Armen Melkumyan • 1st
Technical / Solutions Architect
2w •

In a senior-level React interview, demonstrating a deep understanding of useRef goes beyond just “it references DOM elements.” Here’s how to articulate its purpose and pitfalls in a way that shows you can handle complex use cases:

Why We Use useRef

Direct DOM Access

When you need low-level operations—such as focusing an input field, measuring dimensions, or integrating with non-React libraries—useRef is the go-to. It ensures you’re not juggling any hacky workarounds for DOM access.

Persisting Mutable Data Across Renders

- A classic scenario is storing a mutable variable (like a timer ID or a previous render’s value) without causing re-renders. Since useRef values stay intact across renders and don’t trigger updates themselves, it’s perfect for data that’s “behind the scenes.”

- Performance Considerations

Because updating ref.current doesn’t invoke a re-render, you can avoid unnecessary component cycles. This can be especially useful in performance-sensitive scenarios like real-time animations.

Potential Drawbacks

- Not Reactive

Data in a ref isn’t observed by React. If you need your UI to reflect changes in that data, you’re out of luck with useRef alone. This can lead to subtle bugs if you store values that should influence rendering in a ref instead of useState.

- Misuse Can Cause Unpredictable State

Senior developers know that scattering logic between refs and state can become a debugging nightmare. Overusing refs for data that should be controlled by React’s state model compromises clarity and can lead to out-of-sync UI.

- Ref-Heavy Code Is Harder to Maintain

Throwing too many variables into refs can create a “kitchen sink” scenario. Without careful documentation, it’s difficult for future maintainers—or even future you—to understand how refs drive behavior. This undermines React’s declarative paradigm, which is especially problematic in complex apps.

Conclusion

useRef is an essential tool in scenarios where stateful reactivity isn’t required. You’ll reach for it to interact with DOM elements directly or to hold onto mutable data that changes outside React’s rendering cycle. Just keep in mind that, if your UI must update based on some value, store it in state—not in a ref. As a senior engineer, your role is to strike the right balance between using refs for performance optimizations and keeping your code declarative and straightforward.