summary of State Management in React Native:

🧠 State Management in React Native – Summary

State management is a key concept in React Native, allowing developers to control how data flows and changes within the app. It enables the app to respond to user input, API responses, or other interactions.

📌 1. Local State (using useState)

- **Usage:** For handling data in a single component (e.g., form input, toggles).
- **Example:**
- const [count, setCount] = useState(0);

📌 2. Global State (using Context API)

- **Usage:** When multiple components need access to shared data (e.g., user info, theme).
- **Example:**
    - Create a context using createContext
    - Wrap components in a Provider
    - Access context using useContext

📌 3. State Lifting

- **Purpose:** Move state to a common parent component when child components need to share it.

📌 4. Third-party State Management Libraries

When the app becomes complex, global state can be managed using libraries like:

- **Redux**
    - Central store for all app data.
    - Requires actions, reducers, and middleware.

- **Zustand**
    - Simpler alternative to Redux.
    - Less boilerplate and easier to learn.
- **MobX**
    - Uses observables and reactions to manage state reactively.

🛠️ When to Use What

Scenario	Best Approach
Simple UI state (form, toggles)	useState
Data shared across few components	Context API
Complex app with many state changes	Redux / Zustand / MobX

✅ Best Practices

- Keep state as minimal as possible.
- Use derived values with useMemo where applicable.
- Split state logically (UI state vs business logic).
