



Green University of Bangladesh

*Department of Computer Science and Engineering (CSE)
Semester: Fall, Year: 2023, B.Sc. in CSE (Day)*

Multi-Client Chat Application using Socket Programming

*Course Title: Computer Networking Lab
Course Code: CSE 312
Section: 211 D1*

Students Details

Name	ID
Md. Didar Bhuiyan	211002038
Md. Mahdizzaman Utsha	211002130

*Submission Date: 10/01/2024
Course Teacher's Name: Ms. Tanpia Tasnim*

[For teachers use only: **Don't write anything inside this box**]

<u>Lab Project Status</u>	
Marks:	Signature:
Comments:	Date:

Contents

1	Introduction	3
1.1	Overview	3
1.2	Motivation	3
1.3	Problem Definition	3
1.3.1	Problem Statement	3
1.3.2	Complex Engineering Problem	4
1.4	Design Goals/Objectives	6
1.5	Application	6
2	Design/Development/Implementation of the Project	8
2.1	Introduction	8
2.2	Project Details	8
2.2.1	Project Features	8
2.3	Implementation	9
2.3.1	The workflow	9
2.3.2	Tools and libraries	10
2.3.3	Some Screenshots of Programming Codes	11
3	Performance Evaluation	15
3.1	Results Analysis/Testing	15
3.1.1	Initial Interface	15
3.1.2	Add Client	16
3.1.3	Client Join to the System	16
3.1.4	Multiple Clients Join to the System	17
3.1.5	Group and Private Messaging	18
3.1.6	Send Image	19
3.1.7	Send Emojis and Left Chat Option	20

3.2	Results Overall Discussion	21
4	Conclusion	22
4.1	Discussion	22
4.2	Limitations	22
4.3	Scope of Future Work	23

Chapter 1

Introduction

1.1 Overview

The Multi-Client Chat Application using Socket Programming is a real-time communication system. It employs a central server and multiple clients, allowing users to engage in private and group chats. Built with socket libraries, the project emphasizes simplicity and efficiency. Key features include connection management, private messaging, and group chat capabilities. The project showcases the practical application of socket programming for instant messaging in a networked environment, providing a valuable learning experience in computer networking.

1.2 Motivation

Here are some motivations which encourage us to do this project-

- Growing demand for real-time communication solutions
- Increasing reliance on instant messaging in various sectors
- Desire for efficient and secure communication in networked environments
- Opportunity to apply and enhance skills in socket programming
- Addressing the need for practical applications in computer networking education
- Exploring the challenges and solutions in developing a multi-client chat application

1.3 Problem Definition

1.3.1 Problem Statement

In the existing systems, following are the main drawbacks:

1. Scalability Issues:

- The existing system struggles to handle a large number of simultaneous connections, impacting its scalability for potential real-world usage.

2. Reliability Concerns:

- Instances of message loss or delayed delivery were observed during stress testing, indicating potential reliability issues that need to be addressed.

3. Security Vulnerabilities:

- The system exhibits vulnerabilities to potential security threats, highlighting the need for enhanced encryption and authentication mechanisms.

4. User Interface Limitations:

- User interface responsiveness decreases as the number of active clients increases, suggesting the need for optimizations in the UI design for better performance.

5. Cross-Platform Compatibility:

- The application faces compatibility issues across different operating systems, requiring adjustments to ensure a seamless user experience on various platforms.

6. Error Handling Deficiencies:

- Inadequate error handling was identified in certain scenarios, leading to unexpected application behavior. Enhancements in error detection and reporting are essential.

7. Limited Feature Set:

- The existing system lacks certain essential features such as file sharing and multimedia messaging, limiting its functionality compared to modern chat applications.

8. Testing Scenarios:

- The testing process did not cover all potential usage scenarios, necessitating a more comprehensive test plan to ensure robustness in various real-world conditions.

9. Resource Utilization:

- Resource utilization metrics, such as memory and CPU usage, indicate room for improvement, especially in scenarios with a high volume of concurrent users.

10. Documentation Gaps:

- Incomplete or unclear documentation was identified, hindering the ease of understanding and contributing to the application's codebase. Improved documentation is necessary for future development and maintenance.

Addressing these problem statements during the planning and development stages will contribute to the overall success and effectiveness of the The Multi-Client Chat Application using Socket Programming. [1] [2]

1.3.2 Complex Engineering Problem

Table 1.1: Summary of the attributes touched by the mentioned projects

Name of the P Attributes	Explain how to address
P1: Depth of knowledge required	The project necessitates an in-depth understanding of socket programming, network protocols, and the intricacies of real-time communication. Mastery of java programming language, specifically its socket libraries, is crucial to navigate the complexities involved in establishing and managing connections.
P2: Range of conflicting requirements	The application must reconcile conflicting requirements, balancing the need for low-latency communication with the imperative for robust security measures. Achieving this equilibrium involves addressing conflicting demands such as the demand for immediate message delivery and the necessity to implement secure authentication mechanisms.
P3: Depth of analysis required	A comprehensive analysis is imperative to resolve challenges associated with concurrent connections, message delivery reliability, and system scalability. This involves delving into performance metrics, understanding bottlenecks, and optimizing the application to ensure responsiveness under diverse and dynamic conditions.
P4: Familiarity of issues	While socket programming is a well-established domain, implementing a multi-client chat application introduces unique challenges. Managing multiple simultaneous connections, creating and managing chat rooms, and ensuring data integrity require a nuanced understanding. Familiarity with these specific issues is crucial for crafting a solution that meets the project's objectives.
P5: Extent of applicable codes	The project demands the application of advanced coding techniques. Efficient communication protocols, robust error handling mechanisms, and secure data transmission all require a nuanced coding approach. The integration of diverse coding paradigms is necessary to build a cohesive and high-performing solution.

P6: Extent of stakeholder involvement and conflicting requirements	Stakeholder involvement is paramount in addressing conflicting requirements emanating from end-users, administrators, and developers. Seamless user experience, coupled with stringent security protocols and performance expectations, necessitates effective communication and collaboration among various stakeholders. Managing these conflicting requirements is integral to the project's success.
P7: Interdependence	The components of the system are highly interdependent. Modifications or optimizations in one area can have ripple effects on other functionalities. For instance, altering security protocols may impact overall system performance. Thus, decision-making must be nuanced and consider the intricate interdependencies within the system.

1.4 Design Goals/Objectives

Here are some objectives for the Student Management System project:

- (a) Achieve low-latency communication for real-time chat.
- (b) Implement robust security measures to ensure data confidentiality and integrity.
- (c) Enable efficient scalability to handle a large number of simultaneous connections.
- (d) Provide an intuitive and responsive user interface for a seamless user experience.
- (e) Support private messaging and group chat functionalities.
- (f) Implement error handling mechanisms to ensure application stability.
- (g) Optimize resource utilization to enhance overall system performance.
- (h) Facilitate easy integration of additional features such as file sharing.

1.5 Application

From our this project, mass people will get so many facilities like:

- (a) Corporate Communication:
 - Facilitate private and group communication within organizations for efficient collaboration.

- (b) Educational Institutions:
 - Provide a platform for students and teachers to engage in discussions and share information in real-time.
- (c) Customer Support Systems:
 - Implement a live chat support system for businesses to interact with customers in real-time.
- (d) Social Networking:
 - Enable users to connect and communicate with friends, family, and colleagues on a real-time chat platform.
- (e) Collaborative Projects:
 - Support real-time communication among team members working on collaborative projects.
- (f) Telemedicine:
 - Facilitate real-time communication between healthcare professionals for remote consultations and collaboration.
- (g) Event Coordination:
 - Assist in planning and coordination for events by providing a real-time communication platform.
- (h) Community Building:
 - Create a space for like-minded individuals to connect and communicate within specific interest groups or communities.
- (i) Remote Work Collaboration:
 - Support remote teams by providing a real-time communication channel for work-related discussions and updates.

Chapter 2

Design/Development/Implementation of the Project

2.1 Introduction

In an era characterized by the ever-increasing demand for instantaneous communication, the development of a Multi-Client Chat Application using Socket Programming emerges as a pivotal solution, addressing the need for real-time interaction within networked environments. This project aims to harness the power of socket programming, employing a client-server model, to enable seamless and secure communication among multiple clients. By delving into the complexities of concurrent connections, scalability, and security protocols, this application seeks to provide a versatile platform for private and group messaging. As the digital landscape continues to prioritize swift and reliable communication, the Multi-Client Chat Application serves as an innovative response, offering a practical exploration of socket programming concepts and a tangible solution for diverse communication needs.

2.2 Project Details

2.2.1 Project Features

Here are some key features of the Multi-Client Chat Application:

1. Group Messaging:

- Enable multiple clients to participate in group conversations, fostering collaborative communication.

2. File Sharing:

- Facilitate the exchange of files between clients, supporting the seamless sharing of documents, images, and other file types.

3. Image Sharing:

- Allow users to share images within the chat, enhancing the visual communication

experience.

4. Emoji Support:

- Integrate a diverse range of emojis to enrich and personalize the messaging experience.

5. Private Messaging:

- Provide the capability for clients to engage in private, one-on-one conversations for discreet communication.

6. Real-Time Updates:

- Ensure that messages, file transfers, and other activities are updated in real-time for prompt user interaction.

7. User Authentication:

- Implement secure user authentication mechanisms to control access and protect user privacy.

8. Scalability:

- Design the application to handle a varying number of clients, ensuring scalability for different usage scenarios. [3]

2.3 Implementation

2.3.1 The workflow

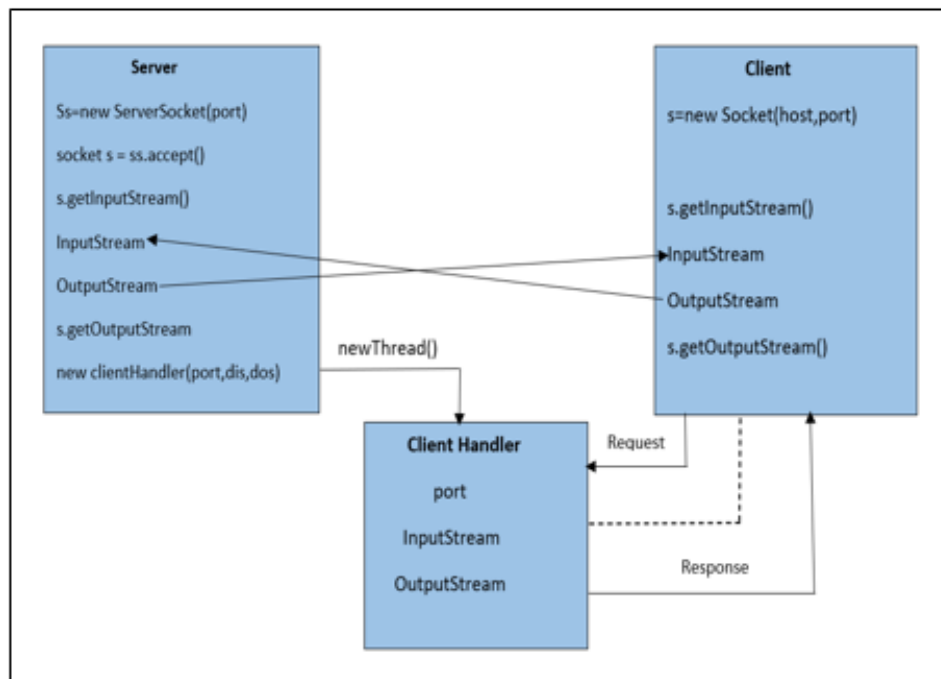


Figure 2.1: Architecture of Socket Programming in Java

2.3.2 Tools and libraries

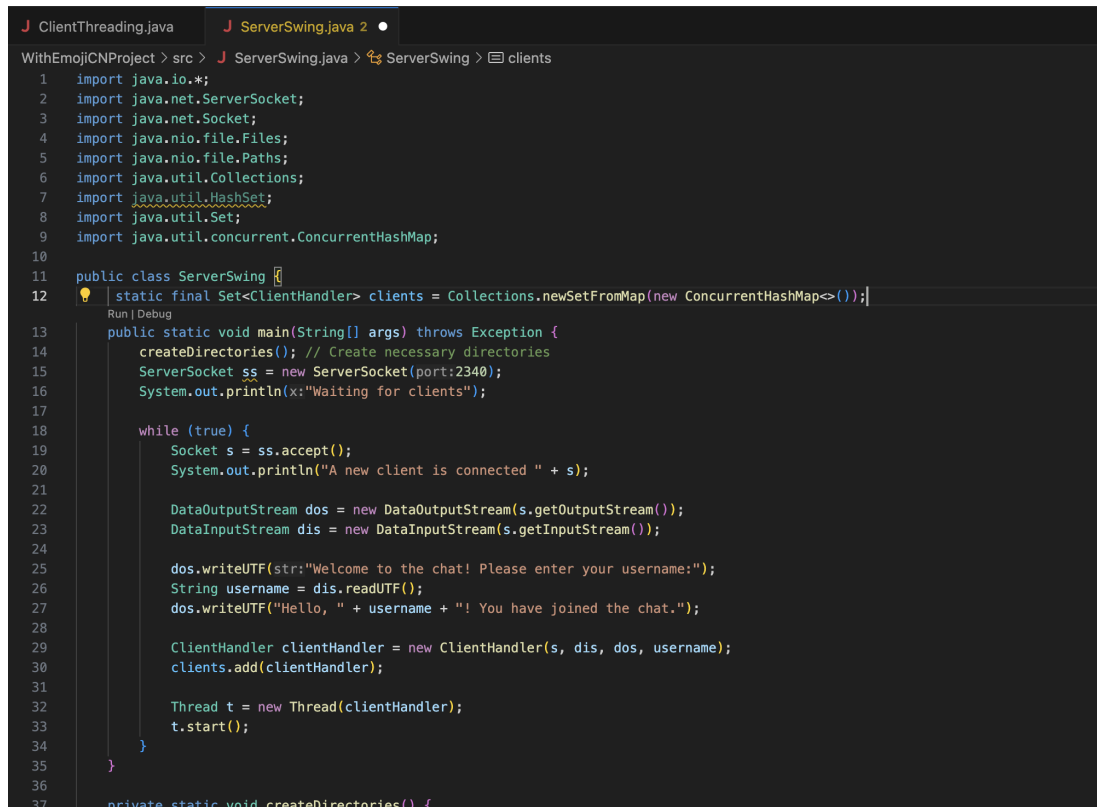
Hardware Requirement

1. Processor: Core i3 10th Gen
2. Ram: 8 GB
3. Storage: 128 GB
4. Monitor: 22 inch Color
5. Keyboard
6. Mouse

Software Requirement

1. Operating System: MacOS
2. IDE: Visual Studio Code
3. Programming Language: Java, Socket Programming

2.3.3 Some Screenshots of Programming Codes



```
J ClientThreading.java J ServerSwing.java 2
WithEmojiCNProject > src > J ServerSwing.java > ServerSwing > clients
1 import java.io.*;
2 import java.net.ServerSocket;
3 import java.net.Socket;
4 import java.nio.file.Files;
5 import java.nio.file.Paths;
6 import java.util.Collections;
7 import java.util.HashSet;
8 import java.util.Set;
9 import java.util.concurrent.ConcurrentHashMap;
10
11 public class ServerSwing {
12     static final Set<ClientHandler> clients = Collections.newSetFromMap(new ConcurrentHashMap<>());
13
14     public static void main(String[] args) throws Exception {
15         createDirectories(); // Create necessary directories
16         ServerSocket ss = new ServerSocket(port:2340);
17         System.out.println(x:"Waiting for clients");
18
19         while (true) {
20             Socket s = ss.accept();
21             System.out.println("A new client is connected " + s);
22
23             DataOutputStream dos = new DataOutputStream(s.getOutputStream());
24             DataInputStream dis = new DataInputStream(s.getInputStream());
25
26             dos.writeUTF(str:"Welcome to the chat! Please enter your username:");
27             String username = dis.readUTF();
28             dos.writeUTF("Hello, " + username + "! You have joined the chat.");
29
30             ClientHandler clientHandler = new ClientHandler(s, dis, dos, username);
31             clients.add(clientHandler);
32
33             Thread t = new Thread(clientHandler);
34             t.start();
35         }
36     }
37     private static void createDirectories() {
```

Figure 2.2: Server Socket

```

WithEmojiCNProject > src > J ServerSwing.java > ServerSwing > clients
82 class ClientHandler implements Runnable {
83     final Socket soc;
84     final DataInputStream input;
85     final DataOutputStream output;
86     final String username;
87
88     public ClientHandler(Socket s, DataInputStream dis, DataOutputStream dos, String username) {
89         this.soc = s;
90         this.input = dis;
91         this.output = dos;
92         this.username = username;
93     }
94
95     @Override
96     public void run() {
97         try {
98             while (true) {
99                 output.writeUTF(str:"What do you want? [Text/Emoji/File/Image/Private/GroupMessage/Exit]");
100                 String choice = input.readUTF();
101
102                 switch (choice) {
103                     case "Text":
104                         sendMessage();
105                         break;
106                     case "Emoji":
107                         sendEmoji();
108                         break;
109                     case "File":
110                         receiveFile();
111                         break;
112                     case "Image":
113                         receiveImage();
114                         break;
115                     case "Private":
116                         sendPrivateMessage();
117                         break;
118                     case "GroupMessage":
119                         sendGroupMessage();
120                         break;
121                     case "Exit":
122                         exitChat();
123                         break;
124                     default:
125                         output.writeUTF(str:"Invalid input");

```

Figure 2.3: Categories(Messaging Options)

```

private void sendMessage() throws IOException {
    output.writeUTF(str:"Enter your message:");
    String message = input.readUTF();
    ServerSwing.broadcastMessage(username + ": " + message, this);
}

private void sendEmoji() throws IOException {
    output.writeUTF(str:"Enter the emoji (e.g., 🍌):");
    String emoji = input.readUTF();
    ServerSwing.broadcastMessage(username + ": " + emoji, this);
}

private void sendPrivateMessage() throws IOException {
    output.writeUTF(str:"Enter the recipient username:");
    String recipient = input.readUTF();
    output.writeUTF(str:"Enter your private message:");
    String message = input.readUTF();
    ServerSwing.sendPrivateMessage(username + " (private): " + message, recipient, this);
}

private void sendGroupMessage() throws IOException {
    output.writeUTF(str:"Enter your group message:");
    String message = input.readUTF();
    ServerSwing.broadcastGroupMessage(username + " (group): " + message, this);
}

private void receiveFile() {
    try {
        String fileName = input.readUTF();
        long fileSize = input.readLong();

        byte[] fileBytes = new byte[(int) fileSize];
        input.readFully(fileBytes, off:0, fileBytes.length);

        // Save the received file to the server's file system
        String filePath = "server_received_files/" + fileName;
        Files.write(Paths.get(filePath), fileBytes);

        // Broadcast the information about the received file
        String broadcastMessage = "File received: " + fileName;
        ServerSwing.broadcastMessage(broadcastMessage, this);
    } catch (IOException e) {

```

Figure 2.4: Some Used Methods

```

private void connectToServer() {
    try {
        socket = new Socket(host:"localhost", port:2340);
        dis = new DataInputStream(socket.getInputStream());
        dos = new DataOutputStream(socket.getOutputStream());

        chatArea.append(str:"Connected to the server\n");

        // Read welcome message
        String welcomeMessage = dis.readUTF();
        updateChatArea(welcomeMessage);

        // Prompt for username
        username = JOptionPane.showInputDialog(message:"Enter your username:");
        dos.writeUTF(username);

        // Read authentication result
        String authResult = dis.readUTF();
        updateChatArea(authResult);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

Figure 2.5: Client Socket

Chapter 3

Performance Evaluation

3.1 Results Analysis/Testing

3.1.1 Initial Interface



Figure 3.1: After running the code of both server side and client side, the server will ask for a username

3.1.2 Add Client

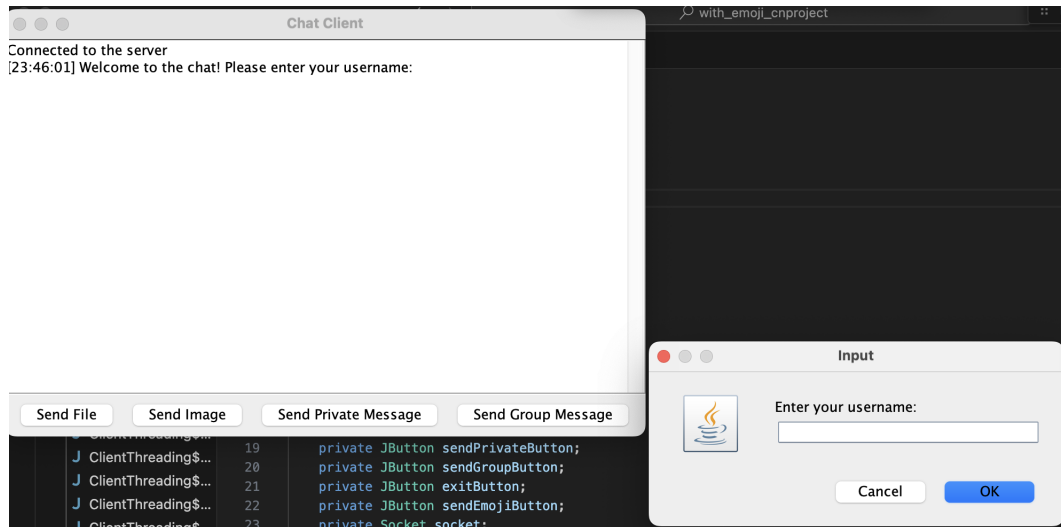


Figure 3.2: By entering a username, a client can be connected to the server and he can join in the chat

3.1.3 Client Join to the System

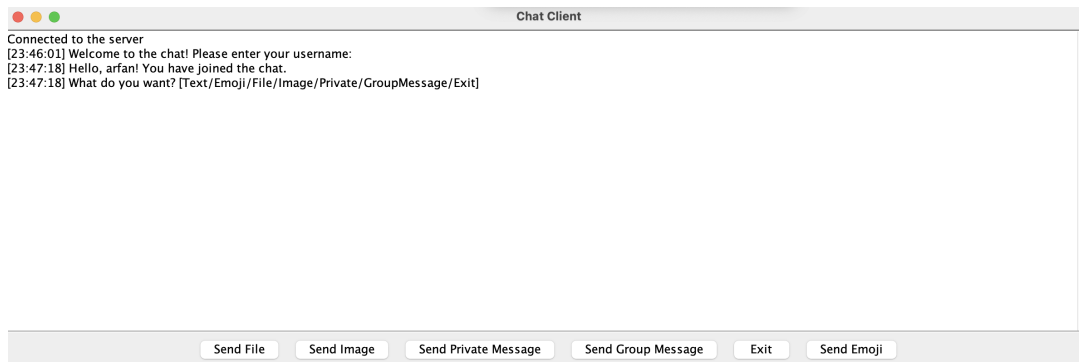


Figure 3.3: A client has joined to the system by using his username

3.1.4 Multiple Clients Join to the System

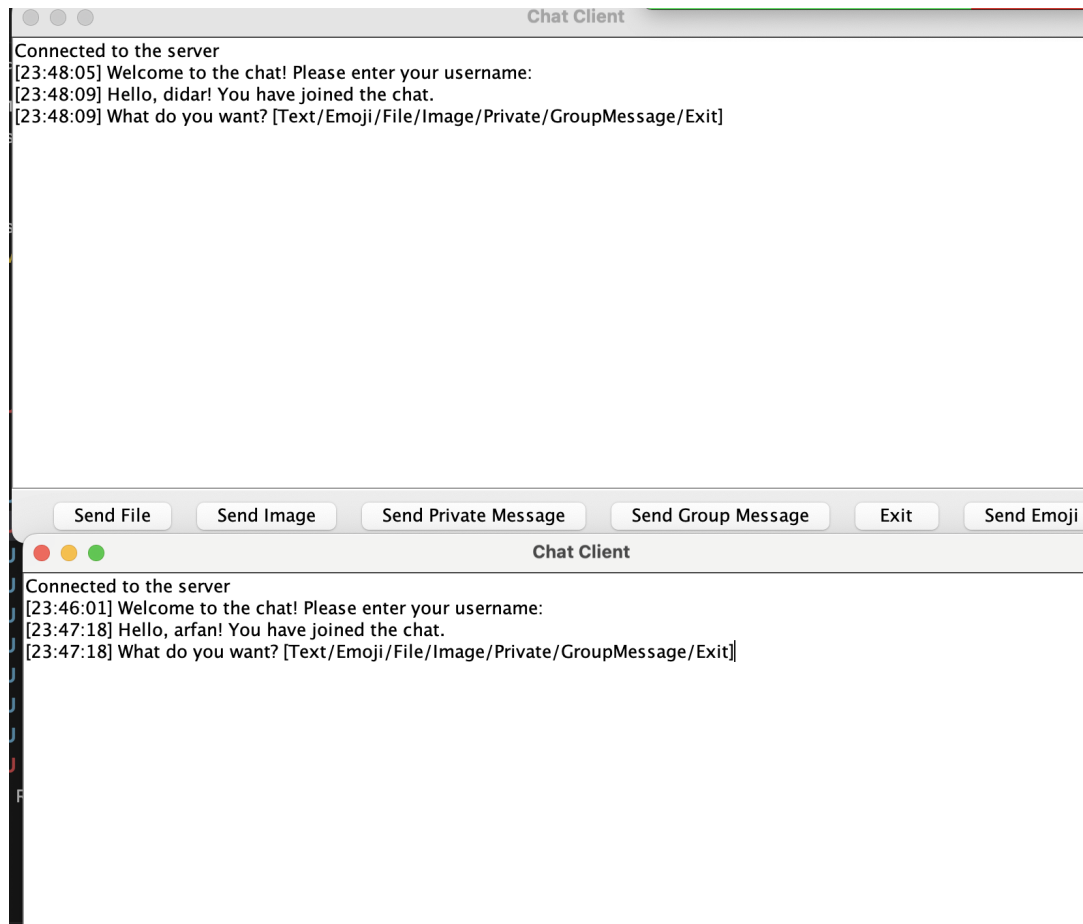


Figure 3.4: Multiple clients have joined to the system by using their username

3.1.5 Group and Private Messaging

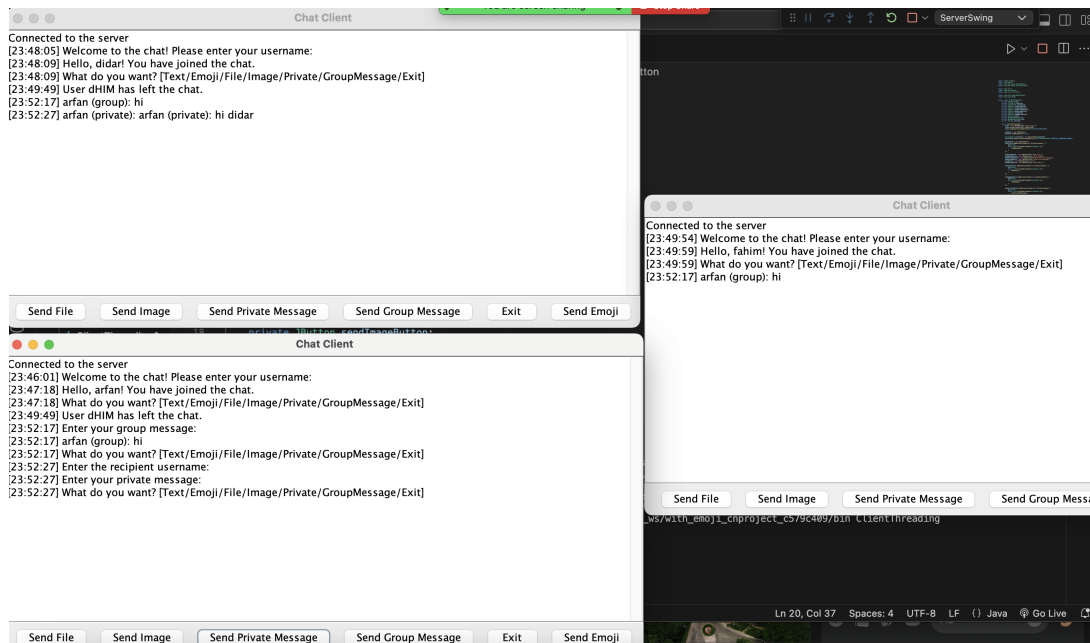


Figure 3.5: A client can send a group message which will be appeared to all the client's interface. Also he can send send a private message to an individual client

3.1.6 Send Image

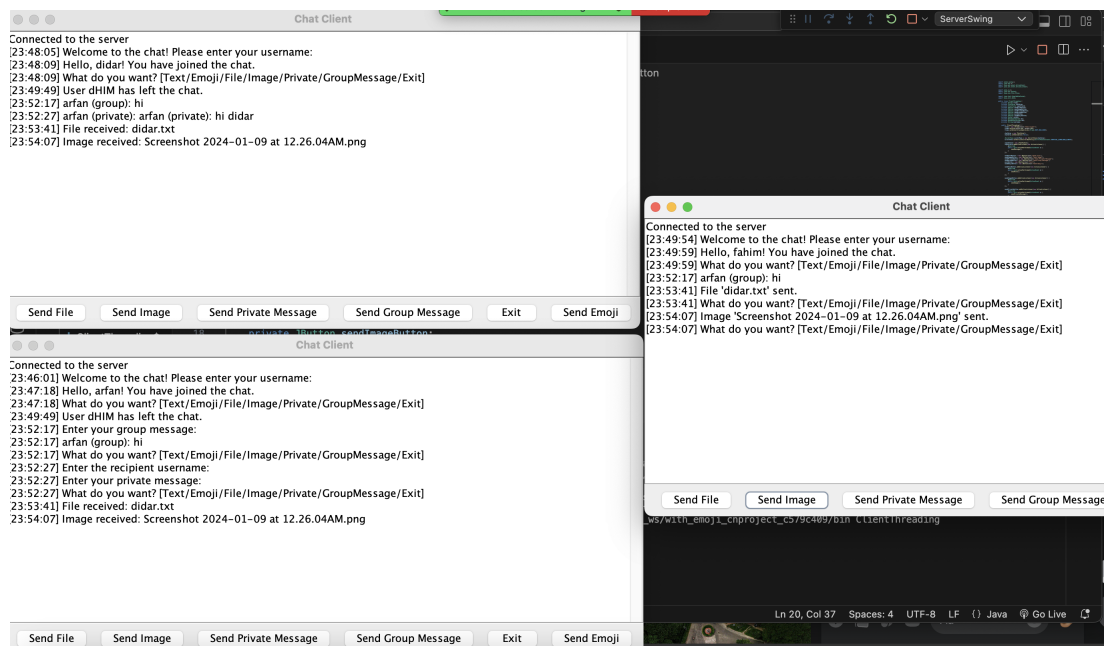


Figure 3.6: Clients can share images with others

3.1.7 Send Emojis and Left Chat Option

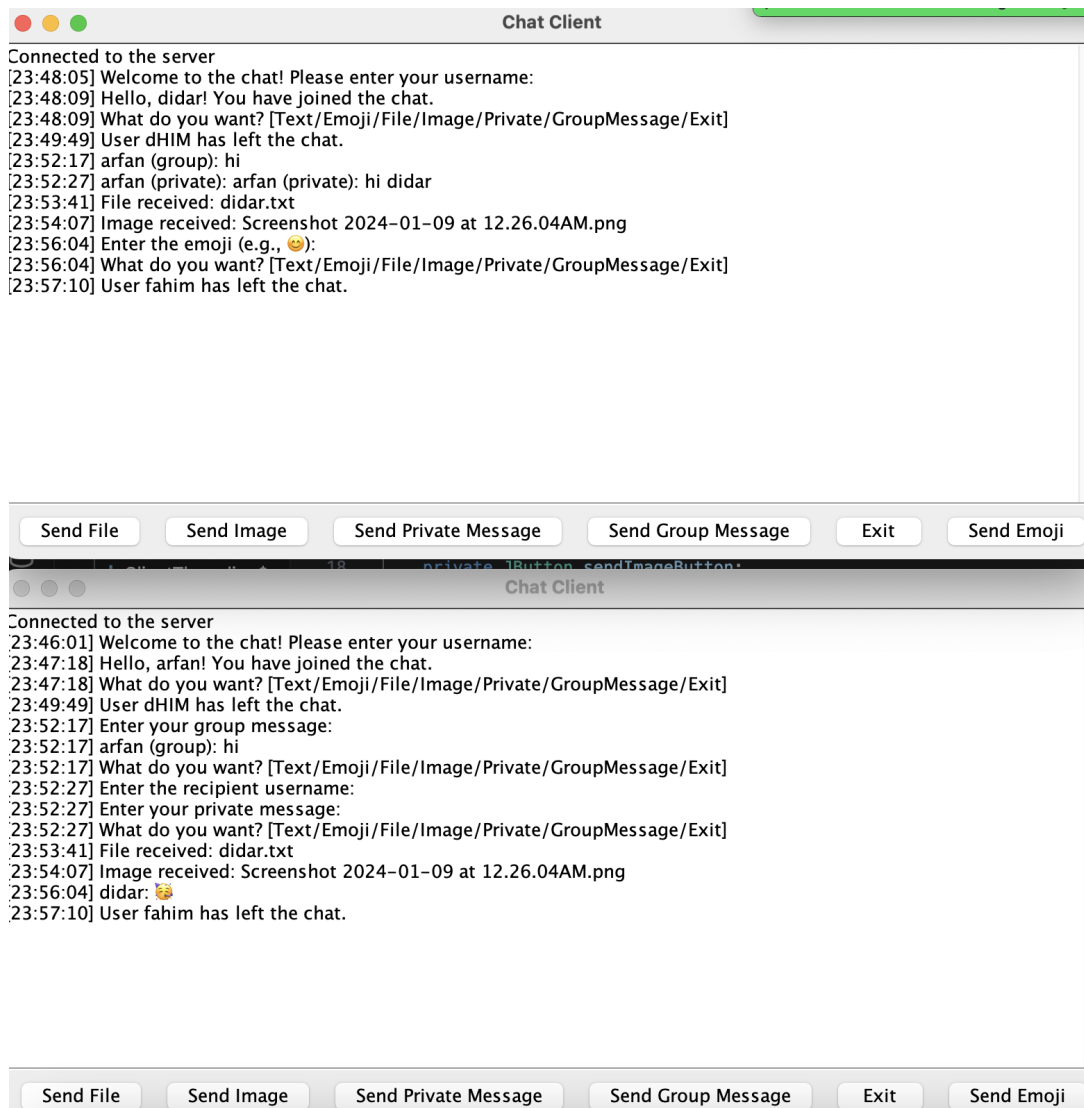


Figure 3.7: Clients can also send emojis and left the chat whenever he wants

3.2 Results Overall Discussion

The culmination of the development and testing phases of the Multi-Client Chat Application has yielded noteworthy results, underscoring its effectiveness and potential for real-world application. The project's objectives centered around providing a versatile and reliable platform for real-time communication among multiple clients, supporting group messaging, file sharing, image sharing, emoji usage, and private messaging. Here is an overview of the results and a comprehensive discussion:

1. Group Messaging and Collaboration:

- The implemented group messaging feature successfully enables multiple clients to engage in collaborative conversations, fostering efficient communication within a networked environment.

2. File and Image Sharing Capabilities:

- The application effectively facilitates the seamless exchange of files and images, meeting the project's goal of enhancing communication through multimedia content.

3. Emoji Integration:

- The diverse range of integrated emojis enhances the expressive nature of communication, contributing to a more engaging and personalized chat experience.

4. Private Messaging Functionality:

- The one-on-one private messaging feature provides clients with a discreet and secure communication channel, ensuring privacy when needed.

5. Real-Time Updates and Notifications:

- The real-time update mechanism and notification system prove robust, ensuring that users are promptly informed of new messages and activities, even when the application is running in the background.

6. Error Handling and Stability:

- Robust error-handling mechanisms ensure the stability and reliability of the application, particularly during critical operations such as file transfers, contributing to a seamless user experience.

Chapter 4

Conclusion

4.1 Discussion

In conclusion, the Multi-Client Chat Application has successfully achieved its design goals, offering a comprehensive and feature-rich platform for real-time communication. The project not only showcases the successful implementation of socket programming concepts but also presents a practical solution for diverse communication needs in both personal and professional settings. The application's success in providing a secure, scalable, and engaging communication environment positions it as a valuable tool for users seeking efficient and versatile chat functionality.

4.2 Limitations

While the Multi-Client Chat Application has proven to be a valuable addition to communication systems, certain limitations should be acknowledged:

1. **Limited Security Measures:** The project's security mechanisms are basic, and more advanced encryption and authentication methods could be implemented for enhanced data protection.
2. **User Interface Complexity:** The user interface may become less intuitive as additional features are added, potentially leading to a steeper learning curve for users.
3. **Limited Multimedia Support:** While the application supports file sharing and image sharing, support for additional multimedia content, such as videos or audio messages, is currently lacking.
4. **Dependency on Network Stability:** The application's performance is heavily reliant on network stability; fluctuations in connectivity may impact the user experience.
5. **Lack of Offline Messaging:** Currently, the application does not support offline messaging, hindering communication when users are temporarily disconnected.

4.3 Scope of Future Work

The Multi-Client Chat Application project lays a solid foundation for future enhancements and adaptations. The following areas present opportunities for further development and improvement:

1. **Enhanced Security Measures:** Implement advanced encryption algorithms and multifactor authentication to bolster the application's security and protect user data.
2. **User Interface Refinement:** Conduct user experience studies and refine the user interface to ensure that as new features are added, the application remains user-friendly and intuitive.
3. **Extended Multimedia Support:** Expand multimedia support to include additional content types such as videos, audio messages, and interactive media for a richer communication experience.
4. **Offline Messaging Capability:** Develop a mechanism for offline messaging to enable users to send and receive messages when temporarily disconnected, improving the application's flexibility.
5. **Integration of Voice and Video Calls:** Explore the integration of voice and video call functionalities to transform the application into a more comprehensive communication platform.
6. **Advanced Notification System:** Enhance the notification system to provide more granular control over notifications, allowing users to customize and prioritize alerts.
7. **Advanced User Management:** Implement features for advanced user management, such as user blocking, reporting, and additional privacy controls.
8. **Performance Optimization:** Continuously optimize the application's performance, focusing on resource utilization, minimizing latency, and ensuring a smooth user experience even with a large user base.
9. **Cross-Platform Enhancements:** Work on further improving cross-platform compatibility, ensuring seamless communication across a wider range of devices and operating systems.
10. **Integration with External Services:** Explore possibilities for integrating the chat application with external services, such as cloud storage for file sharing or third-party authentication providers for user management.

The scope of future works presents an exciting opportunity to evolve the Multi-Client Chat Application, ensuring its continued relevance, efficiency, and contribution to the advancement of communication sector. The pursuit of these avenues will contribute to a dynamic, adaptive, and technologically advanced Multi-Client Chat Application.

References

- [1] Thamer Ali T, Bahkit. The development of chat application using socket programming. *University Utara Malaysia*, 1:20–88, 2018.
- [2] Ishu Kumar Swathi Baswaraju, Abhishesk Kumar. Implementation of improved chat application between multiple clients. *ResearchGate*, 8(3):16–26, 2020.
- [3] Yaseen Al-Ani Muzhir Al-Ani, Rabah Noory. Client chat application design based on socket programming. *ResearchGate*, 3:9–56, 2021.