Comprehensive Instructions and To-Do List for React Native Task Manager App

This document provides a detailed, step-by-step guide and a comprehensive to-do list for developing a **Simple Task Manager App** using **React Native** and **Expo**. The instructions are designed to be executed by an AI assistant (like Cursor AI or ChatGPT) to ensure a perfect implementation that meets all assignment requirements, focuses on **perfect UI/UX**, and adheres to high **Code Quality** standards.

---

1. Project Setup and Initialization

Objective: Create the initial project structure using Expo, as recommended in the assignment.

Step	Action	Details
1.1	**Initialize Project**	Use `npx create-expo-app` with a suitable name (e.g., `ReactNativeTaskManager`). Select a blank template.
1.2	**Install Dependencies**	Install any necessary UI library (e.g., `react-native-paper` or `styled-components` for better UI/UX) and utility libraries. *Decision: Use a minimal, modern UI approach with standard React Native components and a clean design system.*
1.3	**Initial Cleanup**	Clean up the initial `App.js` file to prepare for the main component structure.

---

2. Core Data Structure and State Management

Objective: Define the task data structure and implement local state management as required.

Requirement: Use **local component state** to manage the tasks.

Step	Action	Details
2.1	**Define Task Structure**	Define a clear JavaScript object structure for a single task
		Structure: `{ id: string, description: string, isComplete: boolean }`
2.2	**Initialize State**	In the main component (`App.js` or a dedicated `TaskManagerScreen.js`), initialize the state using `useState` with an array of tasks. Include a few dummy tasks for initi UI development (e.g., 2 complete, 2 incomplete).
2.3	**Implement State Handlers**	Create dedicated functions to manage the state: `addTask` , `toggleTaskComplete` , `deleteTask` . These functions must be passed down as props to child components.

---

3. User Interface (UI) and User Experience (UX) Design

Objective: Implement a clean, intuitive, and responsive UI/UX that provides visual feedback.

Evaluation Criteria Focus: UI/UX Design (intuitive, responsive, good user experience)

3.1. Design System and Styling

Step	Action	Details
3.1.1	**Global Styles**	Implement a consistent color palette (e.g., a primary color, a background color, and a color for completed tasks). Use a clean, modern font stack.
3.1.2	**Layout**	Use `SafeAreaView` and consistent padding/margins for all screens. Ensure the layout is responsive on different device sizes.

3.2. Task List Component (`TaskList`)

Step	Action	Details
3.2.1	**List View**	Use `FlatList` for efficient rendering of the task list.
3.2.2	**Task Item Rendering**	Create a dedicated `TaskItem` component.
3.2.3	**Visual Distinction**	**Crucial for UX:** Visually distinguish complete tasks (e.g., strikethrough text, faded color, checkmark icon).
3.2.4	**Interaction Feedback**	Implement subtle visual feedback on tap/press (e.g., slight color change or ripple effect) for the toggle and delete actions.
3.2.5	**Empty State**	Display a friendly message and illustration/icon when the task list is empty.

3.3. Add Task Component (`AddTaskForm`)

Step	Action	Details
3.3.1	**Input Field**	Use a `TextInput` for the task description. Ensure it has a clear placeholder.
3.3.2	**Add Button**	Implement a prominent "Add" button. The button should be disabled when the input is empty.
3.3.3	**User Flow**	After a task is added, the input field should clear, and the user should receive a brief, non-intrusive confirmation (e.g., a toast message or a subtle animation).

---

4. Feature Implementation (Functionality)

Objective: Implement all required application features.

Evaluation Criteria Focus: Functionality (Add, Complete, Delete tasks)

Step	Feature	Implementation Details
4.1	**Add Task**	Implement the `addTask` handler. The new task must be added to the state array with `isComplete: false` and a unique `id` .
4.2	**Mark Task as Complete**	Implement the `toggleTaskComplete` handler. This function must find the task by `id` and flip its `isComplete` status.
4.3	**Delete Task**	Implement the `deleteTask` handler. This function must filter the task array to remove the task with the given `id` .
4.4	**Task List Display**	Ensure the `FlatList` renders **all** tasks (both complete and incomplete) as required. *UX suggestion: Optionally, display complete tasks at the bottom of the list or in a separate section for better organization, while still maintaining the requirement to "Display all tasks in a list view".*

---

5. Code Quality and Documentation

Objective: Ensure the code is clean, well-structured, and fully documented.

Evaluation Criteria Focus: Code Quality (well-organized, clean, commented) and **Documentation** (README.md).

5.1. Code Structure

Step	Action	Details
5.1.1	**Component Separation**	Separate the application into logical components: `App.js` (or a main screen), `TaskList.js` , `TaskItem.js` , `AddTaskForm.js` , and a `Styles.js` file for styling constants.
5.1.2	**Prop-Types/TypeScript**	Use PropTypes (or TypeScript, if preferred) for clear definition of component props.
5.1.3	**Clean Code**	Use modern JavaScript/React features (e.g., arrow functions, destructuring). Avoid deeply nested ternary operators.

5.2. Documentation (`README.md`)

Step	Action	Details
5.2.1	**Create README**	Create a `README.md` file in the project root.
5.2.2	**Setup Instructions**	Include clear, step-by-step instructions for setting up and running the app (e.g., `npm install` , `npx expo start`).
5.2.3	**Features Overview**	Provide a brief overview of the app's features (Add, Complete, Delete, List).
5.2.4	**Library Mention**	Mention any third-party libraries used (e.g., Expo, React Native Paper) and their purpose.

---

6. Final Review and Submission Checklist

Objective: Verify all requirements and submission guidelines are met.

Checkpoint	Requirement	Status
C.1	**Functionality**	Can add, complete, and delete tasks?
C.2	**State Management**	Uses local component state (no external storage)?
C.3	**UI/UX**	Clean, intuitive, responsive, and provides visual feedback?
C.4	**Visual Distinction**	Completed tasks are visually distinct?
C.5	**Code Quality**	Code is clean, well-organized, and commented?
C.6	**Documentation**	`README.md` is complete with setup, features, and library list?
C.7	**Submission Prep**	Project is initialized as a Git repository and ready for GitHub push.

Final Instruction for AI: Proceed with development following these steps. Prioritize the **UI/UX** aspects to ensure a "perfect" user experience, using modern, clean design principles. All code must be well-commented, especially the state management logic.