre_id	ordre_spec	pris	tid_bestilt	tid_ønsket	tid_færdig

ordre_spec ville være en ordrebeskrivelse; choko-bund orange-top 2 stk.

Dataen bringes på **normalform**:

1NF: Vi gør alle værdier atomarer, ved at opdele cupcake-detaljer, og skaber et unikt id, samt flytter

ordredetaljer over i deres egne tabeller, for at undgå gentagne rækker (teknisk set kan email også bruges som unikt id for brugere):

Tabel 2.2: users

id	email	role	pwd	saldo

Tabel 2.3: order

id	cust_id	create_date	desired_date	finish_date	bot	top	quant	price

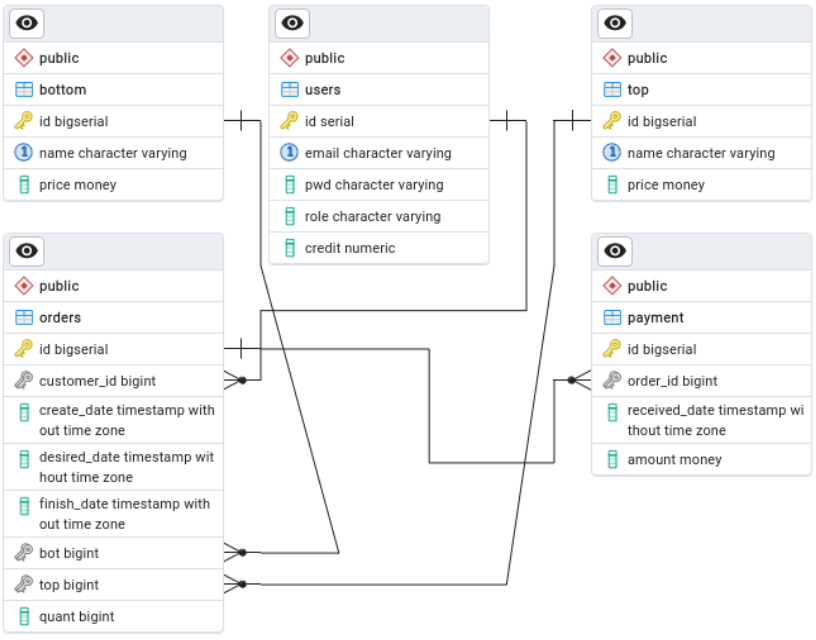
2NF & 3NF: 2. normalform er allerede opfyldt, da vi ikke har nogen sammensatte nøgler. Dog er data ikke på 3. normalform, da `price` afhænger af `bot`, `top` og `quant`, som ikke selv er en del af primærnøglen. Derfor flyttes cupcake-muligheder og priser over i deres egen tabel:

Tabel 2.4: bot & top

id	name	price

2.4 ERD

Her er et **ERD**-diagram over den endelige database, som er genereret vha. **pgadmin**:



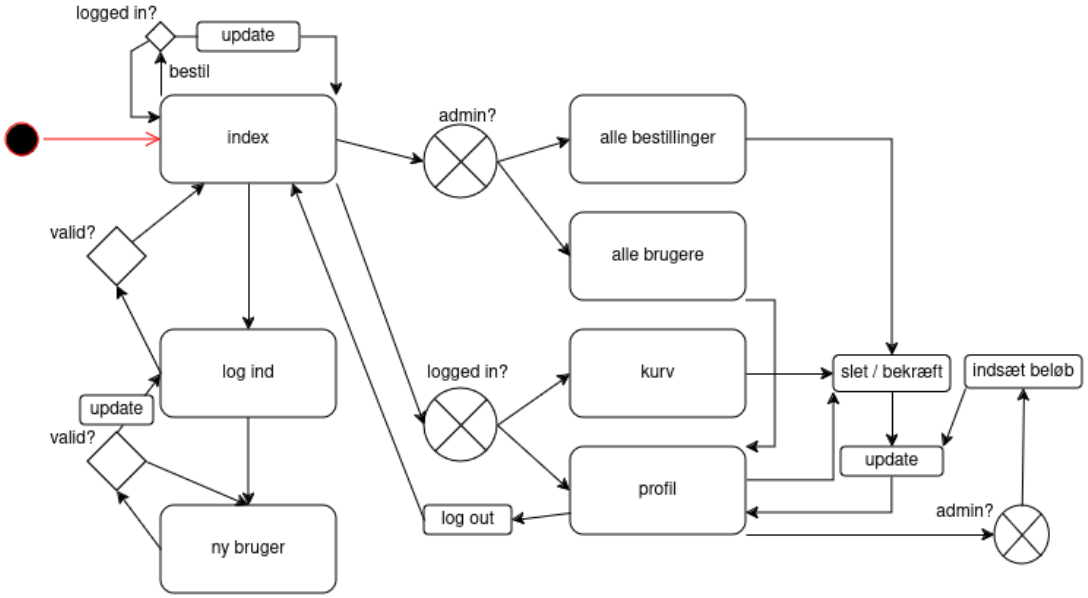
(2.2)

Alle relationer er en-til-mange (repræsenteret med *crow's foot ERD notation*), og der gøres brug af **foreign-keys** til at sikre, at f.eks. en odre skal være knyttet til en eksisterende bruger osv.

2.5 Navigation

Ovenstående domænemodel var ikke tilstrækkeligt konkret, til at implementere systemet, og vi havde behov for at skitsere en model over hjemmesidenavigationen. Nedenstående er et slags tilstandsdiagram, som er udarbejdet med det formål, at gøre implementationsfasen mere gnidningsfri, idet der allerede er skabt en logisk struktur over systemet på forhånd.

Figur 2.1: Navigations-/Tilstandsdiagram



med `update` *forstås en opdatering af systemets data*.

Entry-point er altså `index`, hvorfra man har forskellige muligheder, afhængig af, om man er logget ind, og hvilken *rolle* man har; admin/almindelig bruger.

$\otimes$ indikerer om en tilstand er vist/opnålig afhænging af et prædikat, og $\Diamond$ indikerer en beslutning der afhænger af et prædikat.

2.6 Særlige forhold

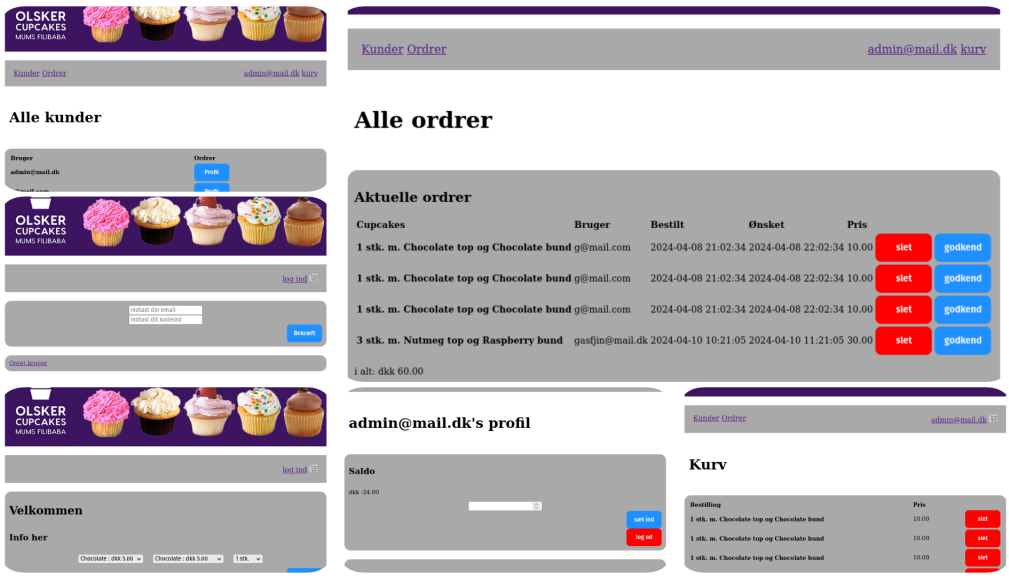
Da betaling og afhentning sker fysisk i butikken, er der ikke taget højde for disse funktionaliteter i selve systemet. Der er blot en simpel funktion til at indsætte og trække fra på brugernes konto, for en administrator.

3 Implementation

Der er implementeret en funktionel prototype af det beskrevne system i design-afsnittet. Det er pt. muligt at oprette en bruger, logge ind, bestille frit fra de på forhånd udleverede valgmuligheder. En administrator konto har derudover de muligheder som krævet, og systemet sørger for, at *kun* en administrator har adgang til disse.

3.1 Udseende

Hjemmesidens udseende er skabt med `html` og `css`. Det er gjort nogenlunde så simpel som muligt. Her ses alle prototypens endelige sider:



(3.1)

Der er ikke pt. understøttelse for mindre skærme (under 600pixel). Der vil kanterne på siden forsvinde.

3.2 Kode

Hjemmesiden er som skrevet indledningsvist implementeret i java, og gør brug af `javalin`, som bruger en `Jetty`-http-server til at servicere brugere. Routes er sat op i en **controller** (`CupcakeController.java`) således:

```
public static void addRoutes(Javalin app, ConnectionPool cp) {
    app.get("/", ctx -> renderIndex(ctx, cp));
    app.get("/index", ctx -> renderIndex(ctx, cp));
    ....
    }
```

Og styrer dermed tilstandsskift, som skitseret i tilstandsdiagrammet. Programmet kan således route brugere rundt på følgende `html`-sider:

```
allorders.html
create.html
customers.html
index.html
login.html
myorders.html
orderconfirm.html
profile.html
```

Nogle routes kræver som beskrevet administratorrettigheder, som f.eks. sikres ved kode som dette:

```
    User user = ctx.sessionAttribute("user");
    if (!isAdmin(user)) {
        ctx.attribute("message", "adgang nægtet");
        renderIndex(ctx, cp);
        return;
    }
```

Derudover bruges **thymeleaf**, til selektivt at vise visse elementer:

```
    <div class="navbar">
        <div class="navbarLeft">
            <a th:if="${session.admin == true}" th:href="customers">Kunder</a>
            <a th:if="${session.admin == true}" th:href="allorders">Ordrer</a>
        </div>
```

Med hvilket man sikrer, at selve html-kildekoden der sendes til en klient ikke indeholder følsom information, som kun en administrator må kunne tilgå.

Når en bruger lægger en bestilling i sin **kurv**, bliver oplysningerne faktisk kun gemt i en `Context.sessionAttribute` (dvs. i en **cookie**). Det er først ved bekræftelse under **profil**, at bestillingen bliver lagret i databasen.

For at opdatere systemets database, bruges `jdbc` til at tilgå en **postgres**-server. Alt databasetilgang foregår i filen `CupcakeMapper.java`. Der gøres brug af `preparedstatement`-klassen fra `java.sql`-biblioteket, for at undgå f.eks. **SQL**-injections.
Her er et eksempel:

```
public static int AddCredit(ConnectionPool cp, User ...)
throws DatabaseException
{
    int retval = 0;
    String sql = "UPDATE users SET credit = credit::numeric + ? WHERE id=?";
    try (
            Connection c = cp.getConnection();
            PreparedStatement ps = c.prepareStatement(sql)) {
        ps.setBigDecimal(1, amount);
        ps.setInt(2, user.Id());
        retval = ps.executeUpdate();
    } catch (SQLException e) {
        throw new DatabaseException(
        "fejl ved indsættelse af penge på konto " + user.Email()
        );
    }
    return retval;
}
```

Hvori der skal indsættes et beløb på en brugers konto (kaldet saldo eller *credit* i systemet).

4 Konklusion

Afslutningsmæssigt kan vi konkludere, at hjemmesiden lader til at virke, som den skulle ift. designet. Dog er koden langt fra robust, eller pæn. Dvs. det vil formentlig være nemt at få programmet til at crashe eller vise noget forkert ved mere kritiske test. Produktet er ikke testet godt igennem, kun overfladisk *acceptance*-testing er fortaget, for at se om produktet opfylder de ni udleverede *user-stories*.

Det at koden ikke er pæn hentyder til, at der ikke er en god struktur, især i `CupcakeController.java`, hvis hjemmesiden skulle videreudvikles, ville der formentlig skulle foretages en *re-write* på i hvert fald den klasse. Den overholder f.eks. ikke **Single-Responsibility**-princippet fra **SOLID**; klassen har nok flere opgaver end den burde. Derudover er de enkelte metoder en del rodede.
Det kan skyldes, at der ikke blev afsat nok tid til kodning, men måske også en lidt forhastet designfase, hvor programstrukturen måske ikke blev hamret godt nok fast.

En note angående kodeord; som de er lagt ind i databasen i klartekst-format ville man ændre til en hash, sådan at man ikke kan få direkte adgang til brugeres private kodeord, som de måske bruger på flere konti.

Hjemmesidens udseende er ikke responsiv ift. mindre skærme, som smartphones osv. Det nåede vi ikke at få styr på.