Klasse: A

Dato:11/04-2024

Gruppe: 5

Cupcake

Deltagere:

Mahdi Michael Karimi (GitHub: cphmk)
E-mail: cph-mk864@cphbusiness.dk

Claus Peter Jørgensen (GitHub: Claus0200)
E-mail: cph-cj539@cphbusiness.dk

Benjamin Sebastian Barrales Hernandez (GitHub: BenjaminHernandez95)
E-mail: cph-bh226@cphbusiness.dk

Ferdinand Amstrup Vestergaard (GitHub: FerdinandAV)
E-mail: cph-fv49@cphbusiness.dk

Indholdsfortegnelse

Indledning

Baggrund

Dette er et Cupcake projekt til bageriet ”Olsker Cupcakes”, der ligger på Bornholm. Projektet består af en hjemmeside, der kan bruges til at bestille Cupcakes online. Kravet til systemet er at kunden skal kunne bestille en cupcake, og vælge en valgfri top, bund og antal af cupcakes. Bunden og toppen der vælges, kunne være chokolade og jordbær.

Teknologi valg

Indenfor teknologier har vi brugt:

Teknologi og version	Beskrivelse	Front- eller Backend
Java - version 17	Hoved programmering sprog, bruges til at skrive vores program i.	Backend
Javalin – version 6.1.3	Web framework, bruges til at skabe overgang mellem Java programmet og html, igennem ruter.	Backend
JDBC (postgres sql driver) – version 42.7.2	Java database connectivity, bruges til at forbinde til databasen i Java.	Backend
Thymeleaf – version 3.1.2	Modern server-side Java template engine, bruges til at erstatte data på vores hjemmeside, og gøre den dynamisk.	Backend
HTML og CSS	Hypertext markup language og cascading style sheets, bruges til at bygge og style hjemmesiderne.	Frontend
PgAdmin – version 8.3	Managment tool for PostgreSQL database, som bruges at lave ERDiagrammer, og indsætte/oprette data på databasen.	Backend
Postgres – version 16.2	Database managment system, bruges til at holde på vores data i form af tabeller med kolonner og rækker	Backend

Krav

Firmaets vision for systemet:

Firmaet Olsker Cupcakes ambition, er at implementere et online bestillingssystem, der vil revolutionere kundens oplevelse, og samtidig forbedre firmaets effektivitet. Visionen er at tilbyde kunderne en problemfri og bekvem platform, som de nemt kan anvende til at bestille deres fortrukne cupcakes, med et personligt valg af top og bund, som kunne bestå af chokolade og chokolade. Der sigtes efter at leverer en enestående kundeservice, og langvarige kunderelationer, gennem brugervenlige funktioner, og pålidelig ordrehåndtering.

User stories:

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 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 e-mail og kodeord. Når jeg er logget på, skal jeg kunne se min e-mail 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.

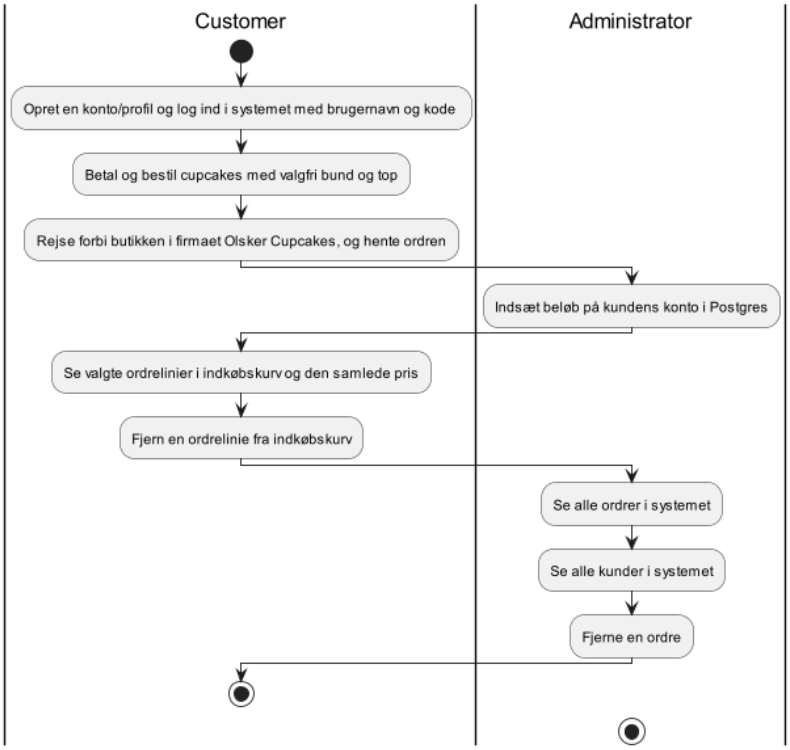
US-7: Som administrator kan jeg se alle kunder i systemet og deres ordrer, sådan at jeg kan følge op på ordrer og holde styr på mine kunder.

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

De følgende 9 user stories er kommet ud fra dialog med kunden.

Aktivitetsdiagram



Figur 1 - Aktivitets diagram

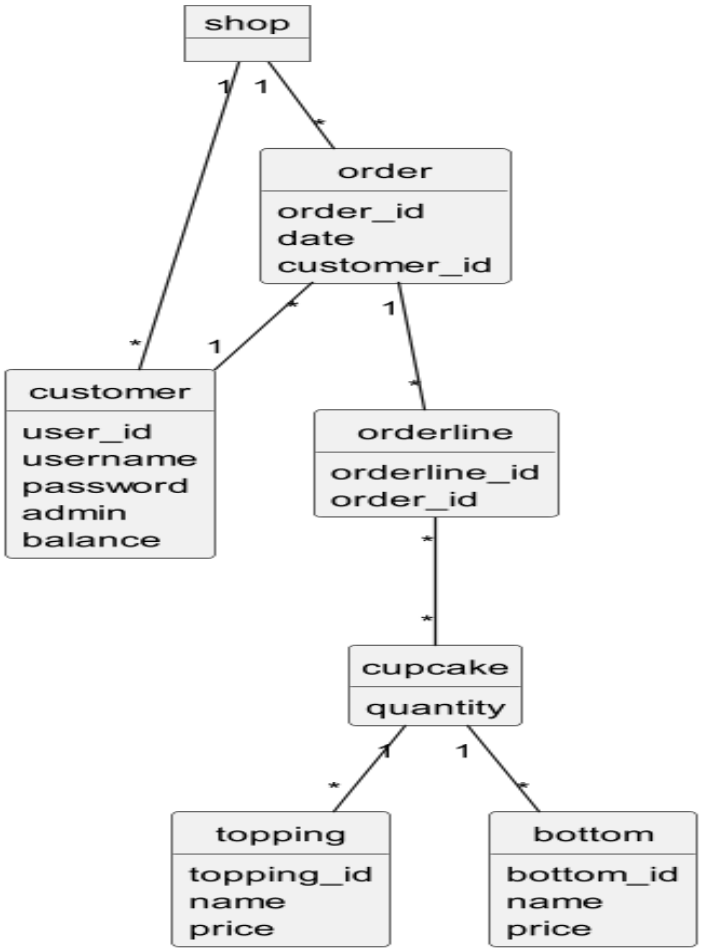
Vores aktivitets diagram beskriver trinene i vores arbejdsgang. Vi har kunden i diagrammet, der skal kunne oprette en konto, og logge ind på hjemmesiden. Derefter har vi en beskrivelse af hvad kunden skal kunne gøre, som betale, og bestille cupcakes så kunden kan rejse forbi og hente bestillingen. Kunden skal have adgang til sin ordrelinje i indkøbskurven, og den samlet pris på sin bestilling, og har også mulighed for at fjerne sin bestilling i systemet.

I diagrammet ser vi en beskrivelse på, hvad Administratoren skal kunne. Administratoren er i stand til at kunne indsætte et beløb, på kundens konto i Postgres, det kan være fordi, hvis der opstår en fejl i betalingssystemet, eller hvis en kunde oplever problemer med at betale for deres ordrer, kan administratoren have brug for manuelt at indsætte et beløb på kundens konto for at rette fejlen, så sikre man sig også at kunden får den ønskede vare. Administratoren har adgang til alle ordrer, og skal også være i stand til at kunne fjerne ordrerne, fra systemet.

Det her Aktivitets diagram repræsenter en TO-BE arbejdsgang, da det beskriver en ønsket fremtidig arbejdsgang, hvor forskellige handlinger udføres af kunden, og administratoren efter implementeringen af det nye system. Hvis Diagrammet repræsenter den aktuelle arbejdsgang som også kaldes for AS-IS, så eksisterer den på tidspunktet for diagrammets udarbejdelse. Dette er ikke tilfældet, da diagrammet så vil afspejle, hvordan tingene fungerer på nuværende tidspunkt, inden nogen ændringer er foretaget.

Domæne model og ER diagram

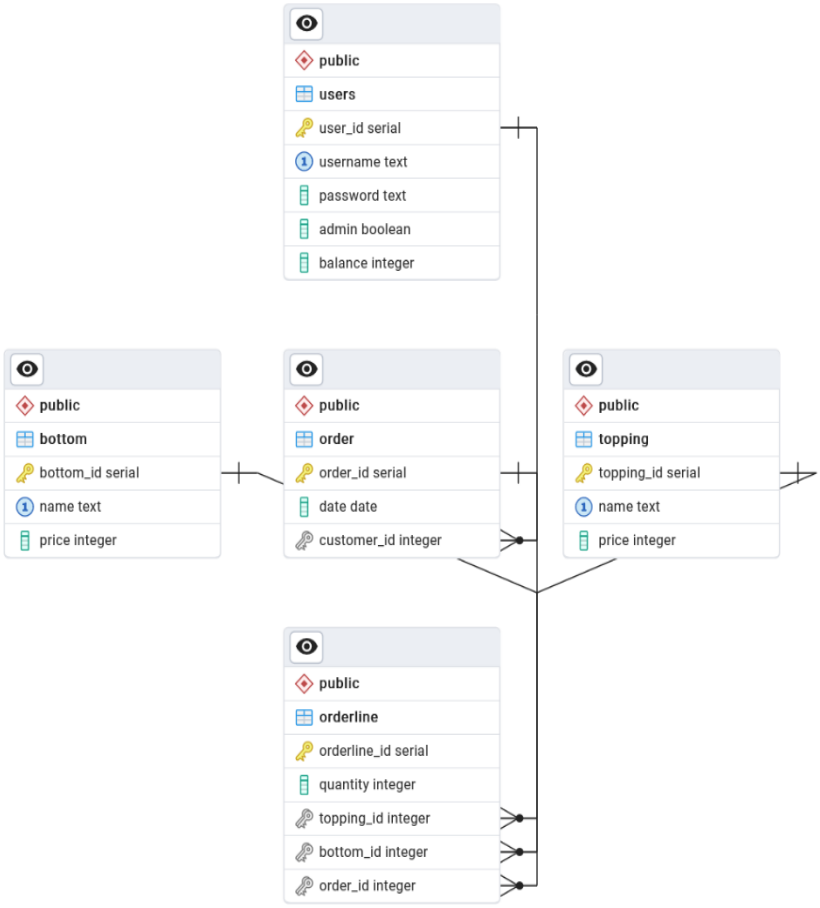
Domænemodel:



Figur 2 - Domænemodel

Domænemodellen repræsenterer virkeligheden og hvordan systemet skal bygges op. Den har en shop, som har en-til-mange relation til både kunder og ordre. Kunden har et id, brugernavn, kodeord, admin (En værdi der fortæller om brugeren er admin), og en balance. Desuden er der en forbindelse mellem kunden og ordre, da en kunde har flere ordre i systemet, og en ordre tilhører kun en kunde, altså en-til-mange. Ordre har også et id, en dato, og et id der fortæller hvilken kunde den, hører til. Ordrelinjer har så en mange-til-mange relation med cupcakes, da der kan være flere cupcakes på en Ordrelinjer, og cupcakes kan tilhøre flere Ordrelinjer. Cupcakes har også en bund og top, med navn, pris og id.

ER diagram:

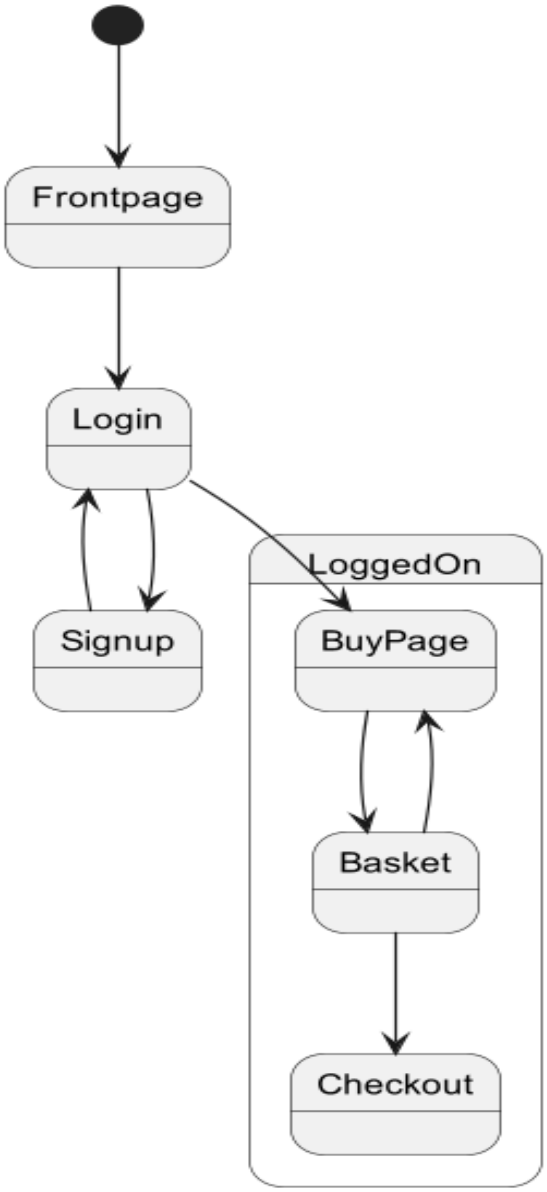


Figur 3 - ERDiagram

Entity-Relationship Diagram er vores diagram der repræsenterer databasen. Den består af 5 tabeller som er: 1. users 2. order 3. orderline 4. bottom 5. topping

Tabellerne har en primær nøgle, hvor der kun er atomarværdier og ingen gentagne kolonner. Der er ingen sammensatte nøgler eller transitive afhængigheder. Der er heller ikke et felt i en tabel som entydigt afhænger af et andet felt, hvor det felt ikke er en primærnøgle. Derfor er ER diagrammet i overensstemmelse, med den tredje normalform (3NF) Vi fik ikke implementeret kundens: personid -> postnr -> by (x -> y -> z), da vi følte at det ikke var så nødvendigt igen.

Navigationsdiagram



Figur 4 - Navigationsdiagram

I dette Navigationsdiagram, ser vi hvordan man skal kunne navigere sig igennem siderne, på hjemmesiden. Man starter på Frontpage, og kan så klikke på logge ind, for at gå videre. Derefter har man muligheden for at logge ind, eller oprette en konto. Her kan man så navigere sig frem og tilbage imellem login og signup, men for at komme videre skal man logge ind efter man har oprettet en konto. Når man så har logget ind, så kommer man videre til LoggedOn status, og her kan man bestille cupcakes på buypage og tjekke sin kurv på basket. Desuden kan man navigere frem og tilbage imellem buypage og basket. Når man er færdig, klikker man på checkout på basket siden, og så kommer man til checkout siden.

Særlige Forhold og kode eksempler

Hvilke informationer gemmes i session:

Når man opretter en ny bruger på hjemmesiden, så bliver dit brugernavn og kode brugt til at tjekke om der eksisterer en bruger i database. Hvis det lykkedes, så er brugeren logget ind og derefter oprettes der et User objekt (hvilket indeholder de samme værdier som i databasen) som bliver gemt i sessionen. Cart (ordre) med Cartlines (ordrelinjer) bliver også gemt i en session således at man kan se sine Cartlines og ændre på dem, uden at det bliver gemt på databasen med det samme. Herefter man går videre til checkout, så bliver begge session attributter User og Cart brugt til at sende data videre til databasen og gemme det. Som administrator kan man hente alle ordre og ordrelinjer fra databasen, så man kan se og fjerne dem. De bliver også gemt i session, så de er nemmere at tilgå, når man ændrer side.

Hvordan bliver exceptions håndteret:

Vi bruger try-catch blokke til at håndtere vores exceptions. I vores mappers der har vi exceptions som bliver sendt, når noget går galt med at forbinde til databasen, oprette data eller at logge ind. Derefter når mappers bliver brugt i vores controllers, så bruger vi også try-catch blokke til at tjekke om der er sket en exceptions, hvorefter der skal ske en handling. Et eksempel er ved loginmetoden i User Controller (se. figur 5), her fanger vi vores database exceptions, og hvis den har fejlmeddelsen ”Fejl i login. Prøv igen”, så var brugernavn eller kodeordet forkert.

Hvordan bliver routes håndteret:

Routes bliver håndteret inde i Main klassen (se. Figur 6), hvor de alle sammen bliver kaldt igennem addRoutes metoder, og bliver tilføjet til Javalin app’en. Et eksempel på en af routes der bliver kaldt, er login. POST /login routen anvendes, når en POST-anmodning sendes til /login. Den kalder login-metoden og sender Context-objektet og ConnectionPool-instansen med. Routen håndterer brugerlogin-funktionalitet ved at forsøge at godkende brugeren baseret på det angivne brugernavn og adgangskode. Hvis de indtastede data matcher med data i databasen, sætter den currentUser-sessionsattributten og omdirigerer brugeren til buypage.html. Hvis det mislykkes på grund af databasefejl eller den indtastede data ikke matcher dem i databasen, sender den brugeren tilbage til login siden med en fejlmeddelelse. (se. figur 5 & figur 6)

Status på implementation

Dette afsnit skal liste hvor langt man er nået med implementation. Typiske ting man kan have sprunget over.

US-?	Afsnit	Kommentar	Status
US-1:	Krav side. 4	Vi har sørget for at kunden kan bestille og betale cupcakes med en valgfri bund og top.	Færdig.
US-2:	Krav side. 4	Kunden kan oprette en konto/profil for at kunne betale og gemme en ordre.	Færdig
US-3:	Krav side. 4	Vi har sørget for at administratoren kan indsætte beløb på en kundes konto direkte i Postgres, så kunden kan betale for sine ordrer.	Færdig
US-4:	Krav side. 4	Kunden kan se sine valgte ordrelinier, i indkøbskurven. og den samlet pris på sin bestilling.	Færdig
US-5:	Krav side. 4	Vi har brugt brugernavn i stedet for e-mail, til at logge ind med. Desuden så kan man hellere ikke se e-mail/navn på hver side i en top menu.	Næsten færdig.
US-6:	Krav side. 4	Vores administrator kan se ordrerne i systemet. Administratoren kan ikke se selve bestillingens indhold af cupcakes, da den giver fejl.	Halv Færdig.
US-7:	Krav side. 4	Vi er nået at starte på US-7	Ikke begyndt.
US-8:	Krav side. 4	Kunden kan fjerne sine ordrelinier fra indkøbskurven.	Færdig.
US-9:	Krav side. 4	Administratoren kan ikke fjerne ordrerne i systemet. Der var en fejl, der gjorde at den ikke virket.	Næsten færdig.

Proces

Hvad var jeres planer for teamets arbejdsform og projektforløbet?

Planen var at hver udvikler fra teamet arbejder med hver sin userstory, html side og css. Alle udviklerne fokuserer på selve koden i starten, mens en af de fire udviklere begyndte på rapporten inden der blev codefreeze. Alle udviklerne kommer og hjælper til med hver deres input til rapporten. Desuden så var planen at der skal lave mappers og controllers, til de diverse user stories. Når to user stories, som for eksempel login (US-5) og signup (US-2) skal begge bruge et User objekt, så skal mappers og controllers laves i fælles (to og to), for at undgå merge-konflikt, og gøre det nemmere at kode.

Hvordan kom det til at forløbe i praksis?

Forløbet gik præcis efter planen i praksis, med henblik på teamets arbejdsform. Så planen gik meget godt, med at hver person har en user-story som de arbejder på. Vi lavede kun controllers og mappers sammen hvis der var to user-stories der skulle benytte den samme.

Hvad gik godt og hvad kunne have været bedre?

Vi føler i gruppen at vi godt kunne havde brugt rollemodellen fra Belbin, hvor hver især indtager hver deres rolle. Det ville kunne hjælpe med at tage beslutninger, og fordele arbejdet mere effektivt.

Hvad har vi lært af processen og hvad vil vi evt. gøre anderledes næste gang?

Som et team af udviklere har vi lært at vi skal være bedre til at løse de fejl der kan opstå når vi har programmeret noget færdigt. Det kan vi gøre ved at vi afsætter nogle dage der bruges til at løse fejl, så vi sørger for at bruger oplevelsen ikke bliver ødelagt på grund af fejl.

En anden ting er at være enige i starten om hvad vi vil kalde de filer som vi arbejder med i hver vores branch, så sikrer vi at der ikke kommer merge konflikter.

Video demo

Her er link til en video af en demo af vores hjemmeside: https://youtu.be/MaxZKwFryUk

Bilag

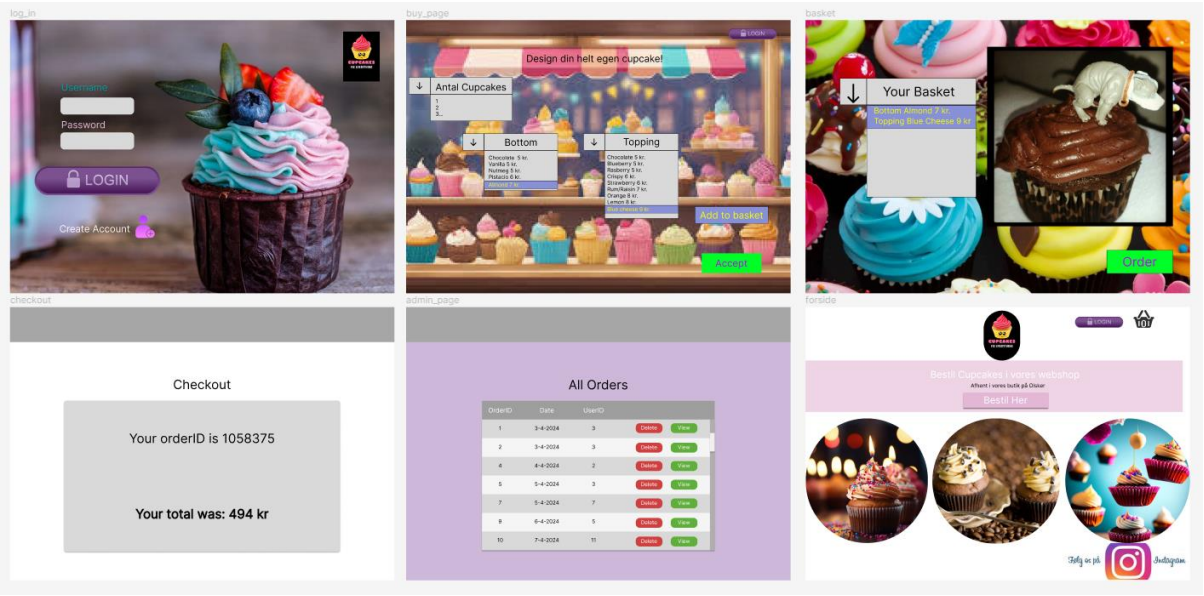
```java
public static void login(Context ctx, ConnectionPool connectionPool)
{
    // Hent form parametre
    String username = ctx.formParam( key: "username");
    String password = ctx.formParam( key: "password");

    // Check om bruger findes i DB med de angivne username + password
    try {
        User user = UserMapper.login(username, password, connectionPool);
        ctx.sessionAttribute("currentUser", user);
        // Hvis ja, send videre til forsiden med login besked
        ctx.render( filePath: "buypage.html");
    }
    catch (DatabaseException e) {
        // Hvis nej, send tilbage til login side med fejl besked
        if (e.getMessage().equals("DB fejl")) {
            ctx.attribute("message", "Can't connect to db");
        }
        if (e.getMessage().equals("Fejl i login. Prøv igen")) {
            ctx.attribute("message", "Wrong username or password");
        }
        ctx.render( filePath: "login.html");
    }
}
```

Figur 5 - UserController: Login

```java
public static void addRoutes(Javalin app, ConnectionPool connectionPool)
    app.post( path: "login", ctx -> login(ctx, connectionPool));
    app.get( path: "login", ctx -> ctx.render( filePath: "login.html"));
    app.get( path: "logout", ctx -> logout(ctx));
    app.get( path: "signup", ctx -> ctx.render( filePath: "signup.html"));
    app.post( path: "signup", ctx -> signup(ctx, connectionPool));
}
```

Figur 6 - UserController: addRoutes



Figur 7 - Figma mockup af hjemmesiden