Communications Project

Group 7

Software Requirements Specification

Revision History

Date	Revision	Description	Author
02/13/2025	1.0	Initial Version	Group 7
02/13/2025	1.0.1	Drafted: Purpose, Constraints	Kamari
02/25/2025	1.0.2	Scope, Overview, Constraints, External Interface Requirements	Anna, Kamari, Josiah, Adam
02/26/2025	1.0.3	Organized and reformatted functional requirements	Jacob
03/02/2025	1.0.4	Reformatted constraints, assumptions and dependencies. Wrote 4.3.3-4.3.4	Jacob
03/03/2025	1.0.5	Created Basic Class Concepts	Adam
03/04/2025	1.0.6	Updated Modules and Assumptions, Refined requirements, Added definitions and acronyms, added description to 2.1	Group 7
03/04/2025	1.0.7	Added Several Use Cases into Use Case Specification	Josiah
03/05/2025	1.0.8	Made a Basic Class Diagram	Adam
03/05/2025	1.0.9	Made Sequence Diagram	Kamari
03/06/2025	1.1	Updated Table of contents, Refined Marketing Talk, Refined Functional Requirements, Refined internal interface requirements	Josiah
03/06/2025	1.1.1	Added the Location of Reference	Adam
03/06/2025	1.1.2	Added Use Case Diagram	Anna
03/06/2025	1.1.3	Added Gantt Chart	Jacob
04/08/2025	1.1.4	Removed Displaying User as Online or Offline Requirement	Josiah

Table of Contents

1. Purpose

1.1. Scope

The purpose of this project is to develop a text-based messaging app that can be used by large organizations. The project will support both asynchronous and synchronous communication which will allow users to chat privately or in groups. All messages will be immutable.

1.2. Definitions, Acronyms, Abbreviations

GUI- Graphic User Interface

IDE- Integrated Development Environment

Chat - Message room for more than two users.

1.3. References

Use Case Specification Document: (Section 5, pg. 9)

Use Case Diagram: (Section 6, pg. 12)

Class Diagram: (Section 7, pg. 13)

Sequence Diagram: (Section 8, pg. 14)

Gantt Chart: (Section 9, pg. 15)

1.4. Overview

This project aims to develop a text-based messaging app for a large organization, supporting both synchronous and asynchronous communication. Users will be able to chat privately or in groups, with all conversations logged and accessible to designated admins.

2. Overall Description

2.1. Product Perspective

This product enhances team cohesion and accountability with fast message boards, unchanging messages, and full view of everything. This product ensures your team is on track and productive throughout projects and meetings.

2.2. Product Architecture

The system will be organized into 3 major modules: the User module, the Client module, and the Server 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):

Github: https://github.com/yoshiyahoo/Communication-App

2.4. Constraints

2.4.1. General Constraints

2.4.1.1. Requires a server application and client application.
2.4.1.2. Data won't be encrypted

2.4.2. Programming Constraints

2.4.2.1. Program must be written in Java.
2.4.2.2. GUI must be implemented using Java Swing
2.4.2.3. Networking must be implemented using Java Sockets
2.4.2.4. Programming must be done in Eclipse IDE
2.4.2.5. Github must be used for version control of code base, requirements, and design. It must be used for assigning tasks

2.5. Assumptions and Dependencies

2.5.1. Assumption: It is assumed that the maximum number of users at a given time is 250,000.
2.5.2. Dependency: Use the Java standard libraries.
2.5.3. Dependency: Have one GitHub repository for all developers.
2.5.4. Assumption: Server is always online.

3. Specific Requirements

3.1. Functional Requirements

3.1.1. Common Requirements

3.1.1.1. Messages are immutable. Nobody can edit a message after it's sent.
3.1.1.2. Messages consist of: User who sent it, content of message in plaintext, time message was sent, if message was received

3.1.2. Client Module Requirements

3.1.2.1. Client must be able to create chats
3.1.2.2. Clients must send messages in the format specified by 3.1.1.2.
3.1.2.3. Clients must log in with a username and password
3.1.2.4. Client username and passwords have a maximum of 64 characters
3.1.2.5. If a message fails to send 3 times, there should be a pop-up window showing that the message didn't send.
3.1.2.6. Clients may logout at any time

3.1.3. Server Module Requirements

3.1.3.1. Server should be able to handle numerous clients.
3.1.3.2. Server should deny or accept a user based on their login credentials.
3.1.3.3. Server should handle saving/loading data asynchronously, to allow users to send and receive messages concurrently.
3.1.3.4. Server should only send messages to Client when Client is online.
3.1.3.5. If the client fails to send a message, the server should let the client know.
3.1.3.6. Server should store all chats, messages inside chats, and user authentication information.

3.1.4. User Module Requirements

3.1.4.1. There are two types of users: Administrator and Employee.
3.1.4.2. Employees can see and send messages in the chats they are a part of.
3.1.4.3. Administrators can see and send messages in any chat at any time.
3.1.4.4. Both administrators and employees can create chats.

3.2. External Interface Requirements

3.2.1. The system must provide an interface to the log-in window so that users can log in.

3.2.1.1. The interface should contain fields to enter username and password and an enter button.

3.2.2. The system must provide an interface for the chat window.

3.2.2.1. It should include a text message box along with a send button so that users can send messages.

3.2.2.2. It should have a logout button.

3.2.2.3. It should have minimize, full screen, and exit buttons for the user to be able to control the size of the window and close it.

3.2.2.4. The user should be able to view and scroll through a chat window of users.

3.3. Internal Interface Requirements

3.3.1. All messages passed through the client and server must be a comma separated file with each message entry following the order specified in 3.1.1.2.

3.3.2. All login & logout requests will be sent via plaintext with comma separated username & password.

3.3.3. All open message requests will be sent via plaintext with comma separated username, password, and chat to open.

3.3.4. All chats passed through the client and server must be a comma separated file with the first entry being the chat name, the chat members, and date chat was created. The subsequent entries must be all chat messages.

4. Non-Functional Requirements

4.1. Security and Privacy Requirements

4.1.1. In order to access the program, valid login credentials are required.
4.1.2. Clients connecting can only be employees or administrators of the company.
4.1.3. Employees will have only one account with premade credentials.
4.1.4. Messages & Requests passed between Client/Server will not be encrypted.

4.2. Environmental Requirements

4.2.1. Internet connection is required to login, view, access chats, and logout.

4.3. Performance Requirements

4.3.1. Messages should take no more than 5-10 seconds to send; 60 seconds max.
4.3.2. The user interface should take no more than 5-10 seconds to load.
4.3.3. User auth should take 1-3 seconds on average; 60 seconds max
4.3.4. Server should take 1-3 seconds to reply to any request; 60 seconds max

4.4. Developer Requirements

4.4.1. The system cannot use any other technology not specified in section 3.1.3.

5. Use Cases Specification

Use Case 1: User Logs In

Relevant Requirements: 3.1.2.3. - 3.1.2.5., 3.1.3.1. - 3.1.3.2., 3.2.1., 3.3.2., 4.1., 4.2.1., 4.3.3. - 4.3.4.

Primary Actor: User

PreCondition: User is not logged in
PostCondition: User is logged in and can start seeing chats

Basic Flow of Main Scenario:
1. User enters username
2. User enters password
3. Client sends login request to server
4. Server sends login confirmation to client
5. Client window updated to message board

Exceptions:
- Login Credentials Invalid
- Request timed out

Related Use Cases
- 2, 3, 4, 5

Use Case 2: User Sends a Message

Relevant Requirements: 3.1.1.2., 3.1.2.2, 3.1.3.1., 3.1.3.3. - 3.1.3.6., 3.1.4.2. - 3.1.4.3., 3.2.2., 3.3.1., 3.3.4., 4.1.4., 4.2.1., 4.3.1. - 4.3.2.

Primary Actor: User

PreCondition: User hasn’t sent their message yet
PostCondition: User sent their message for all to see

Basic Flow of Main Scenario:
1. User types in message
2. User clicks send button
3. Client sends message send request to server
4. Server receives the message send request
5. Client sends message to server
6. Server receives the message
7. Server updates the chat log with new message
8. Server sends new chat log back to client
9. Client receives chat log
10. Client displays chat log for user

Alternate Flow: Message fails to send
1. User types in message
2. User clicks send button
3. Client sends message send request to server
4. Server receives the message send request
5. Client sends message to server

6. Server fails to receive the message
7. Server informs client that message failed to send
8. Client displays box saying message failed to send
9. User tries to resend message or cancels

Exceptions:

- Server times out

Related Use Cases

- 1, 3, 4, 5

Use Case 3: User Opens a Chat With Other Users

Relevant Requirements: 3.1.2.1., 3.1.3.1., 3.1.3.3., 3.1.3.6., 3.1.4.2 - 3.1.4.3, 3.2.2., 3.3.4.,

Primary Actor: User

PreCondition: User is about to open the chat and is in a different chat
PostCondition: Chat is opened and user is ready to type

Basic Flow of Main Scenario:

1. User Double Clicks chat icon
2. Client sends open chat to the server
3. Server finds log file of the messages between those users
4. Server returns log file of messages back to the client
5. Client overrides previous chat log with new one
6. Client updates UI with new window

Alternate Flow: Administrator Accesses Chat they're not a part of

1. User Double Clicks chat icon that they're not a member in
2. Client sends open chat to the server
3. Server finds log file of the messages between those users
4. Server returns log file of messages back to the client
5. Client overrides previous chat log with new one
6. Client updates UI with new window

Exceptions:

- Server fails to open chat

Related Use Cases

- 1, 2, 4, 5

Use Case 4: User creates chat with other users

Relevant Requirements: 3.1.1., 3.1.2.5., 3.1.2.2., 3.1.2.1., 3.1.3.5., 3.1.3.4., 3.1.3.3., 3.1.3.1., 3.2.2.1., 3.3.4., 3.3.3., 3.3.1.

Primary Actor: User

PreCondition: User hasn't created group chat
PostCondition: User has created group chat and users can start chatting

Basic Flow of Main Scenario:

1. User Clicks create new group chat

2. User adds new users into group chat
3. Client sends open chat request with these users
4. Server finds no log file with these users
5. Server creates new log file
6. Server sends back new log file to client
7. Client overrides previous log with new one
8. Client updates UI

Alternate Flow:

1. User types in message
2. User clicks send button
3. Client sends create chat request to server
4. Server receives the message send request
5. Server fails to create the chat
6. Server informs client that chat failed to create
7. Client displays box saying hat failed to be created
8. User tries to recreate chat or fails

Exceptions:

- Server times out, The other users aren't online

Related Use Cases

- 1, 2, 3, 5

Use Case 5: User Logs Out

Relevant Requirements: 3.1.2.6., 3.1.3.1., 3.2.2.2., 4.1.2., 4.1.4, 4.2.1, 4.3.4.,

Primary Actor: User

PreCondition: User is logged in
PostCondition: User is logged out

Basic Flow of Main Scenario:

1. User presses logout button
2. Client sends logout request to the server
3. Server revieces request
4. Server logs client out
5. Server sends success message back to client
6. Client is sent back to login screen

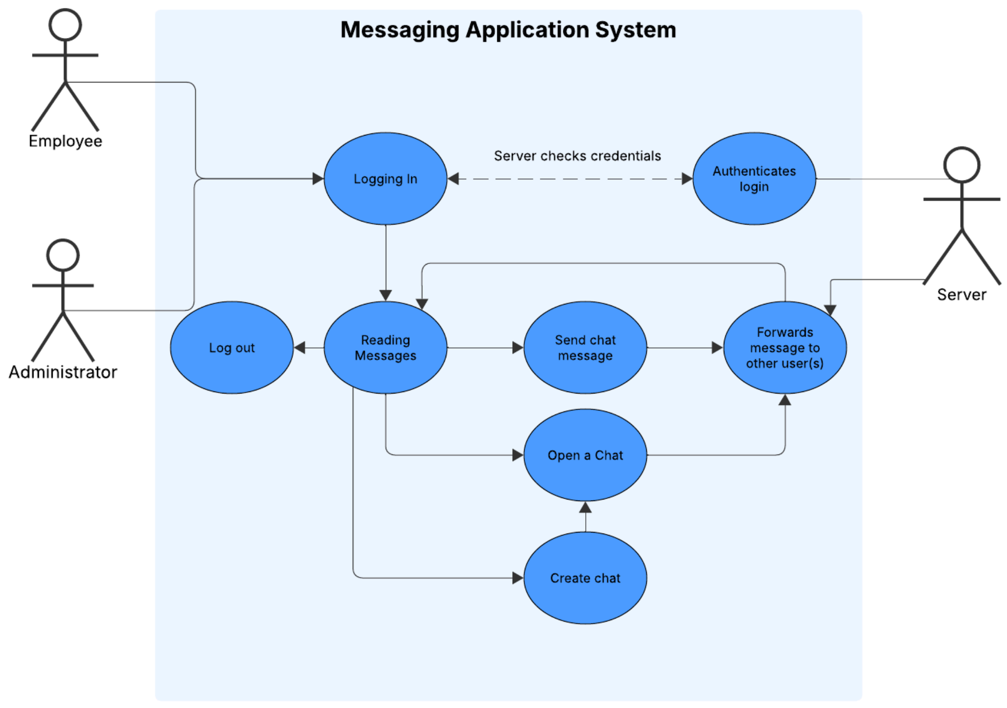
Exceptions:

- Request fails to send
- Server times out

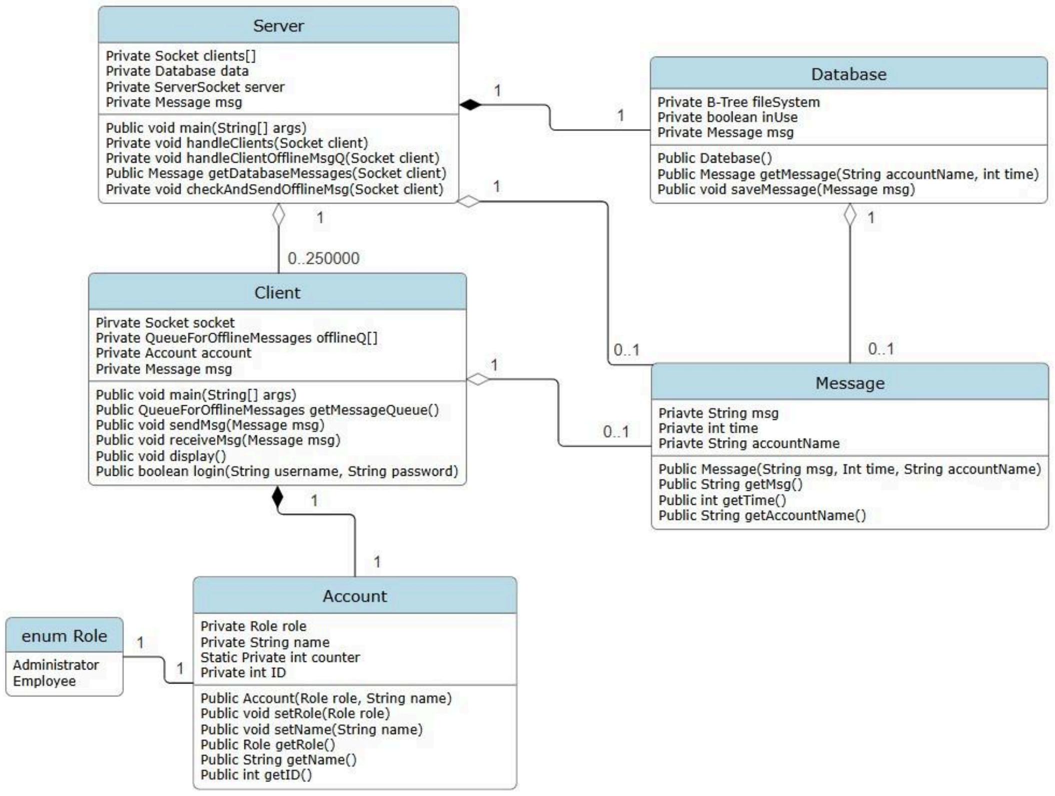
Related Use Cases

- 1, 2, 3, 4

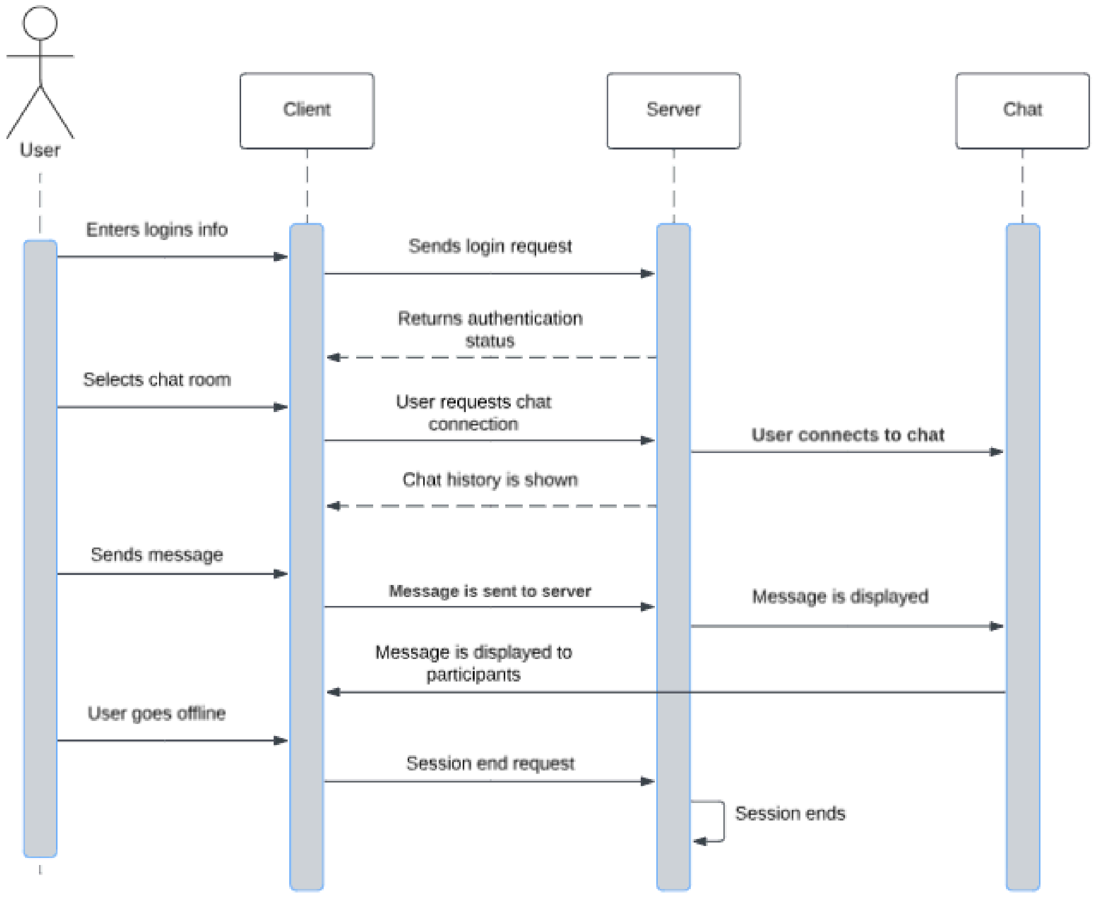
6. Use Case Diagram



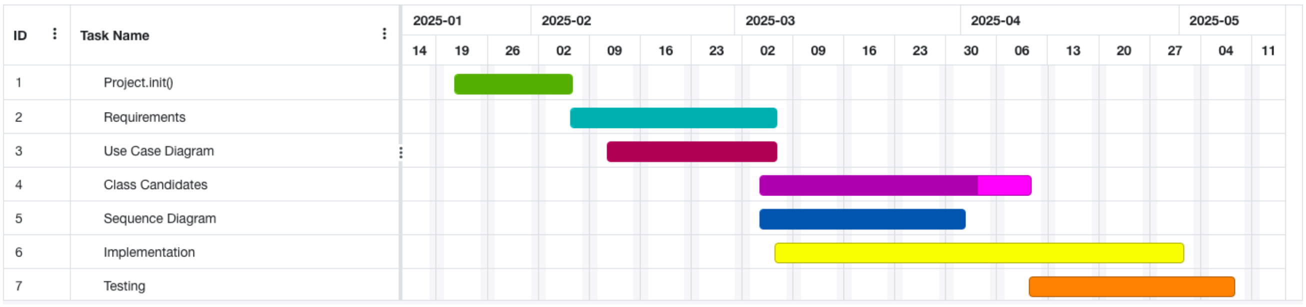
7. Class Diagram


Server
Private Socket clients[]
Private Database data
Private ServerSocket server
Private Message msg
Public void main(String[] args)
Private void handleClients(Socket client)
Private void handleClientOfflineMsgQ(Socket client)
Public Message getDatabaseMessages(Socket client)
Private void checkAndSendOfflineMsg(Socket client)
1
1
1
1
0..250000
Database
Private B-Tree fileSystem
Private boolean inUse
Private Message msg
Public Datebase()
Public Message getMessage(String accountName, int time)
Public void saveMessage(Message msg)
1
Client
Pirvate Socket socket
Private QueueForOfflineMessages offlineQ[]
Private Account account
Private Message msg
Public void main(String[] args)
Public QueueForOfflineMessages getMessageQueue()
Public void sendMsg(Message msg)
Public void receiveMsg(Message msg)
Public void display()
Public boolean login(String username, String password)
0..1
0..1
1
0..1
Message
Priavte String msg
Priavte int time
Priavte String accountName
Public Message(String msg, Int time, String accountName)
Public String getMsg()
Public int getTime()
Public String getAccountName()
1
1
Account
Private Role role
Private String name
Static Private int counter
Private int ID
Public Account(Role role, String name)
Public void setRole(Role role)
Public void setName(String name)
Public Role getRole()
Public String getName()
Public int getID()
enum Role
Administrator
Employee
1
1

8. Sequence Diagram



9. Gantt Chart



ID	Task Name
1	Project.init()
2	Requirements
3	Use Case Diagram
4	Class Candidates
5	Sequence Diagram
6	Implementation
7	Testing