CDIO - D1

‘Anvendelse af 3-lags modellen med interfaces’

Videregående programmering 02324, Softwareteknologi

Gruppe Nr.: 13

Afleveringsfrist: Lørdag 25/02-2017 05:00

Institut: DTU Compute

Vejleder: Ronnie Dalsgaard

Denne rapport er afleveret via Campusnet (der skrives ikke under)

Denne rapport indeholder 22 sider eks. denne sider.

Jesper Bang, s144211

Bertram Christian Henning, s153538

Jia Hao Johnny Chen, s165543

Jonathan Yngve Friis, s165213

Christopher David Carlson Chytræus, s165230

Thomas Kristian Lorentzen, s154424

Abstract

The project documented in this report is a user admin module using the 3-layer model. For interaction from the users, there is a simple textual user interface (TUI) implemented. For the project to work properly, we have created a controller communicating between the interface and entity classes. For the interface, we've implemented the following features for the interface: creating a user, deleting a user, listing the users, modifying the user's, giving roles to the users and terminating the program. As a tool to help us make our development process easier, we've been using UML models and diagrams to create a more manageable view of how to structure our program. To ensure that our code has been developed correctly, we've been documenting the functionalities by creating JUnit tests, which have been successful. Lastly, we can conclude that our project has been successfully developed.

Contents

Timeregnskab

Timeregnskab	**Uge 07**					
Deltager	**Design**	**Impl.**	**Test**	**Dok.**	**Andet**	**I Alt**
Bertram	5	3		0		8
Christopher	7	1		1		9
Johnny	6	1		1		8
Jonathan	6	2		1		9
Thomas	7	1		1		9
Jesper	6	2		1		9
Sum	37	10		5		52

Timeregnskab	**Uge 08**					
Deltager	**Design**	**Impl.**	**Test**	**Dok.**	**Andet**	**I Alt**
Bertram	2	12	2	3		19
Christopher	5	4	0	4		13
Johnny	6	4	0	5		15
Jonathan	3	8	2	3		16
Thomas	6	3	0	6		15
Jesper	4	8	1	3		16
Sum	26	39	5	24		94

Formål

Formålet er at lave et bruger-administrationsmodul program som skal blive designet efter principperne i 3-lags modellen. Programmet benyttes til administration af brugere/operatører, hvor man kan oprette, vise, opdatere og slette brugere. Derudover er der også forskellige typer af brugere.

Kravspecifikationer

Functional:

1. Systemet skal kunne oprette brugere.
2. Systemet skal kunne vise brugere.
3. Systemet skal kunne opdatere brugere.
4. Systemet skal kunne slette brugere.
5. Systemet skal have et tekstbaseret brugergrænseflade (TUI).
6. Systemet skal kunne lade en bruger have flere forskellige roller (Se bilag 1).
7. Systemet skal kunne tildele nye brugere initialiserings-password, som genereres automatisk og overholder de samme regler som anvendes på DTU.

Reliability:

8. Systemet skal kunne gemme data af brugere i programmet.

Implementation:

9. Systemet skal udvikles i Eclipse.
10. Systemet skal bruge UTF-8 som tegnsæt.
11. Systemet skal kodes i Java.

Hovedafsnit

Code snippets

I Controlleren benytter vi os af HashSet når vi sætter rollerne, da HashSet kun kan have en af samme element. Så hvis man prøver at give en user den samme rolle flere gange så vil den fortsat kun blive noteret i hashset en gang, da elementet allerede eksistere. Dette gør at vi ikke skal bekymre os om at sortere eller fjerne elementer der opstår op mod flere gange i settet.

```
System.out.println("hvilke roller vil du have?");
Set<String> roles= new HashSet<String>();
loop:
while(true){
```

Efterfølgende laver vi vores HashSet med alle rollerne om til en liste da vore interface kræver dette.

```
List<String> roles2= new ArrayList<String>();
roles2.addAll(roles);
user.setRoles(roles2);
```

I UserStore benytter vi os af HashMap til fordel for ArrayList da HashMap har flere fordele for os, heriblandt at HashMap har en key til hver af værdierne og ikke gemmer elementerne i sorteret rækkefølge. HashMap minder mest om databasestrukturen vi senere ønsker at implementere og er også hurtigere at gennemføre basale operations, som vi blandt andet bruger, såsom add, remove, contains osv. end ArrayList.

```
public class UserStore implements Serializable {
        private static final long serialVersionUID = -9162958810235685263L;
        private Map<Integer, UserDTO> users = new HashMap<Integer, UserDTO>();
```

Et lille eksempel på hvordan vi udnytter at HashMap har keys til deres værdier.

```
if (!users.containsKey(userId)) {
    throw new IUserDAO.DALException("User findes ikke");
}
```

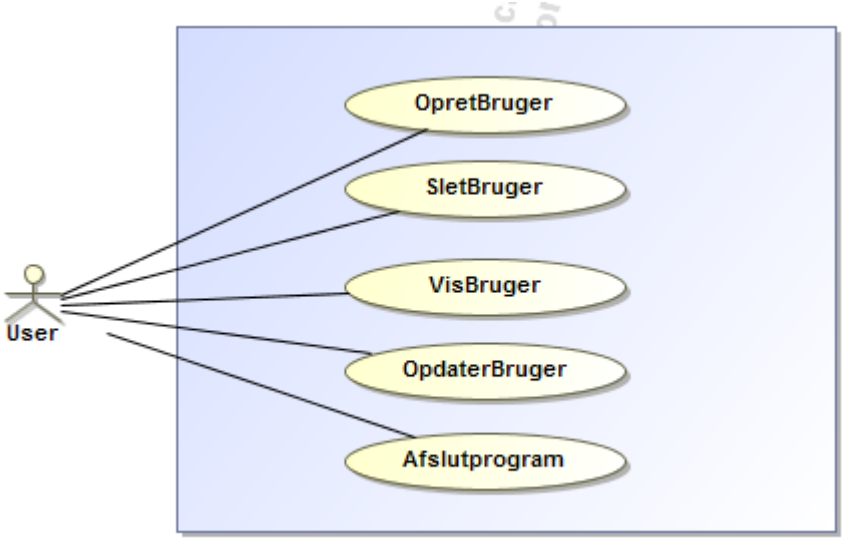
Vi har så vidt muligt holdt det meste logik ude af vores TUI og sørge for at det er delt ordentligt op i metoder så vi kan kalde på den del af TUI som vi bruger flere gange og derved nemt kan kalde igen og igen når vi har behov for at kalde dele af TUI flere gange.

```
public String antalRoller(){
        String out = "";
        System.out.println("Indtast antal roller");
        return out;
}
public String roles(){
        String out = "";
        System.out.println("Indtast hvilke roller du vil have");
        return out;
}
```

3-lags modellen

I 3-lags modellen har man grænsefladelag, funktionalitetslag og datalag. I vores system har vi TUI som vores grænseflade, som indeholder tekst der bliver udskrevet til brugeren og modtager input fra brugeren. I vores funktionalitetslag har vi vores Controller, der indeholder programmets logik og håndterer kommunikationen mellem grænsefladelaget og datalaget. I den ene løsning, hvor den ikke gemmer data på en disk, har vi vores NonPersistentDAO som datalag. Her gemmes data midlertidigt så længe programmet kører. Så snart programmet afsluttes, vil dataet blive glemt. I vores anden løsning, hvor dataet bliver gemt på en fil i disken, har vi SerialDAO og UserStore som vores datalag. Grunden til at vi ikke har lavet løsningen, hvor dataene bliver gemt på en database, er fordi vi ikke har lært at forbinde Java sammen med SQL-database i kursus 02327 endnu.

Use case diagram



Her ses vores use case diagram ovenfor. Diagrammet viser en oversigt over hvilke muligheder en bruger har i programmet. Nedenfor kan man se vores use case beskrivelser.

Use cases

Use Case: OpretBruger
ID: 01
Brief description: Brugeren vælger at oprette en bruger og succesfuldt får oprettet en bruger i programmet.
Primary actors: Nuværende bruger.
Preconditions: Brugeren skal være admin i programmet.
Main flow: 1. Brugeren vælger “Opret bruger”. 2. Brugeren indtaster ID. 3. Brugeren indtaster navn.

4. Brugeren indtaster cpr-nr.
5. Brugeren indtaster antal roller.
6. Brugeren vælger hvilke roller der ønskes.

Postconditions: Brugeren har succesfuldt oprettet en bruger.

Use Case: SletBruger

ID: 02

Brief description: Brugeren vælger at slette en bruger fra programmet.

Primary actors: Nuværende bruger.

Preconditions: Brugeren skal være admin i programmet.

Main flow:

1. Brugeren vælger “Slet bruger”
2. Brugeren indtaster ID’et på den bruger der skal slet.

Postconditions: Programmet sletter succesfuldt den valgte bruger.

Use Case: VisBruger

ID: 03

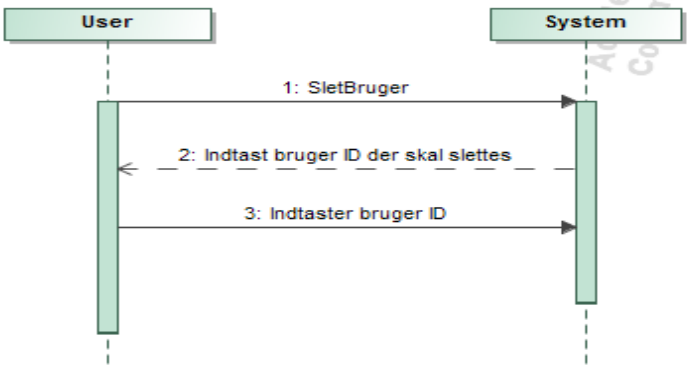
Brief description: Brugeren vælger at vise brugere fra programmet.
Primary actors: Nuværende bruger.
Preconditions: Brugeren skal være admin i programmet.
Main flow: 1. Brugeren vælger “Vis bruger”.
Postconditions: Programmet viser en liste over alle brugerne.

Use Case: OpdaterBruger
ID: 04
Brief description: Brugeren vælger at opdatere en bruger.
Primary actors: Nuværende bruger.
Preconditions: Brugeren skal være admin i programmet.
Main flow: 1. Brugeren vælger “Opdater bruger”. 2. Brugeren vælger den ønskede bruger der skal opdateres. 3. Brugeren opdater de ønskede egenskaber.
Postconditions: Programmet opdatere den valgte brugers egenskaber.

Use Case: AfslutProgram
ID: 05
Brief description: Brugeren vælger at afslutte programmet.
Primary actors: Nuværende bruger.
Preconditions: Brugeren skal være admin i programmet.
Main flow: 1. Brugeren vælger “Afslut program”.
Postconditions: Programmet afsluttes.

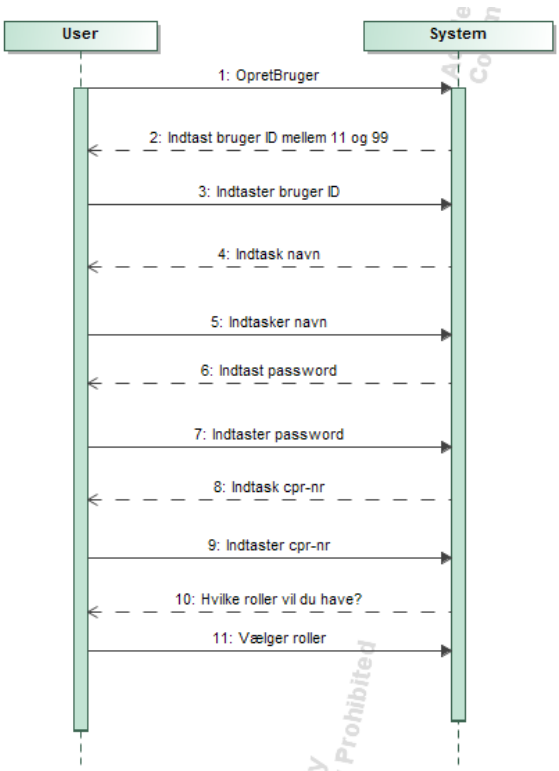
System sekvens diagram

SletBruger:



Her ses forløbet for sekvensen, når administratoren sletter en bruger fra databasen. Brugeren vælger funktionen for at slette en bruger, hvorefter systemet sender en meddelelse til brugeren om at han skal indtaste navnet på det brugerID som skal slettes. Når dette er gjort, er brugeren fjernet fra systemet.

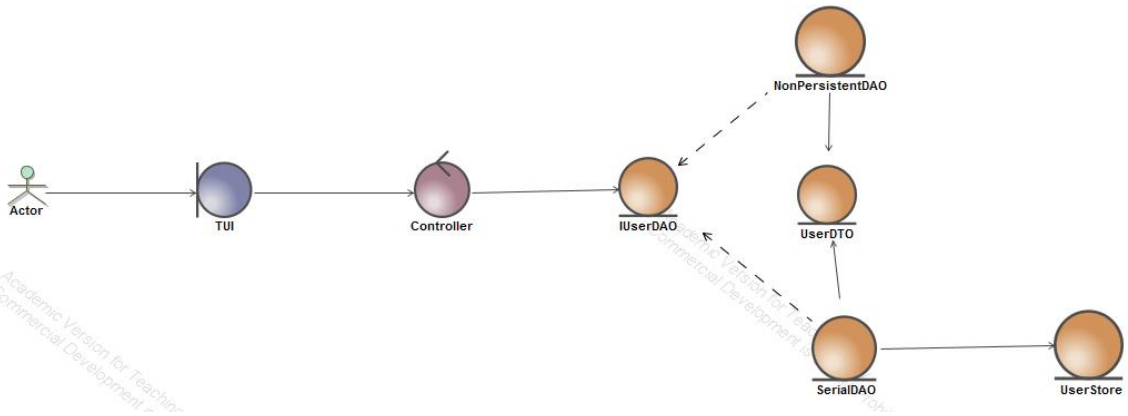
OpretBruger:



I ovenstående diagram afbildes interaktionen mellem brugeren og systemet, når administratoren skal oprette en ny bruger. Når brugeren vælger funktionen for at oprette en bruger, sender systemet en besked tilbage, hvor brugeren bedes indtaste et ID mellem 11 og 99. Når dette er indtastet, får brugeren derefter en besked om at indtaste et navn. Når dette er gjort, får brugeren dernæst besked fra systemet om at indtaste et password. Til sidst bedes brugeren at indtaste CPR-nr for den bruger, som

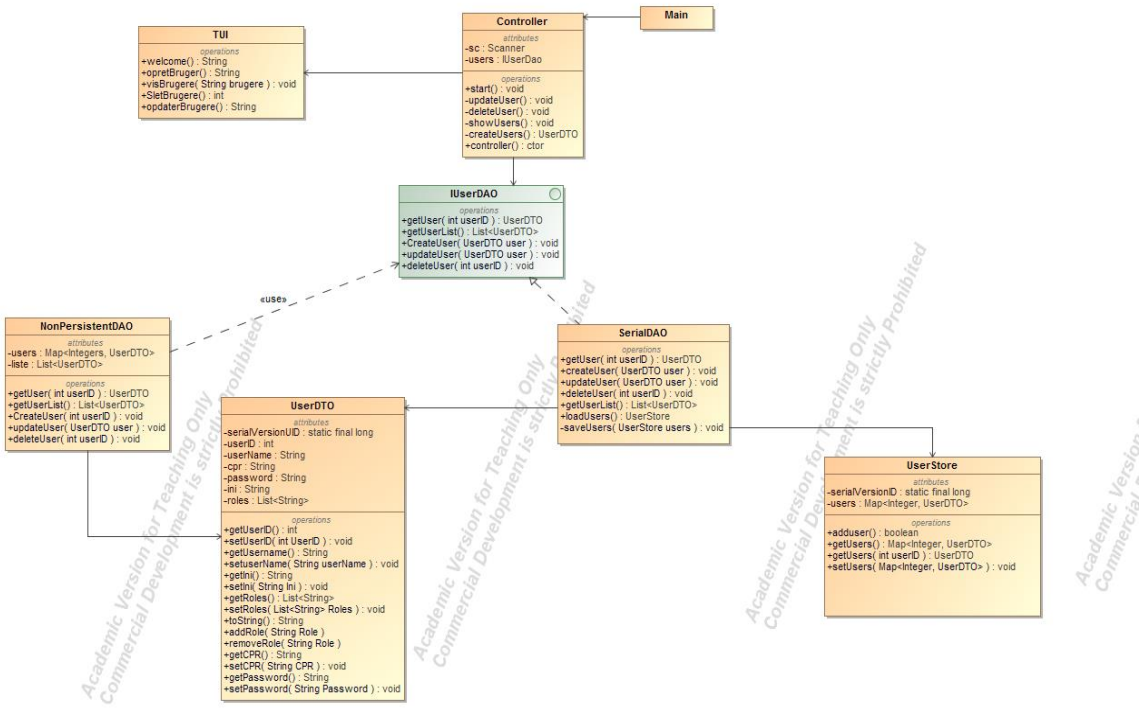
skal oprettes. Når denne oplysning er givet til systemet fra brugeren, sender systemet en besked til brugeren, hvor han skal vælge en rolle. Når brugeren har valgt det, og systemet har modtaget det, er der oprettet en ny bruger.

BCE-model



Ovenstående diagram er en model over vores system givet ved en BCE-model. TUI er vores boundary da det er grænsefladen, som er det brugeren ser, den er altså grænselaget. Controller er vores controller, heraf navnet, der står for logikken og selve programmet. Controlleren er altså funktionslaget. IUserDAO er interfacet der forbinder datalaget med funktionslaget. NonPersistentDAO er den klasse man bruger hvis det ønskes at have et program hvor det ikke er muligt at gemme brugerne. SerialDAO er den klasse man bruger hvis det ønskes at have et program hvor det er muligt at gemme brugerne og UserStore er den klasse der sørger for at brugerne bliver gemt ned på en fil.

Design klassediagram



Ovenstående diagram viser, hvordan de forskellige klasser hænger sammen. I midten har vi vores brugergrænseflade, som er markeret med en grå farve til forskel fra de andre klasser. Dette skyldes, at det ikke er en klasse på samme måde som de andre, men i stedet indeholder metoder, som andre klasser kan implementere. I vores program bruger både SerialDAO- og NonPersistentDAO-klassen metoderne fra vores interface. Det samme gør vores Controller klasse.

Test

Testcase ID	01
Summary	The program shall use a TUI to interact with the user.
Requirements	5

Preconditions	The program has been started
Postconditions	The program used a TUI to display information to the user
Test Procedure	1. Start the program. 2. Check if the program uses a TUI to "guide" the user. 3. Terminate the program
Test Data	none
Expected Result	The system uses a TUI to "guide" the user.
Actual Result	The system used a TUI to "guide" the user.
Status	Passed
Tested by	Jonathan Friis
Date	24-02-2017
Date environment	Eclipse jee neon 2.0 on windows 10

Testcase ID	02
Summary	The system gives a randomly generated password to the user upon creation.

Requirements	7
Preconditions	The system has been started
Postconditions	The user has a randomly generated password
Test Procedure	1. Start the system 2. Create user 3. Show user 4. check if the system gave the user a random password. 5. Redo the procedure 10 times and check if the next users have a different password.
Test Data	Eks: Password first user: ADTad786 Password second user: GHkol12
Expected Result	The system gives the users a random password.
Actual Result	The system gave the users random password.
Status	Passed
Tested by	Jonathan Friis

Date	24-02-2017
Date environment	Eclipse jee neon 2.0 on windows 10

Testcase ID	03
Summary	The system shall save the users info on a seperate file.
Requirements	8
Preconditions	The system has been started
Postconditions	The users is saved on a seperate file.
Test Procedure	1. Start the program 2. Create user 3. Terminate the program 4. Start the program again 5. Show users 6. Check if the users is still there
Test Data	None
Expected Result	The system saves created users on a seperate file.
Actual Result	The system successfully saved users on a seperate file.

Status	Passed
Tested by	Jonathan Friis
Date	24-02-2017
Date environment	Eclipse jee neon 2.0 on windows 10

JUnit test 1 (Krav 1)

Test 1 tester at systemet kan lave en bruger, det gør den ved at lave en ny bruger med en test id og gemme den i vores datalag, så henter vi brugeren med test id'et og tester om den er blevet lavet ved at se at den ikke er null.

JUnit test 2 (Krav 2)

Test 2 tester at systemet kan vise brugere, det gør den ved at lave en bruger med navn, initialer og roller, så gemmer den brugeren og tester om testbrugeren fra datalaget giver en String der passer med de informationer den har.

JUnit test 3 (Krav 3)

Test 3 tester at systemet kan opdatere brugere, det gør den ved at lave en bruger med test id og navnet "ABC" og gemme den, så laver den en ny bruger med samme id og navnet "DEF" og sender den til datalaget for at opdatere. For at se om brugeren er blevet opdateret henter testen navnet på brugeren med test id og tester at det er "DEF".

JUnit test 4 (Krav 4)

Test 4 tester at systemet kan slette brugere, det gør den ved at lave en bruger med test id og gemme den i datalaget, så sletter den brugeren med test id'et og prøver at hente den, så tester den at den har fået beskeden "Ingen user med den id".

JUnit test 5 (Krav 6)

Test 5 tester at systemet kan give en bruger mere end en rolle, det gør den ved at oprette en bruger, tilføje to forskellige roller til den og gemme den. så henter testen brugerens roller og tester om der er de to roller som den tilføjede.

Test-Matrix

	Junit 1	Junit 2	Junit 3	Junit 4	Junit 5	TC 1	TC 2	TC 3
Krav 1	x							
Krav 2		x						
Krav 3			x					
Krav 4				x				
Krav 5						x		
Krav 6					x			
Krav 7							x	
Krav 8								x

Konfiguration

Programmet kræver en computer med java 1.7 eller nyere installeret, eclipse og en fil til at gemme programmet på installeret.

Guide til importering af git repository:

1. Start browser

2. Gå ind på siden: https://github.com/BertramHenning/CDIO_1

3. Klik på “clone or download” og kopier derefter linket

4. Åben eclipse

5. Klik på “file” og vælg “import”

6. Vælg “Git-> Projects from Git”

7. Tryk “next”

8. Klik på “clone URL”

9. Klik på “next”

10. Klik på “next” igen

11. Vælg sti at gemme det på

12. Klik på “next”

13. Vælg “import existing eclipse projects” og tryk “next”

14. Sørg for at project mappen har et flueben ud for sig eller klik i den tomme kasse

15. Tryk “finish”

16. Åben projektet i eclipse

17. Åben “src”

18. Åben pakken “(Default Package)”

19. Åben Main.java

20. Klik på den grønne knap med den hvide pil i.

Konklusion

Vores program er designet efter 3-lags modellen og stemmer overens med vores kravspecifikationer. Programmet funktionaliteter er blevet testet og virker som de skal. Dermed kan vi konkludere at programmet har fyldt afleveringsopgaven.

Litteratur

- CDIO-D1-Instrucktions:

  https://docs.google.com/document/d/1YRFCH9x6z5dYTgL_dco6docqFHBXmn8IHQlYAieqk40/edit

Bilag

Bilag 1:

Felt	Datatype	Beskrivelse
userID	int	Entydig identifikation af en bruger (unik nøgle). Benyttes når der skal foretages en afvejning til at identificere operatøren over for vægten. Valg af userID foretages af brugeren i intervallet 11-99.
userName	String	Brugernavn (min.2 max 20 tegn)
ini	String	Initialer - 2 til 4 tegn
cpr	Vælg selv	Brugerens cpr-nummer
password	Vælg selv	Ukrypteret password
roles	String	Gyldige roller: Admin, Pharmacist, Foreman, Operator