**System Design Document**

**1. Introduction**

* **Purpose:** This document outlines the design of a real-time chat application with video and audio calling capabilities.
* **Goals:**
  + Provide a seamless and intuitive chat experience.
  + Enable one-on-one and group chat.
  + Support real-time messaging with typing indicators.
  + Ensure secure user authentication and data privacy.
  + Integrate video and audio calling features.[80% only implemented]

**2. Architecture**

* **Client-Server Architecture:** The application follows a client-server model, where clients (web or mobile) interact with a central server to send and receive messages and manage calls.
* **Frontend:** React with Chakra UI for the user interface.
* **Backend:** Node.js with Express.js for handling API requests and Socket.io for real-time communication.
* **Database:** MongoDB for storing user data, chat history, and call logs.

**3. Components**

* **Authentication Service:**
  + Handles user registration and login.
  + Uses JSON Web Tokens (JWT) for secure authentication.
* **Chat Service:**
  + Manages chat rooms and message delivery.
  + Provides real-time updates using Socket.io.
  + Supports one-on-one and group chat.
* **Video/Audio Call Service:**
  + Uses WebRTC for peer-to-peer communication.
  + Handles call signalling and negotiation.
  + Manages media streams and user interactions (mute/unmute, end call).[80% completed]

**4. Data Flow**

1. User Registration/Login: Clients send authentication requests to the backend. The backend verifies credentials and generates JWTs.
2. Chat Initiation: Clients connect to the Socket.io server and join relevant chat rooms.
3. Messaging: Clients send messages through the Socket.io server, which broadcasts them to the appropriate recipients.
4. Call Initiation: Clients send call requests through the Socket.io server. The server notifies the recipient and manages signalling.
5. Call Establishment: Clients exchange WebRTC offers and answers to establish a peer-to-peer connection.
6. Media Streaming: Clients send and receive audio/video streams directly between each other using WebRTC.

**5. Dependencies and Libraries**

* **Frontend:**
  + react: For building the user interface.
  + @chakra-ui/react: For UI components and styling.
  + socket.io-client: For real-time communication with the backend.
  + axios: For making API requests.
* **Backend:**
  + express: For creating the API server.
  + socket.io: For real-time communication with clients.
  + mongoose: For interacting with the MongoDB database.
  + jsonwebtoken: For generating and verifying JWTs.

**6. Setup and Run Instructions**

1. **Prerequisites:**
   * Node.js and npm installed.
   * MongoDB server running.
2. **Clone the repository:** git clone <repository-url>
3. **Install dependencies:**
   * Frontend: cd frontend && npm install
   * Backend: cd backend && npm install
4. **Configure environment variables:**
   * Create a .env file in the backend directory.
   * Add your MongoDB URI, JWT secret, and any other necessary configurations.
5. **Start the server:** npm start run[In the directory]
6. **Start the frontend:** npm start in the frontend directory.[Run it inside frontend folder in directory]

**7. Technology Choices**

* **React:** Chosen for its component-based architecture, flexibility, and large community support.
* **Chakra UI:** Provides a comprehensive set of accessible and customizable UI components, simplifying frontend development.
* **Node.js and Express.js:** Well-suited for building scalable and efficient backend APIs.
* **Socket.io:** Enables real-time, bidirectional communication between clients and the server, ideal for chat and call signalling.
* **MongoDB:** A NoSQL database that provides flexibility in data modelling and scalability for handling large amounts of chat data.
* **WebRTC:** An open-source technology that enables peer-to-peer communication in web browsers, making it suitable for video and audio calling.

**8. Future Improvements**

* **Enhanced call features:** Add features like screen sharing, call recording, and multi-user video calls.
* **Improved error handling:** Implement more robust error handling and logging.
* **Security enhancements:** Strengthen security measures to protect user data and privacy.
* **Mobile app development:** Develop mobile applications for iOS and Android using React Native or a similar framework.

This documentation provides a comprehensive overview of the chat application's design, setup, and technology choices. It serves as a guide for developers and stakeholders to understand the application's architecture and functionality.

**Deployment:Deployed on render -** <https://charmander-chatter.onrender.com>

**Already Created Users: 1-Ash\_New[email,password both are same]**

**2-Ash\_New1[email,password both are same]**

**3-Ash\_New3[email,password both are same]**