Report

Semester Project Part-1

Client-Server Chat Application

[Computer Networks]
[Spring 2025]

ABDUL HANAN (68469)

BHOOMI (67036)

UMMUL BANEEN (68539)

[2nd]

Submitted to:
[Poma Panezai]

[Department of Computer Science]
Faculty of Information and Communication Technology (FICT), BUITEMS, Quetta

Abstract

This project focuses on the implementation of a simple but practical Client-Server Chat Application using Java. The application is designed to allow multiple clients to connect to a single server, exchange messages in real-time, and understand the fundamental principles of computer network communication. The chat system uses socket programming in Java to establish a TCP/IP connection between clients and the server. Through this project, we explored concepts such as multithreading, socket communication, client-server architecture, and error handling. It gave us hands-on experience with Java's networking libraries and improved our understanding of how real-world applications like messaging services, multiplayer games, and remote services work.

CONTENTS

Table of Contents

1. PROBLEM STATEMENT:

In today's interconnected networks, seamless communication between devices is crucial. When a client initiates a request, it sends a message to a server, which must efficiently process multiple requests concurrently and respond accordingly. Our project aimed to design a robust client-server model that enables:

- Clients to connect to a central server via IP address and port number

- Multiple clients to be connected simultaneously

- Messages sent by one client to be broadcast to all others

- The server to handle connections and disconnections without interruption - Smooth error handling for issues like connection failures and invalid data

By achieving these goals, we developed a basic chat application that mirrors the functionality of real-world communication systems, providing a reliable and scalable solution for device interaction.

2. INTRODUCTION:

In the digital age, network communication is vital for nearly all online services, powering everything from web browsing to messaging apps. At the heart of these services lies the client-server architecture, a fundamental model where clients send requests and servers respond with relevant data or services. Our project brought this concept to life by developing a simple chat application that showcases this architecture. Leveraging Java's robust networking capabilities through socket APIs, we created a platform that allows multiple clients to connect to a server, exchange messages, and receive responses. By utilizing multithreading, the server efficiently manages multiple connections simultaneously. This compact implementation not only demonstrates the structure of real-world applications but also provides valuable insight into the data exchange process between machines over a network.

3. LITERATURE REVIEW:

To deepen our understanding of client-server systems, we researched various online resources, tutorials, and articles related to networking, socket programming, and Java-based communication systems. Our exploration began with the fundamentals of client-server architecture, which underpins most modern internet services, including email, websites, online gaming, and cloud storage. In this model, clients send requests, and servers respond with the required data or actions, enabling better organization, scalability, and efficient communication. A particularly informative resource was the YouTube video "Client-Server Architecture Explained", which provided a clear overview of network communication protocols like TCP/IP. Additionally, a video on Java Client-Server Programming offered a step-by-step guide to building a client-server system using Java sockets, which was instrumental in helping us develop our own chat application. Through our research, we gained both theoretical knowledge and practical skills, acquiring the confidence to design and implement our project. We learned about message structure, socket connections, and server handling of multiple clients, laying a solid foundation for our chat application development.

4. METHODOLOGY (DESIGN AND SIMULATION)

Design Approach

Our primary objective was to develop a real-time chat system that enables multiple clients to connect to a central server, send messages, and receive messages from other users instantly. We selected Java as our programming language due to its robust support for socket programming, multithreading, and input/output streams. The system employs a client-server architecture, where the server continuously runs and listens for incoming client connections. Each client connects to the server using a specified IP address and port number. To facilitate simultaneous communication among multiple clients, we implemented multithreading on the server side. Whenever a client connects, the server spawns a new thread to handle that client's communication, ensuring the server remains responsive and can manage multiple users concurrently. We utilized the TCP protocol through Java sockets to guarantee reliable data transfer. Messages are transmitted as plain text strings, and the server broadcasts every received message to all connected clients.

EXECUTION STEPS

1. **Server Startup**: The server program is initiated, binding to a specific port (such as 1234) and waiting for incoming connections.

2. **Client Connection:** Multiple clients can be launched, each connecting to the server by specifying the IP address and port number.
3. **Message Sending**: Once connected, clients can send messages to the server.
4. **Message Broadcasting:** The server receives the messages and broadcasts them to all other connected clients.
5. **Disconnection Handling:** If a client disconnects, the server removes it from the list and continues to serve the remaining clients without interruption.
6. **Server Shutdown:** The server can be manually stopped, and it will close all client connections gracefully.

We tested the system by running multiple client instances on the same machine, and the results confirmed that the server handled all clients correctly, delivering messages instantly and accurately.

USED TOOLS

- **Java:** The primary programming language for development and implementation.

-**Apache Netbeans:** The Integrated Development Environment (IDE) for writing, compiling, and running Java programs.

- **(link unavailable):** A tool for creating system flowcharts and visualizing the application's architecture.
- **Console (Terminal):** Used to simulate client-server communication and test the application.
- **GitHub:** A platform for sharing code and facilitating collaborative development. These tools enabled efficient development, collaboration, and testing of the chat application.

5. PROJECT IMPLEMENTATION:

Our chat system leverages Java's socket programming capabilities and consists of two primary components: the client and the server. **Client-Side Implementation** The

client is responsible for:

- Establishing a connection to the server using its IP address and port number.
- Sending user-input messages to the server.
- Receiving and displaying messages from the server.
- Handling exceptions when the server is unavailable or the connection is lost.
- Properly closing the connection when the user exits the chat.

To ensure seamless communication, we utilized:

- Input and output streams via Java's **`BufferedReader`** and **`PrintWriter`.**
- Multithreading to concurrently listen for incoming messages and send user-input messages.

Server-Side Implementation

The server is the central component, performing the following tasks:

- Binding to a specific port and waiting for incoming connections.
- Accepting new client connections using a **`ServerSocket`.**
- Creating a new thread for each client to handle multiple connections simultaneously.
- Receiving messages from clients and broadcasting them to all connected clients.
- Maintaining a list of active client connections.

- Handling disconnections, logging events, and shutting down gracefully when needed.

To manage client connections and broadcast messages efficiently, we:

- Used a `List` to store active client handlers.

- Implemented synchronized methods to send messages to all clients concurrently.

6. RESULTS AND DESIGN:

Upon completing the project, we conducted thorough testing by running the server and connecting multiple clients. The application performed as expected, demonstrating the following key features:

- **Real-time Messaging:** Clients could send and receive messages instantly.

- **Efficient Multithreading**: The server handled multiple clients simultaneously using threads, without any crashes or performance issues.

- **Robust Error Handling:** The application effectively managed connection losses and incorrect inputs.

- **Instant Broadcasting**: Messages were broadcast to all connected clients in realtime.

- **Functional Simplicity:** The user interface was straightforward, and the application's functionality was reliable and efficient.

7. CONCLUSION AND FUTURE WORK:

This project provided invaluable experience in network programming, offering a deep understanding of client-server communication in real-world applications. By developing the chat application from the ground up, we gained hands-on knowledge of managing multiple connections, handling real-time data, and designing a software system. Additionally, the project enhanced our collaboration, problem-solving, and documentation skills. The insights we acquired will be highly beneficial in future courses related to networking and software development

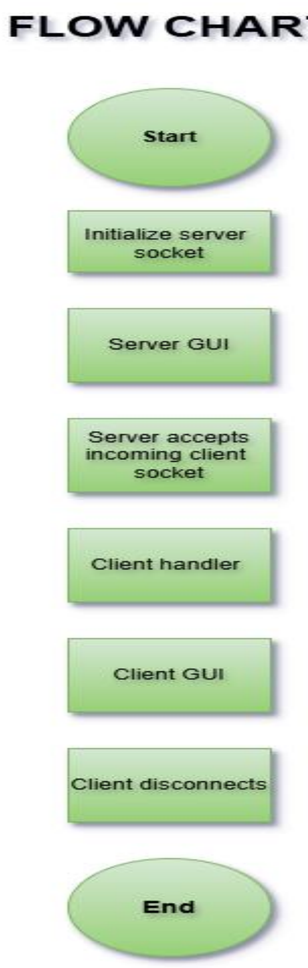
Future Plans

To further improve the application, we intend to:

.Implement a graphical user interface: utilize java FX or swing to create a more user-friendly interface.

. Store chat logs: save chat history in a text file or database for future reference.
.private messaging: introduce private messaging capabilities between users.
.message encryption: enhance security by encrypting messages to protect user data.

8.Flow chart:



9. REFERENCES:

- Client-Server Architecture Explained – YouTube
- Java Client-Server Programming – YouTube
- Client-Server Architecture from Scratch – Medium
- Client-Server Design – GeeksforGeeks