Assignment: Advanced Todo App with Weather Integration

What you will do:

You will create a Todo Application with React Native with four main views:

1. **Login View:** A simple login screen with dummy data (hardcoded or from a mock backend).
2. **List View:** Display a list of tasks sorted by day. Tasks can be marked as completed or removed (either in view or in view and list).
3. **Task Management View:** Add, edit, or view task details (CRUD in a single view). Editing and creating tasks can be non-persistent and removed after a reload if needed.
4. **Profile View:** Display user information and a logout button.

Additionally:

- Implement a feature to show or hide completed tasks.
- Fetch and display the current weather for Tallinn, Estonia on the task detail view.

Detailed Requirements:

1. **User Authentication:**
    - Implement a simple dummy login as the default view (hardcoded usernames and passwords either on front-end or backend).
    - Provide two types of accounts: Admin and User.
    - Admins can create and edit tasks, while users can only view tasks.
    - Display a profile page that can be accessed via a sidebar or a button on the top/button bar.
    - The profile page should be read-only with a logout button.
2. **Todo List:**
    - Display a list of todos sorted by day in a clean and user-friendly UI (either table view or list view).
    - Allow admins to add, edit, and view tasks.
    - Users can only view tasks (list and edit/create view).
    - Tasks should include the name, description, and location (default to Estonia).
    - Provide options to mark tasks as completed or remove them using swipes or selection, or from edit view (only one option could be implemented).
    - By default, show only non-completed tasks. Include an option to show all tasks.
3. **Weather Integration:**
    - Display the current weather for the task's location.
    - Use a public API to fetch weather data (e.g., OpenWeatherMap for example, could choose any).
    - Show temperature in Celsius. (Optional: Add a switch to toggle between Celsius and Fahrenheit, with user preference stored locally).
    - Cache weather requests for a configurable duration (e.g., 30 minutes for production, 30 seconds for testing – optional feature, but great to have that you won’t consume a lot of free API requests).

    - Don’t include API key directly into code – small tip, enviroments or just provided it separetly ☺
4. **Backend:**
    - If familiar with Express or Node.js, create a simple CRUD server with static data and a dummy login.
        - Use two accounts: Admin and User.
    - If not familiar with backend development or feel like you won’t have enough time, use json-server and json-server-auth for mock endpoints and authentication.
        - Provide endpoints for todos and users.
5. **Optional Enhancements:**
    - Allow location changes for tasks with map integration.
    - Cache weather requests manually or with query tools to avoid frequent API calls (good to have).
    - Store user preferences such as temperature format locally using local storage or a local database.
    - Ensure that the UI/UX is intuitive and visually appealing.

Task Breakdown:

1. **Setup and Authentication:**
    - Set up the React Native environment.
    - Implement the dummy login screen.
    - Create the profile page with a logout button.
2. **Todo List Management:**
    - Develop the UI for displaying tasks sorted by day.
    - Implement functionality to add, edit, and view tasks for admins.
    - Integrate options to mark tasks as completed or remove them.
    - Include an option to toggle the visibility of completed tasks.
3. **Weather Integration:**
    - Fetch weather data from a public API.
    - Display the current temperature in the task view.
    - Implement caching for weather requests (optional: you could do a single request only on a task creation and save it into the apps storage)
4. **Backend Integration:**
    - Set up a simple Express/Node.js server with CRUD operations (if applicable).
    - Alternatively, configure json-server with json-server-auth.
5. **Optional Features:**
    - Add the ability to change task locations with map integration.
    - Implement a switch to toggle temperature units and store user preferences.

Mandatory Libraries and Tools:

- **React Native:** Framework for building the mobile app.
- **Node.js (v20 or later):** JavaScript runtime environment.
- **Express.js:** For setting up the backend server (if applicable) **or json-server (or any other mock server):** For a mock backend (**json-server-auth:** For mock authentication).
- **Axios or Fetch:** For making API requests.
- **React Navigation:** For handling navigation within the app (or similar).

- **React Native Supported UI components** (shadcn/ui, NativeWind, r eact-native-reusables, • React Native Paper or similar).
- **S**tate management (any for example Redux or global states).
- **Moment.js, Date-fns or Day.js:** For handling dates.

Optional Libraries and Tools (suitable options/examples):

- **AsyncStorage:** For storing data locally.
- **react-query or SWR:** For data fetching and caching.
- **Formik and Yup:** For form handling and validation.
- **Expo:** To simplify the development workflow.
- **React Native Gesture Handler:** For handling gestures (swipes).

Clarifications:

- **Dummy Login:** Implement a simple login screen that does not need real authentication. You can hardcode the admin and user credentials in the frontend or backend.
- **Static Data for Todos:** Use static data for the todos initially (just an initial array of todos, actullay can be used to store data there as well). You can replace this with dynamic data from the backend or json-server as you progress.
- **Weather Data:** For the weather integration, start by displaying static weather data. Integrate the API call once the static data display works correctly.
- **Profile Page:** Keep the profile page simple and read-only. You can just display hardcoded user information taken from API call.

Deliverables:

- Source code hosted on a public repository (e.g., GitHub), links provided.
- Clear instructions on how to run the application locally in README file.
- Brief documentation of implemented features and any additional configurations, if needed.

Tips:

- Focus on functionality first, then refine the UI/UX.
- Ensure code is clean and well-documented.
- Test thoroughly, especially the caching mechanism and task management features.
- Keep the user experience intuitive and responsive.

This assignment is designed to test your React Native skills, attention to detail, and ability to integrate multiple features into a cohesive application.

A screenshots of how the application should look is also provided. These are only examples; you’re free to design the UI/UX as you feel and think is best.