Web Performance Optimisation

Group 2: Celine Florian, Stephan Robinig, Piotr Siewiera and Nina Tschikof

2 Dec 2025

Information Architecture and Web Usability WS 2025

Overview

1. Improving Download Performance
2. Improving Perceived Performance
3. Improving Runtime Performance
4. Checking Web Performance
5. Performance Demo

1. Improving Download Performance

Caching

- Stores some of the website's data on client side.[1]
- Uses two types of cache:
  - Memory cache - extremely fast but short-lived, lasts only while a page is open.
  - Disk cache - persists across tabs and sessions, can hold much larger resources.
- Use far-future `Expires` header[2] - use long cache durations for versioned resources, short for unversioned ones.

[1] https://jonoalderson.com/performance/http-caching/

[2] https://youtube.com/watch?v=BTHvs3V8DBA&t=1618s

Caching Techniques

- Use Cache-Control headers like like `Date`, `Cache-Control` to take advantage of browser cache:
  - `date: Sat, 17 Jan 2026 00:40:36 GMT`
  - `Cache-Control: max-age=86400`
- Take control of caching with a Service Worker.[1]

[1] https://medium.com/@ddylanlinn/optimizing-frontend-caching-with-service-worker-and-cache-strategy-4131ae1d9aa8

Use HTTP/3

- Turn on HTTP/3, fall back to HTTP/2.
- LCP faster with HTTP/3 (1.44s vs 1.67s for HTTP/2, data from DebugBear from 30 June 2025).[1]
- Only Samsung Internet, Opera Mini, QQ Browser and Kai OS do not support HTTP/3 (data from September 2024).[2]

HTTP	1.x	2.x	3.x
Usage	9.2%	60.4%	30.4%

HTTP version usage. Data from 15 Nov 2025[3]

[1] https://www.debugbear.com/blog/http3-vs-http2-performance

[2] https://caniuse.com/?search=http+3

[3] https://radar.cloudflare.com/

Ship Fewer Bytes

- Use tree shaking for JS to eliminate unused code.
- Minify text assets (HTML, SVG, CSS, JS).
- Turn on Brotli (br) compression for text assets (HTML, SVG, CSS, JS), fall back to gzip.

File Compression

- Gzip/Brotli: Use Brotli and fall back to GZIP.
- Compress HTML, CSS and JS before sending.
- Smaller files sent = faster transfer.
- Compression only once.
- Decompression multiple times.

File Compression Results

- Brotli 27% smaller.
- Compressing: Gzip 4000% faster.
- Decompressing: Brotli 37% faster.

Data 1GB	Size Reduction	Compression Speed	Decompression Speed
Brotli	32%	0.7Kb/s	380 Mb/s
Gzip	23%	29Mb/s	270 Mb/s

Data compression results (local hardware).[1]

[1] https://www.mattmahoney.net/dc/textdata.html

Minification

- Remove whitespace, comments, shorten variable names.
- Trade-offs: reduced readability & debug capability.

```
function addNumbers(e,r){return e+r}
function multiplyNumbers(e,r){if(0===r||0===e)return 0;
const n=e<0&&r>0||e>0&&r<0,t=Math.abs(e),o=Math.abs(r);let u=0;
for(let e=0;e<o;e++)u=addNumbers(u,t);return n&&(u=-u),u}
```

Minified JS code

```
function addNumbers(a, b) {
    const result = a + b;
    return result;
}

function multiplyNumbers(a, b) {
    // Handle edge cases
    if (b === 0 || a === 0) {
        return 0;
    }

    // Handle negative numbers
    const isNegative = (a < 0 && b > 0) || (a > 0 && b < 0);
    const absA = Math.abs(a);
    const absB = Math.abs(b);

    let result = 0;

    for (let i = 0; i < absB; i++) {
        result = addNumbers(result, absA);
    }

    // Apply sign if needed
    if (isNegative) {
        result = -result;
    }

    return result;
}
```

Original JS code

Other Techniques to Improve Download Performance

- Serve images as WebP or AVIF through `<picture>` element. Widely supported by all modern browsers. [1]
- Serve appropriate resolution images with `srcset` and `sizes`.
- Consider using a CDN (Content Delivery Network).
- Consider bundling: [2]
  - Merge JS and CSS into bundles.
  - Reduces number of requests and load time.
  - Trade-off with cache invalidation.

[1] https://caniuse.com/?search=avif+webp

[2] https://learn.microsoft.com/en-us/aspnet/mvc/overview/performance/bundling-and-minification

2. Improving Perceived Performance

Load What Matters First

- Place CSS in `<head>` (immediate download & early rendering).
- Non-critical JS at the bottom (blocks the parser).[1]

[1] https://strapi.io/blog/frontend-performance-checklist

Manage Non-Critical Resources

- Use `async` and `defer`:[1]
  - `async` attribute ensures that the JS resource is loaded asynchronously in the background and does not block rendering.
  - `defer` attribute tells the browser to run the script after the document has been parsed.
- Use `preload` and `prefetch` judiciously.
- Lazy load off-screen images later.[2]

[1] https://debugbear.com/blog/async-vs-defer

[2] https://cloudfour.com/thinks/stop-lazy-loading-product-and-hero-images/

3. Improving Runtime Performance

Reduce Main-Thread Work

- Use CSS rather than JS, wherever equivalent functionality possible:[1]
  - CSS is more efficient, since implemented natively in browser.
  - JS is not error-tolerant, it can sometimes break.
- Offload tasks from main thread with Web Workers:[2]
  - Web Workers run in parallel in background threads.

[1] https://stackoverflow.com/questions/24012569

[2] https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers

Optimise JS Runtime Performance

- Design and write efficient code and algorithms.
- Optimise memory usage.

4. Checking Web Performance

👁

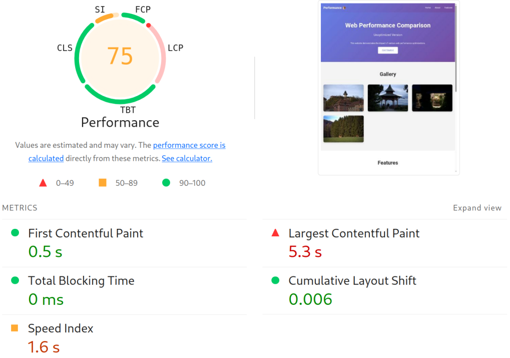
Core Web Vitals Measures

- Largest Contentful Paint (LCP):
  - Measures *loading* performance.
  - Should be $\leq 2.5s$. Achieved by $\approx$ 67.7% of pages[1]
- Interaction to Next Paint (INP):
  - Measures *interactivity*.
  - Shoul be $\leq 200ms$. Achieved by $\approx$ 85.9% of pages[1]
- Cumulative Layout Shift (CLS):
  - Measures *visual* stability.
  - Should be ≤ 0.1. Achieved by $\approx$ 80.3% of pages [1]
- 54.4% of pages satisfy all Core Web Vitals. [1]

[1] https://developer.chrome.com/docs/crux/release-notes#202510

Lighthouse

- Comprehensive performance reporting.
- Measures Core Web Vitals.
- Integrated into Chrome.
- Provides recommendations.



Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 ■ 50–89 ● 90–100

METRICS

Expand view

▲ Largest Contentful Paint
5.3 s

● First Contentful Paint
0.5 s

● Total Blocking Time
0 ms

● Cumulative Layout Shift
0.006

■ Speed Index
1.6 s

Lighthouse performance report.

5. Performance Demo

💡

Side-by-Side Comparison

- Compare web page with optimisations enabled and disabled.
- Python script for page generation:
  - Allow for disabling and enabling different optimisations.
  - Compare different file sizes.
- Self host web page for full control.

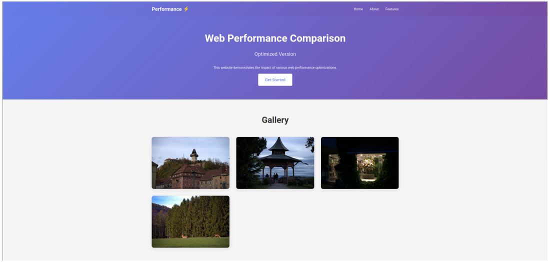
Setup

- Two servers (npm http-server) on same hardware.
- Raspberry Pi Zero 2W 64-bit with Pi OS (Trixie 2025-10-02).
- Cloudflare (synthetic) RUM and/or Lighthouse.
- Repo: `https://github.com/StofflR/WebPerformanceComparison`

Optimisations

- Minification.
- Inline CSS / JS.
- Lazy Loading.
- Image fetch priority.
- Preconnect / Prefetch.
- Remove unused CSS / JS.

Live Demo



Web Performance Comparison

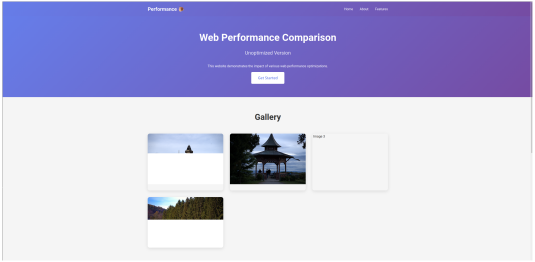
Optimized Version

This website demonstrates the impact of various web performance optimizations.

Gallery

Optimized site: https://webcomp-opt.stofflnet.work

Result: https://webcomp-opt.stofflnet.work/page2.html



Web Performance Comparison

Unoptimized Version

This website demonstrates the impact of various web performance optimizations.

Gallery

Unoptimized site: https://webcomp-unopt.stofflnet.work

Result: https://webcomp-unopt.stofflnet.work/page2.html

Questions?