Kerschel James

814004296

COMP 6601

Chat Messenger Assignment 1

My program executed very well but gives a warning when I send messages from one user to the other. It does not affect the functionality of the chat messenger. I think I completed 99% of the requirements. The 1% was that I was not able to have the chat bubbles alternate between left and right.

To run the application run ChatServer and ChatClient

I created my chat application with JavaFX and JavaFXML for the UI components.

```
public static void main(String[] args){
    launch(args);
}

@Override
public void start(Stage primaryStage) throws Exception {
    window = primaryStage;
    window.setTitle(Name);

    Parent root = FXMLLoader.load(getClass().getResource( name: "chat.fxml"));

    Scene chatpage = new Scene(root, width: 650, height: 500);
    Scene loginpage = new Scene(setGridLogin(chatpage), width: 300, height: 150);


    window.setScene(loginpage);
    window.show();
```

This code snippet show how the UI component of the FXML was created and initialized. Start is called in the main as launch(args); then the UI is created.

UI Design code

```
public GridPane setGridLogin(Scene chatpage) {
    GridPane grid = new GridPane();
    grid.setPadding(new Insets( topRightBottomLeft: 10));
    grid.setVgap(8);
    grid.setHgap(10);

    Label nameLabel = new Label( text: "Username");
    GridPane.setConstraints(nameLabel, columnIndex: 0, rowIndex: 0);
    TextField nameInput = new TextField();
    nameInput.setPromptText("Name");
    GridPane.setConstraints(nameInput, columnIndex: 1, rowIndex: 0);


    try {
        IPAddress = InetAddress.getLocalHost().getHostAddress();
    } catch (UnknownHostException e) {
        e.printStackTrace();
    }
      Server IP
```

Public GridPane setGridLogin(chatpage)

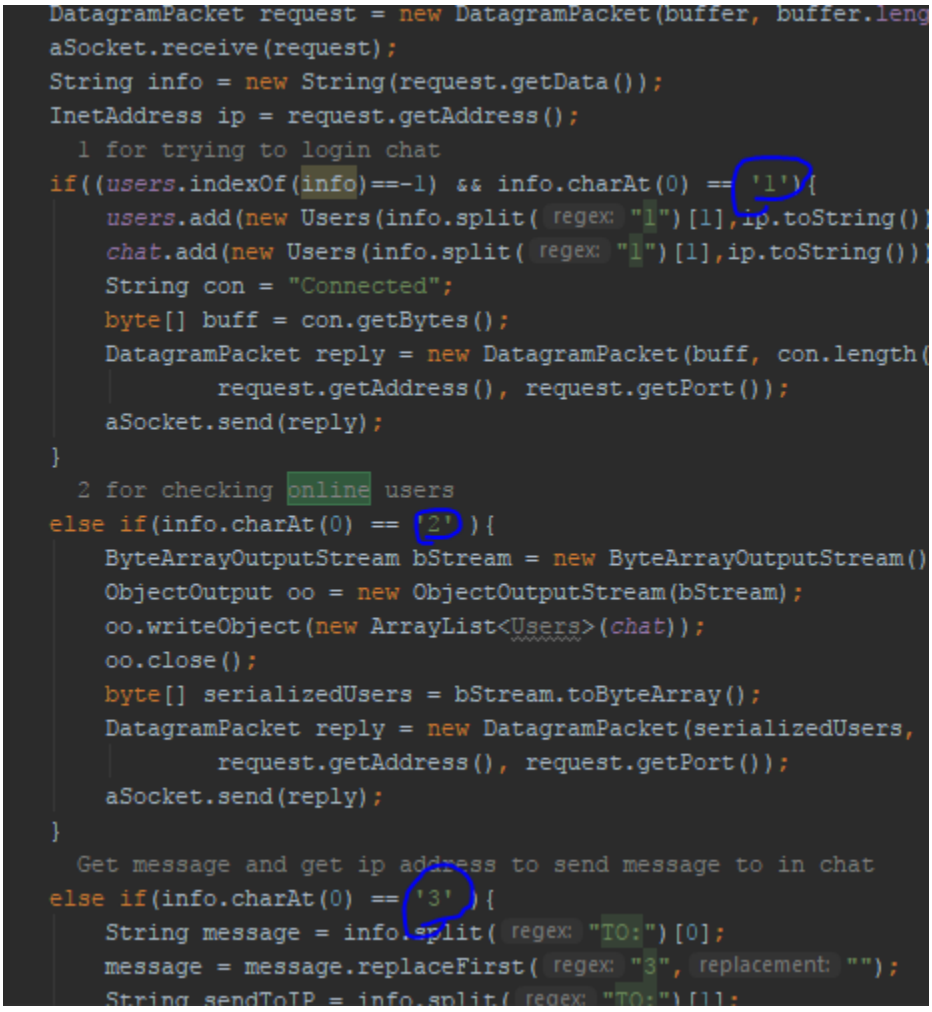
Although I used FXML as a UI component before I learnt about it I started to design the UI components manually through code. Therefore, there are two different styles of designing UI, which I used XML and code based.

Server (ChatServer.java)

The server uses an infinite loop thread and is able to accept multiple different messages from the client. The server listens on port 6578

1. The input of “1” means that the client is trying to connect to the server
2. The input of “2” means that the client is asking for a list of online users
3. The input of “3” means that the client is trying to chat with another user
4. The input of “4’ means that the client is asking the server to block chats from and to a specific IP address

All of these options are outlined in the diagram below



The server has a basic UI which just saves the user’s name and IP address in a table.

Client (ChatClient.java)

After the client logs into the application a thread is started to allow them to receive messages. The listening port for the client is port 6500.

Receiving Messages

```
@Override
public void run() {
    DatagramSocket aSocket = null;
    byte[] buffer;
    try{
        aSocket = new DatagramSocket( port: 6500);
        // create socket at agreed port

        while(true){
            buffer = new byte[1000];
            DatagramPacket request = new DatagramPacket(buffer,
            aSocket.receive(request);
            state=1;
            String message = new String(request.getData());
            System.out.println("Receiving");
            try {
                chat.add(message);
            }
            catch (Exception e){
                System.out.println("THe mesg " + e);
            }
```

This is the thread for the client. The chat.add(message); adds the message to the UI when it is received.

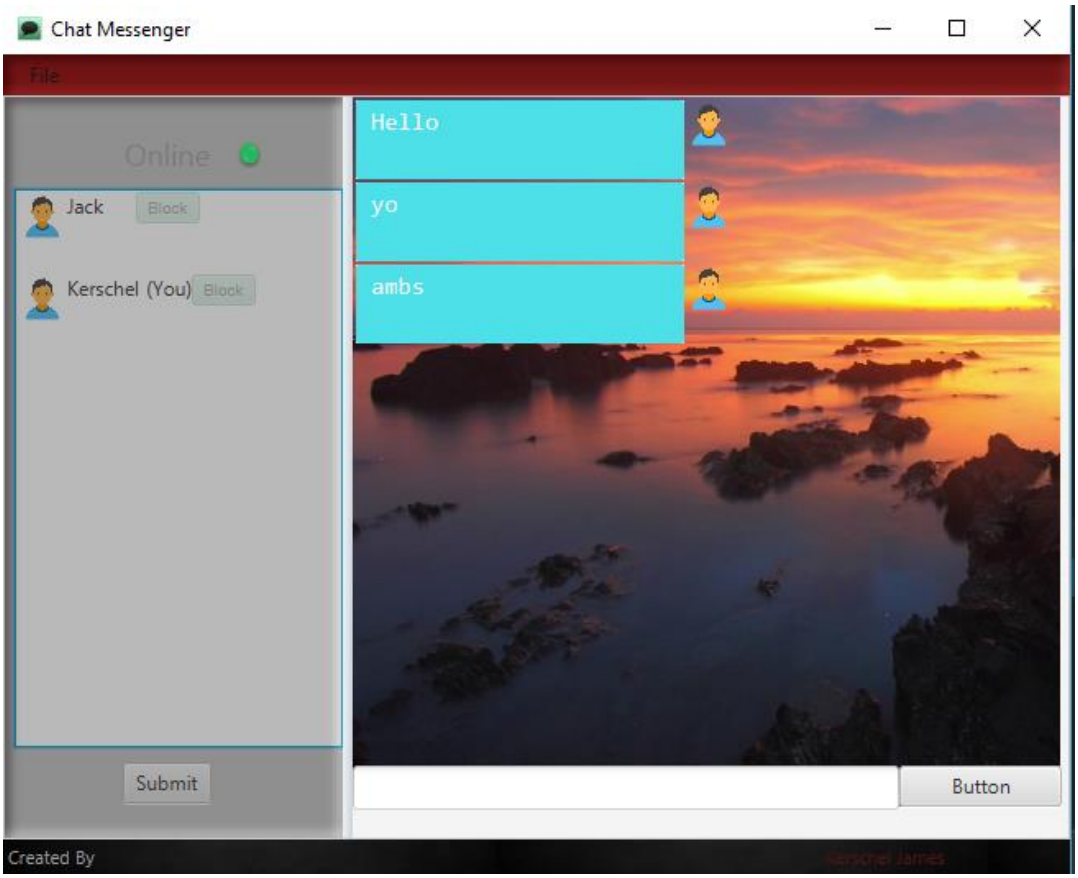
Sending Messages

```
public void sendChat(String message) {
        state =0;
        DatagramSocket aSocket = null;
        try {
            aSocket = new DatagramSocket();
            // create socket at agreed port
            int serverPort = 6789;
                  Send the message to the server with the ip concatenated
                message += "TO:" + sendToIP;
                byte[] send = message.getBytes();
                    DatagramPacket reply = new DatagramPacket(send,send.length,
                            InetAddress.getByName(serverIP),serverPort);
            message = message.split( regex: "TO:")[0];
            message = message.replaceFirst( regex: "3", replacement: "");
                    chat.add(message);
                aSocket.send(reply);

        } catch (SocketException e) {
            System.out.println("Socket: " + e.getMessage());
        } catch (IOException e) {
```

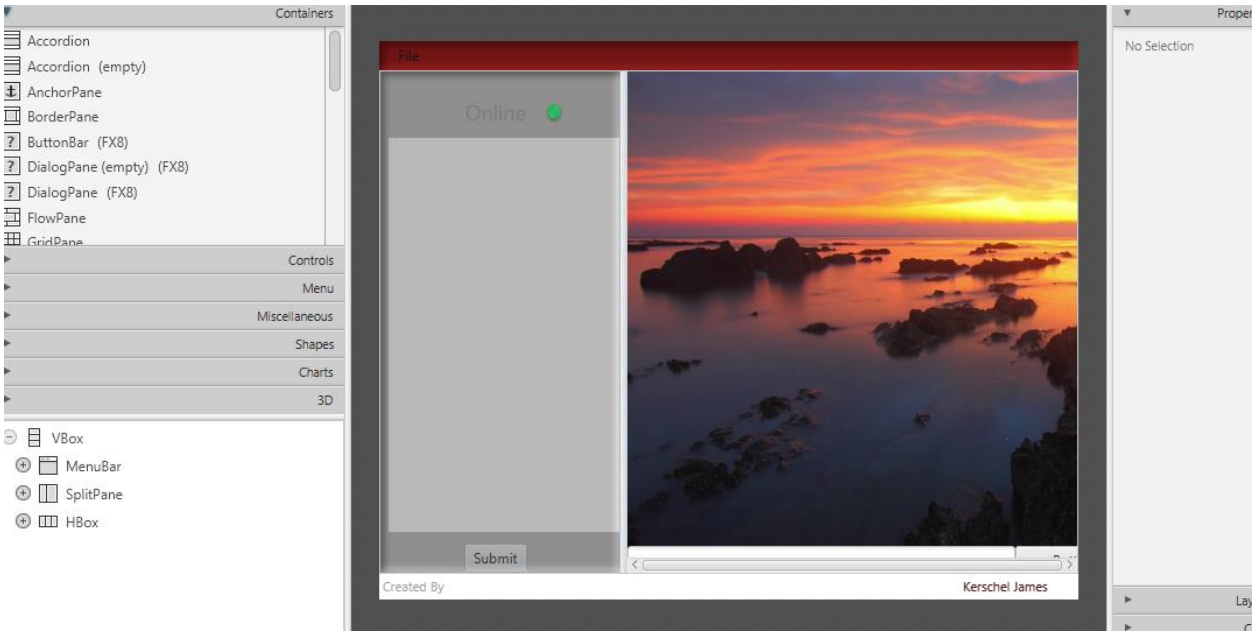
UI Design of ChatClient.java

This UI screen was created using FXML. It is a language similar to XML which I used before in android development therefore it was not as difficult to understand.



FXML file chat.fxml

It could also be edited though ScreenBuilder GUI.



Extra Feature

The extra feature I added was to allow a user to block another from chatting with them. Once this blockChat() is called it sends a message to the client “4block:” + IP. The “4” indicates a block operation on the server’s side. In order to unblock the user the server needs to reset.

```
    public static void blockChat() {

        DatagramSocket aSocket = null;
        try {
            aSocket = new DatagramSocket();
            // create socket at agreed port
            int serverPort = 6789;
//                  Send the message to the server with the ip concatenated
            String message = "4block:" + sendToIP;

            byte[] send = message.getBytes();
            DatagramPacket reply = new DatagramPacket(send,send.length,
                    InetAddress.getByName(serverIP),serverPort);
            aSocket.send(reply);

        } catch (SocketException e) {
            System.out.println("Socket: " + e.getMessage());
        } catch (IOException e) {
            System.out.println("IO: " + e.getMessage());
        } finally {
            if (aSocket != null) aSocket.close();
        }
    }
```