Real-Time ChatApp with Socket.IO

1. Introduction

Real-time communication is an essential feature in modern applications such as WhatsApp, Slack, and Discord. Traditional HTTP request-response cycles are not efficient for real-time interaction because they require frequent refreshing.

This project, **Real-Time ChatApp**, uses **Node.js**, **Express.js**, **Socket.IO**, and **MongoDB** to create a scalable, responsive, and instant messaging platform. The app allows multiple users to connect, send and receive messages instantly, and experience seamless communication.

2. Abstract

The Real-Time ChatApp demonstrates how WebSockets enable persistent, bi-directional communication between server and clients. The backend, built with Node.js + Express.js, manages socket connections, while Socket.IO ensures real-time message exchange.

The frontend developed using HTML, CSS, and JavaScript, provides a simple UI for users to interact. Additional assets like chat.png (icon/logo) and iphone.mp3 (message notification sound) improve the user experience. MongoDB integration can be used to store chat history.

3. Tools Used

- **Node.js** → JavaScript runtime environment for server-side development.
- **Express.js** → Lightweight backend framework for handling routes and APIs.
- **Socket.IO** → Enables real-time, bidirectional communication between client and server.
- MongoDB → NoSQL database for storing user data and chat history (optional).
- HTML5, CSS3, JavaScript → For building the chat interface (index.html, style.css, client.js).
- **Assets:**
    - chat.png → Used as application icon.
    - iphone.mp3 → Sound notification for new messages.

4. Steps Involved in Building the Project

Step 1: Project Setup

- Initialize Node.js project with npm init.

- Install required dependencies:
  npm install express socket.io mongoose
- Create main files: server.js, client.js, index.html, style.css.

Step 2: Backend (server.js)

- Set up Express server and integrate **Socket.IO**.
- Handle socket events (connection, chat message, disconnect).
- Broadcast messages to all connected users.
- (Optional) Save messages in **MongoDB**.

Step 3: Frontend (index.html, client.js, style.css)

- index.html: Provides chat interface with input box and message display area.
- style.css: Adds styling to make the chat UI more user-friendly.
- client.js: Connects to the server via **Socket.IO client**, sends messages, and updates DOM with new messages.

Step 4: Assets Integration

- chat.png displayed as chat icon/logo.
- iphone.mp3 played whenever a new message arrives for notification.

Step 5: Testing

- Run server: node server.js.
- Open multiple browser tabs.
- Verify messages appear in real-time across all connected clients.

5. Conclusion

The **Real-Time ChatApp with Socket.IO** successfully demonstrates instant, bidirectional communication using WebSockets. By combining **Node.js**, **Express.js**, **Socket.IO**, and **MongoDB**, the app allows multiple users to chat seamlessly in real-time.

The project also integrates UI styling (style.css), branding (chat.png), and sound notifications (iphone.mp3) for an enhanced user experience.

This application serves as a foundation for building real-world chat platforms and can be extended with **authentication, private chats** to become production-ready.

Project Submission Date : 08/09/2025

Submitted to- Elevate Labs
Submitted by- Sneh Soni