

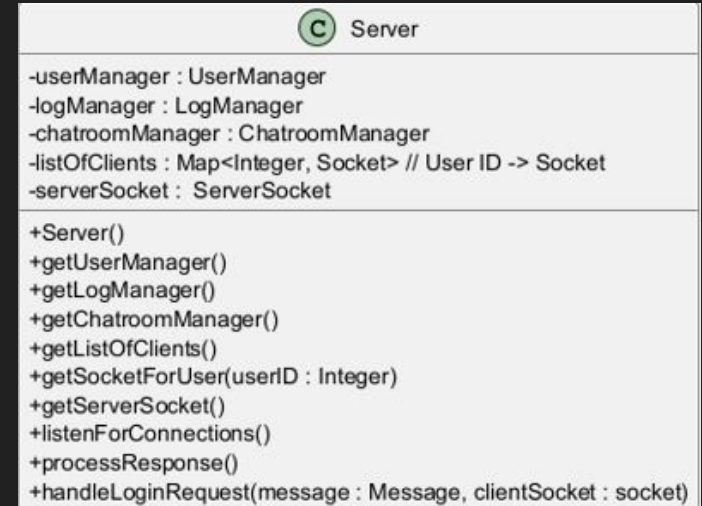
Group 7

Phase_2: WeDiscuss

String Names [] = {"Isaiah", "Joseph", "Marcos", "Roman", "Salvador"}

Class: Server

- The server's job is to listen for connections from clients and handle those incoming requests by creating a thread to process it based on the message type.
- Our design of the server keeps track of every connected client by mapping their socket to their user ID. Keeping track of this mapping aids the thread runners in functionalities that require broadcasting messages to clients other than the one who sent the original request.

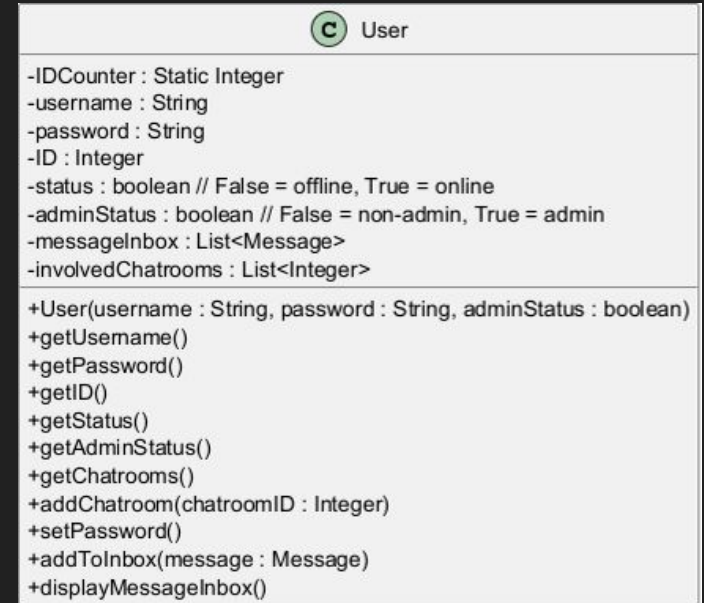


C Client
<pre> -userName : String -userID : int -messageList : List<Message> -chatrooms : Map<Integer, Chatroom> // Stores all the chatrooms user is apart of -userMap : Map<Integer, String> // All ID : Username mapping -serverConnection : Socket -toServer : ObjectOutputStream // To send messages to the server use this! +connectToServer() // Creates server socket connection & Seperate thread to listen for incoming messages from server +listenForMessages() // Seperate thread +sendLoginRequest(String user, String password) // Creates a message w/ Message Type & send to server +sendLogoutRequest() // Creates a message w/ Message Type & Send to server +sendPasswordChangeRequest(String Username, String password) +sendMessageToUser(String message, String toUserName) +sendMessageToChatroom(String message, int ChatroomId) +getChatroom(chatroomId : Integer) +getMessageLogs(userName : String) +getChatLogs(chatRoomID : Integer) +addUser(userName : String, password : String) +deleteUser(username: String) </pre>

- The Client class is responsible for managing the interaction between users and WeDiscuss' core functionalities
- These functionalities include logging in, sending messages, data retrieval, and user interface integration.
- Upon logging in, the client retrieves and stores a mapping of all the Chatrooms the user is a part of and the list of messages directed to the user.
- The Client stores a copy of the list of messages directed to the Client as well as the involved chatrooms so they can update both on their end when updates are broadcasted by the server.

Class: User

- The User class is an encapsulation of all the user data pertaining to each client of the communications system.
- Each user object is given a unique user ID for identification purposes, stores the client's login credentials, tracks the user's online and admin status, holds a list of all the messages that have been sent to the user so that asynchronous communication is possible, and holds a list of chatroom IDs that the user is involved in. The client side caches a list of involved chatrooms upon login so it is important that a lookup of every involved chatroom for a user is easy to find.



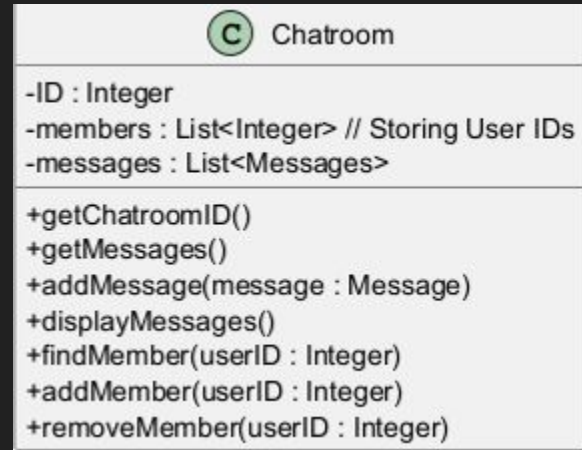
Class: UserManager

- The UserManager class is responsible for all operations that involve any user objects.
- These operations include user lookup, user authentication, an addition of a new user, any alteration of user data including the modification of passwords, the deletion of a user, or messages passed from one user to another.


C UserManager
<pre>-userList : Map<Integer, User> -userIDToUserName : Map<Integer, String> -usernameToUserID : Map<String, Integer> -userFile : String +UserManager() +handleClient(socket : Socket, message : Message, server : Server) // Pass server in so we can get List of Sockets +getUserName(userID : Integer) +getUserID(username : String) +getUserMap() +authUser(socket : Socket, message : Message) +addUser(socket : Socket, message : Message) // Contents Formatted Like : {User} {Password} +changeUserPassword(socket : Socket, message : Message) +sendMessage(fromSocket : Socket, message : Message, toSocket : Socket) +deleteUser(clientSocket : Socket, message : Message, removeSocket : Socket) +getUsers(filePath : String) +saveUsers()</pre>

Class: Chatroom

- The Chatroom class is the encapsulation of all the data that would pertain to a created chatroom in the communications system.
- This class holds the list of User IDs that have joined the chatroom and the list of messages that have been sent within the chatroom.
- Encapsulating this data makes it easier to manage user participation and message history.



Class: ChatroomManager

 ChatroomManager

-chatrooms : Map<Integer, Chatroom>

+ChatroomManager()

+handleClient(socket : Socket, message : Message, server : Server) // Pass server in so we can get List of Sockets

+sendMessageToChatroom(socket : Socket, message : Message, List<Socket>)

+getChatroomMap()

+getChatroom(chatroomId : Integer)

+addMessageToChatroom(message : Message) // This adds message to Chatroom

+addUserToChatroom(clientSocket : Socket, message : Message, AddSocket : Socket) // The "To" fields respond to User added

+createChatroom(clientSocket : Socket, message : Message)

+deleteChatroom (clientSocket : Socket, message : Message, List<Socket>)

+removeUserFromChatroom(clientSocket : Socket, message : Message, RemoveSocket : Socket) // The "To" fields correspond to user removal

- The ChatroomManager class is responsible for all operations that involve any chatroom objects.
- These operations include chatroom lookup, creation, and deletion, as well as adding or removing users from chatrooms.
- The ChatroomManager is also responsible for the broadcasting of messages to all participants within a chatroom.

Class: LogManager

 LogManager
<pre>-userMessageLogs : Map<Integer, List<Message>> // Map UserID : Messages -chatroomMessageLogs : Map<Integer, List<Message>> // Map ChatroomID: Messages -logFile : String</pre>
<pre>+LogManager() // Retrieves Logs before anything connects. Init in Server Driver +handleClient(socket : Socket, message : Message) // Each thread of LogManager needs a socket & a message +getUserMessages(userID : Integer) +getChatroomMessages(chatroomID : Integer) +answerLogRequest(message : Message) +storeMessage(message : Message) +getLogs(filePath : String) +saveLogs()</pre>

- The LogManager is responsible for anything involving data logs for messages sent by users and within chatrooms.
- Every message processed on the server-side that involves sending a message is forwarded to the LogManager, which records it in the appropriate log.
- Other than reading or writing to logs, the LogManager is also responsible for responding to any log requests from an admin on the client side.

Class: Message

- The Message class is the encapsulation of all data that is going to be sent between the client & server, as well as between the internals of the server.
- Every message is designed to be immutable with the exception of toUserID as it is possible that a message could be sent from a client without that field filled out.
- Every message is also associated with a messageType to aid the server & client in determining the correct processing mechanism to employ.

C	Message
<pre>-contents : String -dateSent : Date -toUserName : String -toUserID : int -fromUserName : String -fromUserID : int -toChatroom : int -messageIDCounter : int -messageID : int -messageType : MessageType</pre>	
<pre>+Message(messageCreator : MessageCreator) +getContents() +getDateSent() +getToUsername() +getToUserID() +getFromUsername() +getFromUserID() +getChatroom() +getMessageID() +getMessageType() +setToID(toID : Integer) // Since its possible that a message is not constructed w/ a toUserID +outputMessage(messageType : MessageType) // Different output depending on where it goes to</pre>	

Class: MessageCreator

- The MessageCreator class is responsible for building the attributes that are going to be set inside of a message object.
- These setters were separated from the Message class to emphasize and ensure the semi-immutability that the Message class is designed to have.
- To create a message, the attributes of the MessageCreator class are set and then the MessageCreator object is passed into the Message class's constructor.

C MessageCreator
<pre>-contents : String -toUserName : String -toUserID : Integer -fromUserName : String -fromUserID : Integer -toChatroom : Integer -messageType : MessageType // ENUM -user : User -chatroomMapping : Map<Integer, Chatroom> -userMap : Map<Integer, String></pre>
<pre>+MessageCreator(messageType : MessageType) +setContents(contents : String) +setToUserName(toUserName : String) +setToUserID(toUserID : Integer) +setFromUserName(fromUserName : String) +setFromUserID(fromUserID : Integer) +setToChatroom(chatroomId : Integer) +setUser(user : User) +setUserMap(userMap : Map<Integer, String>) +setChatroomMap(chatroomMap : Map<Integer, Chatroom>) +getContents() +getUserName() +getUserID() +getChatroom() +getUser() +getUserMap() +getChatroomMap() +createMessage() // returns a Message object by passing itself into the Message constructor</pre>

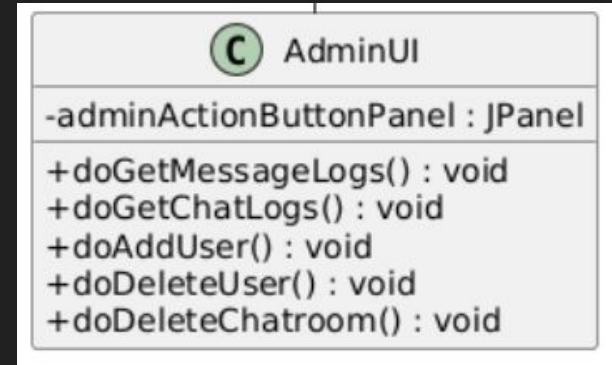
Class: ClientUI

- The ClientUI class is the base front-end user interface for the WeDiscuss application.
- Provides functionality for all users, such as logins and logouts, chatroom managing, user searches, etc.
- Refreshes objects and components throughout the activity of the application.

C ClientUI
<div><div>-mainFrame : JFrame</div><div>-menuBar : JMenuBar</div><div>-listModel : DefaultListModel<Message></div><div>-userJList : JList<Message></div><div>-chatPanel : JPanel</div><div>-profilePanel : JPanel</div><div>-imageLabel : JLabel</div><div>-profileIcon : ImageIcon</div><div>-textArea : JTextArea</div><div>-passwordField : JPasswordField</div></div>
<div><div>+processCommands()</div><div>+weDiscuss(mainFrame : JFrame) //initializer for GUI</div><div>+doLogin() : void</div><div>+doLogout() : void</div><div>+loadUsers(Map<Integer, String> userMap) : void</div><div>+loadChatrooms(Map<Integer, Chatroom> chatrooms) : void</div><div>+doChangePassword() : void</div><div>+doSearchUser() : void</div><div>+doSendMessageToUser() : void //async msg</div><div>+doSendMessageToChatroom() : void //sync msg</div><div>+doCreateChatroom() : void</div><div>+doJoinChatroom() : void</div><div>+doLeaveChatroom() : void</div><div>+selectImage(event : ActionEvent)</div></div>

Class: AdminUI

- The AdminUI class is the administrator user interface for WeDiscuss.
- Provides extra functionality from the ClientUI class, such as getting message and chatroom logs, managing users, and deleting chatrooms.



```

C Client
-username: String
-id: int
-messageList: List<Message>
-chatrooms: Map<Integer, Chatroom> // Stores all the chatrooms user is apart of
-userMap: Map<Integer, String> // All ID : Username mapping
-serverConnection: Socket
-toServer: ObjectOutputStream // To send messages to the server use this!

+connectToServer() // Creates server socket connection & Separate thread to listen for incoming messages from server
+listenForMessages() // Separate thread
+sendLoginRequest(String user, String password) // Creates a message w/ Message Type & send to server
+sendLogoutRequest() // Creates a message w/ Message Type & Send to server
+sendPasswordChangeRequest(String Username, String password)
+sendMessageToUser(String message, String toUserName)
+sendMessageToChatroom(String message, int ChatroomId)
+getChatrooms(chatroomId: Integer)
+getMessageLog(chatroomId: String)
+getChatLog(chatroomId: Integer)
+addUser(username: String, password: String)
+deleteUser(username: String)

```

```

C Server
-userManager: UserManager
-logManager: LogManager
-chatroomManager: ChatroomManager
-loggedInClients: Map<Integer, Socket> // User ID -> Socket
-serverSocket: ServerSocket

+Server()
+getUserManager()
+getLogManager()
+getChatroomManager()
+getListOfClients()
+getSocketForUser(userID: Integer)
+getServerSocket()
+listenForConnections()
+processResponse()
+handleRequest(message: Message, clientSocket: socket)

```

```

C MessageCreator
-contents: String
-toUserName: String
-toUserID: Integer
-fromUserName: String
-fromUserID: Integer
-toChatroom: Integer
-messageType: MessageType // ENUM
-user: User
-chatroomMapping: Map<Integer, Chatroom>
-userMap: Map<Integer, String>

+MessageCreator(messageType: MessageType)
+setContents(contents: String)
+setToUserName(toUserName: String)
+setToUserID(toUserID: Integer)
+setFromUserName(fromUserName: String)
+setFromUserID(fromUserID: Integer)
+setToChatroom(chatroomId: Integer)
+setUser(user: User)
+setUserMap(userMap: Map<Integer, String>)
+setChatroomMap(chatroomMap: Map<Integer, Chatroom>)
+getContents()
+getUserName()
+getUserID()
+getChatroom()
+getUser()
+getChatroomMap()
+createMessage() // returns a Message object by passing itself into the Message constructor

```

```

C ClientUI
-mainFrame: JFrame
-menuBar: JMenuBar
-listModel: DefaultListModel<Message>
-userList: JList<Message>
-chatPanel: JPanel
-profilePanel: JPanel
-imageLabel: JLabel
-profileIcon: ImageIcon
-textArea: JTextArea
-passwordField: JPasswordField

+processCommands()
+weDiscuss(mainFrame: JFrame) // initializer for GUI
+doLogin(): void
+doLogout(): void
+doLogout(): void
+loadUsers(Map<Integer, String> userMap): void
+loadChatrooms(Map<Integer, Chatroom> chatrooms): void
+doChangePassword(): void
+doSearchUser(): void
+doSendMessageToUser(): void // async msg
+doSendMessageToChatroom(): void // async msg
+doCreateChatroom(): void
+doJoinChatroom(): void
+doLeaveChatroom(): void
+doDeleteChatroom(): void
+selectImage(event: ActionEvent)

```

```

C UserManager
-userList: Map<Integer, User>
-userIDToUserName: Map<Integer, String>
-usernameToUserID: Map<String, Integer>
-userFile: String

+UserManager()
+handleClient(socket: Socket, message: Message, server: Server) // Pass server in so we can get List of Sockets
+getUserName(userID: Integer)
+getUserID(username: String)
+getUserMap()
+addUser(socket: Socket, message: Message)
+changeUserPassword(socket: Socket, message: Message)
+sendMessage(fromSocket: Socket, message: Message, toSocket: Socket)
+deleteUser(clientSocket: Socket, message: Message, removeSocket: Socket)
+getUsersFilePath(): String
+saveUsers()

```

```

E MessageType
LOGIN // Login Request
LOGOUT // Informing Server of LOGOUT
ADDUSER // Admin Add User Account Request
DELUSER // Admin Delete User Account Request
GETDEL // Message Informing User of Account Deletion
PWD // Change Password Request
GUL // Get User Log
GCL // Get Chatroom Log
JUC // Invite User to Chatroom
JC // Join Chatroom
LC // Leave Chatroom
UTU // User to User
UTC // User to Chatroom

```

```

C Message
-contents: String
-dateSent: Date
-toUserName: String
-toUserID: Integer
-fromUserName: String
-fromUserID: Integer
-toChatroom: Integer
-messageDCounter: Integer
-messageType: MessageType
-user: User
-chatroomMapping: Map<Integer, Chatroom>
-userMap: Map<Integer, String>

+Message(messageCreator: MessageCreator)
+getContents()
+getDateSent()
+getUserName()
+getFromUserName()
+getFromUserID()
+getChatroom()
+getMessageD()
+getMessageType()
+setToID(toID: Integer) // Since its possible that a message is not constructed w/ a toUserID
+outputMessage(messageType: MessageType) // Different output depending on where it goes to

```

```

C AdminUI
-adminActionButtonPanel: JPanel

+doGetMessageLogs(): void
+doGetChatLogs(): void
+doAddUser(): void
+doDeleteUser(): void

```

```

C User
-IDCounter: Static Integer
-username: String
-password: String
-ID: Integer
-status: boolean // False = offline, True = online
-adminStatus: boolean // False = non-admin, True = admin
-messageInbox: List<Message>
-involvedChatrooms: List<Integer>

+User(username: String, password: String, adminStatus: boolean)
+getUsername()
+getPassword()
+getID()
+getStatus()
+getAdminStatus()
+getChatrooms()
+addChatroom(chatroom: Chatroom)
+setPassword()
+addToInbox(message: Message)
+displayMessageInbox()

```

```

C LogManager
-userMessageLogs: Map<Integer, List<Message>> // Map UserID : Messages
-chatroomMessageLogs: Map<Integer, List<Message>> // Map ChatroomId: Messages
-logFile: String

+LogManager() // Retrieves Logs before anything connects, Init in Server Driver
+handleClient(socket: Socket, message: Message) // Each thread of LogManager needs a socket & a message
+getUserMessages(userID: Integer)
+getChatroomMessages(chatroomId: Integer)
+answerLogRequest(message: Message)
+storeMessage(message: Message)
+getLogs(filePath: String)
+saveLogs()

```

```

C ChatroomManager
-chatrooms: Map<Integer, Chatroom>

+ChatroomManager()
+handleClient(socket: Socket, message: Message, server: Server) // Pass server in so we can get List of Sockets
+sendMessageToChatroom(socket: Socket, message: Message, List<Sockets>)
+getChatroomMap()
+getChatroom(chatroomId: Integer)
+addMessageToChatroom(message: Message) // This adds message to Chatroom
+addUserToChatroom(clientSocket: Socket, message: Message, AddSocket: Socket) // The "To" fields respond to User added
+createChatroom(clientSocket: Socket, message: Message)
+deleteChatroom(clientSocket: Socket, message: Message, List<Sockets>)
+removeUserFromChatroom(clientSocket: Socket, message: Message, RemoveSocket: Socket) // The "To" fields correspond to user removal

```

```

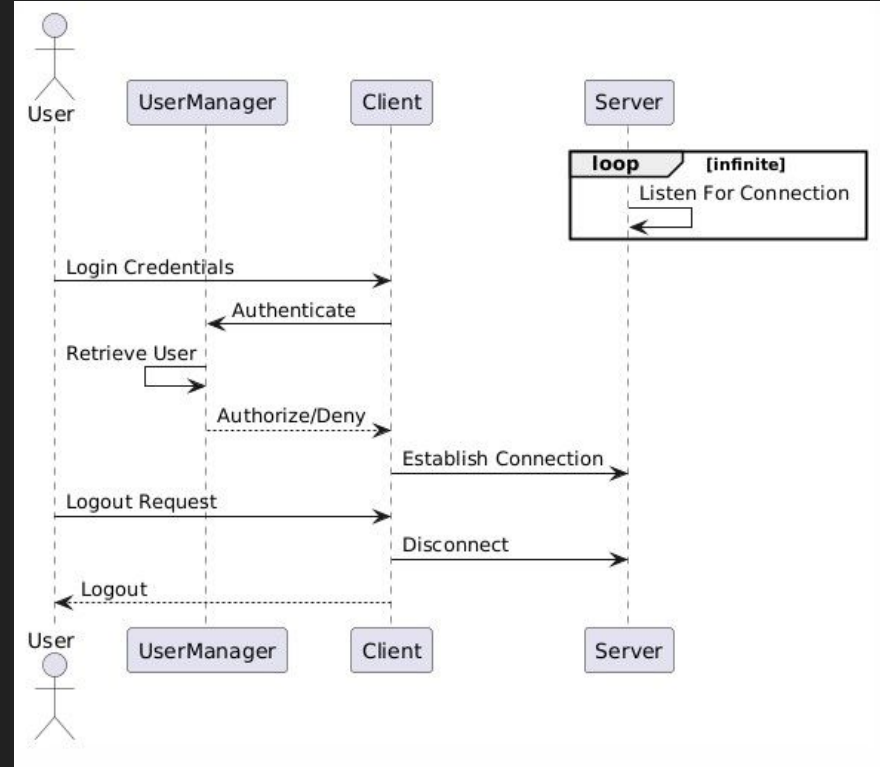
C Chatroom
-ID: Integer
-members: List<Integer> // Storing User IDs
-messages: List<Messages>

+getChatroomID()
+getMessages()
+addMessage(message: Message)
+displayMessage()
+findMember(user: User)
+addMember(user: User)
+removeMember(user: User)

```

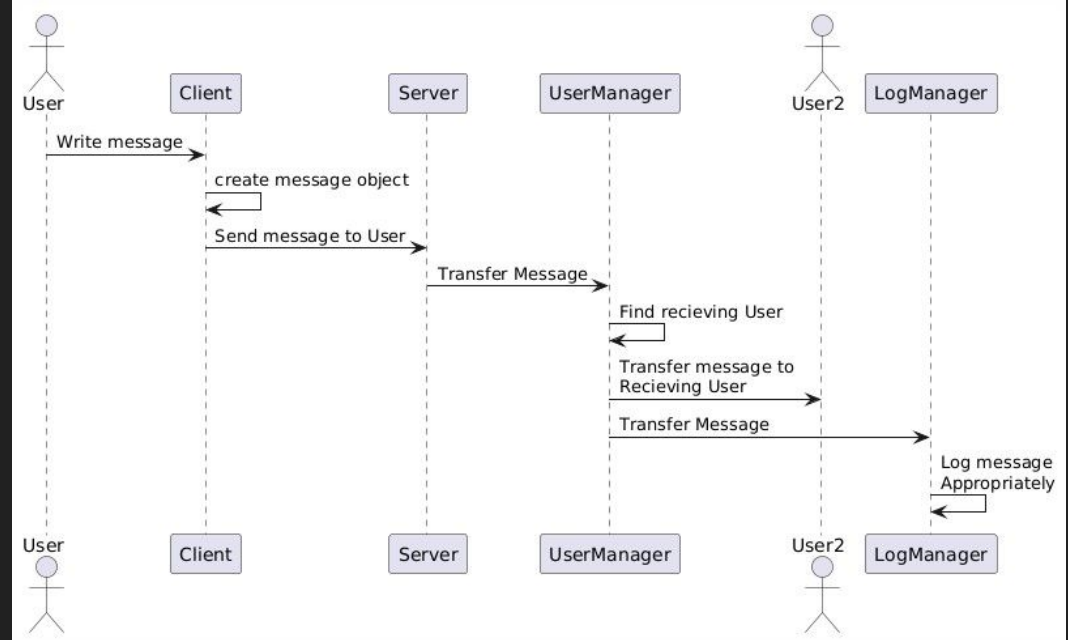
Sequence: Login and Logout

- User requests login from Client to connect to Server
- User is granted access to the application from valid credentials



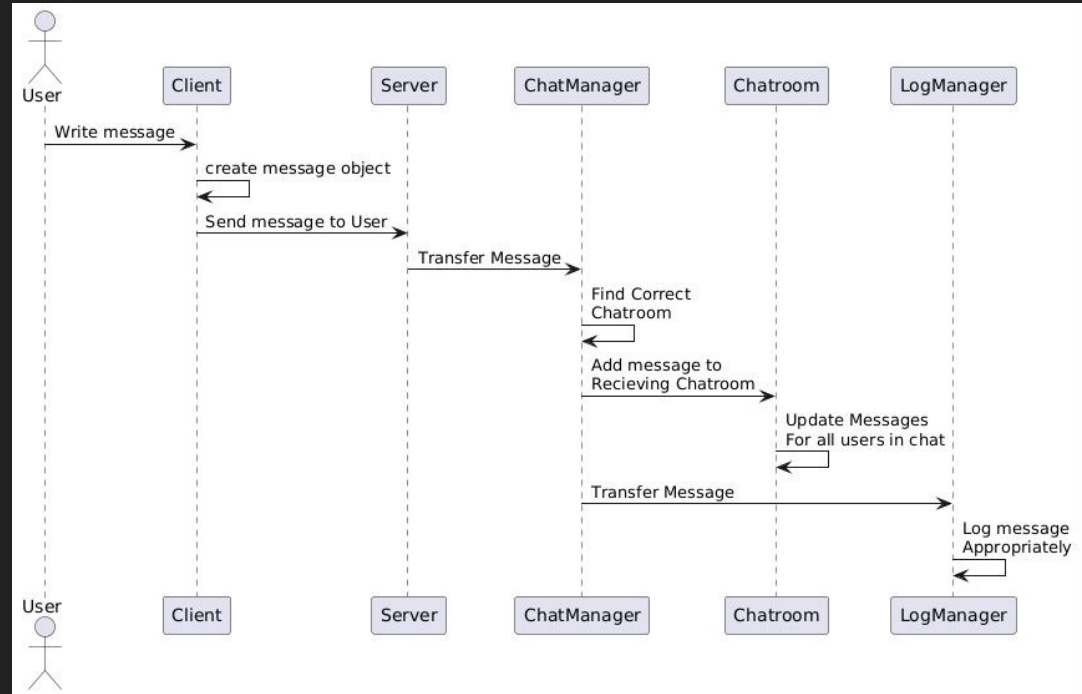
Sequence: Sending message to user

- User writes a message from the Client to send to another User.
- The Server will pass the message to the UserManager for correct handling.
- UserManager relays message to LogManager for logging



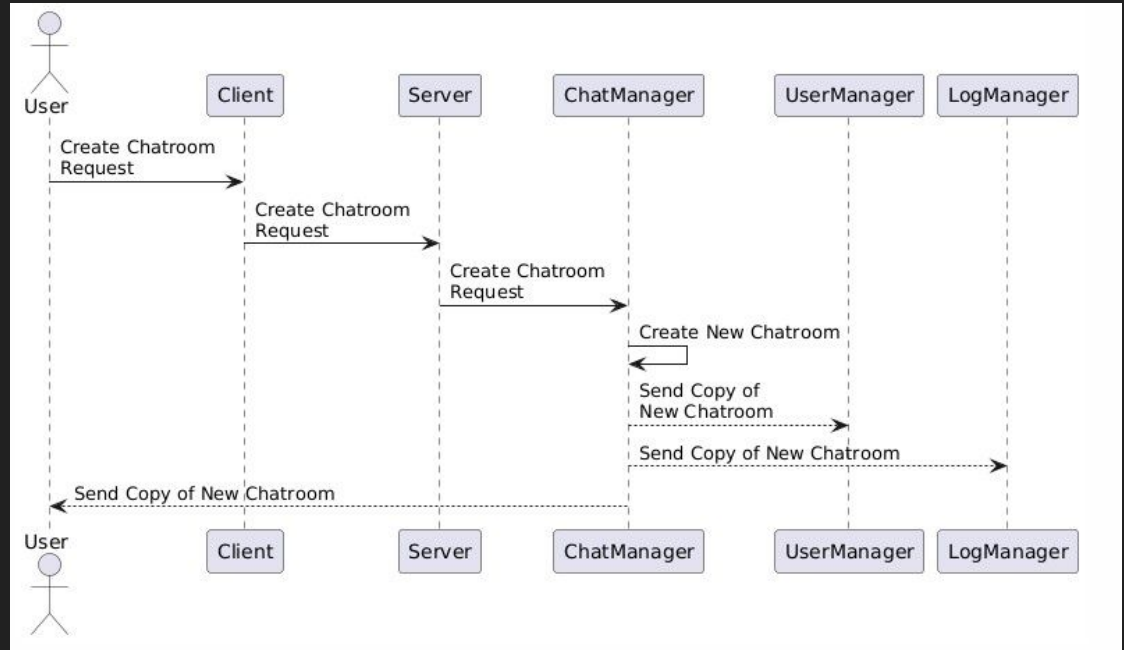
Sequence: Sending Message to Chatroom

- User Module sends write message request to Client
- Client creates message object and sends message to server
- Server relays message to ChatManager
- ChatManager finds chatroom and add's message to Chatroom
- ChatManager relays the message to the log manager which logs the message



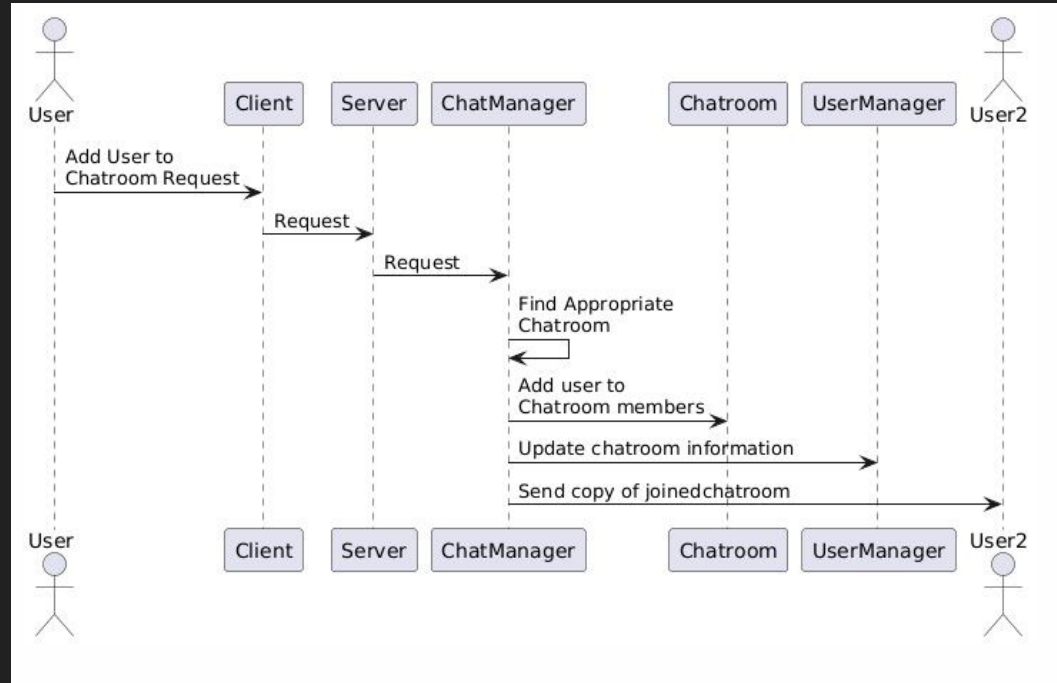
Sequence: Creating Chatroom

- User on client side clicks a button to create a chatroom
- Server receives the create chatroom request
- Server processes the request by creating a thread using the ChatRoomManager.
- ChatroomManager adds a new chatroom object to its mapping of ChatroomId : Chatroom.
- UserManager & LogManager are passed a copy of the ChatroomId so they can add it to their respective lists.
- A copy of the chatroom is transmitted back to the User



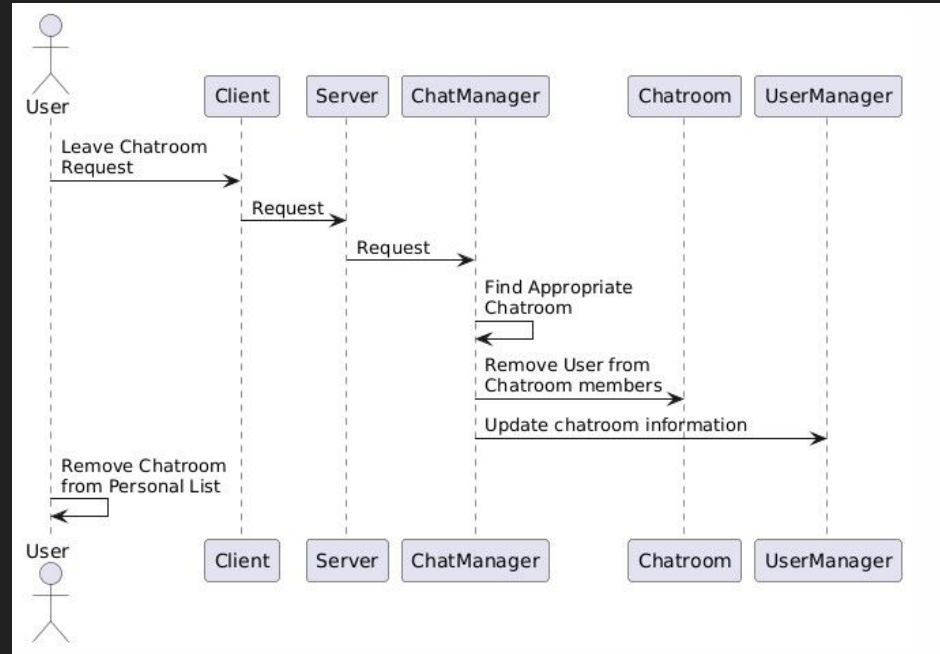
Sequence: Inviting User to Chatroom

- The User will request to join a chatroom from the Client, where the Server will pass this request to the ChatManager.
- The ChatManager will update both Chatroom and UserManager objects, and allow the User to join requested room.



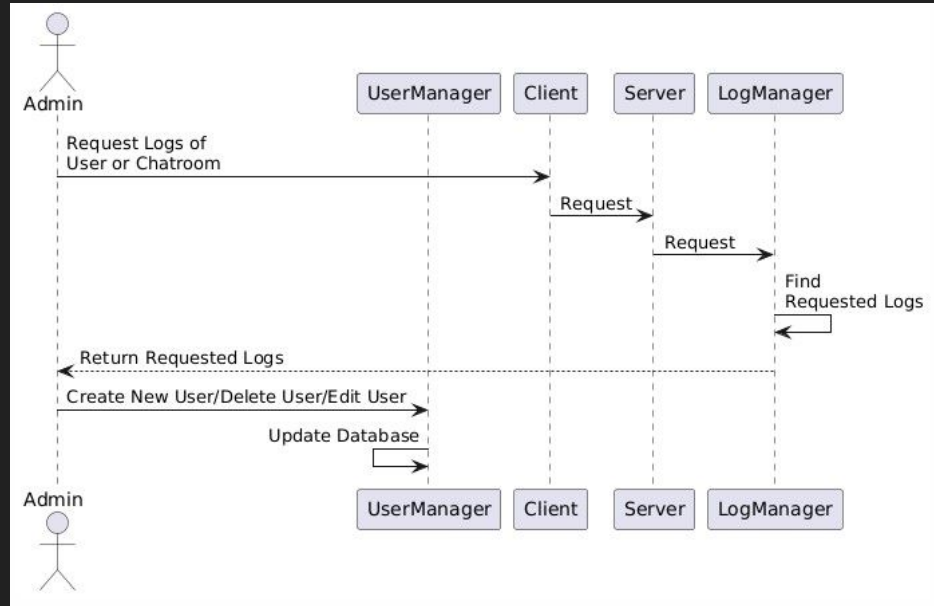
Sequence: Leaving Chatroom

- User inputs ChatroomID to leave or clicks a button corresponding to the chatroom
- Chatroom removed from client chatroom Map & client-side User involved chatrooms list
- Message sent to server informing that User has left the chatroom
- ChatroomManager removes User from the list of involved user for the corresponding chatroom
- ChatroomManager sends this removal update to every online client that holds that chatroom
- UserManager updates the server-side User object to not include that ChatroomID

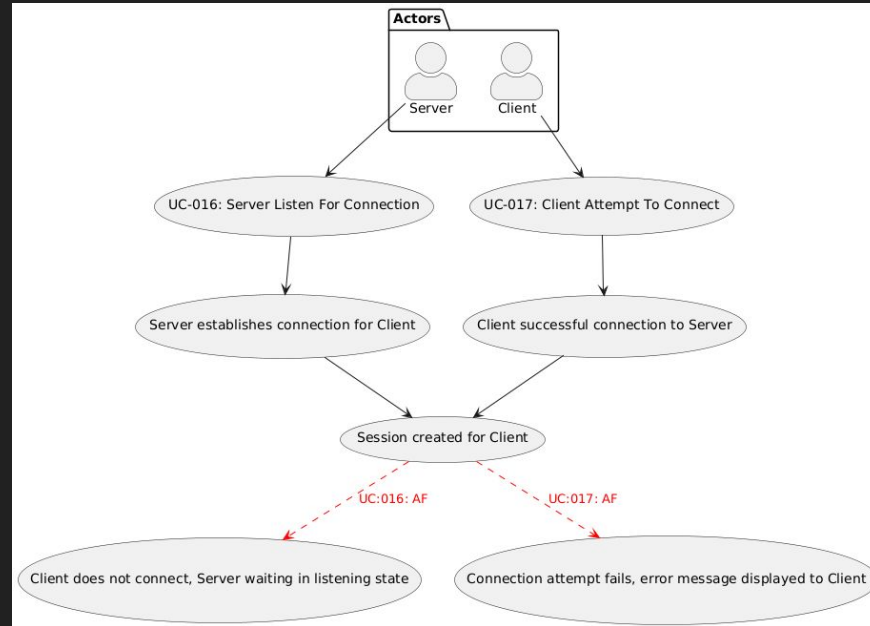


Sequence: Admin Actions

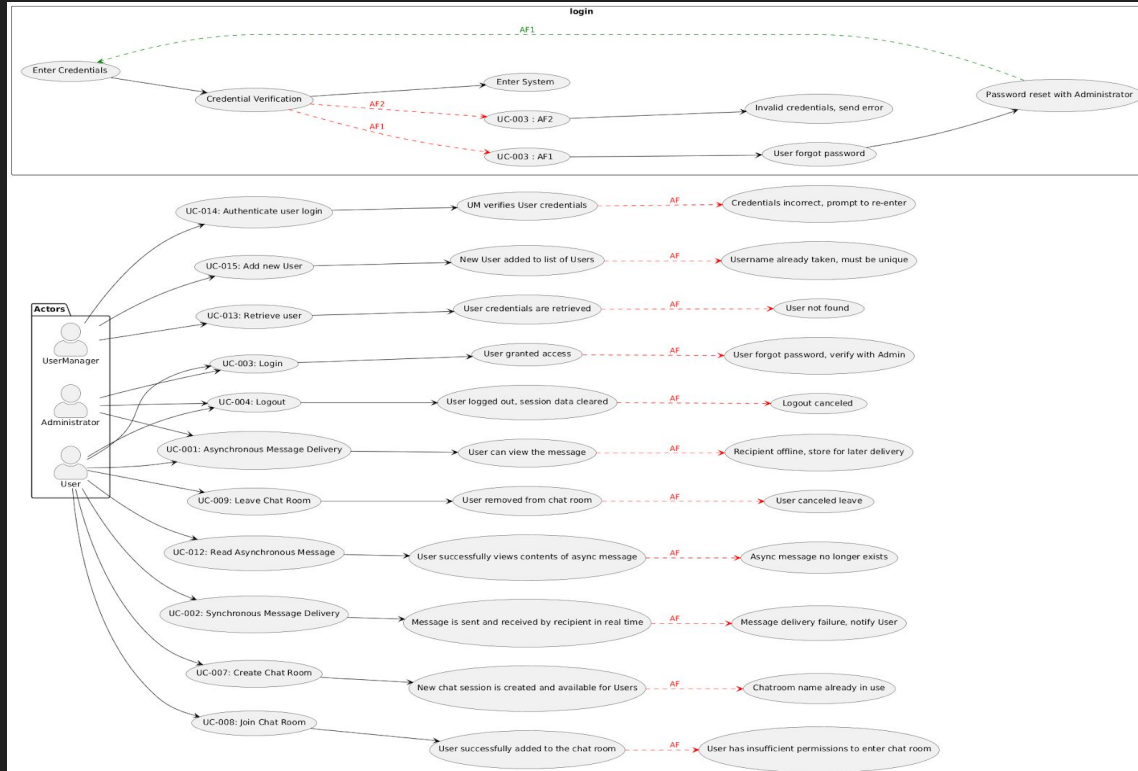
- Admin sends request logs of Users or Chatrooms to the Client
- Client sends request to the server which server relays request to the LogManager
- LogManager finds the requested logs and returns to the Admin
- Admin sends create new/delete/edit user request to the UserManager
- The UserManager updates the database



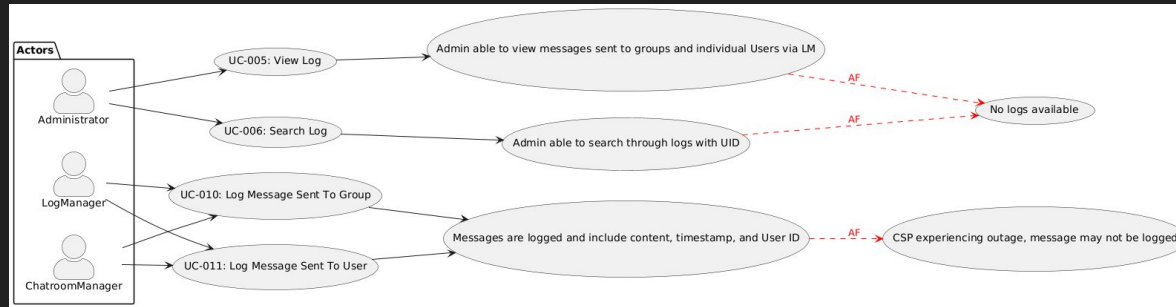
Server/Client Module Use Case Diagram



User/Chat Module Use Case Diagram



Log Module Use Case Diagram



Gantt Chart

2	Project Definition and Planning					
2.1	Design Phase Meeting	All members	10/15/24	10/15/24	1	100%
2.2	Design Document	All members	10/17/24	10/31/24	14	100%
2.3	User & Chatroom Design	Salvador Z	10/15/24	10/26/24	11	100%
2.4	userManager & ChatroomManager Design	Isaiah M	10/15/24	10/26/24	11	100%
2.5	Client & Server Design	Joseph D	10/15/24	10/26/24	11	100%
2.6	LogManager & Message Design	Joseph D	10/15/24	10/26/24	11	100%
2.7	ClientUI and AdminUI Design	Roman C	10/15/24	10/26/24	11	100%
2.8	Design - UML Use Case Diagram	Roman C	10/15/24	10/31/24	16	100%
2.9	Design - UML Sequence Diagram	Marcos G	10/15/24	10/31/24	16	100%
2.1	Design - UML Class Diagram	All members	10/15/24	10/31/24	16	100%
3	Project Conception and Initiation					
3.1	Server Class	Salvador A	11/3/24	12/3/24	0	0%
3.2	Client Class	Joseph D	11/3/24	12/3/24	0	0%
3.3	User Class	Salvador A	11/10/24	12/3/24	0	0%
3.4	Chatroom Class	Isaiah M	11/10/24	12/3/24	0	0%
3.5	Message & MessageCreator Classes	Isaiah M	11/10/24	12/3/24	0	0%
3.6	userManager Class	Marcos	11/17/24	12/3/24	0	0%
3.7	ChatroomManager Class	Marcos	11/17/24	12/3/24	0	0%
3.8	LogManager Class	Roman C	11/17/24	12/3/24	0	0%
3.9	ClientGUI	Roman C	11/10/24	12/3/24	0	0%