Resource 1 - Performance optimization

Article/Website Name:

“Front-End Performance Optimization: A Comprehensive Guide,” by Narender Reddy Karka

Link to resource:

https://doi.org/10.32628/CSEIT251112389

Notes:

- Studies in cognitive psychology show that users perceive interactions as "instantaneous" when they occur within 100ms, as "responsive" when they occur within 300ms, and as "sluggish" when they take longer than 1 second.
- Users judge a site's speed not just by how long it takes to fully load, but by how quickly they can see useful content and begin interacting with it.
- Performance optimization includes:
    - Network optimization
        - Reducing the number and size of HTTP requests
    - Rendering optimization
        - Eliminating render-blocking resources, implementing critical rendering path optimization, and employing server-side rendering.
    - JavaScript execution optimization
        - Reducing main thread blocking. This includes optimizing third-party scripts that contribute significantly to performance degradation.
    - Resource delivery optimization
        - Implementing intelligent caching strategies, service workers, and predictive prefetching. This creates resilient experiences that work well across varying network conditions.
- **Largest Contentful Paint (LCP)** measures loading performance by how quick the largest content element is visible to users.
    - Implementation strategies:
        - Optimize server response times through efficient backend code, proper database indexing, and caching layers
        - Eliminate render-blocking CSS and JavaScript through critical path optimization

        - Implement resource hints like preload, preconnect, and prefetch for critical assets
        - Use responsive image techniques with proper sizing and format selection
        - Consider server-side rendering or static generation for faster initial content delivery
        - Optimize web font loading to prevent text invisibility during page load
- **First Input Delay (FID)** measures when a user first interacts with a page and when the browser can respond to that interaction. Directly impacted by JavaScript execution time and main thread blocking
    - Implementation strategies:
        - Break up long-running JavaScript tasks into smaller chunks under 50ms
        - Defer or remove non-critical third-party scripts that may compete for main thread resources
        - Use Web Workers to move intensive computations off the main thread
        - Implement efficient event delegation patterns instead of numerous individual event listeners
        - Optimize JavaScript execution by removing unnecessary code and dependencies
        - Employ requestAnimationFrame and requestIdleCallback for non-essential processing
        - Implement debouncing and throttling techniques for frequent events like scrolling or resizing
- **Cumulative Layout Shift (CLS)** measures how much unexpected movement of visible elements occurs during the page loading process. This causes users to misclick or lose their place.
    - Implementation strategies:
        - Always specify width and height attributes on images, videos, iframes, and other embedded content
        - Reserve adequate space for dynamic content like ads or embeds using aspect ratio boxes
        - Avoid inserting new content above existing content unless in response to user interaction
        - Implement content placeholders that match the expected final dimensions
        - Use transform animations instead of properties like width/height that trigger layout recalculation

        - Ensure web fonts are loaded efficiently to minimize text reflow
        - Structure CSS to establish layout early in the page load process
- **Render Blocking Optimization: Critical CSS Management**
    - Implementation strategies:
        - Analyze viewport content: Identify which HTML elements appear in the initial viewport across key device sizes and breakpoints
        - Extract critical rules: Use automated tools or manual analysis to determine which CSS rules apply to these above-the-fold elements
        - Minimize the critical CSS: Remove unnecessary whitespace, comments, and optimize selectors to keep the inline CSS as small as possible (ideally under 14KB to fit within initial TCP packets)
        - Inline the critical CSS: Place these extracted styles directly in the <head> of the document to eliminate the need for an external request
        - Load remaining CSS asynchronously: Use resource hints and non-blocking patterns to load the complete stylesheet without blocking rendering
        - Implement fallbacks: Include proper handling for browsers with JavaScript disabled
- **Resource Optimization: Image and JavaScript Asset Optimization Pipeline**
    - Optimization strategies:
        - Format selection: Choosing the optimal image format based on content type and browser support. Vector formats (SVG) for graphics and logos, WebP/AVIF for photographic content with broad browser support, and JPEG/PNG as fallbacks. This often involves content negotiation or the <picture> element to serve different formats to different browsers.
        - Responsive images: Implementing techniques to serve appropriately sized images based on viewport dimensions, pixel density, and network conditions. This includes using the srcset and sizes attributes to provide multiple resolution options, as well as art direction through the <picture> element to crop or modify images for different screen sizes.
        - Lazy loading: Deferring the loading of off-screen images until users scroll near them, reducing initial page weight and prioritizing visible content. This can be implemented using the native loading="lazy" attribute or more sophisticated intersection observer-based solutions for older browsers.
        - Content-aware compression: Applying variable compression levels based on image content type, importance, and placement. Critical

hero images might use higher quality settings, while supplementary images could use more aggressive compression. Modern tools can analyze image content to apply compression intelligently, preserving detail in important areas while compressing background elements more aggressively.
        - Image CDNs and transformation services: Implementing on-the-fly image optimization through specialized CDNs that can resize, format, compress, and cache images automatically based on request parameters.
        - Next-gen formats: Adopting cutting-edge formats like AVIF which offer superior compression-to-quality ratios, with appropriate fallback mechanisms for browsers that don't yet support them.
- **HTTP/2 and Caching Optimization:**
    - Optimization techniques:
        - Resource prioritization: Implementing server-side prioritization hints that tell browsers which resources are most critical, ensuring bandwidth is allocated efficiently. This includes using HTTP/2 PRIORITY frames and headers to indicate the relative importance of different assets.
        - Connection pooling: Configuring servers to maintain persistent connections efficiently, allowing multiple requests to reuse the same TCP connection. This reduces connection establishment overhead and takes advantage of HTTP/2's multiplexing capabilities.
        - Strategic compression: Implementing content-specific compression algorithms beyond simple GZIP, such as Brotli for text-based assets (offering 15-25% better compression than GZIP) and specific optimizations for different content types. This includes configuring appropriate compression levels that balance CPU usage with compression ratio.
        - Cache policy optimization: Implementing sophisticated caching directives that balance freshness and performance. This includes using "immutable" cache control directives for versioned assets, implementing ETags and conditional requests efficiently, and varying cache durations based on content type and update frequency.
        - Server push: Strategically implementing HTTP/2 server push to proactively send critical resources to the client before they're explicitly requested, particularly for render-blocking resources needed during initial page load.

        - Domain consolidation: Reversing the HTTP/1.1 best practice of domain sharding, instead consolidating resources onto fewer domains to maximize connection reuse under HTTP/2's multiplexing capabilities.
- **Performance monitoring:**
    - Implementation strategies:
        - Define critical metrics: Identifying the specific performance indicators most relevant to your application and user experience. This might include Core Web Vitals (LCP, FID, CLS), custom metrics like Time to Interactive or First Meaningful Paint, or business-specific metrics like checkout completion time for e-commerce sites.
        - Set appropriate thresholds
        - Implement accountability measures

Resource 2 - Overview of trends in front-end engineering

Article/Website Name:

“A Decade of Progress in Front-End Engineering: A Review of Trends, Technologies, and Challenges in Modern Software Development,” by Harish Reddy Bonikela

Link/DOI to resource:

http://dx.doi.org/10.47392/IRJASH.2025.068

Notes:

- Five key fronts of front-end engineering:
    - Design approach
    - Frameworks and technology
    - Performance optimization
    - Accessibility
    - Developer productivity
- Conclusions found by other papers:

    - AngularJS provided more complete tooling than Backbone.hs and Ember.js
    - AngularJS's two-way data binding and MVC design improved developer productivity but added runtime complexity
    - Automated testing tools like Axe give better accessibility outcomes
    - React offered best performance for large-scale apps, Vue excelled in simplicity, and Angular was best suited for enterprise use
    - Lean UX and design tokens allow for more effective collaboration
    - PWAs significantly improved offline experience. Recommended for use in mobile-focused apps/websites
    - Low-code tools improve prototyping speed but sacrifice flexibility and code maintainability in long-term projects
    - Growing need for unified tooling and better onboarding documentation
- The integration of design tools like Figma, Adobe XD, and Sketch into the development pipeline ensures seamless translation from wireframes to code
- Tools like Lighthouse and Web Vitals have become essential in quantifying performance benchmarks
- Accessibility tools: Axe, Lighthouse Accessibility
- Platforms like Vite, Webpack, and Parcel support efficient development environments tailored to modern engineering workflows.
- Vue offers simplicity and fast onboarding, React excels in ecosystem flexibility and scalability. Angular remains powerful but may lag in performance and accessibility metrics unless carefully configured
- Adoption of SSR, static site generation, and progressive web apps has led to measurable improvements in load time and SEO
- A strong focus on developer experience translates to faster development cycles and reduced error rates, enhancing overall software quality

Resource 3 - Web Accessibility

Article/Website Name:

W3C Web Accessibility Initiative (WAI)

Link to resource:

https://www.w3.org/WAI/

Notes:

- Has resources on how to make video captions, audio and video, etc. accessible
- Free online course on “Intro to Web Accessibility” here

Resource 4 - Tools

Article/Website Name:

“Frontend Development Best Practices for 2025 - The Complete Developer's Guide,” by Amaresh Adak

Link/DOI to resource:

https://thesyntaxdiaries.com/frontend-development-best-practices

Notes:

- Good overview of best tools and practices. Includes code examples

Resource 5 - UI Design

Article/Website Name:

“Web design best practices to attract more website visitors,” by Nadya A.

Link/DOI to resource:

https://www.hostinger.com/tutorials/web-design-best-practices

Notes:

- **Branding**
    - Audience analysis
    - Competitor Analysis
- **Color Palette**
    - Color theory
        - Color themes: monochromatic, triadic, analogous

    - High contrast improves readability and accessibility.
    - 60/30/10 rule
        - Choose a primary color that will make up 60% of your site and a secondary color for 30%. The final 10% should use an accent color that either contrasts or complements the primary color.
    - Complementary colors
        - On a long-scrolling site, consider using complementary colors such as yellow and purple to distinguish sections.
- **Typography**
    - Limit number of fonts
    - Web-safe fonts
    - Proportional text size
        - A good rule of thumb is to set your website text to a minimum size of 16 pixels or 12 pt. To experiment with text sizes and preview how they will appear on the site, use online tools like Gridlover.
    - 50-75 characters per line
- **Visual Hierarchy:** guides visitors to important parts of the page
    - Symmetrical design
        - Place elements evenly across the page's centerline. For instance, if you have a visually heavy item on the right, you should add an equally heavy item on the left. This type of design is quite popular as it is convenient for all screen sizes.
    - Asymmetrical design
        - Arrangement does not follow the centerline. To achieve balance, designers combine colors and textures or manipulate perspective. For example, if you position a more prominent element near the centerline, it's good to place a smaller one a bit further from it.
    - Mosaic design
        - Generates a feeling of unease. Indicates motion and action, managing to draw attention with a modern and dynamic style.
- **Composition:** organization of elements to give the site a cohesive structure
    - Rule of thirds
        - Split a design or photo into thirds using a grid of nine boxes, providing guidelines to align text, adjust objects, and generally arrange elements.
- **Scale:** brings attention to important details
    - 3 sizes max:
        - Small, medium, and large sizes are enough to give you variety while keeping a clear website hierarchy. Generally, the sizes range

        - from 14 px to 16 px for body copy, 18 px to 22 px for subheadings, and up to 32 px for headings.
    - Enlarge important elements
        - Pick up to two key elements to enlarge so they will stand out.
- **Pattern:** people follow a viewing pattern to scan content, they generally follow the shape of F and Z
    - Z pattern
        - Readers scan the page from the top left to the top right. Then, they scan down diagonally to the lower left and across the page to the lower right. This pattern is ideal for designing pages with minimal copy and design elements, such as landing pages.
    - F pattern
        - Visitors scan the content from the top left to the top right and repeat the process on the following lines. Placing the most important content at the top is best to engage visitors as it is the first thing they see. This layout is a great fit for text-heavy sites.
- **Whitespace:** empty space between elements on a page. Used to break up text, direct attention to certain points, and streamline user experience. Color, texture, and images can be used as whitespace too.
    - Use micro and macro whitespace
        - Micro whitespace improves readability by putting sufficient space between words and paragraphs. Meanwhile, macro whitespace surrounds larger elements such as the website logo or headings, making the site design balanced and less cluttered.
- **Grouping:** items near each other are perceived as belonging to the same group.
    - Use whitespace to group and separate elements
    - Borders and backgrounds can help distinguish/group elements
- **Textures:** give contrast to design, reinforce site structure, and make text-heavy pages more readable
    - Simplicity in texture
    - Can use to divide sections or highlight images
    - Shouldn’t be the center of attention
- **Visuals**
    - Feature a hero image: displayed in the upper section of a page. Provides a first impression
    - Correct image formats improve site performance and user experience
- **Navigation**
    - Horizontal menu
    - Vertical sidebar menu
    - Drop-down menu

        - Lists all pages available after users click or hover over it.
    - Hamburger menu
        - Appears as a three-line icon. When visitors click on it, a drop-down or sidebar menu will show up to reveal the links. A great option for mobile sites as it uses less screen space.

Resource 6 - Website security

Article/Website Name:

“The Ultimate Guide to Secure Web Development: Frontend Best Practices”

Link to resource:

https://www.geeksforgeeks.org/javascript/the-ultimate-guide-to-secure-web-development-frontend-best-practices/

Notes:

- Common vulnerabilities:
    - **Cross-site scripting (XSS):** attackers put malicious code to content that users receive. This script can run in the target's browser, generating several attacks, like data theft, or session hijacking.
        - **Saved XSS:** malicious code is stored in a database and transmitted to users when they request data
            - Ex: if a comment section on a website doesn't properly clean inputs, an attacker could inject a script that gets executed every time the page loads.
        - **Reflected XSS:** malicious code shown on user’s browser through web app. Occurs when data is sent to a website (via query parameters, form submissions, etc.) then reflected back to the user without proper verification
        - **Impacts:**
            - Malware distribution
            - Data theft: stealing cookies, session tokens, or sensitive data from browser
            - Session hijacking: attacker copies user and operate actions on their support (e.g. passing funds or altering account setting) by taking session tokens
        - **Protections:**

            - Input validation and sanitization: confirm all user inputs are validated before being executed or retained
            - Content Security Policy (CSP): bounds sources from which scripts can be conducted
    - **Cross-site request forgery (CSRF):** forces authenticated user to run unwanted actions on a web app. When a user is logged into a web application, an attacker can fraud them into making requests to that application (e.g., submitting a form, changing settings, etc.) without their knowledge.
        - **Impacts:**
            - Unauthorized transactions
            - Compromise of user accounts
            - Data theft or manipulation
        - **Protections:**
            - CSRF tokens: use CSRF tokens for form submission that give data to server side. They're unique, changeable values generated by the server and added with form submissions. The server checks the token to confirm the request and that it came from the targeted user.
            - SameSite cookies: Make cross-origin requests that do not include cookies or turn the 'SameSite' attribute on. Restrictions on how cookies are sent with cross-site requests can help prevent CSRF attacks, though.
    - **Clickjacking:** you click on something different than what you see. This happens when there's a hidden layer over a real button or link.
        - **Impact:**
            - Unauthorized actions
            - Phishing attacks
        - **Protections:**
            - Content Security Policy (CSP): Apply the 'frame-ancestors' directive in your Content Security Policy to process which domains are approved to put in your content in iframes. For example, 'frame-ancestors 'self'' permits only your own domain to place your content.
            - UI redress testing: Frequently test your web applications for UI redressing risks, which can be exploited in clickjacking attacks. This includes checking iframes, external content embedding, and overlay elements.
    - **Man-in-the-Middle (MITM) Attacks:** attacker blocks communication between two parties without either knowing

        - **Impact:**
            - Data theft
            - Session hijacking
        - **Protections:**
            - Use HTTPS: Make sure all communication between the client and server is defended with HTTPS. HTTPS applies encryption to secure data, making it rigid for attackers to block and read the information.
            - Secure WiFi: Warn users to escape using public Wi-Fi networks for sensitive transactions. Use VPNs (Virtual Private Networks) to encrypt traffic over non-secure networks.
- Secure Authentication:
    - Strong passwords
        - Minimum length
        - Complexity requirements
        - Password expiration/updates
    - Use Multi-Factor Authentication (MFA)
        - One-time passwords (OTPS)
        - Biometrics
    - Reliable cookies
        - Kept session information in cookies marked as 'HttpOnly' and 'Secure' . The 'HttpOnly' flag prevents javascript access to cookies, lessens the risk of cross-site scripting (XSS) attacks. The secure flag assures cookies are only transmitted over HTTPS and securing them from man-in-the-middle attacks.
    - Session expiry
        - Timeouts to limit the duration of a session without active user interaction or absolute timeouts
    - Session IDs
        - Use long, unique session identifiers that are tough to guess. Never use linear or predictable session IDs. Moreover, store session IDs securely on the server side, and transmit them over secure channels (HTTPS).
    - Logout mechanisms
        - Make it easy for users to log out. Verify all session data is removed on the server side at logout
    - Account lockout
        - Lock out after multiple failed login attempts to prevent brute force attacks

- Data validation and sanitization
    - Type checking
        - Verify that the data matches the predicted type (e.g., integer, string, date). Like, if a field is planned to be an integer, then application should confirm that the input is numeric
    - Length checking
        - Validate that the length of the input is within suitable range. This is mainly necessary for fields like usernames and passwords. For example, a username should have a minimum and maximum length to avoid overly short or long entries.
    - HTML sanitization
        - When user input is shown on a web page, it must be filtered to prevent XSS attacks. It contains removing out or encoding possibly dangerous HTML tags and attributes. For example, a user might enter alert('XSS'), which should be cleaned to block execution.
    - SQL injection prevention
        - Escaping or parameterizing your sql queries so malicious code is not executed. Further, the use of prepared statements (along with parameterized queries), limits the ability that an attacker has to alter control structures as well.
- Secure storage of sensitive data
    - Encrypt sensitive data
    - Use strong encryption algorithms (e.g. AES. MD5 and SHA1 are outdated)
    - Use web cryptography API
    - Secure communication channels
        - Data in transfer between client and server **must** be transmitted over HTTPS
        - Use HSTS (HTTP Strict Transport Security)
        - Use an SSL/TLS certificate by a certified CA (Certificate Authority)
    - Regularly clear client-side data
        - Session expiration
        - Manual data clearing