Advanced React Interview Questions with In-Depth Answers

✅Complete List of Advanced React Interview Questions Covered in This Document:

1. **1. What is React Fiber and how does it differ from the old reconciliation algorithm?**

React Fiber is the new core algorithm of React that enables better rendering capabilities. Before Fiber, React used a stack-based reconciliation algorithm which was synchronous and could not be paused once it started rendering. That meant if React was doing something complex and needed a lot of time to finish, your UI could freeze or lag.

With React Fiber, rendering became interruptible. Think of it like this — React can now start a task, pause it if something more important comes up (like a user typing), and come back to finish it later. This allows React to prioritize updates and deliver a much smoother user experience.

Key advantages of Fiber: - **Interruptible rendering**: React can pause, resume, or cancel rendering tasks. - **Prioritization**: Updates can be categorized as urgent or non-urgent. - **Concurrency support**: Enables features like time slicing and Suspense.

In short: React Fiber makes the rendering system smarter, allowing for smoother performance and paving the way for modern features like Concurrent Mode and Suspense. 2. How does React determine when to re-render a component? 3. What are concurrent features in React and how do they help? 4. Explain React's batching behavior and what changed in React 18. 5. What is the difference between useMemo and useCallback, and when not to use them? 6. How does Suspense work, and what are some real use cases beyond lazy loading? 7. What is useImperativeHandle and when should you use it? 8. How do you optimize large lists in React? 9. How does useRef differ from useState? 10. How does React handle hydration in SSR, and what problems can arise?

☑️Additional Advanced React Interview Questions and Answers:

11. What is the difference between controlled and uncontrolled components?

- **Controlled components** have their form data controlled by React state.
- **Uncontrolled components** manage their own state using the DOM.

```
// Controlled
<input value={name} onChange={e => setName(e.target.value)} />

// Uncontrolled
<input defaultValue="John" ref={inputRef} />
```

Use controlled for React-driven logic. Use uncontrolled for simple cases or when integrating with non-React libraries.

---

12. What are custom hooks and why should you use them?

Custom hooks let you extract reusable logic from components.

```
function useAuth() {
  const [user, setUser] = useState(null);
  useEffect(() => {
    // Fetch auth status
  }, []);
  return user;
}
```

They help you: - Keep code DRY - Improve readability - Share logic between components

---

13. What's the difference between useEffect and useLayoutEffect?

- `useEffect` runs **after** paint (non-blocking)
- `useLayoutEffect` runs **before** paint (blocking)

Use `useLayoutEffect` when you need to read/write layout measurements before the browser paints — like scrolling or animation setup.

```
useLayoutEffect(() => {
  const height = ref.current.offsetHeight;
}, []);
```

---

14. What are React portals and when to use them?

React portals allow rendering children into a DOM node **outside the parent hierarchy**.

```
ReactDOM.createPortal(<Modal />, document.getElementById('modal-root'))
```

Useful for: - Modals - Tooltips - Popovers

They maintain React's event bubbling even though they're outside the DOM tree.

---

15. What is key prop in React and why is it important?

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

Avoid using indexes as keys when the list is dynamic:

```
items.map((item, index) => <li key={item.id}>{item.name}</li>)
```

Improper keys can lead to: - Bugs in stateful lists - Poor performance

---

16. How does React handle state updates — synchronously or asynchronously?

React batches and handles state updates **asynchronously**, meaning updates might not reflect immediately after calling `setState`.

```
setCount(count + 1);
console.log(count); // Might log old value
```

Use functional updates if relying on previous state:

```
setCount(prev => prev + 1);
```

---

17. What is reconciliation in React?

Reconciliation is React's process of diffing the virtual DOM tree to determine what changed and updating the real DOM efficiently.

- Uses keys to identify changes
- Tries to minimize DOM operations

Fiber made reconciliation interruptible and smarter.

---

18. Explain lazy loading in React.

Lazy loading delays the loading of components until they're needed.

```
const LazyComp = React.lazy(() => import('./HeavyComponent'));
```

Use it with `<Suspense>` to show fallback UI during load.

Improves performance by reducing initial bundle size.

---

19. What is Server Components in React?

React Server Components (RSC) let you render components on the server **without sending JS to the client**.

- Can fetch data on server
- Send only minimal HTML/JSON to browser
- Works with Suspense for streaming

They're experimental but very promising for performance.

---

20. What are render props and how do they compare to hooks?

Render props are a pattern for sharing logic using a function-as-children.

```
<DataFetcher render={data => <UI data={data} />} />
```

Hooks are now preferred for readability and reuse, but render props still work in older patterns.
