# **Design Document: EventChirp Web and Mobile App**

## **Overview**

EventChirp is a social platform that connects users based on shared interests and facilitates the creation and discovery of events. The platform will have both web and mobile applications developed using React for the web, Flutter for mobile, and the backend will be powered by Node.js with MongoDB as the database. The web application will follow the file architecture of Next.js for optimal organization and maintainability.

## **Goals**

* **User Base:** Build a scalable user base by providing a seamless and engaging experience for users to connect, create events, and discover activities.
* **Event Creation:** Allow users to create events, providing details such as date, time, location, and category, fostering a vibrant and diverse event ecosystem.
* **3-Degree Mutual Friends Connection Detection:** Implement a feature to identify mutual friends within a 3-degree connection radius, enhancing user networking and connections.

## **Architecture**

### **Frontend:**

#### Web (React with Next.js)

* **File Architecture:**
  + Utilize Next.js file architecture, including pages, components, and styles for organized and modular development.
* **Components:**
  + User Authentication
  + Event Feed
  + Event Creation Form
  + User Profile
  + Mutual Friends Widget
* **State Management:**
  + Use React Context API or Redux for state management.
  + Handle user authentication, event data, and user profiles.
* **Routes:**
  + / - Home page displaying event feed
  + /login - User authentication and login
  + /register - User registration
  + /create-event - Form for creating events
  + /profile/:username - User profile page
  + /mutual-friends/:userId - Page displaying 3-degree mutual friends

#### Mobile (Flutter)

* **Screens:**
  + Authentication Screens
  + Event Feed
  + Create Event Screen
  + User Profile
  + Mutual Friends Screen
* **State Management:**
  + Leverage Flutter's Provider or Riverpod for state management.
  + Manage user authentication, event data, and user profiles.
* **Routes:**
  + / - Home screen displaying event feed
  + /login - User authentication and login
  + /register - User registration
  + /create-event - Screen for creating events
  + /profile/:username - User profile screen
  + /mutual-friends/:userId - Screen displaying 3-degree mutual friends

### **Backend:**

#### Node.js

* **Express Framework:**
  + Develop a RESTful API using Express for handling requests.
* **Authentication:**
  + Implement JWT (JSON Web Tokens) for secure user authentication.
* **Database (MongoDB):**
  + Use MongoDB as the primary database to store user data, event details, and friendship connections.
* **Event Management:**
  + Create endpoints for event creation, retrieval, and updates.
* **Friendship and Connection Detection:**
  + Implement algorithms for detecting 3-degree mutual friends connections.

## **Features**

### **1. User Base**

* **User Registration and Authentication:**
  + Allow users to register and log in securely.
  + Use JWT for token-based authentication.
* **User Profiles:**
  + Users can create and update their profiles with personal information and preferences.

### **2. Create Events**

* **Event Creation:**
  + Users can create events, specifying details such as date, time, location, and category.
  + Include image uploads for event cover photos.
* **Event Discovery:**
  + Implement a feed where users can discover and join events based on their interests and connections.

### **3. 3-Degree Mutual Friends Connection Detection**

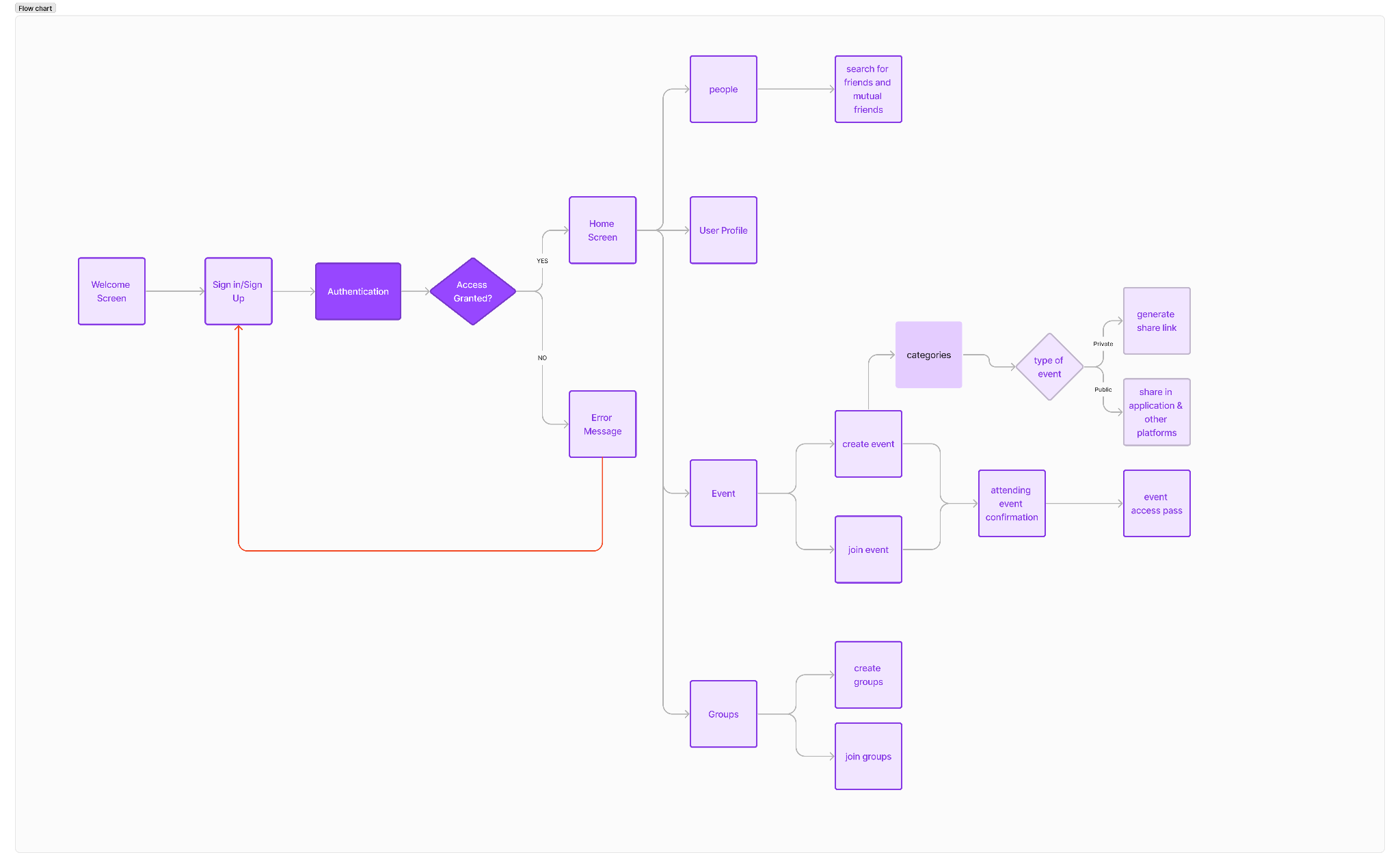
* **Friendship Network:**
  + Store user friendships in the database.
  + Implement algorithms to identify mutual friends within a 3-degree radius.
* **User Connections:**
  + Display mutual friends on user profiles and event details pages.

## **Additional Considerations**

* **Real-time Updates:**
  + Implement WebSocket communication for real-time updates on events and friend connections.
* **Notifications:**
  + Provide notification features for event updates, friend requests, and other relevant activities.
* **Scalability:**
  + Design the architecture to be scalable, considering potential growth in user base and events.
* **Security:**
  + Implement security best practices, including input validation, secure token handling, and data encryption.
* **Documentation:**
  + Maintain comprehensive documentation for developers, including API documentation and code comments.

## **Conclusion**

The EventChirp platform aims to create a vibrant and connected community by combining user-friendly interfaces with robust features for event creation and friend connections. The chosen technologies, along with the file architecture of Next.js, provide a solid foundation for scalability, performance, and maintainability. Regular updates and improvements will ensure a positive user experience and the continued growth of the platform.



# **Database Design for EventChirp**

## **Overview**

The database design for EventChirp is crucial for efficiently storing and retrieving user data, event details, and friendship connections. MongoDB is chosen as the database to leverage its flexibility and scalability.

## **Collections**

### **1. Users Collection**

#### User Document Structure:

json

{ "\_id": "ObjectId", "username": "string", "email": "string", "password": "string","friends": ["ObjectId"], "friendRequests": ["ObjectId"], "numberOfFriends": "number","myCreatedEvents": ["ObjectId"], "eventsAttended": ["ObjectId"], "numberOfEventsCreated":"number", "numberOfEventsAttended": "number", "createdAt": "timestamp", "updatedAt":"timestamp", "\_\_v": "number" }

### **2. Events Collection**

#### Event Document Structure:

json

"\_id": "ObjectId", "mainCategory": "string", "subCategory": "string", "subSubCategory":"string", "dateOfEvent": "string", "startTime": "string", "endTime": "string","location": "string", "nameOfPlace": "string", "address": "string", "maxPeople":"string", "eventMode": "string", "ageRange": "string", "gender": "string", "femaleCount":"string", "maleCount": "string", "occupation": "string", "eventTitle": "string","eventDes": "string", "coverImgUrl": "string", "Img1Url": "string", "Img2Url": "string","Img3Url": "string", "Img4Url": "string", "createdAt": "timestamp", "updatedAt":"timestamp", "\_\_v": "number" }

## **Relationships**

### **1. Friendship Relationship**

* **Friends Collection:**
  + **User1ID - User2ID Relationship:**
    - A document represents a friendship between two users.
    - Fields: \_id, user1ID, user2ID, createdAt, updatedAt

## **Indexing**

* Indexing should be applied to fields used for frequent queries, such as username and email in the Users Collection.
* Consider compound indexes for fields used together, like the dateOfEvent and location in the Events Collection.

## **Conclusion**

The database design focuses on providing a clear structure for storing user data, event details, and friendship connections. Proper indexing and relationships are implemented to optimize query performance. Regular maintenance and monitoring of the database will ensure the scalability and efficiency of the EventChirp platform.