Senior Frontend Engineer

Interview Preparation Sheet

JavaScript · TypeScript · React · Next.js · Browser & Performance
Security · Machine Coding · System Design · Build Tools · DSA

150	6	13
Questions	DSA Patterns	Topic Sections

Difficulty Legend

Easy | Medium | Hard

v7.0 · June 2026 · FAANG Edition · Curated · Structured Learning Order

Contents

13 Leadership & Behavioural	1E 2M 3H — 6Q
DSA — 25 Questions, 6 Patterns	25Q

Phase 1 — JavaScript: Core & Advanced

01 JavaScript Core Concepts 3E · 3M · 3H — 9Q

001	Easy	Explain var, let, and const — scope, hoisting, temporal dead zone, and mutability differences.
002	Easy	Difference between null, undefined, and NaN? How do type-coercion rules affect comparisons involving each?
003	Easy	Difference between function declaration, function expression, and arrow function — how does each affect this binding and hoisting?
004	Med	Explain the JavaScript event loop: call stack, Web APIs, microtask queue, and macrotask queue. What executes first — Promise.then or setTimeout(0)?
005	Med	What is a closure? Describe the classic var-in-loop bug and three ways to fix it.
006	Med	Explain == vs === with at least three type-coercion surprises interviewers love to use.
007	Hard	How does async/await work under the hood? Explain the relationship to Promises, microtasks, and generators.
008	Hard	Explain 'this' in: regular functions, arrow functions, class methods, event handlers, and after .bind() / .call() / .apply().
009	Hard	What are generator functions? Show how you'd use one for lazy pagination or a cancellable async workflow.

02 JavaScript Coding — Utility Implementations 2E · 4M · 3H — 9Q

010	Easy	Implement debounce(fn, delay) — then add a leading-edge option.
011	Easy	Implement throttle(fn, delay) — explain the difference from debounce with a UI example.
012	Med	Implement an EventEmitter with on, off, emit, and once.
013	Med	Implement Promise.all, Promise.allSettled, and Promise.race from scratch.
014	Med	Implement deep clone: handle objects, arrays, Date, Map, Set, null, and circular references.
015	Med	Implement curry(fn) so curry(add)(1)(2)(3) === add(1,2,3) for any arity.
016	Hard	Implement an LRU Cache with O(1) get and put using HashMap and Doubly Linked List.
017	Hard	Implement pipe(...fns) and compose(...fns) — then an async pipe that awaits each step.
018	Hard	Implement a task scheduler: run at most N async tasks concurrently from a queue of M.

02B JavaScript Design Patterns 1E · 2M · 3H — 6Q

019	Easy	Explain the Module pattern vs the Revealing Module pattern — how do ES modules improve on both?
020	Med	Observer vs Pub/Sub pattern — key difference and when does each fit a frontend use case?
021	Med	Decorator pattern without classes — wrap a fetch call to add retry, logging, and auth headers via pure function composition.

022	Hard	Command pattern — implement an undo/redo stack for a text editor using a command queue.
023	Hard	Proxy pattern for reactive state — implement a minimal Vue-style reactivity system using ES6 Proxy.
024	Hard	Mediator pattern — design a component communication bus for a microfrontend shell where iframes post typed messages through a central mediator.

Phase 2 — TypeScript

03 TypeScript — Type System & Patterns 3E · 3M · 4H — 10Q

025	Easy	Difference between unknown, any, never, and void — when is each the correct choice?
026	Easy	What does strict mode enable? Name the five most impactful compiler checks.
027	Easy	interface vs type alias — when does the distinction actually matter?
028	Med	Explain generics with a real custom hook: useFetch with loading / error / data states.
029	Med	Utility types: Partial, Required, Readonly, Pick, Omit, ReturnType, Parameters, Awaited — when do you reach for each?
030	Med	Discriminated union — type a multi-state UI component: idle | loading | success | error.
031	Hard	Conditional types, the infer keyword, and mapped types — write a DeepReadonly using all three.
032	Hard	Template literal types — show a real example: typed event names or CSS property accessors.
033	Hard	Type a fully type-safe generic DataTable where column accessors are bound to keys of T.
034	Hard	Declaration merging and module augmentation — extend Express.Request or a third-party component's props.

Phase 3 — React: Core, Hooks, Advanced Patterns & Next.js

04 React Core & Hooks 3E · 4M · 3H — 10Q

035	Easy	Virtual DOM and reconciliation — how does React's diffing algorithm decide what to re-render?
036	Easy	React keys — why is array index as key a bug? Give a concrete broken example.
037	Easy	useRef — three distinct use cases: DOM access, mutable value storage, preserving value without re-render.
038	Med	useEffect vs useLayoutEffect vs useInsertionEffect — when do you use each and why?
039	Med	React context and re-renders — three patterns to prevent unnecessary consumer re-renders.
040	Med	Suspense in React 18/19 — how does it work with lazy loading and the use() hook for data fetching?
041	Med	React.memo vs useMemo vs useCallback — when does over-memoisation hurt more than it helps?
042	Hard	React Fiber — render phase vs commit phase, diffing heuristics that break down, and concurrent interruption.

043	Hard	useTransition vs useDeferredValue — real use case for each and how they interact with Suspense.
044	Hard	React Server Components — hard constraints vs Client Components: what can't they do and why?

05 Advanced React Patterns 1E · 3M · 4H — 8Q

045	Easy	Prop drilling — Context vs component composition vs state manager: when do you reach for each?
046	Med	Compound component pattern — build a Tabs example showing how it eliminates a monolithic props API.
047	Med	Fully controlled vs uncontrolled components — when should a library component support both modes?
048	Med	Polymorphic components (as prop) with full TypeScript safety so infers anchor props.
049	Hard	Render Props vs HOCs vs custom hooks — give a 2026 scenario where each is still the right choice.
050	Hard	Headless component library design (Radix-style) — core API principles and accessibility contracts.
051	Hard	React 19: use() hook, Server Actions as form actions, useOptimistic — how do they reshape data fetching?
052	Hard	Plugin system in React — allow third-party code to inject UI into slots without coupling (registry pattern).

05B React Performance Patterns 1E · 2M · 3H — 6Q

053	Easy	What is tearing in concurrent React? When can it happen with external stores and how does useSyncExternalStore prevent it?
054	Med	Explain the windowing / virtual list technique — implement a fixed-height row virtualiser from scratch without a library.
055	Med	Avoid expensive child re-renders using children-as-props (lifting content up) without useMemo — explain why this works structurally.
056	Hard	React Compiler (React Forget) — how does auto-memoisation work at the compiler level and what manual patterns does it replace?
057	Hard	Islands architecture in React — selective hydration, how it maps to Next.js Partial Prerendering, and bundle-size impact.
058	Hard	Design a render budget system: cap render work per frame using scheduler.scheduleCallback and startTransition to keep INP under 200ms.

06 Next.js & Modern SSR 1E · 2M · 4H — 7Q

059	Easy	SSR vs SSG vs ISR vs CSR — concrete example of when you'd choose each.
060	Med	Next.js App Router architecture — how does it differ from Pages Router in rendering model and data fetching?
061	Med	revalidatePath vs revalidateTag — practical cache invalidation example for each.
062	Hard	Streaming SSR — how it works with Suspense boundaries and how it improves TTFB and LCP.
063	Hard	Partial Prerendering (PPR) — static shell + dynamic islands at the component level.
064	Hard	Server Actions — walk through optimistic updates, error states, and revalidation in a form submission.

065	Hard	Next.js caching layers — full route cache, data cache, router cache, request memoization: how they compose and conflict.

07 State Management 1E · 2M · 2H — 5Q

066	Easy	Server state vs client state — why does this distinction change your library choice?
067	Med	Zustand vs Jotai vs Redux Toolkit vs React Query — trade-offs, use cases, re-render performance.
068	Med	Zustand selector model — how does it prevent unnecessary re-renders? Show with code.
069	Hard	State design for a real-time trading dashboard: 20+ components, WebSocket data, optimistic updates, rollback on error.
070	Hard	React Query staleTime vs gcTime — tune for a live feed. How do optimistic mutations roll back cleanly?

Phase 4 — Browser, Performance & Security

08 Browser Internals & Networking 1E · 2M · 3H — 6Q

071	Easy	CORS preflight — when triggered, which headers required, risk of wildcard config.
072	Med	Full browser rendering pipeline from HTML bytes to pixels — where does JS block rendering and how do you fix it?
073	Med	Web Workers and SharedArrayBuffer — real use case: image processing or large data transform off the main thread.
074	Hard	URL to Enter: DNS, TCP, TLS handshake, HTTP/2 multiplexing, browser rendering — step by step.
075	Hard	V8 optimisation — JIT compilation, hidden classes, inline caches, and what triggers deoptimisation.
076	Hard	Browser memory model — heap, GC generations, diagnosing and fixing memory leaks in a React SPA with DevTools.

09 Performance Optimisation 1E · 2M · 3H — 6Q

077	Easy	preload vs prefetch vs preconnect — concrete Next.js example for each.
078	Med	LCP, INP, CLS — diagnose each with Chrome DevTools and fix the most common root causes.
079	Med	Performance API and Long Tasks observer — diagnosing main-thread jank in a React SPA.
080	Hard	Virtualise 100,000 rows updating in real time from WebSocket without dropping frames.
081	Hard	Code splitting strategies — route-level, component-level, library-level: measure and validate impact.
082	Hard	Critical rendering path — full optimisation strategy for a React SPA targeting sub-2s LCP on 3G.

10 Security 1E · 2M · 3H — 6Q

083	Easy	Same-Origin Policy and CORS — danger of Access-Control-Allow-Origin: *.
084	Med	JWT storage — HttpOnly cookies vs localStorage: security trade-offs for each.

085	Med	dangerouslySetInnerHTML — security implications and when DOMPurify is not enough.
086	Hard	XSS — stored, reflected, DOM-based. Show a React-specific DOM-based vector and how to close it.
087	Hard	CSRF — how SameSite cookie attribute and a CSRF token together close the attack surface.
088	Hard	OAuth 2.0 / OIDC from a SPA perspective — PKCE: why required for public clients, token exchange flow.

Phase 5 — Build Tools & Web Internals

10B Build Tools & Module Systems 2E · 2M · 3H — 7Q

089	Easy	CommonJS vs ES Modules — differences in resolution, tree-shaking eligibility, and browser support.
090	Easy	What is tree-shaking? What makes a module tree-shakeable and what patterns silently defeat it?
091	Med	Vite vs Webpack — key architectural difference (native ESM dev server vs bundled), and when you'd still choose Webpack.
092	Med	Code splitting with dynamic import() — how webpack chunks are named, loaded, and how to avoid waterfall loading.
093	Hard	Rollup vs esbuild vs SWC — compare transform speed, plugin ecosystems, and which is right for a library vs an app.
094	Hard	Module Federation (Webpack 5) — host/remote contract, shared dependency deduplication, and runtime version negotiation.
095	Hard	Design a monorepo build pipeline: Turborepo task graph, caching strategy, affected-only CI, and publishable package versioning with Changesets.

10C Internal Web App Architecture 1E · 2M · 3H — 6Q

096	Easy	Difference between a SPA, MPA, and hybrid app — when does each win on Core Web Vitals?
097	Med	Feature flags in a frontend app — implement a flags context supporting remote config (LaunchDarkly-style), local overrides, and typed flag keys.
098	Med	Error boundaries in production — pair React error boundaries with a monitoring SDK (Sentry) to capture component stack, user context, and breadcrumbs.
099	Hard	Design a client-side analytics SDK: batched beacon sends, offline queue with IndexedDB, page visibility flush, and privacy-safe PII scrubbing.
100	Hard	Progressive Web App (PWA) — service worker lifecycle (install, activate, fetch), cache-first vs network-first, background sync, and push notifications.
101	Hard	Design a design token pipeline: raw JSON tokens to Style Dictionary to CSS custom properties + Tailwind config + TypeScript types, with automated version diffs on PR.

Phase 6 — Machine Coding, System Design & Behavioural

11 Machine Coding — React & JavaScript 2E · 4M · 4H — 10Q

102	Easy	Build a Tabs component: keyboard nav (ArrowLeft/Right, Home/End) and correct ARIA roles.
103	Easy	Build a debounced search input: async API call, loading spinner, cancel stale requests on rapid typing.
104	Med	Build a toast system: queue, configurable auto-dismiss, manual dismiss, success/error/info variants.
105	Med	Build a modal system: nested modals, Tab-cycle focus trap, Escape to close, backdrop click, animation.
106	Med	Build a resizable split-pane: draggable divider, horizontal and vertical modes, persist ratio to localStorage.
107	Med	Build a calendar date-picker: month nav, range selection, disabled dates, full keyboard accessibility.
108	Hard	Build a virtual scroll list: only visible rows rendered from 50,000 items, variable row heights, smooth wheel scroll.
109	Hard	Build a Kanban board: drag-and-drop cards across columns using Pointer Events API (no libraries), add/delete, persist state.
110	Hard	Build an autocomplete: async fetch, debounce, keyboard nav, highlight matched substring, ARIA combobox pattern.
111	Hard	Build a data grid: client-side sort, column filter, pagination, column resize via drag, multi-row Shift+Click selection.

12 Frontend System Design 0E · 3M · 5H — 8Q

112	Med	Design an infinite scroll feed: skeleton loading, optimistic like/unlike, error recovery, scroll-position restoration on back-nav.
113	Med	RBAC in a React frontend for multi-tenant SaaS: route guards, component-level gating, token claims.
114	Med	Multi-step form wizard: per-step validation, draft auto-save, navigation guards, deep-linkable steps.
115	Hard	Google Docs-style collaborative editor: CRDTs/OT, WebSocket sync, cursor presence, offline conflict resolution.
116	Hard	Notification system at Slack/GitHub scale: real-time WebSocket, grouping, read state, pagination, cross-tab sync.
117	Hard	File upload component: drag-and-drop, chunked multipart, pause/resume, per-chunk progress, virus-scan webhook.
118	Hard	Design system component library: token architecture, accessible headless primitives, versioning, documentation site.
119	Hard	Microfrontend architecture: shell app, module federation, shared auth, independent deploys, cross-MFE event bus.

13 Leadership & Behavioural 1E · 2M · 3H — 6Q

120	Easy	What does 'senior engineer' mean to you? How do you articulate the gap between senior and staff impact?
121	Med	Tell me about a time you disagreed with a technical or product decision — what did you do?
122	Med	How do you balance shipping velocity with code quality in a fast-moving startup?
123	Hard	Most technically complex frontend problem you've solved — trade-offs, approach, outcome.

124	Hard	How do you drive adoption of a new tool or pattern across a resistant team?
125	Hard	Describe eliminating significant technical debt — how did you justify the investment and measure success?

DSA — 25 Questions, 6 Patterns

FAANG frontend roles typically have 1–2 LeetCode rounds at Medium difficulty. Master these 25 before your first screen. Focus on the pattern, not the solution.

Pattern 1 — Arrays & Hash Map Warm-up & most common

D01	Easy	Two Sum — hash map complement lookup O(n).
D02	Easy	Contains Duplicate — hash set membership O(n).
D03	Med	Group Anagrams — sort each string as map key; bucket results.
D04	Med	Top K Frequent Elements — hash map + bucket sort or min-heap.
D05	Med	Product of Array Except Self — prefix + suffix passes, no division, O(1) extra space.

Pattern 2 — Two Pointers Strings, arrays, dedup

D06	Easy	Valid Palindrome — two pointers inward, skip non-alphanumeric.
D07	Med	3Sum — sort + two-pointer, skip duplicates carefully.
D08	Med	Container With Most Water — greedy shrink shorter side.
D09	Hard	Trapping Rain Water — two-pointer with running max walls.
D10	Hard	Merge Intervals — sort by start, merge overlaps. Maps to calendar UI.

Pattern 3 — Sliding Window Substrings, subarrays

D11	Easy	Best Time to Buy and Sell Stock — sliding min price, single pass.
D12	Med	Longest Substring Without Repeating Characters — variable window + last-seen map.
D13	Med	Find All Anagrams in a String — fixed window + freq diff count.
D14	Hard	Minimum Window Substring — shrinkable window with need/have counters.

Pattern 4 — Stack & Queue Parsing, monotonic, LRU

D15	Easy	Valid Parentheses — stack bracket matching. Maps to code editor parsing.
D16	Med	Daily Temperatures — monotonic decreasing stack, next-greater-element.
D17	Hard	Largest Rectangle in Histogram — monotonic stack + area sweep.

Pattern 5 — Trees & BFS/DFS Virtual DOM, state trees

D18	Easy	Max Depth / Invert Binary Tree — recursive DFS base cases.
D19	Med	Binary Tree Level Order Traversal — BFS with queue, return levels array.
D20	Med	Lowest Common Ancestor of BST — BST property traversal O(h).

D21	Hard	Serialize and Deserialize Binary Tree — BFS encoding. Mirrors state persistence.

Pattern 6 — Dynamic Programming Autocomplete, schedulers

D22	Easy	Climbing Stairs — Fibonacci DP, most common DP entry point.
D23	Med	Coin Change — unbounded knapsack, min coins.
D24	Med	House Robber — 1D rolling max, no-adjacent constraint.
D25	Hard	Word Break — DP segmentation. Maps directly to autocomplete engines.