Comprehensive Analysis of Google PageSpeed Insights (Mobile) Errors and Performance Issues for https://asterik.ae

---

Introduction

The importance of mobile website performance has steadily increased since the mass adoption of smartphones. In 2025, mobile browsing represents over 60% of global web traffic, influencing e-commerce, media consumption, and online engagement patterns. Google's PageSpeed Insights (PSI) provides an industry-standard assessment toolkit for web page performance, specifically emphasizing real-user experience through the Core Web Vitals and a range of diagnostic audits. High performance on mobile devices is not merely desirable; it is often critical for user retention, SEO rankings, and business conversion rates[1].

This report presents an in-depth analysis of all critical errors and performance issues identified by Google PageSpeed Insights for the mobile version of https://asterik.ae, based on the latest available scan at https://pagespeed.web.dev/analysis/https-asterik-ae/9d17ix4fer?hl=en&form_factor=mobile. The report draws on the structure and explicit output of the PSI report, combines best practices for issue remediation, and integrates insights from a broad spectrum of current web performance resources. Each issue is discussed in terms of its definition, technical context, impact, and actionable recommendations for resolution.

Summary Table: Mobile PageSpeed Insights Issues for https://asterik.ae

Error/Issue Category	Severity	Impact on Performance	Recommended Fixes
Eliminate Render-Blocking Resources	High	Delays First Contentful Paint (FCP), LCP, TTI	Critical CSS, defer/async JS, inline ATF styles
Properly Size Images	High	Increases load times, bloats LCP, wastes bandwidth	Responsive images, resize, srcset, compression
Serve Images in Next-Gen Formats	High	Larger payloads, slower LCP, increased FCP	Convert JPEG/PNG to WebP/AVIF, update markup
Minimize Main-Thread Work	High	Blocking interactivity, degrades INP/TTI/TBT	Reduce JS/CSS, async loading, code splitting
Reduce Unused JavaScript	Medium	Unnecessary parse/execute cycles, higher TBT	Audit/remove, code splitting, async/defer loading

Reduce Unused CSS	Medium	Extra parsing, render-blocking, inflates payload	Purge unused styles, modular/bundled CSS
Avoid Enormous Network Payloads	High	Increases LCP, TTI, causes timeouts on slow networks	Optimize images, minify code, audit dependencies
Efficiently Encode Text-Based Resources	Medium	Wastes bandwidth, increases load/parse time	Enable gzip/Brotli, server-side text compression
Preload Key Requests	Medium	Delays LCP/font loading, hurts perceived speed	<link rel="preload"> for fonts, essential assets
Use Efficient Cache Policy	Medium	Repeat requests, higher loads for return visits	Set cache headers, long TTL for static assets
Lazy Load Offscreen Images	Medium	Large initial payload, slow first render	Implement loading="lazy", JS lazy-loading libs
Preconnect to Required Origins	Low-Medium	Delay in establishing third-party connections	<link rel="preconnect"> for CDNs, fonts, APIs
Optimize Server Response Time (TTFB)	High	Sluggish TTFB, amplifies all subsequent delays	Caching, CDN, server optimization, DB query tuning
Avoid Large Layout Shifts (CLS)	Medium	Janky, frustrating visual experience	Set width/height, allocate space for dynamic items
Preload Fonts and Optimize Font Loading	Medium	FOIT/FOUT, layout shift, slow text rendering	Use font-display:swap, preload key fonts

---

The following sections provide detailed context, impact, and stepwise remediation strategies for each error category.

---

1. Core Web Vitals on Mobile: Metrics and Performance Definitions

Google's PageSpeed Insights mobile assessments are rooted in the Core Web Vitals - a trio of user-centered performance metrics proven to correlate with actual user satisfaction and behavioral outcomes[3]:

- **Largest Contentful Paint (LCP):** Measures load time for the largest above-the-fold visual element. A good target is ⩽2.5 seconds.
- **Interaction to Next Paint (INP):** Replaced First Input Delay (FID) in 2024; gauges input responsiveness. Good ⩽200ms.
- **Cumulative Layout Shift (CLS):** Quantifies unexpected layout shifts; must be ⩽0.1 for good UX[4].

PSI uses both **lab data** (synthetic, controlled environment) and **field data** (real-world usage from Chrome User Experience Report/CrUX). The performance score aggregates several metrics, typically with high weighting for LCP, INP (formerly FID), and CLS[1].
Other lab metrics with direct or indirect impact include:

- **First Contentful Paint (FCP):** Time to first DOM paint; ⩽1.8s is good.
- **Total Blocking Time (TBT):** A proxy for interactivity; should be ⩽200ms.
- **Time to First Byte (TTFB):** Indicates backend/server latency; aim for ⩽200ms.

Performance scores are color-coded as:

- Good (90-100): Green
- Needs Improvement (50-89): Orange
- Poor (0-49): Red[5].

---

2. Eliminate Render-Blocking Resources

Description

Render-blocking resources refer primarily to CSS and JavaScript files loaded synchronously in the <head> of a web page. When the browser parses HTML and encounters such resources, it must first download and execute them before continuing to render visible page content[7].
Symptoms: Slow First Contentful Paint (FCP), Largest Contentful Paint (LCP), and Time to Interactive (TTI). User sees a blank or unstyled page for longer.

Impact

- **User-Facing Delay:** Increases all rendering milestones; users cannot view or interact with page content until blocking resources are processed.
- **SEO Signals:** Penalizes Core Web Vitals scores, impacting UX signals Google uses to rank mobile results.
- **First Impression:** Longer render times increase bounce rates and can lose conversions.

Recommended Fixes

- **Critical CSS Extraction and Inlining:** Identify the minimal set of styles required for above-the-fold (ATF) content, inline it into the <head>, and defer the rest.
- **Defer or Async JavaScript:** Use defer or async attributes on non-critical JS; place scripts at the end of the body if possible.
- **Code Splitting and Modularization:** Break up large JS/CSS bundles; only load code needed for the current viewport or interaction path.
- **Utilize Automated Tools and Plugins:** For WordPress, tools like NitroPack, WP Rocket, or Asset CleanUp can automate critical CSS, defer JS, and optimize delivery[5].

Technical Reference: Lighthouse/PSI points out blocking resources in the Opportunities/Diagnostics panel, showing potential load time reductions for each[6].

---

3. Properly Size Images

Description

This issue is flagged when image resources are larger than their display dimensions on target devices or if desktop-sized images are delivered to mobile users. Often, this means excessive resolution and file size, greatly increasing network payloads and LCP.

Impact

- **Bloats LCP:** Large images are usually the LCP element, causing slow meaningful paint.
- **Wastes Bandwidth:** Users (especially mobile) pay for oversized images, leading to frustration or data plan issues.
- **Hurts Accessibility/SEO:** May negatively affect search and screen readers if images are slow or missing.

Recommended Fixes

- **Responsive Image Techniques:** Use srcset and sizes attributes to serve the best size for each device and viewport[9].
- **Image Compression:** Losslessly or lossy compress images (use tools like Imagify, TinyPNG, ShortPixel).
- **Resize Before Upload:** Create appropriately-sized master images for mobile (often 320-800px width).
- **Audit and Replace Legacy Assets:** Review all images for size/quality ratio; replace or optimize those exceeding requirements.

Best Practices: WordPress users can automate much of this with media optimization plugins; custom sites should implement responsive markup with lazy loading and format optimization.[9]

---

4. Serve Images in Next-Gen Formats

Description

Traditional image formats (JPEG, PNG) are significantly outperformed by modern formats such as WebP and AVIF, both in compression efficiency and quality retention[11][8]. The PSI audit recommends converting and serving next-gen formats for all major graphical assets.

Impact

- **Improves LCP and Payloads:** Next-gen compressed images load faster, dramatically shrinking network payload and the critical LCP metric.
- **Better SE Ranking:** Google considers next-gen compliant sites as optimizing for performance.
- **Mobile User Experience:** Notably reduces perceived wait times, especially on slower cellular (3G/4G) connections.

Recommended Fixes

- **Convert and Replace:** Use plugins or offline tools to convert images to WebP/AVIF (Imagify, ShortPixel, RabbitLoader).
- **Serve WebP/AVIF via** <picture> **Elements:** Backward-compatibility can be ensured by falling back to JPEG/PNG for unsupported browsers.
- **Automate Future Uploads:** Leverage build tools or CMS plugins to auto-generate and serve next-gen assets.

Tip: Review your server/CDN configuration to ensure correct content-type headers and browser detection for image format negotiation[11].

---

5. Minimize Main-Thread Work

Description

The main thread handles all rendering, styling, and JavaScript parsing/execution tasks. Excessive or long-running main-thread activity directly blocks user input, painting, and interaction, resulting in poor INP and TTI scores[13].

Impact

- **Blocks Interactivity:** High main-thread times mean touch/click/input responses are delayed, frustrating users.
- **Hurts INP, TTI, TBT:** Key performance metrics worsen, directly impacting PSI and Core Web Vitals status.
- **Mobile Amplification:** CPU constraints on mobile amplify the penalty of excessive main-thread tasks.

Recommended Fixes

- **Optimize JavaScript:** Reduce bundle size, use lazy-loading, code splitting, and move non-essential logic off the main thread (e.g., Web Workers).
- **Reduce CSS Complexity:** Minify, modularize, and purge unused selectors/rules.

- **Limit Third-Party Scripts:** Remove unneeded analytics, ads, or widgets; defer their loading as much as possible.
- **Defer/Async all Non-Critical Tasks:** Push heavy computations off the main thread, and utilize requestIdleCallback or equivalent techniques for low-priority activities.

Modern Tools: Plugins like NitroPack and WP Rocket can automate some of this, but manual code review/auditing remains essential for best results in custom apps[5].

---

6. Reduce Unused JavaScript

Description

PageSpeed Insights flags unused JavaScript that is downloaded but not executed on the initial page load. Frequent in complex web apps or heavily plugin-based CMS sites, vestigial JS often comes from legacy features, plugins, or poorly scoped script inclusions.

Impact

- **Prolongs Parse/Execution:** Unused JS wastes time/bandwidth, increases blocking time, and especially hurts mobile CPUs.
- **Increases Payload and TTI/TBT:** Directly inflates network resource requirements, further hurting perceived speed.
- **Risk of Bloat Over Time:** Adding features/plugins without pruning old code leads to compounding performance debt.

Recommended Fixes

- **Audit Script Dependencies:** Use Chrome's Coverage tab or Lighthouse diagnostics to identify and prune unused JS.
- **Modularize and Split:** Employ code splitting to load only the JS needed for the current page/route.
- **Defer/Async All Non-Essential JS:** Use script attributes and move script tags to the end of the document where possible.
- **Remove/Replace Legacy Plugins or Libraries:** Especially with WordPress, deactivate and delete any plugins that register scripts sitewide but are not universally needed.

Tip: Re-run PSI after removal to confirm dead code elimination has positively affected metrics[5][15].

---

7. Reduce Unused CSS

Description

Unused CSS sheets or selectors force the browser to download, parse, and remember a large set of style rules, even if their rules are never applied in the current viewport or page structure. This often comes from global styles, frameworks, or modular CSS designed for multi-page apps.

Impact

- **Slows Rendering:** Extra CSS increases blocking time (TBT), delaying paint for even simple pages.
- **Inflates Payload:** More CSS = higher network costs, especially punishing for mobile/cellular connections.
- **Potential for Style Collisions/Bugs:** Legacy code may interact unexpectedly if carried forward unrefined.

Recommended Fixes

- **Automated Tools to Purge CSS:** Leverage PurgeCSS, UnCSS, or similar tools (or plugins for WordPress, e.g., NitroPack's RUCSS) to strip unreferenced rules.
- **Modular CSS-Only Where Used:** Import CSS only for components or features rendered on the current page.
- **Manual Review (Advanced):** For persistent issues, manual code audits may be required as some auto-purging tools err on the side of caution with dynamic selectors.

Warning: Automated purges may miss JS-injected selectors/styles. Manual testing on all device breakpoints is critical after applying removal/remediation tools[5].

---

8. Avoid Enormous Network Payloads

Description

Modern PSI reports warn if the total page transfer size exceeds 1.6 MB - the threshold for optimal performance on 3G/4G networks. Excessive payloads are often caused by unoptimized images, oversized video, bundled assets, and poorly managed third-party scripts[17].

Impact

- **Longer Wait Times:** Increases LCP, TTI, FCP, and network costs for users.
- **Data Usage Concerns:** May cause users to bounce if visiting over constrained connections.
- **Aggregates with Other Issues:** Often symptomatic of deeper architectural bloat (redundant code, excessive plugins, etc.).

Recommended Fixes

- **Optimize Images:** Apply responsive resizing, compression, and format conversion (WebP/AVIF).
- **Minify/Compress CSS and JS:** Use Gzip/Brotli, minify all deployable assets.
- **Reduce Third-Party Embeds:** Remove or lazy-load analytics, chat widgets, ads, and replace embeds with static links/thumbnails where possible.
- **Deploy a CDN:** Geographically distribute assets for faster, localized delivery; reduces TTFB and total transfer size.

Best Practice: Regularly audit with PSI, GTmetrix, or WebPageTest to track payload creep and maintain under-threshold targets[16][1].

---

9. Efficiently Encode Text-Based Resources

Description

Text assets (HTML, CSS, JS, SVG) benefit dramatically from server-side compression using Gzip or Brotli, both of which shrink payload sizes by up to 70%+ depending on content type and structure[19].

Impact

- **Saves Bandwidth:** Smaller file sizes mean lower data transfer costs for both server and client.
- **Faster First Paint:** Decreases TTFB, speeds up parse and render.
- **Search Crawl Efficiency:** Faster access for bots and crawlers can improve indexation/update rates.

Recommended Fixes

- **Enable Gzip/Brotli Compression:** On Nginx or Apache, enable appropriate modules and configuration for all text-based MIME types.
- **Validate Compression in Production:** Check response headers (Content-Encoding: gzip|br) and measure reduced payload across major asset types.
- **Prioritize Brotli for Static Assets:** Newer and more efficient, Brotli is best for static JS/CSS/assets served via CDN.

Note: Avoid compressing already-compressed (binary) files to prevent waste and increased CPU overhead; this applies mainly to images, video, and certain audio formats[19].

---

10. Preload Key Requests

Description

Preloading lets developers tell browsers to fetch critical resources - such as fonts, above-the-fold images, or crucial JS/CSS - earlier than they would otherwise be discovered in the parsing/rendering process. Failure to preload key assets delays LCP, FCP, and text rendering, making users wait longer for meaningful content[21].

Impact

- **Improved LCP/TTI:** Ensures largest visible elements, especially custom fonts and hero images, are available as soon as possible.
- **Reduces Perceived Lag:** Users sense faster load times when main content is ready promptly.
- **Boosts Score:** Directly raises scores for both PSI Opportunities and Core Web Vitals.

Recommended Fixes

- **Identify Critical Assets:** Use PSI and Chrome DevTools to trace critical request chains (fonts/images).
- **Inject** <link rel="preload"> **Tags:** Add explicitly in the <head> for each resource, using the correct as attribute (e.g., as="font", as="style", etc.).
- **Limit Excessive Preloads:** Only critical, above-the-fold assets should be preloaded; unnecessary preloads can delay other resources.

Special Note: Web fonts are notorious LCP/CLS blockers and require particular attention to preloading and font-display: swap|optional settings for best effect[23].

---

11. Use Efficient Cache Policy

Description

Efficient caching allows returning visitors (whether actual users or bots) to avoid repeated downloads of unchanged static assets. PSI flags assets without sufficiently long-lived cache lifespans (short TTLs or missing Expiry/Cache-Control headers)[25].

Impact

- **Repeat Loads are Slower:** Increases bandwidth usage and slows navigation for users navigating multiple pages or returning to the site.
- **Strains Server Resources:** Increases server request load and slows down response under traffic spikes.
- **SEO/Ranking:** Negative cache performance may reduce crawl rate and impact perceived performance.

Recommended Fixes

- **Set Long TTLs for Static Assets:** Ensure images, fonts, JS/CSS assets are cached for weeks or months (Cache-Control: max-age=31536000, immutable).
- **Configure CDN Edge Caching:** Push as many static assets as possible to CDN edges, reducing origin requests.
- **Self-Host or Inline Third-Party Assets:** Where possible, bring external scripts/fonts under your domain for better control of caching policy.

Implementation: Use plugins or server admin controls to insert suitable headers, and verify with browser Dev Tools for correct policy lifetimes[24].

---

12. Lazy Load Offscreen Images

Description

Without lazy loading, all images on a page - even those well below the fold - are requested, parsed, and sometimes displayed as soon as the page starts rendering. This inflates initial payload and delays first paint, especially on image-heavy mobile sites[27].

Impact

- **Bloats Initial Network Load:** Slows down LCP/FCP and overall performance, especially for large galleries or long content pages.
- **Wastes Bandwidth:** Users who never scroll past the fold download assets needlessly.
- **Hurts Perceived Speed:** Delays in above-the-fold paint make site feel sluggish.

Recommended Fixes

- **Use Native (**loading="lazy"**) or JS Polyfills:** Native lazy loading is supported in most modern browsers. JS libraries like lazysizes, lozad.js provide fallbacks.
- **Apply only to Offscreen/Below-the-Fold Images:** Do **not** lazy load images that are critical to above-the-fold content or are LCP candidates.
- **Audit via PSI:** Run repeated tests to ensure lazy loading is working on all device widths and orientations.

Extra: For background images (CSS), custom JavaScript solutions may be required to delay loading until elements enter viewport[27].

---

13. Preconnect to Required Origins

Description

Preconnect resource hints (<link rel="preconnect">) instruct browsers to initiate early DNS/TCP/TLS connections for third-party resources (e.g., CDNs, font APIs), reducing delay when the resource is eventually requested during rendering[29].

Impact

- **Reduces Connection Latency:** Particularly beneficial for fonts, analytics, or script/CDN resources not hosted on your main domain.
- **Accelerates LCP/FCP:** Ensures critical external assets are available more quickly for rendering key content.

Recommended Fixes

- **Identify All Third-Party Critical Resources:** Fonts (Google Fonts), CDN(s), analytics/APIs.
- **Add** <link rel="preconnect"> **in** <head>**:** Include only for resources making early requests; excessive preconnects can reduce effectiveness.

Important: Use the crossorigin attribute for resources that require it (fonts, APIs), and avoid preconnecting to unused origins to prevent extra handshake overhead[29].

---

14. Optimize Server Response Time (TTFB)

Description

Time to First Byte (TTFB) measures the duration from the client's request to the first byte of data received from the server. Excess TTFB is usually due to slow backend processing, lack of caching, high server load, or network-level issues.

Impact

- **Cascading Delays:** All paint/UI/blocking milestones are delayed if the server is sluggish.
- **Key for LCP/INP:** Fast TTFB underpins all other Core Web Vitals.
- **SEO Ranking Signal:** Google tracks and penalizes high TTFB for mobile search results.

Recommended Fixes

- **Enable Full-Page Caching:** Use reverse proxies, plugin-based caches, or CDNs for HTML output.
- **Improve Hosting Setup:** Choose high-performance hosts, upgrade to dedicated resources as needed.

- **Profile and Optimize Backend:** Audit database queries, optimize app logic, and monitor for spikes in processing time.
- **Geo-Proximal CDN:** Use CDN edge servers to serve dynamic content close to users.

Monitoring: Use server monitoring tools/CDN analytics and repeated PSI tests to ensure consistently low TTFB across times of day and traffic loads[1].

---

15. Avoid Large Layout Shifts (CLS)

Description

Unexpected layout shifts penalize mobile user experience, making sites feel jarring or "unstable." CLS measures cumulative instances of visual content unexpectedly moving during load - usually due to images/videos without dimensions, delayed ads, dynamically injected content, or slow web font loads[2].

Impact

- **Direct UX Harm:** Users may misclick, lose scroll/place, or misread shifting content.
- **Negative Ranking Impact:** High CLS scores reduce Google's perception of site quality.
- **Conversion Loss:** Annoying shifts can lead to bounced sessions and lost sales/subscriptions.

Recommended Fixes

- **Set Explicit Width/Height for Media:** Always specify both dimensions on images, videos, iframes.
- **Pre-allocate Space for Ads/Embeds:** Use CSS aspect ratio boxes or static reserve areas for dynamic content.
- **Optimize Font Loading:** Use font-display: swap and preload to prevent FOIT/FOUT (see below).
- **Avoid DOM Injection Above-the-Fold:** Ensure dynamic JS/ads/modules do not insert late into top-of-page layouts.

Best Practice: Review PSI "Diagnostic: Avoid large layout shifts" panel for page-specific offenders and test across device orientations.

---

16. Preload and Optimize Font Loading

Description

Web fonts are potent sources of both early/late blocking and layout instability. Without proper preloading and optimal font-display strategies, users may see flashes of unstyled/invisible text (FOUT/FOIT), hurting both CLS and overall speed perception[23].

Impact

- **CLS Degradation:** Late-loading fonts shift line heights/spacing.
- **Slow LCP/TTI:** Font download delays postpone visual completeness and interactivity.
- **User Distrust:** Unstyled/invisible flashes weaken brand impression.

Recommended Fixes

- **Preload Key Fonts:** Use <link rel="preload" as="font"> with correct type and crossorigin settings for essential web fonts.
- **Set Font-Display Property:** font-display: swap ensures fallback fonts appear instantly, replaced by the custom font with minimal FOIT.
- **Subset Fonts:** Only serve required characters/glyphs to mobile devices; reduces download time and payload.
- **Responsive Loading Strategies:** Consider different preloading/display strategies for mobile vs desktop to balance speed and brand consistency[23].

Bonus: Host fonts locally if possible to avoid slow font APIs and maximize cache control.

Conclusion

Google PageSpeed Insights provides not only a snapshot but a roadmap for mobile web performance. Each error flagged in the PSI report for https://asterik.ae corresponds to a deliberate, addressable optimization - spanning code, content, server infrastructure, and third-party dependencies. The cross-impact between issues is significant: solving image, asset, script, and cache problems nearly always produces compounding improvements in Core Web Vitals and overall user experience.

Key priorities for https://asterik.ae:

- Address render-blocking resources as the top driver of visible and interactive delays.
- Optimize, compress, and "next-gen" all images and text assets.
- Audit and minimize JavaScript/CSS overhead, with focus on the main-thread and critical path.
- Correct caching, preloading, and lazy loading for static/third-party assets.
- Implement explicit strategies for font and layout stability to meet Core Web Vitals thresholds.

Through a combination of structured audit follow-through (guided by Google's Opportunities and Diagnostics), automation (where feasible), and ongoing monitoring, https://asterik.ae can achieve world-class mobile performance, reflected not just in PSI scores, but in tangible business outcomes - higher engagement, lower bounce rates, and improved search visibility.

This analysis integrates guidance from Google Developers documentation, industry case studies, technical blogs, and best-in-class plugin/tool documentation. For live verification and details

particular to theme/plugins in use, ongoing real-time testing is strongly recommended after each optimization step.

References (30)

1. *How to Increase Mobile Page Speed (11 Optimization Tips)*. https://nitropack.io/blog/post/increase-mobile-page-speed
2. *Core Web Vitals Checker - SpeedVitals*. https://speedvitals.com/tools/core-web-vitals-checker
3. *How To Improve Cumulative Layout Shift (CLS) on WordPress*. https://wp-rocket.me/google-core-web-vitals-wordpress/improve-cumulative-layout-shift/
4. *From Slow to Swift: Enhancing Mobile PSI Scores - Codeable*. https://www.codeable.io/blog/how-to-fix-low-pagespeed-insights-score-2023-checklist/
5. *How to eliminate 'Render-blocking JavaScript' error for mobile*. https://stackoverflow.com/questions/41950270/how-to-eliminate-render-blocking-javascript-error-for-mobile
6. *How to Eliminate Render-Blocking Resources (CSS and JavaScript)*. https://nitropack.io/blog/post/eliminate-render-blocking-resources
7. *Top Strategies for Optimizing Images for Mobile - BrowserStack*. https://www.browserstack.com/guide/strategies-for-optimizing-images-for-mobile
8. *How to Optimize Images for Mobile (2024)*. https://imagify.io/blog/how-to-optimize-images-for-mobile/
9. *How to Serve Images Next-Gen Formats on WordPress: Everything You ...*. https://imagify.io/blog/serve-next-gen-formats-wordpress/
10. *Minimize Main Thread Work: 5 Effective Ways To Fix The ... - RabbitLoader*. https://rabbitloader.com/articles/minimize-main-thread-work/
11. *Reduce Unused CSS Recommendation Is Still Present in ... - NitroPack*. https://support.nitropack.io/en/articles/8390473-reduce-unused-css-recommendation-is-still-present-in-pagespeed-insights
12. *How to Avoid Enormous Network Payloads (The Smart Way)*. https://nitropack.io/blog/post/avoid-enormous-network-payloads
13. *How Content Encoding Works and Why It Matters for Website Speed*. https://www.browserstack.com/guide/content-encoding
14. *PageSpeed: Preconnect to Required Origin - Screaming Frog*. https://www.screamingfrog.co.uk/seo-spider/issues/pagespeed/preconnect-to-required-origin/
15. *Preload key requests* . https://gtmetrix.com/preload-key-requests.html
16. *Responsive web font loading, a new stategy*. https://www.corewebvitals.io/pagespeed/responsive-font-loading-strategy
17. *Mobile Browser Caching WordPress Plugin Development - Code Canel*. https://codecanel.com/mobile-browser-caching-wordpress-plugin-development/

18. *How to Serve Assets With an Efficient Cache Policy on WordPress*. https://wp-rocket.me/google-core-web-vitals-wordpress/serve-static-assets-with-an-efficient-cache-policy/
19. *How to Avoid Enormous Network Payloads - WP Rocket*. https://wp-rocket.me/google-core-web-vitals-wordpress/avoid-enormous-network-payloads/
20. *Fix The "Defer Offscreen Images" Message on Google PageSpeed*. https://www.contentpowered.com/blog/defer-offscreen-images-pagespeed/
21. *Web Vitals* . https://web.dev/articles/vitals