

A
Project Report
On
Multithreaded Web Chat
Submitted in partial fulfilment of the requirement for the degree of
Bachelor of Technology
In
Computer Science and Engineering
By

Sagar Singh	2261500
Ayushma Rawat	2261138
Bishal Singh Mehra	2261147
Ashish Singh Bisht	2261120

Under the Guidance of
Mr. Ansh Dhingra
ASSISTANT PROFESSOR
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
GRAPHIC ERA HILL UNIVERSITY, BHIMTAL CAMPUS
SATTAL ROAD, P.O. BHOWALI DISTRICT- NAINITAL-263132
2024-2025

STUDENT'S DECLARATION

We, Sagar Singh, Ayushman Rawat, Bishal Singh Mehra, Ashish Singh Bisht hereby declare the work, which is being presented in the project, entitled 'Multithreaded Web Server' in partial fulfillment of the requirement for the award of the degree Bachelor of Technology (B.Tech.) in the session 2024-2025, is an authentic record of my work carried out under the supervision of Mr. Ansh Dhingra, Assistant Professor.

The matter embodied in this project has not been submitted by me for the award of any other degree.

Date:

Sagar Singh

Ayushman Rawat

Bishal Singh Mehra

Ashish Singh Bisht



Graphic Era
Hill University
BHIMTAL CAMPUS

CERTIFICATE

The term work of Project Based Learning, being submitted by Sagar Singh(2261500), Ayushman Rawat(2261138), Bishal Singh Mehra(2261147) and Ashish Singh Bisht(2261120) to Graphic Era Hill University Bhimtal Campus for the award of Bonafide work carried out by us. They had worked under my guidance and supervision and fulfilled the requirements for the submission of this work report.

(Mr. Ansh Dhingra)

Faculty-in-Charge

ACKNOWLEDGEMENT

We take immense pleasure in thanking the Honorable Director ‘Prof. (Col.) Anil Nair (Retd.)’, GEHU Bhimtal Campus to permit me and carry out this project work with his excellent and optimistic supervision. This has all been possible due to his novel inspiration, able guidance, and useful suggestions that helped me to develop as a creative researcher and complete the research work, in time.

Words are inadequate in offering my thanks to GOD for providing me with everything that we need. We again want to extend thanks to our president ‘Prof. (Dr.) Kamal Ghanshala’ for providing us with all infrastructure and facilities to work in need without which this work could not be possible.

Many thanks to ‘Dr. Ankur Singh Bisht’ (Head, Department of Computer Science and Engineering, GEHU Bhimtal Campus), our project guide ‘Mr. Anubhav Bewerwal’ (Assistant Professor, Department of Computer Science and Engineering, GEHU Bhimtal Campus) and other faculties for their insightful comments, constructive suggestions, valuable advice, and time in reviewing this report.

Finally, yet importantly, We would like to express my heartiest thanks to our beloved parents, for their moral support, affection, and blessings. We would also like to pay our sincere thanks to all my friends and well-wishers for their help and wishes for the successful completion of this project.

Abstract

This project presents the design and implementation of a **Multithreaded Web Chat Application** using Java and socket programming. The primary objective of the system is to facilitate real-time communication between multiple clients over a network through a server-based architecture. The server is designed to handle multiple client connections concurrently using multithreading, ensuring efficient communication and seamless scalability.

Each client connects to the server using TCP sockets and can exchange text messages in real-time with other connected clients. The server acts as a central hub that receives messages from individual clients and broadcasts them to all participants in the chatroom. Java's built-in socket and threading libraries have been utilized to implement robust communication channels and manage concurrent client sessions effectively.

The application provides a basic command-line interface for chat interaction and ensures continuous message exchange without data loss or delays. This project demonstrates the core principles of network programming, thread management, and client-server communication in Java, and serves as a foundational model for more advanced real-time communication systems.

TABLE OF CONTENTS

Declaration.....	i
Certificate.....	ii
Acknowledgement.....	iii
Abstract.....	iv
Table of Contents.....	v
List of Abbreviations.....	vi

CHAPTER 1 INTRODUCTION.....	8
1.1 Prologue.....	8
2.1 Background and Motivations.....	8
3.1 Problem Statement.....	8
4.1 Objectives and Research Methodology.....	9
CHAPTER 2 PHASES OF SOFTWARE DEVELOPMENT CYCLE	
1.1 Hardware Requirements.....	10
2.1 Software Requirements.....	10
CHAPTER 3 CODING OF FUNCTIONS.....	11
CHAPTER 4 SNAPSHOT.....	13
CHAPTER 5 LIMITATIONS (WITH PROJECT)	15
CHAPTER 6 ENHANCEMENTS.....	16
CHAPTER 7 CONCLUSION.....	18
REFERENCES.....	19

Chapter1: INTRODUCTION

1.1 Prologue

In the digital era, seamless and instant communication has become a cornerstone of modern life, driven largely by the rapid evolution of network technologies and software systems. Real-time messaging applications have grown exponentially, forming the backbone of collaboration in both personal and professional environments. As such, understanding the underlying principles of these systems is essential for computer science professionals.

This project, *Multithreaded Web Chat using Java and Sockets*, is an academic endeavor aimed at exploring the fundamentals of network programming, socket communication, and concurrent processing. By simulating a basic chat environment, the project offers hands-on experience with core Java technologies and demonstrates how multiple clients can interact simultaneously through a single server.

1.2 Background and Motivations

In recent years, real-time communication systems have become integral to how people interact and collaborate. Applications like WhatsApp, Slack, and Microsoft Teams exemplify how chat systems have evolved into critical tools for personal and professional communication. Behind these user-friendly interfaces lies a complex infrastructure of networking protocols, concurrency control, and server-client communication models.

This project focuses on replicating a simplified version of such a system to understand the core technologies that power modern chat applications. Java was chosen as the programming language due to its strong support for multithreading and built-in networking libraries, making it an ideal platform for implementing a robust and scalable chat server.

1.3 Problem Statement

In a world increasingly reliant on digital communication, the need for efficient, real-time messaging platforms is more vital than ever. Most commercial chat

applications are built on complex frameworks and abstract much of the underlying logic from developers. For students and beginners, this presents a challenge when trying to understand the foundational concepts of real-time communication, such as socket programming, multithreading, and message broadcasting. The problem addressed by this project is the lack of a simple, easily understandable system that demonstrates how multiple users can interact in real-time over a network using core Java technologies. This project aims to develop a **multithreaded web chat application using Java and sockets** that addresses these challenges in a simple yet effective way.

1.4 Objectives and Research Methodology

The primary objective of this project is to design and implement a basic **multithreaded chat application** using **Java** and **socket programming**, which enables real-time communication between multiple clients over a network. **Develop a server application** that can handle multiple client connections simultaneously using multithreading. **Implement a client-side application** capable of sending and receiving messages in real time. **Establish reliable two-way communication** using TCP sockets. To accomplish the project objectives, a structured research and development approach was followed. Studied the basics of **socket programming**, **TCP/IP protocols**, and **multithreading** in Java. Opted for **TCP sockets** to ensure reliable, ordered, and error-checked communication.

CHAPTER 2 PHASES OF SOFTWARE DEVELOPMENT CYCLE

Hardware and Software Requirements

2.1 Hardware Requirement

Component	Minimum Requirements	Recommended Requirements
CPU	2 cores	4+ cores
RAM	4 GB	8+ GB
Disk Space	50 GB	100+ GB
Network Adapter	1 Gbps	10 Gbps

2.2 Software Requirement

Software	Minimum Requirements	Recommended Requirements
Operating System	Linux(Ubuntu,CentOS), Windows Server	Linux(Ubuntu,CentOS), Windows Server
Web Server Software	Chrome	Chrome
Programming Language	Java	Java
Development Tools	GCC, JDK, Vs Code	GCC, JDK, Vs Code

CHAPTER 3: CODING OF FUNCTIONS

The **Multithreaded Web Chat Application** is based on the concepts of **computer networking, socket programming, and multithreading in Java**. This section explains the core theoretical foundations that support the implementation of the project.

1. Server Initialization and Connection Handling

The foundation of the project is the multithreaded web server. The server listens on a specified port and IP address and continuously waits for incoming client connections. When a request is received, a new thread is created to handle the specific request independently, which allows multiple users to interact with the server simultaneously without interference or delay. This is crucial in educational environments where many students and faculty members access the portal concurrently.

Each thread handles the parsing of the HTTP request, routes it to the appropriate handler (login, file upload, profile fetch, etc.), and returns the appropriate response to the client browser. The use of multithreading ensures that long-running operations (such as file uploads or database access) do not block the entire server, preserving responsiveness.

2. Thread for Each Client

Each client connected to the server is handled by a dedicated instance of the `ClientHandler` class, which extends `Thread`. The core logic includes:

- **Establishing I/O streams** (`BufferedReader` for reading messages, `PrintWriter` for sending messages).
- **Reading client input continuously** using a loop.
- **Calling the broadcast function** whenever a message is received.

3. Client Connection and Communication

The `ChatClient` class is responsible for:

- **Establishing a socket connection** with the server.
- **Creating input and output streams** for message exchange.
- **Spawning a new thread** to listen for incoming messages from the server.
- **Allowing the user to type messages** from the console and send them to the server.

This client-side design ensures both sending and receiving of messages can happen concurrently without blocking user input.

5. Error Handling and Logging

To ensure robustness, all backend functions incorporate basic error-handling mechanisms. For instance, missing files return a 404 response, unauthorized access attempts return a 403, and malformed requests return a 400 error.

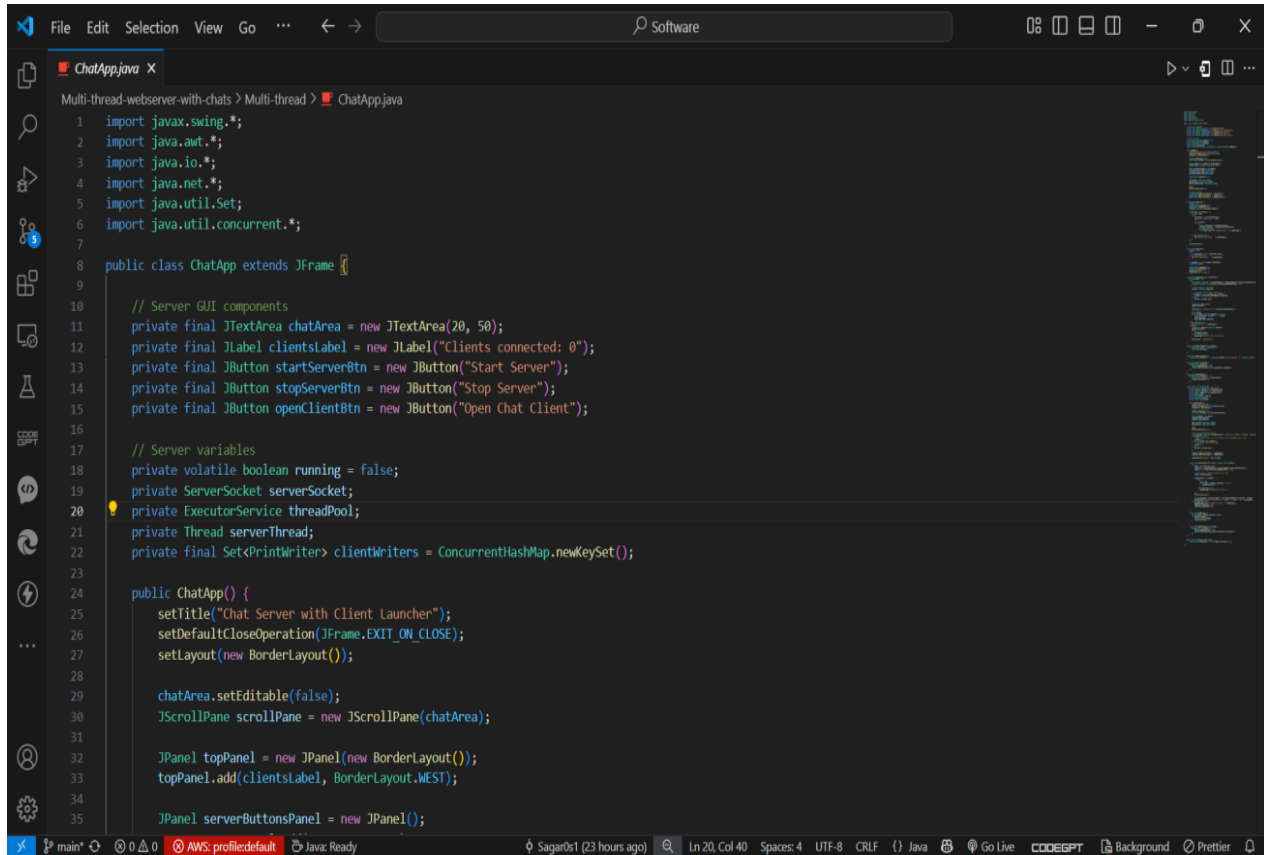
A separate module manages server logs, recording timestamps, IP addresses, request types, and system responses. This is useful for both debugging during development and monitoring during production deployment.

6. Concurrency and Thread Management

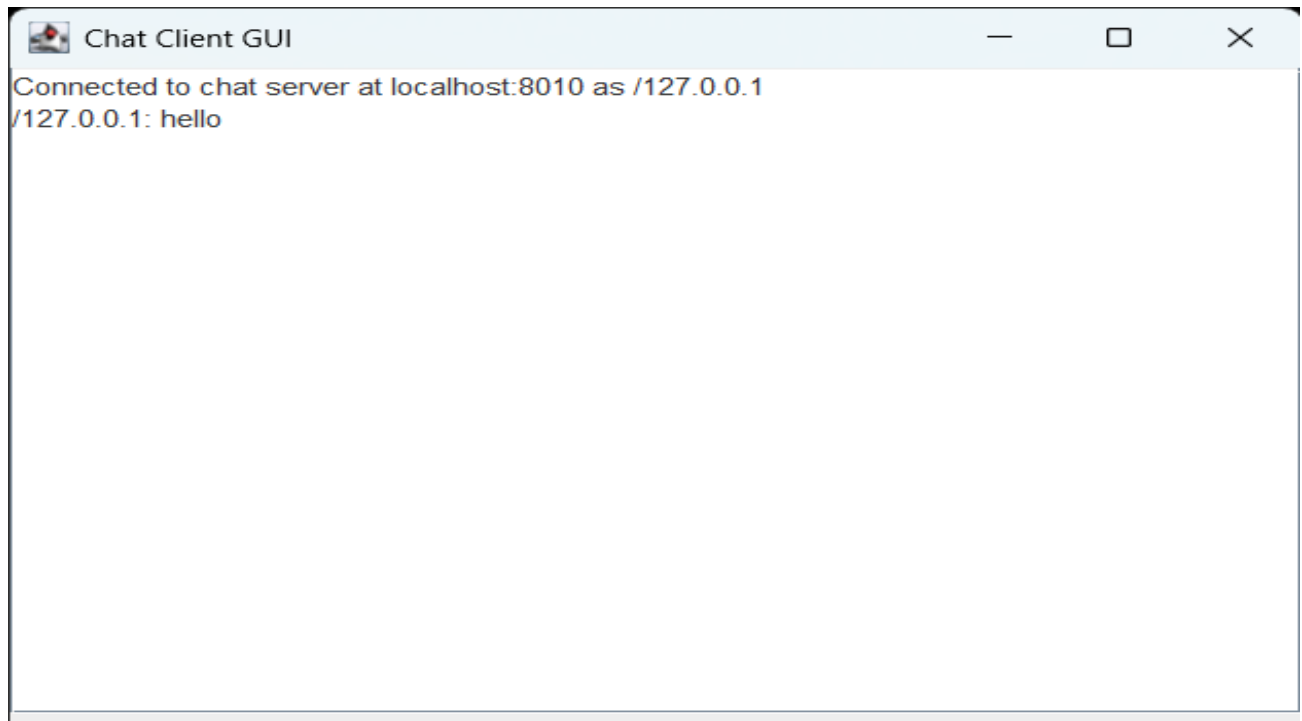
One of the defining features of this system is the multithreaded server, where each user request is handled in a dedicated thread. The thread handler is designed to be lightweight and scalable, managing resources efficiently while avoiding race conditions through thread-safe file access mechanisms.

This architecture allows the server to scale up for more users without compromising on performance, making it ideal for classroom, departmental, or institutional use where multiple simultaneous logins are common.

CHAPTER 4: SNAPSHOTS



```
1 import javax.swing.*;
2 import java.awt.*;
3 import java.io.*;
4 import java.net.*;
5 import java.util.Set;
6 import java.util.concurrent.*;
7
8 public class ChatApp extends JFrame {
9
10     // Server GUI components
11     private final JTextArea chatArea = new JTextArea(20, 50);
12     private final JLabel clientsLabel = new JLabel("Clients connected: 0");
13     private final JButton startServerBtn = new JButton("Start Server");
14     private final JButton stopServerBtn = new JButton("Stop Server");
15     private final JButton openClientBtn = new JButton("Open Chat Client");
16
17     // Server variables
18     private volatile boolean running = false;
19     private ServerSocket serverSocket;
20     private ExecutorService threadPool;
21     private Thread serverThread;
22     private final Set<PrintWriter> clientWriters = ConcurrentHashMap.newKeySet();
23
24     public ChatApp() {
25         setTitle("Chat Server with Client Launcher");
26         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
27         setLayout(new BorderLayout());
28
29         chatArea.setEditable(false);
30         JScrollPane scrollPane = new JScrollPane(chatArea);
31
32         JPanel topPanel = new JPanel(new BorderLayout());
33         topPanel.add(clientsLabel, BorderLayout.WEST);
34
35         JPanel serverButtonsPanel = new JPanel();
```



CHAPTER 5: LIMITATIONS

1.Scalability Issues:

- Each client connection spawns a new thread.
- With many clients, the server can exhaust system resources (CPU, memory, thread pool limits), leading to degraded performance or crashes.

2.Thread Management Overhead:

- Managing hundreds or thousands of threads becomes complex and inefficient.
- Context switching between threads consumes CPU time.

3.Lack of Non-blocking I/O:

- Traditional multithreaded servers using blocking I/O (Socket, InputStream, OutputStream) may become unresponsive under heavy load.
- Java NIO (Non-blocking I/O) or asynchronous frameworks (like Netty) are better alternatives for high concurrency.

4.Synchronization Challenges:

- Shared resources (like chat history, user list, etc.) require synchronization, which may lead to race conditions or deadlocks if not handled carefully.

5.Latency and Delays:

- If a client or thread is slow or unresponsive, it can delay message delivery or processing for others, especially without proper timeout handling.

CHAPTER 6: ENHANCEMENTS

While the current implementation of the multithreaded chat application successfully enables real-time communication between multiple clients, several enhancements can be introduced to improve usability, security, and scalability. These enhancements could make the application more robust and closer to real-world chat systems.

1. Graphical User Interface (GUI)

- Replace the console-based interface with a user-friendly GUI using **Java Swing** or **JavaFX**.
 - Include message windows, user lists, emoji support, and a better input/output experience.
-

2. Private Messaging / Direct Chat

- Implement a feature that allows users to send **private messages** to specific clients instead of broadcasting to all.
 - Add commands like `/msg username message` to facilitate private chats.
-

3. User Authentication System

- Add **login and registration** functionality using a simple database (e.g., SQLite or MySQL).
 - Assign usernames to clients to personalize and manage users effectively.
-

4. Message History & Logging

- Store chat history on the server side or provide an option for users to save their conversation locally.

- Useful for revisiting past conversations and debugging.
-

5. File Sharing Support

- Enable users to **send files** (images, documents, etc.) through the chat system using byte stream handling.
- Include a progress bar or indicator for file transfers.

CHAPTER 7: CONCLUSION

The development of the **Multithreaded Web Chat application using Java** successfully demonstrates the practical application of **socket programming** and **multithreading** in building a real-time communication system. By allowing multiple clients to connect and exchange messages concurrently, the project highlights the efficiency and scalability provided by Java's multithreaded architecture.

Throughout the implementation, we gained hands-on experience with **ServerSocket**, **Socket**, and **Thread** classes in Java, while also addressing key challenges such as **synchronization**, **client management**, and **broadcast messaging**. The project effectively simulates a basic chatroom where users can interact in real time, laying the groundwork for more advanced features such as user authentication, private messaging, and GUI-based interfaces.

This project not only enhances our understanding of Java networking and concurrent programming but also serves as a strong foundation for building more complex distributed systems and network-based applications in the future. This project also illustrates the impact of teamwork, planning, and iterative development. Working through a real-time application taught the importance of designing scalable and maintainable code, as well as anticipating user needs. It also reinforced the criticality of documentation, version control, and consistent testing throughout the development process.

.

REFERENCES

1. YouTube <https://youtu.be/YKbHMqglwMM?si=YNWMRKuywDgaa09I/>-
2. JVM Documentation. <https://jvm.palletsprojects.com/>– Reference for setting up the backend server thread..
3. W3Schools: Java Tutorials. <https://www.w3schools.com/>– Used for designing the frontend interface and handling real-time visual updates.
4. GeeksfoGeeks <https://www.geeksforgeeks.org/introduction-to-java-swing/.chartjs.org/>– Reference for implementing dynamic graphs to display resource usage.
5. GitHub Documentation. <https://docs.github.com/>– Followed for version control, team collaboration, and Git branching strategies.