**Question 1: What is the Context API in React? How is it used to manage global state across multiple components?**

**✅ What is the Context API?**

The Context API is a built-in feature in React that allows you to share state or data globally across the component tree without prop drilling (i.e., manually passing props through every level).

It’s useful for managing global data such as:

* User authentication
* Theme settings (dark/light mode)
* Language selection
* Cart or app settings

**✅ How it manages global state**:

1. Create a Context using createContext().
2. Wrap your app (or part of it) with a Provider component to supply the data.
3. Use useContext() in any component to access the shared data directly.

**✅ Benefits:**

* Cleaner code with no need to pass props through many layers
* Centralized global state
* Works well with small to medium apps (for larger apps, consider Redux or Recoil)

**Question 2: Explain how createContext() and useContext() are used in React for sharing state.**

**✅ Step-by-Step**:

**1. createContext()**

Used to create a Context object.

jsx

CopyEdit

import React, { createContext } from 'react';

export const ThemeContext = createContext(); // Create context

**2. Provide context value using a Provider**

Wrap part of your app in the Context Provider and pass the value.

jsx

CopyEdit

import { ThemeContext } from './ThemeContext';

function App() {

const theme = 'dark';

return (

<ThemeContext.Provider value={theme}>

<Navbar />

</ThemeContext.Provider>

);

}

**3. useContext()**

Any child component can access the value using useContext().

jsx

CopyEdit

import { useContext } from 'react';

import { ThemeContext } from './ThemeContext';

function Navbar() {

const theme = useContext(ThemeContext);

return <div>Current Theme: {theme}</div>;

}

✅ Summary:

| Function | Purpose |
| --- | --- |
| createContext() | Creates a context object |
| Provider | Supplies the shared value |
| useContext() | Allows access to the context value |

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

Redux is a predictable state management library often used in large or complex React applications. It helps manage global state—shared data that needs to be accessed or updated across multiple components—in a centralized and consistent way.

🔹 Why use Redux in React apps?

* Helps manage complex state logic.
* Provides single source of truth (central store).
* Enables easy debugging and time travel debugging (via Redux DevTools).
* Makes state changes predictable with strict rules.
* Encourages clean separation of concerns.

Core Concepts of Redux:

✅ 1. Store

* The store holds the entire state of the application.
* It is a JavaScript object that acts as the centralized state container.

js

CopyEdit

import { createStore } from 'redux';

const store = createStore(reducer);

✅ 2. Actions

* Actions are plain JavaScript objects describing what happened.
* They must have a type field (string) and can include additional data.

js

CopyEdit

const addTodoAction = {

type: 'ADD\_TODO',

payload: { id: 1, text: 'Learn Redux' }

};

✅ 3. Reducers

* A reducer is a pure function that takes the current state and an action, and returns a new state.
* It defines how the state changes in response to actions.

js

CopyEdit

function todoReducer(state = [], action) {

switch (action.type) {

case 'ADD\_TODO':

return [...state, action.payload];

default:

return state;

}

}

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

Recoil is a modern state management library designed specifically for React. It integrates deeply with React’s rendering lifecycle and makes managing shared and local state much simpler than Redux in many cases.

🔸 How Recoil Simplifies State Management:

| Feature | Recoil | Redux |
| --- | --- | --- |
| Boilerplate | Minimal code | Requires actions, reducers, and store setup |
| Learning Curve | Easy for beginners | More complex for new developers |
| Data Flow | React-style (bidirectional) | Strictly unidirectional |
| State Scope | Local and global state | Mostly global state only |
| React Integration | Hooks-based, seamless | Needs connect() or custom hooks |
| Async Support | Built-in via selector and atom | Needs middleware (e.g., Redux Thunk, Redux Saga) |

🔹 Core Concepts in Recoil:

✅ 1. Atom – The piece of state

js

CopyEdit

import { atom } from 'recoil';

const countState = atom({

key: 'countState',

default: 0,

});

✅ 2. Selector – Derived or computed state

js

CopyEdit

import { selector } from 'recoil';

const doubledCount = selector({

key: 'doubledCount',

get: ({ get }) => get(countState) \* 2,

});

✅ 3. useRecoilState / useRecoilValue – Hooks to use state

js

CopyEdit

const [count, setCount] = useRecoilState(countState);

const doubled = useRecoilValue(doubledCount);