Communications Project
Group 4

Software Requirements Specification

Revision History

Date	Revision	Description	Author
9/26/23	1.00	Initial Version	Benjamin Newton Aaron Nguyen Kosta Nikolaou Jagjot Nijjar
10/3/23	1.01	Section 1.3 - Added Use Case Specifications and Diagram	Aaron Nguyen
10/3/23	1.02	Section 1.3 - Added Class Diagram	Aaron Nguyen
10/3/23	1.03	Section 1.3 - Added Sequence Diagram	Aaron Nguyen
10/4/23	1.04	Section 1.3 - Modify Class Diagram	Aaron Nguyen
10/4/23	1.05	Section 3 - Added Specific Requirements	Benjamin Newton
10/5/23	1.06	Section 4 - Added Non-Functional Requirements	Benjamin Newton
10/10/23	1.07	Section 1.3 - Modify Class Diagram Section 3.1 - Add functional requirements	Aaron Nguyen Jagjot Nijjar Kosta Nikolaou
10/17/23	1.08	Section 2.1 - Add product perspective Section 3.2, 3.3, 4.2, 4.3 - Add additional requirements	Benjamin Newton Aaron Nguyen Jagjot Nijjar Kosta Nikolaou
10/24/23	1.09	Section 1.3 - Update UML class diagrams and sequence diagrams	Benjamin Newton Aaron Nguyen Jagjot Nijjar Kosta Nikolaou

Table of Contents

1. Purpose

This document outlines the requirements for our Communication System (CS).

1.1. Scope

This document will catalog the user, system, and hardware requirements for the Communication System.. It will not, however, document how these requirements will be implemented.

1.2. Definitions, Acronyms, Abbreviations

CS - Communication System

1.3. References

Use Case Specifications

Use Case ID: 1

Use Case Name: Login

Relevant Requirements: N/A

Primary Actor: User

Pre-conditions:

1. Server is running
2. Client is connected to the server
3. User has an account registered in the system

Post-conditions: User is logged into the messaging system client

Basic Flow or Main Scenario:

1. User enters their username and password in the client
2. Client sends a login request to the server
3. Server authenticates the user
4. On successful authentication, the server responds with a success indicator

Extensions or Alternate Flows:

4. On unsuccessful authentication, the server responds with failure indicator
5. User attempts to fix incorrect credentials and attempts login again

Exceptions: N/A

Related Use Cases: Creating a room (2), Selecting a room (3), Sending a message (4), Updating user status (5)

Use Case ID: 2

Use Case Name: Creating a room

Relevant Requirements: Login (1)

Primary Actor: User

Pre-conditions:

1. Client is connected to the server
2. User is logged into the client

Post-conditions: A room is created where the user can message other user(s)

Basic Flow or Main Scenario:

1. User clicks on a button in the client to create a new room
2. User selects participants to be invited to the room
3. If only one participant is selected, no room name is required

Extensions or Alternate Flows:

3. If more than one participant is selected, the user must enter a name for the room

Exceptions: N/A

Related Use Cases: Sending a message (4)

Use Case ID: 3

Use Case Name: Selecting a room

Relevant Requirements: Login (1)

Primary Actor: User

Pre-conditions:

1. Client is connected to the server
2. User is logged into the client

Post-conditions: The selected room is displayed in the user's client

Basic Flow or Main Scenario:

1. After logging into the client, the user is presented with a list of rooms they are a participant in
2. Clicking on a room displays the room in their client which includes messages from others and a list of participants

Extensions or Alternate Flows: N/A

Exceptions: If no room is available, the list of rooms will be empty

Related Use Cases: Sending a message (4)

Use Case ID: 4

Use Case Name: Sending a message

Relevant Requirements: Login (1)

Primary Actor: User

Pre-conditions:

1. Client is connected to the server
2. User is logged into the client
3. User has a room selected

Post-conditions: A message is sent to the appropriate group

Basic Flow or Main Scenario:

1. After entering a room, the user is presented with a text field
2. The user can input text into this field and press 'Enter' to send the message
3. If the message is successfully delivered, they are presented with a success indicator

Extensions or Alternate Flows:

4. If the message fails to be delivered, they are presented with a failure indicator

Exceptions: N/A

Related Use Cases: Selecting a room (3)

Use Case ID: 5

Use Case Name: Updating user status

Relevant Requirements: Login (1)

Primary Actor: User

Pre-conditions:

1. Client is connected to the server
2. User is logged into the client

Post-conditions: The status of the user is updated

Basic Flow or Main Scenario:

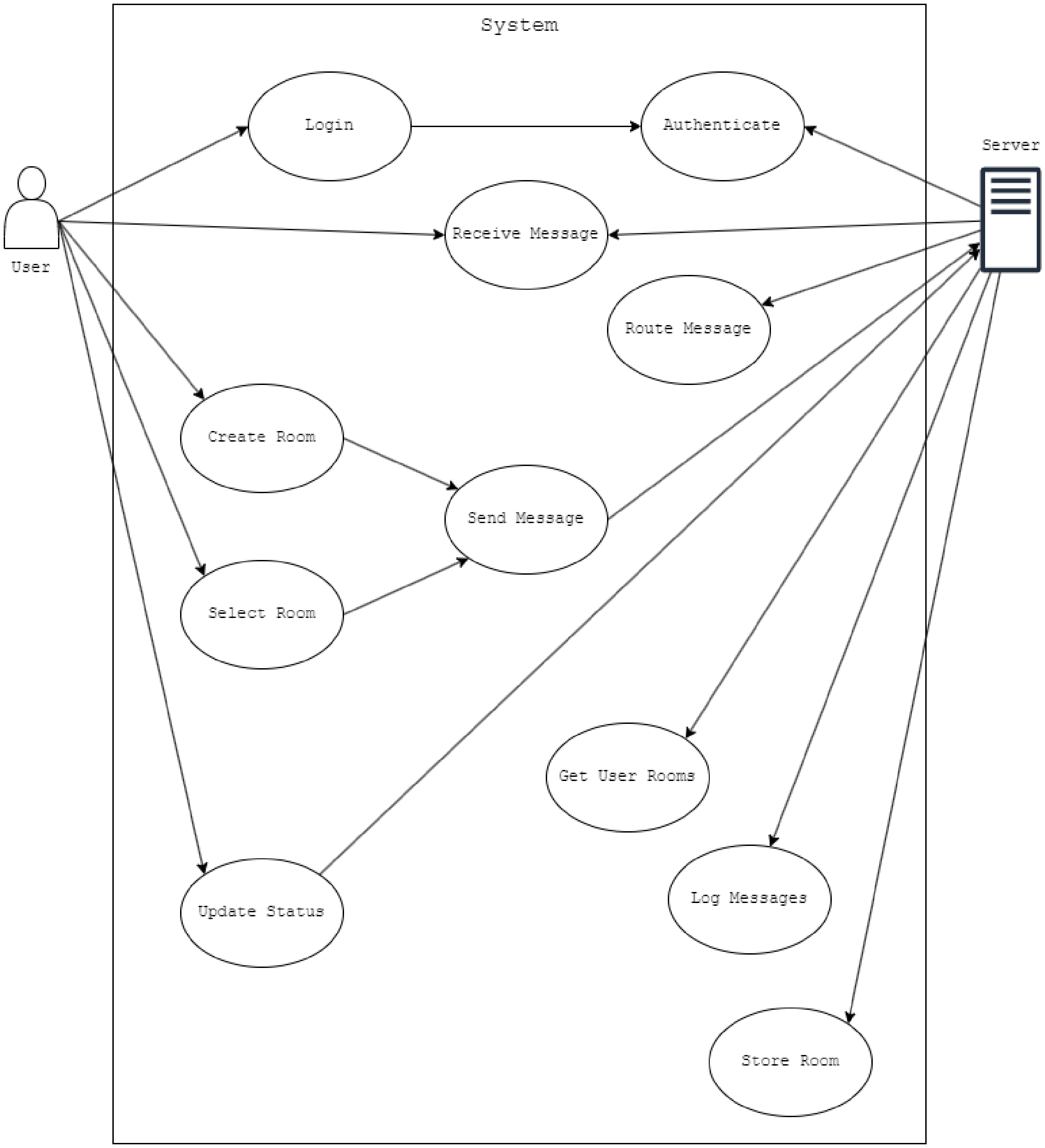
1. After logging into the client, the user can see a "Change Status" button
2. Clicking this button opens a dropdown menu with a list of statuses
3. Clicking on a status will update their status to their selection
4. Other users will be able to see the updated status

Extensions or Alternate Flows: N/A

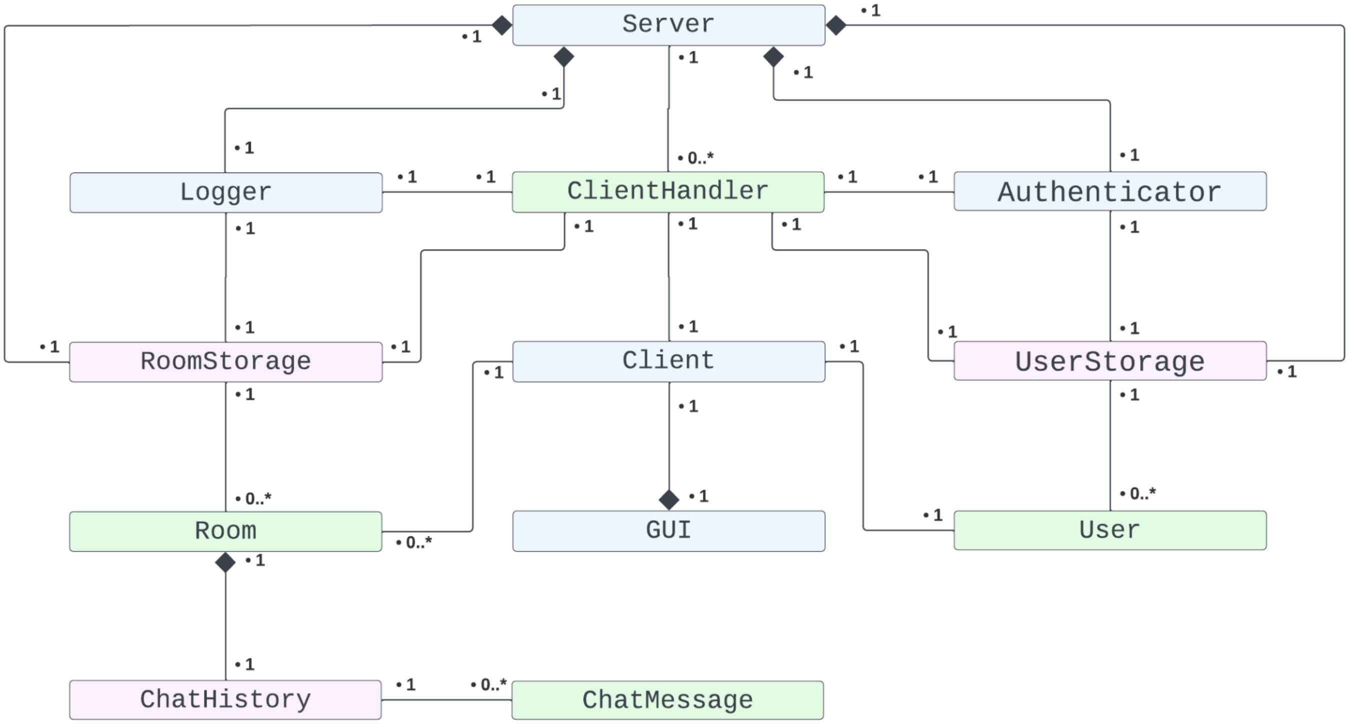
Exceptions: The 'Offline' status is available internally but is not available for selection by the user

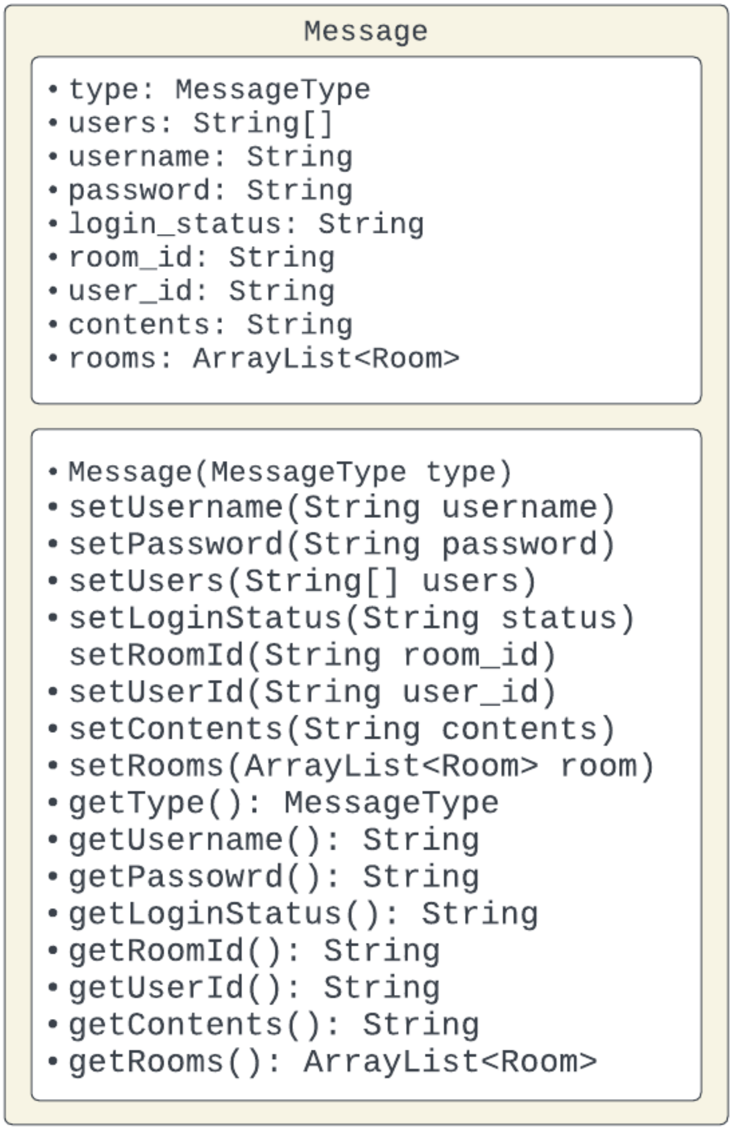
Related Use Cases: N/A

Use Case Diagram



Class Diagrams

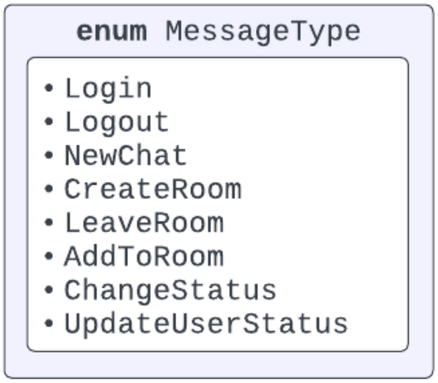




Message

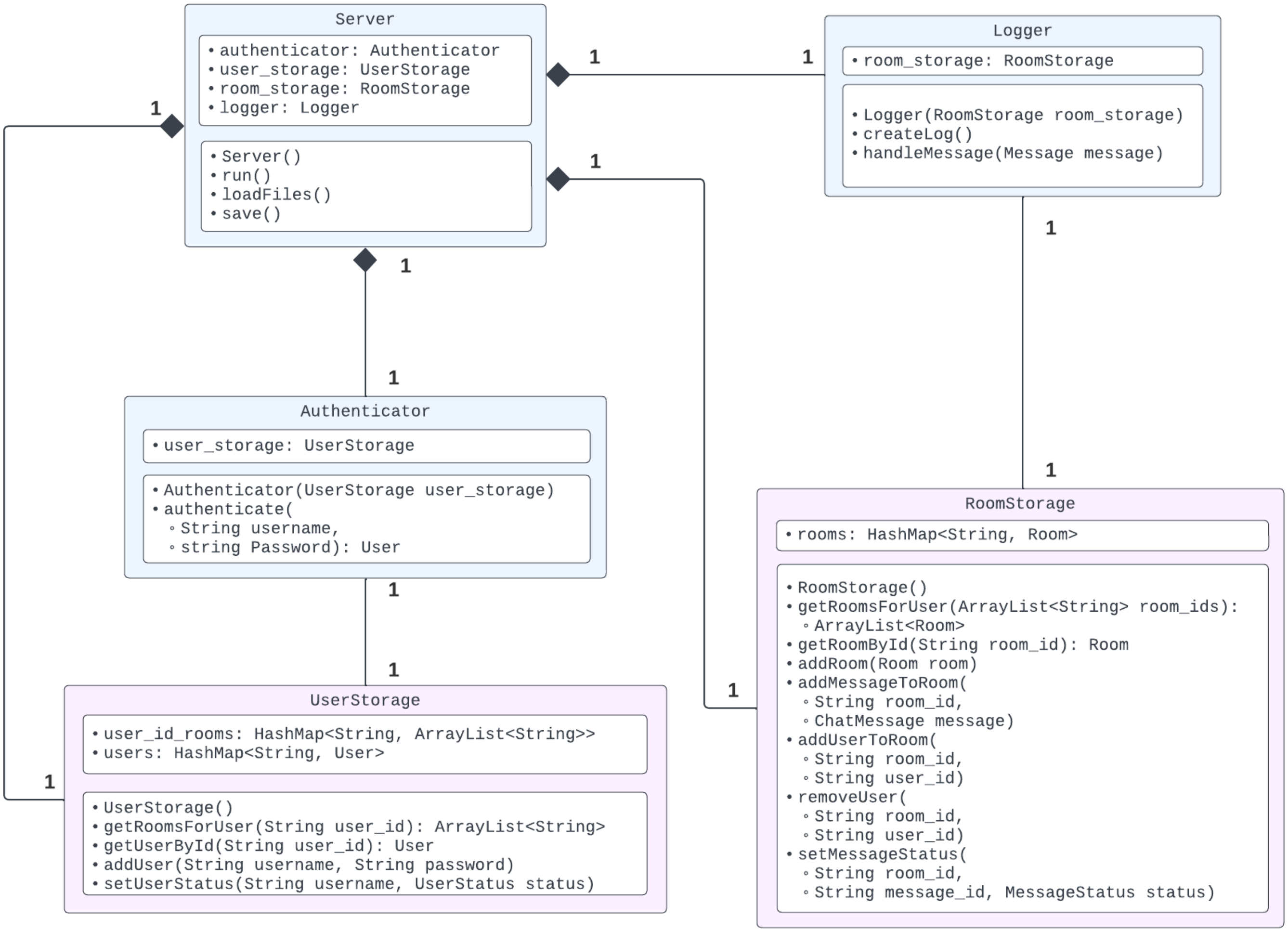
- type: MessageType
- users: String[]
- username: String
- password: String
- login_status: String
- room_id: String
- user_id: String
- contents: String
- rooms: ArrayList<Room>

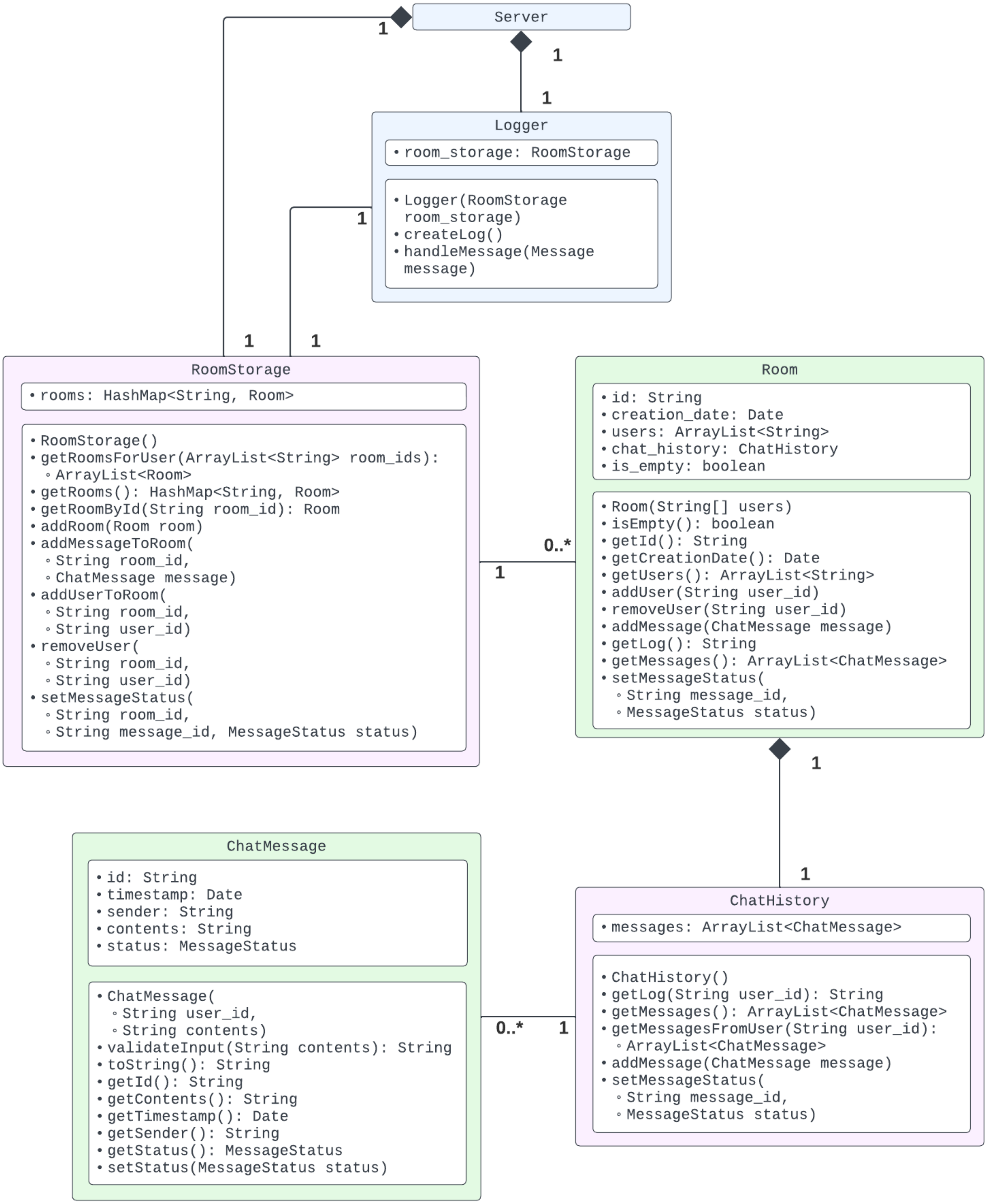
- Message(MessageType type)
- setUsername(String username)
- setPassword(String password)
- setUsers(String[] users)
- setLoginStatus(String status)
  setRoomId(String room_id)
- setUserId(String user_id)
- setContents(String contents)
- setRooms(ArrayList<Room> room)
- getType(): MessageType
- getUsername(): String
- getPassowrd(): String
- getLoginStatus(): String
- getRoomId(): String
- getUserId(): String
- getContents(): String
- getRooms(): ArrayList<Room>



enum MessageType

- Login
- Logout
- NewChat
- CreateRoom
- LeaveRoom
- AddToRoom
- ChangeStatus
- UpdateUserStatus


Server
• authenticator: Authenticator
• user_storage: UserStorage
• room_storage: RoomStorage
• logger: Logger
• Server()
• run()
• loadFiles()
• save()
Logger
• room_storage: RoomStorage
• Logger(RoomStorage room_storage)
• createLog()
• handleMessage(Message message)
Authenticator
• user_storage: UserStorage
• Authenticator(UserStorage user_storage)
• authenticate(
◦ String username,
◦ string Password): User
RoomStorage
• rooms: HashMap<String, Room>
• RoomStorage()
• getRoomsForUser(ArrayList<String> room_ids):
◦ ArrayList<Room>
• getRoomById(String room_id): Room
• addRoom(Room room)
• addMessageToRoom(
◦ String room_id,
◦ ChatMessage message)
• addUserToRoom(
◦ String room_id,
◦ String user_id)
• removeUser(
◦ String room_id,
◦ String user_id)
• setMessageStatus(
◦ String room_id,
◦ String message_id, MessageStatus status)
UserStorage
• user_id_rooms: HashMap<String, ArrayList<String>>
• users: HashMap<String, User>
• UserStorage()
• getRoomsForUser(String user_id): ArrayList<String>
• getUserById(String user_id): User
• addUser(String username, String password)
• setUserStatus(String username, UserStatus status)
1
1
1
1
1
1
1
1
1
1
1
1
1
1



```
Server
```

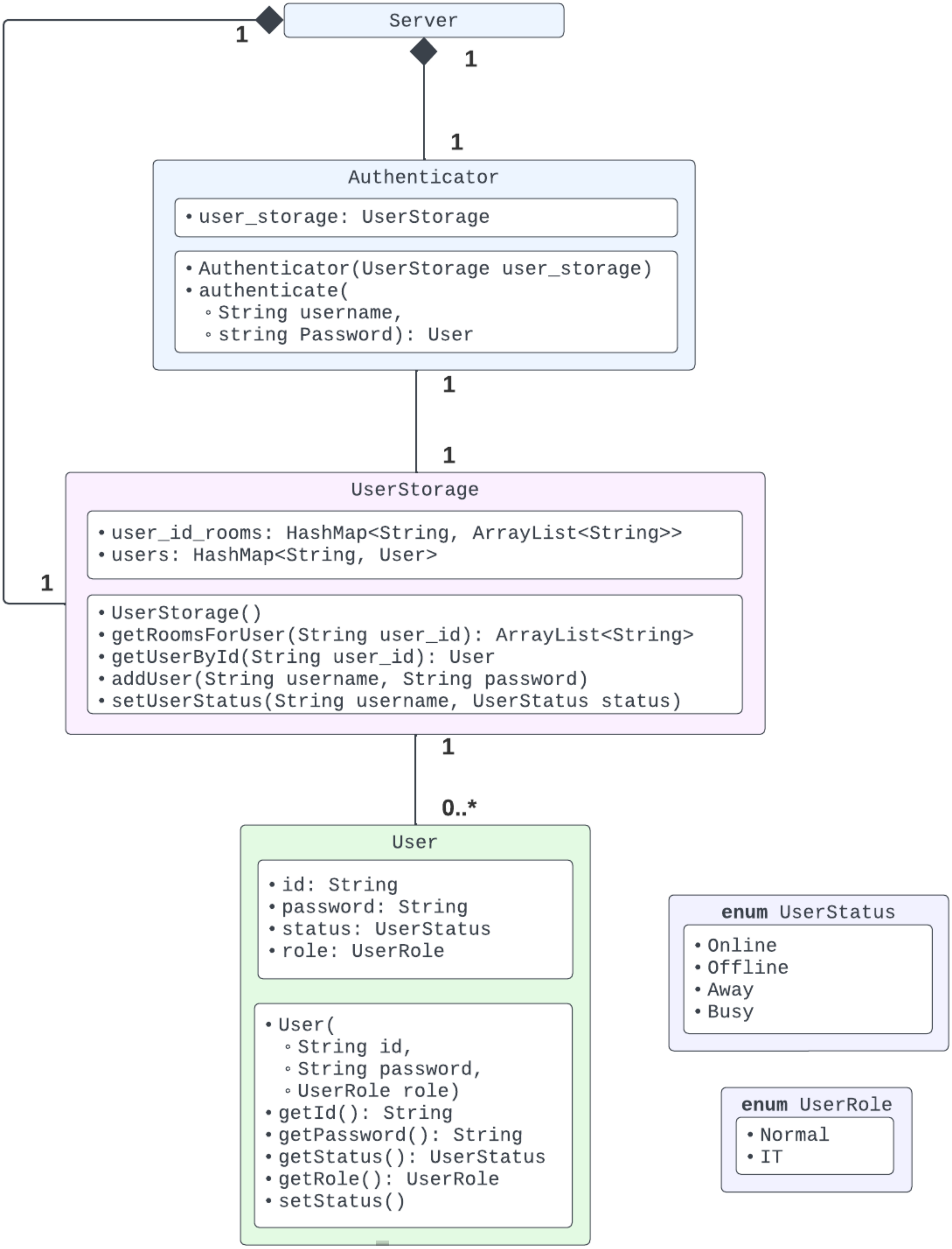
```
Logger
• room_storage: RoomStorage

• Logger(RoomStorage
  room_storage)
• createLog()
• handleMessage(Message
  message)
```

```
RoomStorage
• rooms: HashMap<String, Room>

• RoomStorage()
• getRoomsForUser(ArrayList<String> room_ids):
   ◦ ArrayList<Room>
• getRooms(): HashMap<String, Room>
• getRoomById(String room_id): Room
• addRoom(Room room)
• addMessageToRoom(
   ◦ String room_id,
   ◦ ChatMessage message)
• addUserToRoom(
   ◦ String room_id,
   ◦ String user_id)
• removeUser(
   ◦ String room_id,
   ◦ String user_id)
• setMessageStatus(
   ◦ String room_id,
   ◦ String message_id, MessageStatus status)
```

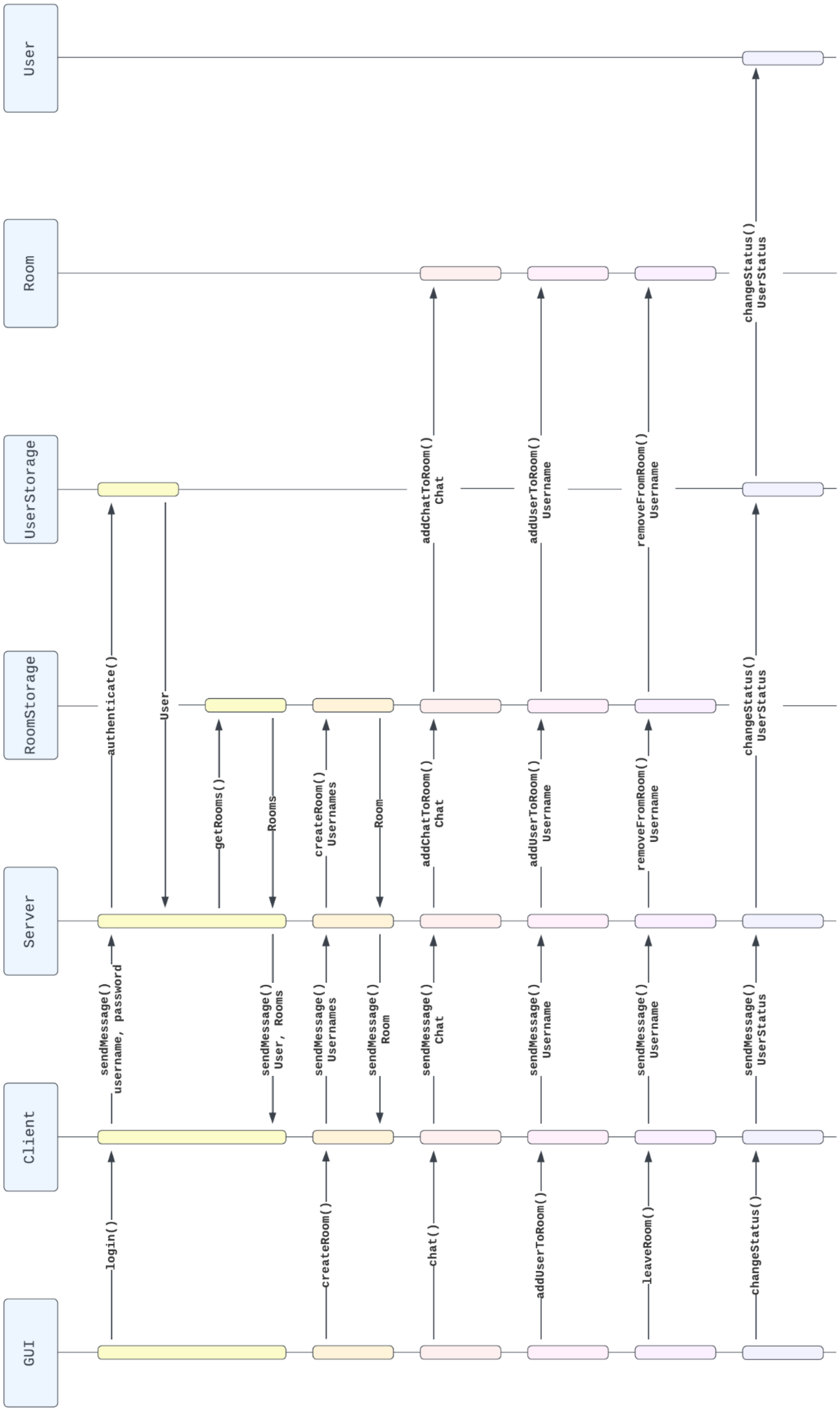
```
Room
• id: String
• creation_date: Date
• users: ArrayList<String>
• chat_history: ChatHistory
• is_empty: boolean

• Room(String[] users)
• isEmpty(): boolean
• getId(): String
• getCreationDate(): Date
• getUsers(): ArrayList<String>
• addUser(String user_id)
• removeUser(String user_id)
• addMessage(ChatMessage message)
• getLog(): String
• getMessages(): ArrayList<ChatMessage>
• setMessageStatus(
   ◦ String message_id,
   ◦ MessageStatus status)
```

```
ChatMessage
• id: String
• timestamp: Date
• sender: String
• contents: String
• status: MessageStatus

• ChatMessage(
   ◦ String user_id,
   ◦ String contents)
• validateInput(String contents): String
• toString(): String
• getId(): String
• getContents(): String
• getTimestamp(): Date
• getSender(): String
• getStatus(): MessageStatus
• setStatus(MessageStatus status)
```

```
ChatHistory
• messages: ArrayList<ChatMessage>

• ChatHistory()
• getLog(String user_id): String
• getMessages(): ArrayList<ChatMessage>
• getMessagesFromUser(String user_id):
   ◦ ArrayList<ChatMessage>
• addMessage(ChatMessage message)
• setMessageStatus(
   ◦ String message_id,
   ◦ MessageStatus status)
```


Server
1
1
1
Authenticator
• user_storage: UserStorage
• Authenticator(UserStorage user_storage)
• authenticate(
◦ String username,
◦ string Password): User
1
1
UserStorage
1
• user_id_rooms: HashMap<String, ArrayList<String>>
• users: HashMap<String, User>
• UserStorage()
• getRoomsForUser(String user_id): ArrayList<String>
• getUserById(String user_id): User
• addUser(String username, String password)
• setUserStatus(String username, UserStatus status)
1
0..*
User
• id: String
• password: String
• status: UserStatus
• role: UserRole
• User(
◦ String id,
◦ String password,
◦ UserRole role)
• getId(): String
• getPassword(): String
• getStatus(): UserStatus
• getRole(): UserRole
• setStatus()
enum UserStatus
• Online
• Offline
• Away
• Busy
enum UserRole
• Normal
• IT

Sequence Diagram



1.4. Overview

The Communication system will allow users to message people directly or groups of people through creating rooms.

2. Overall Description

2.1. Product Perspective

The product consists of a communication system that permits asynchronous and synchronous messaging between clients. Users will be able to message directly as well as in groups and IT users will be able to view all logs.

2.2. Product Architecture

This system will be broken into three main modules: The server, the client, and the room

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): Users will be able to message other users either directly or in a group by creating a room. Rooms will handle synchronous and asynchronous messaging as well as having a history function.

2.4. Constraints

Users will only be able to send other users text. Files will not be supported at this time.

2.5. Assumptions and Dependencies

It is assumed that users will not spam other users in rooms as there will not be strict spam filtering on rooms.

Users are dependent on themselves to know other people to send messages too, there will not be a social network aspect of the communication system.

3. Specific Requirements

3.1. Functional Requirements

3.1.1. Common Requirements:

3.1.1.1 Users should be able to message other users.

3.1.1.2 Users should be able to see when a message was sent.

3.1.1.3 Users should be able to see who sent a message.

3.1.1.4 Users should be able to see a list of users available to message.

3.1.1.5 Users should be able to see a list of groups they are in.

3.1.1.6 Users should be able to create groups.

3.1.1.7 Users should be able to search for messages or content within messages.

3.1.1.8 Users should be able to set a status indicating their availability or presence.

3.1.2. Server Module Requirements:

3.1.2.1 The server should authenticate users trying to log in from the client.

3.1.2.2 The server should create rooms for users when a new message room is established

3.1.2.3 The server should handle delivering messages to a room.

3.1.2.4 The server should keep an array of all the currently created rooms

3.1.2.5 The server should allow multiple users to be online and send messages at the same time

3.1.3. Client Module Requirements:

3.1.3.1 Users should be allowed to log in using their issued id and password, both of which are alphanumeric strings between 6 and 20 characters in length.

3.1.3.2 In the client, users should be allowed to see a list of rooms that they have already sent messages in.

3.1.3.3 In the client the user should be allowed to see a user list of who is currently connected to the server.

3.1.3.4 Users should receive all of the messages that they missed while they were logged out of the system.

3.1.4. Room Module Requirements:

3.1.4.1 In a room the user should be able to see a history of all messages sent before in that room

3.1.4.2 The user should be able to add new messages to any room they are a part of.

3.1.4.3 The room should keep a history of every message for IT level users.

3.2. External Interface Requirements

3.2.1 The system will need to be opened each time that the computer is turned on. It will not open on startup.

3.2.2 The system must alert users when a new message appears with a sound.

3.2.3 The client application should provide a text field for input.

3.2.4 The client should have a notification system.

3.3. Internal Interface Requirements

3.3.1 The system must allow an IT user to see all of the history of all users.

3.3.2 The system must have access to the filesystem for storing files.

3.3.3 The server application should return a list of all rooms a user is a participant in upon login

4. Non-Functional Requirements

4.1. Security and Privacy Requirements

4.1.1 The system must always send data using TCP/IP.
4.1.2 Server securely stores user passwords.
4.1.3 Messages between users should not be encrypted.
4.1.4 Information between the server and client is not encrypted.

4.2. Environmental Requirements

4.2.1 System cannot require any other software in order to run properly.
4.2.2 The server application must be running on a computer connected to the internet
4.2.3 Any client application(s) must be running a computer connected to the internet

4.3. Performance Requirements

4.3.1 The client must be able to open up to the login screen in less than 5 seconds.
4.3.2 The system must route messages in less than 2 seconds.
4.3.3 The system must authenticate a user in less than 2 seconds.