IBY Project Documentation

Software Engineer Role

Message Service Prototype (Deployed)

-By Rutam S Rajhansa
Roll No:B22ME055
Email: b22me055@iitj.ac.in
Contact: +91 9881916705
Indian Institute Of Technology, Jodhpur

Deployed Application - Link
Repository Link- Link
NOTE: For Understanding how to run the application please refer the readme file of the repository.

1. Introduction

This report outlines the system design and features of a messaging application developed using the MERN stack (MongoDB, Express.js, React.js, Node.js) integrated with Socket.io for real-time communication. The application enables users to authenticate, create chat rooms, engage in one-on-one chats, and manage profiles.

Key Features:

- **User Authentication**: Secure login and signup.
- **Chat Room Creation**: Users can create and manage chat rooms with multiple participants.
- **Individual User Chats**: Direct messaging between users.
- **User Profiles**: Editable user profiles for personalized user experiences.

2. System Design Overview

2.1. Architecture

The architecture follows a classic client-server model:

- **Client-side (Frontend)**: React.js is used to build the user interface, where users can interact with chat rooms, messages, and their profiles.

- **Server-side (Backend)**: Node.js and Express.js manage user sessions, chat requests, and message routing.
- **Database**: MongoDB stores user credentials, chat room details, messages, and profiles.
- **Real-Time Communication**: Socket.io handles bidirectional real-time communication for chat updates and notifications.

2.2. Components

1. **Frontend (React.js)**
    - **React Components**:
        - **Login/Signup Pages**: Provide forms for user authentication.
        - **Chat Room**: Displays existing chat rooms and allows users to create new ones.
        - **Chat Window**: Interface for viewing and sending messages in a real-time chat.
        - **User Profile**: Editable section for user-specific details.
    - **API Integration**: Axios or Fetch API is used for communication between React components and backend services.
2. **Backend (Node.js + Express.js)**
    - **Authentication Service**:
        - Implements user signup/login using **JWT (JSON Web Tokens)** for session management.
    - **Chat Room Service**:
        - Handles creation, deletion, and management of chat rooms, including participant lists.
    - **Message Service**:
        - Manages real-time messaging between users, including message storage and retrieval.
    - **Socket.io Integration**:
        - Implements real-time messaging and notifications for chat room creation, user joining, and message updates.
3. **Database (MongoDB)**
    - **User Collection**: Stores user credentials (hashed passwords) and profile information.
    - **Chat Room Collection**: Stores chat room metadata and participant information.
    - **Message Collection**: Stores chat messages along with timestamps and sender details.

3. Detailed System Design

3.1. User Authentication Flow

- **Signup**: Users register by providing an email and password. The password is hashed using bcrypt and stored in MongoDB. A JWT is generated upon successful registration and sent to the client.
- **Login**: The user logs in with credentials. The server validates credentials, and if successful, returns a JWT token to the client. The token is stored in local storage for session persistence.
- **Middleware**: JWT is used in middleware to authenticate and authorize requests to protected routes (chat rooms, user data, etc.).

3.2. Chat Room Creation

- Users can create new chat rooms by specifying a room name and inviting other users.
- A new chat room entry is created in the **Chat Room Collection**, with a reference to the users who joined the room.
- The room creator becomes the admin of the room and has the authority to invite others.

3.3. Real-time Chat using Socket.io

- **Real-time Communication**: Upon entering a chat room, the frontend establishes a WebSocket connection using Socket.io.
- **Message Flow**:
    1. When a user sends a message, it is emitted via the Socket.io connection to the backend.
    2. The server listens for messages and broadcasts them to all participants of the chat room.
    3. Messages are persisted in the **Message Collection** with metadata like sender, room ID, and timestamp.

3.4. One-on-One Messaging

- The system allows users to search for another user and initiate a direct conversation.
- A dedicated one-on-one chat room is created for direct messaging, which is handled similarly to group chat rooms.

3.5. User Profile Management

- Users can view and update their profiles, including details like username, email, and status.
- Profile data is stored in the **User Collection** and can be fetched and updated via the backend's API.

4. Database Design

Collections:

1. **User Collection**:
    - Fields: `userID`, `username`, `email`, `hashedPassword`, `profilePicURL`, `status`, `lastLogin`.
2. **Chat Room Collection**:
    - Fields: `roomID`, `roomName`, `participants` (array of userIDs), `adminID`, `createdAt`.
3. **Message Collection**:
    - Fields: `messageID`, `senderID`, `roomID`, `messageText`, `timestamp`.

5. Real-Time Communication Using Socket.io

Socket.io Flow:

- **Connect/Disconnect Events**:
    - When a user connects, they join a Socket.io room corresponding to the chat room ID.
    - When a user disconnects, the server tracks this and notifies other users in the room.
- **Message Handling**:
    - Messages are transmitted over the WebSocket connection, ensuring real-time updates to all users in the chat room.
    - The server also stores each message in MongoDB for future retrieval.

6. Scalability Considerations

1. **Horizontal Scaling**:

    - Multiple instances of the server can be run behind a load balancer to distribute traffic evenly. Socket.io can be scaled using adapters like Redis to handle connections across multiple servers.
2. **Database Sharding**:
    - MongoDB can be sharded to distribute large datasets (messages, chat rooms) across multiple machines to maintain performance as the user base grows.
3. **Caching**:
    - Popular chat rooms or frequently accessed data can be cached using Redis to reduce database load and improve response times.

7. Security Measures

1. **JWT Authentication**: All protected routes and user actions are authenticated using JWT tokens.
2. **Password Hashing**: Passwords are hashed with bcrypt before being stored in the database.
3. **Data Validation**: Input validation is performed on both frontend and backend to prevent malicious input.
4. **CSRF and XSS Protection**: Measures are taken on the frontend and backend to avoid cross-site scripting (XSS) and cross-site request forgery (CSRF) attacks.

8. Conclusion

This MERN-based messaging application offers a scalable, real-time communication platform with chat rooms, individual messaging, and user profile management. By leveraging Socket.io for real-time updates and MongoDB for scalable data storage, the system is well-equipped to handle multiple users and large amounts of data efficiently.