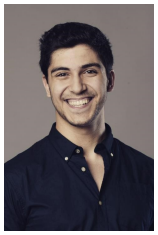
CDIO1 02324 Videregående Programmering

February 23, 2018

Mads
Seydel
s175197

Frederik
Husted
s164101

Mette
Rejmers
s175184



Isaac
Irani
s175176

Benjamin
Damore
s155891

Lea
Mellerup-Lund
s175195

Abstract

This paper examines the software development of a simple interactive administrative user interface, conducted through a text based user interface. The product was developed through unified process and iterations of distinct classes along with how they collaborate with one another. The buildout faced the process of analysis, design, implementation, and test, corresponding to the professors vision and requirements for the program. We've ended up with a fully functional program, in which one can add, edit and administrate different users, and save them in a database.

Contents

1 Indledning

Denne rapport dokumenterer fremgangsmåden af konstruktionen af denne opgave, hvori der en database skulle etableres. Rapporten indeholder en gennemgang af analysen, implementeringen og tests af dette program. Der er i dette forløb anvendt Unified Process.

1.1 Formål

Formålet med denne delopgave er at udforske nogle af de nye principper og features vi har lært i de første uger på kursus 02324. Det er bl.a. tekstbaseret brugergrænseflade (TUI), interface, exceptions, lidt MySQL database programmering mm. Som nævnt ovenfor er dette en delopgave. Dvs, at i løbet af kursus vil vi udvide yderligere og implementere flere features.

2 Analyse

2.1 Vision

Der ønskes en fuldt fungerende database som indeholder en række brugere, som alle indeholder et navn, CPR nummer, bruger ID, initialer og et password. Disse brugere skal kunnes vises og gemmes i et persistent storage eller, hvis muligt, i en mysql database.

2.2 Kravliste

Vi har opstillet en række krav til vores program ud fra kravet om det endelige program.

2.2.1 Funktionelle krav

- Opret bruger:
  - Det skal være muligt at oprette brugere med et unikt ID, brugernavn, CPR nr., password og rolle.
- Vis brugere
  - Det skal være muligt at vise brugere efter angivede kommando.
- Opdater bruger
  - Man skal kunne opdatere bruger information hvis ønsket. Det er dog kun muligt at ændre Navn, initialer, rolle og password.
- Find bruger
  - Det skal være muligt at finde specifikke brugere.
- Slet bruger
  - Det skal være muligt at slette bruger udfra ID.
- Afslut program
  - Man skal kunne afslutte programmet.

2.2.2 ikke funktionelle krav

- Java programmering
  - Programmet skal skrives i Java.
- Databar på DTU
  - Programmet burde kunne operere på DTU's databar vha windows operativ system.
- Oversættelse
  - Det skal være let at oversætte programmet til andre sprog.
- Git
  - Brug git til versionskontrollering.

2.3 Risici

Denne del af rapporten er tildelt for at vise de forskellige risici i projektets forløb. Risici vil bliver vurderet på en skalar fra 1 til 10, hvor den højeste prioriteret er den med det højeste negative påvirkning. Risici er organiseret i en tabel nedenfor hvor der bliver beskrevet hvor signifikant, sandsynlighed og hvordan vi vil forebygge/håndtere dette.

2.3.1 Risici tabel

Significance er rankeret fra en skalar fra 1 til 10, hvor 10 er den højeste og 1 er den laveste. Dette rankeres på hvor kritisk risikoen ville være og hvad sandsynligheden er for at det forekommer.

Risiko	**Significance**	**Indflydelse**
Sygdom	3	2
Troubleshooting (implementerings problemer)	6	3

Sygdom: dette er lidt svære at forebygge siden der meget nemt kan opstå næsten spontan sygdom hvilket afhænger af mange faktorer. Hvis sygdom opstår, kan holdet som enhed kan dog supplere hinanden men ekstra arbejde og kompensation.

Troubleshooting: Problemer med implementering af kode kan forebygges vha tæt samarbejde med hinanden og god kommunikation. bl.a. kan man også forebygge indflydelsen af problemer ved implementering af koden ved at afsætte decideret arbejdsdage til at troubleshoote problemer.

2.4 Stakeholders

Role	**Betingelse**	**Type**
Administrator	Oprette, vis, opdater, find bruger og slut programmet uden problemer	Primær

2.5 Use Cases

Vi har til dette program beskrevet en række use cases. De er bygget over de funktionelle krav til programmet.

Use Case Name: Opret bruger.
Primary Actor: Administratorerne.
Secondary Actors: Brugerne.
Preconditons: Programmet kører og brugeren er valideret (admin).
Postconditions: Ny bruger er oprettet.
Main Flow:

1. Vælg opret bruger.

2. Indtast bruger ID.

3. Vælg brugernavn, initialer, CPR nr, password og rolle.

4. Vælg afslut.

Use Case Name: Vis brugere.
Primary Actor: Administratorerne.
Secondary Actors: Brugerne.
Preconditons: Minimum en bruger er oprettet og der er logget ind.
Postconditions: Liste af brugere vises som valgt.
Main Flow:

1. Vælg list brugere.

2. Vælg alfabetisk eller numerisk rækkefølge.

Use Case Name: Opdater brugere.
Primary Actor: Administratorerne.
Secondary Actors: Brugerne.
Preconditons: Minimum en bruger er oprettet og der er logget ind.
Postconditions: Brugeren er opdaret som ønsket.
Main Flow:

1. Vælg opdater bruger.

2. Indtast bruger ID.

3. Ændr bruger (Navn, Initialer, rolle og password).

4. Vælg afslut.

Use Case Name: Slet bruger.
Primary Actor: Administratorerne.
Secondary Actors: Brugerne.
Preconditons: Minimum en bruger er oprettet og der er logget ind.
Postconditions: Brugeren er slettet.
Main Flow:

1. Vælg slet bruger.

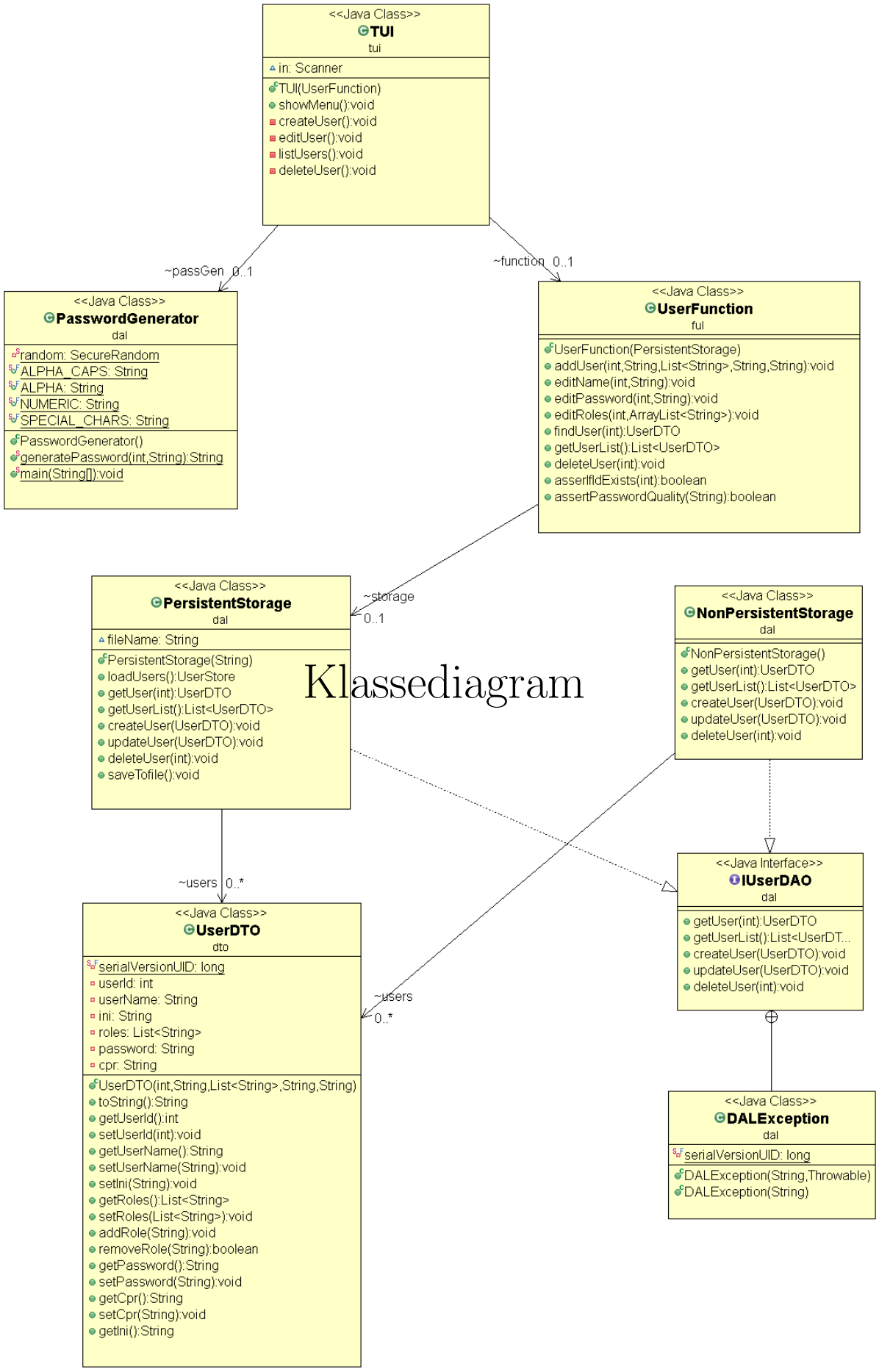
2. Indtast bruger ID.

3. Bekræft sletning.

Use Case Name: Afslut prgram.
Primary Actor: Administratorerne.
Secondary Actors: Brugerne.
Preconditons: Programmet kører.
Postconditions: Brugeren er opdaret som ønsket.
Main Flow:

1. Vælg afslut program.

2. Bekræft afslutningen.


<<Java Class>>
TUI
tui
in: Scanner
TUI(UserFunction)
showMenu():void
createUser():void
editUser():void
listUsers():void
deleteUser():void
~passGen 0..1
~function 0..1
<<Java Class>>
PasswordGenerator
dal
random: SecureRandom
ALPHA_CAPS: String
ALPHA: String
NUMERIC: String
SPECIAL_CHARS: String
PasswordGenerator()
generatePassword(int,String):String
main(String[]):void
<<Java Class>>
UserFunction
ful
UserFunction(PersistentStorage)
addUser(int,String,List<String>,String,String):void
editName(int,String):void
editPassword(int,String):void
editRoles(int,ArrayList<String>):void
findUser(int):UserDTO
getUserList():List<UserDTO>
deleteUser(int):void
asserIfIdExists(int):boolean
assertPasswordQuality(String):boolean
<<Java Class>>
PersistentStorage
dal
fileName: String
PersistentStorage(String)
loadUsers():UserStore
getUser(int):UserDTO
getUserList():List<UserDTO>
createUser(UserDTO):void
updateUser(UserDTO):void
deleteUser(int):void
saveTofile():void
~storage
0..1
Klassediagram
<<Java Class>>
NonPersistentStorage
dal
NonPersistentStorage()
getUser(int):UserDTO
getUserList():List<UserDTO>
createUser(UserDTO):void
updateUser(UserDTO):void
deleteUser(int):void
~users 0..*
<<Java Interface>>
IUserDAO
dal
getUser(int):UserDTO
getUserList():List<UserDT...
createUser(UserDTO):void
updateUser(UserDTO):void
deleteUser(int):void
<<Java Class>>
UserDTO
dto
serialVersionUID: long
userId: int
userName: String
ini: String
roles: List<String>
password: String
cpr: String
UserDTO(int,String,List<String>,String,String)
toString():String
getUserId():int
setUserId(int):void
getUserName():String
setUserName(String):void
setIni(String):void
getRoles():List<String>
setRoles(List<String>):void
addRole(String):void
removeRole(String):boolean
getPassword():String
setPassword(String):void
getCpr():String
setCpr(String):void
getIni():String
~users
0..*
<<Java Class>>
DALException
dal
serialVersionUID: long
DALException(String,Throwable)
DALException(String)

3 Implementering

Vi har valgt at opbygge vores program efter 3-lagsmodellen. Vi definerede vores datalag efter den udleverede interface. Vi sørgede for at datalaget kunne bruge UserDTO objekter, som vi havde defineret ud fra UserDTO-klassen. Efter dette tilføjede vi en funktionsklasse som også er vores controller og en TUI-klasse. Vi skrev hovedmenuens muligheder op i en switch og fik hvert ansvar for en funktion. Som nødvendigt tilføjede vi så også metoder til funktionslaget. Vi brugte DBTester-klassen til løbende at sikre os at det, vi indtil videre havde lavet, virkede. Vi har udfra GRASP princippet gået efter lav kobling og høj samhørighed. Det er ses blandt andet igennem vores controller, og vores User Tranfer Object klasse som har ansvaret for vores objekter (brugere).

4 Test

4.1 Aktør Test

Der er lavet en række aktør tests der dækker vores use cases, for at vise og verificere at vores use cases er implementeret og fungerer.
TC1: Opret bruger.
Preconditons: Programmet kører og der er verificeret.
Test procedure:

1. Der vælges opret bruger.
2. Systemet beder om oplysninger til brugeren.
   (a) Brugernavn.
   (b) CPR nummer.
   (c) Roller.
   (d) Bruger ID.
3. Systemmet melder bruger oprettet.

Testdata: Bruger data.
Expected result: Brugeren er oprettet og gemt.
Actual result: Expected result.
Status: Succes.
Tested by: Frederik.
Date:22/02/18
Test Environment MacOS High Sierra

TC2: Vis brugere.
Preconditons: Programmet kører og der er verificeret.
Test procedure:

1. Vælg vis brugere.

Expected result: Liste af brugere vises.
Actual result: Expected result.
Status: Succes.
Tested by: Frederik.
Date:22/02/18
Test Environment MacOS High Sierra

TC3: Opdater bruger.
Preconditons: Programmet kører og der er verificeret.
Test procedure:

1. Vælg opdater bruger.
    (a) Ændre navn.
    (b) Ændre Rolle.
    (c) Ændre Id.
2. Vælg afslut.

Testdata: Bruger data.
Expected result: Brugeren er opdateret.
Actual result: Expected result.
Status: Succes.
Tested by: Frederik.
Date:22/02/18
Test Environment MacOS High Sierra

TC4: Slet bruger.
Preconditons: Programmet kører og brugeren er verificeret.
Test procedure:

1. Vælg slet bruger.
2. Vælg bruger der skal slettes.
3. Vælg vis brugere.

Test data: Slet bruger
Expected result: Brugeren indgår ikke længere på listen over hvis brugere.
Actual result: Expected result.
Status: Succes.
Tested by: Frederik.
Date:22/02/18
Test Environment MacOS High Sierra

4.2 JUnit

Der er desuden lavet en række JUnit tests til vores metoder. De er alle sammen succesfulde, dog er der ikke lavet negative tests. Vores JUnit viser at vores use cases og metoder er implementeret så det fungerer ved brug af vores program, og der kan hentes og gemmes i datalaget som ønsket.

- **Test Edit Name:** Tester om der kan ændres navn og at denne bliver gemt i datalaget.
- **Test Edit Password:** Tester om der kan ændres password og at denne gemmes korrekt og kan anvendes.
- **Test Edit Roles:** Tester om man kan tilføje roller efterfølgende.
- **Test Delete User:** Tester om bruger er slettet og fjernet fra arrayet.

5 Konklusion

Vi er endt op med en et fungerende program i hvilket man kan tilføge brugere, ændre i brugere, vise brugerne på en liste samt slette brugere. Bruger arrayet bliver gemt i en tekstfil, og kan derfor vises igen selvom programmet har været lukket ned. Der kan tildeles flere roller til hver bruger, og passwordet bliver autogenereret ud fra DTUs regler. Programmet er gennemtestet med JUnit og aktør tests. Vi har arbejdet ud fra GRASP principperne og Unified Process. Vi har fået implementeret alle vores krav og use cases.

6 Glossary

Glossary	**Beskrivelse**
Persistent Storage	Kan gemme til en fil.
Non Persistent Storage	Gemmer kun når programmet kører
Generate Password	Genererer password ud definerede karaktersæt.
Initialer	Der genereres initialler på baggrund af forbogstaver i navnene.
Setters	Setter attributterne til noget bestemt.
TUI	*Text User Interface* Det visuelle tekstbaserede view.
DTO	*Data Transfer Object* Vores user objekter.
Unified Process (UP)	Arbejdsmetoden til dette forløb.