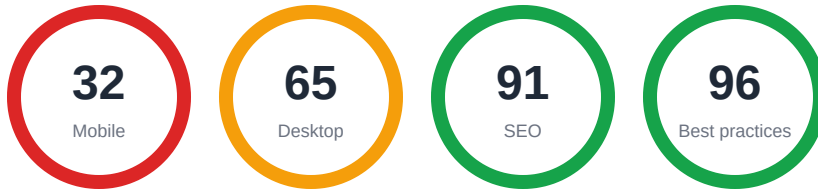




www.oakandpine.example

Prepared for Oak & Pine Furniture · 2026-10-09

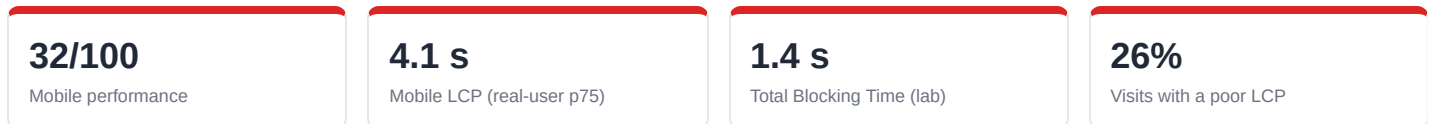


1. Executive summary & business impact
2. Core Web Vitals: real users vs lab
3. Asset breakdown

4. Developer fixes
5. Priority matrix & action plan

1 Executive summary & business impact

The homepage is slow on phones and borderline on desktop: mobile performance scores 32/100, and 26% of real mobile visits wait more than 4 s for the main content (real-user LCP p75 4,100 ms). In the lab, mobile LCP is 15.9 s and nothing appears on screen until 7.5 s. Three root causes account for almost all of it: two synchronous scripts and one stylesheet block rendering (Lighthouse estimates up to 6,830 ms), the 1,876 KiB hero.png is served uncompressed at more than twice its display size, and no static file has a cache lifetime. All three are fixable with configuration and markup changes, without a redesign.



What this means for your visitors

A visitor on a typical phone currently looks at a blank screen for several seconds before any text or image appears, and roughly one in four real mobile visits waits more than 4 s for the main product image. Google/SOASTA's research found that the probability of a mobile visitor bouncing rises by 32% as load time goes from 1 s to 3 s and by 90% from 1 s to 5 s, so this page sits in the range where most of that drop-off happens. Deloitte and Google's 2020 study associated each 0.1 s of mobile speed improvement with 8.4% more retail conversions. Because returning visitors re-download every file, the slowness also repeats on every visit rather than only the first.

- Mobile LCP (real-user p75) is **4.1 s**, which is **1.6 s** above Google's "good" threshold of 2.5 s.
- **26%** of real mobile visits to this page waited more than 4 s for the main content to appear.
- On a throttled phone, the first text or image appears after **7.5 s** (lab).

Published research on speed and revenue

Deloitte & Google, "Milliseconds Make Millions" (2020): A 0.1 s improvement in mobile site speed was associated with 8.4% more conversions and a 9.2% higher average order value for retail sites.

Google / SOASTA mobile page speed research (2017): As mobile page load time goes from 1 s to 3 s, the probability of a visitor bouncing increases by 32%; from 1 s to 5 s it increases by 90%.

Revenue worksheet

	EXAMPLE STORE	YOUR FIGURES
Monthly visitors	10,000	
Conversion rate	2.0%	
Average order value	\$80	
Monthly online revenue (visitors × rate × AOV)	\$16,000	
Benchmark value of each 0.1 s faster (revenue × 8.4%)	≈ \$1,344 / month	

Illustrative only. The 8.4% figure is the average conversion lift Deloitte measured for retail sites per 0.1 s of mobile speed improvement; it is not a forecast for this site, and gains are not linear.

2 Core Web Vitals: real users vs lab

METRIC	REAL USERS, MOBILE (P75)	LAB, MOBILE	LAB, DESKTOP	GOOD
Largest Contentful Paint When the main image or headline appears	4.1 s  41% good · 33% needs work · 26% poor	15.9 s	2.7 s	≤ 2.5 s
Interaction to Next Paint Responsiveness to taps and clicks	310 ms  62% good · 27% needs work · 11% poor	field only	field only	≤ 200 ms
Cumulative Layout Shift How much the layout jumps	0.02  91% good · 6% needs work · 3% poor	0	0	≤ 0.1
First Contentful Paint First text or image on screen	2.9 s  48% good · 34% needs work · 18% poor	7.5 s	1.5 s	≤ 1.8 s
Time to First Byte Server response for the page itself	1.1 s  55% good · 33% needs work · 12% poor	1 ms	1 ms	≤ 800 ms
Total Blocking Time Lab proxy for INP: main thread busy with JavaScript	lab only	1.4 s	323 ms	≤ 200 ms
Speed Index How quickly the page visibly fills in	lab only	7.5 s	1.5 s	≤ 3.4 s

Why lab and real-user numbers differ. Real-user figures come from the Chrome UX Report: the 75th percentile of actual Chrome visits over the last 28 days. Lab figures come from a single Lighthouse 13.5.0 run that simulates a mid-range phone on a slow mobile connection: 150 ms round-trip time, 1.6 Mbps throughput and a 4× CPU slowdown. The lab run is deliberately pessimistic and shows *why* the page is slow; the real-user data shows *how many* visitors feel it. Google uses the real-user data for search ranking.

Where the LCP time goes (mobile)

LCP element: ``



■ Time to first byte 1% ■ Resource load delay 1% ■ Resource load duration 4% ■ Element render delay 94%

Share of the LCP time spent in each phase. A large render delay usually points to render-blocking CSS/JS or client-side rendering; a large load duration points to a heavy LCP image.

Lighthouse scores

	PERFORMANCE	SEO	BEST PRACTICES
Mobile	32	91	96
Desktop	65	91	92

3 Asset breakdown

All figures from the mobile Lighthouse run. Own-domain files are shown by path; third-party files with their host.

Page weight by type (mobile, 2,905 KiB total)

TYPE	REQUESTS	TRANSFER SIZE	
Image	1	1,876 KiB	
Script	2	783 KiB	
Stylesheet	1	245 KiB	
Document	1	1 KiB	

Render-blocking requests · Est savings of 6,830 ms Lighthouse estimate: FCP −6,850 ms

Files the browser must download and process before it can paint anything.

RESOURCE	SIZE	EST. DELAY
/vendor.js	587 KiB	5,401 ms
/theme.css	245 KiB	3,601 ms
/app.js	196 KiB	3,151 ms

Unused JavaScript · Est savings of 578 KiB Lighthouse estimate: FCP −4,200 ms, LCP −3,150 ms

Code downloaded on this page that never runs during load.

RESOURCE	SIZE	UNUSED	UNUSED %
/vendor.js	587 KiB	578 KiB	98%

Unused CSS · Est savings of 245 KiB Lighthouse estimate: FCP −900 ms, LCP −1,200 ms

Style rules downloaded but not used on this page.

RESOURCE	SIZE	UNUSED	UNUSED %
/theme.css	245 KiB	245 KiB	100%

JavaScript execution time · 1.5 s Lighthouse estimate: TBT −1,400 ms

Main-thread time per script on the throttled phone. This drives TBT and INP.

RESOURCE	CPU TIME	SCRIPT EVAL
/vendor.js	1,466 ms	1,400 ms
Unattributable	159 ms	6 ms
/	76 ms	7 ms

Image delivery · Est savings of 1,822 KiB

Lighthouse estimate: LCP –7,950 ms

Oversized, uncompressed or legacy-format images.

RESOURCE	SIZE	EST. SAVINGS
/hero.png	1,876 KiB	1,822 KiB
Using a modern image format (WebP, AVIF) or increasing the image compression could improve this image's download size. This image file is larger than it needs to be (1000x640) for its displayed dimensions (721x461). Use responsive images to reduce the image download size.		

Cache lifetimes · Est savings of 2,904 KiB

Lighthouse estimate: FCP –5,100 ms, LCP –13,500 ms

Static files that repeat visitors must download again.

RESOURCE	CACHE TTL	SIZE
/hero.png	none	1,876 KiB
/vendor.js	none	587 KiB
/theme.css	none	245 KiB
/app.js	none	196 KiB

Layout shift culprits

Elements that move content after it appears.

RESOURCE
/hero.png

4 Developer fixes

Ordered by expected impact. Code blocks are starting points written for the files found on this site: test on a staging copy first and adapt paths marked in comments.

1 vendor.js, app.js and theme.css block the first paint

CRITICAL IMPACT **LOW EFFORT** FCP

Evidence Render-blocking requests: Est savings of 6,830 ms (FCP). vendor.js (587 KiB, 5,401 ms), theme.css (245 KiB, 3,601 ms), app.js (196 KiB, 3,151 ms). Lab mobile FCP is 7.5 s.

Why it matters The browser cannot draw anything until all three files have downloaded and run, so every visitor stares at a blank page for the full download time.

Expected gain **could cut up to 6,850 ms from mobile FCP**

1. Add the defer attribute to both script tags in the <head> so they download in parallel and run after the HTML is parsed.
2. Check that no inline script calls code from vendor.js before DOMContentLoaded; wrap any such call in a DOMContentLoaded listener.
3. Extract the CSS needed for the header and hero (above the fold) and inline it in a <style> block.
4. Load the full theme.css asynchronously with the preload/onload pattern and a <noscript> fallback.
5. Re-test with PageSpeed and confirm the render-blocking audit passes.

HTML <head>: defer scripts, inline critical CSS, load theme.css async

HTML

```
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width,initial-scale=1">
<title>Oak & Pine Furniture</title>

<!-- Critical CSS for header + hero only (generate with e.g. `npx critical`) -->
<style>
  body{font-family:Inter,sans-serif;margin:0}
  .hero img{width:100%;height:auto}
</style>

<!-- Full stylesheet no longer blocks rendering -->
<link rel="preload" href="/theme.css" as="style" onload="this.onload=null;this.rel='stylesheet'">
<noscript><link rel="stylesheet" href="/theme.css"></noscript>

<!-- Scripts download in parallel, run after parsing, in order -->
<script src="/vendor.js" defer></script>
<script src="/app.js" defer></script>
</head>
```

2 hero.png is 1,876 KiB and served at twice its display size

CRITICAL IMPACT **LOW EFFORT** LCP

Evidence Improve image delivery: Est savings of 1,822 KiB (LCP 7,950 ms). hero.png is the LCP element; Lighthouse reports it is 1000x640 but displayed at 721x461 and not in a modern format.

Why it matters The hero image is the largest element on the page, so its download time is the LCP. On a slow mobile connection 1.8 MB takes many seconds.

Expected gain **could cut up to 7,950 ms from mobile LCP and up to 1,822 KiB of transfer**

1. Export hero.png as AVIF and WebP at 480, 800 and 1200 px wide (e.g. with Squoosh or sharp).
2. Serve them with a <picture> element and srcset/sizes so phones get the small file.
3. Add fetchpriority="high" to the hero image so the browser requests it first.
4. Add explicit width and height attributes (also fixes the unsized-image warning).
5. Keep the original PNG only as the fallback .

Responsive, prioritised hero image

HTML

```
<div class="hero">
  <picture>
    <source type="image/avif"
      srcset="/hero-480.avif 480w, /hero-800.avif 800w, /hero-1200.avif 1200w"
      sizes="100vw">
    <source type="image/webp"
      srcset="/hero-480.webp 480w, /hero-800.webp 800w, /hero-1200.webp 1200w"
      sizes="100vw">
    
  </picture>
</div>
```

3 No static file has a cache lifetime

HIGH IMPACT

LOW EFFORT

PAGE WEIGHT

Evidence Use efficient cache lifetimes: Est savings of 2,904 KiB. hero.png (1,876 KiB), vendor.js (587 KiB), theme.css (245 KiB) and app.js (196 KiB) all have a cache TTL of 0.

Why it matters Every returning visitor downloads the full 2.9 MB again, so repeat visits and multi-page sessions are as slow as the first.

Expected gain **up to 2,904 KiB less transfer on repeat visits**

1. Add long-lived Cache-Control headers for images, fonts, CSS and JS.
2. Make sure file names change when content changes (e.g. vendor.3f9a.js) or append a version query string, so long caching is safe.
3. Keep HTML on a short cache or no-cache so content updates appear immediately.
4. Reload nginx and verify the header with curl -I.

nginx: long-lived cache for static assets

NGINX

```
# inside the server { } block for www.oakandpine.example
location ~* \.(?:css|js|png|jpe?g|webp|avif|svg|woff2)$ {
  expires 1y;
  add_header Cache-Control "public, max-age=31536000, immutable";
  access_log off;
}

location = / {
  add_header Cache-Control "no-cache";
}

# Apache (.htaccess) equivalent:
# <FilesMatch "\.(css|js|png|jpe?g|webp|avif|svg|woff2)$">
#   Header set Cache-Control "public, max-age=31536000, immutable"
# </FilesMatch>
```

4 98% of vendor.js is never used on this page

HIGH IMPACT

MEDIUM EFFORT

LCP

Evidence Reduce unused JavaScript: Est savings of 578 KiB (FCP 4,200 ms, LCP 3,150 ms). vendor.js is 587 KiB, of which 578 KiB is unused.

Why it matters Even after deferring, the phone still downloads and parses almost 600 KiB of code it does not need, competing with the hero image for bandwidth.

Expected gain **could cut up to 4,200 ms from FCP and 3,150 ms from LCP**

1. Open Chrome DevTools > Coverage on the homepage to see which functions in vendor.js run.
2. If vendor.js is a bundle, split it with dynamic import() so homepage code is separate from code for other pages.
3. Remove libraries that are not used anywhere (check the bundle with source-map-explorer).
4. Load the rest only on the pages that need it.

Load heavy code only when it is needed

JAVASCRIPT

```
// app.js: keep the homepage bundle small and load the rest on demand
document.addEventListener('DOMContentLoaded', () => {
  const gallery = document.querySelector('[data-gallery]');
  if (gallery) {
    // adapt path: the module that used to live in vendor.js
    import('/js/gallery.js').then((m) => m.init(gallery));
  }
});

// Bundler (Vite/webpack) config: split vendor code per route
// build.rollupOptions.output.manualChunks = (id) =>
//   id.includes('node_modules') ? 'vendor-' + id.split('node_modules/')[1].split('/')[0] : undefined;
```

5 vendor.js keeps the main thread busy for 1.5 s

HIGH IMPACT

MEDIUM EFFORT

TBT

Evidence Reduce JavaScript execution time: 1.5 s (TBT 1,400 ms). vendor.js uses 1,466 ms of CPU, 1,400 ms of it script evaluation. Lab mobile TBT is 1,420 ms; real-user INP p75 is 310 ms.

Why it matters While the script runs, taps and scrolls are ignored, which is why real visitors see sluggish responses (INP needs work).

Expected gain **could cut up to 1,400 ms from mobile TBT**

1. Record a Performance profile in DevTools (CPU 4x slowdown) and find the long task inside vendor.js.
2. Move work that is not needed for the first view behind requestIdleCallback or user interaction.
3. Break long loops into chunks that yield to the browser with scheduler.yield() or setTimeout.
4. Re-test TBT in PageSpeed and watch INP in the CrUX data over the following 28 days.

Yield to the browser during long work

JAVASCRIPT

```
// Run non-urgent work in small slices so taps are handled in between
const yieldToMain = () =>
  'scheduler' in window && 'yield' in scheduler
    ? scheduler.yield()
    : new Promise((r) => setTimeout(r, 0));

async function runInSlices(tasks) {
  let deadline = performance.now() + 40;
  for (const task of tasks) {
    task();
    if (performance.now() > deadline) {
      await yieldToMain();
      deadline = performance.now() + 40;
    }
  }
}

// Defer the rest until the browser is idle
requestIdleCallback(() => runInSlices(nonCriticalTasks));
```

6 theme.css ships 245 KiB of rules this page does not use

MEDIUM IMPACT

MEDIUM EFFORT

FCP

Evidence Reduce unused CSS: Est savings of 245 KiB (FCP 900 ms, LCP 1,200 ms). Almost all of theme.css (245 KiB) is unused on the homepage.

Why it matters Once theme.css loads asynchronously (fix 1) it no longer blocks paint, but it still costs bandwidth and style calculation time.

Expected gain **could cut up to 1,200 ms from LCP and 245 KiB of transfer**

1. Run PurgeCSS against the site's templates to remove unused selectors.
2. Add a safelist for classes added by JavaScript at runtime.

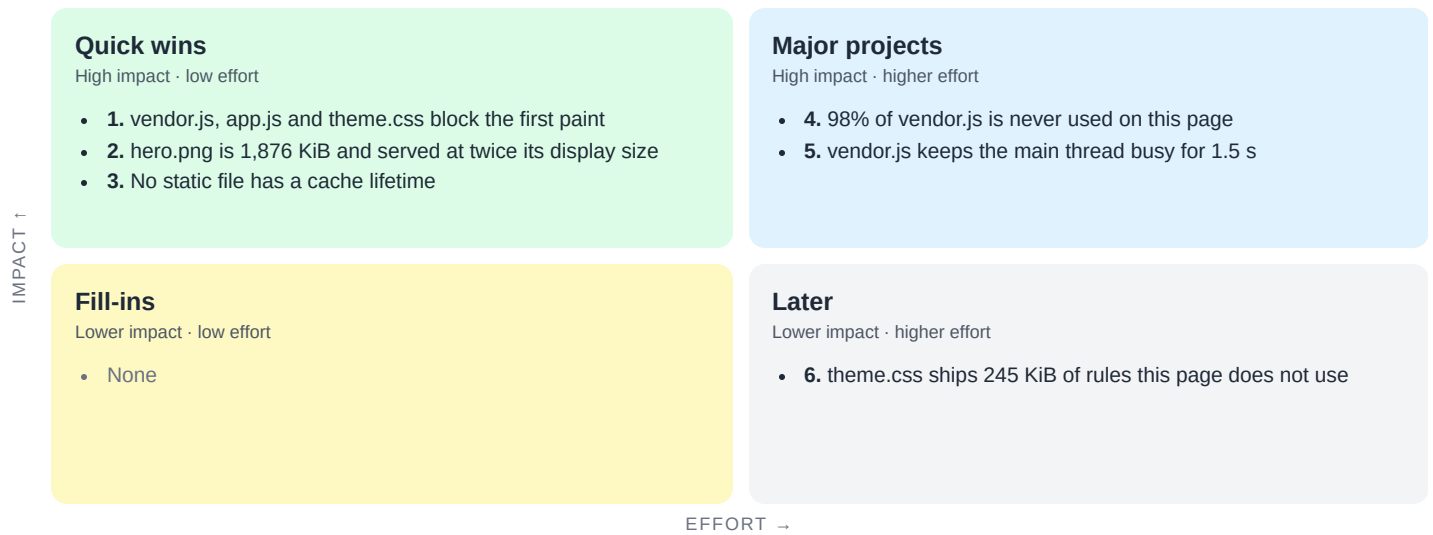
3. Compare the page visually before and after on mobile and desktop.
4. Serve the purged file with a new versioned name.

purgecss.config.js

JAVASCRIPT

```
// npx purgecss --config purgecss.config.js
module.exports = {
  content: ['./**/*.html', './js/**/*.js'], // adapt to your templates
  css: ['./theme.css'],
  output: './dist/',
  safelist: {
    standard: [/^is-/, /^has-/], // classes toggled by JS
    deep: [/^modal/],
  },
};
```

5 Priority matrix & action plan



Action plan

#	ACTION	METRIC	EFFORT	EXPECTED GAIN	WHEN
1	vendor.js, app.js and theme.css block the first paint (fix 1)	FCP	LOW	could cut up to 6,850 ms from mobile FCP	Do this week
2	hero.png is 1,876 KiB and served at twice its display size (fix 2)	LCP	LOW	could cut up to 7,950 ms from mobile LCP and up to 1,822 KiB of transfer	Do this week
3	No static file has a cache lifetime (fix 3)	page weight	LOW	up to 2,904 KiB less transfer on repeat visits	Do this week
4	98% of vendor.js is never used on this page (fix 4)	LCP	MEDIUM	could cut up to 4,200 ms from FCP and 3,150 ms from LCP	Plan next
5	vendor.js keeps the main thread busy for 1.5 s (fix 5)	TBT	MEDIUM	could cut up to 1,400 ms from mobile TBT	Plan next
6	theme.css ships 245 KiB of rules this page does not use (fix 6)	FCP	MEDIUM	could cut up to 1,200 ms from LCP and 245 KiB of transfer	Only if time allows

Rolling out and verifying

1. Ship fixes 1-3 together (all low effort), then re-run PageSpeed on mobile and compare FCP and LCP with this report.
2. Schedule fixes 4 and 5 as one JavaScript refactor and test interactions on a real mid-range Android phone.
3. Watch the real-user LCP and INP in PageSpeed Insights or Search Console; CrUX data updates over a rolling 28-day window.
4. Add a PageSpeed or Lighthouse CI check to your deploy process so regressions are caught before release.

Prefer to have this done for you?

We can implement the 5 high-impact fixes in the priority matrix on www.oakandpine.example for you:

- Fixed price from \$650, agreed before any work starts
- Changes made and tested on a staging copy first
- Before-and-after PageSpeed results for the same pages
- The \$149 you paid for this report is deducted if you book within 30 days

Reply "Speed fix"

Or reply to the email this report came with, with "Speed fix" in the subject (radoslav@getsitevitals.eu). We confirm the scope and the price with you before anything is touched.

Methodology & sources

- Tests run on 2026-10-09 with Google PageSpeed Insights (Lighthouse 13.5.0), mobile and desktop, against <https://www.oakandpine.example/>.
- Real-user data: Chrome UX Report (CrUX), 75th percentile over the trailing 28 days.
- "Estimated savings" are Lighthouse's own projections for a single simulated load. Actual gains depend on implementation and on your visitors' devices and networks.
- Deloitte & Google, "Milliseconds Make Millions" (2020).
- Google / SOASTA mobile page speed research (2017).
- Thresholds: Google Core Web Vitals (web.dev/articles/vitals).