VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELAGAVI 590018

Project Report on

RT-CHAT APPLICATION USING JAVA SWING & SOCKET

By

MOHAMMAD TALIB KHAN(1BF24CS176)
MOHIT KUMAR(1BF24CS178)
MUZAMIL ZAHOOR(1BF24CS184)
MOHAMMAD ASIF(1BF24CS174)

Under the Guidance of

MONISHA H M
Assistant Professor, Department of CSE
BMS College of Engineering

Work carried out at

Department of Computer Science and Engineering
BMS College of Engineering
(Autonomous college under VTU)
P.O. Box No.: 1908, Bull Temple Road, Bangalore-560 019

2025-2026

BMS COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the OOPS with JAVA project titled **RT-CHAT APPLICATION USING JAVA SWING & SOCKET** has been carried out by MOHAMMAD TALIB KHAN (1BF24CS176), MOHIT KUMAR(1BF24CS178), MUZAMIL ZAHOOR(1BF24CS184), MOHAMMAD ASIF(1BF24CS174) during the academic year 2025-2026.

Signature of the guide

MONISHA H M

Assistant Professor,

Department of Computer Science and Engineering

BMS College of Engineering, Bangalore

BMS COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, MOHAMMAD TALIB KHAN(1BF24CS176), MOHIT KUMAR(1BF24CS178), MUZAMIL ZAHOOR(1BF24CS184), MOHAMMAD ASIF(1BF24CS174), students of 3rd Semester, B.E, Department of Computer Science and Engineering, BMS College of Engineering, Bangalore, hereby declare that, this project work entitled **RT-CHAT APPLICATION USING JAVA SWING & SOCKET** has been carried out by us under the guidance of **MONISHA H M,** Assistant Professor, Department of CSE, BMS College of Engineering, Bangalore during the academic semester Sep-Dec 2025. We also declare that to the best of our knowledge and belief, the project reported here is not from part of any other report by any other students.

Signature of the Candidates

MOHAMMAD TALIB KHAN(1BF24CS176)

MOHIT KUMAR(1BF24CS178)

MUZAMIL ZAHOOR(1BF24CS184)

MOHAMMAD ASIF(1BF24CS174)

TABLE OF CONTENTS

Sr. No.	Title	Page No.
1	Problem Statement	1
2	Introduction	2
3	Overview of the Project (Block Diagram)	4
4	Tools Used	5
5	OOPs Concepts Used & Explanation	6
6	Implementation / Code	11
7	Result / Snapshots	14
8	References	15

1. Problem Statement

Real-time communication applications are widely used today, but building a **responsive and scalable chat system** that supports concurrent users, private and global messaging, and a graphical interface using core Java technologies can be challenging. This project addresses the need for a **multi-threaded chat system** that allows multiple users to communicate simultaneously over a network with a clean UI and real-time message delivery.

The RT-CHAT system aims to solve:

- Efficient handling of **multiple client connections concurrently**
- Real-time message routing between clients
- Intuitive graphical user interface for users
- Support for **both private and global messaging**

This project demonstrates knowledge of sockets, threads, GUI design, and network communication in Java.

2. Introduction

RT-CHAT is a **multi-threaded real-time chat application** developed in **Java** that enables users connected over a network to exchange messages instantly. The system follows a **classic client–server architecture** where a central server manages incoming connections and routes messages between connected clients.

The application uses:

- **Java Socket programming** to establish TCP/IP communication
- **Swing/AWT** for building the user-interface
- **Threads** to ensure concurrent handling of multiple clients

The project illustrates the principles of network programming, concurrency, and GUI development in Java. It supports both **private one-to-one chats** and **global broadcasts** to all connected users.

3. Overview of the Project

What is RT-CHAT?

RT-CHAT is a sophisticated **chat system written in Java** that enables real-time text communication between multiple users on a network. It implements a **centralized server** that accepts client connections and manages message delivery in real time.

Key Components

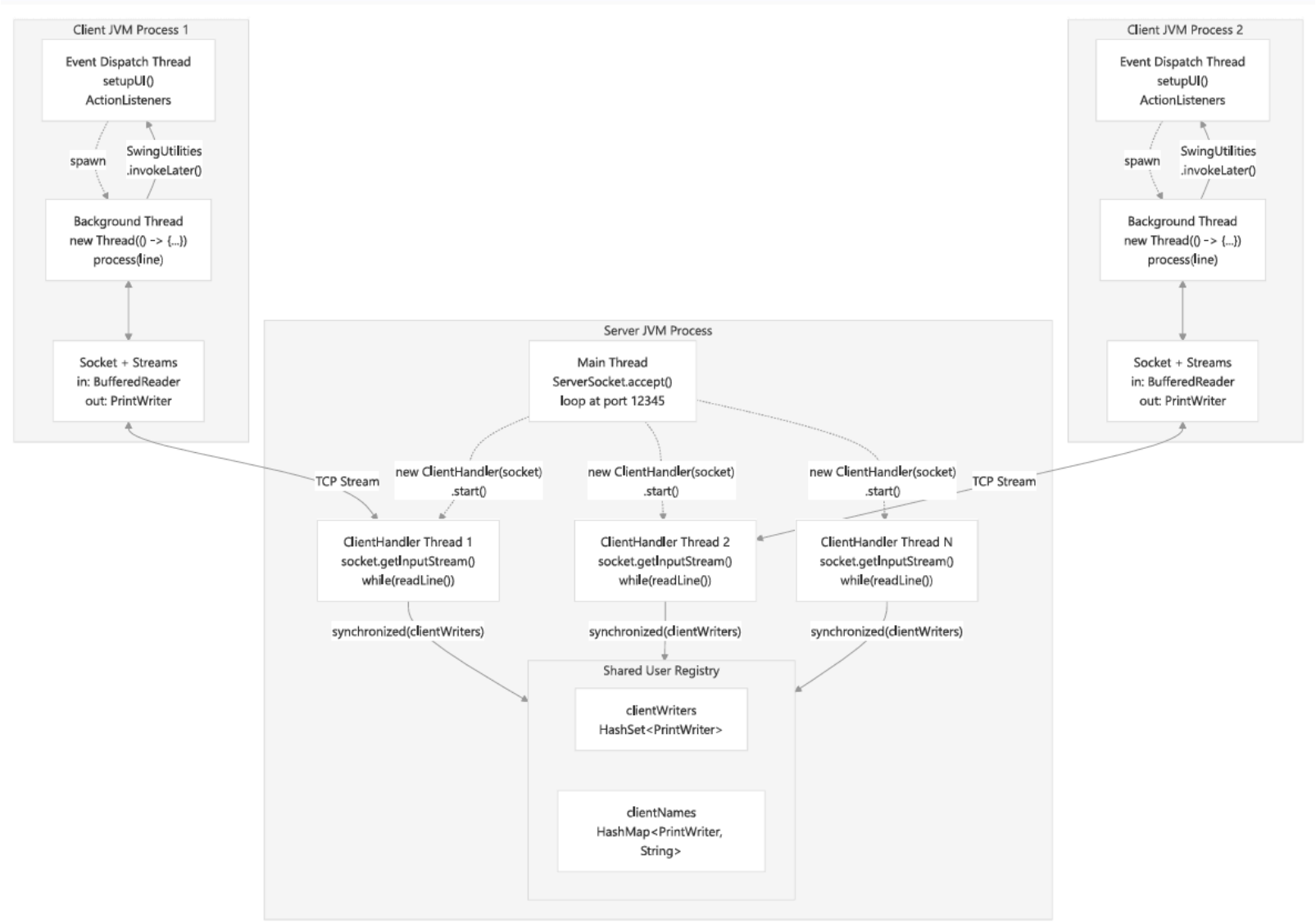
- **ChatServer** – Listens for and manages client connections, routes messages
- **ClientHandler** – Dedicated thread per client to handle communication
- **ChatClient** – Java GUI client that allows users to send and receive messages

Features

- **Modern UI:** Sleek interface with a toggleable Dark/Light Mode.
- **Private Messaging:** Secure, direct messaging between users.
- **Global Announcements:** A dedicated channel for broadcasting messages to all connected users.
- **Unread Counters:** Visual notification counters for missed messages.
- **Emoji Support:** Integrated emoji picker for expressive communication.
- **Dynamic Avatars:** Auto-generated user avatars with initials.
- **Real-time Updates:** Instant message delivery and status updates (Join/Leave).
- **Chat History Retention:** New users receive the last 50 messages upon joining.
- **Offline Status:** Users who leave are marked as "(Offline)" and can still be messaged or viewed.

System Architecture

RT-CHAT uses a **multi-threaded server model**, where each client connection spawns a separate thread to handle its communication. This allows multiple clients to chat concurrently without blocking operations. The communication protocol uses **line-delimited messages** with simple prefixes to distinguish message types (like private message, global broadcast, user join/leave actions).



4. Tools Used

1. **Java (JDK)**
   Java is used as the core programming language to develop the RT-CHAT application. It provides built-in support for networking, multithreading, and GUI development.

2. **Java Socket Programming**
   Socket programming is used to establish real-time communication between the server and multiple clients using TCP/IP.

3. **Java Swing / AWT**
   Swing and AWT are used to design the graphical user interface of the chat client, including message display, input fields, buttons, and emojis.

4. **Multithreading**
   Multithreading is used on the server side to handle multiple client connections simultaneously, ensuring smooth real-time communication.

5. **GitHub**
   GitHub is used for version control, source code management, and collaboration.

5. OOPs Concepts Used & Explanation

1. **Encapsulation**
   Encapsulation is used to bind data and methods within classes. Important data such as socket connections, input/output streams, and user details are accessed only through defined methods.

2. **Inheritance**
   Inheritance is used to extend existing Java classes, especially for GUI components and thread handling, which helps in code reusability and reduces redundancy.

3. **Polymorphism**
   Polymorphism allows methods to behave differently based on the object invoking them. This is mainly used in message handling and thread execution.

4. **Abstraction**
   Abstraction hides complex implementation details such as networking operations and message routing by separating them into dedicated classes.

5. **Multithreading**
   Multithreading is used so that each client connection runs on a separate thread, allowing multiple users to communicate simultaneously without blocking the server.

6. **Synchronization**
   Synchronization is used to control access to shared resources such as the list of connected clients, preventing data inconsistency when multiple threads operate at the same time.

7. **Exception Handling**
   Exception handling is used to manage runtime errors such as network failures and unexpected client disconnections, ensuring smooth application execution.

6. Implementation / Code

Server Implementation

The `ChatServer` class acts as the central hub of the application. It listens for incoming client connections on **port 12345** and maintains two important data structures:

- **clientWriters**: A `HashSet<PrintWriter>` used to keep track of all active client output streams.
- **clientNames**: A `HashMap<PrintWriter, String>` used to map each client's output stream to its corresponding username.

The server continuously runs a loop that accepts new client connections. For every new connection, a separate `ClientHandler` thread is created and started, allowing multiple clients to communicate concurrently.

```
public class ChatServer {

private static final int PORT = 12345;

    private static Map<PrintWriter, String> clientNames = new HashMap<>();

    private static Set<PrintWriter> clientWriters = new HashSet<>();

    public static void main(String[] args) {

        System.out.println("Chat Server is running on port " + PORT);

        try (ServerSocket serverSocket = new ServerSocket(PORT)) {

            while (true) {

                new ClientHandler(serverSocket.accept()).start();}

        } catch (IOException e) {

            e.printStackTrace();}}}
```

Client Handler Implementation

Each client connection is handled by an individual `ClientHandler` object. The core logic is implemented inside the `run()` method:

- Reads the client's username during the initial handshake process
- Registers the client with the server's shared data structures
- Enters a continuous message-processing loop
- Handles **private messages** using the `PM:` prefix
- Handles **global broadcast messages** sent to all connected clients
- Performs proper cleanup when a client disconnects

This design ensures isolated handling of each client while maintaining synchronized access to shared resources.

```
public class ClientHandler extends Thread {

    private Socket socket;

    private PrintWriter out;

    private BufferedReader in;

    public void run() {

        try {

                                    in      =      new      BufferedReader(new
InputStreamReader(socket.getInputStream()));

            out = new PrintWriter(socket.getOutputStream(), true);



            // Read name first
```

```
            String nameLine = in.readLine();

            String myName = "Unknown";

            if (nameLine != null && nameLine.startsWith("NAME:")) {

                myName = nameLine.substring(5);

            }



            ChatServer.addClient(myName, out);

            out.println("ID:" + myName);
```

Client Implementation

The `ChatClient` class represents the complete client-side application and is responsible for both the user interface and network communication:

- **UI Components**: Uses `CardLayout` to switch between the contact list view and the chat window.
- **Network Layer**: Executes all network operations on a background thread to ensure the UI remains responsive.
- **Message Processing**: The `process()` method interprets incoming server messages and updates the user interface accordingly.
- **Custom Components**: Implements `ContactRenderer` for styled contact lists and `Bubble` components for displaying chat messages in a conversational format.

This class integrates networking, UI rendering, and message handling into a cohesive client application.

```
public class ChatClient {

    JFrame frame = new JFrame("Chat App");

    CardLayout cardLayout = new CardLayout();

    JPanel mainPanel = new JPanel(cardLayout);

    // Connection data

    String myName, serverAddress = "localhost";

    int serverPort = 12345;

    String currentChatPeer = null;

    PrintWriter out;

    // Data models

    DefaultListModel<String> contactsModel = new DefaultListModel<>();

    Map<String, JPanel> chatPanels = new HashMap<>();

    Map<String, Integer> unreadMap = new HashMap<>();

}
```

Communication Protocol

RT-CHAT uses a simple **text-based communication protocol** to exchange messages between the server and clients:

- `NAME:username` – Client registration during connection
- `PM:target:message` – Private message to a specific user

- *Plain text* – Global broadcast message
- `ID:`, `PRESENT:`, `JOIN:`, `LEFT:` – Server notifications for user status and presence

This protocol allows efficient message interpretation and routing with minimal overhead.

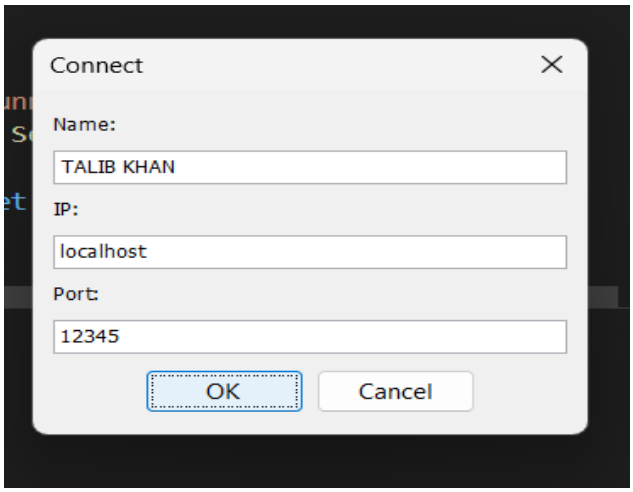
```
void connect() {
    new Thread(() -> {
        try (Socket s = new Socket(serverAddress, serverPort)) {
            out = new PrintWriter(s.getOutputStream(), true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(s.getInputStream())
            );
            out.println("NAME:" + myName);
            String l;
            while ((l = in.readLine()) != null) {
                final String line = l;
                SwingUtilities.invokeLater(() -> process(line));}
        } catch (Exception e) {
            SwingUtilities.invokeLater(() ->
                JOptionPane.showMessageDialog(frame, "Connection Error")
            ); }
    }).start();
}
```

7. Results:

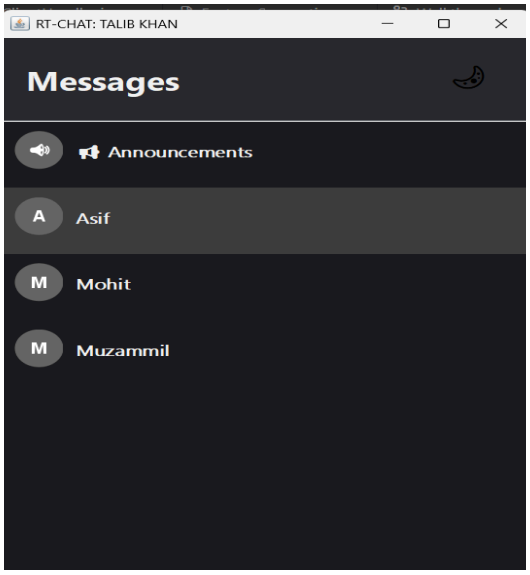
The **RT-CHAT** application delivers a fully functional **real-time chat experience** with the following visible results.

User Interface Features

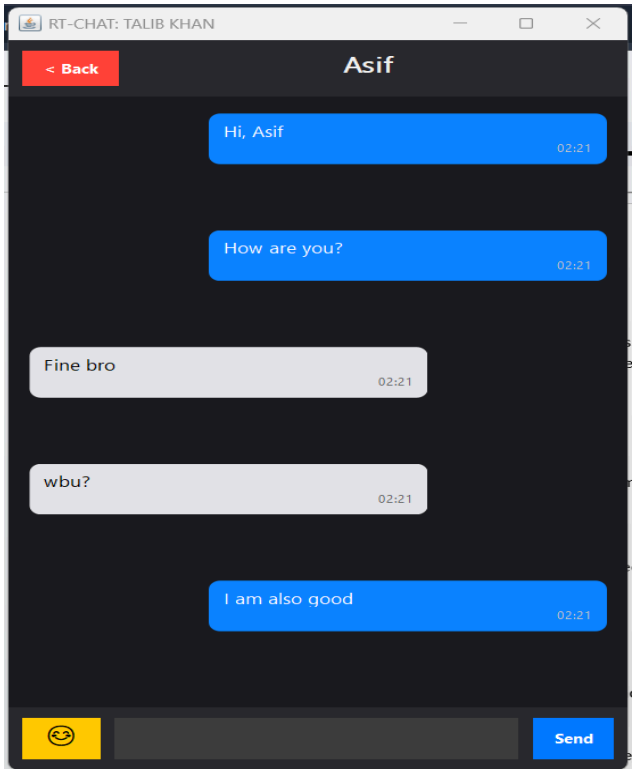
- **Login Dialog**
  Captures the username, server IP address, and port number required for connection setup.



- **Contact List**
  Displays all connected users with circular avatars showing user initials and unread message counters highlighted in red brackets.



- **Chat Interface**
  Provides individual chat panels where message bubbles are aligned to the left for received messages and to the right for sent messages, along with timestamps.



- **Theme Toggle**
  Allows seamless switching between dark mode and light mode with appropriate

color schemes applied across the interface.

- **Emoji Picker**
  Includes a popup grid of 16 commonly used emojis to enhance text messages.

Functional Results

- **Real-time Messaging**
  Enables instant delivery of both private and global messages between users.

- **Presence Management**
  Automatically updates the contact list when users join or leave the chat application.

- **Unread Notifications**
  Displays visual indicators showing the number of unread messages for each contact.

- **Multi-client Support**
  Supports concurrent communication between multiple users without performance degradation.

- **Responsive Design**
  Ensures automatic scrolling to the latest messages and smooth transitions between different views.

8. References

- **Server Implementation**
  src/chat/ChatServer.java – Main server class handling client registration and message routing.
  (ChatServer.java: lines 1–79)

- **Client Handler**
  src/chat/ClientHandler.java – Thread responsible for per-client connection management.
  (ClientHandler.java: lines 1–68)

- **Client Implementation**
  src/chat/ChatClient.java – Complete client-side user interface and networking logic.
  (ChatClient.java: lines 1–378)

- **README File**
  README.md – Project overview and setup instructions.
  (README.md: lines 1–72)

- **Architecture Documentation**
  DeepWiki page detailing system architecture and component interactions.

- **Source Code Repository (GitHub)**
  RT-CHAT Project Source Code:
  https://github.com/talibk90/RT-CHAT

- **Project Documentation (DeepWiki)**
  Architecture, implementation details, and documentation:
  https://deepwiki.com/talibk90/RT-CHAT