ICT ENGINEERING

Project Report

Client/Server Banking System

Supervisors

Troels Mortensen (TRMO)

Jens Cramer Alkjærsig (JCA)

Group

Bozhidar Nedyalkov (260383)

Daniel Molnar (260059)

Nikola Vasilev (260099)

Patrik Ihnat (260010)

Zahari Dzhelepov (253925)

Abstract

The aim of this project is to create a simulation of a client/server banking system. This software is created for people who would like to use banking activities through online. Each bank has to perform many activities daily, which required big infrastructure and hiring more staff members. The banking system allows bank to perform many of its activities, in a faster way, without involving employees directly in the process. The major idea behind the banking system is to provide an array of services to the customer in a way that the user will have more flexibility and lose less time operating the system. Before we started off with the implementation of the system, we had to create the requirements based on the wishlist of the customer. We had to plan the whole system by making use-case diagram and class diagram modeling in order to create a functional. The whole system is coded using Java language. For the connection between Server/client Remote Method Invocation (RMI) is used. Design patterns which are used are Adapter and MVC. System needs to store information about the customers like account number, name, balance and more. Database, created with SQL, comes along with the software so information can be stored there. Graphical user interface(GUI) provide the user with easy access and usability. As a result, all mandatory requirements are fully functional. Due to lack of time not everything intended is implemented but the system is open for improvements.

Contents

Introduction

Internet banking offers an array of different advantages to the user, including: account balances and history including year-to-date information, the ability to transfer money from one account to another inside of the bank, check history, check balances and more. Internet banking basically allows you to be able to do everything that you can in your regular banking institution, only with the benefit that you don't need to go to the bank. Not only is this saving time because you can be comfortable and have peace of mind knowing that you can keep track by yourself of all your banking issues, but as well it allows for more flexibility because you don't have to worry about rushing out and making it to the bank. The aim of this project is creating a client/server system for a bank. Main objectives which are included are: user has the ability to send and receive money, login and log out, check balance and check transaction history. Technical problems and challenges which are presented in this report are server/client connection, connecting the database to the server and creating a functional GUI. Delimitations of the system include inability to recover a username or a password if they are forgotten by the user. Password chosen by the user is not encrypted and the user is unable to log in using Nem-ID security login. Next chapters include functional and non-functional requirements which cover and show the objectives of the project according to the aim. Analysis part demonstrates the logic behind the architecture, identify objects in the problem domain and display how they are cooperating. The design chapter presents how the system is structured, interface design choices, which design patterns are used and data models. Implementation part trace how server/client connection is established, how design patterns are implemented and how data base is connected to the server.

Functional system requirements

1. The user can login.
2. The user can check balance on his account.
3. The user can transfer money between accounts.
4. The user can see his transactions.
5. The user can logout.
6. The user can view personal details (phone no., email).
7. The user can update personal details (phone no., email).
8. The user can set monthly payments.
9. The user can take a loan.

Non-Functional requirements

- Client/Server(RMI)
- Database
- GUI (Graphical user interface)
- Login based on password.

Analysis

This chapter covers the analytical process which precedes the development of the software. Core feature of the system is the server/client connection. This feature allows the user to send money to different accounts and receive money from different accounts. In order to establish this connection Remote Method Invocation (RMI) is used.

The banking system features are listed in the Functional system requirements chapter. In order to transfer money (send money to another account) the user must input some information such as account number of the receiver and the amount of money which the client wants to be transferred. The system also has other functionalities such as editing personal information (i.e. email/phone number) and viewing their transactions. In order to do that a database and the ability to manipulate it is required. So a database was created and connected to the server. Password and nickname information of the user are as well stored in the database.

To make accessibility easier a graphical user interface (GUI) is created. From the GUI the user can log in and use the software. GUI is divided in tabs, where features of the program are put together in a logical manner. From GUI the user is able to send money, edit personal information and check past transactions.

Java is used for the server/client connection side and SQL is used for the database part of the project.

Use Case Modeling

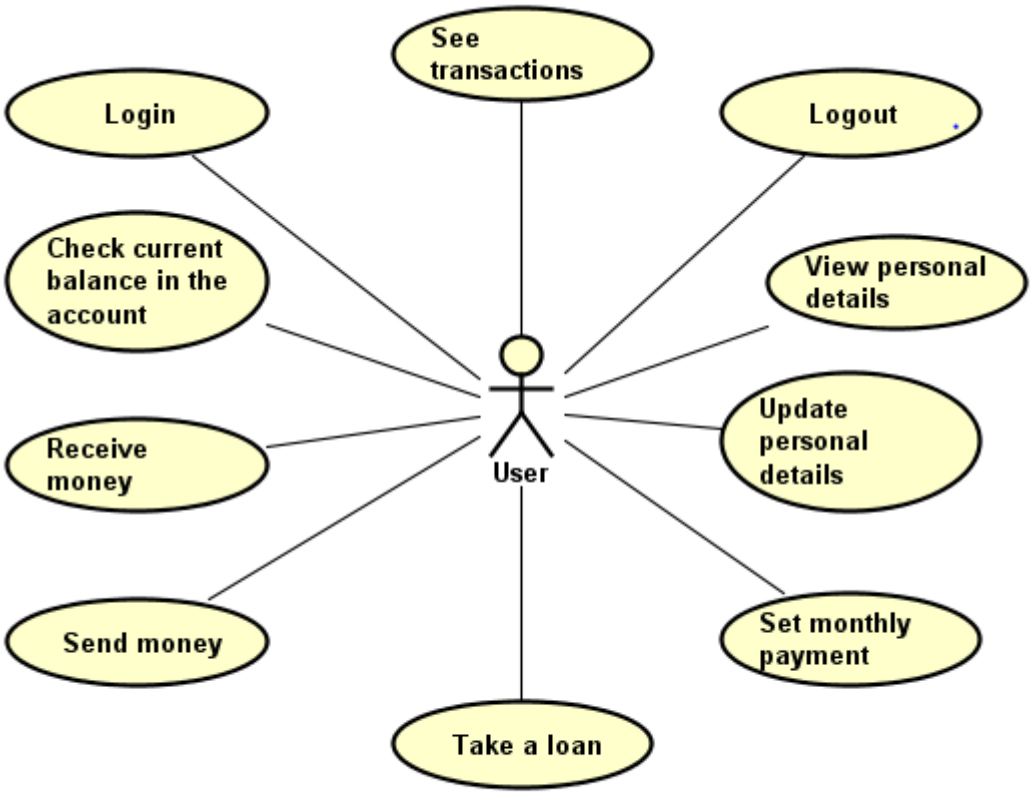


Figure 1: Use case diagram

Figure 1 Illustrates some of the various scenarios, which the user can encounter during a typical usage. Expected capabilities of the system are shown in the user case diagram.

Use Case Description

Use Case	Send Money
Summary	User should be able to send money to other customer.
Actor	User
Precondition	User wants to send money to a different customer.
Post condition	Money is sent. Balance of the user is updated.
Base Sequence	1. User enters account number. 2. User enters receiver account number.

	3. User enters transaction amount. 4. User enters date in a specific format. 5. User finishes sending money by clicking a button.
Exception Sequence	Data not valid: 1-5 System asks the user to enter valid information.

Table 1: Use case description of "Transfer money" use case

Table 1 contains description of “Transfer money” use case. It is one of the most common operation which can be done via server/client banking software.

Design

GUI Design

This part of the report represents part of the user interface responsible for log in.

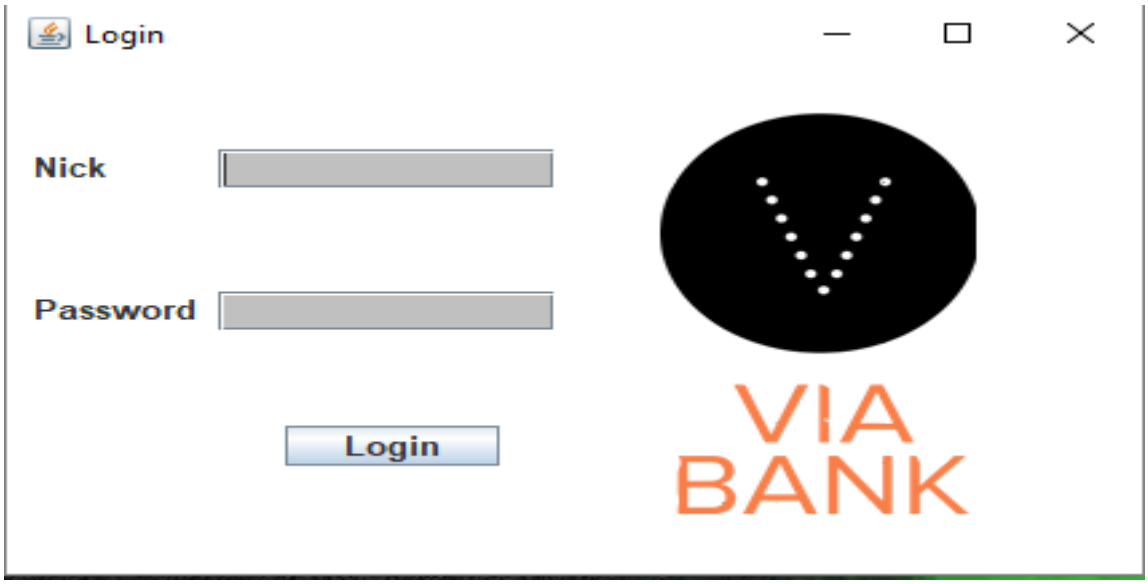


Figure 4: GUI “Log in” window

This is the first tab which the user sees upon starting the program. It contains two fields: Nick and Password. The user fills in personal nick and password and press Login button to enter his account.

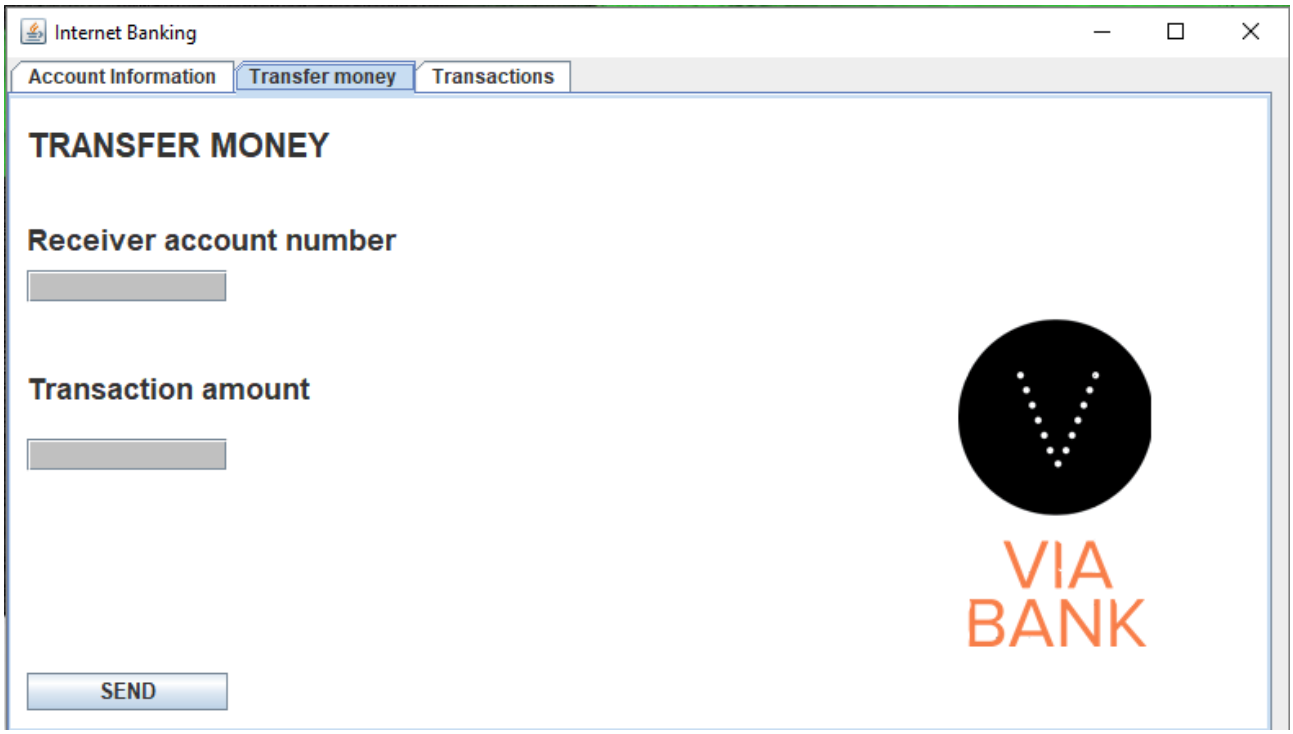


Figure 5: GUI "Transfer money" WINDOW

Transfer money tab is used to transfer money between accounts in the same bank. To transfer money, the user must fill in the account number of the receiver and fill in the desired amount of money, which they want to be transferred. Transfer is done when the user press the SEND button.

For more information and guide on the use of the GUI please refer to Appendix-E.

Design Class Diagram

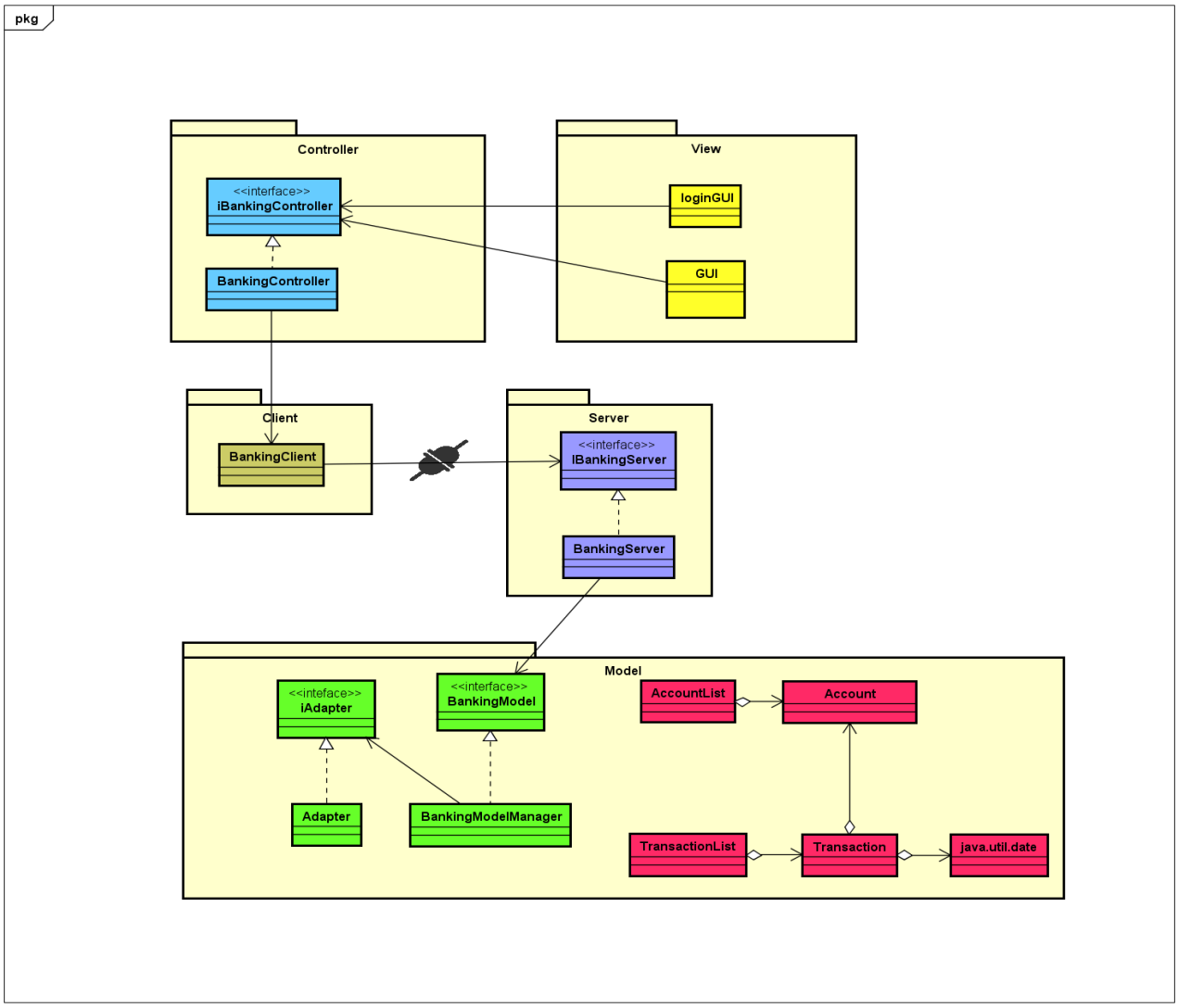


Figure 6: Minimized version of the Class diagram

Figure 6 shows a minimized version of the class diagram which describe our system. It shows two design patters which are used: MVC and Adapter. Account class holds fields and methods for the accounts. AccountList class contains all the accounts. Transaction class contains fields and methods for transactions. TransactionList holds information for all the transactions. Java.util.date is from java library.

Adapter class is used to take information from database and with the help of the interface iAdapter sends it to the BankingModelManager, which also by the help of its interface BankingModel allows the data from the database to be sent to the BankingServer. Then the BankingServer class itself process it to the BankingClient with the help of iBankingServer interface. Then we have the BankingController class which is used to process incoming requests and handles user input. The view part of the MVC is the two Graphical User Interface classes' loginGUI and GUI.

For full diagram with all the methods please refer to Appendix-B.

Design Pattern

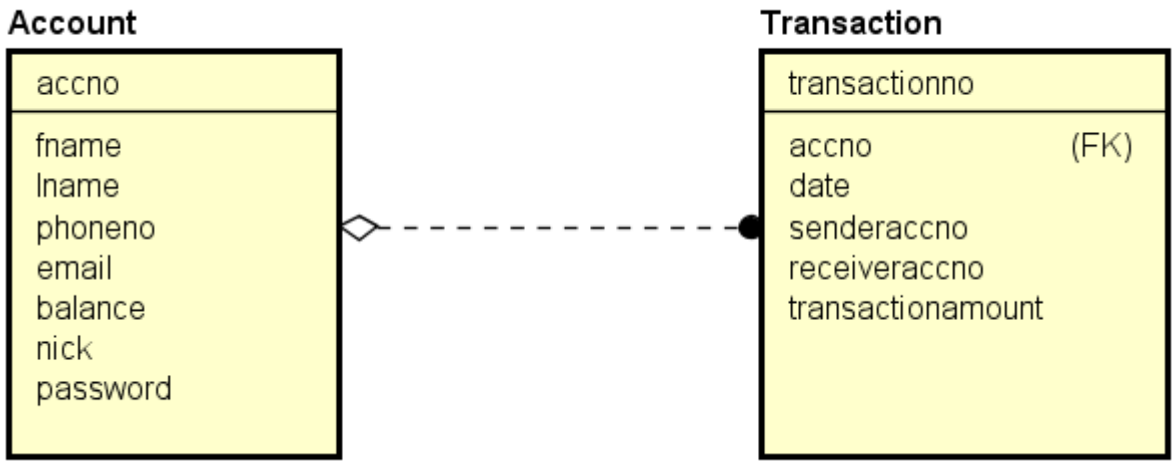
- MVC

Model View Control pattern is used in this project because of the importance of the controller which is used for sending or retrieving data from database and its ability to handle incoming requests and user inputs. Both the view and the model depend on the controller. The view is important for displaying information or a user can input data and the controller needs to handle that. The model is used for storing information in sort of a cache data so that when an error in the database occurs, to still get information and controller again handles from where that should happen.

- Adapter

Adapter is used to make existing classes work with others without modifying their source code. So if another class` interface from our system is incompatible with another one we use the Adapter design pattern to connect our database to the Server in our system.

Data base design



A very simple database is created to store/retrieve data.Two tables are used to store all of the data of accounts and transactions. Account table store data such as account number (primary key), full name and details about user. Transaction table is used to store all of the transactions between users.

Implementation

- Adapter (inserting transactions)

In this method with the use of JDBC we are inserting transactions to the database. At the beginning we have to declare database driver and establish connection. Then we are using

command “PreparedStatement” for inserting specific data to database using SQL language mixed with java variables.

```
//INSERT TRANSACTION
public void insertTransaction(Transaction t) throws ClassNotFoundException, SQLException, RemoteException
{
  //Driver
  Class.forName("org.postgresql.Driver");
  Connection con = null;
  con = DriverManager.getConnection ("jdbc:postgresql://localhost:5432/BANK","postgres", "Meandiou123");

  //Inserting
  PreparedStatement insert = con.prepareStatement ("INSERT INTO "
        +"Bank.transaction"+ " (AccNo,Date ,senderAccNo,recieverAccNo, transactionAmount) "+
        "VALUES("+t.getReceiverAccNo()+",'"+t.getDate()+"',"+t.getSenderAccNo()+","+t.getReceiverAccNo()+","+t.getTransactionAmount()+ ")");
  insert.executeUpdate();

  //Closing
  if (con != null)
  con.close ();
}
```

- Adapter (view account info)
  This method has the same beginning as the previous one, we decided to use database declaration in every method inside adapter to be sure that our connection to database is always established. Than there is declaration for “DefaultTableModel” that is our model for getting account information from database and being able to insert them into JTable(GUI). Also, there is used JDBC for obtaining information from database based on account number. At the end we are putting them in the correct order.

```
//VIEW ACC INFO
public DefaultTableModel viewAccountInfo(int AccNo) throws ClassNotFoundException, SQLException, RemoteException {

    //Driver
    Class.forName("org.postgresql.Driver");
    Connection con = null;
    con = DriverManager.getConnection ("jdbc:postgresql://localhost:5432/BANK","postgres", "Meandiou123");

    //Model
    DefaultTableModel mod = new DefaultTableModel(new String[] {"First name: ","Last name: ","Account Number: ","Phone Number: ","Email: ","Balance: ","Login Nickname"},0);

    //Personal info view
    PreparedStatement read = con.prepareStatement ("SELECT * FROM Bank.accounts WHERE AccNo =" +AccNo );
    ResultSet rs = read.executeQuery();
    while(rs.next())
    {
        String fname = rs.getString(1);
        String lname = rs.getString(2);
        String accno = rs.getString(3);
        String phone = rs.getString(4);
        String balance = rs.getString(5);
        String email = rs.getString(6);
        String nick = rs.getString(7);

    mod.addRow(new Object[] {fname,lname,accno,phone,balance,email,nick});

    //Closing
    if (con != null)
    con.close ();
    }
    return mod;
}
```

- Controller (login-logout)
  At the beginning we are declaring “DefaultTableModel” which is called “tab”. In for loop the password and nickname are compared with all passwords and nick names which are stored in database. Through this login we are obtaining account number and putting out all data about specific account from database.

```
public boolean login(String nick, String pass) throws ClassNotFoundException, RemoteException, SQLException {
   DefaultTableModel tab = new DefaultTableModel();
   tab = client.viewAllAccountInfo();

   for (int i = 0; i < tab.getRowCount(); i++) {
      if ((nick.equals(tab.getValueAt(i, 5).toString()))&&(pass.equals(tab.getValueAt(i,6).toString()))) {
      accNo = Integer.parseInt(tab.getValueAt(i,2).toString());
      return true;
      }
   }
   return false;

}
public void logout() {
   accNo = 0;
}
```

For full code please refer to Appendix-A.

Test

For the purpose of this project Scenario testing via GUI is used.

Use case	Tested
The user can log in.	Yes
The user can send money.	Yes
The user can see his transactions.	Yes
The user can check balance on his account.	Yes
The user can receive money.	Yes
The user can logout.	Yes
The user can view personal details (phone no., email).	Yes
The user can update personal details (phone no., email).	Yes
The user can set monthly payments.	No
The user can take a loan.	No

Test Specification

1. Log in

- Type in nick and password into text fields
- Pressing (login) to log into your account

Expectation: System opens new window with information about logged in user

Error: NONE

2. Log in

- Type in incorrect nick or password into text fields

- Pressing (login) to log into account

Expectation: System pop-out message wrong nick name / password.

Error: NONE

3. Transfer money

- Pressing (Transfer Money) tab opens a new page
- Type in receiver account number
- Type in amount of money to be sent
- Pressing (Send) button completes the transaction

Expectation: Money was transferred inside the database and pop-out message will show text “Relog to see changes”.

Error: NONE

4. See transactions

- Pressing (Transactions) tab opens up a new page
- List displays all transactions

Expectation: List displays all transactions were logged in user was part of (Sender or receiver).

Error: NONE

5. Change phone number

- Filling text field for phone no.
- Pressing “Change Phone no.” button

Expectation: New phone number will be added to database and pop-out message will say “relog to see changes”.

Error: NONE

6. Change email

- Filling text field for email
- Pressing “Change E-mail” button

Expectation: New email will be added to database and pop-out message will say “relog to see changes”.

Error: NONE

Results and Discussion

The purpose of the results and discussion section is to present the outcome and achieved results of the project.

Requirement	Implementation	Test
The user can login.	Done	Pass
The user can send money.	Done	Pass
The user can see his transactions.	Done	Pass
The user can check balance on his account.	Done	Pass
The user can receive money.	Done	Pass
The user can logout.	Done	Pass
The user can view personal details (phone no., email).	Done	Pass
The user can update personal details (phone no., email).	Done	Pass
The user can set monthly payments.	Not done	-
The user can take a loan.	Not done	-

All of the tested features have passed the test successfully.

Conclusion

Project spans through three weeks. At start all requested requirements are structured and divided in functional and nonfunctional. Then they are put in hierarchical order by importance. Useful diagrams are prepared to help the analysis part so the core architecture of the software is understandable for everyone in the team. Design part of the project includes design of the server/client connection, data base and GUI. According to design choices the features of the program are implemented and connected to each other. In the end stage the program is tested. Result is that Server/client banking system meets all the obligatory requirements and it is fully functional.

Future of the project

Reflect on your project from a technical viewpoint and describe what you would change if you could.

Suggest how the project could be improved or made ready for production. Discuss scalability, suggest possible spin offs, what is needed, missing, etc.?

Due to the lack of time some possible features are not implemented or are implemented in not the best way possible. Server/client connection could be implemented using Sockets instead of RMI. This way users don't have to be connected to the same network connection when they want to send or receive money.

Log in procedure could also be improved. At the moment the password is not encrypted. Another possible way of logging in could be by using a different code every time the user tries to log in (similar to Danish Nem-ID).

References

Larman, C. (2004). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. *Analysis*. https://doi.org/10.1016/j.nec.2006.05.008

Connolly, T., & Begg, C. E. (2010). *Database Systems: a practical approach to design, implementation, and management*. *Database Systems: a practical approach to design, implementation, and management*.

https://studienet.via.dk/Class/IT-SDJ2X-A17/Session%20Material/Forms/Default.aspx

Supplementing material

Description of the materials supporting our solution are attached in appendices, as following:

Appendix A: **Code.**

Appendix B: **Class Diagram.**

Appendix C: **ER Diagram.**

Appendix D: **User guide.**