CDIO - 1

User administration tool for managing user information.

Af gruppe 22

Peter El Habr
s165202

Simon Engquist
s143233

Arvid Langsø
s144265

Mikkel Lund
s165238

Jeppe Nielsen
s093905

Mads Stege
s165243

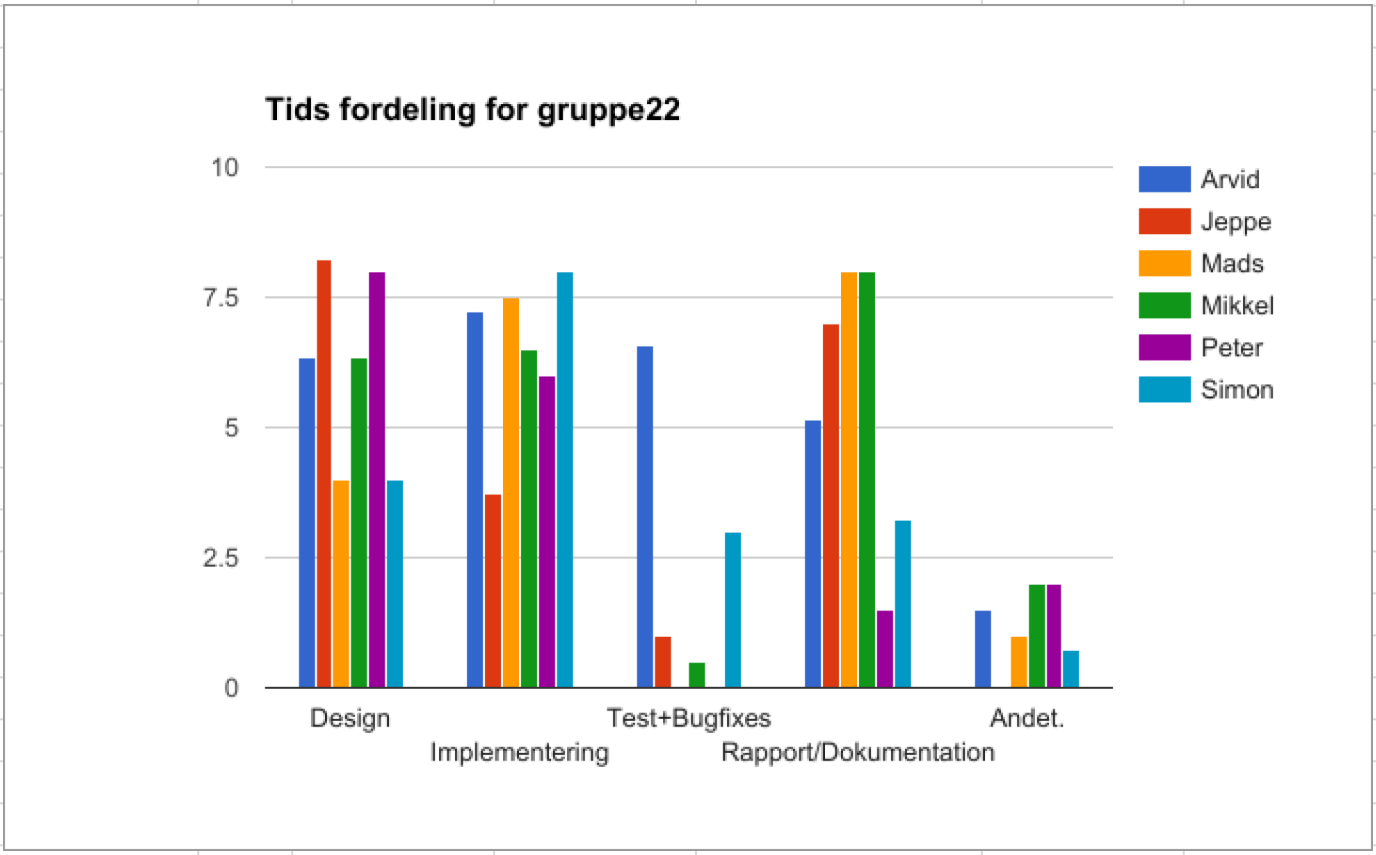
Danmarks Tekniske Universitet
DTU Compute

25. februar 2017 - Kl. 05:00

Side antal (med bilag): 15

Kurser: 02324

	Design	Implementering	Test+Bugfixes	Rapport/Dokumentation	Andet.	Total
Arvid	6,33	7,25	6,56	5,16	1,5	26,8
Jeppe	8,25	3,75	1	7	0	20
Mads	4	7,5	0	8	1	20,5
Mikkel	6,33	6,5	0,5	8	2	23,33
Peter	8	6	0	1,5	2	17,5
Simon	4	8	3	3,25	0,75	19
Total:	36,91	39	11,06	32,91	7,25	127,13



Content

1 Introduction

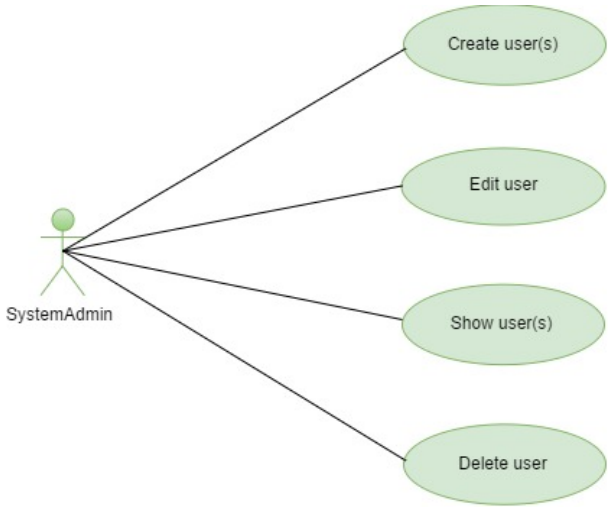
This report describes the design and testing of a user administration tool, for managing user-data in a data file. The tool will be developed in java, and shall implement a textual user interface. The tool shall be runnable from the commando prompt on systems with a windows operating system installed. The tool is expected to be able to do the following:

- Create new users, and save them with the other users in a data-file.
- Edit existing user-information.
- View information on users in the system.
- Delete existing user from the data.

A full requirement specification can be found in the appendix.

2 Design

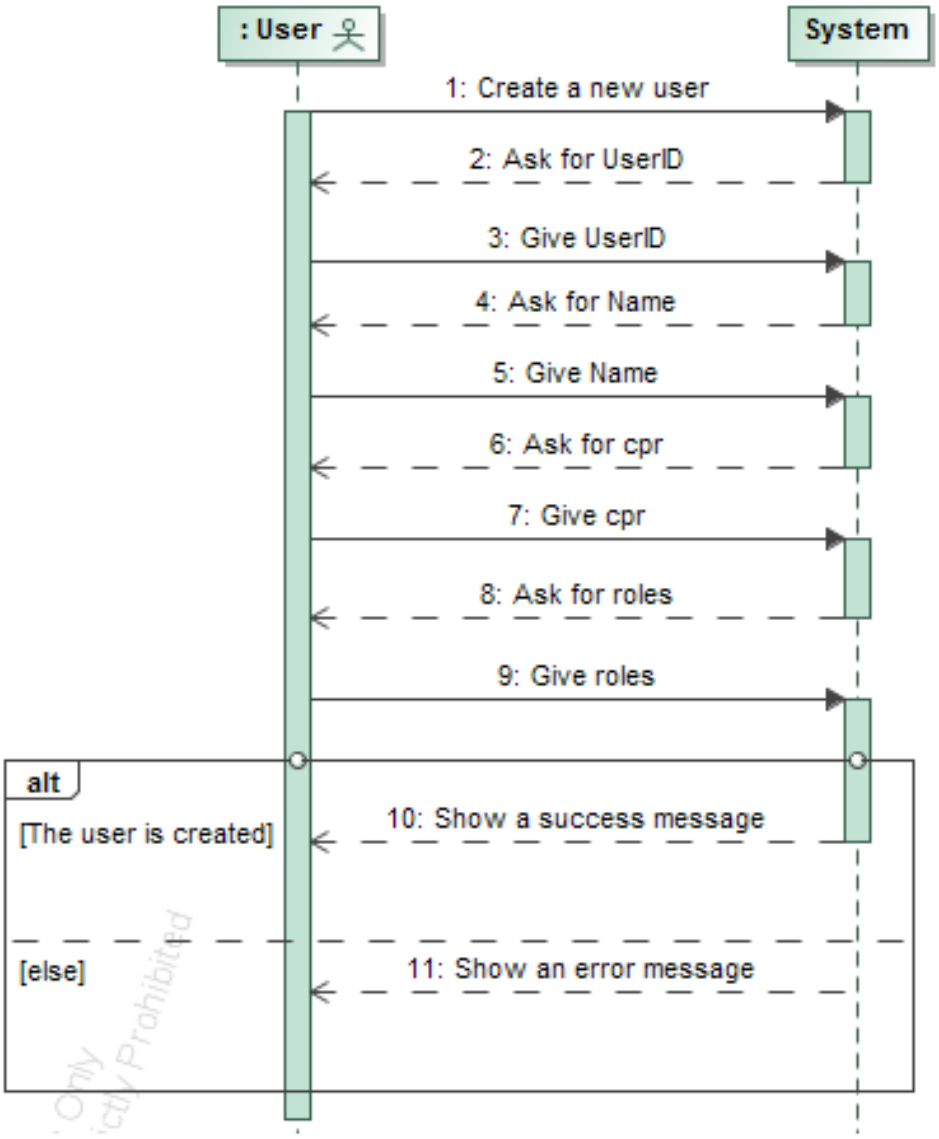
2.1 Use-Case diagram



Figur 3: Use-Case diagram for the user administration tool.

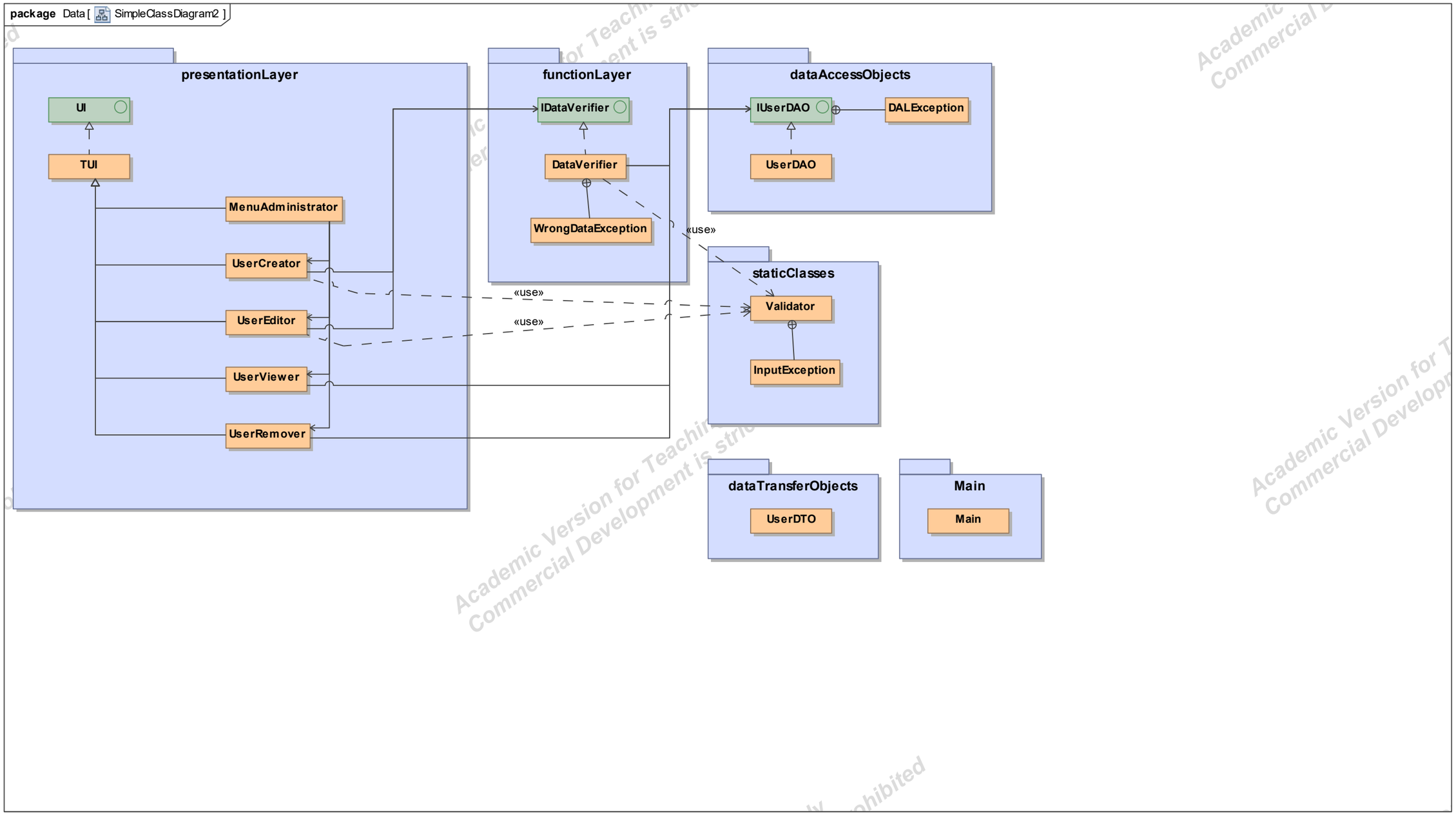
The above Use Case diagram shows the different use cases. A detailed use case description can be found in the appendix.

2.2 System Sequence Diagram - UserCreator



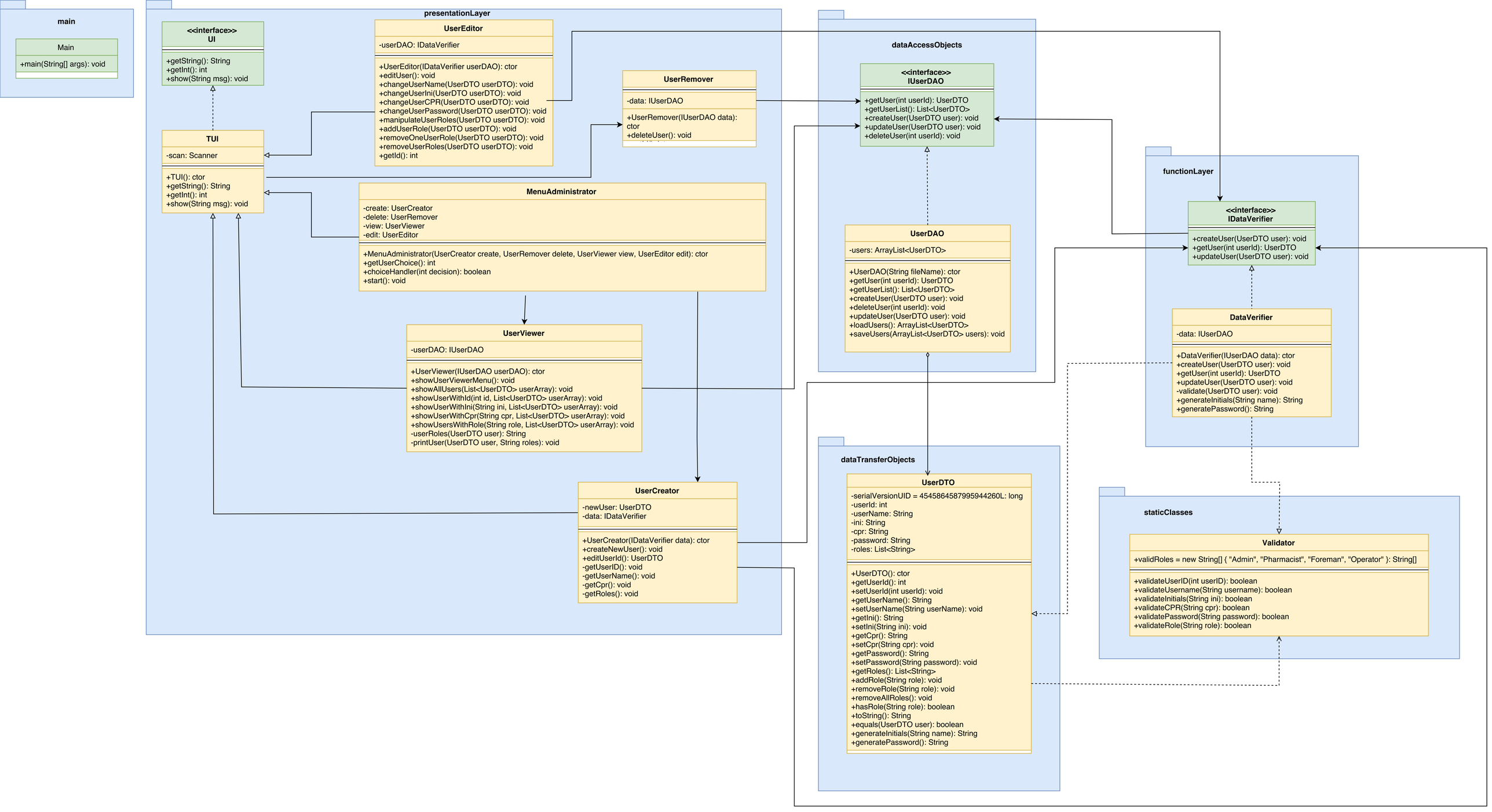
Figur 4: System sequence diagram for creation of a user.

2.3 System Class Diagram

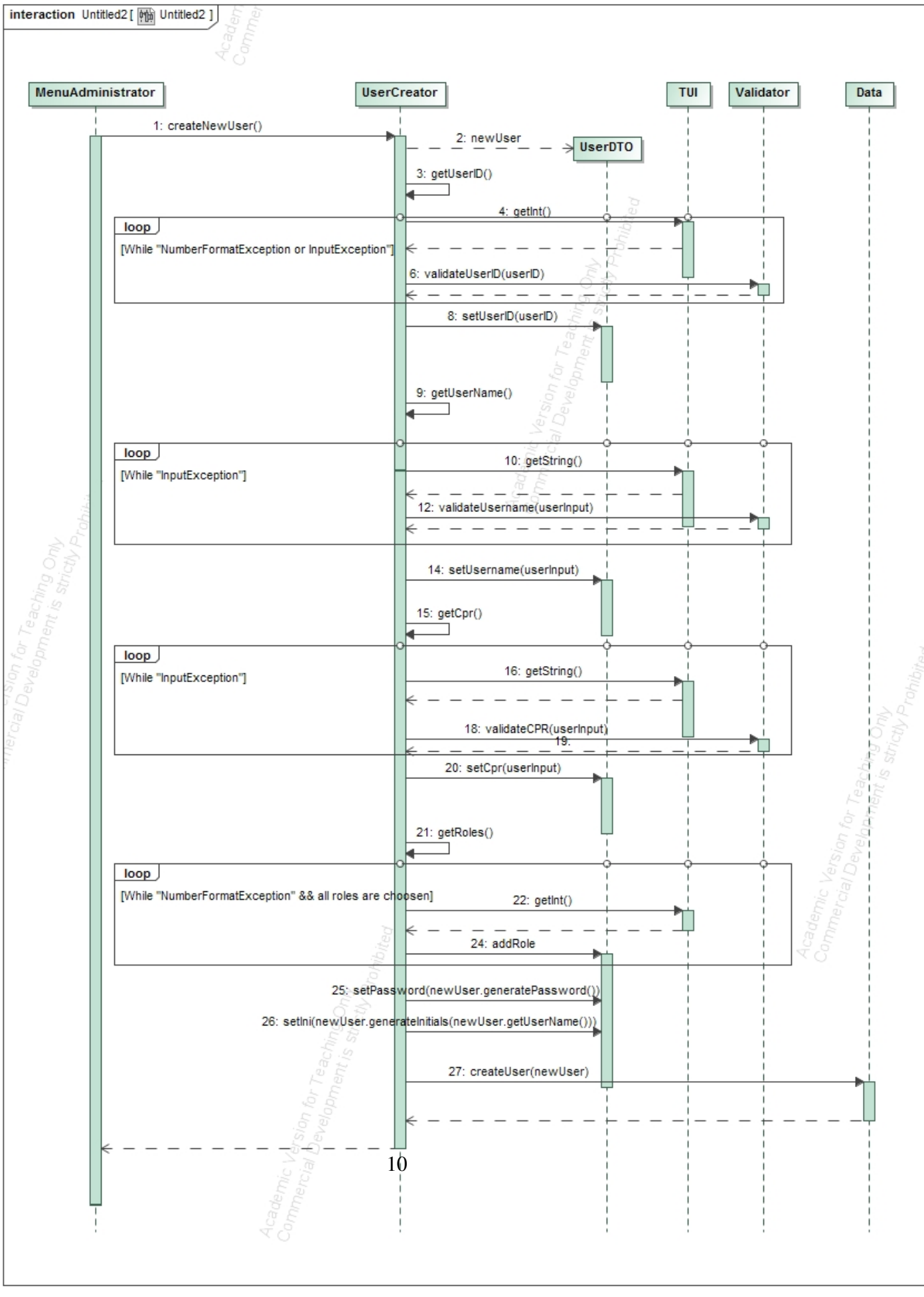


The system class diagram displays a simple overview of the different classes and their relations.
The UserDTO class is used by the other classes to transfer data.
The Main class contructs instances of the classes.
To retain simplicity, the arrows from these two classes have been left out.

2.4 Class Diagram


main
Main
+main(String[] args): void
presentationLayer
<<interface>>
UI
+getString(): String
+getInt(): int
+show(String msg): void
UserEditor
-userDAO: IDataVerifier
+UserEditor(IDataVerifier userDAO): ctor
+editUser(): void
+changeUserName(UserDTO userDTO): void
+changeUserIni(UserDTO userDTO): void
+changeUserCPR(UserDTO userDTO): void
+changeUserPassword(UserDTO userDTO): void
+manipulateUserRoles(UserDTO userDTO): void
+addUserRole(UserDTO userDTO): void
+removeOneUserRole(UserDTO userDTO): void
+removeUserRoles(UserDTO userDTO): void
+getId(): int
UserRemover
-data: IUserDAO
+UserRemover(IUserDAO data): ctor
+deleteUser(): void
TUI
-scan: Scanner
+TUI(): ctor
+getString(): String
+getInt(): int
+show(String msg): void
MenuAdministrator
-create: UserCreator
-delete: UserRemover
-view: UserViewer
-edit: UserEditor
+MenuAdministrator(UserCreator create, UserRemover delete, UserViewer view, UserEditor edit): ctor
+getUserChoice(): int
+choiceHandler(int decision): boolean
+start(): void
UserViewer
-userDAO: IUserDAO
+UserViewer(IUserDAO userDAO): ctor
+showUserViewerMenu(): void
+showAllUsers(List<UserDTO> userArray): void
+showUserWithId(int id, List<UserDTO> userArray): void
+showUserWithIni(String ini, List<UserDTO> userArray): void
+showUserWithCpr(String cpr, List<UserDTO> userArray): void
+showUsersWithRole(String role, List<UserDTO> userArray): void
-userRoles(UserDTO user): String
-printUser(UserDTO user, String roles): void
UserCreator
-newUser: UserDTO
-data: IDataVerifier
+UserCreator(IDataVerifier data): ctor
+createNewUser(): void
+editUserId(): UserDTO
-getUserID(): void
-getUserName(): void
-getCpr(): void
-getRoles(): void
dataAccessObjects
<<interface>>
IUserDAO
+getUser(int userId): UserDTO
+getUserList(): List<UserDTO>
+createUser(UserDTO user): void
+updateUser(UserDTO user): void
+deleteUser(int userId): void
UserDAO
-users: ArrayList<UserDTO>
+UserDAO(String fileName): ctor
+getUser(int userId): UserDTO
+getUserList(): List<UserDTO>
+createUser(UserDTO user): void
+deleteUser(int userId): void
+updateUser(UserDTO user): void
+loadUsers(): ArrayList<UserDTO>
+saveUsers(ArrayList<UserDTO> users): void
functionLayer
<<interface>>
IDataVerifier
+createUser(UserDTO user): void
+getUser(int userId): UserDTO
+updateUser(UserDTO user): void
DataVerifier
-data: IUserDAO
+DataVerifier(IUserDAO data): ctor
+createUser(UserDTO user): void
+getUser(int userId): UserDTO
+updateUser(UserDTO user): void
-validate(UserDTO user): void
+generateInitials(String name): String
+generatePassword(): String
dataTransferObjects
UserDTO
-serialVersionUID = 4545864587995944260L: long
-userId: int
-userName: String
-ini: String
-cpr: String
-password: String
-roles: List<String>
+UserDTO(): ctor
+getUserId(): int
+setUserId(int userId): void
+getUserName(): String
+setUserName(String userName): void
+getIni(): String
+setIni(String ini): void
+getCpr(): String
+setCpr(String cpr): void
+getPassword(): String
+setPassword(String password): void
+getRoles(): List<String>
+addRole(String role): void
+removeRole(String role): void
+removeAllRoles(): void
+hasRole(String role): boolean
+toString(): String
+equals(UserDTO user): boolean
+generateInitials(String name): String
+generatePassword(): String
staticClasses
Validator
+validRoles = new String[] { "Admin", "Pharmacist", "Foreman", "Operator" }: String[]
+validateUserID(int userID): boolean
+validateUsername(String username): boolean
+validateInitials(String ini): boolean
+validateCPR(String cpr): boolean
+validatePassword(String password): boolean
+validateRole(String role): boolean

2.5 Sequence Diagram - UserCreation



The depicted System Sequence Diagram for the creation of a new user. The diagram shows the incoming and outgoing messages sent between the user of the system, and the system itself.

The system contains multiple instances of data verifications, ensuring that the inserted user inputs are valid for the specific types of data.

3 Test

3.1 Test cases

1. Create user - Test that user can be created with valid inputs, and not with invalid inputs.
2. View user - Test if all relevant information on a user can be viewed.
3. Edit user - Test if user information can be edited to valid values.
4. Delete user - Test if a user can be deleted.
5. Exit program -Testing if the program exits.

3.2 JUnit

1. **DataVerifierTest** Tests if a user's initials and password are generated correctly.
2. **ValidatorTest** Tests if the Validator class validates the user input correctly.
3. **UserDAOTest** Tests if the data access layer functions correctly, can users be viewed, added, edited and deleted. Also test if users can be saved and loaded from a file.

4 Conclusion

During this project a user administration module has been developed. The user data are stored locally in a text file, and the module features an intuitive and logical Textual User Interface.

All of the requirements have been implemented, tested and makes for an easy to use user experience.

The system has been testet using the black box method. Any errors found have been corrected and retested. This has improved the system quality and operation. A few unit tests has also been made on specific parts of the program that are more complex.

This project has proven to be a solid foundation upon which to build further iterations, and expand the uses of the data file.

5 Appendix

5.1 Requirements specification

The list of requirements has been setup using the FURPS+ model.

5.1.1 Functionality

- F1: The system shall be runnable on a DTU Databar computer system, operating a Windows 10 64-bit operating system.
- F2: The system shall be able to create, delete, edit and show a list of all registered users.
  - F2.1: All users will have the following data assigned to them at creation:
    - An unique userID (ranging from numerals 11 to 99).
    - A userName (ranging from 2 to 20 characters).
    - A pair of Initials (2 to 4 characters).
    - A CPR number (a valid cpr number consisting of 10 numbers in total).
    - A password (Following the same rules as DTU's login systems).
    - A list of zero or more roles (Admin, Pharmacist, Foreman and Operator).
- F3: The system shall have an easy to read and use Textual Interface, that exist in a console.
  - F3.1: The console shall start upon opening the program, and shall close when asked to.
- F4: The system shall save every user to a persistent datalayer.
  - F4.1: The data will be saved as a local file.
- F5: The database shall be able to save, and store 89 individual users simultaneously (each with an individual userIDs ranging from 11 to 99).

5.1.2 Usability

- U1: The console shall present an easily read and understandable list of available commands and requirements upon starting.
  - U1.1: The console shall provide adequate information when creating, or editing a user.
  - U1.2: The console shall provide a detailed description of the commands selected, so as to avoid confusion and errors.

  - U1.3: The console shall provide users with the possibility to cancel a selected command (i.e. cancel the deletion of an user, or removing all of the roles from a specified user).

5.1.3 Reliability

- R1: The system shall be able to recognize and handle unexpected inputs, return an error message, and continue afterwards without closing down unexpectedly.
  - R1.1: The system shall also be able to distinguish between correct and wrong userinputs and handle each accordingly.
  - R1.2: Upon entering an incorrect input, the user should be prompted to enter a new, valid input.
- R2: The program shall start and operate correctly in 99,9% of all start-up operations.
- R3: The program shall be able to run and operate for at least 24 hour before requiring a system restart.
- R4: All user data shall be saved, and must be available for the next start-up sequence.

5.1.4 Performance

- P1: The system shall have no more than a 0,5 second delays between recognizing a userInput, and executing the proper action.
- P2: The system shall have no more than a 2 second delays when saving users to a data file.

5.1.5 Supporting

- S1: All code must be easily maintainable.
  - S1.1: This should be achieved by following the MVC model and the principles of GRASP.
- S2: All output to the user must be easily understandable. The language of this output should not disturb the readability.

5.1.6 Constraints (+)

- +1: The TUI and all related code shall be programmed in Java 8, and shall require the use of a command prompt (Windows Operating systems).
- +2: The whole program - including the file database - shall fill no more than 1024 MB of disc storage.

5.2 Use-case Description

Use-case 01 - Create User(s):

- The SystemAdmin selects to create a new user.
- The system enters the Create User mode, and asks for a series of information for the new user.
- The SystemAdmin enters the relevant information, and confirms.
- The system saves the newly created user into the database, and exits Create User mode.

Use-case 02 - Edit User:

- The SystemAdmin selects to edit an existing user.
- The system asks the SystemAdmin to enter a valid userID.
- The SystemAdmin enters a valid userID.
- The system recognizes the userID, retrieves the user information and asks the SystemAdmin how he wishes to proceed.
- The SystemAdmin selects a specific way to manipulate the data.
- The system asks for a valid input in relation to the desired data edit.
- The SystemAdmin enters an input.
- The System recognizes the input as a valid input, and changes the related user information. The system then prints out a confirmation of the changed user data to the SystemAdmin.
- The SystemAdmin choses to exit the Edit user mode, and returns to the prior menu.

Use-case 03 - Show User(s):

- The SystemAdmin chooses to show all the currently registered users in the database.
- The system enters Show User mode, and asks the SystemAdmin which characteristic to show the user(s) by, i.e.by their userID, initials, CPR number or role.
- The SystemAdmin chooses a valid option, and enters an input.
- The system retrieves the relevant information, and returns the related user-information.
- The SystemAdmin can now choose to make a new search in the database, or to close the Show User mode, and return to the prior menu.

Use-case 04 - Delete user:

- The SystemAdmin chooses to delete an existing user in the database.
- The system enters Delete User mode, and asks the SystemAdmin for a userID to delete.
- The SystemAdmin enters a valid userID.
- The system searches the database, and if the userID is found, deletes the related user.
- The system then returns a confirmation message, saying that it has found the userID, and deleted the related user.