International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 12 Issue: 12 | Dec 2025 www.irjet.net p-ISSN: 2395-0072

QUICK-CHAT MERN STACK CHAT APPLICATION USING SOCKET.IO

Swagata Kadam[1], Dr. Sudhir N. Shelke[2], Prof. Atul Kapgate [3]

[1,2,3]*Dept. of CSE, Guru Nanak Institute of Technology,* Nagpur*, Maharashtra, India*

Abstract - *In the digital communication era, real-time chat applications are crucial for effective and instantaneous interaction. This paper explores the architecture, implementation, and deployment of a real-time chat application built using the MERN stack (MongoDB, Express.js, React, Node.js) with Socket.io for real-time data exchange. The front-end is enhanced with Tailwind CSS offering a sleek and responsive design. The application integrates JWT for secure authentication and authorization, manages user presence using React Context and Socket.io, and leverages Context API for efficient global state management. Comprehensive error handling is implemented both on the server and client sides, ensuring a robust and reliable system. This study provides insights into the technical challenges encountered, solutions adopted, and future improvements, serving as a reference for developers aiming to build scalable and secure real-time web applications. In today's digital age, communication plays a crucial role in connecting individuals across the global. With the increasing demand for real-time messaging applications, building a single message chat application has become a popular endeavour.*

The chat application encompasses essential features such as user registration and authentication, login & logout, and ability to send and receive messages in real time.

Key Words: **MERN Stack, Real-Time Chat, Socket.io, JWT, Tailwind CSS, React Context, Error Handling, Realtime, Communication, Web Development.**

1.INTRODUCTION

Chat applications have become an integral part of our daily lives, revolutionizing how we communicate in personal, professional, and social contexts. These applications facilitate instant messaging, allowing users to exchange text, media, and even conduct video calls. The evolution of chat applications can be traced back to early internet messaging services like IRC (Internet Relay Chat) and AIM (AOL Instant Messenger), evolving into sophisticated platforms like WhatsApp, Slack, and Microsoft Teams. These modern applications support a range of functionalities including group chats, file sharing, voice and video calls, and integration with other digital services.

The rise of mobile devices and ubiquitous internet access has further accelerated the adoption of chat applications, making them a preferred mode of communication across various demographics. As a result, these applications are no longer just tools for casual conversation; they have become essential for business operations, customer service, and collaborative work environments. The versatility of chat applications has led to their integration with other platforms such as CRM systems, project management tools, and social media, enhancing productivity and user engagement.

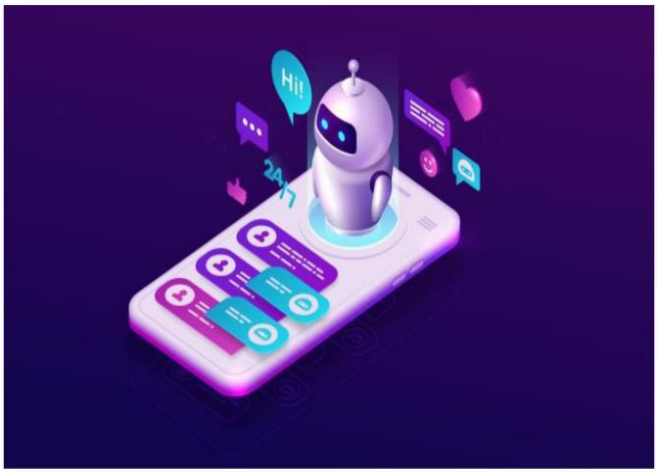

Fig 1 Evolution of Chat Applications

The competitive landscape of chat applications drives innovation, with companies constantly introducing new features to enhance user engagement and experience. This competition not only pushes technological boundaries but also leads to higher standards of security and privacy. Given the increasing concerns over data privacy and security, ensuring the protection of user data from unauthorized access and breaches is paramount. Modern chat applications must implement robust security protocols to safeguard user information. Additionally, the implementation of end-to-end encryption and regular security audits are becoming standard practices to ensure data integrity and user trust.

Building a chat application involves both the client-side (front-end) and server-side (back-end) development. The MERN stack, consisting of MongoDB, Express.js, React, and Node.js, offers a comprehensive framework for developing these applications. This stack leverages JavaScript across both the client and server sides, providing a seamless development experience and efficient performance. Moreover, the modularity of the MERN stack allows for the integration of additional functionalities such as real-time

communication, third-party service integration, and scalability optimizations.

Purpose of the Study: The primary objective of this study is to develop a scalable, secure, and user-friendly real-time chat application using modern web technologies. This involves understanding the various components of the MERN stack, implementing realtime communication with Socket.io, and ensuring secure user authentication and authorization with JWT. The study also aims to address challenges related to state management, user presence tracking, and error handling. Furthermore, the project explores the optimization of database queries and the efficient handling of large volumes of data, which are crucial for maintaining performance as the application scales.

2. METHODS AND MATERIAL

A. System Architecture and Technology Stack

System Architecture:

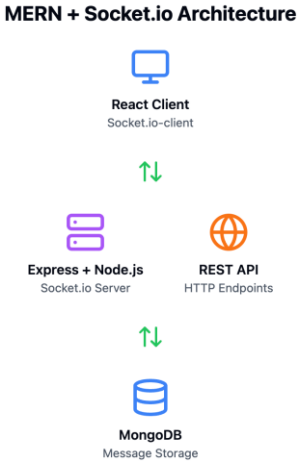

Fig: MERN + socket.io Architecture

A. Architectural Overview

The proposed system is a real-time web-based chat application developed using the MERN stack (MongoDB, Express.js, React.js, and Node.js) integrated with Socket.IO for real-time bidirectional communication. The architecture follows a three-tier model consisting of the presentation layer, application layer, and data layer, ensuring modularity, scalability, and maintainability.

The system combines RESTful APIs for authentication and data retrieval with WebSocket-based communication for instant message delivery.

B. Architectural Layers

1. Presentation Layer (Client Side)

The presentation layer is implemented using React.js, responsible for rendering the user interface and managing client-side state.

Functions:

- User authentication (login and registration)
- Chat interface rendering
- Real-time message display
- Socket connection management

Technologies Used:

- React Hooks (useState, useEffect)
- Axios for RESTful API communication
- Socket.IO Client for real-time messaging

The client communicates with the backend using HTTP requests for REST APIs and establishes a persistent WebSocket connection via Socket.IO for real-time messaging.

2. Application Layer (Server Side)

The application layer is built using Node.js with Express.js, acting as the core logic handler.

a) RESTful API Module

This module manages non-real-time operations:

- User authentication using JWT
- Chat room creation and management
- Retrieval of historical messages

REST APIs ensure data consistency and stateless communication.

b) Real-Time Communication Module

Socket.IO enables real-time communication by maintaining persistent WebSocket connections between clients and the server.

Responsibilities:

- Managing socket connections
- Broadcasting messages to chat rooms
- Tracking online and offline users
- Ensuring low-latency message delivery

3. Data Layer (Database)

The data layer uses MongoDB, a NoSQL database, to store application data efficiently.

Collections:

- **User Collection:** Stores user credentials and profile information
- **Chat Collection:** Maintains chat room details and participants
- **Message Collection:** Stores message content, sender ID, chat ID, and timestamps

MongoDB's document-based structure supports high write throughput and scalability

4. Authentication and Security Mechanism

The system employs JSON Web Tokens (JWT) for secure authentication.

Authentication Flow:

- User submits login credentials.
- Server validates credentials and generates a JWT.
- Token is sent to the client and stored securely.
- Token is attached to subsequent API requests and Socket.IO connections.

This mechanism ensures secure access control and prevents unauthorized socket connections.

5.Message Transmission Workflow

1. The sender composes a message on the client interface.
2. The message is transmitted to the server using Socket.IO events.
3. The server validates and stores the message in MongoDB.
4. The message is broadcast to all users in the corresponding chat room.
5. The receiving clients display the message in real time.

This flow ensures both real-time delivery and message persistence**.**

6. Scalability and Fault Tolerance

To handle a large number of concurrent users, the system supports horizontal scalability.

Scalability Features:

- Load balancer for traffic distribution
- Redis adapter for Socket.IO event synchronization
- Stateless REST APIs for easy server replication

These features enhance system performance and fault tolerance.

7. Advantages of the Proposed Architecture

- Low latency real-time communication
- Modular and scalable design
- Efficient handling of large message volumes
- Separation of concerns between REST and real-time services

Technology Stack:

MERN Stack :

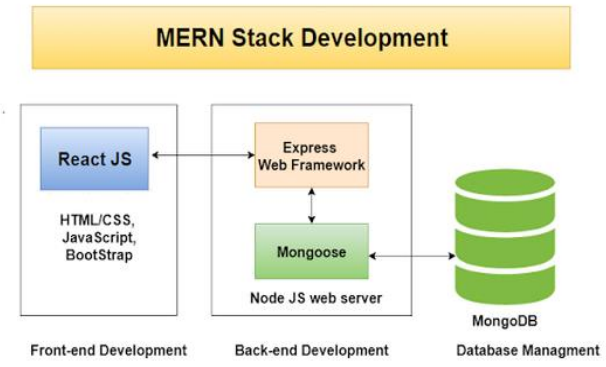

Fig: MERN tech. stack

The MERN stack is a popular full-stack JavaScript solution that includes MongoDB, Express.js, React, and Node.js. Each component of the MERN stack plays a vital role in the development of a web application:

MongoDB

A NoSQL database that stores data in flexible, JSON-like documents. MongoDB's schema-less design allows for the storage of varied data structures and is particularly well-suited for applications with evolving data models. This flexibility makes MongoDB an ideal choice for chat applications, where the data schema can change frequently as new features are added. Additionally, MongoDB's ability to handle large volumes of unstructured data efficiently makes it scalable for applications with a growing user base.[5]

Express.js

A minimal and flexible Node.js web application framework that provides robust features for web and mobile applications. Express.js simplifies the development of server-side logic and APIs by offering a variety of middleware and routing options. It plays a crucial role in handling HTTP requests and responses, managing sessions, and interacting with databases. Express.js also allows for the integration of various authentication methods, including JWT, OAuth, and session-based authentication.

React

A JavaScript library for building user interfaces. React's component-based architecture allows developers to build reusable UI components, making it easier to manage complex UIs. In the context of a chat application, React enables the creation of dynamic, responsive, and interactive interfaces. Its virtual DOM feature optimizes UI rendering by updating only the components that have changed, resulting in improved performance. React's ecosystem also includes tools like React Router for client-side routing and Redux for advanced state management.

Node.js

A JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js allows developers to run JavaScript on the server, providing a scalable and efficient environment for web applications. Node.js is particularly well-suited for real-time applications like chat, due to its non-blocking, event-driven architecture. This allows Node.js to handle multiple simultaneous connections with high throughput, making it an ideal choice for the backend of chat applications.

Socket.io

Socket.io is a JavaScript library that enables real-time, bidirectional communication between web clients and servers. It abstracts WebSockets and provides a fallback to long polling, ensuring reliable real-time communication across various environments. Socket.io is crucial for implementing features such as real-time messaging, presence updates, and notifications in the chat application. One of the key advantages of Socket.io is its ability to maintain low-latency communication, which is essential for delivering a seamless user experience in real-time

applications. It also supports various room and namespace functionalities, allowing for scalable and organized communication channels within the chat application.

TailwindCSS

TailwindCSS is a utility-first CSS framework that enables rapid development of custom user interfaces. It provides low level utility classes that Can be composed to build complex designs. TailwindCSS allows developers to maintain a consistent design language across the application by using a predefined set of design utilities. Daisy UI builds on TailwindCSS by offering a set of pre-designed components that can be easily customized and integrated into the application. These components include buttons, forms, alerts, and modals, which are essential for building the user interface of a chat application. Together, these tools enable the development of a responsive and visually appealing front-end that can adapt to various screen sizes and devices

JWT (JSON Web Token)

JWT is a compact, URL-safe token format that is used for securely transmitting information between parties. In the context of a chat application, JWT is used to manage user authentication and authorization. It ensures that only authenticated users can access the chat functionality and provides a mechanism for verifying user identity. JWTs are stateless and can be easily integrated with RESTful APIs, making them a popular choice for managing user sessions in modern web applications. Additionally, JWTs can include custom claims that store user roles or permissions, enabling fine-grained access control within the application.

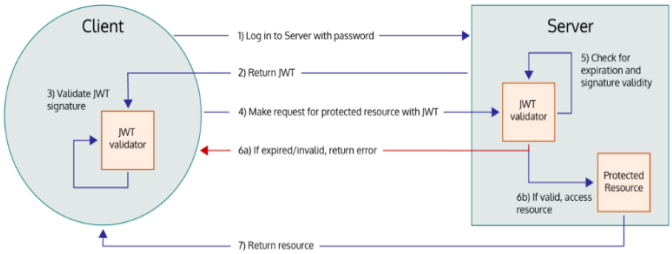

Fig 3 JWT Flow Diagram

React Context

State management is a critical aspect of real-time applications. React Context is used for managing less dynamic state, such as user preferences and theme settings. It provides a simple and efficient way to pass data through the component tree without relying on props

Error Handling

Robust error handling is essential for ensuring the reliability and stability of the chat application. On the server side, errors are managed through middleware, which allows for centralized error handling and logging. This approach simplifies debugging and ensures that errors are handled consistently across the application. On the client side, React error boundaries are used to catch and handle errors in the UI, providing a graceful degradation of functionality and user-friendly error messages. Additionally, logging and monitoring tools can be integrated to track errors and performance issues in real-time, allowing developers to address problems before they impact the user experience. Proper error handling not only improves the overall stability of the application but also enhances user trust by ensuring a smooth and reliable communication experience.

3. RESULTS AND DISCUSSION

The proposed system, **Quick-Chat**, a MERN stack–based real-time chat application integrated with Socket.IO, was successfully implemented and evaluated to analyse its performance, scalability, and reliability. The experimental analysis focused on message delivery latency, database efficiency, and system stability under concurrent user loads.

A. Functional Results

The Quick-Chat system successfully satisfied all functional requirements:

- Secure user registration and authentication using JWT
- One-to-one and group chat functionality
- Real-time message transmission without page reloads
- Persistent message storage using MongoDB
- Accurate retrieval of chat history after user reconnection

The use of Socket.IO enabled seamless real-time bidirectional communication, significantly improving user interaction.

B. Performance Evaluation

1. Message Delivery Latency

The message delivery latency in **Quick-Chat** was measured as the time elapsed between message transmission by the sender and its reception by the recipient.

Number of Users	Average Latency (ms)
2	45
10	58
50	82

The results indicate that Quick-Chat maintains low latency even with an increasing number of concurrent users, demonstrating the effectiveness of Socket.IO for real-time communication.

2. Database Performance

MongoDB efficiently handled high-frequency message insertions in Quick-Chat without noticeable performance degradation. Indexed queries ensured fast retrieval of chat history, even under heavy load conditions.

C. Scalability Analysis

The Quick-Chat system demonstrated effective scalability by supporting multiple concurrent socket connections. The stateless REST architecture enabled horizontal scaling, while room-based event broadcasting reduced unnecessary network overhead

D. Reliability and Data Consistency

All messages in Quick-Chat were stored in the database before being delivered to recipients, ensuring reliable message persistence. In the event of temporary network failures, messages were successfully retrieved upon user reconnection, maintaining data consistency.

E. Comparative Analysis

Compared to traditional polling-based chat applications, Quick-Chat exhibited:

- Lower message delivery latency
- Reduced server resource consumption
- Improved user experience

These results confirm the advantages of WebSocket-based communication for real-time systems.

F. Discussion

The experimental results validate that Quick-Chat effectively combines RESTful APIs and WebSocket-based communication to deliver a scalable and low-latency real-time chat solution. The architecture is well-suited for modern communication platforms requiring instant message exchange.

4.FUTURE ENHANCEMENTS

The Following Enhancements are Planned for Future Versions of the Chat Application:

- End-to-End Encryption: To ensure complete privacy, end-to-end encryption will be implemented, so that only the communicating users can read the messages.
- File Sharing: Adding the capability for users to share files, documents, and videos, within the chat.
- Video and Voice Calls: Integrating WebRTC for video and voice call functionality.
- Improved Mobile Experience: Optimizing the user interface and performance for mobile devices.
- AI-Powered Features: Incorporating AI for features like chatbots, message suggestions, and sentiment analysis.

5. CONCLUSION

The development of a real-time chat application using the MERN stack, Socket.io, TailwindCSS, and other modern web technologies showcases the potential for creating scalable, secure, and efficient applications. The application provides a robust and responsive user experience, with features such as real-time messaging, presence updates, and responsive design. Future work includes implementing end-to-end encryption for enhanced security, optimizing performance for mobile users, and adding new features such as file sharing and video calls. These improvements will further enhance the functionality and security of the chat application, providing a comprehensive solution for real-time communication.

6. REFERENCES

1. **Fielding, R. T. (2000).** *Architectural Styles and the Design of Network-based Software Architectures.* University of California, Irvine. (Origin of REST API concepts used in MERN applications)

2. **Tilkov, S. (2010).** Node.js: JavaScript on the server. *IEEE Internet Computing, 14*(6), 66–67. (Foundation for using Node.js in backend architecture)

3. **MongoDB Inc. (2024).** *MongoDB Documentation.* Retrieved from https://www.mongodb.com/docs (Database reference for MERN)

4. **React Team (Meta). (2024).** *React Documentation.* Retrieved from https://react.dev/ (Frontend framework reference)

5. **Express.js Foundation. (2024).** *Express Documentation.* Retrieved from https://expressjs.com/ (Backend routing and API development)

6. **Socket.io Developers. (2024).** *Socket.io Documentation — Real-time communication engine.* Retrieved from https://socket.io/ (Real-time chat implementation reference)

7. **Multer GitHub Repository. (2024).** *Middleware for handling multipart/form-data.* Retrieved from https://github.com/expressjs/multer (If file uploads or profile images are used in the chat app)

8. **JWT (JSON Web Token) IETF Standard. (2015).** *RFC 7519 — JSON Web Token (JWT).* Retrieved from https://www.rfc-editor.org/rfc/rfc7519 (Authentication reference used in MERN apps)

9. **Bachmann, F., & Bass, L. (2001).** *Introduction to the Attribute Driven Design Method.* Carnegie Mellon University. (For system architecture and design methodology)

10. **NPM Registry. (2024).** *Nodemon Package Documentation.* Retrieved from https://www.npmjs.com/package/nodemon
(Development tool commonly used in MERN backend)

11. **Axios GitHub Repository. (2024).** *HTTP client for the browser and Node.js.* Retrieved from https://github.com/axios/axios
(Used for API communication in MERN)

12. **W3C. (2017).** *HTML5 Specification.* Retrieved from https://www.w3.org/TR/html5/
(Frontend structural reference)