## **Core Features to Include**

### **1. Authentication Mechanisms**

* **Email & Password Login:**Use proper validation (e.g., strong passwords, validation on both frontend and backend).
* **Phone Number & OTP Login:**Use providers like Twilio, Firebase, or other SMS APIs.
* **Social Media Logins:**Support **Google** and **Apple** login using Expo AuthSession or libraries like expo-auth-session.
* **Biometric Authentication:**Add biometric login as a second layer (FaceID/TouchID) using expo-local-authentication.
* **MPIN Authentication:**Add MPIN login as a second layer
* **Session Management:**Handle **JWT tokens** (access & refresh tokens) with automatic renewal and storage in a secure manner (e.g., SecureStore).
* **Logout:**Include a proper logout flow to clear credentials and tokens securely.

### **2. Registration Mechanisms**

* Email-based registration with **user details** (e.g., Name, Country, etc.).
* Phone number registration with OTP verification.
* Social media registration using Google/Apple accounts.

### **3. Authorization & Security**

* **Role-Based Access Control (RBAC):**Include roles such as Admin, User, Guest, etc., and ensure backend permissions align with roles.
* **Two-Factor Authentication (2FA):**Allow users to enable/disable 2FA with email, SMS, MPIN, biometric or other third party options.
* **Secure Storage:**Use expo-secure-store to store sensitive data like tokens.
* **Validation:**Validate inputs on both the client and server sides.
* **Rate Limiting:**Protect against brute-force attacks for login/OTP.

### **4. Global Configuration**

* **Toggleable Features:**Add a global configuration file to enable/disable login/registration options (e.g., allow/disable OTP login, biometrics).
* Default app settings like themes, language, and API base URLs.

### **5. UI/UX Enhancements**

* Prebuilt, customizable login and registration screens:
  + Use libraries like **React Native Paper** or **Native Base** for theming and prebuilt components.
  + Add animations for a polished user experience (e.g., react-native-reanimated).
* Responsive design for different screen sizes.
* Accessibility compliance (e.g., proper contrast, screen reader support).

### **6. API Integrations**

* Use axios or fetch with proper error handling and interceptors for JWT token management.

### **7. State Management**

* **Global State Management:**Use **React Context**, **Zustand**, or **Redux** to manage user authentication state and app-wide settings.
* Persist state using libraries like AsyncStorage or expo-secure-store.

### **8. Global Error Handling**

* **Global ErrorBoundary:**Introduce centralized error handling (e.g., global error boundary for unhandled errors).

### **9. Testing**

* Unit tests for components and utilities.
* End-to-end tests for authentication flows using **Detox, Jest** or **Cypress**.
* Mock API calls during tests.

## 

## 

## 

## 

## 

## 

## 

## 

## **Best Practices**

### **Security**

1. Always use **HTTPS** for API communication.
2. Store sensitive information (like access tokens) securely using expo-secure-store.
3. Avoid hardcoding API keys in the codebase; use .env files with libraries like react-native-dotenv.
4. Implement proper error messages to prevent information leakage.
5. Use **CSRF protection** for APIs where applicable.
6. Protecting Authenticated Routes.

### **Scalability**

1. Modularize code:
   * Separate screens, components, services (e.g., API calls), and utilities.
   * Reusable Components
   * Follow a proper folder structure:
   * Ensure extensibility to add more login methods or customize flows easily.

### **Documentation**

1. Write clear documentation for:
   * Setup instructions.
   * How to enable/disable specific features (e.g., toggle OTP login in config).
   * Adding new authentication methods.
2. Add inline comments and docstrings.

### **Performance**

1. Use lazy loading for screens and components.
2. Optimize assets with expo-asset and caching mechanisms.
3. Minimize the number of API calls by caching where applicable.

### 

### **Theming**

1. Support light and dark modes.
2. Centralize theme-related styles to make the app consistent.

### **Developer Experience**

1. Use **TypeScript** for type safety.
2. Include linting and formatting tools (e.g., ESLint, Prettier).
3. Set up **Husky** for pre-commit hooks to ensure quality checks.

## **Optional Add-ons**

* **Multilingual Support:**Use libraries like i18next or react-intl for localization.
* **Analytics:**Integrate Firebase Analytics or Amplitude for tracking user behavior.
* **Push Notifications:**Use expo-notifications for features like OTP alerts and login reminders.