Group 7 Software Design Document

WeDiscuss Communications Application

Github: https://github.com/rmncbr/cs401-group7-communications

Revision History

Date	Revision	Description	Author
10/17/2024	1.0	Initial Release	Roman Cabrera
10/21/2024	1.5	Redesigned Document Layout	Joseph Dominguez
10/29/2024	1.6	ClientUI and AdminUI class design	Roman Cabrera
10/29/2024	1.7	Client, Server, Message, MessageCreator, LogManager class	Joseph Dominguez
10/29/2024	1.8	User & UserManager Class	Salvador Alvarez
10/29/2024	1.9	Chatroom & ChatroomManager Class	Isaiah Meza
10/30/2024	2.0	Use Cases Added	All Members
10/30/2024	2.1	Section 4: All Sequence Diagrams Added	Marcos Gomez
12/3/2024	2.2	Updated UML Diagrams	Salvador Alvarez

Table of Contents

1. Purpose

The Group 7 Software Design Document outlines the architecture and system design for WeDiscuss, a communications system project. WeDiscuss is designed to help employees of an organization communicate effectively both synchronously and asynchronously, enabling private and public messaging. This document is intended for Project Managers, Software Engineers, and anyone else who will be involved in the implementation of the system.

1.1. Scope

This document describes the implementation details of WeDiscuss Communications Application. WeDiscuss will consist of five major components: The Server/Client module, the User module, the Log module, the Chat module, and the GUI module. Each of the components will be explained in detail in this design document.

1.2. Definitions, Acronyms, Abbreviations

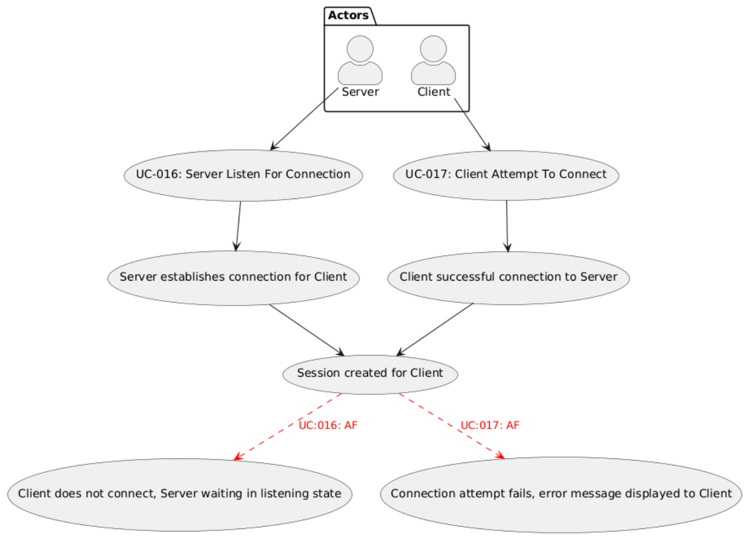
- UTU - User to User message
- UTC - User to Chatroom message

1.3. References

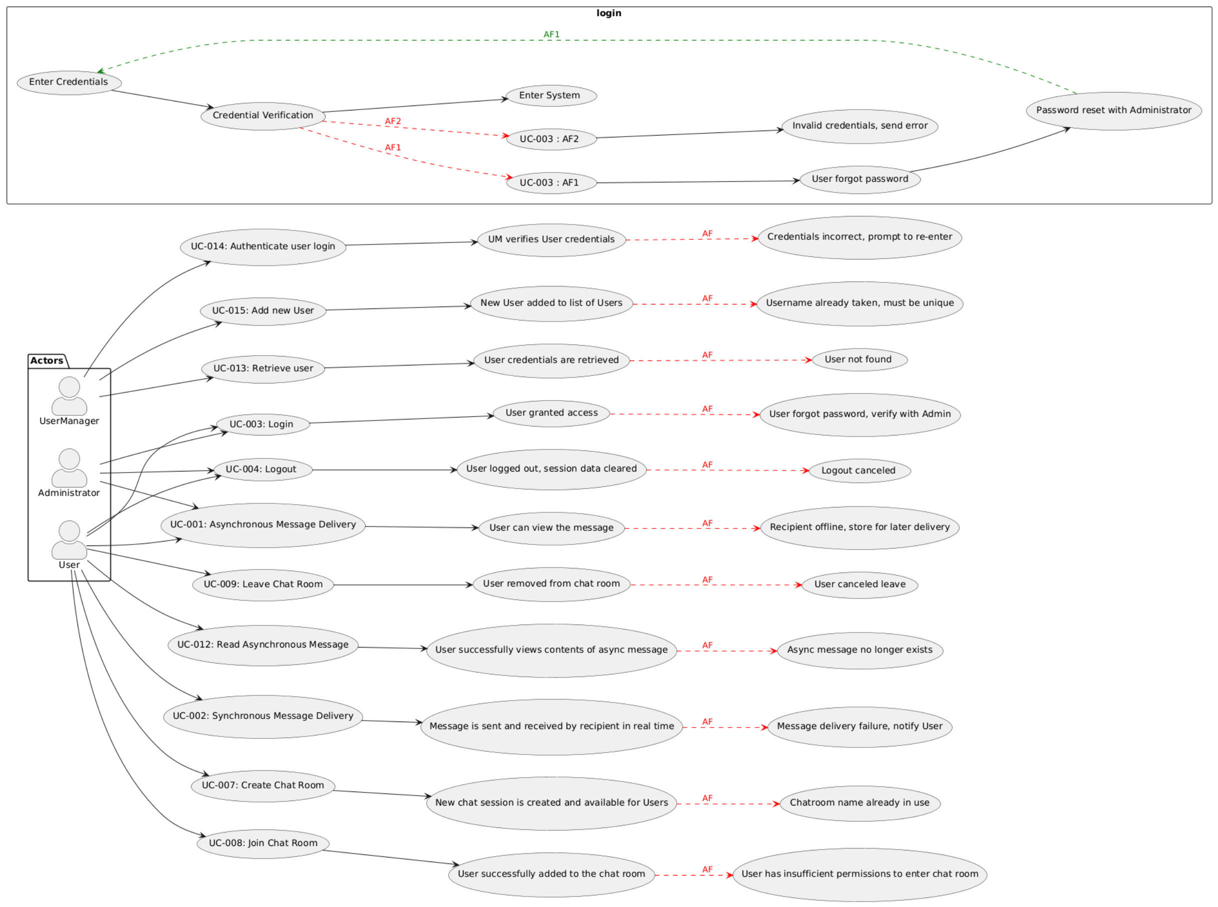
Group 7 *Software Specifications Document*

2. Use Case Overview

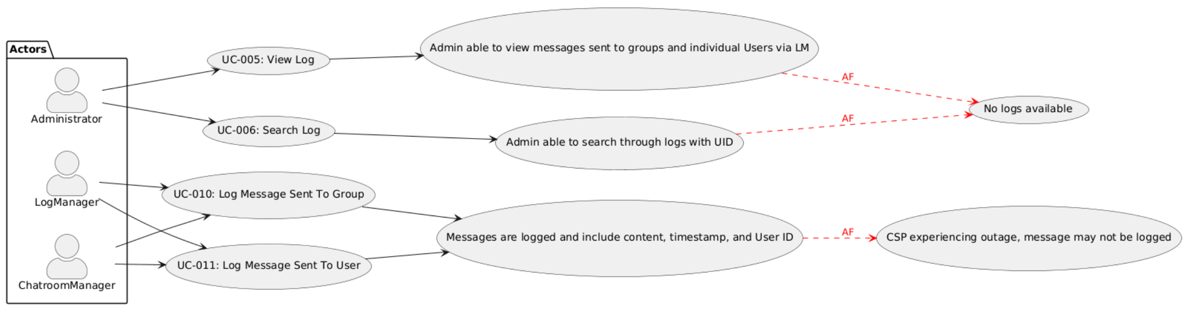
2.1 Server/Client Module Use Case Diagram


Actors
Server
Client
UC-016: Server Listen For Connection
UC-017: Client Attempt To Connect
Server establishes connection for Client
Client successful connection to Server
Session created for Client
UC:016: AF
UC:017: AF
Client does not connect, Server waiting in listening state
Connection attempt fails, error message displayed to Client

2.2 User/Chat Module Use Case Diagram

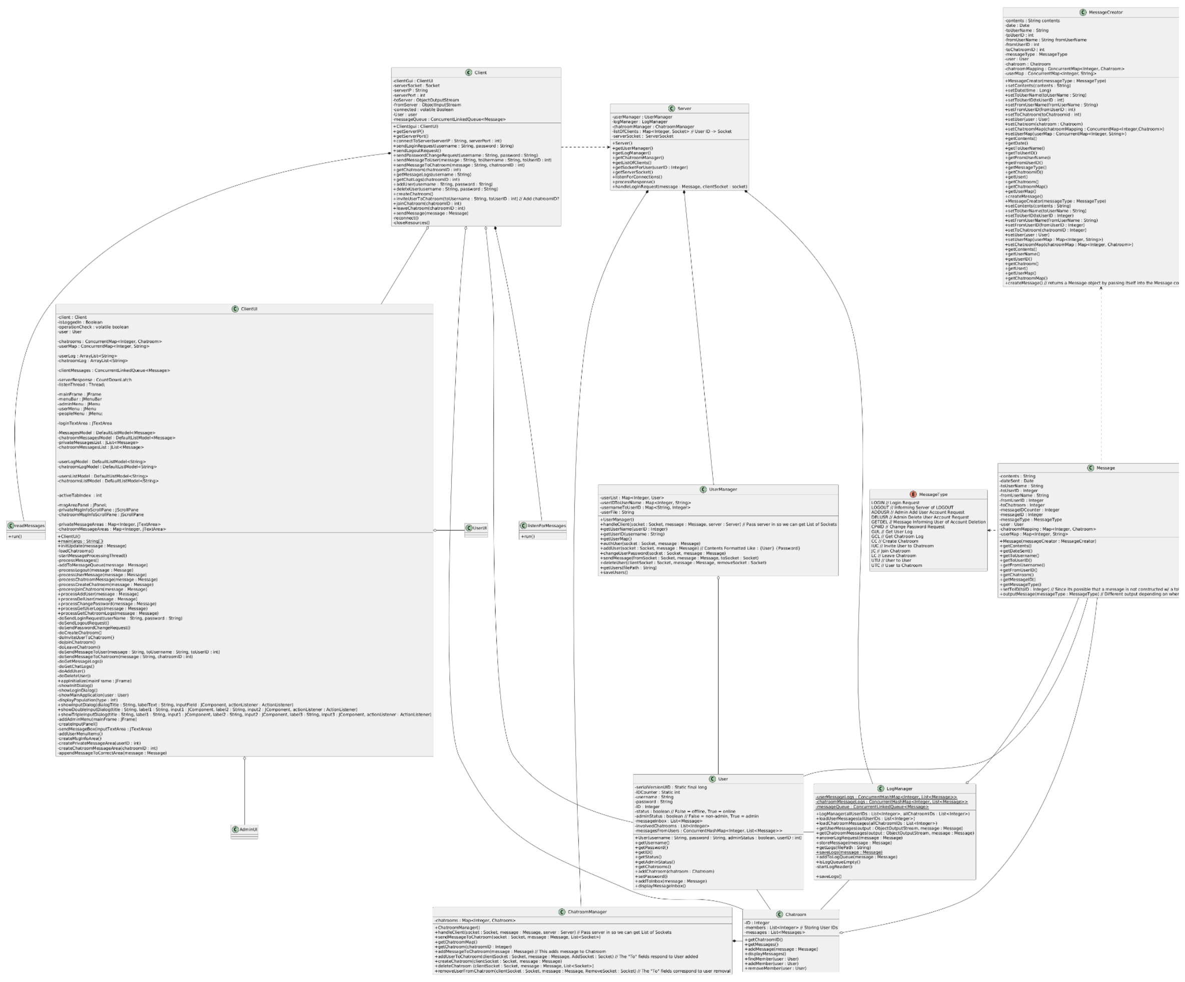

login
AF1
Enter Credentials
Credential Verification
Enter System
AF2
UC-003 : AF2
AF1
UC-003 : AF1
Invalid credentials, send error
User forgot password
Password reset with Administrator
Actors
UserManager
Administrator
User
UC-014: Authenticate user login
UM verifies User credentials
AF
Credentials incorrect, prompt to re-enter
UC-015: Add new User
New User added to list of Users
Username already taken, must be unique
UC-013: Retrieve user
User credentials are retrieved
User not found
UC-003: Login
User granted access
User forgot password, verify with Admin
UC-004: Logout
User logged out, session data cleared
Logout canceled
UC-001: Asynchronous Message Delivery
User can view the message
Recipient offline, store for later delivery
UC-009: Leave Chat Room
User removed from chat room
User canceled leave
UC-012: Read Asynchronous Message
User successfully views contents of async message
Async message no longer exists
UC-002: Synchronous Message Delivery
Message is sent and received by recipient in real time
Message delivery failure, notify User
UC-007: Create Chat Room
New chat session is created and available for Users
Chatroom name already in use
UC-008: Join Chat Room
User successfully added to the chat room
User has insufficient permissions to enter chat room

2.3 Log Module Use Case Diagram


Actors
Administrator
LogManager
ChatroomManager
UC-005: View Log
UC-006: Search Log
UC-010: Log Message Sent To Group
UC-011: Log Message Sent To User
Admin able to view messages sent to groups and individual Users via LM
Admin able to search through logs with UID
Messages are logged and include content, timestamp, and User ID
AF
AF
AF
No logs available
CSP experiencing outage, message may not be logged

3. Overall Class Design

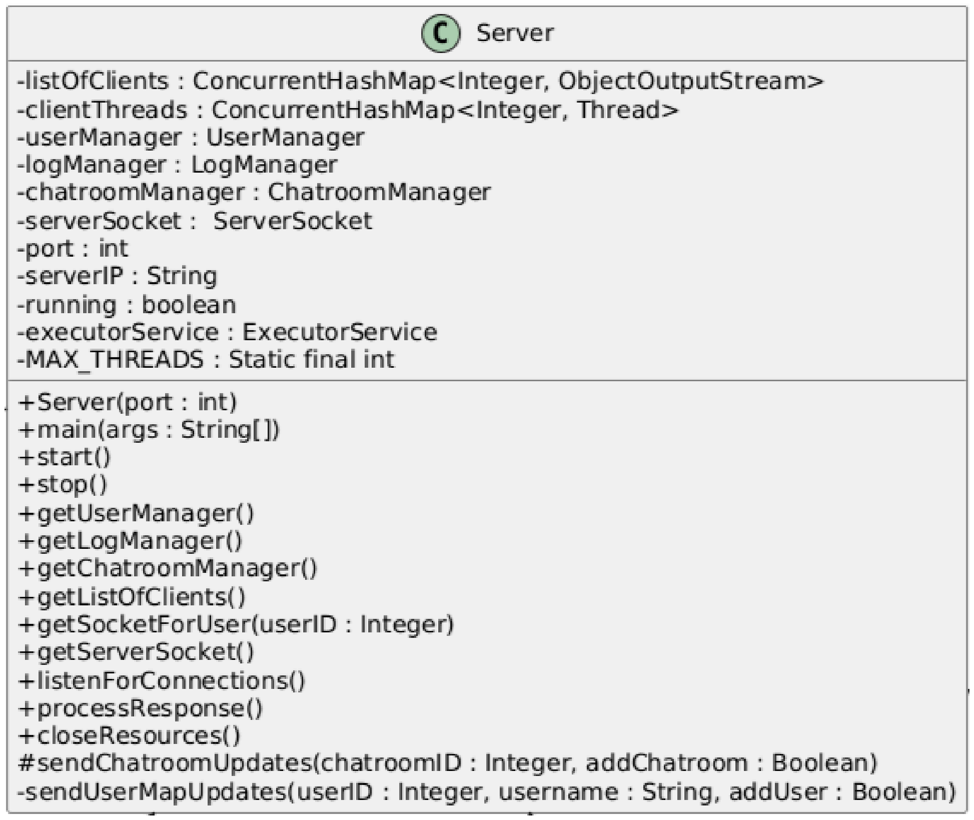
3.1. Top-Level UML Class Diagram Design



3.2. Individual Class Designs

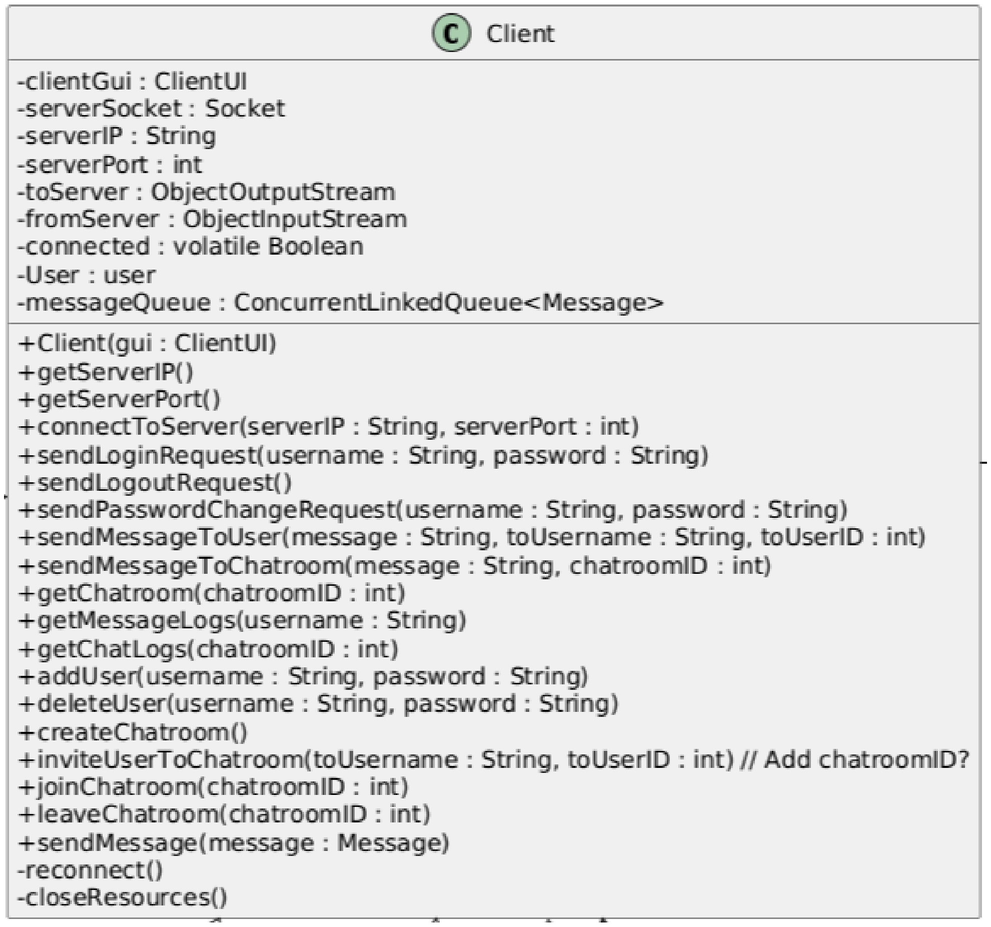
3.2.1. Server/Client Module

i. Server Class


Server
-listOfClients : ConcurrentHashMap<Integer, ObjectOutputStream>
-clientThreads : ConcurrentHashMap<Integer, Thread>
-userManager : UserManager
-logManager : LogManager
-chatroomManager : ChatroomManager
-serverSocket : ServerSocket
-port : int
-serverIP : String
-running : boolean
-executorService : ExecutorService
-MAX_THREADS : Static final int
+Server(port : int)
+main(args : String[])
+start()
+stop()
+getUserManager()
+getLogManager()
+getChatroomManager()
+getListOfClients()
+getSocketForUser(userID : Integer)
+getServerSocket()
+listenForConnections()
+processResponse()
+closeResources()
#sendChatroomUpdates(chatroomID : Integer, addChatroom : Boolean)
-sendUserMapUpdates(userID : Integer, username : String, addUser : Boolean)

The server class is an integral part of the client-server design pattern. The server's job is to listen for connections from clients and handle those incoming requests by creating a thread to process it and passing it to the appropriate handler 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.

ii. Client Class

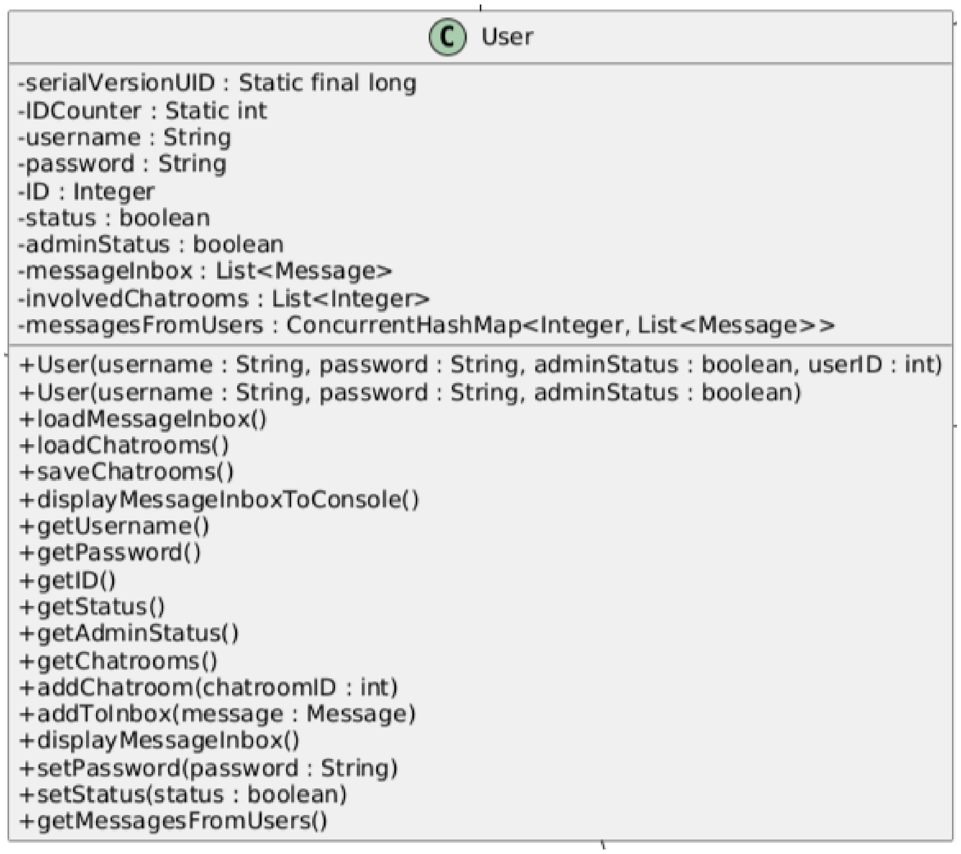


```
Client
-clientGui : ClientUI
-serverSocket : Socket
-serverIP : String
-serverPort : int
-toServer : ObjectOutputStream
-fromServer : ObjectInputStream
-connected : volatile Boolean
-User : user
-messageQueue : ConcurrentLinkedQueue<Message>
+Client(gui : ClientUI)
+getServerIP()
+getServerPort()
+connectToServer(serverIP : String, serverPort : int)
+sendLoginRequest(username : String, password : String)
+sendLogoutRequest()
+sendPasswordChangeRequest(username : String, password : String)
+sendMessageToUser(message : String, toUsername : String, toUserID : int)
+sendMessageToChatroom(message : String, chatroomID : int)
+getChatroom(chatroomID : int)
+getMessageLogs(username : String)
+getChatLogs(chatroomID : int)
+addUser(username : String, password : String)
+deleteUser(username : String, password : String)
+createChatroom()
+inviteUserToChatroom(toUsername : String, toUserID : int) // Add chatroomID?
+joinChatroom(chatroomID : int)
+leaveChatroom(chatroomID : int)
+sendMessage(message : Message)
-reconnect()
-closeResources()
```

The Client class is responsible for managing the interaction between users and WeDiscuss' core functionalities, which include user authentication, user and communication management, data handling, and user interface integration. The Client class will store maps of users and chatrooms for easy retrieval and display when a client connects to the server. 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.

3.2.2. User Module

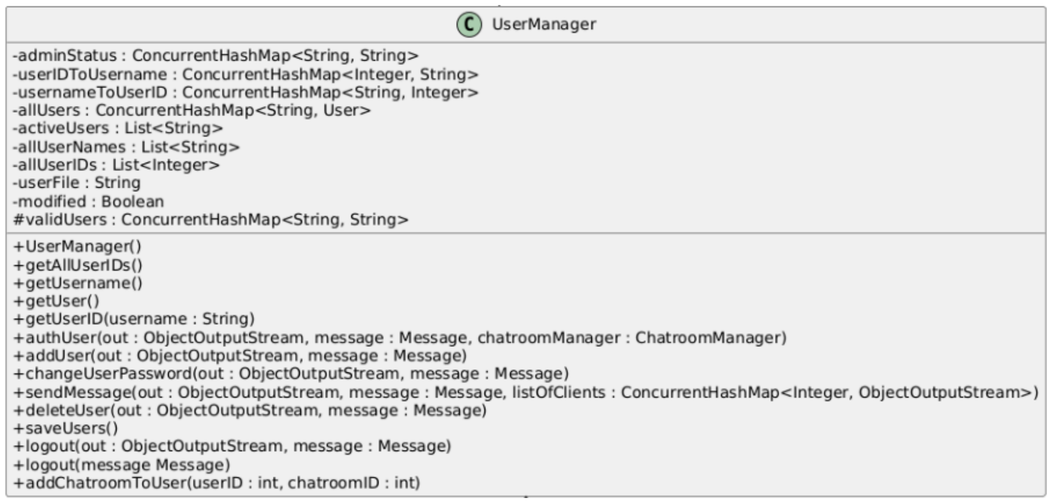
i. User Class



```
User
-serialVersionUID : Static final long
-IDCounter : Static int
-username : String
-password : String
-ID : Integer
-status : boolean
-adminStatus : boolean
-messageInbox : List<Message>
-involvedChatrooms : List<Integer>
-messagesFromUsers : ConcurrentHashMap<Integer, List<Message>>
+User(username : String, password : String, adminStatus : boolean, userID : int)
+User(username : String, password : String, adminStatus : boolean)
+loadMessageInbox()
+loadChatrooms()
+saveChatrooms()
+displayMessageInboxToConsole()
+getUsername()
+getPassword()
+getID()
+getStatus()
+getAdminStatus()
+getChatrooms()
+addChatroom(chatroomID : int)
+addToInbox(message : Message)
+displayMessageInbox()
+setPassword(password : String)
+setStatus(status : boolean)
+getMessagesFromUsers()
```

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.

ii. UserManager Class

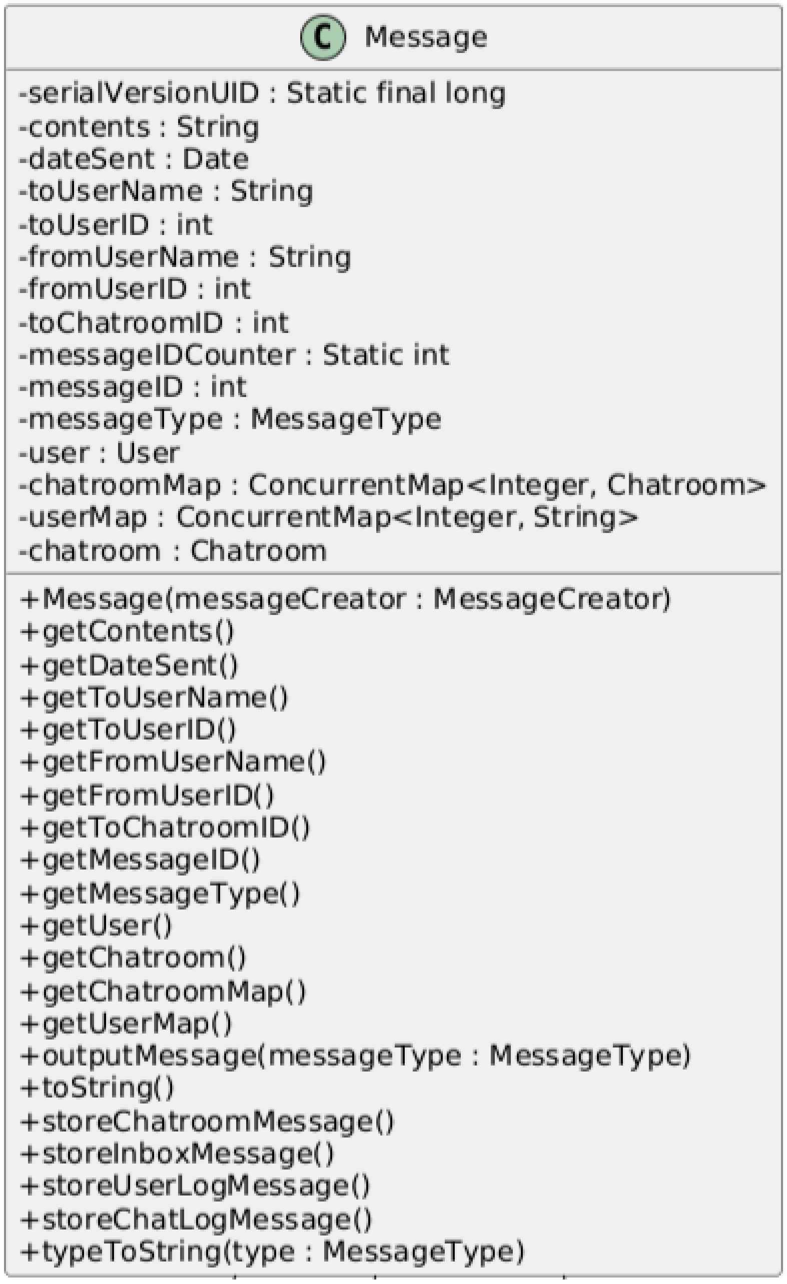


```
UserManager
-adminStatus : ConcurrentHashMap<String, String>
-userIDToUsername : ConcurrentHashMap<Integer, String>
-usernameToUserID : ConcurrentHashMap<String, Integer>
-allUsers : ConcurrentHashMap<String, User>
-activeUsers : List<String>
-allUserNames : List<String>
-allUserIDs : List<Integer>
-userFile : String
-modified : Boolean
#validUsers : ConcurrentHashMap<String, String>
+UserManager()
+getAllUserIDs()
+getUsername()
+getUser()
+getUserID(username : String)
+authUser(out : ObjectOutputStream, message : Message, chatroomManager : ChatroomManager)
+addUser(out : ObjectOutputStream, message : Message)
+changeUserPassword(out : ObjectOutputStream, message : Message)
+sendMessage(out : ObjectOutputStream, message : Message, listOfClients : ConcurrentHashMap<Integer, ObjectOutputStream>)
+deleteUser(out : ObjectOutputStream, message : Message)
+saveUsers()
+logout(out : ObjectOutputStream, message : Message)
+logout(message Message)
+addChatroomToUser(userID : int, chatroomID : int)
```

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.

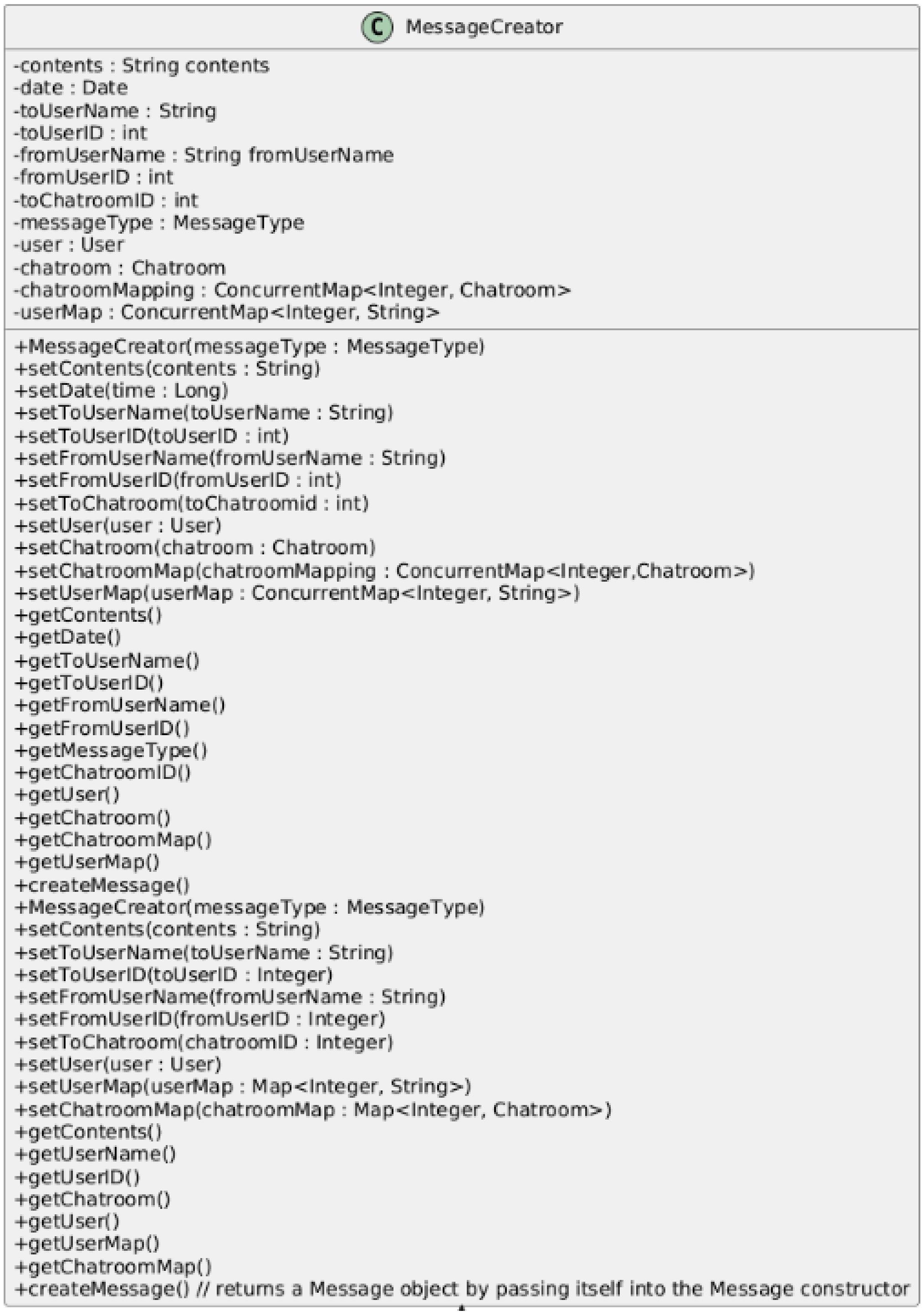
3.2.3. Log Module

i. Message Class



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 and client in determining the correct processing mechanism to employ.

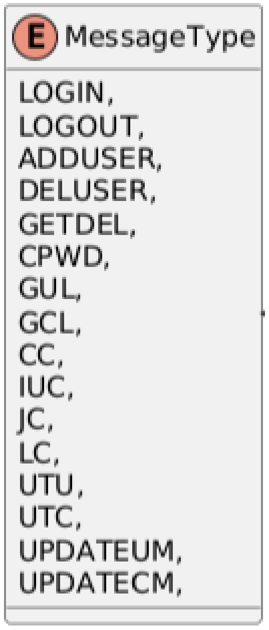
ii. MessageCreator Class



```
MessageCreator
-contents : String contents
-date : Date
-toUserName : String
-toUserID : int
-fromUserName : String fromUserName
-fromUserID : int
-toChatroomID : int
-messageType : MessageType
-user : User
-chatroom : Chatroom
-chatroomMapping : ConcurrentMap<Integer, Chatroom>
-userMap : ConcurrentMap<Integer, String>
+MessageCreator(messageType : MessageType)
+setContents(contents : String)
+setDate(time : Long)
+setToUserName(toUserName : String)
+setToUserID(toUserID : int)
+setFromUserName(fromUserName : String)
+setFromUserID(fromUserID : int)
+setToChatroom(toChatroomid : int)
+setUser(user : User)
+setChatroom(chatroom : Chatroom)
+setChatroomMap(chatroomMapping : ConcurrentMap<Integer,Chatroom>)
+setUserMap(userMap : ConcurrentMap<Integer, String>)
+getContents()
+getDate()
+getToUserName()
+getToUserID()
+getFromUserName()
+getFromUserID()
+getMessageType()
+getChatroomID()
+getUser()
+getChatroom()
+getChatroomMap()
+getUserMap()
+createMessage()
+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
```

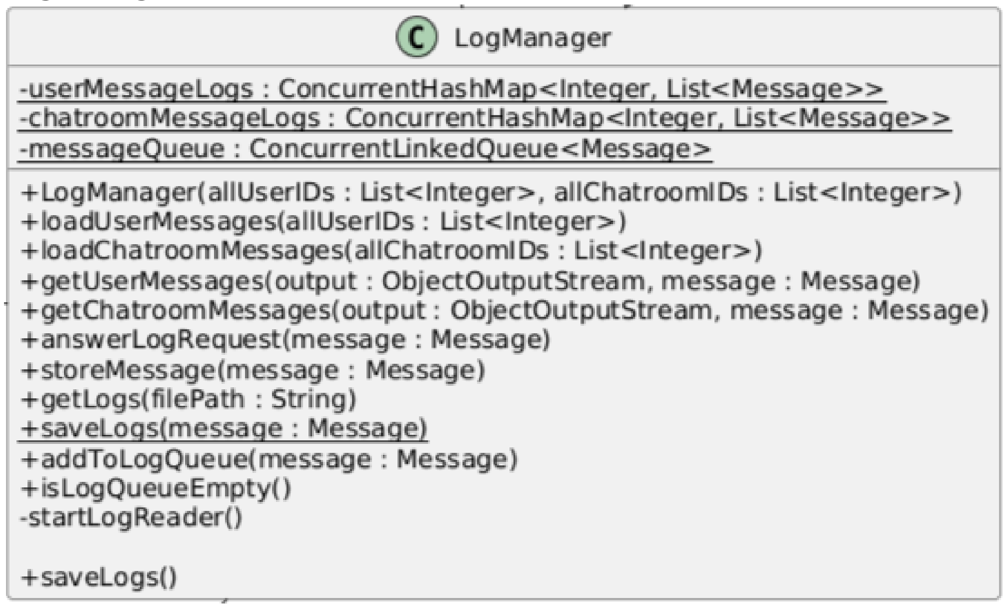
The MessageCreator class is responsible for building the attributes that are going to be set inside of a message object. These setters were separated from the Message class to emphasize and ensure the semi-immutability that the Message class is designed to have. To create a message, the attributes of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

iii. MessageType Enum



The MessageType enumeration is useful for helping to identify the type of processing that a transmitted message object should go through when received on both the client and server side.

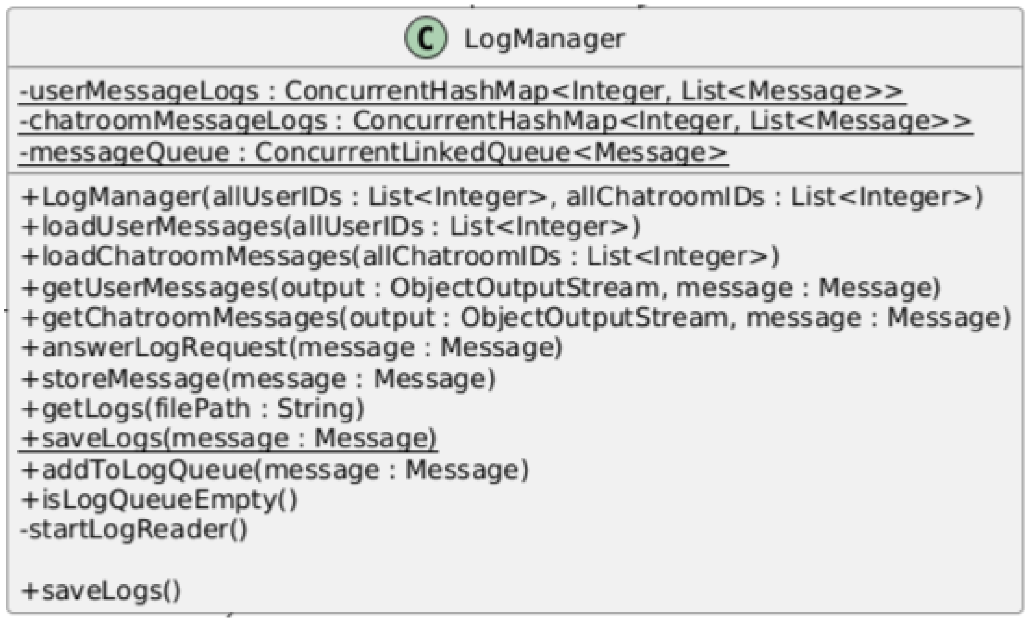
iv. LogManager Class


C
LogManager
-userMessageLogs : ConcurrentHashMap<Integer, List<Message>>
-chatroomMessageLogs : ConcurrentHashMap<Integer, List<Message>>
-messageQueue : ConcurrentLinkedQueue<Message>
+LogManager(allUserIDs : List<Integer>, allChatroomIDs : List<Integer>)
+loadUserMessages(allUserIDs : List<Integer>)
+loadChatroomMessages(allChatroomIDs : List<Integer>)
+getUserMessages(output : ObjectOutputStream, message : Message)
+getChatroomMessages(output : ObjectOutputStream, message : Message)
+answerLogRequest(message : Message)
+storeMessage(message : Message)
+getLogs(filePath : String)
+saveLogs(message : Message)
+addToLogQueue(message : Message)
+isLogQueueEmpty()
-startLogReader()
+saveLogs()

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.

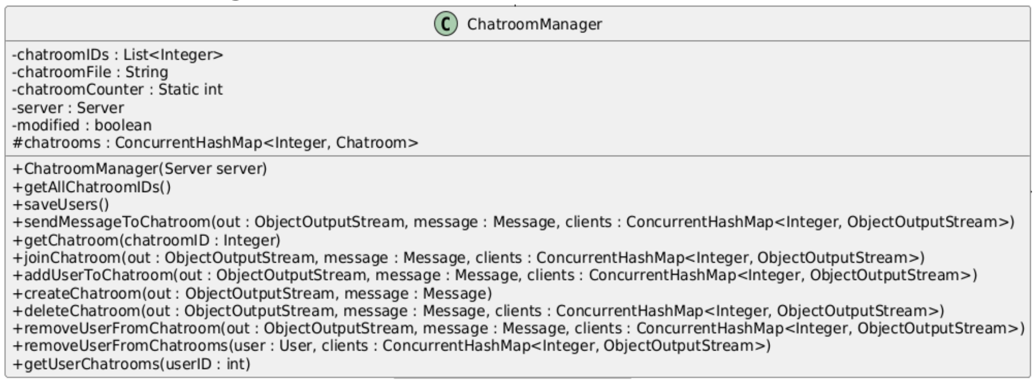
3.2.4. Chat Module

i. Chatroom Class


LogManager
-userMessageLogs : ConcurrentHashMap<Integer, List<Message>>
-chatroomMessageLogs : ConcurrentHashMap<Integer, List<Message>>
-messageQueue : ConcurrentLinkedQueue<Message>
+LogManager(allUserIDs : List<Integer>, allChatroomIDs : List<Integer>)
+loadUserMessages(allUserIDs : List<Integer>)
+loadChatroomMessages(allChatroomIDs : List<Integer>)
+getUserMessages(output : ObjectOutputStream, message : Message)
+getChatroomMessages(output : ObjectOutputStream, message : Message)
+answerLogRequest(message : Message)
+storeMessage(message : Message)
+getLogs(filePath : String)
+saveLogs(message : Message)
+addToLogQueue(message : Message)
+isLogQueueEmpty()
-startLogReader()
+saveLogs()

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.

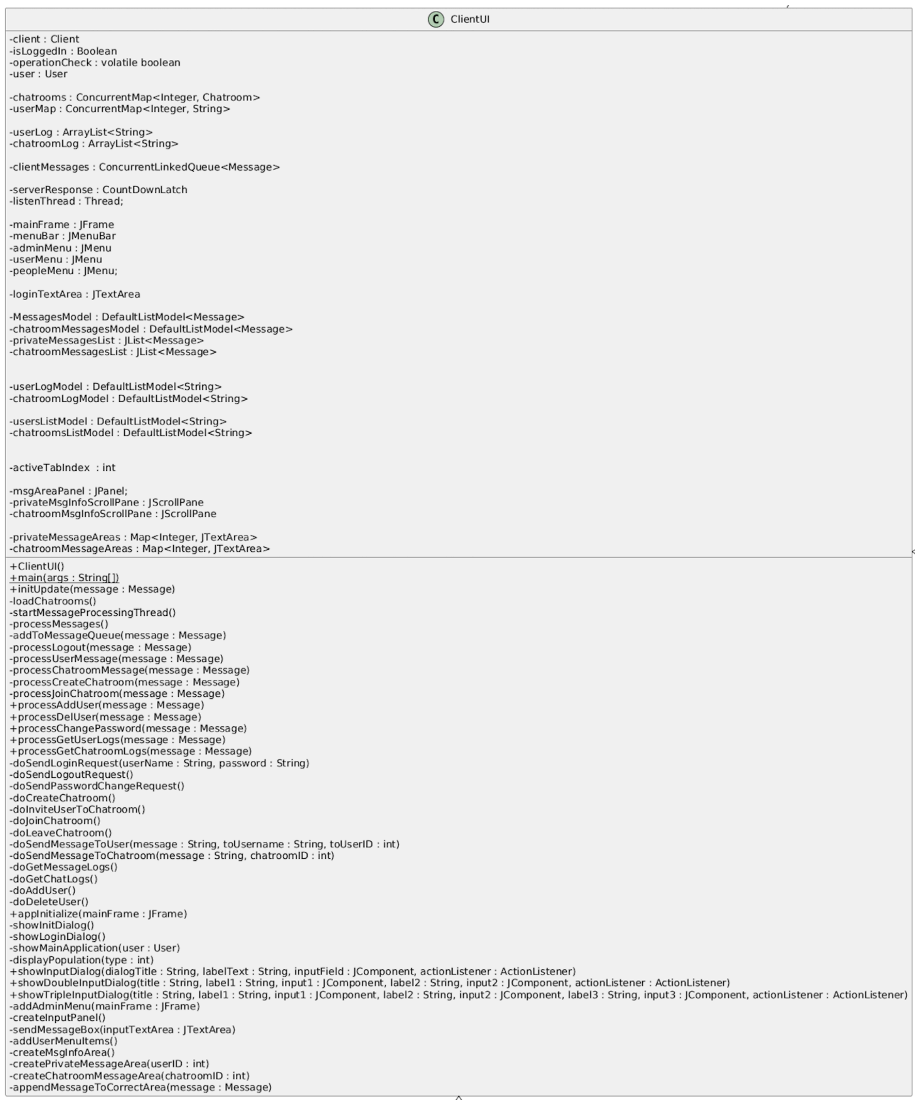
ii. Chatroom Manager Class


ChatroomManager
-chatroomIDs : List<Integer>
-chatroomFile : String
-chatroomCounter : Static int
-server : Server
-modified : boolean
#chatrooms : ConcurrentHashMap<Integer, Chatroom>
+ChatroomManager(Server server)
+getAllChatroomIDs()
+saveUsers()
+sendMessageToChatroom(out : ObjectOutputStream, message : Message, clients : ConcurrentHashMap<Integer, ObjectOutputStream>)
+getChatroom(chatroomID : Integer)
+joinChatroom(out : ObjectOutputStream, message : Message, clients : ConcurrentHashMap<Integer, ObjectOutputStream>)
+addUserToChatroom(out : ObjectOutputStream, message : Message, clients : ConcurrentHashMap<Integer, ObjectOutputStream>)
+createChatroom(out : ObjectOutputStream, message : Message)
+deleteChatroom(out : ObjectOutputStream, message : Message, clients : ConcurrentHashMap<Integer, ObjectOutputStream>)
+removeUserFromChatroom(out : ObjectOutputStream, message : Message, clients : ConcurrentHashMap<Integer, ObjectOutputStream>)
+removeUserFromChatrooms(user : User, clients : ConcurrentHashMap<Integer, ObjectOutputStream>)
+getUserChatrooms(userID : int)

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.

3.2.5. GUI Module

i. ClientUI Class

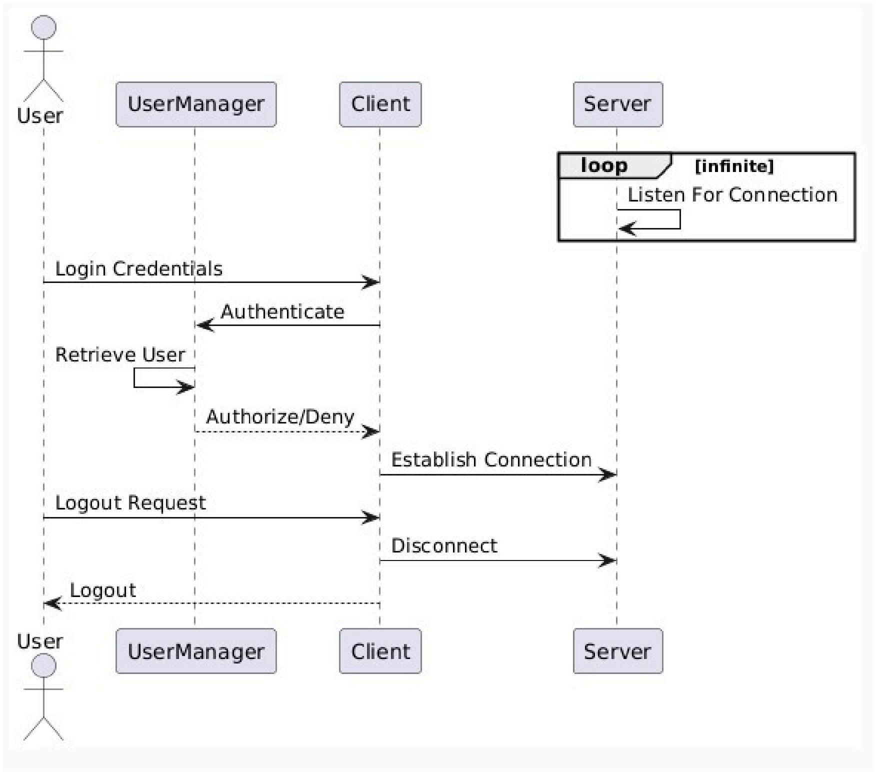


```
ClientUI
-client : Client
-isLoggedIn : Boolean
-operationCheck : volatile boolean
-user : User

-chatrooms : ConcurrentMap<Integer, Chatroom>
-userMap : ConcurrentMap<Integer, String>

-userLog : ArrayList<String>
-chatroomLog : ArrayList<String>

-clientMessages : ConcurrentLinkedQueue<Message>

-serverResponse : CountDownLatch
-listenThread : Thread;

-mainFrame : JFrame
-menuBar : JMenuBar
-adminMenu : JMenu
-userMenu : JMenu
-peopleMenu : JMenu;

-loginTextArea : JTextArea

-MessagesModel : DefaultListModel<Message>
-chatroomMessagesModel : DefaultListModel<Message>
-privateMessagesList : JList<Message>
-chatroomMessagesList : JList<Message>


-userLogModel : DefaultListModel<String>
-chatroomLogModel : DefaultListModel<String>

-usersListModel : DefaultListModel<String>
-chatroomsListModel : DefaultListModel<String>


-activeTabIndex  : int

-msgAreaPanel : JPanel;
-privateMsgInfoScrollPane : JScrollPane
-chatroomMsgInfoScrollPane : JScrollPane

-privateMessageAreas : Map<Integer, JTextArea>
-chatroomMessageAreas : Map<Integer, JTextArea>
+ClientUI()
+main(args : String[])
+initUpdate(message : Message)
-loadChatrooms()
-startMessageProcessingThread()
-processMessages()
-addToMessageQueue(message : Message)
-processLogout(message : Message)
-processUserMessage(message : Message)
-processChatroomMessage(message : Message)
-processCreateChatroom(message : Message)
-processJoinChatroom(message : Message)
+processAddUser(message : Message)
+processDelUser(message : Message)
+processChangePassword(message : Message)
+processGetUserLogs(message : Message)
+processGetChatroomLogs(message : Message)
-doSendLoginRequest(userName : String, password : String)
-doSendLogoutRequest()
-doSendPasswordChangeRequest()
-doCreateChatroom()
-doInviteUserToChatroom()
-doJoinChatroom()
-doLeaveChatroom()
-doSendMessageToUser(message : String, toUsername : String, toUserID : int)
-doSendMessageToChatroom(message : String, chatroomID : int)
-doGetMessageLogs()
-doGetChatLogs()
-doAddUser()
-doDeleteUser()
+appInitialize(mainFrame : JFrame)
-showInitDialog()
-showLoginDialog()
-showMainApplication(user : User)
-displayPopulation(type : int)
+showInputDialog(dialogTitle : String, labelText : String, inputField : JComponent, actionListener : ActionListener)
+showDoubleInputDialog(title : String, label1 : String, input1 : JComponent, label2 : String, input2 : JComponent, actionListener : ActionListener)
+showTripleInputDialog(title : String, label1 : String, input1 : JComponent, label2 : String, input2 : JComponent, label3 : String, input3 : JComponent, actionListener : ActionListener)
-addAdminMenu(mainFrame : JFrame)
-createInputPanel()
-sendMessageBox(inputTextArea : JTextArea)
-addUserMenuItems()
-createMsgInfoArea()
-createPrivateMessageArea(userID : int)
-createChatroomMessageArea(chatroomID : int)
-appendMessageToCorrectArea(message : Message)
```

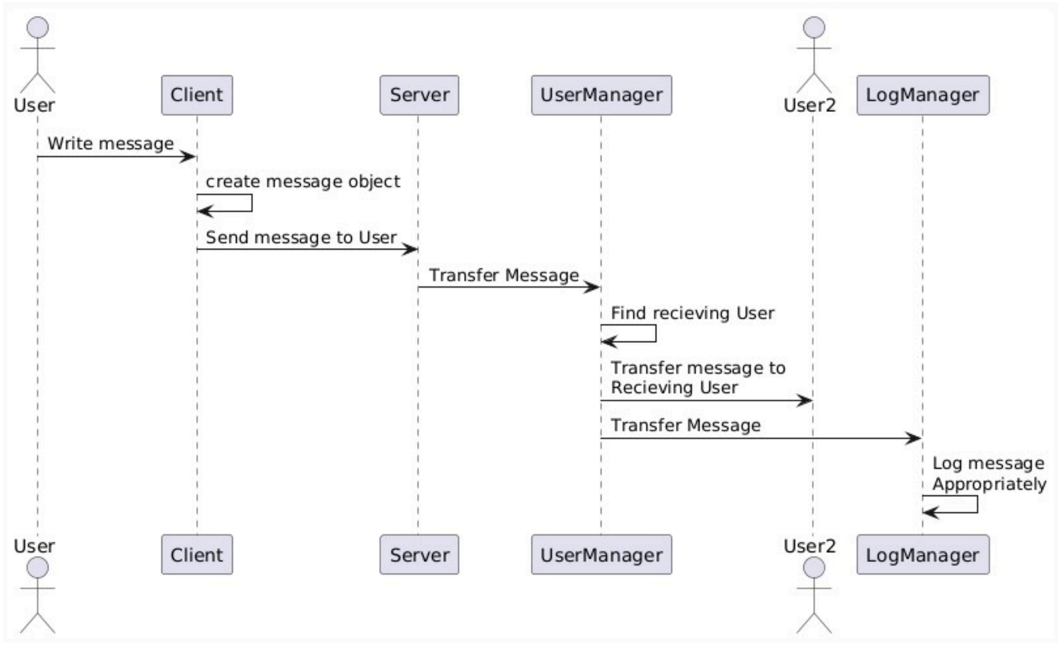
The ClientUI class is the base front-end user interface for the WeDiscuss application. It will provide functionalities for all users of the application, such as logging in and out, chatroom management, searching for users, and selecting profile pictures. The ClientUI will refresh objects and components throughout the activity of the application. Interfaces for both normal users and administrators will stem from the ClientUI class.

4. UML Sequence Diagrams

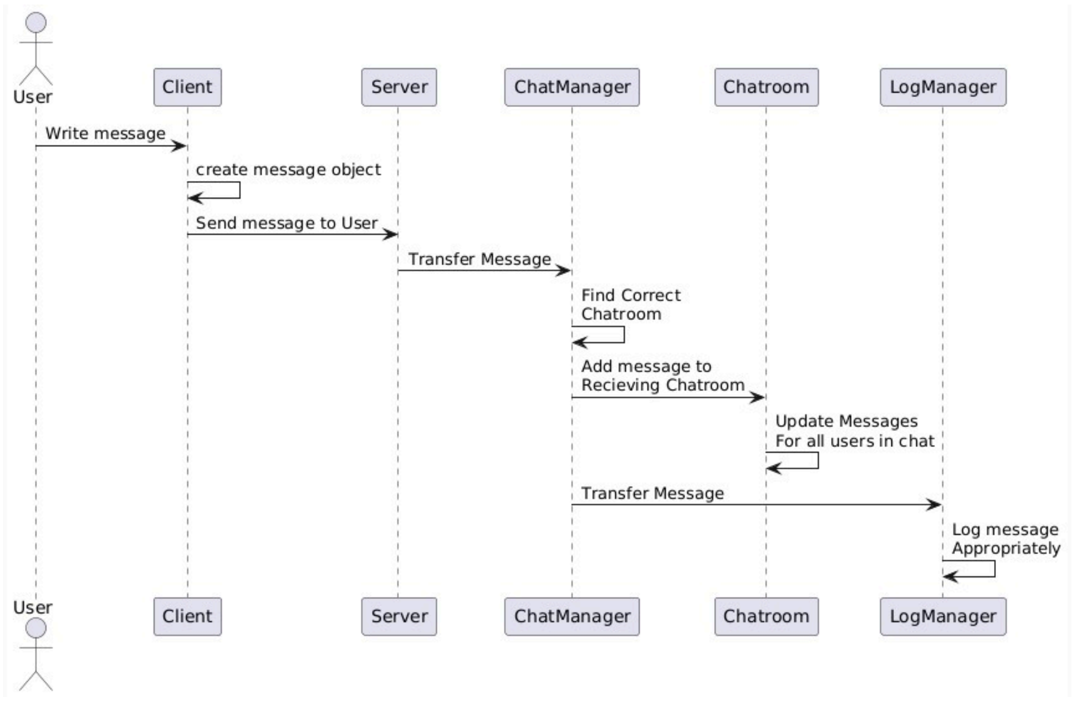
4.1.Login/Logout



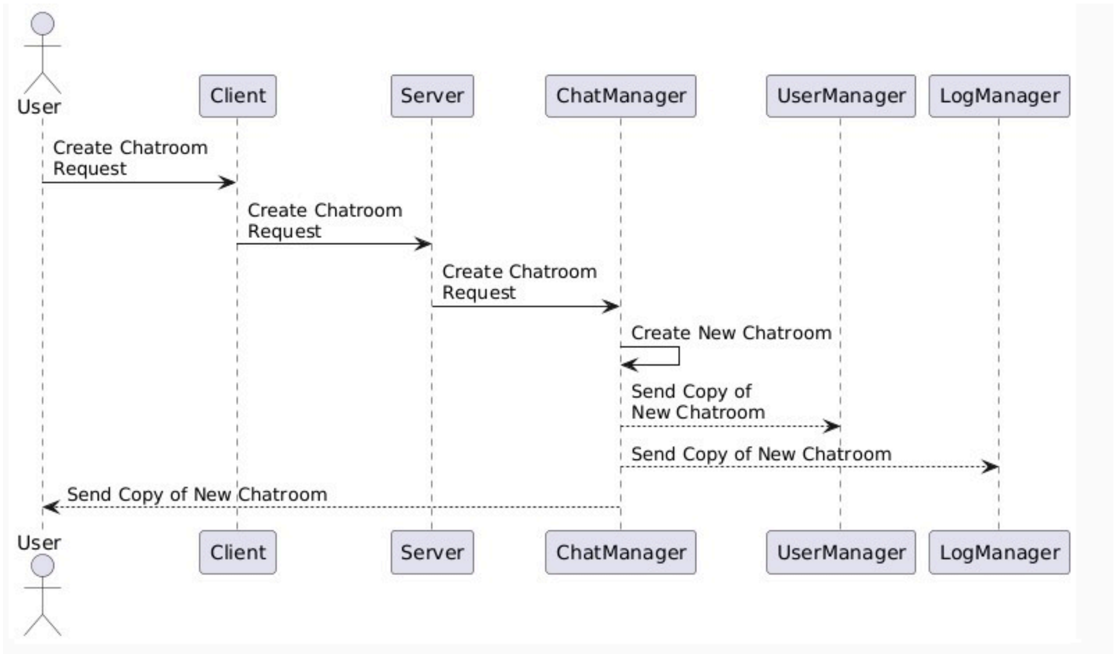
4.2.Send Message To User



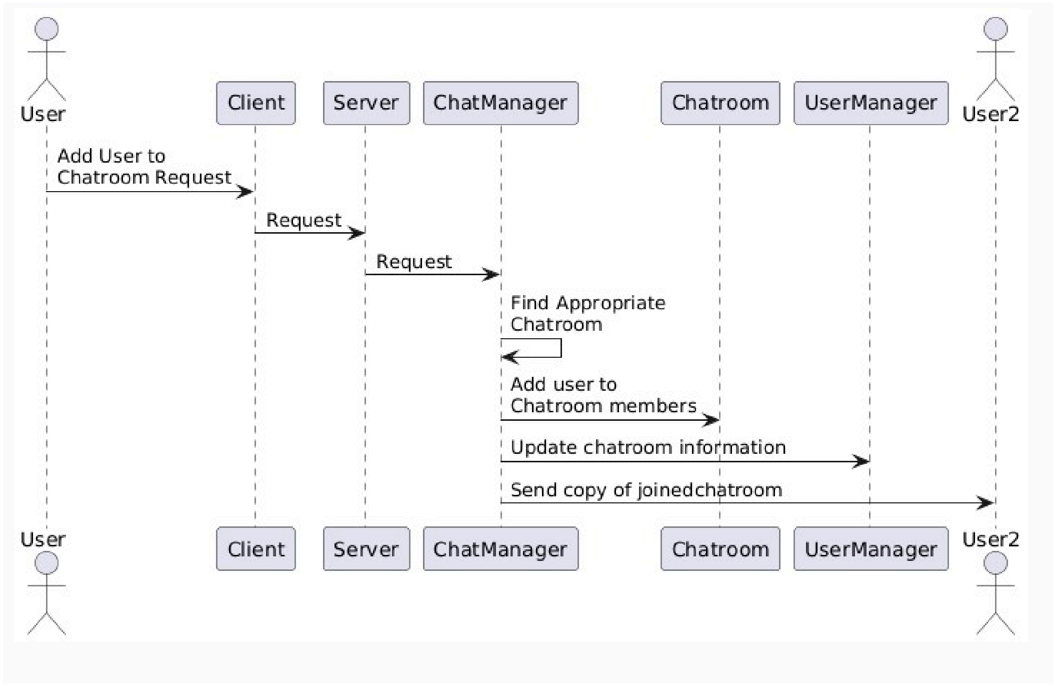
4.3. Send Message To Chatroom



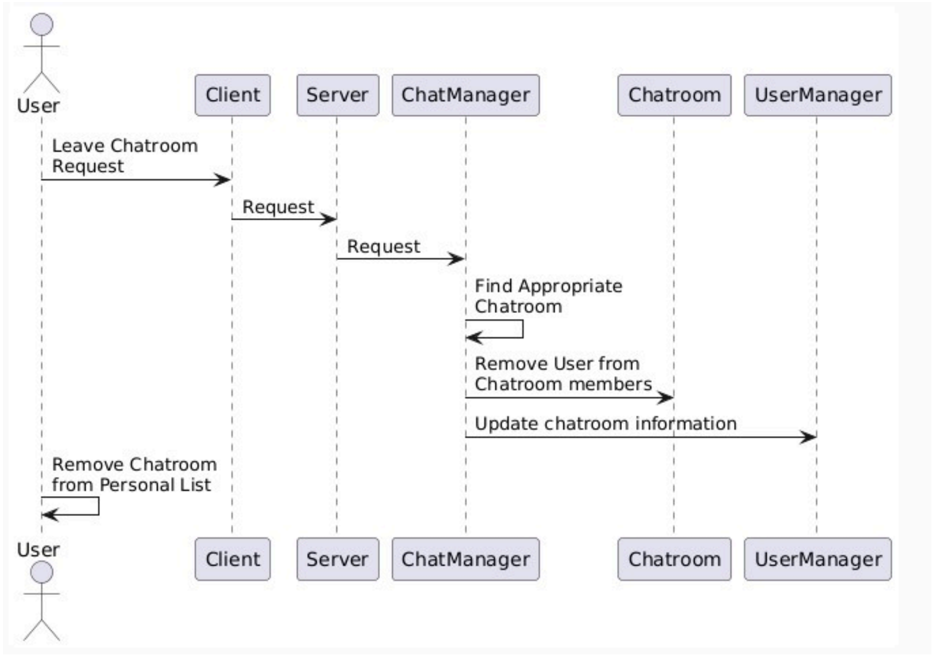
4.4. Creating Chatroom



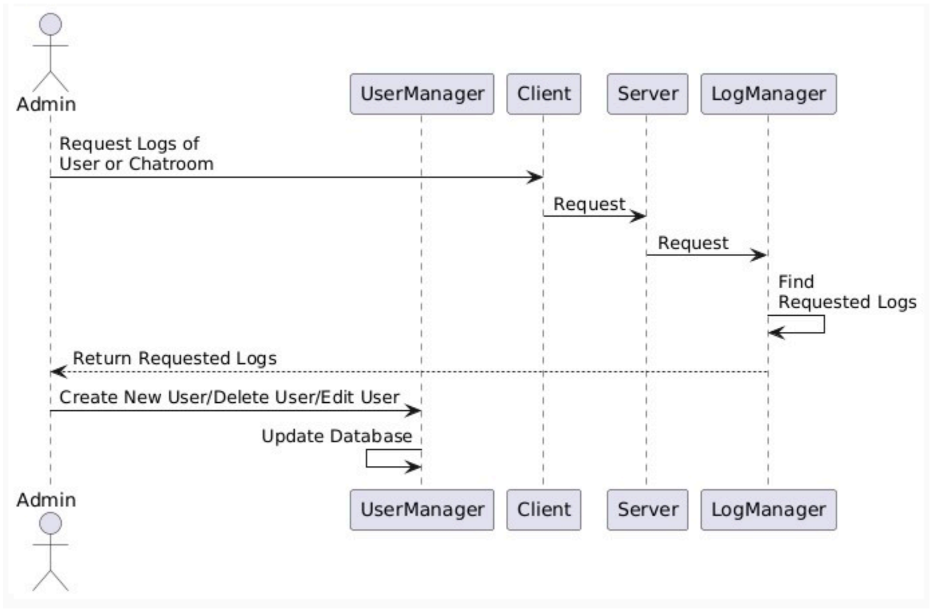
4.5. Inviting User To Chatroom


User
Client
Server
ChatManager
Chatroom
UserManager
User2
Add User to Chatroom Request
Request
Request
Find Appropriate Chatroom
Add user to Chatroom members
Update chatroom information
Send copy of joinedchatroom

4.6. Leaving Chatroom


User
Client
Server
ChatManager
Chatroom
UserManager
Leave Chatroom Request
Request
Request
Find Appropriate Chatroom
Remove User from Chatroom members
Update chatroom information
Remove Chatroom from Personal List

4.7. Admin Actions


Admin
UserManager
Client
Server
LogManager
Request Logs of User or Chatroom
Request
Request
Find Requested Logs
Return Requested Logs
Create New User/Delete User/Edit User
Update Database
Admin
UserManager
Client
Server
LogManager