A SYNOPSIS ON

Multi-Threaded Chat Application

Submitted in partial fulfilment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

In

Computer Science & Engineering

Submitted by:

Pankaj Joshi	**2261407**
Harshit Pandey	**2261253**
Jatin Singh Mehra	**2261281**
Gaurav Chandra Karnatak	**2261215**

Under the Guidance of

Mr. Ansh Dhingra

Lecturer

Project Team ID: 64

Department of Computer Science & Engineering

Graphic Era Hill University, Bhimtal, Uttarakhand

March-2025

Graphic Era Hill University
BHIMTAL CAMPUS

CANDIDATE'S DECLARATION

We hereby certify that the work which is being presented in the Synopsis entitled **"Multithreaded chat app"** in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science & Engineering of the **Graphic Era Hill University, Bhimtal Campus** and shall be carried out by the undersigned under the supervision of **Mr. Ansh Dhingra, Lecturer**, Department of Computer Science & Engineering, Graphic Era Hill University, Bhimtal.

Pankaj Joshi	**2261407**
Harshit Pandey	**2261253**
Jatin Singh Mehra	**2261281**
Gaurav Chandra Karnatak	**2261215**

The above mentioned students shall be working under the supervision of the undersigned on the **"Multithreaded Chat App".**

Signature

Supervisor

Signature

Head of the Department

Internal Evaluation (By DPRC Committee)

Status of the Synopsis: Accepted / Rejected

Any Comments:

Name of the Committee Members: **Signature with Date**

1.

2.

Table of Contents

Chapter No.	Description	Page No.
Chapter 1	Introduction and Problem Statement	4
Chapter 2	Background/ Literature Survey	6
Chapter 3	Objectives	8
Chapter 4	Hardware and Software Requirements	9
Chapter 5	Possible Approach/ Algorithms	10
	References	11

Chapter - 1

Introduction and Problem Statement

In the following section, a brief introduction and the problem statement for the work has been introduced,

1. Introduction

In today's digital world, **real-time communication** is essential for businesses, education, and personal interactions. Chat applications provide an efficient way for users to communicate instantly, but many existing systems suffer from **high latency, poor concurrency management, and security vulnerabilities**. A **Multi-Threaded Chat Application (MTCA)** addresses these challenges by utilizing **multithreading, socket programming, and Interprocess communication (IPC)** to create a **fast, scalable, and secure** messaging system.

This chat application ensures seamless interaction between multiple users by enabling concurrent connections without performance bottlenecks. The **multi-threaded approach** allows the server to handle multiple clients simultaneously, ensuring efficient message exchange. The application can be implemented using either:

1. **Client-Server Model:** A central server manages all client connections, receiving messages and forwarding them to other users. This approach is commonly used in platforms like **WhatsApp Web and Slack**.
2. **Peer-to-Peer (P2P) Model:** Clients communicate directly without relying on a central server, as seen in **Tox Messenger**. This model enhances privacy and reduces server dependency but requires more complex message routing.

To enhance real-time responsiveness, **Web Sockets** maintain a persistent connection between clients and the server, minimizing delays. Additionally, **encryption techniques (AES/RSA)** secure messages, preventing unauthorized access. **Concurrency control mechanisms** such as mutex locks and semaphores ensure smooth operation without conflicts.

The backend can be built using **Node.js (Express.js)** while the frontend can be developed using **HTML, CSS, JavaScript, or React.js**. Additional features like **user authentication, multimedia sharing, and message storage** can further enhance the application.

By addressing the limitations of existing systems, this project aims to develop a **reliable, scalable, and secure** chat platform capable of handling multiple users efficiently in diverse communication scenarios.

2. Problem Statement

Effective real-time communication is crucial for businesses, education, and social interactions. However, existing chat applications often face **scalability issues, inefficient message handling, and security vulnerabilities**. These challenges lead to **delays in message delivery, connection failures, and system crashes**, making them unreliable for large-scale communication. A **Multi-Threaded Chat Application (MTCA)** addresses these concerns by

implementing **multithreading, socket programming, and Interprocess communication (IPC)** to enhance efficiency and reliability.

One of the primary problems in traditional chat applications is **poor concurrency management**. When multiple users send messages simultaneously, servers without proper threading mechanisms may experience delays or drop messages. A **multi-threaded** approach ensures that the server can handle multiple clients at once, preventing bottlenecks and improving response times.

Additionally, **message security** is a major concern. Many chat applications **lack encryption**, making them vulnerable to cyber threats like **eavesdropping and data breaches**. Implementing **end-to-end encryption (AES/RSA)** ensures that messages remain private and protected from unauthorized access.

Another challenge is **real-time message delivery**. Traditional HTTP-based chat systems require frequent polling, which increases server load and latency. **Web Sockets** resolve this by maintaining a persistent connection, enabling instant message transmission with minimal delay.

There are two key architectures to consider:

1. **Client-Server Model:** A central server manages all user connections, efficiently routing messages (e.g., **WhatsApp Web, Slack**).
2. **Peer-to-Peer (P2P) Model:** Clients communicate directly, reducing reliance on a server (**Tox Messenger**), but requiring complex connection handling.

By integrating **thread synchronization, encryption, and real-time messaging**, this project aims to develop a **secure, scalable, and high-performance chat application** that overcomes the limitations of existing systems, providing a **seamless user experience**.

Chapter – 2

Background/ Literature Survey

Several real-time chat applications have been developed using **multithreading, socket programming, and concurrency management**, allowing multiple users to communicate efficiently. These applications rely on **client-server or peer-to-peer models**, utilizing **OS concepts like process scheduling, Interprocess communication (IPC), and concurrency control** to handle multiple concurrent users.

Examples of some early developed applications :

1. **Internet Relay Chat (IRC)**
    - One of the earliest chat applications, based on a **client-server model**.
    - Utilized **TCP sockets** for communication and **multithreading** to manage multiple users.
    - Messages were broadcasted through channels, similar to modern group chats.
2. **WhatsApp (Early Implementation)**
    - Initially followed a **centralized client-server model** with messages relayed through a server.
    - Implemented **multithreading and concurrency control** for real-time messaging.
    - Later incorporated **end-to-end encryption (E2EE)** and optimized peer-to-peer synchronization.
3. **Facebook Messenger**
    - Originally relied on **socket programming and multithreading** to manage concurrent user requests.
    - Used **XMPP (Extensible Messaging and Presence Protocol)** for structured message exchange.
    - Migrated to **Web Sockets** for more efficient and scalable real-time messaging.
4. **Telegram**
    - Uses a **multi-threaded message queue system** for fast and efficient message delivery.
    - Employs **client-server architecture** with **distributed processing** for high scalability.

- Implements **AES encryption** for secure communication while using **asynchronous synchronization** across multiple devices.

A review of existing literature and real-world implementations has highlighted several important aspects of multi-threaded chat applications:

- **Multithreading and Concurrency Control** are essential for efficiently handling multiple client connections.
- **Web Sockets** are commonly used for real-time message transmission with minimal latency.
- **Security Measures**, such as **AES encryption and TLS protocols**, are necessary for protecting data.
- **Scalability and Load Balancing** techniques ensure that the system can handle large numbers of concurrent users.
- **Event-Driven and Asynchronous Programming** (such as Node.js) provides better performance over traditional threading in high-load environments.

Chapter - 3

Objectives

The objectives of building a **Multi-Threaded Chat Application (MTCA)** are as follows:

1. **Enable Real-Time Communication:**
   The application ensures instant message exchange between multiple users using socket programming and multithreading, providing a seamless chat experience.

2. **Support Concurrent Connections:**
   By utilizing multithreading and concurrency control, the system efficiently handles multiple users simultaneously without performance degradation, ensuring smooth scalability.

3. **Efficient Resource Management:**
   The application will optimize CPU, memory, and bandwidth usage by implementing techniques like thread pooling, caching, and efficient data structures, ensuring smooth performance even under heavy load.

4. **Implement a Robust Client-Server Model:**
   The server manages all client connections, message routing, and broadcasting while maintaining stable and secure communication between users.

5. **Provide a Scalable Architecture:**
   The system supports load balancing and distributed server deployment to handle an increasing number of users efficiently, ensuring future scalability**.**

Chapter 4

Hardware and Software Requirements

Hardware Requirements –

Sno.	**Name**	**Specifications**
1	Processor	Inter core i5 12500+
2	Ram	4Gb or higher
3	Storage	Minimum 10GB free
4	Internet Connection	Minimum 1 Mbps

Software Requirements –

Sno.	**Name**	**Specifications**
1	Operating System	Windows/Linux/macOS
2	Programming Languages	JavaScript
3	Backend Framework	Node.js, Express.js
4	Frontend Framework	React.js
5	Database	MongoDB
6	WebSocket Library	Socket.io
7	Version Control	Git and GitHub
8	Package Manager	npm or yarn
9	Deployment	Vercel
10	CI/CD	Github Actions

Chapter 5

Possible Approach/ Algorithms

The chat application will be built using the following approach:

1. **Client-Server Communication:**
    - Web Sockets (Socket.io) enable real-time messaging by maintaining persistent connections between clients and the server.
    - REST APIs handle user authentication and message retrieval using Express.js for routing.
    - The server listens for incoming messages and broadcasts them to relevant users.
2. **Multithreading Implementation:**
    - Node.js's event-driven, non-blocking architecture ensures smooth performance by handling multiple requests asynchronously.
    - Worker threads are used to manage CPU-intensive tasks, ensuring efficient load distribution.
    - Each client connection is handled in a separate thread, preventing performance bottlenecks.
3. **Database Management:**
    - MongoDB is used to store users, messages, and chat history in a structured document format.
    - A caching mechanism (e.g., Redis) can be added to reduce database load and enhance performance.

4.**Authentication & Security:**

- JWT is used for stateless authentication, ensuring secure user sessions.
- Messages are encrypted using AES encryption before storage to prevent data leaks.
- HTTPS and secure Web Sockets (WSS) are implemented to prevent man-in-the-middle attacks.

5. **User Interface & Experience:**
    - React.js provides a fast and responsive UI, supporting both mobile and desktop users.
    - Notifications are integrated using browser APIs and Web Push to alert users of new messages.
    - UI enhancements like typing indicators and online/offline status improve user experience.

References

1. Tanenbaum, A. S., & Van Steen, M. (2007). Distributed Systems: Principles and Paradigms. Pearson Education.

2. Stevens, W. R., Fenner, B., & Rudoff, A. M. (2003). Unix Network Programming, Volume 1: The Sockets Networking API. Addison-Wesley.

3. Stallings, W. (2006). Operating Systems: Internals and Design Principles. Prentice Hall.

4. Comer, D. E. (2018). Computer Networks and Internet. Pearson.