Redux in React

🔹 **What is Redux?**

- Redux is a **state management library**.
- It helps **manage and centralize application state**.
- Mostly used with React, but is library-agnostic.

🔹 **Why Use Redux?**

- Manages **complex state** logic outside of components.
- Useful in **large applications** where multiple components share data.
- Helps maintain a **single source of truth** (centralized store).
- Improves **predictability** and **debuggability** (especially with dev tools).

🔹 **Core Principles of Redux:**

1. **Single Source of Truth** – Entire app state is in a single object stored in one store.
2. **State is Read-Only** – You cannot modify state directly; only via actions.
3. **Changes via Pure Functions** – Reducers specify how state changes in response to actions.

🔹 **Key Concepts:**

- **Store**: Holds the state.
- **Action**: A plain JS object with a `type` field. Describes *what* happened.
- **Reducer**: A pure function that takes current state and action, returns new state.

- **Dispatch**: Method to send actions to the reducer.
- **Selector**: A function to get specific data from the store.
- **Middleware**: Handles side effects (e.g., API calls with redux-thunk or redux-saga).

🔹 **Redux Flow Summary:**

1. **User Interaction →**
2. **Dispatch Action →**
3. **Reducer Receives Action →**
4. **Returns New State →**
5. **UI Re-renders with New State**

🔹 **Integration with React:**

- Use the `react-redux` library.
- **Provider**: Wrap your app to connect it with Redux store.
- **useSelector**: Access state.
- **useDispatch**: Send actions.

🔹 **Modern Redux (Redux Toolkit - RTK):**

- Recommended way to write Redux logic.
- Reduces boilerplate.
- Includes utilities like:
  - `createSlice` (combines action creators + reducers)
  - `createAsyncThunk` (for async logic)

- Cleaner and easier syntax.

🔹 **When NOT to Use Redux?**

- For small apps or simple component state.
- If you don't need global/shared state.
- When Context API or useState/useReducer hooks are sufficient.

🔹 **Benefits:**

- Centralized state.
- Predictable state transitions.
- Easy to debug (with time-travel debugging).
- Scalable architecture.

🔹 **Common Middleware:**

- **redux-thunk**: For async logic inside actions.
- **redux-saga**: Uses generator functions for side effects.
- **logger**: Logs actions and state changes (useful for dev).

🔹 **Best Practices:**

- Keep reducer functions pure.
- Normalize your state shape (avoid deeply nested structures).
- Avoid putting non-serializable values in state.
- Use Redux Toolkit to simplify setup.

- Use selectors to avoid tight coupling to state shape.

Feature	**Redux**	**Context API**
Purpose	State management (especially large apps)	Prop drilling solution (simple apps)
Data Flow	Unidirectional (strict)	Unidirectional
Boilerplate	More (less with Redux Toolkit)	Minimal
Async Support	Yes (via middleware like thunk/saga)	No built-in support
DevTools Support	Excellent (Redux DevTools)	Limited
Performance Optimization	Built-in via `connect`, selectors	Might cause re-renders if not optimized
Middleware Support	Yes	No
State Sharing	Excellent for global state	Works but not ideal for complex state
Learning Curve	Moderate to High	Easy

Best For	Large apps, complex state flows	Small to medium apps, simple state

✅ When to Use What?

- **Use Context API** when:
    - You need to share **static** or **lightweight** state (e.g., theme, auth status).
    - You don't want to add extra libraries.
    - State isn't changing frequently.
- **Use Redux** when:
    - App state is **large, shared, or complex**.
    - You need **middleware**, async handling, or **devtools**.
    - State transitions need to be **predictable** and **testable**.