Communications System

Design Document

Github Link:
https://github.com/elle-zej/CommunicationsSystem

Revision History

Date	Revision	Description	Author
03/23/25	1.0	Initial Version - format - class descriptions - use cases	Jezelle Overstreet
03/23/25	1.1	Added few Class Descriptions	Sushant
04/06/25	1.2	Revision on Class Descriptions - Employee Class contains a role enum	Jezelle Overstreet
04/06/25	1.2.1	Sequence diagram -revised sending message diagram -added new sequence diagrams	Sushant
04/08/25	1.2.2	Added use case diagram Revised Class Descriptions Updated table of contents	Jezelle Overstreet
04/09/25	1.2.3	Fixed a little error on the sequence diagram -Authentication done on the server	Sushant
04/09/25	1.2.4	Revisions on use cases	Jezelle Overstreet
04/10/25	1.3	Class Diagram Use Case Diagram Github Link	Jezelle Overstreet
4/10/25	1.4	Added UI Prototype	Timothy Miu

Table of Contents

1. Purpose

This document outlines the requirements for the Communications System.

1.1. Scope

This document will describe the communication system design and basic flow of use cases.

1.2. Definitions, Acronyms, Abbreviations

User: Employee or IT member within the communication system.

Employee: Standard user, holds no special privileges.

IT: Member of information technology team, has administrative privileges.

Group: Any congregation of two or more users.

Asynchronous: Messages can be delivered in such a way that they “persist” even when one user is not online. In other words, they work in a similar fashion to text messaging; there is a mechanism that allows propagation of messages even when communication is not immediate between the two users.

1.3. Overview

The communications system is a platform designed for users to send messages to and receive messages from other users. Users are able to login and chat with other users privately or in groups. The messages between users are only text.

2. Overall Design and Classes

2.1. Product Architecture/Modules

The system will be organized into 3 major modules: the user module, the messaging module, and the server/client module

2.2. 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):

1. Send and receive messages privately or in groups synchronously and asynchronously

2.3. Constraints

Messages sent by all users are immutable

Employees are only able to view their own chat logs

Messages sent are only allowed to be text

Messages are not stored in an external database

No databases, libraries, frameworks, or other technologies may be used

2.4. Design Pattern

2.4.1. Server-Client Design Pattern

Features both multithreaded server and client

2.5. Classes

2.5.1. Server

Handles all requests from client
Routes clients and messages to proper destinations
Handles multiple clients at the same time
ClientHandler

2.5.2. Client

Allows users to log into the communications system using a unique ID and password
Sends requests for login and messaging to the server

2.5.3. Message

A message object will have information about the sender, receiver, date/time the message was sent and will also contain the content of the message the user intends to send

2.5.4. User

A User is a member of the communication group. Each user is assigned a role (enum), Employee or IT Manager. Each user has the ability to send/receive messages, create groups and view their own chat logs. Each user will have attributes such as their name, Employee ID, Password and the conversations they are a part of. The only difference between an employee and an IT manager is that an IT manager has administrative privileges. This is an additional method that gives the IT manager the ability to view the chat history of all the employees in the company.

2.5.5. *Group*

Class that keeps track of all members of a conversation and creates a unique ID for each conversation

2.5.6. *Conversation*

A conversation is a collection of messages for any group of two or more people which is a collection of messages in the chronological order they were sent.

2.5.7. *ConversationStorage*

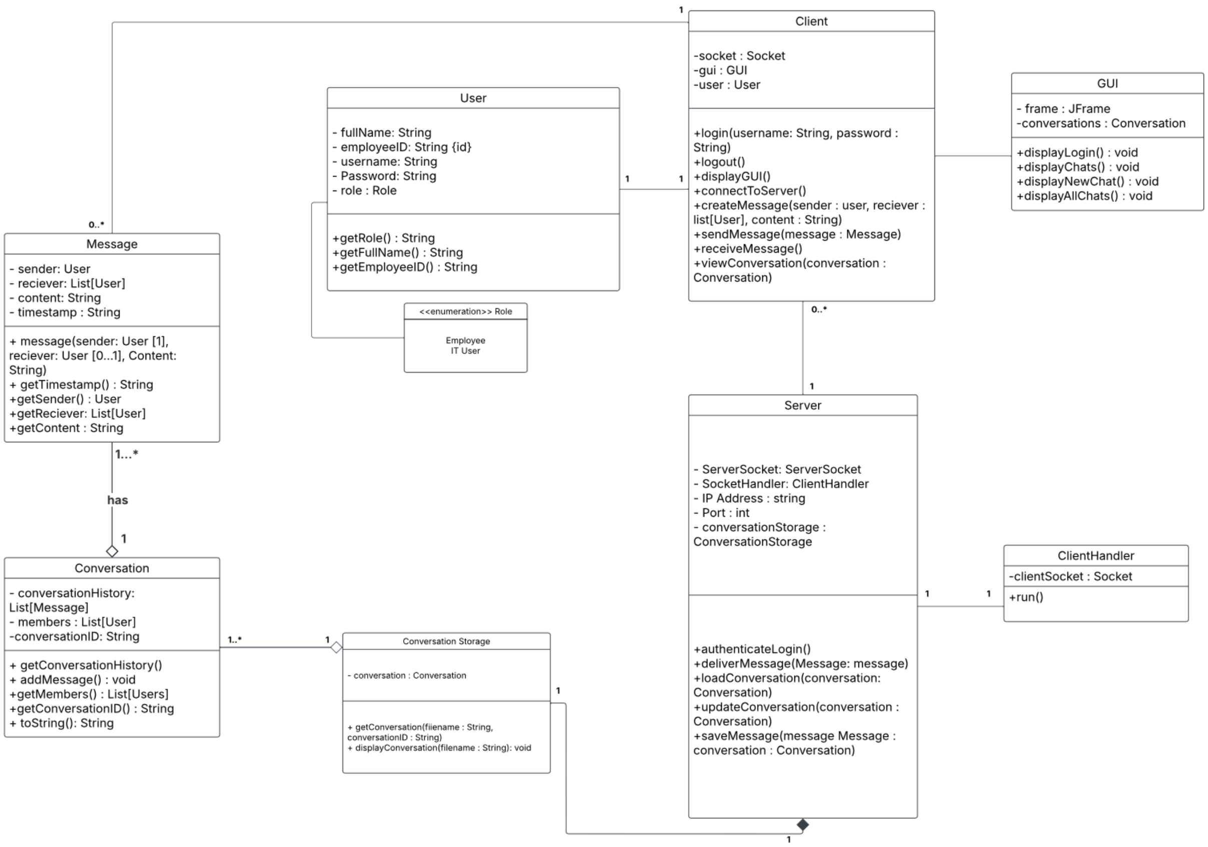
Retrieves a conversation ID
Saves Conversations and messages
Saves each conversation and message in a text file
Each text file corresponds to a different conversation
Function to display the conversation

2.5.8. *CommunicationGUI*

Handles all logic that has to do with the GUI. Creates buttons that users will interact with to view conversations and create conversations. GUI also allows users to scroll through messages.

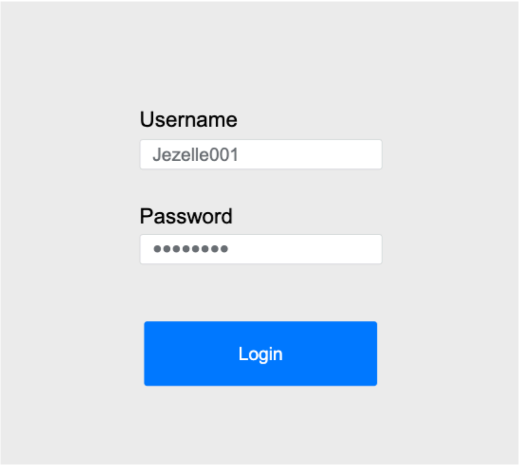
- Login Page
- Messaging page that features each conversation and a create new message button
- Message button prompts user to enter recipients (if multiple recipients, then a group is created) and message content

2.6. Class Diagram


Client
-socket : Socket
-gui : GUI
-user : User
+login(username: String, password : String)
+logout()
+displayGUI()
+connectToServer()
+createMessage(sender : user, reciever : list[User], content : String)
+sendMessage(message : Message)
+receiveMessage()
+viewConversation(conversation : Conversation)
GUI
- frame : JFrame
-conversations : Conversation
+displayLogin() : void
+displayChats() : void
+displayNewChat() : void
+displayAllChats() : void
User
- fullName: String
- employeeID: String {id}
- username: String
- Password: String
- role : Role
+getRole() : String
+getFullName() : String
+getEmployeeID() : String
<<enumeration>> Role
Employee
IT User
Message
- sender: User
- reciever: List[User]
- content: String
- timestamp : String
+ message(sender: User [1], reciever: User [0...1], Content: String)
+ getTimestamp() : String
+getSender() : User
+getReciever: List[User]
+getContent : String
1...*
has
1
Conversation
- conversationHistory: List[Message]
- members : List[User]
-conversationID: String
+ getConversationHistory()
+ addMessage() : void
+getMembers() : List[Users]
+getConversationID() : String
+ toString(): String
1..*
1
Conversation Storage
- conversation : Conversation
+ getConversation(fiiename : String, conversationID : String)
+ displayConversation(filename : String): void
1
1
Server
- ServerSocket: ServerSocket
- SocketHandler: ClientHandler
- IP Address : string
- Port : int
- conversationStorage : ConversationStorage
+authenticateLogin()
+deliverMessage(Message: message)
+loadConversation(conversation: Conversation)
+updateConversation(conversation : Conversation)
+saveMessage(message Message : conversation : Conversation)
ClientHandler
-clientSocket : Socket
+run()
1
1
0..*
1
1
1
0..*
1

2.7. UI Prototype

Login screen:



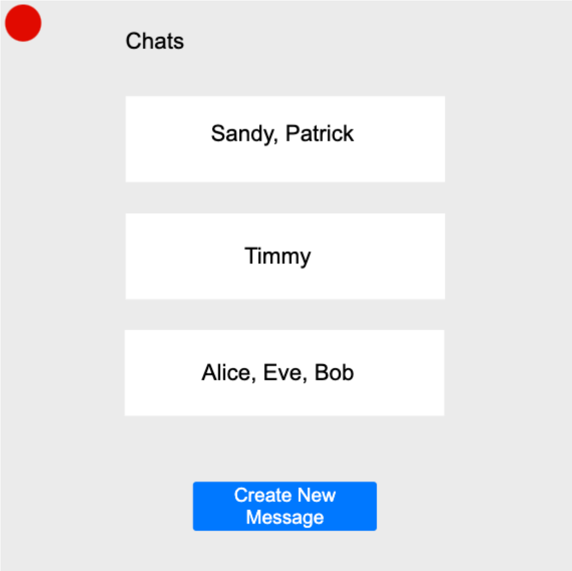
Username

Jezelle001

Password

••••••••

Viewing chats:

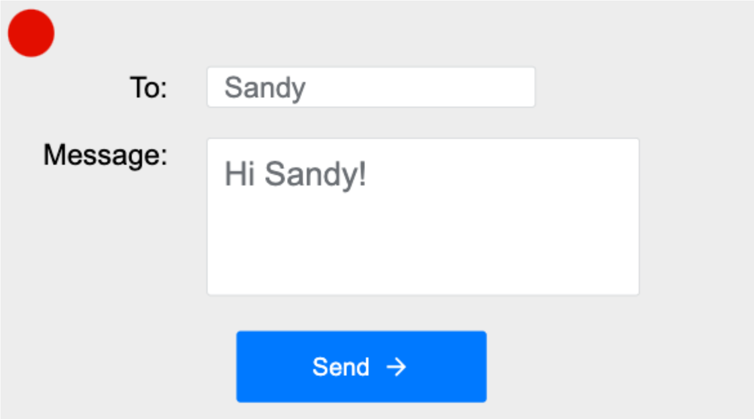

Chats

Sandy, Patrick

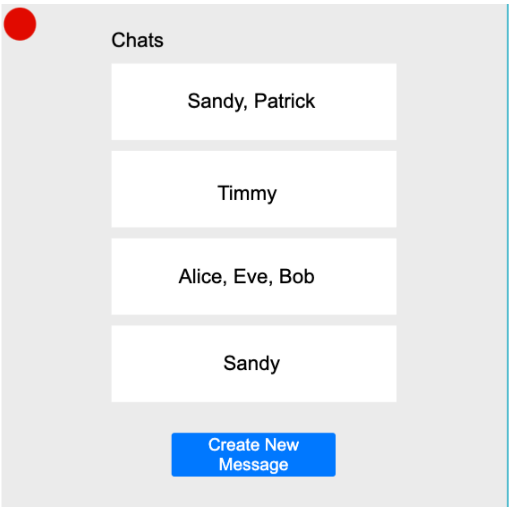
Timmy

Alice, Eve, Bob

Creating new chat:


To: Sandy

Message: Hi Sandy!


Chats

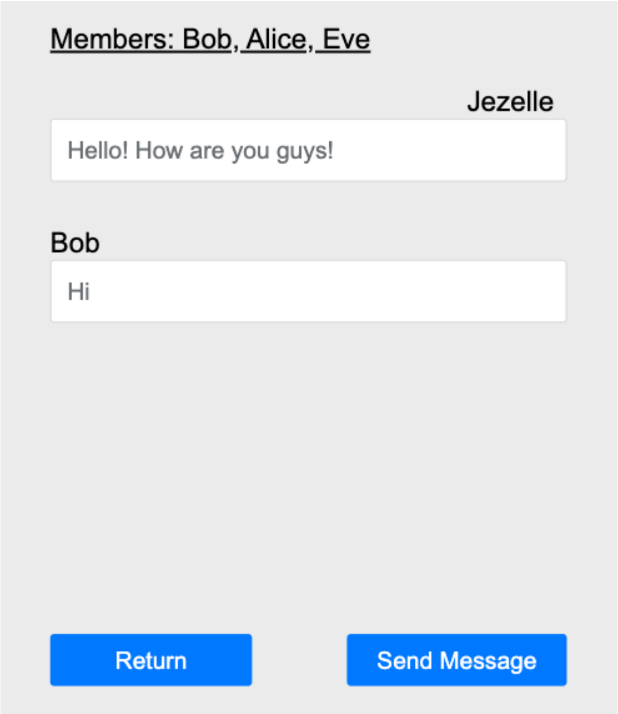
Sandy, Patrick

Timmy

Alice, Eve, Bob

Sandy

Sending message in existing chat:



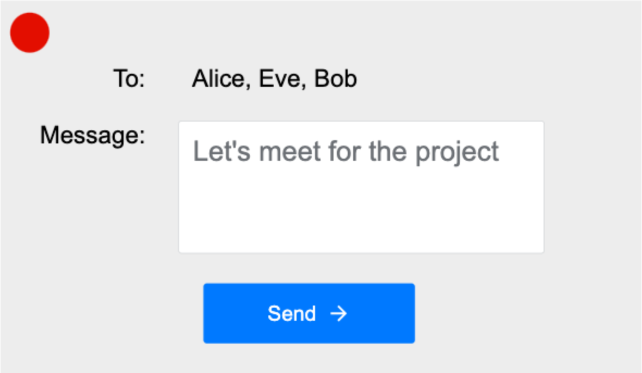
Members: Bob, Alice, Eve

Jezelle

Hello! How are you guys!

Bob

Hi



To: Alice, Eve, Bob

Message: Let's meet for the project

3. Use Cases & Sequence Diagrams

3.1. Use Case Specification

Use Case ID: 0001
Use Case Name: System login
Primary Actor: System
Pre-conditions: User (Employee or IT User) has valid credentials
Post-conditions: User accesses the appropriate interface (Employee or IT User) based on the entered credentials
Basic Flow or Main Scenario:

1. User enters login credentials
2. Login credentials are sent to the server
3. Server validates credentials and sends user to employee or IT user interface
4. User is able to access messaging system

Extensions or Alternate Flows: If login credentials are never inputted, user will stay on login page
Exceptions: Login credentials are invalid
Related Use Cases:

- System logout

Use Case ID: 0002
Use Case Name: System Logout
Primary Actor: System
Pre-conditions: User must press the logout button
Post-conditions: User will be redirected to the login page
Basic Flow or Main Scenario:

1. User selects the logout button
2. The server receives this signal and sends user back to login page
3. User is redirected to login page

Extensions or Alternate Flows: none
Exceptions: none
Related Use Cases:

- System Login

Use Case ID: 0003
Use Case Name: Employee sends a new private message
Primary Actor: Employee
Pre-conditions: Employee has entered valid login credentials
Post-conditions: Employee sends a private message to a different user
Basic Flow or Main Scenario:

1. Employee successfully logs in
2. Employee enters the want to send a message to
3. Employee writes a message and sends it
4. Other user receives message

Extensions or Alternate Flows:
Exceptions: Other user never receives the message
Related Use Cases:

- Employee creates group chat
- Employee views their conversations
- Employee receives/reads message

Use Case ID: 0004
Use Case Name: Employee creates new group chat
Primary Actor: Employee
Pre-conditions: Employee successfully logs in and enters people to chat with
Post-conditions: Employee creates a group chat with other users
Basic Flow or Main Scenario:

1. Employee successfully logs in
2. Employee enters multiple people to chat with
3. Employee enters message to send to group
4. Employee sends message to group chat

Extensions or Alternate Flows: User sends message to only one person
Exceptions: Group chat is not created successfully across all users
Related Use Cases:

- Employee sends a private message
- Employee views their conversations
- Employee receives/reads message

Use Case ID: 0005
Use Case Name: Employee sends message to existing group chat
Primary Actor: Employee
Pre-conditions: Employee successfully logs in
Post-conditions: Employee sends a message to the group chat
Basic Flow or Main Scenario:

1. Employee successfully logs in
2. Employee chooses the group chat they wish to send a message in
3. Employee writes a message and sends it

Extensions or Alternate Flows: Employee only views chat and does not send a message
Exceptions: Employee's message fails to end
Related Use Cases:

- Employee views their conversations

Use Case ID: 0006
Use Case Name: Employee receives/reads message
Primary Actor: Employee
Pre-conditions: Employee has successfully logged in
Post-conditions: Employee is able to view messages that were sent to them
Basic Flow or Main Scenario:

1. Employee successfully logs in
2. Employee selects a chat to view

Extensions or Alternate Flows: none
Exceptions: Employee fails to login
Related Use Cases:

1. Employee views their conversations
2. Employee sends a private message

Use Case ID: 0007
Use Case Name: Employee views their conversations
Primary Actor: Employee
Pre-conditions: Employee has logged in successfully, all conversations have been saved
Post-conditions: Employee is able to view their conversations
Basic Flow or Main Scenario:

1. Employee successfully logs in
2. Employee is able to choose a conversation to view

Extensions or Alternate Flows: Employee views and replies to a message
Exceptions: Employee is not able to log in, messages and conversation do not load
Related Use Cases:

- Employee sends a private message
- Employee receives/reads message

Use Case ID: 0008
Use Case Name: IT sends a new message
Primary Actor: IT
Pre-conditions: IT User successfully logged in
Post-conditions: IT User chooses someone to send a message to and writes and sends a message
Basic Flow or Main Scenario:

1. IT successfully logs in
2. IT selects create new message button
3. IT enters someone to message
4. IT writes and sends the message

Extensions or Alternate Flows: IT creates a group chat
Exceptions: IT does not successfully log in
Related Use Cases:

- IT creates a group chat

Use Case ID: 0009
Use Case Name: IT creates a new group chat
Primary Actor: IT
Pre-conditions: IT user successfully logged in
Post-conditions: IT user has successfully created a group chat
Basic Flow or Main Scenario:

1. IT user successfully logs in
2. IT selects create a new message button
3. IT user enters multiple users to create a group chat with
4. IT user enters a message to send to the group

Extensions or Alternate Flows: IT user only choose one person
Exceptions: IT user does not successfully log in
Related Use Cases:

- IT user sends a message

Use Case ID: 0010
Use Case Name: IT views their own conversations
Primary Actor: IT
Pre-conditions: IT user successfully logs in, All messages have been saved
Post-conditions: IT user is able to access their own conversations
Basic Flow or Main Scenario:

1. IT user successfully logs in
2. IT user selects their own chats to view

Extensions or Alternate Flows: IT user views all conversations
Exceptions: IT user does not log in successfully
Related Use Cases:

- IT views all messages

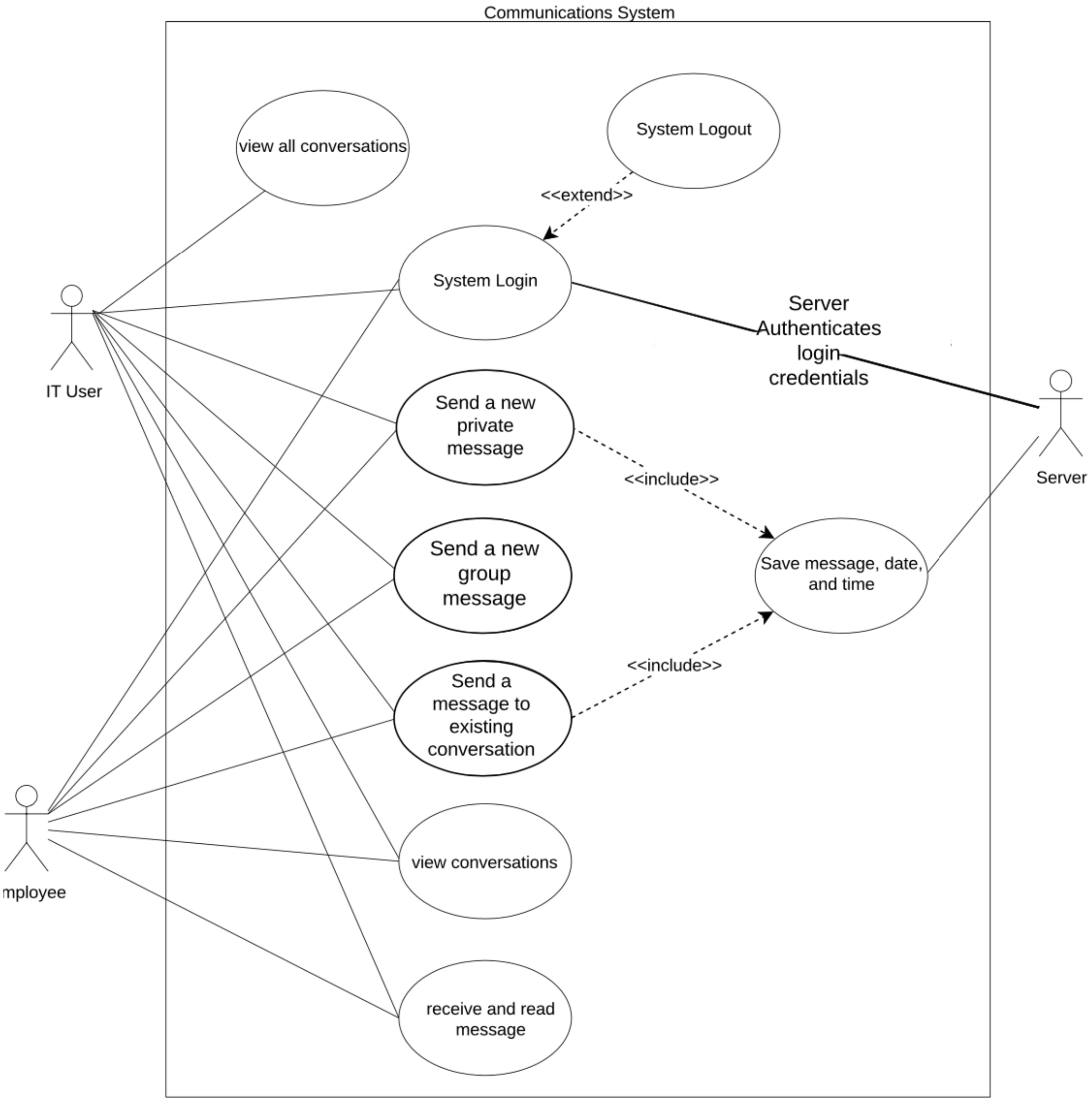
Use Case ID: 0011
Use Case Name: IT views all messages
Relevant Requirements: Software Requirements Specification - 3.1.2.4, 3.1.3.1, 3.1.4.1
Primary Actor: IT
Pre-conditions: IT user successfully logs in
Post-conditions: IT user is able to view all conversations
Basic Flow or Main Scenario:

1. IT user successfully logs in
2. IT user chooses to view all conversations
3. IT user is now able to access all conversation records

Extensions or Alternate Flows: IT views their own messages
Exceptions: IT user does not log in successfully, messages are not able to load
Related Use Cases:

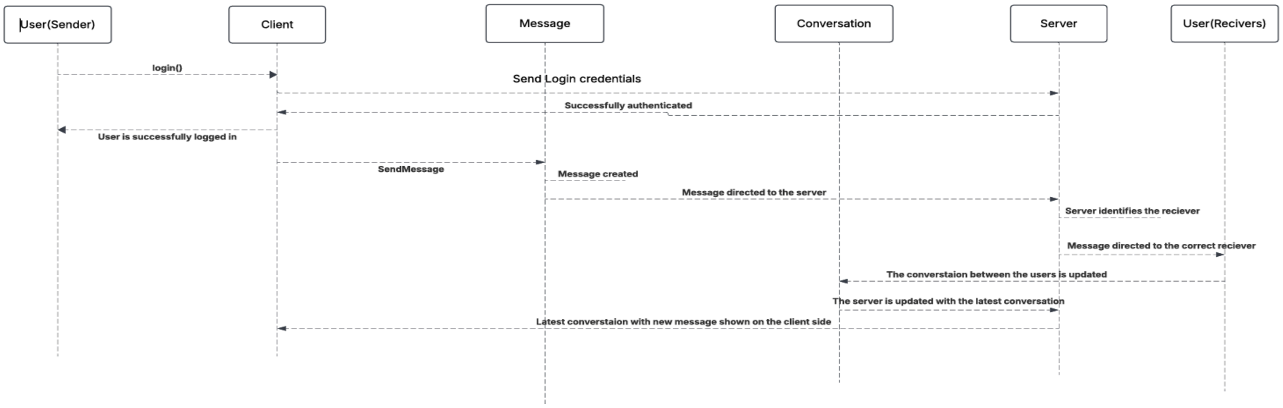
4. IT user views their own conversations

3.2 Use Case Diagram

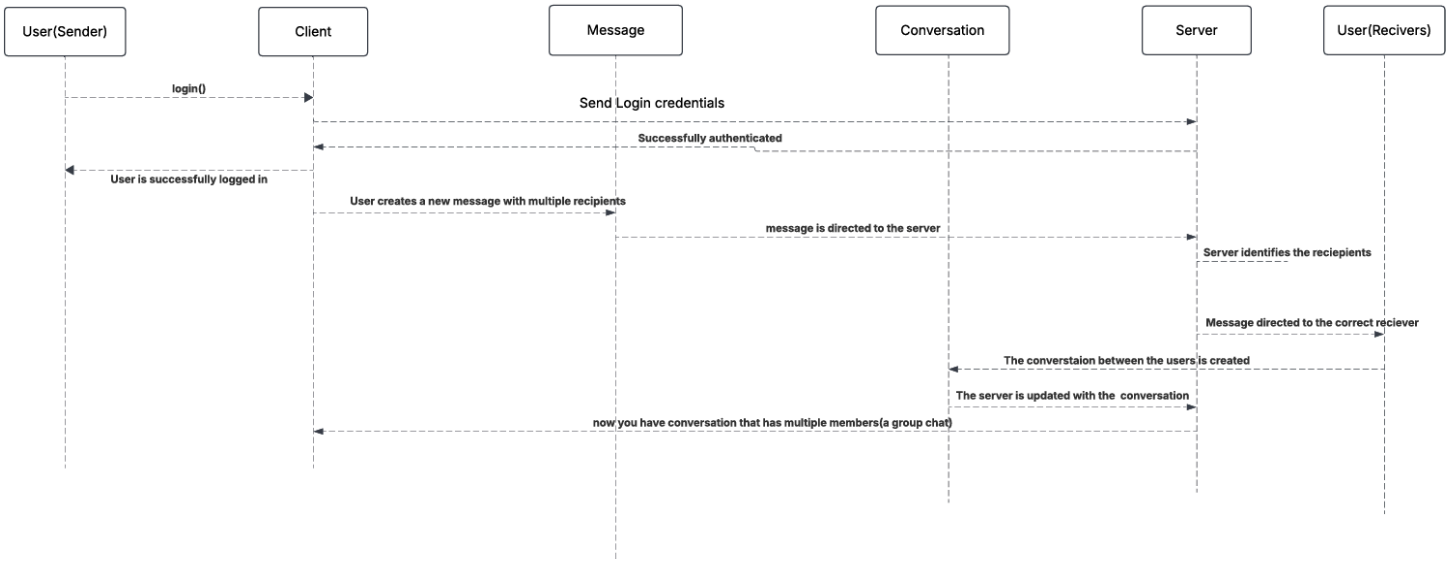


3.3 Sequence Diagrams

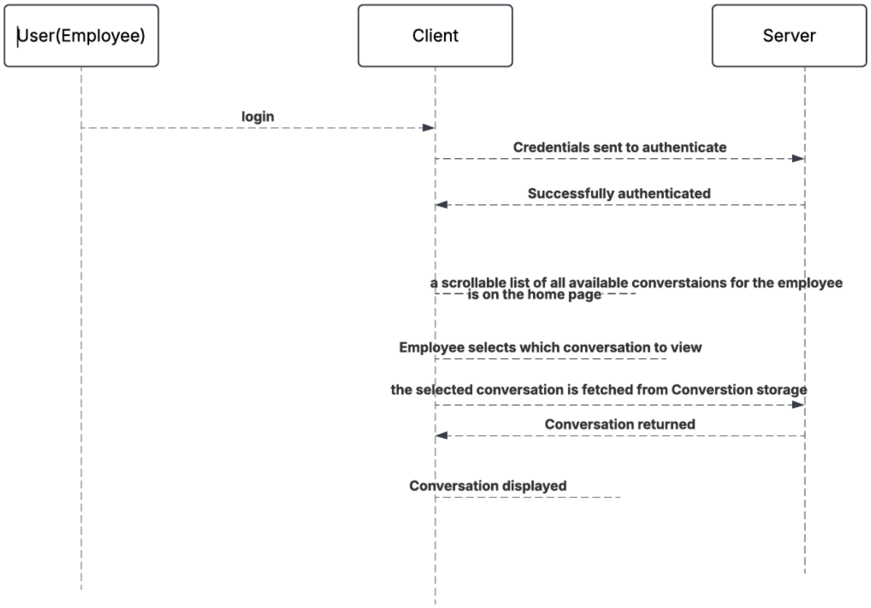
3.3.1. Sending a message to another user


User(Sender)
Client
Message
Conversation
Server
User(Recivers)
login()
Send Login credentials
Successfully authenticated
User is successfully logged in
SendMessage
Message created
Message directed to the server
Server identifies the reciever
Message directed to the correct reciever
The converstaion between the users is updated
The server is updated with the latest conversation
Latest converstaion with new message shown on the client side

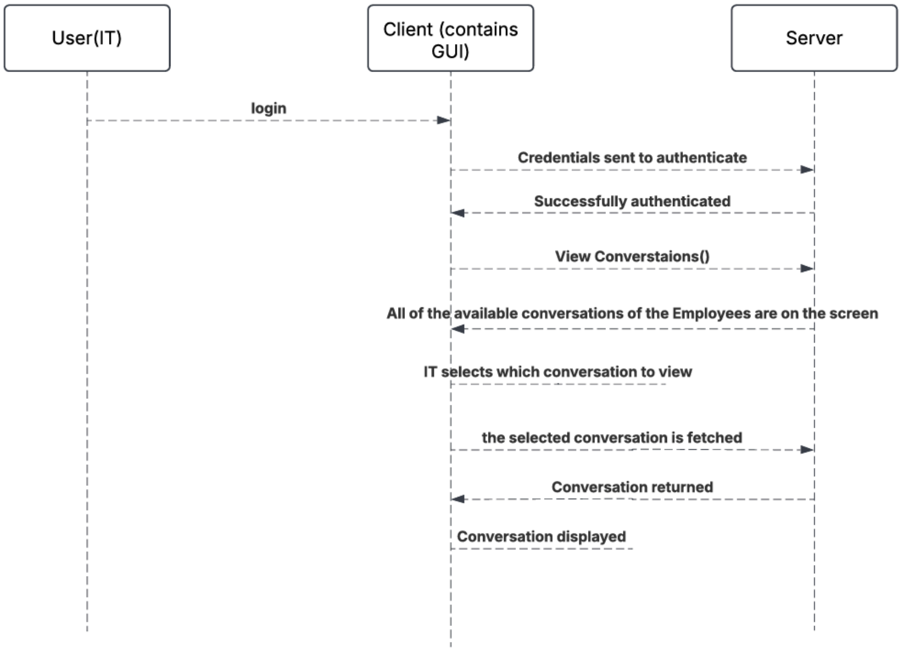
3.3.2. User creating a group


User(Sender)
Client
Message
Conversation
Server
User(Recivers)
login()
Send Login credentials
Successfully authenticated
User is successfully logged in
User creates a new message with multiple recipients
message is directed to the server
Server identifies the reciepients
Message directed to the correct reciever
The converstaion between the users is created
The server is updated with the conversation
now you have conversation that has multiple members(a group chat)

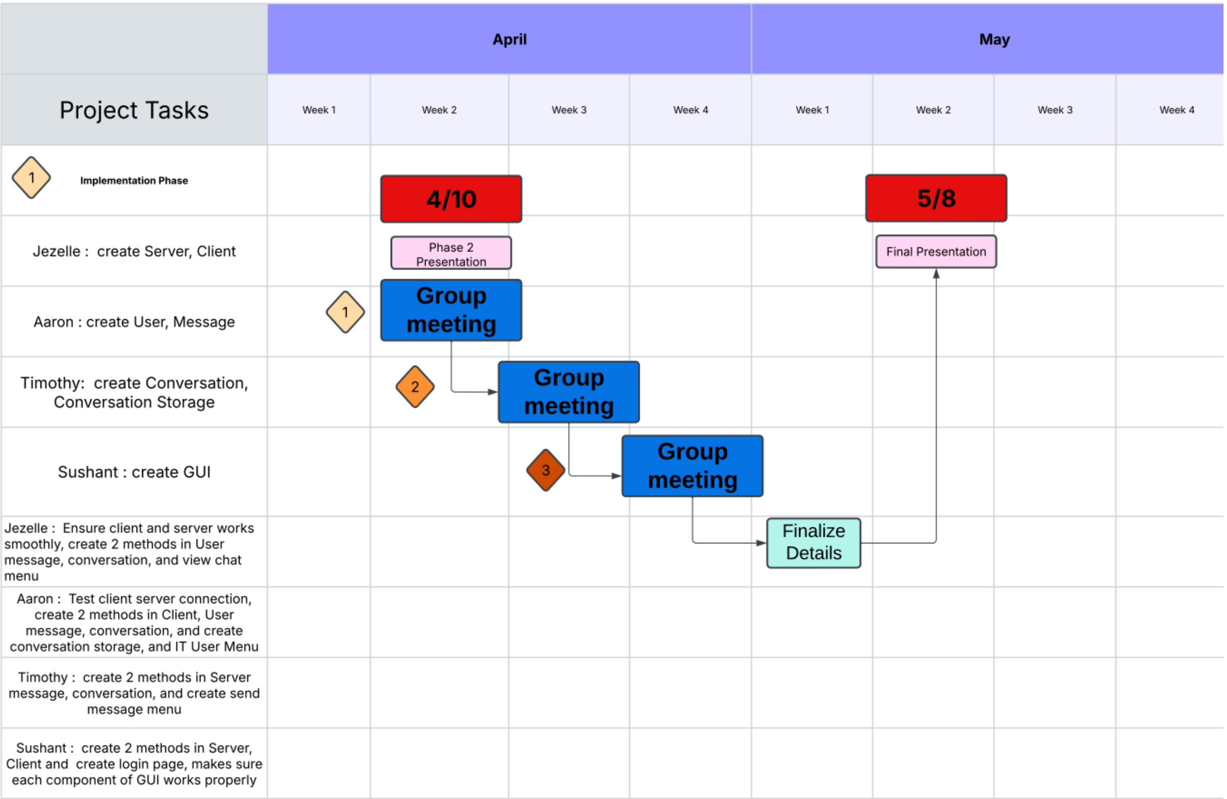
3.3.3 User (Employee) viewing their conversations


User(Employee)
Client
Server
login
Credentials sent to authenticate
Successfully authenticated
a scrollable list of all available converstaions for the employee is on the home page
Employee selects which conversation to view
the selected conversation is fetched from Converstion storage
Conversation returned
Conversation displayed

3.3.4 User (IT) viewing Employee conversations


User(IT)
Client (contains GUI)
Server
login
Credentials sent to authenticate
Successfully authenticated
View Converstaions()
All of the available conversations of the Employees are on the screen
IT selects which conversation to view
the selected conversation is fetched
Conversation returned
Conversation displayed

4. Gantt Chart



	April				May			
Project Tasks	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2	Week 3	Week 4
1 Implementation Phase		4/10				5/8		
Jezelle : create Server, Client		Phase 2 Presentation				Final Presentation		
Aaron : create User, Message	1	Group meeting						
Timothy: create Conversation, Conversation Storage		2	Group meeting					
Sushant : create GUI			3	Group meeting				
Jezelle : Ensure client and server works smoothly, create 2 methods in User message, conversation, and view chat menu					Finalize Details			
Aaron : Test client server connection, create 2 methods in Client, User message, conversation, and create conversation storage, and IT User Menu								
Timothy : create 2 methods in Server message, conversation, and create send message menu								
Sushant : create 2 methods in Server, Client and create login page, makes sure each component of GUI works properly								