CDIO D1

02324 Advanced Programming

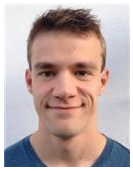
Project title: CDIO D1

Group number: 20

Deadline: Saturday 25th February 2017 - 4:59

Version: 1.0

This report contains 19 pages, including this page.

Student nr., Surname, First name	**Signature**
S165248, Gadegaard, Theis 	_____
S165208, Højsgaard, Tobias André 	_____
S165209, Jønsson, Danny 	_____
S165227, Petersen, Gustav Hammershøi 	_____
S165237, Poulsen, Joakim Thorum 	_____
S145005, von Scholten, Frederik 	_____

Work Distribution

24 February 2017							
Participant	**Analysis**	**Design**	**Impl.**	**Test**	**Doc.**	**Other**	**Total Hours**
Gadegaard, Theis	2	2	12			0,75	**16,75**
Højsgaard, Tobias André	2	2	3		2,5	0,75	**10,25**
Jønsson, Danny				2,5	3		**5,5**
Petersen, Gustav Hammarshøi	2	2	9		2	0,5	**15,5**
Poulsen, Joakim Thorum	1		0,5		2		**3,5**
von Scholten, Frederik	0,5			4,5	1	2	**8**
Total	**7,5**	**6**	**24,5**	**7**	**10,5**	**4**	**59,5**

Resumé

The objective of this project is to develop a user-administration system with a simple textual user interface. This report's purpose is to provide the necessary documentation to give an understanding behind the thought process of the design, implementation and testing of the program.

Table of Contents

1 Inception

1.1 Vision

Our task is to write a user administration module with a simple TUI. It shall be used to administrate users/operators.

2 Analysis

2.1 Requirement specification

2.1.1 Functional requirements

1.1	The program shall have a menu with the following options available 1. Add users 2. Show users 3. Delete users 4. Assign user roles 5. Update users 6. End program
1.2	The program shall auto-generate a password according to following rules 1. 6 or more characters 2. with at least 3 of the following 4 categories a. lowercase letters (a to z) b. uppercase letters (A to Z) c. digits (0 to 9) d. special characters {'.', '-', '_', '+', '!', '?', '='}
1.3	The program must be able to assign users the following roles: Admin, Pharmacist, Foreman, Operator
1.4	Users shall only have access to features allowed to the users assigned role
1.5	The program shall have a simple Text User Interface (TUI)

2.1.2 Usability requirements

	None

2.1.3 Reliability requirements

	None

2.1.4 Performance requirements

	None

2.1.5 Supportability requirements

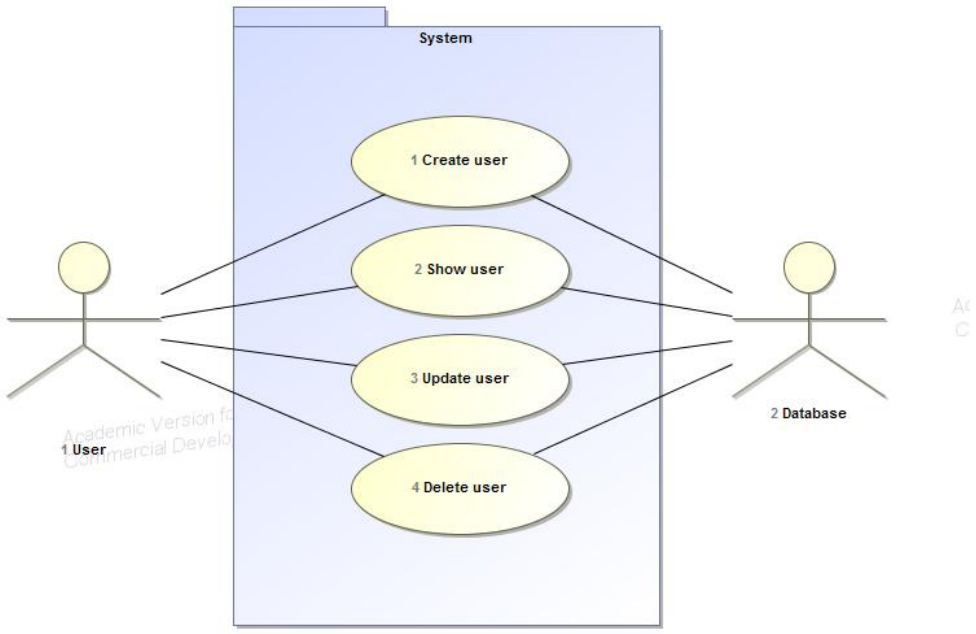
5.1	The program shall store user information for future use

2.1.6 Additional requirements

6.1	The system must have implemented a finished version of the UserDTO (User Data Transfer Object) class, which has been described in the vision document.
6.2	The interface to the data access layer of the system has been defined beforehand. The system must implement the IUserDAO (User Data Access Object) interface described in the vision document.
6.3	The system must have implemented some form of persistent data storage. This can be either a file system or a database.

3 Design

3.1 Use case diagram



3.2 Use cases

UC1 Create user

Use case ID:	UC1
Use case name:	Create user
Description:	A function to create new users which in turn gets stored in a server database
Primary actors:	User
Secondary actors:	Database
Preconditions:	None
Postconditions:	1. A new user is successfully created in the database
Main flow:	1. User initiates create user process

	2. User inputs required information into appropriate inputs and ends user creation process
Alternate flows:	1. User inputs invalid or wrong syntax in inputs 2. An error is thrown

UC2 Show users

Use case ID:	UC2
Use case name:	Show users
Description:	The user has the ability to see all existing users in the server database
Primary actors:	User
Secondary actors:	Database
Preconditions:	None
Postconditions:	1. All users currently in the server database is shown
Main flow:	1. User requests for users to be shown
Alternate flows:	None

UC3 Update user

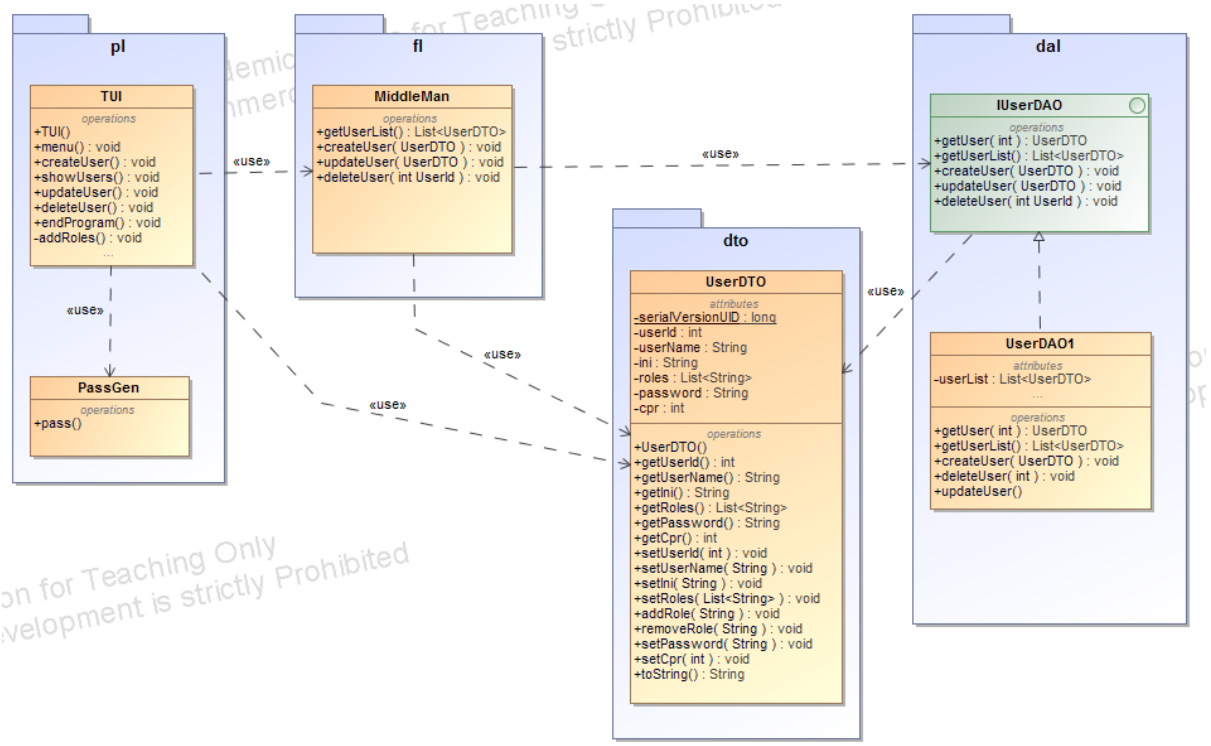
Use case ID:	UC3
Use case name:	Update user
Description:	Gives the user the ability to change data about any user.
Primary actors:	User
Secondary actors:	Database
Preconditions:	1. The database is running
Postconditions:	A field of data has been changed in the database
Main flow:	1. The user selects the “update user” option 2. The user selects the data that they wish to change 3. The user writes in what new data should be assigned to the given field

Alternate flows:	1. The input given by the user does not match the datatype of the selected data 2. An exception is thrown telling the user that their input is invalid 3. The user is asked to give a new input

UC4 Delete user

Use case ID:	UC4
Use case name:	Delete user
Description:	Gives a user the ability to delete own user from the server database
Primary actors:	User
Secondary actors:	Database
Preconditions:	1. User exists in the server database
Postconditions:	1. User is removed from the server database
Main flow:	1. User initiates deletion process 2. User completes deletion process
Alternate flows:	None

3.3 Analysis class diagram



TUI is responsible for handling user inputs and showing the values returned from the data layer in a human-readable format. It has methods for each of the use cases described in section 3.2.

The MiddleMan class is responsible for receiving the data transfer objects from the TUI and passing them along to the data layer.

UserDAO1 is responsible for storing our users. This implementation of our data layer stores the information as an arraylist.

UserDTO is a class made for transferring all the data needed for a user over the internet. Any time we move data between our presentation/functional and data layers, we use this object.

PassGen is used to generate a random password for new users.

4 Implementation

4.1 UserDAO1

We have created a data access object which implements the interface IUserDAO. It is only a relatively simple, nonpersistent implementation, which is built with an ArrayList, but it works as intended. Its biggest flaw is that the data disappears when the program is closed.

4.2 TUI / Middleman

We use a switch/case integrated within a while loop to determine which option the user takes. The switch is made on a string input, so we do not need to worry about the user not typing an int and the menu will simply loop if the user types something that is not an option.

Of the methods available inside the TUI class, the Update user method was the most difficult to implement. We use an input from the user to determine the userId of the user that needs to be updated after which we go down into the data access layer, search for a UserDTO with this id and return it to the middle man class, where the UserDTO can be updated with the relevant information. Once the update is done, the UserDTO that was returned is overwritten by the newly updated UserDTO. This means we do not have to make any new methods in our data access layer.

Most of the other methods were simple to implement with their main challenge being input handling.

4.3 PassGen

Since the password for each created user should be generated, and follow some specific requirements made by DTU, have we gone and made a password generator class. This class has one method which when called upon will create a 10 letter long password with 3 random lowercase letters, 3 random uppercase letters and 4 random digits all in random order. By doing this we have fulfilled the requirements for passwords set by DTU but the generator could be expanded to include special signs (+,-,/,* etc.) which it dosnt support at the moment.

5 Test

5.1 Unit test

5.1.1 Database Test

Our program uses a non-persistent ArrayList solution as database and is therefore not testable.

5.1.2 Password Generator

The password generation and validity of passwords generated.
The first test (test1) tests the set and getter. We use the set method with a predetermined password and prints it with the get method. If the setter and getter is working properly then we will get a return matching the predetermined password.
Then we check if the generator is able to generate a random password and shows it.

The second test (test2) test the validity of the password generated. According to the rules set by DTU. By asserting the length of the password and checking, and counting the character properties.If a password contains the right amount of legal characters it will be deemed as valid. The test generates and tests 100000 passwords and count the amount of valid and invalid passwords. Finally it assert the expected results (100000) against the actual result.

We are able to test every rule defined på the DTU password ruleset. Though the test does not test for special characters as the generator doesn't generate any. The latest version of the password generator has passed the test with 100000/100000 valid passwords .

5.2 Integration test

Test of the whole integrated system.

5.2.1 Use case tests

TC01A: Create user

Test case ID	TC01A
Summary	Tests to see if creating a user, actually creates a user with the correct data.
Requirements	R1

Preconditions	1. User is at the start menu 2. Desired ID has not been taken
Test Procedure	1. OPTION: User presses 1, to create a new user 2. NAME: User enters a name (2-20 characters) 3. INITIALS: User enters his initials (2-4 characters) 4. CPR: User enters his cpr number (10 digits) 5. ID: User enters his desired ID (integer 11-99)
Test data	1. OPTION: a. {6, 0, a} b. 1 2. NAME: a. {‘a’, “abcdefghijlmnopqr1921”} (1 and 21 characters) b. {“a1, “abcdefghijklmnopq1920”} (2 and 20 characters) 3. INITIALS: a. {‘d’, “bad45”} (1, 5 initials) b. {“ft”, “ftjd”} (2, 4 initials) 4. CPR: a. {123456789, 12345678901, ‘a’} b. {0102001900} 5. ID: a. {10, 100, f} b. {11, 99}
Expected result	1. OPTION: a. “You entered {6, 0, a} which is not a valid choice.” b. (void) *Continues to next data input* 2. NAME: a. “Only 2-20 signs, try again:” b. (void) *Continues to next data input* 3. INITIALS: a. “Only 2-4 initials, try again:” b. (void) *Continues to next data input* 4. CPR: a. “Wrong input, try again:” b. (void) *Continues to next data input* 5. ID: a. “Only a number range 11-99, try again:” b. (void) *Continues to start menu*
Actual result	All of the actual results are as the expected results
Status	Passed
Tested by	Danny
Date	23/02/2017
Test Environment	Eclipse Neon 4.6.2

TC01B: Create user (when userID is taken)

Test case ID	TC01B
Summary	Tests creating a new user with a desired user ID that is already used.
Requirements	R1
Preconditions	1. A user has already been created with the desired userID.
Test Procedure	1. User walks through TC01A and reaches Test Procedure '5'. a. ID: User enters his desired userID.
Test data	1. ID: a. {17} b. {18}
Expected result	1. ID: a. "ID already taken, try again:" b. (void) *Continues to start menu*
Actual result	The actual results are as the expected results.
Status	Passed
Tested by	Danny
Date	23/02/2017
Test Environment	Eclipse Neon 4.6.2

TC02: Show users

Test case ID	TC02
Summary	Tests if the show user option is showing all of the users properly.
Requirements	R1
Preconditions	User is at the start menu
Test Procedure	1. User enters 2 in the menu, to "Show all existing users". 2. Shows all existing users
Test data	
Expected result	Shows following for each and every existing user 1. user id 2. user name 3. user initials 4. user roles 5. cpr is confidential and shall not be shown
Actual result	Everything but the cpr is shown

Status	Passed
Tested by	Frederik
Date	23/02/2017
Test environment	Eclipse Neon 4.6.2

TC03: Update user

Test case ID	TC03
Summary	Tests if update user is able to update an already existing users username, password, initials, cpr, add roles and remove roles
Requirements	R1
Preconditions	User is at the start menu At least one user already exists with a know id
Test Procedure	1. User enter 3 into the menu, to “Update an existing user” 2. Types in the userID the user want to update 3. Selects an option by typing the corresponding number and pressing "enter" a. (1) “Update username” b. (2) “Update password” c. (3) “Update initials” d. (4) “Update CPR” e. (5) “Add role” f. (6) “Remove role” g. (7) “Main menu”
Test data	1. Same test for option 1, 3 and 4 as NAME, INITIALS and CRP in TC01 “Create User” 2. Option (2) “Update password” a. update password as desired 3. Option (5) “Add role” a. try adding each of the permitted roles[Admin, Pharmacist, Foreman, Operator] b. try adding a random none permitted role “Random” c. try adding a permitted role but with different upper- and lowercase letters fx. admin instead of Admin 4. Option (6) “Remove role” a. try removing a role [Admin] b. try removing a permitted role which isn’t granted to the user. c. removing a non permitted role (none existing) 5. Option (7) “Main menu” a. go to main menu using this option

Expected result	1. Same expected results for option 1, 3 and 4 as NAME, INITIALS and CRP in TC01 “Create User”
	2. Option (2) “Update password”
	  a. returns to “Update user” menu after successful updating the user password
	3. Option (5) “Add role”
	  a. returns to “Update user” menu after successful adding a role
	  b. get error message “Invalid role.”
	  c. get error message “Invalid role.”
	4. Option (6) “Remove role”
	  a. returns to “Update user” menu after successful removing a role
	  b. returns to “Update user” menu after being unable to remove the none granted role
	  c. returns to “Update user” menu after being unable to remove the non permitted and non existing role
	5. Option (7) “Main menu”
	  a. returns to “main menu”

Actual result	The actual results are as the expected results.
Status	Passed
Tested by	Frederik
Date	23/02/2017
Test environment	Eclipse Neon 4.6.2

TC04: Delete user

Test case ID	TC04
Summary	Tests to see if the option ‘delete user’ will delete the chosen user
Requirements	R1
Preconditions	User is at the start menu A user has already been created with the userID 17
Test Procedure	1. User enters 4 in the menu, to “Delete user”. 2. User chooses to delete userID 17.
Test data	1. {16, a} 2. {17}

Expected result	1.
	  a. “No user with the given userID was found”
	  b. (void) *Return to main menu*
	2.

	a. “The user was successfully deleted” b. (void) *Return to main menu*
Status	Passed
Tested by	Danny
Date	23/02/2017
Test environment	Eclipse Neon 4.6.2

5.3 Conclusion on test

All functional requirements have been tested and passed beside requirement 1.2.2d and 1.1.6 as they are either too trivial to test (end program) or not yet implemented and therefore not testable. The tested requirements are shown in table 1 and 2.

Table 1: Traceability matrix

	R1.1.1	R1.1.2	R1.1.3	R1.1.4	R1.1.5	R1.1.6
TC01A & TC01B	x					
TC02		x				
TC03			x	x		
TC04					x	

Table 2: Traceability matrix

	R1.2.1	R1.2.2a	R1.2.2b	R1.2.2c	R1.2.2d
PassGenTest	x	x	x	x	

6 Conclusion

Everything has been implemented and works as intended, though there are a few minor details about the password generator - It does not work with special characters, yet - the passwords however does still fulfill the requirements set, except for the requirements involving persistent data storage.

There are also a few features that we'd like to see in our upgrade of the program. For instance a possibility to exit/cancel when creating a user, or manipulating it's data. We would also have liked to implement a data access layer with persistence, like a file system or a database instead of the non-persistent ArrayList solution we currently have, but it is relatively easy to update our system to another data access implementation in the future because of our use of the IUserDAO inteface.

We conclude that the program works after having use case tested all existing methods within the system.