Interview Cheat Sheet solution

Frontend Answers

Easy

1. **Difference between `==` and `===`**
   - `==` checks equality after type coercion (e.g., `2 == "2"` → true).
   - `===` checks both value and type without coercion (`2 === "2"` → false).
   - Best practice: Always use `===` to avoid unpredictable behavior.
2. **Difference between `null` and `undefined`**
   - `null`: Explicitly set empty value, intentional absence of value.
   - `undefined`: Default for uninitialized variables or missing properties.
3. **Semantic HTML tags**
   - Tags with inherent meaning: `<header>`, `<footer>`, `<article>`, `<nav>`.
   - Improve accessibility, SEO, and code readability.
4. **Inline, inline-block, and block**

- Inline: Does not start new line, width = content (`<span>`).
- Block: Starts new line, width = full container (`<div>`).
- Inline-block: Behaves inline but allows width/height (`<img>`).

5. **Difference between `var`, `let`, `const`**
   - `var`: Function-scoped, hoisted, prone to bugs.
   - `let`: Block-scoped, no redeclaration.
   - `const`: Block-scoped, immutable reference.
6. **CSS Box Model**
   - Content → Padding → Border → Margin.
   - Total element size = content + padding + border + margin.
7. **CSS positioning**
   - Relative: Moves relative to normal position.
   - Absolute: Positioned relative to nearest positioned ancestor.
   - Fixed: Stays relative to viewport.
   - Sticky: Hybrid, switches between relative/fixed based on scroll.
8. **Synchronous vs Asynchronous JS**
   - Sync: Code executes line by line, blocking.
   - Async: Code can pause (e.g., network calls) and resume later via callbacks, promises, async/await.
9. **localStorage vs sessionStorage vs cookies**
   - localStorage: Persistent storage (until cleared).
   - sessionStorage: Temporary, cleared when tab closes.
   - cookies: Small data sent with requests, size-limited, used for authentication.
10. **Event bubbling vs capturing**

- Bubbling: Event propagates child → parent.

- Capturing: Event propagates parent → child.
- You can control with `addEventListener(type, handler, true/false)`.

Medium

1. **Virtual DOM**
   - Lightweight JS representation of DOM.
   - React diffs (reconciliation) and applies minimal DOM updates for performance.
2. **React Hooks**
   - Functions to use state & lifecycle in functional components.
   - Examples: `useState`, `useEffect`, `useMemo`, `useContext`.
3. **useEffect vs useLayoutEffect**
   - `useEffect`: Runs async after render/paint → non-blocking.
   - `useLayoutEffect`: Runs before paint → blocking, useful for DOM measurements.
4. **React Reconciliation**
   - Diffing algorithm compares Virtual DOM trees.
   - Uses keys to optimize list rendering.
5. **Controlled vs Uncontrolled Components**
   - Controlled: State managed via React (`value` + `onChange`).
   - Uncontrolled: Managed by DOM, access via refs.
6. **Service Worker in PWA**
   - JS worker between app and network.
   - Enables caching, offline access, background sync.
7. **Browser rendering pipeline**
   - Steps: JS → Style → Layout → Paint → Composite.

- Reflow: Layout recalculation.
- Repaint: Only visual changes.

8. **CORS**
   - Cross-Origin Resource Sharing.
   - Prevents unauthorized cross-domain requests.
   - Fixed by backend headers or frontend proxy.
9. **SSR vs CSR vs SSG**
   - SSR: Server renders HTML each request (Next.js).
   - CSR: Browser renders everything after JS load (React SPA).
   - SSG: Pre-built HTML files (Gatsby).
10. **Web Components & Shadow DOM**

- Web Components: Encapsulated reusable elements (`<custom-element>`).
- Shadow DOM: Scoped DOM/styles isolated from global CSS.

Senior

1. **React Fiber**
   - New reconciliation algorithm (incremental rendering).
   - Breaks rendering into units, schedules work based on priority.
2. **Hydration in SSR**
   - Server sends pre-rendered HTML, React attaches event listeners and makes it interactive.
3. **Optimizing large lists**
   - Virtualization (react-window, react-virtualized).
   - Infinite scroll, pagination, windowing.
4. **Code splitting**
   - Split JS bundle into smaller chunks.

- Implement via `React.lazy`, `import()`, Webpack.

5. **Browser JS optimization**
   - Parsing, Just-In-Time (JIT) compilation, hidden classes, inline caching.
6. **Tree Shaking**
   - Remove unused imports during bundling.
   - Works with ES6 module syntax (`import/export`).
7. **Reducing bundle size**
   - Lazy loading, compression (Brotli/Gzip), image optimization, removing polyfills.
8. **Drawbacks of CSR at scale**
   - Slower first render, SEO challenges, larger JS payload.
9. **Micro-frontend architecture**
   - Split monolithic frontend into smaller, independently deployable apps.
   - Integration via iframes, Webpack Module Federation, custom loaders.
10. **Advanced state management**

- Beyond Redux: XState (state machines), Recoil, Zustand, RxJS streams.

Rare but Asked

1. **Difference between repaint and reflow**
   - **Reflow (layout):** Occurs when changes affect the geometry or structure of the DOM, e.g., adding/removing elements, changing width/height, font size, etc. Reflows are expensive because the browser recalculates positions and layouts of elements.
   - **Repaint:** Occurs when only the visual appearance changes without affecting layout, e.g., color, background, visibility. Repaints are lighter than reflows but still impact performance if frequent.
   - **Optimization:** Minimize layout thrashing, batch DOM updates, use `transform` and `opacity` for animations.

2. **How DNS resolution affects frontend performance**
    - Every request to a new domain requires a DNS lookup, translating the domain into an IP address.
    - **Impact:** Adds latency (usually 20–120ms per lookup).
    - **Optimization:** Use DNS caching, CDN to reduce lookups, or preconnect to frequently used domains.
3. **Difference between `async` and `defer` in script tags**
    - `async`**:** Script downloads in parallel with HTML parsing, executes immediately when ready, can block DOM parsing if ready early.
    - `defer`**:** Script downloads in parallel but executes after HTML parsing, preserving execution order.
    - **Use:** `defer` is preferred for scripts dependent on DOM; `async` for independent scripts.
4. **Browser caching (ETag, Cache-Control, etc.)**
    - **Cache-Control:** Directs browser how long to cache resources (`max-age`, `no-cache`, `must-revalidate`).
    - **ETag:** Unique identifier for resource versions; browser requests server only if resource changed.
    - **Expires header:** Older standard, sets absolute expiry date.
    - **Optimization:** Proper caching reduces redundant requests, speeds up page load.
5. **HTTP/1.1 vs HTTP/2 vs HTTP/3**
    - **HTTP/1.1:** One request per TCP connection; head-of-line blocking; plain text headers.
    - **HTTP/2:** Multiplexing multiple requests/responses over a single connection; header compression; server push.
    - **HTTP/3:** Runs over QUIC (UDP), faster connection setup, better handling of packet loss.

- **Impact:** HTTP/2/3 significantly reduce latency and improve page load for multiple resources.

6. **CSS specificity rules in depth**
   - Determines which CSS rule applies when multiple match.
   - **Hierarchy:** Inline styles > IDs > classes/attributes/pseudo-classes > elements/pseudo-elements.
   - Example:

```
#id .class p { color: red; } /* specificity: 1-1-1 */
p { color: blue; }          /* specificity: 0-0-1 */
```

     → Red wins because of higher specificity.

7. **Preventing layout shift (CLS)**
   - **CLS (Cumulative Layout Shift):** Unexpected movement of elements during page load.
   - **Prevention:**
     - Specify width/height for images, videos, iframes.
     - Reserve space for dynamic content (ads, banners).
     - Avoid inserting content above existing content.

8. `requestAnimationFrame`
   - API to schedule animations efficiently.
   - Browser calls the callback before next repaint.
   - Advantages: Smooth animations, synced with display refresh rate (~60fps), avoids unnecessary CPU work.

9. **WebSockets vs HTTP**
   - **HTTP:** Request-response protocol; connection closes after response.
   - **WebSockets:** Persistent bi-directional connection; real-time communication.

- Use cases: chat apps, live notifications, multiplayer games.

10. **How a browser paints text**
    - Steps:
        1. Layout determines position of text boxes.
        2. Text is shaped using font metrics and glyphs.
        3. Anti-aliasing applied.
        4. Compositing layer draws glyphs on screen.
    - Heavy DOM and complex font rendering can slow this process.
11. **Critical rendering path**
    - Sequence of steps browser takes to convert HTML/CSS/JS into pixels on screen.
    - Steps: HTML parsing → CSSOM creation → Render tree → Layout → Paint → Composite.
    - **Optimization:** Minimize CSS/JS blocking, inline critical CSS, defer non-essential scripts.
12. **Inline styles vs CSS-in-JS vs external CSS**
    - **Inline:** Applied directly to element (`style=""`), high specificity, not reusable.
    - **CSS-in-JS:** Styles written in JS files, scoped, dynamic, good for React components.
    - **External CSS:** Global stylesheets, cached by browser, reusable across pages.
13. **Optimizing multiple CSS animations**
    - Avoid animating properties that trigger reflow (width, height, top, left).
    - Prefer `transform` and `opacity` (GPU-accelerated).
    - Combine animations into a single layer when possible.
14. **React's concurrent mode**
    - Allows React to interrupt rendering to handle high-priority updates first.

- Improves responsiveness of complex apps.
- Uses “time slicing” and “suspense” to manage async rendering.

15. **RAIL performance model**
    - Framework for web performance: **Response, Animation, Idle, Load**
        - Response: <100ms
        - Animation: 60fps (16ms/frame)
        - Idle: 50ms tasks
        - Load: <1000ms interactive
    - Goal: Build fast, responsive experiences.
16. **Prefetch, preload, preconnect**
    - **preconnect:** Establish early connection (DNS/TCP/TLS).
    - **preload:** Fetch resource early, before browser discovers it.
    - **prefetch:** Fetch resource for future navigation (low priority).
17. **Lazy loading images/scripts**
    - Load resources only when needed (e.g., below-the-fold images).
    - Improves initial page load.
    - Implement via `<img loading="lazy">` or dynamic imports in JS.
18. **Why base64-encoded images**
    - Embeds image data directly in HTML/CSS.
    - Reduces HTTP requests for small images (icons).
    - Downsides: increases HTML size, cannot cache separately.
19. **Hydration mismatch vs reconciliation issue**
    - **Hydration mismatch:** SSR HTML differs from React client render → warnings, broken interactivity.
    - **Reconciliation issue:** React diffing fails to efficiently update DOM → unnecessary re-renders.

20. **Brotli compression**
    - Modern compression algorithm for text-based assets (HTML, CSS, JS).
    - Smaller size than Gzip → faster download → better frontend performance.

Backend Answers

Easy

1. **What is REST API?**
    - REST = Representational State Transfer.
    - Architecture style where resources are represented by URLs and accessed via standard HTTP methods.
    - Principles: statelessness, client-server separation, uniform interface.
2. **GET, POST, PUT, PATCH, DELETE**
    - **GET**: Retrieve resource (safe, idempotent).
    - **POST**: Create new resource (not idempotent).
    - **PUT**: Replace entire resource.
    - **PATCH**: Update partial resource.
    - **DELETE**: Remove resource.
3. **Middleware in Express.js**
    - Functions that run between request and response.
    - Used for authentication, logging, parsing, error handling.
    - Example: `app.use(express.json())`.

4. **SQL vs NoSQL**
    - SQL: Relational, structured schema, ACID (MySQL, PostgreSQL).
    - NoSQL: Flexible schema, distributed, BASE (MongoDB, Cassandra).
5. **ORM**
    - Object Relational Mapper: maps database tables to objects in code.
    - Examples: Sequelize (Node + SQL), TypeORM, Prisma, Mongoose (MongoDB).
6. **Request/Response cycle**
    - Client sends HTTP request → server parses request → processes logic → interacts with DB → sends response back.
7. **Sync vs Async in Node.js**
    - Sync: Blocks thread until complete.
    - Async: Uses callbacks, promises, or async/await to avoid blocking.
8. **JWT (JSON Web Token)**
    - Token format for stateless authentication.
    - Consists of Header, Payload, Signature.
    - Used to verify user without storing session on server.
9. **Authentication vs Authorization**
    - Authentication: Verifying identity (login).
    - Authorization: Checking permissions (access control).
10. **Load Balancing**
    - Distributes traffic across multiple servers.
    - Improves availability and reliability.
    - Algorithms: round-robin, least connections, IP hash.

Medium

1. **Event Loop in Node.js**
    - Single-threaded architecture.
    - Handles async operations via event loop + callback queue.
    - Phases: timers, pending callbacks, idle/prepare, poll, check, close.
2. **Clustering in Node.js**
    - Spawns multiple worker processes to utilize multi-core CPUs.
    - Master distributes requests to workers.
3. **Horizontal vs Vertical scaling**
    - Vertical: Add more resources (CPU, RAM) to one machine.
    - Horizontal: Add more servers to handle load.
4. **ACID properties**
    - Atomicity, Consistency, Isolation, Durability.
    - Guarantees reliability of SQL transactions.
5. **Database Indexing**
    - Data structure (usually B-tree, hash) that speeds up queries.
    - Speeds reads, can slow writes.
6. **CAP Theorem**
    - Distributed systems can only guarantee 2 of: Consistency, Availability, Partition Tolerance.
    - Example: MongoDB → CP, Cassandra → AP.
7. **Transactions in SQL**
    - Group of SQL queries executed atomically.
    - COMMIT → persist, ROLLBACK → undo.
8. **Rate Limiting**
    - Restricts number of requests per user/time.
    - Implement via Redis counters, token bucket, leaky bucket algorithms.

9. **Redis vs DB**
    - Redis: in-memory, super fast, key-value.
    - Used for caching, pub/sub, queues.
    - Traditional DB: persistent storage with complex queries.
10. **Message Queues (RabbitMQ, Kafka)**

- Queues decouple producer and consumer.
- Ensure reliable, async processing.
- Kafka handles high throughput streaming, RabbitMQ for smaller tasks.

Senior

1. **Concurrency in Node.js**
    - Node itself is single-threaded, but uses libuv to delegate async I/O to worker threads.
    - Event loop multiplexes tasks efficiently.
2. **Microservices architecture**
    - Split monolith into smaller, independently deployable services.
    - Communicate via APIs, message queues, or gRPC.
    - Challenges: data consistency, observability, network overhead.
3. **Database Sharding**
    - Splitting large dataset across multiple servers.
    - Improves scalability but adds complexity in joins and transactions.
4. **Distributed Transactions & 2PC**
    - Ensures atomicity across multiple databases.
    - 2-Phase Commit: Prepare → Commit.
    - Slow, risk of blocking if coordinator fails.
5. **Replication in MongoDB & PostgreSQL**

- MongoDB: Primary-Secondary replication.
- PostgreSQL: Streaming replication.
- Used for high availability and failover.

6. **gRPC vs REST vs GraphQL**
   - gRPC: Binary protocol, fast, strongly typed, streaming supported.
   - REST: Simple, text-based, widely adopted.
   - GraphQL: Client defines shape of data, avoids over-fetching.
7. **Securing APIs at Scale**
   - Rate limiting, input validation, HTTPS, OAuth2, API gateways, WAFs.
8. **Consistency in Distributed Systems**
   - Strong consistency: Immediate correctness (Spanner).
   - Eventual consistency: Data eventually consistent (DynamoDB).
9. **Debugging Memory Leaks in Node.js**
   - Tools: Chrome DevTools, heap snapshots.
   - Causes: global variables, event listeners not removed, caches.
10. **Observability in Backend**

- Three pillars: Logs, Metrics, Tracing.
- Tools: ELK stack, Prometheus + Grafana, Jaeger.

Rare but Asked Backend Answers

1. **How does DNS affect backend services?**
   - DNS resolves domain names to IP addresses.
   - Backend services rely on DNS for service discovery and communication between microservices.
   - Slow or failed DNS resolution can add latency or cause request failures.

- **Optimization:** Use DNS caching, local resolvers, or service discovery mechanisms like Consul or Kubernetes DNS.

2. **HTTPS handshake**
   - Establishes secure connection between client and server.
   - Steps:
     1. Client hello (supported TLS versions and ciphers).
     2. Server hello (chosen cipher, certificate).
     3. Key exchange (public key encryption).
     4. Session key creation (symmetric encryption for actual data).
   - Ensures encryption, integrity, and server authentication.
3. **HMAC (Hash-based Message Authentication Code)**
   - Combines a secret key with a hash function to verify data integrity and authenticity.
   - Commonly used in API request signing or token verification.
   - Example: `HMAC_SHA256(key, message)`.
4. **Symmetric vs Asymmetric encryption**
   - **Symmetric:** Same key for encryption/decryption (AES). Fast, used for bulk data.
   - **Asymmetric:** Public/private key pair (RSA, ECC). Slower, used for key exchange, signatures.
5. **OAuth2 Authorization Code flow**
   - Secure way for clients to access resources on behalf of a user.
   - Steps:
     1. User authenticates with auth server.
     2. Auth server returns an authorization code to client.
     3. Client exchanges code for access token.
     4. Client uses token to access resource server.

6. **TLS termination in load balancers**
    - LB decrypts HTTPS traffic before sending it to backend servers.
    - Reduces CPU overhead on servers.
    - Simplifies certificate management, can inspect traffic for routing/security.
7. **Optimistic vs pessimistic locking**
    - **Optimistic:** Assume no conflicts, check at commit. Good for low contention.
    - **Pessimistic:** Lock resource immediately to prevent conflicts. Used in high-contention scenarios.
8. **Zookeeper**
    - Distributed coordination service.
    - Manages configuration, leader election, naming, distributed locks, and synchronization in clusters.
9. **Consistent hashing**
    - Maps keys to nodes in a way that minimizes remapping when nodes join/leave.
    - Useful for sharded caches, distributed databases.
10. **Monolith vs SOA vs Microservices**
    - **Monolith:** Single codebase, tightly coupled.
    - **SOA:** Service-Oriented Architecture, heavier inter-service communication, often uses an enterprise service bus.
    - **Microservices:** Lightweight, independently deployable services, communicate via APIs or messaging.
11. **Circuit breaker pattern**
    - Prevents cascading failures when a service is failing.
    - Opens the circuit to stop requests temporarily, retries after cooldown.
    - Improves fault tolerance in distributed systems.

12. **Raft consensus algorithm**
    - Ensures consistency in distributed systems.
    - Components: Leader election, log replication, safety.
    - Leader receives client requests, replicates logs to followers, commits when majority agrees.
13. **Docker container vs VM**
    - **Container:** Lightweight, shares host OS kernel, fast startup.
    - **VM:** Full OS virtualization, heavier, slower startup, complete isolation.
14. **Kubernetes scaling**
    - Horizontal Pod Autoscaler (HPA) adjusts pod count based on metrics (CPU, memory, custom).
    - Cluster Autoscaler adjusts number of nodes based on resource demands.
15. **Leader election in distributed systems**
    - Ensures a single node coordinates actions to avoid conflicts.
    - Implemented via algorithms like Raft or using Zookeeper.
16. **Eventual consistency in DynamoDB**
    - Writes propagate asynchronously to replicas.
    - Reads may return stale data until replicas catch up.
    - Trades strict consistency for high availability and performance.
17. **Kafka message ordering**
    - Messages are ordered within a partition, not across partitions.
    - Producers can control which partition a message goes to via key hashing.
18. **Synchronous vs asynchronous replication**
    - **Synchronous:** Master waits for replica acknowledgment → safer but slower.
    - **Asynchronous:** Master doesn't wait → faster, risk of temporary inconsistency.

19. **Sidecar containers**
    - Helper container running alongside main application container.
    - Handles logging, monitoring, proxying, or security without modifying main app.
20. **Debugging high-latency issues**
    - Identify bottleneck: CPU, memory, DB, network, external APIs.
    - Tools: APMs (New Relic, Datadog), tracing (Jaeger, OpenTelemetry), profiling.
    - Steps: Measure, isolate, optimize, monitor.