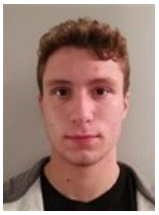
Danmarks Tekniske Universitet

CDIO 2

11 Maj 2020

Lavet af: Gruppe 25

Oliver Poulsen
s185112



Thomas Hohnen
s195455

Mohamad Abdulfatah Ashmar
s176492

Martin Mårtensson
s195469

Andrey Baskakov
s147199

Daniel Styrbæk-Petersen
s143861

Github Repository https://github.com/GRP25/CDIO_2
Production Demo https://mama.sh/

Resumé

The purpose of this projekt is to make a web page to administrate users for a database. There was some different use cases that needed to be made for the project. This involved Create user, Update user, Get user, Get user list and delete user. The main focus of this assignemt was to make the web application correct. This has been done with HTML, CSS and JavaScript. This has been done by building on top of the previous CDIO assignemt.

Before anyone programmed anything there was made som diagrams and talked about how to make the web page. Furthermore, some of the design from CDIO 1 was also used in this assignemt since there was some similarities between them

1 Timeregnskab

Skemaet herunder indeholder gruppens regnskab over timer brugt til hver del af opgaven. Denne er blevet opdateret løbende.

1.1 Timeregnskab

Områder/Navne	Oliver	Thomas	Mohamad	Andrey	Martin	Daniel	I alt
Projektplanlægning	2	2	2	2	2	2	**12**
Rapportskrivning	10	3	3	3	2	2	**23**
Kravspecifikation	1	1	1	1	1	1	**6**
Frontend Programmering	0	15	0	0	3	15	**33**
Funktionalitets Programmering	0	0	10	10	5	0	**25**
Datalag programmering	7	0	0	0	7	0	**14**
Testing	2	2	2	2	2	2	**12**
Gennemlæsning	1	1	1	1	1	1	**6**
Samlet tid i timer	**23**	**24**	**19**	**19**	**23**	**23**	**131**

Tabel 1: Tabel over timeregnskab

Indhold

2 Indledning

Til dette projekt er det blevet lavet et web interface til bruger administration. Til dette er der blevet gjort brug af Java, SQL database, HTML, CSS Javascript og mange flere. Dette er et projekt som er blevet bygget videre fra et tideligere projekt (CDIO 1) hvilket betyder at der er noget kode som er blevet genbrugt. I denne rapport vil det blive oplyst hvordan dette projeket er blevet løst, og hvad tænkerne bag det har været.

3 Krav

3.1 Kravspecifikation

- Der skal udvikles et administrationsmodul som skal kunne tilgås via en web browser
- Alle med adgang til web siden kan oprette alle typer brugere
- System skal understøtte 4 brugertyper hendholdsvis **Admin**, **Pharmaceut**, **Produktionsleder** og **Laborant**.

3.2 Tekniske Krav

- Systemet skal være en *Single page application* med en REST-backend og HTML, CSS og JavaScript fontend.
- Data'en skal lagres i en persistent database mellem sessioner.
- Kommunikationen med REST-backenden skal være serialiseret, så andre kan anvende services.
- Datalaget skal være afkoblet med et interface *IUserDAO*, der indeholder definitionen af de nødvendige metoder. Datalaget skal kunne genbruges selvom der skiftes væk fra REST og Jersey.
- Datalaget skal implementeres som et adskilt lager, som en database i MySQL.

Oplysningerne om brugerne med gyldige og obligatoriske felter for input er defineret i nedenstående data transfer object (DTO):

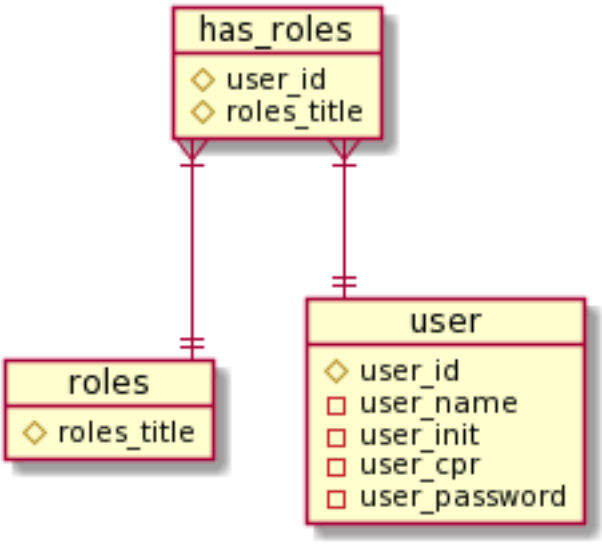
```
1  public class UserDTO {
2      int userId;
3      String userName;
4      String ini;
5      String cpr;
6      String password;
7      List<String> roles;
8  }
```

4 Design

4.1 Database EER diagram

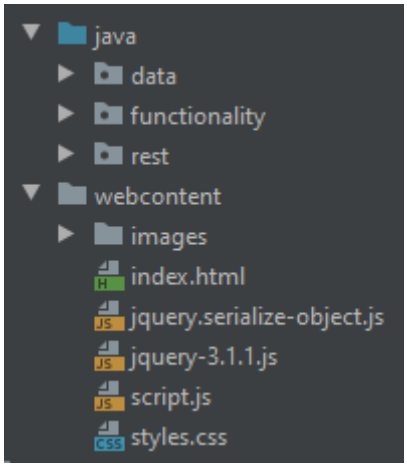
4.2 Struktur over programmet

Til dette projekt er der blevet taget brug af 3-lagsmodellen. Dette er en model som sikre at det kun er de nødvendige metoder der snakker sammen. Altså i bunden er der et datalag som håndtere rå data fra SQL databasen og sender den videre til funktionalitet laget.



Figur 1: EER digram over database anvendt til system

Dette er det lag som skal håndtere det meste funktionalitetter ved systemet. Til sidst er brugergrænsefladen, som i dette projekt er web-applikationen. Under web-applikationen er der HTML, CSS og javascript som er til at programmere hjemmesiden. I figur 2 kan

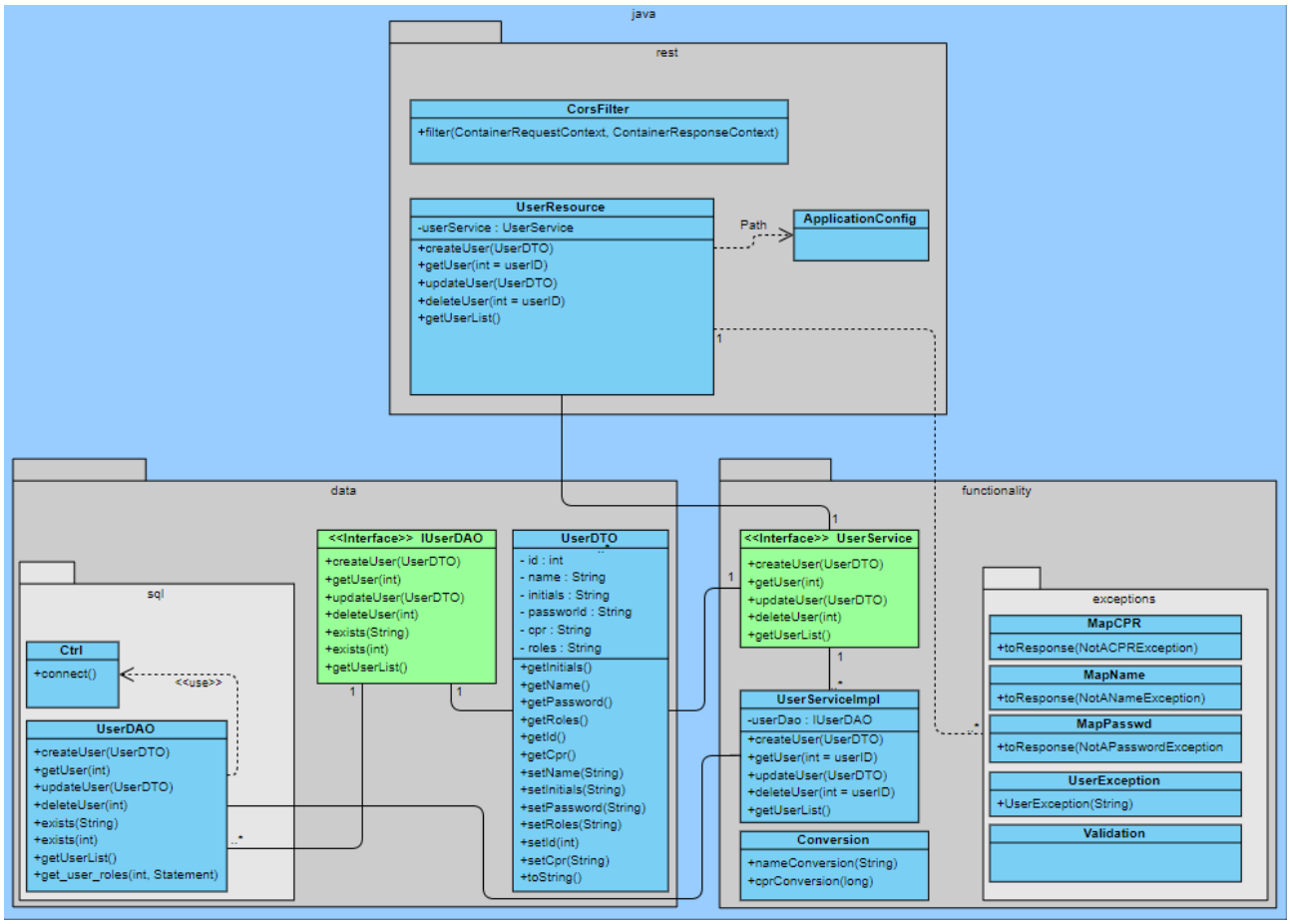


Figur 2: Opbygning af program

man se visuel hvordan programmet er bygget op i de 3 forskellige lag.

4.3 Klassediagram

I klassediagrammet viser vi hvordan backend af projektet er sat op, og hvordan klasserne interagerer med hinanden. Vi har fået lavet en java package som indeholder data, functionality og rest.



Figur 3: Klasse diagram

5 Implementering

5.1 Datalag og interface'et

Datalaget til det udviklede system til håndtering af brugerprofiler, implementeres med *IUserDAO* interface'et, som indeholder metode deklarationerne som systemet bruger til initialisering og opdatering af data'en.

```
1   public interface IUserDAO {
2       UserDTO getUser(int userID);
3       void updateUser(UserDTO);
4       void createUser(UserDTO);
5       void deleteUser(int userID);
6       ArrayList<UserDTO> getUserList();
7   }
```

I system er blevet implementeret en *MySQL* database, som står for opbevaring af brugerprofilerne. Systemet får adgang til databasen med *UserDAO* klassen i *SQL* package'en, som er baseret på *IUserDAO* interface. Klassen har ansvaret for at etablere forbindelsen og kommunikationen mellem systemet og database serveren.

Kommunikationen til serveren foregår ved at sende SQL kommander, og afhængig af funktionen, vil tabellen som bliver retuneret blive konverteret til et UserDTO objekt. Nedenstående funktion viser *getUser*, som henter brugeren fra serveren med det indtastede ID. *String* objektet *sql*, på linje 3, indeholder SQL komandoen som sendes til severen, hvori *"?"*erstattes med *ID*'s værdi på linje 9. *ResultSet st* vil modtage tabellen som sendes retur fra serveren fra *executeQuery*. Fra linje 11 til 18 læses resultatet fra databasen, hvor data'en omdannet til et UserDTO objekt.

```
1   public UserDTO getUser(int ID) {
2       UserDTO user = new UserDTO();
3       String sql = "SELECT * FROM user WHERE user_id = ?;";
4
5       try
6        {
7           Connection conn = connect();
8           PreparedStatement pstmt = conn.prepareStatement(sql);
9           pstmt.setInt(1, ID);
10          ResultSet rs = pstmt.executeQuery();
11           if(rs.next()) {
12               user.setId(ID);
13               user.setName(rs.getString("user_name"));
14               user.setInitials(rs.getString("user_init"));
15               user.setCpr(rs.getString("user_cpr"));
16               user.setPassword(rs.getString("user_password"));
17               user.setRoles(get_user_roles(ID, pstmt));
18           }
19          rs.close();
20          pstmt.close();
21          conn.close();
22      } catch (SQLException e) {
23          System.out.println(e.getMessage());
24      }
```

```
25          return user;
26      }
```

Hele operationen foregår i en *Try-Catch* statement, for at fange evtuelle *SQLExeptions* der kunne opstår under forløbet. Ved oprettelsen af en bruger kunne en fejl for eksempel være en fejl indsendte oplysninger, som serveren ikke ville acceptere.

5.2 JSON

Kommunikationen mellem API'en på serveren og præsentationslaget på clienten, ved data'en er formateret i JSON tekst formatet. Formatet gør det simpel at overføre data og objekter mellem klient og server.

5.3 Funktionalitetslag

Funktionalitetslaget er udviklet med REST api structuren. Den udviklede API'en til oplysningerne om brugerprofilerne bliver tilgået ved */users/*, hvorefter et ID bliver indtastet. Med brugen af HTTP metoderne *POST*, *GET*, *PUT* og *DELETE*, kan klienten tilgå API'en, og de ønskede metoder.

Den nedenstående kodeudklip viser GET metoden for en bruger. For at tilgå metoden, sender klienten en HTTP GET anmodning */users/userID*, hvori man i *userID* definere brugerprofilens ID. I toppen af klassen har man fortalk systemet at linket metoderne tilgåes ved er */users*. I metoden som vises, er på første linje defineret at metoden tilgåes via en GET (*@GET*). Dertil kan man i addressefeltet skrive en parameter som kan bruges i funktionen. Metoden returnere derefter et objekt i JSON, som sendes klienten.

```
1       @GET
2       @Path("/{user_id}")
3       public Response get(@PathParam("user_id") int userID) {
4           UserDTO user = userDAO.getUser(userID);
5           return Response.ok(user).build();
6       }
```

Ligeledes fungere resten af metoderne.

5.4 Frontend/præsentationslag

Præsentationslaget er en *Single page application* udviklet med HTML, CSS og JavaScript teknologierne. Brugerfladen giver mulighed for at oprette, opdater, fjerne og vise informationerne omkring de tilknyttede brugere i systemet.

I menuen i venstre side af skærmen kan man vælge en af funktionerne. Valget i menu'en bliver vist på siden, med brugen af JavaScript til at ændre display værdien fra *display: none* til standard værdien.

5.5 AJAX

Ajax er en funktion i Jquery, som tillader javascript at lave asynkrone HTTP requests, og vedligeholde Jquery syntaxen. Asynkrone kald, tillader programmet at køre uden umiddelbart svar fra metoden, så programmet kan eksekvere anden kode, imens at metoden

får svar fra en ekstern kilde. Jquery gør det unødvendigt at gøre sig brug af den indbyggede javascripts syntax for asynkronitet. Fx callback, Promises og det nye async await. Det som ajax er blevet brugt til i dette projekt, er at lave asynkrone kald til API'en fra frontend (præsentationslaget), i dette projekt forventes der maximalt én bruger af gangen, men i at større projekt der skulle kunne servicere flere klienter af gangen, er asynkronitet nødvendigt for at flere brugere ikke skal vente på hinandens svar før de kan få serveret den data der kommer fra deres requests. i figur 4 kan man se hvordan ajax

```
function createUser() {
    document.getElementById( elementId: "loaderID").style.display = "block";

    event.preventDefault();
    const user = {
        cpr: $("#cpr").val(),
        name: $("#c_name").val(),
        initials: $("#c_initials").val(),
        password: $("#c_userPassword").val(),
        roles: $("#role").val(),
    };
    $.ajax( url: {
        url: "https://api.mama.sh",
        method: "POST",
        data: JSON.stringify(user),
        contentType: "application/json",
        success: function (response) {
            /*
            Hide container -
            empty result container
            input response into result container
            Show result container
             */
            document.getElementById( elementId: "loaderID").style.display = "none";
            $(".createUser").hide();
            $(".showResult").html( value: `<p> User ${user.name} added </p>`);
            $(".showResult").show();
        },
        error: function (jqXHR, text, error) {
            document.getElementById( elementId: "loaderID").style.display = "none";
            alert(jqXHR.status + text + error);
        },
    });
```

Figur 4: Ajax

er blevet implementeret i dette projekt. Til dette er der blevet lavet en funktion til hver use case altså opret bruger, update bruger osv. I denne metode kan man se ajax delen og præcis hvad den gør. Den bliver giver API'en/URL'en, hvilken type af metode der skal udføres og at den skal tage imod et JSON objekt. Derefter kan man skrive hvad succes

scenarioet er og hvad error scenarioet er.

5.6 HTML

Når der tales om fronted til en hjemmeside er det helt grundliggende at have HTML på plads. Til dette projekt er der blevet gjort brug af HTML til at sætte hjemmesiden til databasen op. Dette er blevet gjort ved inputs og formats til at hjælpe med CSS. HTML'en er brygget op af forskellige "containers"som holder forskellig information. f.eks. er der en container til at lave en bruger. Dette har gjort det en del nemmere at kunne anvende singel page application, da disse containers kan skjules meget nemt. Da der er blevet gjort brug af containers er der også blevet gjort meget brug af klasser. Måden dette er blevet gjort i denne opgave er ved at give mange forskellige div's og containers den samme klasse, hvilket har gjort det nemmere at arbejde med det i CSS som gør at hjemmesiden bliver konsekvent.

5.7 CSS

Der er blevet brugt en del CSS til dette projekt. Dette har været til formålet at få hjemmesiden til at se ordenlig ud men også for at gøre den mere overskuelig. Måden der er blevet arbejdet med CSS er ved at bruge det forskellige containers og rykke rundt på dem og ændre størrelse osv. Dette har resulteret i at hjemmesiden bliver meget nemmere at navigere.

```css
.optionsContainer {
    margin: 20px auto auto auto;
    border: 1px solid black;
    width: 300px;
    height: 90%;
    position: absolute;
    left: 0;
    top: 9%;
    -ms-transform: translate(-50%, -50%);
    display: flex;
    flex-direction: column;
}
```

Figur 5: optionContainer

I figur 5 kan man se et eksempel på hvordan disse containers er blevet skrevet i CSS.

5.8 REST

I vores rest har vi lavet en klasse UserResource, som har forbindelse mellem Userinterface og functionality. I klassen er der blevet lavet 5 metoder, som gør brug af 4 af de http requests som har været en del af pensum, GET, POST, PUT og DELETE. Når der bliver hentet data fra serveren til userinterfacet, så bliver der brugt Produces annotation som konverterer data til JSON. Når data bliver sendt til serveren via userinterfacet så bruger vi Consumes annotation som konverterer JSON til brugbart data.

```
@Path("UserResource")
public class UserResource {
    UserService userService = new UserServiceImpl();

    @Path("/{userID}")
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public UserDTO getUser(@PathParam("userID") int userID) {

        return userService.getUser(userID);
    }
    @PUT
    @Consumes(MediaType.APPLICATION_JSON)
    public void updateUser(UserDTO user) throws UserException {
        validateUser(user);
        userService.updateUser(user);
    }
```

Figur 6: Eksempel på GET, PUT og Produces, Consumes

6 Test

REST: Backend er blevet testet succesfuldt ved hjælp af at sætte en tomcat server op i intellij som så bliver kørt. I POSTMAN, har vi testet alle de requests typer som er blevet implementeret i vores program, altså GET, POST, PUT og DELETE. Når vi har brugt GET request fik vi alle vores users fra serveren i JSON form, vi har også gjort brug af POST til at skabe nye brugere skrevet som JSON objekter og succesfuldt gemt på serveren. PUT request er blevet testet ved at updatere en user i JSON format som så blev succesfuldt gemt. Med DELETE request har vi slettet de korrekte brugere identificeret ved deres id.

Test af forbindelse til API'ens endpoints fra clientside, er blevet testet ved at bruge Intellij's hjemmeside fremviser, eller vs codes extention live server. Til at køre hjemmesiden lokalt, da vores database ligger hostet på en server har vi på den måde adgang til den selv lokalt. Da har vi så kunne teste hvorvidt om vores knapper var forbundet til API'en korrekt, og den nødvendige data blev sendt videre til databasen.

7 Projektplanlægning

Gennem dette projekt har næsten alt planlægning forgået over discord hvor der har kunne været opdeling af de forskellige dele af projektet. Da de forskellige opgaver var blevet delt ud blev det skrevet op på Github i en README fil.

CDIO_2

Hvem skal lave hvad til næste gang

- Thomas og Daniel kigger på Frontend (HTML/CSS/JavaScript)
- Mohamed og Andrey kigger på Backend Funktionslaget(Tomcat)
- Oliver og Martin kigger på Datalaget(JavaFilerne)

Datalag (Oliver og Martin)

- Database: 3 tabeller: user, roles, has_roles.
- Ændre i UserDAO til den nye database.

Specifikke opgaver

- Sæt postgreSQL database op
- Ctrl.java connection skal sættes op til serveren

Figur 7: README

I figur 7 kan man se hvem der fik de forskellige opgaver og præcis hvad de skulle arbejde på. Den store udfordring ved dette projekt har været at alt planlægning har skulle forgå virtuelt. Dette har derfor resulteret i at der skulle snakkes meget sammen over discord omkring alt planlægning. Dette har både haft sine fordele og ulemper. Fordelen har været at det har været nemmere at komme i kontakt sammen en ulempe med det har dog været når der skulle vises noget visuelt f.eks. på et white board eller lignede.

Noget som kunne blive forbedret til næste gang har været brugen af projectboards. Dette ville have resulteret i en mere præcis planlægning og måske også mindre møder mellem

gruppe medlemmerne.

8 Konklusion

Der er til dette projekt succesfuldt blevet lavet en web-applikation til bruger administration. Der er gennem projektet blevet uddelegeret opgaver til alle de forskellige gruppe medlemmer. Det kan også konkluderes at alle de krav der blev stillet i opgave formuleringen er blevet opfyldt i dette projekt. Dog vil der i alle projekter være ting der kunne havde været gjort anderledes og dette projekt er ikke en undtagelse. I dette projekt kunne der havde været bedre kommunikation mellem de forskellige gruppe medlemmer da der var nogle ting som kunne havde været blevet gjort meget nemmere havde det været sagen. En meget nem måde at løse dette på ville nok have været brugen af projectsboard i stedet for at aftale inddelingerne i de 3 lag. Dog har gruppen gjort rigtig god brug af branches i arbejdes processen for at undgå merge konflikter og lignede.

Figurer

Tabeller

9 Bilag