Report

Semester Project Part-1

[Client-Server Chat Application]

[Computer Networking]

[Session Fall 2024]

[66920, 68105, 67949]

[Saniya Gul, Pranjal Kumari, Maryam Ejaz]

[Sem: 2nd]

Submitted to:

[Ms. poma panezai]

[Department of FICT]

Faculty of Information and Communication Technology (FICT), BUITEMS, Quetta

Abstract

This project primarily concerns the development of a chat application with communication environment. Where multiple clients can communicate with each other from a server. The purpose of our project is to design and implement a chatting system in order to allow a server and clients to communicate with one another. The system's design is based on the socket concept, which is a software endpoint that enable bidirectional communication between a server program and one or more client programs. The construction of this system will be done using the Java programming language and Java Development Kit (JDK), which is a development environment developed for building applications and components using the Java programming language. NetBeans (IDE) will also be used.

Table of Contents

1. Problem Statement

In modern computer networks, efficient communication between multiple devices is essential for delivering services and resources. The client-server architecture is a foundational model that separates service providers (servers) from service requesters (clients), enabling scalable, organized, and reliable network communication. This project focuses on designing, implementing, and demonstrating a client-server system where the client sends requests (such as data retrieval, file access, or service execution) to the server, and the server processes these requests and give correct response on time. The challenge lies in ensuring correct protocol handling, managing multiple client connections, maintaining data integrity, and achieving smooth interaction between the client and server components. The goal is to build a working implementation that showcases the fundamental principles of client-server communication, highlights potential issues (like connection handling, timeouts, or errors), and demonstrates practical applications using Java.

2. Introduction

The main objective of this project is to develop a Client-Server Chat Application through Java programming language using the NetBeans IDE. The chat application allows many users to chat in a real-time setting over a network by allowing users to mutually and effectively exchange text messages. This project is executed in working with Socket Programming in Java, which in turn illustrates basis application of it. Socket programming is the foundation for many applications or programs of any network-based programming in Java.

A socket is made from an IP address and a port number, which helps create a connection link between two or more systems. So, the existence of both IP and port can create a connection. That is the case when there is a server, and the client must socket programming (software) which communicates using IP and Port number. The programming of those applications is known as Socket Programming; the type of programming is what really matters. Socket programming supports bi-directional communication in which a system is a server and other system(s) are a client(s). The Server acts as an actuator that is a central controller over all the communication tasks of the various clients that are attached. This structure can be referred to as a client-server model, which is one of the most commonly used communication

architectures in computer networking. Communication is established over TCP (Transmission Control Protocol) to provide reliable messaging, as TCP preserves the integrity of the data even if an error occurs. The application provides guaranteed, sequential, and error-checked message delivery, meaning it is an appropriate medium for chat-based communication, which places a premium on message integrity.

Our project can study not just the internals of socket communication but also the additional programming fundamentals, like that of multi-threading, allowing the server to support multiple clients at the same time, and compiling a Graphical User Interface (GUI) with usable elements using Java Swing components, for an easily manipulable interface for users, which provides a sophisticated communication between clients and servers. The GUI is intended to allow users to send and receive text messages, as well as manage the connection.

Moreover, this chat application is an experimental demonstration of how client-server design can be used to create interactive and extensible communications systems. If so, it serves as both a demonstration of how socket programming can be used to implement complex and interactive communications systems, and as an interesting learning exercise where we have related theory to systematic computer programing. This project provides an understanding and an opportunity to learn more.

3. Literature Review

Before we started our project on the Server and Client Chat Application, we reviewed existing research and information to help us understand how these systems work and how basically java works. A chat application is a simple form of illustrating how computer networking allows for the ability to communicate in real time between two or more devices. It also gives an understanding of the concepts of client-server architecture, which is used in so many networking applications today.

The majority of the literature we analyzed discussed how this client-server model works. In this model, the server would serve as the main system that will handle the exchange of messages and the connections, while clients will simply be the users , connects to the server to send and receive messages easily. This setup is a common structure seen in online services like messaging applications, emails, and even web browsing.

We also read about the different networking protocols, specifically TCP (Transmission Control Protocol). This is a commonly used protocol in chat applications, and it is done for a reason; this protocol allows devices to communicate properly without any flaws in the sequence in which messages are delivered to the devices. TCP connections are reliable that is the main reason it's used in chat applications to provide reliability.

❖ ANALYSIS

Developing the client-server chat application required a thorough investigation of the functional and non-functional requirements. At a functional level, we needed to establish a system that permits real-time message exchange, handles multiple client connections, and displays messages to users in a readable way. On the non-functional side, we were concerned about performance, scalability, reliability, and usability. We examined how performance might suffer under greater numbers of clients, message load and, share resource concerns. We also looked at some of the security concerns surrounding message privacy and securing our TCP/IP transmission.

Along the same lines, we analyzed user interaction with the system, identifying those features we felt to be critical to creating an intuitive experience, such as connection status indicators, applications that respond to the state of a message and smooth transformations among different chat states.

❖ DEVELOPMENT PHASES

a. Requirement Gathering and Planning

In this phase of the project, we started with a project plan, identifying the goal of the project, together with the technologies needed to make that happen. We decided to use the programming language Java, since it has better networking capabilities than other languages (for example .NET or python). We also decided that we wanted to use NetBeans IDE, since we identified several useful features that would benefit from using it, including project management, debugging, and user interface creation. We identified key functional requirements that were necessary (multi-client, message reliability, user interface usability), and project timeline milestones.

b. **Design Phase**

This phase involved creating the overall architecture of the system. We designed the flow of messages between clients and server, and designed the overall structure of the application with

OOP principles. Overall foreshadowed how many classes would be needed, and what socket classes, TCP communication, and thread management strategies would be needed. We also created wireframes of the GUI to support a consistent user experience for the overall message delivery system.

c. Implementation Phase

The implementation was one of the most complex phases because it required the development of the server and client programs separately, with the end goal of integrating both programs together for testing. Server-side implementation focused on establishing socket connections and managing the client socket that represented the thread, while the client side implementation focused on accepting the socket connection, sending the message, and managing user input. The GUI was created with Java Swing, and the components were developed and tested through a series of iterations to ensure usability and responsiveness.

d. Testing and Debugging

Once a basic communication was achieved, we tested the system under a variety of conditions, such as numerous simultaneous users, sudden disconnections, and incorrect inputs. Debugging was done with the NetBeans tools to trace runtime issues, throw exceptions, and stabilize the system. This stage also included slowly improving the layout and functionality of the GUI.

e. Evaluation and Enhancement

At this point, we had all the core functionality we planned or needed; we began evaluating the functionality and usability of the system. We put together a list that captured all of the feedback from the users to understand where we could improve the application. Items for improvement included times on messages, better error messages, and a nicer user interface. We were also able to put together some ideas for things we would like to add or improve in the future, including security encryption, and logging messages.

4. Methodology (Design and Simulation)

- **DESIGN PHASE**

The design phase of the Client-Server Chat Application represents an important part of the project. This phase requires careful consideration around the design of the application architecture, communication protocols, security, and functionality to ensure it is scalable,

secure and capable of providing uninterrupted user experiences across a variety of varied networking environments. The design phase develops how certain features will work (i.e., messaging, authentication and multi-user). The system itself is designed based on a client-server architecture, where there are many clients that connect through a single central server. The server receives incoming client connections, manages active sessions and client requests, and distributes messages to users. Each client is an interface for the users to send messages and receive messages, and initiate application interaction.

a. System Architecture

This system uses the most common architecture design, the client/server model, where the clients create a connection with a server. Where Several clients connect to a central server, and with that server moderating the messaging with all clients from it.

Server: It will accept incoming client connections, fulfill messaging logic when sending messages, and maintain sessions per client. It will accept control/command over chat rooms, and authentication.

Client: It creates a socket connection with the server, sends messages by the user, and displays messages sent by the server.

b. Network communication

The communication between the client and server is facilitated through socket programming and uses the TCP/IP protocol. TCP was selected due to its reliability, so messages are sent and received in order, which is important for a chat system. Communication between both the client and server uses the Socket and Server socket classes in Java. The server will handle these clients using separate threads allowing them to exchange messages concurrently.

c. The message format

All messages will use the same format for consistency in parsing and delivery. Each message will contain a sender ID/username, content, and the timestamp.

The message format could be plain text, or serialized form using lightweight data formats like JSON for added flexibility and future scalability.

d. User authentication

The system integrates user authentication functionality as a simple mechanism to restrict access and tailor communications to users. Users can create an account by providing a username and

password and log in before entering the chat. The server authenticates users and validates their credentials.

e. Message handling

Within the application, message flow is separated into two areas: server-side messaging and client-side messaging. Server-side messaging will take care of receiving messages, approving the message, and broadcasting the message to the respective clients (if the client joined as a private chat room) or public chat room. Client-side messaging will accept messages from the server and display the messages in a presentable format, and include the ability for users to enter and send messages through the GUI.

f. Error handling

The error management in this application is robust on both the server and the client, and maintains its stability through the use of error handling.

Connection errors: to see if the connection is lost, it will automatically be discovered and update the user to whether the chat has lost its connectivity.

Message errors: will determine if data is malformed and run a verification in order to avoid crashing the system.

Authentication errors: this informs the user when their credentials were incorrect.

g. User Interface Design

A clean and responsive Graphical User Interface (GUI) was built using Java Swing to provide users with an intuitive experience.

The features included:

message history, user list, input area, button to send or connect/disconnect to chat room, navigation through chat rooms. The layout is designed with a real-time concept, but without compromising clarity and ease of use.

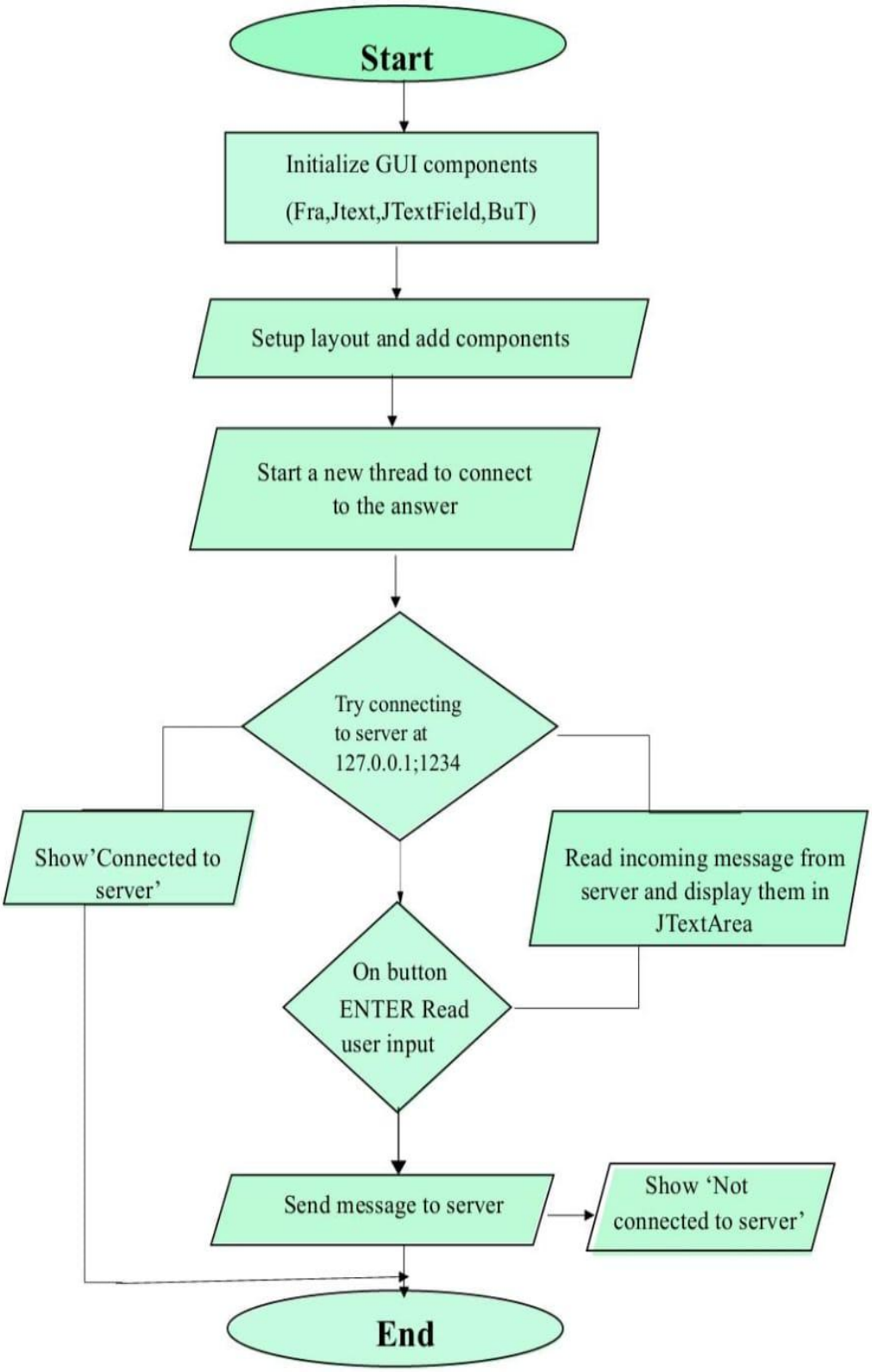
Design Approaches The design phase followed several design principles.

Performance: The system addressed low latency and high throughput needs.

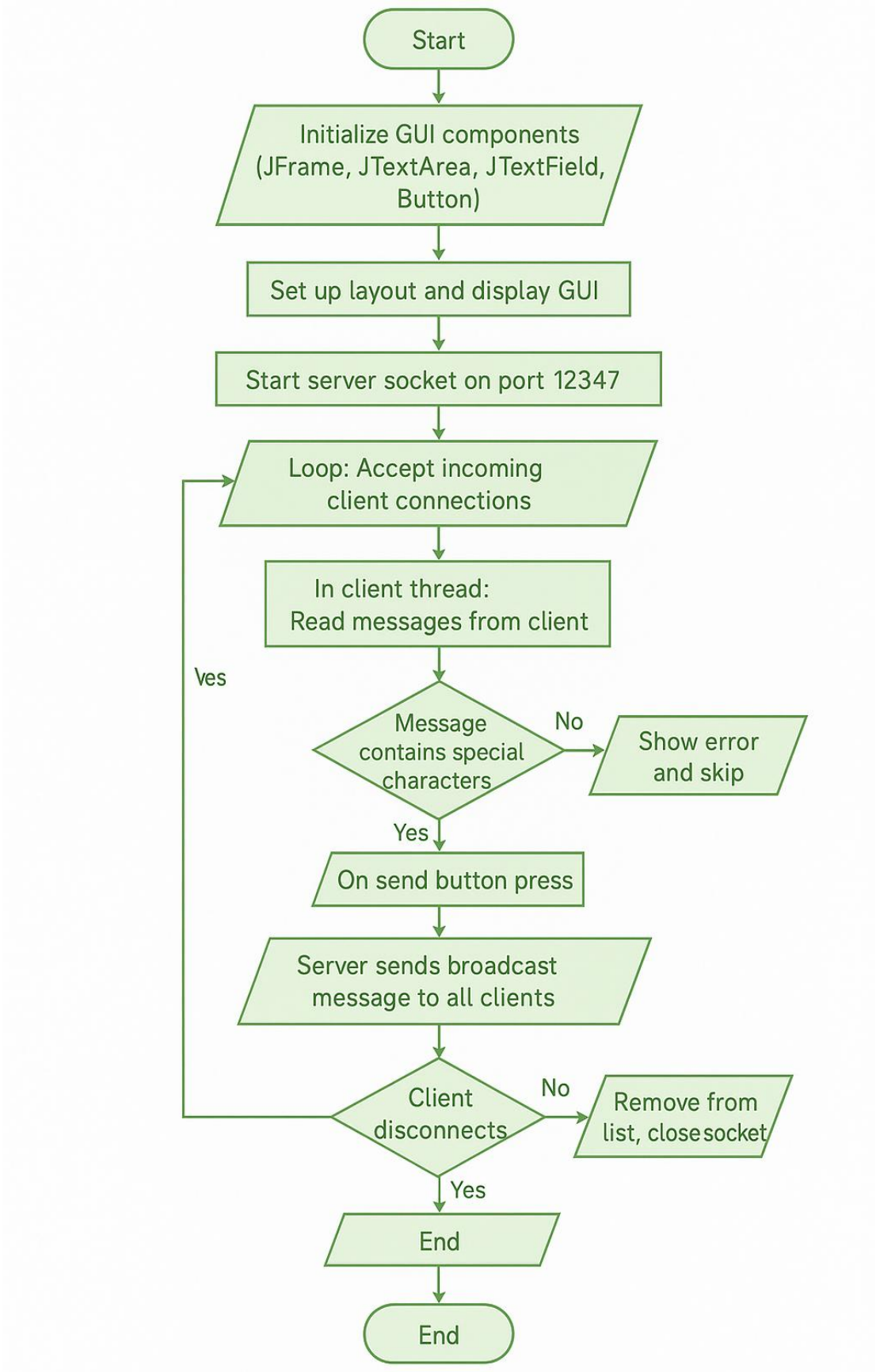
Reliability: The system delivers reliable messages using TCP and our error handling mappings.

User Experiences: The GUI is clean, accessible, responsive and assuring positive user engagement. By recognizing these aspects during the design phase, the chat application is described to be resilient, durable, and user-friendly.

❖ CLIENT-SIDE FLOWCHART



❖ SERVER-SIDE FLOWCHART



❖ SIMULATION AND TESTING

a. Basic Functionality

We tested the first case where one client connects to the server and sends messages to it and receives replies. It worked great, the server was able to connect to the client and process the messages.

b. Multiple Clients

We tested the server with multiple clients. All users were able to send and receive messages in real time. Messages were correctly published to all clients connected at that time. We didn't observe crashing or get error messages either on the server or clients after they login, send and receive messages, and disconnect with one or two clients connected to the server during the tests. The server continued working for clients at any time even as more clients arrive or disconnect.

c. Message Flow

Messages were sent quickly and in order. For example, when one client sends a message, it instantly updates other clients without delay.

d. Disconnection Response

We also tested for disconnected clients and server shut down situations. The server continued working for the remaining clients after disconnection and none crashed. The only exception to this point was that the disconnected client returning must login again because the code does not allow automatic reconnections.

e. Performance

With a maximum of 5-6 clients simultaneously, the application works great. There are no delay times or excessive resource consumption.

f. Limitations

We have identified some limitations:

- No private chat support.
- No user interface (only text).
- No encryption/secure messaging.

5. Results and Discussion

Once we finished constructing our Server and Client Chat Application, we thoroughly tested the system and got the results we anticipated. Our application was able to have multiple clients connect to a single server to send and read messages in real-time, as desired. When one client sends a message, all other connected clients received the message, showing that we distributed the message correctly to all users connected to the server. The server was able to manage a busy server by allowing many users to be connected at the same time without crashing, partially due to the benefits of multi-threading. Each time a new message or client connects to the application, the server will create a separate thread to manage that user's connection to the server. This allows users to communicate in real-time without blocking the communication of other users.

The user interface we constructed was simple (text-based) and functioned properly for basic communication. After testing sending a message, it was received immediately, showing that the TCP connection was operational.

We also conducted testing for sending special characters (i.e., symbols and emojis), and incomplete messages (messages that were sent without using a newline) and our server handled this input very well. Our server showed some proper error messages or prompts without crashing in most cases. This shows that During the course of our development, we faced some challenges like Handling multiple client connections simultaneously was one challenge. In the beginning, the server was only able to talk to one client, but once we figured out how to use threading, we solved the problem. Error handling and managing edge cases was another issue we had to contend with, because there are a few scenarios I never thought about such as when a client is forcefully disconnected or when unexpected input is sent to the server.

Overall, we met our overall expectations. We learned a significant amount of information on how server-client systems operate, how data is transmitted and shared over a network, and how to build a real-time communication system using socket programming. The project was useful because it not only gave us insight into the theoretical aspects in a real-world instance, but we actually learned how to work through real problems.

TABLE :

Test Scenario	Expected Behavior	Actual Result	Status
Server startup	Server starts and displays waiting message	Server socket bound to port 12347 successfully	☑ Operational
Client connection	Client connects to server, server logs client address	Server GUI updates with connection details	☑ Connection established
Sending normal message	Message sent by client appears in server and all connected clients	Chat shown to all clients with proper formatting	☑ Message delivered
Special character message	Server rejects message with warning	Client receives "Your message is incorrect and incomplete"	⚠ Input rejected
Broadcast from server GUI	Server types message and it's broadcasted to all clients	All clients display the server message	☑ Broadcast successful
Client disconnects	Server logs disconnect, others continue unaffected	Disconnected cleanly, no errors	☑ Handled gracefully
Multiple clients sending messages concurrently	All clients' messages are received and broadcasted properly	Threaded handling ensured smooth simultaneous messages	☑ Concurrent success

The table summarizes all tests conducted on the client-server chat application. The tests have shown that the server started correctly, clients connected successfully, and normal messages are broadcasted to all users. Special character messages were correctly rejected with a warning. Server-initiated messages are received by all clients, disconnections from a client are handled

smoothly, and more than one client was able to send a message simultaneously without any issues. Overall, the application performs as expected in all conditions tested.

6. Conclusion and Future Work

Our Server and Client Chat Application is a practical project that shows how real-time communication works in a networked environment, it helped us learn about the main components, protocols, and methods used in real-time messaging systems. This knowledge guided us during the project and helped us solve technical challenges. It also helped us connect theoretical knowledge with real implementation, provided a deep understanding of how messages travel across a network and set the stage for more advanced features in future developments.

References

1. Basic concepts for understanding java language

https://youtu.be/UmnCZ7-9yDY?si=8XfLuq2xG0XfzaG

2.Client-Server Architecture Explained (YouTube)

https://www.youtube.com/watch?v=BqBKEXLqdvI

3. Client-Server Architecture from Scratch (Medium Article)
https://medium.com/@paritosh_30025/client-server-architecture-from-scratch-java-e5678c0af6c9

4. Client-Server Architecture – System Design (GeeksforGeeks)

https://www.geeksforgeeks.org/client-server-architecture-system-design/?ref=oin_asr7

5. Understanding Client-Server Architecture in Java

https://medium.com/@anirbandhara1997/understanding-client-server-architecture-in-

6. Complete playlist for the understanding of client server chat application
https://youtube.com/playlist?list=PLFXgDZCORpn1h_RXVTem2H61BkHwYlYkW&si=jlkdivMl3i_CuFPj