**Internet Technology Lab Report**

**Assignment 2**

Name – **Nikhil Badyal**

Roll – **001810501069**

Class – **BCSE 3rd year**

Group – **A2**

Assignment Numbers – **2**

Write a multi-client chat application consisting of both client and server programs. In this chat application simultaneously, several clients can communicate with each other. For this you need a single server program that clients connect to. The client programs send the chat text or image (input) to the server and then the server distributes that message (text or image) to all the other clients. Each client then displays the message sent to it by the server. The server should be able to handle several clients concurrently. It should work fine as clients come and go.

Develop the application using a framework based on Node.JS. How are messages handled concurrently? Which web application framework(s) did you follow? Prepare a detailed report of the experiments you have done, and your observations on performance of the system.

**DESCRIPTION & EXPERIMENTS**

* The server program is being written in JavaScript with the help of EXPRESS framework and the Node.JS framework has been used to run the program.
* To handle concurrent messages, WebSockets have been used. The following is a brief description about WebSockets:
  + The WebSocket specification defines an API establishing "socket" connections between a web browser and a server. In plain words: There is a persistent connection between the client and the server and both parties can start sending data at any time.
  + Connection limitations are no longer a problem since WebSockets represent a single TCP socket connection.
  + Cross domain communication has been considered and is dealt with within the connection handshake.
  + Using WebSocket creates a whole new usage pattern for server-side applications. While traditional server stacks such as LAMP are designed around the HTTP request/response cycle they often do not deal well with a large number of open WebSocket connections. Keeping a large number of connections open at the same time requires an architecture that receives high concurrency at a low performance cost. Such architectures are usually designed around either threading or so-called non-blocking IO.
* The client program is being written in REACT, and the inner script in REACT uses JavaScript. The client can be run on any web browser.
* The following experiments have been performed or errors has been handled:
  + Unique username – A user cannot have same client id at the same time. If a new user tries to enter an existing client id of a running client, an alert is being shown which says “Duplicate Client Username”.
  + Empty input text while sending a text or image. Users need to at least enter a character, otherwise it raises a alert showing that the input text is empty.
  + Multiple clients (multiple users) can join the chat group, wherein each user is being notified about “joining” and “leaving” of any user in the group.
  + Each client can send a text message or image. If the image has to be sent, the user should enter the complete path of the image present in the system.
  + Each client is being notified whenever an existing client sends a message (text or input), and the sender’s username is also shown before the message.
  + While pressing carriage return key on writing a text, the client adds a new line.

Hereby, a user can enter many new line-separated sentences in the same message.

**Features**

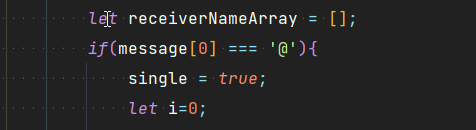
|  |  |
| --- | --- |
| Multicast | ✅ |
| Unicast | ✅ |
| Broadcast | ✅ |
| Welcome Message | ✅ |
| Left Message | ✅ |
| Encryption | ✅ |
| Image support | ✅ |
| Multiple chat group | ✅ |
| Login Page | ✅ |
| Front end Tech | React/SocketIo-Client |
| Backend Tech | Express/SocketIo |

**Broadcast/Multicast/Unicast**

By default, application does in the broadcast in their respective groups/rooms. If sender doesn’t mention anyone’s name in the messages. The message is forwarded to everyone present in that room. AND not in other rooms.

To decide among the options available. Server reads first few bits of the message. And check if it has **@**

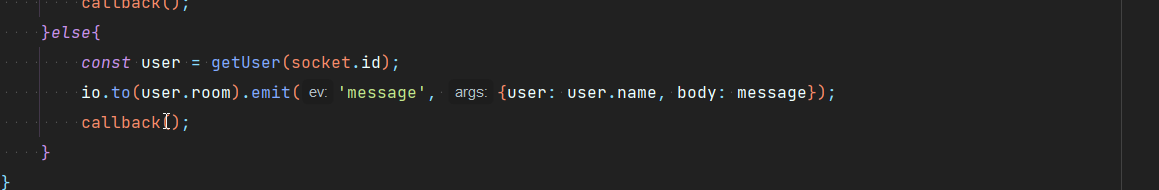
In the message. If it has, then message is not broadcasted.



Little snippet which server uses to check whether it’s a broad/multicast/unicast.

**Broadcast**

If single is false then the message is broadcasted.



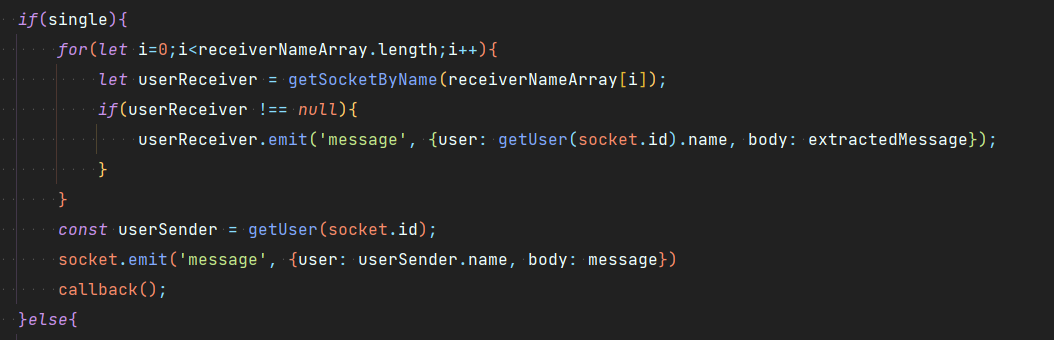
This is the calls done by the server to do the broadcast.

Unicast/Multicast



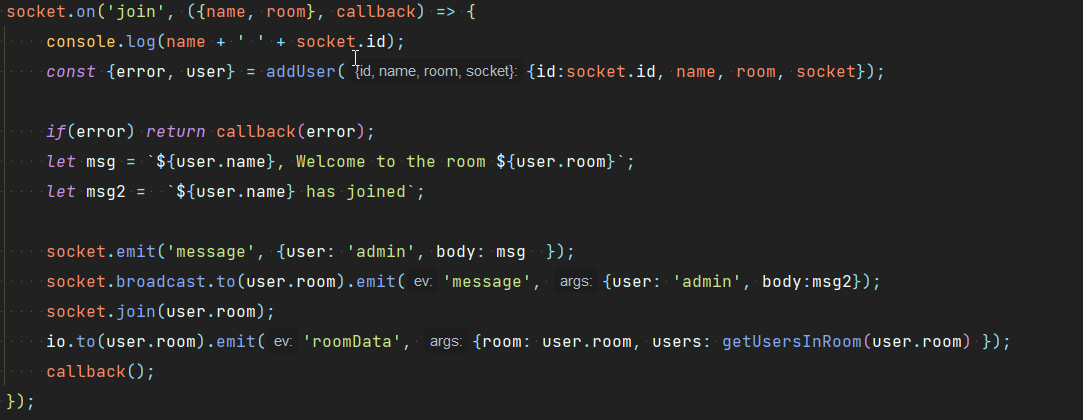
In order to do multicast/unicast. Server extract names from message and then push them to a receiver array. Now since single will be true in this case. Following actions are taken

We query the database to get the UNIQUE socket id of the interested users. And then forward that message to only these sockets.



**Welcome message**

Whenever any user logs into the system. Server acts as a bot and send WELCOME MESSAGE to the new user AND notify existing users that someone joined.



**LEFT MESSAGE**

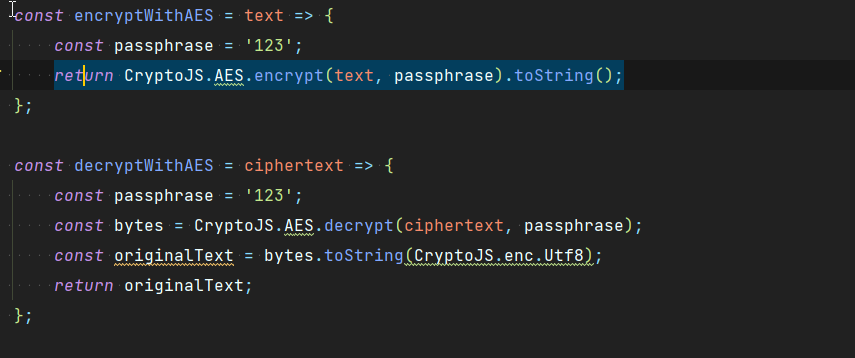
Whenever any user logs out from the system. Server acts as a bot and send USER LEFT THE CHAT message to the existing users.



**ENCRYPTION/DECRYPTION**

Whenever user send message. They are encrypted on the client side. With the CRYPTOJS library in which we use AES encryption.

In the example we are using a dummy passphrase.



**MESSAGE VALIDATION**

Most of the validation is handled itself by the SocketIO library. And we don’t need to do much in the case. Some of the validation which we handle are EMPTY MESSAGE check. With the validation check. User can’t do injection and try to run the js code on receiver side.

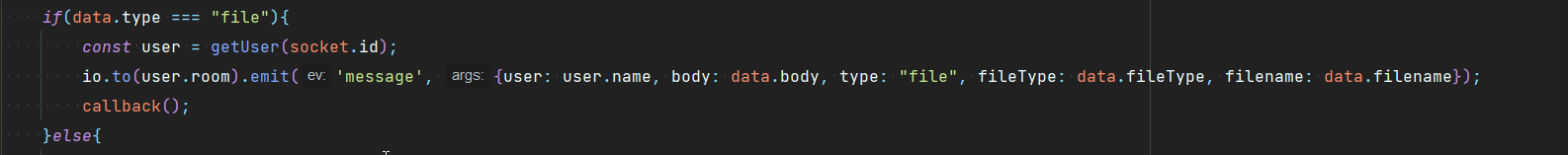
**IMAGE SUPPORT**

We have a data type field in the message body. Hence, we first check if the message is a file type. If it is then we forward it to the users.

Client Side

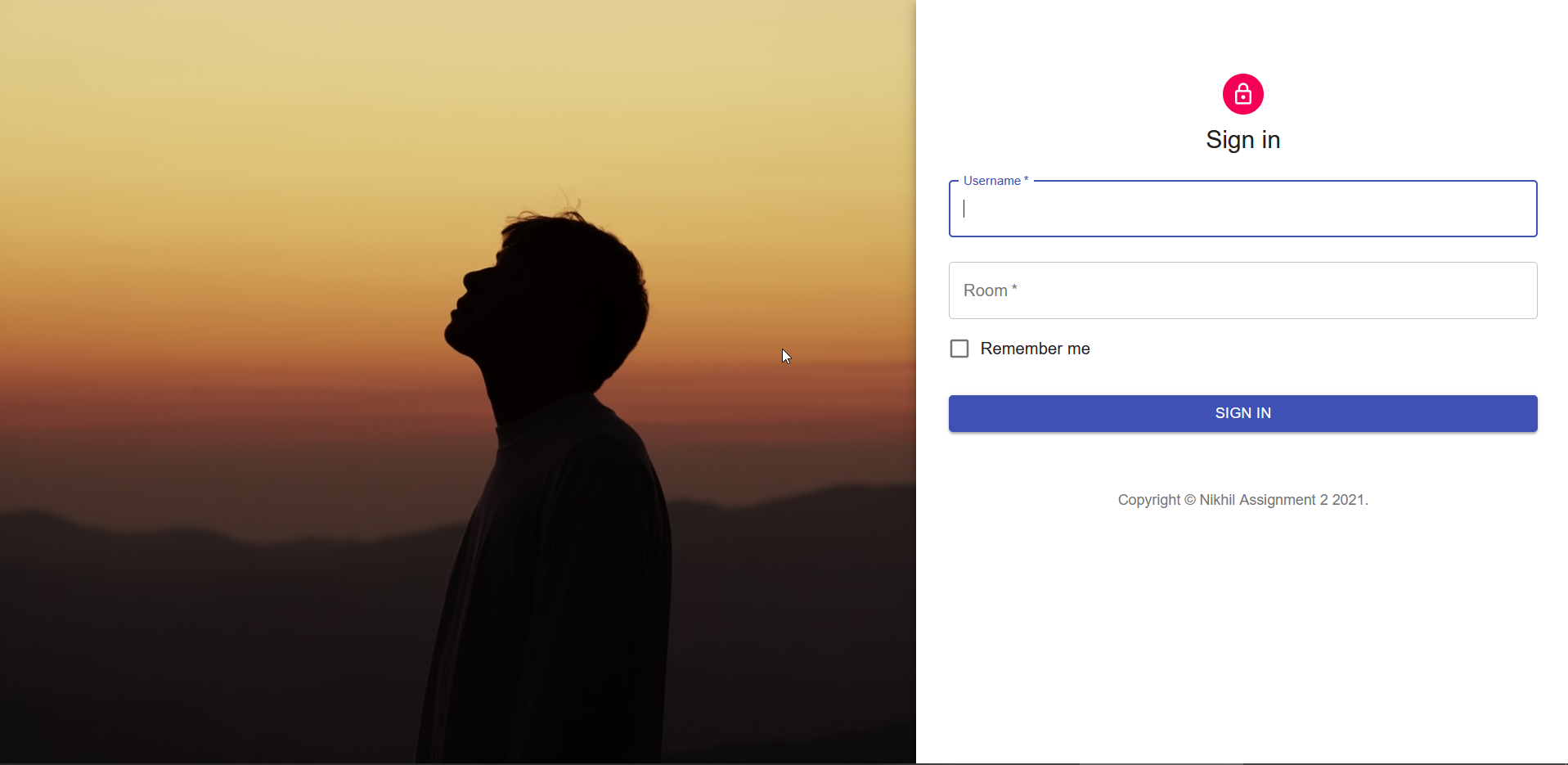
In the client side we read file as stream of bytes. And then forward that stream to server. Which server forward to the users in the room.



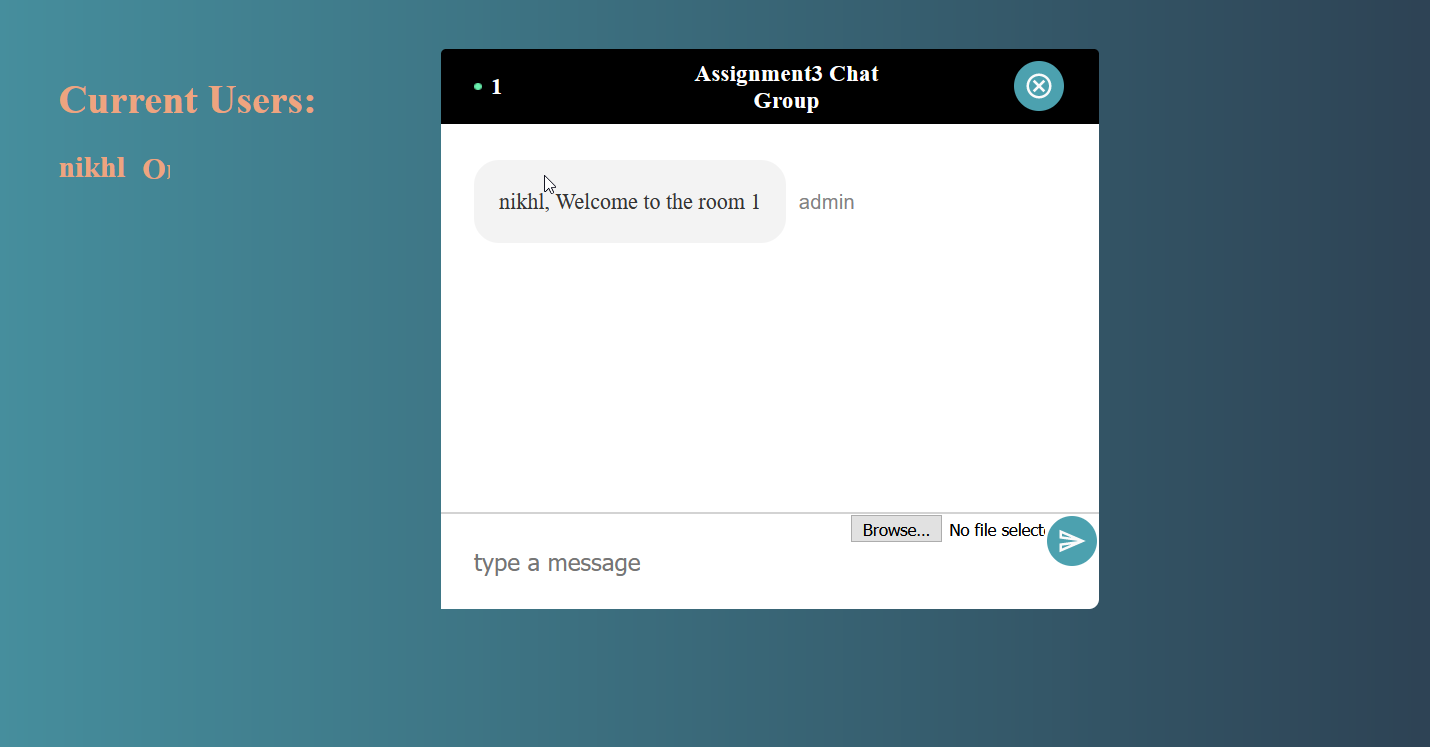


**Login Page**

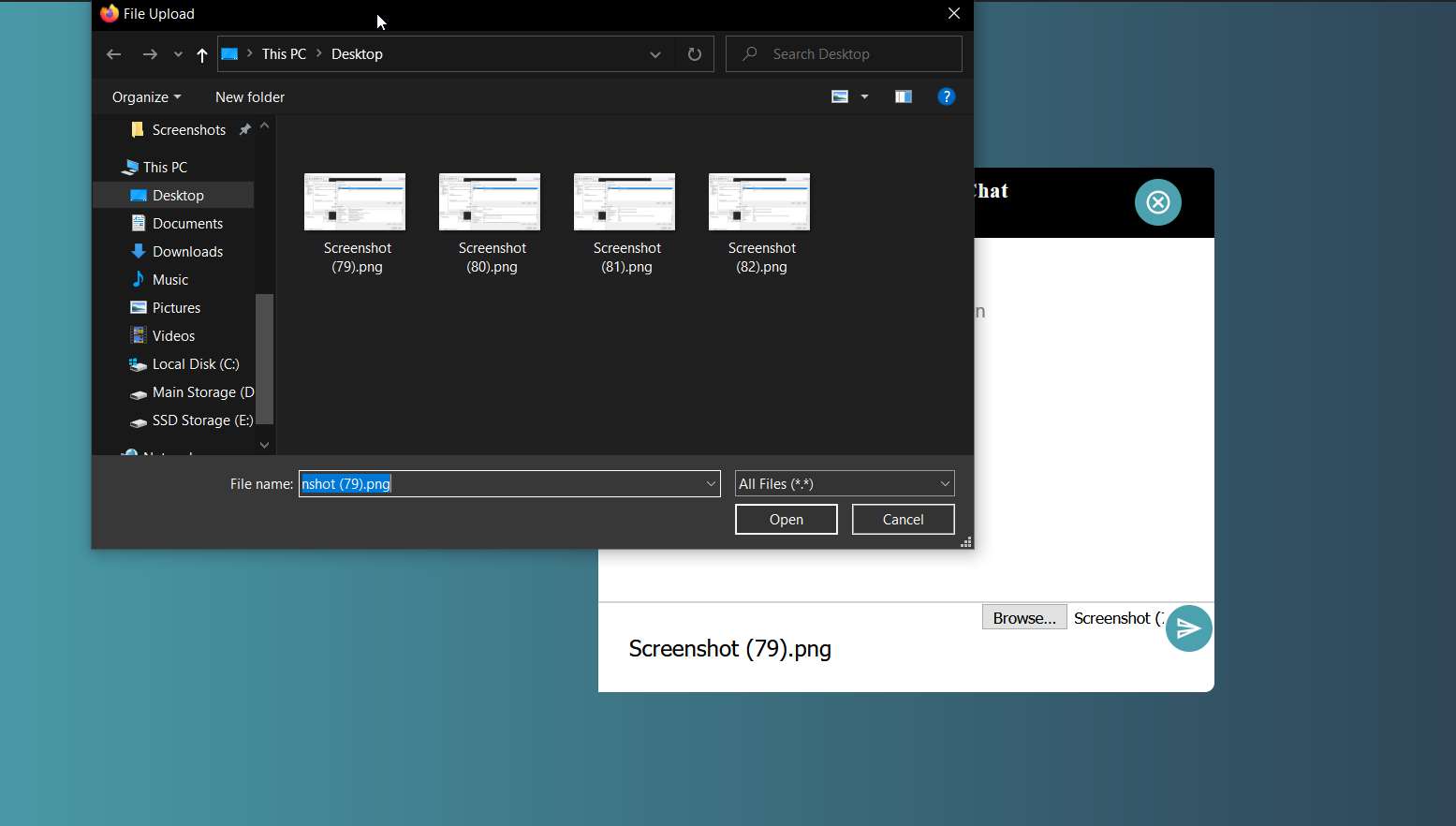
The app has a nice looking login UI. Which uses unsplash random image API, which renders new images on every log in.

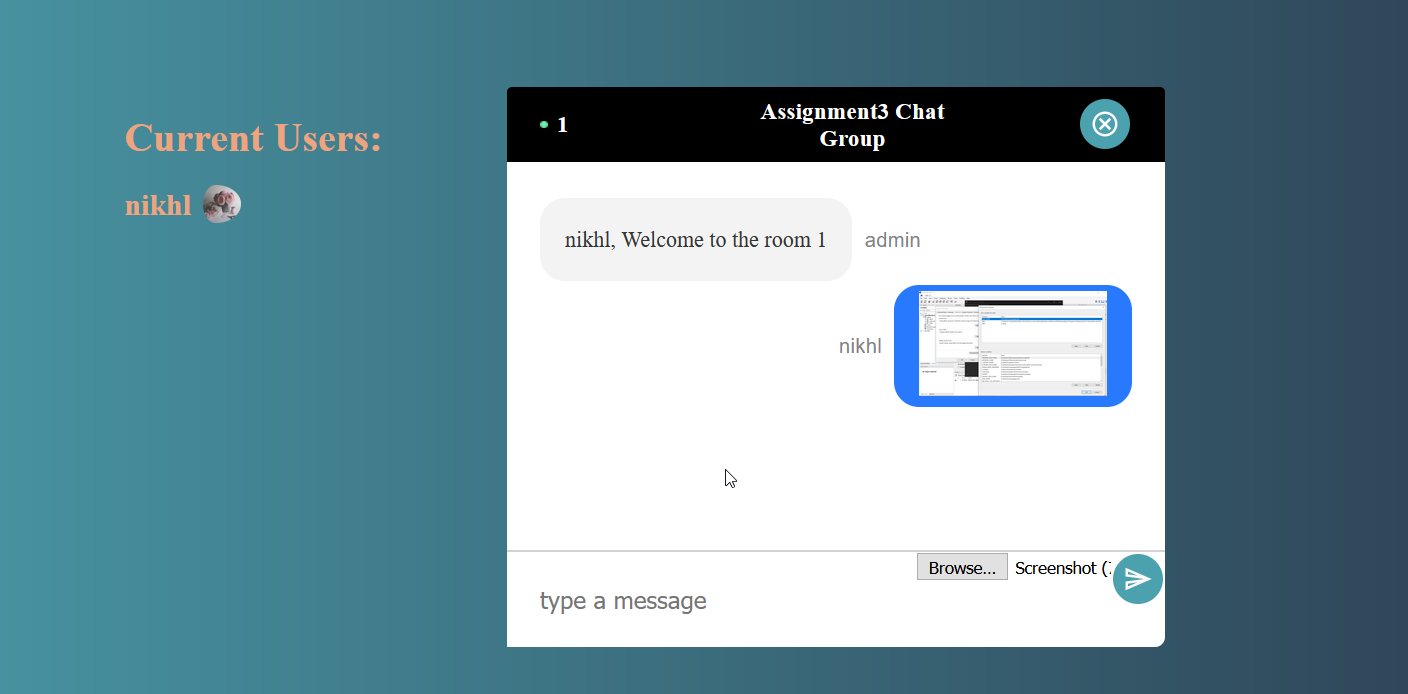


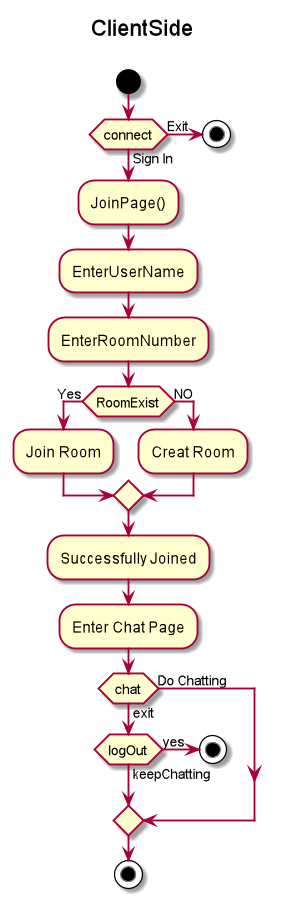
**CHAT UI**



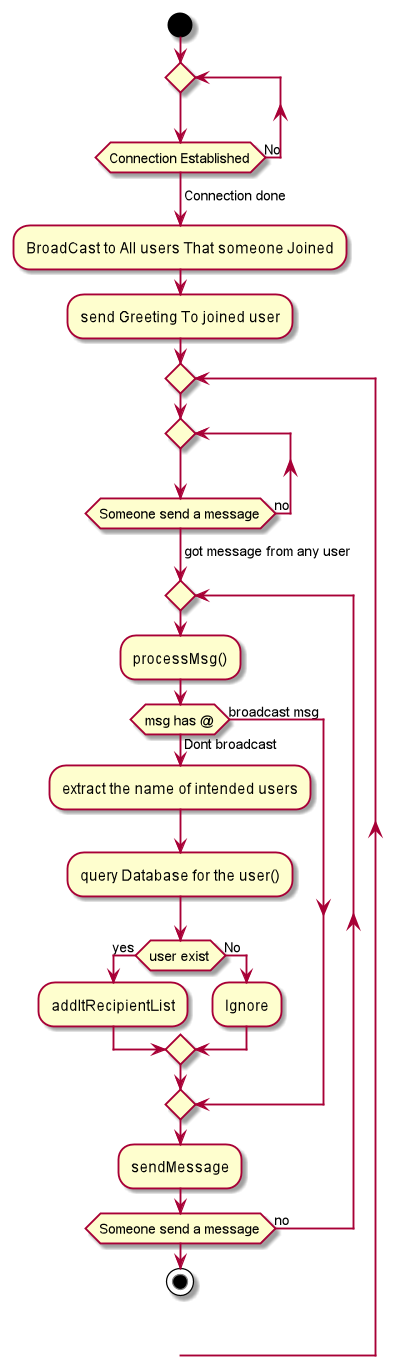
**IMAGE**







**SERVER**



**SERVER**

