# 02324 - Videregåendeprogrammering - CDIO

## Gruppe: 14

David s147197

Johan s195491

Mohammad s184174

Christian s195480

William s195490

Max s195842

**Dato: 13. Marts 2020**

Github Repository

# Abstract

This paper is based on the planning and execution of the first iteration of the CDIO project, concerning a user management system. The content of the report will therefore take shape according to the requirements set in CDIO sub-task 1. Our main focus is on the 3-layer model and the use of interfaces as well as exception handling in the form of exceptions. The paper extends to analysis, design, and implementation of the project, as evidenced by the usual standard procedure for report writing.


# Forord

Denne rapport tager udgangspunkt i planlægningen og udførelsen af første iteration i CDIO-projektet, omhandlende et brugeradministrationssystem. Indholdet i rapporten vil derfor tage form efter de krav som er stillet i CDIO-delopgave 1.Vores hovedfokus omhandler 3-lagsmodellen og anvendelsen af interfaces samt undtagelseshåndtering i form af exceptions. Rapporten strækker sig over analyse, design, samt implementering af projektet, hvilket fremgår som følge af den sædvanlige standardprocedure for rapportskrivning.

# Indholdsfortegnelse

# 1 Vision

Vi ønsker at udvikle et brugeradministrationssystem hvis kernefunktionalitet dækker over oprettelse, sletning, samt opdatering og visning af brugerinformationer.

# 2 Krav & analyse

## 2.1 Use case model

Vi anvender et use case diagram til at identificere aktøren og mulige use cases, samt relationerne mellem disse. Derudfra kan de mest interessante use cases udvælges og beskrives yderligere.

### 2.1.1 Use case diagram

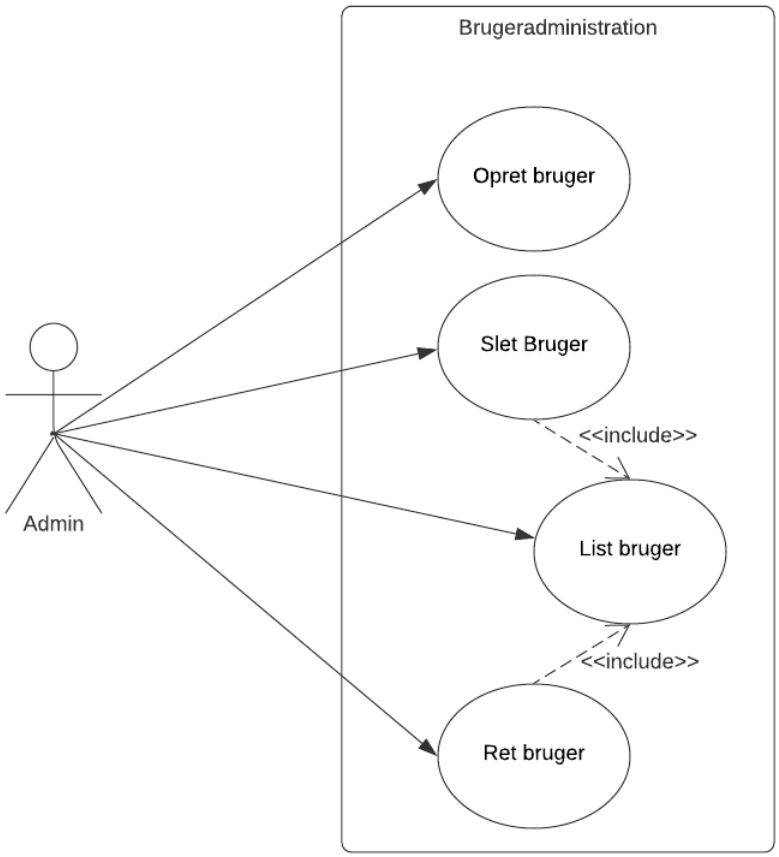


Figure 2.1: Use case over bruger-administrationsmodulet

### 2.1.2 Use case beskrivelse

Vi har valgt at beskrive alle disse use cases som brief, og "opret bruger" use casen som fully dressed.

**Opret bruger**
Brugeren vælger "opret ny bruger" muligheden i menuen og indtaster de påkrævede informationer. Brugeren modtager en bekræftelses-besked og menuen fremvises igen.

**Slet bruger**
Brugeren vælger "Slet bruger" muligheden fra menuen. Brugeren indtaster bruger-ID'et på den bruger som skal slettes. Systemet sletter brugeren og fremviser en bekræftelses-besked.

**Ret bruger**
Brugeren vælger "Ret bruger" muligheden. Brugeren indtaster bruger-ID'et på den ønskede bruger. Brugeren vælger hvilken information, som der skal ændres og indtaster de nye oplysninger.

**List bruger**
Brugeren vælger "List brugere" i menuen. Programmet viser så alle nuværende brugere i systemet og deres informationer såsom ID og navn.

**Opret bruger - fully dressed**
Use case: Opret bruger
Primær aktør: Admin
Forudsætninger: Programmet er startet.

Success scenarie:
1. Brugeren vælger "Opret ny bruger" i menuen.
2. Brugeren indtaster brugernavn, CPR og rolle.
3. Systemet opretter bruger og tildeler brugerID, initialer og password
4. Brugeren modtager en bekræftelsesbesked, at brugeren er oprettet samt brugerens informationer.
5. Menuen fremvises for brugeren igen

Alternative flows:
3a. Ugyldigt brugernavn
   1. Brugeren får en fejlmeddelelse
      1a. Brugernavn for langt
      1b. Brugernavn for kort
      1C. Brugernavnet er ikke tilgængeligt
   2. Systemet går tilbage til menuen
3b. Ugyldigt CPR
   1. Brugeren får en fejlmeddelelse
      1a. CPR for langt/kort
      1b. CPR dato ugyldig
      1c. CPR ugyldigt
   2. Systemet går tilbage til menuen
3c. Ugyldig rolle
   1. Brugeren får en fejlmeddelelse
   2. Brugeren bliver bedt om at vælge rolle igen

# 3 Design

## 3.1 Designklassediagram

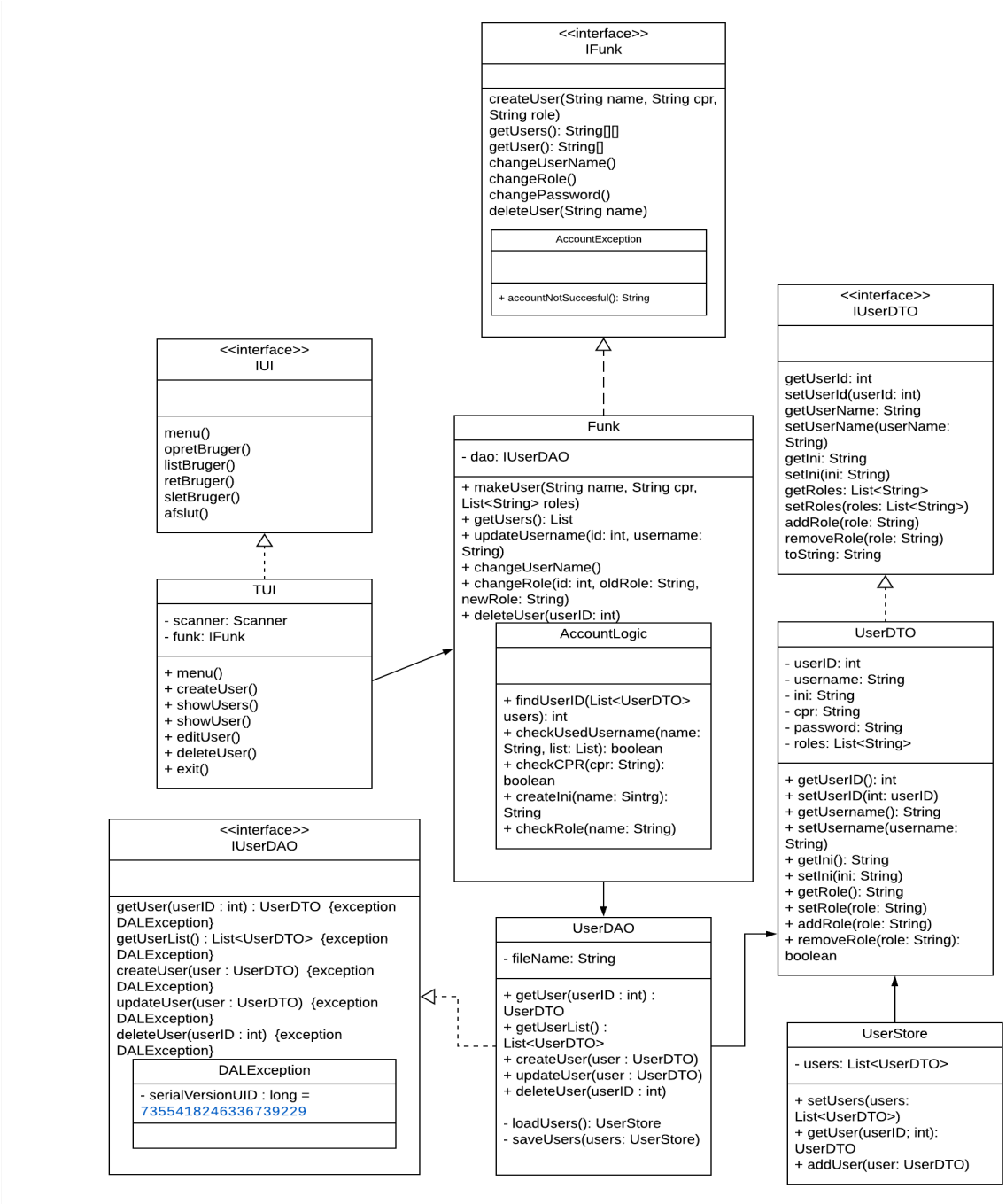


Figure 3.1: Designklassediagram over programmet

## 3.2 Pakkediagram

Vi har lavet et pakkediagram over den logiske struktur i vores program. Vi har valgt at inddele vores program efter 3-lags-modellen.

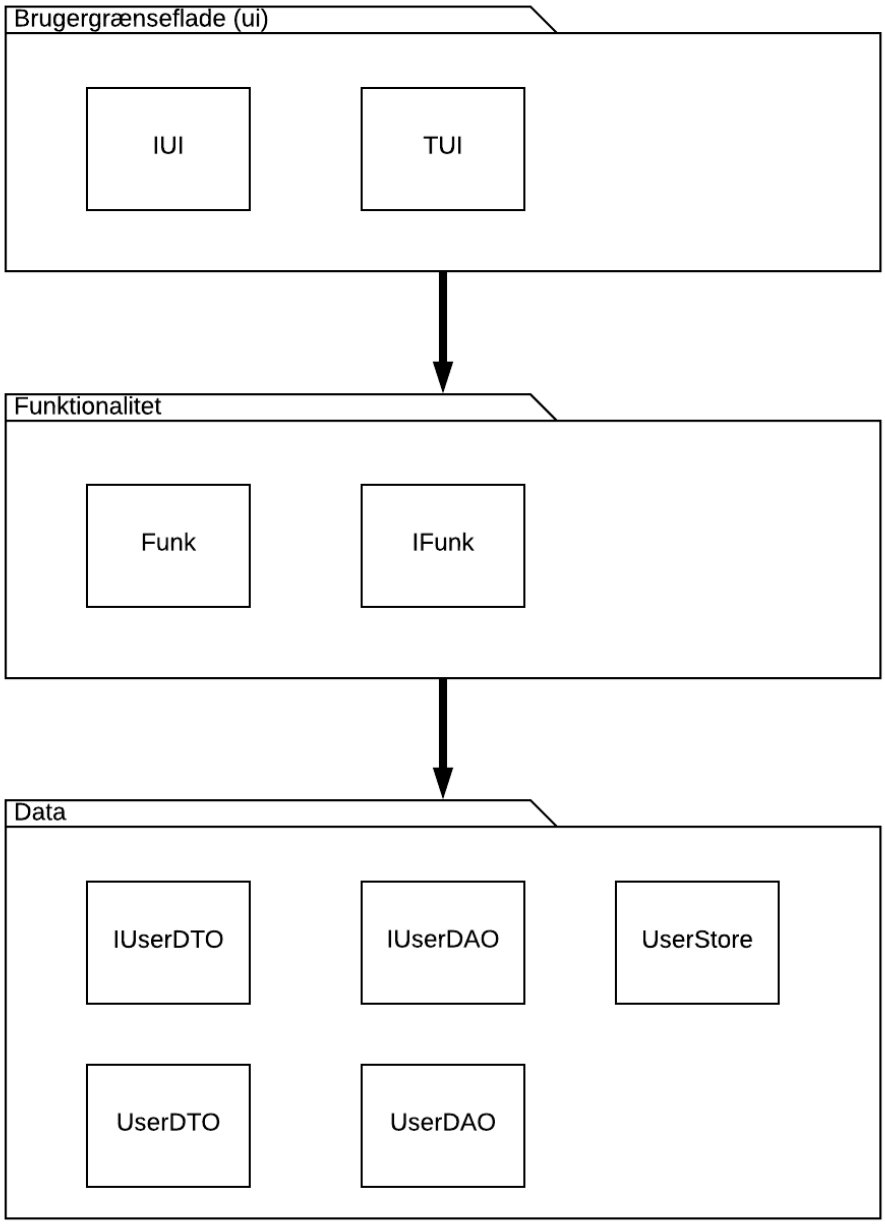


Figure 3.2: Pakkediagram over pakkestrukturen

## 3.3 SD-diagram

Vi har lavet et sekvensdiagram over "make user", som viser interaktionen mellem instanserne af de respektive klasser gennem metodekald. Vores SD tager udgangspunkt i at brugeren allerede har valgt funktionen "make user" gennem TUI'en, derudover er det, den generelle interaktion der vises for at skabe et overblik. Vi har derfor valgt at undlade flere små detaljer og ser interaktionen fra et lavere abstraktionsniveau.

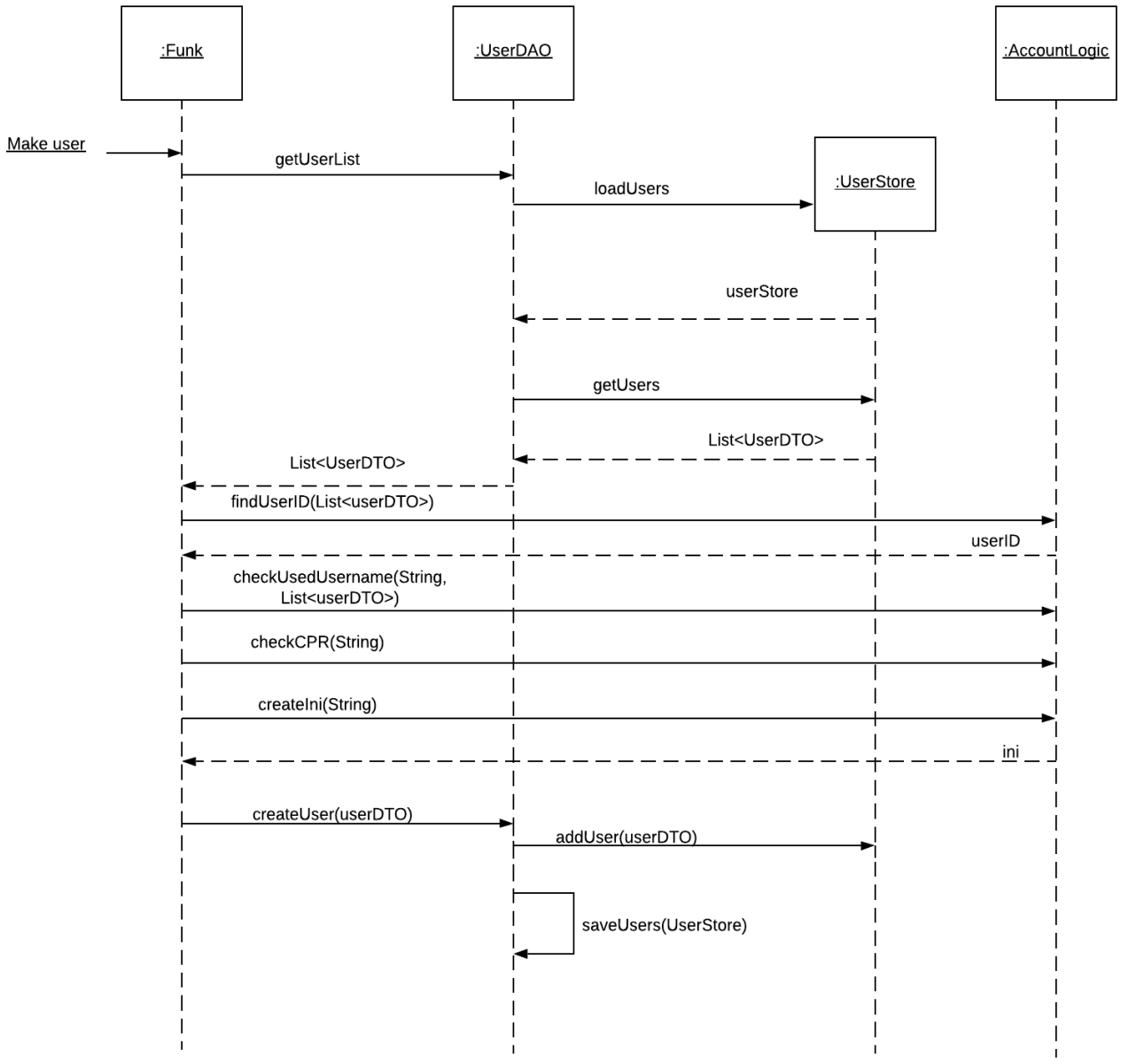


Figure 3.3: SD-diagram over opret bruger

# 4 Dokumentation & implementering

## 4.1 Interface & nedarvning

I vores program har vi brugt interfaces til at prædefinere vores klasser. Ved hjælp af interfaces, har vi vidst hvilke metoder der skulle laves, og hvilke parametre metoderne skulle have, for at kunne opfylde deres funktioner, samt hvilke exceptions der skulle kastes. Alle interface'rne er nedarvet til deres respektive standarde klasser, som også kan ses i klassediagrammet.

## 4.2 Kildekode forklaring

For at gøre vores program mere fleksibel har vi lavet interfaces til nogle af de centrale klasser. Hvis man fx er interesseret i at implementere en anden ui

### 4.2.1 Opbevaring af data - 3 lags strukturen

**Hvordan gemmes data fra input?**
Brugeren interagerer med grænseflade-laget, hvor brugeren kan vælge at oprette, vise eller at administrere brugere. Derefter vil vores grænseflade-lag kommunikere til funktions-laget, som undersøger om de modtaget inputs er gyldige, og derefter generer et gyldigt password, userID, og initialerne af brugerens navn.
Funktions-laget vil så kommunikere til data-laget. Data-laget kan gemme data, finde data frem, opdatere og slette data. Efter at data-laget har gjort sit arbejde, vil programmet enten have opfyldt brugerens ønsker eller kaste en exception, som vil blive catched i TUI'en.

### 4.2.2 Nested class(accountlogic)

Vi har lagt en klasse 'AccountLogic' nested inde i 'funk' klassen. Det har vi gjort for at gøre klassen mere overskuelig at læse. AccountLogic indeholder metoder der kan checke om input er gyldig og genere nødvendig information for at oprette en bruger. Det ville også være muligt at have metoderne inde i 'Funk' klassen eller gøre 'AccountLogic' public og ligge den inde i funktionalitet's pakken seperat.

### 4.2.3 Interfaces

**IFunk**

IFunk klassen indeholder alle de metoder og exceptions, som vores funktionelle klasse 'Funk', skal kunne eksekvere og catche. Det er bl.a. funktionerne for at kunne oprette, opdatere, tilføje/fjerne roller, slette, og vise brugere.

```
1   package funktionalitet;
2
3   import ...
6
7   public interface IFunk {
8       void makeUser(String username, String cpr, List<String> roles) throws AccountException, IUserDAO.DALException;
9       void updateUsername(int id, String username) throws AccountException, IUserDAO.DALException;
10      void addRole(int id, String role) throws IUserDAO.DALException, AccountException;
11      void removeRole(int id, String role) throws IUserDAO.DALException, AccountException;
12      void changeRole(int id, String oldRole, String newRole) throws IUserDAO.DALException, AccountException;
13      List getUsers() throws IUserDAO.DALException;
14      void deleteUser(int userID) throws IUserDAO.DALException;
15
16      class AccountException extends Exception {
17
18          public AccountException(String message) { super(message); }
21          public String accountNotSuccessful() { return "Account creation not successful: " + getMessage(); }
24      }
25  }
```

Figure 4.1: IFunk klassens kildekode

I interfacet befinder der sig en klasse 'AccountException', som extender Exception klassen. Denne exception skal fortælle om noget gik galt under oprettelsen af brugeren.

**IUI**

Ved at have et interface til vores userinterface klasse, vil det være muligt at tilføje nye implementeringer i fremtiden. Fx hvis en GUI skal implementeres, så vil man kunne implementere metoderne fra interface-klassen i GUI'en.

### 4.2.4 Exception

Der er oprettet en brugerdefineret Exception-klasse, DALException. DALException kastes af metoderne i datalaget, hvis der opstår fejl. Fejl kunne fx være angivelse af forkert ID, når man skal hente en enkelt persons oplysninger, eller hente en liste med alle bruger, hvor listen er tom. Dermed vil der blive kastet en DALException med en tilhørende fejlmeddelelse, afhængig af fejlen.

# 5 Konklusion

Det kan konkluderes af rapporten at vi har formået, at opbygge et brugeradministrationssystem med undtagelseshåndtering og interfaces med en 3-lags struktur. Vi overvejede at tilkoble en database, men da vi ikke har mulighed for at have en server der kan køre 24/7 har vi valgt at gemme det som filer.