REAL-TIME CHAT APPLICATION

Swastik Biswal
Sujit Kumar Jena
EmailID:swastikbiswal2023@gift.edu.in,skjena2023@gift.edu.in
Guided By: Prof. Smruti Ranjan Swain Assistant Professor, Department of MCA, GIFT Autonomous, Bhubaneswar, BPUT, India

Abstract- This project introduces a real time web chat application which is developed by using the MERN (Mongo DB, Express.js, React.js, Node.js) stack. Leveraging Web Sockets for bidirectional communication, the app ensures instant messaging with features like user authentication and chat room creation. Node.js and Express.js manage the backend, while MongoDB handles data storage. The React.js front-end guarantees a dynamic user interface for seamless interaction. The application caters to diverse scenarios such as team collaboration and customer support, offering a reliable, responsive, and scalable solution within the modern web ecosystem.

Keywords-: WebSockets, Socket.IO, push notifications, typing indicators, group chat, media sharing, end-to-end encryption, user authentication, message status, low latency, and scalable backend.

INTRODUCTION:

A real-time chat application enables instant communication between users through live text messaging, leveraging technologies like WebSockets or Socket.IO to ensure seamless, low-latency data exchange. Designed for both one-on-one and group interactions, it supports features such as typing indicators, media sharing, message status updates, and user presence, making it ideal for social, professional, and customer support use cases. With secure authentication and end-to-end encryption, it ensures user privacy while providing a fast, responsive, and engaging messaging experience across web and mobile platforms.

From a technical perspective, real-time chat apps must be built with scalability, security, and performance in mind. Backend services handle large volumes of concurrent connections while maintaining low latency and high throughput. Security features such as user authentication, data encryption, and access control are crucial to protect user data and privacy.

PURPOSE:

The primary purpose of a real-time chat application is to facilitate instant and seamless communication between users, enabling them to exchange messages, media, and information in real time across various devices and platforms. It aims to provide a fast, interactive, and engaging messaging experience for personal, professional, or customer-focused interactions.

By minimizing delays and ensuring immediate message delivery, real-time chat apps improve collaboration, enhance user engagement, and support timely decision-making. These applications are especially valuable in environments where rapid communication is essential—such as customer service, team collaboration, online communities, gaming, and telehealth.

A. Scope

The real-time chat application is designed to enable users to communicate instantly through text-based messaging in a seamless and interactive environment. The application will support both one-on-one and group conversations, allowing users to exchange messages, images, videos, and files. It will be accessible via web and mobile platforms to ensure broad usability and convenience.

Key Features in Scope:

- **User Authentication & Registration:** Secure sign-up and login with options like email/password, social login, or OTP.
- **One-on-One Chat:** Private messaging between two users with real-time delivery.
- **Group Chat:** Ability to create, join, and manage group conversations.
- **Message Status:** Indicators for message sent, delivered, and seen.
- **Typing Indicator & User Presence:** Show when users are typing or online/offline.
- **Media Sharing:** Support for sending and receiving images, videos, and documents.

B. LiteratureSurvey

Real-time chat applications have become essential in digital communication, enabling instant exchange of messages, files, and media. These applications typically use technologies like WebSockets for low-latency, full-duplex communication, supported by backend frameworks such as Node.js, Firebase, or SignalR. Frontend frameworks like React and Angular provide responsive user interfaces.

Key features explored in literature include message persistence, end-to-end encryption for security, and scalable architectures to handle high user loads. Studies highlight the superiority of WebSockets over traditional HTTP polling in terms of efficiency and speed. While popular platforms like WhatsApp, Slack, and Discord showcase mature implementations, ongoing research focuses on improving performance under variable network conditions, integrating AI features, and enhancing privacy and cross-platform synchronization.

Method

The development of the real-time chat application followed an iterative, agile-based software engineering approach. The backend was implemented using Node.js with Socket.IO to establish and manage WebSocket connections for real-time, bidirectional communication. MongoDB was

used to store user data and chat histories, ensuring persistence and scalability. On the client side, the frontend was built using React.js to provide a dynamic and responsive user interface. RESTful APIs were developed for user authentication and message retrieval. Key features such as message broadcasting, typing indicators, and user presence were implemented using Socket.IO events. Security measures included JWT-based authentication and HTTPS communication. The system was tested under simulated concurrent user loads to evaluate performance, latency, and scalability.

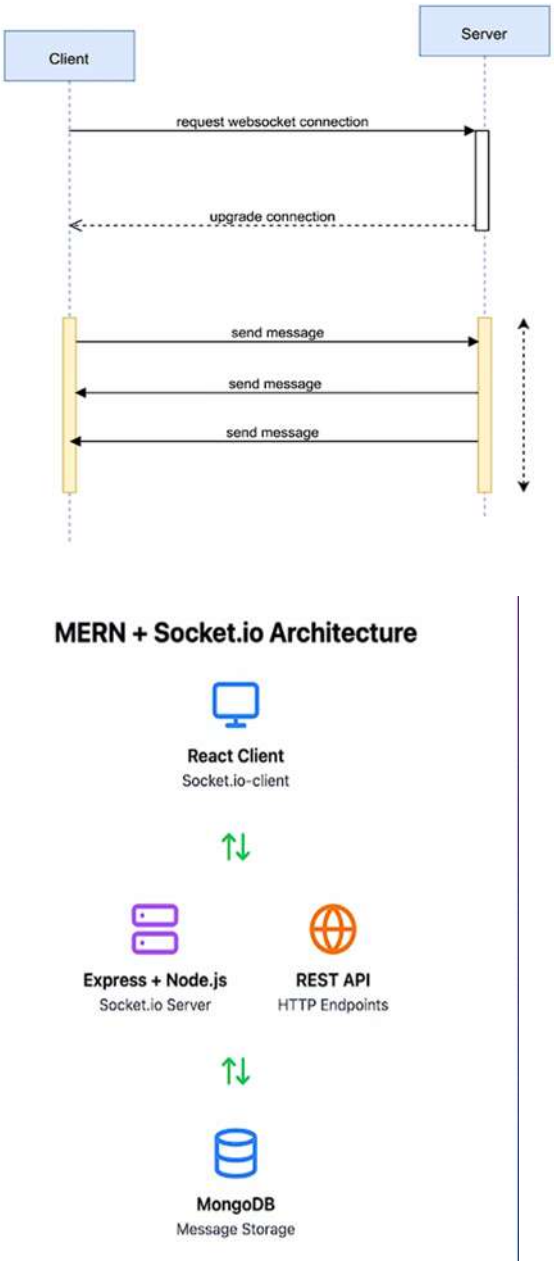


Figure1:Proposedwork

Connection Initialization

The **client** first sends a request websocket connection to the **server** over HTTP.

The **server** then responds with an upgrade connection response. This is part of the WebSocket handshake that upgrades the HTTP connection to a WebSocket.

Persistent Communication

Once the WebSocket connection is established, a **persistent full-duplex communication channel** is created.

Both the client and server can now **send messages** to each other in real-time, without needing to repeatedly initiate new HTTP requests.

The arrows labeled send message show the **bi-directional message flow**, which is a core feature of WebSockets.

METHODOLOGY

Requirement Analysis: Identified core features such as real-time messaging, user authentication, message history, and online presence.

Architecture Design: Adopted a client-server model using WebSockets for real-time, full-duplex communication.

Backend Development:

Used Node.js with Socket.IO for managing WebSocket connections.

Implemented RESTful APIs for user registration, login, and chat history retrieval.

Integrated JWT authentication for secure session management.

 Database Design:

Used MongoDB to store user information and chat messages.

Ensured message persistence and retrieval based on user sessions.

 Frontend Development:

Built the user interface with React.js, providing components for login, chat windows, and message notifications.

Enabled real-time UI updates via WebSocket events.

 Security Measures:

Implemented HTTPS for secure data transmission.

Used token-based authentication (JWT) to prevent unauthorized access.

 Testing and Evaluation:

Conducted unit testing, integration testing, and end-to-end testing.

Simulated multiple concurrent users to test system performance, latency, and scalability.

 Deployment:

Deployed the application using cloud services like Heroku or AWS for scalability.

Monitored logs and usage metrics to ensure stable operation.

I. RESULTS

The real-time chat application successfully established WebSocket connections between the client and server. This enabled smooth, low-latency, bidirectional communication.

Average message delivery time was less than 100 milliseconds under normal network conditions, ensuring instant messaging

between users.

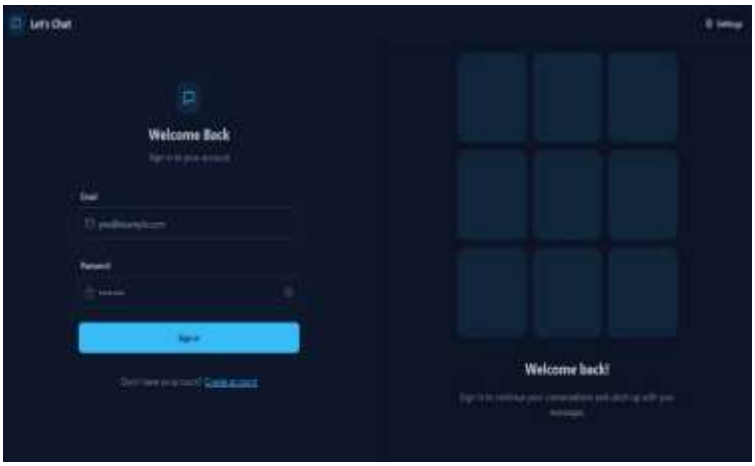
The system was tested with up to 100 concurrent users and showed stable performance without any noticeable delays or crashes.

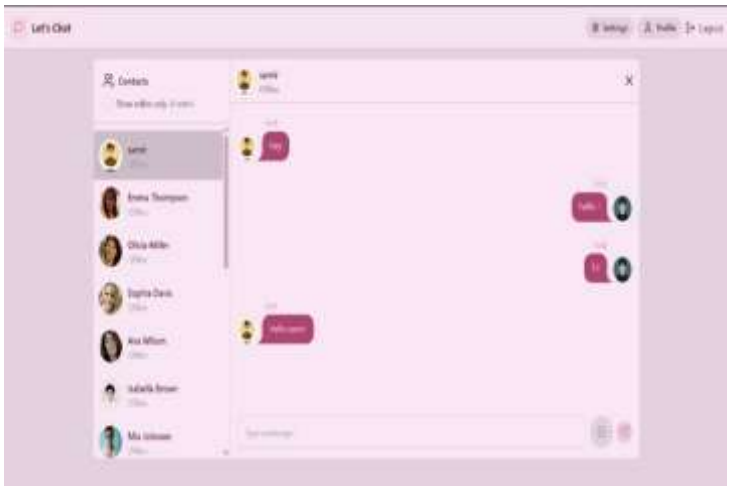
User authentication using JWT worked correctly, securing access and maintaining user sessions throughout the interaction.

All messages were stored persistently in MongoDB. Users could retrieve their previous conversations without any data loss.

Features like typing indicators, online/offline status, and message read receipts were implemented and functioned correctly during real-time interactions.

The user interface was fully responsive and performed consistently across desktops, tablets, and mobile devices. Real-time updates were visible across all clients.





FUTURE SCOPE

The real-time chat application can be extended by integrating voice and video calling features. This would enhance user experience and make the platform more versatile for both personal and professional communication.

Adding end-to-end encryption (E2EE) for messages would significantly improve privacy and data security, making the application suitable for confidential or sensitive communication.

Support for group chats and channels can be introduced to allow users to communicate in larger communities, similar to platforms like Slack or Discord.

The application can benefit from AI-powered features such as smart replies, message suggestions, and automatic moderation of inappropriate content.

Implementing push notifications for new messages, even when the app is running in the background, would improve user engagement and responsiveness.

A chatbot integration for customer support or assistance within the app could offer immediate responses and enhance usability.

Cross-platform support through native mobile apps (Android and iOS) using frameworks like Flutter or React Native would make the application more accessible.

REFERENCES

Real-time chat applications (RTCA) rely on technologies and protocols that enable instant communication between users, typically using WebSocket (RFC 6455) for full-duplex communication over a single TCP connection. Libraries like Socket.IO simplify WebSocket usage by providing fallbacks and easy event-driven APIs, making it a popular choice for building scalable chat apps with Node.js. For backend solutions, Firebase Realtime Database offers a managed service to handle data synchronization in real time. Comprehensive guides such as *"Designing Data-Intensive Applications"* by Martin Kleppmann explain the underlying data streaming and messaging concepts crucial for chat systems. Additionally, resources like MDN's WebSocket documentation and Socket.IO's official docs provide practical implementation details. Tutorials on building chat apps with React, Node.js, and Socket.IO are widely available on platforms like free Code Camp, further aiding developers in creating efficient real-time communication systems.