React.js Interview Prep

That's a great move you're planning! Companies like Capgemini, Infosys, Wipro, and similar MNCs often focus on both **technical fundamentals** and **practical project experience** during interviews, especially for someone with **3 years of experience in React.js**.

Here's a curated list of **React.js project-based use cases** and **coding/interview questions** you can expect – all in a **Q&A format** to help you prep better:

⚙️ Project-Based Use Cases (Scenario/Case-Study Type Questions)

Q1: Describe a project where you implemented state management in React. What tools did you use?

A:
In my last project, I worked on a dashboard for internal employee management. For state management, I used Redux Toolkit to manage global state like user authentication, employee lists, and role-based access. React Context was used for theme toggling and user preferences. Async data fetching was handled using Redux Thunk.

Q2: Have you implemented lazy loading in your React app? How and why?

A:
Yes, in a React-based e-commerce portal. We used React's `React.lazy` and `Suspense` for code-splitting and lazy loading components like product details and user profile pages. This improved initial load time and performance by 30%.

Q3: How did you manage API calls and error handling in your project?

A:
We used Axios for API calls and centralized all requests in a service layer. Each API call returned a consistent structure using a try-catch block. Errors were caught globally using an Axios interceptor, where we handled token expiry, unauthorized access, and showed user-friendly messages.

Q4: How do you handle forms in React? Did you use any library?

A:
Yes, for complex forms we used **Formik** along with **Yup** for validation. It simplified validation, error handling, and controlled components, especially for dynamic forms like user registration, product configuration, etc.

Q5: What kind of folder structure did you follow in your React projects?

A:
We followed a feature-based folder structure:

```bash
/src
  /components
  /features
    /auth
    /dashboard
    /users
  /services
  /utils
  /hooks
```

Each feature had its own components, services, and styles to improve scalability and maintainability.

🧠 Coding/Technical Questions (React + JS + Basic DSA)

Q6: What's the difference between controlled and uncontrolled components?

A:

- Controlled: Form data is handled by React using state. E.g., `value={formData.name}`.
- Uncontrolled: Uses ref to access DOM directly, not ideal for most apps. E.g., `ref={inputRef}`.

Controlled is preferred for complex logic and validation.

Q7: Write a simple React component that fetches user data and displays it in a list.

A:

```jsx
import React, { useEffect, useState } from 'react';
import axios from 'axios';

function UserList() {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    axios.get('https://jsonplaceholder.typicode.com/users')
      .then(res => setUsers(res.data))
      .catch(err => console.error(err));
  }, []);

  return (
    <ul>
      {users.map(user => <li key={user.id}>{user.name}</li>)}
    </ul>
  );
}
export default UserList;
```

Q8: Explain useEffect dependency array. What happens if you leave it empty vs not adding it?

A:

- `[]` : Runs only once on mount (like `componentDidMount`).
- No array: Runs after every render (can cause performance issues).
- `[count]` : Runs when `count` changes.

Q9: How would you optimize a React app for performance?

A:

- Code splitting via lazy loading
- Memoization with `React.memo`, `useMemo`, `useCallback`
- Avoid unnecessary re-renders
- Debouncing input
- Virtualization for large lists (e.g., `react-window`)

Q10: Write a simple debounce hook in React.

A:

jsx

```jsx
import { useEffect, useState } from 'react';

function useDebounce(value, delay) {
  const [debounced, setDebounced] = useState(value);

  useEffect(() => {
    const handler = setTimeout(() => setDebounced(value), delay);
    return () => clearTimeout(handler);
```

```
  }, [value, delay]);

  return debounced;
}
```

Q11: What is the virtual DOM and how does React use it?

A:
The virtual DOM is a lightweight copy of the real DOM. React compares the current and previous virtual DOM trees using a process called **reconciliation** and updates only the changed nodes in the actual DOM (efficient and fast).

Q12: What are keys in React and why are they important?

A:
Keys help React identify which items have changed, are added, or removed in a list. Without keys or with improper keys (like index), React may cause rendering issues or poor performance.

💼 Bonus: System Design-ish Project Scenario

Q13: How would you design a notification system in a React app?

A:

- Use a global state or context to store notifications.
- Trigger notifications from anywhere (e.g., via Redux actions or context methods).
- Display via a `Toast` component.
- Optionally, integrate with WebSocket for real-time updates.

Want me to create a **mock interview** set or build a **sample project structure** for your resume and portfolio?

Absolutely! Here are **10 more questions and answers**, continuing in the same style — focused on **React.js concepts, coding ability, and real-world project use cases** that are likely to be asked in interviews at companies like **Capgemini, Infosys, Wipro**, etc.

🔧 Real-World React Questions & Coding Tasks (Part 2)

Q14: What are React Hooks? Can you list some commonly used ones?

A:
React Hooks let you use state and lifecycle methods in functional components. Common hooks:

- `useState`
- `useEffect`
- `useContext`
- `useRef`
- `useMemo`
- `useCallback`
- `useReducer`

Q15: Write a component that shows a countdown timer from 10 to 0.

A:

```jsx
```

```jsx
import React, { useEffect, useState } from 'react';

function Countdown() {
  const [time, setTime] = useState(10);

  useEffect(() => {
    if (time === 0) return;
    const timer = setTimeout(() => setTime(time - 1), 1000);
    return () => clearTimeout(timer);
  }, [time]);

  return <h1>{time}</h1>;
}
```

Q16: What is the difference between `useMemo` and `useCallback`?

A:

- `useMemo`: Caches the **result** of a function.
- `useCallback`: Caches the **function itself**.

Use `useMemo` for expensive computations, `useCallback` for memoizing callbacks passed to child components.

Q17: What is Prop Drilling and how do you avoid it?

A: Prop Drilling is when you pass data through many nested components unnecessarily.
To avoid it:

- Use **React Context**
- Use **Redux** or **Zustand** for global state
- Use **custom hooks**

Q18: Create a reusable modal component in React.

A:

```jsx
function Modal({ isOpen, onClose, children }) {
  if (!isOpen) return null;

  return (
    <div className="modal-backdrop" onClick={onClose}>
      <div className="modal-content" onClick={e => e.stopPropagation()}>
        {children}
        <button onClick={onClose}>Close</button>
      </div>
    </div>
  );
}
```

Usage:

```jsx
<Modal isOpen={showModal} onClose={() => setShowModal(false)}>
  <p>Hello from Modal!</p>
</Modal>
```

Q19: Explain the difference between `useEffect` and `useLayoutEffect`.

A:

- `useEffect`: Runs **after** DOM paint. Ideal for data fetching and subscriptions.
- `useLayoutEffect`: Runs **before** DOM paint. Use when DOM mutations or measurements are needed.

Q20: How do you handle authentication in a React app?

A:

- Store token in **HTTP-only cookie** or **localStorage** (for less secure apps)
- Use context or Redux to manage auth state
- Protect routes with a **PrivateRoute** component
- Use interceptors to attach token to requests

Q21: What is React Context API and how have you used it?

A: Context API is a way to pass data globally without prop drilling.
I've used it for:

- User authentication state
- Theme switching (dark/light mode)
- App-level config and language preferences

Q22: Write a hook that tracks window size (responsive design).

A:

```jsx
import { useState, useEffect } from 'react';

function useWindowSize() {
  const [size, setSize] = useState({ width: window.innerWidth, height:
window.innerHeight });

  useEffect(() => {
    const handleResize = () => setSize({ width: window.innerWidth, height:
window.innerHeight });
    window.addEventListener('resize', handleResize);
    return () => window.removeEventListener('resize', handleResize);
  }, []);
```

```
  return size;
}
```

Q23: What is reconciliation in React?

A:
Reconciliation is the process React uses to compare the previous virtual DOM with the new one and apply minimal changes to the real DOM. It uses a diffing algorithm for better performance.

If you'd like, I can also help you prep:

- 🔍 A **sample project repo** to showcase in your interviews
- 📄 A **React-focused resume section**
- 🎤 A **mock interview Q&A** with follow-ups

Would you like help with one of those next?

Of course! Here are **10 more React.js interview questions and answers**, designed to reflect the kind of depth expected for **3 years of experience**, especially in service-based MNCs like **Infosys, Capgemini, Wipro**, etc.

🔄 React.js Interview Questions (Part 3)

Q24: What is the purpose of `key` prop in lists, and what happens if you don't use it?

A:
The `key` prop helps React identify which items have changed, are added, or are removed. Without it, React will re-render all items unnecessarily, which can lead to performance issues or UI bugs (especially with animations or input fields).

Q25: Can you explain how `useRef` works and give a use case?

A:

`useRef` gives you a persistent, mutable value that doesn't cause re-renders.
Common use cases:

- Accessing DOM elements directly (e.g., focusing an input)
- Storing previous values or timers

```jsx
const inputRef = useRef();
<input ref={inputRef} />;
inputRef.current.focus();
```

Q26: What are higher-order components (HOC)?

A:

A **HOC** is a function that takes a component and returns a new component with enhanced behavior.
Example: Adding authentication, logging, or theming.

```jsx
const withAuth = (Component) => {
  return function AuthWrapper(props) {
    if (!props.isLoggedIn) return <Redirect to="/login" />;
    return <Component {...props} />;
  };
};
```

Q27: What is the difference between React Router v5 and v6?

A:

- v6 uses a simplified `Routes` wrapper instead of `Switch`
- v6 supports **nested routing** more cleanly
- Route matching is more **precise** in v6 (no need for `exact`)
- More declarative and fewer bugs due to ambiguity

Q28: How would you implement dark mode in a React application?

A:

- Store theme in Context or local state
- Save preference to localStorage
- Toggle a class (e.g., `dark`) on the `<body>` or root `<div>`
- Use CSS variables or Tailwind's dark mode classes

Q29: Explain `useReducer`. When would you prefer it over `useState`?

A:
`useReducer` is used when state logic is **complex** (e.g., multiple sub-values or actions).
Prefer `useReducer` when:

- You have multiple state transitions
- You want to manage actions like in Redux

Q30: How do you prevent unnecessary re-renders in React?

A:

- Use `React.memo` for pure functional components
- Use `useCallback` to memoize functions passed as props

- Use `useMemo` for expensive calculations
- Avoid creating inline functions inside JSX

Q31: Explain React's component lifecycle using hooks.

A:

- **Mount:** `useEffect(() => {}, [])`
- **Update:** `useEffect(() => {}, [dependencies])`
- **Unmount:** `return () => {}` inside `useEffect`

```jsx
useEffect(() => {
  console.log("Mounted or updated");
  return () => console.log("Cleanup on unmount");
}, [someState]);
```

Q32: What is code splitting and why is it useful?

A:

Code splitting breaks your bundle into smaller chunks, improving initial load performance.

React uses:

- `React.lazy()` + `Suspense`
- Dynamic `import()` syntax Helps reduce time-to-interactive in large apps.

Q33: How do you handle global notifications (e.g., toast messages) in your React apps?

A:

- Use **React Context** or **Redux** to store messages
- Create a reusable `Toast` component
- Use libraries like `react-toastify` or `notistack`

Example:

```jsx
toast.success("User created successfully!");
```

Want me to give you a **mock interview sheet** with mixed questions or build a **React project roadmap** to strengthen your switch? Let me know how you'd like to continue!

You're on a roll! 💪 Here's another set of **10 React.js questions and answers** — this batch dives a bit deeper into performance, testing, advanced patterns, and practical coding knowledge expected from someone with **3 years of experience**.

💡 React.js Interview Questions (Part 4)

Q34: What is the difference between `useEffect` and `useLayoutEffect` in terms of timing?

A:

- `useEffect`: Runs **after** the DOM has painted.
- `useLayoutEffect`: Runs **synchronously after all DOM mutations**, but **before** the paint.

Use `useLayoutEffect` when you need to measure layout or apply styles immediately.

Q35: What are custom hooks? Can you give an example?

A:

Custom hooks let you extract and reuse logic between components. They are just JavaScript functions that use other hooks.

```jsx
function useLocalStorage(key, initialValue) {
  const [value, setValue] = useState(() =>
    JSON.parse(localStorage.getItem(key)) || initialValue
  );

  useEffect(() => {
    localStorage.setItem(key, JSON.stringify(value));
  }, [key, value]);

  return [value, setValue];
}
```

Q36: What is the role of error boundaries in React?

A:

Error boundaries catch JavaScript errors in **child components** during rendering and lifecycle methods. They prevent the entire app from crashing.

They are class components that implement `componentDidCatch` and `getDerivedStateFromError`.

Q37: How do you write unit tests for React components?

A:

Using **Jest + React Testing Library**:

- Test rendering, props, events, state
- Use `render()`, `fireEvent`, and `screen.getByText`, etc.

```jsx
```

```
test('renders button and clicks it', () => {
  render(<MyButton />);
  const btn = screen.getByText('Click Me');
  fireEvent.click(btn);
  expect(screen.getByText('Clicked')).toBeInTheDocument();
});
```

Q38: How do you manage side effects in a Redux-based React app?

A:

You can use:

- **Redux Thunk**: Middleware to dispatch async actions
- **Redux Saga**: Generator-based control over async flows
- **RTK Query**: Modern approach built into Redux Toolkit for data fetching

---

Q39: What are render props in React?

A:

A render prop is a function prop that returns JSX, allowing for **shared logic** across components.

```jsx
<MouseTracker render={position => <h1>X: {position.x}</h1>} />
```

Q40: What is memoization in React and how does it help performance?

A:
Memoization prevents unnecessary computations and re-renders.
React tools for this:

- `React.memo` – memoize a component
- `useMemo` – memoize a value
- `useCallback` – memoize a function

Q41: What is server-side rendering (SSR) in React?

A:
SSR renders React components **on the server** and sends HTML to the browser. Improves SEO and initial page load.
Frameworks like **Next.js** enable SSR in React apps.

Q42: What is hydration in React?

A:
Hydration is when React attaches event listeners to the server-rendered HTML (SSR). It's how the app becomes interactive after loading on the client.

Q43: How do you persist login sessions in a React app?

A:

- Store JWT/token in `localStorage`, `sessionStorage`, or **HTTP-only cookies**
- On app load, check if token exists
- Refresh token before expiry or on background
- Store user data in global state (Context/Redux)

🔥 Would you like a list of **React mini-project ideas** for your portfolio or some **mock HR + technical round questions** that follow these? Let me know how you want to level up next.