Gruppuppgift DA343A VT19
Grupp 40

Medlemmar
Mohanad Oweidat
Sahand Poursadeghi Khiavi
Susanne Vikström
Sara Bakdach Fakhro
Bojana Filipovic
Hazem el-khalil

Innehållsförteckning

Arbetsbeskrivning

Mohanad Oweidat
Mohanad är ansvarig för ClientController och ServerMessageObject, StartClient
Mohanad är ansvarig för sekvensdiagram 2

Sahand Poursadeghi Khiavi
Sahand är ansvarig för LoggedInView och signUpView
Sahand är ansvarig för sekvensdiagram 4

Susanne Vikström
Susanne är ansvarig för User, Logg, StartServer
Susanne är ansvarig för sekvensdiagram 5
Susanne är ansvarig för klassdiagrammen.

Sara Bakdach Fakhro
Sara är ansvarig för SynchronizedHashSet, Message
Sara är ansvarig för sekvensdiagram 3

Bojana Filipovic
Bojana är ansvarig för LoggerView och MessageType
Bojana är ansvarig för sekvensdiagram 6

Hazem el-khalil
Hazem är ansvarig för ServerController samt Server och ServerView.
Hazem är ansvarig för sekvensdiagram 1

Instruktioner för programstart

Det finns en package som heter starters, man ska starta klassen StartServer för servern och StartClient för clienten.

Systembeskrivning

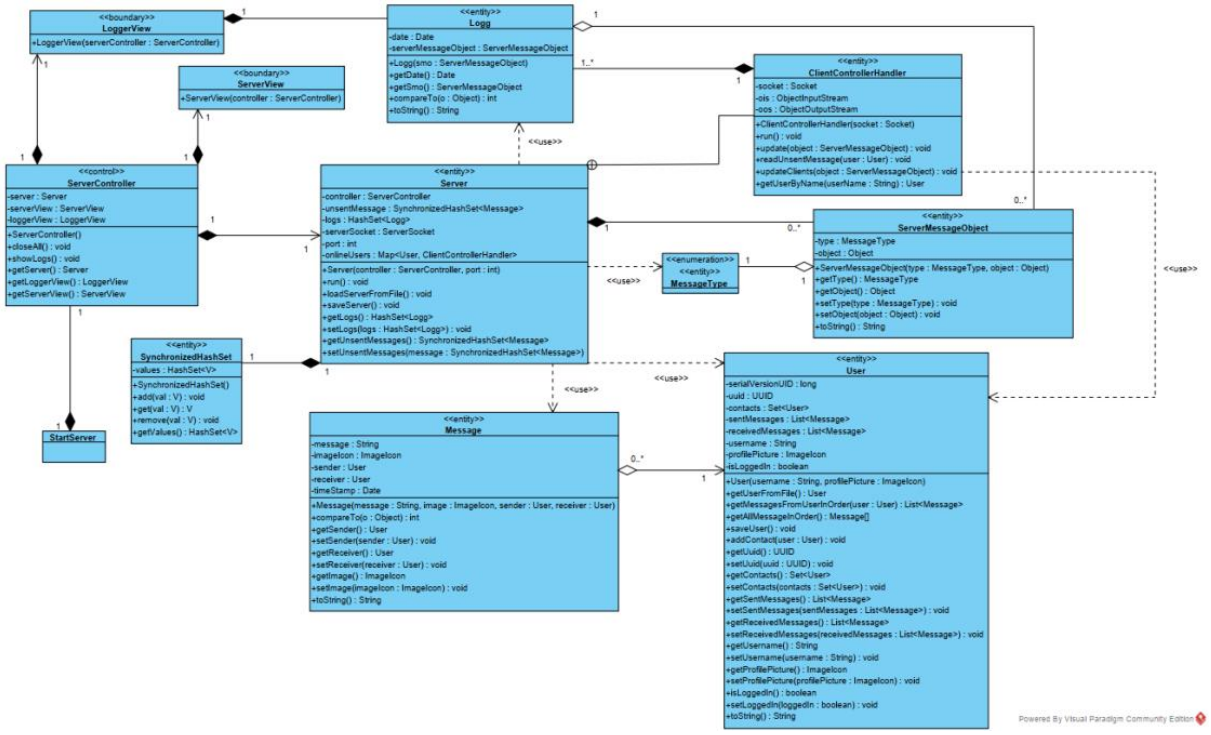
Systemet består av en flertrådatserver, vilket innebär att många klienter kan koppla upp sig mot det. Dessutom så har klienten möjligheten att välja en profilbild samt ett användarnamn. Inuti själv programmet då kan klienten skicka både en textmeddelande eller text och bild. Meddelanandera kan skickas till en specifik uppkopplad klient eller till alla. Därtill så kan klienten lägga till andra klienter i sin vänlista, där man kan skicka meddelande till de också.

De meddelandena som skickades till personer som inte är uppkopplade sparas och skickas på nytt när personen har kopplat upp sig igen.

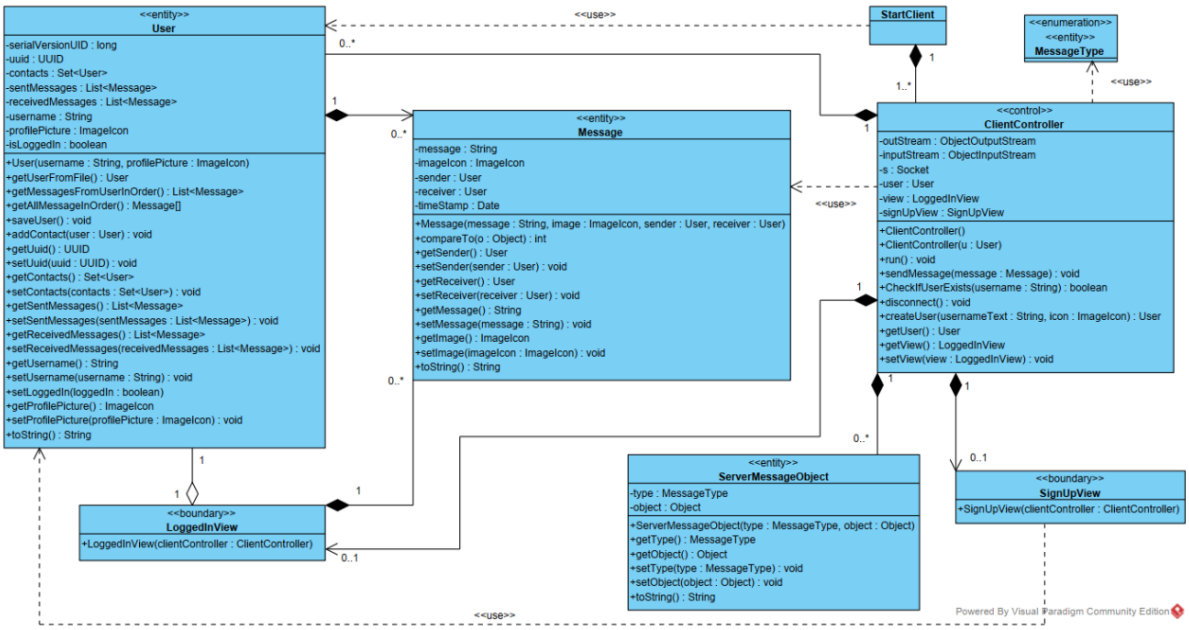
Man behöver inte logga in två gånger utan bara en gång för senare då loggas man in automatiskt.

Klassdiagram

Klassdiagram server



Klassdiagram klient

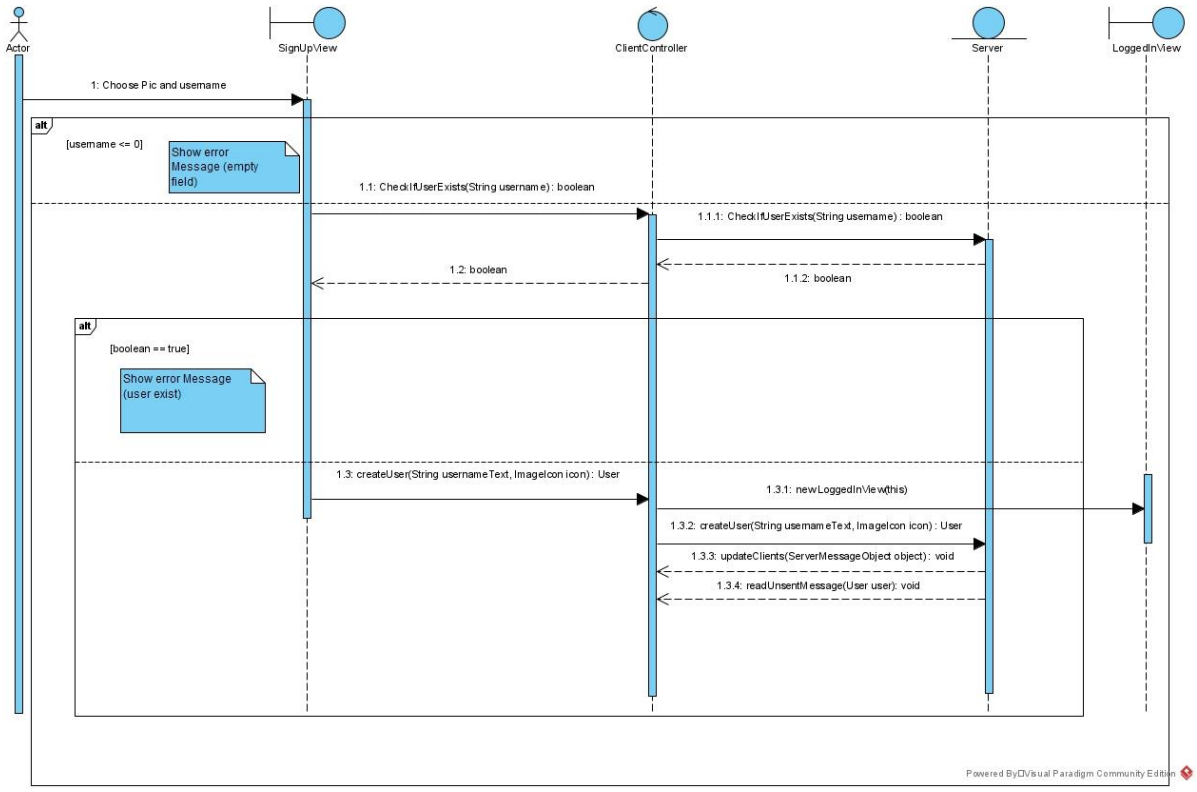


Sekvensdiagram

Sekvens 1

1. **Vad som sker på serversidan när en klient ansluter till systemet. Detta fall ska inkludera utskick av ej levererade meddelanden.**

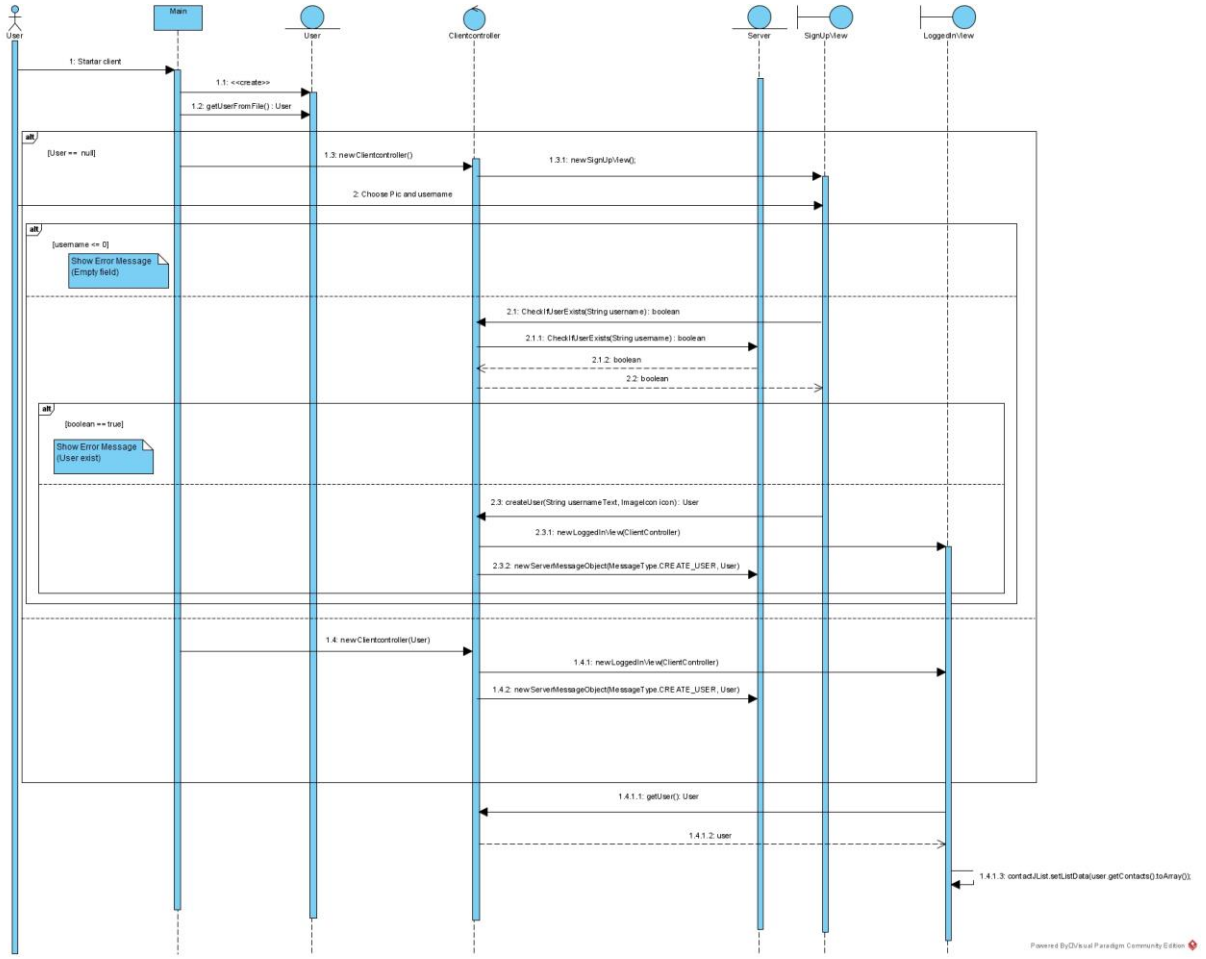
   Ansvarig: Hazem el-khalil



Sekvens 2

2. **Vad som sker på klientsidan när en användare startar klienten. Detta fall ska inkludera inläsning av Kontakter.**

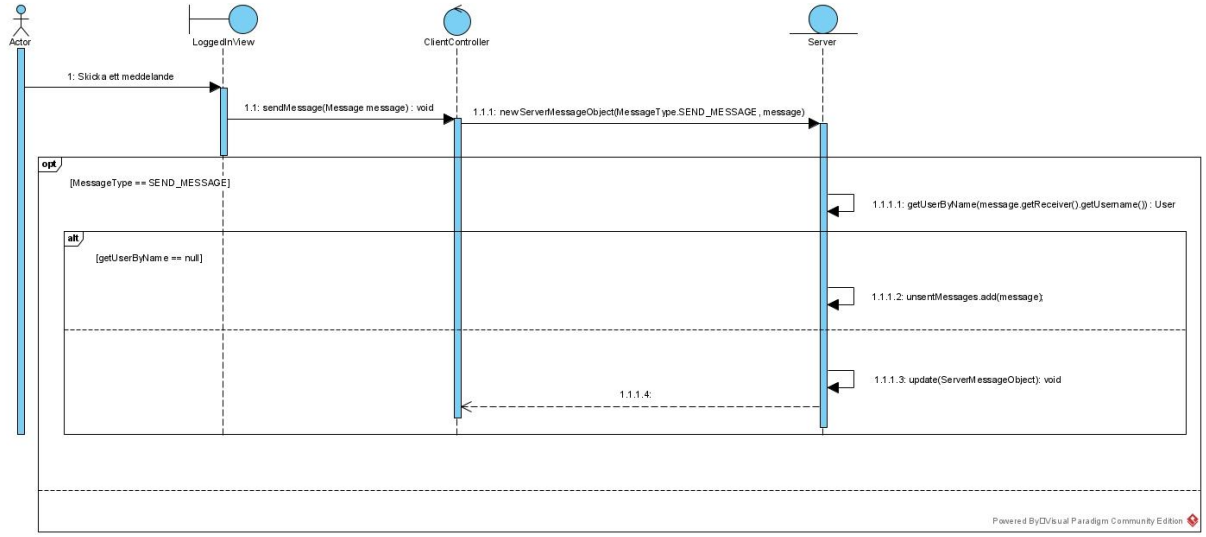
Ansvarig: Mohanad Oweidat



Sekvens 3

3. **Vad som sker på servern när en klient skickar ett meddelande. Detta fall ska inkludera utskick av meddelande till uppkopplade mottagare och lagring av meddelande till ej uppkopplade mottagare.**

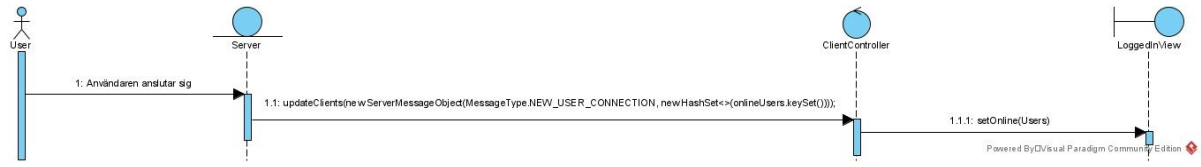
   Ansvarig: Sara Bakdach Fakhro



Sekvens 4

4. **Vad som sker i klienten då servern skickar uppdatering av anslutna användare.**

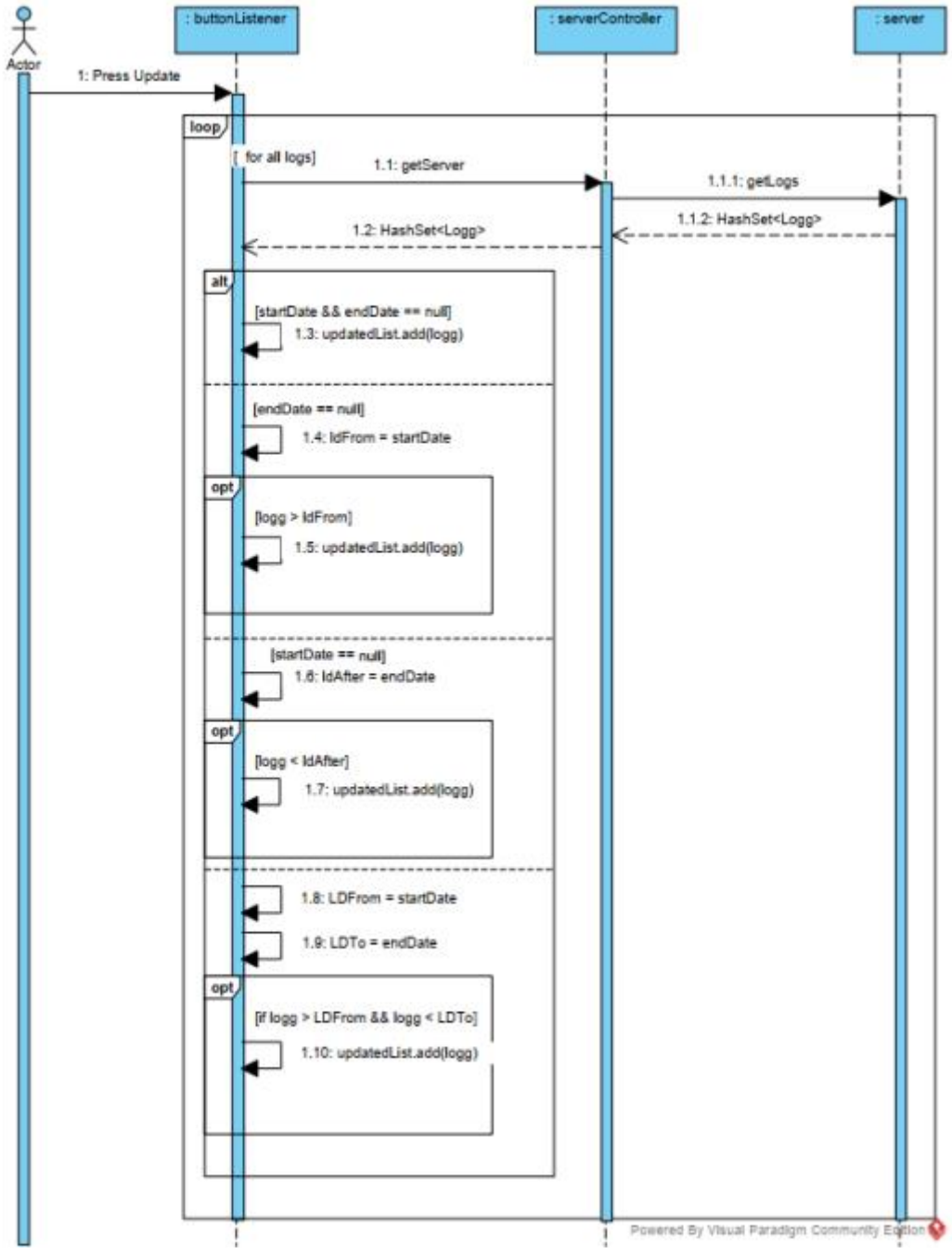
Ansvarig: Sahand Poursadeghi Khiavi



Sekvens 5

5. **Vad som sker i servern då loggning mellan två tidpunkter ska visas i server UI.**

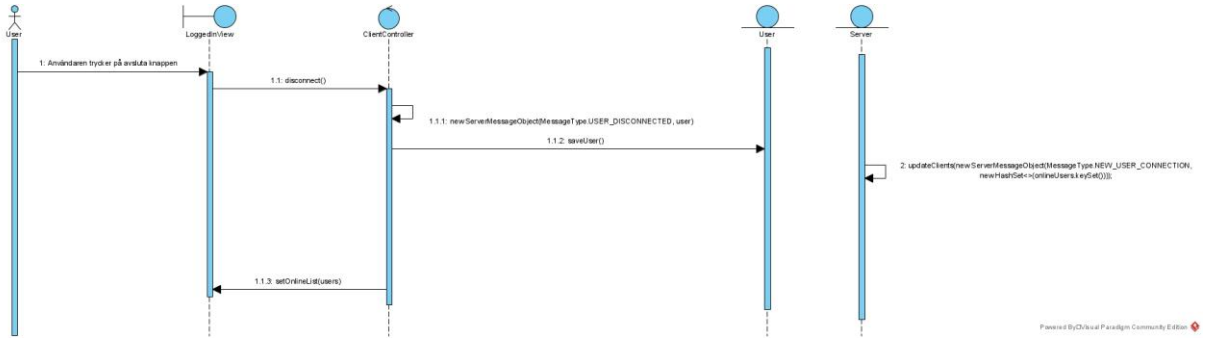
Ansvarig: Susanne Vikström



Sekvens 6

6. **Vad som sker på klientsidan när en klient avslutar sin anslutning. Detta fall ska inkludera att skriva Kontakter till hårddisken.**

Ansvarig: Bojana Filipovic



Källkod server

Logg

```
package Model.server;

import Controller.ServerMessageObject;

import java.io.Serializable;
import java.util.Date;

public class Logg implements Comparable, Serializable
{
   private Date date;
   private ServerMessageObject serverMessageObject;

   public Logg(ServerMessageObject smo)
   {
      this.serverMessageObject = smo;
      this.date = new Date(System.currentTimeMillis());
   }

   public Date getDate()
   {
      return date;
   }

   public ServerMessageObject getSmo()
   {
      return serverMessageObject;
   }

   @Override
   public int compareTo(Object o)
   {
      Logg val = (Logg) o;
      return date.compareTo(val.date);
   }
   public String toString(){
      return date.toString() + ": " + serverMessageObject.toString();
   }
}
```

Server

```
package Model.server;

import Controller.MessageType;
import Controller.ServerController;
import Controller.ServerMessageObject;
import Model.Shared.Message;
import Model.Shared.SynchronizedHashSet;
import Model.Shared.User;

import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;


public class Server extends Thread
{
   private ServerController controller;
   private SynchronizedHashSet<Message> unsentMessages;
   private HashSet<Logg> logs;
   private ServerSocket serverSocket;
   private int port;
   public static Map<User, ClientControllerHandler> onlineUsers = new
HashMap();


   public Server(ServerController controller, int port) throws IOException
   {
      this.controller = controller;
      this.port = port;
      serverSocket = new ServerSocket(port);
      logs = new HashSet<>();

      this.unsentMessages = new SynchronizedHashSet<>();
      controller.getServerView().serverMessageBoardAppend("Message Server
running..");
      controller.getServerView().serverMessageBoardAppend("Vi kör på port:"
+ port);
      controller.getServerView().serverMessageBoardAppend("----------------
--------------------");
      controller.getServerView().serverMessageBoardAppend("Waiting for the
clients...");

      loadServerFromFile();
      this.start();
   }
```

```
   @Override
   public void run()
   {
      while (!this.isInterrupted())
      {
         try
         {
            System.out.println("Looking for client!");
            Socket socket = serverSocket.accept();
            System.out.println("NEW CONNECTION FOUND");
            new ClientControllerHandler(socket);
         }
         catch (IOException e)
         {
            e.printStackTrace();
         }
      }
   }

   public void loadServerFromFile()
   {
      File dirr = new File("C:\\server_chat");
      if (!dirr.exists())
      {
         dirr.mkdirs();

      }
      else
      {
         File obj = new File(dirr, "info.txt");
         if (obj.exists())
         {
            try
            {
               FileInputStream fi = new FileInputStream(obj);
               ObjectInputStream oi = new ObjectInputStream(fi);
               HashSet<Logg> logs = (HashSet<Logg>) oi.readObject();
               SynchronizedHashSet<Message> unsentMessages =
(SynchronizedHashSet<Message>) oi.readObject();
               this.setLogs(logs);
               this.setUnsentMessages(unsentMessages);
               this.controller.getLoggerView().updateLoggList(logs);
            }
            catch (IOException | ClassNotFoundException e)
            {
               e.printStackTrace();
            }
         }
      }
   }

   public void saveServer()
   {
      File dirr = new File("C:\\server_chat");
```

```
      if (!dirr.exists())
      {
         dirr.mkdir();
      }
      try
      {
         FileOutputStream fileOut = new
FileOutputStream("C:\\server_chat\\info.txt");
         ObjectOutputStream objectOut = new ObjectOutputStream(fileOut);
         objectOut.writeObject(logs);
         objectOut.writeObject(unsentMessages);

         objectOut.close();
      }
      catch (Exception ex)
      {
         ex.printStackTrace();
      }
   }

   public HashSet<Logg> getLogs()
   {
      return this.logs;
   }
   public void setLogs(HashSet<Logg> logs)
   {
      this.logs = logs;
   }
   public SynchronizedHashSet<Message> getUnsentMessages()
   {
      return this.unsentMessages;
   }
   public void setUnsentMessages(SynchronizedHashSet<Message> messages)
   {
      this.unsentMessages = messages;
   }

   private class ClientControllerHandler extends Thread
   {
      Socket socket;
      ObjectInputStream ois;
      ObjectOutputStream oos;

      public ClientControllerHandler(Socket socket)
      {
         this.socket = socket;
         try
         {
            oos = new ObjectOutputStream(socket.getOutputStream());
            ois = new ObjectInputStream(socket.getInputStream());
         }
         catch (Exception e)
         {
         }
         this.start();
```

```
      }

      @Override
      public void run()
      {
         System.out.println("Running client socket");
         while (!this.isInterrupted())
         {
            if (!socket.isConnected())
            {
               this.interrupt();
            }
            try
            {
               if (socket.getInputStream().available() == 0)
               {
                  continue;
               }
               ServerMessageObject o = (ServerMessageObject)
ois.readObject();
               Logg logg = new Logg(o);
               logs.add(logg);
               controller.getLoggerView().updateLoggList(logs);
               switch (o.getType())
               {
                  case CREATE_USER:
                     User user = (User) o.getObject();
                     onlineUsers.put(user, this);

controller.getServerView().addUserToList(user.getUsername());

controller.getServerView().serverMessageBoardAppend("New user connected: "
+ user.getUsername());
                     updateClients(new
ServerMessageObject(MessageType.NEW_USER_CONNECTION, new
HashSet<>(onlineUsers.keySet())));
                     readUnsentMessage(user);
                     break;
                  case VERFY_NAME:
                     String uName = (String) o.getObject();
                     boolean bool = false;
                     for (User uc : onlineUsers.keySet())
                     {
                        if (uc.getUsername().equals(uName))
                        {
                           bool = true;
                        }
                     }
                     oos.writeObject(new
ServerMessageObject(MessageType.VERFY_NAME, bool));
                     oos.flush();
                     break;
                  case USER_DISCONNECTED:
                     User u = (User) o.getObject();
                     onlineUsers.remove(u);
```

```
controller.getServerView().removeUserFromList(u.getUsername());

controller.getServerView().serverMessageBoardAppend("User disconnected: " +
u.getUsername());
                    updateClients(new
ServerMessageObject(MessageType.NEW_USER_CONNECTION,
                            new HashSet<>(onlineUsers.keySet())));
                case SEND_MESSAGE:
                    Message message = (Message) o.getObject();
                    if (message.getReceiver() == null)
                    {
                        updateClients(o);
                        continue;
                    }
                    User reciver =
getUserByName(message.getReceiver().getUsername());
                    ClientControllerHandler reciverHandler =
onlineUsers.get(reciver);

                    if (reciver == null)
                    {

controller.getServerView().serverMessageBoardAppend("User: " +
message.getReceiver().getUsername() + " is not currently online saving
message");
                        unsentMessages.add(message);
                    }
                    else
                    {
                        reciverHandler.update(o);
                    }
                }
            }
            catch (java.io.IOException | java.lang.ClassNotFoundException
e)
            {
                e.printStackTrace();
            }
        }
    }

    public void update(ServerMessageObject object)
    {
        try
        {
            System.out.println(object.getType());
            oos.writeObject(object);
            oos.flush();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
```

```
    }

    public void readUnsentMessage(User user)
    {
        for (Message value : unsentMessages.getValues())
        {
            if (value.getReceiver().getUsername().equals(user.getUsername()))
            {
                onlineUsers.get(user).update(new
ServerMessageObject(MessageType.SEND_MESSAGE, value));
                unsentMessages.remove(value);
            }
        }
    }

    public void updateClients(ServerMessageObject object)
    {
        for (User user : onlineUsers.keySet())
        {
            ClientControllerHandler cch = onlineUsers.get(user);
            cch.update(object);
        }
    }

    public User getUserByName(String userName)
    {
        for (User u : onlineUsers.keySet())
        {
            if (userName.equals(u.getUsername()))
            {
                return u;
            }
        }
        return null;
    }
}
```

ServerController

```java
package Controller;

import Model.server.Server;
import View.server.LoggerView;
import View.server.ServerView;

import java.io.IOException;

public class ServerController {
    private Server server;
    private ServerView serverView;
    private LoggerView loggerView;


    public ServerController() throws IOException {
        serverView = new ServerView(this);
        loggerView = new LoggerView(this);
        server = new Server(this, 3500);
    }

    public LoggerView getLoggerView() {
        return loggerView;
    }


    public ServerView getServerView() {
        return serverView;
    }

    public void closeAll() {
        serverView.dispose();
        loggerView.dispose();
        server.interrupt();
        server.saveServer();
        server.stop();
    }

    public void showLogs() {
        loggerView.setVisible(!loggerView.isVisible());
    }

    public Server getServer() {
        return server;
    }

    public void setServer(Server server) {
        this.server = server;
    }

    public void setServerView(ServerView serverView) {
        this.serverView = serverView;
    }
```

```
    public void setLoggerView(LoggerView loggerView) {
        this.loggerView = loggerView;
    }
}
```

LoggerView

```
package View.server;

import Controller.ServerController;
import Model.server.Logg;
import com.github.lgooddatepicker.components.DateTimePicker;

import javax.swing.*;
import javax.swing.border.EmptyBorder;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.time.LocalDate;
import java.util.List;
import java.util.*;

public class LoggerView extends JFrame {

    private JPanel contentPane;
    private JTextArea textArea;
    private JButton btnUpdate;
    private JList<Logg> loggListInfo;


    private DateTimePicker dtpStart;
    private DateTimePicker dtpEnd;

    private ServerController serverController;

    public LoggerView(ServerController ServerController) {
        this.serverController = ServerController;
        init();
    }

    private void init() {
        setTitle("View logs");
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setBounds(100, 100, 950, 600);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        JScrollPane scrollPane = new JScrollPane();
        scrollPane.setBounds(10, 11, 613, 539);
        contentPane.add(scrollPane);
```

```
        loggListInfo = new JList();
        scrollPane.setViewportView(loggListInfo);

        Font f = new Font("Arial Bold", Font.BOLD, 32);

        JLabel lblStart = new JLabel("From");
        lblStart.setFont(f);
        lblStart.setBounds(750, 50, 100, 32);
        contentPane.add(lblStart);

        dtpStart = new DateTimePicker();
        dtpStart.setBounds(640, 100, 275, 32);
        contentPane.add(dtpStart);

        JLabel lblEnd = new JLabel("To");
        lblEnd.setFont(f);
        lblEnd.setBounds(750, 250, 100, 32);
        contentPane.add(lblEnd);

        dtpEnd = new DateTimePicker();
        dtpEnd.setBounds(640, 300, 275, 32);
        contentPane.add(dtpEnd);

        btnUpdate = new JButton("Update");
        btnUpdate.addActionListener(new ButtonListener());
        btnUpdate.setBounds(725, 450, 102, 23);
        contentPane.add(btnUpdate);
        this.setResizable(false);

    }

    public void updateLoggList(HashSet<Logg> log) {
        List<Logg> loggs = new ArrayList(log);
        Collections.reverse(loggs);
        Logg[] finishedVersion = new Logg[loggs.size()];
        for (int x = 0; x < loggs.size(); x++) {
            finishedVersion[x] = loggs.get(x);
        }

        loggListInfo.setListData(finishedVersion);
    }

    public void updateTextArea(String str) {
        textArea.setText(str);
    }

    private class ButtonListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            if (e.getSource() == btnUpdate) {


                ArrayList<Logg> updatedList = new ArrayList<>();

                for (Logg logg : serverController.getServer().getLogs()) {
                    if (dtpStart.datePicker.getDate() == null &&
```

```
dtpEnd.datePicker.getDate() == null) {
                        updatedList.add(logg);
                    } else if (dtpEnd.datePicker.getDate() == null) {
                        Date ldFrom =
convertToDate(dtpStart.datePicker.getDate());
                        if (logg.getDate().after(ldFrom))
                            updatedList.add(logg);
                    } else if (dtpStart.datePicker.getDate() == null) {
                        Date ldAfter =
convertToDate(dtpEnd.datePicker.getDate());
                        if (logg.getDate().before(ldAfter))
                            updatedList.add(logg);
                    } else {
                        Date LDFrom =
convertToDate(dtpStart.datePicker.getDate());
                        Date LDTo =
convertToDate(dtpEnd.datePicker.getDate());
                        if (logg.getDate().after(LDFrom) &&
logg.getDate().before(LDTo))
                            updatedList.add(logg);
                    }

                }

                Logg[] finishedVersion = new Logg[updatedList.size()];
                updatedList.toArray(finishedVersion);
                loggListInfo.setListData(finishedVersion);

            }
        }
    }

    public Date convertToDate(LocalDate dateToConvert) {
        return java.sql.Date.valueOf(dateToConvert);
    }
}
```

ServerView

```
package View.server;
import Controller.ServerController;
import javax.swing.*;

public class ServerView extends JFrame {
    private JTextArea serverMessageBoard;
    private JList allUserNameList;
    private JList activeClientList;

    private JButton logViewrButton;

    public JList getAllUserNameList() {
        return allUserNameList;
    }

    public JList getActiveClientList() {
        return activeClientList;
    }

    private DefaultListModel<String> activeUsers = new
DefaultListModel<String>();
    private DefaultListModel<String> allUsers = new
DefaultListModel<String>();


    public DefaultListModel<String> getActiveUsers() {
        return activeUsers;
    }

    public DefaultListModel<String> getAllUsers() {
        return allUsers;
    }

    private ServerController controller;

    public ServerView(ServerController controller) {
        this.controller = controller;
        initialize();
    }

    // Creates the frame components
    private void initialize() {

        setBounds(100, 100, 796, 530);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        getContentPane().setLayout(null);
        setTitle("Server View");
        setVisible(true);

        serverMessageBoard = new JTextArea();
        serverMessageBoard.setEditable(false);
        serverMessageBoard.setBounds(12, 29, 489, 435);
        getContentPane().add(serverMessageBoard);
```

```
        serverMessageBoard.setText("Starting the Server...\n");


        // allUserNameList
        allUserNameList = new JList();
        allUserNameList.setBounds(526, 324, 218, 140);
        getContentPane().add(allUserNameList);

        // activeClientList
        activeClientList = new JList();
        activeClientList.setBounds(526, 78, 218, 156);
        getContentPane().add(activeClientList);


        //All Usernames label
        JLabel all_usernames = new JLabel("All Usernames");
        all_usernames.setHorizontalAlignment(SwingConstants.LEFT);
        all_usernames.setBounds(530, 295, 127, 16);
        getContentPane().add(all_usernames);

        // Active Users label
        JLabel active_users = new JLabel("Active Users");
        active_users.setBounds(526, 53, 98, 23);
        getContentPane().add(active_users);

        //Save button
        JButton saveButton = new JButton("Stop");
        saveButton.addActionListener(e -> controller.closeAll());
        saveButton.setBounds(640, 469, 100, 20);
        getContentPane().add(saveButton);

        JButton showLogs = new JButton("logs");
        showLogs.addActionListener(a -> controller.showLogs());
        showLogs.setBounds(526, 469, 100, 20);
        getContentPane().add(showLogs);

        this.setResizable(false);
    }

    public void addUserToList(String userName) {
        controller.getServerView().getActiveUsers().addElement(userName);
        controller.getServerView().getAllUsers().addElement(userName);

controller.getServerView().getActiveClientList().setModel(controller.getSer
verView().getActiveUsers());

controller.getServerView().getAllUserNameList().setModel(controller.getServ
erView().getAllUsers());
    }

    public void removeUserFromList(String userName) {
        getActiveUsers().removeElement(userName);

controller.getServerView().getActiveClientList().setModel(controller.getSer
verView().getActiveUsers());
```

```
    }

    //To show Message in ServerMessageBoard
    public void serverMessageBoardAppend(String message) {
        serverMessageBoard.append(message + "\n");
    }
}
```

Källkod klient

ClientController

```
package Controller;

import Model.Shared.Message;
import Model.Shared.User;
import View.client.LoggedInView;
import View.client.SignUpView;

import javax.swing.*;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.HashSet;

public class ClientController extends Thread {

    private ObjectInputStream inputStream;
    private ObjectOutputStream outStream;
    private Socket s;
    private User user;
    private LoggedInView view;
    private SignUpView signUpView;

    public User getUser() {
        return user;
    }

    public ClientController() throws IOException {
        s = new Socket("217.115.51.195", 3500);
        inputStream = new ObjectInputStream(s.getInputStream());
        outStream = new ObjectOutputStream(s.getOutputStream());
        signUpView = new SignUpView(this);
    }

    public ClientController(User u) throws IOException {
        this.user = u;
        s = new Socket("217.115.51.195", 3500);
        inputStream = new ObjectInputStream(s.getInputStream());
```

```
        outStream = new ObjectOutputStream(s.getOutputStream());
        view = new LoggedInView(this);
        outStream.writeObject(new
ServerMessageObject(MessageType.CREATE_USER, user));
        outStream.flush();
        this.start();
    }


    @Override
    public void run() {
        try {
            while (true) {
                if (s.getInputStream().available() < 1) {
                    continue;
                }

                ServerMessageObject messageObject = (ServerMessageObject)
inputStream.readObject();
                switch (messageObject.getType()) {
                    case SEND_MESSAGE -> {
                        Message message = (Message)
messageObject.getObject();

System.out.println(message.getSender().getUsername() + ": " +
message.getMessage());
                        user.getReceivedMessages().add(message);

view.getClientMessageList().setListData(user.getAllMessageInOrder());
                    }
                    case NEW_USER_CONNECTION -> {
                        HashSet<User> users = (HashSet<User>)
messageObject.getObject();
                        //For second constructor --> with user
                        view.setOnlineList(users);
                        //For first constructor
                        //loggedInView().setOnlineList(users);
                    }
                }
            }
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    public void sendMessage(Message message) throws IOException {
        System.out.println(this.user.getUsername() + "> " +
message.getMessage());
        outStream.writeObject(new
ServerMessageObject(MessageType.SEND_MESSAGE, message));
        outStream.flush();
        user.getSentMessages().add(message);

view.getClientMessageList().setListData(user.getAllMessageInOrder());
```

```
    }

    //TODO: Kanske flytta alla dessa metoder till  Connection
    // Check if the user is already connected to the server
    public boolean CheckIfUserExists(String username) throws IOException,
ClassNotFoundException {
        outStream.writeObject(new
ServerMessageObject(MessageType.VERFY_NAME, username));
        outStream.flush();

        while (s.getInputStream().available() == 0) {

        }
        ServerMessageObject serverMessageObject = (ServerMessageObject)
inputStream.readObject();
        return (boolean) serverMessageObject.getObject();
    }

    public void disconnect() throws IOException {
        outStream.writeObject(new
ServerMessageObject(MessageType.USER_DISCONNECTED, user));
        outStream.flush();

        user.saveUser();
    }


    // It will create a user and send it to the server
    public User createUser(String usernameText, ImageIcon icon) {
        user = new User(usernameText, icon);
        this.setView(new LoggedInView(this));
        try {
            outStream.writeObject(new
ServerMessageObject(MessageType.CREATE_USER, user));
            outStream.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
        this.start();
        return user;
    }


    public LoggedInView getView() {
        return view;
    }

    public void setView(LoggedInView view) {
        this.view = view;
    }
}
```

LoggedInView

```
package View.client;

import Controller.ClientController;
import Model.Shared.Message;
import Model.Shared.User;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.io.File;
import java.io.IOException;
import java.util.Set;

public class LoggedInView extends JFrame {
    private JTextField clientTypingBoard;
    private JList<Message> clientMessageList;

    public JList<Message> getClientMessageList() {
        return clientMessageList;
    }

    private JList<User> clientActiveUsersList;

    private JList contactJList;


    private JRadioButton oneToNBtn;
    private JRadioButton broadcastBtn;
    private JRadioButton contactListBtn;
    private JButton uploadBtn;
    private JButton clientEndSessionButton;

    private JLabel user_icon;
    private JLabel user_id;


    private DefaultListModel<User> listModel;
    private DefaultListModel<User> contactListModel;
    private JFileChooser fileChooser;

    private ClientController clientController;

    private User user;
    private ImageIcon MessageImage;


    public LoggedInView(ClientController clientController) {
        this.clientController = clientController;
        user = clientController.getUser();
        init();
```

```
    }

    //TODO: Just for debug --> delete later!
    public static void main(String[] args) {
        new LoggedInView(null);
    }


    public void init() {
        //Frame
        //setTitle("Client Frame-" + user.getUsername());
        setTitle("Messenger - Logged in as: " + user.getUsername());
        setBounds(100, 100, 926, 680);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        getContentPane().setLayout(null);
        this.setResizable(false);


        //Components

        //clientMessageList
        clientMessageList = new JList<>();
        clientMessageList.setCellRenderer(new MessageBoxRenderer());
        clientMessageList.setBounds(12, 63, 530, 457);
        getContentPane().add(clientMessageList);


        //user_icon
        user_icon = new JLabel();
        ImageIcon icon;
        icon = user.getProfilePicture();
        Image image = icon.getImage(); // transform it
        Image newimg = image.getScaledInstance(50, 50, Image.SCALE_SMOOTH);
// scale it the smooth way
        icon = new ImageIcon(newimg);  // transform it back
        user_icon.setIcon(icon);
        user_icon.setHorizontalAlignment(SwingConstants.LEFT);
        user_icon.setBounds(12, 5, 50, 50);
        getContentPane().add(user_icon);

        //user_id
        user_id = new JLabel(user.getUsername());
        user_id.setHorizontalAlignment(SwingConstants.LEADING);
        user_id.setFont(new Font("Verdana", Font.BOLD, 10));
        user_id.setBounds(90, 12, 70, 50);
        getContentPane().add(user_id);

        //clientTypingBoard
        clientTypingBoard = new JTextField();
        clientTypingBoard.setHorizontalAlignment(SwingConstants.LEFT);
        clientTypingBoard.setBounds(12, 533, 470, 70);
        getContentPane().add(clientTypingBoard);
        clientTypingBoard.setColumns(10);
```

```
        //Send button
        JButton clientSendMsgBtn = new JButton("Skicka");
        clientSendMsgBtn.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String textAreaMessage = clientTypingBoard.getText();
                ImageIcon selectedImg = MessageImage;

                if (selectedImg != null ||
!textAreaMessage.equalsIgnoreCase("")) {
                    if (clientActiveUsersList.getSelectedValue() != null &&
oneToNBtn.isSelected()) {
                        sendMessageLogic(textAreaMessage, selectedImg);
                    } else if (broadcastBtn.isSelected()) {
                        sendMessageLogic(textAreaMessage, selectedImg);
                    } else if (contactJList.getSelectedValue() != null &&
contactListBtn.isSelected()) {
                        sendMessageLogic(textAreaMessage, selectedImg);
                    } else {
                        JOptionPane.showMessageDialog(null, "Please choose
one receiver from the list!", "OBS!",
                                JOptionPane.ERROR_MESSAGE);
                    }

                } else {
                    JOptionPane.showMessageDialog(null, "Please write a
message or upload a picture!", "OBS!",
                            JOptionPane.ERROR_MESSAGE);
                }
            }
        });
        clientSendMsgBtn.setBounds(625, 533, 115, 70);
        getContentPane().add(clientSendMsgBtn);


        //clientActiveUsersList
        clientActiveUsersList = new JList<>();
        clientActiveUsersList.setCellRenderer(new UserBoxRenderer());
        listModel = new DefaultListModel<>();
        clientActiveUsersList.setModel(listModel);
        clientActiveUsersList.setToolTipText("Online användare:");
        clientActiveUsersList.setBounds(554, 63, 350, 460);
        getContentPane().add(clientActiveUsersList);
        clientActiveUsersList.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                if (e.getClickCount() > 1) {
                    User selectedUser =
clientActiveUsersList.getSelectedValue();
                    user.getContacts().add(selectedUser);
                    contactJList.setListData(user.getContacts().toArray());
                }
            }
        });
```

```
        //contactJList
        contactJList = new JList();
        contactListModel = new DefaultListModel<>();
        contactJList.setModel(contactListModel);
        contactJList.setToolTipText("Kontakter:");
        contactJList.setBounds(554, 63, 350, 460);
        contactJList.setCellRenderer(new UserBoxRenderer());
        contactJList.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                if (e.getClickCount() > 1) {
                    User selectedUser = (User)
contactJList.getSelectedValue();
                    user.getContacts().remove(selectedUser);
                    contactJList.setListData(user.getContacts().toArray());
                }
            }
        });
        getContentPane().add(contactJList);


        //Exit button
        clientEndSessionButton = new JButton("Avsluta");
        clientEndSessionButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    clientController.disconnect();
                } catch (IOException exception) {
                    exception.printStackTrace();
                }
                dispose();
            }
        });
        clientEndSessionButton.setBounds(755, 533, 115, 70);
        getContentPane().add(clientEndSessionButton);

        //Upload pic button
        uploadBtn = new JButton("Bild");
        uploadBtn.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                fileChooser = new JFileChooser();
                fileChooser.setAcceptAllFileFilterUsed(false);
                int option = fileChooser.showOpenDialog(null);
                if (option == JFileChooser.APPROVE_OPTION) {
                    File pic = fileChooser.getSelectedFile();
                    MessageImage = new ImageIcon(pic.getPath());
                }
            }
        });
        uploadBtn.setBounds(490, 533, 120, 70);
        getContentPane().add(uploadBtn);
        JLabel activeUserLabel = new JLabel("Online:");
        activeUserLabel.setHorizontalAlignment(SwingConstants.LEFT);
```

```
        activeUserLabel.setBounds(559, 43, 95, 16);
        getContentPane().add(activeUserLabel);

        //oneToNBtn button
        oneToNBtn = new JRadioButton("PM");
        oneToNBtn.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                clientActiveUsersList.setVisible(true);
                clientActiveUsersList.setEnabled(true);
            }
        });
        oneToNBtn.setSelected(true);
        oneToNBtn.setBounds(600, 24, 100, 20);
        getContentPane().add(oneToNBtn);


        //broadcastBtn
        broadcastBtn = new JRadioButton("Broadcast");
        broadcastBtn.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                clearSelectedUser();
                clientActiveUsersList.setEnabled(false);

            }
        });
        broadcastBtn.setBounds(700, 24, 100, 20);
        getContentPane().add(broadcastBtn);


        //contactListBtn
        contactListBtn = new JRadioButton("Kontakter");
        contactListBtn.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                clientActiveUsersList.setVisible(false);
                updateContacts();
                contactJList.setVisible(true);
            }
        });
        contactListBtn.setBounds(800, 24, 100, 20);
        getContentPane().add(contactListBtn);


        ButtonGroup btngrp = new ButtonGroup();
        btngrp.add(oneToNBtn);
        btngrp.add(broadcastBtn);
        btngrp.add(contactListBtn);
        setVisible(true);

    }

    private void updateContacts() {
        this.contactJList.setListData(user.getContacts().toArray());
```

```
    }


    public void clearSelectedUser() {
        clientActiveUsersList.clearSelection();
    }

    public void addUserToOnlineList(User user) {
        listModel.addElement(user);
        clientActiveUsersList.setModel(listModel);
    }

    public void setOnlineList(Set<User> users) {
        listModel.clear();
        users.forEach(u -> listModel.addElement(u));
        clientActiveUsersList.setModel(listModel);

    }

    public void sendMessageLogic(String textAreaMessage, ImageIcon
selectedImg) {
        User to;
        if (clientActiveUsersList.getSelectedValue() == null) {
            to = (User) contactJList.getSelectedValue();
        } else {
            to = clientActiveUsersList.getSelectedValue();
        }

        Message message = new Message(textAreaMessage, selectedImg, user,
to);
        try {
            clientController.sendMessage(message);
        } catch (IOException exception) {
            exception.printStackTrace();
        }
        MessageImage = null;
        clientTypingBoard.setText("");
    }


    class MessageBoxRenderer extends DefaultListCellRenderer {

        Font font = new Font("helvitica", Font.BOLD, 12);

        @Override
        public Component getListCellRendererComponent(JList list, Object
value, int index, boolean isSelected,
                                                      boolean cellHasFocus)
{
            Message message = (Message) value;
            JLabel label = (JLabel)
super.getListCellRendererComponent(list, message, index, isSelected,
cellHasFocus);
            if (message.getImage() != null) {
                Image image = message.getImage().getImage(); // transform
```

```
it
                Image newimg = image.getScaledInstance(100, 100,
java.awt.Image.SCALE_SMOOTH); // scale it the
                // smooth way
                ImageIcon icon = new ImageIcon(newimg);
                label.setIcon(icon);
            }
            label.setHorizontalTextPosition(JLabel.LEFT);
            label.setFont(font);
            return label;
        }
    }

    class UserBoxRenderer extends DefaultListCellRenderer {

        Font font = new Font("helvitica", Font.BOLD, 12);

        @Override
        public Component getListCellRendererComponent(JList list, Object
value, int index, boolean isSelected,
                                                      boolean cellHasFocus)
{
            User user = (User) value;
            JLabel label = (JLabel)
super.getListCellRendererComponent(list, user.getUsername(), index,
isSelected,
                    cellHasFocus);
            Image image = user.getProfilePicture().getImage(); // transform
it
            Image newimg = image.getScaledInstance(50, 50,
java.awt.Image.SCALE_SMOOTH); // scale it the
            // smooth way
            ImageIcon icon = new ImageIcon(newimg);
            label.setIcon(icon);
            label.setHorizontalTextPosition(JLabel.RIGHT);
            label.setFont(font);
            return label;
        }
    }

}
```

SignUpView

```
package View.client;

import Controller.ClientController;
import Model.Shared.User;

import javax.swing.*;
import javax.swing.border.EmptyBorder;
import javax.swing.border.LineBorder;
import javax.swing.filechooser.FileNameExtensionFilter;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.File;

public class SignUpView extends JFrame
{
   private ClientController clientController;

   private JPanel avatarPane;

   private JPanel contentPane;
   private JTextField txtUsername;
   private JButton btnSignUp = new JButton("Connect");
   private JButton btnChooseImage = new JButton("Choose image");

   private ImageIcon icon;
   private String path;


   public SignUpView(ClientController clientController)
   {
      this.clientController = clientController;
      setResizable(false);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setBounds(100, 100, 700, 500);
      contentPane = new JPanel();
      contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
      setContentPane(contentPane);
      contentPane.setLayout(null);
      this.setResizable(false);
      txtUsername = new JTextField();
      txtUsername.setToolTipText("Your username");
      txtUsername.setFont(new Font("Tahoma", Font.PLAIN, 20));
      txtUsername.setBounds(236, 235, 200, 30);
      contentPane.add(txtUsername);
      txtUsername.setColumns(10);
      txtUsername.setEditable(false);
```

```
      this.addWindowListener(new WindowAdapter()
      {
         public void windowOpened(WindowEvent e)
         {
            txtUsername.requestFocus();
         }
      });

      JLabel lblUsername = new JLabel("Username");
      lblUsername.setBounds(300, 210, 75, 15);
      contentPane.add(lblUsername);

      btnSignUp.setBounds(289, 385, 89, 23);
      btnSignUp.addActionListener(new ButtonListener());
      contentPane.add(btnSignUp);


      avatarPane = new JPanel();
      avatarPane.setBorder(new LineBorder(new Color(0, 0, 0)));
      avatarPane.setBounds(310, 50, 50, 50);
      contentPane.add(avatarPane);

      btnChooseImage.setBounds(275, 135, 125, 23);
      btnChooseImage.addActionListener(new ButtonListener());
      contentPane.add(btnChooseImage);
      setVisible(true);

   }

   private class ButtonListener implements ActionListener
   {
      public void actionPerformed(ActionEvent e)
      {

         if (e.getSource() == btnChooseImage)
         {
            JFileChooser JFC = new JFileChooser();
            FileNameExtensionFilter filter = new
FileNameExtensionFilter("JPG & PNG Images", "jpg", "png");
            JFC.setFileFilter(filter);
            int optionpicked = JFC.showOpenDialog(null);
            if (optionpicked == JFileChooser.APPROVE_OPTION)
            {
               File file = JFC.getSelectedFile();
               icon = new ImageIcon(file.getPath());
               Image image = icon.getImage(); // transform it
               Image newimg = image.getScaledInstance(50, 50,
java.awt.Image.SCALE_SMOOTH); // scale it the
               // smooth way
               icon = new ImageIcon(newimg);  // transform it back

               avatarPane.removeAll();
               avatarPane.add(new JLabel(icon));
```

```
                avatarPane.updateUI();
                path = file.getAbsolutePath();
                if (path != null)
                {
                   txtUsername.setEditable(true);
                }
             }
          }
          if (e.getSource() == btnSignUp)
          {
             if (txtUsername.getText().length() <= 0)
             {
                JOptionPane.showMessageDialog(null, "Please choose a picture
and then enter a username", "ERROR",
                      JOptionPane.ERROR_MESSAGE);
             }
             else
             {
                String usernameText = txtUsername.getText();
                try
                {

                   //Check if the entered username is not Connected to the
server
                   boolean exists =
clientController.CheckIfUserExists(usernameText);
                   //Username already exists (Connected)
                   if (exists)
                   {
                      JOptionPane.showMessageDialog(null, "Username already
Connected, please login with " +
                            "another username!", "ERROR",
JOptionPane.ERROR_MESSAGE);
                      txtUsername.setText("");
                   }

                   //Username not Connected to the server ---> Done login
                   else
                   {
                      User user = clientController.createUser(usernameText,
icon);
                      //loggedInView = new LoggedInView(clientController);
                      System.out.println("User:" + usernameText + "are now
logged!!");
                      user.setLoggedIn(true);
                      setVisible(false);
                   }
                }
                catch (Exception ex)
                {
                   ex.printStackTrace();
                }
             }
          }
```

```
        }
    }
}
```

Källkod för gemensamma klasser i klient och server

MessageType

```
package Controller;

import java.io.Serializable;

public enum MessageType implements Serializable
{

    VERFY_NAME, CREATE_USER, SEND_MESSAGE, NEW_USER_CONNECTION,
USER_DISCONNECTED;

}
```

ServerMessageObject

```
package Controller;

import java.io.Serializable;

public class ServerMessageObject implements Serializable
{
    private MessageType type;
    private Object object;

    public ServerMessageObject(MessageType type, Object object)
    {
        this.type = type;
        this.object = object;
    }


    public MessageType getType()
    {
        return type;
    }

    public Object getObject()
    {
        return object;
    }
```

```
    public void setType(MessageType type)
    {
        this.type = type;
    }

    public void setObject(Object object)
    {
        this.object = object;
    }

    @Override
    public String toString()
    {
        return object.toString() + " - " + type.name();
    }
}
```

Message

```
package Model.Shared;

import javax.swing.*;
import java.io.Serializable;
import java.util.Date;

public class Message implements Serializable, Comparable {
    private String message;
    private ImageIcon imageIcon;
    private User sender;
    private User receiver;
    private Date timeStamp;

    public Message(String message, ImageIcon image, User sender, User
receiver) {
        this.message = message;
        this.imageIcon = image;
        this.sender = sender;
        this.receiver = receiver;
        this.timeStamp = new Date(System.currentTimeMillis());
    }

    @Override
    public int compareTo(Object o) {
        return this.timeStamp.compareTo(((Message) o).timeStamp);
    }

    public User getSender() {
        return sender;
    }

    public void setSender(User sender) {
        this.sender = sender;
    }
```

```
    public User getReceiver() {
        return receiver;
    }

    public void setReceiver(User receiver) {
        this.receiver = receiver;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    public ImageIcon getImage() {
        return imageIcon;
    }

    public void setImage(ImageIcon image) {
        this.imageIcon = image;
    }

    @Override
    public String toString() {

        String messageText;
        if (receiver != null)
            messageText = sender.getUsername() + " > " + message;
        else
            messageText = sender.getUsername() + " > " + message + " (Sent
to Everyone)";

        return messageText;
    }
}
```

SynchronizedHashSet

```java
package Model.Shared;

import java.io.Serializable;
import java.util.HashSet;

public class SynchronizedHashSet<V> implements Serializable
{
   private HashSet<V> values;

   public SynchronizedHashSet()
   {
      this.values = new HashSet<>();
   }

   public synchronized void add(V val)
   {
      values.add(val);
   }

   public synchronized V get(V val)
   {
      for (V v : values)
      {
         if (v == val)
         {
            return v;
         }
      }
      return null;
   }

   public synchronized void remove(V val)
   {
      values.remove(val);
   }

   public HashSet<V> getValues()
   {
      return values;
   }
}
```

User

```java
package Model.Shared;

import javax.swing.*;
import java.io.*;
import java.util.*;

public class User extends Thread implements Serializable {
    @Serial
    private static final long serialVersionUID = 3178179156010420956L;
    private UUID uuid;
    private Set<User> contacts;
    private List<Message> sentMessages;
    private List<Message> receivedMessages;
    private String username;
    //private Icon profilePicture;
    private ImageIcon profilePicture;
    private boolean isLoggedIn;


    public User(String username, ImageIcon profilePicture) {
        this.receivedMessages = new ArrayList<>();
        this.sentMessages = new ArrayList<>();
        this.contacts = new HashSet<>();
        uuid = UUID.randomUUID();
        this.username = username;
        this.profilePicture = profilePicture;
        isLoggedIn = false;
    }

    public static User getUserFromFile() {
        File dirr = new File("C:\\client_chat");

        if (!dirr.exists()) {
            return null;
        } else {
            File obj = new File(dirr, "info.txt");
            if (obj.exists()) {
                try {
                    FileInputStream fi = new FileInputStream(obj);
                    ObjectInputStream oi = new ObjectInputStream(fi);
                    User user = (User) oi.readObject();
                    System.out.println(user.getContacts().size());
                    return user;
                } catch (IOException | ClassNotFoundException e) {
                    e.printStackTrace();
                }
            } else {
                return null;
            }

        }
        return null;
    }
```

```
    public List<Message> getMessagesFromUserInOrder(User user) {
        List<Message> messages = new ArrayList<>();
        for (Message message : getReceivedMessages()) {
            if (message.getSender() == user) {
                messages.add(message);
            }
        }
        for (Message message : getSentMessages()) {
            if (message.getReceiver() == user) {
                messages.add(message);
            }
        }

        Collections.sort(messages);

        return messages;
    }

    public Message[] getAllMessageInOrder() {
        List<Message> allMessages = new ArrayList<>();
        allMessages.addAll(sentMessages);
        allMessages.addAll(receivedMessages);
        Collections.reverse(allMessages);
        Message[] array = new Message[allMessages.size()];

        allMessages.toArray(array);

        return array;
    }


    public void saveUser() {
        File dirr = new File("C:\\client_chat");

        if (!dirr.exists()) {
            dirr.mkdir();
        }
        try {
            FileOutputStream fileOut = new
FileOutputStream("C:\\client_chat\\info.txt");
            ObjectOutputStream objectOut = new ObjectOutputStream(fileOut);
            System.out.println(this.getContacts().size());
            objectOut.writeObject(this);
            objectOut.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    public void addContact(User user) {
        contacts.add(user);
    }
```

```java
    public UUID getUuid() {
        return uuid;
    }

    public void setUuid(UUID uuid) {
        this.uuid = uuid;
    }

    public Set<User> getContacts() {
        return contacts;
    }

    public void setContacts(Set<User> contacts) {
        this.contacts = contacts;
    }

    public List<Message> getSentMessages() {
        return sentMessages;
    }

    public void setSentMessages(List<Message> sentMessages) {
        this.sentMessages = sentMessages;
    }

    public List<Message> getReceivedMessages() {
        return receivedMessages;
    }

    public void setReceivedMessages(List<Message> receivedMessages) {
        this.receivedMessages = receivedMessages;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public ImageIcon getProfilePicture() {
        return profilePicture;
    }

    public void setProfilePicture(ImageIcon profilePicture) {
        this.profilePicture = profilePicture;
    }

    public boolean isLoggedIn() {
        return isLoggedIn;
    }

    public void setLoggedIn(boolean loggedIn) {
        isLoggedIn = loggedIn;
    }
```

```
    @Override
    public String toString() {
        return this.username;
    }
}
```