CDIO 1

CDIO-1 Opgave Feb 2017

Projektnavn: CDIO1
Gruppe: 17
Afleveringsfrist: Lørdag d. 25. Februar 2017

 s147153, Köse, Duran

 s165245, Hjort-Trærup, Carl-Emil

 s165235, Ghodrat, S.Ali

 s153558, Nybroe Pascal, Oliver

 s165244, Boran, Taygun

 s146711, Nicolajsen, Thomas

Denne rapport er afleveret via Campusnet (der skrives ikke under).
Denne rapport indeholder 15 sider ekskl. forside og bilag.

Danmarks Tekniske Universitet
DTU Compute
02324 Videregående programmering
Vejleder: Stig Høgh, Finn Gustafsson

1 Abstract

This report is the the starting point of CDIO projects which covers Conceive, Design, Implement and Operate through analysis of the project by creation of a domain model and use cases by design UML Sequence Diagrams, UML Class Diagrams with respect to GRASP principles.

In this sub task we should write a user administration module with a simple text-based user interface (TUI). You will see an overview of our first project CDIO-1 which shows the process of a developing a fully functional piece of software from the beginning to the top.

2 Timeregnskab

Tabel 1: Deltager Timeregnskab

Navn	I alt per deltager
Köse, Duran	85,5
Hjort-Trærup, Carl-Emil	85,5
Nybroe Pascal, Oliver	85,5
Ghodrat, S.Ali	85,5
Boran, Taygun	85,5
Nicolajsen, Thomas	85,5
I alt for alle deltagere	513

Tabel 2: Opgave Timeregnskab

Opgave	I alt pr. opgave
Design	18
Impl.	228
Test	39
Dok.	64
Andet	16
Ialt	365

Se Bilag 10.2 for en detaljeret oversigt

Indhold

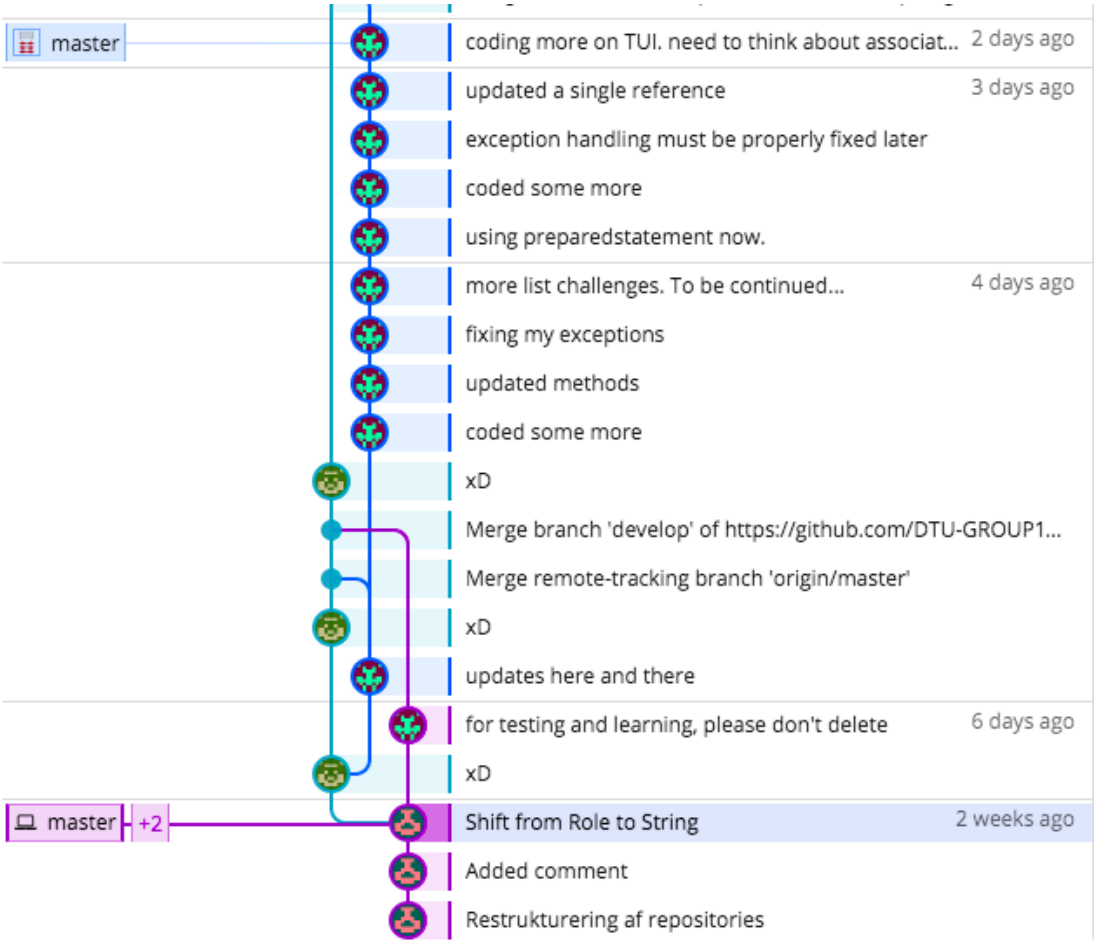
3 Indledning

I denne opgave har vi til formål at udvikle et bruger-administrationsmodul med en simpel tekstbaseret brugergrænseflade (TUI).Vi vil benytte os af 3 lagsmodellen og diverse interfaces for at gøre det nemmere at vedligeholde programmet i de kommende iterationer. Når vi har lavet kravsspecifikation så skal vi designe og kode systemet, bl.a vha. use cases, domæne klassediagram og system sekvensdiagrammer.

4 Metoder

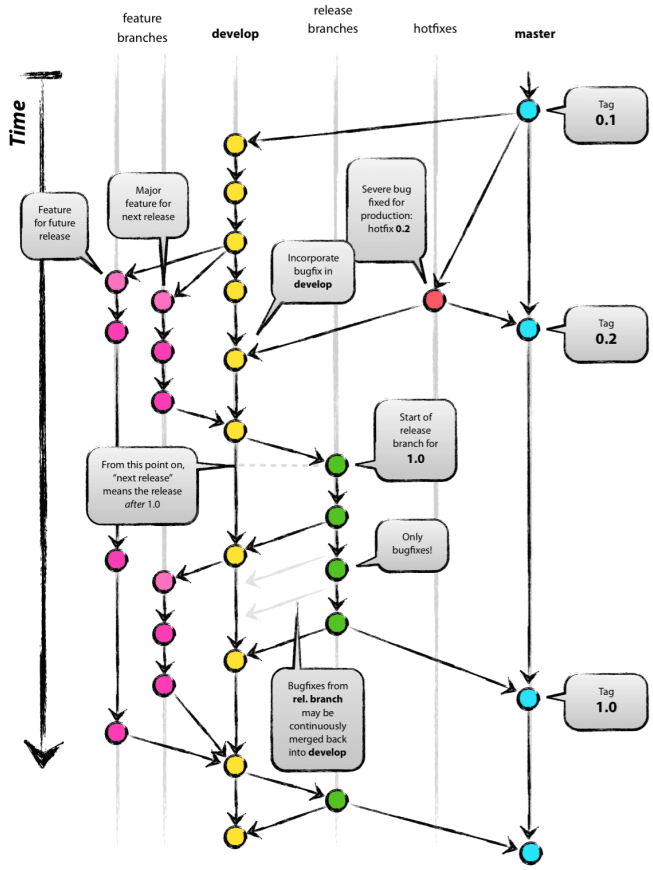
4.1 GIT

Vi valgt at anvende git-flow strategien til udvikling af projektet , til versions- og releasestyring. Git-Flow model er en git forgrening og frigivelse ledelsesstrategi, der hjælper udviklere at holde styr på funktioner, hotfixes og udgivelser i større softwareprojekter det stemmer fra princippet om branch-per-feature.



Figur 1: Git overblik gennem gitkraken GUI

Releases nummereres med versionsnumre startende med 0.1 for første pre-release og 1.0 for første færdige release. Figur 2 viser hvordan et længerevarende projekt udviklet med git-flow kunne se ud. Her kan det ses at der varetages features, releases, hotfixes og tags, som en del af arbejdsgangen.



Figur 2: Git Flow

4.2 FURPS+

Vi har valgt at opstille vores kravspecifikation ud fra FURPS+, hvilket er et godt redskab til at kategorisere og opdele vores krav. FURPS+ giver et lidt andet syn på de opstillede krav, og ikke mindst kundens vision. FURPS+ er altså en måde hvorpå vi yderligere kan overveje og gennemtænke aspekter af kravene og kundens vision, og udfra dette prøve så vidt muligt at optimere det system som vi har tænkt os at udarbejde.

Oversigt over FURPS+

- Functional *(features, capabilities and security)*
- Usability *(human factors, help and documentation)*
- Reliability *(frequency of failure, recoverability and predictability)*
- Performance *(response times, throughput. accuracy. availability amd resource usage*
- Supportability *(adaptability, maintainability, internationalization and configureability)*

- Implementation *(ressource limitations, languages and tools, hardware...)*
- Interface *(constraints imposed by interfacing with external systems)*
- Operations *(system management in its operational setting)*
- Packaging *(for example, a physical box)*
- Legal *(licensing and so forth)*

Tabel 3: FURPS+,
Applying UML and Patterns (3rd edition), af Craig Larman, kapitel 5.4

5 Teori

Vores mål er at opbygge programmet ved at benytte os af 3 lagsmodellen. Vi er i den overbevisning at det vil gavne os i de fremtidige videreudviklinger. Hvert lag har et uafhængigt ansvarsområde. Udfordringen er at skabe kontakt mellem lagene således, at det kun er de relevante klasser der taler med hinanden. Alt input og output fra og til brugeren har vi smidt over i grænsefladelaget, mens programmets selve logik foregår i funktionslaget. Datalaget indeholder vores kontakt til data. Det er her vi gemmer vores brugere i memory, tekstfil eller database. Vi kan kategorisere lagene således:

- Grænsefladelaget
  - UI
  - TUI
- Funktionslaget
  - App
  - Starter
  - UserController
  - Action
  - CreateUser
  - DeleteUser
  - UpdateUser
  - ViewUser
- Datalaget
  - DAO
  - UserDAO
  - InMemoryDAO
  - InMemoryUserDAO
  - JDBCDAO
  - JDBCUserDAO
  - Model
  - User
  - DALException
  - NotConnectedException
  - NotFoundException

6 Analyse

6.1 Kravsspecifikation

Functional

R.1 Bruger-administrationsmodulet skal have en simpel tekstbaseret brugergrænseflade (TUI).

R.2 Bruger-administrationsmodulet skal benyttes til at administrere brugere og operatøre.

- R.2.1 Det skal være muligt at oprette en bruger.
- R.2.2 Det skal være muligt at vise brugere.
- R.2.3 Det skal være muligt at opdatere brugere.
- R.2.4 Det skal være muligt at slette brugere.

R.3 Brugergrænsefladen skal have en hovedmenu der indeholder mulighederne.

- R.3.1 Opret ny bruger.
- R.3.2 List bruger.
- R.3.3 Ret bruger.
- R.3.4 Slet bruger.
- R.3.5 Afslut program.

R.4 Programmet skal designes efter principperne i 3-lags modellen.

R.5 Der skal være forskellige brugertyper (roller).

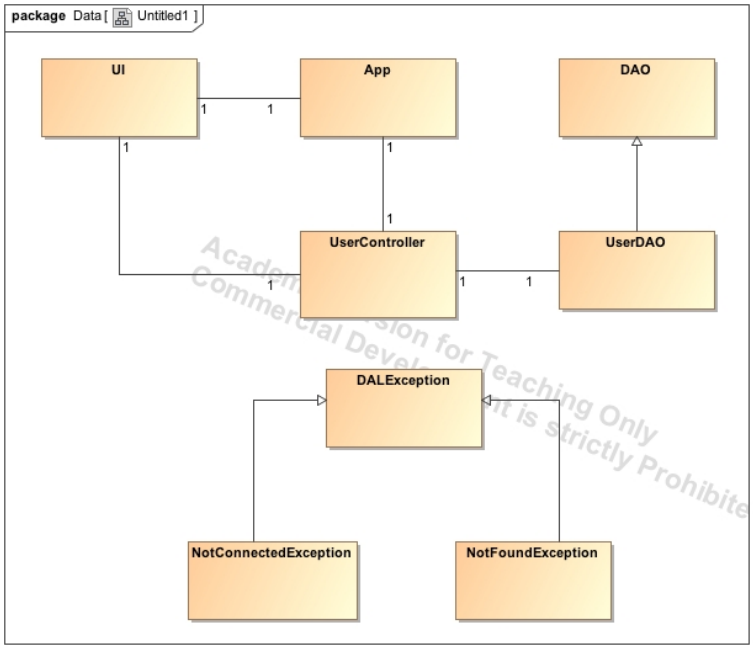
R.6 Datalaget forventes implementeret med et 'persistent storage'.

6.2 Navneordsanalyse

1. User
2. TUI
3. JDBC
4. Query - Builder
5. Bruger-administrationsmodul
6. Brugere/operatører

Tabel 4: Navneordsanalyse

6.3 Domænemodel



Figur 3: Domænemodel

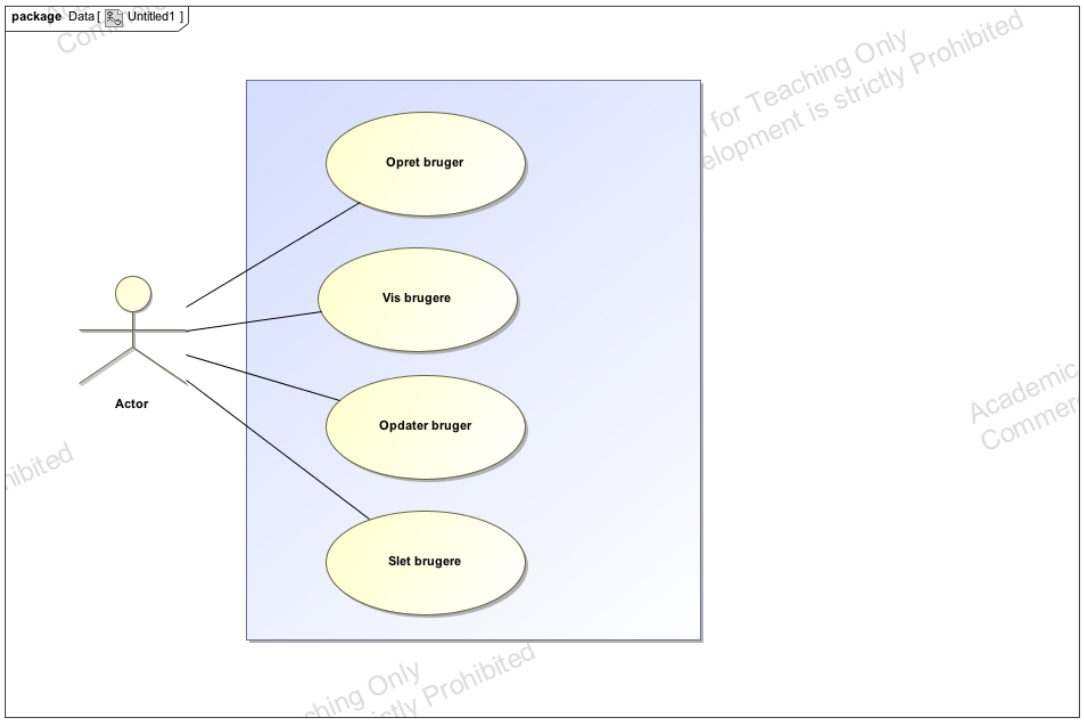
Diagrammet viser en overordnet illustration over vores Operatør-administration. Vi har prøvet på, at opnå så lav kobling som muligt. Muliciplicitеterne illustrerer forholdet mellem de forskellige klasser.

6.4 Udsagnsordsanalyse

1. Opret
2. Vis
3. Opdater
4. Slet

Tabel 5: Udsagnsordsanalyse

6.5 Use-case Diagram



Figur 4: Use case diagram

Vores use-cases ses på ovenstående use-case diagram, i alt 4 use-cases lyder som følger:

- Opret bruger
- Vis brugere
- Opdater bruger
- Slet brugere

6.6 Use Case Scenarier

6.6.1 Use-Case 01: OpretBruger

Use Case Name: OpretBruger

ID: 1

Brief description: Operatører skal kunne oprette brugere.

Primary Actor Operatører.

Secondary actors System.

Preconditions

- Programmet skal være startet.

Main Flow

- Brugeren oprettes med et unikt id fra 11 og op efter, et brugernavn (tekststræng) på 2-20 tegn, initialer (tekststreng) på 2-4 tegn, cpr (tekststreng) på 10 tegn, password (tekststreng) på 255 tegn og roller der vælges blandt (Admin, Pharmacist, Foreman, Operator).

Alternative Flow

- Bruger har indtastet for kort/ langt brugernavn/ password.
  - Bruger bliver bedt om at indtaste nyt antal.
- Bruger har indtastet ugyldige symboler.
  - Bruger bliver bedt om at indtaste nyt brugernavn.
- Bruger har indtastet fornavn, efternavn, eller brugerID i passwordet.
  - Bruger bliver bedt om at indtaste nyt password.

Postconditions

- Bruger er oprettet.

6.6.2 Use-Case 02: Vis brugere

Use Case Name: Vis brugere

ID: 2

Brief description: Brugerne vises.

Primary Actor Operatør.

Secondary actors System.

Preconditions

- Programmet skal være startet, og brugere skal være oprettet.

Main Flow

- Systemet tager alle brugere og viser dem på skærmen.

Alternative Flows

- Systemet kan ikke tilgå data.

Postcondition

- Brugerne kan ses på skærmen.

6.6.3 Use-Case 03: Opdatér bruger

Use Case Name: Opdatér bruger

ID: 3

Brief description: Operatør opdaterer brugerens attributter.

Primary Actor Operatør.

Secondary actors System.

Preconditions

- Programmet skal være startet, og bruger skal være oprettet.

Main Flow

- Operatør foretager ændringer på attributterne af brugeren.
- Systemet lægger ændringerne i data'en.

Alternative Flows

- Bruger har indtastet ugyldig ændring.
  - Bruger bliver bedt om at indtaste ny gyldig ændring.

Postcondition

- Brugeren er opdateret.

6.6.4 Use-Case 04: Slet brugere

Use Case Name: Slet brugere

ID: 4

Brief description: Operatøren sletter brugeren.

Primary Actor Operatør.

Secondary actors System.

Preconditions

- Programmet skal være startet, og brugere skal være oprettet.

Main Flow

- Brugerne slettes.

Alternative Flows

- Brugeren findes ikke og kan ikke slettes.

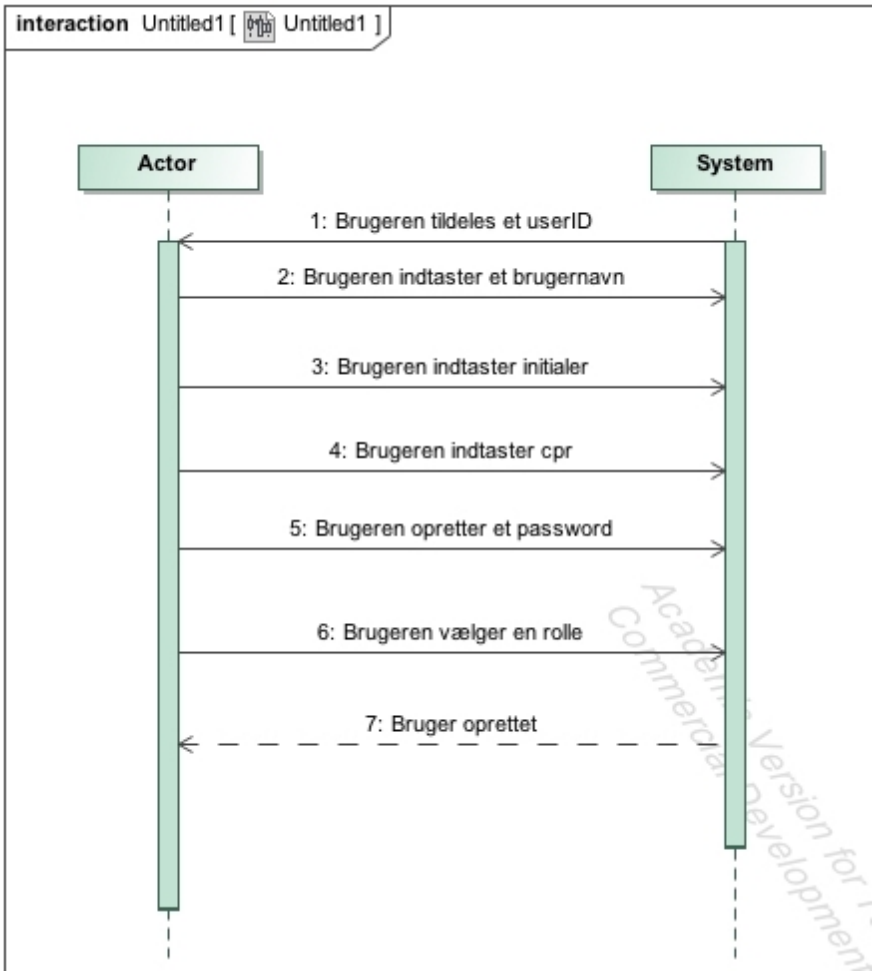
Postcondition

- Brugerne er slettet.

6.7 System Sekvensdiagram

6.7.1 System Sekvensdiagram: Opret bruger

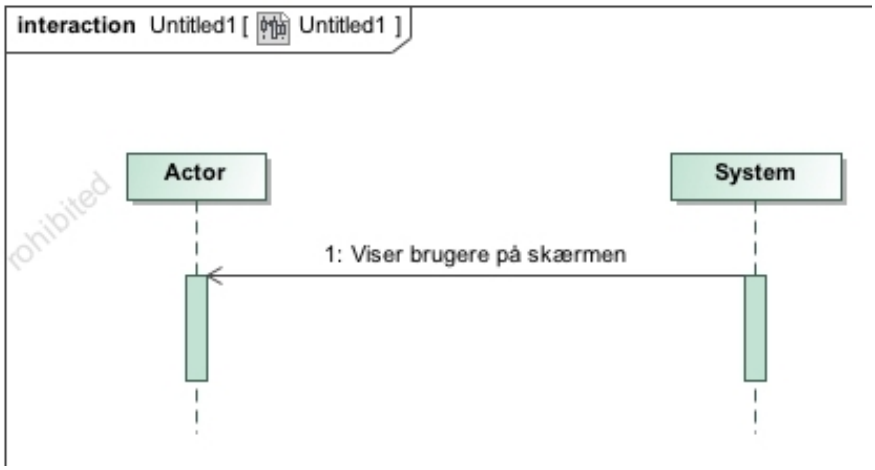
I nedenstående diagram vises hvordan mainflowet for Opret bruger use-casen bliver udført. Det fremgår af diagrammet at systemet automatisk generere et userID til brugeren, hvorefter brugeren indtaster sine oplysninger.



Figur 5: System Sekvens Diagram til Afsnit 6.6.1

6.7.2 System Sekvensdiagram: Vis brugere

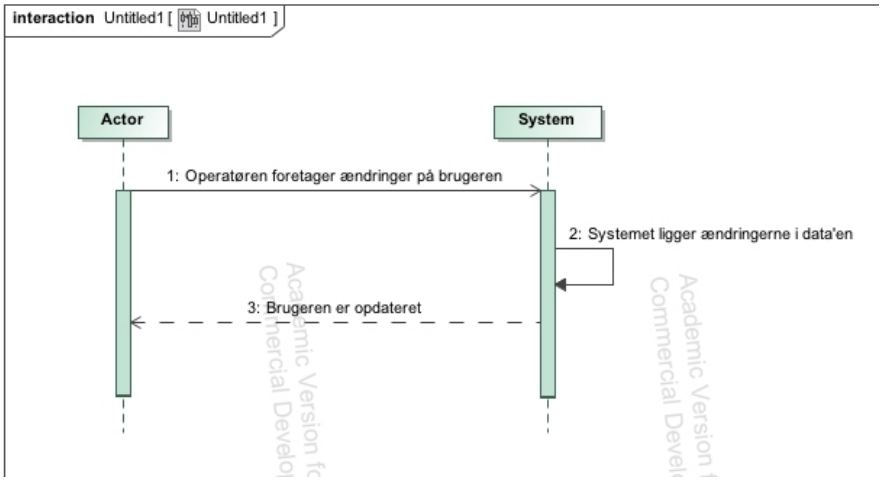
I nedenstående diagram er der vist, hvordan mainflowet for Vis brugere use-casen bliver udført.



Figur 6: System Sekvens Diagram til Afsnit 6.6.2

6.7.3 System Sekvensdiagram: Opdatér bruger

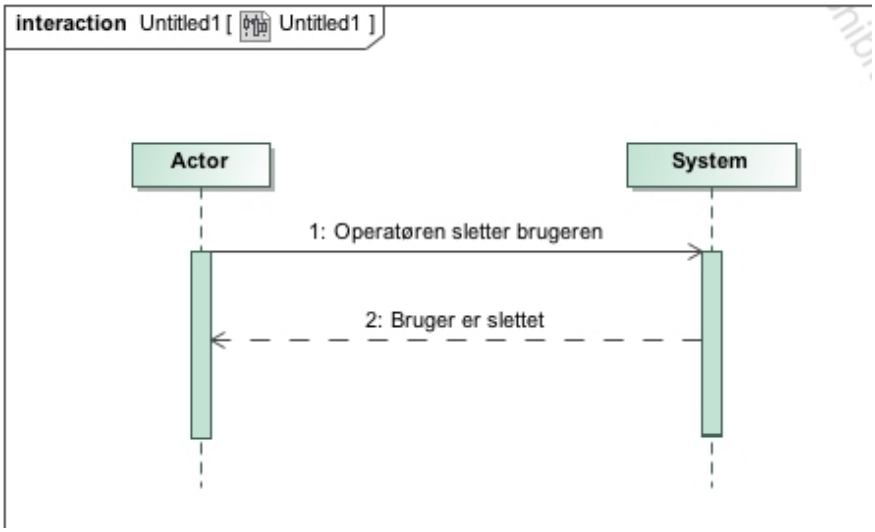
I nedenstående diagram er der vist, hvordan mainflowet for Opdatér bruger use-casen bliver udført. Brugeren opdaterer sine oplysninger. Dernæst lagrer systemet ændringerne i data'en.



Figur 7: System Sekvens Diagram til Afsnit 6.6.3

6.7.4 System Sekvensdiagram: Slet brugere

I nedenstående diagram er der vist, hvordan mainflowet for Slet bruger use-casen bliver udført.



Figur 8: System Sekvens Diagram til Afsnit 6.6.4

6.8 Requirement Tracing

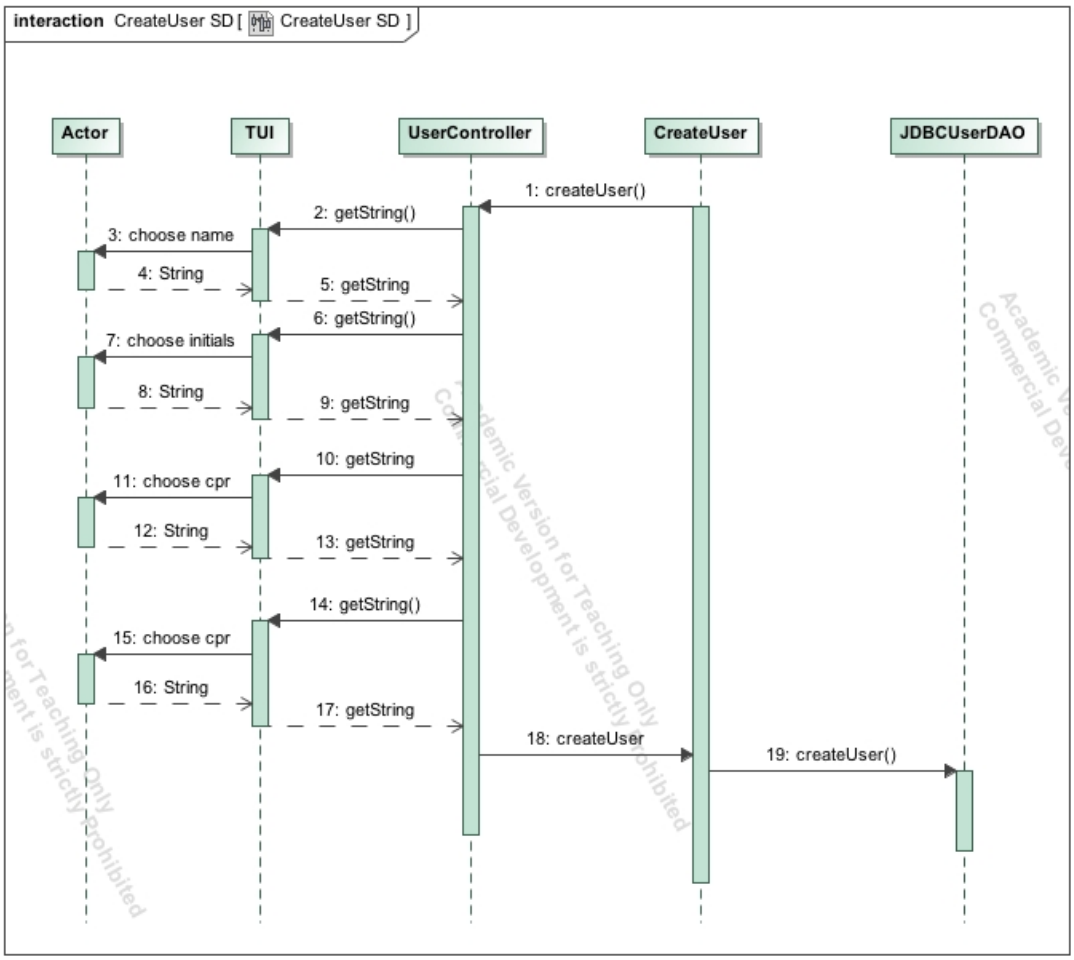
	Use cases			
Requirements	U1	U2	U3	U4
R.1	■			
R.2	■			
R.2.1	■			
R.2.2		■		
R.2.3			■	
R.2.4				■
R.3	■			■
R.3.1	■			
R.3.2		■	■	
R.3.3			■	
R.3.4				■
R.3.5	■	■	■	■

Figur 9: Requirement Tracing

Kravene R.4, R.5 og R.6 indgår ikke i modellen, da disse krav er non-funktionelle.

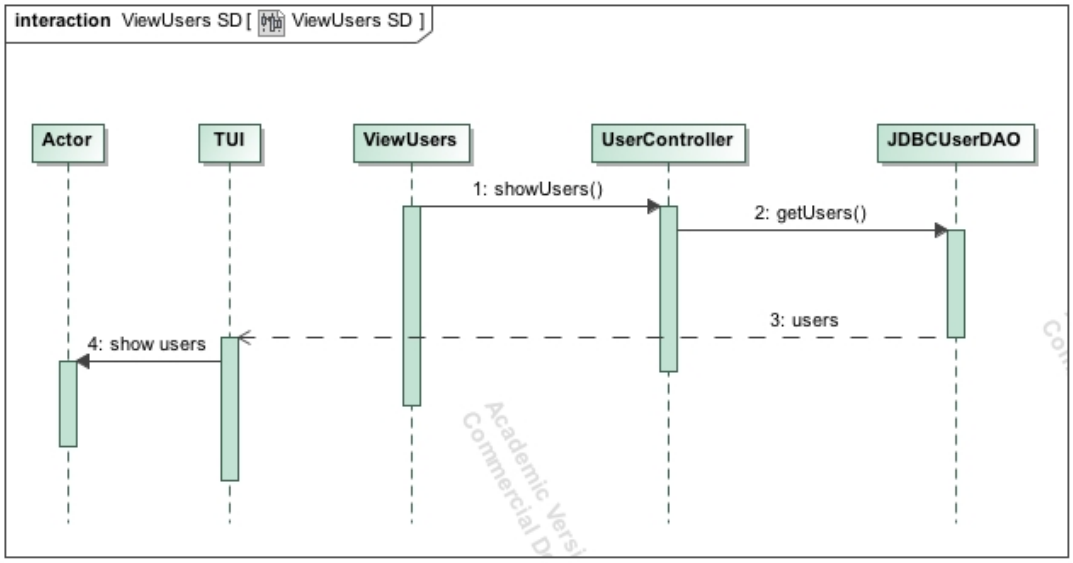
7 Design

7.1 Sekvensdiagrammer



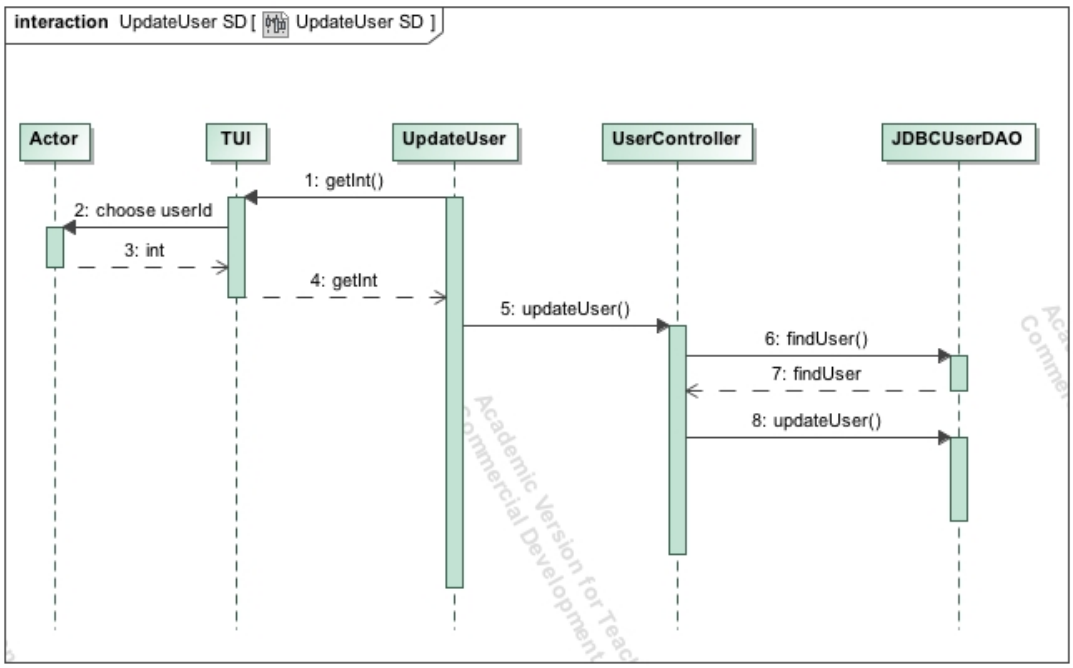
Figur 10: Sekvensdiagram for opret bruger

7.1.1 Sekvensdiagram for vis brugere



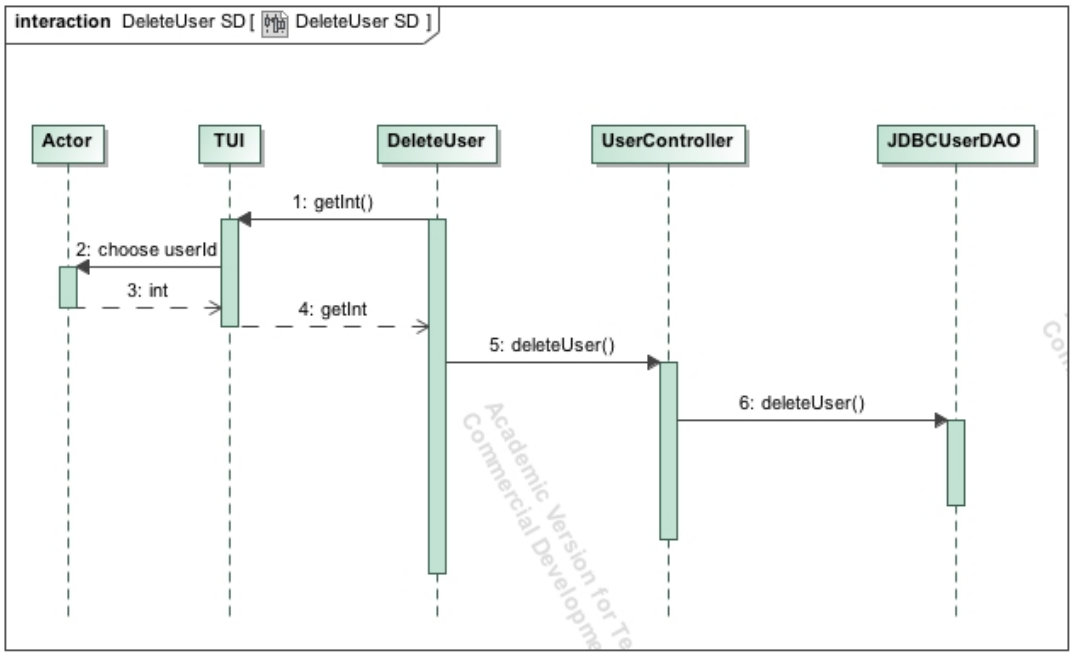
Figur 11: Sekvensdiagram for vis brugere

7.1.2 Sekvensdiagram for opdatér bruger



Figur 12: Sekvensdiagram for vis brugere

7.1.3 Sekvensdiagram for slet brugere



Figur 13: Sekvensdiagram for vis brugere

8 Implementering

Vi har i vores program brugt interfaces og abstrakte klasser, til at illustrere generalisering og specialisering.

Programmet er delt op i 9 pakker.

- app
  - **App.java**
    Den står for ansvaret til valget af action.
  - **Starter.java**
    Den definerer hvilke 3 lag der skal bruges i 3 lagsmodellen.
- app.actions
  - **Action.java**
    Abstrakt klasse som definerer handlinger som brugeren kan foretage.
  - **CreateUser.java**
    Handlingen til at oprette bruger.
  - **DeleteUser.java**
    Handlingen til at slette bruger.
  - **UpdateUser.java**
    Handlingen til at opdatere bruger.

  - **ViewUsers.java**
    Handlingen til at vise brugere.

- controllers
  - **UserController.java**
    Den står for logik relevans for brugere.

- dal.contracts
  - **DAO.java**
    Den håndterer data access.
  - **UserDAO.java**
    Et object som accesser data om users.

- dal.exceptions
  - **DALException.java**
    Exceptions relateret til datalaget.
  - **NotConnectedException.java**
    Exceptions relateret til når man ikke er forbundet.
  - **NotFoundException.java**
    Exceptions relateret til når data ikke er fundet.

- dal.jdbcdao
  - **JDBCDAO.java**
    Den håndterer data access til JDBC.
  - **JDBCUserDAO.java**
    Et object som accesser data om users gennem JDBC.

- factories
  - **Factory.java**
    Abstract klasse der bruges til at lave DTO'ere.
  - **UserFactory.java**
    Bruges til at lave user DTO'ere.

- models
  - **InvalidInputException.java**
    Exceptions relateret til når bruger inputter forkert data.
  - **Model.java**
    Objekter der repræsenterer data fra vores database.
  - **User.java**
    Objekter der repræsenterer data om users.

- ui
  - **UI.java**
    Interface for brugergrænsefladeladet.
  - **TUI.java**
    Tekst implementering af UI.

9 Konklusion

Vi har i vores projekt lavet et brugeradministrationsmodul, hvor man kan oprette, opdatere, vise og slette brugere. Vi har udarbejdet kravspecifikation og forskellige analysediagrammer for programmet. I programmet har vi anvendt interfaces og abstrakte klasser, har ladet os generalisere, og specialisere klasserne i projektet. Vi brugte UP til at gennemløbe iterationer af analyse, design, implementering, og test. Vi har så vidt passende, opdelt klasserne efter 3-lagsmodellen, og inddelt klasserne i lagene UI, Control, Data.

10 Bilag

10.1 Git

10.1.1 Kildekode

```java
package app.actions;

import app.App;

public abstract class Action{

    protected App app;

    public Action(App app){
        this.app = app;
    }

    public abstract void perform();

}

```

```
 1 package app;
 2
 3 import java.util.Hashtable;
14
15 public class App {
16
17     private UI ui;
18     private UserController userController;
19     private Hashtable<String, Action> actions;
20
21     public App(UI ui, DAO dao) {
22         this.ui = ui;
23         this.userController = new
   UserController(
24                 this.ui,
25                 dao
26         );
27         this.actions = new Hashtable<>();
28         this.actions.put("Create user", new
   CreateUser(this));
29         this.actions.put("View users", new
   ViewUsers(this));
30         this.actions.put("Delete user", new
   DeleteUser(this));
31         this.actions.put("Update user", new
   UpdateUser(this));
32         this.actions.put("exit", new
   CloseProgram(this));
33     }
```

```
34
35      private void chooseAction(String
    actionKey){
36          if
    (this.actions.containsKey(actionKey)){
37
    this.actions.get(actionKey).perform();
38          } else {
39              this.getUi().showMessage("No such
    action '" + actionKey + "'");
40          }
41      }
42
43      public void start() {
44          do {
45              this.chooseAction(
46                  this.getUi().getString(
47                      "Choose action: " +
    this.actions.keySet().toString()
48                  )
49              );
50          } while (true);
51      }
52
53      public UI getUi() {
54          return ui;
55      }
56
57      public UserController getUserController()
```

```
58          return userController;
59      }
60  }
61
```

```java
package app.actions;

import app.App;

public class CloseProgram extends Action {

    public CloseProgram(App app) {
        super(app);
    }

    @Override
    public void perform() {
        System.exit(0);
    }

}

```

```java
package app.actions;

import app.App;


public class CreateUser extends Action {
    public CreateUser(App app) {
        super(app);
    }

    @Override
    public void perform() {
this.app.getUserController().createUser();
    }
}

```

```java
package dal.exceptions;

public abstract class DALException extends Exception {
    private Exception e;

    public DALException(){
        super();
    }

    public DALException(String msg) {
        super(msg);
    }

    public DALException(Exception e){
        super(e.getMessage());
        this.e = e;
    }

    @Override
    public void printStackTrace() {
        this.e.printStackTrace();
    }


}
```

```java
package dal.contracts;

/**
 * Contract for all DOA's
 */
public interface DAO {

    /**
     * Gets the DAO for a user
     *
     * @return the chosen userDAO
     */
    UserDAO getUserDAO();

    /**
     * Gets the DAO for a role
     *
     * @return the chosen roleDAO
     */
    RoleDAO getRoleDAO();

}
```

```sql
CREATE DATABASE IF NOT EXISTS Weight;
use Weight;

DROP TABLE IF EXISTS Roles_Users;
DROP TABLE IF EXISTS Users;
DROP TABLE IF EXISTS Roles;

CREATE TABLE Users (
    id INT(11) UNSIGNED
        AUTO_INCREMENT
        NOT NULL
        PRIMARY KEY,
    name VARCHAR(255),
    ini VARCHAR(4),
    cpr VARCHAR(10),
    psswrd VARCHAR(255)
);
ALTER TABLE Users AUTO_INCREMENT=10;

CREATE TABLE Roles (
    id INT(11) UNSIGNED
        AUTO_INCREMENT
        NOT NULL
        PRIMARY KEY,
    name VARCHAR(255) UNIQUE
);
CREATE TABLE Roles_Users (
    user_id INT(11) UNSIGNED
        NOT NULL,
```

```
30      role_id INT(11) UNSIGNED
31          NOT NULL,
32      FOREIGN KEY (user_id) REFERENCES Users(id)
33          ON DELETE CASCADE,
34      FOREIGN KEY (role_id) REFERENCES Roles(id)
35          ON DELETE CASCADE
36  );
37
```

```java
package app.actions;

import app.App;

public class DeleteUser extends Action {
    public DeleteUser(App app) {
        super(app);
    }

    @Override
    public void perform() {

this.app.getUserController().deleteUser(
            this.app.getUi().getInt("Choose
user id")
        );
    }
}

```

```sql
INSERT INTO roles (name) VALUES ("manager");
INSERT INTO roles (name) VALUES ("user");
INSERT INTO roles (name) VALUES ("pharmacist");
INSERT INTO roles (name) VALUES ("foreman");
```

```
1 package models;
2
3 public interface DTO {
4 }
5
```

```java
package factories;

import java.util.LinkedList;

public abstract class Factory {
    static Faker faker = new Faker();
    public static Factory factory;

    @SuppressWarnings("unchecked")
    public <T> T make(int times) throws InvalidInputException {
        if (times == 1) {
            return (T) this.instantiate();
        }
        LinkedList<? extends DTO> data = new LinkedList<>();
        for (int i = 0; i < times; i++)
            data.add(this.instantiate());
        return (T) data;
    }

    public <T> T make() throws InvalidInputException {
        return this.make(1);
    }

    abstract <T extends DTO> T instantiate() throws InvalidInputException;

```

```
31 }
32
```

```
1 package factories.exceptions;
2
3 public class InvalidCprException extends
  InvalidInputException {
4     public InvalidCprException() {
5     }
6
7     public InvalidCprException(String message)
  {
8         super(message);
9     }
10 }
11
```

```
1 package factories.exceptions;
2
3 public class InvalidInitialsException extends
  InvalidInputException {
4     public InvalidInitialsException() {
5     }
6
7     public InvalidInitialsException(String
  message) {
8         super(message);
9     }
10 }
11
```

```java
package factories.exceptions;


public abstract class InvalidInputException extends Exception {

    public InvalidInputException() {
    }

    public InvalidInputException(String message) {
        super(message);
    }
}

```

```
1 package factories.exceptions;
2
3 public class InvalidPasswordException extends
  InvalidInputException {
4     public InvalidPasswordException() {
5     }
6
7     public InvalidPasswordException(String
  message) {
8         super(message);
9     }
10 }
11
```

```
1 package factories.exceptions;
2
3 public class InvalidUsernameException extends
  InvalidInputException {
4     public InvalidUsernameException() {
5     }
6
7     public InvalidUsernameException(String
  message) {
8         super(message);
9     }
10 }
11
```

```
 1 package dal.jdbcdao;
 2
 3 import java.sql.Connection;
11
12 public class JDBCDAO implements DAO {
13
14     private JDBCUserDAO userDAO;
15     private JDBCRoleDAO roleDAO;
16     private Connection connection;
17     private String uri;
18
19     public JDBCDAO(String uri) {
20         this.uri = uri;
21         this.userDAO = new JDBCUserDAO(this);
22         this.roleDAO = new JDBCRoleDAO(this);
23     }
24
25     public Connection getConnection() throws
   NotConnectedException{
26         if (this.connection==null){
27             try {
28
   Class.forName("com.mysql.cj.jdbc.Driver").newI
   nstance();
29                 this.connection =
   DriverManager.getConnection(
30                         "jdbc:mysql://localhost:
   3306/" + this.uri,
31                         "root",
```

```
32                      ""
33                  );
34              } catch (SQLException |
  InstantiationException |
  IllegalAccessException |
  ClassNotFoundException e) {
35                  throw new
  NotConnectedException();
36              }
37          }
38          return this.connection;
39      }
40
41      @Override
42      public UserDAO getUserDAO() {
43          return this.userDAO;
44      }
45
46      @Override
47      public RoleDAO getRoleDAO() {
48          return this.roleDAO;
49      }
50
51  }
52
```

```
 1 package dal.jdbcdao;
 2
 3 import java.sql.ResultSet;
11
12 public class JDBCRoleDAO implements RoleDAO {
13
14     private JDBCDAO parent;
15
16     public JDBCRoleDAO(JDBCDAO parent){
17         this.parent = parent;
18     }
19
20     @Override
21     public List<Role> getRoles() throws
   NotConnectedException {
22         List<Role> roles = new ArrayList<>();
23         try {
24             String query =
25                 "SELECT " +
26                     "id, " +
27                     "name " +
28                 "FROM roles";
29             ResultSet results =
   this.parent.getConnection().prepareStatement(q
   uery).executeQuery();
30             while (results.next()) {
31                 roles.add(
32                     new Role(
33                         results.getString(2),
```

```
34                          results.getInt(1)
35                      )
36                  );
37              }
38          } catch (SQLException e) {
39              throw new NotConnectedException();
40          }
41          return roles;
42      }
43  }
44
```

```
 1 package dal.jdbcdao;
 2
 3 import java.sql.PreparedStatement;
16
17 public class JDBCUserDAO implements UserDAO {
18
19     private JDBCDAO parent;
20
21     public JDBCUserDAO(JDBCDAO parent){
22         this.parent = parent;
23     }
24
25
26     private List<Role> getUserRoles(int
   userID) throws NotFoundException,
   NotConnectedException{
27         try {
28             String query =
29                 "SELECT roles.name " +
30                 "FROM roles_users " +
31                 "INNER JOIN roles " +
32                     "ON roles.id =
   roles_users.role_id " +
33                 "WHERE
   roles_users.user_id=?";
34             PreparedStatement statement =
   this.parent.getConnection().prepareStatement(
   query);
35             statement.setInt(1, userID);
```

```
36              ResultSet results =
    statement.executeQuery();
37              List<Role> roles = new
    ArrayList<>();
38              while (results.next()){
39                  roles.add(new
    Role(results.getString(1)));
40              }
41              return roles;
42          } catch (SQLException e) {
43              throw new
    NotConnectedException();
44          }
45      }
46
47      @Override
48      public User findUser(int userId) throws
    NotFoundException, NotConnectedException {
49          try {
50              String query =
51                  "SELECT " +
52                      "name, " +
53                      "ini, " +
54                      "cpr, " +
55                      "psswrd " +
56                  "FROM users " +
57                  "WHERE id = ?";
58              PreparedStatement statement =
    this.parent.getConnection().prepareStatement(
```

```
query);
59              statement.setInt(1, userId);
60              ResultSet results =
statement.executeQuery();
61              if(!results.first()) {
62                  throw new
NotFoundException();
63              }
64              return new User(
65                  userId,
66                  results.getString(1),
67                  results.getString(2),
68                  results.getString(3),
69                  results.getString(4),
70                  this.getUserRoles(userId)
71              );
72          } catch (SQLException e) {
73              throw new
NotConnectedException();
74          }
75      }
76
77      @Override
78      public List<User> getUsers() throws
NotConnectedException {
79          List<User> users = new ArrayList<>();
80          try {
81              String query =
82                  "SELECT " +
```

```
83                      "id," +
84                      "name," +
85                      "ini," +
86                      "cpr," +
87                      "psswrd " +
88                  "FROM users";
89              ResultSet results =
    this.parent.getConnection().prepareStatement(
    query).executeQuery();
90              while (results.next()) {
91                  int id = results.getInt(1);
92                  users.add(
93                      new User(
94                          id,
95                          results.getString(2),
96                          results.getString(3),
97                          results.getString(4),
98                          results.getString(5),
99                          this.getUserRoles(id)
100                     )
101                 );
102             }
103         } catch (SQLException |
    NotFoundException e) {
104             throw new
    NotConnectedException();
105         }
106         return users;
107     }
```

```
108
109     @Override
110     public void createUser(User user) throws
    NotConnectedException {
111         try {
112             String query =
113                 "INSERT INTO users (" +
114                     "name, " +
115                     "ini, " +
116                     "cpr, " +
117                     "psswrd" +
118                 ") " +
119                 "VALUES (?,?,?,?)";
120
121             PreparedStatement statement =
    this.parent.getConnection().prepareStatement(
    query, Statement.RETURN_GENERATED_KEYS);
122             statement.setString(1,
    user.getName());
123             statement.setString(2,
    user.getInitials());
124             statement.setString(3,
    user.getCpr());
125             statement.setString(4,
    user.getPassword());
126             statement.executeUpdate();
127             ResultSet results =
    statement.getGeneratedKeys();
128             int userId;
```

```
129             if (results.next()){
130                 userId = results.getInt(1);
131             } else {
132                 throw new
    NotConnectedException();
133             }
134             query =
135                 "INSERT " +
136                 "INTO roles_users (user_id,
    role_id) " +
137                 "VALUES";
138             for (int i = 0;
    i<user.getRoles().size(); i++){
139                 query += "(?, (SELECT id FROM
    roles WHERE name=?)),";
140             }
141             query = query.substring(0,
    query.length()-1)+';';
142             statement =
    this.parent.getConnection().prepareStatement(
    query);
143             int counter = 0;
144             for (String role :
    user.getRoles().stream().map(Role::getName).c
    ollect(Collectors.toList())){
145                 statement.setInt(++counter,
    userId);
146                 statement.setString(+
    +counter, role);
```

```
147                 }
148                 statement.execute();
149             } catch (SQLException e) {
150                 throw new
    NotConnectedException();
151             }
152         }
153 
154         @Override
155         public void updateUser(User user) throws
    NotFoundException, NotConnectedException {
156             try {
157                 String updateStatement =
158                     "UPDATE users " +
159                     "SET " +
160                         "name=?, " +
161                         "ini=?, " +
162                         "cpr=?, " +
163                         "psswrd=? " +
164                     "WHERE id=?";
165                 PreparedStatement statement =
    this.parent.getConnection().prepareStatement(
    updateStatement);
166                 statement.setString(1,
    user.getName());
167                 statement.setString(2,
    user.getInitials());
168                 statement.setString(3,
    user.getCpr());
```

```
169             statement.setString(4,
    user.getPassword());
170             statement.setInt(5,
    user.getId());
171             statement.execute();
172             String query =
173                 "DELETE " +
174                 "FROM roles_users " +
175                 "WHERE user_id=?";
176             statement =
    this.parent.getConnection().prepareStatement(
    query);
177             statement.setInt(1,
    user.getId());
178             query =
179                 "INSERT " +
180                 "INTO roles_users (user_id,
    role_id) " +
181                 "VALUES";
182             for (int i = 0;
    i<user.getRoles().size(); i++){
183                 query += "(?, (SELECT id FROM
    roles WHERE name=?)),";
184             }
185             query = query.substring(0,
    query.length()-1)+';';
186             statement =
    this.parent.getConnection().prepareStatement(
    query);
```

```
187             int counter = 0;
188             for (String role :
    user.getRoles().stream().map(Role::getName).c
    ollect(Collectors.toList())){
189                 statement.setInt(++counter,
    user.getId());
190                 statement.setString(+
    +counter, role);
191             }
192             statement.execute();
193 
194         } catch (SQLException e) {
195             throw new
    NotConnectedException();
196         }
197     }
198 
199     @Override
200     public void deleteUser(int userId) throws
    NotFoundException, NotConnectedException{
201         try {
202             String deleteStatement =
203                 "DELETE " +
204                 "FROM users " +
205                 "WHERE id=? ";
206             PreparedStatement statement =
    this.parent.getConnection().prepareStatement(
    deleteStatement);
207             statement.setInt(1, userId);
```

```
208             if(statement.executeUpdate() ==
    0)
209                 throw new
    NotFoundException();
210         }
211         catch (SQLException e) {
212             throw new
    NotConnectedException();
213         }
214     }
215
216 }
217
```

```java
package dal.exceptions;

public class NotConnectedException extends
 DALException {

    public NotConnectedException(){
        super();
    }

    public NotConnectedException(String msg) {
        super(msg);
    }

}

```

10.2 Detaljeret Timeregnskab

Dato	Deltager	Design	Impl.	Test	Dok.	Andet	I alt
02/02/2017	Carl-Emil Hjort-Trærup	1,5	6				7,5
03/02/2017	Carl-Emil Hjort-Trærup		4				4,0
06/02/2017	Carl-Emil Hjort-Trærup		6			1	7,0
07/02/2017	Carl-Emil Hjort-Trærup		3,5	1	2		6,5
08/02/2017	Carl-Emil Hjort-Trærup		4		2		6,0
10/02/2017	Carl-Emil Hjort-Trærup		5,5	1	0,5		7,0
13/02/2017	Carl-Emil Hjort-Trærup		5				5,0
15/02/2017	Carl-Emil Hjort-Trærup		8,5	2,5	2	1	14,0
17/02/2017	Carl-Emil Hjort-Trærup		3	2		2	7,0
20/02/2017	Carl-Emil Hjort-Trærup		0,5		4,5		5,0
22/02/2017	Carl-Emil Hjort-Trærup				5		5,0
23/02/2017	Carl-Emil Hjort-Trærup	1,5	6				7,5
24/02/2017	Carl-Emil Hjort-Trærup		4				4,0
02/02/2017	Oliver Nybroe Pasca	1,5	6				7,5
03/02/2017	Oliver Nybroe Pasca		4				4,0
06/02/2017	Oliver Nybroe Pasca		6			1	7,0
07/02/2017	Oliver Nybroe Pasca		3,5	1	2		6,5
08/02/2017	Oliver Nybroe Pasca		4		2		6,0
10/02/2017	Oliver Nybroe Pasca		5,5	1	0,5		7,0
13/02/2017	Oliver Nybroe Pasca		5				5,0
15/02/2017	Oliver Nybroe Pasca		8,5	2,5	2	1	14,0
17/02/2017	Oliver Nybroe Pasca		3	2		2	7,0
20/02/2017	Oliver Nybroe Pasca		0,5		4,5		5,0
22/02/2017	Oliver Nybroe Pasca				5		5,0
23/02/2017	Oliver Nybroe Pasca	1,5	6				7,5
24/02/2017	Oliver Nybroe Pasca		4				4,0

Tabel 6: Detaljeret Timeregnskab 1/2

Dato	Deltager	Design	Impl.	Test	Dok.	Andet	I alt
02/02/2017	S. Ali Ghodrat	1,5	6				7,5
03/02/2017	S. Ali Ghodrat		4				4,0
06/02/2017	S. Ali Ghodrat		6			1	7,0
07/02/2017	S. Ali Ghodrat		3,5	1	2		6,5
08/02/2017	S. Ali Ghodrat		4		2		6,0
10/02/2017	S. Ali Ghodrat		5,5	1	0,5		7,0
13/02/2017	S. Ali Ghodrat		5				5,0
15/02/2017	S. Ali Ghodrat		8,5	2,5	2	1	14,0
17/02/2017	S. Ali Ghodrat		3	2		2	7,0
20/02/2017	S. Ali Ghodrat		0,5		4,5		5,0
22/02/2017	S. Ali Ghodrat				5		5,0
23/02/2017	S. Ali Ghodrat	1,5	6				7,5
24/02/2017	S. Ali Ghodrat		4				4,0
02/02/2017	Taygun Boran	1,5	6				7,5
03/02/2017	Taygun Boran		4				4,0
06/02/2017	Taygun Boran		6			1	7,0
07/02/2017	Taygun Boran		3,5	1	2		6,5
08/02/2017	Taygun Boran		4		2		6,0
10/02/2017	Taygun Boran		5,5	1	0,5		7,0
13/02/2017	Taygun Boran		5				5,0
15/02/2017	Taygun Boran		8,5	2,5	2	1	14,0
17/02/2017	Taygun Boran		3	2		2	7,0
20/02/2017	Taygun Boran		0,5		4,5		5,0
22/02/2017	Taygun Boran				5		5,0
23/02/2017	Taygun Boran	1,5	6				7,5
24/02/2017	Taygun Boran		4				4,0
02/02/2017	Thomas Nicolajsen	1,5	6				7,5
03/02/2017	Thomas Nicolajsen		4				4,0
06/02/2017	Thomas Nicolajsen		6			1	7,0
07/02/2017	Thomas Nicolajsen		3,5	1	2		6,5
08/02/2017	Thomas Nicolajsen		4		2		6,0
10/02/2017	Thomas Nicolajsen		5,5	1	0,5		7,0
13/02/2017	Thomas Nicolajsen		5				5,0
15/02/2017	Thomas Nicolajsen		8,5	2,5	2	1	14,0
17/02/2017	Thomas Nicolajsen		3	2		2	7,0
20/02/2017	Thomas Nicolajsen		0,5		4,5		5,0
22/02/2017	Thomas Nicolajsen				5		5,0
23/02/2017	Thomas Nicolajsen	1,5	6				7,5
24/02/2017	Thomas Nicolajsen		4				4,0
02/02/2017	Duran Köse	1,5	6				7,5
03/02/2017	Duran Köse		4				4,0
06/02/2017	Duran Köse		6			1	7,0
07/02/2017	Duran Köse		3,5	1	2		6,5
08/02/2017	Duran Köse		4		2		6,0
10/02/2017	Duran Köse		5,5	1	0,5		7,0
13/02/2017	Duran Köse		5				5,0
15/02/2017	Duran Köse		8,5	2,5	2	1	14,0
20/02/2017	Duran Köse		0,5		4,5		5,0
17/02/2017	Duran Köse		3	2		2	7,0
22/02/2017	Duran Köse				5		5,0
23/02/2017	Duran Köse	1,5	6				7,5
24/02/2017	Duran Köse		4				4,0

Tabel 7: Detalieret Timeregnskab 2/2