**React - Applying Redux**

**Question 1: What is Redux, and why is it used in React applications?**

**🧠 What is Redux?**

**Redux** is a **state management library** for JavaScript applications, commonly used with React. It provides a **centralized store** where all application state lives and can be managed in a predictable way.

Redux follows a **unidirectional data flow** and is based on three key principles:

1. **Single source of truth** – State is stored in one central place.
2. **State is read-only** – You can't change state directly; instead, you dispatch actions.
3. **Changes are made with pure functions (reducers)** – Reducers take the current state and an action, and return the new state.

### 🎯 Why Use Redux in React Applications

React’s built-in state (useState, useContext) works well for small to medium apps. But Redux becomes useful when:

#### 🔁 1. **You need to share state across many components**

Redux allows deeply nested or unrelated components to access and update shared data without prop drilling.

#### 🔄 2. **Your app state is complex or large**

Redux makes it easier to organize and manage global state such as user authentication, theme settings, or shopping cart data.

#### 🧪 3. **You need predictable state updates**

Redux uses **pure functions** (reducers) to ensure state changes are easy to debug, test, and track over time.

#### 🛠️ 4. **You want powerful developer tools**

Redux DevTools allow you to:

* Inspect every state change
* Time-travel debug
* Undo/redo actions

### 🧱 Basic Redux Workflow

1. **Store** – Holds the app’s state.
2. **Action** – A plain object that describes what happened.
3. **Reducer** – A pure function that describes how the state changes in response to an action.

**Explain the coreconcepts of actions, reducers, and the store.**

### 📝 1. ****Actions****

**What they are:**  
Actions are **plain JavaScript objects** that describe an event or intention to change the state.

### ⚙️ 2. ****Reducers****

**What they are:**  
Reducers are **pure functions** that take the **current state** and an **action**, and return a **new state**.

### 🏬 3. ****Store****

**What it is:**  
The **store** is the **single source of truth** in Redux. It holds the entire application’s state and provides methods to access and update it.

**Question 2: How does Recoil simplify state management in React compared to Redux?**

### ✅ How Recoil Simplifies State Management in React Compared to Redux

Recoil is a state management library for React that offers a **simpler, more React-friendly** way to manage both local and global state—especially compared to Redux.

### 📌 Why Recoil Feels More Natural in React

* Recoil’s API (like useRecoilState, useRecoilValue, selector) is **hook-based**, matching the way you already write React components.
* It allows **shared state** without prop drilling, **without** needing to architect a whole state system like Redux.

### 🧠 When to Use Recoil Over Redux

Use **Recoil** if:

* You want to share state between components with **minimal setup**.
* You prefer a **React-hooks style** API.
* You’re building a **medium-sized app** that doesn’t need Redux’s advanced features like middleware or dev tools.

Use **Redux** if:

* You have a **large-scale app** with complex state logic.
* You need **middleware**, **time-travel debugging**, or **integration with non-React** systems.