SEPI2-Y-A16 – BLOOD CENTER

PROJECT REPORT

Group members:	*Andreea Lacramioara Carst*	*240286*
	Carlos Cristiano Alberto	*247903*
	Muhammad Allak	*245496*
	Raluca Balogh	*245490*
Supervisors:	*Henrik Kronborg Pedersen*	
	Troels Mortensen	

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

ABSTRACT

Giving blood saves lives. The blood people give is a lifeline in an emergency and for people who need long-term treatments. Many people would not be alive today if donors had not generously given their blood. One unit of blood can be separated into several components: red blood cells, plasma, platelets and cryoprecipitate. It mean's 1 blood bag saves 3 lives.

Only about 3% of eligible donors donate blood [1]. Why is this? Human blood cannot be duplicated and most people know at least one person that has had or will need a blood transfusion during the course of their life. Even so people do not realise the need for donated blood.

Philosophers such as Jean Jacques Rousseau argued that people were born good, instinctively concerned with the welfare of others. Starting from this idea we went further thinking that people need to fill that they are doing something positive for helping others. If a blood donor will receive a message when his blood saved a life, this will encourage him to donate again. He will fill that his gesture is good and important.

We have developed a system allows a constant communication between hospitals and blood centres. Every time a bag with blood is used the hospital sends the blood centre a message. They will now send an e-mail or text message to the blood donor.
A similar system was already successfully implemented in Sweden and England. The feedback received from the donors is highly positive, saying that it has a nice touch [2].

INTRODUCTION

There are chronic shortages in the supply of blood available for transfusions. Hopefully, the need for donated blood will someday be eliminated by the development of artificial blood. In the meantime, however, the Red Cross is working on getting more people to donate blood. [3] In order to come in their help, we have developed a program that encourage the blood donation by allowing the blood centre members to receive information about when blood was used. They can now react and send a message to the blood donors. Finding out that the gesture of donating had a "touchable meaning" (someone is living in this moment thanks to donated blood) will encourage the donor to donate again.

Our program allows a constant communication between the hospitals and the blood centres. The system is split into two different access points. First access point is used by the staff of the blood collecting point, a blood bank. The purpose is to store data regarding the donor, such as name, address, blood type, age, telephone number and email. Each staff member will have his or her own user name and password, which will protect their access from other unauthorized entries. The data from the donors will be stored in a shared database and will be available to the both sides using this system.

The second access point is used by the hospital staff. Each member of the hospital will be able to see the basic data of the patient, such as the ID, name and blood type. The hospital staff will also have a username and password, that will grant them access to the system and will prevent unauthorized access.

ANALYSIS

Doctors need blood to save lives. Hospitals need to communicate with blood centres to request necessary blood. Donors need to know the finality of the blood donation they made. Our system was implemented to come in help for this entire process.

Requirements

Functional

1. The Hospital (the first client) should send a request to the server for an amount of blood type whenever the hospital needs it.
2. In order to use the system, the user must log in.
3. In order to exit the system, the user must log off.
4. The hospital should send a notification to the server whenever a blood transfusion happens.
5. The blood centre should send an amount of a specific blood type to the hospital whenever it received a request.

Non-Functional

1. In order to send a request, the request button must be selected.
2. In order to log in to the system, the user must insert his/her id as a username and his/her password.
3. The user can log off the system by selecting either a log off button or an exit button.
4. A Notification process can happen, only by selecting a withdraw blood button.
5. In order to send the blood from the blood centre to the hospital, the blood centre's user must select the requested blood type from the database and specify the amount and then select a send blood button.
6. Whenever the send blood button is selected the blood id will be removed from the blood centre's database and added to the hospital's database.

Use Case Modelling

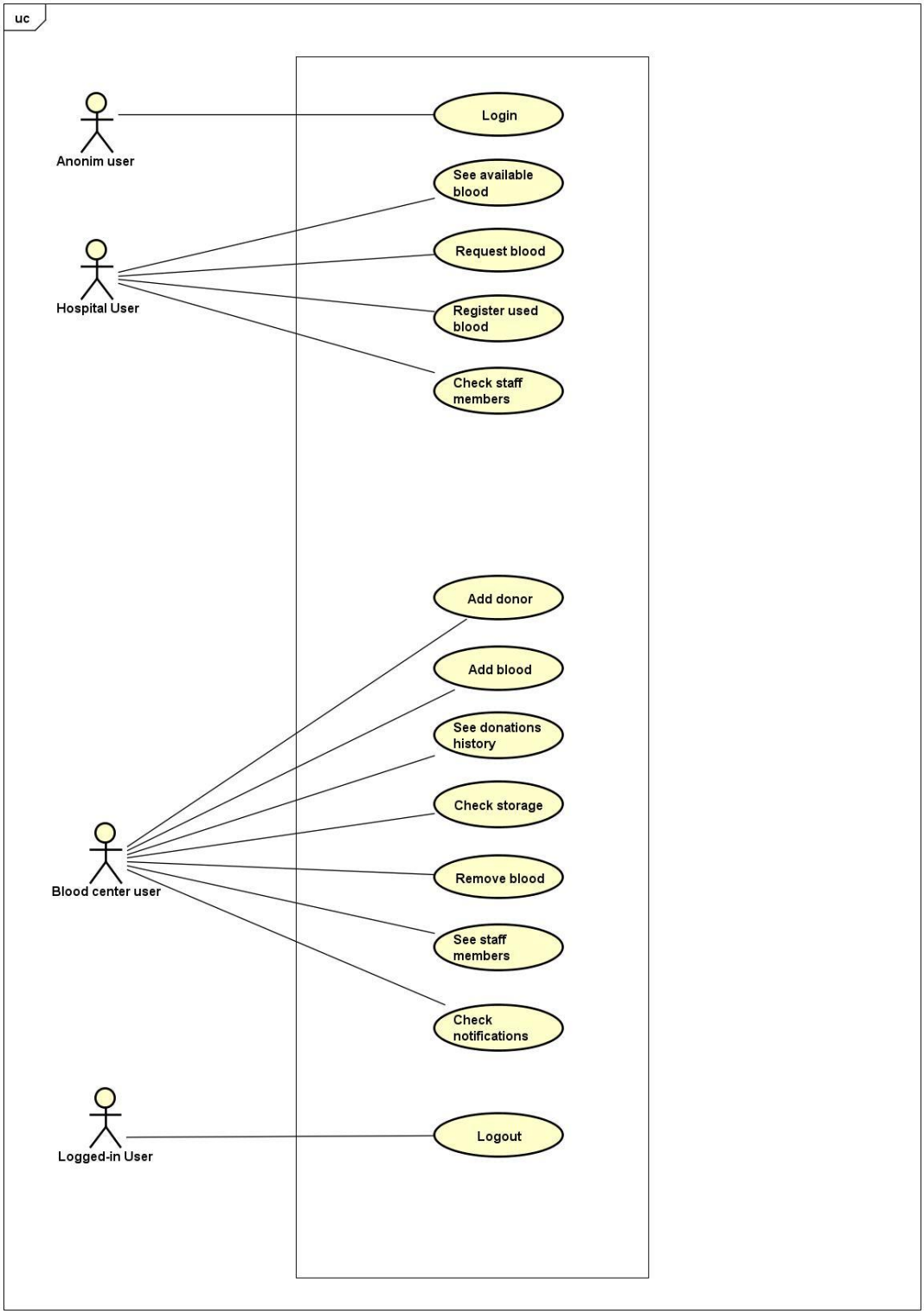

Figure 1 - Use case diagram

Use Case Diagram

The use case above shows all the requirements that have been implemented for the system. First, the system will require the user to login by entering the username and the password. An anonym user will login to the system. If the user is part of the Blood centre staff he will be able to register a new donor, a new blood donation, see the history of all donation, check the blood storage, remove blood, see a list of staff, check the notifications received from the hospitals. If the user is part of the hospital staff he will be able to see the available blood, request blood to the blood bank, register used blood and check staff members. All the users, after finishing their activity will logout for security reasons.

Use Case Description

LOGIN:

Use case: Login

Summary: The user has to login in order to have access to the system

Actor: Anonym User

Precondition: The user has to be registered in the system with a username and a password.

Post condition: The user needs to type correctly his username and the password

Base sequence: The user needs to insert his username and password to receive access to the systems functionality.

Exception sequence: If username and password does not match the user needs to type it again.

ADD DONOR:

Use case: Add donor

Summary: The user adds a new donor

Actor: Blood centre User

Precondition: The user is logged in.

Post condition: All the fields are completed

Base sequence: The user registers a new blood donor. He fills in the fields with information about the donor.

Exception sequence: If the user didn't fill all the fields he will get a notification: "Please fill in all the required information".

ADD BLOOD: *Use case:* Add blood

Summary: The user can register a new blood donation

Actor: Blood centre User

Precondition: The user is logged in.

Post condition: All the fields are completed

Base sequence: The user registers a new blood donation. He fills in the fields with information about the blood.

Exception sequence: If the user didn't fill all the fields he will get a notification: "Please fill in all the required information".

SEE DONATIONS HISTORY: *Use case:* See donations history

Summary: The user can see a history off all the donations

Actor: Blood centre User

Precondition: The user is logged in.

Post condition: None

Base sequence: The user can see a history of all the donors.

Exception sequence: None

CHECK STORAGE: *Use case:* Check storage

Summary: The user checks the blood available in the storage.

Actor: Blood centre user

Precondition: The user is logged in.

Post condition: None

Base sequence: The user checks the available blood in the storage by blood types.

Exception sequence: None

REMOVE BLOOD:

Use case: Remove blood

Summary: The user removes the blood that has expired or has been sent to the hospitals.

Actor: Blood centre User

Precondition: The user is logged in.

Post condition: None

Base sequence: The user removes blood. The blood was expired or the blood have been sent to hospitals.

Exception sequence: None

SEE STAFF MEMBERS:

Use case: See staff members

Summary: The user can see the list with the staff from the blood centre.

Actor: Blood centre user

Precondition: The user is logged in.

Post condition: None

Base sequence: The user sees a list of the staff working for the blood bank.

Exception sequence: None

CHECK NOTIFICATIONS:

Use case: Check notifications

Summary: The user can see the notifications received from the hospital.

Actor: Blood centre user

Precondition: The user is logged in.

Post condition: None

Base sequence: The user can see the notifications received from the hospital regarding the necessary blood and the used blood.

Exception sequence: None

LOGOUT:

Use case: Logout

Summary: The user has to logout from the system

Actor: Logged-in user

Precondition: The user is logged in.

Post condition: The user will not have access to the system after logging out.

Base sequence: For security reasons the user needs to logout from the system after finishing his activity.

Exception sequence: None

REQUEST BLOOD:

Use case: Request blood

Summary: The user sends a request to the blood centre

Actor: User in Hospital

Precondition: The user is logged in.

Post condition: None

Base sequence: The user checks the available blood in the storage and sends a request to the blood centre for more blood of a certain blood type.

Exception sequence: None

SEE AVAILABLE BLOOD:

Use case: See available blood

Summary: The user sees the available blood

Actor: User in Hospital

Precondition: The user is logged in.

Post condition: None

Base sequence: The user sees all the available blood in the hospitals storage. He can search by blood type.

Exception sequence: None

REGISTER USED BLOOD:

Use case: Register used blood

Summary: The user uses a bag of blood from the hospitals storage.

Actor: User in Hospital

Precondition: The user is logged in.

Post condition: None

Base sequence: The user register that he used a bag of blood. The blood centre will receive a message

Exception sequence: None

CHECK STAFF MEMBERS:

Use case: Check staff members

Summary: The user can see the list with the staff from the blood centre.

Actor: User in Hospital

Precondition: The user is logged in.

Post condition: None

Base sequence: The user sees a list of the staff working for the hospital.

Exception sequence: None

DESIGN

The program consists of three parts: the database the Java class and the GUI.

JAVA CLASSES:

The Java classes make the functioning of the program possible by creating and manipulating necessary Java objects, as well as connecting, retrieving and sending information to the database. Numerous Java classes have been implemented in order for the program to have a strong object-oriented character.

DATABASE:

The database contains all the information regarding the donors, the staff from the blood bank and hospital

GUI:

The Graphical User Interface is formed of one Java class which contains numerous other classes. Thus, all the repetitive methods needed for implementing the buttons, tags, text-field and other graphical user interface related classes are not mixed with other classes, which allows for a better maintenance.

Database Design

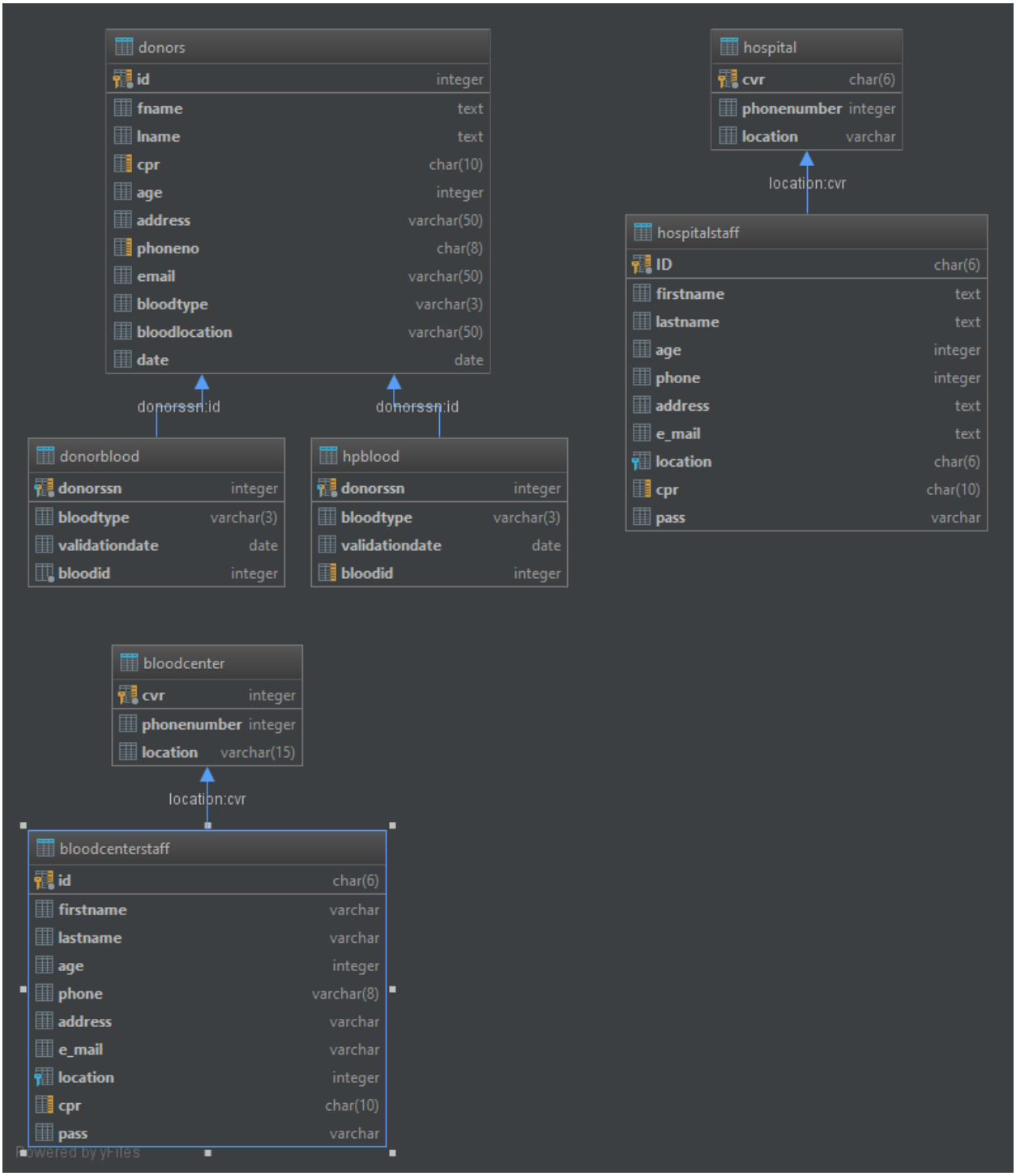

Figure 2 - Database diagram

In Order to store the donated blood in the system, database was needed. This means that the database should hold the information for both the donor and the blood. The donor's information was saved in the system to make it easier for the blood centre if the same donor decides to donate again and to send a message to the donor afterwards when his/her blood is used in the hospital to save someone else's life.

The database consists of seven tables. First of all, we implemented the 'donors' table which included all the contact details required for making a donation. Whenever the donor donate a blood bag he/she is going to receive an ID number which will be used to access to the donor's information in case the donor decides to donate again which means that there's no need to ask the donor for all the information every time.

The staff tables for the blood banks and hospitals were mainly requested mainly for security reasons and are being used for logging into the system.

The 'hospital' and 'bloodcenter' tables include the ID in order to identify the blood bank or the hospital, the location and the phone number. The blood centre's staff will be able to login to the blood centre system, add a blood, add donor, send an encouraging message to the donor after his/her blood was used and remove a blood ID from the database in case that blood was sent to the hospital. The hospital staff will be able to login to the hospital system, request a specific blood type in case the hospital has a lack of that blood type and notify the blood centre when a blood bag is used.

And lastly, the 'donorblood' and 'hpblood' tables were necessary for making a donation. The blood information database holds an ID number for each blood bag to distinguish between the donated blood bags that have a similar type or donor. If a user donates blood, the database will generate an ID and save it in the 'donorblood' table. If the hospital will make any blood requests, the donor blood ID, donor ID, validation date and blood type will be moved in the 'hpblood' table.

The database is designed to be easily implemented in java but still be a fully functional database form PostgreSQL point of view.

GUI Design

Blood Bank GUI Design

The main feature of graphical user interface for the blood bank is to make a donation. Besides this, after logging in, the user can access a menu, see the donors' history, the blood storage, the staff of the blood bank and the notification received from the hospital.

All the buttons for the menu tab represent the functionalities of the system. There is a corresponding tab for each button. Whenever it is pressed, the GUI will display the right tab with different functionalities.

The functions represent the Use Case of the system and each function works according to Use Case Descriptions. The full interface can be found in Appendix A.

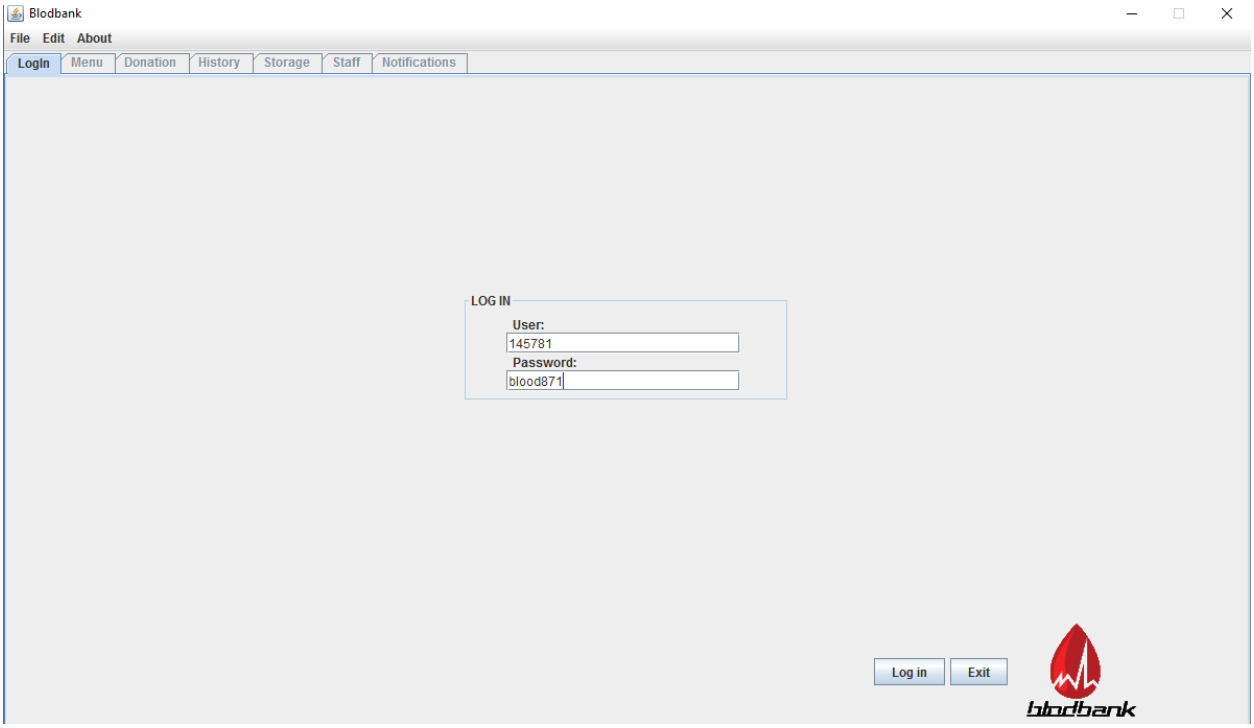

Figure 3 - Log in for the blood bank user

The program was organized in tabs, according to the activity the user wants to access. This way, the user gets maximum visibility of all the actions available at all times.

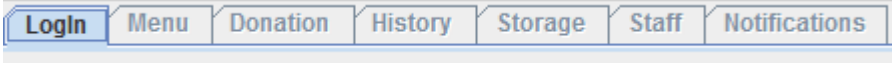

Figure 4 - Options for the blood bank user

Hospital GUI Design

From the beginning of the GUI implementation we have established that the hospital should not have the same features as the blood bank. The main focus was when a blood is being used, therefore no other unnecessary details concerning the donor can be accessed from the hospital, except for the donor id, blood id, blood type and the validation date.

Just as the graphical user interface for the blood bank, the GUI's hospital also has a log in tab, followed by the blood storage and the staff. Since there are not as many functionalities as the blood bank, there wasn't any necessity to implement a menu.

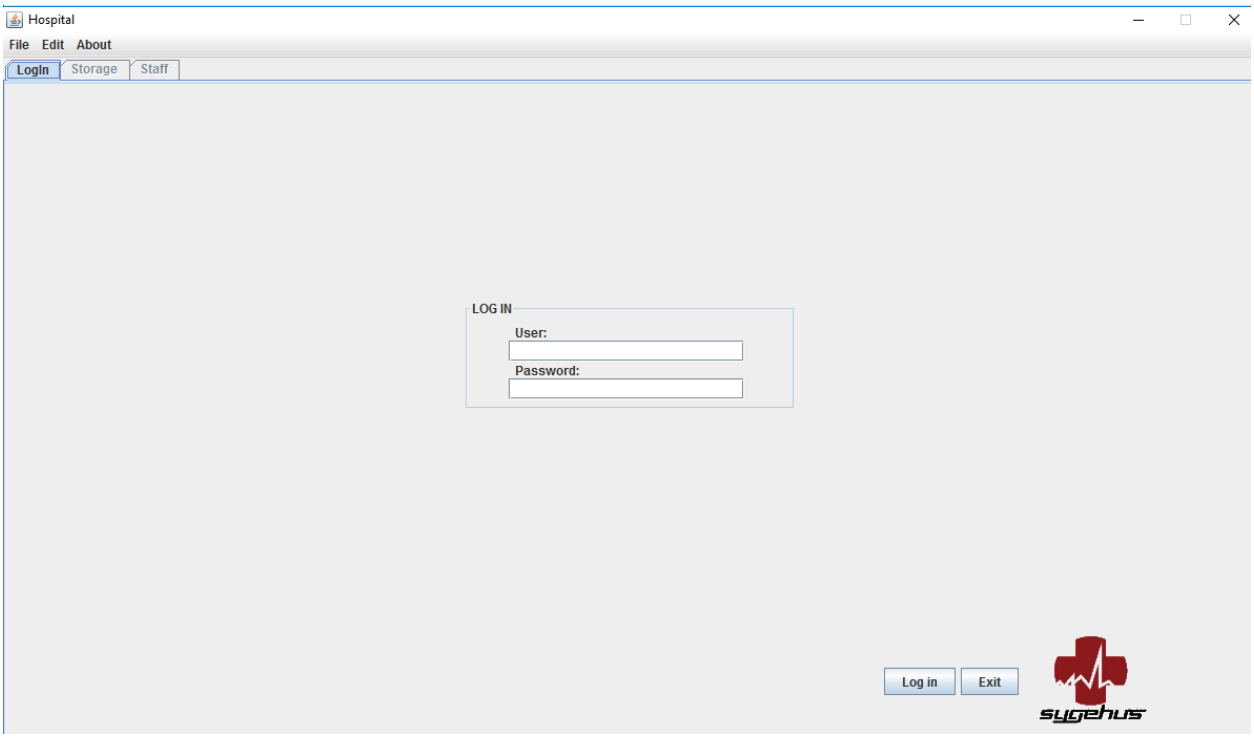

Figure 5 - Log in for the hospital user

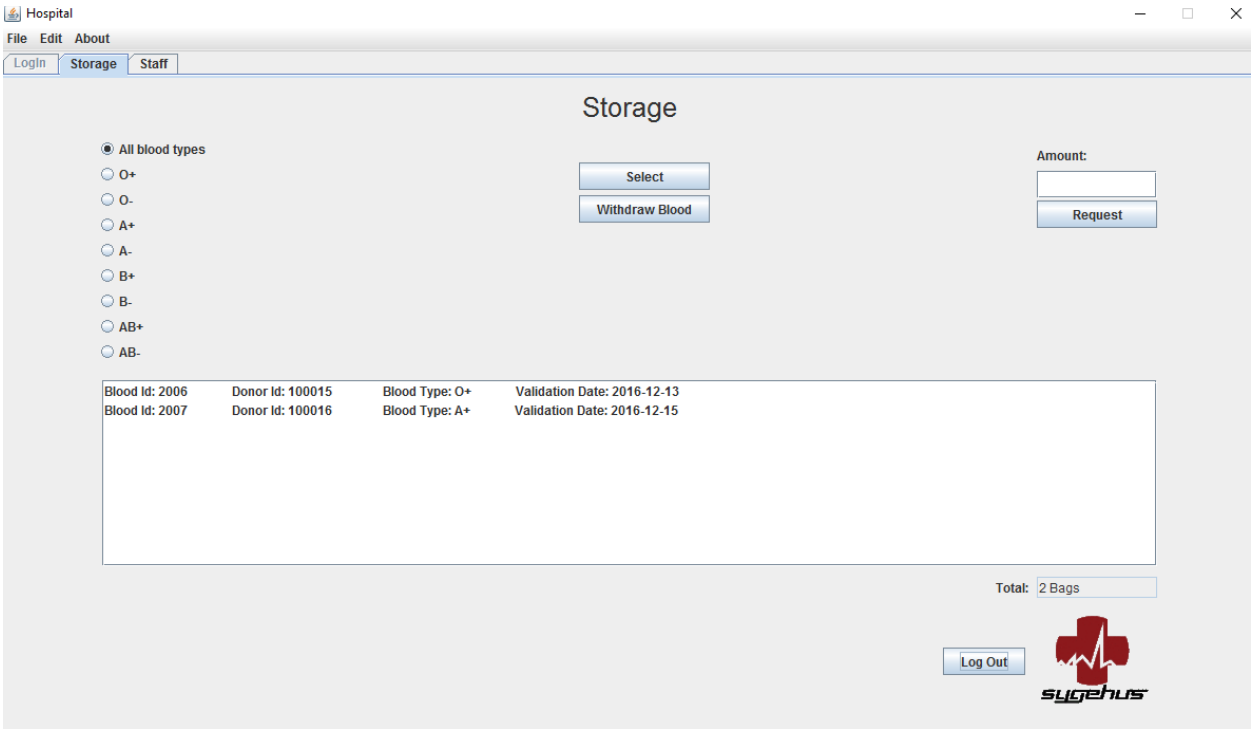

Figure 6- Blood storage

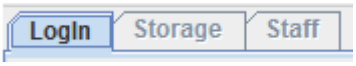

Figure 7 - Options for the hospital user

Class Diagram

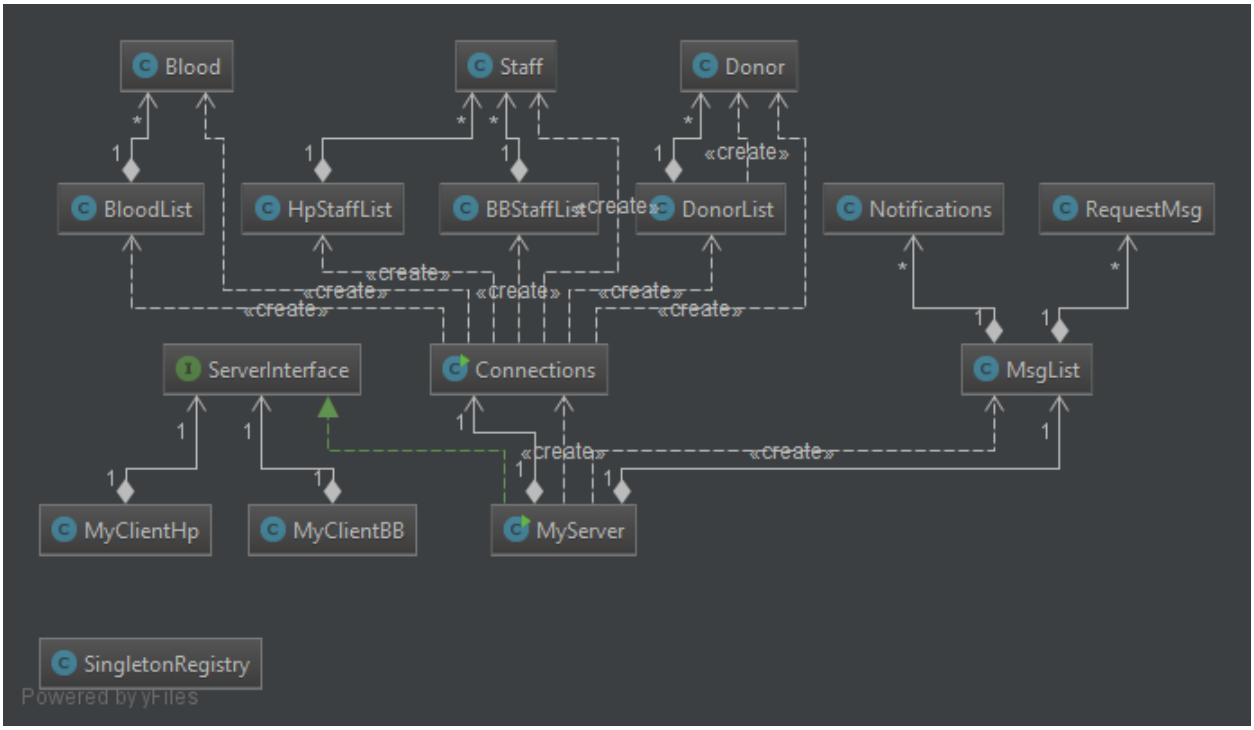

Figure 8 - Minimalistic class diagram

For full class diagram, see Appendix C

In order to facilitate the blood donation process, client/server system is needed. This means two clients can connect to a central server, those two clients are a blood centre and a hospital.

The server will make sure that the blood centre has access to the blood list and the donor stored in the Database.

The server will also hold all the communication between the blood centre and the hospital in which the hospital can request an amount of blood type from the blood centre, send a message to the blood centre when a specific blood bag was used and the blood centre can send the requested amount of blood to the hospital. All these operations go through the server which is going to deliver them to the other client.

Sequence Diagram

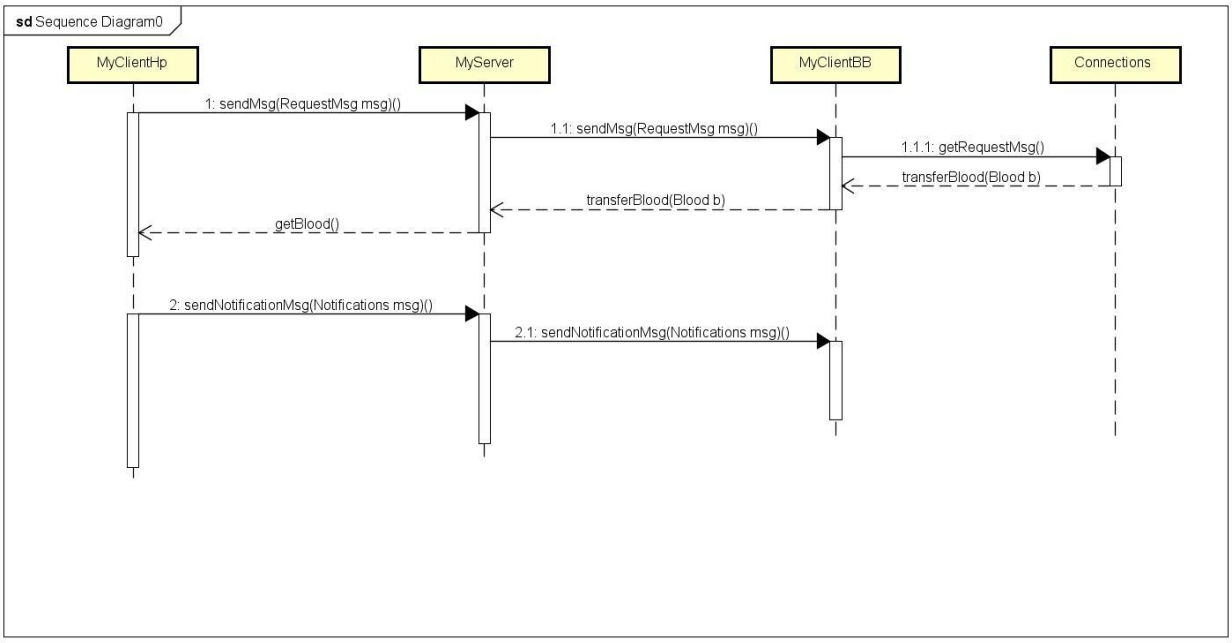

Figure 9 - Sequence diagram

In the figure above, in order for the hospital to make a request for an amount of blood type, it needs to be connected to the server. The server will then transform that request to the blood centre which will then supply the hospital with the requested amount of blood.

After every transfusion a notification process will begin in which the hospital will send a notification to the server. The notification holds the ID for the blood that was used, then the server will transform the notification to the blood centre which will send a message to the donor informing him/her that he/she saved someone else's life.

IMPLEMENTATION

The actual implementation of our system will be described in this chapter. It will include the Client-Server architecture and its implementations.

Client-Server architecture

In order for a user to access the Blood Bank Software, a Client-Server architecture can be used for this to be accomplished.

As the work was done only within java environment the network technology of choice was Remote Method Invocation(RMI).

The Blood Bank system was split into two clients part (Blood Centre Interface, Hospital Interface) and one server, with the first one holding the information about the donation process, managing the communication to and from the user (through a GUI) while the second one, holds the information about usage of blood, managing the communication to and from the user (through a GUI).

The server manages system's logic and provides the clients with the means to interact with it. The networking part is managed by the JAVA's built-in abstraction layer named RMI. It allows for an implementation where the methods on the server can be called by the clientas if they were on the same system, without the Need to define a new protocol.

Remote Method Invocation (RMI)

RMI allows and requires that certain objects are shareable through the Network.

Creating the connection:

First the RMI Registry needs to be started. The RMI listens for new connections on port 1099.

This connection is created by using a single instance with the SingletonRegistry class.

Then ServerInterface tells the RMI Registry to associate or "bind" its service name with the object. This allows the registry to know where to direct client requests and what ServerInterface stub class to use.

```
serverInterface=(ServerInterface)Naming.lookup("rmi://localhost:1099/BBank");
```

The whole project is divided in 4 packages (GUI, MC, RMI, Singleton);

In the RMI package there 4 classes MyClientBB, MyClientHp, ServerInterface, MyServer.
Being the ServerInterface the Interface which extends Remote and through which the objects are accessed on the Client sides.

```
public interface ServerInterface extends Remote {

   DonorList getAllDonors() throws RemoteException;
   ArrayList<String> getBloodLocations() throws RemoteException;
   HpStaffList getAllHospStaff() throws RemoteException;
   HpStaffList getHospitalStaffByID(int id) throws RemoteException;

   <.....>
}
```

These methods are then implemented in the MyServer class which is the main server with all the system's logic.

```
public class MyServer extends UnicastRemoteObject implements ServerInterface, Serializable {

   private Connections connections;
   private MsgList msgList;

   protected MyServer() throws RemoteException {
      connections = new Connections();
      msgList = new MsgList();
   }
```

As seen above we have an instance of the class Connections which is the one that connects to the DataBase. We can also see a list to store objects inside the server. In this we use both ArrayList and Database to store our data.
All features are implemented inside this class and every method throws a RemoteException.

```
public static void main(String args[]) {
   try {
      Registry registry = SingletonRegistry.getInstance().getRegistry();
      ServerInterface rmiServer = new MyServer();
      Naming.rebind("BBank", rmiServer);
      System.err.println("Server Ready");
   } catch (Exception e) {
      System.err.println("Server exception: " + e.toString());
      e.printStackTrace();
   }
}
```

We mentioned above we use the SingletonRegistry to create the registry used by the server.

The Registry can be accessed by the client's side when the Server is running by the Naming.lookup("rmi://localhost:1099/BBank");

```
public class MyClientBB implements Serializable {

   private ServerInterface serverInterface;

   public MyClientBB () throws RemoteException {
      super();
      try {
            serverInterface=(ServerInterface)Naming.lookup("rmi://localhost:1099/BBank");
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
```

```
public class MyClientHP implements Serializable {

   private ServerInterface serverInterface;

   public MyClientHP () throws RemoteException {
      super ();
      try {
            serverInterface=(ServerInterface)Naming.lookup("rmi://localhost:1099/BBank");
      } catch (Exception e) {
         e.printStackTrace();
      }
   }


public void addDonor(String fName, String lName, String cpr, int age, String address, int phoneno,
                String email, String bloodType, String location, Date date) {
   try {
      Statement statement = connection.createStatement();
      String sql = "INSERT INTO donors(id, fname, lname, cpr, age, address, phoneno, email,
bloodtype, bloodlocation, date)" +
            " " + "VALUES (nextval('incrementing'),'" + fName + "','" + lName + "','" + cpr + "','" +
age + "','" + address +
            "','" + phoneno + "','" + email + "','" + bloodType + "','" + location + "','" + date + "');";

      statement.executeUpdate(sql);
      connection.commit();

      ResultSet resultSet = statement.executeQuery("SELECT * FROM donors;");
      int ID = 0;
      while (resultSet.next()) {
         if (cpr.equals(resultSet.getString("cpr"))) {
            ID = resultSet.getInt("id");
         }
      }
      addDonorBlood(bloodType, ID);
      statement.close();
```

```
    } catch (Exception e) {
        System.err.println(e.getClass().getName() + ": " + e.getMessage());
        e.printStackTrace();
    }
    System.out.println("Records created successfully");
}
```

Above is the addDonor() method, throws RemoteException and its main functionality is to create a donation, by adding a donor and a Blood object to the Database

```
  public void transferBlood(String type, int amount) {
   try {
      Statement statement = connection.createStatement();

      String transfer = "INSERT INTO hpblood(bloodtype, donorssn, validationdate, bloodid) " +
            "SELECT bloodtype,donorssn,validationdate,bloodid FROM donorblood WHERE
bloodtype ='" + type + "' LIMIT " + amount + "";

      String delete = "DELETE FROM donorblood WHERE bloodid IN(SELECT bloodid FROM
hpblood WHERE hpblood.bloodid=donorblood.bloodid)";

      statement.executeUpdate(transfer);
      statement.execute(delete);

      statement.close();
      connection.commit();
    } catch (Exception e) {
      System.err.println(e.getClass().getName() + ": " + e.getMessage());
      e.printStackTrace();
    }
}
```

Above is the transferBlood() method, also throws RemoteException and its main functionality is to transfer blood from the BloodCenter to the Hospital

Took place through the entire development process. Upon implementing a new feature to the system, tests were made to determine if the new feature is working properly.
For the final test of the system, several users were asked to try and use the application in the purpose that it was built for.

USE Case TEST Implementation

HOSPITAL

Log in:

The test was performed by filling the User text field and the Password text field with the correct data and by pressing the Enter Key, or by pressing the Log In Button.
When pressed or clicked, an action will unblock the next tabs of the GUI, and changes from the first tab to the second tab and blocks the first tab

Testing Result: Success, First try.
Testing exceptions or errors: None.

See Available Blood

The test was performed by selecting a radio button and press the Select button.
When the button is clicked the data is to appear on the list below, depending on which radio button is selected, different type of data shows up.

Testing Result: Success, First try.

Testing Exceptions or errors:

None.

Register Used Blood

The test was performed by selecting an object from the jList and click the Withdraw Blood button.
The system will then delete the object selected and send a message type string to the server and store it in an ArrayList.

Testing Results: All cleared

Testing Exceptions or errors:

None.

Check Staff Members

The test was performed by press the Show All Staff Members button. All staff members are to show up in the listArea below.

Testing Results: All cleared

Testing Exceptions or errors:

None.

BLOODCENTER

Add Donor / Add Blood

The test was performed by selecting any of the two radio buttons, depending on the situation, filling the text fields required and by clicking the donate button.

The system will then add a donor to the table DONORS in the Database, and add a Blood object to the DONORBLOOD table in the Database.

Testing Results: We had some problems when the input on the fields was incorrect.

Testing Exceptions:

Exception 1: Some sort of PostgresSQL Exception, when the wrong input was given on the text fields

Donation History

The test was performed by pressing the Show History button, All Former DONORS are to show up in the list below.

Testing Results: All cleared

Testing Exceptions or errors:

None.

See the Storage

The test was performed by selecting a radio button and press the Select button.
When the button is clicked the data is to appear on the list below, depending on which radio button is selected, different type of data shows up.

Testing Result: Success, First try.

Testing Exceptions or errors:

None.

Remove Blood

The test was performed by selecting an element from the jList and by pressing the Remove Blood button. When the button is pressed the data selected is to be removed from the Database;

Testing Results: Success, First try.

Testing Exceptions or errors:

None.

Check Staff Members

The test was performed by press the Show All Staff Members button. All staff members are to show up in the listAerea below.

Testing Results: All cleared

Testing Exceptions or errors:

None.

Check Notifications

The test performed by selecting a radio button and click the select button. The notifications should then appear in the small listArea with a certain
format depending on the radio button selected.

Testing Results: All correct

Testing Exceptions or errors:

None.

Logout

The test was performed by pressing the Log Out button laid out on the menu. After pressing the button, a yes or no pop up should appear asking if the user really wants
to the log out. If yes is pressed the user will be taken to the Log IN tab.

Testing Results: All correct

Testing Exceptions or errors:

None.

RESULT

The results of this project is a functional system for a Blood bank where you have a donation process and transfer the donated blood to the Hospital.
Those results were achieved by fully developing the main features of this project.

DISCUSSION

All the functions that were considered to be important for a blood bank were implemented successfully. The least important features of the system were cancelled, for example, notify the donor when his/her blood was used was one of the functions cancelled and was not part of the system, since this was a human part and it's up to the blood centre's staff to send the message to the donor and inform him/her.

CONCLUSION

The main objective part of this project was to transfer the requested amount of a blood type from the blood centre's database to the hospital's database.
It can be concluded that overall the project was a success, as it managed to fulfil all the requirements. Since the project period was only for three weeks, it prevented the system from receiving many additions, which ended up becoming part of the delimitations

APPENDICES

- Appendix A – User Guide
- Appendix B – Project Description
- Appendix C – Class diagram
- Appendix D – Controller diagram
- Appendix E – RMI diagram
- Appendix F – RMI and controller diagram

REFERENCES

[1] https://www.quora.com/Why-dont-more-people-donate-blood

[2] http://www.independent.co.uk/news/uk/blood-donors-to-receive-a-text-before-their-blood-is-used-in-hospital-a7081136.html

[3] Chou, E., & Murnighan, J., 2013. *Life or Death Decisions: Framing the Call for Help PLoS ONE* [Online]
Available at: http://stochasticscientist.blogspot.dk/2013/03/how-to-get-people-to-donate-blood.html

[4] [7] [9] Asian Journal of Transfusion Science, 2010. *Blood donor incentives: A step forward or backward* [Online]
Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2847338/

[5] Elsevier B.V., 2016. *EBioMedicine* [Online]
Available at: http://www.sciencedirect.com/science/article/pii/S2352396416302481

[6] NHS Blood and Transplant Organization, 2015. New blood donors in decline: 40% fewer new blood donors in 2014/5 than 2004/5 [Online]
Available at: https://www.blood.co.uk/news-and-campaigns/news-and-statements/news-new-blood-donors-in-decline-40-fewer-new-blood-donors-in-20145-than-20045/

[8] World Health Organization, 2016. *Blood safety and availability* [Online]
Available at: http://www.who.int/mediacentre/factsheets/fs279/en/