

**Shripriti**

Educational & IT Hub

Step towards your Success

**Project Report**

**On**

**Group Chatting Application**

**Submitted by**

**Chetan Kailas Banait**

**MCA II YEAR**

**SEM IV**

**Under the Guidance of**

**Prof. Pranjali Ulhe**

**Assistant Professor**

**Submitted to**

**Shripriti**

**Educational & IT Hub**

**Academic Year 2025-26**



“**Group Chatting Application**”



**Shripriti**

Educational & IT Hub

Step towards your Success

## DECLARATION

We hereby declare that the Project work titled **“Group Chatting Application”** submitted to **Shripriti Educational & IT Hub** is a record of an original work done by us under the guidance of Prof. Pranjali Ulhe, Assistant Professor, School of Allied Sciences, Datta Meghe Institute of Higher Education & Research, Sawangi (Meghe) Wardha in the AY 2025-26 for a partial fulfillment of requirement.

This report has not been submitted to any other University or Institute for the award of any Internship

**MCA II Year Semester – IV : Chetan Kailas Banait**

**Place: Wardha, MH**

**Date: \_\_/\_\_/\_\_\_\_**



**Shripriti**

Educational & IT Hub

Step towards your Success

## ACKNOWLEDGEMENT

We would like to express our special thanks of gratitude’s to our Project guide Prof. Pranjali Ulhe, Faculty of Science and Technology, School of Allied Sciences, Datta Meghe Institute of Higher Education & Research, Sawangi (Meghe), Wardha for his/her able guidance and support in completing this report. We would like to extend my gratitude to and **Shripriti Educational & IT Hub ,** for providing me with all the facility that was required to complete this report successfully.

We also thank the management of **Shripriti Educational & IT Hub**  for providing me/us state of the art infrastructure and the opportunity to embark this Project. At last but not the least we are thankful to all my all teachers, staff who have been always helping and encouraging me/us throughout the period of this project.

**MCA II Year Semester – IV : Chetan Kailas Banait**

**Place: Wardha, MH**

**Table of Content**

|  |  |  |
| --- | --- | --- |
| **Chapter** | **Chapter Name** | **Page No** |
| 1 | Cover Page |  |
| 2 | Acknowledgment |  |
| 3 | Table of Contents |  |
| 4 | Introduction • Purpose of the Internship • Overview of Expectations • Technologies and Stack Covered |  |
| 5 | Weekly Breakdown • Week-by-week Summary • Challenges and Solutions |  |
| 6 | Mini Projects • Description and Features • Screenshots |  |
| 7 | Final Project:  • Project Overview  • Objective and Motivation  • Features Implemented  • Frontend & Backend Technologies Used  • Database Schema  • APIs Created  • Deployment Details |  |
| 8 | Learning Outcomes • Key Takeaways • Tools Learned • Real-world Experience |  |
| 9 | Conclusion • Internship Impact • Future Scope |  |
| 10 | Appendix • Code Snippets • GitHub Links • Extra Screenshots • References |  |

### Introduction

### ****Purpose of the Internship****

The primary purpose of this internship was to gain practical, hands-on experience in software development through the design and implementation of a real-world application. By working on an Employee Management System, I aimed to apply theoretical concepts learned during my academic training, understand the software development lifecycle in a professional setting, and enhance my problem-solving and critical thinking skills. This experience helped bridge the gap between academic knowledge and its application in the industry.

### ****Overview of Expectations****

Throughout the internship, I was expected to:

Understand the core requirements of a company-oriented HR management system.

Learn and apply full-stack development techniques using modern tools and technologies.

Design and manage a backend database to securely store and handle employee information.

Develop intuitive and responsive user interfaces for both Admin and Employee roles.

Test and debug the application to ensure robust and error-free functionality.

Create comprehensive documentation covering system design, code structure, and user guidance.

Deploy the final application on a local machine or a cloud server, as required.

#### Technologies and Stack Covered

**Programming Language:** Java  
**GUI Libraries:**

**AWT (Abstract Window Toolkit):** For foundational components such as Labels, TextFields, and Buttons.

**Swing:** To build enhanced, user-friendly GUIs with components like JFrame, JPanel, JTable, and JButton.

**Database:**

File Handling (or optionally **MySQL** for more advanced implementations)

**IDE Used:**

IntelliJ IDEA / NetBeans / Eclipse (based on user preference)

### ****Key Concepts Practiced****

Event Handling

Exception Handling

MVC Architecture (if followed)

CRUD Operations via GUI

This technology stack laid a strong foundation in desktop GUI development using Core Java. It also improved my ability to write structured, maintainable code while focusing on user experience and system functionality.

**Weekly Breakdown**

### ****Week 1: Project Overview & Setup****

**Summary:**  
The first week was dedicated to understanding the project scope and setting up the development environment. I analyzed the requirements of the **Employee Management System**, which is designed to manage employee profiles, job roles, and attendance data. A basic framework of the system was laid out.

**Challenges & Solutions:**  
Setting up the database connection was initially a hurdle. I addressed this by integrating JDBC and ensuring the appropriate driver libraries were configured in the IDE.

### ****Week 2: Core Functionalities – Employee Registration****

**Summary:**  
Developed the **Employee Registration** module, allowing input of essential details such as name, employee ID, designation, department, and contact info.

**Challenges & Solutions:**  
Input validation and managing null or malformed data proved to be challenging. I implemented conditional logic and Java exception handling to ensure data integrity and provide feedback to users.

### ****Week 3: Employee Data Retrieval and Display****

**Summary:**  
Implemented features for retrieving and displaying employee data in a tabular format. Added search and filter functionalities to enhance data accessibility.

**Challenges & Solutions:**  
Efficiently displaying large datasets required optimization. This was achieved through optimized SQL queries and pagination logic to break data into manageable chunks.

### ****Week 4: Updating and Deleting Employee Records****

**Summary:**  
Focused on adding features for **updating** existing employee records and **deleting** records when necessary.

**Challenges & Solutions:**  
Ensuring data consistency and avoiding accidental deletions was a key concern. I introduced confirmation dialogs and transaction handling to manage critical operations securely.

### ****Week 5: Employee Attendance Management****

**Summary:**  
Built a module for employees to **mark attendance**, enabling the system to track working hours and generate attendance status.

**Challenges & Solutions:**  
Preventing multiple entries for a single day required strict validation. I implemented date-specific checks to allow only one entry per employee per day.

### ****Week 6: Reporting & Exporting Data****

**Summary:**  
Created a reporting module that generates summaries for employee performance, attendance, and working hours. Integrated export options in **CSV** format.

**Challenges & Solutions:**  
Formatting and exporting data reliably was a challenge. I utilized Java I/O and the **OpenCSV** library to ensure structured and readable data export.

### ****Week 7: Final Testing & Bug Fixing****

**Summary:**  
This week involved thorough testing of all modules. I focused on identifying bugs, especially in edge cases, and enhancing system stability.

**Challenges & Solutions:**  
Minor inconsistencies such as duplicate records or misaligned data were found. These were resolved through improved validation and extended test cases.

### ****Week 8: Documentation & Final Presentation****

**Summary:**  
Compiled detailed documentation covering all modules, including technical specifications, usage instructions, and system flow diagrams. Presented the system to peers and mentors, and received valuable feedback.

**Challenges & Solutions:**  
Condensing technical details into user-friendly documentation was challenging. I tackled this by breaking down each component with clear visuals and step-by-step guides.

#### Mini Projects

#### ****1. Group Chatting Application****

**Description:**  
 This desktop-based application is developed using **Core Java**, **Socket Programming**, and **Multithreading**. It enables real-time communication between multiple users in a group chat format. Designed for LAN-based networks, the system includes a Server and multiple Client modules for message broadcasting and reception.

**Key Features:**

**User Login:** Basic username-based login for identification.

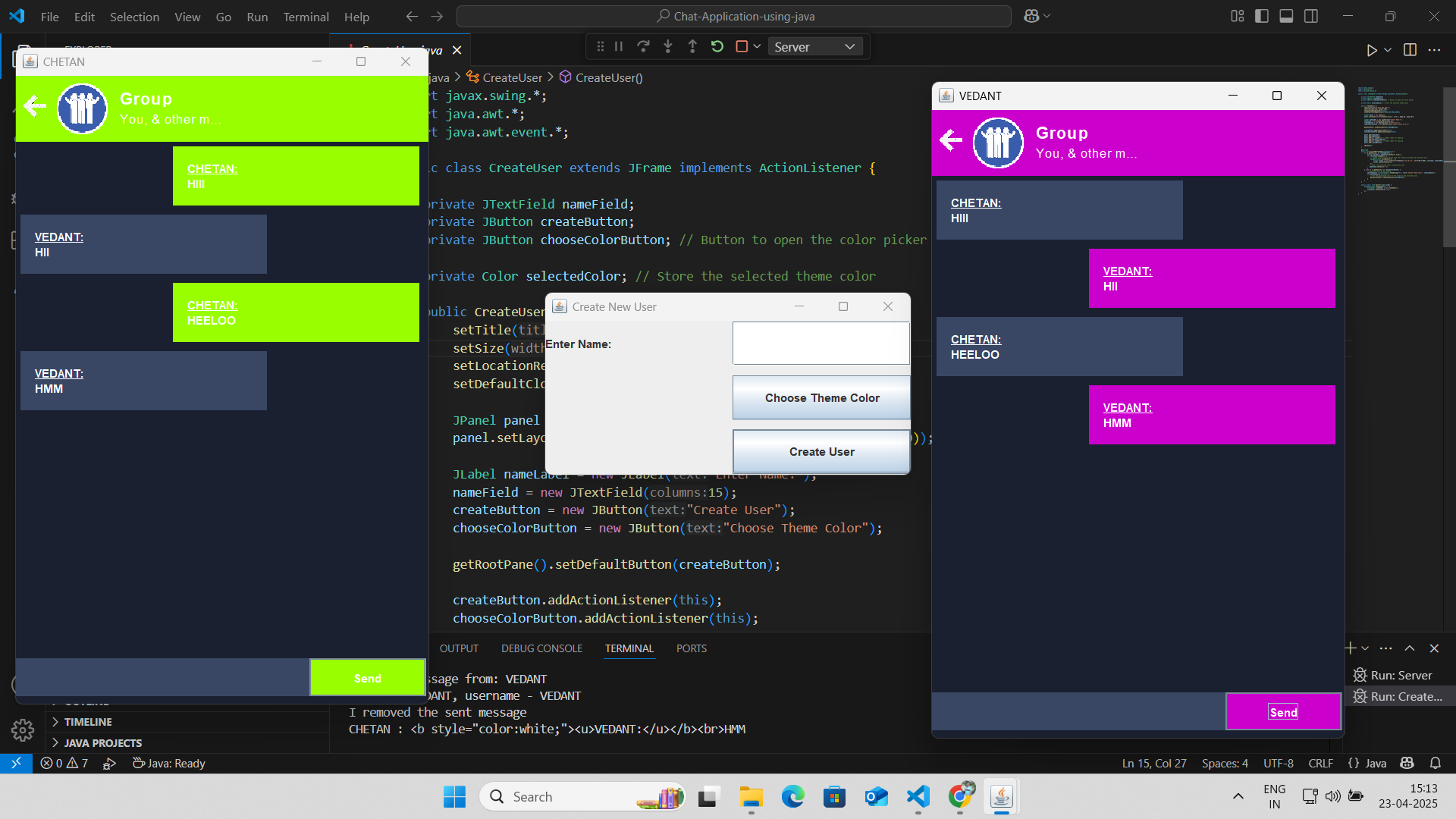
**Group Chat Functionality:** Real-time messaging between connected clients.

**Server-Client Architecture:** Centralized server manages all connections and message distribution.

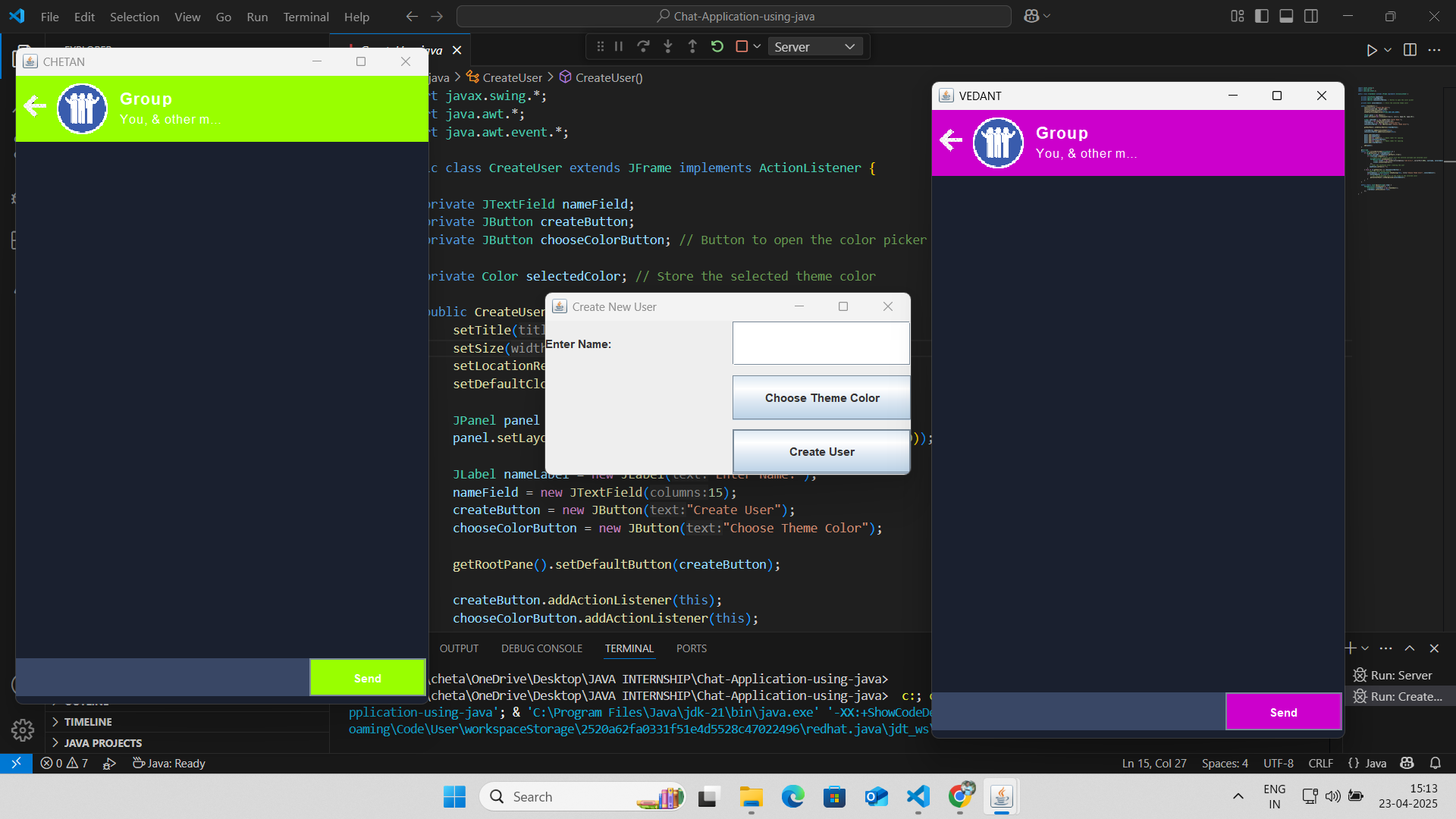
**Multithreading:** Each client connection handled via separate threads to support simultaneous communication.

**Message Broadcast:** Messages sent by any user are broadcast to all connected users.

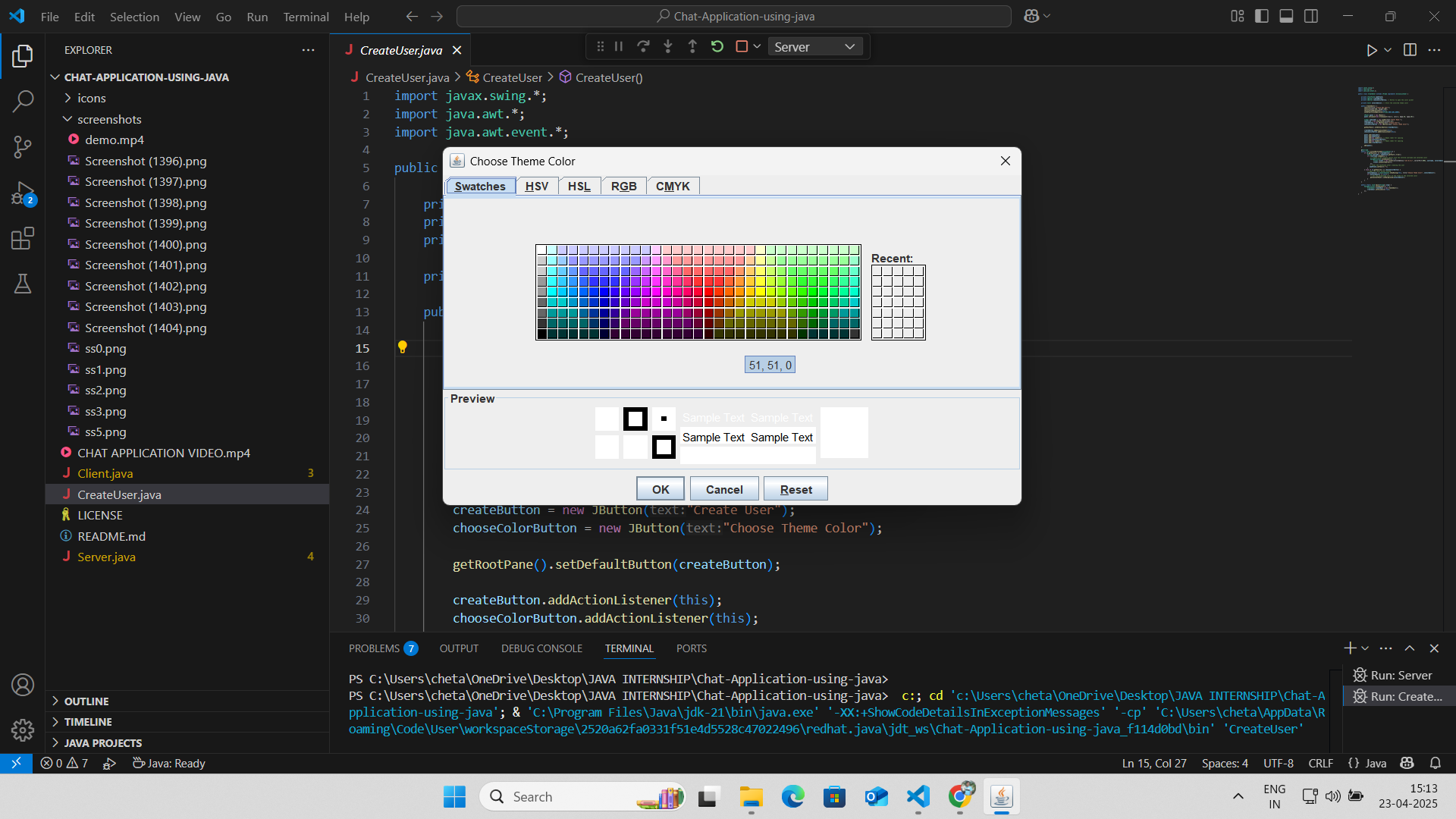
**Clean UI:** Simple and user-friendly GUI using **Swing**.



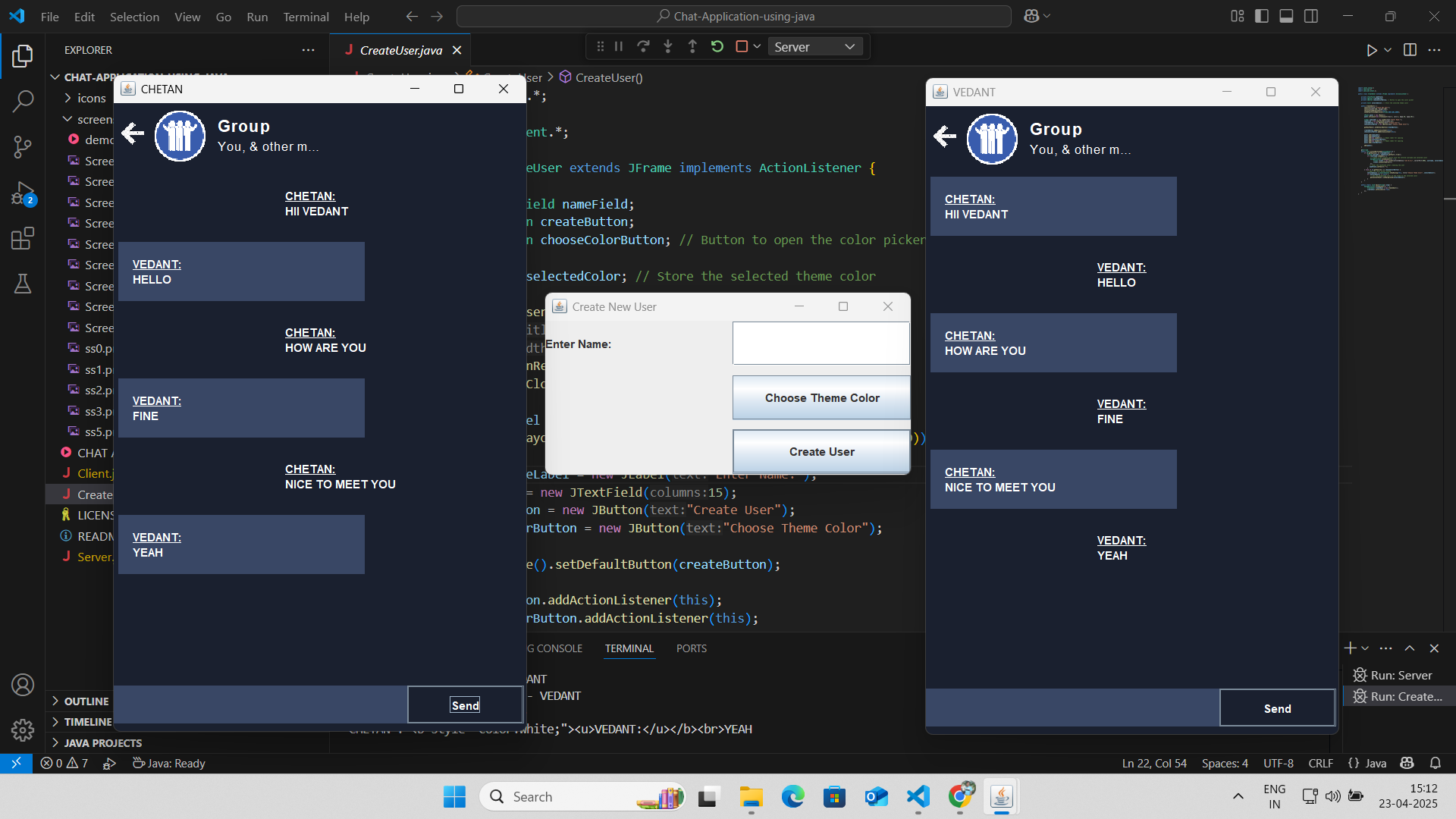
*Figure 1.:- Chatting with each other*



*Figure 3.:- Creating a User*



*Figure 2.:- Choose the Color Design*



*Figure 4.:- chatting with Two User*

**Final Project:**

### ****Project Overview****

The **Group Chatting Application** is a desktop-based LAN chat system developed using **Core Java**, with key implementation of **Socket Programming** and **Multithreading**. It allows multiple users to engage in real-time group conversations within a local network environment. The application follows a client-server architecture, where a centralized server handles all the message routing, ensuring that messages from one client are broadcasted to all other connected clients.

This system is especially useful in environments where internal communication is required but external internet access is limited or unavailable, such as in computer labs, corporate offices, or educational institutions.

### ****Objective and Motivation****

The primary objective of this project is to provide an efficient and user-friendly internal communication platform for small networks. Existing chat applications often require internet connectivity or have high system requirements. This application is lightweight, runs on LAN, and does not require any third-party installations or servers.

The motivation behind the project came from recognizing the need for secure, fast, and local communication tools within institutes and organizations, especially during lab sessions or collaborative activities.

### ****Features Implemented****

**Username-based Login:** Each user enters a unique name on login to identify themselves in the chat.

**Real-time Messaging:** Messages sent by one user are instantly received by all connected users in the chatroom.

**Centralized Server:** A server application handles all incoming messages and broadcasts them to connected clients.

**Multithreaded Client Handling:** Each client connection is managed on a separate thread, enabling simultaneous communication.

**Message Broadcasting:** Messages from one user are distributed to all users in the network via the server.

**Java Swing GUI:** A user-friendly graphical interface allows users to type and read messages easily in a text area and send them using a button or keyboard.

### ****Frontend & Backend Technologies Used****

| **Layer** | **Technology** |
| --- | --- |
| Frontend | Java Swing (for GUI design) |
| Backend | Core Java (Socket Programming, Threads) |
| IDE Used | NetBeans / Eclipse |
| Development | Java SDK (JDK 8 or above) |

### ****System Architecture****

The Group Chatting Application is based on a **Client-Server model**. Below are the main components:

**Server.java**

Listens on a specific TCP port.

Accepts incoming connections using ServerSocket.

Spawns a new thread for each connected client.

Receives messages and broadcasts to all connected clients.

**Client.java**

Connects to the server via IP and port.

Allows user to input messages via GUI.

Sends messages to the server using Socket output stream.

Listens to incoming messages from the server via a dedicated thread.

#### ****Communication Protocol****

Based on TCP/IP for reliable message delivery.

Uses predefined port (e.g., 1234).

Messages are sent and received as plain text strings via data streams.

### ****Core Java APIs & Custom Methods****

| **Method Name** | **Description** |
| --- | --- |
| startServer() | Initializes server socket and begins accepting clients. |
| startClient() | Connects the client to the server and initializes GUI. |
| broadcastMessage() | Sends messages from one client to all others via server. |
| receiveMessage() | Listens and displays incoming messages from the server. |

All messages are handled using BufferedReader, PrintWriter, Socket, and ServerSocket.

### ****Deployment Details****

The application is compiled and exported as a .jar file.

**Server** application is hosted on one machine in the local network.

**Client** applications are run on other machines connected to the same LAN.

All clients must know the server's IP address and connect to the same TCP port.

**Requirements:** Java Runtime Environment (JRE) must be installed on all machines.

**Learning Outcomes**

### ****Key Takeaways****

Developing the **Group Chatting Application** gave me hands-on experience in applying **Core Java** concepts to solve real-time communication problems. This project reinforced my understanding of key programming principles such as **multithreading**, **socket programming**, and **event-driven GUI development** using Java Swing.

I learned how to design a system based on the **client-server model**, handle concurrent user sessions through threads, and maintain stable communication across the network. The importance of **synchronization**, **data streams**, and **modular programming** became clearer as I structured the server to handle multiple client requests in parallel.

Beyond technical skills, this project emphasized the need for **proper planning, code documentation, and testing**, all of which are critical for building scalable and maintainable software solutions.

### ****Tools Learned****

During the development of this application, I worked extensively with a range of industry-relevant tools and technologies, including:

**Core Java** – For socket programming, multithreading, and backend logic

**Java Swing** – For designing an interactive and responsive desktop user interface

**Socket API (TCP/IP)** – For establishing real-time communication between server and clients

**NetBeans / Eclipse IDEs** – For writing, debugging, and running the application

**Java SDK** – For compiling and deploying the application

**Git** – For version control and collaboration, helping track and manage code changes

These tools not only strengthened my technical abilities but also introduced me to **standard practices in professional software development**, such as modularization, code versioning, and testing strategies.

### ****Real-world Experience****

This project closely resembled a real-world software development lifecycle, beginning from **requirement analysis** to **final deployment and testing**. Some practical challenges I encountered and resolved include:

Managing **multiple client threads** on the server without collision or message loss

Handling **network exceptions and failures**, ensuring clients reconnect smoothly

Designing a **simple yet functional GUI** for users to communicate effortlessly

Ensuring **synchronous broadcasting** of messages to all clients

Writing clean, maintainable, and **well-documented code**

This experience improved my **problem-solving mindset**, taught me to approach challenges analytically, and enhanced my communication skills when explaining technical progress or solutions. Most importantly, it provided a realistic preview of working in **team-based, real-world IT projects**, preparing me for professional software development roles.

**Conclusion**

### ****Internship Impact****

This internship played a pivotal role in shaping both my technical skills and professional outlook. Through the development of the Group Chatting Application, I was able to apply theoretical knowledge of Core Java, Socket Programming, and Multithreading in a practical, real-world scenario.

Building a LAN-based chat system deepened my understanding of client-server architecture, synchronous message handling, and real-time communication protocols. I learned how to create responsive user interfaces using Java Swing, manage concurrent threads efficiently, and ensure reliable message broadcasting to multiple users.

In addition to enhancing my programming capabilities, the project taught me how to independently manage tasks, troubleshoot networking errors, and meet project timelines. Working with tools like NetBeans, Java SDK, and Git familiarized me with industry-standard development environments and best practices such as version control, code documentation, and modular design.

### ****Future Scope****

The current version of the Group Chatting Application provides a stable foundation for LAN-based communication, but it holds considerable potential for future expansion:

* Private Messaging: Add support for one-to-one or group-private chats in addition to broadcast messaging.
* Message Encryption: Implement end-to-end encryption using libraries like Java Cryptography Architecture (JCA) for secure communication.
* User Authentication: Enhance login with user verification and role-based access (admin, regular users).
* File Sharing: Allow users to exchange documents or images through the chat interface.
* Online Deployment: Extend the app to function over the internet using RESTful APIs, allowing communication beyond LAN.
* Web & Mobile Integration: Rebuild the system using Spring Boot (backend) and React or Flutter (frontend) for cross-platform support.
* Chat History & Storage: Integrate SQLite or MySQL to store chat history locally or on a server.
* Notifications & Status Indicators: Add real-time typing indicators and online/offline user tracking.

These features would transform the current LAN-only desktop application into a fully-featured, scalable, and secure messaging system. The internship has laid a strong foundation for such advancements, fueling my motivation to continue exploring network programming and full-stack development.

**Appendix**

### ****1. Code Snippets – Group Chatting Application****

Below are key code snippets that demonstrate core functionalities implemented in the Group Chatting Application.

### ****a. Server.java – Starting the Server and Accepting Clients****

java

Copy code

public class Server {

private static final int PORT = 1234;

private static Set<Socket> clientSockets = ConcurrentHashMap.newKeySet();

public static void main(String[] args) {

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

System.out.println("Server started on port " + PORT);

while (true) {

Socket clientSocket = serverSocket.accept();

clientSockets.add(clientSocket);

new Thread(new ClientHandler(clientSocket)).start();

}

} catch (IOException e) {

e.printStackTrace();

}

}

public static void broadcastMessage(String message, Socket sender) {

for (Socket client : clientSockets) {

if (!client.equals(sender)) {

try {

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

out.println(message);

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

}

### ****b. ClientHandler.java – Handling Individual Clients (Multithreading)****

java

Copy code

public class ClientHandler implements Runnable {

private Socket clientSocket;

public ClientHandler(Socket socket) {

this.clientSocket = socket;

}

@Override

public void run() {

try (

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

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

) {

String inputLine;

while ((inputLine = in.readLine()) != null) {

System.out.println("Received: " + inputLine);

Server.broadcastMessage(inputLine, clientSocket);

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

### ****c. Client.java – Sending and Receiving Messages****

java

Copy code

public class Client {

public static void main(String[] args) {

try (

Socket socket = new Socket("localhost", 1234);

BufferedReader userInput = new BufferedReader(new InputStreamReader(System.in));

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

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

) {

// Thread to listen for messages from server

new Thread(() -> {

String serverMessage;

try {

while ((serverMessage = in.readLine()) != null) {

System.out.println("Server: " + serverMessage);

}

} catch (IOException e) {

e.printStackTrace();

}

}).start();

// Sending messages to server

String inputLine;

while ((inputLine = userInput.readLine()) != null) {

out.println(inputLine);

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

These code snippets form the backbone of the Group Chatting Application and reflect practical applications of Java networking, concurrent programming, and basic client-server communication principles. Each module ensures robust interaction between users in a local network environment.

### ****2. GitHub Links****

You can find the complete source code and project files on GitHub:

### ****3. References****

· **Java™ Platform, Standard Edition Documentation – Oracle**  
Comprehensive reference for Core Java classes, packages, and APIs used for building the application.  
[https://docs.oracle.com/javase/8/docs/](https://docs.oracle.com/javase/8/docs/" \t "_new)

· **Java Networking (Socket Programming) – Oracle Tutorials**  
Guides used for implementing TCP-based client-server communication.  
[https://docs.oracle.com/javase/tutorial/networking/sockets/](https://docs.oracle.com/javase/tutorial/networking/sockets/" \t "_new)

· **Java Swing Tutorial – Oracle**  
Helped design and structure the graphical user interface for both server and client components.  
[https://docs.oracle.com/javase/tutorial/uiswing/](https://docs.oracle.com/javase/tutorial/uiswing/" \t "_new)

· **Java Concurrency Tutorial – Oracle**  
Reference for using threads to manage simultaneous chat sessions from multiple clients.  
[https://docs.oracle.com/javase/tutorial/essential/concurrency/](https://docs.oracle.com/javase/tutorial/essential/concurrency/" \t "_new)

· **GitHub: Java Chat Server Examples**  
Open-source references were studied for understanding architectural patterns in socket-based chat apps.  
[https://github.com](https://github.com" \t "_new)