

VIA University
College

SEP2 CLASS Y - PROJECT REPORT

CLIENT SERVER SYSTEM:

CHAT SYSTEM FOR A SMALL COMPANY

CREATED BY:

Edi Trifonov:	241926
Titas Jackus:	239853
Daniel Borisov:	239925
Mario Burgov:	240303

SUPERVISED BY:

Steffen Vissing Andersen

Hand in date: 03.06.2016

TABLE OF CONTENTS

ABSTRACT	3
INTRODUCTION	4
ANALYSIS	4
REQUIREMENTS	4
Functional.....	4
Non-Functional.....	5
USE CASE MODELLING	5
USE CASE DESCRIPTION	7
ACTIVITY DIAGRAM	8
SEQUENCE DIAGRAM	10
Model-View-Controller	10
CLASS DIAGRAM FOR MODEL	12
DESIGN	12
Graphical User Interface	12
Implementation	14
TEST	16
RESULT.....	17
DISCUSSION	18
CONCLUSION	19
REFERENCES.....	19
APPENDICES.....	20

ABSTRACT

The chat system is a system supposed to help out companies solve their problems with communication between employees. It creates a connection between users and administrators so they have the possibility to communicate between each other. It enables them to connect, send messages, receive messages and disconnect from the system. The system differentiates between users and administrators, giving the latter more options to choose from. All features are applicable while working with PostgreSQL. The system is constructed to be easily maintainable and efficient, making it easier for future changes. It is built upon a user interface which makes the user experience better overall.

The project was built upon the usage of Scrum theories and methodology as well as Agile Unit Processing. It was split on different Sprints and AUP phases which eventually led to the completion of the project. Problems, of course appeared throughout the course of completing the project and we gained a lot of experience from them.

INTRODUCTION

Communication has turned out to be an important factor for the development of modern companies. That is why they need a good way of communication between their employees. Members of certain companies have to spend a lot of time communicating between each other every day which is a hugely time consuming task.

This is why, the Product Owner has looked into the problem and decided to order a software system that could fix that problem. The software system will allow sending messages between employees in a company.

The Chat system's purpose is to help companies solve the problem with the time consuming communication between their employees. Based on the needs of the Product Owner a set of requirements was established. This set of requirements contains his requests, which he wishes the new system should have. Starting from those requirements a fully operating system was made. The system was developed entirely in Java. It features a somewhat user friendly GUI, developed from scratch, just to make sure the user experience is kept at the highest standard.

ANALYSIS

The system will allow users or administrators to login into the system and then log out whenever they want. The user and administrator will be able to send messages, edit or delete them. On top of that the administrator will have the rights to modify, create or delete a user. He will also be able to get a list of chat history for a certain period.

REQUIREMENTS

Below is the list of all the requirements of the system:

Functional

1. CLIENT-SERVER SYSTEM: The software should be based on a client-server connection.

2. SENDING MESSAGE: The user or administrator can send message(s) into the system and the message(s) should be received by the other users and administrators.
3. LOGGING IN AND OUT: The user or administrator should be able to log into the system with this own credentials and log out whenever he wants.
4. ALL INFORMATION STORED IN A DATABASE: All data should be stored in a database to prevent data loss after program quits.
5. GUI: Everything will be in graphical interface, so user can easily manage with the program.

Non-Functional

6. CREATING USER: The administrator should be able to create a new user.
7. DELETE USER: The administrator should be able to delete a user.
8. MODIFY USER: The administrator should be able to modify a user's credentials.
9. SWITCH CHAT ROOMS: The user or administrator should be able to switch chat rooms.
10. SHOW ONLINE USERS AND AMINISTRATORS: The user or administrator should be able to see a list of online users and administrators.
11. SHOW HISTORY OF MESSAGES: The administrator should be able to get a text file with chat history for a certain period.

USE CASE MODELLING

After each Sprint was completed, the main user and administrator diagrams were updated.

The actor in the shown diagram is the administrator of our system.

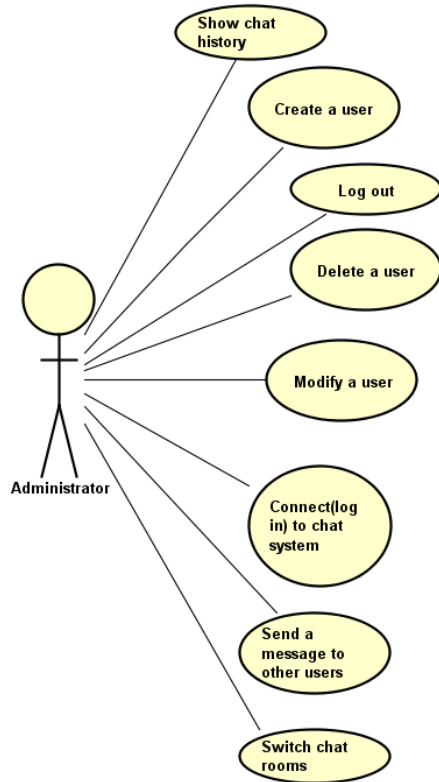


Figure 1 – Use case Diagram for Administrator(See Appendix A for all Use Cases)

Show Chat History- The administrator is able to get a chat history for a certain period.

Create a user- The administrator is able to create a new user.

Delete a user- The administrator is able to delete an existing user.

Modify a user- The administrator is able to modify a user's information.

Log out- The administrator is able to log out of the system.

Connect(log in) to chat system – The administrator is able to log into the system.

Send a message to other users – The administrator is able to send messages to other users in the system.

Switch chat rooms- The administrator is able to switch chat rooms.

The actor in the shown diagram is the user in our system.

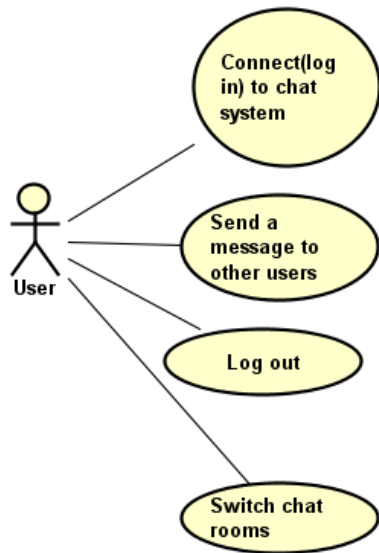


Figure 2- Use case Diagram for User(See Appendix A for all Use cases)

Log out- The user is able to log out of the system.

Connect(log in) to chat system – The user is able to log into the system.

Send a message to other users – The user is able to send messages to other users/administrators in the system.

Switch chat rooms- The user is able to switch chat rooms.

USE CASE DESCRIPTION

Below you can see a description of “Send a message to other users” use case, which is one of the most important features in the system. (To see all the Use Case Descriptions and Use Case Diagrams, see Appendix A).

UseCase:	Send a message to other users
Summary:	The user/administrator is able to send messages(s) to other user(s)/administrator(s)
Actor:	User/Administrator
Precondition:	There are some users/administrators connected to the system
Postcondition:	The other users/administrators receive user's message.
Base Sequence:	1. User/Administrator types a message.



	<ol style="list-style-type: none">2. Server receives the message and broadcasts it to the other users/administrators(clients).3. Users/administrators receive the message.
Exception Sequence:	<p>In case the server crashes:</p> <ol style="list-style-type: none">1. User/administrator types a message.2. User/administrator receives an error message.

Figure 3- Use case Description for “Send a message to other users”(See Appendix A for all use cases and activity diagrams)

ACTIVITY DIAGRAM

The activity diagram found below shows in detail all the capabilities and actions of the user or administrator when sending a message.

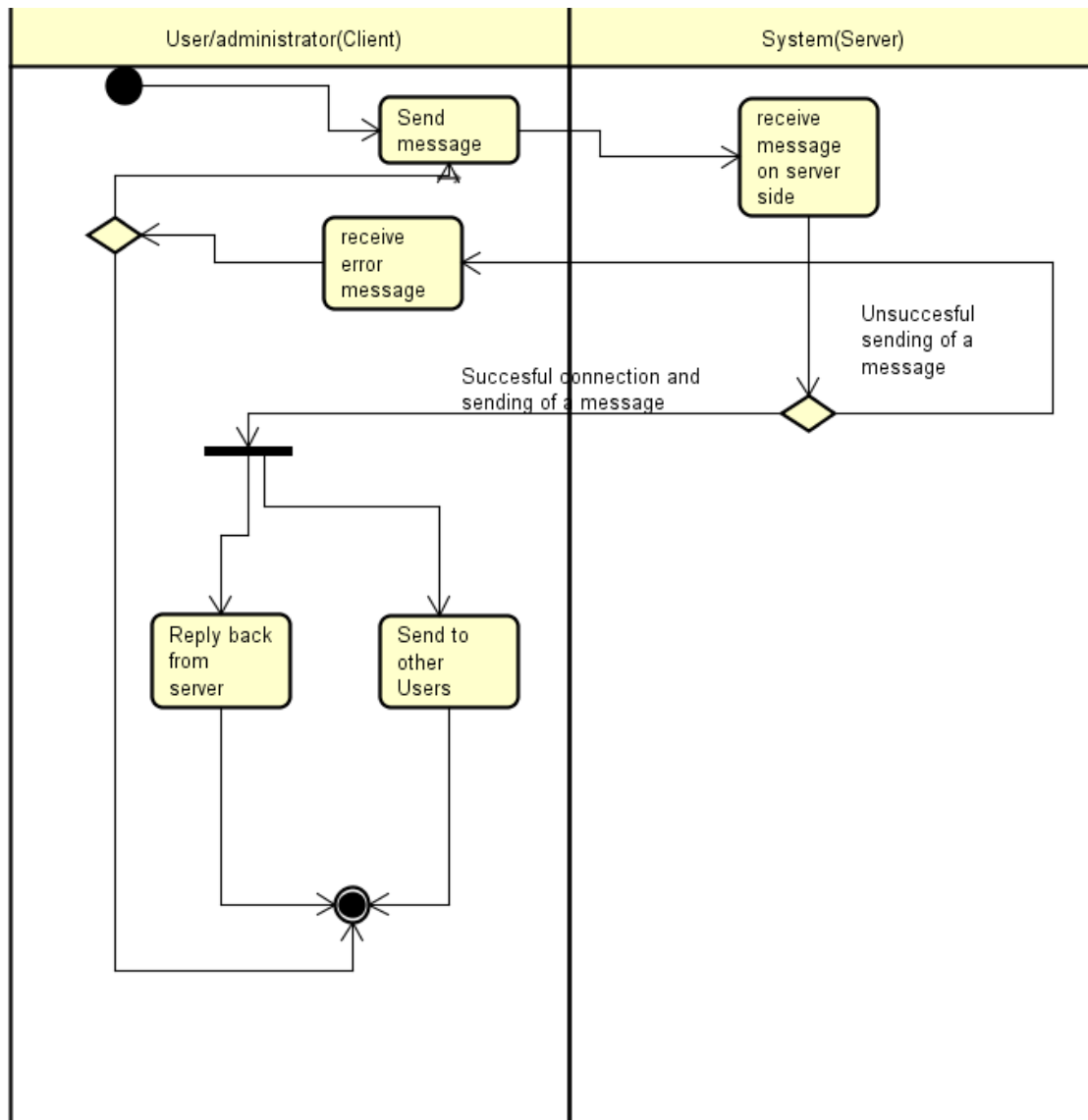


Figure 4 – Activity Diagram(See Appendix A for all Activity Diagrams)

The course of action while sending a message is simple: User or administrator types a message, server receives that message and if it successfully connects with the client it gives the client a reply and sends the message to the other users. If it doesn't connect successfully the user receives an error and he can choose to send a new message.

SEQUENCE DIAGRAM

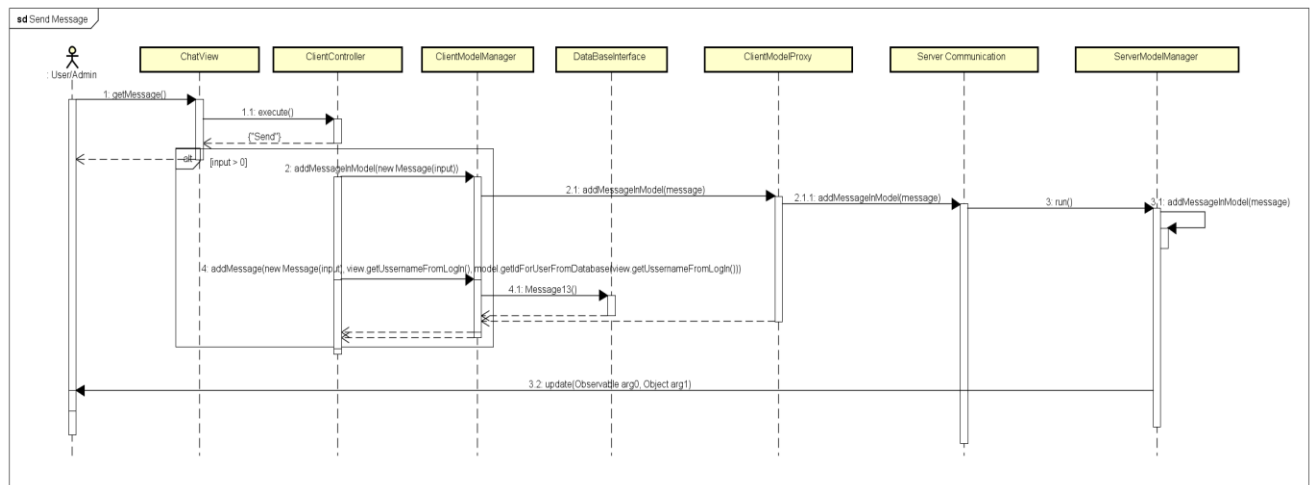


Figure 5 – Sequence Diagram

Model-View-Controller

The system is based on a Model-View-Controller (MVC) design pattern consisting of the following parts:

Model, which consists of the model package, taking care of storing and processing data.

Two controllers(ClientController,AdminController) that act on both the model and the view. They control the data flow into the model objects and update the view whenever data changes. The controllers have access to the model through ClientModelManager and ServerModelManager.

Views, which are retrieving requests from the user and passing them to the controllers and also updating whenever there are changes to the model.

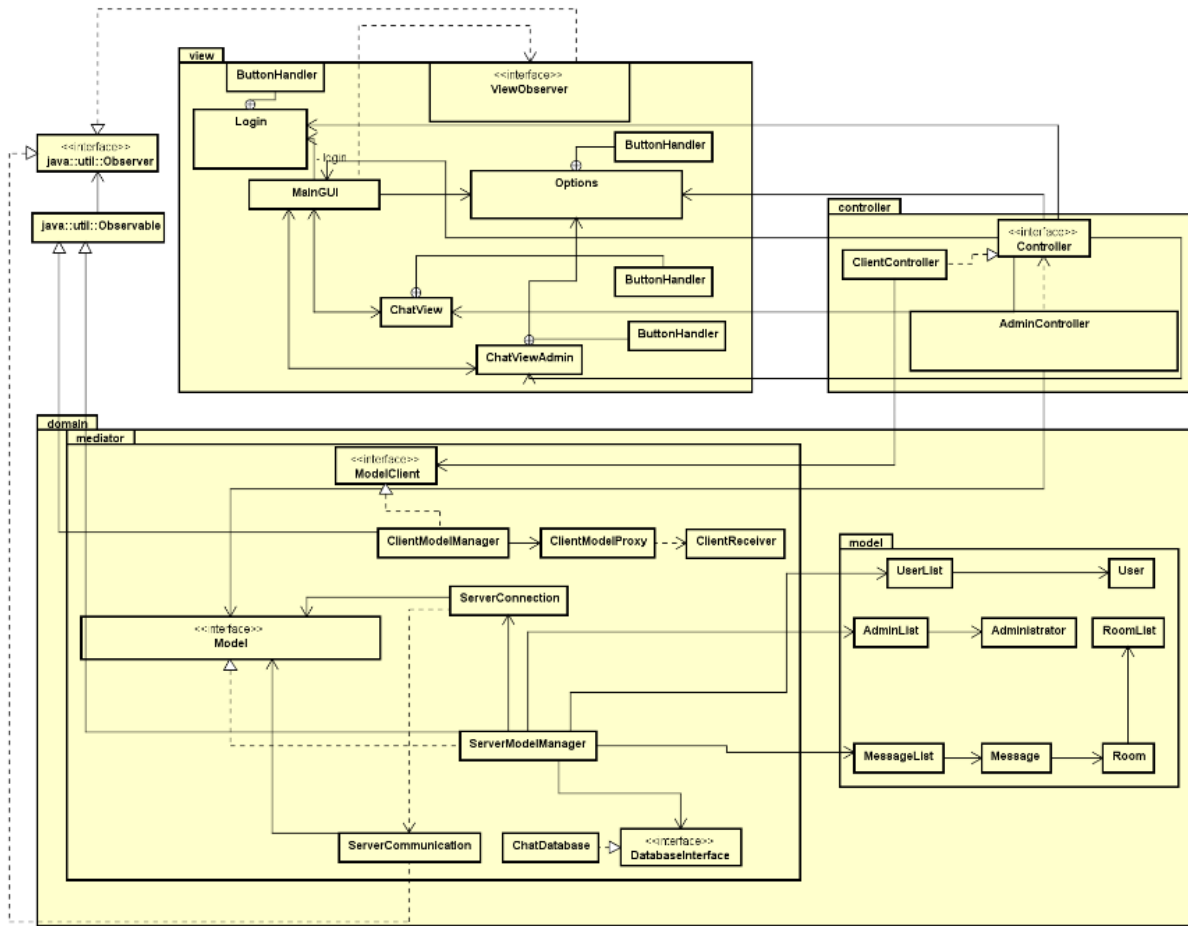


Figure 6 – MVC Pattern(See Appendix E for full Class Diagram)

In order to show updated data in the View, an observer pattern is added. ClientModelManager and ServerModelManager extend the Observable class from Java API while the ServerCommunication and ClientModelProxy implement the Observer interface.

One of the most important parts on the server side is the relation between ServerModelManager and the interface of the database. This allows the ServerModelManager to communicate with the database through SQL statements which was a key element in the system.

CLASS DIAGRAM FOR MODEL

The class diagram below represents the model from MVC design pattern which contains Objects and lists of Objects of the system.

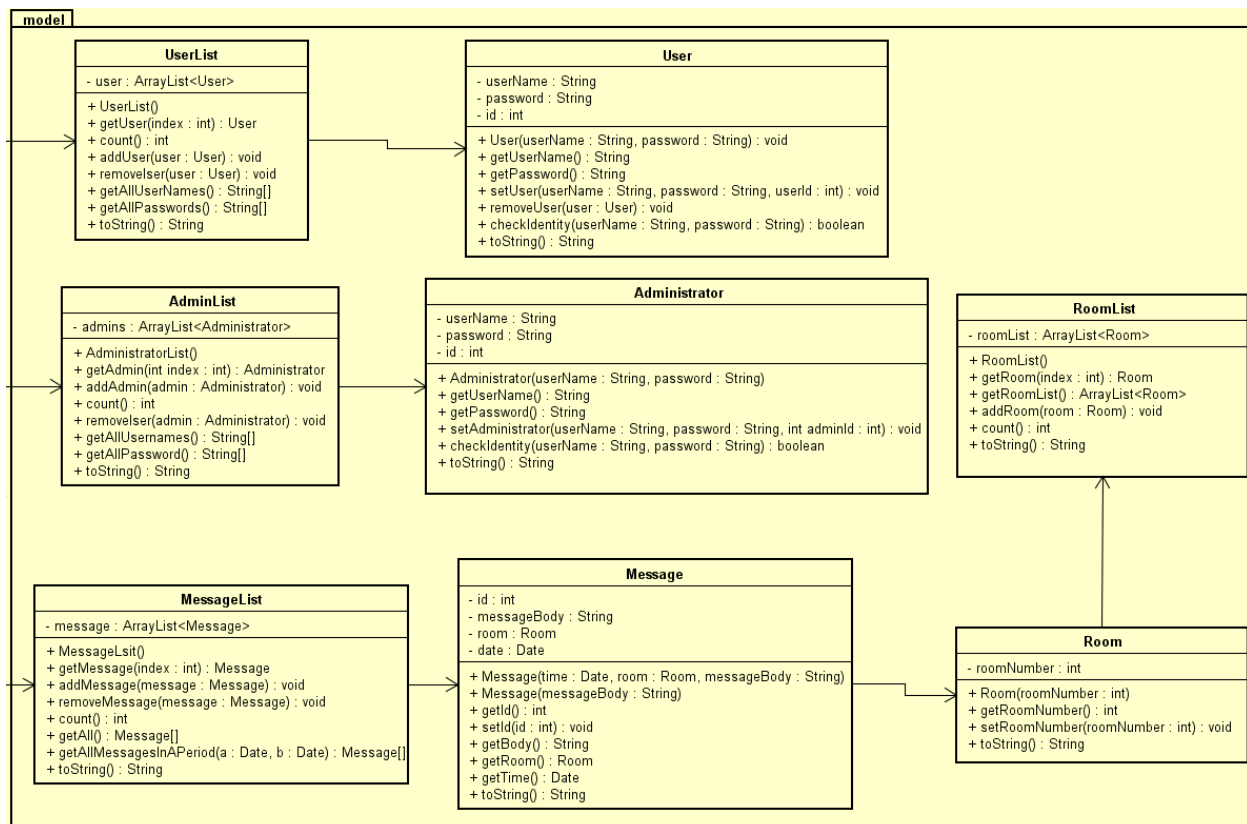


Figure 7 – Model package from MVC(See Appendix E for Full Class Diagram)

The **Message**, **Administrator**, **Room** and **User** classes are the object classes. All of the list classes are holding their corresponding objects and have methods for managing the list. For example, in **UserList** you can add additional user, remove user and etc.

DESIGN

Graphical User Interface

Our system is based on a GUI. It features a user-friendly design and is pretty straight forward.

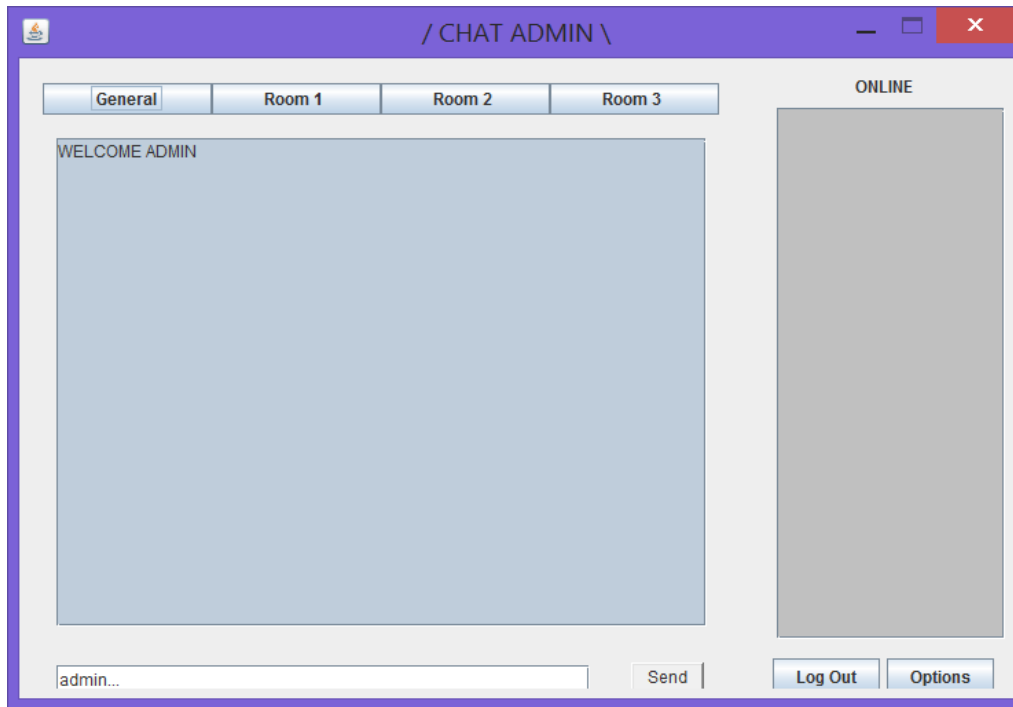


Figure 8 – Chat panel for admin(Check Appendix F-User Guide for all GUI panels and explanations)

This is how the chatting panel for administrator looks like. It has a functional text box and buttons. Each button has its own functionality, which makes it unique. For instance, Send button sends message to the other logged users or administrators. Options button's functionality is to open a new panel that contains different features for the administrator.

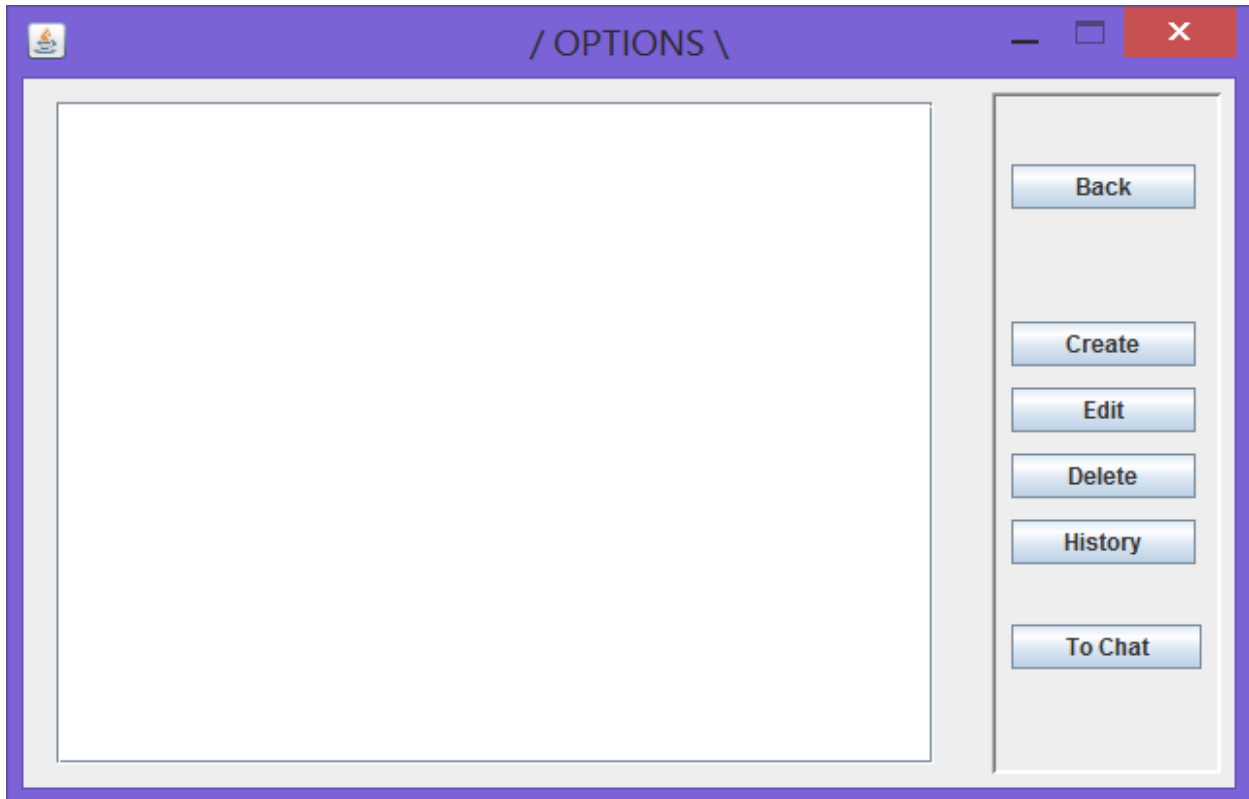


Figure 9 – Options panel for admin(Check User Guide for all GUI panels and explanations)

Implementation

An interesting part of the system is how the GUI's implementation looks. There is a Main GUI class which is the Observer and it controls which GUI to be active and updates only the active GUI.

```

public class MainGUI implements ViewObserver {

    private static Controller controller;
    private LogIn logIn;
    private ChatView chatView;
    private ChatViewAdmin chatViewAdmin;
    private Options option;

    public MainGUI() {
        this.controller = controller;
    }

    public int activeGUI() {
        if (chatView == null)
            return -1;
        if (chatView.isActive()) {
            return 1;
        } else if (chatViewAdmin.isActive())
            return 2;
        return -1;
    }

    @Override
    public void update(Observable o, Object arg) {

        if (activeGUI() == 1 && activeGUI() == 2) {
            System.out.println("both places");
            chatView.show(arg);
            chatViewAdmin.show(arg);
        }
        if (activeGUI() == 1) {
            System.out.println("userss");
            chatView.show(arg);
        }
        if (activeGUI() == 2) {
            System.out.println("adminn");
            chatViewAdmin.show(arg);
        }
    }

    public void LogInUser()
    {
        logIn.dispose();// To close the current window
        chatView.setVisible(true);// Open the new window
    }

    public void LogInAdmin()
    {
        logIn.dispose();// To close the current window
        chatViewAdmin.setVisible(true);// Open the new windows
    }

    public String getMessage()
    {
        return chatView.getMessage();
    }

    public String getMessageAdmin()
    {
        return chatViewAdmin.getMessage();
    }

    public String getUsernameFromLogIn()
    {
        return logIn.getUsername();
    }

    public String getPasswordFromLogIn()
    {
        return logIn.getPassword();
    }

    public void start(Controller controller)
    {
        logIn = new LogIn(controller);
        chatView = new ChatView(controller);
        chatViewAdmin = new ChatViewAdmin(controller);
        option = new Options(controller);
    }
}

```

Figure 10 – Part of implementation of Main GUI class

The update method of the Main GUI class is using two methods from the other GUI's in order to show the things on the panel only for the active panel. Below you can see how those methods look like:

```

public void show(Object arg1)
{
    if (arg1 == null || arg1.toString().length() < 1)
    {
        return;
    }
    String old = messlist.getText();
    if (old.length() > 1)
    {
        old += "\n";
        messlist.setText(old + arg1);
    }
}

public String getMessage()
{
    if (text == null || text.toString().length() < 1)
    {
        return null;
    }
    return text.getText().trim();
}

public boolean isActive()
{
    java.awt.Frame win[] = java.awt.Frame.getFrames();
    boolean isOpen = false;
    for (int i = 0; i < win.length; i++)
    {
        if (win[i].getName().equals(" / CHAT \\\n"))
        {
            isOpen = true;
        }
    }
    return isOpen;
}

```

Figure 11– Part of implementation of GUI classes

TEST

Before and during the coding we have made countless number of tests so that we are certain all of the requirements are done. We tried to check each and every component when the implementation was done and then when a certain segment is done we checked the entire functionality.

After the program was finished we made several test according to the requirements and use case descriptions.

The results can be found below:


CLIENT-SERVER SYSTEM	PASSED
SENDING MESSAGE	PASSED
LOGGING IN AND OUT	PASSED
ALL INFORMATION STORED IN A DATABASE	PASSED
GUI	PASSED
CREATING USER	NOT WORKING
DELETE USER	NOT IMPLEMENTED
MODIFY USER	NOT IMPLEMENTED
SWITCH CHAT ROOMS	NOT IMPLEMENTED
SHOW HISTORY OF MESSAGES	NOT IMPLEMENTED

Figure 12- Test cases for requirements

RESULT

The system consists of two applications, a client and a server.

The server part differentiates between users and administrators but only allows administrators to log in the system giving them full rights in the program. The client part differentiates



between users and administrators giving the same options for both of them. This is achieved by checking credentials before giving access.

The user is able to send messages, receive messages and log out of the system whereas the administrator has additional access to an Options panel including creating a user, deleting a user, modifying a user and showing history of messages. Messages, administrators and users are stored in a database. The server and in some cases the client establishes connection to the database whenever a request has been made. A GUI was created on both the client and server side to make it easier for the users of the program.

Creating a user has been implemented but it does not work in every situation because it does not always get the correct information entered. Modifying and deleting a user has not been implemented because a list with all users and administrators from the database is not implemented in the User Interface. Showing history of messages also has not been implemented.

DISCUSSION

After testing the system and evaluating the result, it can be concluded that the system can perform the most important features. Using Sockets makes it possible to expand the system to work over different networks. Implementing the system in a structured manner ensured that the system will be easier to maintain in the future. A user friendly GUI has been done so the user can have better experience with the program.

It is worth noting that almost all of the additional features have not been implemented or not working due to a strict time frame and failure of using Scrum. The tasks that had to be done in some of the Sprints weren't properly split. They included work over the whole system which unfortunately turned out to be the wrong decision and led to the failure of some requirements. Because of the time frame that was chosen for the Sprints, they still had to be finished even though they were not completed and a butterfly effect appeared on the result.

CONCLUSION

We started out with ideas and ended with a solution to a problem, because we know that our product owner had a problem with his small company and we did our best to make sure those problems disappear.

Working through a GUI, the possibilities for users are to log in and out of the system, send and receive messages.

We found out the hard way that having a solid documentation foundations is extremely important for our project. Now we know a lot more about SCRUM and AUP and we believe that for the next project we will have a better start and all the experience from this project can only help us achieve better results in the future.

If we had the knowledge that we have now after the project, we would have done something much, much better.

REFERENCES

AUP. (n.d.). Retrieved June 2, 2016, from <https://studienet.via.dk/Class/IT-SDJ2Y-S16/Session Material/SDJ2-S16 Philosophy Agility and Unified Process.pdf>

MVC CD. (n.d.). Retrieved June 2, 2016, from <https://studienet.via.dk/Class/IT-SDJ2Y-S16/Session Material/SDJ2-S16-12MVC.pdf>

MVC Chat system. (n.d.). Retrieved June 2, 2016, from <https://studienet.via.dk/Class/IT-SDJ2Y-S16/Session Material/SDJ2-S16-13SocketMVC.pdf>

E/R modelling. (n.d.). Retrieved June 2, 2016, from [https://studienet.via.dk/Class/IT-DBS1Y-S16/Session Material/DBS1 Lecture notes \(E-R modelling\).pdf](https://studienet.via.dk/Class/IT-DBS1Y-S16/Session Material/DBS1 Lecture notes (E-R modelling).pdf)

JDBC how to. (n.d.). Retrieved June 2, 2016, from [https://studienet.via.dk/Class/IT-DBS1Y-S16/Session Material/DBS1 Lecture notes \(JDBC\).pdf](https://studienet.via.dk/Class/IT-DBS1Y-S16/Session Material/DBS1 Lecture notes (JDBC).pdf)

Creating tables PostgreSQL. (n.d.). Retrieved June 2, 2016, from [https://studienet.via.dk/Class/IT-DBS1Y-S16/Session Material/DBS1 Lecture notes\(Creating tables\).pdf](https://studienet.via.dk/Class/IT-DBS1Y-S16/Session Material/DBS1 Lecture notes(Creating tables).pdf)

JPanel (Java Platform SE 7). (n.d.). Retrieved April 7, 2016, from <https://docs.oracle.com/javase/7/docs/api/javax/swing/JPanel.html>

Establishing a Connection (The Java™ Tutorials > JDBC(TM) Database Access > JDBC Basics). (n.d.). Retrieved April 7, 2016, from <https://docs.oracle.com/javase/tutorial/jdbc/basics/connecting.html>

Java TCP Sockets and Swing Tutorial. (n.d.). Retrieved April 7, 2016, from <http://www.cise.ufl.edu/~amyles/tutorials/tcpchat/>

java - How to generate Javadoc HTML in Eclipse? - Stack Overflow. (n.d.). Retrieved June 3, 2016, from <http://stackoverflow.com/questions/4468669/how-to-generate-javadoc-html-in-eclipse>

java - How to use MigLayout? - Stack Overflow. (n.d.). Retrieved June 3, 2016, from <http://stackoverflow.com/questions/352781/how-to-use-miglayout>


SWING JPanel Class. (n.d.). Retrieved June 3, 2016, from http://www.tutorialspoint.com/swing/swing_jpanel.htm

SWING Layouts. (n.d.). Retrieved June 3, 2016, from http://www.tutorialspoint.com/swing/swing_layouts.htm

Event Handling. (n.d.). Retrieved June 3, 2016, from http://www.tutorialspoint.com/swing/swing_event_handling.htm

SWING Event Listeners. (n.d.). Retrieved June 3, 2016, from http://www.tutorialspoint.com/swing/swing_event_listeners.htm

APPENDICES



Appendix E- Class Diagram

Appendix F- User Guide