USMAN INSTITUTE OF TECHNOLOGY UNIVERSITY

OBJECT-ORIENTED PROGRAMMING

FINAL LAB PROJECT REPORT

SPRING 2025, SEMESTER-II, CS

A REAL TIME CLIENT-SERVER CHAT APPLICATION

Made by:

Hisham Ahmed(24FA-013-CS)

Laiba Khan(24FA-053-CS)

Instructor: Shiza Hasan.

Abstract:

This project presents the development of a Java-based client-server chat application designed to facilitate real-time text communication between multiple users over a network. The system incorporates essential features such as user authentication, registration, message broadcasting, and a graphical user interface (GUI) built using Java Swing. A multithreaded server handles concurrent connections from clients, ensuring efficient communication without blocking operations. Each connected client is assigned a dedicated thread for message processing, enhancing performance and scalability. User credentials are managed securely using an SQLite database, allowing for persistent storage and retrieval during login and registration operations. Communication between clients and the server is established using Java sockets, with clearly defined message formats for different actions. The application ensures smooth interaction by continuously listening for incoming messages and dynamically updating the chat window. The modular design, separation of concerns, and use of core Java technologies make this project a practical demonstration of networking, multithreading, database integration, and GUI programming in Java. This system is suitable for educational purposes and can be extended with advanced features such as private messaging, encryption, or file sharing. Overall, the project exemplifies how Java can be used to create a functional and user-friendly networked chat system.

Table of Contents

Introduction:

In the modern digital age, real-time communication systems are essential for both personal and professional interactions. This project introduces a Java-based client-server chat application that enables multiple users to communicate seamlessly over a local or remote network. The primary goal is to develop a functional, user-friendly chat system that demonstrates key concepts in Java programming, including socket-based networking, multithreading, GUI development using Swing, and database integration with SQLite (sqlite-jdbc-3.49.1.0). We are implementing this project using NetBeans IDE.

The application has two main components: a multithreaded server and a Swing-based client. The server searches for incoming client connections and manages user sessions concurrently using individual threads. Clients can register or log in through a graphical interface, and user credentials are verified against an SQLite database to ensure secure access. Once authenticated, users enter a chat environment where messages are broadcast in real time to connected clients.

This system emphasizes modularity and code reusability, with clear separation between the user interface, networking logic, and database operations. This architecture facilitates future enhancements such as private messaging, secure encryption, message history, and more. Through this project, core Java skills are applied in a practical setting, making it an excellent educational tool and a solid foundation for more advanced communication platforms.

System Architecture:

This chat system is based on the client-server architecture, where the server acts as the central communication hub and each client communicates only with the server, not directly with each other. The server handles user authentication, message reception, and broadcasting. Each client connects to the server through sockets. The server listens for new client connections and manages communication using multithreading. SQLite is used as the backend to store and retrieve user credentials.

i. The server runs independently and is responsible for managing client connections, validating credentials, and forwarding messages to all connected users.
ii. The client provides a Swing-based GUI for users to register, log in, and chat in real-time.
iii. All communication happens over TCP sockets.

TCP sockets are a communication mechanism used in computer networks to enable reliable, two-way communication between devices over the Internet or a local network. They are based on the **Transmission Control Protocol (TCP)**, which ensures that data is delivered accurately and in the correct order. TCP sockets are the backbone of many network applications, including chat servers, file transfers, and web browsing, due to their reliability and robustness.

Component Breakdown:

Client Component:

i. It is responsible for user input, interface rendering, and sending/receiving messages.
ii. It communicates with the server over a persistent socket connection.
iii. It maintains a background thread for listening to server broadcasts.

Server Component:

i. It waits and searches for incoming connections using a Server Socket.
ii. It uses a thread pool or individual threads to manage each client.
iii. This receives messages from one client and broadcasts them to others.

Database Component (SQLite):

i. It contains a user table to store login credentials.
ii. It supports methods for validating login and inserting new user records.
iii. It communicates with the server via the DatabaseHelper class.

Detailed Design:

Database Design:

The database consists of a single table named users. It is implemented using SQLite due to its lightweight, serverless nature which integrates seamlessly with desktop applications. The schema is as follows:

i. **id**: A unique integer for each user.
ii. **username**: Must be unique to prevent duplicate registrations.
iii. **password**: Stored in plaintext or encrypted depending on implementation.

Class Design:

ChatServer.java:

i. It implements the server-side logic of the chat system.
ii. Each new client is handled in a separate thread via the ClientHandler inner class.
iii. Broadcasts messages from one client to all others.
iv. This manages client connection list and handles graceful disconnections.

ChatClient.java:

i. It provides the main GUI for the client using Java Swing.
ii. It allows the user to register or log in, then enter the chat window.
iii. It includes a message input area and a display for ongoing chat.
iv. This uses a background thread to receive messages from the server in real-time.

DatabaseHelper.java:

i. It abstracts all database-related operations.
ii. It provides methods such as registerUser (String username, String password) and validateUser (String username, String password).
iii. It ensures the separation of concerns by isolating database logic from GUI and server communication.

MainLauncher.java:

i. This acts as the main entry point of the application.
ii. It allows the user (developer) to choose whether to launch the server or the client interface.
iii. It simplifies deployment and testing by consolidating startup options in one location.

User Manual (How to Run):

1. Java JDK 17+ must be installed on both server and client machines.
2. SQLite JDBC Driver (e.g., sqlite-jdbc-3.36.0.3.jar).
3. Internet/Network connection (if running client and server on different machines).

Setup Instructions:

Server:

- Place files in Server.java, ClientHandler.java, DatabaseManager.java, etc.
- Ensure that all are in the same package or proper directory structure.
- Initialize SQLite database and run the server once to auto-generate chat.db, or manually create it and then compile the Server Code.

Client:

- Place files as ChatClient.java, LoginFrame.java, RegisterFrame.java, etc, and compile Client Code.
- Update Host/Port in ChatClient.java, set the correct server IP and port (e.g., localhost, port 12345).

Start the Server:

- The server will listen on port 12345 (or configured port).
- The console will log connections, login attempts, and messages.

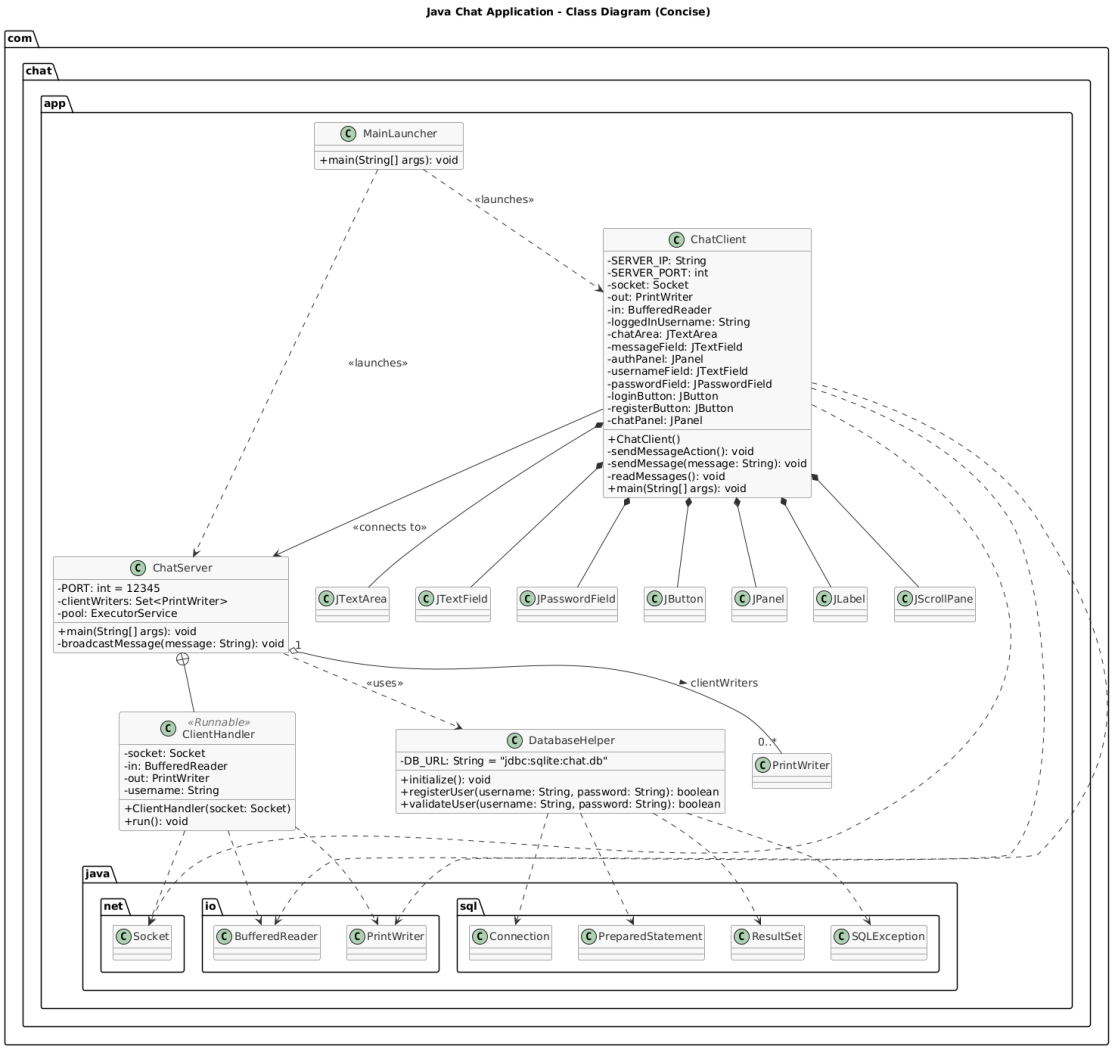
Start the Client:

- A Swing window opens with options to log in or register.
- Register if new to it, login if already resgistered.

Basic Usage Guide:

- Click Register on the login window.
- Enter a unique username and password.
- If successful, you're redirected to the login screen.
- In the main chat window: Enter recipient's username and your message.
- Click **Send** to deliver the message.
- If the recipient is online, they'll receive it immediately.
- Close the window to disconnect.
- The server removes you from the online user list.

UML Diagram:



Problems and Debugging:

During the development of the Java Chat Application, several common challenges were encountered, typical of client-server and database-integrated applications. Addressing these issues involved systematic debugging and applying best practices in Java network programming and GUI development.

Initial Compilation and Runtime Errors:

Early stages of development were marked by various compilation errors and ClassNotFoundException at runtime. This was due to javac indicating undeclared exception such as SQLException not caught or similar was displayed upon attempting to run JAR files. For SQLException, JBDC operations that could to throw such exceptions were identified (DatabaseHelper.initialize, registerUser, validateUser) and try-catch blocks were implemented on methods where appropriate and SQLException was declared in the method signature of remaining methods.

Client Connection Issues:

The ChatClient failed to connect to the server, displaying a "Connection refused: connect" error. There would be a GUI popup stating the error as well as sometimes the GUI would not fully start or close immediately after starting. It seemed that the server was connection correctly and it was listening on correct IP and port. We verified using the “netstat -ano” command in PowerShell. We also verified that the constant server IP we were using was the correct one. In the end, we created a launch_chat.bat file in which we gave server a 15 second head start, and after which the client would start. This solved the problem for us and the client now started correctly.

Database Driver Issue:

On some startup attempts, the server would report "No suitable driver found for jdbc:sqlite:chat.db". Server console gave the output: Server encountered an error during startup: No suitable driver found for jdbc:sqlite:chat.db. Confirmed the sqlite-jdbc-3.49.1.0.jar file was present in the correct directory. Verified that the JAR file name in the -cp (classpath) argument matched the actual file name exactly. Ensured the -cp argument correctly included the full path or relative path to the driver JAR. Double-checked and corrected the sqlite-jdbc-3.49.1.0.jar file name in the launch_chat.bat script's SET SQLITE_JDBC_JAR variable and the java -cp command. Ensured the driver JAR was physically present in the directory specified by the classpath.

Conclusion:

This Java client-server chat application demonstrates key concepts in network programming, multithreading, GUI development with Swing, and secure database handling using SQLite. The system supports user registration, login authentication, and real-time one-to-one messaging. By integrating these components effectively, the project offers a practical and functional chat solution. It also serves as a solid foundation for building more advanced communication tools.

Future Enhancements:

To extend functionality and improve user experience, several enhancements can be considered:

- **Group Chats:** Allow multiple users to communicate in shared chat rooms.
- **Message History:** Store and retrieve chat logs using the database.
- **File Sharing:** Support for sending files between users.
- **End-to-End Encryption:** Secure messages using AES or RSA.
- **Better UI/UX:** Include sound alerts, themes, and user status indicators.
- **Deployment:** Package the application with Docker or Java Web Start for easy distribution.

References:

- Oracle Java SE Documentation.
- SQLite JDBC Driver by Xerial.
- TutorialsPoint & GeeksforGeeks – for Java networking and GUI examples.
- Stack Overflow community – for troubleshooting during development
- Special thanks to faculty, peers, and mentors who provided feedback and support during this project