Assignment 2

Client-Server Chat Room using Socket in Erlang

Manual

This project is completed using Erlang. The client-server multi-chat room is called ChatApp. There are three components

1. chatApp_server.erl
2. chatApp_listener.erl
3. chatApp_client.erl

On the **Server** side the server socket port is set to default as **8080,** but we can provide our port while starting the server. ChatApp server accepts multiple clients, and for the test purpose, localhost IP(IPv4) is used to establish a socket connection. Here server allows Users to Signup themself to the chatroom and then join the multi-user chatroom. Users can send a message privately to other users or broadcast it to one or more users using TCP protocol. Two components **chatApp_server.erl** and **chatApp_listener.erl** handles the client and client request. The **chatApp_listener** accepts the incoming connection request on a listening socket. The component handles the user action and calls server methods to provide the service as per the user action.

Similarly, the **Client**-side component i.e., **chatApp_client.erl**, Packets can be sent to the returned socket Socket using send/2. Packets sent from the peer are delivered as messages (unless {active, false} is specified in the options list for the listening socket, in which case packets are retrieved by calling recv/1).

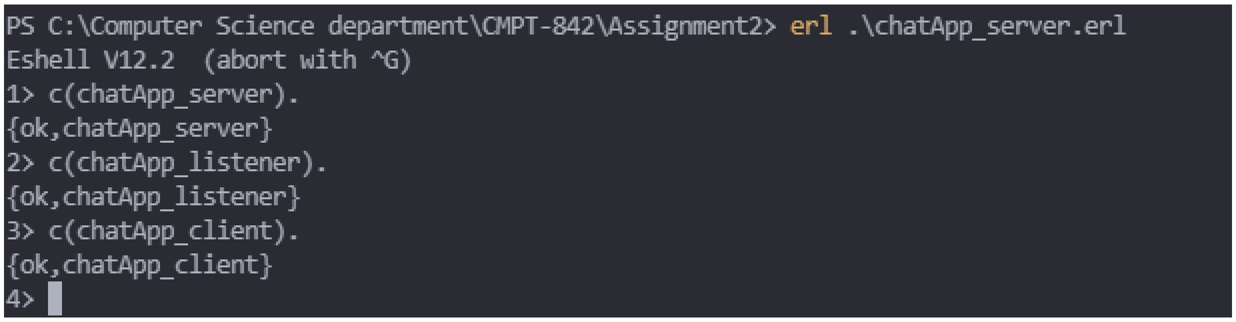
Here the Client/User connects to the server port and can sign up in the chatroom. The server will be displaying the list of active Users' connections and their respective ports. These ports are the channel to send messages to particular users and even broadcast messages to all the active users.

Steps to Run the chatApp.

1. The system needs to have Erlang installed. For the project (Erlang OTP 24) was installed and Vscode was used as an IDE.
2. As previous mentioned the three files
   a. **chatApp_server.erl**
   b. **chatApp_listener.erl**
   c. **chatApp_client.erl**

   For all of these erlang files. Their respective .beam file needs to be created. These Beam files are the executable file generated by the Erlang compiler. And these files can be run with the Erlang virtual machine (VM).

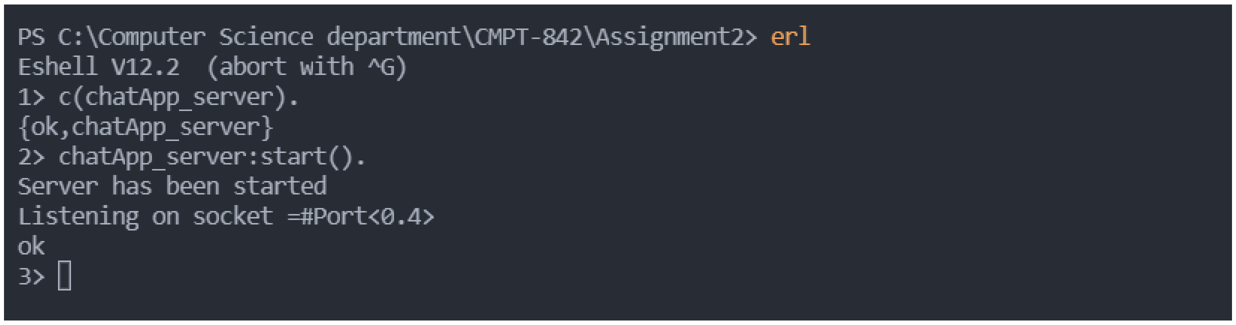
   If we compare to previous assignment 1. The .class file of java file is similar to the beam file. Here .class files contain bytecode that can run on the Java virtual machine (JVM).



```
PS C:\Computer Science department\CMPT-842\Assignment2> erl .\chatApp_server.erl
Eshell V12.2  (abort with ^G)
1> c(chatApp_server).
{ok,chatApp_server}
2> c(chatApp_listener).
{ok,chatApp_listener}
3> c(chatApp_client).
{ok,chatApp_client}
4> 
```

Fig1: - Compiling the three Erlang files and creating their Beam files.

3. The first step would be starting the server. While starting the server, we can start the server either by selecting a port or using the default one. **8080** is the default port which gets selected if no argument is passes in the method.
   Server should be active in order to establish client server chat environment. After starting the server, it would notify the listening socket port.

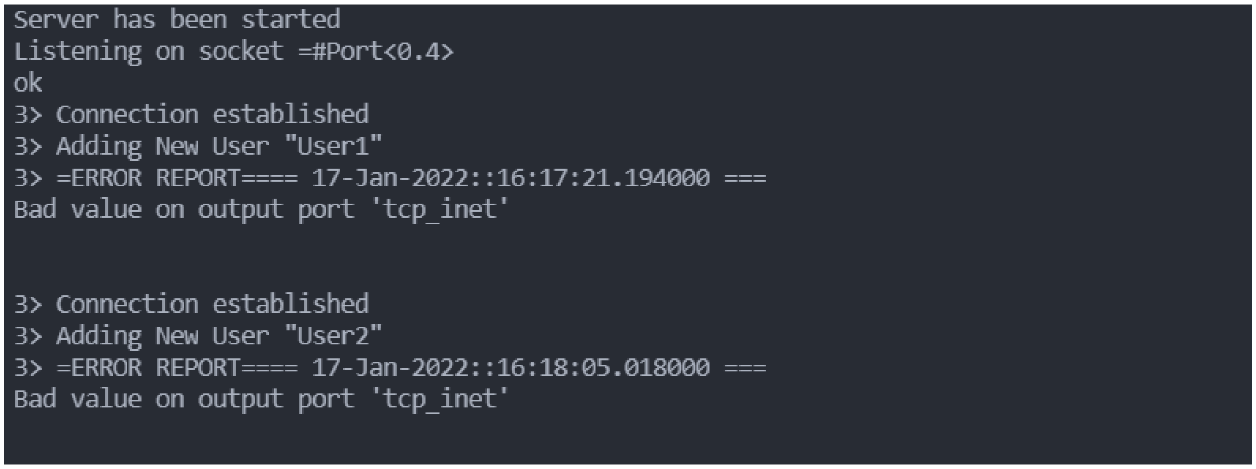


```
PS C:\Computer Science department\CMPT-842\Assignment2> erl
Eshell V12.2  (abort with ^G)
1> c(chatApp_server).
{ok,chatApp_server}
2> chatApp_server:start().
Server has been started
Listening on socket =#Port<0.4>
ok
3> 
```

Fig2:- Starting chatApp server.

4. After the server is good to go. We now signup clients or users to chat. The second terminal will be the interface for the user chat. Open new terminals. In the second terminal, create a port parament in order to store the port value. Then the send method is called, with parameters being port and your message. The message will contain your request and the UserId.

   Here three client consoles are created, and three users are connected to the server. User1, User2, User3 are the users. The list of connected users will be present on the server.

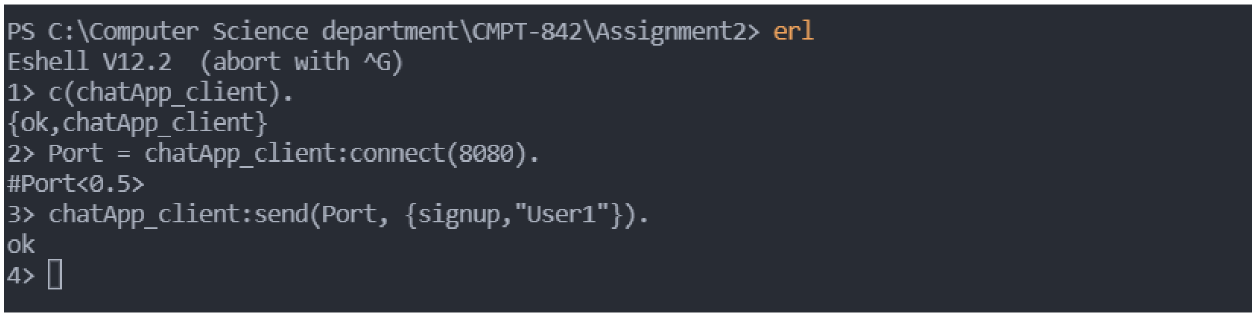


```
Server has been started
Listening on socket =#Port<0.4>
ok
3> Connection established
3> Adding New User "User1"
3> =ERROR REPORT==== 17-Jan-2022::16:17:21.194000 ===
Bad value on output port 'tcp_inet'


3> Connection established
3> Adding New User "User2"
3> =ERROR REPORT==== 17-Jan-2022::16:18:05.018000 ===
Bad value on output port 'tcp_inet'
```

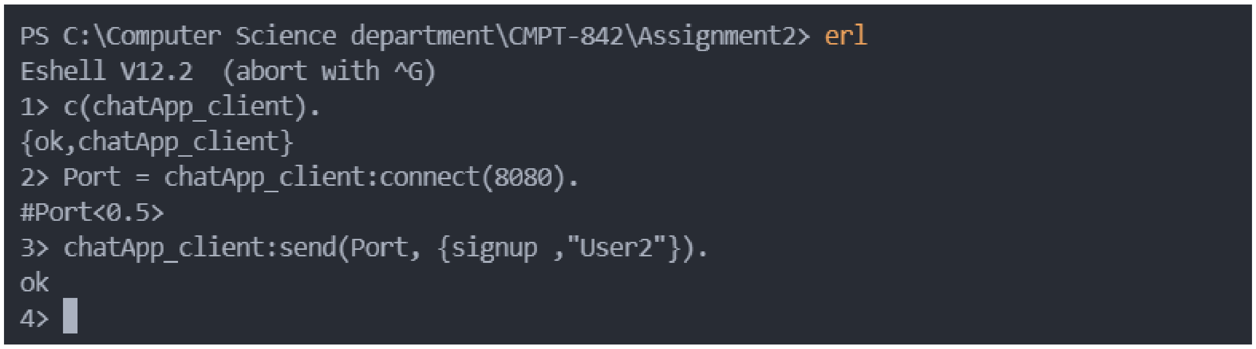
Fig3: Server response when a new user connects.

Similar in the Client-side console. We have



```
PS C:\Computer Science department\CMPT-842\Assignment2> erl
Eshell V12.2  (abort with ^G)
1> c(chatApp_client).
{ok,chatApp_client}
2> Port = chatApp_client:connect(8080).
#Port<0.5>
3> chatApp_client:send(Port, {signup,"User1"}).
ok
4> 
```

Fig: - User1 getting connected to the chat server.

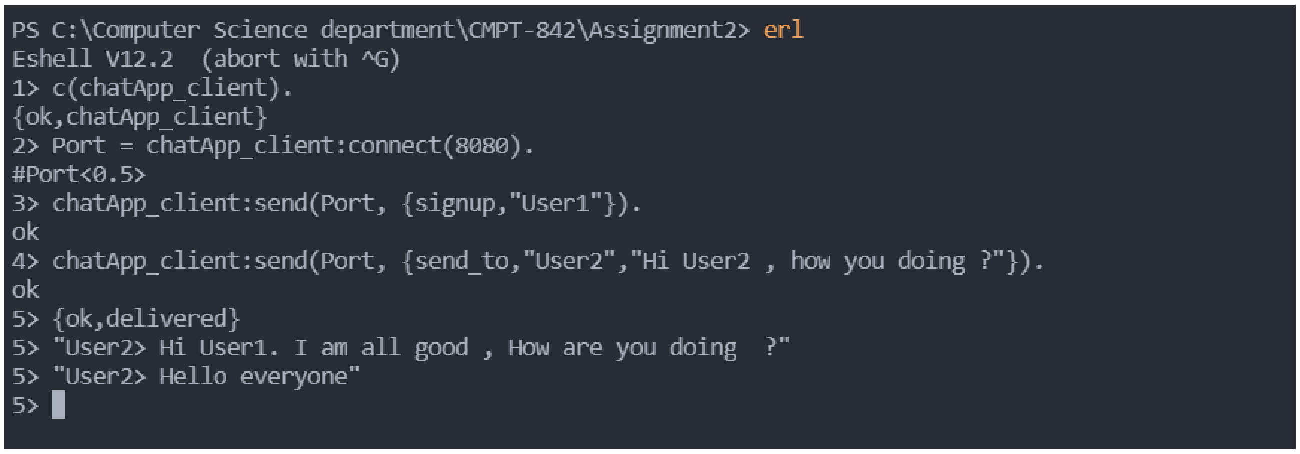


```
PS C:\Computer Science department\CMPT-842\Assignment2> erl
Eshell V12.2  (abort with ^G)
1> c(chatApp_client).
{ok,chatApp_client}
2> Port = chatApp_client:connect(8080).
#Port<0.5>
3> chatApp_client:send(Port, {signup ,"User2"}).
ok
4> 
```

Fig: - User2 getting connected to the chat server.

5. After **signup** in the chatApp, users can now send and receive a message from all the connected users. There is a feature called **broadcast** to let all active uses see the message. Users can send private messages to Users as well. If the user leaves the chat, The keywords signup and broadcast should be explicitly used in order to perform those actions.

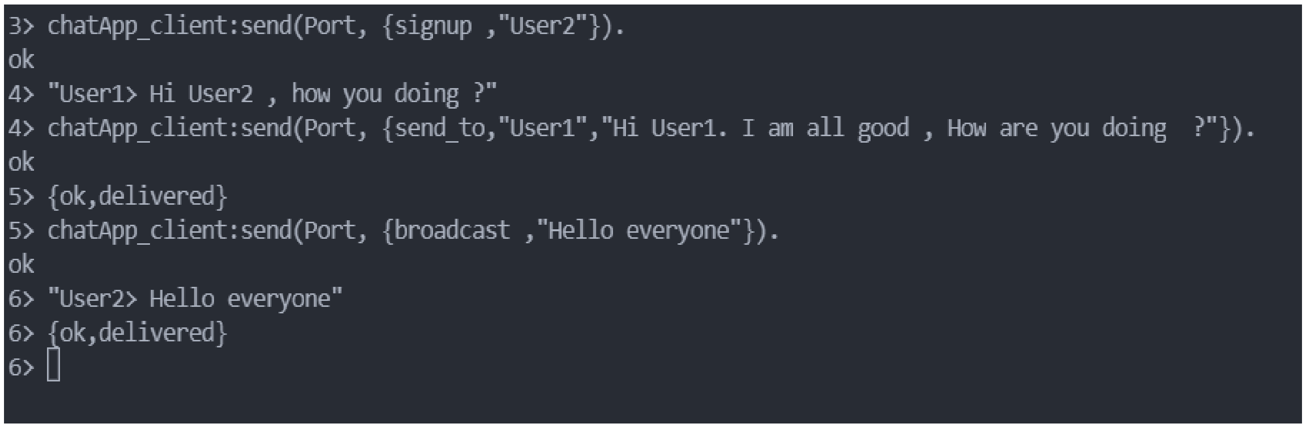
In this step, after the user has signup into the chat server. He can send messages to the active user. For example, User2 is being messaged here. The user gets notified if it is delivered or not. And When the Active user exits the chatroom, the server will get notified and broadcast it to all the currently active users of the chatroom.



```
PS C:\Computer Science department\CMPT-842\Assignment2> erl
Eshell V12.2  (abort with ^G)
1> c(chatApp_client).
{ok,chatApp_client}
2> Port = chatApp_client:connect(8080).
#Port<0.5>
3> chatApp_client:send(Port, {signup,"User1"}).
ok
4> chatApp_client:send(Port, {send_to,"User2","Hi User2 , how you doing ?"}).
ok
5> {ok,delivered}
5> "User2> Hi User1. I am all good , How are you doing  ?"
5> "User2> Hello everyone"
5> 
```

Fig: User1 Terminal

User2 will receive the message and can reply to user1 privately or even broadcast the message to all the active users.



```
3> chatApp_client:send(Port, {signup ,"User2"}).
ok
4> "User1> Hi User2 , how you doing ?"
4> chatApp_client:send(Port, {send_to,"User1","Hi User1. I am all good , How are you doing  ?"}).
ok
5> {ok,delivered}
5> chatApp_client:send(Port, {broadcast ,"Hello everyone"}).
ok
6> "User2> Hello everyone"
6> {ok,delivered}
6> 
```

Fig: - User2 Terminal