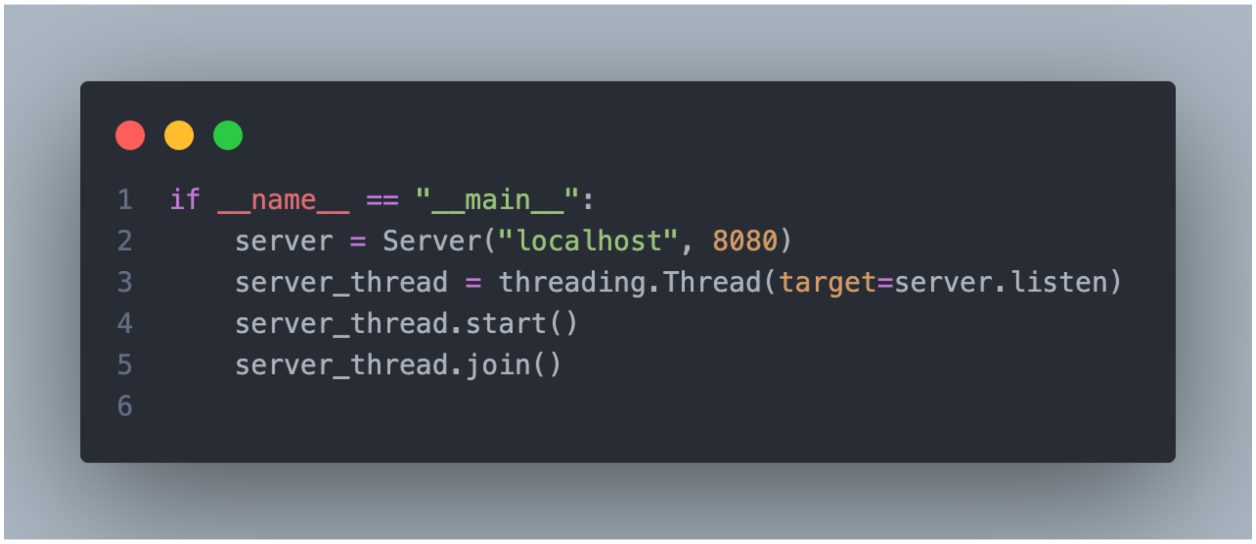
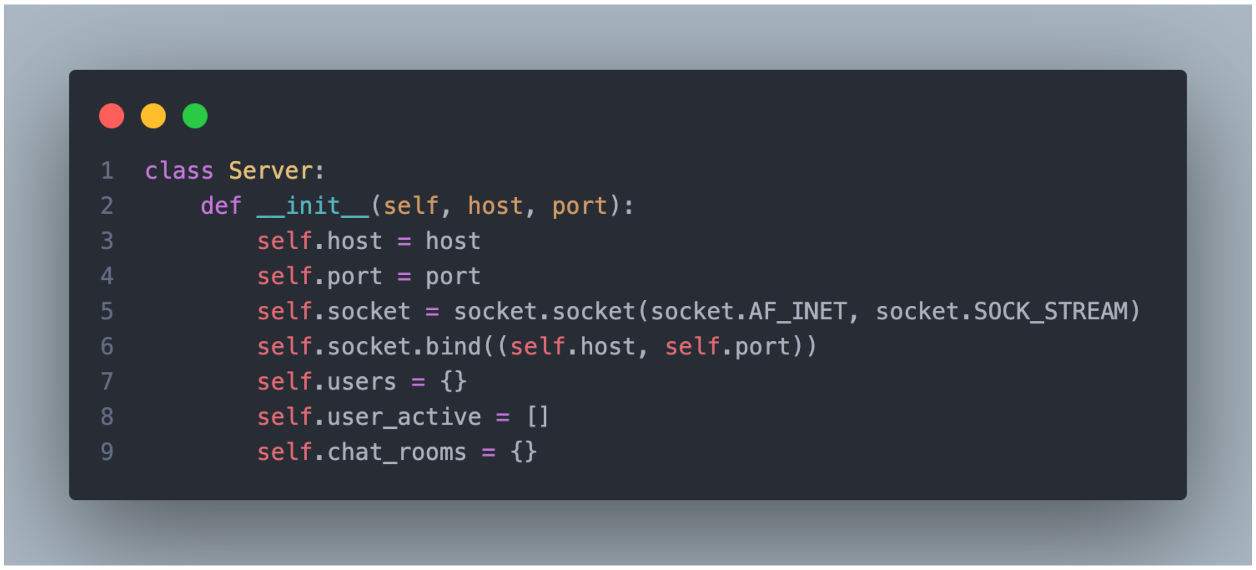
Real Time Chat Application

- This realtime chat application is based on 2 tier architecture, which consist of a client and a server. Server is also serving as a database storage tier.



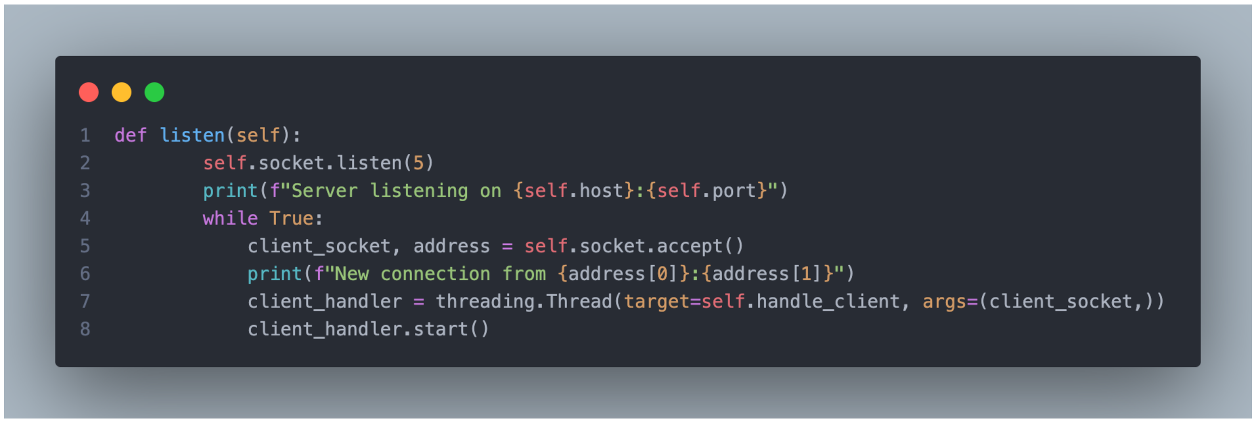
```python
if __name__ == "__main__":
    server = Server("localhost", 8080)
    server_thread = threading.Thread(target=server.listen)
    server_thread.start()
    server_thread.join()

```

- Server saves information about user authentication such as username and password, chatrooms (it’s name, members and messages), and a list which consists of active members.



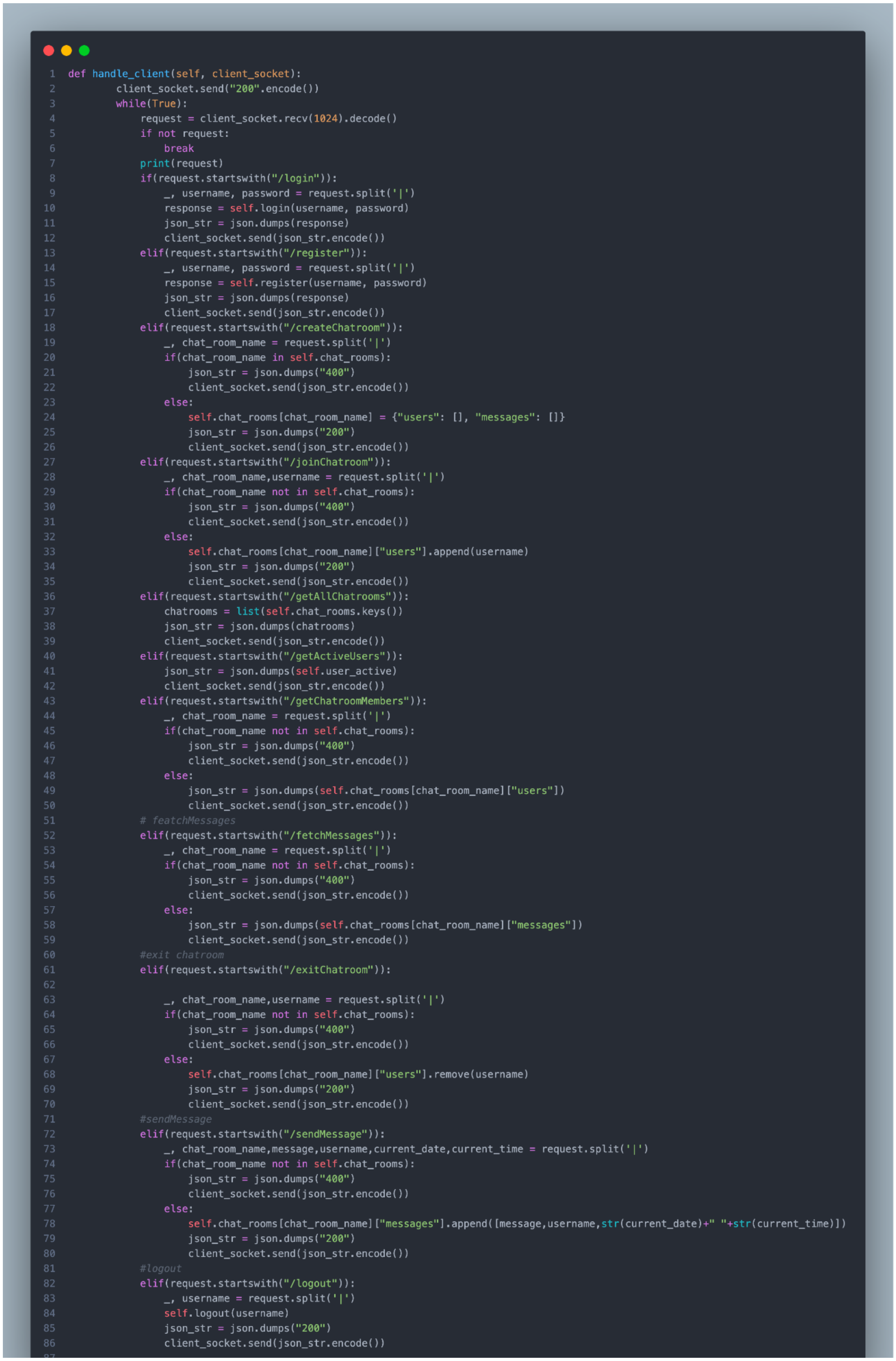
```python
class Server:
    def __init__(self, host, port):
        self.host = host
        self.port = port
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.socket.bind((self.host, self.port))
        self.users = {}
        self.user_active = []
        self.chat_rooms = {}
```

- Server continuously listens for any incoming connection requests.



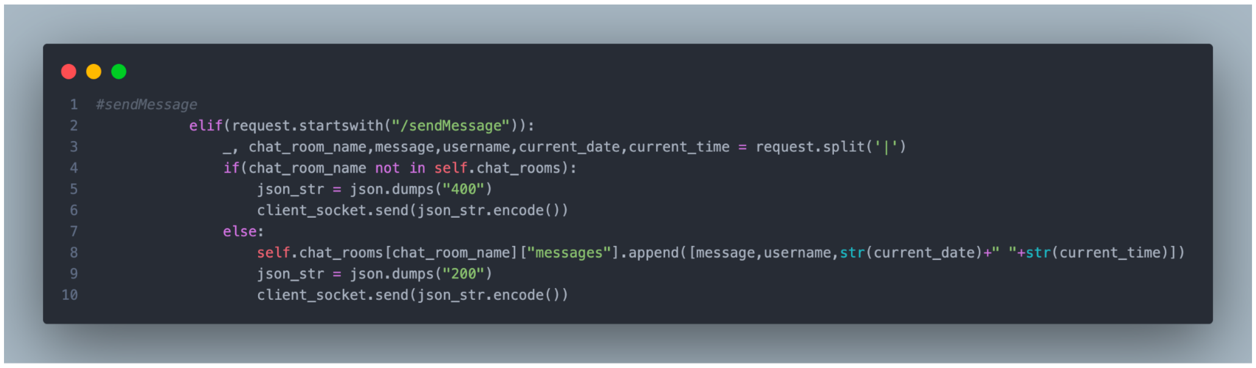
```
def listen(self):
        self.socket.listen(5)
        print(f"Server listening on {self.host}:{self.port}")
        while True:
            client_socket, address = self.socket.accept()
            print(f"New connection from {address[0]}:{address[1]}")
            client_handler = threading.Thread(target=self.handle_client, args=(client_socket,))
            client_handler.start()
```

- For each request server , it creates a new thread with the target function as “handle_client”.



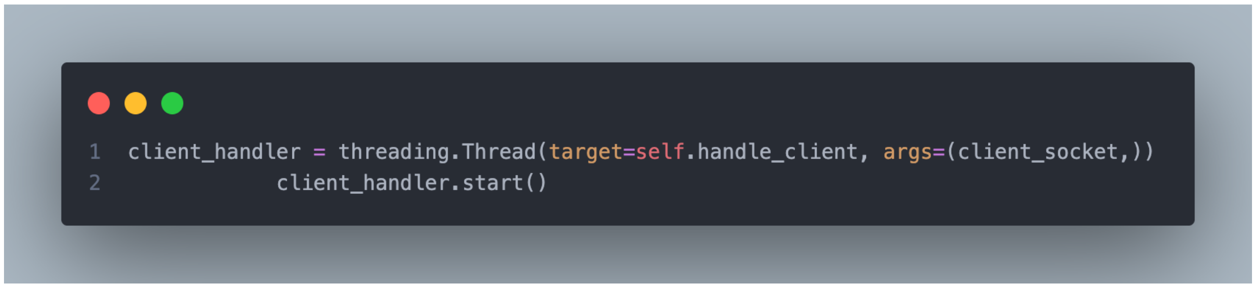
```python
def handle_client(self, client_socket):
        client_socket.send("200".encode())
        while(True):
            request = client_socket.recv(1024).decode()
            if not request:
                break
            print(request)
            if(request.startswith("/login")):
                _, username, password = request.split('|')
                response = self.login(username, password)
                json_str = json.dumps(response)
                client_socket.send(json_str.encode())
            elif(request.startswith("/register")):
                _, username, password = request.split('|')
                response = self.register(username, password)
                json_str = json.dumps(response)
                client_socket.send(json_str.encode())
            elif(request.startswith("/createChatroom")):
                _, chat_room_name = request.split('|')
                if(chat_room_name in self.chat_rooms):
                    json_str = json.dumps("400")
                    client_socket.send(json_str.encode())
                else:
                    self.chat_rooms[chat_room_name] = {"users": [], "messages": []}
                    json_str = json.dumps("200")
                    client_socket.send(json_str.encode())
            elif(request.startswith("/joinChatroom")):
                _, chat_room_name,username = request.split('|')
                if(chat_room_name not in self.chat_rooms):
                    json_str = json.dumps("400")
                    client_socket.send(json_str.encode())
                else:
                    self.chat_rooms[chat_room_name]["users"].append(username)
                    json_str = json.dumps("200")
                    client_socket.send(json_str.encode())
            elif(request.startswith("/getAllChatrooms")):
                chatrooms = list(self.chat_rooms.keys())
                json_str = json.dumps(chatrooms)
                client_socket.send(json_str.encode())
            elif(request.startswith("/getActiveUsers")):
                json_str = json.dumps(self.user_active)
                client_socket.send(json_str.encode())
            elif(request.startswith("/getChatroomMembers")):
                _, chat_room_name = request.split('|')
                if(chat_room_name not in self.chat_rooms):
                    json_str = json.dumps("400")
                    client_socket.send(json_str.encode())
                else:
                    json_str = json.dumps(self.chat_rooms[chat_room_name]["users"])
                    client_socket.send(json_str.encode())
            # featchMessages
            elif(request.startswith("/fetchMessages")):
                _, chat_room_name = request.split('|')
                if(chat_room_name not in self.chat_rooms):
                    json_str = json.dumps("400")
                    client_socket.send(json_str.encode())
                else:
                    json_str = json.dumps(self.chat_rooms[chat_room_name]["messages"])
                    client_socket.send(json_str.encode())
            #exit chatroom
            elif(request.startswith("/exitChatroom")):

                _, chat_room_name,username = request.split('|')
                if(chat_room_name not in self.chat_rooms):
                    json_str = json.dumps("400")
                    client_socket.send(json_str.encode())
                else:
                    self.chat_rooms[chat_room_name]["users"].remove(username)
                    json_str = json.dumps("200")
                    client_socket.send(json_str.encode())
            #sendMessage
            elif(request.startswith("/sendMessage")):
                _, chat_room_name,message,username,current_date,current_time = request.split('|')
                if(chat_room_name not in self.chat_rooms):
                    json_str = json.dumps("400")
                    client_socket.send(json_str.encode())
                else:
                    self.chat_rooms[chat_room_name]["messages"].append([message,username,str(current_date)+" "+str(current_time)])
                    json_str = json.dumps("200")
                    client_socket.send(json_str.encode())
            #logout
            elif(request.startswith("/logout")):
                _, username = request.split('|')
                self.logout(username)
                json_str = json.dumps("200")
                client_socket.send(json_str.encode())
```

- I designed architecture to mimic the way client-server works, similar to that of nodeJS server and any front end communicates.
- I have created various apis on server side, and server checks if the request start's with that request for example "/sendMessage, it will split the request with delimiter '|' , and get the required components to process the request, and send appropriate response code.



```
#sendMessage
            elif(request.startswith("/sendMessage")):
                _, chat_room_name,message,username,current_date,current_time = request.split('|')
                if(chat_room_name not in self.chat_rooms):
                    json_str = json.dumps("400")
                    client_socket.send(json_str.encode())
                else:
                    self.chat_rooms[chat_room_name]["messages"].append([message,username,str(current_date)+" "+str(current_time)])
                    json_str = json.dumps("200")
                    client_socket.send(json_str.encode())
```

- handle _client function inside server, handles the client independently through each thread.



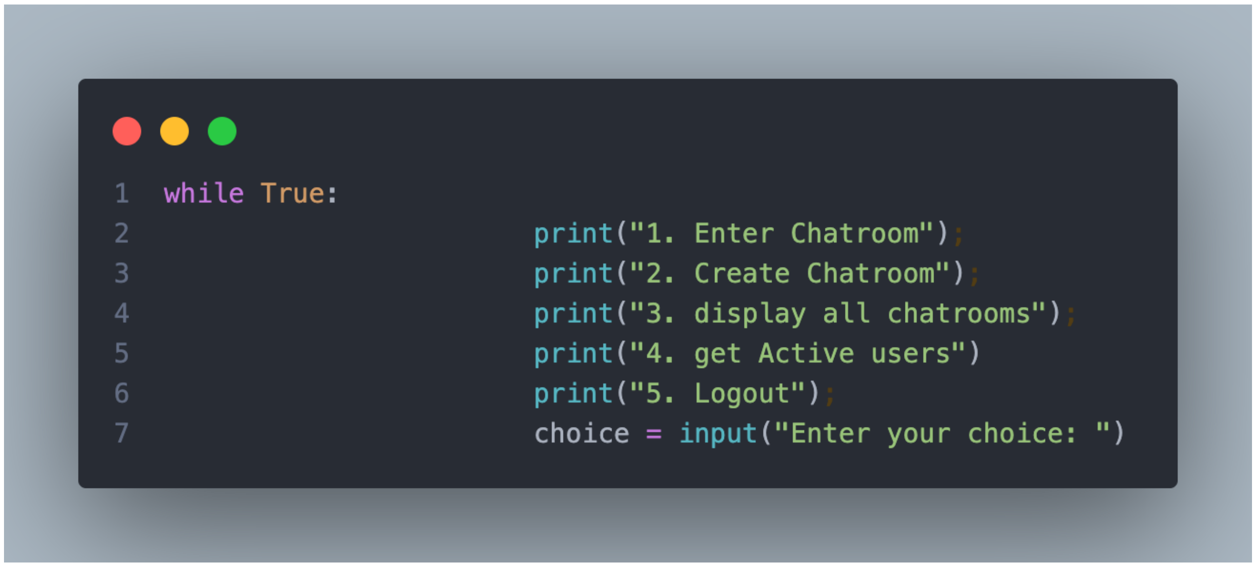
```
client_handler = threading.Thread(target=self.handle_client, args=(client_socket,))
            client_handler.start()
```

- Handle_client function listens for requests, and responds accordingly as it consists of series of if-else statements that branches the request to appropriate response.

User flow:

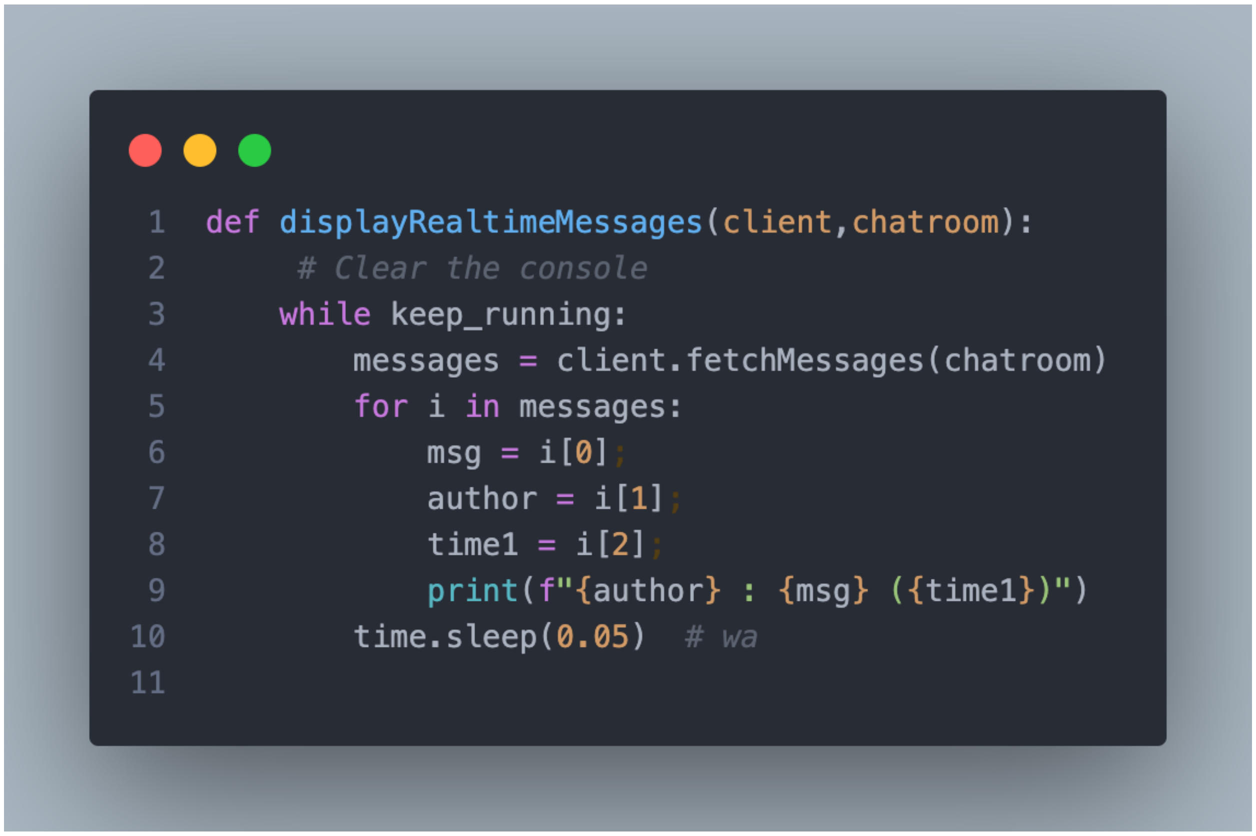
- User has to first register in the application.

- Once done, user will be redirected to main menu, where user can login with the registered credentials.
- Once successfully logged in, user will get list of active users, which user can request any number of time.



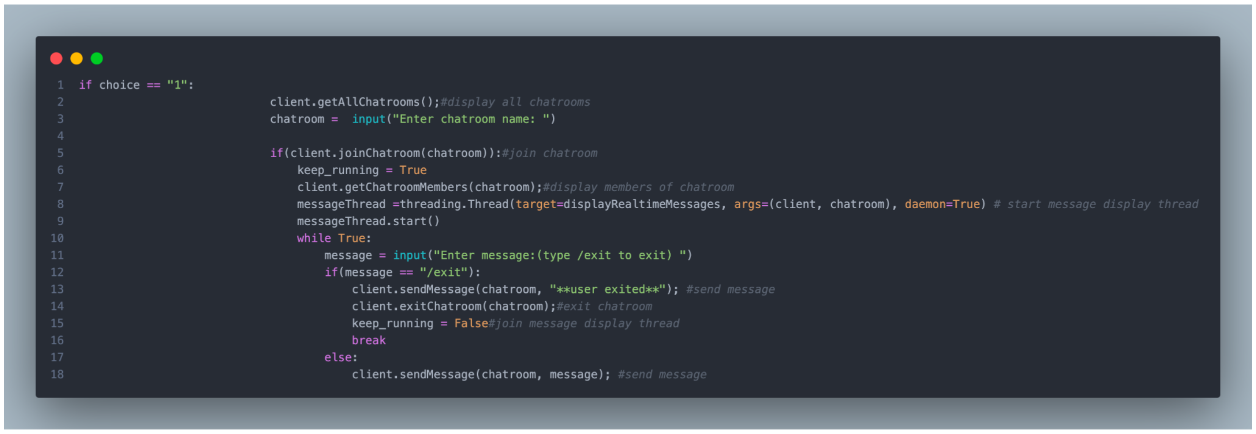
```python
while True:
                            print("1. Enter Chatroom");
                            print("2. Create Chatroom");
                            print("3. display all chatrooms");
                            print("4. get Active users")
                            print("5. Logout");
                            choice = input("Enter your choice: ")
```

- User can create a chatroom or join one.
- After successful joining , the client will make a backend api request to fetch messages in every 0.05 seconds. This is handled by multithreading. This gives the illusion of "realtime".



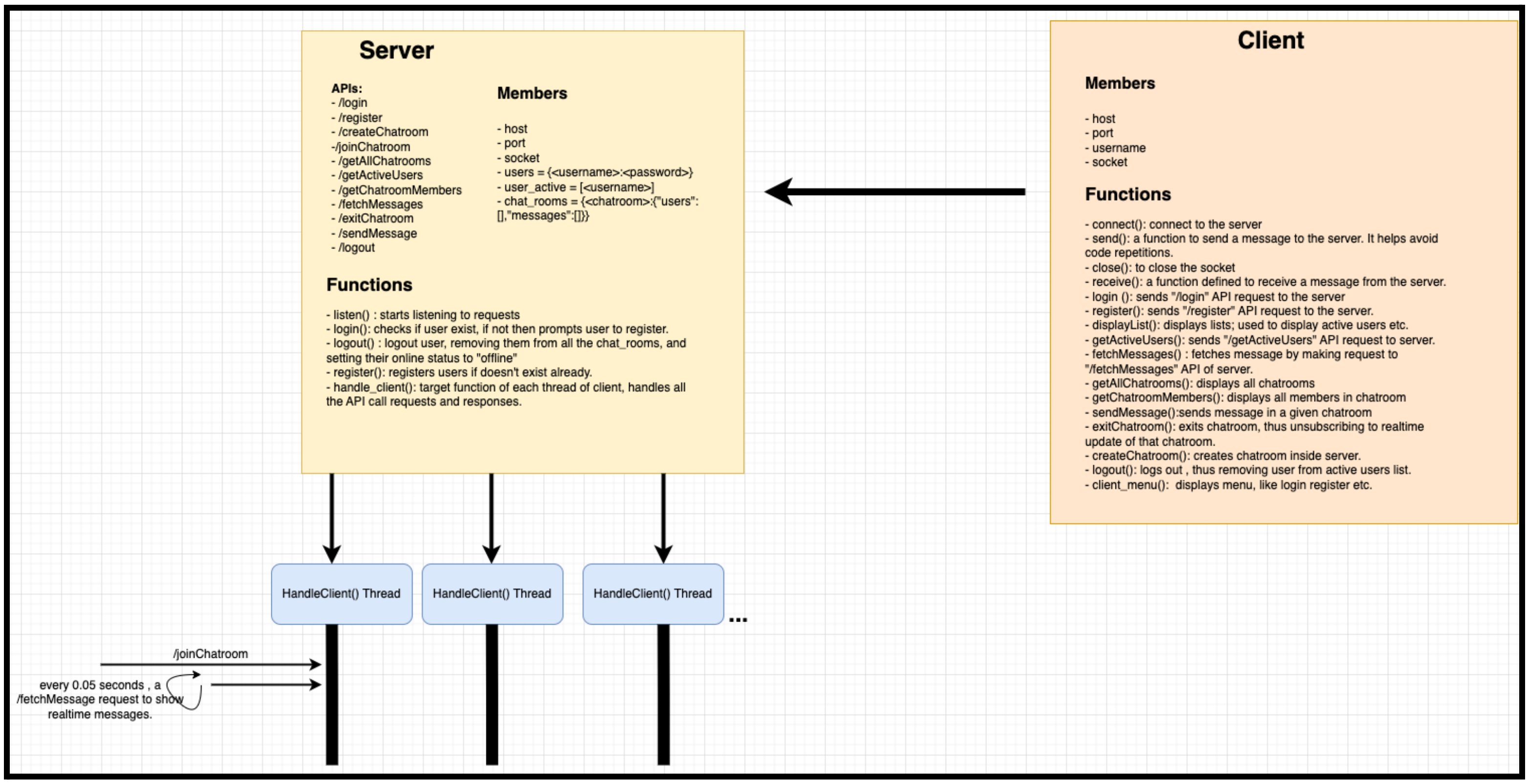
```python
def displayRealtimeMessages(client,chatroom):
     # Clear the console
    while keep_running:
        messages = client.fetchMessages(chatroom)
        for i in messages:
            msg = i[0];
            author = i[1];
            time1 = i[2];
            print(f"{author} : {msg} ({time1})")
        time.sleep(0.05)  # wa

```

- If joined, to exit the chatroom, user has to enter “/exit” in his/her message bar.



```
if choice == "1":
                            client.getAllChatrooms();#display all chatrooms
                            chatroom =  input("Enter chatroom name: ")

                            if(client.joinChatroom(chatroom)):#join chatroom
                                keep_running = True
                                client.getChatroomMembers(chatroom);#display members of chatroom
                                messageThread =threading.Thread(target=displayRealtimeMessages, args=(client, chatroom), daemon=True) # start message display thread
                                messageThread.start()
                                while True:
                                    message = input("Enter message:(type /exit to exit) ")
                                    if(message == "/exit"):
                                        client.sendMessage(chatroom, "**user exited**"); #send message
                                        client.exitChatroom(chatroom);#exit chatroom
                                        keep_running = False#join message display thread
                                        break
                                    else:
                                        client.sendMessage(chatroom, message); #send message
```

Architecture



Server

APIs:
- /login
- /register
- /createChatroom
-/joinChatroom
- /getAllChatrooms
- /getActiveUsers
- /getChatroomMembers
- /fetchMessages
- /exitChatroom
- /sendMessage
- /logout

Members

- host
- port
- socket
- users = {<username>:<password>}
- user_active = [<username>]
- chat_rooms = {<chatroom>:{"users":[],"messages":[]}}

Functions

- listen() : starts listening to requests
- login(): checks if user exist, if not then prompts user to register.
- logout() : logout user, removing them from all the chat_rooms, and setting their online status to "offline"
- register(): registers users if doesn't exist already.
- handle_client(): target function of each thread of client, handles all the API call requests and responses.

Client

Members

- host
- port
- username
- socket

Functions

- connect(): connect to the server
- send(): a function to send a message to the server. It helps avoid code repetitions.
- close(): to close the socket
- receive(): a function defined to receive a message from the server.
- login (): sends "/login" API request to the server
- register(): sends "/register" API request to the server.
- displayList(): displays lists; used to display active users etc.
- getActiveUsers(): sends "/getActiveUsers" API request to server.
- fetchMessages() : fetches message by making request to "/fetchMessages" API of server.
- getAllChatrooms(): displays all chatrooms
- getChatroomMembers(): displays all members in chatroom
- sendMessage():sends message in a given chatroom
- exitChatroom(): exits chatroom, thus unsubscribing to realtime update of that chatroom.
- createChatroom(): creates chatroom inside server.
- logout(): logs out , thus removing user from active users list.
- client_menu(): displays menu, like login register etc.