Report

Semester Project Part-1

Client-Server Chat Application

[Computer Networks]

[Spring 2025]

Aryan Khan (66369)

Syed Mohammad Zohaib (67689)

Muhammad Bilal (66324)

[2nd]

Submitted to:

[Poma Panezai]

[Department of Computer Science]

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

Abstract

This project takes advantage of the implementation of a multi-client real-time chat system with Java's socket programming to show fundamental concepts of client-server programming and TCP/IP communication. The system is capable of supporting multiple clients connecting to the main server, broadcasting and receiving messages in real-time, emulating aspects of programs such as WhatsApp or Slack. The main features are multithreaded server handling to accommodate multiple client connections, strong error handling to accommodate lost connections, and broadcasting of messages to all connected clients.

By this deployment, we learned critical networking concepts including:

Socket programming (Java ServerSocket and Socket classes) for reliable TCP connections.

I/O streams (BufferedReader, PrintWriter) to pass data.

Synchronization of the threads to avoid race conditions between client interactions.

The project also includes a simple GUI to display connection status and messaging exchange, in addition to terminal-based execution. By addressing issues such as scalability and stability of the connections, this application is a good example of learning about networked systems in real-world applications, such as multiplayer games and remote collaboration software. The exercise reinforced our skill with Java's networking APIs and emphasized the best practices in system design and protocol reliability.

CONTENTS

Table of Contents

1. Objective:

The objective of this project is to make an effective Client-Server chat application using Java. Which is intended to connect several clients to one server, share messages in real-time, and learn the basics of computer network communication. The chat systems use socket programming in Java to create a TCP connection among clients and servers. Further into the project we should be learning about multithreading, socket communication, client-server system/architecture, and error handling as a part of this project.

2. Introduction:

This Project is a basic representation of chat application used for messaging between client and server using java sockets. It communicates through input and output stream using the TCP protocol. This project helps us to understand of how real world messaging applications operate. How data is moved/exchanged between devices. Java socakets are used to establish and manage these kind of connections, ensuring the reliability. The most common examples of such communications in real world platforms are Whatsapp, Discord or Slack at a basic level. Furthermore the project also gives us an insight into how protocols like TCP ensure that data transimission is reliable, secure, and oredered.

3. Tools Used:

- Java (Version 8 or above)
- Apache NetBeans 26
- Command prompt / Terminal
- Draw.io
- MS Word
- GitHub

4. Client-Server Architecture:

The use of Java in such architecture is used to make simple yet effective chat systems where multiple clients could connect to a central server and exchange messages in real time. Plus, the language comes with a built-in support for socket programming, multithreading, and input/output streams, which is crucial for our project as beginners.

The architecture is based on client-server application. As long as the server stays active it will be constantly listening for incoming connections. Each client, when online, connects to the server using an IP address and port number. Furthermore, to allow multiple connections the server uses multithreading, this ensures that if one client is sending or receiving data, others can still connect to server without any problem.

All communication in this architecture uses TCP protocol, which is reliable. Each message sent to server from client is routed, which broadcasts it to all other connected clients, simulating a real-time group chat.

❖ Execution flow:

1. Start Server
2. Connect Clients
3. Messaging
4. Broadcasting
5. Disconnect Handling (If a client disconnects, it's removed from the active list)
6. Shutdown

❖ Concepts Used/Involved:

- Sockets: Used for creating communication channels between client and server.
- Streams: InputStreamReader, BufferReader, and PrintWriter are used to send and receive messages.
- Multithreading: Allows multiple clients to chat with server.

❖ How to compile and Run the Program:

1. Save the files as server.java and client.java.
2. Open the terminal and navigate to the source directory.
3. Compile using:
   javac Server.java
   javac Client.java
4. Run server:
   Java server
5. In a new terminal, run client:
   Java client
6. And then repeat step 5 because the multithreading is active.

5. Screenshots:

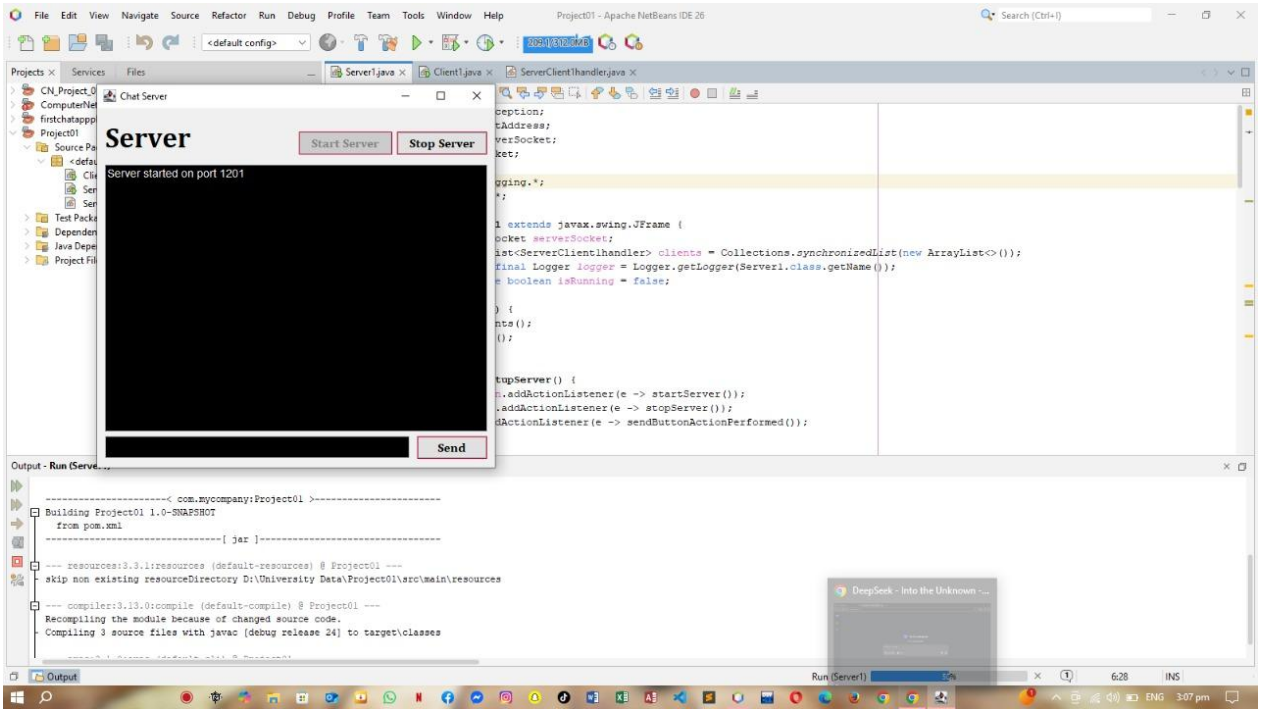


Figure: Server Terminal Started

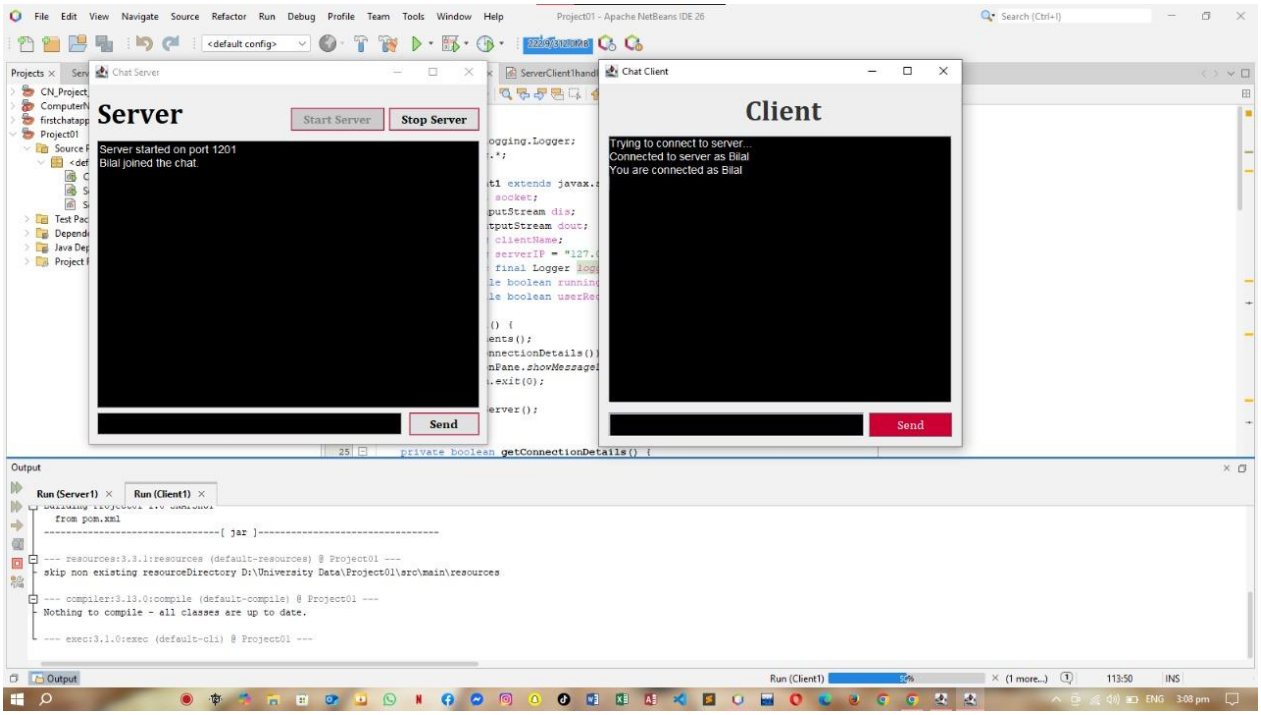


Figure: Client terminal displaying successful connection

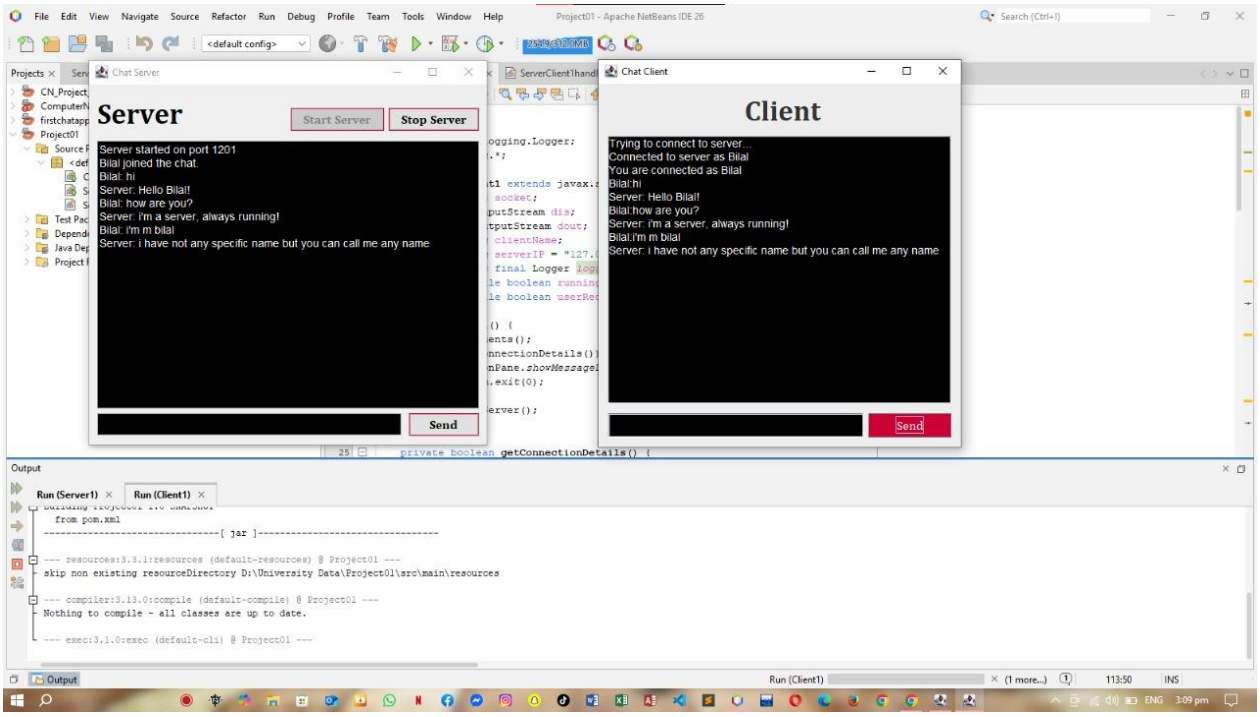


Figure: Client sends message, server replies

6. Project Implementation:

The server was designed to accept multiple client connections at the same time and use maximum advantage of multithreading to manage each connection separately. When the server starts, it listens on a fixed port. Clients can then connect to the server using the IP address and port.

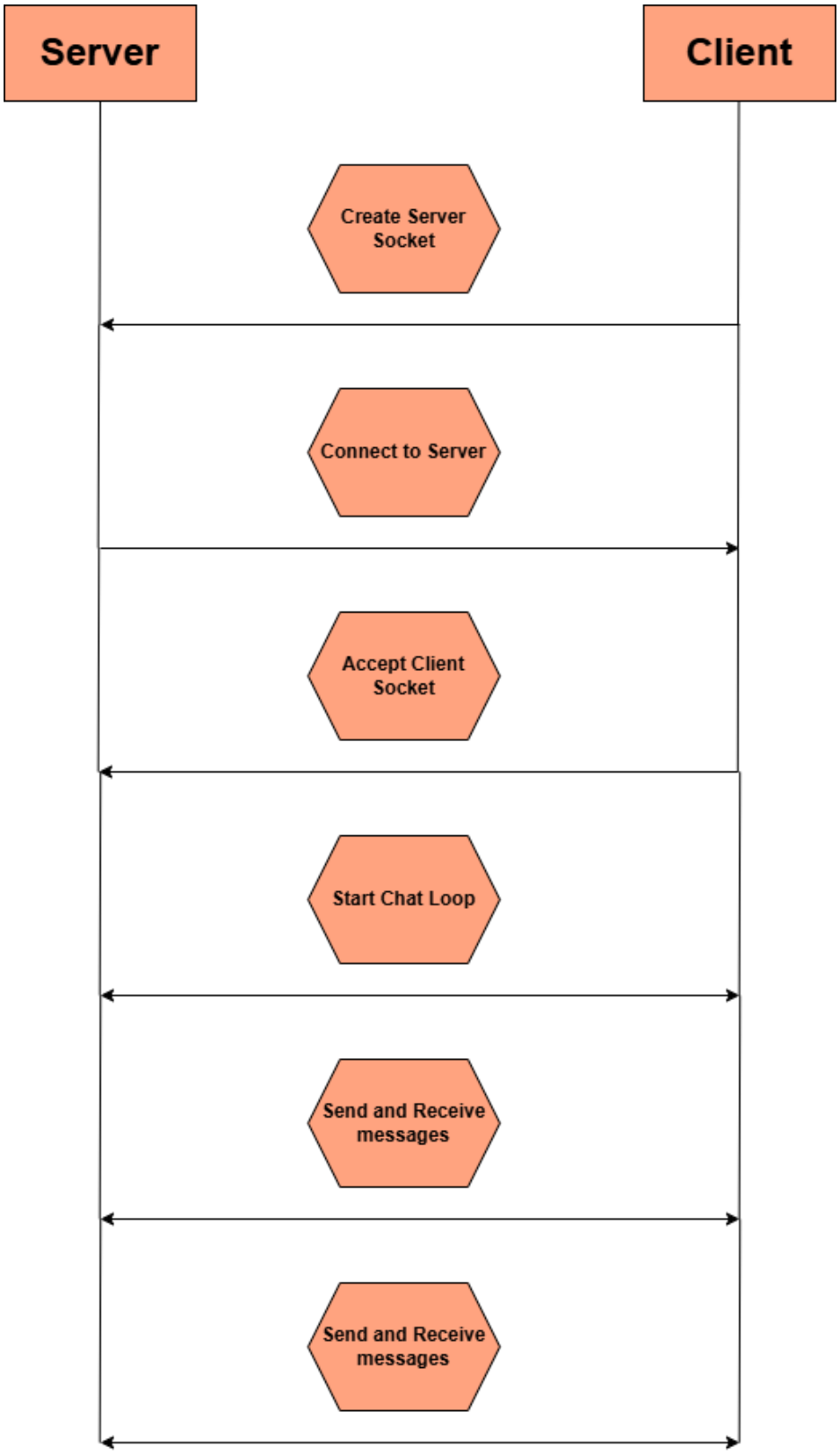
Once the connection process is completed, communication between the server and the client occurs using a reliable TCP stream.

Messages sent by a client are first received by the server and then broadcast to all connected clients, simulating a real-time group chat.

The application also shows / includes basic GUI elements that show connection status and message flow for both server and client interfaces.

- ❖ Key implementation features include:
  - Real-time message broadcasting to all clients
  - Auto connection and reconnection are supported
  - Basic error handling of dropped connections
  - Use of standard I/O streams and java socket

7. Flow Chart:


Server
Client
Create Server Socket
Connect to Server
Accept Client Socket
Start Chat Loop
Send and Receive messages
Send and Receive messages

8. Conclusion:

This project successfully showcases the design and implementation of a basic Client-Server chat application using java. It reflects / uses core concepts like socket programming, multithreading, TCP communication, and real-time data exchange. By building this system, we gained hands-on experience with how chat-based applications function internally.

Also, the use of Java's built in networking libraries provided a simplified yet powerful approach to understanding how multiple clients can connect and communicate efficiently with a single server. In addition, this project has improved our understanding of how to handle multiple users and maintain communication stability.

Overall, the project not only fulfilled academic objectives but also laid a foundation for more advanced networking projects and real-world applications.

9. References:

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