System Design Document

1. Overview

The project is a messaging service application that allows real-time chat between users. It supports user authentication, individual chat, group messaging (to be implemented later), and can be extended to include AI-powered chatbots and video/audio calls.

Core Features:

- User registration and authentication
- Real-time messaging
- Handling of message conversations
- Socket.io for live user activity tracking (commented out but planned)

2. Architecture

The architecture is based on **REST API** principles, with a **frontend-backend separation**:

- **Frontend**: Built with **React** (using Redux for state management and react-router-dom for routing).
- **Backend**: Built using **Node.js/Express** and **MongoDB** as the database.
- **Communication**: Axios is used on the frontend for making API requests to the backend.

Backend components:

- **User Management**: Registration, login, and user list retrieval.
- **Messaging Service**: Message sending and retrieval based on conversations.
- **Authentication**: JWT-based authentication middleware that ensures only authenticated users can access messages and user-related data.

3. Component Breakdown

- **Frontend (React)**:
    - **App.jsx**: Main routing and state integration.
    - **useGetMessages.jsx**: Custom hook for fetching messages for a selected user.
    - **useOtherUser.jsx**: Custom hook for fetching all other users.
    - **Redux**: Manages user, message, and socket states.
- **Backend (Node.js)**:

- **Routes**:
  - /api/v1/user: Handles user registration, login, logout, and user retrieval.
  - /api/v1/message: Handles sending and retrieving messages.
- **Models**:
  - **User Model**: Stores user details such as username, password, and profile.
  - **Message Model**: Stores individual messages.
  - **Conversation Model**: Manages conversation threads between users.

4. Database

MongoDB was chosen for its:

- **Document-based model**: Suited for storing complex data like messages and conversations.
- **Scalability**: Easy to scale horizontally for large user bases.
- **Speed**: Fast read/write for messaging systems.

5. System Workflow

1. **User Registration**: A user registers using the /register endpoint. Passwords are hashed with bcrypt, and JWT tokens are generated for session management.
2. **Login**: Upon successful login via /login, the server returns a JWT which is stored in cookies.
3. **Real-Time Messaging**: Messages are sent using the /send/:id API and stored in MongoDB under the respective conversation.
4. **Message Retrieval**: The frontend fetches messages from the backend for a selected user.
5. **Authentication**: Middleware (isAuthenticated.js) ensures users have valid tokens before accessing private routes.

Setup and Installation Guide

1. Dependencies

Backend:

- **Express.js**: Backend framework for routing and middleware management.

- **Mongoose**: MongoDB object modeling tool used for schema and data modeling.
- **JWT (jsonwebtoken)**: For user authentication.
- **Bcrypt.js**: Used for hashing passwords.
- **Cookie-parser**: Parses cookies to manage user sessions.
- **Cors**: To enable cross-origin requests between the frontend (React) and backend (Node.js).
- **dotenv**: To manage environment variables.

Frontend:

- **React.js**: Main frontend framework.
- **React-router-dom**: Enables routing between different frontend views (home, login, signup).
- **Redux**: For global state management (user and message states).
- **Axios**: For making API requests to the backend.
- **react-hot-toast**: For user-friendly notifications.

Dev Tools:

- **Tailwind CSS**: Utility-first CSS framework for styling the frontend.
- **Socket.io-client**: Used to establish WebSocket connections (though currently commented).

2. Installation Steps

1. **Backend Setup**:
    - Install Node.js if not installed.
    - Clone the repository.
    - Navigate to the backend folder.
    - Install dependencies using:

      npm install
    - Set up your .env file with the following environment variables:

      MONGO_URI=<your_mongo_db_connection_string>

      JWT_SECRET_KEY=<your_jwt_secret>

      PORT=3000

- Start the backend server:

  npm start

2. **Frontend Setup**:

- Navigate to the frontend folder.
- Install dependencies using:

  npm install

- Start the React development server:

  npm run dev

3. **Running the Full Stack**:

- Ensure MongoDB is running.
- Start both the frontend and backend servers.
- Open the frontend at http://localhost:5173.

3. Why These Tools?

- **MongoDB**: Provides a flexible schema for handling chat data where messages may vary in length and complexity.
- **Node.js/Express**: Simple yet powerful, great for creating RESTful APIs and handling asynchronous operations.
- **React/Redux**: React allows the creation of dynamic user interfaces, while Redux handles state management efficiently, especially for real-time data like user status and messaging.
- **JWT**: Secure and scalable authentication mechanism for managing user sessions.
- **Tailwind CSS**: Fast prototyping and consistent design system for the frontend.
- **Socket.io**: Enables real-time, bidirectional communication (to be integrated for live messaging).