Communication System

Software Requirements Specification

Revision History

Date	Revision	Description	Author
09/17/2025	1.0	Changed Title to "Communication System"	Oscar Velazquez Castillejo
9/24/2025	2.0	Collaboratively Completed Sections 1-4	Group 1
9/26/2025	3.0	Collaboratively Worked on Use Cases	Group 1
10/1/2025	4.0	Edited/Cleaned Up some of the SRS	Group 1
10/2/25	5.0	Polished SRS added Diagrams	Group 1

Table of Contents

1. Purpose

This document outlines the requirements for the Communication System Application.

1.1. Scope

This document will catalog the requirements for the Communication System application. It will not, however, document how these requirements will be implemented.

1.2. Definitions, Acronyms, Abbreviations

CS	Communication System
TCP/IP	Transmission Control Protocol/Internet Protocol
GUI	Graphical User Interface

1.3. References

GitHub Repo - https://github.com/ovelazqz/Communications-System---Group-1

Use Case Specification Document - 9

UML Use Case Diagrams Document - 11

Class Diagrams - 12

Sequence Diagrams - 13

1.4. Overview

The Communication System (CS) is designed for a very large organization. This system will allow employees to communicate over chat both synchronously and asynchronously. Users will be able to chat privately and in groups. All conversations will be logged and viewable by the IT users. Privacy will be minimized. Only text is required at this time.

2. Overall Description

2.1. Product Perspective

2.2. Product Architecture

The system will be organized into 5 major modules: the GUI module, the User module, the Chat module, the Server module, and the Client module.

2.3. Product Functionality/Features

The high-level features of the system are as follows (see section 3 of this document for more detailed requirements that address these features):

2.4. Constraints

2.4.1 All messages (private and group) must be logged and stored in a .txt file and viewable by IT.

2.4.2 All users will not be able to delete or permanently hide any communication.

2.4.3 Users will have no expectation of privacy. No end-to-end encryption or user-side encryption is to be implemented.

2.4.4 Only plain text messaging is supported in this version. No file sharing, emojis, voice, or video features will be implemented at this stage.

2.4.5 The Communication System must operate over TCP/IP.

2.5. Assumptions and Dependencies

2.5.1 It is assumed that all users will access the system from within the organization's internal secured network.

2.5.2 It is assumed that designated IT administrators will be responsible for monitoring and user permissions.

2.5.3 It is assumed that all communications will be text-only during the initial deployment, and users will not require support for multimedia content (e.g., images, videos, file attachments).

2.5.4 The system depends on IT administrators to create and manage user accounts.

3. Specific Requirements

3.1. Functional Requirements

1.1.1. Common Requirements:

3.1.1.1 All components of the system must log user activity, including login/logout, message sending, group creation, and any administrative access, with a timestamp.

3.1.1.2 System responses (e.g., message sent confirmation, errors) must be returned to the user.

3.1.1.3 The system requires real-time logging of all messages for IT administrators.

3.1.1.4 The system requires each user to have a unique username and password to access the application.

3.1.1. GUI Module Requirements:

3.1.1.1 The GUI is dependent on the Client module; it will only send requests to update the GUI.
3.1.1.2 The GUI will display notifications.
3.1.1.3 The GUI will have three different states: LOG_IN, EMPLOYEE, and IT_ADMINISTRATOR.
3.1.1.4 The GUI will display both private chats and group chats.
3.1.1.5 The GUI will display a username/password input field for the Login.
3.1.1.6 The GUI will provide IT Admin tools if the user has IT_ADMINISTRATOR role.
3.1.1.7 The GUI will allow users to discover other users and create chats.

3.1.2. User Module Requirements:

3.1.2.1 The User module will create a unique identifier for each user object.

3.1.2.2 The User module will have a username, password, and role attribute.

3.1.2.3 The User module will have a static counter to track user instances

3.1.3. Chat Module Requirements:

3.1.3.1 The Chat will be given a unique ID

3.1.3.2 The Chat will store a list of participants and messages

3.1.3.3 The Chat will have a chatType to be either private or group

3.1.3.4 The Chat will have a method that returns the unique ID

3.1.3.5 The Chat will have an appending method to include the new messages to the conversation

3.1.3.6 The Chat will have a retriever that returns the complete message history for the chat

3.1.4. Server Module Requirements:

3.1.4.1 The server must accept multiple concurrent client connections using TCP/IP sockets.
3.1.4.2 The Server will reject any connections from unauthorized clients.
3.1.4.3 The Client will send login credentials to the server upon connection.
3.1.4.4 The Server will log every transmitted message.

3.1.4.5 The Server will allow IT Administrators to retrieve chat logs.
3.1.4.6 The Server shall support real-time (synchronous) messaging between online users.
3.1.4.7 The Server shall support asynchronous messaging by storing undelivered messages for offline users and delivering them when they reconnect.
3.1.4.8 The Server will include timestamps, sender/receiver, status (read/delivered) for messages.
3.1.4.9 The server will store, retrieve, maintain User accounts and Messages.
3.1.4.10 The server will give users the option to manually log out, or it will automatically log them out when inactive.
3.1.4.11 The server will lock the users account after 5 failed login attempts.
3.1.4.12 The server will not unsend or delete any messages once sent.

3.1.5. Client Module Requirements:

3.1.5.1 The Client will present a login interface where employees can enter their username and password.
3.1.5.2 The Client will present a list of users to chat with.
3.1.5.3 The Client will display messages in real-time.
3.1.5.4 The Client will allow users to create group chats.
3.1.5.5 The Client will allow users to initiate one-on-one (private) chats
3.1.5.6 The Client shall retrieve undelivered messages from the Server Client upon login.

3.2. External Interface Requirements

3.2.1 The system must provide a GUI for employees to log in, send one-to-one messages, and participate in group chats. The system will consist of window pages for the log in, chats, and IT Administrator tools.

3.2.2 The system will deny users who are not found in the system and log the failed attempt. This will help the IT track down any strange behavior happening improving security.

3.2.3 The system will back up information whenever there is a crash. Users will be back where they left off, their messages won't be lost.

3.3. Internal Interface Requirements

3.3.1 The system will allow users to find other users to chat with.
3.3.2 The system will allow users to create private one on one chats or create groups for collaboration.
3.3.3 The system will allow users to send and receive direct messages with other users in real time (synchronous) and offline (asynchronous).

4. Non-Functional Requirements

4.1. Security and Privacy Requirements

4.1.1 User will use a username and password to access the application
4.1.2 There will be no end-to-end encryption for this application

4.2. Environmental Requirements

4.2.1 Server application will run on one system, while client application can run on any other system.
4.2.2 System must use the University network for communication with the server application and the client application using TCP/IP
4.2.3 Group chat should allow up to N participants in real time.

4.3. Performance Requirements

4.3.1 Typing indicators should update in real time.
4.3.2 Message logs should be stored within 2 seconds.
4.3.3 Logs should be searchable by IT with filters (user, date, keyword) in real-time.

5. USE CASE SPECIFICATIONS

Use Case ID: UC-LOGIN
Use Case Name: User Login
Relevant Requirements: Only authenticated users can have access to the system. If they are not authenticated it will lock them out. Login validates their credentials.
Primary Actor: User (IT Admin and Employee)
Pre-conditions: User has a valid username and password.
Post-conditions: Once they are authenticated it redirects them to the chat dashboard.
Basic Flow or Main Scenario: 1. The user opens the log in page. 2. They enter their credentials. 3. System checks to make sure credentials are good. 4. System redirects them to the main dashboard.
Extensions or Alternate Flows: If the user account is not found or blocked then it will deny the login. If the user forgets their password, IT will give it to them.
Exceptions: Incorrect username or password an Eror message will be shown saying that the login is denied. System crashes or unavailable the user will see a maintenance message.
Related Use Cases: UC-MANAGE, UC-GROUPCHAT, UC-CHAT_LOGS

Use Case ID: UC-MANAGE
Use Case Name: Manage Accounts
Relevant Requirements: Only authenticated IT Administrators are given the ability to manage accounts.
Primary Actor: IT Administrator
Pre-conditions: The Client connects to the server successfully followed by the IT Administrator successfully logging in and requesting to manage accounts.
Post-conditions: The server reflects the requested changes.
Basic Flow or Main Scenario: 1: IT Administrator requests to manage accounts from the server. 2: Server responds with the following options: Create Account, Update Account, Remove Account. 3: IT Administrator selects an option. 4. Server validates input. 5. Server updates the account list accordingly. 6. Server responds with a confirmation of said updates.
Extensions or Alternate Flows: The IT Administrator chooses Create Account, system prompts for username, password, role, then adds account and confirms changes.
Exceptions: Attempting to create an account with an existing username will result in an error message. All fields (username, password, and role) must be filled out correctly or will be rejected by server. Account not found. The connection between the Client and Server is lost, resulting in the cancelation of the current operation.
Related Use Cases: UC-LOGIN

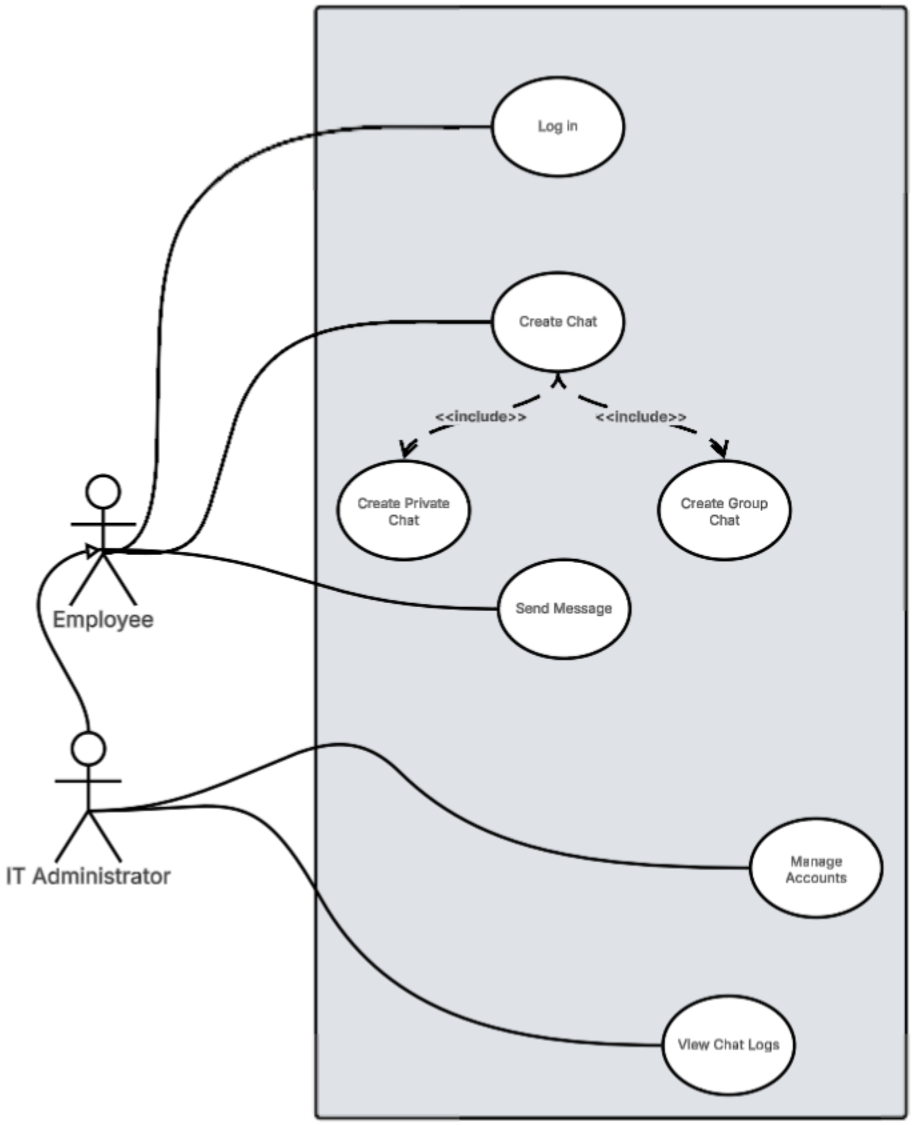
Use Case ID: UC-GROUPCHAT
Use Case Name: Making a group chat
Relevant Requirements: Once in the application, the user can find other users and add them to group chat.
Primary Actor: User (IT Admin and Employee)
Pre-conditions: Employees and IT employees must be login into the application to start making group chats with other employees in the company.
Post-conditions: the employees and IT employees can now talk to other employees in group chats
Basic Flow or Main Scenario: 1. employees and IT employees first login. 2. can search for other employees. 3. can add then to group chat. 4. can now talk to other employees in group chat.
Extensions or Alternate Flows: typing name wrong in the search for other employees. Can't find employees in the search.
Exceptions: adding employees that are fired.
Related Use Cases: UC-LOGIN

Use Case ID: UC-PRIVATECHAT
Use Case Name: Making a private chat
Relevant Requirements: Once in the application, the user can find other users and start a private chat.
Primary Actor: User (IT Admin and Employee)
Pre-conditions: User must be logged into the application.
Post-conditions: The user can now create a private one on one chat with another user.
Basic Flow or Main Scenario: 1. User first logs into their account. 2. Can search for other employees. 3. Select who to chat with. 4. Then create chat.
Extensions or Alternate Flows: typing name wrong in the search for other employees. Can't find employees in the search.
Exceptions: Adding employees that are fired.
Related Use Cases: UC-LOGIN

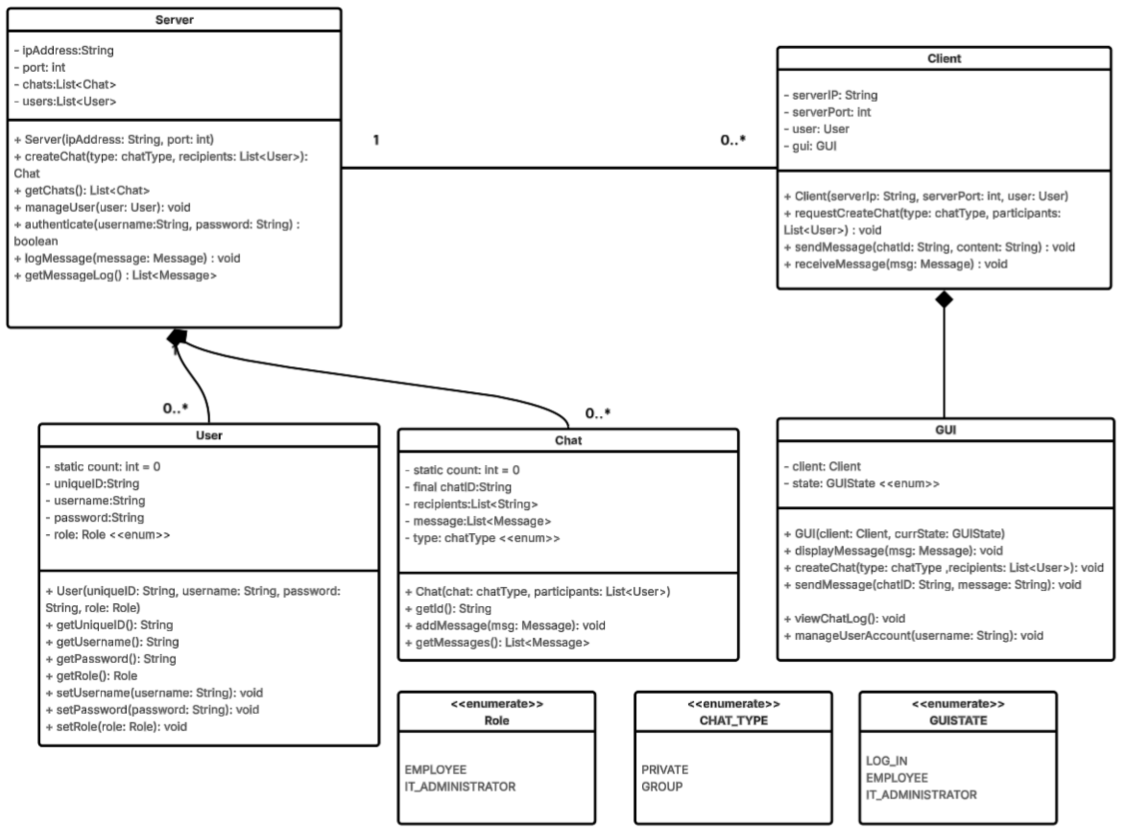
Use Case ID: UC-CHAT_LOGS
Use Case Name: IT Reviews Chat Logs
Relevant Requirements: All messages are logged. IT can view all conversations.
Primary Actor: IT Administrator
Pre-conditions: IT user is authenticated.
Post-conditions: IT admin views the chat logs
Basic Flow or Main Scenario: 1. IT user logs in and has a view of the chat dashboard like other users alongside the chat log view. 2. IT can select the user or chat name to view. 3. The system will fetch the user or chat that the IT wants to view, and it will show the chat history. 4. IT can filter by using user, chat name, date, timestamp. 5. The views for IT will be logged.
Extensions or Alternate Flows: IT can review all group or private messages.
Exceptions: If the system crashes and the log service is not available. If the way they are searching is invalid.
Related Use Cases: UC-LOGIN, UC-MANAGE, UC-GROUPCHAT

Use Case ID: UC-SEND_MESSAGE
Use Case Name: Send Message
Relevant Requirements: User must be logged in. There must be other existing users. The server must log all messages.
Primary Actor: User (IT Admin and Employee)
Pre-conditions: The User client is connected to the server. The User is signed in. At least one valid recipient.
Post-conditions: The message is transmitted to the intended recipient. The recipient receives the message in their chat window. The message is logged.
Basic Flow or Main Scenario: 1: User chooses a chat recipient (private or group). 2: User writes their message. 3: User clicks send or ENTER/RETURN on keyboard. 4: Client forwards the message to the server. 5: Server validates the recipient and routes the message. 6: Recipient client(s) receive and display the message. 7: Server logs the message
Extensions or Alternate Flows: User selects a group chat as the recipient; therefore, the server will have to broadcast the message to all group members.
Exceptions: If user is not found, the server will reply with an error. If the connection is lost before the message was received by the server, then an error will be displayed. If the message is empty or contains non-text, then an error will be displayed.
Related Use Cases: UC-LOGIN, UC-CHAT_LOGS, UC-GROUPCHAT, UC-PRIVATECHAT

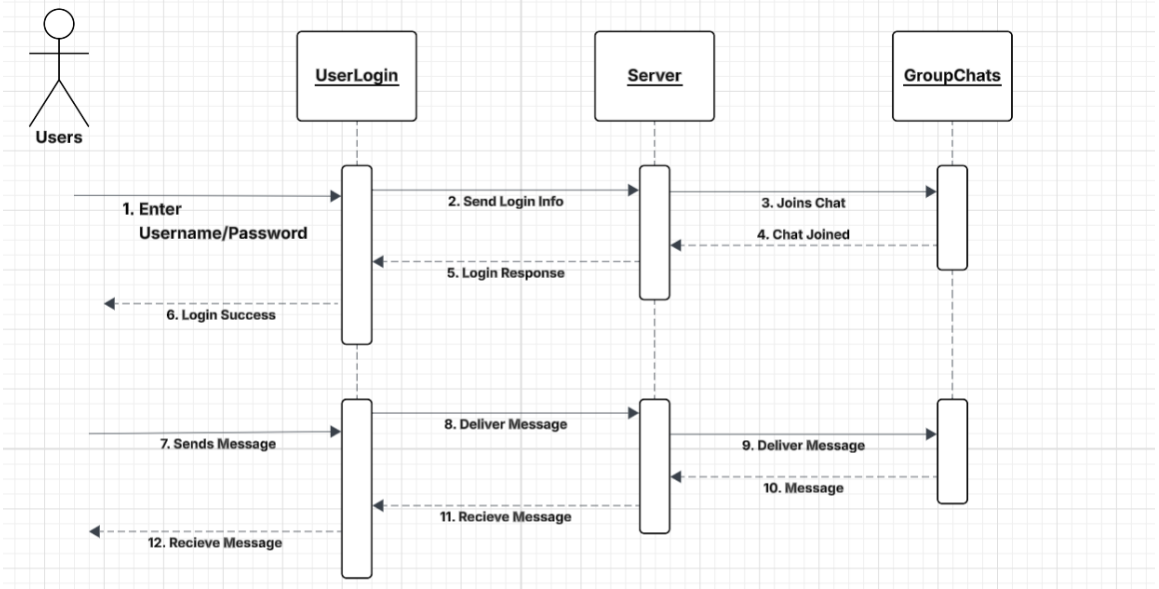
6. UML USE CASE DIAGRAM

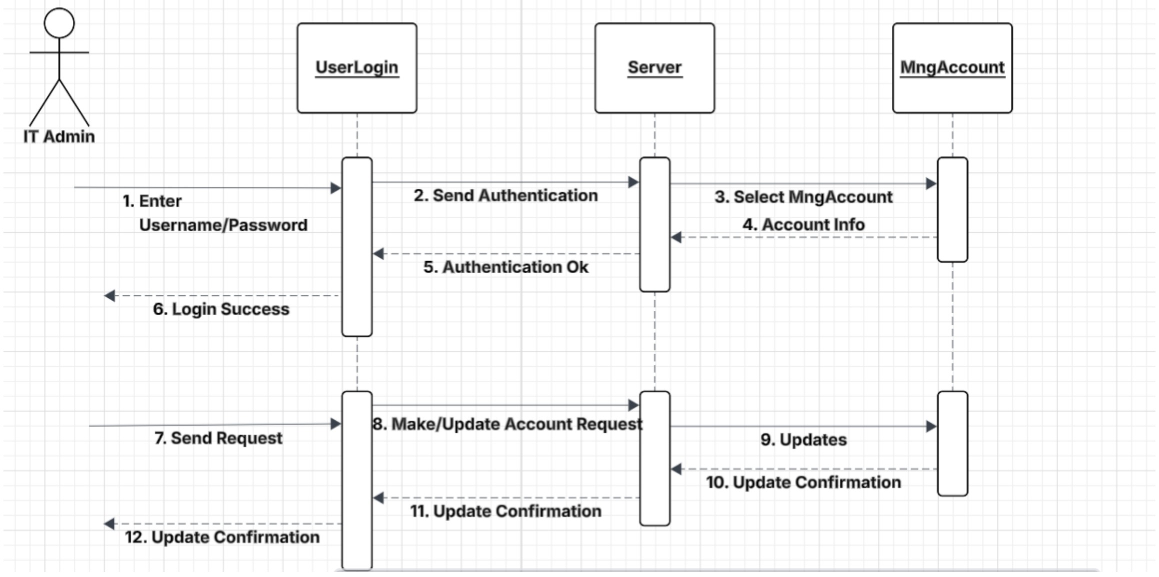


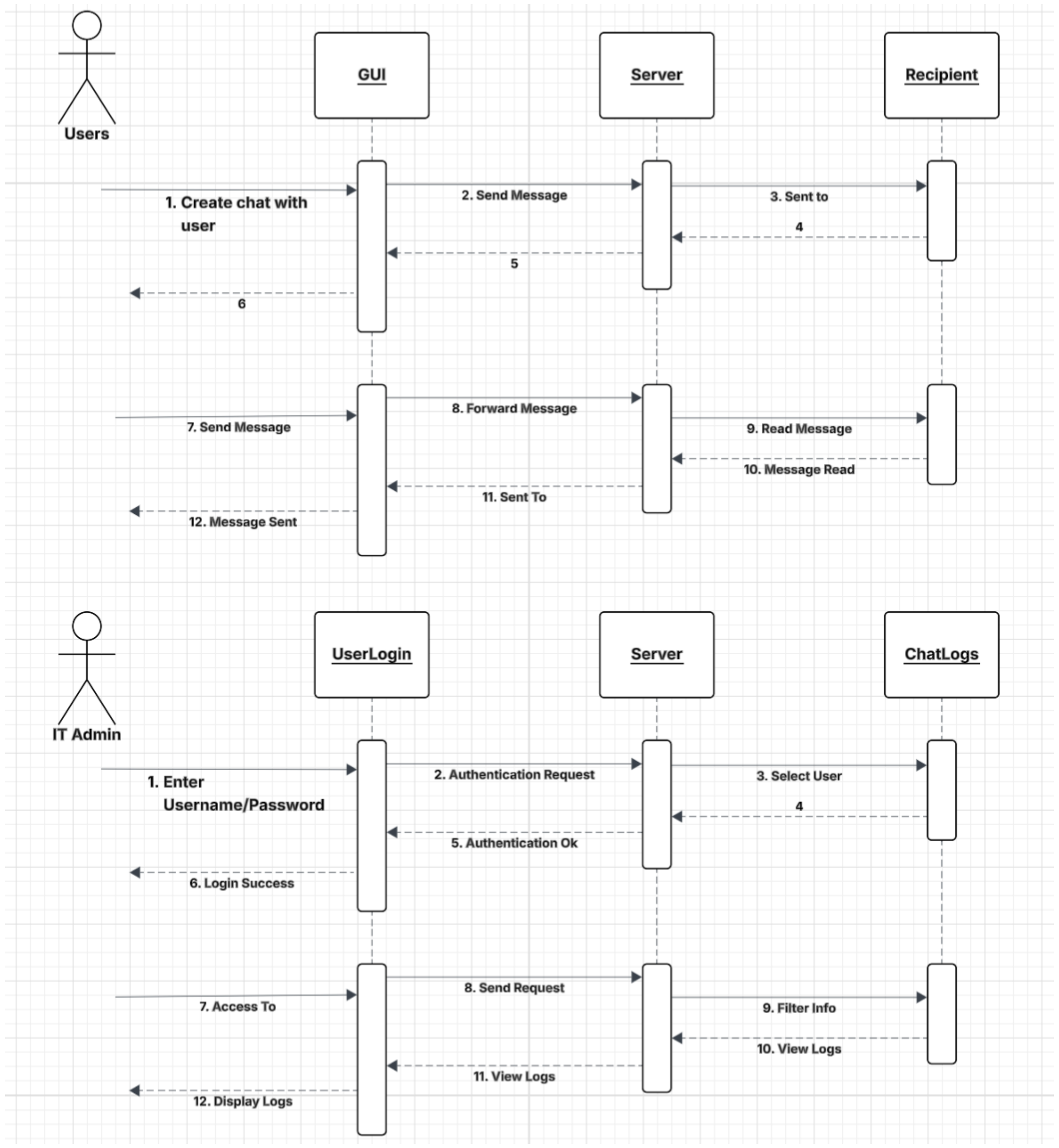
7. CLASS DIAGRAMS


Server
- ipAddress:String
- port: int
- chats:List<Chat>
- users:List<User>
+ Server(ipAddress: String, port: int)
+ createChat(type: chatType, recipients: List<User>): Chat
+ getChats(): List<Chat>
+ manageUser(user: User): void
+ authenticate(username:String, password: String) : boolean
+ logMessage(message: Message) : void
+ getMessageLog() : List<Message>
1
0..*
Client
- serverIP: String
- serverPort: int
- user: User
- gui: GUI
+ Client(serverIp: String, serverPort: int, user: User)
+ requestCreateChat(type: chatType, participants: List<User>) : void
+ sendMessage(chatId: String, content: String) : void
+ receiveMessage(msg: Message) : void
0..*
0..*
User
- static count: int = 0
- uniqueID:String
- username:String
- password:String
- role: Role <<enum>>
+ User(uniqueID: String, username: String, password: String, role: Role)
+ getUniqueID(): String
+ getUsername(): String
+ getPassword(): String
+ getRole(): Role
+ setUsername(username: String): void
+ setPassword(password: String): void
+ setRole(role: Role): void
Chat
- static count: int = 0
- final chatID:String
- recipients:List<String>
- message:List<Message>
- type: chatType <<enum>>
+ Chat(chat: chatType, participants: List<User>)
+ getId(): String
+ addMessage(msg: Message): void
+ getMessages(): List<Message>
GUI
- client: Client
- state: GUIState <<enum>>
+ GUI(client: Client, currState: GUIState)
+ displayMessage(msg: Message): void
+ createChat(type: chatType ,recipients: List<User>): void
+ sendMessage(chatID: String, message: String): void
+ viewChatLog(): void
+ manageUserAccount(username: String): void
<<enumerate>>
Role
EMPLOYEE
IT_ADMINISTRATOR
<<enumerate>>
CHAT_TYPE
PRIVATE
GROUP
<<enumerate>>
GUISTATE
LOG_IN
EMPLOYEE
IT_ADMINISTRATOR

8. SEQUENCE DIAGRAMS






Users
GUI
Server
Recipient
1. Create chat with user
2. Send Message
3. Sent to
4
5
6
7. Send Message
8. Forward Message
9. Read Message
10. Message Read
11. Sent To
12. Message Sent
IT Admin
UserLogin
Server
ChatLogs
1. Enter Username/Password
2. Authentication Request
3. Select User
4
5. Authentication Ok
6. Login Success
7. Access To
8. Send Request
9. Filter Info
10. View Logs
11. View Logs
12. Display Logs