Smart Doc

Michał Ciebien(266908)

Eduard Costea(266078)

Remedios Pastor Molines(266100)

Boris Šídlo(251341)

Mihai Tirtara(266097)

Supervisors: Mona Wendel Andersen, Joseph Chukwudi Okika, Ib Havn, Knud Erik Rasmussen, Steffen Vissing Andersen

ICT Engineering

2nd Semester

08/06/2018

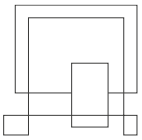
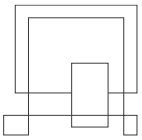
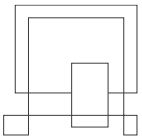


Table of Contents



Abstract

The purpose of the project was to develop a system that would help improve the treatment information exchange between patients and doctors in the hospital. The system was supposed to store data about patients and doctors working in the hospital and allow the doctors writing prescriptions, recommendations, updating appointments and writing diagnosis for patients. The paper begins with Requirements part that lists out the requirements for the system and show how they were derived into Use Cases. The Analysis part analyzes requirements that were set for the system and presents the analysis in form of various UML Diagrams. The Design part shows how the system was Designed with respect on what was derived from the Analysis part. It includes the details of what technologies were used in the system as well as describe the patterns and data persistence used. The Implementation part shows examples of code and describe most interesting parts of it. Test chapter describes the testing of the whole system. Results and Discussion summarize what was done in the project and explain what could be done better. Next part – Conclusions analyze how each part included in the paper valued in the final result. Lastly the Project Future part discusses how the system could be developed in the future.

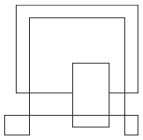


1. Introduction

The healthcare administration has its roots in the early 20th century when the number of hospitals grew significantly. In this way, creating the necessity of having trained staff in administrative tasks. Thus, the first modern day "health systems management program" was born in 1934 at the University of Chicago. (Healthcare Administration, 2012). Furthermore, in the decade of the 50's, Homer Warner (American cardiologist) arose with the idea of introducing computers in hospitals to improve patients' care (Owens-Liston, 2012).

Hospital information systems are part of health informatics which tries to fulfil the administrative needs of hospitals. Offering to the patients a convenient manner to view their personal healthcare information and to the doctors a reliable method to diagnose or prescribe medicine hence facilitating the interaction between the two parties.

Nowadays we can see that the industry is shifting from reactive sick care to proactive health management with preventive recommendations from doctors, thus switching from fragmented niche care to a cross-continuum care system and from reward for volume to reward for quality, efficiency and safety(Cerner 's Senior Vice President of Population Health John Glaser , 2017). The approach to healthcare is radically changing worldwide. Where we used to focus on understanding and treating disease, we now realize the power of providing individualized, patient-centric care (Diana Venter, 2018). Focusing on the patients demands doctors can provide them with personalized recommendations and easy access to view their prescription or upcoming appointments. Furthermore, information privacy is an important feature of hospital systems assuring that the users have their health records stored and encrypted.



2. Requirements

The system's requirements that also refer to its delimitation and problem, are divided between functional requirements an non-functional requirements, which represent the problems the system will need to handle.

2.1. Functional Requirements

1. Administrator should be able to create doctor profiles
2. Administrator should be able to delete doctor profiles
3. Patient should be able to introduce their data
4. Each system user should be able to access their authorized data
5. Patients should be able to see their personal information
6. Patient should be able to see ones upcoming appointment
7. Patient should be able to see ones treatment data
8. General doctor should be able to assign patient to a specific doctor.
9. Doctor should be able to write diagnose for the patient
10. Doctor should be able to see patient's personal information
11. Doctor should be able to schedule upcoming appointment
12. Doctor should be able to prescribe medicine to patient
13. The system should store data for each entity that include the cpr, first name, last name, phone number, email, gender, entity type, date of birth
14. The system should store the data about the specialty of the doctor
15. The system should store data for every patient's recommendations, prescription, diagnosis and upcoming meeting

2.2. Non-Functional Requirement

1. The system has to be implemented in JAVA.
2. The system has to be using PostgreSQL for the data persistence
3. The system has to be running on Windows 10
4. The system must be a client-server-system

2.3. Use cases

The following use case diagram and use case descriptions show what the system should do, and how the system users should interact with it to achieve certain goals. They are shown here as an example of all the possible actions the system user can take. All the use case diagrams and their descriptions can be found in the Appendix A.

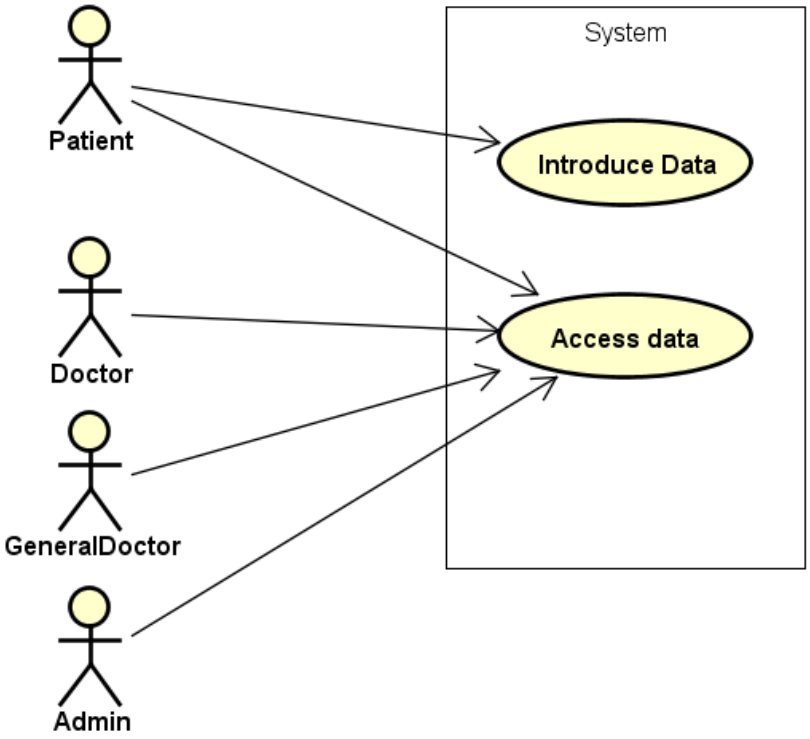
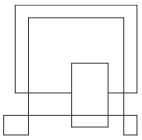


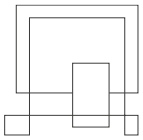
Figure 1 Use Case Diagram for introducing data/accessing data

The above use case diagram (Figure 1) show different parties (Patients, Doctors, General doctors, Administrator) which goal is to access their secured data, or in the case of patients also to introduce the data in order to be registered in the system. The following use case descriptions (Figure 2 and Figure 3) for these two goals describe how can that be achieved.



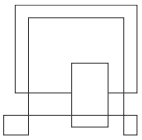
ITEM	VALUE
UseCase	Access data
Summary	The user can acces his data by logining into the system and the system will display his personal information which include his name, date of birth, cpr and e mail.
Actor	Patient Doctor GeneralDoctor Admin
Precondition	The user must already have an account created.
Postcondition	The system display the homepage with his personal information.
Base Sequence	1. The user enter his username and password. 2.The user submit his username and password. 3.The system verifies his username and password. 4.The system validate his username and password. 5.The system display the homepage with the personal information.
Branch Sequence	
Exception Sequence	1.a The user enter the wrong username or the wrong password. 1.The system prompts "Wrong username or password" 2. Use case return to step 1 1.b The user forgot his password 1.The system prompts the user to introduce his username 2.Use case return to step 1
Sub UseCase	
Note	

Figure 2 Use Case description for accessing data



ITEM	VALUE
UseCase	Introduce Data
Summary	The patient will be registered in the system by filling the fields with his person al information which include his first name, last name, cpr,date of birth,email and choosing his username and password.
Actor	Patient
Precondition	The patient must not be already registered.
Postcondition	The patient will be registered in the system and added to the database.
Base Sequence	1.The user enter his personal information(first name, last name, cpr,phone nu mber, date of birth,username,password, email,gender) 2.The user submit his personal informations 3.The system display "Succes "
Branch Sequence	
Exception Sequence	1.a The user enter an existing username 1.The system prompts "Existing username" 2.Use case return to step 1 2.a 1.The user clear the information 2.Use case return to step 1
Sub UseCase	
Note	

Figure 3 Use Case description for new patient registering



3. Analysis

The analysis of the system displays how the system was analyzed before it was designed, implemented and tested. This part of the paper shows how the tasks that the system should handle were analyzed, as well as shows the process that was followed to precisely set the goals for interaction with the system users. The analysis part mainly emphasizes on what are the problems the system is facing, and trying to find solutions for them keeping in mind the delimitations for it. The problems were stated in the Project Description, thus the analysis bases on what are the answers for these questions. The questions are as follows:

- The main question is **how to improve information transfer between doctors and patients**?
- How the healthcare information can be easily accessed by the patients and doctors?
- What are the information patients would like to know?
- How can we make sure the information will be stored?
- Who will insert the patients' information?
- Why do patients need the better access to their healthcare information?
- How the information will be presented?
- The delimitation for the project were also stated in the Project Description, and are as follows:

One of project's delimitation are:

- Only doctors can insert patients' information
- The system will store personal information about the patients as well as recommendations from doctors
- Patients will only be able to read their information

3.1 Activity Diagrams

Activity diagrams were used to visualize most complex Use Cases scenarios as well as to make sure they can be easily understandable. The following Activity Diagram is shown as an example of the scenario for the Use Case nr 8.

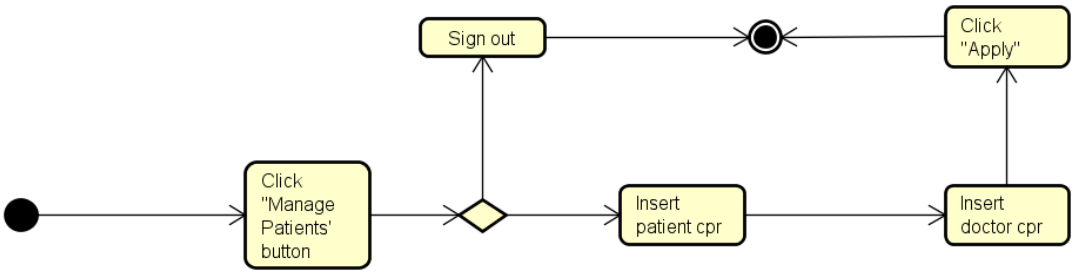


Diagram 1 Activity Diagram for Use case 8

3.2 System Sequence Diagrams

For a better view of the Use Cases, the following System Sequence Diagrams (SSDs) were made to better visualize the exact path of what the system does while the user interacts with it. All of the System Sequence Diagrams can be found in the appendices.

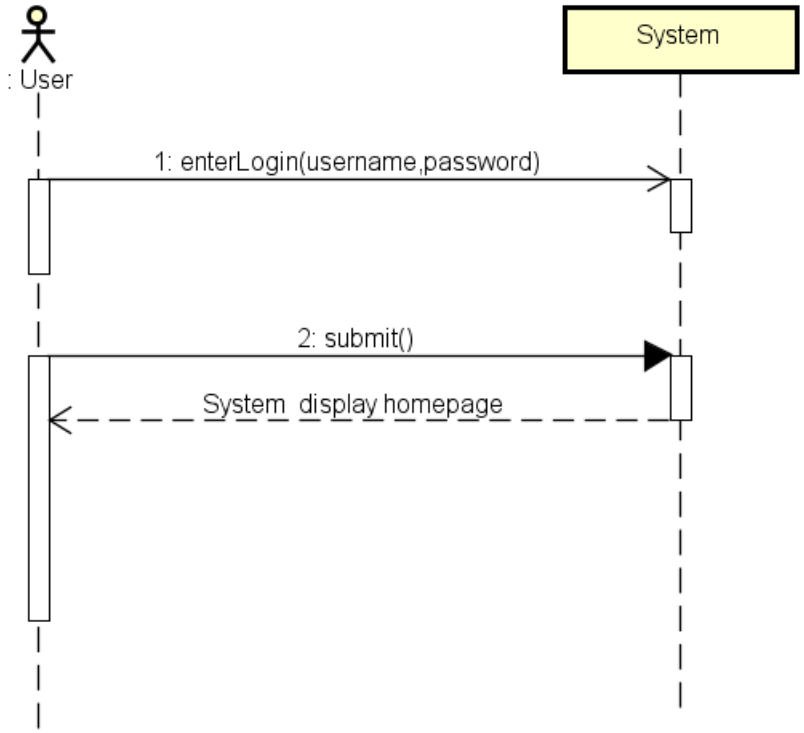


Figure 4 SSD for system users accessing data (Use Case nr 4)

In the above figure it is the shown the sequence diagram for accessing data which represent how users can access their data. The first step is for the user to introduce the username and password then submitted it subsequently the system responds by displaying the specific homepage for every user. In the case when the user enters a wrong username or password the system will prompt the user to reintroduce his credentials as it can be seen in the picture below.

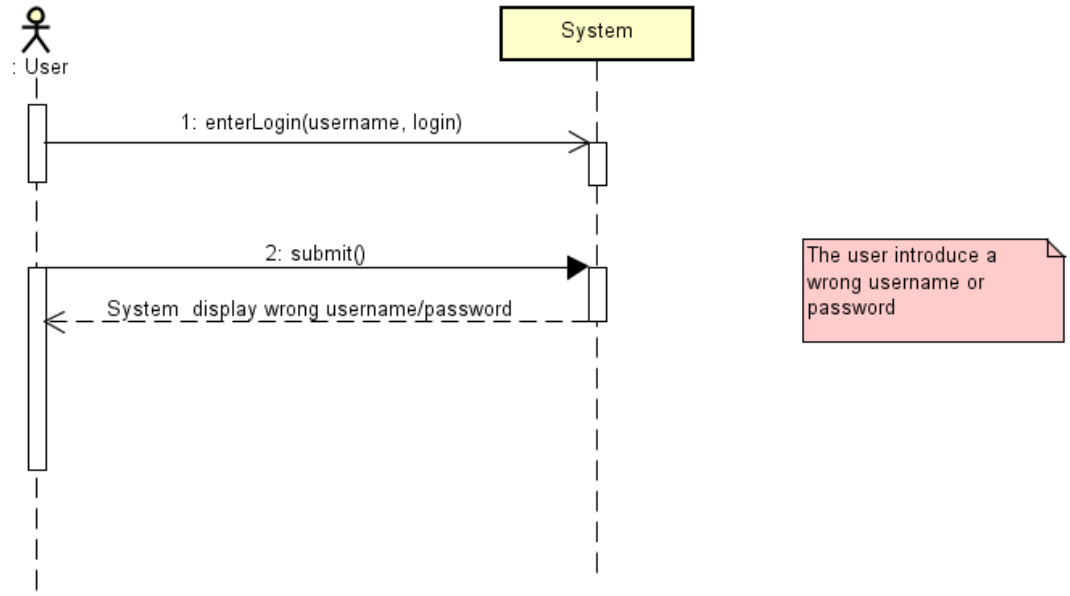


Figure 5 SSD for accessing data with wrong username or password

The following SSD (Figure 6) shows how the system behaves with a General Doctor as an actor, whose goal is to assign a patient to a specific doctor. The sequence starts with the *1:clickManageButton()* action (not by accessing data, as this is indicated by the **precondition** in the Use Case for this figure), the system as a return displays a new window with the input for the CPR number of a patient. After the General Doctor inserts CPR of patient on *2:insertCpr(patientCpr)* and clicks the apply button on *3:clickApply()*, the system then displays a new window with a input for doctor's CPR. The following actions are *4:insertCpr(doctorCpr)*, where General Doctor inserts the CPR of the specific doctor and *5:clickApply()*, after which the system displays a popup with a success message.

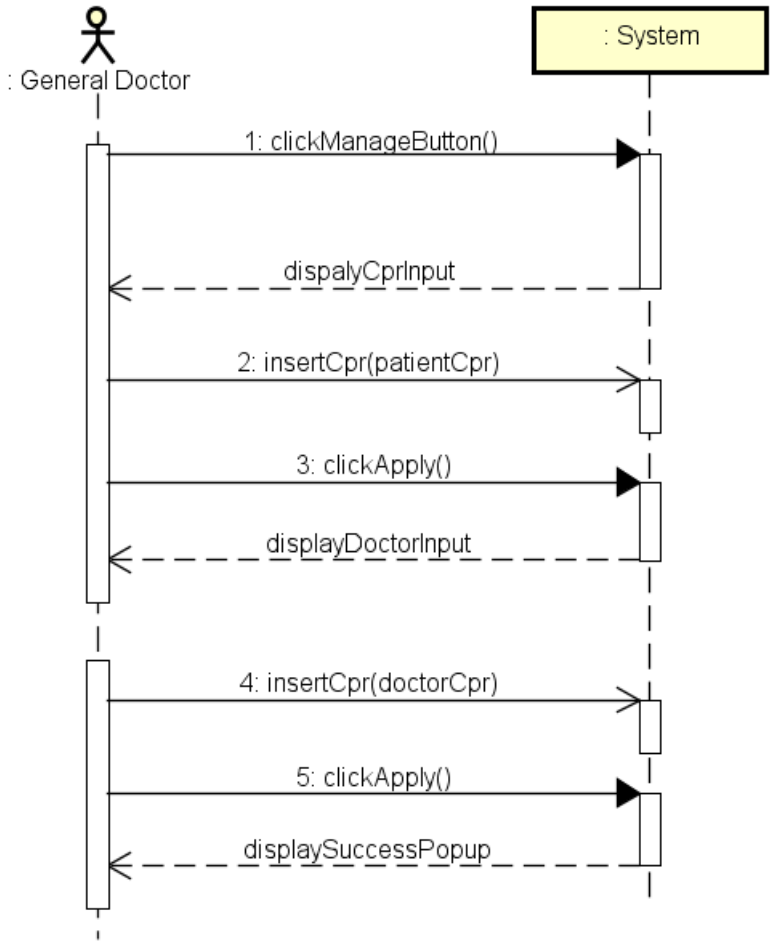


Figure 6 - System Sequence Diagram for assigning patients to specific doctor (Use Case nr 8)

3.3 Analysis Diagram

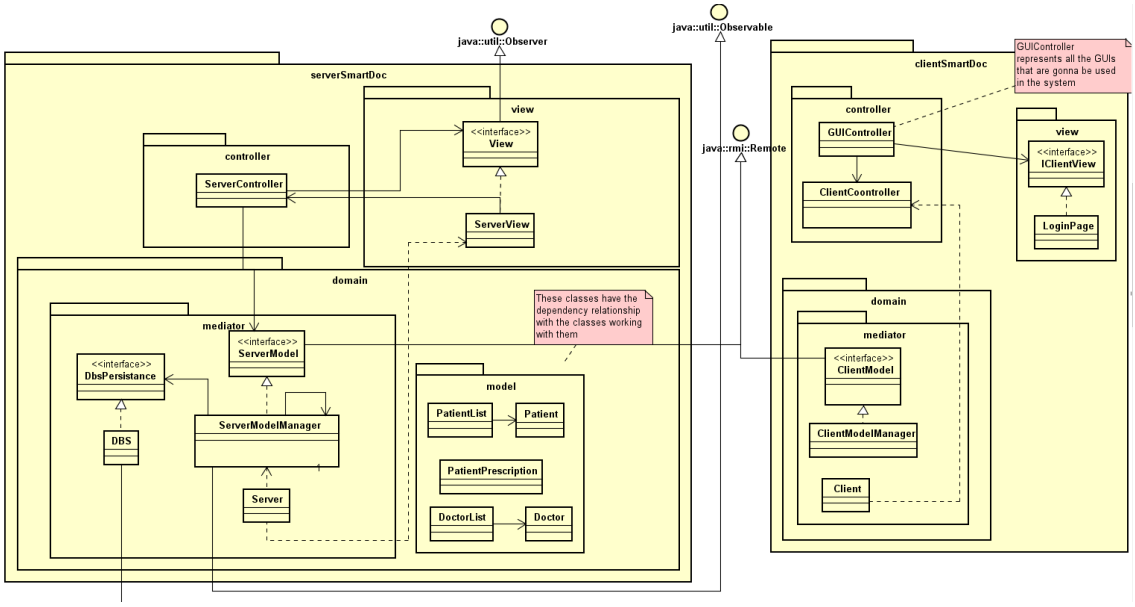


Figure 7 Analysis Diagram for the system

In the above figure it is presented the analysis diagram which makes the foundation for the system also the primary structure for Design and Implementation. From the picture it can be observed the client-server architecture also the Model-View-Controller pattern.

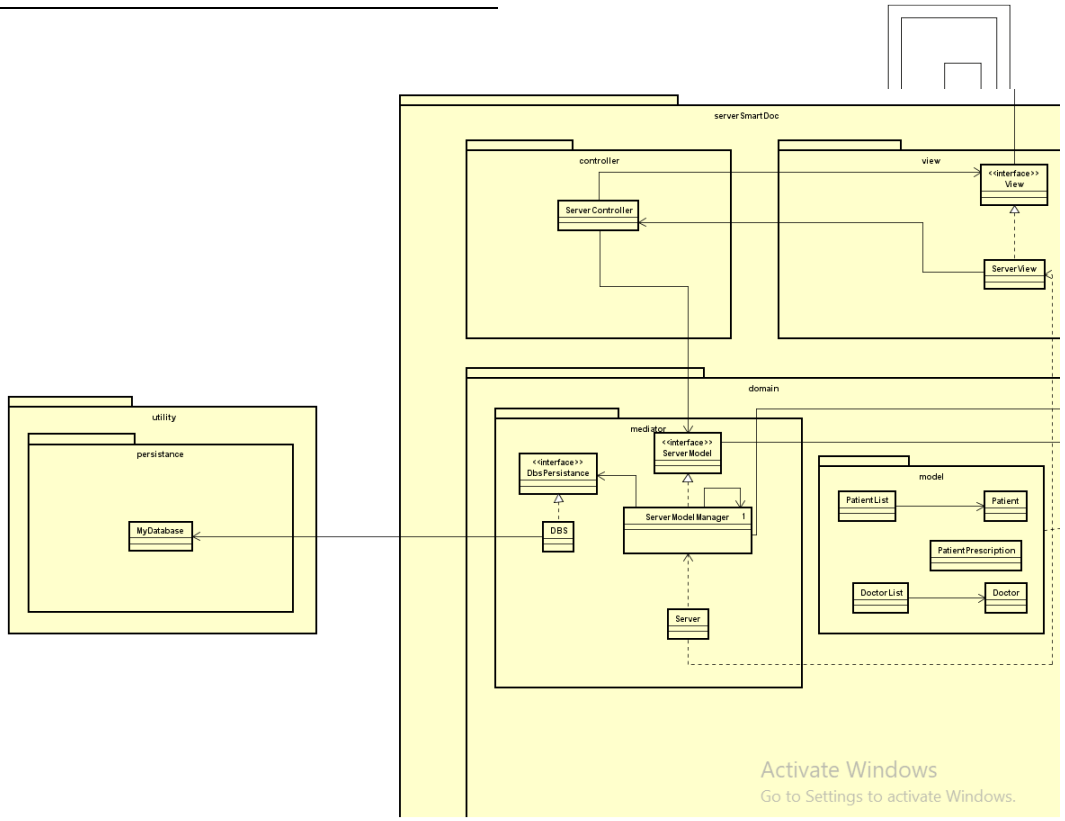


Figure 8 Server-side of analysis diagram

In the above picture it is shown the server part of the system also the separation of concerns through MVC having the model, the controller that performs the actions, the view showing the model state and the mediator which makes the connection between the controller and model, moreover the relationship between the database and the server.

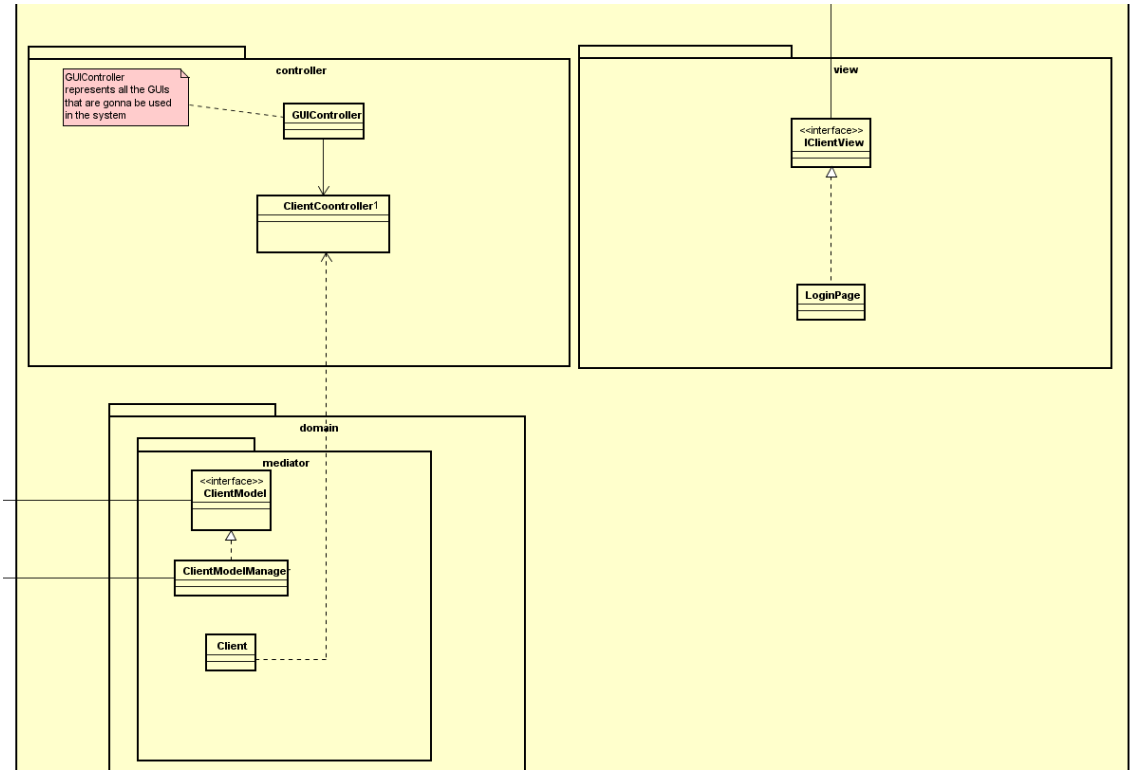


Figure 9 Client-side of Analysis Diagram

In the client side are used similar principles as in server, the exception being the absence of the model because it is using the model from the server.

4. Design

The design part of the paper is to show the system design choices, patterns used, technologies and other features that were used to build the system.

4.1 MVC Pattern

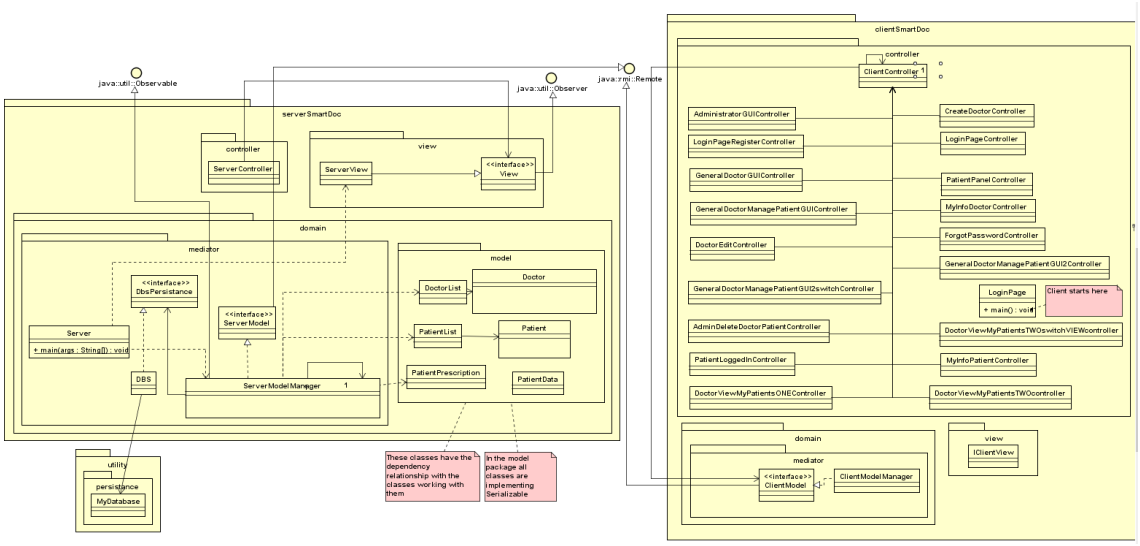


Diagram 2 Class diagram of the system(can be viewed in Appendix C, including all methods etc.)

The system has been built upon MVC (Model-View-Controller) pattern, which representation can be seen in the above class diagram(Diagram 1). As for the delimitation, that the system has to be implemented as a client-server-system, MVC has defined separately both the client and the server.

We can see multiple packages on the diagram, that contain various classes. The client side contains:

- Package ***controller***, which contains controllers for the GUIs and the ClientController class responsible for operations happening between views and model
- Package ***domain***, which in itself contains the package ***mediator***, that's responsible for connecting the client side of the system with the server side and sanding data to related objects

The server side contains:

- Package ***view***, which is responsible for displaying data in a console when some actions were taken by the system user

- Package ***controller***, which may be helpful for future development of the server-side of the system
- Package ***domain***, which contains two packages ***model***, which contains all the model-like classes used in the system, and a ***mediator*** which is responsible for receiving connection from the client, and sending and retrieving data from it

4.2 Observer Pattern

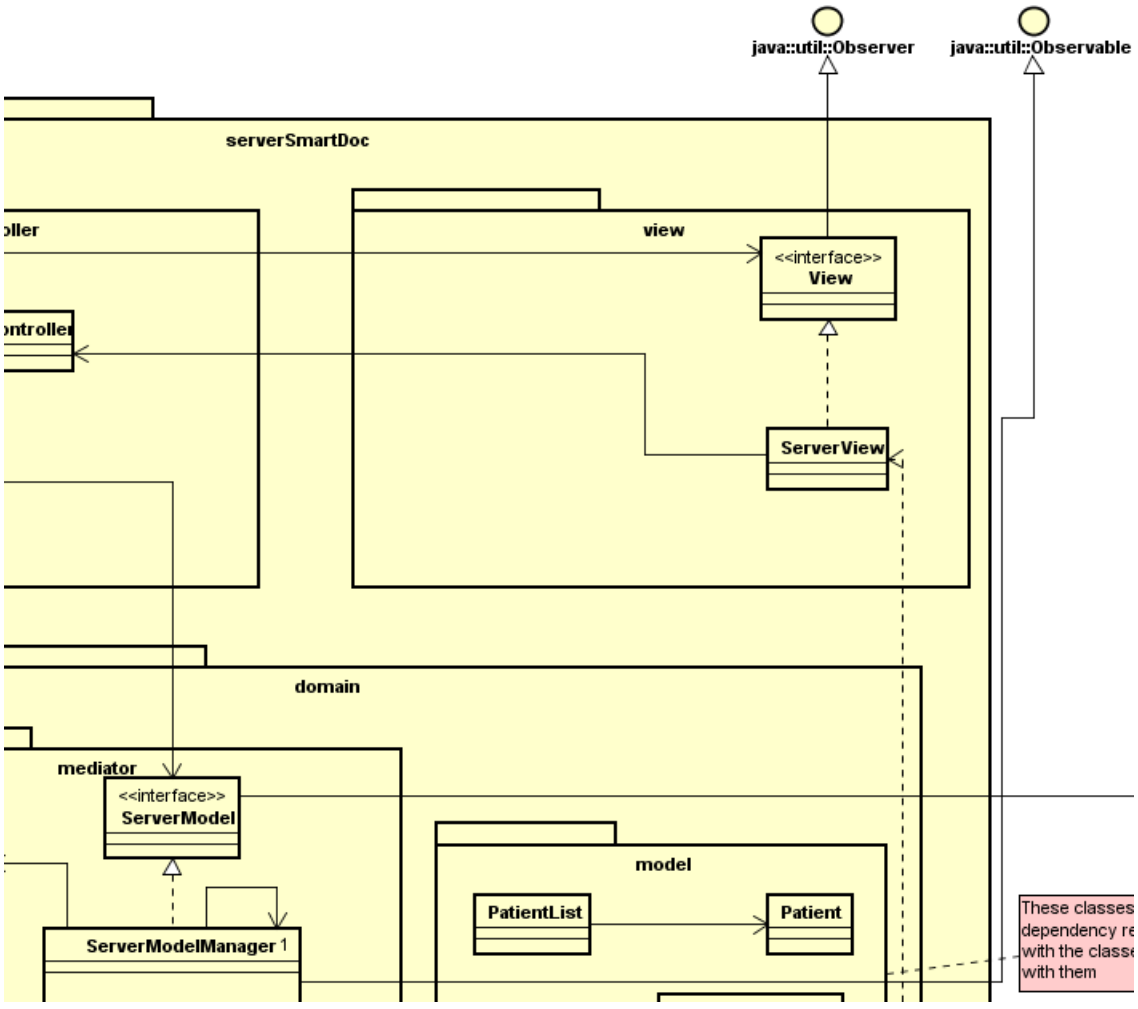


Diagram 3 Observer Pattern between ServerModelManager (Subject) and ServerView (Observer)

The observer pattern is used in the system between the ServerModelManager class and the ServerView, which allows the view to handle information console printing by the ServerView (Observer) after specific methods in ServerModelManager (Subject) are called. The inheritance of classes can be seen on the above Diagram 2

4.3 Adapter Pattern

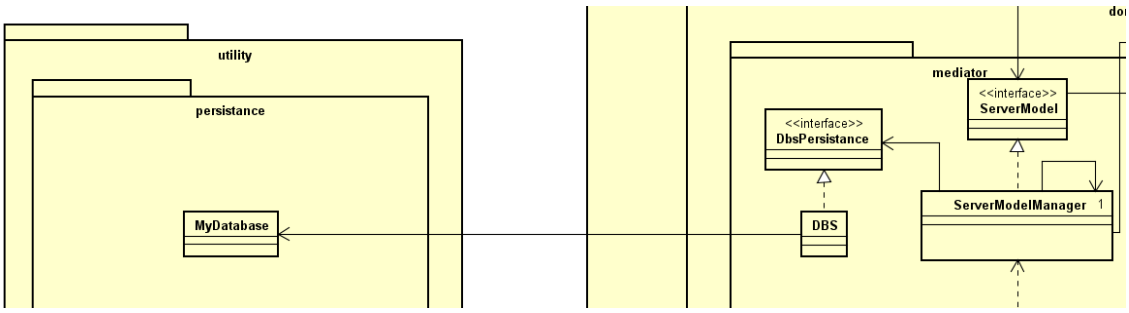


Diagram 4 Adapter pattern DbsPersitance + DBS

The adapter pattern is used in the system between the ServerModelManager class and MyDatabase class, which is the **DBS** and DbsPersistance classes. They allow the java specific data to be stored in database, or retrieved from it by querying and parsing. The pattern is shown on the above Diagram 3.

4.4 Singleton Pattern

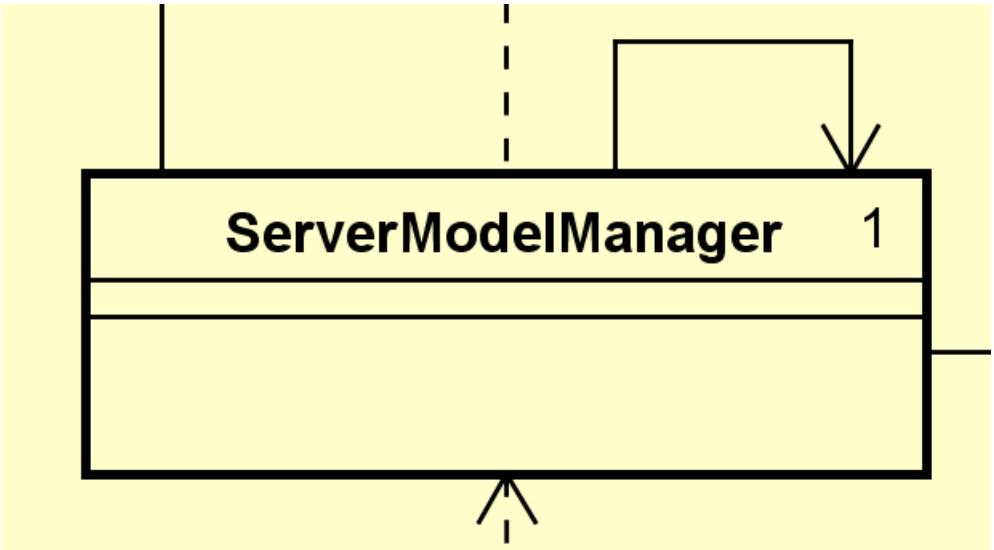


Diagram 5 Example of Singleton in the system

The singleton pattern is used in the system for two classes, the “ServerModelManager” and “ClientController” to ensure that only one instance exists restricting the instantiation of the class to one object furthermore simplifying the relationship between classes by eliminating the necessity to create the same object throughout the system.

4.5 Sequence Diagrams

The Sequence Diagrams were made to help visualize how the system should behave while different actions are taken by the users. They rely fully on the Use Cases and their scenarios. On the diagram below there is presented an example of a Sequence Diagram that refers to the Use Case nr 8. All of the Sequence Diagrams can be viewed in Appendix B.

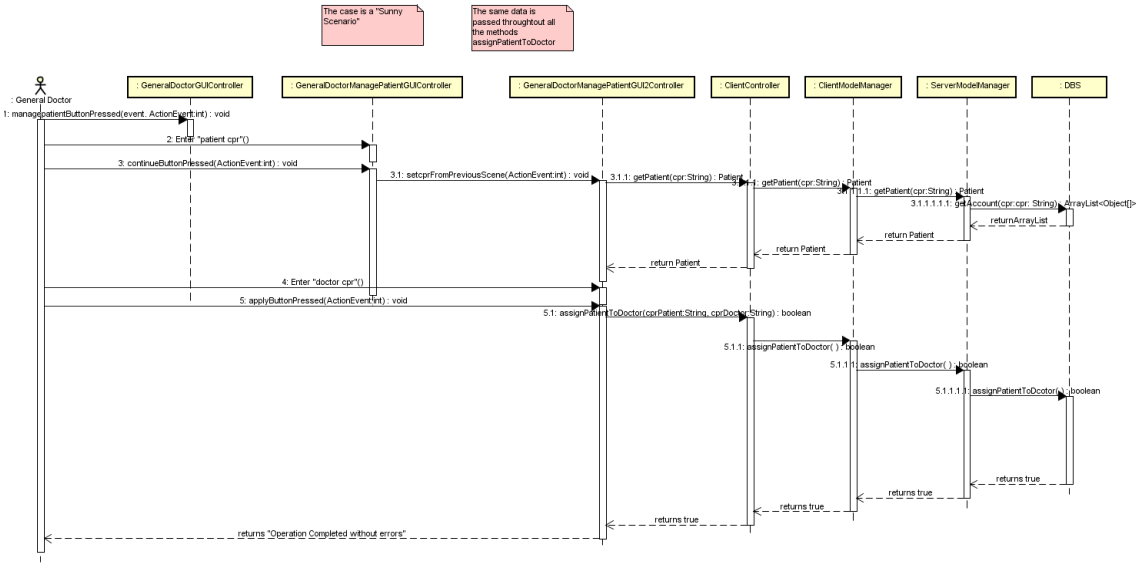
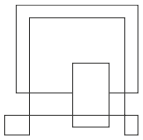


Diagram 6 Sequence Diagram for Use Case nr 8

The first method that is called is the “ManagePatientButtonPressed()” from the ‘’GeneralDoctorGUIController”, then the General Doctor needs to introduce the cpr of the patient and press the “Continue” button. The system then retrieves the data (following the path of the objects) of the Patient from the database of specific CPR in order to display it later in the GUI. Afterwards the specific doctor is chosen by inserting his cpr and pressing Apply calling the method “applyButtonPressed()”. The system then sends both the cpr od patient and doctor to database and writes it in the relation. System responds by returning the “Operation completed without errors” pop-up.



4.6 ER Diagram & Data normalization

Clinique Normalization

Zero normal form (0NF):

login (UNIQUE), password, fname, lname, cpr, phone, dob, email, speciality, gender, type, illnesses, allergies, height, weight, smoking, vaccines, familyIllnesses, insurance, pregnancy, prescription, appointments, problem, recommendations

First normal form (1NF):

login, password, fname, lname, cpr, phone, dob, email, type, speciality, gender

cpr, illnesses, allergies, height, weight, smoking, vaccines, familyIllnesses, insurance, pregnancy, prescription, appointments, problem,, recommendations

Second normal form (2NF):

login, password, fname, lname, cpr, phone, dob, email, type, speciality, gender

cpr, illnesses, allergies, height, weight, smoking, vaccines, familyIllnesses, insurance, pregnancy, prescription, appointments, problem, recommendations

Third normal form (3NF):

cpr, login, password,

cpr, fname, lname, phone, dob, email, type, gender

cpr, speciality

cpr, illnesses, allergies, height, weight, smoking, vaccines, familyIllnesses, insurance, pregnancy,

cpr, prescription, appointments, problem, recommendations

Text Figure 1 Database normalization

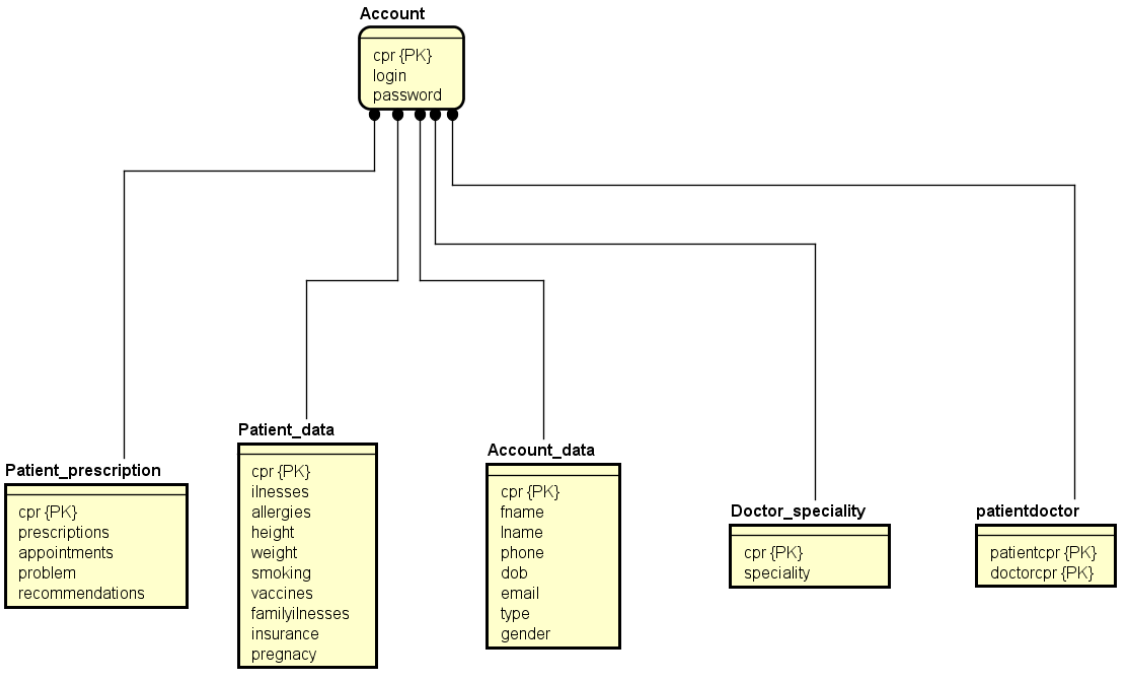


Diagram 7 ER Diagram for Database Design

Database normalization(Text Figure 1) and the ER Diagram(Diagram 2) show the database design and how the specific data are related to each other. The aim of the normalization is to ensure the database will work well with the system requirements and their objectives. The **Primary Key** in most of the relations is the CPR number of a person registered in the system, which is always referenced to the *Account* relation. This provides the system clarity on what data should different entities rely on. Specifically, only *patientdoctor* relation has the primary kay which contains of cpr number or patient and doctor. That ensures that when a patient is assigned to a doctor, this can only take place once between particular patient and doctor.

4.7 Technologies

The decisions about technologies are listed below and were made with a respect of the latest trends, but also complexity of the system.

Technologies that were decided to be used are:

- **GitHub** – which provided a good way to exchange and merge code done by the group members
- **Java –** according to the projects requirements – it had to be built using Java language
- **PostgreSQL** – as a database management system
- **JavaFx** – the library that the GUI of the system is using
- **Scene Builder** – software for JavaFx to simplify the GUI building process
- **Eclipse IDE** – for Java code writing
- **DataGrip/pgAdmin 4** – for PostgreSQL database implementation

4.8 UI Design

As stated in the Technologies choices, the GUI for the system is made using JavaFx, which ensured the layout will look fresh and modern. This includes the Scene Builder for the simplicity of implementation. The example of the UI can be seen below.

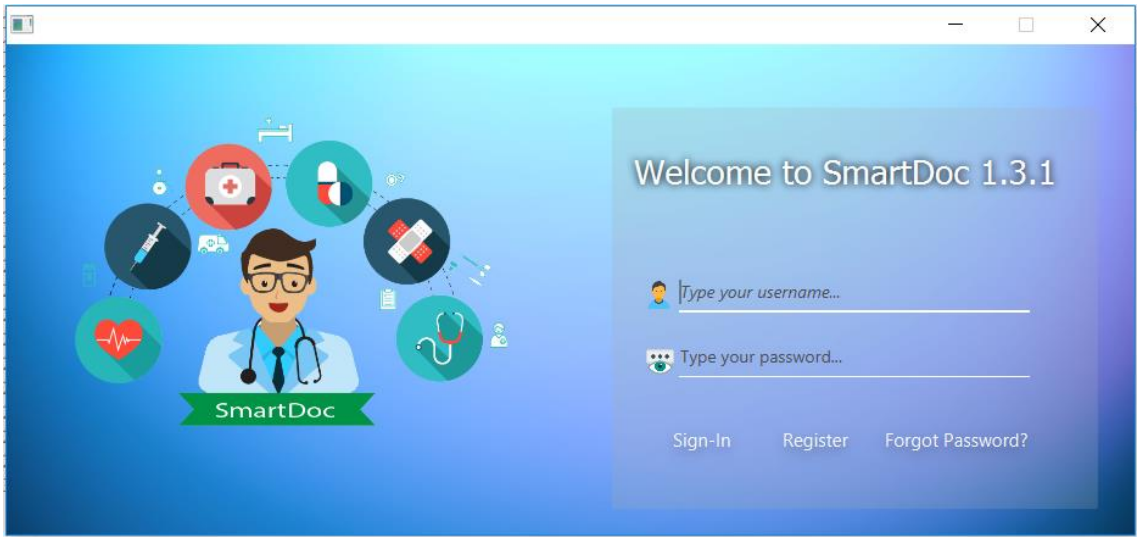
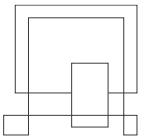


Figure 10 Login window for the system



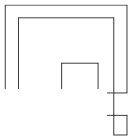
5. Implementation

The implementation of the system has been done using upper mentioned technologies and solutions to make the process simpler and quicker. Some of the most interesting parts of the system are contained in the GUI parts, but also in the DBS class, which will be showed and discussed in the following code snippets.

```
@Override
public boolean createPatient(String login, String password, String fname, String lname, String cpr, int phone,
        String email, LocalDate dob, String gender) {

    String passwordHex = passwordToHex(password);
    String sql = "insert into account values (?,?,?)";
    String sql1 = "insert into account_data values(?,?,?,?,?,?,?,?)";
    String sql2 = "insert into patient_data values(?,?,?,?,?,?,?,?,?,?)";
    String sql3 = "insert into patient_prescription values(?,?,?,?,?)";

    Date dateSQL = parseDateToDbs(dob);
    String type = "P";

    try {
        myDatabase.update(sql, cpr, login, passwordHex);
        myDatabase.update(sql1, cpr, fname, lname, phone, dateSQL, email, type, gender);
        myDatabase.update(sql2, cpr, '0','0',0,0,false,'0','0',false,false);
        myDatabase.update(sql3, cpr, '0',parseDateToDbs(LocalDate.now()),'0','0');
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        return false;
    }
    return true;
}
```

Figure 11 Code Snippet from DBS class, createPatient method

The above snippets (Figure 11) shows the example of passing data to database in case of registering a new patient. The data are passed to the method, and then assigned a place in one of the SQL strings that insert data to specific relations and columns in the database. The password set by the patient is first encrypted to hexadecimal string with method *passwordToHex()*, which uses the SHA-256 hash function. This process also takes place with registration of all different parties using the system to ensure their data safety. You can also see the internal class method *parseDateToDbs()* is called in the code, which parses the *LocalDate* object to *Date* – in order to be able to be read by PostgreSQL as the *Date* variable. Code for both of these methods can be seen below (Figure 12, Figure 13).



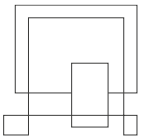
```
public String passwordToHex(String password) {
    MessageDigest md = null;
    try {
        md = MessageDigest.getInstance("SHA-256");
    } catch (NoSuchAlgorithmException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    byte[] buffer = password.getBytes();
    md.update(buffer);
    byte[] digest = md.digest();

    StringBuffer sb = new StringBuffer();
    for (byte b : digest) {
        sb.append(Integer.toHexString((int) (b & 0xff)));
    }
    String passwordHex = sb.toString();
    return passwordHex.toUpperCase();
}
```

Figure 12 passwordToHex method in DBS class

```
public Date parseDateToDbs(LocalDate date) {
    String[] dateVariables = date.toString().split("-");
    String trimYear = dateVariables[0];
    String trimMonth = dateVariables[1];
    String trimDay = dateVariables[2];

    System.out.println(trimYear + " " + trimMonth + " " + trimDay);

    int year = Integer.parseInt(trimYear) - 1900;
    int month = Integer.parseInt(trimMonth);

    month--;

    int day = Integer.parseInt(trimDay);

    Date dateSQL = new Date(year, month, day);
    return dateSQL;
}
```

Figure 13 parseDateToDbs method code snippet

The database has been coded in SQL, that could be shared within the group members to ensure the same version of database is shared. The following figure shows how that was made.



```
create table account (cpr varchar(11) primary key, login VARCHAR(50), password
VARCHAR(150));

create table account_data (cpr varchar(11) references account(cpr) primary key,
fname VARCHAR(50),
lname VARCHAR(50),
phone integer,
dob date,
email VARCHAR(50),
type varchar(1),
gender VARCHAR(1));

create table doctor_speciality (cpr varchar(11) references account(cpr) primary key,
speciality VARCHAR(50)
);


create table patient_data (cpr varchar(11) references account(cpr) primary key,
illnesses text, allergies text, height integer, weight integer,
  smoking boolean, vaccines text, familyIllnesses text, insurance boolean,
  pregnancy BOOLEAN
);

create table patient_prescription(cpr varchar(11) references account(cpr) primary key,
prescription text, appointments date, problem VARCHAR(200), recommendations text);


create table patientDoctor (patientCpr varchar(11) references account(cpr),
            doctorCpr varchar(11) references account(cpr), primary key(patientCpr,doctorCpr));

alter table account add constraint account_login_key UNIQUE (login);
```

Figure 14 SQL code sample for database implementation

The above snippets show how the relations for database were created. Data types for the columns were created strictly following the requirements for the system to ensure they contain correct data and correct ranges for them. You can see that after creating all the tables, one of the tables is also changed with “alter table account”, this statement adds the constraint of uniqueness for the usernames(login) in the account relation, that ensures only one unique username exists in the system.

6. Test

The first part of tests were made as unit tests, which were making sure the core parts of the system classes work well. This was done using Junit library for Java. You can see an example of the test on the following snippet that tests the DBS class.

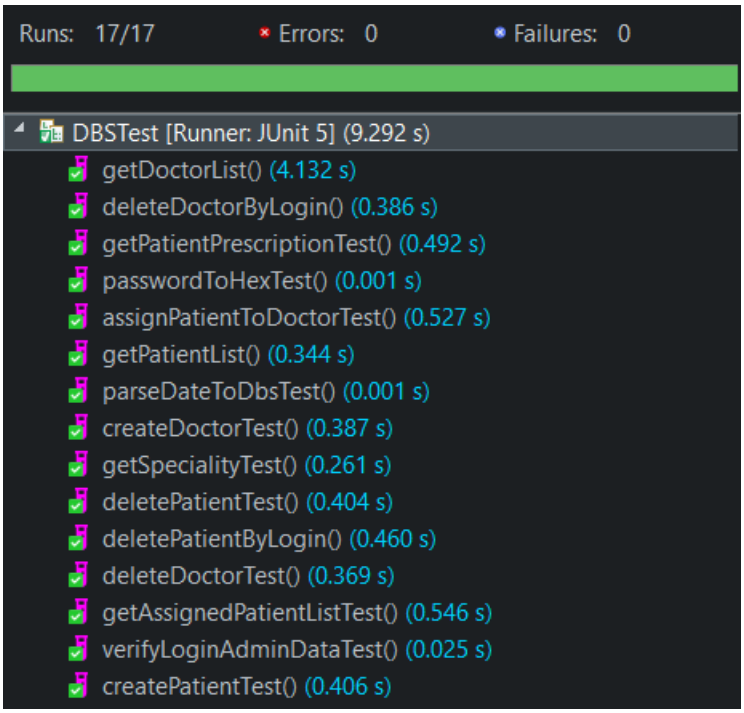
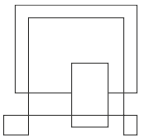


Figure 15 Test Snippet

The second part of tests were based on the Use Cases and their scenarios, each of it having each path user can take shown, followed and tested to make sure all the Use Cases can correctly refer to what is happening in the system so these parts could always be relied on.

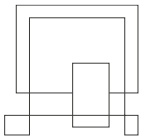
On the table below you can see the how one of the Test Cases looks like. It refers to a specific Use Case (Use Case 8 - see appendix A). The Test Case is given an id, as well as every scenario it shows is. The "V" means that something is correct, or true – in contrast "X" means that something is false. The *Expected result* column says what should happen when a given scenario, with given data is fired, as well as the implementation column shows if a certain scenario for a use case has already been implemented.



TC ID#5	Scenario	Patient CPR	Doctor CPR	Expected result	Implemented
AS1.	Successfully assigned patient to specific doctor	V	V	Patient is assigned to the specific doctor, system displays "success"	V
AS2.	Wrong Patient CPR	X	V	No response from the system	V
AS3	Wrong Doctor CPR	V	X	System prompts “Wrong CPR”	V

Figure 16 Test Case for Use Case nr 8

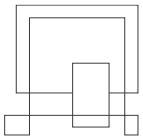
Testing included all the most vital parts of the system and the Test Cases as well as table of “Use Case – Test Case” relations can be viewed in the Appendix D.



7. Results and Discussion

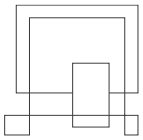
The system keeps track of data of various parties that are set to be using the system - like patients and doctors. It is also ensuring the security of the data of these parties by a logging system and encrypting the password. Moreover, ensures patient's treatment data are kept private by a complex data visibility concerns. Patients are able to register in the system, after which they are able to access their data – and if they are assigned to a specific doctor, the doctor is able to provide them with updated data about their treatment giving prescription, diagnosis, recommendations and updating the upcoming meeting.

The system however, has some minor mistakes in the implementation part. The model classes that are used in the GUI (on the Client-side view) could have been implemented in the Client-side of the system and not externally imported from the Server-side. This would ensure more integrity in the system. Also the multi-functionality of the server-side classes violates SOLID principles, which could be solved by creating more specific and defined classes.



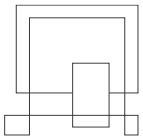
8. Conclusions

The system development has been following all of the above stages from Analysis, through Design, Implementation until Testing. With the Analysis part, the requirements as well as the delimitations were deeply analyzed to make sure all future features will rely on them. In the Design part, the diagrams from Analysis part were brought and followed to design a system for Java, in this part also all the various technologies and patterns were decided. The system then could be successfully implemented in an Implementation part. Lastly, the Use Cases were tested in the Test Cases, that included unit testing. The main purpose from Project Description that stated to "*facilitate the interaction between doctors and patients thus offering efficient and accurate administration of health information*" has been fulfilled with satisfying results.



9. Project future

If the project would be further developed there would be a lot of features that could be improved in the future. One of the most obvious parts is definitely the complexity of the data that could be shared between doctors and their patients. For instance the patient could see not only the prescription, but also what is the price of a specific medicament. He could also be able to see his treatment history, the previous appointments, more precise data about his treatment etc. The system could also be further developed as a website, that would broaden the target consumer scope and eventually become a really large and complex hospital system that could be widely deployed in many places.



10. Sources of information

Anon., 2014. *Top Mobile Trend.* [Online]
Available at:
https://web.archive.org/web/20140530024928/http://topmobiletrends.com/mobile-technology-contributions-patient-experience-parmar/

Anon., 2017. *Cerner.* [Online]
Available at: https://www.cerner.com/blog/8-health-it-trends-to-watch-in-2018

Anon., 2017. *Healthcare Informatics.* [Online]
Available at: http://www.amritatech.com/healthcareInformatics.html

Anon., n.d. [Online].

Cerner's Senior Vice President of Population Health John Glaser , 2017. *Cerner.* [Online]
Available at: https://www.cerner.com/blog/8-health-it-trends-to-watch-in-2018

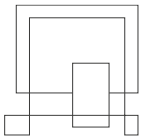
Healthcare Administration, 2012. *HealthCare Administration.* [Online]
Available at: http://www.healthcareadministration.com/healthcare-management-historical-background/
[Accessed February 2018].

Larman, C., 2005. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition).* 3rd ed. s.l.:s.n.

Michal Ciebien, R. P. M. M. T. E. C. B. S., 2018. *Project Description SmartDoc,* s.l.: s.n.

Owens-Liston, P., 2012. *Health Feed.* [Online]
Available at: https://healthcare.utah.edu/healthfeed/postings/2012/11/113012homeobit.php
[Accessed March 2018].

VENTER, D., 2018. *The evolution of Hospital Information Systems.* [Online]
Available at: https://orionhealth.com/uk/knowledge-hub/blogs/the-evolution-of-hospital-information-systems/
[Accessed May 2018].



11. Appendices

1. Project Description
2. User Guide
3. Source code + documentation
4. Appendix A (Analysis)
5. Appendix B (Design)
6. Appendix C (Analysis Diagram + Design Class Diagram)
7. Appendix D (Test Cases)
8. Group Contract