

ChatBuddy

PROJECTREPORT

Submitted by

**Manav Patel
Rudra Vaidya
Harshil Reddy**

**2201201082
2201201117
2201201043**

**Under the Guidance of
Ms. Niyati Mevada**

In partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

Computer Science and Engineering

Krishna School of Emerging Technology and Applied Research



Drs. Kiran & Pallavi Patel Global University

May, 2025



KPGU
Vadodara

Drs. Kiran & Pallavi Patel Global University, Vadodara
Krishna School of Emerging Technology and Applied Research

CERTIFICATE

This is to certify that the mini-project report entitled “**ChatBuddy Real Time Chat App**” submitted by **Manav Patel (2201201082) Rudra Vaidya (2201201117) Harshil Reddy (2201201043)** has been carried out under my guidance in partial fulfillment for the degree of Bachelor of Technology in **Computer Science and Engineering, 6th Semester of KSET, KPGU University, Vadodara, during the academic year 2024-2025.**

Ms. Niyati Mevada

Internal Guide

Head of the Department (KSET)

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to Ms. Niyati Mevada, our project guide, for her invaluable support, guidance, and encouragement throughout this mini- project. Her insights and expertise greatly contributed to the success of this project.

We also want to thank , Head of the Department(KSET), for providing the resources and conducive environment necessary for this project. His leadership and commitment to academic excellence have been a source of inspiration.

Finally, our thanks to everyone who assisted us during this project, including our fellow classmates and friends, for their collaboration, assistance, support and feedback. This project would not have been possible without the contributions of all those involved.



ABSTRACT

In today's digitally connected world, real-time communication platforms have become an integral part of both personal and professional interactions. From social networking to team collaboration, users demand seamless, fast, and responsive communication tools. This project, titled "**AI-Secured Real-Time Chat Application**", addresses this growing need by designing and developing a secure, scalable, and user-friendly chat system using the MERN stack, integrated with **Socket.IO** for real-time, bidirectional messaging capabilities.

A clean, responsive frontend interface built with **React.js** ensures cross-device compatibility and an intuitive user experience.

Security is a key focus of the application. User credentials are securely stored with **bcrypt hashing**, and all user sessions are authenticated and protected through encrypted tokens. The communication between users is facilitated in real-time using **Socket.IO**, allowing instantaneous message delivery without the need to refresh or reload the interface. This ensures a smooth and immersive chatting experience, similar to popular platforms like WhatsApp or Messenger.

This real-time chat system serves not only as a functional communication tool but also as a comprehensive learning model for full-stack development. It demonstrates how to integrate multiple technologies into a cohesive, fully operational web application, and offers significant insights into the domains of software engineering, real-time systems, secure communication, and cloud deployment.

In conclusion, the real-time chat application built with the MERN stack represents a modern, open-source solution for real-time communication needs. It has practical applications in educational institutions, small businesses, and developer communities looking for customizable, lightweight, and efficient alternatives to commercial chat platforms.

List of Figures

Flowchart of Chatapp	25
Flowchart of JWT auth	26
Screenshot of app interface	26.27
Database Screenshot.....	28
Sentiment analysis.....	29



Abbreviations

Abbreviation	Full Form
MERN	MongoDB, Express.js, React.js, Node.js
API	Application Programming Interface
JWT	JSON Web Token
UI	User Interface
UX	User Experience
DB	Database
CRUD	Create, Read, Update, Delete



Table of Contents

Acknowledgement	i
Abstract	ii
List of Figures	iii
List of Abbreviations	iv
Mini Project Activity Report	v

Contents

Chapter 1: Introduction	7
1.1 Introduction to Project (Literature Survey)	7
Project Category: Full Stack Development	8
1.2 Objectives	9
1.3 Identification/Recognition of Need	10
1.4 Existing System	11
1.5 Proposed System: Introducing ChatBuddy	12
1.6 Unique Features of ChatBuddy	14
1. Real time Communication with Socket.Io	14
2. Auth Using JWT	14
3. User status indicators	14
4. Responsive UI Design	14
5. Real World Tech Stack Exposure	14
6. Expandable architecture	14
Chapter 2: Requirement Analysis	16
2.1 Requirement Collection	16
2.2 Feasibility Study	17
Chapter 3 System Analysis	19
3.1 Problem Defination	19
3.2 System Objectives	19
3.3 System Scope	19
3.4 User Characteristic	19
3.5 System Features Overview	20
3.6 System Requirements	20
3.7 Data flow and interaction	21
3.8 Constraints and assumptions	21
Chapter 4 System Design	22

4.1 Design Goals	22
4.2 Architecture Design	22
4.3 System Architecture Diagram.....	23
4.4 Components Design.....	23
4.5 Database Design	23
4.6 DataFlow Design	24
4.7 Security Design	24
4.8 Responsiveness & Ui/UX consideration	24
4.9 Extensibility in Design	24
Chapter 5 Conclusion and Discussion.....	30



Chapter 1: Introduction

1.1 Introduction To Project (Overview Of Project)

- This project aims to build a fully functional real-time chat application using the MERN stack, which includes MongoDB, Express.js, React.js, and Node.js. The system allows users to register, log in securely, and exchange messages in real time through a responsive and modern interface.
- The application uses Socket.IO for establishing a persistent connection between clients and the server, enabling instant delivery of messages without refreshing the page. Messages are stored in MongoDB, which ensures persistence and allows users to access their chat history.
- The backend is built with Express.js and Node.js, handling API requests, authentication using JWT (JSON Web Tokens), and real-time communication logic. The frontend is developed with React.js, offering a seamless and dynamic user experience with real-time UI updates.
- This project provides hands-on experience with full-stack web development and demonstrates the practical application of real-time data transfer in modern web apps. It also serves as a foundation for building more complex communication platforms in the future.

Context and Relevance

- In the current digital age, instant communication is essential. Whether it's workplace collaboration, customer support, or social networking, users expect real-time responsiveness from the applications they use. Traditional web apps using request-response models struggle to meet these expectations.
- With the introduction of WebSocket technology and tools like Socket.IO, developers can now implement bi-directional real-time communication, drastically improving responsiveness and user engagement.
- The MERN stack is a popular choice for full-stack web development due to its use of JavaScript across all layers, enabling faster development and easier maintenance. When combined with Socket.IO and JWT for authentication, it becomes a powerful toolkit for building secure and scalable real-time applications.

Existing Literature and Similar Projects

A number of existing platforms and studies provide valuable insight into the development of real-time messaging systems:

- Slack and Microsoft Teams implement advanced WebSocket communication, enabling smooth, multi-user chats, notifications, and integrations.
- WhatsApp Web uses a combination of WebSockets and secure messaging protocols to provide real-time communication and encryption across devices.
- Rocket.Chat, an open-source chat system built with Meteor and MongoDB, demonstrates the viability of scalable messaging platforms with extensive customization options.

Project Category

This project is classified under the category of **Full-Stack Web Development**, with a specific focus on Real-Time Communication Systems. It involves the end-to-end development of a modern web application that provides real-time messaging functionality, using the MERN stack—MongoDB, Express.js, React.js, and Node.js—along with Socket.IO for real-time communication.

In today's digital environment, full-stack applications that offer live interactivity are crucial across various domains such as social networking, online collaboration, education platforms, and customer support systems. This project demonstrates the complete lifecycle of a real-time web application, covering frontend development, backend services, real-time communication, secure authentication, and persistent data storage.

Characteristics of the Project Category:

Full-Stack Web Development

- Utilizes a combination of frontend (React.js) and backend (Node.js + Express.js) technologies.
- Ensures a seamless flow of data between the client and server.
- Encourages modular development, maintainability, and scalability.

Real-Time Communication System

- Implements **Socket.IO**, a powerful JavaScript library for enabling real-time, event-based communication.
- Facilitates instant message delivery between users without the need to refresh the page.
- Provides bidirectional, low-latency communication using WebSockets.

Frontend

- Developed using **React.js**, a component-based framework known for its speed, flexibility, and dynamic rendering capabilities.
- Includes state management using `useState` and `useEffect` for real-time updates.
- Designed with responsive UI to function efficiently on both desktop and mobile devices.

Backend

- Powered by **Node.js** with **Express.js**, enabling a fast and non-blocking I/O environment suitable for scalable web servers.

- RESTful APIs are used to handle user authentication, message retrieval, and data posting.
- Authentication is handled securely using **JSON Web Tokens (JWT)**.

Database

- Utilizes **MongoDB**, a NoSQL document database, to store user data and chat history efficiently.
- Mongoose ODM (Object Document Mapper) is used to structure data models and schemas.

Security and Session Management

- User authentication is implemented using **JWT**, ensuring secure access to protected routes and components.
- Includes basic validation and middleware to protect endpoints and prevent unauthorized access.

Educational and Practical Relevance

- This project serves both academic and real-world purposes by providing an example of how modern full-stack applications are structured and function.
- It helps learners understand integration of real-time features into scalable applications.
- Can be extended with advanced features like group chats, file sharing, message notifications, or AI-based moderation.

1.2 Objectives

The main objective of this project is to design and develop a real-time chat application using the MERN stack (MongoDB, Express.js, React.js, Node.js), which facilitates secure, fast, and responsive communication between users through a web interface.

General Objective

- To build a full-stack, real-time web application that enables users to send and receive messages instantly using modern web technologies and secure authentication practices.

Specific Objectives

1. To implement a real-time communication system

Enable instant messaging using Socket.IO to establish and manage WebSocket connections for live data exchange.

2. To create a responsive user interface
Design the frontend using React.js to provide a seamless and dynamic chat experience that adapts to different screen sizes (desktop and mobile)..
3. To provide secure user authentication
Use JSON Web Tokens (JWT) to securely manage user sessions, ensuring that only authenticated users can access the chat features.
4. To store and manage user and chat data
Integrate MongoDB for storing user profiles, login credentials, and chat histories in a structured and persistent way.
5. To ensure low latency and reliability
Optimize the architecture to reduce delays in message transmission and improve user experience.
6. To enhance learning and practical knowledge
Provide a hands-on learning opportunity in integrating frontend, backend, database, and real-time technologies to build a functional and modern web application.

1.3 Recognition of Need

In the era of digital transformation, communication has evolved from traditional, time-delayed methods such as emails and SMS to instantaneous, real-time messaging systems. This evolution has significantly impacted the way individuals, teams, and organizations communicate—especially in the context of remote work, e-learning, customer service, healthcare, and social media.

With the growing reliance on internet-based communication, there is an increasing demand for systems that allow users to exchange messages, data, and files instantly and reliably. Users expect minimal latency, secure data transfer, and a smooth, responsive interface that works seamlessly across all devices.

Many existing commercial platforms—such as Slack, Microsoft Teams, WhatsApp, and Discord—offer robust real-time messaging features. However, these platforms are typically proprietary, not open-source, and often lack flexibility for customization. Organizations, developers, and educational institutions may require their own tailored chat systems to integrate with their workflows, products, or learning environments.

From an educational and developmental perspective, there is a strong need for project-based learning models that teach students and developers how to build and deploy such applications from the ground up. Learning how to implement real-time systems using technologies like the MERN stack and Socket.IO is not only highly relevant but also aligns with current industry demands.

Identified Needs Addressed by This Project:

- The need for customizable and open-source real-time communication tools
- The demand for practical applications in full-stack web development education
- The requirement for secure, scalable messaging systems in various industries
- The necessity of cross-platform communication tools accessible from any device
- The importance of user-friendly, responsive UI design for better user engagement
- The need for a modular architecture to allow further expansion (e.g., file sharing, group chat, video calls)

1.4 Existing System

Several real-time messaging platforms already exist and are widely used across different domains. These systems are built using advanced technologies that offer fast, secure, and interactive communication experiences. The most notable examples include Slack, WhatsApp, Microsoft Teams, Discord, and Telegram. These applications offer a broad range of features, including group messaging, media sharing, message history, notification systems, and integration with other tools or bots.

While these existing systems are powerful, they are also proprietary, resource-intensive, and often not fully customizable. Moreover, they may not provide open-source accessibility for educational or research-based development purposes. Understanding how these systems work and what they offer helps identify the gaps and opportunities that this project can address.

Overview of Prominent Existing Systems

WhatsApp

- A mobile-first messaging app offering secure, end-to-end encrypted communication.
- Supports media sharing, voice/video calls, and group chats.
- Built using a proprietary backend system with limited web access.
- Not open-source; customization or extension is not feasible for developers.

Microsoft Teams

- Integrated with Microsoft 365 and designed for enterprise-level team communication.
- Provides chat, video conferencing, task assignments, and file sharing.
- Uses a combination of real-time and asynchronous messaging services.
- Proprietary system with no access to source code or architecture for learning or modification.

Discord

- Initially developed for gamers, now widely adopted across communities and educational groups.
- Offers real-time messaging, voice/video calls, and community (server)-based organization.
- Built on scalable, real-time WebSocket-based infrastructure.
- Partially open for bot integration but not customizable at the core level.

Common Features Among Existing Systems

- Real-time bi-directional communication
- Multimedia sharing (images, files, audio)
- Push notifications
- Group chat and broadcast capabilities
- Device synchronization (web/mobile/desktop)
- Some level of encryption and user privacy
- APIs for basic bot/integration support

1.5 Proposed System

The proposed system aims to build a secure, responsive, and scalable real-time chat application using the MERN stack (MongoDB, Express.js, React.js, Node.js) in combination with Socket.IO for bi-directional communication. This application is designed as a complete full-stack solution that demonstrates the integration of frontend and backend components while enabling real-time functionality.

Unlike existing commercial messaging platforms, which are mostly proprietary and rigid, the proposed system is built with modularity and customization in mind. It

serves as both a learning platform and a working prototype that can be extended to suit specific needs in education, enterprise communication, or social interaction.

System Components:

1. Frontend (Client-Side) – React.js

- Built with React for a dynamic single-page interface.
- Utilizes state management and hooks for real-time UI updates.
- Ensures mobile responsiveness and optimized user experience.

2. Backend (Server-Side) – Node.js with Express.js

- Handles API requests, authentication, and message logic.
- Utilizes Express.js for fast routing and RESTful service design.

3. Database – MongoDB with Mongoose

- Stores user data, chat logs, timestamps, and user sessions.
- Ensures high availability and scalability through NoSQL schema.

4. Real-Time Communication – Socket.IO

- Enables real-time message transmission between users.
- Implements features like typing indicators, live status updates, etc.

5. Authentication – JWT (JSON Web Token)

- Secure user login and session management.
- Prevents unauthorized access to chat services.

Key Benefits:

- **Educational Utility:** Serves as an ideal project for learning full-stack and real-time application development.
- **Customizability:** Fully open-source and modular for customization in future use-cases.
- **Scalability:** Easily extendable to include additional modules such as video calling, file sharing, group chat, and emoji reactions.
- **Lightweight & Efficient:** Built to function efficiently on moderate resources for academic or small business environments.

Use-Case Scenarios:

- Internal team communication within startups or educational institutions.
- One-on-one student-instructor communication platforms.
- Lightweight alternative to enterprise chat apps for small organizations.
- Learning environment for students exploring MERN stack development.

1.6 Unique Features

The proposed system introduces several unique features that distinguish it from existing chat solutions and enhance both its functionality and learning value. These features are designed to be modular, efficient, and secure, while also providing a clear pathway for future expansion.

1. Real-Time Communication with Socket.IO

- Enables bi-directional, event-based communication over WebSockets.
- Unlike traditional polling methods, this ensures messages are transmitted with near-zero latency.

2. Authentication using JWT

- Secure session management using tokens.
- No need for server-side session storage, improving scalability and performance.

3 User Status Indicators

- Real-time user status (Online/Offline).
- Typing notifications and message seen/read receipts (optional extensions).

4 Responsive UI Design

- Built with modern React components and responsive layout.
- Seamless experience across desktop, tablet, and mobile devices.

5 Real-World Tech Stack Exposure

- Introduces developers to the same stack (MERN + WebSockets) used in real-world applications such as Slack clones, gaming chats, and customer service tools.

6 Expandable Architecture

- Easily upgradable to include:

- Group messaging
- Multimedia support
- AI integration for chatbot services
- Push notifications
- Admin panel



Chapter 2: Requirement Analysis

Requirement Analysis

The Requirement Analysis phase plays a crucial role in the success of any software development project. It involves identifying, gathering, and analyzing both functional and non-functional requirements to ensure that the final system meets the needs of its intended users. This section details the requirements collection process and assesses the feasibility of the proposed real-time chat application.

2.1 Requirement Collection

Requirement collection involves gathering all necessary information from the stakeholders (in this case, developers, potential users, and instructors) to define what the system is supposed to do. The requirements are categorized as follows:

Functional Requirements

These are the specific features and functionalities that the system must provide:

- User Registration and Login
 - Users should be able to register and securely log in using email and password.
 - Authentication is managed using JWT.
- Real-Time Messaging
 - Users can send and receive messages in real time using Socket.IO.
 - Messages should appear instantly without refreshing the page.
- User List Display
 - A list of all registered users should be visible to start a conversation.
- Chat History
 - Past messages should be stored in a database and loaded when the chat window is opened.
- Typing Indicators (optional)
 - Users should see a typing indicator when the other user is typing.
- Online Status Indicator
 - Users should be able to see which contacts are currently online.

Non-Functional Requirements

- Performance
 - The system should handle concurrent users with low latency.
- Scalability
 - The architecture should support scaling up to group messaging and media sharing in the future.
- Security
 - User data and credentials must be protected using hashing (e.g., bcrypt) and secure tokens (JWT).
- Responsiveness
 - The UI should be mobile-friendly and function seamlessly across devices.
- Maintainability
 - The codebase should be modular, well-commented, and easy to update.

2.2 Feasibility Study

A feasibility study assesses whether the project is practical and achievable within the given constraints. It evaluates the project from different angles to ensure it is worth developing.

1. Technical Feasibility

- The MERN stack (MongoDB, Express.js, React.js, Node.js) is a modern and well-supported technology stack suitable for building full-stack web applications.
- Real-time communication is effectively implemented using Socket.IO, which is highly compatible with Node.js.
- Hosting options like Heroku, Vercel, and Render are available for free or low-cost deployment, making this project technically feasible.

2. Economic Feasibility

- Since the technologies used are open-source, there is no licensing cost.
- Development and deployment can be done using free-tier cloud services, minimizing infrastructure expenses.
- Ideal for academic and personal use where budget constraints are common.

3. Operational Feasibility

- The system is designed with simplicity and user-friendliness in mind, making it easy to use even for non-technical users.
- All planned features have been designed with practicality, ensuring that users can interact without a steep learning curve.

4. Schedule Feasibility

- The development of the MVP (Minimum Viable Product) can be completed within a few weeks.
- Each phase—frontend, backend, real-time communication—can be handled in parallel or sequential sprints.



Chapter 3: System Analysis

3. System Analysis

System Analysis is a crucial phase in the software development life cycle (SDLC), where the existing system (if any) is studied in detail and a comprehensive plan is created to develop the proposed system. This stage identifies system requirements, analyzes how various components will interact, and lays the foundation for the system's architecture and functionality.

In the context a Real-Time Chat Application built using the MERN stack and Socket.IO—system analysis provides insights into data flow, user interactions, technical needs, and logical components.

3.1 Problem Definition

Modern communication relies heavily on real-time interaction. While many commercial chat applications exist, most are closed-source, costly, or too complex for educational or small-scale use. The lack of open, customizable, and scalable platforms for learning and internal communication creates a need for a lightweight real-time chat solution that is easy to develop, deploy, and adapt.

3.2 System Objectives

- Enable users to send and receive messages in real time.
- Provide secure user authentication.
- Maintain chat history using a database.
- Allow status and presence indicators (online/offline).
- Deliver a responsive, intuitive user interface.
- Be fully customizable and open-source.

3.3 System Scope

The system is designed for:

- Academic and learning purposes.
- Small team collaboration.
- Lightweight deployment scenarios.

The system does not initially include video/audio calls, media file sharing, or advanced moderation tools—but it is structured to support those in future iterations.

3.4 User Characteristics

- General Users: Can register, log in, send/receive messages, view other online users.
- Admin (optional extension): May monitor user activity, manage access, or view logs (not included in MVP but can be added).

3.5 System Features Overview

Feature	Description
User Authentication	Login/Signup with JWT, password hashing via bcrypt
Real-Time Chat	Instant messaging using Socket.IO and WebSockets
Persistent Chat History	MongoDB stores past messages and user data
Online Presence	Users can see who is currently online
Responsive Design	React-based UI adapts across devices (desktop, tablet, mobile)
Modular Codebase	Separate components for routing, services, and UI

3.6 System Requirements

a) Hardware Requirements

- Minimum 4GB RAM system
- Dual-core processor
- Stable internet connection (for deployment/testing)

b) Software Requirements

- Frontend: React.js (with Vite or Create React App)

- Backend: Node.js with Express.js
- Database: MongoDB (local or Atlas cloud DB)
- Real-time Engine: Socket.IO
- Development Tools: VS Code, Postman, Git, npm/yarn

3.7 Data Flow and Interaction

- User logs in via frontend → Token is sent to backend → Backend validates and returns a response.
- After login, a WebSocket connection is established.
- Messages typed in the frontend are sent via Socket.IO to the backend → Stored in MongoDB → Broadcast to recipient.
- The chat history loads automatically when a conversation is opened.

3.8 Constraints and Assumptions

- System depends on constant internet connectivity for WebSocket communication.
- Initially supports only text messages (future updates may include media).
- MVP is designed for 1-on-1 chat; group chat is a potential future module.
- Assumes users are on modern browsers and not using outdated systems.

Chapter 4: System Design

4. System Design

System Design is a critical phase in the software development life cycle, where the architecture of the system is carefully planned. It involves translating the functional requirements into structured components, defining how data flows within the system, and ensuring that the design aligns with performance, scalability, and maintainability goals.

In this project, the Real-Time Chat Application is designed using the MERN stack for full-stack JavaScript development and Socket.IO for real-time communication.

4.1 Design Goals

- To ensure scalability for handling multiple users in real-time.
- To achieve separation of concerns between the frontend, backend, and database layers.
- To implement modular and reusable components.
- To support secure and fast communication through real-time data transmission.
- To allow easy customization and extension in future development.

4.2 Architectural Design

The architecture of this chat application follows a three-tier architecture enhanced with WebSocket-based communication:

1. Presentation Layer (Frontend)

- Built using React.js
- Handles user interaction (chat UI, login/register forms)
- Communicates with backend through REST APIs and WebSockets (Socket.IO)
- Uses responsive design for mobile and desktop views

2. Application Layer (Backend)

- Built using Node.js and Express.js
- Provides RESTful API endpoints for authentication, user management, and chat history
- Handles WebSocket connections for real-time messaging
- Performs JWT-based authentication

3. Data Layer (Database)

- MongoDB is used as a NoSQL database
- Stores users, chat messages, and conversation history
- Uses Mongoose for schema definition and data modeling

4.3 System Architecture Diagram

4.4 Component Design

Frontend Components (React.js)

Component	Description
Login/Register	Authentication UI with form validation
ChatWindow	Main interface for chatting, shows messages and user input field
UserList	Sidebar listing all registered/online users
MessageBubble	Displays individual chat messages (own and others')
Header/Footer	Optional layout and navigation components

4.5 Database Design

Collections in MongoDB

1. Users

- _id
- username
- email
- passwordHash
- onlineStatus

2. Messages

- _id
- senderId
- receiverId
- messageText
- timestamp

4.6 Data Flow Design

Message Transmission Flow (Real-Time)

1. User A types a message in the frontend UI.
2. Socket.IO emits the message to the backend server.
3. Backend receives the message, stores it in MongoDB.
4. Backend broadcasts the message to the intended recipient using Socket.IO.
5. User B receives and sees the message in real time.

Login Flow

1. User enters email/password on login form.
2. Data is sent to backend via REST API.
3. Backend validates credentials and sends JWT.
4. Frontend stores token and initiates WebSocket connection.

4.7 Security Design

- Passwords are stored securely using bcrypt hashing.
- All routes are protected using JWT authentication middleware.
- WebSocket connections are initialized after authentication to ensure secure messaging.
- API rate-limiting (optional) and input validation prevent common attacks (e.g., brute force, injection).

4.8 Responsiveness & UI/UX Considerations

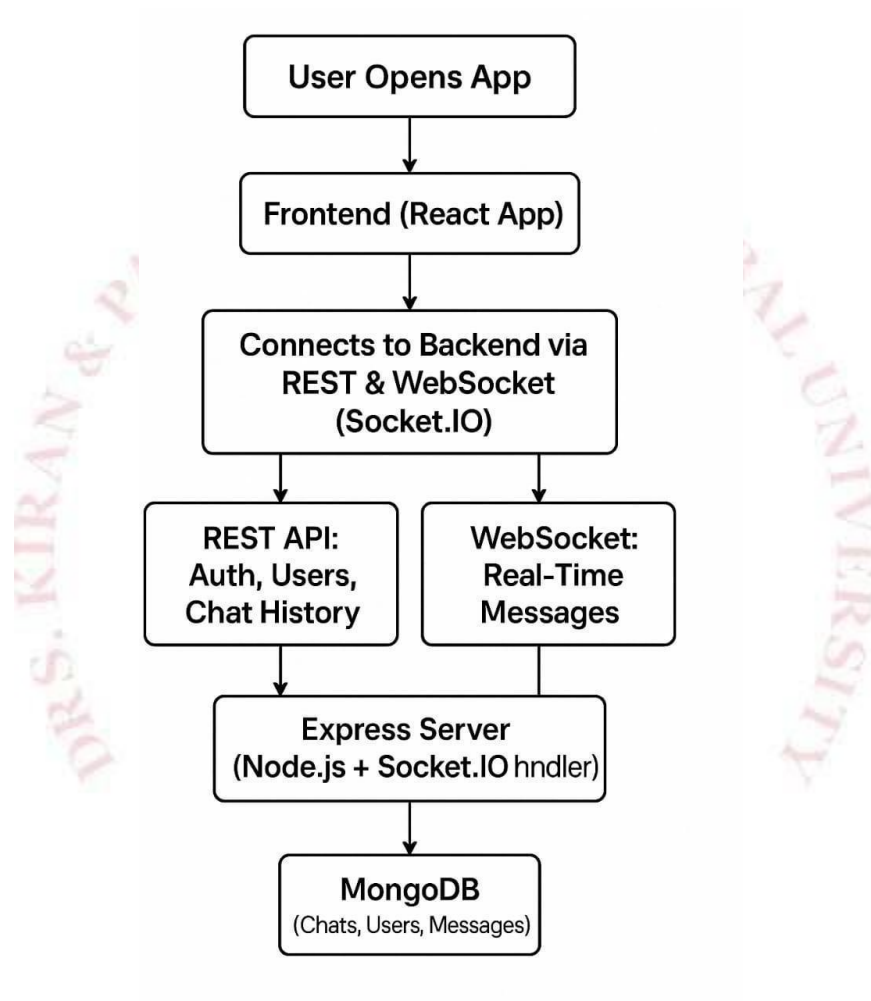
- The frontend uses CSS Flexbox/Grid for responsive layout.
- Media queries ensure proper rendering on both desktop and mobile devices.
- UI is minimalistic and intuitive, ensuring ease of use for all user categories.

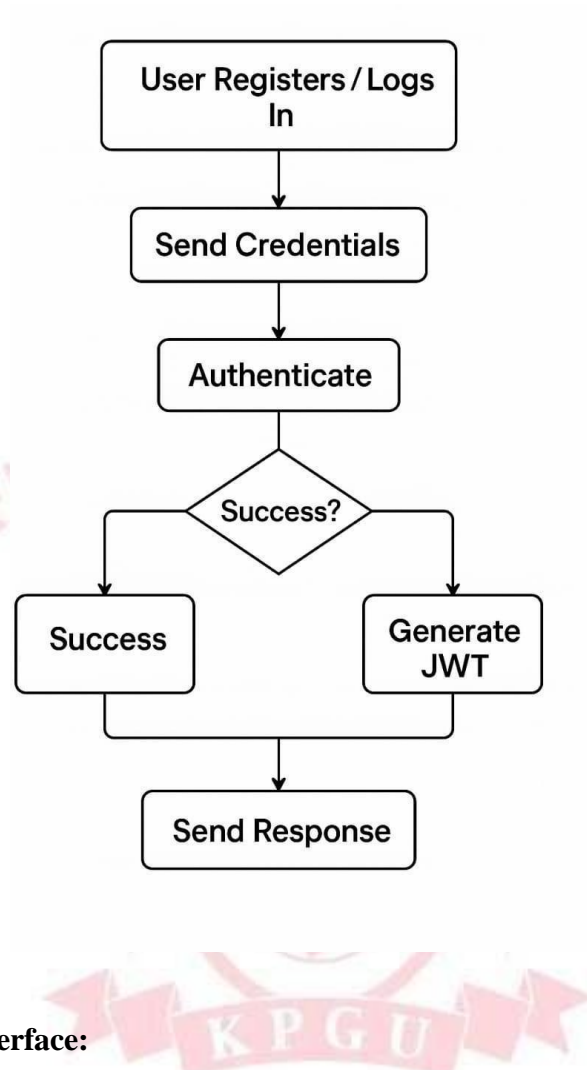
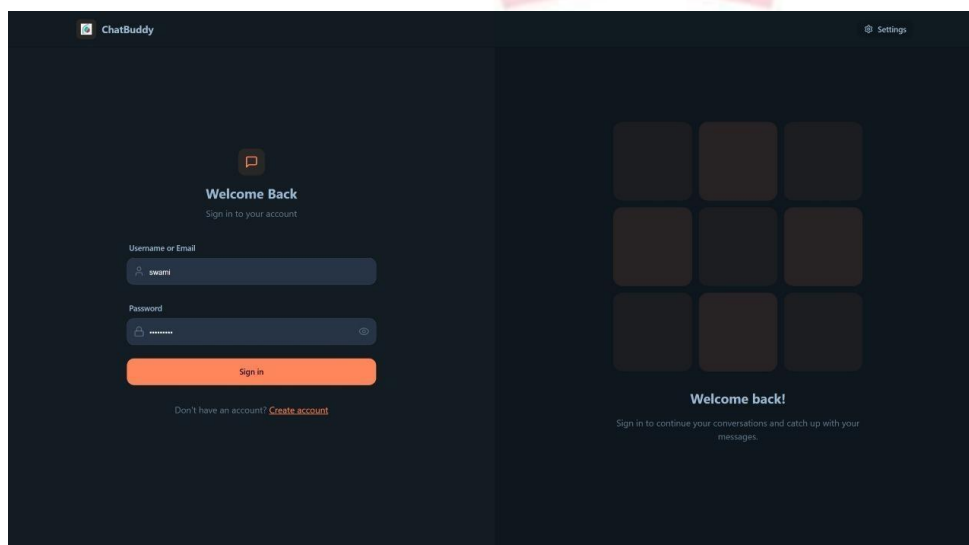
4.9 Extensibility in Design

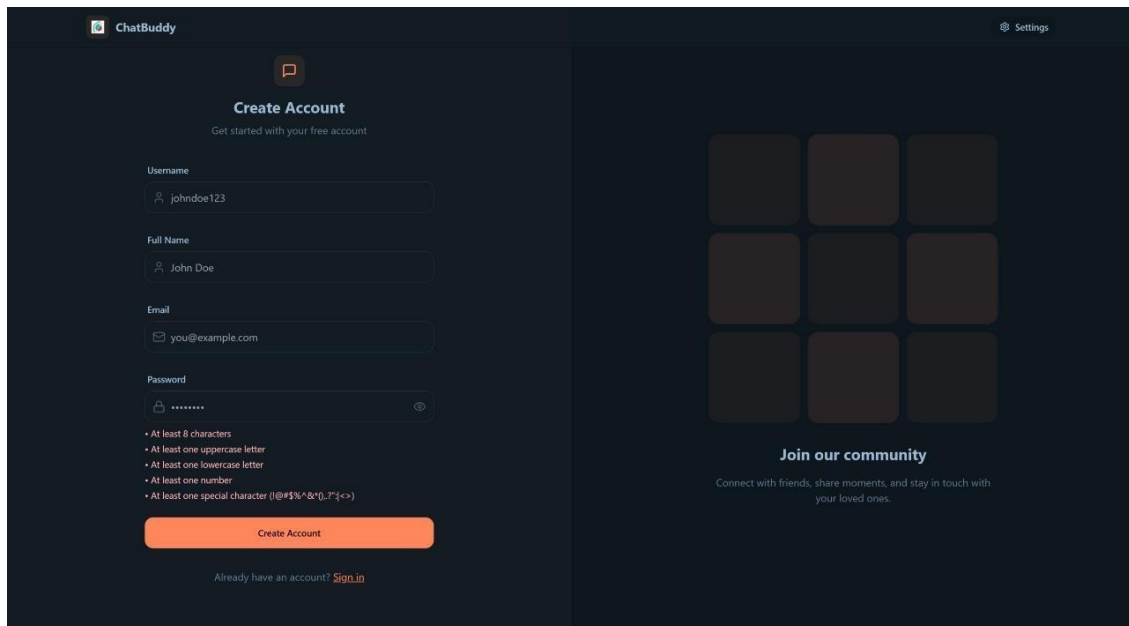
The modular system design allows future upgrades such as:

- Group chat

- Push notifications
- Emoji reactions
- Chatbot integration

Flowchart of Chatapp:

Flowchart of JWT authentication:**Screenshots of App Interface:**



The image shows the 'Create Account' form for ChatBuddy. The form is titled 'Create Account' with a subtitle 'Get started with your free account'. It contains four input fields: 'Username' (with placeholder 'johndoe123'), 'Full Name' (with placeholder 'John Doe'), 'Email' (with placeholder 'you@example.com'), and 'Password' (with a strength indicator). Below the password field, there are four bullet points listing password requirements: 'At least 8 characters', 'At least one uppercase letter', 'At least one lowercase letter', 'At least one number', and 'At least one special character (!@#\$%^&*()_~`{<=>)'. A red 'Create Account' button is at the bottom. A link 'Sign in' is provided for users who already have an account. To the right of the form, there is a section titled 'Join our community' with a subtitle 'Connect with friends, share moments, and stay in touch with your loved ones.' and a grid of nine placeholder images for user avatars.

ChatBuddy

Settings

Create Account

Get started with your free account

Username

johndoe123

Full Name

John Doe

Email

you@example.com

Password

• At least 8 characters

• At least one uppercase letter

• At least one lowercase letter

• At least one number

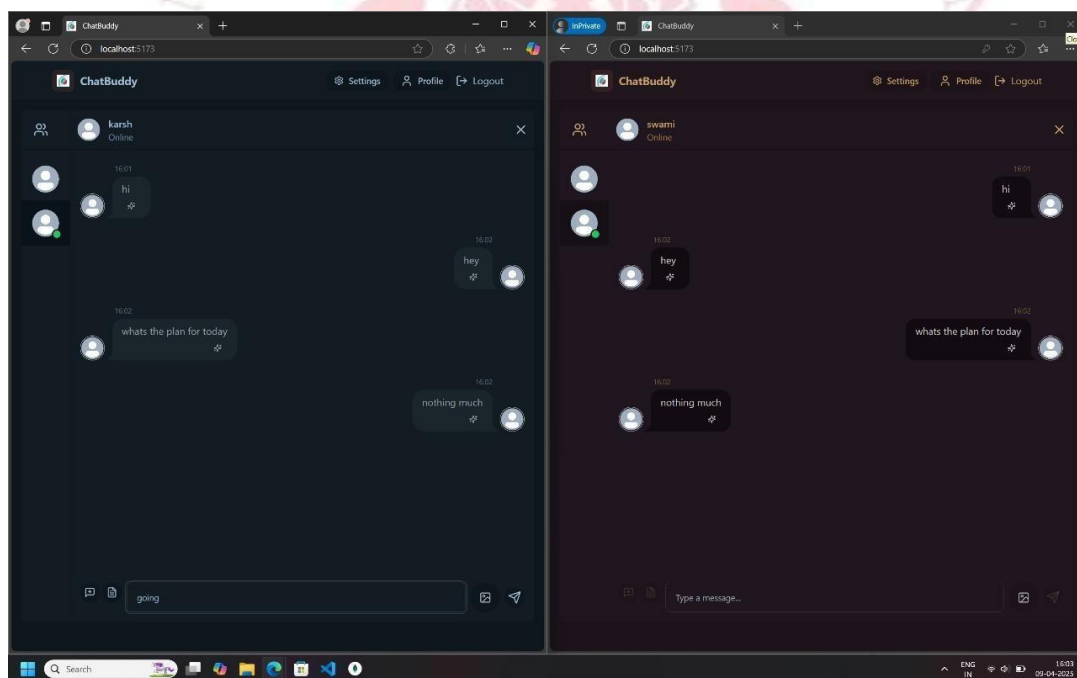
• At least one special character (!@#\$%^&*()_~`{<=>)

Create Account

Already have an account? [Sign in](#)

Join our community

Connect with friends, share moments, and stay in touch with your loved ones.



Database Screenshots:

The screenshot shows the MongoDB Atlas interface for the 'Chatapp' database. The left sidebar contains navigation options: Overview, DATABASE, Clusters, SERVICES, Atlas Search, Stream Processing, Triggers, Migration, Data Federation, SECURITY, Quickstart, Backup, Database Access, Network Access, Advanced, and Goto. The main panel displays the 'Chatapp.users' collection. At the top, it shows metadata: STORAGE SIZE: 36KB, LOGICAL DATA SIZE: 492B, TOTAL DOCUMENTS: 3, INDEXES TOTAL SIZE: 108KB. Below this are tabs for Find, Indexes, Schema, Anti-Patterns, Aggregation, and Search Indexes. A search bar is present with the text 'Type a query: { field: 'value' }'. The main area displays two document snippets in JSON format:

```
{
  "_id": ObjectId("67f613a1428be381ea80d9a6"),
  "username": "swami",
  "email": "swamigmail.com",
  "fullName": "swami",
  "password": "52a518976c3u0k1kEaLuzm2Rw4Cnuw9JK70/bTm2TD56d50318kyu6r p5EUC",
  "profilePic": "",
  "createdAt": 2025-04-09T06:26:49.938+00:00,
  "updatedAt": 2025-04-09T06:26:49.938+00:00
}
```

```
{
  "_id": ObjectId("67f64c85d97ad57a5a8ab822"),
  "username": "karsh",
  "email": "karsh@gmail.com",
  "fullName": "karsh",
  "password": "52a51891mhr5XURCm09.qA4Qz8Bie92Iegad9uF3KERYr3Qkji/1byGv5yC",
  "profilePic": "",
  "createdAt": 2025-04-09T10:31:33.886+00:00,
  "updatedAt": 2025-04-09T10:31:33.886+00:00
}
```

At the bottom, the system status is 'All Good' and the copyright notice is '©2025 MongoDB, Inc. Status Terms Privacy Atlas Blog Contact Sales'.

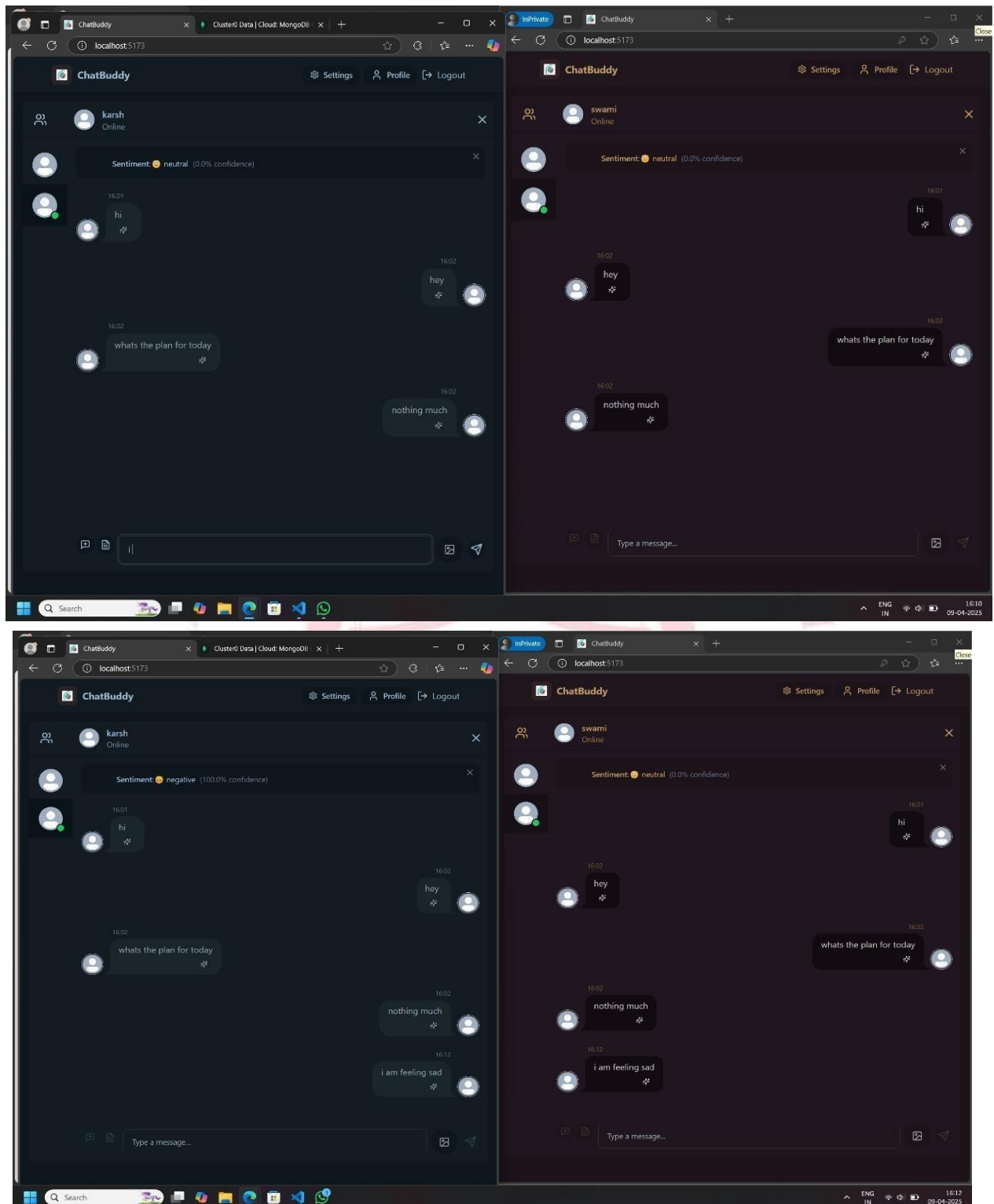
The screenshot shows the MongoDB Atlas interface for the 'Chatapp' database, specifically the 'Chatapp.messages' collection. The left sidebar is identical to the previous screenshot. The main panel displays the 'Chatapp.messages' collection with metadata: STORAGE SIZE: 36KB, LOGICAL DATA SIZE: 17KB, TOTAL DOCUMENTS: 9, INDEXES TOTAL SIZE: 36KB. Below this are tabs for Find, Indexes, Schema, Anti-Patterns, Aggregation, and Search Indexes. A search bar is present with the text 'Type a query: { field: 'value' }'. The main area displays three document snippets in JSON format:

```
{
  "_id": ObjectId("67f6147c428be381ea80d9be"),
  "senderId": ObjectId("67f613fa428be381ea80d9a9"),
  "receiverId": ObjectId("67f613a1428be381ea80d9a6"),
  "text": "hi",
  "createdAt": 2025-04-09T06:32:28.397+00:00,
  "updatedAt": 2025-04-09T06:32:28.397+00:00
}
```

```
{
  "_id": ObjectId("67f6192a428be381ea80da08"),
  "senderId": ObjectId("67f613fa428be381ea80d9a9"),
  "receiverId": ObjectId("67f613a1428be381ea80d9a6"),
  "text": "hey",
  "createdAt": 2025-04-09T06:52:26.240+00:00,
  "updatedAt": 2025-04-09T06:52:26.240+00:00
}
```

```
{
  "_id": ObjectId("67f6192c428be381ea80da0a"),
  "senderId": ObjectId("67f613fa428be381ea80d9a9"),
  "receiverId": ObjectId("67f613a1428be381ea80d9a6"),
  "text": "hey",
  "createdAt": 2025-04-09T06:52:28.457+00:00
}
```

At the bottom, the system status is 'All Good' and the copyright notice is '©2025 MongoDB, Inc. Status Terms Privacy Atlas Blog Contact Sales'.

Sentiment analysis of chat:

Chapter 5: Conclusion and Discussion

5.1 Conclusion

This project successfully demonstrates the development of a real-time chat application using modern full-stack web development technologies—namely, the MERN stack (MongoDB, Express.js, React.js, Node.js) combined with Socket.IO for real-time, bidirectional communication.

The project fulfills its initial objectives of enabling user authentication, seamless messaging, real-time updates, and persistent storage. By integrating all parts of the stack efficiently, the application showcases the potential of open-source technologies to build high-performance, interactive, and scalable communication platforms.

The system's core architecture supports real-time data transfer with low latency, secure user management through JWT-based authentication, and an intuitive frontend UI for smooth user experience. In addition, the application is responsive across different screen sizes and devices, making it accessible and versatile.

This chat app not only meets functional expectations but also sets the groundwork for more complex features such as media sharing, group messaging, voice/video integration, and chatbot services. Its modularity and clean architecture allow for easy future enhancements.

5.2 Discussion

1. Educational and Technical Impact

This project serves as a practical learning tool for students and developers seeking to understand:

- How full-stack development works in a real-world scenario.
- The implementation of real-time data exchange using WebSockets.
- The integration of frontend and backend systems for complete application flow.
- The security implications of user authentication and data handling.

Working on this project strengthens key skills in React.js development, RESTful API creation, backend server logic, WebSocket communication, and database design using MongoDB. It also introduces DevOps elements like deployment strategies and environment management.

2. Strengths of the System

- Real-time communication is efficiently implemented using Socket.IO.
- Authentication and security using JWT and password hashing.
- Scalability and extensibility due to modular coding practices.
- Cross-platform compatibility and responsive design.

- Open-source flexibility for customization and deployment.

3. Limitations

While the system achieves its core objectives, there are certain areas that can be enhanced in the future:

- Currently supports only one-on-one messaging.
- No support for media attachments or emojis.
- Lacks an admin panel or content moderation tools.
- No push notification system implemented yet.
- No chat encryption beyond secure transmission—end-to-end encryption is a future consideration.

4. Future Scope

Given the foundational work done, this project can be further developed into a more robust communication platform. Possible future enhancements include:

- Group chats and channels
- Voice and video calling using WebRTC
- AI-based chatbot integration