SOCIAL MEDIA CHAT SYSTEM

Work Report

CS 212 Object Oriented Programming Project

ACKNOWLEGEMENT

It is our privilege to express our heartfelt gratitude to our OOP Lab Project Co-ordinator, **Prof. Aparajita Datta**, for her valuable insights, continuous encouragement, and dedicated guidance throughout the development of our project titled **"Social Media Chat System"**. Her unwavering support and constructive feedback at every stage were instrumental in shaping our understanding and application of Object-Oriented Programming concepts.

Prof. Aparajita Datta provided us with a clear sense of direction, patiently guiding us through challenges and ensuring we stayed on track. Her expertise in programming and practical applications helped us bridge the gap between theory and implementation. Her detailed reviews and thoughtful suggestions motivated us to improve our work continuously and strive for excellence.

The project offered us a hands-on experience in applying object-oriented programming principles such as classes, objects, inheritance, polymorphism, abstraction, and encapsulation. With her guidance, we successfully integrated these concepts into the design and functionality of our system. Her insights helped us refine our code structure, adopt better design patterns, and ensure our system was scalable and user-friendly.

Beyond technical learning, this project also enhanced our skills in collaboration, time management, and critical thinking. Prof. Datta's regular feedback and mentorship ensured a smooth and productive workflow, while also encouraging innovation and independent thought.

We also take this opportunity to sincerely thank our friends and classmates for their support, whether it was through idea-sharing, feature testing, or moral encouragement.

This project has been a valuable part of our learning journey, offering practical exposure that will benefit us in future software development endeavors. We are deeply thankful to **Prof. Aparajita Datta** for her mentorship and for making this experience both educational and inspiring

Project Team:

Priya Das (2312106)
Abhay Sharma (2312131)

ABSTRACTION

This project presents the design and development of a multithreaded social media chat system, aimed at providing an efficient and scalable platform for real-time communication. The system enables seamless exchange of private messages and group messages between users, along with comprehensive logging mechanisms to maintain chat history and user interactions. The architecture of the application is based on a client-server model, ensuring structured communication and centralized control over data flow.

Developed in C++, the system leverages key features of object-oriented programming, incorporating classes, inheritance, encapsulation, and polymorphism to manage various components such as users, messages, and chat rooms. The application employs socket programming to facilitate network communication between the client and server, allowing multiple clients to connect and interact concurrently. To handle multiple user requests efficiently, the server utilizes multithreading, ensuring that each client is served independently without blocking the operations of others.

For persistent data storage, the system integrates with a MySQL database, which stores user credentials, chat records, and group information securely. This ensures data reliability, consistency, and the ability to retrieve historical conversations when needed. The use of SQL queries enables robust interaction with the database for user authentication, message retrieval, and data management.

The report details the system architecture, implementation methodology, and functional flow of the chat system. It also presents a critical evaluation of the system's performance, discussing key features such as concurrency handling, data storage efficiency, and user experience. Furthermore, the project outlines potential areas for future enhancement, such as the integration of file sharing, user presence indicators, end-to-end encryption, and a web-based or mobile interface.

Overall, this project demonstrates a practical implementation of core concepts in network programming, multithreading, and database management, forming a foundation for scalable real-time communication systems.

TABLE OF CONTENT

1) Problem statement

a) Background
b) Problem Description

2) Introduction

a) Overview of the project
b) Purpose
c) scope

3) Code structure

a) Object Oriented Programming concepts used
b) Class Description
c) Other Function

4) Flow chart

a) Flowchart

5) Algorithm for all activities

a) Creating the server
b) Creating the client

6) Working of code

a) Output images
b) Result
c) Discussion

7) References

a) References and GitHub

8) Conclusion

a) Conclusion

PROBLEM STATEMENT

Background:

In today's interconnected world, communication systems form the backbone of social interactions, whether for personal relationships, professional collaborations, or online communities. The rise of social media platforms has brought a demand for robust, scalable, and secure messaging systems that provide seamless real-time communication.

The **Social Media Chat System** project stems from this necessity, aiming to address key challenges of communication, such as real-time message delivery, efficient group messaging, and secure data storage. This project is also motivated by the need for an accessible platform that leverages cutting-edge technologies in networking, multithreading, and database management.

Problem Statement:

This project aims to develop a **Social Media Chat System** that addresses key communication challenges, including:

- Real-time private and group messaging capabilities to foster individual and collaborative interactions.
- Efficient concurrent handling of multiple users using multithreading to ensure seamless communication without delays.
- Persistent logging of chat history with sender and receiver details stored securely in a database for accountability and future reference.
- Scalable architecture to accommodate growing user bases while maintaining consistent performance.

INTRODUCTION

1.**Overview of the Project:** The Social Media Chat System is a real-time communication platform designed to facilitate seamless interaction among users through private messaging, group chats, and persistent message logging. Built using C++ and MySQL, the project leverages a client-server architecture with multithreading for concurrency management, ensuring that multiple users can engage simultaneously without compromising performance. Its key features include efficient private and group messaging, robust error handling, and secure chat logging in a relational database, making it both functional and reliable.

2.Purpose: The primary purpose of this project is to create a user-friendly and scalable chat system for real-time communication, addressing common challenges such as concurrent user management, secure data storage, and efficient group communication. It aims to provide a reliable solution that balances simplicity with functionality while also serving as a practical implementation of networking and database concepts. This project is designed not only to meet immediate communication needs but also to serve as a foundation for future enhancements like encryption and multimedia sharing.

3.Scope: The scope of this project encompasses the development of a robust communication system with core functionalities, including:

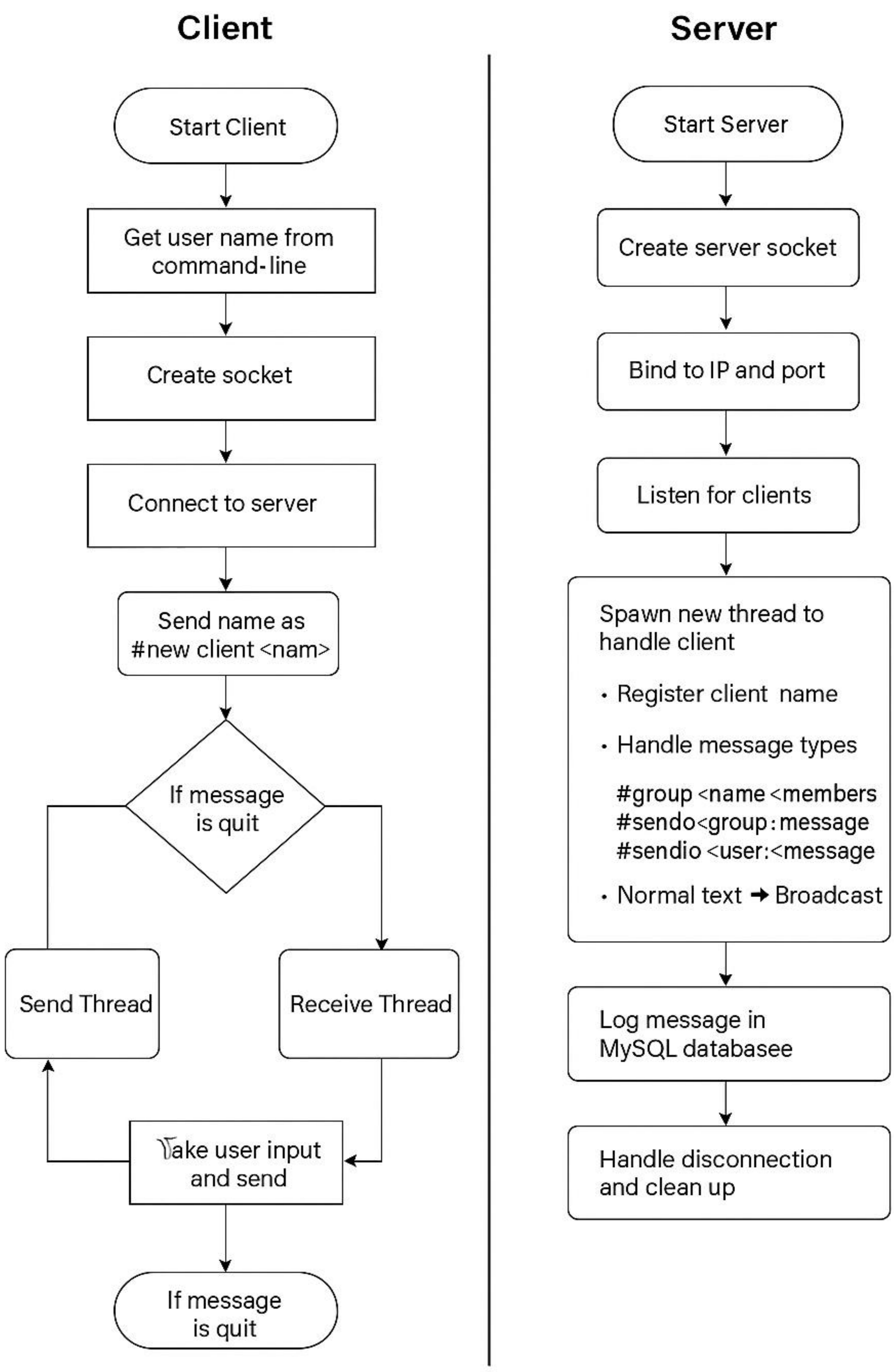
- **Private Messaging**: Secure and formatted communication between individual users.
- **Group Messaging**: Collective interactions among predefined groups of users.
- **Message Logging**: Persistent storage of chat history with sender and receiver details for accountability. Additionally, the system is designed to handle scalability through multithreading and manage error handling effectively, making it adaptable to larger user bases and diverse use cases. The modular architecture allows for future expansion into advanced features such as encryption, graphical user interfaces, and cross-platform compatibility.

CODE STRUCTURE:

Methodology The implementation of the social media chat system followed a structured approach:

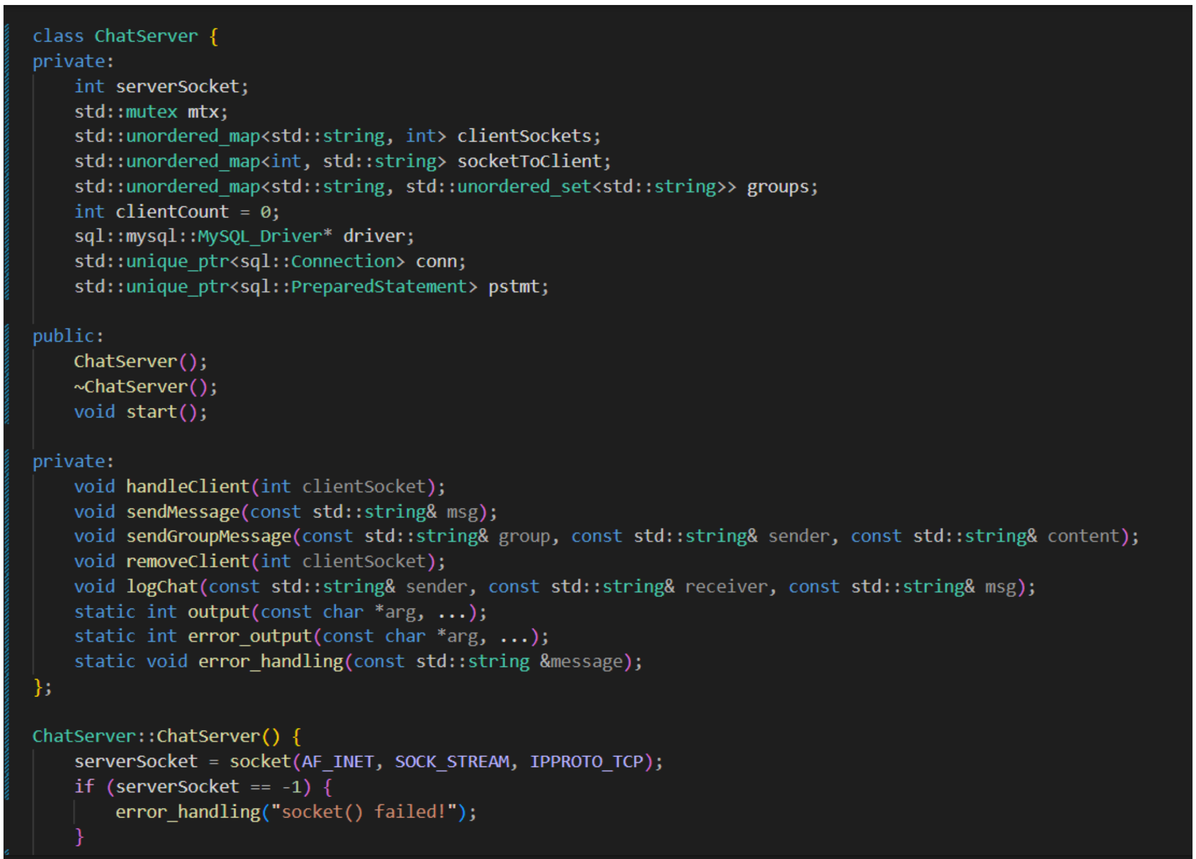
1. **Client-Server Architecture:**
   - A server listens for incoming connections and handles communication between multiple clients using multithreading for concurrency.
   - Clients connect to the server, exchange messages, and display them in real-time.
2. **Technologies Used:**
   - C++ for programming and socket API for network communication.
   - MySQL for chat log storage and management.
3. **Key Features:**
   - Private Messaging: Users can send messages directly to one another.
   - Group Messaging: Groups can be created with multiple members, allowing collective communication.
   - Message Logging: All messages are stored in a MySQL database with sender, receiver, and timestamp information for auditing.
4. **Error Handling and Validation:**
   - Comprehensive error-handling mechanisms ensure robust communication and data storage.

FLOW CHART

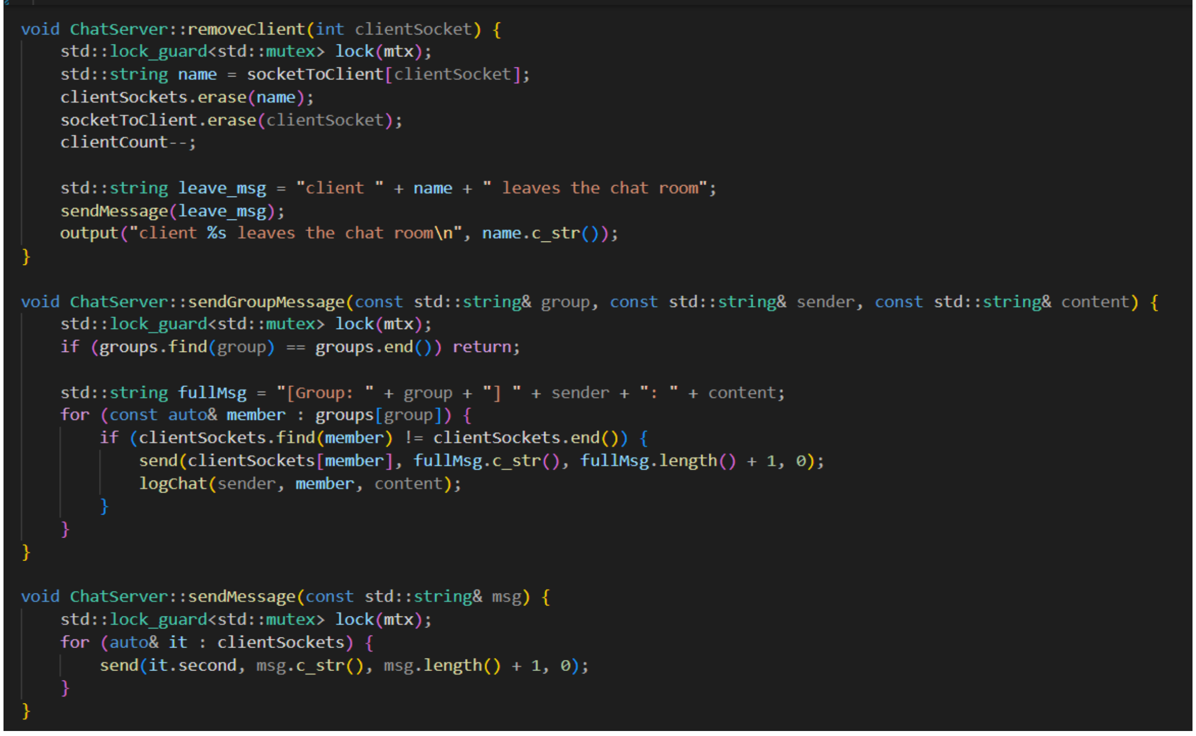


Client
Server
Start Client
Get user name from command-line
Create socket
Connect to server
Send name as #new client <nam>
If message is quit
Send Thread
Receive Thread
Take user input and send
If message is quit
Start Server
Create server socket
Bind to IP and port
Listen for clients
Spawn new thread to handle client
• Register client name
• Handle message types
#group <name <members
#sendo<group:message
#sendio <user:<message
• Normal text ➜ Broadcast
Log message in MySQL databasee
Handle disconnection and clean up

CREATING THE SERVER

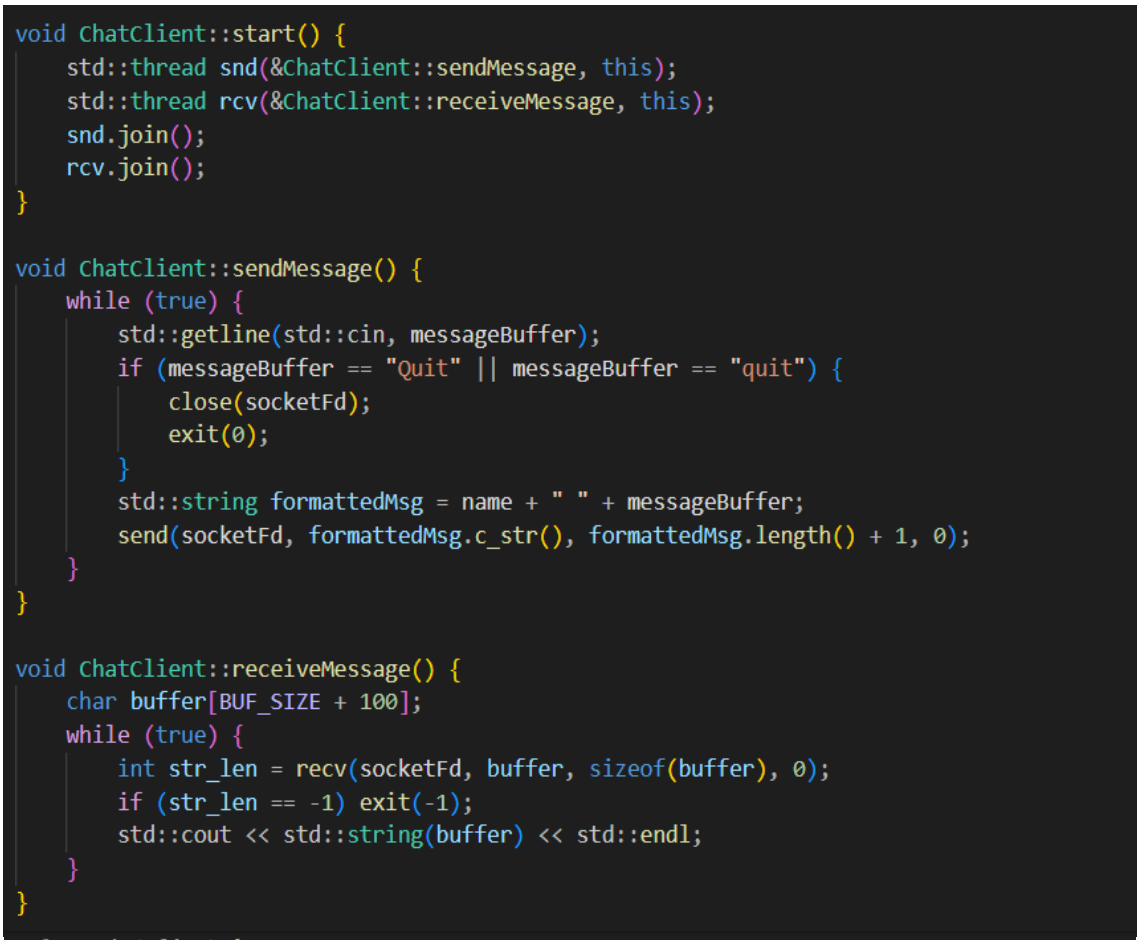


```
class ChatServer {
private:
    int serverSocket;
    std::mutex mtx;
    std::unordered_map<std::string, int> clientSockets;
    std::unordered_map<int, std::string> socketToClient;
    std::unordered_map<std::string, std::unordered_set<std::string>> groups;
    int clientCount = 0;
    sql::mysql::MySQL_Driver* driver;
    std::unique_ptr<sql::Connection> conn;
    std::unique_ptr<sql::PreparedStatement> pstmt;

public:
    ChatServer();
    ~ChatServer();
    void start();

private:
    void handleClient(int clientSocket);
    void sendMessage(const std::string& msg);
    void sendGroupMessage(const std::string& group, const std::string& sender, const std::string& content);
    void removeClient(int clientSocket);
    void logChat(const std::string& sender, const std::string& receiver, const std::string& msg);
    static int output(const char *arg, ...);
    static int error_output(const char *arg, ...);
    static void error_handling(const std::string &message);
};

ChatServer::ChatServer() {
    serverSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (serverSocket == -1) {
        error_handling("socket() failed!");
    }
```

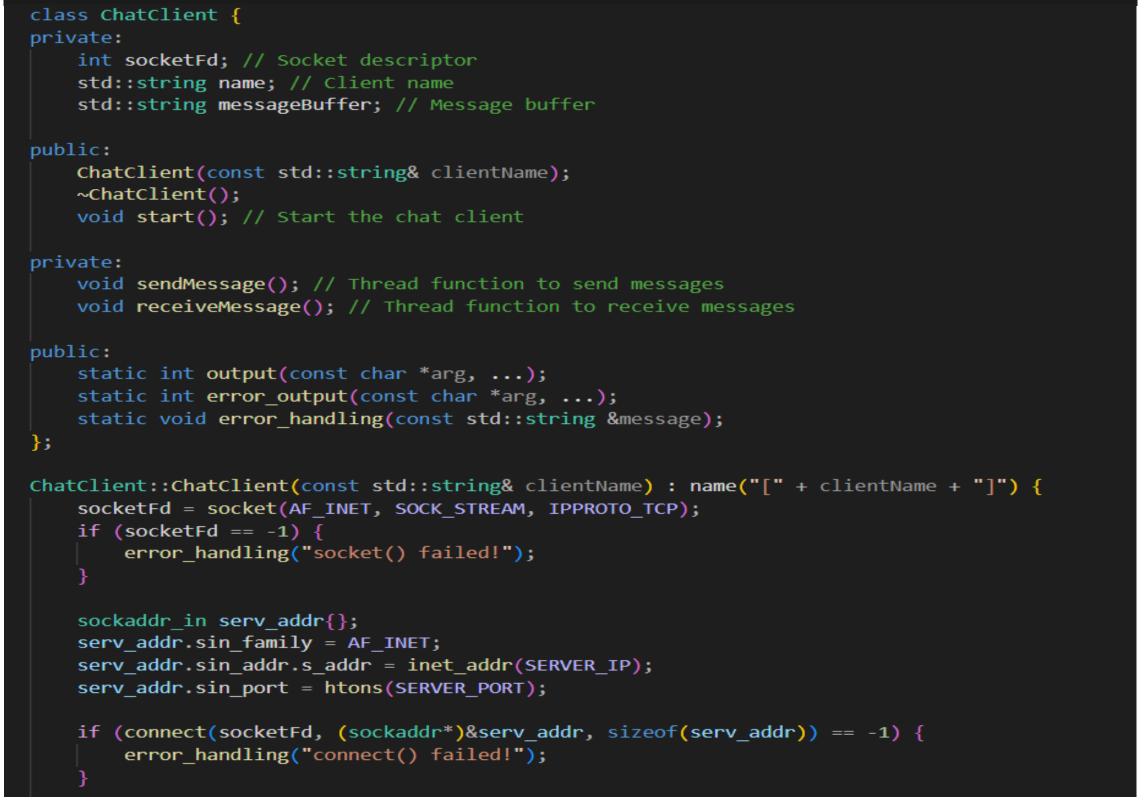


```
void ChatServer::removeClient(int clientSocket) {
    std::lock_guard<std::mutex> lock(mtx);
    std::string name = socketToClient[clientSocket];
    clientSockets.erase(name);
    socketToClient.erase(clientSocket);
    clientCount--;

    std::string leave_msg = "client " + name + " leaves the chat room";
    sendMessage(leave_msg);
    output("client %s leaves the chat room\n", name.c_str());
}

void ChatServer::sendGroupMessage(const std::string& group, const std::string& sender, const std::string& content) {
    std::lock_guard<std::mutex> lock(mtx);
    if (groups.find(group) == groups.end()) return;

    std::string fullMsg = "[Group: " + group + "] " + sender + ": " + content;
    for (const auto& member : groups[group]) {
        if (clientSockets.find(member) != clientSockets.end()) {
            send(clientSockets[member], fullMsg.c_str(), fullMsg.length() + 1, 0);
            logChat(sender, member, content);
        }
    }
}

void ChatServer::sendMessage(const std::string& msg) {
    std::lock_guard<std::mutex> lock(mtx);
    for (auto& it : clientSockets) {
        send(it.second, msg.c_str(), msg.length() + 1, 0);
    }
}
```

CREATING THE CLIENT SERVER



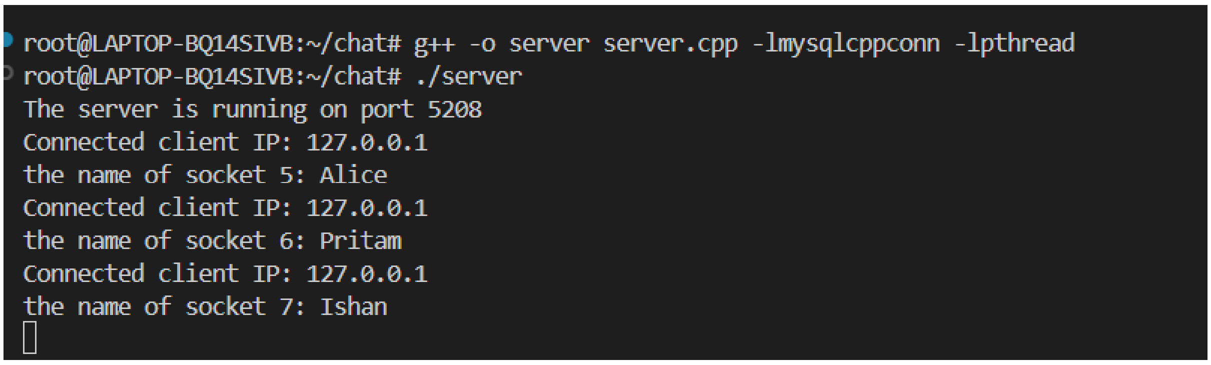
```cpp
void ChatClient::start() {
    std::thread snd(&ChatClient::sendMessage, this);
    std::thread rcv(&ChatClient::receiveMessage, this);
    snd.join();
    rcv.join();
}

void ChatClient::sendMessage() {
    while (true) {
        std::getline(std::cin, messageBuffer);
        if (messageBuffer == "Quit" || messageBuffer == "quit") {
            close(socketFd);
            exit(0);
        }
        std::string formattedMsg = name + " " + messageBuffer;
        send(socketFd, formattedMsg.c_str(), formattedMsg.length() + 1, 0);
    }
}

void ChatClient::receiveMessage() {
    char buffer[BUF_SIZE + 100];
    while (true) {
        int str_len = recv(socketFd, buffer, sizeof(buffer), 0);
        if (str_len == -1) exit(-1);
        std::cout << std::string(buffer) << std::endl;
    }
}
```



```cpp
class ChatClient {
private:
    int socketFd; // Socket descriptor
    std::string name; // Client name
    std::string messageBuffer; // Message buffer

public:
    ChatClient(const std::string& clientName);
    ~ChatClient();
    void start(); // Start the chat client

private:
    void sendMessage(); // Thread function to send messages
    void receiveMessage(); // Thread function to receive messages

public:
    static int output(const char *arg, ...);
    static int error_output(const char *arg, ...);
    static void error_handling(const std::string &message);
};

ChatClient::ChatClient(const std::string& clientName) : name("[" + clientName + "]") {
    socketFd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (socketFd == -1) {
        error_handling("socket() failed!");
    }

    sockaddr_in serv_addr{};
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr(SERVER_IP);
    serv_addr.sin_port = htons(SERVER_PORT);

    if (connect(socketFd, (sockaddr*)&serv_addr, sizeof(serv_addr)) == -1) {
        error_handling("connect() failed!");
    }
```

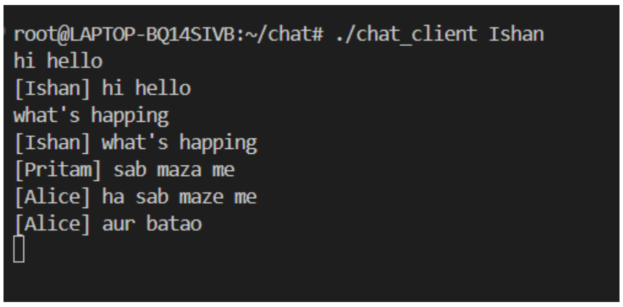
OUTPUT OF THE CODE

Server:

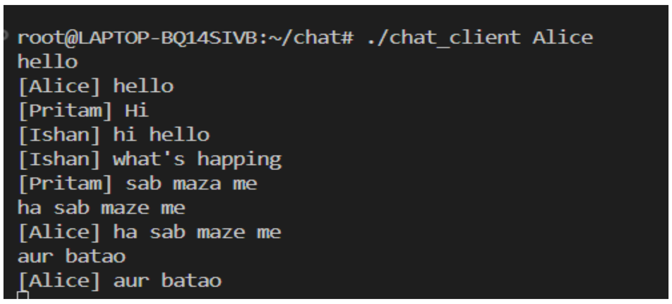


```
root@LAPTOP-BQ14SIVB:~/chat# g++ -o server server.cpp -lmysqlcppconn -lpthread
root@LAPTOP-BQ14SIVB:~/chat# ./server
The server is running on port 5208
Connected client IP: 127.0.0.1
the name of socket 5: Alice
Connected client IP: 127.0.0.1
the name of socket 6: Pritam
Connected client IP: 127.0.0.1
the name of socket 7: Ishan
```

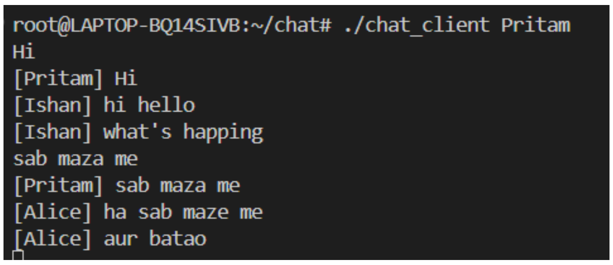
Chat Inbox:



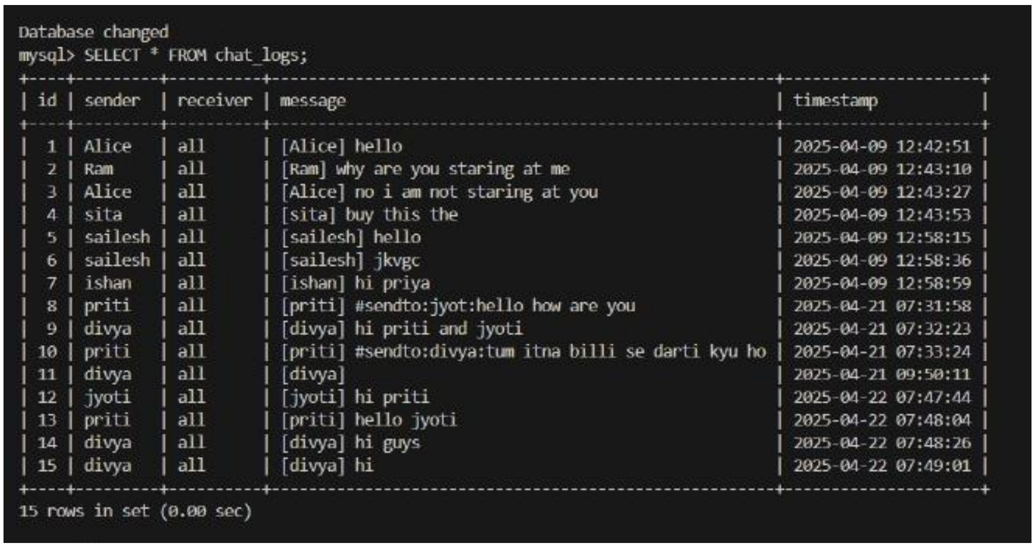
```
root@LAPTOP-BQ14SIVB:~/chat# ./chat_client Ishan
hi hello
[Ishan] hi hello
what's happing
[Ishan] what's happing
[Pritam] sab maza me
[Alice] ha sab maze me
[Alice] aur batao
```



```
root@LAPTOP-BQ14SIVB:~/chat# ./chat_client Alice
hello
[Alice] hello
[Pritam] Hi
[Ishan] hi hello
[Ishan] what's happing
[Pritam] sab maza me
ha sab maze me
[Alice] ha sab maze me
aur batao
[Alice] aur batao
```



```
root@LAPTOP-BQ14SIVB:~/chat# ./chat_client Pritam
Hi
[Pritam] Hi
[Ishan] hi hello
[Ishan] what's happing
sab maza me
[Pritam] sab maza me
[Alice] ha sab maze me
[Alice] aur batao
```



```
Database changed
mysql> SELECT * FROM chat_logs;
+----+---------+----------+-------------------------------------------------+---------------------+
| id | sender  | receiver | message                                         | timestamp           |
+----+---------+----------+-------------------------------------------------+---------------------+
|  1 | Alice   | all      | [Alice] hello                                   | 2025-04-09 12:42:51 |
|  2 | Ram     | all      | [Ram] why are you staring at me                 | 2025-04-09 12:43:10 |
|  3 | Alice   | all      | [Alice] no i am not staring at you              | 2025-04-09 12:43:27 |
|  4 | sita    | all      | [sita] buy this the                             | 2025-04-09 12:43:53 |
|  5 | sailesh | all      | [sailesh] hello                                 | 2025-04-09 12:58:15 |
|  6 | sailesh | all      | [sailesh] jkvgc                                 | 2025-04-09 12:58:36 |
|  7 | ishan   | all      | [ishan] hi priya                                | 2025-04-09 12:58:59 |
|  8 | priti   | all      | [priti] #sendto:jyot:hello how are you          | 2025-04-21 07:31:58 |
|  9 | divya   | all      | [divya] hi priti and jyoti                      | 2025-04-21 07:32:23 |
| 10 | priti   | all      | [priti] #sendto:divya:tum itna billi se darti kyu ho | 2025-04-21 07:33:24 |
| 11 | divya   | all      | [divya]                                         | 2025-04-21 09:50:11 |
| 12 | jyoti   | all      | [jyoti] hi priti                                | 2025-04-22 07:47:44 |
| 13 | priti   | all      | [priti] hello jyoti                             | 2025-04-22 07:48:04 |
| 14 | divya   | all      | [divya] hi guys                                 | 2025-04-22 07:48:26 |
| 15 | divya   | all      | [divya] hi                                      | 2025-04-22 07:49:01 |
+----+---------+----------+-------------------------------------------------+---------------------+
15 rows in set (0.00 sec)
```

RESULT:

Results The completed chat system demonstrates the following functionality:

1. Real-time private and group messaging with clear message formatting.
2. Concurrency achieved through multithreading, allowing multiple users to interact simultaneously.
3. Secure storage of chat history in a MySQL database for future retrieval.
4. Seamless addition of new clients and groups without affecting ongoing operations.
5. Effective handling of invalid inputs, client disconnections, and edge cases like duplicate usernames.

DISCUSSION:

Discussion The project highlights the significance of leveraging multithreading and database integration for building scalable communication platforms. While the system performs well under testing, areas for improvement include:

1. **Security Enhancements:** Incorporating encryption for message transmission to ensure data confidentiality.
2. **User Interface:** Developing a graphical interface for better usability compared to the current text-based console setup.
3. **Scalability:** Extending functionality to handle significantly larger client bases by optimizing resource usage and utilizing advanced database indexing.
4. **Cross-Platform Support:** Modifying the codebase to support platforms beyond desktop environments, such as mobile and web.

FUTURE ENHANCEMENTS:

Currently, the system supports real-time multithreaded chat between multiple clients effectively. However, it lacks strong security features, especially end-to-end encryption. This means messages could potentially be accessed or altered while in transit.

- **Implement End-to-End Encryption (E2EE):** Ensure message confidentiality and integrity by encrypting messages so they can't be accessed or altered while in transit.
- **Add User Authentication:** Secure the login process to ensure only authorized users can access the chat system.
- **Use TLS/SSL for Encrypted Socket Communication:** Encrypt all socket communications to prevent unauthorized access and protect data in transit.
- **Store Passwords and Sensitive Data Using Secure Hashing Techniques:** Ensure that sensitive user data, such as passwords, are stored securely using proper hashing methods.
- **Create a GUI for a Better Public Experience:** Develop a user-friendly graphical interface that enhances the user experience and makes the chat system more accessible to the public.

REFERENCE

- Object Oriented Programming in C++ (OOPs) – YouTube Playlist by **Coder Army**.

  Available at: **https://youtube.com/playlist?list=PLQEaRBV9gAFujcBWJhBT2XXsuMlIfETBy**
- The **TutorialsPoint C++ Socket Programming** tutorial provides a comprehensive guide to network communication in C++. It covers the creation and management of sockets for both client and server applications using TCP and UDP protocols.
- Titled **Socket Programming in C/C++**, is an excellent resource for learning socket programming. It covers the fundamentals of socket programming in both C and C++, providing practical examples and explanations to help you understand how to implement network communication between applications.
- **GitHub repository**: Used as a reference to understand the code structure, file organization, and formatting standards

CONCLUSION

The Social Media Chat System project successfully demonstrates the implementation of a real-time messaging platform that mimics the core functionality of modern chat applications. By integrating socket programming with MySQL database support, the system ensures efficient message exchange and persistent user data management. The project showcases the practical use of C++ for networking, multithreading for handling multiple clients, and database connectivity for user authentication and chat history.

Through this project, we gained hands-on experience in designing a client-server architecture, handling concurrent connections, managing user sessions, and structuring code for scalability. This chat system lays a strong foundation for future enhancements, such as media sharing, encryption, user profiles, and a graphical interface, which could further align it with real-world social media applications.

Overall, the project has been a valuable learning experience in both network programming and system design, bridging theoretical concepts with real-time application development.