Interview Process (Stage II)

Frontend Tech Interview Prep Guide

1. React Core & Rendering Model

- How React's **virtual DOM diffing and reconciliation** works.
- What causes **component re-renders** (state, props, context changes).
- How React batches state updates.
- useState, useReducer, and useMemo – what they do and what they don't prevent.
- When and how to use useCallback and React.memo to prevent unnecessary renders.
- Common patterns for **lifting state up** and for **component memoization**.
- Debugging re-renders using **React DevTools → Components → Render count profiling**.

Practice debugging:

- Identify why a component is re-rendering unexpectedly (prop identity changes, missing memoization, stale closures).

2. Hooks and Side Effects

- Hooks i.e. useEffect lifecycle behavior: when it runs, cleanup timing, mount vs. update.
- Dependency arrays — how React determines when to re-run effects.
- Common pitfalls:
- Stabilizing deps with useMemo/useCallback.
- Difference between hooks.
- When to avoid effects (derive state from props or render instead).

Practice debugging:

- Spotting infinite render loops.
- Fixing stale closures in async effects.
- Using eslint-plugin-react-hooks to catch dependency issues.

3. State Management: Client vs. Server State

- What **client state** is (UI, modals, form input, filters, active tab, etc.).
- What **server state** is (data fetched from APIs, canonical on server).
- Proper tools:
    - Client state?
    - Server state?
- When **not** to store server data in Redux
- Deriving computed values instead of duplicating state.

Practice:

- Designing a component tree where both client and server state coexist cleanly.
- Handling refetching and cache invalidation (React Query invalidateQueries).

4. Security & XSS Prevention

- How React automatically escapes HTML by default.
- Dangers of dangerouslySetInnerHTML and how to sanitize user-generated HTML.
- Libraries: **DOMPurify**, sanitize-html.
- Content Security Policy (CSP): basics, preventing inline scripts.
- Safe link attributes (rel="noopener noreferrer", etc).
- Avoiding injection of user input into event handlers or URLs.

Practice:

- Implement safe rendering of Markdown or HTML from users.
- Test with malicious payloads.

5. Real-Time Data & Live Updates

- Strategies for live data updates:
- When to use each
- Implementing reconnect logic, exponential backoff, and heartbeat messages.

Practice:

- Implementing a dashboard that updates from live data streams.
- Debugging connection drop / reconnection issues.

6. Forms, Validation & Data Flow

- Controlled vs. uncontrolled components.
- Libraries: React Hook Form, Formik.
- Schema-based validation
- Managing multi-step/wizard forms:
- Handling async validation (API-based checks).
- Accessibility for forms: labels, errors, keyboard navigation.

Practice debugging:

- React Hook Form's watch and trigger.
- Handling errors and showing inline validation messages.
- Recovery from lost drafts or navigation interruptions.

7. Testing (Unit + Integration)

- Testing with **Jest/Vitest + React Testing Library**.
- Mocking API calls and asserting side effects (toasts, redirects).
- Testing async behavior with waitFor and findBy*.
- Testing form validation, success, and error flows.
- Querying elements by role, label, text for accessibility.

- Snapshot vs. behavior-driven testing.
- When to mock libraries

Practice:

- Write a test that fills out a form, triggers an API mock, and asserts a success toast.
- Debugging failed async tests (flaky waits, unmocked timers).

8. Performance Optimization

- useMemo, useCallback, React.memo interplay.
- Lazy loading, code splitting, suspense for data fetching.
- Profiling with React DevTools Profiler.
- Avoiding excessive context re-renders.
- Virtualization for long lists.
- Avoiding layout thrashing (measurements in useLayoutEffect).

Practice:

- Measure and fix re-renders using React Profiler.
- Identify and memoize expensive derived values.

9. Debugging & Tooling

- React DevTools (render count, props/state diff).
- Network tab for API and WebSocket events.
- Console debugging + breakpoints in VSCode/Chrome.
- Inspecting component tree performance.
- Common debugging patterns:
    - Finding state updates causing double renders
    - Locating async bugs
    - Tracing state transitions

Practice:

- Debugging “why did this re-render?”
- Tracing API errors through hooks (useEffect + fetch).

10. General Frontend Fundamentals

- DOM, event bubbling, capturing, delegation.
- Browser APIs (Fetch, AbortController, localStorage, sessionStorage).
- Security best practices (CORS, CSRF, CSP basics).
- Accessibility basics (roles, ARIA, focus management).
- Responsive design, CSS layouts (Flexbox, Grid).
- Performance metrics (TTFB, CLS, LCP).

General Practice building a small project that includes:

- A form with validation and autosave.
- A data list fetched via React Query.
- A live updates component
- A few tests (validation, success flow). Then, **profile and debug** it