COMP3331 Report

**The application layer message format and a brief description of how your system works. Also discuss any design tradeoffs considered and made. Describe possible improvements and extensions to your program and indicate how you could realise them. If your program does not work under any particular circumstances, please report this here. Also indicate any segments of code that you have borrowed from the Web or other books.**

**Design Choices**

**Message formatting**

**Dictionaries for both Socket and ObjectOutputStream**

As we had to send socket information and packets to different Clients, I kept a HashMap of each client username to their respective

**Having Functions send the Object rather than within my Main on Server side**

**Error Checking**

I did not do much input error checking on the client side and assumed . as I thought it would unnecessarily complicate the code and was also not within the scope of the project.

Server side however, I did some basic checks including ensuring n

IMPROVEMENTS/TRADEOFFS

**ServerSocket Address**

Currently when I’m storing each clients’ ServerSocket IP address, I am returning the localhost / loopback address 127.0.0.1 using the code in the line below:   
I believe that this method could be improved by using the proper way to get the ServerSocket address through the getLocalInetAddress() function.

However, I could not do so in this case because I have instantiated ServerSockets using only the port number as a parameter, whereby this method returns 0.0.0.0 as the sockets IP. As this IP listens to all IPv4 addresses on your computer, you cannot establish a specific socket connection using 0.0.0.0 as the address parameter.

The way I could improve this design is by giving an IP address as an additional parameter when I start my client program and use this address to instantiate the Client’s ServerSocket.

**Logging out on Client Side**As in the requirements, the client terminal must be terminated when the client failed to login 3 times, logged out or timed out. I was unsure of how to do this but after reading this article (<https://www.baeldung.com/java-thread-stop>), I decided to stop my threads through interruptions and exceptions rather than deliberately setting a flag in my program.

I have two different types of threads within my program, a thread for listening to client commands and threads to listen to each peer to peer connection.

For my client command thread, I stop the thread through an interruption. However, because the *‘for’* loop is checking for when an interruption is received by the thread, it does not immediately close and requires the user to press the enterkey (write to STDIN) an additional time before the program is terminated.

For the threads that listen to peer connections, I close the thread by closing the socket connection. As this raises an EOFException, I catch this exception and then ask the thread to return. I believe that I could of improved this by using a thread interrupt again as currently I would not know whether the connection was deliberately closed or closed through an error.

APPLICATION LAYER PROTOCOL

For the application layer protocol, I tried to replicate HTTP and created an object that I could easily transfer between my clients to server and peer to peer. I named this object *‘TCPackage’.* It implements the ‘*Serializable’* interface and as such it can be transferred between clients or server through an ‘*ObjectOutputStream’.*

The *‘TCPackage’* class contains 5 different fields including:

**Content** – the message that is sent between clients or between the client and server e.g. messages between clients or error messages sent by the server. This field is ALWAYS printed out to the terminal by the client program.

**User** – stores the username of the person you want to message/interact with. Only prior to login, this field is used to store your own username

**IPAddress & Port** – Used to store the InetAddress and port number for private messaging / P2P connection.

**Header –** Below is the table of headers that the respective client or server would expect to receive and parse.

## **CLIENT**

|  |  |  |
| --- | --- | --- |
| **Header** | **Description** | **Action** |
| Login/pass | User has successfully logged in | Spawns a thread that reads STDIN and sends data to the server |
| Login/fail/retry | Correct username but wrong password was entered | User is prompted to re-enter their password |
| Login/fail/user | Username does not exist in “credentials.txt” | User is prompted to re-enter their username and password |
| Logout/user | User has either logged out, timed out or been blocked by the server for failed password entries | All threads and socket connections to the user is closed. Program is terminated |
| Msg/user | Default heading for all standard messages between client & server | Do nothing (as package content is already printed out) |
| Private/connect | User accepts any incoming socket connections for peer to peer messaging | ServerSocket accepts connection. Spawns a new thread that listens for messages on that connection |
| Private/start | User starts connection with peer | Server returns packet with IP and port number of peer. Client creates a new socket with those parameters |

## SERVER

|  |  |  |
| --- | --- | --- |
| **Header** | **Description** | **Action** |
| User/authenticate | User wants to login | Checks username and password to ensure user exists and password is correct |
| User/broadcast | User wants to broadcast a message | Goes through list of all logged in users and sends them the user’s message |
| User/msg | User wants to send a message to another user | Finds output stream of other user and sends them the message if not blocked |
| User/whoelse | User wants to see who else is logged on | Sends list of all logged in users (excluding current user) back to the user |
| User/whoselsesince | User wants to see who else is logged on within the last ‘X’ seconds | Sends list of all logged in users within last ‘X’ seconds (excluding current user) back to user |
| User/block | User wants to block ‘X’ user | Adds user to ‘X’ users list of users that have blocked him/her |
| User/unblock | User wants to unblock ‘X’ user | Removes user from ‘X’ users list of users that have blocked him/her |
| User/logout | User wants to logout | Removes user from list of users loggedIn and updates other respective lists. Sends request to user to close connection |
| User/startprivate | User wants to start a private connection with ‘X’ user | Finds port number and IP address of ‘X’ user’s ServerSocket. Sends these parameters back to the User. Then sends a packet to ‘X’ user to accept the incoming connection |