### **🔹 Core React Concepts (You *must* be confident here)**

* **JSX** – How it works and how it's different from HTML
* **Components** – Functional components and when to break UI into components
* **Props & State** – Passing data and managing internal state
* **useState, useEffect** – You *must* know these hooks inside out (dependencies, cleanup, etc.)
* **Conditional Rendering** – Using ternary, short-circuit, etc.
* **Lists and Keys** – .map() and why key is important
* **Forms** – Controlled vs uncontrolled inputs

### **🔹 React Hooks (Intermediate)**

* **useContext** – For managing simple state or themes without props drilling
* **useRef** – For DOM refs, and sometimes to hold mutable values
* **useMemo, useCallback** – Know how to prevent unnecessary re-renders
* **Custom Hooks** – Why and how to make them

### **🔹 React Router**

* **Basic Routing** – <BrowserRouter>, <Route>, <Link>, useNavigate(), etc.
* **Dynamic Routing** – URL params
* **Nested Routes** (if your app uses it)

### **🔹 State Management**

* If your app uses **Redux**, **Zustand**, or **Context API**, know how and why they were used
* Be able to explain why you chose it and the pros/cons

### **🔹 API Integration**

* **Fetching data** with fetch() or **Axios**
* Handling **loading**, **error**, and **success** states
* Using useEffect properly with async data

### **🔹 Component Communication**

* **Parent to child** and **child to parent** communication
* **Lifting state up**

### **🔹 Performance Optimization (Basic understanding)**

* Why components re-render
* How to memoize components using React.memo, useMemo, useCallback
* Lazy loading components using React.lazy and Suspense

### **🔹 Styling**

* CSS Modules / Styled-components / Tailwind / SCSS (whichever you use)
* Responsive design and media queries
* Conditional styles

### **🔹 Project Structure and Best Practices**

* Folder structure
* Code splitting and lazy loading
* Reusable components
* Clean code and naming conventions

### **🔹 Testing (Basic awareness)**

* Basics of unit tests in React (e.g., with React Testing Library or Jest)
* Not mandatory unless the company has test focus

### **🔹 Other Useful Skills to Mention in Interviews**

* Familiarity with **TypeScript** (if you’ve used it)
* Familiarity with **Git & GitHub**
* Knowledge of **eslint, prettier, husky** (if your team uses them)
* Experience with any UI library (Material UI, Ant Design, etc.)

### **Bonus: Soft Skills / Communication**

* Be ready to **explain your projects**: What the app does, what tech you used, your role in the team
* Be able to explain **why you wrote something the way you did**
* Talk about a bug or performance issue you solved

### **🔹 React Basics**

1. **What is React and why would you use it?**
2. **What are the differences between a class component and a functional component?**
3. **What is JSX? Can a component return multiple elements?**
4. **What is the virtual DOM and how does React use it?**
5. **How does React handle rendering and updating the UI?**

### **🔹 Hooks**

1. **What is useState? How is it used?**
2. **What is useEffect? What happens if you don’t pass a dependency array?**
3. **What is the difference between useEffect(() => {}, []) and useEffect(() => {}, [someValue])?**
4. **What is useRef and when would you use it?**
5. **Have you created any custom hooks? Why and how?**

### **🔹 Props & State**

1. **How do you pass data between components in React?**
2. **How can a child component communicate with a parent component?**
3. **What’s the difference between state and props?**
4. **What is "lifting state up"?**

### **🔹 Component Behavior**

1. **What happens if you update state inside a loop or a condition?**
2. **What is conditional rendering and how do you handle it in React?**
3. **What happens when a parent component re-renders — will child components re-render too?**

### **🔹 Routing**

1. **How do you set up routing in React?**
2. **How do you pass parameters through the URL?**
3. **What is the difference between <Link> and <a> in React Router?**

### **🔹 API Handling**

1. **How do you fetch data from an API in React?**
2. **Where and when do you fetch the data in a component?**
3. **How do you handle loading and error states?**
4. **What if two API calls need to happen one after the other — how would you handle that?**

### **🔹 Optimization / Real-world Problems**

1. **What causes a component to re-render?**
2. **What is the purpose of useMemo and useCallback?**
3. **Have you used React.memo? How is it useful?**
4. **How do you lazy-load components in React?**

### **🔹 UI and Forms**

1. **How do you handle forms in React?**
2. **What’s the difference between controlled and uncontrolled components?**
3. **How do you handle multiple form inputs with a single change handler?**

### **🔹 Project-based Questions (They *will* ask this!)**

1. **What was your role in the last project?**
2. **How did you structure your React app?**
3. **How did you manage the state across multiple components?**
4. **Did you face any performance issues? How did you solve them?**
5. **How did you handle errors and loading in the UI?**
6. **Tell me about a bug you found and fixed.**

### **🔹 Bonus / Tools & Ecosystem**

1. **Have you used any UI libraries like Material UI, Ant Design, etc.?**
2. **Have you worked with TypeScript in React?**
3. **Do you use ESLint or Prettier in your projects?**

### **React Interview Questions and Answers**

#### **1. What is React and why would you use it?**

React is a JavaScript library for building user interfaces. It allows developers to build reusable UI components and efficiently update the UI when data changes.

#### **2. Differences between class and functional components?**

* Class components use ES6 classes and support lifecycle methods.
* Functional components are simpler and use hooks for state and lifecycle.

#### **3. What is JSX? Can a component return multiple elements?**

JSX is a syntax extension that looks like HTML but works in JavaScript. Yes, components can return multiple elements wrapped in a single parent (e.g., a fragment <>...</>).

#### **4. What is the virtual DOM and how does React use it?**

The virtual DOM is a lightweight copy of the real DOM. React compares the virtual DOM with the previous version to find and apply only the changes needed.

#### **5. How does React handle rendering and updating the UI?**

React re-renders components when state or props change, compares virtual DOMs, and applies minimal updates to the real DOM.

#### **6. What is useState?**

useState is a hook that lets you add state to functional components. It returns a state variable and a setter function.

#### **7. What is useEffect?**

useEffect runs side effects in functional components. It can run on mount, on every render, or when dependencies change.

#### **8. Difference between useEffect(() => {}, []) and useEffect(() => {}, [someValue])?**

* [] means run once on mount.
* [someValue] means run on mount and when someValue changes.

#### **9. What is useRef and when to use it?**

useRef is used to hold a reference to a DOM element or to persist values across renders without causing re-renders.

#### **10. Have you created any custom hooks?**

Yes. Custom hooks are reusable functions that use React hooks inside them. They help to extract logic from components.

#### **11. How do you pass data between components?**

Using props. For parent to child, pass data via props. For child to parent, use callbacks.

#### **12. How can a child communicate with a parent?**

By calling a function passed as a prop from the parent.

#### **13. Difference between state and props?**

* Props are read-only, passed from parent.
* State is local and managed within the component.

#### **14. What is "lifting state up"?**

Moving shared state to the closest common ancestor to share it between child components.

#### **15. What happens if you update state inside a loop or condition?**

React batches updates but updating state in a loop without care can cause unexpected re-renders.

#### **16. What is conditional rendering?**

Displaying UI elements based on a condition using ternary operators or logical &&.

#### **17. When does a child re-render if the parent re-renders?**

Always, unless wrapped with React.memo or controlled with shouldComponentUpdate or useMemo.

#### **18. How do you set up routing in React?**

Using react-router-dom with <BrowserRouter>, <Routes>, <Route>, <Link>, etc.

#### **19. How do you pass parameters through the URL?**

Using path parameters like /user/:id and useParams() to access them.

#### **20. Difference between <Link> and <a> in React Router?**

<Link> prevents page reload and works with React Router. <a> reloads the page.

#### **21. How do you fetch data in React?**

Use fetch or Axios inside useEffect for API calls.

#### **22. Where and when do you fetch the data?**

In useEffect, after the component mounts.

#### **23. How do you handle loading and error states?**

Use state variables like isLoading and error and conditionally render messages or spinners.

#### **24. How to handle two dependent API calls?**

Use async/await inside useEffect and make second API call after first one succeeds.

#### **25. What causes a component to re-render?**

Changes in state, props, or context.

#### **26. Purpose of useMemo and useCallback?**

* useMemo: memoizes a computed value.
* useCallback: memoizes a function to avoid re-creation.

#### **27. Have you used React.memo?**

Yes, to prevent re-renders of functional components when props haven't changed.

#### **28. How do you lazy-load components in React?**

Using React.lazy() and <Suspense> for dynamic imports.

#### **29. How do you handle forms in React?**

Use useState to control input values and handle onChange and onSubmit.

#### **30. Controlled vs Uncontrolled components?**

* Controlled: value controlled by React.
* Uncontrolled: value managed by the DOM using refs.

#### **31. How to handle multiple inputs with one handler?**

Use the input name attribute and update state using [e.target.name]: e.target.value.

#### **32. What was your role in the last project?**

(You should describe your contributions clearly — components built, APIs integrated, features implemented, etc.)

#### **33. How did you structure your React app?**

Component-based folder structure with folders for components, pages, services, and assets.

#### **34. How did you manage state across components?**

Using useState, useContext, and sometimes third-party libraries like Redux or Zustand.

#### **35. Did you face any performance issues?**

Yes. I used React.memo, useMemo, lazy loading, and code splitting to fix performance problems.

#### **36. How did you handle errors and loading in the UI?**

Used try-catch, isLoading, and error state to show spinners and error messages.

#### **37. Tell me about a bug you fixed.**

(Answer with a real example: what the bug was, how you debugged it, and how you fixed it.)

#### **38. Used any UI libraries like MUI or AntD?**

Yes. (Mention the library you used and why — rapid development, clean design, etc.)

#### **39. Worked with TypeScript?**

(If yes) Yes, for type safety and better developer experience. (If no) No, but I’m learning it and understand the basics.

#### **40. Do you use ESLint or Prettier?**

Yes, to maintain code quality and formatting consistency.