A

Project Report

On

MERNCOM

Submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology

In

Computer Science and Engineering

By

Kiran	**2261322**
Jaya Mehta	**2261283**
Jatin Talaniya	**2261282**
Ayushi Adhikari	**2261136**

Under the Guidance of

Mr. Prince Kumar

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, **Kiran, Jaya Mehta, Jatin Talaniya**and **Ayushi Adhikari** hereby declare the work, which is being presented in the project, entitled ‘ **MERNCOM**’ 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. Prince Kumar.

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

Date: 27/05/2025

Kiran

Jaya Mehta

JatinTalaniya

Ayushi Adhikari

CERTIFICATE

The project report entitled " MERNCOM " being submitted Kiran (2261322) d/o Mr. Jagat Singh , Jaya Mehta (2261283) d/o Mr.Amar Singh Mehta, Jatin Talaniya(2261282) s/o Mr. Ashok Talaniya and Ayushi Adhikari (2261136) d/o Mr. Dhan Singh Adhikari B.Tech.(CSE) to Graphic Era Hill University Bhimtal Campus for the award of bonafide work carried out by them. They have worked under my guidance and supervision and fulfilled the requirement for the submission of a report.

Mr. Prince Kumar — **Dr. Ankur Singh Bisht**

(Project Guide) — **(Head, CSE)**

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. Prince Kumar’** (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.

Kiran	**2261322**
Jaya Mehta	**2261283**
Jatin Talaniya	**2261282**
Ayushi Adhikari	**2261136**

ABSTRACT

This project presents the development and implementation of a modern real-time chat application built using the MERN (MongoDB, Express.js, React, Node.js) stack. The application addresses the growing need for secure, efficient, and user-friendly real-time communication platforms. Leveraging cutting- edge technologies including Socket.io for real-time messaging, JWT for secure authentication, and Cloudinary for media handling, the application delivers a robust and scalable solution for instant messaging needs.

The system implements a comprehensive feature set including real-time messaging, user authentication, media sharing capabilities, and responsive design using TailwindCSS and DaisyUI. The architecture follows modern best practices with a clear separation of concerns between frontend and backend services, ensuring maintainability and scalability. Security is prioritized through implementation of JWT-based authentication, password encryption, and secure session management.

The project demonstrates successful integration of various technologies to create a seamless user experience while maintaining high performance and security standards. The implementation includes thorough testing methodologies, deployment strategies, and documentation, making it a production- ready solution that can be easily maintained and extended for future enhancements.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

- **API** – Functions/protocols for software communication
- **Bcrypt** – Secure password hashing algorithm
- **CLI** – Text-based command interface
- **CSS** – Styles web content
- **CRUD** – Create, Read, Update, Delete
- **DB** – Electronic data storage
- **DOM** – HTML/XML document structure interface
- **EJS** – HTML templating with JavaScript
- **ES6** – Modern JavaScript features (e.g., arrow functions)
- **GUI** – Visual user interface
- **HTML** – Markup language for web pages
- **HTTP** – Web data transfer protocol
- **HTTPS** – Secure HTTP
- **IPC** – Process communication/sync mechanism
- **IDE** – Software dev environment (code, debug, test)
- **IO** – Data input/output operations
- **JS** – Scripting for interactive web pages
- **JSON** – Lightweight data format
- **JWT** – Secure info token
- **JSX** – JavaScript + HTML syntax in React
- **LTS** – Long-term software support
- **MERN** – MongoDB, Express, React, Node stack
- **MVC** – Model-View-Controller pattern
- **NPM** – JavaScript package manager
- **OS** – Manages computer hardware/software
- **PWA** – Web app with native-like features
- **REST** – Stateless web architecture
- **SPA** – Single-page dynamic web app
- **SQL** – Database query language
- **SSR** – Server-side web rendering
- **TCP** – Reliable network protocol
- **UI** – User interface design/layout
- **Zustand** – Lightweight React state manager

1. INTRODUCTION

1.1 Prologue

In the era of digital communication, web-based chat applications have become indispensable tools for personal and professional interactions. This project, "Chat App", presents a modern web-based chat application built using the MERN stack (MongoDB, Express.js, React, Node.js), offering real- time messaging capabilities with a focus on user experience, security, and scalability. The application is designed to provide a seamless communication platform that combines modern web technologies with robust backend services.

1.2 Background and Motivations

The evolution of web technologies has enabled the creation of sophisticated real-time applications that rival traditional desktop applications in functionality and user experience. Modern web frameworks like React, combined with powerful backend technologies like Node.js and MongoDB, have made it possible to build complex applications that run entirely in the browser while maintaining high performance and reliability.

The motivation for this project stems from several key factors:

- The growing demand for real-time web applications in various sectors
- The need to understand and implement modern web development practices
- The opportunity to explore full-stack development using the MERN stack
- The challenge of building a secure and scalable real-time communication system

1.3 Problem Statement

While there are numerous chat applications available, building a custom web-based chat system presents several technical challenges:

- Implementing real-time bidirectional communication
- Ensuring secure user authentication and data transmission
- Managing state and data persistence effectively
- Creating a responsive and intuitive user interface
- Handling media sharing and file uploads
- Maintaining application performance under load

This project addresses these challenges by implementing a modern web application that demonstrates best practices in full-stack development and real-time communication.

1.4 Objectives and Research Methodology

- To design a full-stack chat application using the MERN stack
- To implement real-time chat using **Socket.IO**
- To use **Zustand** for global state management
- To design a responsive UI using **Tailwind CSS + DaisyUI**
- To study how Operating System concepts apply in real-time applications

Research was conducted through:

- Reading technical documentation (MDN, React, Socket.IO)
- Reviewing existing GitHub repositories
- Building and testing components iteratively

1.5 Project Organization

The report is divided into logical chapters:

- Chapter 2 covers system requirements and software environment
- Chapter 3 presents implementation with scripts and commands
- Chapter 4 includes snapshots
- Chapter 5 discusses limitations
- Chapter 6 suggests future enhancements
- Chapter 7 concludes the project

1.6 PHASES OF SOFTWARE DEVELOPMENT CYCLE

1.7 Hardware Requirement

1.7.1 Development Environment Hardware

- Processor : Intel Core i5/AMD Ryzen 5 or higher
- RAM : Minimum 8GB (16GB recommended)
- Storage : 256GB SSD or higher
- Internet Connection : Broadband connection (minimum 10 Mbps)
- Display : 1920 x 1080 resolution or higher

1.7.2 Server Requirements (Production)

- CPU: 2 vCPUs minimum
- RAM: 4GB minimum
- Storage: 20GB SSD minimum
- Network: High-bandwidth connection
- Scalability: Ability to scale vertically and horizontally

1.7.3 Client Requirements (End Users)

- Device : Any modern computer or mobile device
- Browser : Latest versions of Chrome, Firefox, Safari, or Edge
- Internet : Stable internet connection (minimum 5 Mbps)
- Display : Responsive design supports all screen sizes
- Storage : Minimal local storage requirements (< 100MB)

1.8 Software Requirement

1.8.1 Development Environment

1. Core Software

- Node.js (v16.0 or higher)
- npm (v8.0 or higher) or Yarn (v1.22 or higher)
- Git (v2.0 or higher)
- MongoDB (v5.0 or higher)
- VS Code or similar IDE

2. Development Tools

- Chrome DevTools or similar browser developer tools
- Postman or similar API testing tool
- MongoDB Compass for database management
- Docker (optional, for containerization)

1.8.2 Backend Requirements

1. Runtime Environment
- Node.js runtime
- Express.js framework
- MongoDB database
- Socket.io server

2. Key Dependencies

json
```
{
"express": "^4.21.2",
"mongoose": "^8.13.0",
"socket.io": "^4.8.1",
"jsonwebtoken": "^9.0.2",
"bcryptjs": "^3.0.2",
"cloudinary": "^2.6.0",
"cors": "^2.8.5",
"dotenv": "^16.4.7"
}
```

2. CODING OF FUNCTIONS

This chapter presents the functional implementation of the **MERNCom Chat Application**, including backend operations, real-time messaging, frontend interface logic, state management using Zustand, API interactions, utility methods, and database schema design.

4.1 Backend Functions

4.1.1 Authentication Functions

The backend provides user authentication via registration and login functions. Passwords are securely hashed using `bcrypt`, and `JWT` is used for token-based authentication.

```
const userAuth = {
  register: async (req, res) => {
    const { username, email, password } = req.body;
    const hashedPassword = await bcrypt.hash(password, 10);
    const newUser = new User({ username, email, password: hashedPassword });
    await newUser.save();
    const token = jwt.sign({ userId: newUser._id }, process.env.JWT_SECRET);
    res.status(201).json({ token, user: newUser });
  },

  login: async (req, res) => {
    const { email, password } = req.body;
    const user = await User.findOne({ email });
    if (!user) return res.status(404).json({ error: 'User not found' });
    const validPassword = await bcrypt.compare(password, user.password);
    if (!validPassword) return res.status(401).json({ error: 'Invalid password' });
    const token = jwt.sign({ userId: user._id }, process.env.JWT_SECRET);
    res.status(200).json({ token, user });
  }
};
```

4.1.2 Real-Time Message Functions

Using `Socket.IO`, the app enables real-time message transmission and typing indicators.

```
const messageHandler = (io, socket) => {
  socket.on('sendMessage', async (messageData) => {
    const { sender, receiver, content } = messageData;
    const newMessage = new Message({ sender, receiver, content, timestamp: new Date() });
    await newMessage.save();
    io.to(receiver).emit('newMessage', newMessage);
    socket.emit('messageSent', newMessage);
  });
  socket.on('typing', (data) => {
    socket.broadcast.to(data.receiver).emit('userTyping', {
      user: data.sender,
      isTyping: true
    });
  });
};
```

4.2 Frontend Functions

4.2.1 Chat Interface Components

The user interface for chat includes message display and input components.

```
const ChatMessage = ({ message, isOwn }) => (
  <div className={flex ${isOwn ? 'justify-end' : 'justify-start'} mb-4}>
    <div className={rounded-lg p-3 max-w-[70%] ${isOwn ? 'bg-blue-500 text-white' : 
'bg-gray-200'}}>
      <p className="text-sm">{message.content}</p>
      <span className="text-xs opacity-70">{formatTime(message.timestamp)}</span>
    </div>
  </div>
);

const ChatInput = ({ onSendMessage }) => {
  const [message, setMessage] = useState('');
  const handleSubmit = (e) => {
    e.preventDefault();
    if (message.trim()) {
      onSendMessage(message);
      setMessage('');
    }
  };

  return (
    <form onSubmit={handleSubmit} className="flex gap-2 p-4 border-t">
      <input type="text" value={message} onChange={(e) => setMessage(e.target.value)} 
className="flex-1 rounded-lg border p-2" placeholder="Type a message..." />
      <button type="submit" className="bg-blue-500 text-white px-4 py-2 rounded-
lg">Send</button>
    </form>
  );
};
```

4.2.2 State Management

```
const useChatStore = create((set) => ({
  messages: [],
  activeChat: null,
  loading: false,
  error: null,

  setActiveChat: (chatId) => set({ activeChat: chatId }),
  addMessage: (message) => set((state) => ({ messages: [...state.messages, message] })),

  fetchMessages: async (chatId) => {
    set({ loading: true });
    try {
      const response = await axios.get(/api/messages/${chatId});
      set({ messages: response.data, loading: false });
    } catch (error) {
      set({ error: error.message, loading: false });
    }
  }
}));
```

4.3 API Integration Functions

4.3.1 API Service Layer

This layer abstracts API calls for authentication and messaging.

```
const apiService = {
  user: {
    login: async (credentials) => {
      const response = await axios.post('/api/auth/login', credentials);
      return response.data;
    },
    register: async (userData) => {
      const response = await axios.post('/api/auth/register', userData);
      return response.data;
    }
  },

  messages: {
    fetch: async (chatId) => {
      const response = await axios.get(/api/messages/${chatId});
      return response.data;
    },
    send: async (messageData) => {
      const response = await axios.post('/api/messages', messageData);
      return response.data;
    }   }
};
```

4.4 Utility Functions

4.4.1 Helper Functions

The app includes utility functions for formatting time, encrypting messages, and uploading files.

```
export const formatTime = (timestamp) => {
  return new Date(timestamp).toLocaleTimeString('en-US', {
    hour: '2-digit',
    minute: '2-digit'
  });
};

export const encryptMessage = (message, key) => {
  //* Implementation of message encryption here*
};

export const uploadFile = async (file) => {
  const formData = new FormData();
  formData.append('file', file);
  const response = await axios.post('/api/upload', formData, {
    headers: { 'Content-Type': 'multipart/form-data' }
  });
  return response.data.url;
};
```

4.5 Database Models

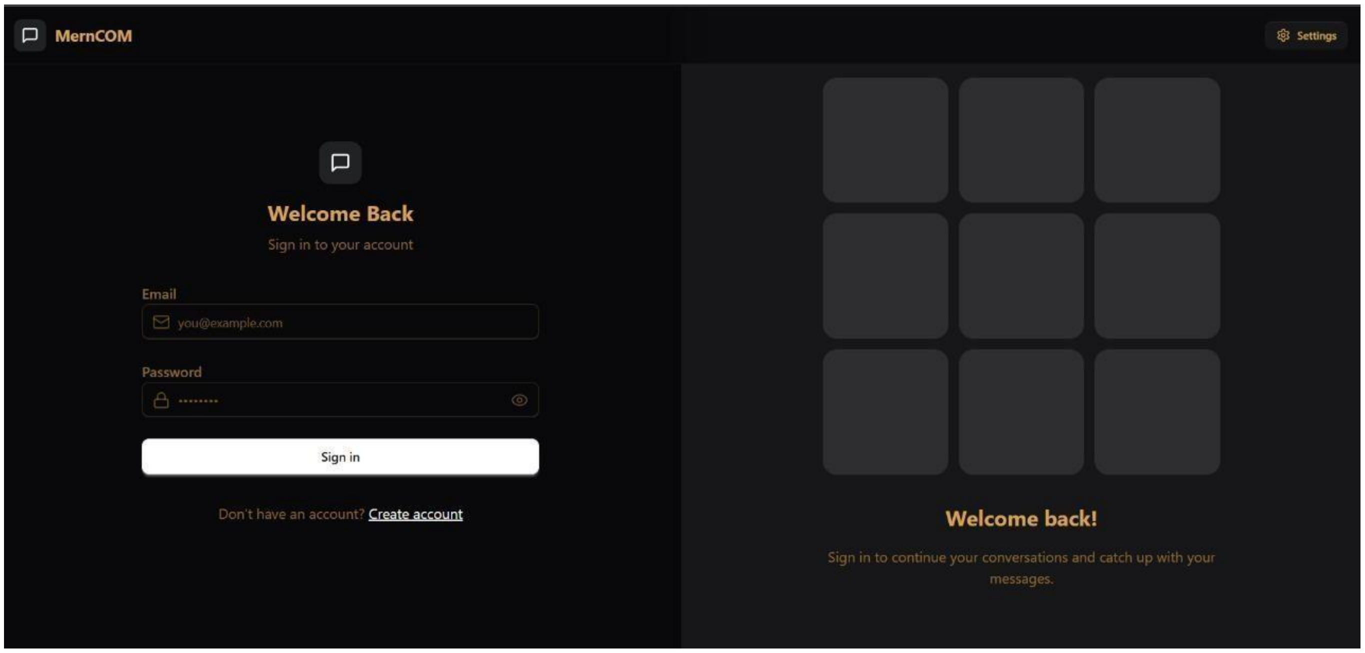
4.5.1 MongoDB Schemas

MongoDB is used for storing user and message data via Mongoose schemas.

```
const userSchema = new mongoose.Schema({
  username: { type: String, required: true, unique: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  avatar: String,
  createdAt: { type: Date, default: Date.now }
});

const messageSchema = new mongoose.Schema({
  sender: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  receiver: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  content: { type: String, required: true },
  timestamp: { type: Date, default: Date.now },
  read: { type: Boolean, default: false }
});
```

3. SNAPSHOTS



MernCOM

Settings

Welcome Back

Sign in to your account

Email

you@example.com

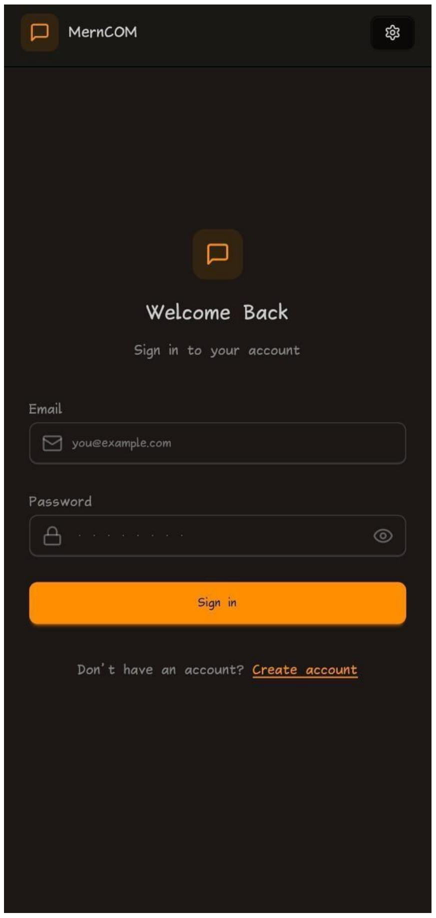
Password

Sign in

Don't have an account? Create account

Welcome back!

Sign in to continue your conversations and catch up with your messages.



MernCOM

Welcome Back

Sign in to your account

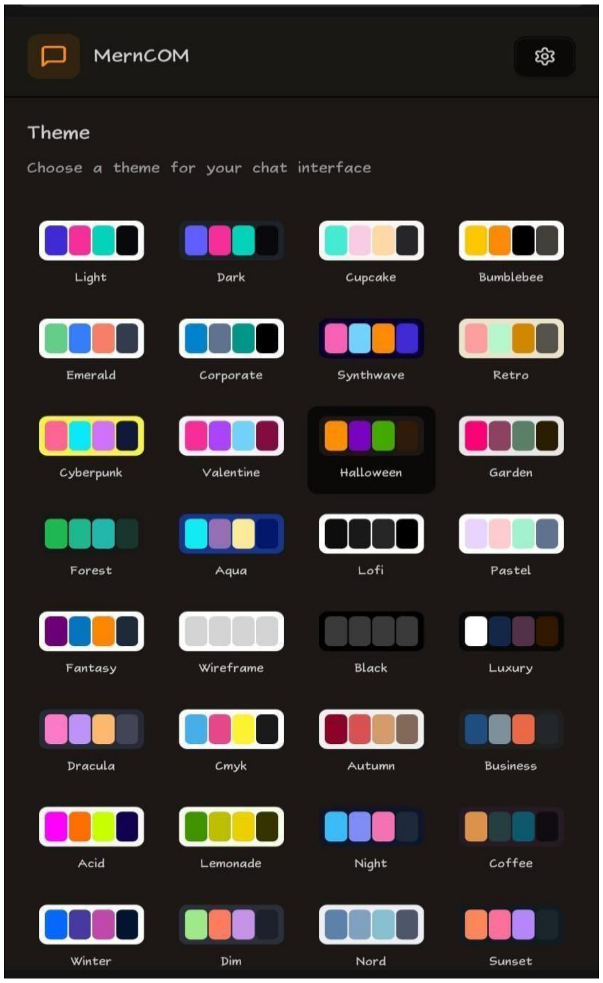
Email

you@example.com

Password

Sign in

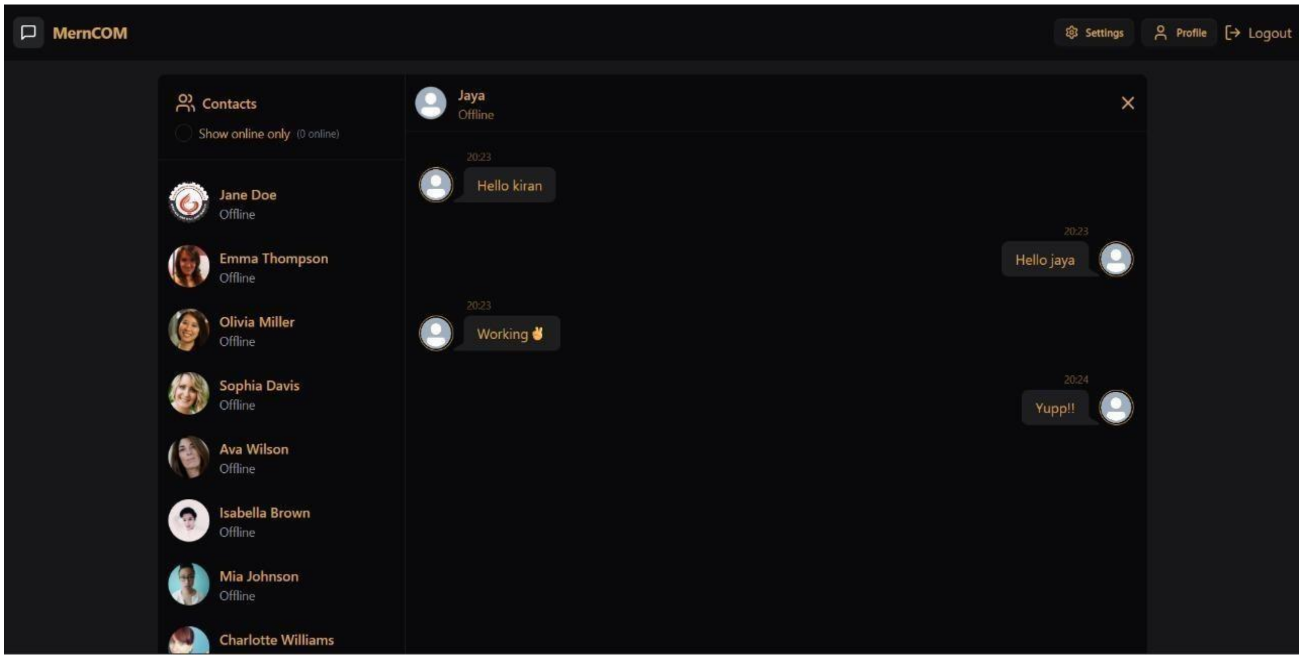
Don't have an account? Create account



Theme

Choose a theme for your chat interface

Light	Dark	Cupcake	Bumblebee
Emerald	Corporate	Synthwave	Retro
Cyberpunk	Valentine	Halloween	Garden
Forest	Aqua	Lofi	Pastel
Fantasy	Wireframe	Black	Luxury
Dracula	Cmyk	Autumn	Business
Acid	Lemonade	Night	Coffee
Winter	Dim	Nord	Sunset



Jane Doe Offline
Emma Thompson Offline
Olivia Miller Offline
Sophia Davis Offline
Ava Wilson Offline
Isabella Brown Offline
Mia Johnson Offline
Charlotte Williams

Jaya
Offline

20:23 Hello kiran

20:23 Hello jaya

20:23 Working ✌

20:24 Yupp!!

5. LIMITATIONS

5.1 Technical Limitations

5.1.1 Performance Constraints

The application supports around 1000 users, limits chat history to 1000 messages, restricts file uploads to 10MB, and has limited real-time connection capacity per server instance.

5.1.2 Database Limitations

Using MongoDB's free tier imposes constraints on storage, media handling, query performance, and lacks robust backup and restoration capabilities.

5.1.3 Real-time Communication

The app may experience reconnection delays, message latency in poor networks, lacks offline message queuing, and has WebSocket connection limits.

5.1.4 Feature Limitations

- Chat Features

  Group chat, message editing, deletion, voice/video calls, and advanced emoji support are not currently available.

- Media Handling

  The app supports limited file types, lacks image compression, video streaming, audio messages, and advanced media previews.

- User Interface

  Customization options are minimal with no dark mode, basic responsive design, limited accessibility, and no multi-language support.

5.2 Security Limitations

- Authentication Constraints

  Only email/password login is supported, with no MFA, social logins, cross-device sessions, or advanced recovery features.

- Security Features

  Encryption is basic, with no end-to-end protection, weak DDoS defense, limited rate limiting, and no IP blocking.

5.3 Privacy Limitations

- User Privacy

  User blocking, privacy settings, message self-destruction, visibility control, and incognito mode are limited or unavailable.

- Data Protection

  Data backups, exports, cleanup automation, and anonymization are basic and lack flexibility.

6.ENHANCEMENTS

6.1 Advanced Communication Features

The project can be enhanced by implementing group chat functionality with a well-defined database schema to support multiple participants, integrating voice and video calling using WebRTC for real-time media communication, and improving message handling to include read receipts, delivery status, and reactions.

6.2 Enhanced User Experience

The user interface can be upgraded with advanced UI components, customizable themes, and smooth transitions; additionally, inter-service communication between chat, notifications, and user management modules can improve responsiveness and modularity.

6.3 Security Enhancements

To strengthen data protection, the system can adopt end-to-end encryption, implement robust key management protocols, and ensure secure message transmission using modern encryption standards like AES and RSA.

6.4 Performance Optimizations

Performance can be improved by introducing local message caching, enabling offline messaging support, and optimizing data synchronization mechanisms for better speed and reliability.

6.5 Scalability Improvements

The app’s scalability can be significantly enhanced by transitioning to a microservices architecture, separating critical services such as chat, authentication, and media handling for better maintainability and load management.

6.6 Analytics and Monitoring

Advanced features like user activity analytics, real-time metrics tracking, and performance monitoring tools can be integrated to assess system health and user engagement, along with rich message interactions such as polls, media previews, and shared content tracking.

CONCLUSION

7.1 CONCLUSION

This project successfully delivered a real-time chat application built using the MERN stack, integrating modern web technologies to create a responsive, scalable, and secure communication platform. Key milestones included implementing real-time messaging with Socket.io, secure authentication, and a clean, user-friendly interface, all backed by efficient backend operations.

7.2 Learning Outcomes

The project strengthened core technical skills in frontend development with React.js, state management using Zustand, and backend architecture with Node.js and Express.js. It also provided practical exposure to WebSocket communication, MongoDB data modeling, and full-stack application deployment, enhancing both development and problem-solving capabilities.

7.3 Challenges and Solutions

Key technical challenges—such as ensuring real-time performance, managing complex application states, and implementing secure data handling—were addressed through optimized Socket.io integration, centralized state control, and JWT-based authentication and encryption, resulting in a robust and reliable user experience.

7.4 Future Scope

Upcoming improvements include group chat, voice/video calling, advanced media sharing, and end-to-end encryption. Long-term plans envision mobile app versions, microservices architecture, AI-driven features, and monetization options, positioning the app for broader adoption and scalability.

7.5 Final Thoughts

This project not only met its technical objectives but also provided a comprehensive learning experience in full-stack development, real-time systems, and secure communication protocols. It lays a strong foundation for further innovation and demonstrates the practical application of advanced development practices in building production-ready web solutions.

REFERENCES

[1] M. Popovic, "Introduction to the MERN Stack: MongoDB, Express.js, React, Node.js," FreeCodeCamp, Apr. 2022.

[2] Socket.IO, "Socket.IO — Real-time Engine," Socket.IO Docs, 2024.

[3] Mongoose, "Mongoose v7.0 Documentation," MongooseJS Official Docs, 2024.

[4] ReactJS, "Getting Started – React," Meta Open Source, 2024.

[5] Zustand, "Bear necessities for state management in React," Zustand Documentation, 2024.

[6] Tailwind Labs, "Tailwind CSS: Rapidly Build Modern Websites Without Ever Leaving Your HTML," TailwindCSS Documentation, 2024.

[7] DaisyUI, "Tailwind Components Plugin – daisyUI," daisyUI Docs, 2024.

.

[8] M. Jackson and R. Florence, “Introduction to REST APIs,” MDN Web Docs, 2024.

[9] D. Crockford, “JSON Web Token (JWT) Authentication Explained,” Auth0 Blog, 2023.