A REALTIME RESEARCH PROJECT REPORT

On

A Secure Chat Application for Instant Communication

Submitted by,

RATHOD CHETAN	24J45A0504
K.SHIVA KUMAR	23J41A0533
M.PRADEEP KUMAR	23J41A0548
SANIYA SHAIK	23J41A0557

In partial fulfillment of the requirements for the award of the degree

Of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Under the Guidance of

Dr.TIRUMALA PARUCHURI

Assoc.Professor, MREC

COMPUTER SCIENCE AND ENGNEERING
MALLA REDDY ENGNEERING COLLAGE
(An UGC Autonomous Institution, Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad) Maisammaguda, Secunderabad, Telangana, India,500100

APRIL-2025

MALLA REDDY ENGINEERING COLLEGE

Maisammaguda, Secunderabad, Telangana, India, 500100

BONAFIDE CERTIFICATE

This is to certify that this Real time research project work entitled "**A Secure Chat Application for Instant Communication**", submitted by

RATHOD CHETAN (24J45A0504), K.SHIVA KUMAR(23J41A0533),M.PRADEEP KUMAR(23J41A0548),SANIYA SHAIK(23J41A0557) to Malla Reddy Engineering College affiliated to JNTUH, Hyderabad in partial fulfillment for the award of **Bachelor of Technology** in **COMPUTER SCIENCE AND ENGINEERING** is a bonafide record of project work carried out under our supervision during the academic year 2024 –2025 and that this work has not been submitted elsewhere for a degree.

SIGNATURE	**SIGNATURE**
Dr.TIRUMALA PARUCHURI	**Dr.PILLA SRINIVAS**
Assoc.Professor, CSE	**Professor & HOD-CSE**
Malla Reddy Engineering College	Malla Reddy Engineering College
Secunderabad,500100	Secunderabad,500100

Submitted for Real Time Research Project viva-voce examination held on____

INTERNAL EXAMINER

MALLA REDDY ENGINEERING COLLEGE

Maisammaguda, Secunderabad, Telangana, India 500100

DECLARATION

I here by declare that the Real time research project titled "**A Secure Chat Application for Instant Communication**", submitted to Malla Reddy Engineering College (Autonomous) and affiliated with JNTUH, Hyderabad, in partial fulfillment of the requirements for the award of a **Bachelor of Technology** in **COMPUTER SCIENCE AND ENGINEERING**, represents my ideas in my own words. Wherever others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity, and I have not misrepresented, fabricated, or falsified any idea, data, fact, or source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute. It is further declared that the project report or any part thereof has not been previously submitted to any University or Institute for the award of degree.

		Signature(s)
RATHOD CHETAN	24J45A0504	
SHIVA KUMAR	23J41A0533	
PRADEEP KUMAR	23J41A0548	
SANIYA SHAIK	23J41A0557	

MALLA REDDY ENGINEERING COLLEGE

Mismanaguda, Secunderabad, Telangana, India 500100

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to **Dr. A.Ramaswami Reddy, Professor& Principal, MREC(Autonomous)** for his invaluable support and encouragement through out the course of our project.

We extend our heartfelt thanks to **Dr. Pilla Srinivas, Professor and Head of the Department of Computer Science and Engineering**, for his insightful guidance and continuous encouragement during the project.

Our sincere appreciation goes to our **Project Coordinator, Ms.Vyshnava Divya, Assistant Professor, Department of Computer Science and Engineering**, for his unwavering cooperation and assistance during the project work.

We are deeply grateful to our **Project Guide, Dr.Tirumala Paruchuri, Assoc.Professor**, **Department of Computer Science and Engineering,** for her consistent guidance, direction, and support, which were instrumental in the successful completion of the project.

We also wish to thank all the teaching and non-teaching staff of the department for their valuable cooperation and support throughout the project.

RATHOD CHETAN	**24J45A0504**
K.SHIVA KUMAR	**23J41A0533**
M.PRADEEP KUMAR	**23J41A0548**
SANIYA SHAIK	**23J41A0557**

TABLE OF CONTENTS

ABSTRACT

The Secure Chat Application for instance Communication is a full-stack, real-time messaging system designed to facilitate seamless, interactive communication between users with a strong emphasis on responsiveness, scalability, and user experience. It combines the power of modern web technologies to deliver a feature-rich environment suitable for both personal and professional messaging use cases. The frontend is developed using **React and bootstrapped with Vite,** ensuring fast performance and modular component-based architecture. Bootstrap is employed to achieve a clean, responsive, and mobile-friendly user interface that enhances usability across devices. The interface includes features such as real-time message updates, chat partner switching, contextual user indicators, and message timestamps for better conversation tracking.The backend is built on Node.js with the Express framework to handle RESTful APIs and WebSocket communication. The core real-time functionality is managed by Socket.IO, which enables instant, bi-directional communication between connected clients without requiring manual page refreshes. For persistent data storage, the system utilizes MongoDB Atlas, a cloud-based NoSQL database, which stores messages and chat histories efficiently using Mongoose for schema modeling and validation. Environmental variables such as server ports and database URIs are securely managed through the dot env(.env) module, while CORS is configured to enable seamless integration between the frontend and backend hosted on different origins. Deployment readiness is also considered, with the application designed for containerization using Docker and scalability through Kubernetes. These tools would allow the system to be deployed efficiently across cloud environments and scaled based on user demand. Media and file sharing, emoji support, and audio/video calling are further envisioned to expand the feature set of the application. In conclusion, SmartConnect serves as a comprehensive chat solution that not only addresses the fundamental requirements of real-time messaging but also provides a robust foundation for the development of intelligent, scalable, and secure communication systems of the future.

LIST OF FIGURES

LIST OF ABBREVIATIONS

S.NO	ABBREVIATION	
1	UI	User Interface
2	UX	User Experience
3	DB	Database
4	API	Application Programming Interface
5	JWT	JSON Web Token
6	CRUD	Create, Read, Update, Delete
7	HTTP	HyperText Transfer Protocol
8	HTTPS	HyperText Transfer Protocol Secure
9	JSON	JavaScript Object Notation
10	REST	Representational State Transfer

CHAPTER – 1

INTRODUCTION

In today’s digitally connected world, real-time communication has become a fundamental requirement for both personal and professional interactions. Chat applications play a vital role in enabling instant messaging, collaboration, and engagement across the globe. This project focuses on the development of a web-based Chat Application that allows users to communicate in real-time through a secure, efficient, and user-friendly interface.

The application leverages modern web technologies, including React.js for the frontend, Node.js and Express.js for the backend, and MongoDB for data storage. Real-time messaging is facilitated using Web Sockets, implemented via Socket.io, allowing users to send and receive messages instantly without the need to refresh the page.

The system includes essential features such as user authentication, chat switching, and message history storage, ensuring both security and usability. By integrating these technologies, the application demonstrates the principles of full-stack development and real-time data handling.

This project not only showcases the practical implementation of software engineering concepts but also provides a foundation for scalable communication platforms with potential extensions like group chats, media sharing, and push notifications.

CHAPTER – 2
LITERATURE SURVEY

The development of chat applications has gone through significant transformation over the years. Early messaging platforms such as Yahoo Messenger and Google Talk relied on traditional client-server models and polling techniques, which were not efficient for real-time communication.

Modern chat applications like WhatsApp, Telegram, and Slack have adopted Web Sockets and other real-time technologies to enable instant, bidirectional communication. These technologies reduce latency and allow features such as typing indicators, read receipts, and real-time updates.

The use of the MERN stack (MongoDB, Express.js, React.js, Node.js) has become popular in recent years due to its full-stack JavaScript capabilities. It allows developers to create scalable and maintainable web applications. React provides dynamic user interfaces, Node.js and Express handle asynchronous backend processes, and MongoDB ensures flexible and fast data storage for messages and user data.

Security is a growing concern in messaging systems. Research highlights the importance of implementing features like user authentication, data encryption, and secure socket connections to protect users from data breaches and unauthorized access.

This project incorporates the above advancements—real-time messaging with Socket.io, full-stack development with MERN, and basic security features—to build a functional and modern chat application.

Real-Time Communication

Real-time communication is the cornerstone of any chat system. Traditional methods like HTTP polling have been replaced by efficient solutions such as **Web Sockets**, which allow full-duplex communication between the server and the client. Libraries like **Socket.IO** simplify real-time event handling and ensure cross-browser compatibility. These technologies help reduce latency and improve performance.

2. Backend Technologies

Node.js is widely adopted for backend development due to its non-blocking, event-driven architecture. This makes it suitable for handling thousands of concurrent connections, which is essential in chat systems. Frameworks like **Express.js** further simplify API development and server-side logic.

3. Frontend Frameworks

Modern frontend frameworks such as **React**, **Vue.js**, and **Angular** allow developers to build responsive and dynamic interfaces. These frameworks support modular UI development and efficient state management, which is important for updating chat views in real time without reloading the page.

4. Database Systems

Storing messages, user profiles, and conversation history efficiently is vital. **MongoDB**, a NoSQL database, is often used due to its document-based structure, scalability, and seamless integration with JavaScript-based backends like Node.js. Other databases like **Firebase Realtime Database** and **Redis** are also used in certain applications to handle real-time data sync and caching.

5. Authentication and Security

Security is a key concern in any chat application. User authentication is typically managed using **JWT (JSON Web Tokens)** for secure, stateless sessions. End-to-end encryption (E2EE) and secure transmission protocols (e.g., HTTPS, TLS) are commonly implemented to ensure confidentiality and data integrity.

6. Scalability and Deployment

To handle large user bases, modern chat apps leverage **cloud platforms** such as **AWS**, **Google Cloud**, and **Heroku**. These platforms provide scalability through auto-scaling, load balancing, and distributed architecture. Technologies like **Docker** and **Kubernetes** are also used for containerization and orchestration.

7. User Experience Enhancements

Features like typing indicators, read receipts, push notifications, file sharing, and emoji support are now standard in chat applications. These improve usability and create a more engaging experience for users. Libraries and APIs are available to implement these features efficiently.

CHAPTER – 3

SYSTEM ANALYSIS

System analysis is a critical step in the software development lifecycle that involves understanding user needs, defining functional and non-functional requirements, and analyzing the overall architecture of the system. For a real-time chat application, the system must ensure efficiency, scalability, security, and responsiveness.

3.1 Purpose of the System

The purpose of the Chat Application is to provide a real-time communication platform that enables users to exchange text messages efficiently and securely. It aims to support individual and group conversations, user authentication, message history, and message persistence. The system ensures smooth communication between users through a user-friendly interface and backend logic powered by technologies such as Node.js, Express.js, and MongoDB.

3.2 Users of the System

The system is designed for the following users:

End Users: Individuals who register and log in to send and receive messages. They can:
Initiate or continue conversations.
View message history.
Switch between chats.

Admin (optional): May monitor messages, manage user access, or analyze usage patterns (for moderation or analytics).

3.3 Core System Functions

The main functionalities of the chat system include:

User Registration & Login: Allows users to create accounts and log in securely.

User Authentication: Ensures only authorized users can access chat features using JWT/session tokens.

Real-time Messaging: Enables users to send and receive messages instantly (via Web Sockets or polling).

Chat Switching: Users can switch between different chat threads or contacts.

Message History: Previously exchanged messages are stored and retrieved when users open a chat.

Message Persistence: Messages are stored in a database (e.g., MongoDB) for future retrieval.

3.4 Inputs and Outputs

Inputs:

User Data: Username, email, password (during registration).

Login Credentials: Email/username and password.

Messages: Text content sent by users.

Chat Selection: Clicking or tapping on a contact or chat thread.

Outputs:

Chat Interface: Display of user list, chat window, and message history.

Notifications: New message alerts or errors.

Responses: Confirmation of successful login, registration, or message sent.

Error Messages: Invalid login, connection issues, etc.

3.5 Storage and Retrieval

Storage:

User Data: Stored in a secure user collection/table (includes hashed passwords).

Messages: Stored in a messages collection with sender, receiver, timestamp, and content.

Chat Threads: Optional - used to manage group chats or separate conversation contexts.

Retrieval:

Login Session: User credentials are verified during login, and user session is maintained.

Message History: When a user selects a chat, the system retrieves all past messages associated with that conversation from the database.

User List: Fetched to show available contacts.

CHAPTER – 4

SYSTEM STUDY

System Study:- The System Study involves a detailed examination of the existing communication methods and the design of a new system that meets user requirements for real-time interaction, efficiency, and ease of use. It includes analyzing the current systems, identifying limitations, and proposing a modern solution using advanced technology.

4.1 Problem Definition

The need for real-time communication is growing rapidly in both personal and professional environments. Existing communication platforms may be overloaded with unnecessary features, lack customization, or may not suit specific organizational needs. There is a requirement for a simple, efficient, and secure chat application that allows users to exchange messages in real-time, with persistent storage and easy retrieval of conversation history.

4.2 Feasibility Study

This includes analyzing the feasibility of developing the chat application across multiple dimensions:

- **Technical Feasibility**:
  - Tools like Node.js, Express, Web Sockets, and MongoDB are readily available and suitable for real-time communication.
  - The development team is capable of implementing the required technologies.
- **Economic Feasibility**:
  - Initial development cost is moderate, especially using open-source tools.
  - Cloud deployment can be scaled based on demand, reducing long-term costs.

4.3 Existing System

- Most communication systems (like WhatsApp, Messenger, Slack) are complex or bound to their ecosystems.
- Existing systems may:
  - Require heavy infrastructure.
  - Be closed-source, limiting customization.
  - Lack control over data privacy for organizations.

4.4 Proposed System

The proposed chat application aims to overcome the limitations of existing systems by offering:

- **Custom-built real-time chat** using modern technologies.
- **User-friendly interface** with easy chat switching and responsive design.
- **End-to-end communication** between users with data stored securely in a centralized database.
- **Authentication system** to protect user data and restrict unauthorized access.
- **Persistent chat history** with fast retrieval capabilities.

4.5 System Requirements

Hardware Requirements:

- **Server**:
    - Processor: Quad-Core or better
    - RAM: 8 GB minimum
    - Storage: SSD preferred, 50+ GB
- **Client**:
    - Any modern device (PC/Tablet/Smartphone) with internet access

Software Requirements:

- **Frontend**: HTML, CSS, JavaScript (React or plain)
- **Backend**: Node.js with Express.js
- **Database**: MongoDB
- **Real-time Communication**: Socket.IO or WebSocket API
- **Authentication**: JWT (JSON Web Tokens)
- **Operating System**: Cross-platform support; deployment on Linux server preferred

CHAPTER – 5

System Design

System design focuses on the architecture, modules, components, data flow, and interfaces of the application. It is divided into two main parts: High-Level Design (HLD) and Low-Level Design (LLD), each helping to visualize how the system will function as a whole.

5.1 Architecture Design

The chat application follows a **Client-Server architecture** with the following components:

1. Frontend (Client)

- Developed using HTML, CSS, JavaScript (or React).
- Handles user interface, sending/receiving messages, and interaction with the backend via APIs and Web Sockets.

2. Backend (Server)

- Built with **Node.js** and **Express.js**.
- Handles API requests (authentication, chat management).
- Manages real-time messaging using **Socket.IO**..

3. Real-time Communication

- Web Sockets (via Socket.IO) establish persistent connections for live message exchange

5.2 Main Modules

1. **User Module**
    - Register, login, logout
    - Authentication (JWT-based)
2. **Chat Module**
    - Start new chat
    - Switch between chats
    - Group or individual messaging

3. **Message Module**
    - Send and receive messages
    - Store messages in DB
4. **Socket Module**
    - Establish WebSocket connection
    - Real-time messaging logic

5.3 Data Flow and Design

1. **User Inputs**
    - Registration/Login credentials
    - Message content
    - Chat partner selection
2. **Processes**
    - Authenticate User
    - Send Message
    - Retrieve Chat History
3. **Outputs**
    - Login success/failure
    - Displayed chat messages
    - Notifications (new message, error

5.4 Entities and Relationships

Entities:

1. **User**
    - User_id (PK)
    - username

    - email
    - password (hashed)
    - created_at

2. **Message**
    - message_id (PK)
    - sender_id (FK → User)receiver_id (FK → User)
    - content
    - timestamp
    - chat_id (FK → Chat)

3. **Chat**
    - chat_id (PK)
    - user_ids (Array of participant IDs)
    - created_at

Relationships:

- **User → Message**: One-to-many (a user can send many messages)
- **Chat → Message**: One-to-many (a chat contains many messages)
- **User ↔ Chat**: Many-to-many (users can participate

CHAPTER – 6

System Requirements:-

System requirements are crucial for understanding the resources and technologies needed to run and develop the chat application. These requirements can be divided into **hardware** and **software** specifications.

6.1 Hardware Requirements

- **Processor**: Minimum Dual Core (Recommended: Intel i5 or higher)
- **RAM**: Minimum 4 GB (Recommended: 8 GB or more)
- **Hard Disk**: Minimum 500 MB free space (for development and deployment)
- **Display**: Standard HD monitor (1280x720 or higher)
- **Peripherals**: Keyboard, Mouse, Microphone (optional for voice chat

6.2 Software Requirements

- **Operating System**:
  - Windows 10 or higher
  - Linux (Ubuntu 18.04+ / Fedora / etc.)
  - macOS 10.14 or higher
- **Programming Languages**:
  - Frontend: HTML, CSS, JavaScript (React.js or similar)
  - Backend: Node.js (Express) or Python (Flask/Django)
- **Database**:
  - MongoDB (or alternative: MySQL/PostgreSQL)
- **Other Tools/Technologies**:
  - Web Browser (Chrome/Firefox/Edge)
  - Code Editor (e.g., VS Code)
  - Postman (for API testing)
  - Git (for version control)

  - npm or yarn (for package management)

6.3 Network Requirements

- **Internet Connection**:
  - Stable connection with at least 1 Mbps upload/download speed
  - Required for real-time messaging and backend API access
- **Ports**:
  - Backend Server: Port 6000 (or as configured)
  - Frontend: Default React port (3000)
  - MongoDB: Default port 27017
- **Protocols**:
  - HTTP/HTTPS for frontend-backend communication
  - WebSocket (or Socket.IO) for real-time chat functionality

CHAPTER 7

SYSTEM TESTING:-

System testing ensures that the Chat Application functions correctly, integrates seamlessly, and provides a user-friendly experience. The following types of testing were conducted:

7.1 Functional Testing

Functional testing verifies that all features of the chat application work as expected according to the requirements.

Tested Features:

- User Registration and Login
- Sending and Receiving Messages
- Real-time message updates (Web Sockets)
- Chatroom Switching
- Chat history retrieval from database
- Error handling (invalid login, empty messages, etc.)

Result:

All functions performed as intended under normal and boundary conditions.

7.2 Integration Testing

Integration testing focuses on verifying the correct interaction between different components of the application.

Components Integrated:

- Frontend with Backend (API communication)
- Backend with Database (MongoDB)
- WebSocket connection for real-time messaging
- Authentication with session/token-based systems

Result:

The system components interacted smoothly, and no critical issues were found during module integration.

7.3 Usability Testing

Usability testing assesses how user-friendly and intuitive the application is.

Test Criteria:

- Ease of Navigation (e.g., switching between chats)

- Interface Clarity (readability, button placement)
- Responsiveness (quick load times and smooth transitions)
- User Feedback (error/success messages, validations)

Method:

Testing was done by allowing users to interact with the app and collecting feedback through surveys and observations.

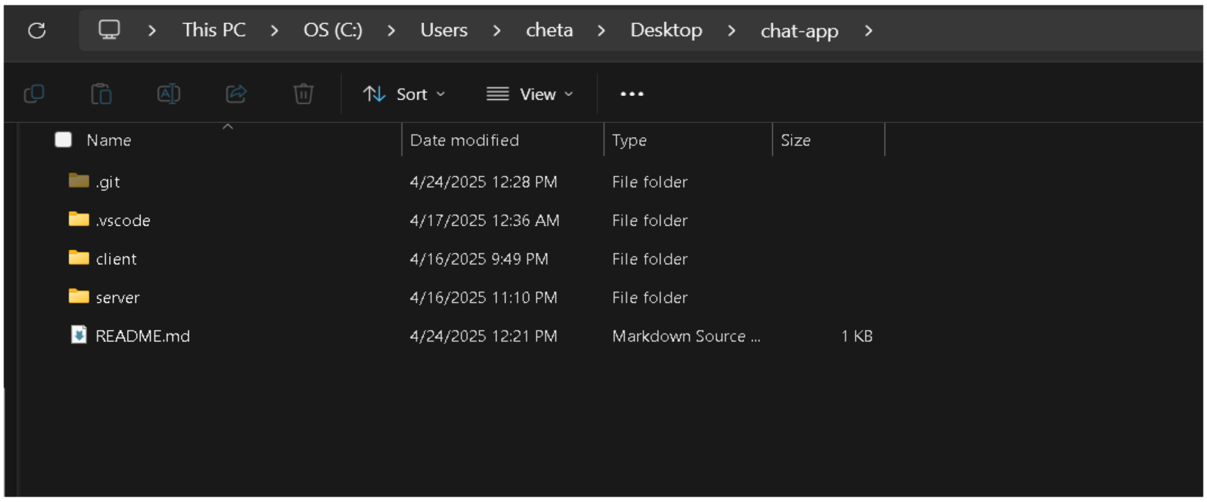
Result:

Most users found the interface intuitive and easy to use. Minor suggestions were implemented to improve layout and button visibility.

CHAPTER – 8

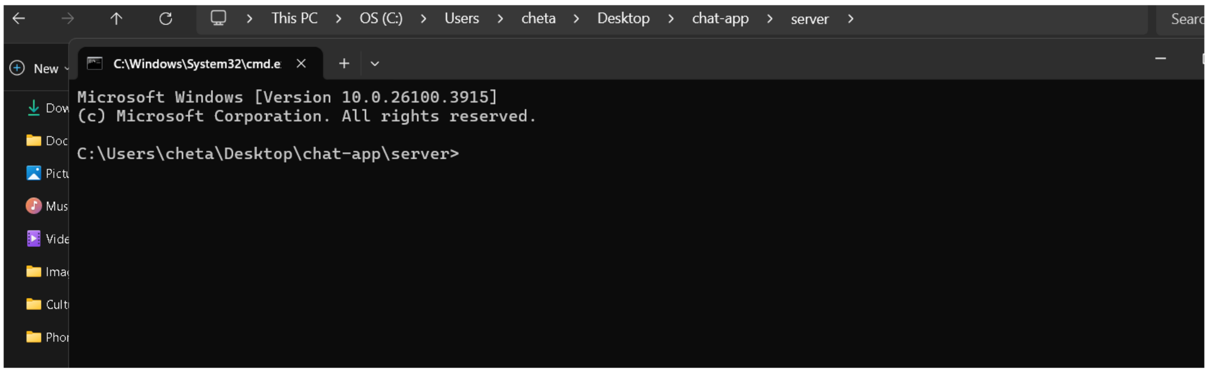
8.1 EXECUTION

Step 1: Open project folder:



```
Name          Date modified        Type                Size
.git          4/24/2025 12:28 PM   File folder
.vscode       4/17/2025 12:36 AM   File folder
client        4/16/2025 9:49 PM    File folder
server        4/16/2025 11:10 PM   File folder
README.md     4/24/2025 12:21 PM   Markdown Source ...   1 KB
```

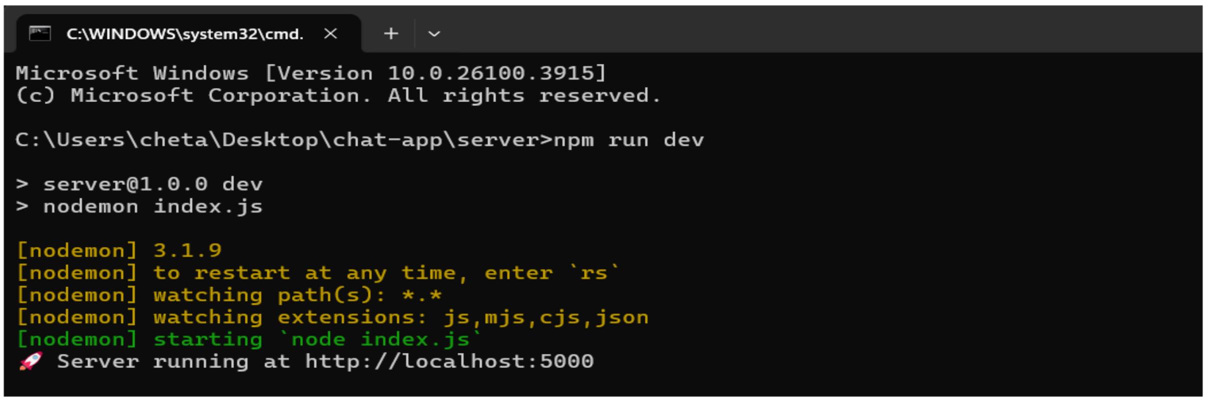
Step 2: Redirect to Server folder in CMD


This PC > OS (C:) > Users > cheta > Desktop > chat-app > server >
C:\Windows\System32\cmd.e

```
Microsoft Windows [Version 10.0.26100.3915]
(c) Microsoft Corporation. All rights reserved.

C:\Users\cheta\Desktop\chat-app\server>
```

Step 3: Run Server:"npm run dev"



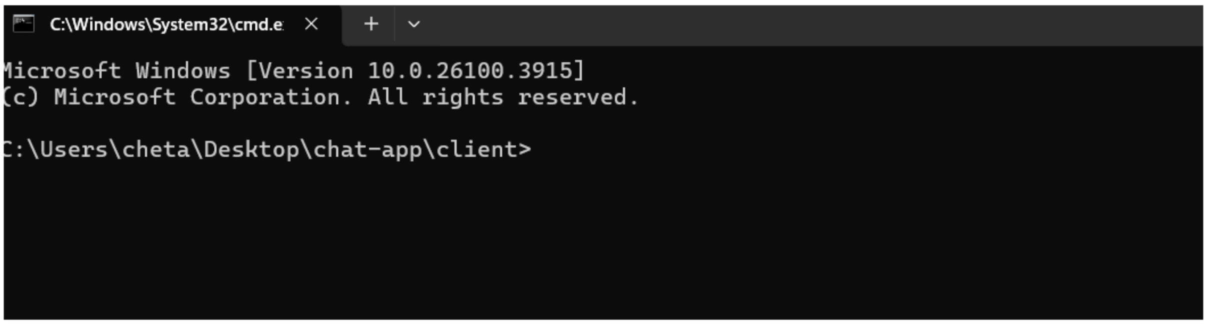
```
Microsoft Windows [Version 10.0.26100.3915]
(c) Microsoft Corporation. All rights reserved.

C:\Users\cheta\Desktop\chat-app\server>npm run dev

> server@1.0.0 dev
> nodemon index.js

[nodemon] 3.1.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
🚀 Server running at http://localhost:5000
```

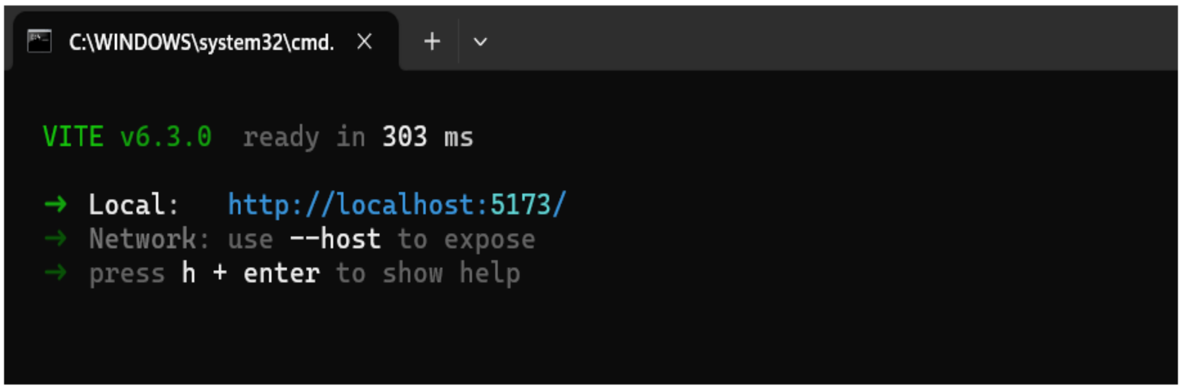
Step 4 : Redirect to Client folder in CMD:



```
Microsoft Windows [Version 10.0.26100.3915]
(c) Microsoft Corporation. All rights reserved.

C:\Users\cheta\Desktop\chat-app\client>
```

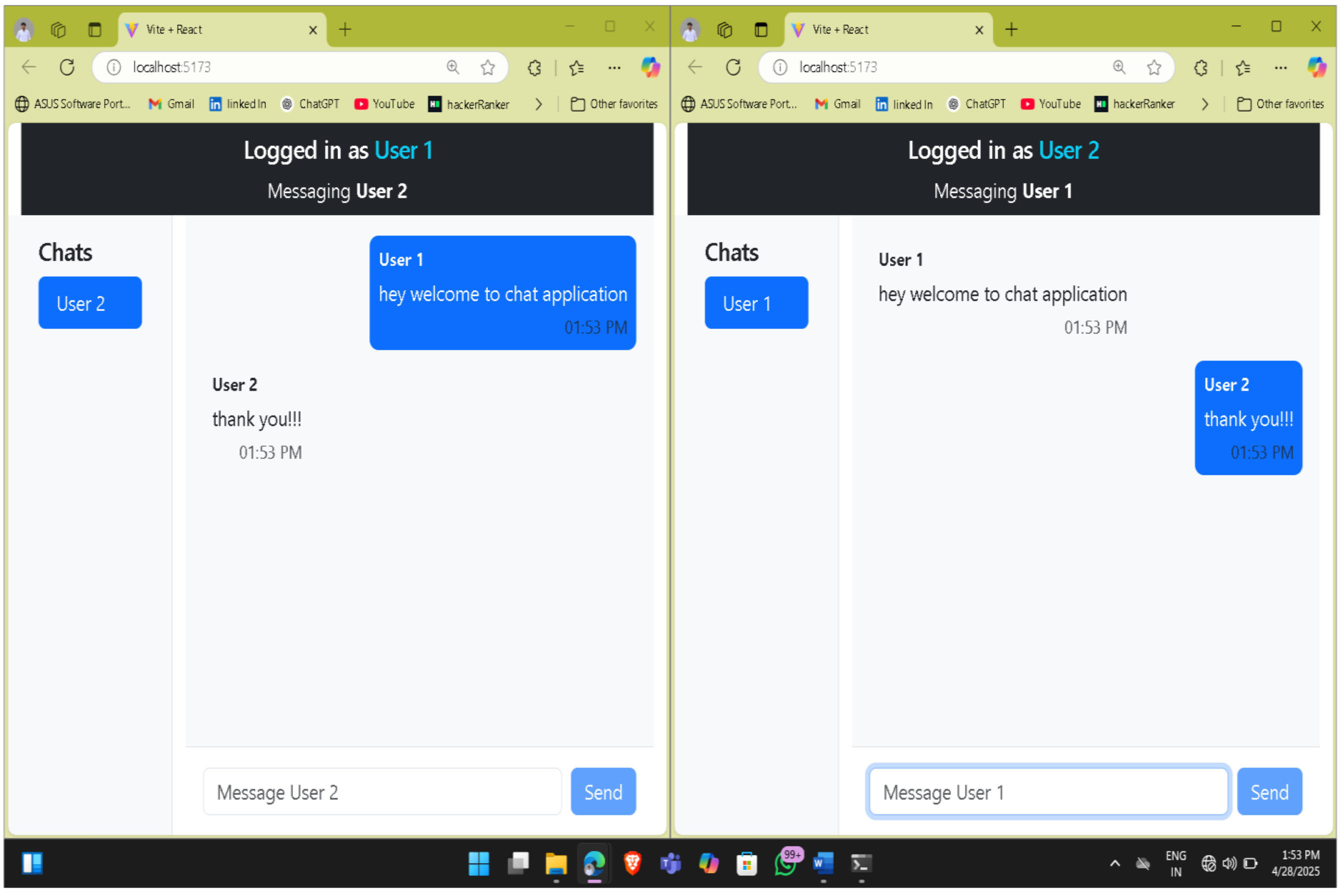
Step 5:Run Client files and Click(Clt +click) on local host:5173/



```
  VITE v6.3.0  ready in 303 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

8.2 Output:



Logged in as User 1
Messaging User 2

Chats
User 2

User 1
hey welcome to chat application
01:53 PM

User 2
thank you!!!
01:53 PM

Message User 2 Send

Logged in as User 2
Messaging User 1

Chats
User 1

User 1
hey welcome to chat application
01:53 PM

User 2
thank you!!!
01:53 PM

Message User 1 Send

CHAPTER - 9

CONCLUSION

The Chat Application successfully demonstrates the implementation of a real-time messaging system using modern web technologies. It integrates a user-friendly frontend built with React, a robust backend powered by Node.js and Express, and a MongoDB database for persistent storage of chat messages. Real-time communication is achieved using Socket.IO, enabling users to send and receive messages instantly.

This project not only highlights key concepts in full-stack development—such as API communication, Web Sockets, and database integration—but also emphasizes scalability, usability, and responsive design. Through functional, integration, and usability testing, the application has proven to be stable, intuitive, and reliable.

Overall, this Chat Application serves as a solid foundation for more advanced features such as private messaging, media sharing, authentication, and real-time notifications, making it suitable for further development or deployment in a real-world environment.

CHAPTER - 10

FUTURE WORK

While the current version of the Chat Application meets the basic requirements for real-time communication, several enhancements can be made to improve functionality, user experience, and scalability. Potential future developments include:

User Authentication & Profiles

Implement secure user registration, login, and profile management using JWT or OAuth2.

Private Messaging & Group Chats

Enable users to create private chats and custom group conversations with selective participants.

Media Sharing

Add support for sharing images, videos, and documents within the chat interface.

Chat Search & Filters

Allow users to search through messages and filter chats by user, date, or keyword.

Typing Indicators & Read Receipts

Enhance interactivity by showing when users are typing or when messages have been read.

Message Reactions and Emojis

Introduce emoji support and the ability to react to messages for a more expressive experience.

Push Notifications

Integrate browser and mobile notifications to alert users of new messages even when inactive.

Mobile App Version

Develop a cross-platform mobile version using React Native or Flutter.

Admin Dashboard

Include admin features for monitoring chats, managing users, and moderating content.

Scalability & Deployment

Deploy the application on cloud platforms (e.g., AWS, Heroku, or Vercel) with support for load balancing and database scaling.

CHAPTER – 11

REFERENCES

1. **Socket.IO Documentation**
   https://socket.io/docs
   Used for implementing real-time communication between client and server.
2. **Node.js Official Website**
   https://nodejs.org
   Used for building the backend server with Express framework.
3. **Express.js Documentation**
   https://expressjs.com
   Used to create RESTful APIs and handle routing in the backend.
4. **MongoDB Documentation**
   https://www.mongodb.com/docs
   Used for storing and retrieving chat messages in the database.
5. **React Official Documentation**
   https://reactjs.org/docs
   Used for building a responsive and dynamic user interface.
6. **Axios GitHub Repository**
   https://github.com/axios/axios
   Used for making HTTP requests from the frontend to the backend.
7. **MDN Web Docs**
   https://developer.mozilla.org
   General reference for JavaScript, HTML, CSS, and web APIs.
8. **W3Schools – Web Development Tutorials**
   https://www.w3schools.com
   Referenced for basic HTML, CSS, and JavaScript styling and structure.