

Lauren Spruce
18011848



CS6002 Distributed and Internet Systems - Year 2020-21

Coursework 1

Report

Lauren Spruce

18011848

Contents

Introduction	3
Design.....	4
Server-Side UML model	4
Client-Side UML model	5
Development.....	6
Server-Side Classes.....	6
Client-Side Classes.....	7
Sequence diagram.....	8
Deployment	9
Installing and running the applications.....	9
Application Walkthrough	10
Connecting the server and clients	10
Messaging Clients and Chatrooms.....	11
Sending Attachments.....	14
Testing.....	16
Unit Testing.....	16
Starting the server	16
Testing Multithreading.....	16
Integration Testing.....	17
Functional Testing.....	19
Running multiple clients	19
Logging into the chatroom.....	20
Peer to Peer messaging.....	21
Client and Server communication.....	21
Sending Attachments	22
Creating Chatrooms	23
Conclusion.....	25

Introduction

In my coursework, I explored the functionalities of using a mixed architecture of a client to server model. I also implemented peer to peer model as clients can message and send files and images privately. I also implemented the ability to have multiple chatrooms, clients can create a chatroom by giving it a name. Other clients can join this chatroom by entering the chatrooms correct name. This is done by using the TCP protocol. I used swing-based GUI input and output for both the client and server side. Server side has an online user list, the ability to connect the server and disconnect, refresh, send files to chatrooms and individual clients. Client side has a GUI for both login/registering and the main window for chatting. All chats are displayed on the main window for the client for the purpose of easy testing with multiple clients.

My report will cover the applications structure and design, the software development, deployment of the system, a walkthrough of how to use the application and finally software testing.

Design

Server-Side UML model

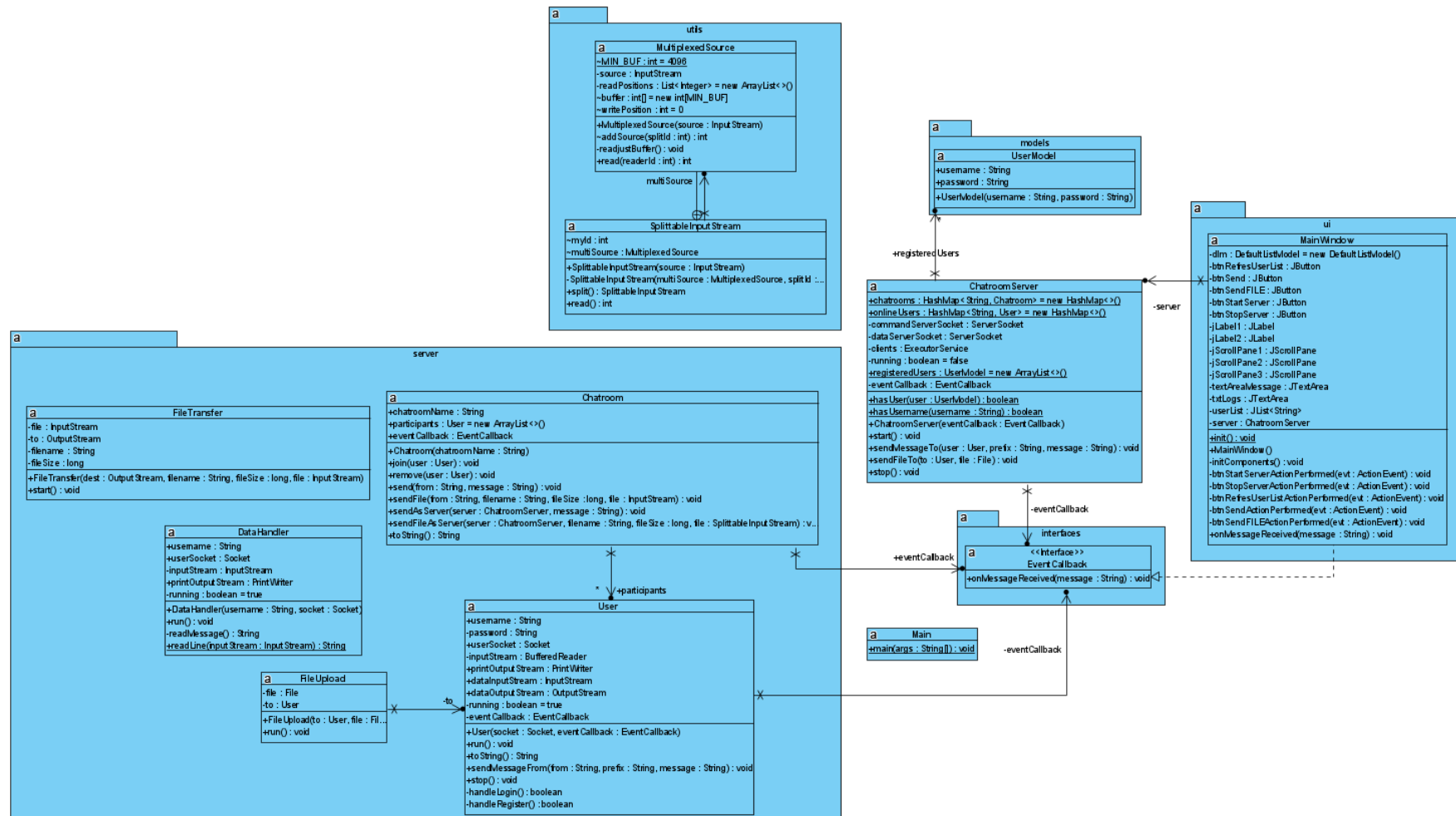


Figure 1: UML model for the client project Figure 2: UML model for the server project

Client-Side UML model

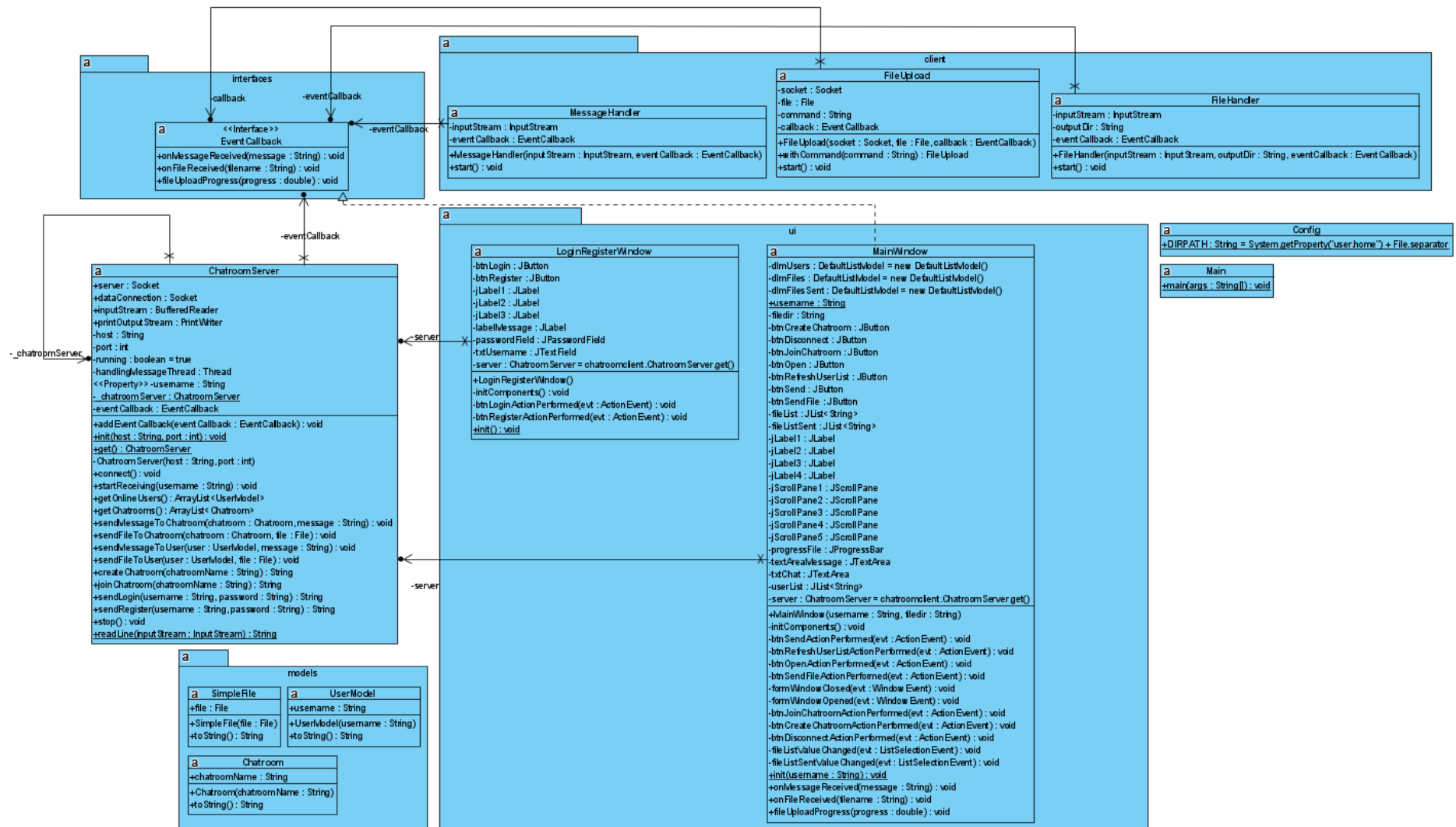
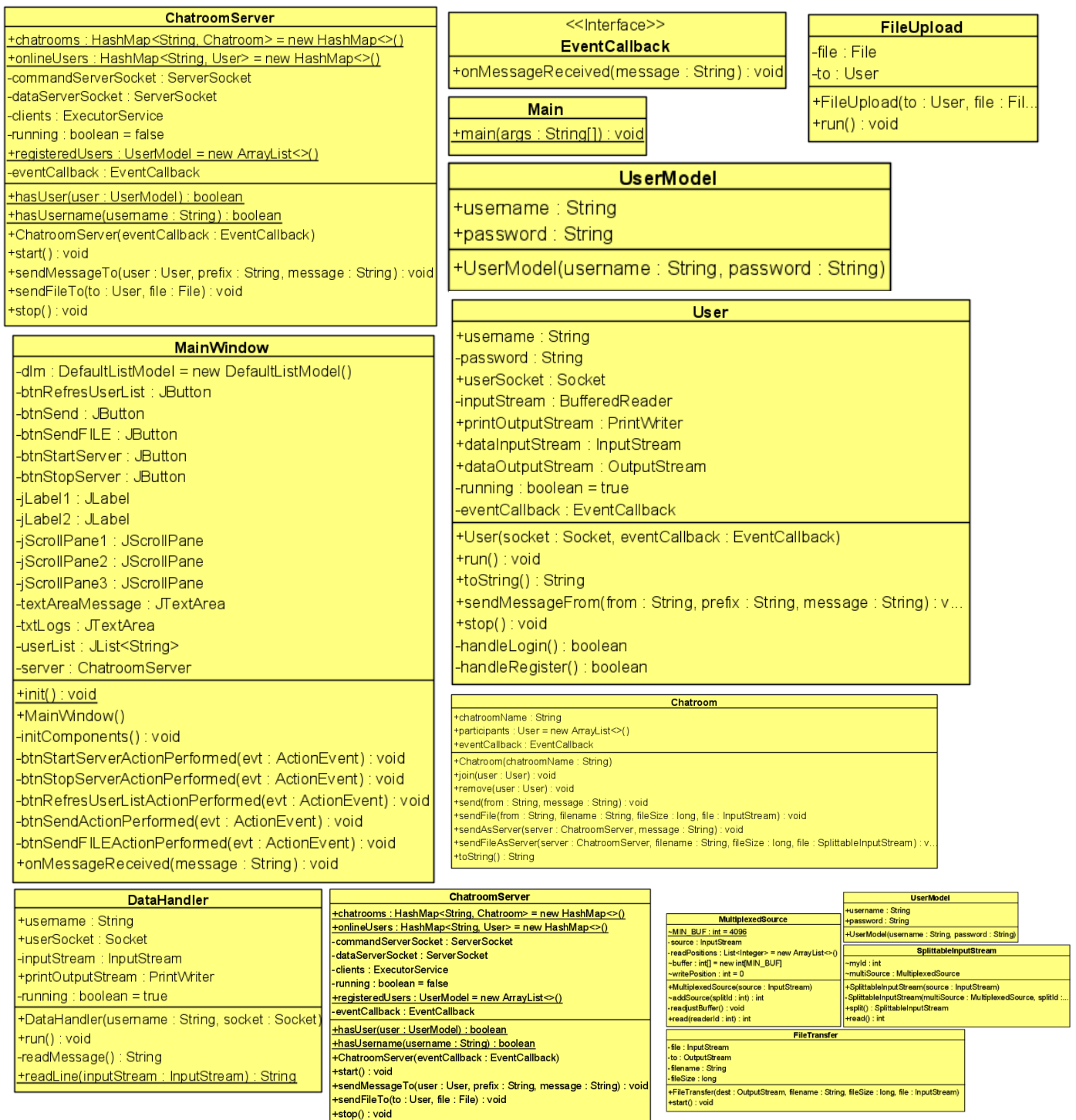


Figure 2: UML model for the client project

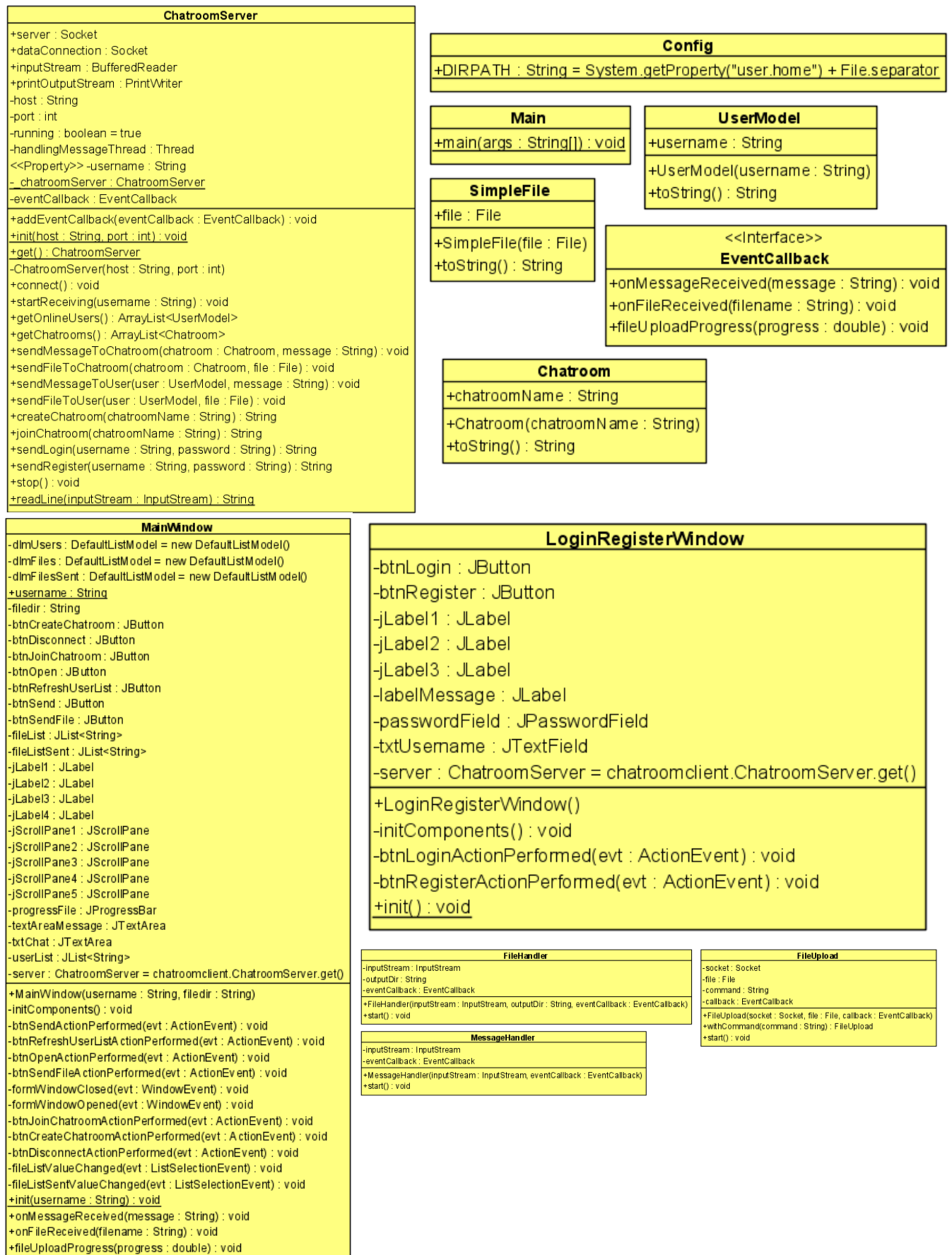
Lauren Spruce
18011848

Development

Server-Side Classes



Client-Side Classes



Sequence diagram

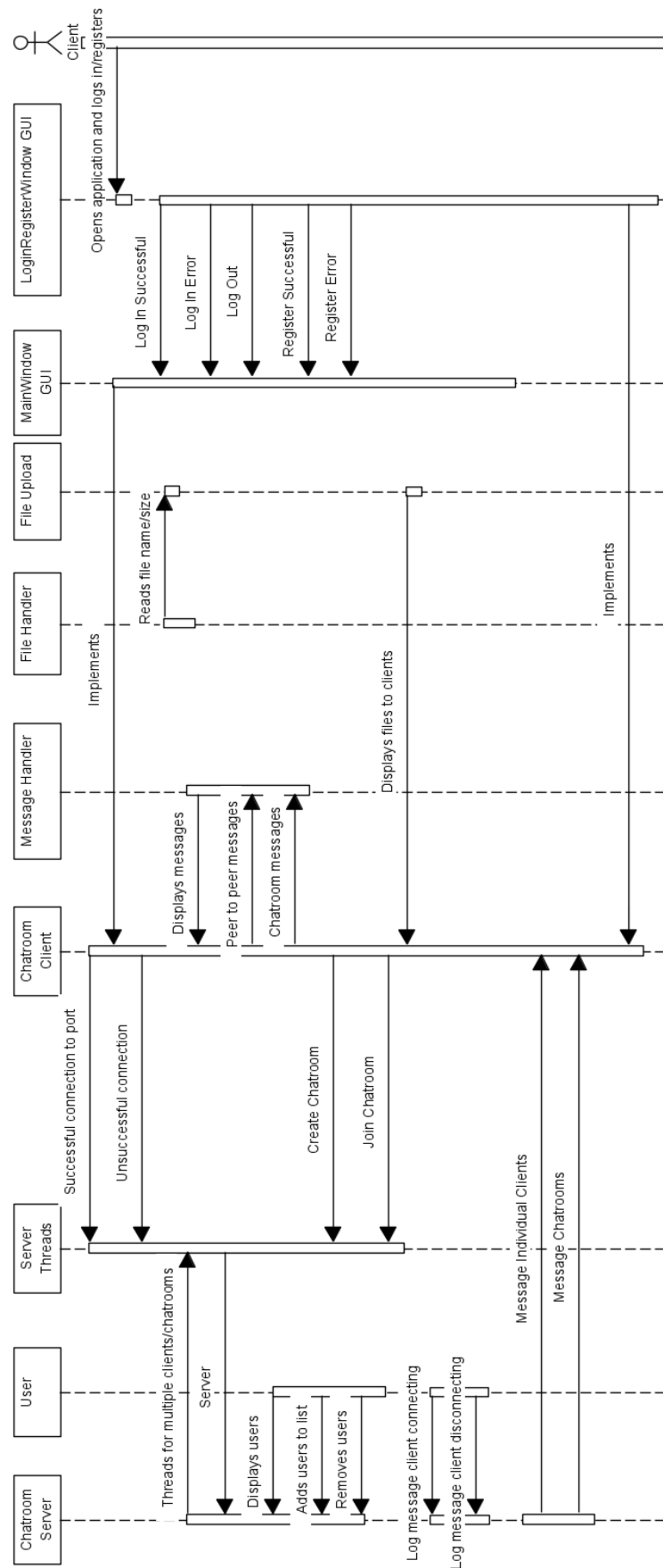


Figure 3: Sequence diagram for the application

Deployment

Installing and running the applications

I have made two JAR files which can be opened to run the server and client. I got these JAR files by cleaning and building both client and server projects, going into the dist folders of both projects and copying them. This allows you to run both client and server side without using NetBeans.

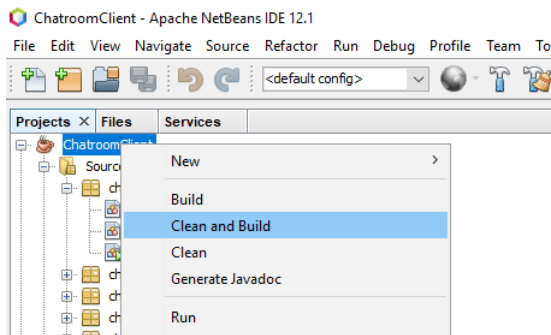


Figure 4: Cleaning and building Client Project

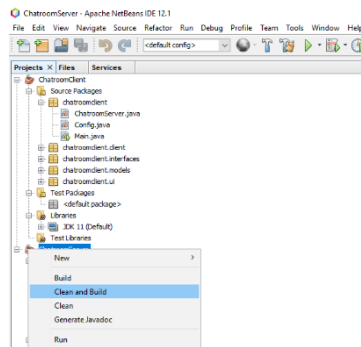


Figure 5: Cleaning and building Server Project

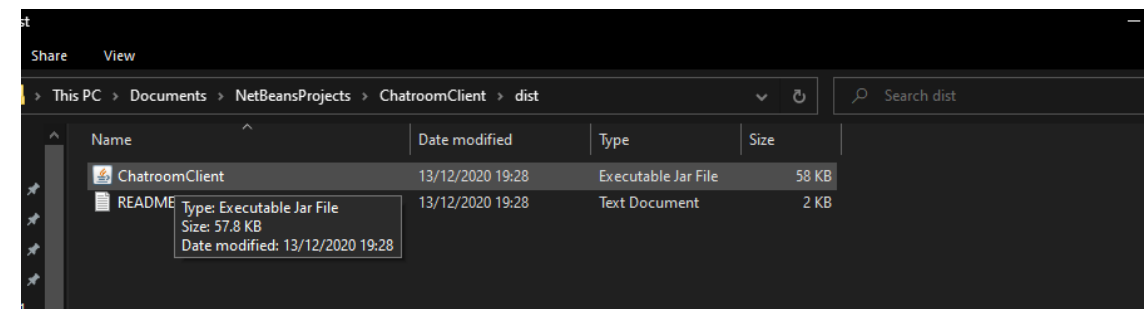


Figure 6: The project dist files copying the JAR file

First, you need to open the Server JAR file first and press the connect button, this will start the server. Then you can open the Client JAR file and register a client, or you could login with one of the premade logins. If a multiple user wants to be added onto the program, simply open the client file again and login/register with a different client.



	ChatroomClient	07/12/2020 14:02	Executable Jar File	58 KB
	ChatroomServer	07/12/2020 14:02	Executable Jar File	54 KB

Figure 7: Both the client and server JAR files

Application Walkthrough

Connecting the server and clients

First, you will need to start the server, to do this you can either open the JAR file or do it through NetBeans, simply right click the chatroom server project and click run. This will open the server UI window. From here, you must click start server in order to start the server. The main chatroom '#general' will appear once the server is started.

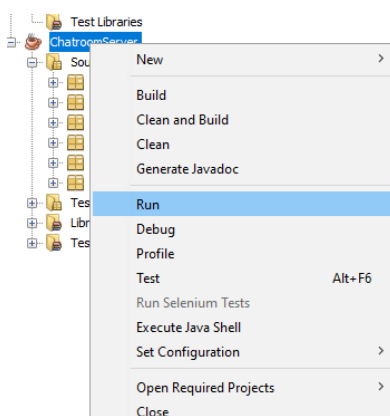


Figure 8: Running the server

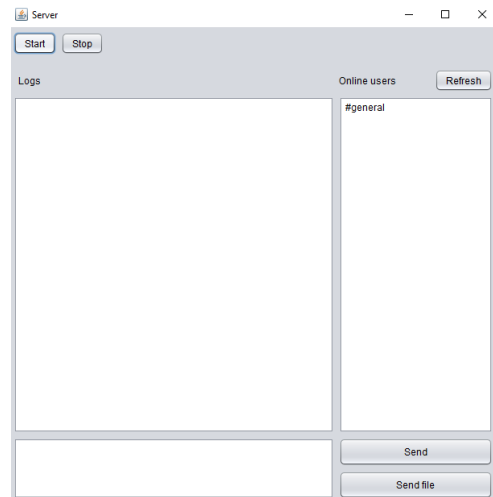


Figure 9: The main window UI for the Server

To add a client, you must either run the 'chatroom client' JAR file, or you can run it in NetBeans by right clicking and selecting run. (Note, you can only connect a client if the server is running.) Once running, the login/register window will appear. You can create a new user by registering, or you can login with one of the pre-existing users I have already made as shown below:

Username	Password
Lauren	pass1
Charlie	pass2
Bailey	pass3
Cooper	pass4

Table 1: Showing the pre-existing credentials

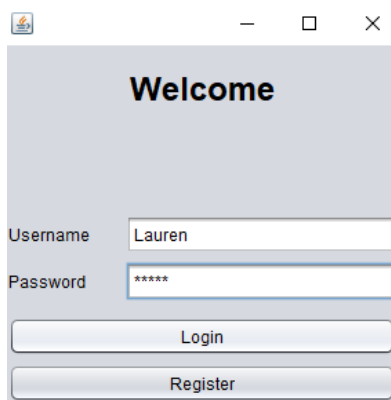


Figure 10: Logging into the client side, Login/Register window

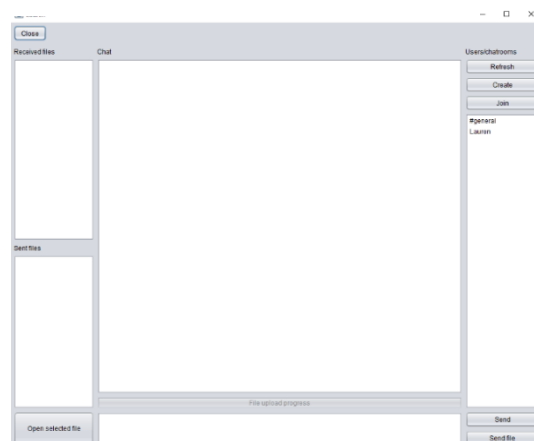


Figure 11: The main window for the client application

```
//Create the default users
registeredUsers.add(new UserModel("Lauren", "pass1"));
registeredUsers.add(new UserModel("Charlie", "pass2"));
registeredUsers.add(new UserModel("Bailey", "pass3"));
registeredUsers.add(new UserModel("Cooper", "pass4"));
```

Figure 12: The code for creating the registered users in a user model

If you try to register a user with the same name as another, it will display an error message saying, 'username is already taken!' Also, if you try to log in with an incorrect username and password to what you registered, then it will also display an error message.

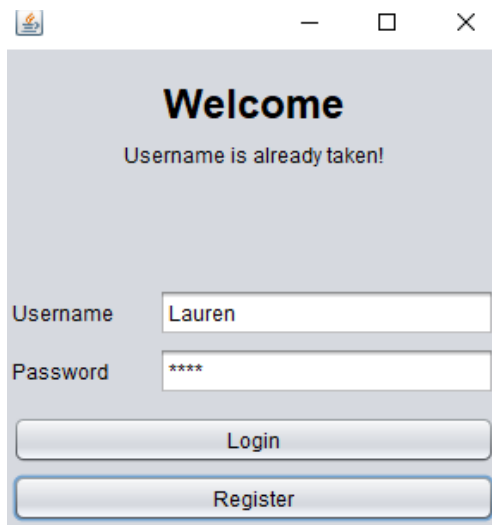


Figure 13: Username is already taken error message

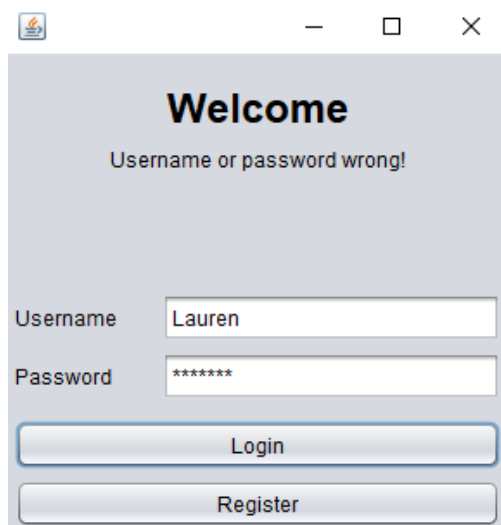


Figure 14: Username or password wrong error message

Messaging Clients and Chatrooms

Once logged in, the client can select the chatroom or individual client to send a message to, they can also send and receive files from different chatrooms. The server can send messages and files to chatrooms and individual clients. The server will also show when a new client connects in the server

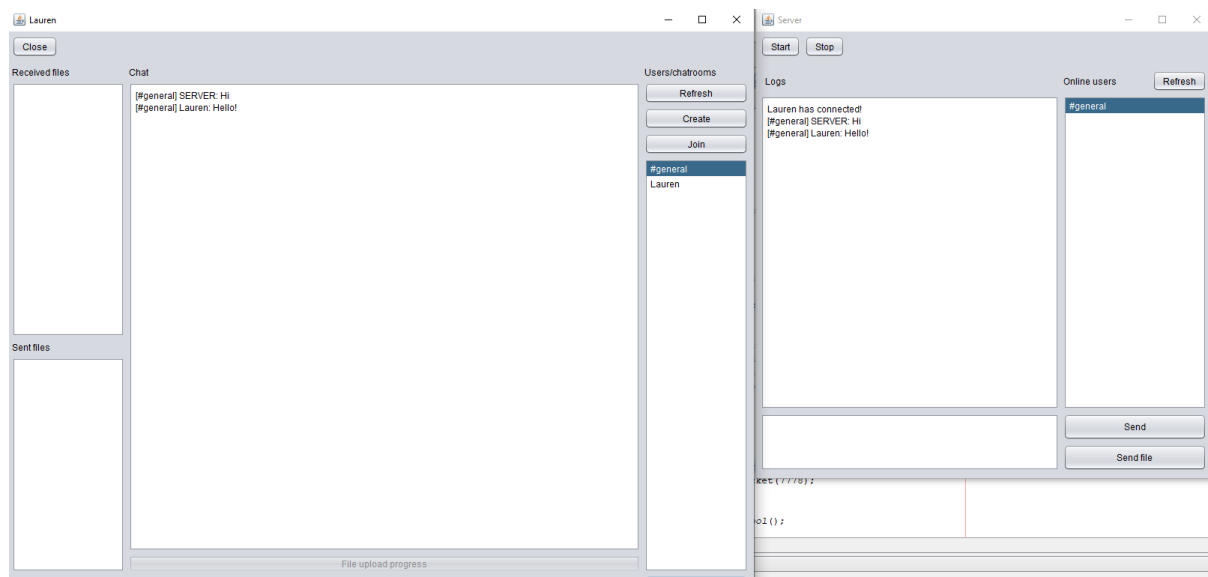


Figure 15: Client and server messaging on the general chatroom

Lauren Spruce
18011848

log, it will also say when they have disconnected. You must refresh the server to update the users/chatrooms field when a client connects or disconnects.

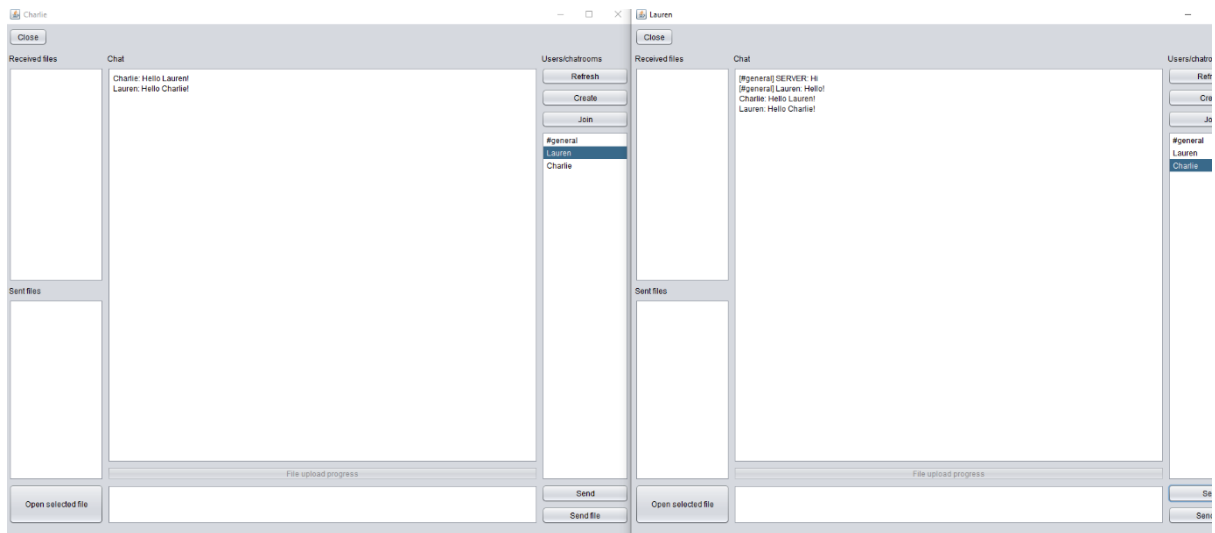


Figure 16: 2 clients communicating peer to peer

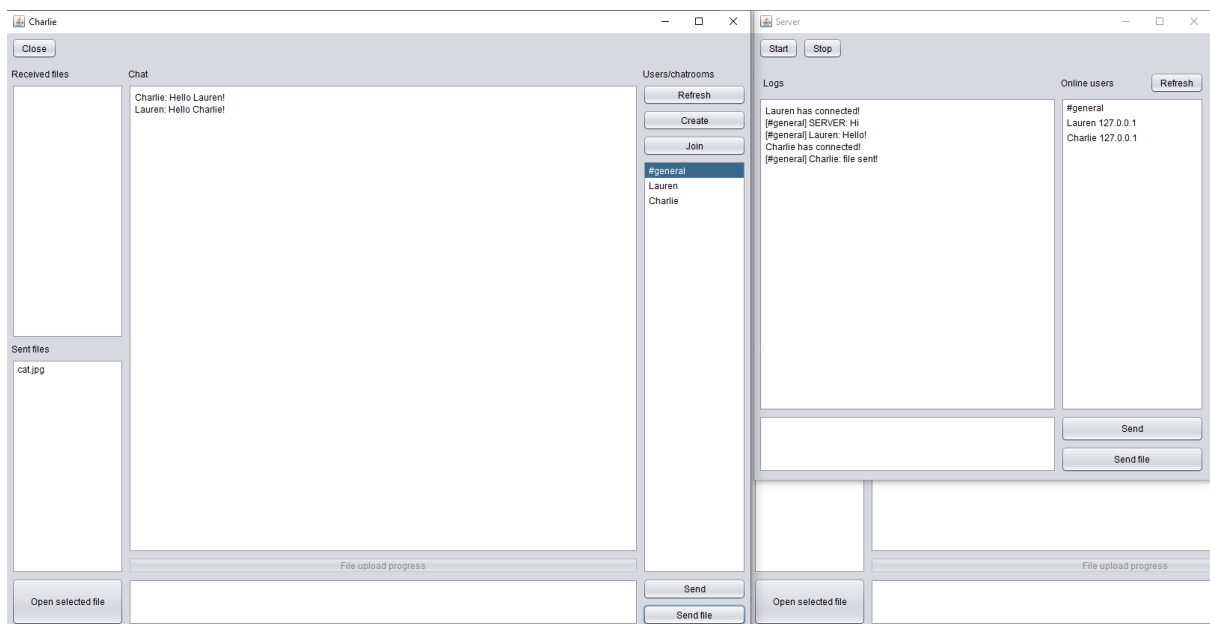


Figure 17: Client sending a file to the general chatroom

Lauren Spruce
18011848

To select who to chat to, the client/server must select the chatroom or client, by clicking their name in the users/chatroom text area. Also, the client has their name displayed as the window title to easily identify which client is which, as it got confusing identifying clients for testing purposes.

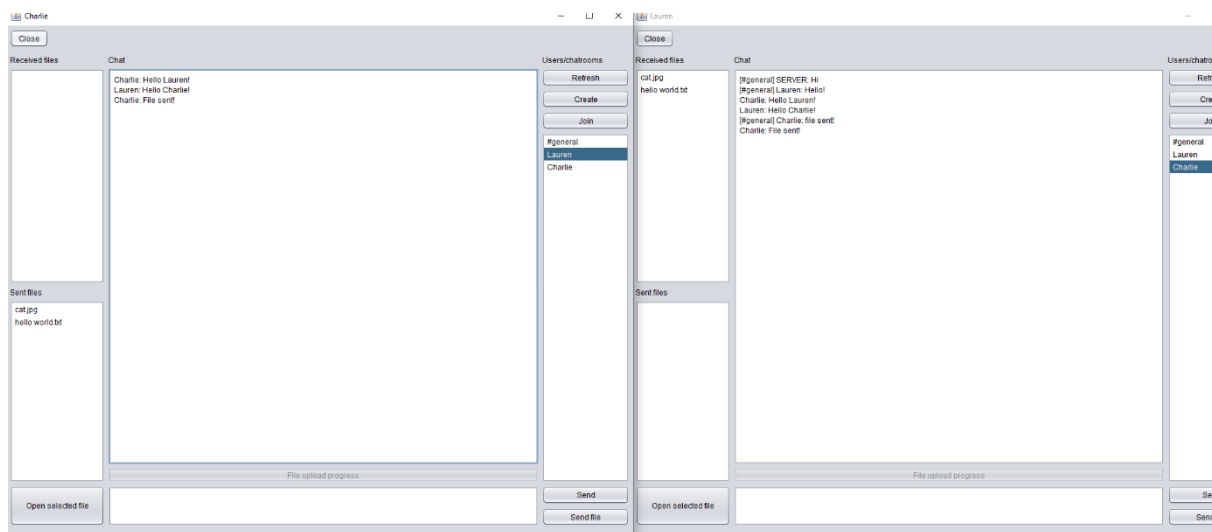


Figure 18: Client sending over a file to another client

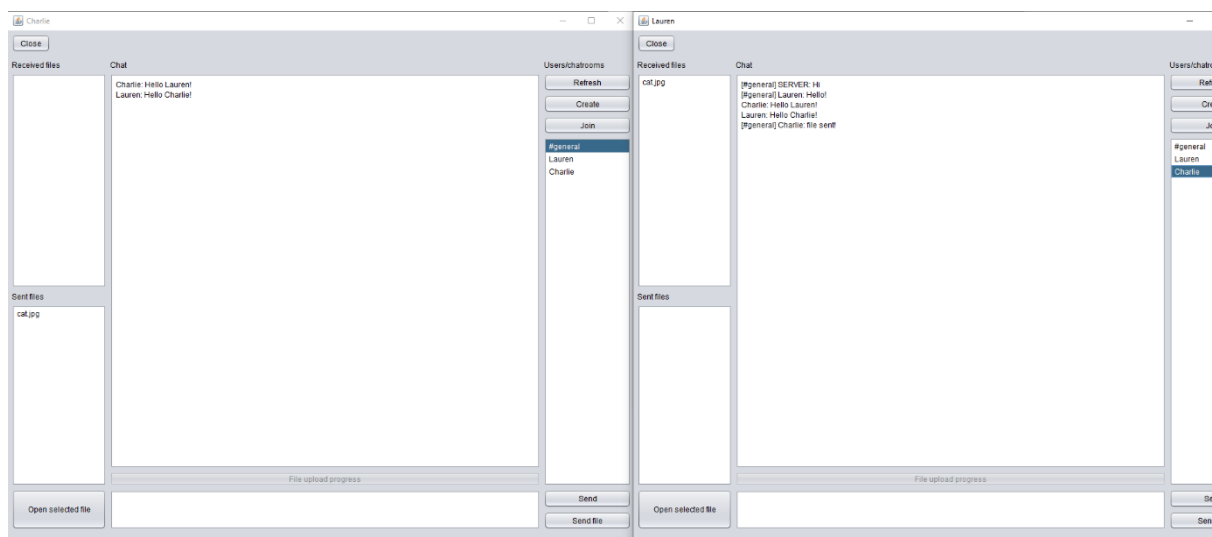


Figure 19: Clients sending over files

Lauren Spruce
18011848

Sending Attachments

Clients can also send files to new chatrooms if needed, these can be images or attachments.

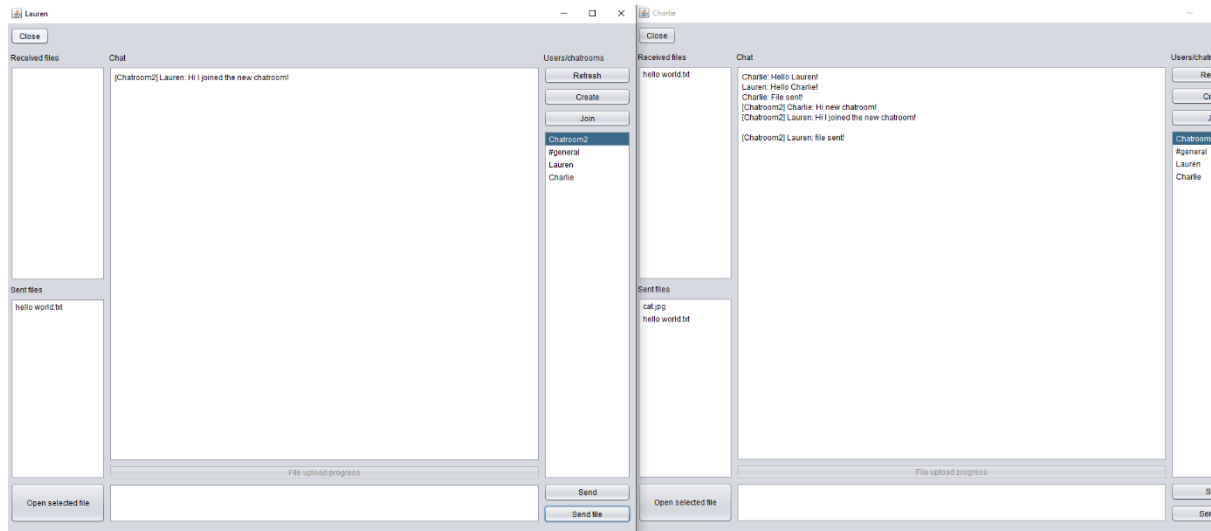


Figure 20: Client sending over file to a new chatroom

The server can also send these files to any client or chatroom.

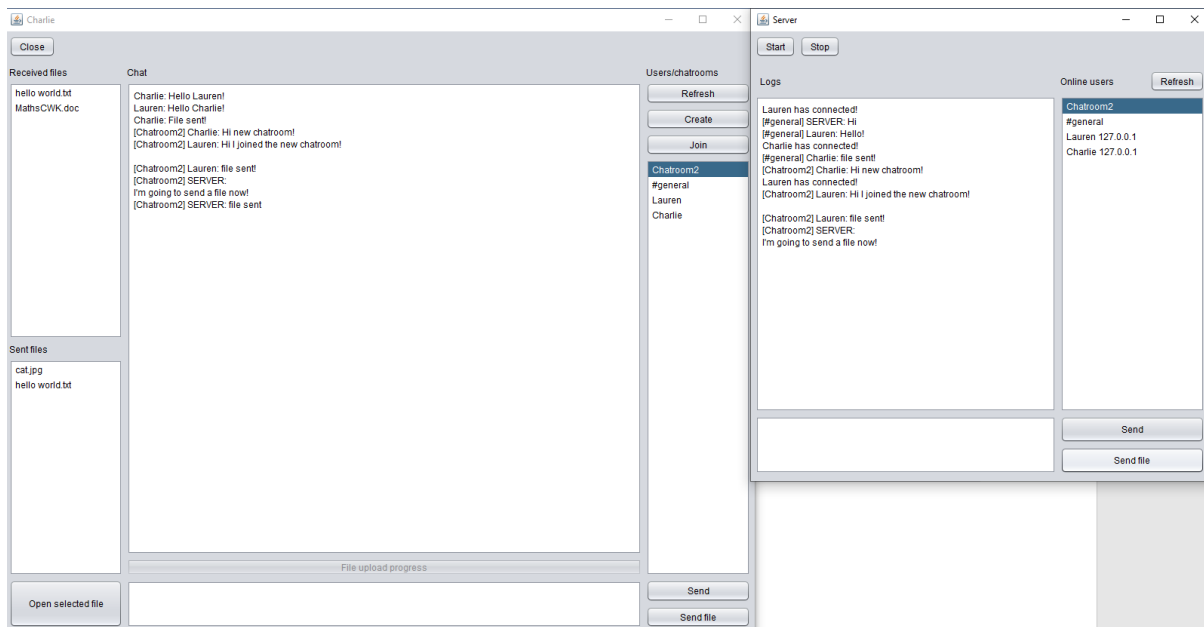


Figure 21: Server sending over attachments to a chatroom

Lauren Spruce
18011848

Clients can open the attachments by selecting the correct file from the 'sent files' or 'received files' text areas and click on the 'open selected file.'

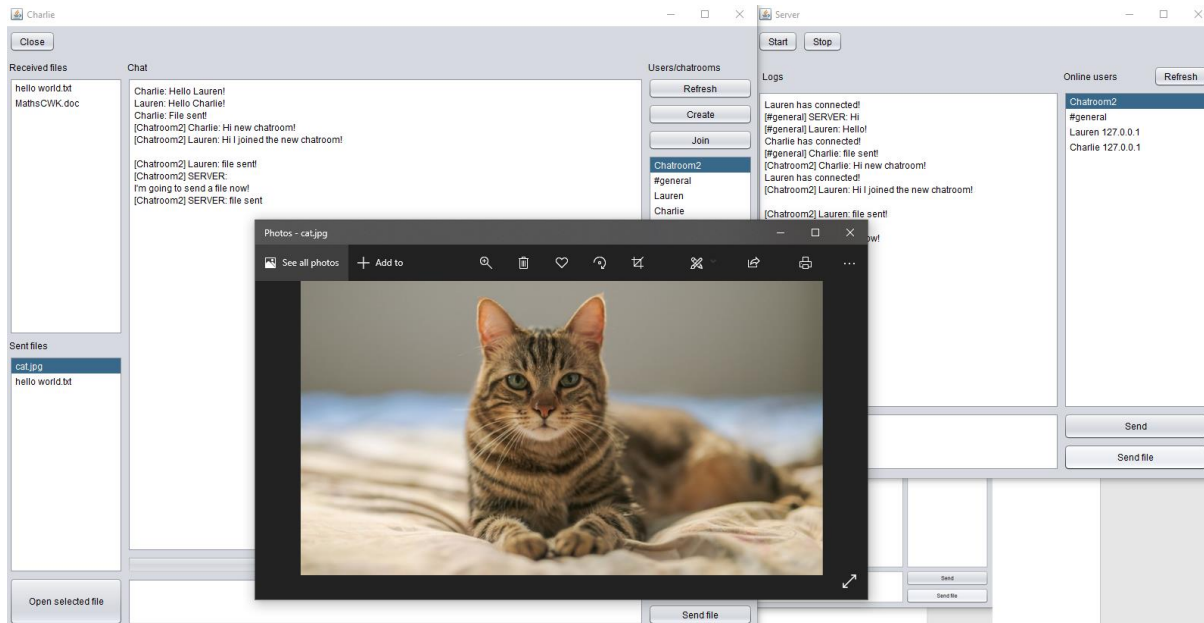


Figure 22: Client opening a sent attachment

Finally, to disconnect, the client can click the 'close' button to disconnect from the server. The server can also be disconnected by pressing the 'disconnect' button.

Lauren Spruce
18011848

Testing

Unit Testing

Starting the server

I clicked run on the Chatroom server project and it gave me the expected outcome. This is running on port 7777 for the command socket, the data socket is on 7778.

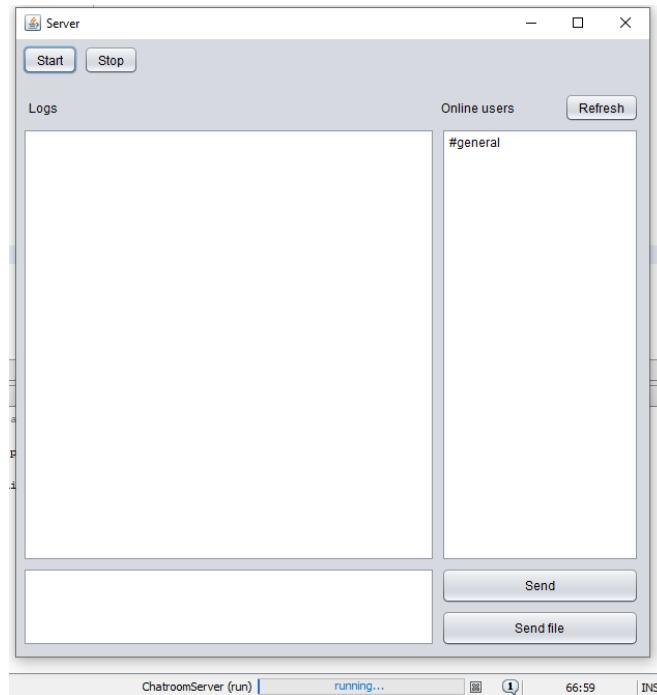


Figure 23: Server running

Testing Multithreading

To test multiple users connecting and messaging in the group chat I logged in using the premade clients and messaged using all 3 clients to the general chatroom.

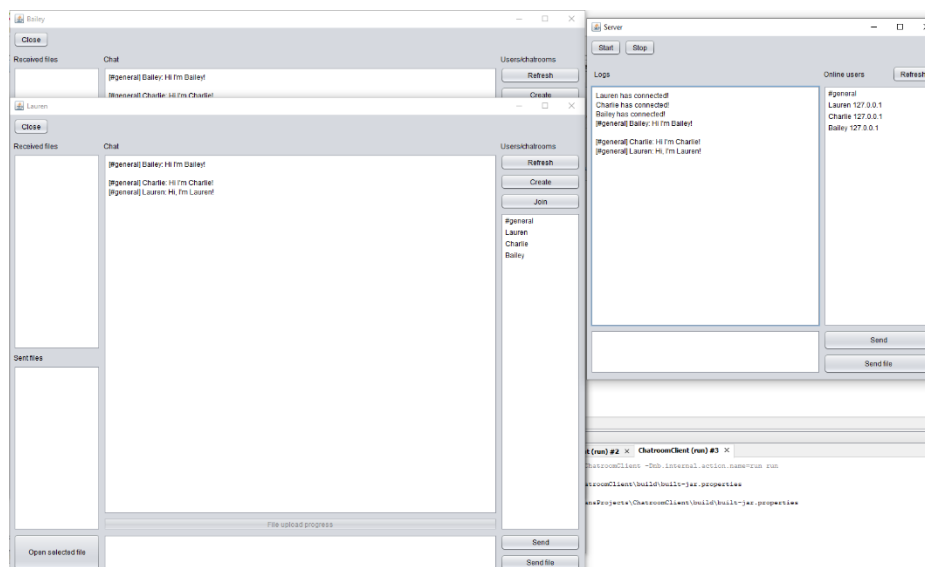


Figure 24: 4 clients connecting to the server with log messages

Lauren Spruce
18011848

Integration Testing

Sending attachments

I experienced a bug where when a client wanted to send a file to another client, the log message would appear to show that the client receiving the file was sending it, when wasn't.

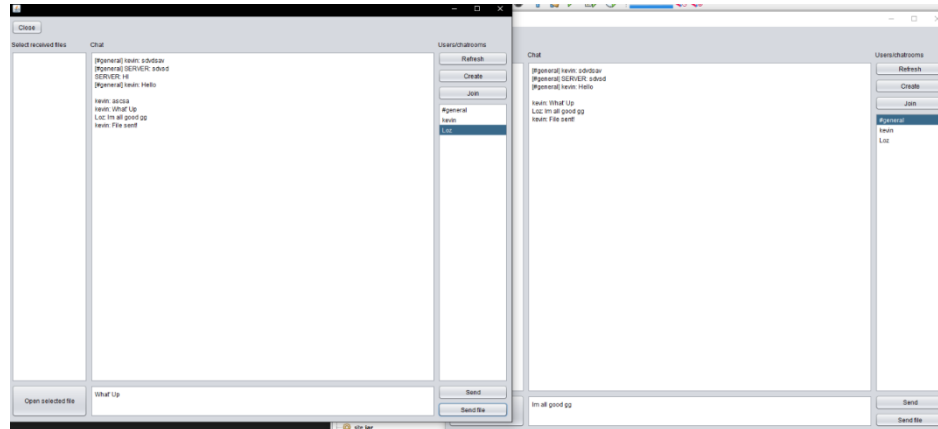


Figure 25: Bug showing clients names mixed up on sending attachment logs

To fix this, I saved the files to the disk, along with the client's username and file name and this solved the bug.

```
//Save file to disk
FileOutputStream fos = new FileOutputStream(outputDir + MainWindow.username + "_" + filename);
BufferedOutputStream bos = new BufferedOutputStream(fos);
```

Figure 26: Code to show new code implemented

Messaging using text field

Bug encountered: The text area wasn't clearing after client or server were sending a message.

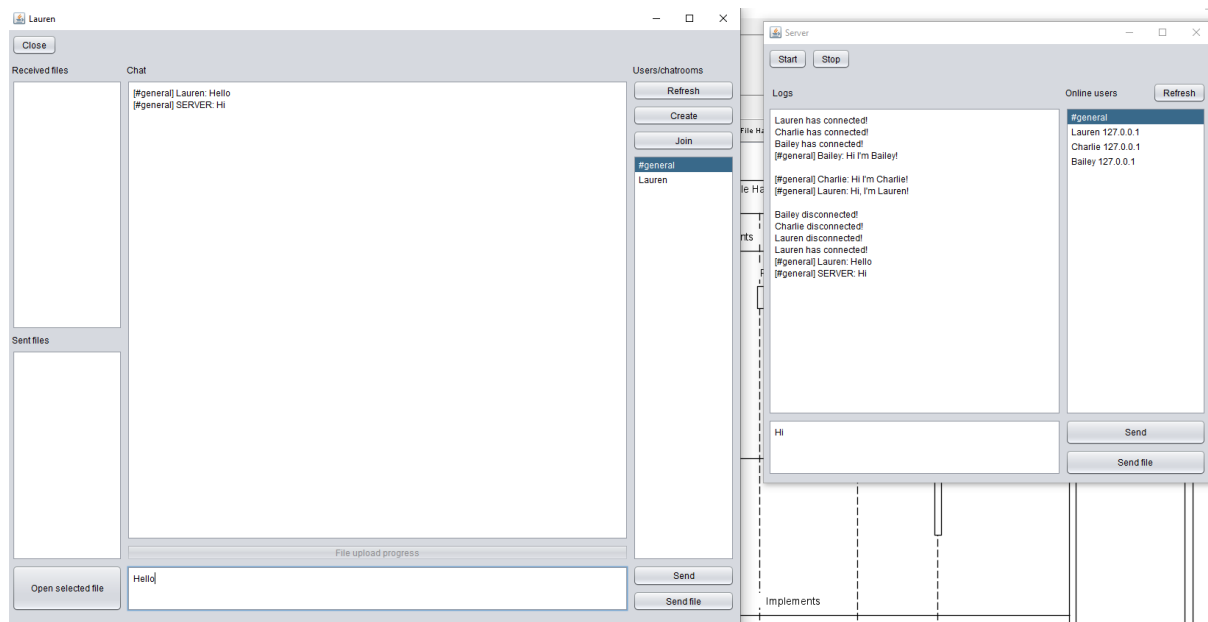


Figure 27: Showing the text field not clearing after messages sent on the client and server side

Solution: I fixed this by setting the text area to an empty string after the message is sent. I did this in both Main window UIs for the client and server 'send' buttons.

```
    }  
  
    textAreaMessage.setText("");
```

Figure 28: Code to show the corrected version

Another bug I encountered was that the client and server could send empty string messages. The client that sent them could see them but no other client or the server could.

To fix this, I added an if statement to not send the message if it was an empty string on the 'send' button.

```
private void btnSendActionPerformed(java.awt.event.ActionEvent evt) {  
    String message = textAreaMessage.getText();  
    int selectedIndex = userList.getSelectedIndex();  
    if(selectedIndex < 0 ) return;  
    if(textAreaMessage.getText().equals("")) return;  
}
```

Figure 29: Code to show empty strings are returned and therefore not sent as a message

When I was testing early on in my prototype, I didn't have any way for the client/server to check who was logged in on the 'users/chatrooms.' So, I added a 'refresh' button, this is to clear the list then update it with all the clients and chatrooms created.

```
private void btnRefreshUserListActionPerformed(java.awt.event.ActionEvent evt) {  
    try {  
        //Clear user list and populate again with new chatrooms and users  
        dlmUsers.clear();  
        for(Chatroom chatroom : server.getChatrooms()) {  
            dlmUsers.addElement(chatroom);  
        }  
  
        for(UserModel user : server.getOnlineUsers()) {  
            dlmUsers.addElement(user);  
        }  
  
        userList.setModel(dlmUsers);  
    } catch (IOException ex) {  
        Logger.getLogger(MainWindow.class.getName()).log(Level.SEVERE, null, ex);  
    }  
}
```

Figure 30: The refresh button implemented in the client main window GUI

Functional Testing

Running multiple clients

I ran the chatroom client and logged in with a premade user 4 times, this connected all 4 of my clients and further proves that the multithreading is working in my application. Once all 4 clients disconnect, the server log will output the correct message, saying the client's username, that they have disconnected.

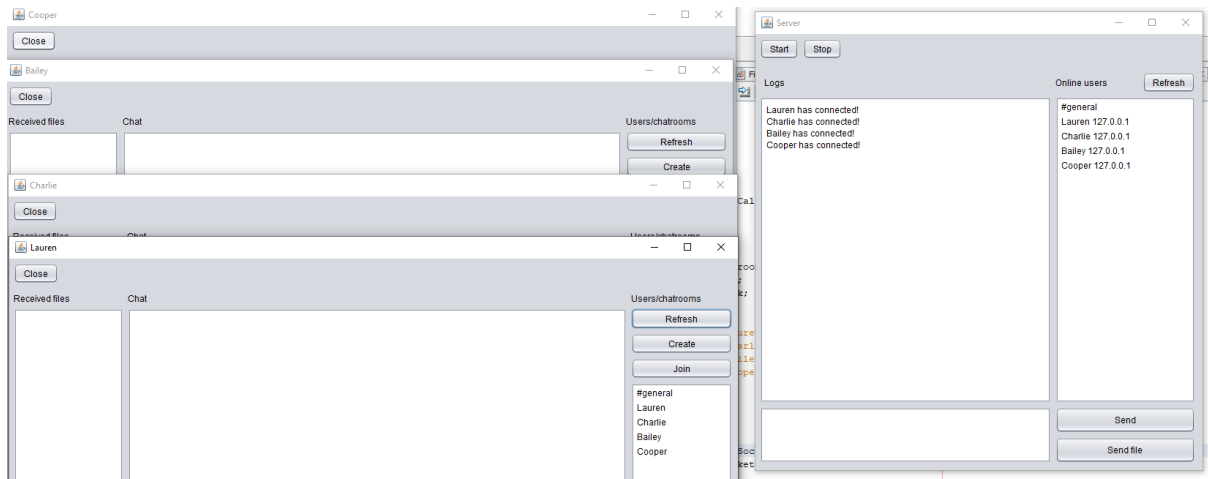


Figure 31: Running multiple clients

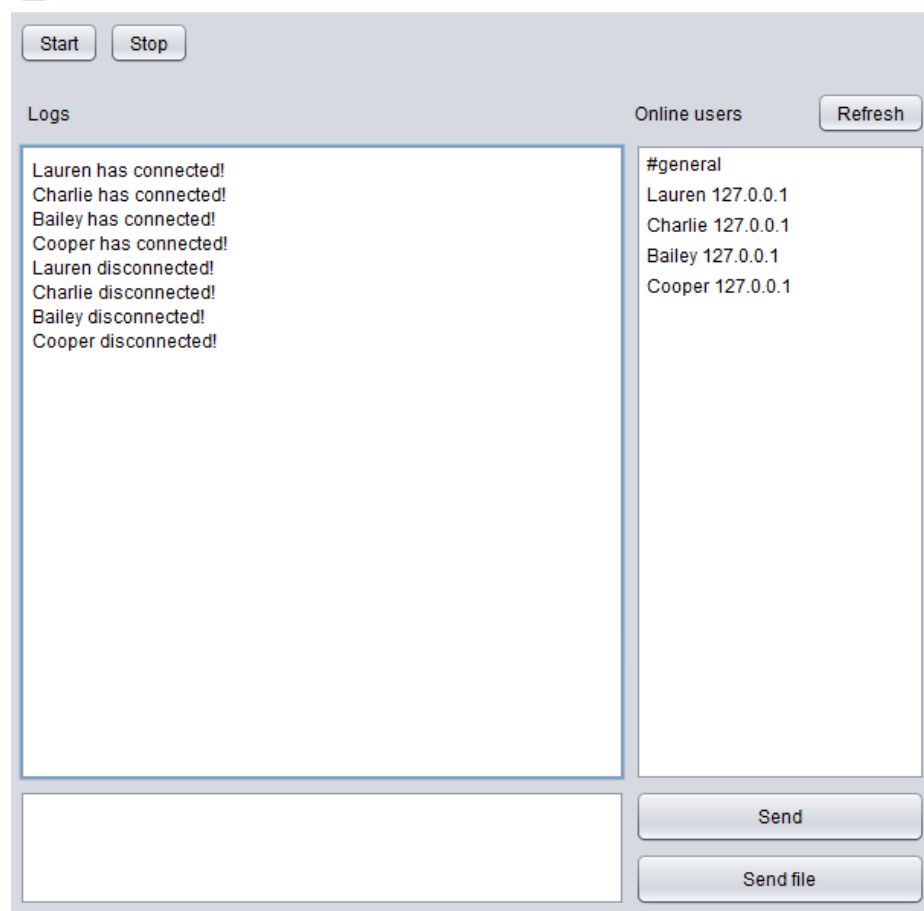


Figure 32: The server log of clients connecting and disconnecting

Logging into the chatroom

If the client enters the wrong credentials, they will receive an error message. If the client tries to register with a pre-existing username another error will be outputted. If the client is successful logging in, they will successfully be taken to the main window UI.

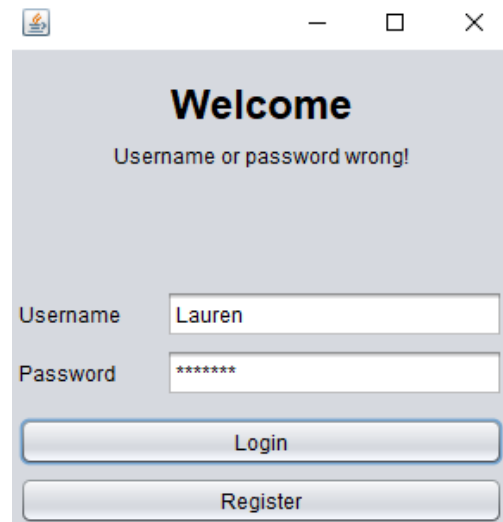
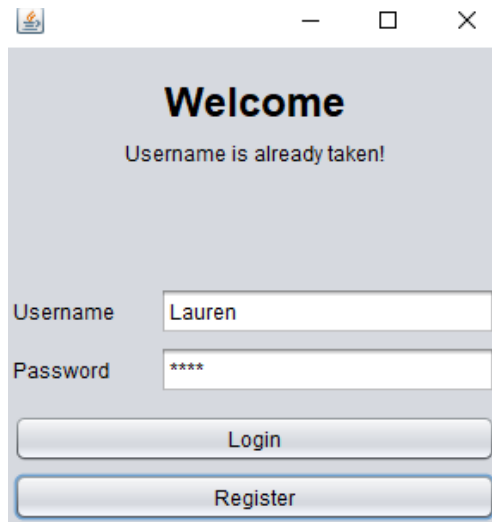


Figure 33: Client username already taken error message Figure 34: Incorrect username or password error message

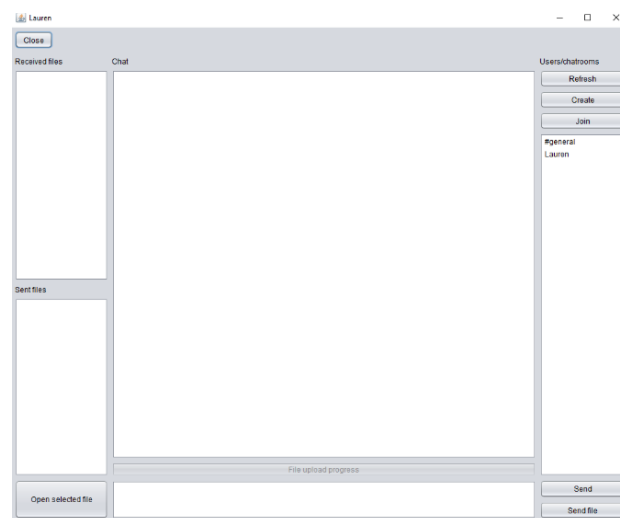


Figure 35: Main window client-side GUI after successful login

Lauren Spruce
18011848

Peer to Peer messaging

Once the client is successfully logged in, they will be able to select another client to message from the 'users/chatrooms' text area. Once highlighted, they can type their message and send, this will only send to the selected client.

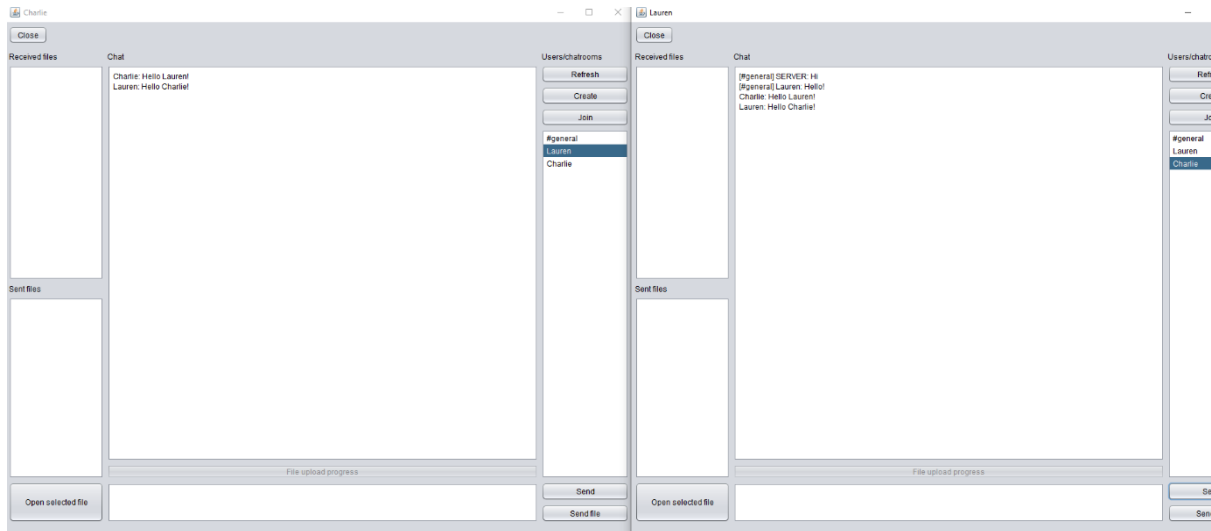


Figure 36: Peer to peer messaging

Client and Server communication

The server can send chatrooms and individual client's messages. Clients can message the general chatroom and communicate with other clients here.

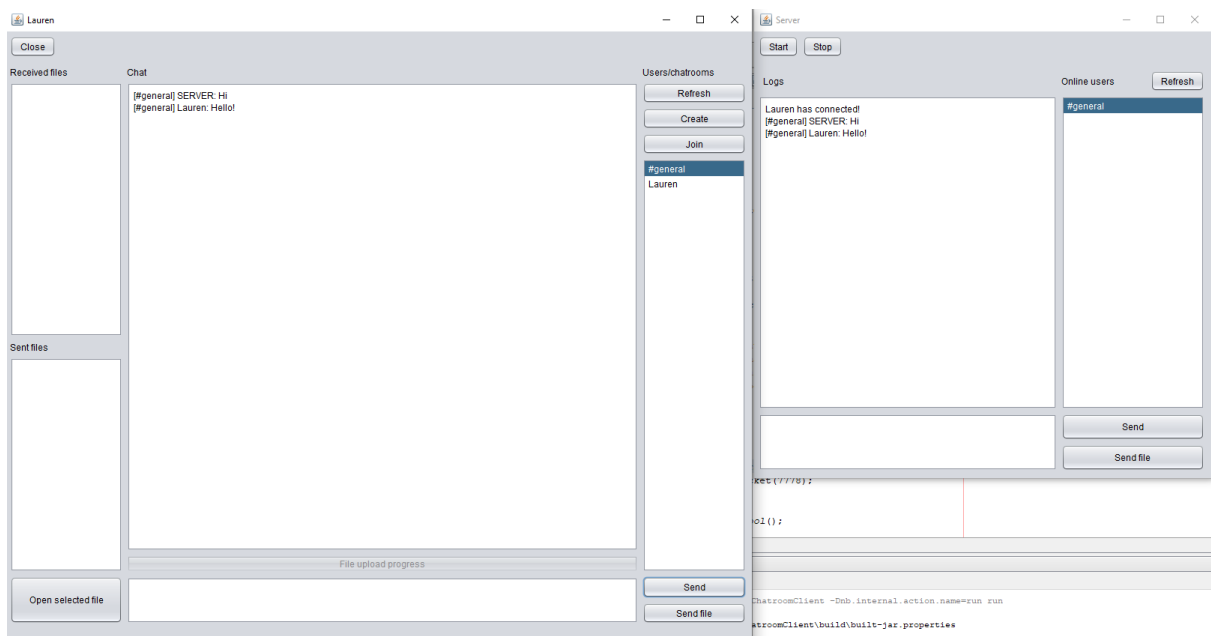


Figure 37: Client and server communicating

Lauren Spruce
18011848

Sending Attachments

Clients can also send files to new chatrooms if needed, these can be images or attachments. The server can also send files to clients and chatrooms. Clients can open the attachments by selecting the correct file from the 'sent files' or 'received files' text areas and click on the 'open selected file'.

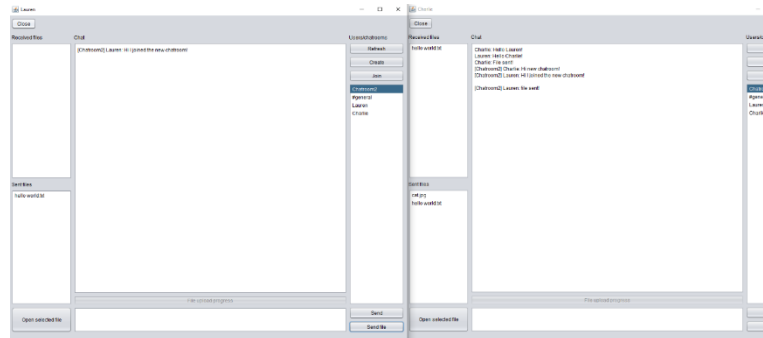


Figure 38: Client sending attachments to a new chatroom

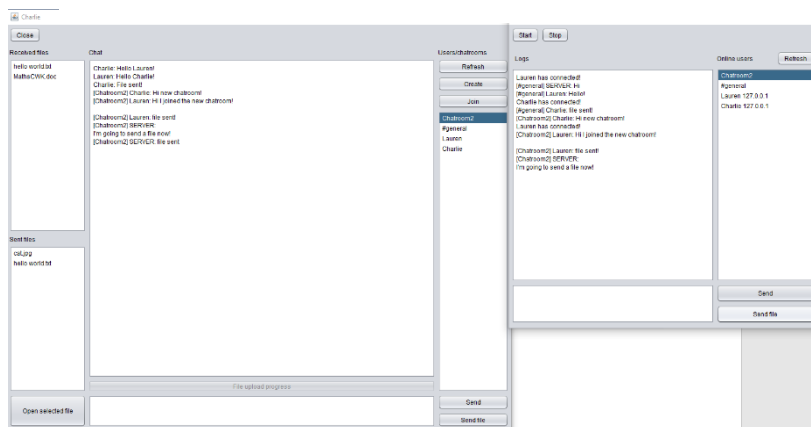


Figure 39: Server sending a file

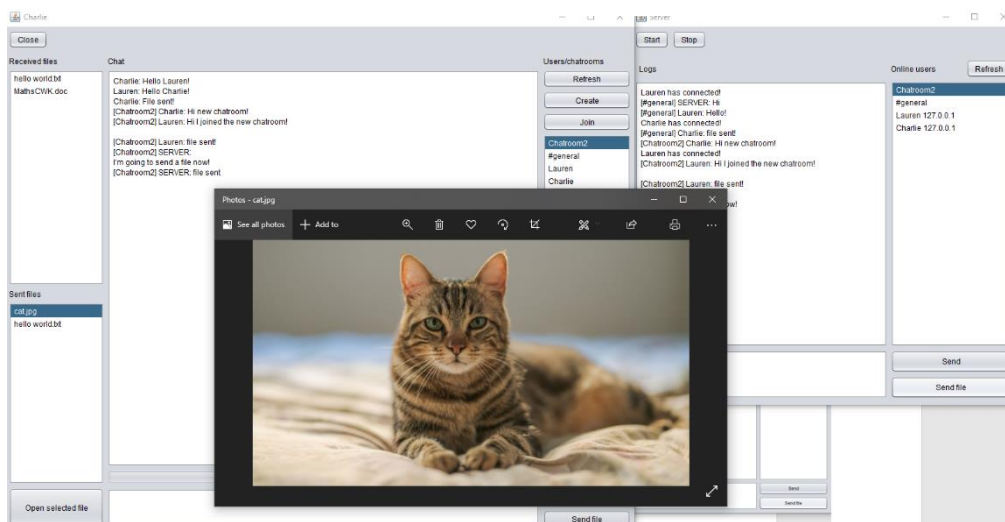


Figure 40: Client opening a file

Lauren Spruce
18011848

Creating Chatrooms

To create a chatroom, the user must click the 'create' button, this will display a window, which will prompt the client to enter a chatroom name. Once entered, the client must refresh the window for it to then appear on the 'users/chatrooms' list. The server will also need to refresh for it to update on its end.

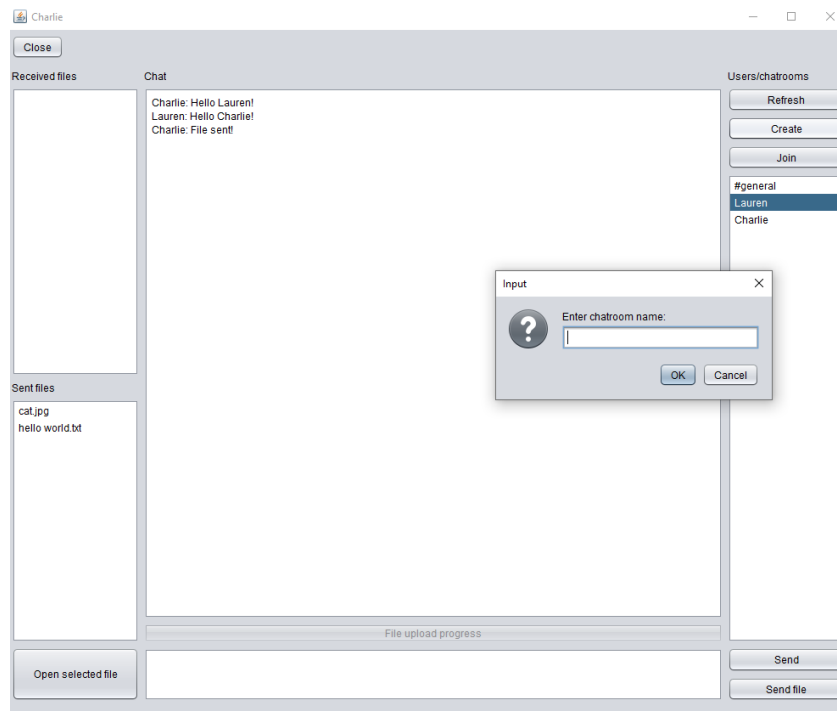


Figure 41: Client entering a new chatroom name

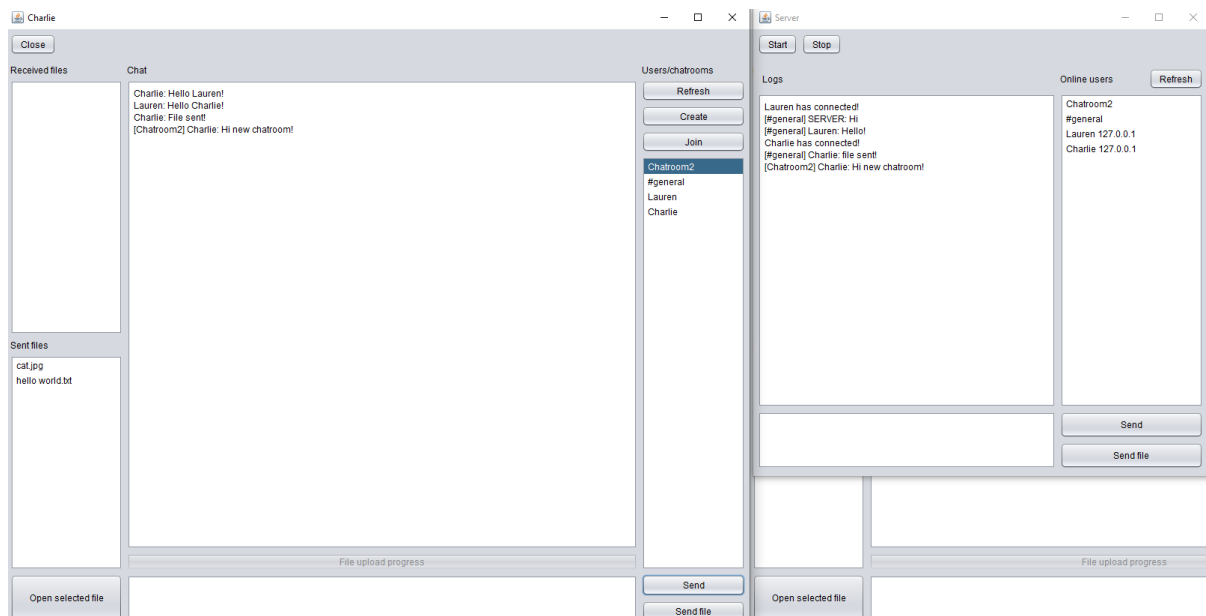


Figure 42: Shows that the new chatroom 'chatroom2' was created and visible to server and client

Other clients cannot see this chatroom or receive messages from the chatroom unless they use the 'join' button to enter. They must enter the name of the chatroom, if it's the right name it will allow them access to see all messages and files. If an incorrect chatroom name is entered, an error message will display 'chatroom doesn't exist.'

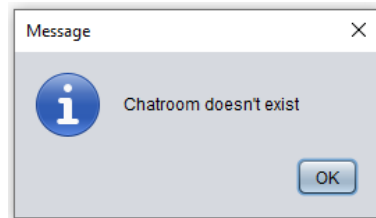


Figure 43: Client trying to join a non-existing chatroom or one that hasn't been created

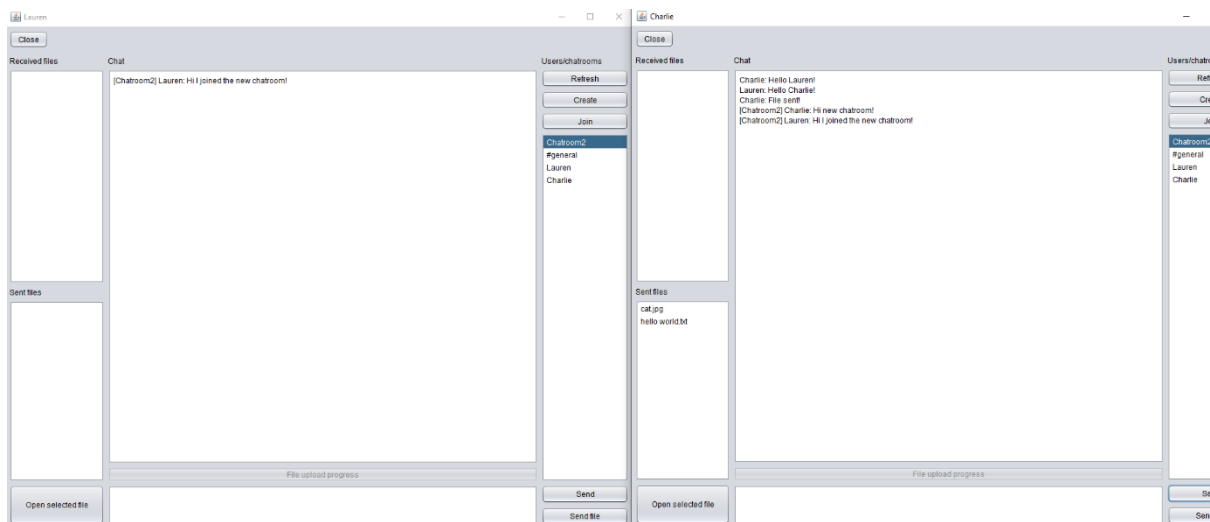


Figure 44: Successful joining of another chatroom

Conclusion

To conclude this report, I feel like I have developed as a programmer significantly because of this assignment. I learnt in detail about the TCP protocol and how to implement it into a simple chatroom program. I really enjoyed experimenting with the multiple features implemented in my work. It was also rewarding to implement data structures I learnt prior, CS5003, to tie my work together. This includes array lists and HashMap for usernames and passwords.

Throughout the prototype development, I struggled on implementing Java Swing to create GUIs for both client and server side, as previously, my only experience with java swing was creating simple GUIs for sorting user inputs with linked lists. To overcome this problem, I spent a lot of my time researching the TCP protocol, also using Oracle's website to get more familiar with how Java Swing functions.

For future work, or if I was to complete this prototype again, I would have improved a few features. First, I would have created a simple database, for the client's login and register credentials, so all logins created could be saved. Also, I would have created the feature to open a new window for each client when they initiate a new chat, the same for creating a new chatroom. For testing, the single main window was easier to keep track of the conversations, but for a more professional program I wouldn't have stuck with this singular window approach.