Roll No: B-89

Indira College of Commerce and Science, Pune.

A Networking Mini Project Report On

“CLIENT- SERVER CHAT APPLICATION USING JAVA”

Submitted by

RUSHIKESH KAILAS NARAWADE
Seat No: 4748675

Under the guidance of
APARNA MAM

Department of Computer Science

Introduction

This chat application project aims to provide a straightforward and effective solution for real-time communication between two users. The application is designed using Java Swing for the graphical user interface (GUI) and Java socket programming for network communication.

Key Features:

- **Real-Time Messaging:** Enables instant messaging between users.
- **User-Friendly Interface:** Both the client and server applications feature intuitive GUIs with chat areas, text fields for message input, and send buttons.
- **Socket-Based Communication:** Utilizes TCP/IP sockets to ensure reliable data transmission between the client and server.

Components:

1. **Client:**
   - **GUI:** Displays the chat history, provides a text field for message input, and a send button to transmit messages.
   - **Functionality:** Connects to the server, sends user messages, and displays received messages.
2. **Server:**
   - **GUI:** Similar to the client's GUI, used primarily to receive and display messages from the client.
   - **Functionality:** Listens for client connections, receives messages from the client, and can send messages back to the client.

Technical Details:

- **Programming Language:** Java is used for both the Backend and Frontend.
- **Networking:** Communication between the client and server is achieved using TCP/IP sockets, ensuring a reliable connection.

Workflow:

1. **Client Initialization:** The client application sets up its chat window and establishes a connection to the server.
2. **Message Sending:** Users can type messages in the text field and send them by clicking the send button. The message is transmitted to the server.
3. **Message Receiving:** Both the client and server continuously listen for incoming messages and update the chat display accordingly.

Benefits:

- **Instant Communication:** Provides real-time message exchange, improving the efficiency of communication.
- **Simplicity:** The application offers a straightforward interface and functionality, making it easy to use.

Networking Details

1. Client Code

Client Initialization:

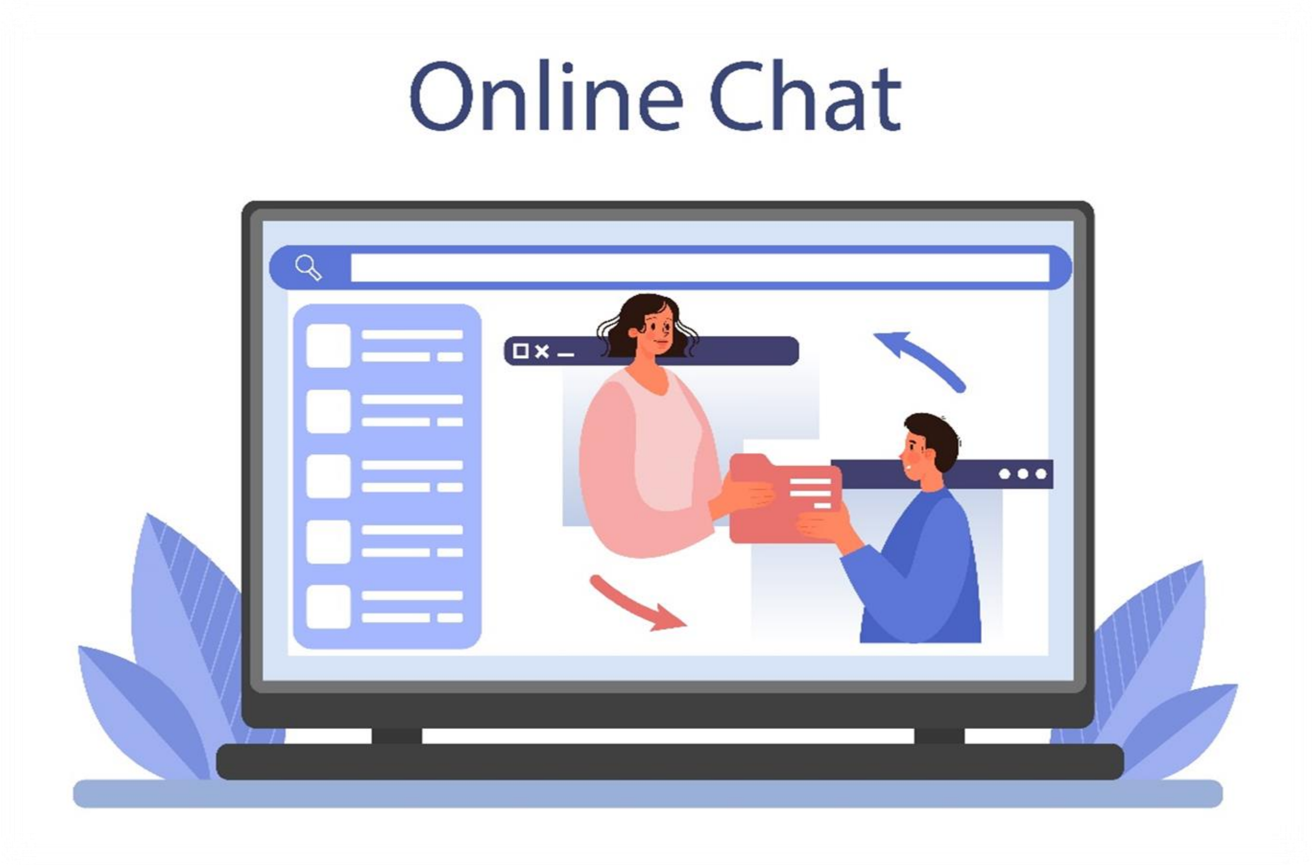
- **UI Setup:** The client sets up a `JFrame` with various components including a chat area (`a1`), a text field (`text`), and a send button (`send`).
- **Connection:**
  - The client establishes a connection to the server using
  - `Socket s = new Socket("127.0.0.1", 6001);`.
  - It uses `DataInputStream` to read messages from the server and `DataOutputStream` to send messages to the server.

Sending Messages:

- The `actionPerformed` method is triggered when the send button is clicked.
- It reads the text from the text field, formats it using `formatLabel(out)`, and sends it to the server using `dout.writeUTF(out);`.

Receiving Messages:

- In the `main` method, the client continuously reads messages from the server using `din.readUTF()` and updates the chat area accordingly.



Networking Details

2. Server Code

Server Initialization:

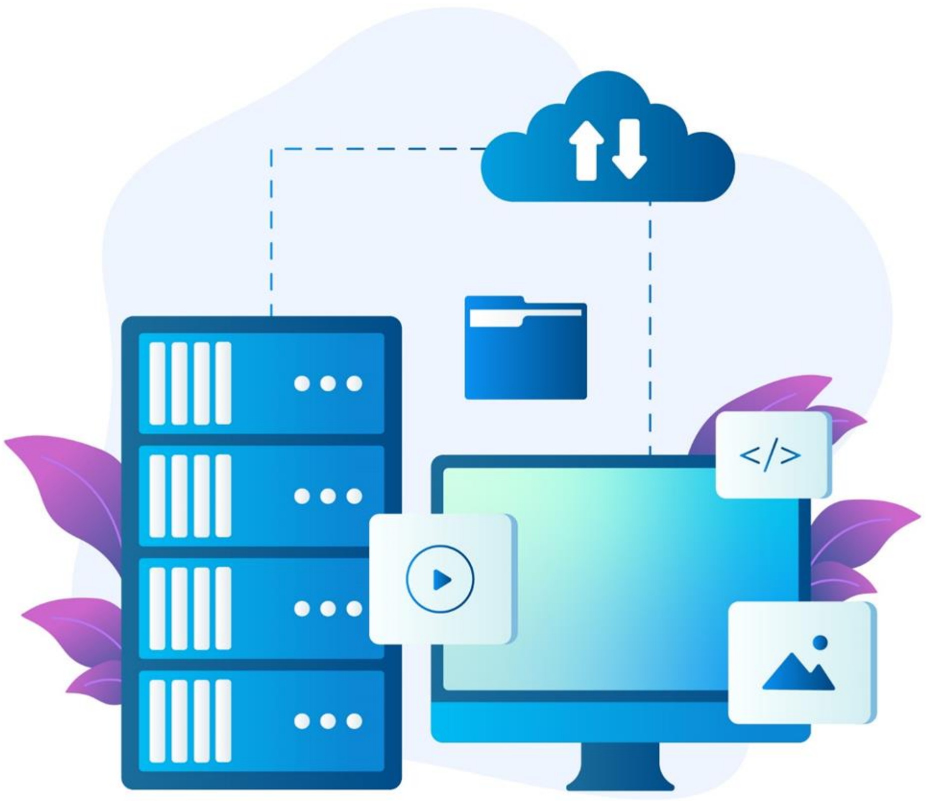
- **UI Setup:** Similar to the client, the server sets up a `JFrame` with similar components.
- **Connection:**
    - The server listens for incoming connections on port `6001` using
    - `ServerSocket skt = new ServerSocket(6001);`.
    - It accepts client connections using `Socket s = skt.accept();`.
    - It then sets up `DataInputStream` and `DataOutputStream` to handle communication.

Receiving Messages:

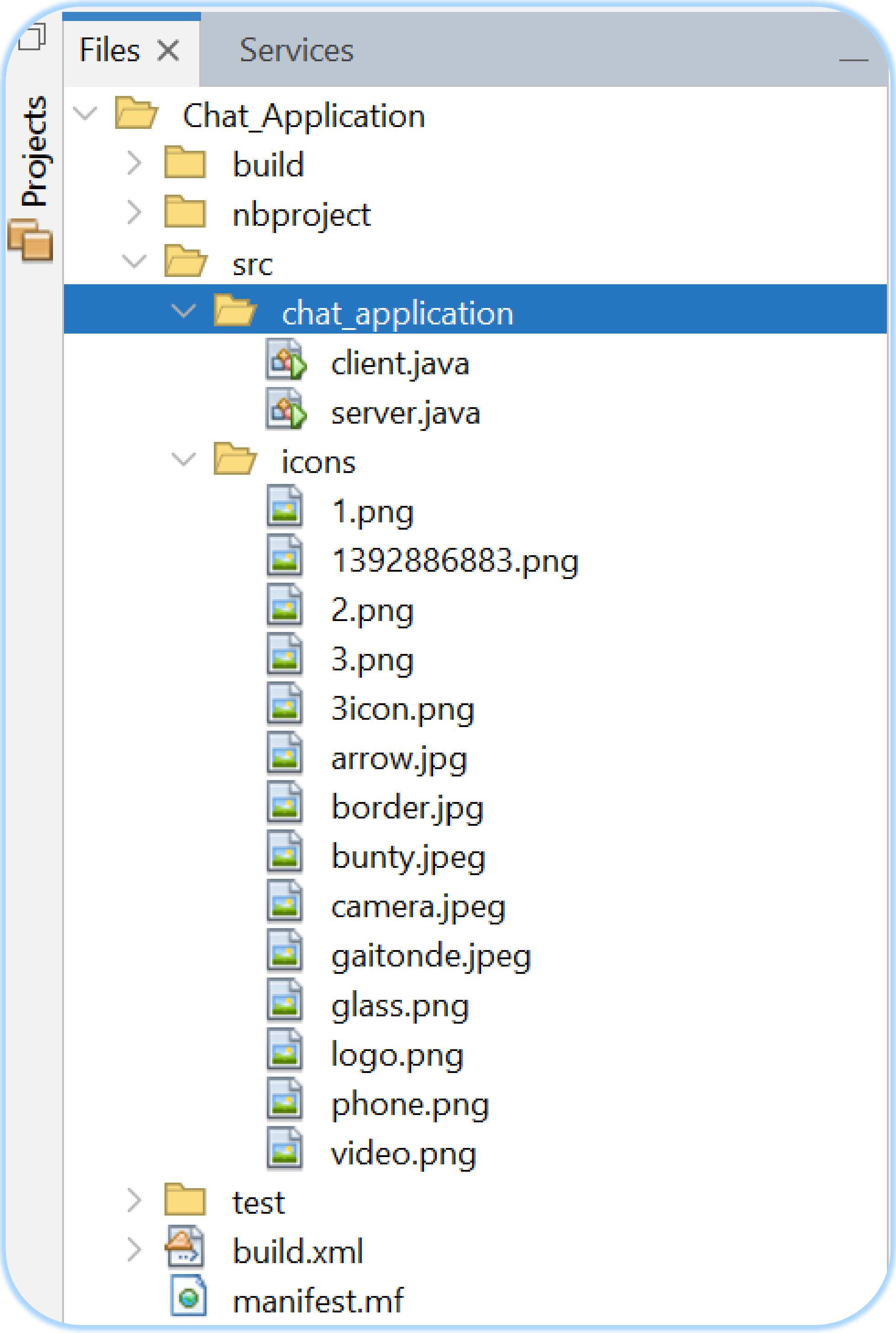
- The server reads messages from the client using `din.readUTF()`.

Sending Messages:

- The server does not send messages back to the client in this code. However, the setup is similar to how messages are received.



Project Structure



```
Chat_Application
    build
    nbproject
    src
        chat_application
            client.java
            server.java
        icons
            1.png
            1392886883.png
            2.png
            3.png
            3icon.png
            arrow.jpg
            border.jpg
            bunty.jpeg
            camera.jpeg
            gaitonde.jpeg
            glass.png
            logo.png
            phone.png
            video.png
    test
    build.xml
    manifest.mf
```

Code Details

Client Code Snippets:

1. **Establishing Connection:**

```
Socket s = new Socket("127.0.0.1", 6001);
DataInputStream din = new DataInputStream(s.getInputStream());
dout = new DataOutputStream(s.getOutputStream());
```

2. **Sending Messages:**

```
dout.writeUTF(out);
```

3. **Receiving Messages:**

```
String msg = din.readUTF();
```

Server Code Snippets:

1. **Listening for Connections:**

```
ServerSocket skt = new ServerSocket(6001);
```

2. **Accepting Client Connection:**

```
Socket s = skt.accept();
DataInputStream din = new DataInputStream(s.getInputStream());
dout = new DataOutputStream(s.getOutputStream());
```

3. **Receiving Messages:**

```
String msg = din.readUTF();
```

Networking Flow

Client Initialization:

1. **Connects to the Server:** The client initiates a connection to the server using a socket.
2. **Sends Messages Typed by the User:** The client captures user input from the text field and sends it to the server through the established socket connection.

Server Initialization:

1. **Listens for Incoming Connections from Clients:** The server creates a server socket and listens for incoming connection requests from clients.
2. **Accepts Connections and Reads Messages Sent by the Client:** When a client requests a connection, the server accepts it and establishes a communication channel. The server then reads messages sent by the client through the socket.

Message Exchange:

1. **Client Sends a Message:** The user types a message in the client's text field and clicks the send button. The message is transmitted to the server via the socket connection.
2. **Server Receives the Message:** The server reads the incoming message from the client through the socket.
3. **Processing or Displaying the Message:**
   - The server can process the received message (e.g., logging, filtering) if necessary.
   - The server displays the received message in its chat window.
4. **Server Sends a Response (if needed):** The server can also send messages back to the client. The client then reads and displays these messages in its chat window.

Summary

This project is a simple chat application with a client and a server, implemented in Java. The client and server communicate over a network using sockets. The main purpose of this project is to enable text-based communication between the client and server.
This project, created by Rushikesh Narawade, demonstrates basic network communication in Java by implementing a chat application with a client-server architecture.