**Use state:**

State is used to store and manage dynamic data that can change over time — and update the UI automatically when that data changes.

So the main motive of state is to make your app dynamic, interactive, and able to respond to user actions without page refresh.

**UseEffect state:**

useEffect is a **React Hook** that lets you run some code **after the component renders**.

It’s mainly used to:

* Fetch data from an API
* Update the DOM
* Set up timers or intervals
* Handle side effects (like page title change, local storage, etc.)

“After React finishes showing something on the screen, I want to run this special task (effect).”

**useRef Hook:**

useRef is a React Hook that is used to:

Access and interact with DOM elements directly (like focusing an input).

Store mutable values (values that don’t cause re-renders when changed).

Keep values between renders (like previous state values).

**Props Drilling:**

What is Props Drilling? Props Drilling happens when you pass data (props) from a top-level (parent) component down to deeply nested child components, even if the middle components don’t need that data — they just pass it along.

**Real-Life Analogy:**

Imagine you have to give a file to your great-grandchild, but you give it to your child → who gives it to your grandchild → who finally gives it to your great-grandchild.

Even if the middle people don’t need the file, they still have to handle it.

**useContext:**

useContext is a React Hook that allows components to share data without having to pass props through every level manually.

It works with the Context API, which helps you create global data that any component can access, no matter how deep it is.

**When to Use useContext?**

When many components need the same data

When you want to avoid props drilling

When working with themes, user info, auth, language, etc.

**React Router**

React Router is a library used to handle navigation in React applications.

**It lets you:**

Create multiple pages (routes) in a single-page application (SPA)

Switch between pages without reloading

Pass data between routes

Make dynamic pages (like user profiles, product pages)

**What is useMemo?**

useMemo is a React Hook that caches the result of a calculation, so it doesn’t run again unless needed.

**Regular Function in React:**

A standard JavaScript function that is recreated every time the component re-renders.

**useCallback Hook:**

A React Hook that memoizes a function, so it is only recreated when its dependencies change — useful for performance optimization.

**Custom Hooks:**

A function created by the developer using built-in React hooks (like useState, useEffect, etc.) to solve a specific need or logic that can be reused in multiple components.

**Think of it like:**

"Built by you, using built-in tools — to handle something repetitive or reusable in a clean way."

**Examples:**

Want to fetch data in many places? → Create a useFetch() custom hook

Handling form inputs repeatedly? → Create a useForm() custom hook

Toggling a theme across app? → Create a useTheme() custom hook

**Controlled Component:**

A controlled component is a form input element whose value is controlled by React state, using useState() and onChange() handlers.

**Uncontrolled Component:**

An uncontrolled component is a form input element whose value is managed by the DOM itself, and accessed using useRef() instead of React state.

**Redux (State Management Library)**

**📌 What is Redux?**

**Redux** is a **state management library** used to manage **global state** in a React application.

Instead of passing data manually through props, Redux lets you store and access state **anywhere in the app**, from a **central store**.

**✅ When to Use Redux?**

* When multiple components need the **same data**
* When your app becomes too complex to manage state with useState
* When data needs to be shared globally (like user login, cart, theme, etc.)

**✅ Core Concepts:**

| **Concept** | **Purpose** |
| --- | --- |
| **Store** | Global place where state lives |
| **Action** | Plain JS object describing what happened |
| **Reducer** | Pure function that updates state based on action |
| **Dispatch** | Sends action to reducer |
| **useSelector** | Gets data from the store |
| **useDispatch** | Triggers an action |

**✅ Example Flow:**

1. User clicks a button
2. An **action** is dispatched: { type: "INCREMENT" }
3. The **reducer** gets this action and updates the state
4. The component **re-renders** with the new state from the **store**

**✅ Code Example:**

**Reducer:**

const counterReducer = (state = 0, action) => {

switch (action.type) {

case 'INCREMENT':

return state + 1;

default:

return state;

}

};

**Dispatching Action:**

const dispatch = useDispatch();

<button onClick={() => dispatch({ type: "INCREMENT" })}>+</button>

**Accessing State:**

const count = useSelector((state) => state.counter);

**✅ Real-Life Analogy:**

Redux is like a **central warehouse**: instead of carrying data around manually, everyone gets and updates it from one central place (the store).

**HOC (Higher Order Component)**

**📌 What is an HOC?**

A **Higher Order Component (HOC)** is a **function** that takes a **component as input**, adds some **extra functionality**, and **returns a new component**.

**✅ Purpose:**

* Reuse logic across multiple components
* Separate concerns (like authentication, logging, themes, etc.)
* Avoid repeating similar code

**✅ Syntax:**

const withExtraInfo = (WrappedComponent) => {

return function EnhancedComponent(props) {

return (

<div>

<p>Extra Info</p>

<WrappedComponent {...props} />

</div>

);

};

};

**Usage:**

const MyComponentWithInfo = withExtraInfo(MyComponent);

**✅ Example:**

function User({ name }) {

return <h2>Hello, {name}</h2>;

}

const withBorder = (Component) => {

return function Wrapped(props) {

return (

<div style={{ border: '2px solid red' }}>

<Component {...props} />

</div>

);

};

};

const BorderedUser = withBorder(User);

**✅ Real-Life Analogy:**

HOC is like **wrapping a gift**: you add **extra decoration (logic)** without changing what’s inside (the original component).

**Difference b/w props, context api, redux?**

