Report Semester

Project Part-1

[Client-Server Chat Application]

[Computer Network]

[Session 2024]

[68080, 68056, 68665]

[Farzana , Tayyaba , Matiullah Safi]

[2nd Semester]

Submitted to:

[Poma Panezai]

[Department of _Computer Science_]

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

PROJECT DOCUMENTATION

Abstract

This project is about creating a simple chat application using Java programming language which helps beginners learn how computers can talk to each other in real time using Java's networking features. The application has two parts: A server with multiple clients. The server can handle many users at the same time by the help of using threads and Each client has a user-friendly

interface made with Java Swing. Users can type and send messages, and all connected users can see them instantly. This project shows how Java can be used to build a real-time, multi-user chat system.

PROJECT DOCUMENTATION

Table of Contents

PROJECT DOCUMENTATION

1. Problem Statement

In today's computerized world, real-time communication between clients may be a key necessity for numerous applications, extending from client back to social interaction. A client-server chat application serves as an essential illustration of real-time informing, where numerous clients can interface to a central server to send and get content messages.

The client-server isolates messages dealing with obligations between the server, which oversees message dissemination and association control, and clients, which start and take an interest in discussions.

This venture centers on planning, executing, and illustrating a Java-based client-server chat framework. In this framework, clients can send content messages to the server, which at that point transfers those messages to other associated clients.

Key challenges include:

1. The challenge lies in ensuring correct protocol handling.
1. Managing multiple client connections.
2. Maintaining data integrity, and achieving smooth interaction between the client server component.
3. The goal is to build a working implementation that showcases the fundamental principles of client-server communication, higher potential issues (like connection handling, timeouts, or errors), and demonstrate practical application using Java.

The objective is to create a dependable and user-friendly chat application that outlines the center standards of client-server communication, handles real-time informing proficiently, and illustrates viable use cases utilizing Java organizing.

2. Introduction

Server

A server is a computer or software system that provides services, resources, or data to other computers, called clients, over a network.

PROJECT DOCUMENTATION

Client

A client is a computer or software application that requests services or resources from a server over a network.

Client-Server Chat Application

A Client-Server Chat Application is a software system that allows users to send and receive messages in real time over a network, using a client-server architecture.

The Function of Server in client and server chat application

Main Functions of the Server:

1. **Accept Client Connections:**
   - The server listens on a network port and accepts connection requests from multiple clients.
2. **Handle Multiple Clients:**
   - It manages simultaneous communication using threads or handling so that many users can chat at the same time.
3. **Receive Messages from Clients:**
   - The server receives messages sent by any client.
4. **Forward Messages to Other Clients:**
   - Once a message is received, the server relays (broadcasts) it to all other connected clients or to specific clients (in case of private messaging).
5. **Manage Client Sessions:**
   - It keeps track of connected users (login/logout), assigns IDs or usernames, and handles session timeouts.
6. **Error Handling:**
   - The server detects and manages issues like lost connections, invalid data, or unexpected errors gracefully.
7. **Ensure Message Order and Delivery:**

- It maintains the proper sequence of messages and ensures that each message reaches its target.

The function of Client in client and server chat application

The main functions are:

1. **Connect to the Server:**
   - The client starts by connecting to the server using the IP address of server and port number.
2. **Send Messages to the Server:**
   - When the user types a message, the client sends that message to the server.
3. **Receive Messages from the Server:**
   - The client listens for new messages from the server (messages sent by other users) and displays them to the user.
4. **Display User Interface:**
   - The client provides a user-friendly interface (text-based or graphical) to chat, view messages, and interact.
5. **Handle User Input:**
   - It takes input from the keyboard and sends it to the server.
6. **Maintain Connection:**
   - The client keeps a suitable connection with the server and may reconnect if disconnected.
7. **Error Display and Handling:**
   - If there's a problem (e.g., can't connect to the server), the client shows appropriate error messages.

3. Literature Review:

The Client-Server Chat Application is a simplified yet effective implementation of a real-time messaging system based on the traditional client-server model. Utilizing Java socket

programming, it enables reliable two-way communication over TCP/IP between a server and multiple clients. To support simultaneous user interactions, the system employs multi-threading, ensuring efficient and parallel communication. The graphical user interface, developed with Java Swing, provides an intuitive platform for users to send and receive messages. Basic input validation is implemented to maintain message integrity and prevent disruptions. While modern systems may adopt more advanced technologies, this application focuses on core networking principles, making it ideal for educational purposes and foundational understanding.

4. Methodology Design and Simulation)

In our project for developing the Client-Server Chat Application first we need to design a simple, interactive communication system using java socket programming for network communication and Java (swing) for Graphical user interface.

To do the process go with planning the system architecture, implementing client and server modules, adding a graphical user interface, and supporting multiple client connections.

We follow the following steps outline to build this project.

Step 1. Project Planning

Our main target for working on this project is, to create a real-time chat system with:

- GUI (Graphical user interface) for client and server.
- Support for multiple clients.
- Broadcasting the server message to all the connected clients.
- Real-time bidirectional messaging.
- Handling errors and connection notifications.

Tools and Technologies used:

- PC or laptop
- Apache NetBeans IDE: runs Java program.
- Java JDK: programing language and development kit.
- GUI Libraries like Java Swing: design GUI for server and clients.
- GitHub: to control version and code sharing.

- Draw.io: for creating diagrams and designing flowchart.
- Zip folder: to compress and package the complete project.

Step 2. Application Design:

This application follows a client-server model where:

Server listens on port, accepts multiple client connections, and handles broadcasting.

Each client connects to server, send messages and listens for incoming messages.

Flow of communication:

Clients send message to the server.

The server receives the message and broadcasts it to all connected clients without the sender.

Client and server both send message from their GUI.

Step 3. Implementation Steps:

1. **Server Implementation:**

1) **Server GUI creation:**
   - make a simple **JFrame** from with TextArea, **TextField, JLebel** and Button to display and broadcast message.
2) **Server socket initialization:**
   - In the **startServer ()** method, create a **serverSocket** to listen on port number **5050**
   - and display "waiting for client…" in the GUI.
3) **Handling multiple clients:** When a client connects
   - it accepts the socket connection
   - Assigns unique name using **AtomicInteger** such as client1, client 2, ...e.t.c
   - A new **ClientHandler** thread will be created for each client.
   - Each connected client will be stored in a shared vector **< ClientHandler>**for broadcasting.
4) **ClientHandler thread class:**

   A subclass of thread, that handles communication for a single client:
   - Reads message from its assigned client.
   - Broadcasts received message to all other clients.
   - Sends welcome message with client names.
   - Removes itself from client list it disconnects or exits.

5) **Broadcast mechanism:**
   - server Admin can type a message in the GUI and click send.
   - Message is validated and no special characters allowed.
   - The message is broadcasted to all clients through **sendMessage()**.
6) **Logging and filtering messages.**
   - Special characters like **@#$%^&*** are not allowed.
   - In case of sending blocked message, server replies with “error” or logs the incident in the server GUI.
7) **Error Handling and disconnection:**
   - If the client disconnects or sends “exit”, it’s thread stops and removes from the client list and the GUI displays disconnection status.

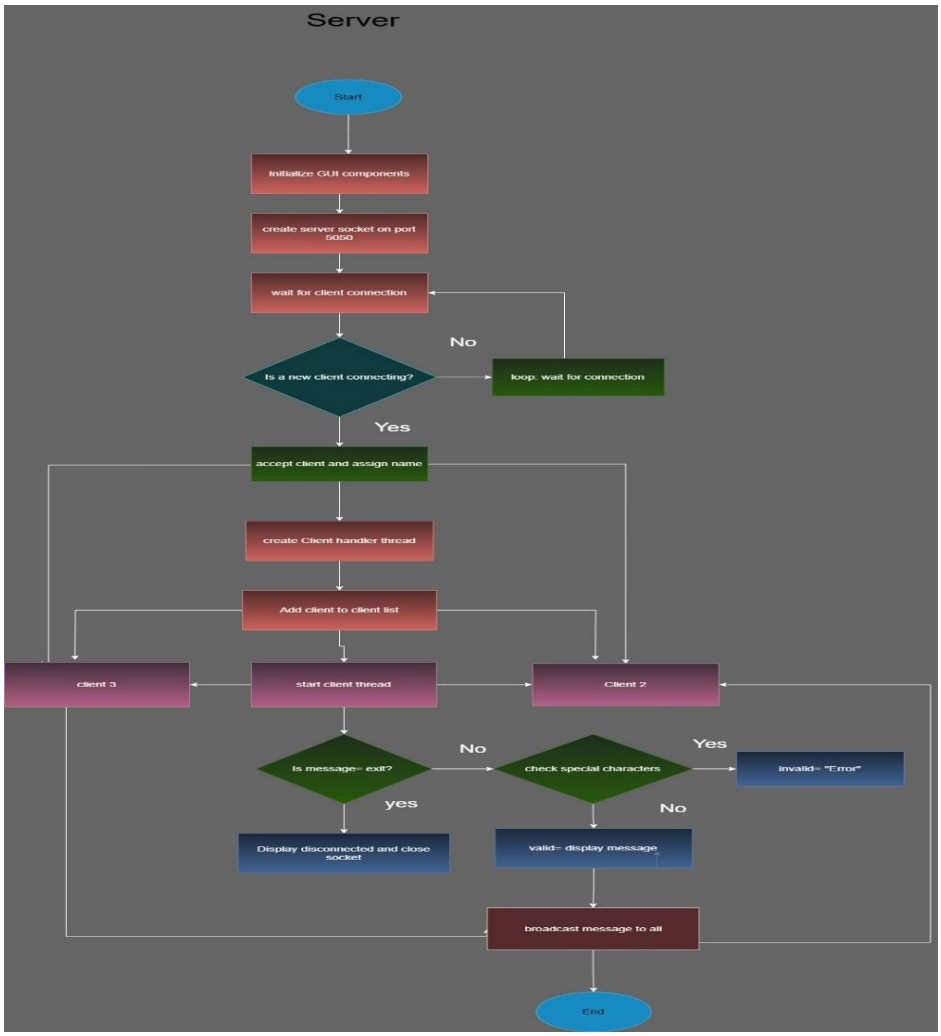

Figure 1Server Flowchart representation

-

2. Client implementation:

1) Client GUI creation:

- Create a **JFrame** with TextArea to display chat messages.
- Create TextField and send Button to type and send message.

2) Connecting to the server:

- In **connetToServer (),** use socket to connect to IP address (127.0.0.1) and port 1201
- Create Input and output streams with **DataInputStream** and **DataOutputStream** to receive and send messages.

3) Start Listening Thread:

- Start a separate thread to listen for incoming message using **readUTF()** and append received message to the TextArea.

4) Sending messages:

- When the user types message and clicks the send button, the message will be checked for special characters. If it is valid then it will be sent to server using **writeUTF().**

5) Receive unique client name:

- When the client connects, the server assigns a name like Client 1.
- The name is displayed in the client window which will be used for identification of the clients.

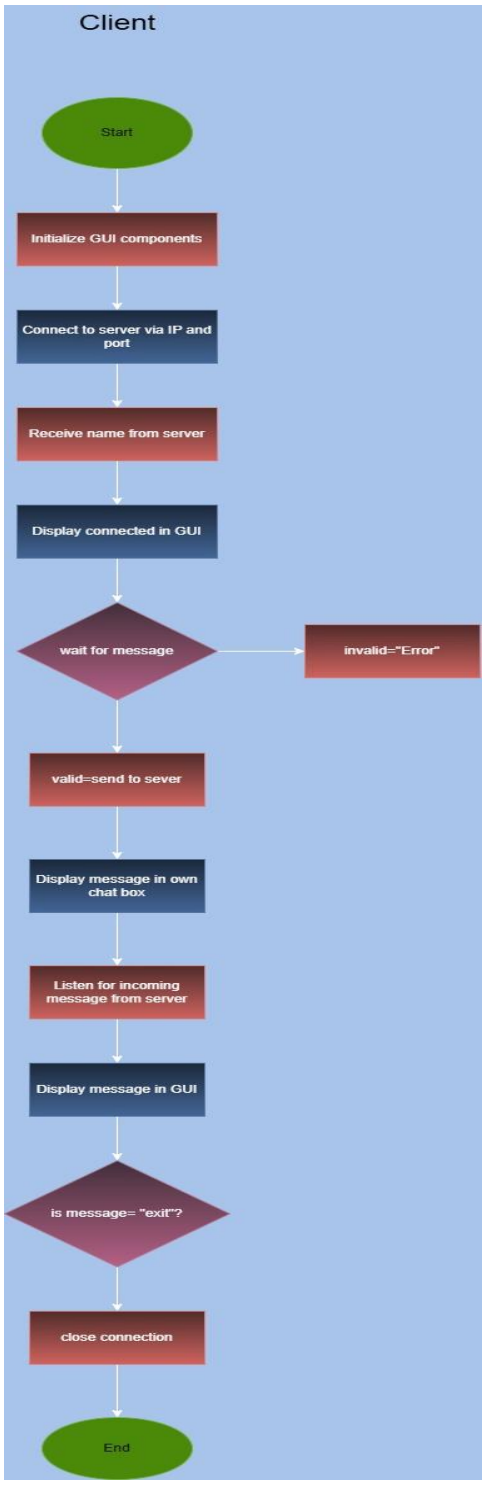

Figure 2 Client flowchart representation

6) **Receive and display message:**
   - Client listens for message continuously and the receives the message on the TextArea.
   - The listener thread ensures that this happens in real-time without blocking the GUI.
7) **Disconnect gracefully:**
   - If the client types "exit" or it closes the application, server will receive a disconnect message and input output stream and socket will close.
   - GUI will show a "disconnected" status.

5. Results and Discussion

The strategy come approximately in a valuable real-time chat application with back for various clients and a user-friendly GUI utilizing Java Swing. Communication between clients and the server was capably managed with utilizing connection programming and multithreading, ensuring reliable message broadcasting. The setup allowed users to connect at the same time and send messages instantly, while error handling kept the app running smoothly. Although the project was successful, it had some issues like no message encryption and a very basic user interface. Overall, the project achieved its goals and created a good base for building more advanced network applications.

Observations:

Test case	Expected outcome	Actual result	discussion
Client and server connection	Client connects to server successfully	Connection established successfully	Stable connection without losing any packet
Message broadcasting	Server broadcasts messages to all connected clients	All client received the message instantly	Efficient multi-client handling is confirmed
Unique client identification	Individual client gets a unique ID	Assigns unique ID like: client 1, client 2	Useful for debugging and tracking communication
GUI responsiveness	During operation interface remains responsive	GUI remained responsive during all test phases	There is no lag (swing-based design was stable and user-friendly)

Disconnection handling	Server has to log and manage client disconnection	Server showed disconnection logs in GUI	Handling of abrupt client exits without crashes is stable

6. Conclusion and Future Work

In conclusion, this project successfully created a basic real-
time chat system with GUI support, multi-
client handling and smooth news transmission. With Java sockets and swings, the system worked and
made it easier to use. However, it does not include characteristics such as encryption, authenticat
ion, or enhanced UI design. In the future, these can be added to improve security and user experi
ence.
Extensions of the system to support file releases or mobile platforms can also improve its usefuln
ess.

7. References

https://www.youtube.com/watch?v=dlacc6831zw&list=PLFXgDZCORpn1h_RXVTem2H61BkHwYlYkW

https://www.youtube.com/watch?v=hE63QdXS588

https://www.youtube.com/watch?v=BbAmT3up_58

https://www.youtube.com/watch?v=-xKgxqG411c

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