🧠 Top React.js Interview Questions (With Explanations)

If you're preparing for a **React developer role**, you *must* go beyond `useState` and `props`. Modern interviews test **deep conceptual understanding**, not just syntax.

Here are 🔟 **frequently asked** (and tricky) **React.js interview questions** you need to master:

1. ⚛️ What's the difference between `useEffect`, `useLayoutEffect`, and `useInsertionEffect`?

- `useEffect`: Runs **after** render.
- `useLayoutEffect`: Runs **synchronously** *after* DOM changes but **before** paint. Useful for layout calculations.
- `useInsertionEffect`: New! Runs **before** styles are injected — used by CSS-in-JS libraries.

📌 Tip: Use `useLayoutEffect` only when DOM measurement is critical (e.g. animations, refs).

2. 🚀 Explain React's Reconciliation Process

Reconciliation is React's algorithm to **update the DOM efficiently** by comparing the new virtual DOM with the previous one (diffing), and applying the minimal required changes.

🧠 Interview Insight: Discuss keys, performance, and how React avoids full re-rendering.

3. 🧩 What's the difference between Controlled and Uncontrolled Components?

- **Controlled**: Form data is handled by React via state (`useState`).
- **Uncontrolled**: Data is handled by the DOM using refs.

✔️ Controlled: more predictable ⚠️ Uncontrolled: faster but harder to validate

4. 🔁 How does `key` help in React lists?

Keys help React identify **which items changed, added, or removed**.

🚫 Wrong: Using index as key ✔️ Right: Use unique, consistent IDs

5. 📦 What are React Server Components?

React Server Components (RSC) allow rendering parts of the UI on the server **without sending unnecessary JS to the client**. Improves performance and load times.

Used heavily with **Next.js App Router**.

6. 🧵 Explain the difference between `Context API` and `Redux`.

- **Context API**: For light, app-wide state (e.g. theme, auth). No middleware support.
- **Redux**: For complex state management, with devtools, middlewares (like thunk, saga), and modular structure.

📌 Tip: Mention modern alternatives like **Zustand**, **Recoil**, **Jotai** too.

7. 🔄 What causes unnecessary re-renders in React?

Common causes:

- Passing new object/array references

- Inline arrow functions
- Not using `React.memo` or `useCallback`

🛠️ Tools like `why-did-you-render` can help detect this in dev mode.

8. ⚙️ Explain `useCallback` vs `useMemo`

- `useCallback(fn, deps)` → returns **memoized function**
- `useMemo(() => compute, deps)` → returns **memoized value**

Use them to **avoid recalculations** or **function re-creation**.

9. 🔐 How to handle authentication in React?

Strategies include:

- Store JWT in HTTP-only cookies
- Protect routes with conditional rendering or route guards
- Use `useContext` for auth state or a global store like Redux

Bonus: Mention **middleware-based auth in Next.js** for bonus points.

Concurrent Mode allows React to interrupt rendering work and continue it later. It helps with:

- Better responsiveness
- Non-blocking rendering
- Features like `startTransition` and `Suspense`

⚠️ It's **experimental** but powering advanced UI/UX in Next.js and React 19+.

💡 Bonus Tip:

Be ready to build something in interviews — a simple `ToDo`, weather app, or counter with `useReducer`. Interviewers value code **quality** more than speed.