CDIO Opgave - Del 3

Gruppe 21

Marie Seindal
s185363

Peter Revsbech
s183760

Christian Kyed
s184210

Ida Schrader
s195483

Anthony Haidari
s141479

Dánjal Dávidsson
s195472

Danmarks Tekniske Universitet

CDIO-opgave

02327 - Indledende databaser og database programmering.

GitHub: `https://github.com/Software2020Hold21/21_CDIO1.git`

14/03-2020

Timeregnskab

Dato	Hvem?	Timer	Noter
Marts 2020	Ida	5	Rapportskrivning
Marts 2020	Marie	10	Rapportskrivning
Marts 2020	Peter	12	Kodning og rapportskrivning
Marts 2020	Christian	10	Kodning og rapportskrivning
Marts 2020	Anthony	9	Kodning
Marts 2020	Danjal	0	

Indhold

1 Introduktion

Vi er i faget 02324 Videregående Programmering på DTU forår 2020 blevet bedt om at udvikle et bruger-administrationsmodul med en simpel tekstbaseret grænseflade. Den skal undersøtte persistent data, hvilket betyder at brugerne gemmes mellem afviklinger af programmet. Vi har en løsning som gemmer i en fil på en disk. Nærmere indblik i de forskellige klasser fås ved at læse designklassediagrammet.

2 Analyse

Dette kapitel vil beskrive hvordan vi har analyseret opgaven og er kommet frem til det design vi ville arbejde med.

2.1 Krav

Vores program skal opfylde følgende krav, vigtigst af alt er CRUD (create, read, update, delete).

- Oprette, vise, opdatere og slette en bruger.
- En bruger har følgende oplysninger:
  - En unik nøgle i form af et brugerID
  - Et brugernavn på mellem 2 og 20 tegn
  - Initialer, fra 2 til 4 tegn
  - Et CPR-nummer på 10 tegn
  - Et password som overholder følgende krav
    - Længden skal være mellem 6 og 50 tegn
    - Skal indeholde tre af følgende fire kategorier
      - små bogstaver a til z
      - store bogstaver A til Z
      - cifre 0 til 9
      - specialtegn: '.', '-', '_', '+', '!', '?', '='
  - Ved initialisering bliver brugeren tildelt et password
  - En eller flere af følgende roller: administrator, farmaceut, formand og operator
- En simpel tekstbaseret brugergrænseflade som tillader at CRUD'e

2.2 Use case

De vigtigste funktionaliteter, vores program skal understøtte er som sagt Create, Read, Update og Delete -funktioner i en database af brugere. Disse er meget almindelige og vi vil ikke uddybe deres formål yderligere her.

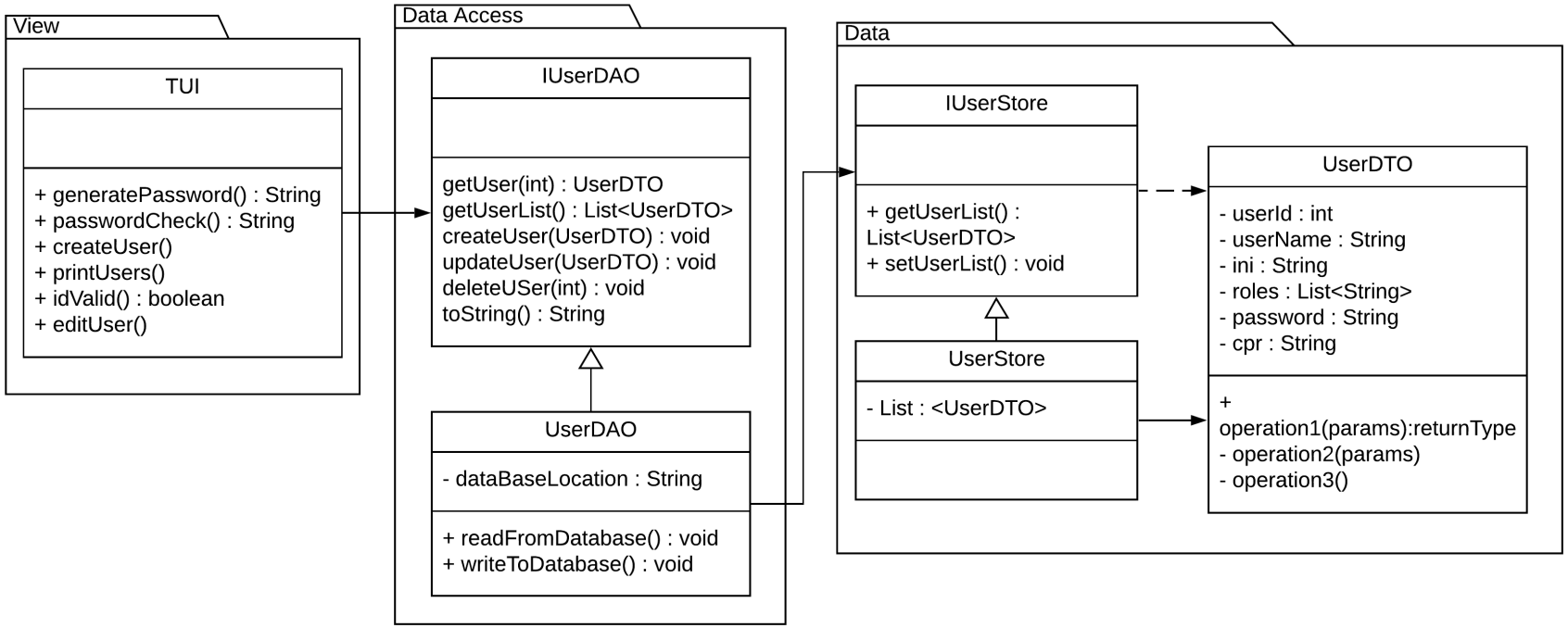
3 Design

Vi har i vores design valgt at benytte os af trelagsmodellen. Derfor er vores program opdelt i følgende tre lag: et datalag, et data-access lag og et viewlag. Mellem lagene er interfaces, så de er i stand til at udveksle oplysninger uden at kende til hinanden internt.
Til at overføre brugere, som indeholder flere forskellige data-elementer, benyttes klassen UserDTO, der er et Data Transfer Object. Dette skal grundlæggende ses som en "udvidelse"af javas primitive datatyper.

Vores brug af interfaces gør, at vi nemt i en anden version af programmet f.eks. vil kunne udskifte vores lokale data-base med en SQL-database, hvis implementationen af SQL-databasen bare benytter det samme interface, altså IUserStore.

3.1 Designklasse Diagram

I dette projekt har vi valgt at benytte 3-lagsmodellen, som ses på designklassediagrammet. Dette er opdelet i 3 pakker, tilsvarende de 3 lag, hhv. view, data access og data - lagene.



Figur 2: Klasse diagram

4 Implementering

4.1 TUI

TUI klassen er vores brugergrænse-flade som benytter sig af konsollen, og skriver forskellige beskeder til brugeren og modtager inputs. I klassen er der forskellige metoder, med dertilhørende hjælpemetoder, som blandt håndtere oprettelse af nye brugere, ændre eksisterende brugere, slette brugere og at få vist alle eksisterende brugere. Metoderne idValid() og generatePassword() kunne man godt have valgt at ligge i en funktionalitetsklasse da den ikke tager noget brugerinput og blot genere eller behandler data. Vi har benyttet os af try-catch til at håndtere eventuelle exceptions som kan opstå i forbindelse med forkerte brugerinput.

4.2 UserDAO

Dette er et Data Acess Object, som benyttes til at tilgå systemets datalag, hvor users gemmes. UserDAO-objektet skal altså understøtte de 4 vigtige funktionaliteter af programmet, Create, Read, Update og Delete. Det følger derfor naturligt, at denne klasse har metoder som getUserList(), createUser(), deleteUser() osv.

Klassen introducerer også en exception-klasse, DALException, som kastes, når der opstår i forsøget på at tilgå data-laget.

4.3 UserStore

UserStore-objektet er en simpel beholder til listen af UserDTO'er dvs. listen af users. Denne understøtter kun get og set -metoder.

4.4 UserDTO

UserDTO (User Data Transfer Object) er en simpel klasse, som blot skal indeholde data for en bestemt user. Klassen bruges til nemmere at kunne overføre de data af simple datatyper, som tilhører en bestemt user, uden at hver enkelt felt skal overføres for sig selv. Et UserDTO-objekt har derfor en userId, et UserName, initialer, en liste med roller, et password og et cpr-nummer. userId er en int, roles er en liste af Strings, og de andre felter er Strings. Vi har valgt at cpr skal være en String, da man alternativt ville have problemer med at gemme cpr-numre, der starter med 0.

5 Konklusion

Alt i alt er vi tilfredse med vores program. Det opfylder de krav vi fremsatte på kravlisten. Man kan oprette en bruger, slette den, opdatere den og printe dens oplysninger ud. I henhold til vores oprindelige plan om at lave en trelagsmodel er dette også en success. Vores program kan implementeres i en større helhed, da det snakker sammen vha interfaces. Implementationen er lykkedes, programmet kører fejlfrit. Vi har haft udfordringer i gruppen med at bruge git, men er kommet nogenlunde helskindet gennem mergekonflikter og andre småproblemer.

Havde vi haft tid til overs ville vi have brugt det på at gennemteste programmet mere systematisk. Senere i forløbet ser vi frem til at kunne sammenkoble vores løsning med en SQL database.