**System Design: Real-Time Chat Application Chat - App**

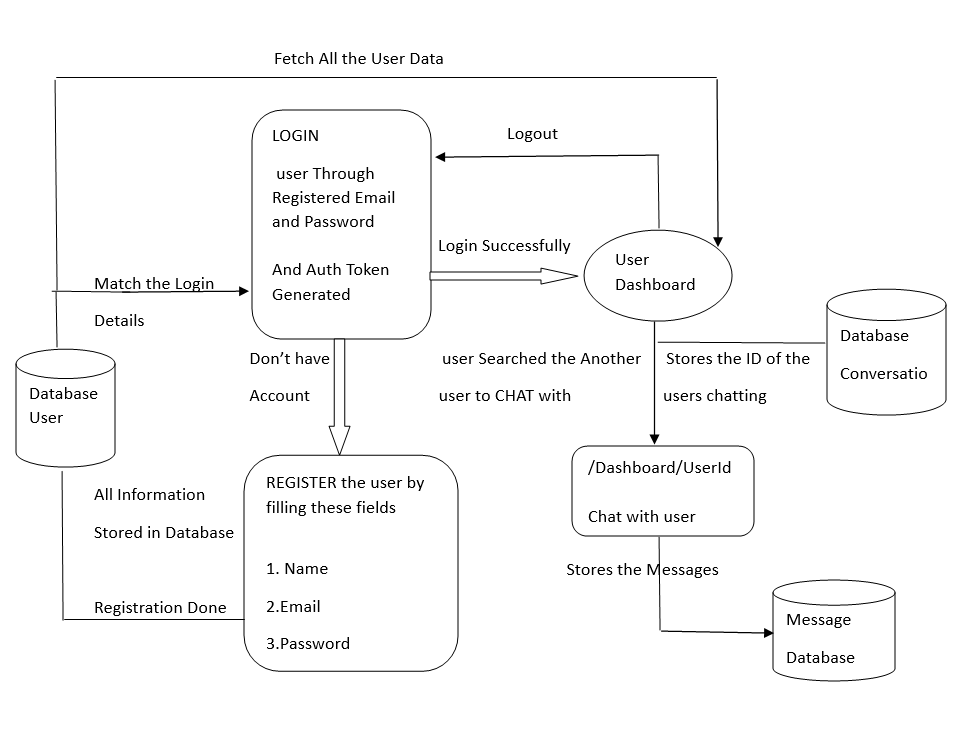
The chat application allows users to register and log in to their accounts securely. After logging in, users can search for other online users through a search box and initiate one-on-one chats. The messaging feature supports real-time text communication and enables users to share images and videos with their chat partners. To enhance user experience, the application provides notifications for unseen messages, keeping users informed of new communications. Additionally, users have the ability to update their profile information, including their username and profile picture, ensuring that their personal details are always current.

**Architecture Overview**

The system is designed using a client-server architecture where the client (frontend) communicates with the server (backend) through HTTP and WebSocket protocols.

* **Frontend**:
  + Built using **React.js**.
  + Handles user authentication, UI rendering for messages, and interacts with the backend through APIs for CRUD operations and Socket.io for real-time messaging.
* **Backend**:
  + Built using **Node.js** with **Express.js**.
  + Handles user management, real-time messaging using **Socket.io**, and performs data management with **MongoDB**.
  + Socket.io is used for real-time updates, while HTTP routes handle user authentication, login, and message history fetching.
* **Database**:
  + Uses **MongoDB** for data storage.
  + Three key collections:
    - **Users**: Stores user information.
    - **Conversations**: Stores metadata about conversations between users.
    - **Messages**: Stores individual messages (text, images, videos).
* **Cloudinary:** 
  + To Store Images and video in cloud and convert them into link

**Features**

* **One-to-One Messaging**:
  + **Feature**: Users can send messages directly to one another.
  + **Implementation**:
    - Utilize Socket.IO for real-time communication.
    - Store message history in a database, associated with the sender and recipient IDs.
* **Media Sharing**:
  + **Feature**: Users can share images and videos as part of their messages.
  + **Implementation**:
    - Allow file uploads through a frontend interface.
    - Store media files in a cloud storage service or a dedicated media server.
    - Reference media files in messages stored in the database.
* **Unseen Messages Count**:
  + **Feature**: Display the number of unseen messages for each user.
  + **Implementation**:
    - Track read/unread status of messages in the database.
    - Update unread message count in real-time using Socket.IO or periodic polling.
    - Show notifications or indicators in the user interface.
* **Profile Updates**:
  + **Feature**: Users can update their profile information (e.g., name, email, profile picture).
  + **Implementation**:
    - Provide a profile update form in the frontend.
    - Update user details in the backend and store changes in the database.
    - Ensure real-time updates using Socket.IO to reflect changes across sessions.
* Architecture of the Chat-app:
  + 

**Frontend**

* **React.js Components**:
  + **Register**: Handles the Registration of user.
  + **Login**: Handles the Login of user.
  + **Home**: Handles user login, Sidebar, and other components like the messaging area.
  + **Sidebar**: Displays user information and conversations. Manages user search and logout.
  + **MessagePage**: Displays the messages in a selected chat and handles sending text, images, and videos.
  + **Search**: Displays all the users which are online and offline and then user can select anyone to chat with.
  + **Redux** **State:** 
    - **User Information**:
      * Tracks user's \_id, name, and email.
      * Manages the token for authentication.
    - **Online Users**:
      * The onlineuser array stores a list of currently online users, useful for showing who is available in a chat app
    - **Socket.IO Connection**:
      * The socketconnection holds the Socket.IO connection for real-time messaging between users.
  + **user:** Stores user details.
  + **Onlineuser**: Stores the list of online users.
  + **Socketconnection**: Manages the Socket.IO connection.

**Backend**

* **Node.js & Express.js**:
  + **Authentication (auth.js)**:
    - Handles user signup (/createuser), login (/login), and updating user details (/updateuser).
    - Uses **JWT (JSON Web Token)** for user authentication and authorization.
    - **bcrypt** is used for password hashing.
  + **API Endpoints**:
    - * /api/auth/getuserdetails: Fetches authenticated user details.
      * /api/auth/login : Login route
      * /api/auth/register: Register route
      * /api/messages: Manages sending and receiving messages, storing them in the database
  + **Socket.io**:
    - Establishes WebSocket connections for real-time messaging.
    - Each user connects with a unique socket ID.
    - Facilitates message exchange, conversation updates, and message status (e.g., seen/unseen).
  + **Conversation & Message Handling**:
    - **Conversations** (CONV model): Tracks which users are in a conversation and stores message references.
    - **Messages** (MESS model): Stores individual messages (text, images, and videos) and tracks whether they are seen.
  + **Helper Functions**:
    - details(token): Decodes JWT to extract user details.
    - getConversation(userId): Fetches conversation details for a user.
* **Database (MongoDB)**
* **User Collection**:
  + Stores user details like \_id, name, email, password (hashed), profile picture, and status (online/offline).
* **Conversations Collection**:
  + Stores details of conversations between users. Includes participants and last message details.
* **Messages Collection**:
  + Stores chat messages between users, including message type (text, image, video), timestamps, and sender information.

**Data Flow**

Frontend:

* User Authentication:
  1. When the user logs in, a JWT token is stored in localStorage.
  2. On page load, the app fetches the user's data from /api/auth/getuserdetails using the token.
* Real-time Messaging:
  1. The frontend establishes a Socket.IO connection with the server using the JWT token.
  2. The Sidebar component fetches and displays conversations by emitting socket events (e.g., 'sidebar').
  3. Messages are sent in real-time by emitting the 'send message' event with message data.
  4. Received messages are displayed by listening for the 'message' event from the server.
* Media Upload:
  1. Users can upload images and videos using the MessagePage component.
  2. Files are uploaded via a helper function (uploadfile), which returns a URL.
  3. The message is then sent with the URL included, allowing other users to see the media in the chat.

Backend:

* User Authentication:
  1. The backend verifies JWT tokens for all protected routes.
  2. Upon successful verification, user details are returned to the frontend.
* Real-time Messaging:
  1. When a user connects, the backend establishes a Socket.IO connection and stores the socket ID.
  2. Messages are sent to the backend via the 'send message' socket event, which stores them in the MongoDB Messages collection.
  3. The backend emits the 'message' event to both the sender and receiver, ensuring real-time delivery.
* File Storage:
  1. The backend processes media files (e.g., images, videos) and stores them in cloud storage like(CLOUDINARY).
  2. The URLs for these files are saved in MongoDB and sent back to the frontend for display.

**Future Enhancements:**

* + Notifications: Add push notifications to alert users when they receive new messages, even when offline.
  + Typing Indicators: Implement real-time typing indicators using additional socket events.
  + Provide an interface for users to initiate and receive video calls.