PERFREVIEWS

Web performance audit report · June 2026

Booking engine NOW

Fourteen performance issues documented across the booking flow, with measured evidence, root cause and fixes prioritised by impact and effort.

14	1	23	6.2s
issues documented	root cause that multiplies the rest	charts built from measured data	of request chain on /reserva/crear

CLIENT

Vantia Hotels & Resorts

SCOPE

`apps/now-web` · `/reserva/crear`

ANALYST

Joan León · PerfReviews

PRIORITY MARKET

Latin America · locale `/es-mx/`

Sample report. This is a real audit, anonymised. The figures, findings and recommendations are the originals; the client name, the sector and the third-party vendors are fictional. The charts are vector reconstructions of the capture data, not DevTools screenshots: the specific evidence file behind each one is named in the tables of every chapter.

CONTENTS

What this report covers

FRONT MATTER

ISSUES — ONE CHAPTER EACH

CLOSING

SECTION A

How to read this report

Every issue is a self-contained chapter. The front matter holds everything that would otherwise be repeated fourteen times: the measurement environment, the Core Web Vitals thresholds and the evidence conventions.

Anatomy of a chapter

Each chapter opens with a number, a title, a one-line summary of the finding, and a metadata strip: priority, scope, affected metrics, effort and owning team. Directly below sits the single most important measured figure for that issue. The body always follows the same order:

Section	What it holds
Problem	What happens in the application and why, with the responsible code path where it has been identified.
Core Web Vitals impact	Which metrics are affected and through what mechanism. Definitions are in Section C.
Measured impact	Figures taken from a real capture. Anything not measured is labelled as an estimate.
Recommended fix	Concrete changes, ordered by return on effort. The code is illustrative, not a ready-to-apply patch.
Evidence	Which capture files exist, which are missing, and the charts derived from them.
Expected improvement	Projected saving per optimisation, plus the remote-market multiplier. Estimates unless stated otherwise.
References	External sources supporting the diagnosis and the fix.

Conventions

Priority

Priority combines impact and effort, not severity alone. A very high-impact issue that demands cross-team infrastructure work only jumps ahead of a quick win when its multiplier effect justifies it — which is exactly the case for one of the fourteen.

The colours follow the same ramp as the Core Web Vitals thresholds: red, amber and green. The label always accompanies the colour, so the document reads the same in black and white or with a colour vision deficiency.

Level	Meaning	Issues
P0	Attack first — very high impact, amplifies everything else	PERF-005
P1	High-impact quick wins	PERF-001, PERF-003, PERF-004, PERF-006, PERF-007, PERF-011, PERF-013 Fixed
P2	Strategic improvements	PERF-002, PERF-008, PERF-009, PERF-014
P3	Polish and infrastructure	PERF-010, PERF-012

Metric labels

Metrics are grouped by what they describe:

`LCP` `FCP` `TTFB` loading · `TBT` `INP` `TTI` interactivity and responsiveness · `CLS` visual stability

Evidence status

Collected the capture exists in the evidence directory. **Pending** still to be captured; the claim around it rests on the evidence that is present, and says so.

The charts

The charts in this report are **vector reconstructions of the capture data**, not DevTools screenshots. Each one names the evidence file it comes from in its caption. No figure has been rounded or smoothed to make a chart look better.

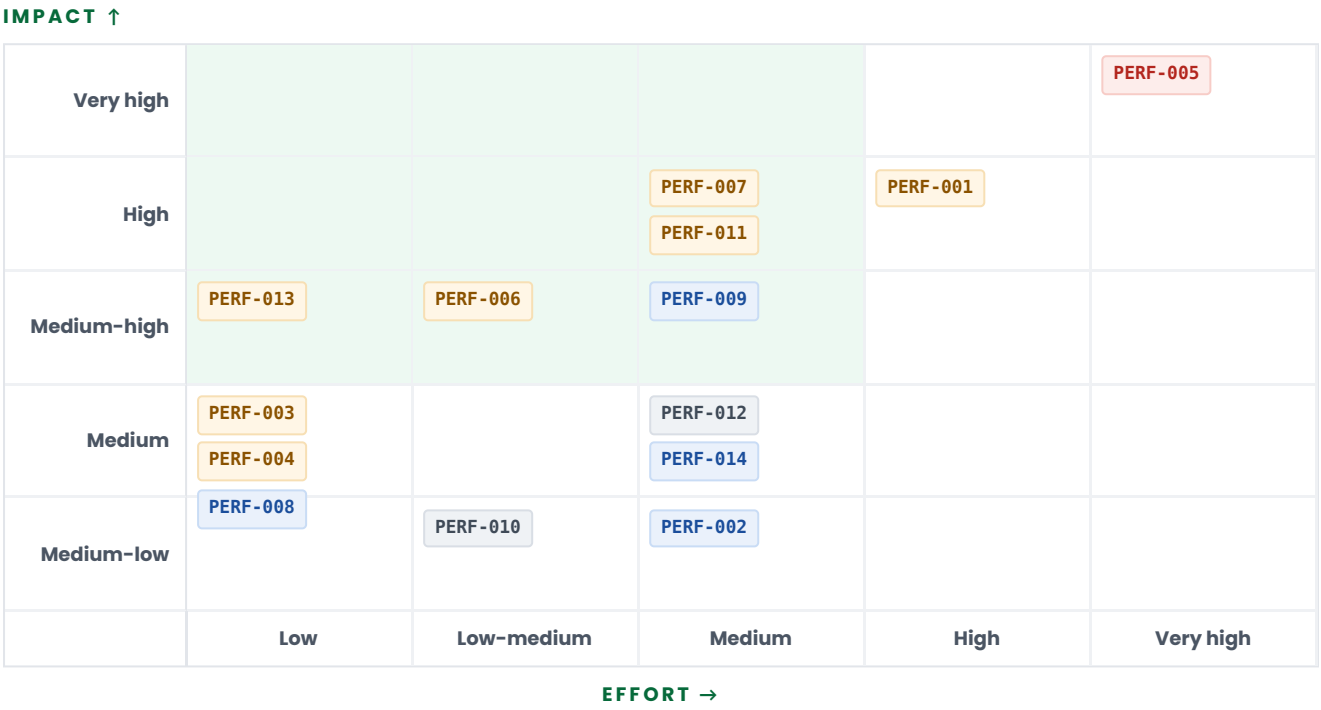
The figures

Measured values are given as they were recorded. Projections use the word *estimated* or a range, and always name the baseline they extrapolate from. Wherever a value is a lower bound relative to the real user — which is nearly everywhere in this document — the chapter says so.

SECTION B

Priority matrix and triage

Impact against effort for the fourteen issues. The shaded zone is the high-impact, low-effort quadrant: the work worth scheduling first.



P0 Attack first P1 Quick win P2 Strategic P3 Polish Shaded: high impact for little effort

P0 — Attack first

ID	Title	Scope	Metrics	Effort	Owner
PERF-005	US-hosted backend and CDN caching disabled by origin headers	All routes	LCP TTI INP	Very high	Infra + Backend

THE MULTIPLIER ON EVERYTHING ELSE

Every other network issue in this report is amplified by the round trip to the US, and again by the distance to the Latin American market. Fixing PERF-005 does not remove the other thirteen, but it changes the size of all their numbers.

P1 — High-impact quick wins

ID	Title	Scope	Metrics	Effort	Owner
PERF-013 Fixed	Load the maps SDK on demand	`/reserva/crear`	TBT TTI	Low	Frontend

ID	Title	Scope	Metrics	Effort	Owner
PERF-006	Defer non-critical JavaScript	All	FCP LCP TBT	Low-medium	Frontend + Platform
PERF-003	Preconnect to external domains	All	LCP FCP	Low	Frontend + Platform
PERF-004	Preload the fonts	All	LCP CLS	Low	Frontend + Platform
PERF-011	Deduplicate the API calls	`/reserva/crear`	TTI INP	Medium	Frontend
PERF-007	Consolidate and defer third-party scripts	All	FCP LCP TBT INP	Medium	Frontend + Analytics
PERF-001	Improve form request chaining	`/reserva/crear`	LCP TTI	High	Frontend + Backend

P2 — Strategic improvements

ID	Title	Scope	Metrics	Effort	Owner
PERF-008	Preload the critical application chunks	`/reserva/crear`	LCP TTI	Low	Frontend + Platform
PERF-009	Speculation Rules for the confirmation navigation	`/crear` → `/confirmada`	LCP TTI (next page)	Medium	Frontend
PERF-014	Optimise the SVG icon sprite	All	LCP TBT	Medium	Frontend + Design
PERF-002	Ship only the CSS you need	All	FCP LCP	Medium	Frontend

P3 — Polish and infrastructure

ID	Title	Scope	Metrics	Effort	Owner
PERF-010	`content-visibility` on off-screen sections	`/reserva/crear`	TBT INP	Low-medium	Frontend
PERF-012	HTTP/2 103 Early Hints	All	FCP LCP	Medium	Infra + Platform

SECTION C

Core Web Vitals reference

The thresholds every chapter refers to. Field metrics are what the user experiences; lab metrics are approximations used during diagnosis.

Metric	Full name	Good threshold	What it measures
LCP	Largest Contentful Paint	≤ 2.5 s	Loading — when the main content appears
FCP	First Contentful Paint	≤ 1.8 s	Loading — when the first pixel appears
TTFB	Time to First Byte	≤ 800 ms	Server and network response time
TBT	Total Blocking Time	≤ 200 ms	Interactivity proxy — lab only
INP	Interaction to Next Paint	≤ 200 ms	Responsiveness across all interactions
TTI	Time to Interactive	≤ 3.8 s	Full interactivity — deprecated, still measured here
CLS	Cumulative Layout Shift	≤ 0.1	Visual stability

SECTION D

Measurement environment and evidence conventions

Applies to every capture in this report unless a chapter says otherwise. Reading it once here saves reading it fourteen times.

Baseline conditions

Variable	Value
Environment	Production, authenticated session
Analyst location	Spain (EU) — roughly 120 ms RTT to the US-hosted origin
Device	MacBook Pro M3
Connection	Symmetric fibre ~600 Mbps, ~10 ms local latency
Artificial throttling	None, except in PERF-010 where 4× CPU throttling is applied and noted
Browser	Chrome stable. Chrome Canary has a DevTools bug that hides the real calls and shows only the `OPTIONS` preflights: unusable for these captures
Target locale	`/es-mx/reserva/crear` — the Latin American market, the worst affected

THESE FIGURES ARE A LOWER BOUND

A MacBook on European fibre is close to the best case this application will ever see. The audience that matters — Latin America, on modest devices and mobile networks — operates at real RTTs of 300 to 500 ms. Every remaining serial hop is multiplied by an estimated **2 to 4×** against this baseline. Read every measured value as the floor, not as the typical user.

Known measurement gaps

- **WebPageTest is blocked by the CDN**, so there is no third-party synthetic lab data for these routes.
- **The development environment does not use the production bundle**, so the bundle-size and blocking-script findings are only reproducible against production.
- **No capture has yet been taken from Latin America.** Every figure for that market is an extrapolation from the European baseline, and is labelled as such.

HOW TO CONTRIBUTE A CAPTURE FROM THE REMOTE MARKET

A capture taken from Mexico, Colombia or Chile is the single most valuable contribution this analysis can receive. Use the same `/es-mx/reserva/crear` URL, follow the `CAPTURE-GUIDE.md` in the relevant evidence directory, and drop the files alongside the existing ones with an `-mx` or `-latam` suffix — for example `PERF-001-no-cache-mx.har`.

Evidence directory structure

Each issue keeps its evidence in its own subdirectory. Not every file exists for every issue: the evidence table in each chapter states the set actually collected.

```
docs/performance/issues/evidence/
└── PERF-XXX/
    ├── CAPTURE-GUIDE.md                 step-by-step capture instructions
    ├── PERF-XXX-no-cache.har            cold network capture
    ├── PERF-XXX-with-cache.har          warm network capture
    ├── PERF-XXX-no-cache-trace.json.gz  cold performance trace
    ├── PERF-XXX-with-cache-trace.json.gz warm performance trace
    ├── PERF-XXX-*.png                   waterfall and flame chart screenshots
    └── PERF-XXX-recording.mp4           screen recording (PERF-001, -002, -010 only)
```

AN INCONSISTENCY WORTH RESOLVING

The source material carries two different figures for the EU→US round trip: roughly 120 ms in the per-issue evidence notes, and roughly 250 ms in the PERF-005 summary. Both have been preserved exactly as recorded, without reconciling them. Before using these numbers to build a business case, confirm one and correct the other.

PART TWO

The issues

Fourteen chapters, one per issue, in identifier order. Chapter *n* is `PERF-0nn`.

01 PERF-001 P1 · Quick win

Improve request chaining in the booking form

Thirty-two schema requests in a single load, seventeen of them redundant, and every one of them waiting behind a bootstrap call that takes 3.9 seconds on its own.

SCOPE	METRICS	EFFORT	OWNER	STATUS
`/reserva/crear`	LCP TTI	High	Frontend + Backend	Open

6,173 ms between the first schema request and the last response, on a warm cache, from Spain to `/es-mx/reserva/crear`. From navigation start to the last schema: **~8,537 ms**.

Problem

The booking form loads its structure through a sequential chain of requests. Each one has to complete before the next begins:

1. HTML download → Angular bootstrap → authentication check
2. Authentication resolved → `GET /reservas/schemas/default`
3. Bootstrap schema resolved → `POST /reservas/schemas/guest-details`
4. Guest details resolved → `POST /reservas/schemas/rate-details`
5. Rates resolved → `POST /reservas/schemas/v2/extra-services`
6. …and so on for every section of the form

The chain is not purely serial: once `/schemas/default` responds, the remaining schemas go out in parallel waves. But every wave sits behind that first response, and that first response takes 3,871 ms.

Core Web Vitals impact

Metric	Effect
LCP	Delayed — the form skeleton, which is the Largest Contentful Paint candidate, cannot render until schema data arrives.
TTI	Badly degraded — the serial requests keep the main thread busy and the form unresponsive.

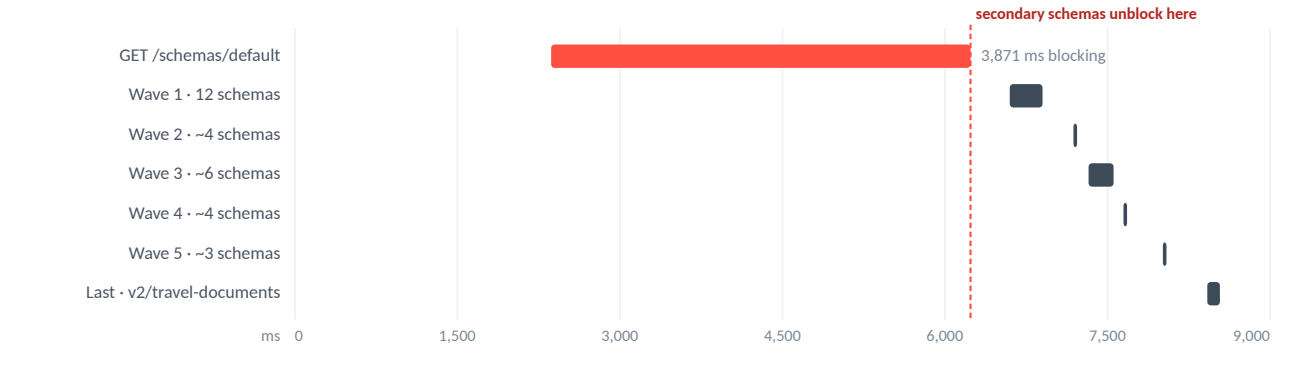
Every additional round trip translates directly into LCP and TTI. **PERF-005** explains why each of those round trips costs what it costs.

Measured impact

Captured with `devtools-snippet-perf001.js` against production, warm cache, Spain (EU) → `/es-mx/reserva/crear`.

32	15	17	3,871 ms
schema requests	unique endpoints	redundant refetches	for /schemas/default alone

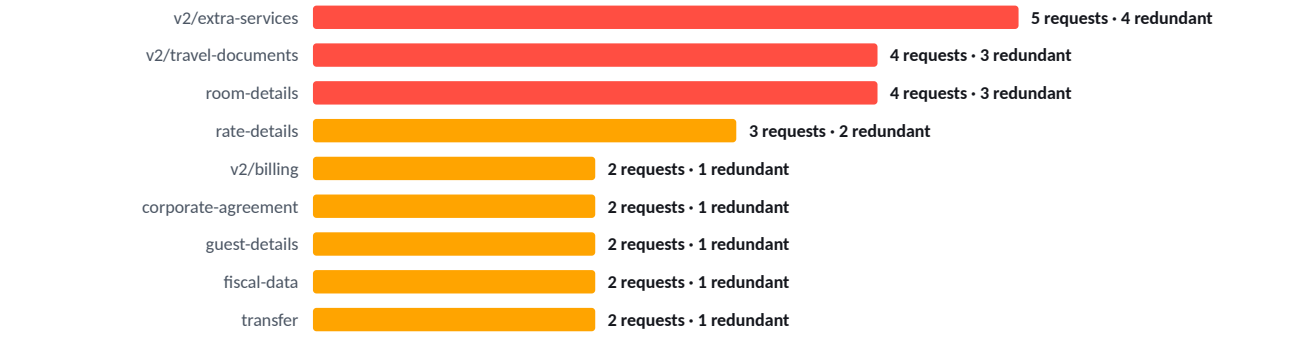
Metric	Value
Total window, first → last schema	6,173 ms
Sum of individual durations	10,911 ms
Overlap recovered by parallelism	4,738 ms
Navigation start → last schema	~8,537 ms



Shape of the chain. The red bar is the gate: until `/schemas/default` responds, none of the five following waves can start. It occupies 43% of the whole window. *Reconstructed from PERF-001-Highlight-schema-requests.json.*

Duplicate requests

The snippet surfaced a finding the waterfall alone does not show: 15 unique endpoints generate 32 HTTP requests within a single page load.



Refetches per endpoint. Seventeen of the thirty-two requests ask again for a schema already loaded in the same page session. *Reconstructed from PERF-001-Highlight-schema-requests.json.*

Recommended fix

1 · Identify the invariant schemas and request them in parallel

Schemas that are always needed regardless of booking type — guest details, room, rates — can be requested alongside the bootstrap schema rather than after it.

```
// libs/booking/data-access/src/services/bootstrap.service.ts
forkJoin({
  bootstrap: this.fetchBootstrapResponse(context),
  guestDetails:
    this.httpClient.get(this.configuration.fetchGuestDetailsSchema(context)),
  roomDetails:
    this.httpClient.post(this.configuration.fetchRoomDetailsSchema(context), payload),
}).pipe(...)
```

2 · Prefetch schemas on link hover

Warm the schemas when the user hovers the "New booking" link, using either a custom prefetch strategy or a direct `HttpClient` call.

3 · Merge bootstrap and default schema

Evaluate whether the backend can return the most common schemas inside `/reservas/schemas/default`, removing the separate round trips entirely.

4 · Deduplicate

Seventeen of the thirty-two requests repeat a schema already loaded. Add `shareReplay(1)` per URL in each schema service, or a shared `Map<url, Observable>` at interceptor level, so each unique endpoint is requested at most once per load. See **PERF-011**, which addresses the same class of problem at interaction level.

Evidence

CAPTURE CONDITIONS

Authenticated production, Spain (EU) → `/es-mx/reserva/crear`, MacBook Pro M3 on fibre, no throttling. Full conditions in **Section D**. These values are a lower bound for the Latin American market: every remaining serial hop is multiplied by an estimated 2 to 4× at 300–500 ms RTT.

File	Status	Notes
`PERF-001-no-cache-waterfall.png`	Collected	Full waterfall including asset downloads
`PERF-001-with-cache-waterfall.png`	Collected	API-only waterfall, isolating the schema chain
`PERF-001-recording.mp4`	Collected	Screen recording of the warm-cache delay
`PERF-001-Highlight-schema-requests.png`	Collected	Snippet console output
`PERF-001-Highlight-schema-requests.json`	Collected	Structured output — 32 requests, 6,173 ms

File	Status	Notes
`PERF-001-no-cache.har · with-cache.har`	Collected	—
`PERF-001-*-trace.json.gz`	Collected	Cold and warm traces

Expected improvement

Based on the measured data. The savings in the first two rows overlap: the combined target is not their sum.

Optimisation	Estimated saving
Remove the 17 redundant refetches	~400–800 ms
Parallelise secondary schemas with `default`	up to ~3,871 ms
Merge `default` and common schemas server-side	up to ~3,871 ms
Realistic combined target	**1,500–3,500 ms**

For users in Latin America at 300–500 ms RTT, every remaining serial hop is multiplied by an estimated 2 to 4× against the European baseline.

References

- **web.dev — LCP** · definition of the headline metric; understand what delays LCP before optimising it
- **web.dev — Optimize LCP** · full guide, with a section dedicated to eliminating resource load delay
- **web.dev — Critical rendering path** · why each additional network hop multiplies perceived latency
- **High Performance Browser Networking — HTTP/2** (Ilya Grigorik) · the canonical chapter on multiplexing and head-of-line blocking; explains why HTTP/2 does not eliminate application-level chains
- **RxJS — forkJoin** · run independent observables in parallel and wait for all of them
- **Chrome DevTools — network waterfall** · measuring timings and visualising the dependency chain

02 PERF-002 P2 · Strategic

Ship only the CSS you need

A single 1.25 MB stylesheet drags in every section of the application, and the chat widget's sheet blocks render for 108 ms while sitting 99.3% unused.

SCOPE	METRICS	EFFORT	OWNER	STATUS
All routes focus on `/reserva/crear`	FCP LCP	Medium	Frontend	Open

99.3% of `assistant-launcher.css` goes unused on initial render — and still costs a blocking 108 ms round trip on every load, even when cached.

Problem

All application styles are bundled into a single `main.scss` entry point that imports every section unconditionally:

Stylesheet	Critical?	Reason
`body.scss`	Critical	Base layout
`card.scss`	Critical	Card container
`compact-form-grid.scss`	Critical	Form grid
`form.scss`	Critical	Form controls
`navigation.scss`	Critical	Navigation bar
`skeleton.scss`	Critical	Loading skeleton
`utils.scss · z-index.scss`	Critical	Shared utilities and stacking
`guided-tour.scss`	Non-critical	Only after interaction
`voucher.scss`	Non-critical	Only after confirmation
`table.scss`	Non-critical	Secondary views
`toast.scss`	Debatable	Rarely needed at first paint

CSS for sections not visible at initial render — billing, extra services, documentation, voucher, guided tour — is parsed and applied anyway, adding unnecessary style recalculation. The icon sprite (861 KB uncompressed, see **PERF-014**) and third-party styles are also loaded globally.

Core Web Vitals impact

Metric	Effect
`FCP`	Delayed — blocking CSS must be fully parsed before the browser paints anything.
`LCP`	Delayed — the form skeleton cannot render until the CSS is applied.

CSS blocks rendering by design. Every extra kilobyte on the critical path delays FCP directly.

Measured impact

Captured with `devtools-snippet-perf002.js`, production, warm cache, Spain (EU) → `/es-mx/reserva/crear`.

CSS coverage. The Coverage tab reveals four files, two more than the snippet detects: `design-system-core.css` arrives via an `@import` inside the main bundle, which makes it invisible to a `<link>` filter. *Reconstructed from PERF-002-coverage.png and PERF-002-CSS_Stylesheet_Audit.json.*

Where the blocking sheets sit. The assistant launcher's sheet starts at 2,673 ms and costs 108 ms of real network time. *Reconstructed from PERF-002-CSS_Stylesheet_Audit.json.*

> **THOSE 108 MS ARE A REAL COST**
>
> Even though the file is cached, the browser still issues a conditional `GET` and receives a `304 Not Modified`. Rendering stays blocked while it waits for that response. **A cached file is not a free file.**

Recommended fix

1 · Extract the critical CSS

Identify the styles needed to render the visible skeleton without shifting — grid, navigation bar, card container, field base — and inline them in a `<style>` block in the shell's `index.html`, or use Angular's `inlineStyleLanguage` with a critical CSS extractor.

2 · Lazy-load non-critical global styles

```
private loadGuidedTourStyles(): void {
  const link = document.createElement('link');
  link.rel = 'stylesheet';
  link.href = '/assets/guided-tour.css';
  document.head.appendChild(link);
}
```

3 · Audit and remove unused global utilities

Run PurgeCSS, or `@angular-builders/custom-webpack` with a CSS tree-shaking pass, against actual template usage.

4 · Move route-specific styles into components

Billing, documentation and booking-detail styles should move their SCSS to component level. Worth verifying no duplication remains in the global bundle.

5 · Defer the assistant's stylesheet

It is third-party, render-blocking, fires at 2,673 ms, costs 108 ms per load even when cached, and is 99.3% unused at initial render. It should load asynchronously once the form is interactive, or only when the chat opens.

```
private loadAssistantStyles(): void {
  const link = document.createElement('link');
  link.rel = 'stylesheet';
  // URL taken from the <link> injected by the assistant script
  link.href = 'https://chat.assistant-vendor.com/...assistant-launcher.css';
  document.head.appendChild(link);
}
```

Evidence

CAPTURE CONDITIONS

Authenticated production, Spain (EU) → `/es-mx/reserva/crear`, MacBook Pro M3 on fibre, no throttling. Full conditions in **Section D**.

File	Status	Notes
`PERF-002-coverage.png`	Collected	Coverage tab with real CSS usage
`PERF-002-CSS_Stylesheet_Audit.png / .json`	Collected	Snippet output, console and structured
`PERF-002-recording.mp4`	Collected	Screen recording of the render delay
`PERF-002.har`	Collected	Waterfall with blocking position and sizes
`PERF-002-trace.json.gz`	Collected	Trace with `Parse Stylesheet` and `Recalculate Style` entries

Expected improvement

Optimisation	Estimated saving
Defer `assistant-launcher.css` removes the 108 ms round trip	~100–120 ms
Defer non-critical global styles	~50–150 ms of FCP
Remove or split the `design-system-core.css` `@import`	~100–300 ms of parsing
Realistic combined target	**200–500 ms of FCP**

For users in Latin America at 300–500 ms RTT, the `assistant-launcher.css` round trip alone can cost 300 to 500 ms on a cache miss.

References

- → **web.dev — FCP** · CSS blocks rendering by default and delays FCP directly
- → **web.dev — Eliminate render-blocking resources** · how CSS blocks and which strategies reduce its impact
- → **web.dev — Defer non-critical CSS** · loading non-critical CSS without blocking the render path
- → **web.dev — Extract critical CSS** · inlining above-the-fold CSS to remove the round trip
- → **Chrome DevTools — Coverage tab** · measuring the share of unused CSS and JS per load
- → **PurgeCSS** · CSS tree-shaking based on real template usage
- → **Angular — Component styles** · component-scoped styles to avoid unnecessary global CSS

03 PERF-003 P1 · Quick win

Preconnect to the external domains

Four external origins are contacted during load and not one of them has a connection hint. The browser pays full DNS, TCP and TLS at exactly the moment it can least afford it.

SCOPE	METRICS	EFFORT	OWNER	STATUS
All routes	FCP LCP	Low	Frontend + Platform	Open

0 `preconnect` hints in the document. The resource-hint audit found 2 `preload` and 1 `prefetch`, all three injected by third parties — and not a single `preconnect` or `dns-prefetch`.

Problem

The page requests resources from several external origins with no advance connection hint. The browser discovers those origins only after parsing HTML and CSS or executing JavaScript, and at that point it has to pay the full connection cost before receiving a single byte of content.

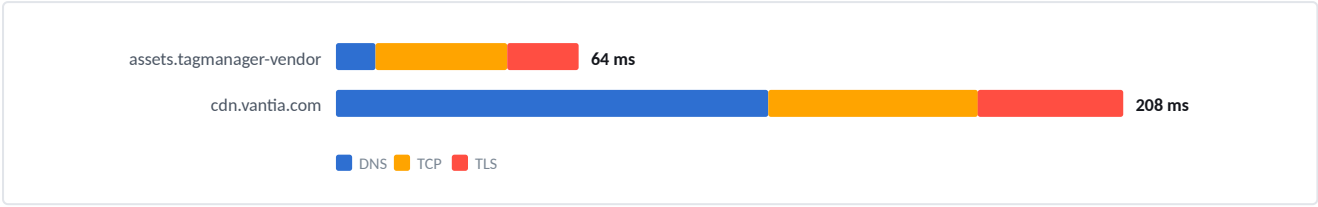
Origin	Purpose	Discovered at
`maps.geo-vendor.com`	Maps SDK and autocomplete	~3.5 s — on JS execution
`tiles.geo-vendor.com`	Map tiles and assets	After the SDK loads
`assets.tagmanager-vendor.com`	Tag manager	HTML parse
`cdn.vantia.com`	Corporate assets: fonts, header and footer	HTML parse
`chat.assistant-vendor.com`	Virtual assistant	Page load

DNS resolution plus the TCP handshake plus TLS negotiation costs between 100 and 300 ms per origin from Spain, and it is paid in full during load.

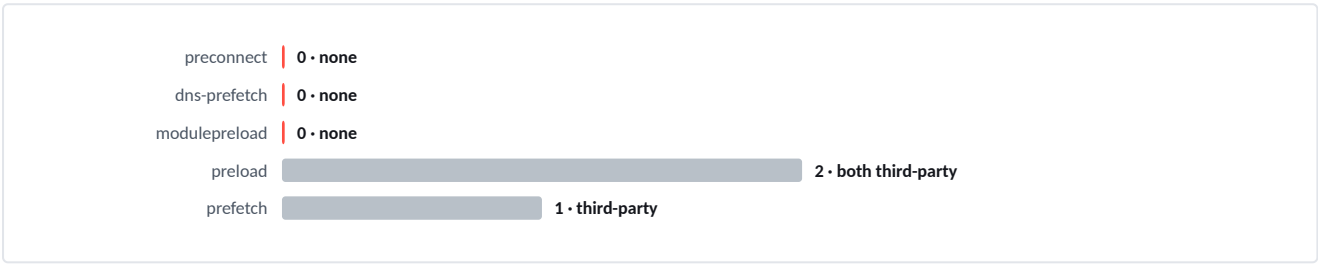
Core Web Vitals impact

Metric	Effect
FCP	Delayed — scripts and styles from external origins block or defer rendering.
LCP	Delayed — late-discovered external resources extend the critical path.

Measured impact



Connection cost per origin. `cdn.vantia.com` alone spends 208 ms on the handshake before even sending the request. That is time no application code can recover. *Reconstructed from PERF-003-tagmanager-timing.png and PERF-003-cdn-timing.png.*



Resource hints present in the document. The only three that exist are injected by third-party scripts. The application contributes none. *Reconstructed from PERF-003-resource-hints-audit.png.*

Recommended fix

1 · Static hints in the shell `<head>`

```
<!-- Critical preconnects: open the connection before JS triggers it -->
<link rel="preconnect" href="https://assets.tagmanager-vendor.com">
<link rel="preconnect" href="https://cdn.vantia.com" crossorigin>
```

2 · Programmatic hint for the maps SDK, on intent only

The map must not be preconnected at initialisation: **PERF-013** is precisely about its eager loading. The hint is added when the user shows intent by hovering the address field.

```
// libs/core/main/src/ui/places-autocomplete/maps-api.service.ts
private preconnectToMaps(): void {
  ['https://maps.geo-vendor.com', 'https://tiles.geo-vendor.com'].forEach((origin) => {
    if (!document.querySelector(`link[href="${origin}"]`)) {
      const link = this.document.createElement('link');
      link.rel = 'preconnect';
      link.href = origin;
      this.document.head.appendChild(link);
    }
  });
}
```

Called on the address field's `mouseenter`: by the time the script tag is added, the connection is already warm.

Evidence

CAPTURE CONDITIONS

Authenticated production, Spain (EU), MacBook Pro M3 on fibre, no throttling. Full conditions in **Section D**. At 300–500 ms RTT the per-origin cost is 2 to 4× higher.

File	Status	Notes
`PERF-003-cold-waterfall.png`	Collected	Cold waterfall with DNS and connection columns
`PERF-003-cold.har`	Collected	Cold HAR with per-origin detail
`PERF-003-tagmanager-timing.png`	Collected	Tag manager Timing panel
`PERF-003-cdn-timing.png`	Collected	`cdn.vantia.com` Timing panel
`PERF-003-resource-hints-audit.png`	Collected	Confirmation of `preconnect: 0`
`PERF-003-maps-timing.png`	Pending	Maps SDK Timing panel (optional)
`PERF-003-trace.json`	Pending	Trace showing DNS/TCP/TLS cost on the timeline

Expected improvement

Each cold handshake removed saves 100 to 300 ms on fibre from Spain, at roughly 120 ms RTT to US-hosted origins. With three or four external origins, a total saving of **300 to 900 ms** on first load is realistic.

For the Latin American market at 300–500 ms RTT the per-origin cost is 2 to 4× higher, so the improvement scales accordingly: between 600 ms and 1.6 s per cold load.

References

- → **web.dev — FCP** · the metric most affected by external connection latency
- → **Ilya Grigorik — Eliminating Roundtrips with Preconnect** · the reference article by the author of *High Performance Browser Networking*; explains the DNS, TCP and TLS layers and how `preconnect` removes each one
- → **web.dev — Preconnect to required origins** · when to use `preconnect` rather than `dns-prefetch`
- → **MDN — Link types: preconnect** · attribute specification
- → **Chrome DevTools — Network request timing** · measuring DNS Lookup, Initial Connection and SSL per request
- → **web.dev — Resource hints** · full guide to `preconnect`, `dns-prefetch`, `preload`, `modulepreload` and `prefetch`

04 PERF-004 P1 · Quick win

Preload the fonts

The first font is not discovered until 4,713 ms after navigation, because the browser has to download and parse the entire CSS bundle before it learns the font exists.

SCOPE	METRICS	EFFORT	OWNER	STATUS
All routes fonts are global	LCP CLS	Low	Frontend + Platform	Open

4,713 ms before the first font request starts. The snippet puts it bluntly: with a correct `<link rel="preload" as="font">` this should be 0 ms. The full font-loading window spans 3,542 ms.

Problem

The corporate fonts load as web fonts referenced by `@font-face` rules in the CSS. The browser cannot discover a font URL until it has:

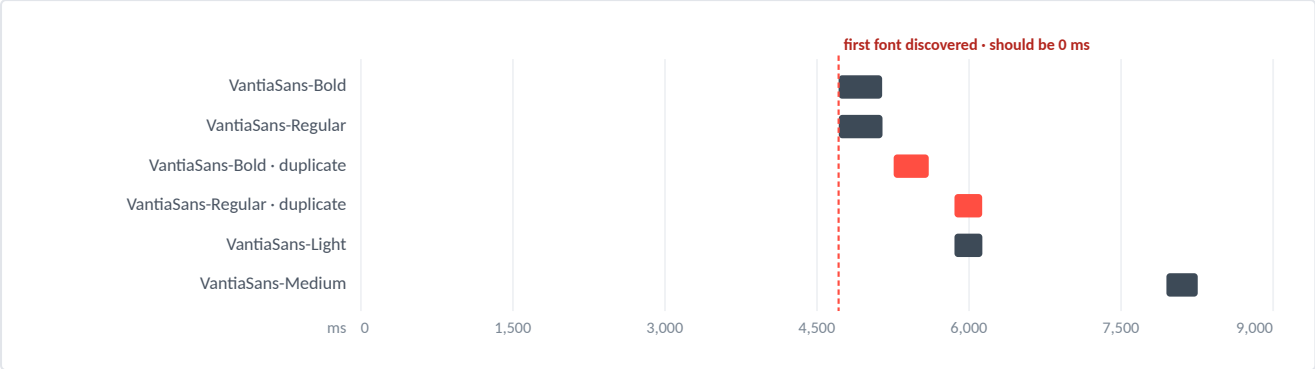
1. Downloaded and parsed the complete CSS bundle
2. Found an element that needs that font
3. Matched the `@font-face` rule

This three-step chain produces 300 to 600 ms of invisible or fallback text before the real face paints. FOUT and FOIT are two visible symptoms of the same discovery delay.

Core Web Vitals impact

Metric	Effect
LCP	Delayed — the LCP element, typically a heading or a form label, cannot paint in the correct face until the file arrives.
CLS	At risk — swapping the fallback for the real face shifts layout when the font metrics differ.

Measured impact



Font loading timeline. Six requests. The two red bars are duplicates: `Bold` and `Regular` are each requested twice. *Reconstructed from PERF-004-no-cache-font-duplicates.png and the snippet output.*

4,713 ms	8,255 ms	3,542 ms	2
first font discovered	all fonts ready	loading window	fonts requested twice

SECOND FINDING — DUPLICATE REQUESTS

`VantiaSans-Bold.woff2` and `VantiaSans-Regular.woff2` are each requested twice. Every duplicate wastes bandwidth and adds load to the critical path. Worth investigating alongside the preload work: preloading a font that is then requested twice does not fix the underlying double reference.

Recommended fix

1 · Preload the critical variants in the `<head>`

```
<!-- Only the fonts on the critical rendering path -->
<link rel="preload" href="/assets/fonts/VantiaSans-Regular.woff2"
      as="font" type="font/woff2" crossorigin>
<link rel="preload" href="/assets/fonts/VantiaSans-Bold.woff2"
      as="font" type="font/woff2" crossorigin>
```

Only those used above the fold. Preloading every weight and style wastes bandwidth and competes with more critical resources.

2 · Add `font-display: swap` to every rule

```
@font-face {
  font-family: 'VantiaSans';
  src: url('../fonts/VantiaSans-Regular.woff2') format('woff2');
  font-display: swap; // text stays visible while loading
}
```

3 · Resolve the duplicates

Trace which two declarations reference the same `Bold` and `Regular` files, and consolidate them.

Evidence

CAPTURE CONDITIONS

Authenticated production, Spain (EU), MacBook Pro M3 on fibre, no throttling. Full conditions in **Section D**.

File	Status	Notes
`PERF-004-no-cache.har · with-cache.har`	**Collected**	Warm serves fonts from disk, but the discovery delay persists
`PERF-004-*-trace.json.gz`	**Collected**	Confirm the gap is not a caching problem
`PERF-004-no-cache-font-detail.png`	**Collected**	Request detail: "Initiated by `main-[hash].css`"
`PERF-004-no-cache-font-duplicates.png`	**Collected**	Font filter: `Bold` and `Regular` ×2
`PERF-004-*-waterfall.png`	**Pending**	Cold and warm waterfalls

Expected improvement

Preloading two or three critical files removes the discovery chain and can improve LCP by **200 to 400 ms** on an uncached first visit, measured from the Spain/fibre baseline. Repeat visits are unaffected, since the fonts come from cache.

At 300–500 ms RTT the discovery delay is proportionally larger, putting the estimated improvement 2 to 4× higher: roughly 400 to 800 ms per cold load.

References

- → **web.dev — LCP** · the LCP candidate is usually text that depends on the custom font
- → **Zach Leatherman — Comprehensive Guide to Web Font Loading Strategies** · the definitive reference in this space; eleven-plus strategies with their trade-offs, FOUT, FOIT and the Critical FOFT pattern
- → **web.dev — Web font best practices** · `preload`, `font-display`, self-hosting versus CDN, and their impact on Core Web Vitals
- → **web.dev / MDN — font-display** · every descriptor value and when to use each
- → **CSS Font Loading API** · detecting when a font is ready without polling or depending on CSS

05 PERF-005 P0 · Attack first

High TTFB and CDN caching disabled by the origin

The CDN sits in front of the backend and caches nothing, because the origin sends `no-store` on every response. Fifty-five calls per session, 342 ms on average, every one of them reaching the origin.

SCOPE	METRICS	EFFORT	OWNER	STATUS
All API routes	LCP TTI INP	Very high	Infra + Backend	Open

18.8 s of cumulative network time in a single session: 55 API calls, every one a CDN cache miss. Served from the edge they would total between 0.5 and 1 s.

THE ROOT CAUSE THAT MULTIPLIES

This is the issue that makes every other one more expensive. Even after PERF-001 removes the serial chaining, each individual call will still carry 235 to 342 ms of overhead. Fixing the chaining reduces the number of round trips; fixing this reduces the price of every one that remains.

Problem

The booking backend is deployed in the corporate cloud, in a US region, behind the corporate CDN. But the CDN effectively acts as a pass-through: every request is a cache miss, because the origin sends `Cache-Control: no-cache, no-store, max-age=0` on all its responses. Every request reaches an origin that is nowhere near the users.

The `server-timing` header on each schema response states it unambiguously:

```
server-timing: cdn-cache; desc=MISS
server-timing: edge; dur=108
server-timing: origin; dur=75
```

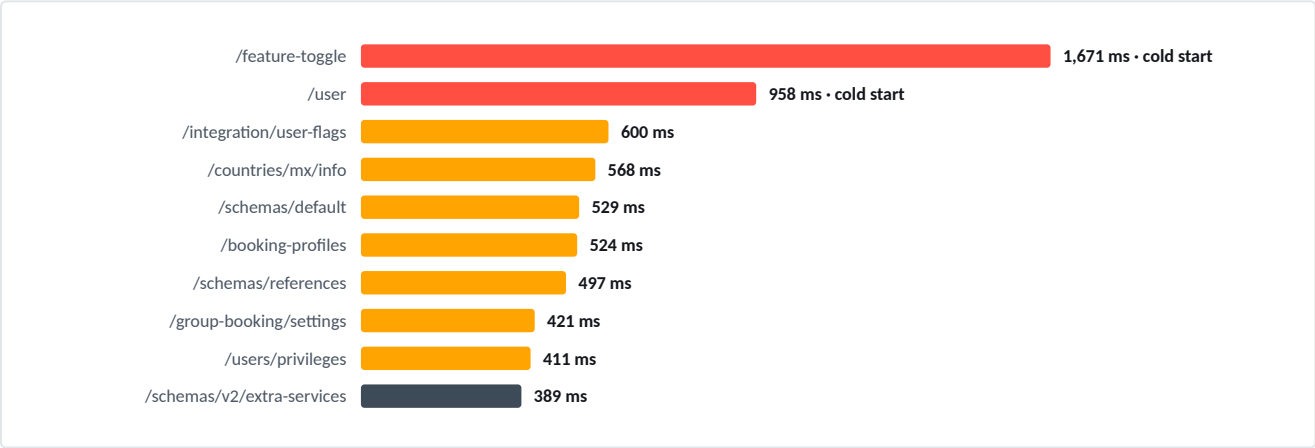
Confirmed measurements

Two captures, Chrome stable, Spain, 1 June 2026.

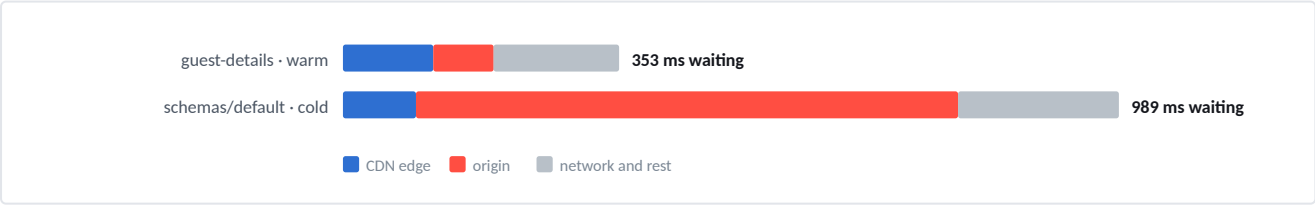
Metric	Warm HAR, 29 calls	Snippet, full session, 55	Served from the edge
CDN cache status	MISS on all	—	HIT
Duration, warm requests	210–345 ms mean ~235 ms	173–957 ms mean ~342 ms	~10–20 ms

Metric	Warm HAR, 29 calls	Snippet, full session, 55	Served from the edge
Cold start · `schemas/default`	4,328 ms TTFB	—	~10–20 ms
Cold start · `/feature-toggle`	—	1,671 ms	~10–20 ms
Cold start · `/user`	—	958 ms	~10–20 ms
Edge processing, warm	103–136 ms	—	—
Origin processing, warm	70–90 ms	—	—
Total calls	29	55	—
Cumulative duration	—	18.8 s	~0.5–1 s
Overhead versus a static CDN asset	**—**	**~211 ms per call**	**~0 ms**

The HAR was captured on a single navigation to /reserva/crear. The snippet ran across a full session with form interactions, which is what exposes the real call volume and the cold starts beyond schemas/default.



The ten slowest calls of the session. The first three are cold starts. The mean across all 55 calls is 342 ms; the maximum is 1,671 ms. *Reconstructed from PERF-005-TTFB-01.png and -02.png.*



Where the waiting time goes. On a warm request the origin already contributes 77 ms out of 353. On a cold start, 691 ms of 989 are pure origin time. *Reconstructed from PERF-005-cdn-cache-miss-headers.png and PERF-005-cold-start-timing.png.*

Root causes

1 · The origin's own headers disable caching

```
cache-control: no-cache, no-store, max-age=0, must-revalidate
pragma: no-cache
expires: 0
```

`no-store` tells the CDN never to cache the response. The CDN infrastructure is already deployed and paid for: the headers are what stop it being effective.

2 · Cold-start penalty on `schemas/default`

The first schema request of each session takes 4,328 ms TTFB, of which 3,667 ms is spent at the origin. Subsequent ones drop to around 75 ms. That signature is consistent with lazy initialisation: connection pooling, VM warm-up or an on-demand computation.

3 · The origin is nowhere near the users

Even warm, a TTFB of 210–345 ms indicates real geographic distance between edge and origin. With 29 calls in the form load, that accumulates fast.

> **NOTE ON AN EARLIER INVESTIGATION GAP**
>
> This issue sat as "pending investigation" because Chrome Canary DevTools only recorded the `OPTIONS` preflight requests and hid the real calls. Chrome stable captures all of them. Every HAR attached here was taken with Chrome stable.

Recommended fix

Options by return on effort.

Option 1 · Remove `no-store` from the schema responses

Highest return, lowest effort. The infrastructure is already there; the only blocker is the origin's `Cache-Control`.

```
Cache-Control: private, max-age=30
```

Or, to let the CDN cache but not the browser:

```
Cache-Control: no-cache, s-maxage=60
Surrogate-Control: max-age=60
```

Schema responses are largely deterministic per account and role. A short TTL with a cache key that includes the account identifier would be safe and effective, taking TTFB from ~235 ms to ~10–20 ms on cached responses.

Option 2 · Fix the cold start

The 3.67 s of origin time on the first call needs backend investigation. Likely causes: lazy service initialisation, connection pool expiry after idle, or an expensive first computation. Fixes: keep-alive connections, eager initialisation

at deploy time, or a scheduled warm-up ping.

Option 3 · Regional gateway

Deploy a lightweight gateway near the market that terminates TLS regionally and proxies to the origin. Shortens the transatlantic leg regardless of caching policy.

Option 4 · Full regional deployment

Deploy the backend in a region close to the market. Maximum latency reduction; requires coordination with backend, infrastructure and the data residency and compliance teams.

Option 5 · Batch requests at the edge

Use CDN edge functions to batch several schema requests into a single round trip to the origin, reducing the number of trips even if the origin stays where it is.

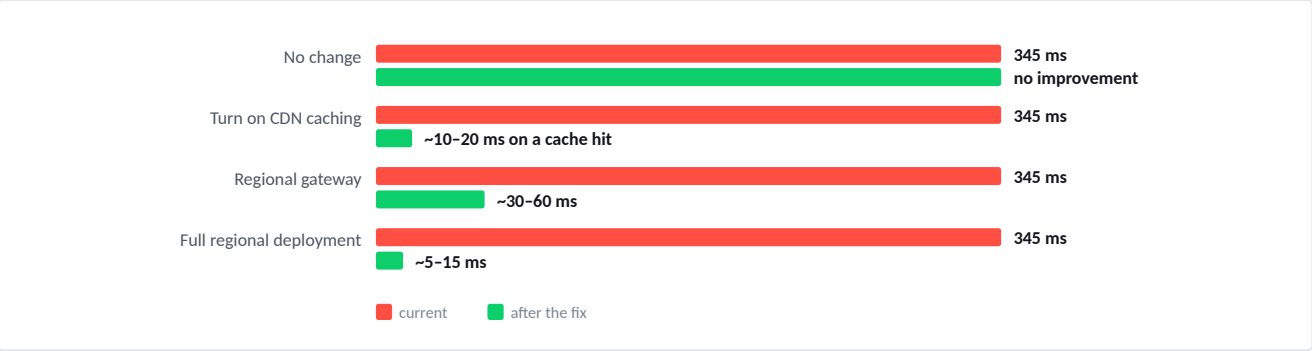
Evidence

CAPTURE CONDITIONS

Authenticated production, Spain (EU), MacBook Pro M3 on fibre, no throttling, **Chrome stable only**. Full conditions in **Section D**.

File	Status	Notes
`PERF-005-warm-cache.har`	Collected	29 calls, `cdn-cache: MISS`, TTFB 210–345 ms
`PERF-005-with-cache.har`	Collected	Second warm capture; confirms the pattern across sessions
`PERF-005-no-cache.har`	Collected	Cold; `schemas/default` TTFB ~4,328 ms
`PERF-005-cdn-cache-miss-headers.png`	Collected	`cdn-cache; desc=MISS` next to `Cache-Control: no-store`
`PERF-005-cold-start-timing.png`	Collected	Timing panel with the TTFB bar at ~4,300 ms
`PERF-005-TTFB-01.png · -02.png`	Collected	Snippet output: TTFB per endpoint
`PERF-005-network-ttfb.png`	Pending	Network panel with a readable TTFB column
`PERF-005-us-baseline.har`	Pending	US baseline to measure the RTT delta
`PERF-005-trace.json`	Pending	Trace with per-call TTFB and the cold-start spike

Expected improvement



TTFB per call by option. Turning on CDN caching is the highest-return, lowest-effort fix in the whole report: the infrastructure already exists and only the headers need to change. *Projections against the measured 210–345 ms baseline.*

Fix	TTFB per call	Cold start
Current baseline Spain/fibre	210–345 ms	4,328 ms
Turn on CDN caching	~10–20 ms on a hit	removed on cached endpoints
Fix the cold start only	210–345 ms	~75 ms
Regional gateway	~30–60 ms	~30–60 ms
Full regional deployment	~5–15 ms	~5–15 ms

The baseline figures reflect Spain on fibre. For the Latin American market at 300–500 ms RTT, warm TTFB is estimated between 500 and 900 ms per call, which worsens the cumulative session cost considerably (55 calls × ~700 ms average) and makes the caching fix proportionally more valuable.

References

- → **web.dev — Optimize TTFB** · CDN strategy, response caching and Early Hints
- → **web.dev — LCP** · every millisecond of TTFB delays LCP by the same amount
- → **web.dev — INP** · interactions that trigger live calls also carry the geographic latency
- → **HTTP `server-timing` header** · CDN and origin breakdown visible directly in the network panel
- → **High Performance Browser Networking — HTTP/2** · why latency dominates even with multiplexing
- → **Your CDN's HTTP response caching documentation** · how it honours and overrides `Cache-Control` headers

06 PERF-006 P1 · Quick win

Defer the non-critical JavaScript

Three scripts in the shell `<head>` carry neither `defer` nor `async`. None is needed to paint the form skeleton, and all three stop HTML parsing at 463 ms.

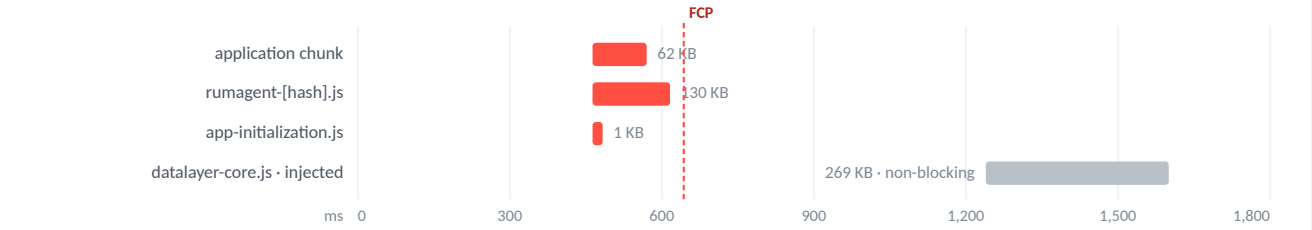
SCOPE	METRICS	EFFORT	OWNER	STATUS
All routes platform shell	FCP LCP TBT	Low-medium	Frontend + Platform	Open

193 KB of blocking JavaScript — roughly 378 KB uncompressed — across three static scripts in the `<head>`, all starting at 463 ms and blocking the 643 ms FCP.

Problem

Several sizeable JavaScript files load synchronously in the `<head>`, blocking parsing and rendering.

Script	Wire size	Behaviour
`application chunk [hash]`	62 KB 185 KB uncompressed	Render-blocking — static, no `defer` / `async`
`rumagent-[hash].js RUM agent`	130 KB	Render-blocking — static, no `defer` / `async`
`app-initialization.production-[hash].js`	1 KB	Render-blocking — static, no `defer` / `async`



Blocking scripts against FCP. All three start simultaneously at 463 ms; FCP does not arrive until 643. The grey bar is the data layer, which the browser does not classify as blocking but which still occupies the main thread. *Reconstructed from PERF-006-network-blocking.png and PERF-006-trace-fcp-blocked.png.*

INJECTED SCRIPTS ARE A SEPARATE PROBLEM

`datalayer-core.js` (~900 KB) and the header-and-footer script (~904 KB) are injected by Angular at runtime. The browser assigns them low network priority and does not classify them as blocking: `renderBlockingStatus` is `"non-blocking"`. They still execute on the main thread during bootstrap and contribute to TBT, so delaying their injection to idle time (fix 2) still applies.

Core Web Vitals impact

Metric	Effect
FCP	Heavily blocked — HTML parsing stops at every synchronous `<script>` until it is downloaded, parsed and executed.
LCP	Delayed — the browser cannot paint until every blocking script finishes.
TBT	High — large scripts generate long main-thread tasks that block interactivity.

Measured impact

Metric	Measured	Conditions
Blocking JS transferred	193 KB	Cold cache, Spain, fibre, no throttling
Blocking JS uncompressed	~378 KB	Cold cache, Spain, fibre, no throttling
Blocking scripts in `<head>`	3	Static, no `defer` / `async`
Download start	~463 ms	Cold cache
FCP	~643 ms	Cold cache
Data layer injection start	~1,239 ms	Cold cache

Recommended fix

1 · Add `defer` to every static script in the `<head>`

```
<!-- Before -->
<script src="/booking/assets/app-initialization.production-[hash].js"></script>
<script src="/booking/rumagent-[hash].js"></script>

<!-- After -->
<script src="/booking/assets/app-initialization.production-[hash].js" defer></script>
<script src="/booking/rumagent-[hash].js" defer></script>
```

2 · Delay the data layer injection

```
// libs/core/main/src/datalayer/datalayer-loader.service.ts
async load(): Promise<void> {
  if (this.configuration.dataLayer.scriptUrl) {
    await new Promise<void>((resolve) =>
      'requestIdleCallback' in window
        ? requestIdleCallback(() => resolve())
        : setTimeout(resolve, 200)
    );
    // existing injection logic...
  }
}
```

3 · When to use `async` and when `defer`

Attribute	Behaviour	Use for
`async`	Downloads in parallel and executes as soon as ready; can interrupt parsing	Fully independent scripts, such as analytics
`defer`	Downloads in parallel and executes after parsing, in document order	Scripts that depend on the DOM, such as header and footer

Evidence

CAPTURE CONDITIONS

Authenticated production, Spain (EU), MacBook Pro M3 on fibre, no throttling. Full conditions in **Section D**.

File	Status	Notes
`PERF-006-no-cache.har · with-cache.har`	Collected	Warm still delays render through parse and evaluation
`PERF-006-*-trace.json.gz`	Collected	"Evaluate Script" blocks before the FCP marker
`PERF-006-network-blocking.png`	Collected	The three blocking scripts at high priority
`PERF-006-trace-fcp-blocked.png`	Collected	FCP delayed by evaluation tasks
`PERF-006-coverage.png`	Collected	Unused percentage of each blocking script

Expected improvement

Adding `defer` to the three static scripts removes the parse-blocking phase at 463 ms. On European desktop fibre FCP is already at ~643 ms, so the headroom under ideal conditions is modest — but the benefit multiplies on slow connections and at 300–500 ms RTT.

Delaying the data layer injection (269 KB, starting at 1,239 ms) to idle time additionally reduces TBT.

References

- → **web.dev — FCP** · every blocking script delays the first pixel on screen
- → **V8 — The cost of JavaScript in 2019** (Addy Osmani) · essential reference from the V8 team: parse, compile and execute are three separate expensive phases; 300 KB of JS is not equivalent to 300 KB of image
- → **web.dev — Eliminate render-blocking resources** · identifying what blocks FCP and moving it off the critical path
- → **MDN — `<script>` defer / async** · execution order guarantees and timing relative to parsing
- → **MDN — requestIdleCallback** · scheduling non-critical work in the browser's idle periods

07 PERF-007 P1 · Quick win

Consolidate and defer the third-party scripts

Analytics and tracking are the largest avoidable source of network and CPU cost in the application — and the chat widget keeps polling for eight minutes for users who never open it.

SCOPE	METRICS	EFFORT	OWNER	STATUS
All routes	FCP LCP TBT INP	Medium	Frontend + Analytics	Open

5.35 MB uncompressed — 828 KB transferred — of third-party code on a cold load, excluding the personalisation tracker and the beacon responses. Across more than 28 requests.

Problem

Tag manager

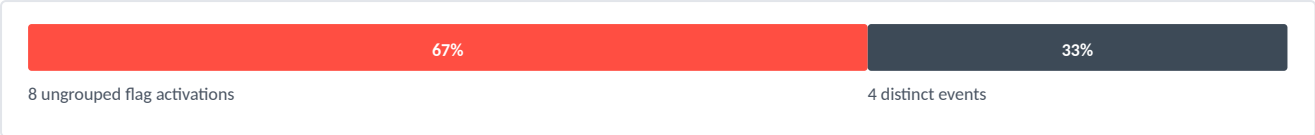
- 466 KB compressed / 2.25 MB uncompressed across 13 requests: one main bundle and twelve extension files
- Queued at 4.06 s into the page lifecycle — injected dynamically by the shell, not a parser-discovered `<script>` tag
- Cold connection cost to the vendor CDN: DNS 10 ms + TCP 33 ms + SSL 18 ms = 61 ms of handshake before a single byte transfers
- `EvaluateScript` for the main bundle: a 51.3 ms long task on the main thread
- Not needed until the page is interactive

Virtual assistant (chat widget)

- 362 KB compressed / 3.1 MB uncompressed across more than 15 requests on first load, excluding API calls
- `chat-launch.js` and `chat-loader.min.js` each load **twice**: once from the page context and once from a high-priority parsed iframe. This is structural duplication caused by the eager initialisation pattern.
- Background activity continues for the whole session — polling was observed for over 8 minutes — even when the user never opens the chat
- Only needed when the user explicitly opens the chat; loads on every visit

Analytics beacons

Twelve beacons per page load to the collection endpoint. Each URL carries a unique random suffix, which is the platform's standard cache-busting mechanism, not a per-event identifier.



Composition of the twelve beacons. Eight correspond to distinct feature flag activations; four are genuinely different events. *Reconstructed from PERF-007-beacon-burst.png*.

THE PROBLEM IS NOT DUPLICATE DATA

Eight of the twelve share the same activation event type, but each carries a different flag identifier and variant: these are eight feature flags active on `/reserva/crear`, not eight copies of the same event. The problem is that they are sent as **eight serial HTTP requests within 430 ms where one batched request would do**. The analytics platform supports event batching; the experimentation platform fires one beacon per activation with no batching at all.

The remaining four are genuinely distinct events: page view (×2), form ready, and a click on the rate schema response.

Personalisation tracker — newly identified

An additional tracker loads from a personalisation vendor's CDN. Its scope and impact are not yet measured. It should be included in the lazy-loading audit alongside the tag manager.

Measured impact

Uncompressed weight per vendor. Cold cache, Spain, fibre, no throttling. The measured total, excluding personalisation and beacons, is 828 KB transferred and 5,350 KB uncompressed across more than 28 requests. *Reconstructed from PERF-007-no-cache.har and the associated traces*.

Recommended fix

Tag manager — load after interactivity

```
// Replace the eager script with a post-render injection
afterNextRender(() => {
  const script = document.createElement('script');
  script.src = '//assets.tagmanager-vendor.com/container-xxx.min.js';
  script.async = true;
  document.head.appendChild(script);
});
```

Assistant — load on user interaction only

```
// libs/shell/src/services/virtual-assistant-display.service.ts
initOnInteraction(): void {
  const trigger = document.querySelector('[data-chat-trigger]');
  trigger?.addEventListener('click', () => this.loadChatWidget(), { once: true });
}

private loadChatWidget(): void {
  // inject the assistant script here, and only now
}
```

Analytics — batch the flag activations

```
private readonly pendingActivations: FlagActivation[] = [];
private flushScheduled = false;

trackFlagActivation(activation: FlagActivation): void {
  this.pendingActivations.push(activation);
  if (!this.flushScheduled) {
    this.flushScheduled = true;
    queueMicrotask(() => this.flushActivations());
  }
}

private flushActivations(): void {
  // send a single beacon with all pending activations
  this.flushScheduled = false;
  this.pendingActivations.length = 0;
}
```

Personalisation — audit and defer

Determine whether the personalisation script is needed for initial render. If not, apply the same `afterNextRender()` pattern.

Evidence

CAPTURE CONDITIONS

Authenticated production, Spain (EU), MacBook Pro M3 on fibre, no throttling. Full conditions in **Section D**.

File	Status	Notes
`PERF-007-no-cache.har · with-cache.har`	Collected	—
`PERF-007-*-trace.json.gz`	Collected	Confirm the 51.3 ms `EvaluateScript` and the iframe duplication
`PERF-007-tracking-waterfall-tagmanager.png`	Collected	13 requests, 466 KB / 2,294 KB
`PERF-007-tracking-waterfall-assistant.png`	Collected	Iframe duplication visible
`PERF-007-beacon-burst.png`	Collected	All 12 beacons and the headers panel

File	Status	Notes
`PERF-007-tagmanager-timing.png`	Pending	Detail: queued 4.06 s, handshake 61 ms, total 138 ms
`PERF-007-assistant-zoom.png`	Pending	Same capture zoomed to 0–15 s

Expected improvement

- **Deferring the tag manager** takes 466 KB (2.25 MB uncompressed) and a 51.3 ms task off the critical path, and removes the 61 ms cold handshake for first-time visitors.
- **Fixing the assistant initialisation** removes the iframe duplication, takes more than 362 KB out of the initial load for the large majority of users who never open the chat, and stops the eight minutes of background polling.
- **Batching the beacons** removes seven of the twelve analytics requests per load.
- **Auditing personalisation** may yield further savings once its scope is measured.

References

→ **web.dev — INP** · third-party scripts are the leading cause of poor INP: they compete with interaction handlers on the main thread

→ **Harry Roberts — Identifying, Auditing, and Discussing Third Parties** · a framework for auditing, classifying and negotiating third-party cost with stakeholders

→ **web.dev — Best practices for tags and tag managers** · when to load them and how to measure their real Core Web Vitals impact

→ **web.dev — Third-party JavaScript** · impact, diagnosis and mitigation: facades, lazy loading, Partytown

→ **Chrome DevTools — Identify slow third-party scripts** · isolating cost per origin from the Performance panel

→ **Angular — afterNextRender** · the correct hook for deferring work until after the first render cycle

08 PERF-008 P2 · Strategic

Preload the critical application chunks

Two chunks totalling 2.8 MB are required to paint the form, and the browser does not learn they exist until it has executed the main bundle. Two gaps, 1,550 ms of pure waiting.

SCOPE	METRICS	EFFORT	OWNER	STATUS
`/reserva/crear`	LCP TTI	Low	Frontend + Platform	Open

~1,550 ms of serial discovery overhead on fibre — 720 ms before the third wave and 830 before the fourth — during which the network sits idle waiting for JavaScript to name the next file.

Problem

Two large lazy-loaded chunks are required to render the form, but are only discovered after the initial bundle executes.

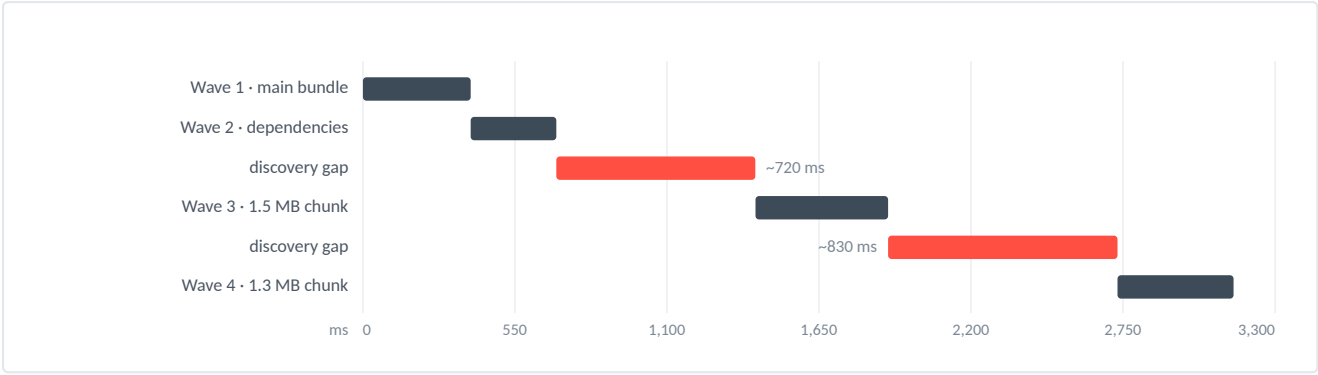
Chunk	Size	Discovery point
`chunk-[hash-a].js`	~1.5 MB ~396 KB compressed	After the main bundle executes
`chunk-[hash-b].js`	~1.3 MB ~262 KB compressed	After the main bundle executes

Because the chunks are discovered through dynamic `import()`, the browser cannot begin requesting them until the Angular router runs, which in turn depends on the main bundle being parsed and executed. The result is a serial dependency where 2.8 MB of critical code waits behind the bundle.

Core Web Vitals impact

Metric	Effect
LCP	Delayed — the form does not paint until the 1.5 MB chunk is downloaded and parsed.
TTI	Delayed — Angular cannot activate the route until both chunks are ready.

Measured impact



Load waves and discovery gaps. The two red bars are not downloads: they are time in which the network is idle because the browser does not yet know what to request. *Reconstructed from PERF-008-key-chunks-waterfall.png and PERF-008-no-cache.har.*

Recommended fix

Add `modulepreload` hints in the shell `<head>`:

```
<link rel="modulepreload" href="/now-web/chunk-[hash-a].js">
<link rel="modulepreload" href="/chunk-[hash-b].js">
```

CHUNK NAMES CARRY CONTENT HASHES

They change on every build, so a hand-written hint goes stale silently. It has to be kept in sync through one of the three mechanisms below.

1 · Build-time generation

```
// scripts/inject-modulepreload.js (inside the build pipeline)
const stats = require('./dist/now-web/stats.json');
const criticalChunks = stats.assets
  .filter(a => a.name.startsWith('chunk-') && a.size > 1_000_000)
  .map(a => `<link rel="modulepreload" href="/${a.name}">`);
// inject into the shell template...
```

2 · Named chunks

The build already sets `namedChunks: true`. It only remains to map names to hashed files in a build artifact manifest.

3 · Coordination with the platform team

If the shell is generated by the corporate platform, the `modulepreload` hints need to be included in its own pipeline.

Evidence

CAPTURE CONDITIONS

Authenticated production, Spain (EU), MacBook Pro M3 on fibre, no throttling. Full conditions in **Section D**.

File	Status	Notes
`PERF-008-no-cache.har · with-cache.har`	**Collected**	Four serial waves; the two large chunks in the third and fourth
`PERF-008-*-trace.json.gz`	**Collected**	Main bundle finishes and a visible gap precedes the chunks
`PERF-008-key-chunks-waterfall.png`	**Collected**	Waterfall with the waves and both gaps
`devtools-snippet-perf008.js`	**Collected**	Reusable snippet

Expected improvement

The two measured gaps total ~1,550 ms on fibre. Preloading both chunks in parallel with the main bundle can remove them, for a TTI improvement of **500 to 1,500 ms**.

At 300–500 ms RTT each discovery round trip costs considerably more, and the improvement is estimated 2 to 4× higher. The Spain/fibre values are a lower bound.

References

- → **web.dev — LCP** · critical chunks on the render path delay LCP directly
- → **web.dev — Module preload** · `rel=modulepreload` versus `rel=preload` : semantic differences and module graph pre-reading
- → **MDN —** `<link rel="modulepreload">` · specification: `as` , `crossorigin` and ES module integration
- → **Angular — Lazy loading feature modules** · how Angular generates chunks with `loadChildren` and its effect on the graph
- → **Angular CLI —** `--stats-json` · identifying the largest chunks and their dependency tree

09 PERF-009 P2 · Strategic

Speculation Rules for the confirmation navigation

The user has spent two to five minutes filling in the form. The final click — the one that confirms the booking — then costs them another one to three seconds of waiting.

SCOPE	METRICS	EFFORT	OWNER	STATUS
`/reserva/crear` → `/reserva/confirmada`	LCP TTI on the next page	Medium	Frontend	Open

1–3 s of navigation delay at the highest-intent moment of the entire funnel, measured on fibre with a warm cache. Estimated at 3–6 s for the Latin American market.

CORRECTION ON THE INTERFACE FLOW

In comfortable mode, pressing "View summary" opens a modal layer within `/reserva/crear` without changing the URL. The real navigation happens when the confirm button inside that layer is pressed, leading to `/reserva/confirmada`. In compact mode, "Confirm booking" navigates directly. **There is no intermediate review route**, although earlier drafts of this tracker described one.

Problem

Once the form is complete, the user navigates to `/reserva/confirmada`. At that point the browser has to:

1. Request the confirmation page's HTML and assets
2. Download the lazy chunk for the confirmation container
3. Make the API calls that confirm the booking

There is no prefetch strategy for this route. The delay lands exactly where frustration is most expensive: the user has already invested minutes of work and is committing to a purchase.

Core Web Vitals impact

Metric	Effect on `/reserva/confirmada`
LCP	No longer a wait — the assets are already preloaded before navigation.
TTI	Reduced — the confirmation-specific chunk is already cached.

Measured impact

Metric	Value
Navigation delay, create → confirmed	1–3 s
LCP on `/reserva/confirmada`	Trace pending
TTI on `/reserva/confirmada`	Trace pending

Recommended fix

Inject the speculation rules once the user has shown real progress through the form — for example, on validating the first step — rather than on page load.

```
// In comfortable-navigation.component.ts — from viewSummary()
// or from the showViewSummaryButton$ subscription
private injectSpeculationRules(): void {
  if (!HTMLScriptElement.supports?.('speculationrules')) return;         // detection
  if (document.querySelector('script[type="speculationrules"]')) return; // already injected

  const script = document.createElement('script');
  script.type = 'speculationrules';
  script.textContent = JSON.stringify({
    prefetch: [{
      urls: ['/booking/es-mx/reserva/confirmada'],
      eagerness: 'moderate',
    }],
  });
  document.head.appendChild(script);
}
```

For browsers without support — Safari and Firefox — fall back to a standard prefetch:

```
<!-- Fallback for browsers without Speculation Rules -->
<link rel="prefetch" href="/booking/es-mx/reserva/confirmada">
```

Eagerness level	Behaviour
`immediate`	Prefetches or prerenders straight away
`moderate`	When the link is visible in the viewport — the default, and the right choice here
`conservative`	Only on hover or pointer down

`moderate` avoids spending bandwidth on users who abandon the form early.

Evidence

CAPTURE CONDITIONS

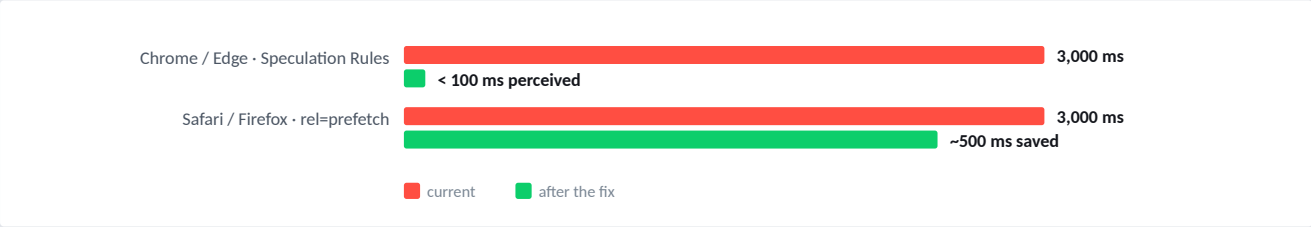
Authenticated production, Spain (EU), MacBook Pro M3 on fibre, no throttling. Full conditions in **Section D**.

File	Status	Notes
`PERF-009-baseline.har`	Collected	Warm cache, no rules active. Every confirmation-page resource starts from scratch on navigation.
`devtools-snippet-perf009.js`	Collected	Reusable snippet
`PERF-009-baseline-trace.json`	Pending	Full navigation timeline; the LCP and TTI cost the fix would remove
`PERF-009-nav-waterfall.png`	Pending	Waterfall with resources starting from scratch
`PERF-009-spec-rules-panel.png`	Pending	Empty Speculation Rules panel, confirming no rules exist

THIS CHAPTER HAS NO MEASUREMENT CHARTS YET

Unlike the rest of the report, PERF-009 rests on a single HAR. The 1–3 s baseline is real but coarse: the trace that would break it down into chunk time versus API time is missing. The improvement figures below are projections, not measurements.

Expected improvement



Projected navigation after the fix. Estimates against the measured 1–3 s baseline, not measurements. *Base: PERF-009-baseline.har.*

For the Latin American market at 300–500 ms RTT, the baseline is estimated between 3 and 6 s and the projected improvement scales in the same proportion.

References

- **Chrome Developers — Prerender pages for instant navigations** · full guide: eagerness levels, prerendering restrictions and behaviour on authenticated pages
- **MDN — Speculation Rules API** · specification: JSON syntax, `prefetch` and `prerender` fields, eagerness and `where` selectors
- **web.dev — `<link rel="prefetch">`** · the fallback that works in Safari and Firefox
- **Chrome DevTools — Debug Speculation Rules** · rule status, errors and active prerendered pages

10 PERF-010 P3 · Polish

Apply `content-visibility` to off-screen sections

Opening a sidebar menu repaints the entire form. The browser lays out 225 of the document's 242 nodes in a single pass, because nothing in the form is contained.

SCOPE	METRICS	EFFORT	OWNER	STATUS
`/reserva/crear` compact view	TBT INP	Low-medium	Frontend	Open

225 / 242 document nodes enter a single layout pass, with `#document` as the root. Consecutive frames of 133.5 ms and 108.4 ms — eight times the 16.6 ms threshold.

Problem

The compact form view renders every section in the DOM at once, even though only one is visible at a time. The summary panel additionally renders the full state of every completed step.

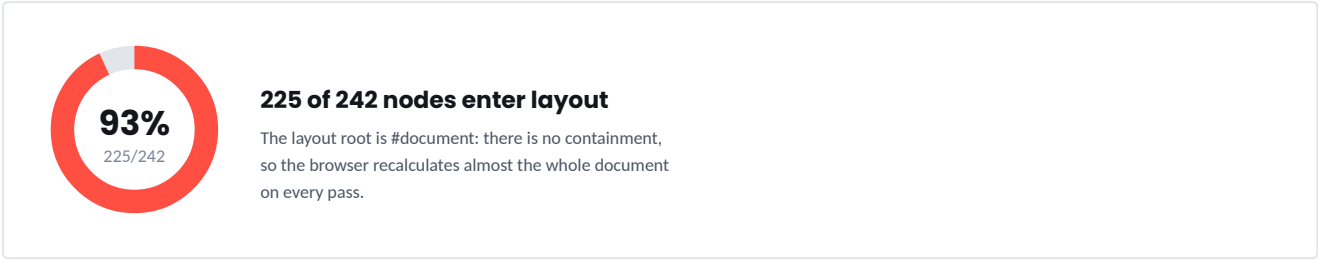
The browser lays out, recalculates style and paints all DOM nodes regardless of visibility. With fifteen or more sections mounted — billing, extra services, guest details, documentation and the rest — that is a lot of unnecessary render work. The step configuration defines more than thirty components, and all of them can exist in the DOM during transitions.

This is a problem felt most on modest devices, where the bottleneck is rendering rather than the network.

Core Web Vitals impact

Metric	Effect
TBT	Reduced by the fix — less layout and paint work means shorter main-thread tasks.
INP	Improved by the fix — fewer rendered elements means faster response to scrolling and input.

Measured impact



Share of nodes entering each layout pass. The DevTools summary panel for the selected task reports 3.41 ms duration, `#document` as root and, as first invalidation, Angular's change detection. *Reconstructed from PERF-010-flame-chart.png.*

Frame duration against the smoothness threshold. Two consecutive frames far above the 16.6 ms that 60 fps requires. *Reconstructed from the Frames track of PERF-010-flame-chart.png.*

THE MOST REVEALING SYMPTOM

With paint flashing enabled in DevTools, opening the sidebar submenu triggers repaint flashes across the main form content: fields with no relationship whatsoever to the interaction. This happens because the sections have no paint containment: any DOM change the browser cannot scope forces a repaint of the whole document. `content-visibility: auto` implicitly applies `contain: layout style paint`, which confines repaints to the element boundary.

Recommended fix

Option A · CSS `content-visibility`, no JavaScript changes

```
// In the compact form section styles
.booking-form-section {
  content-visibility: auto;
  contain-intrinsic-size: auto 300px;  // approximate height, avoids CLS
}
```

Option B · Angular `@defer (on viewport)` — recommended

This additionally defers component initialisation and change detection, not just layout and paint.

```
<guest-details-form [state$]="state$" />   <!-- critical: always -->

@defer (on viewport) {
  <billing-form [state$]="state$" />
}
@placeholder {
  <div class="section-placeholder" style="min-height: 300px"></div>
}

@defer (on viewport) {
  <extra-services-form [state$]="state$" />
}
@placeholder {
  <div class="section-placeholder" style="min-height: 300px"></div>
}
```

Evidence

CAPTURE CONDITIONS — NOTE THE DEVIATION

Authenticated production, Spain (EU), MacBook Pro M3 on fibre. No network throttling. **4× CPU throttling is applied where noted**, because for a rendering problem the representative bottleneck is CPU rather than network. Full baseline in **Section D**.

File	Status	Notes
`PERF-010-flame-chart.png`	**Collected**	Layout task with 225 dirty nodes
`PERF-010-paint-flashing.png`	**Collected**	Repaint triggered by an unrelated interaction
`PERF-010-recording.mp4`	**Collected**	Screen recording of the repaint
`PERF-010-baseline-trace.json`	**Pending**	Total Layout, Recalculate Style and Paint duration, with and without the fix
`PERF-010.har`	**Pending**	Low priority: this is a rendering problem, not a network one

Expected improvement

Applying `content-visibility: auto` or `@defer (on viewport)` excludes off-screen sections from the layout pass and brings the dirty node count down to the 40–60 actually in the viewport. Expected improvement: **a 40 to 60% reduction in layout and paint time**. On modest devices, 200 to 500 ms off Time to Interactive and noticeably smoother scrolling.

The expected remote-market multiplier is 2 to 4×, and here it is driven by weaker hardware rather than the network.

References

→ **web.dev — INP** · less layout and paint work means faster interaction response

→ **web.dev — content-visibility: the new CSS property that boosts your rendering performance** (Una Kravets and Vladimir Lebedev) · the original article, with measurements showing up to 7× reductions in render time

- → **MDN — content-visibility** · the `auto`, `hidden` and `visible` values and their relationship to CSS containment
- → **MDN — contain-intrinsic-size** · reserving space to avoid CLS when `content-visibility: auto` skips layout
- → **Angular — @defer blocks** · `on viewport`, `on idle` and `on interaction` triggers, plus `@placeholder`, `@loading` and `@error` blocks

11 PERF-011 P1 · Quick win

Deduplicate the API calls

Changing a single form field takes the session from 32 calls to 57. Seventeen of them are surplus round trips to endpoints that had already answered.

SCOPE	METRICS	EFFORT	OWNER	STATUS
`/reserva/crear`	TTI INP	Medium	Frontend	Open

~21.8 s of network time estimated as wasted after changing one field, against ~2.4 s on page load. Spain/fibre baseline; the remote market wastes 2 to 4× more.

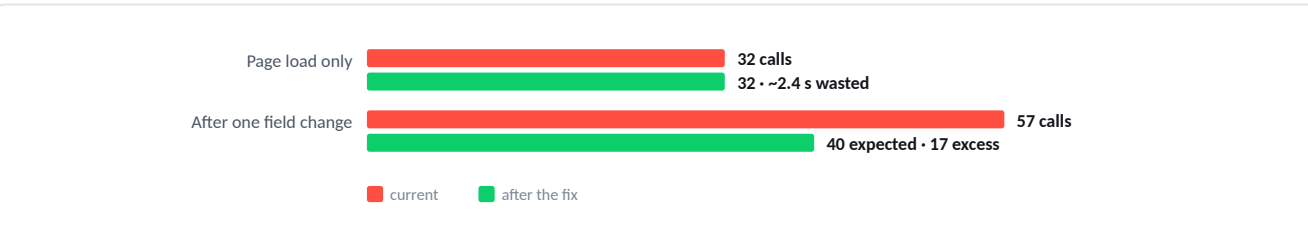
THIS COMPOUNDS WITH PERF-005

Every duplicate call costs a full round trip to the origin. Deduplicating removes the call; fixing PERF-005 makes the remaining ones cheaper. Both are worth doing, and they multiply each other.

Problem

Network analysis shows several endpoints being called far more often than necessary.

Calls per endpoint after one field change. Against the one or two that would be expected in each case. *Reconstructed from PERF-011-call-summary_after.json.*



Total call volume. Page load already carries excess; a single interaction makes it worse. *Reconstructed from PERF-011-call-summary.json and _after.json.*

Every duplicate call wastes a 150 to 250 ms round trip against the Spain/fibre baseline, backend CPU for an identical request, and 50 to 200 KB of the user's bandwidth per response.

Core Web Vitals impact

Metric	Effect
`TTI`	Degraded — redundant calls compete for the HTTP/2 connection's bandwidth and delay schema availability.
`INP`	Degraded — the calls triggered by a field change fire multiple times and produce visible jank.

Measured impact

Metric	Page load	After one field change
Total calls	32	57
Excess over the ~40 expected	—	+17
Endpoints with duplicates	4	11
Wasted network time (estimated)	~2.4 s	~21.8 s
Worst case	`v2/extra-services` ×5, mean 324 ms	`v2/extra-services` ×7, mean 670 ms

Recommended fix

Level 1 · `debounceTime` and `distinctUntilChanged`

```
formValueChanges$.pipe(
  debounceTime(300),
  distinctUntilChanged((a, b) => isEqual(a, b)),          // ignore identical payloads
  switchMap((payload) => this.schemaService.fetch(payload)), // cancel in-flight
)
```

`switchMap` is the critical operator: it cancels in-flight requests when a new trigger arrives, preventing an older response from landing after a newer one.

Level 2 · Deduplicate in-flight requests

```
// libs/shared/schemas/data-access — deduplication layer
@Injectable({ providedIn: 'root' })
export class SchemaRequestDeduplicator {
  private readonly inFlight = new Map<string, Observable<unknown>>();

  deduplicate<T>(key: string, request$: Observable<T>): Observable<T> {
    if (!this.inFlight.has(key)) {
      const shared$ = request$.pipe(
        share({ resetOnComplete: true, resetOnError: true }),
        finalize(() => this.inFlight.delete(key)),
      );
      this.inFlight.set(key, shared$);
    }
    return this.inFlight.get(key) as Observable<T>;
  }
}
```

Use a hash of the booking payload as the key.

Level 3 · Cache deterministic responses

For endpoints whose response is deterministic given the same input payload, cache client-side keyed on the payload hash, and invalidate when the booking state resets.

Evidence

CAPTURE CONDITIONS

Authenticated production, Spain (EU), MacBook Pro M3 on fibre, no throttling. Full conditions in **Section D**. At 300–500 ms RTT each duplicate costs 2 to 4× more.

File	Status	Notes
`PERF-011.har`	Collected	Warm cache, every call from a complete load
`PERF-011-trace.json.gz`	Collected	Repeated XHR blocks: no cancellation or deduplication in place
`PERF-011-duplicate-calls.png`	Collected	Full listing, sorted by name
`PERF-011-endpoint-detail.png`	Collected	Worst-case detail, with one request cancelled in flight
`PERF-011-call-summary.png / .txt`	Collected	Snippet output
`PERF-011-call-summary.json · _after.json`	Collected	Baseline and post-interaction

Expected improvement

After a single interaction, 57 calls are observed where roughly 40 are expected: 17 surplus round trips. At 150 to 670 ms each against the Spain/fibre baseline, removing them cuts the estimated wasted network time from **~21.8 s to**

essentially zero.

At 300–500 ms RTT the gain scales in the same proportion: the fix is worth considerably more to the priority market than the European figures suggest.

References

- → **web.dev — INP** · interaction-triggered duplicate calls degrade INP directly
- → **RxJS — switchMap** · cancels the previous observable when a new emission arrives; the antidote to reactive cascades
- → **RxJS — debounceTime** · suppresses intermediate emissions over an interval; essential on form changes
- → **RxJS — share** · multicasting a cold observable so several subscribers share one execution
- → **RxJS — distinctUntilChanged** · prevents emissions when the value has not changed; the necessary companion to `debounceTime`
- → **Angular — HttpClient** · interceptors as the centralised point for caching and deduplication

12 PERF-012 P3 · Infrastructure

Enable HTTP/2 103 Early Hints

For the first 463 ms of a cold load the browser has sent its request and received nothing. It cannot download a single resource. Early Hints turns that dead window into download time.

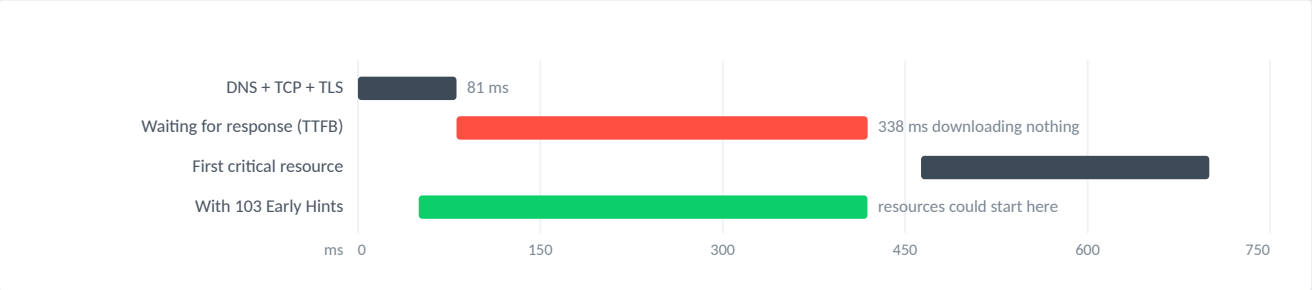
SCOPE	METRICS	EFFORT	OWNER	STATUS
All routes served by the shell	FCP LCP	Medium	Infra + Platform	Open

~413 ms of recoverable idle time per cold load: an observed 463 ms window minus roughly 50 ms of estimated latency for the 103 response.

Problem

The booking engine's TTFB sits around 250 ms, and in the capture below it measured 338 ms. For that entire window the browser has sent its request and received nothing: it cannot start requesting any resource because it does not yet know what they are.

HTTP/2 103 Early Hints lets the server send a preliminary response carrying `Link` preload headers *before* the full HTML response is ready. The browser can act on those hints immediately.



The idle window, and what Early Hints would fill it with. The full handshake is 81 ms; waiting for the response is 338. The green bar marks where critical resources could start downloading. *Reconstructed from PERF-012-ttfb-timing.png and PERF-012-ttfb-summary.png.*

Core Web Vitals impact

Metric	Effect
FCP	Improved — critical CSS and JS begin downloading during server processing time.
LCP	Improved — critical resources arrive earlier and unblock rendering.

The technique pays off precisely when TTFB is high, which per **PERF-005** is the normal condition here.

Measured impact

Cold connection — 2 June 2026, Spain → `/es-mx/`

Phase	Duration
DNS resolution	20.11 ms
Initial connection	38.98 ms
SSL	21.90 ms
Request sent	0.91 ms
Waiting for response — TTFB	338.11 ms
Content download	2.18 ms
Total	**402.58 ms**

The 338 ms TTFB is above the ~250 ms baseline in PERF-005, consistent with server variability. DNS, TCP and TLS together total around 81 ms: four times less than TTFB alone, which makes the idle window the dominant cost by a wide margin.

Warm connection — HTTP/2 reuse

Phase	Duration
Queueing and stalling	1.57 ms
Request sent	0.23 ms
Waiting for response — TTFB	228.40 ms
Content download	1.89 ms
Total	**232.09 ms**

Warm TTFB is lower because there is no connection cost, but it is still 228 ms during which nothing can download.

Absence of hints confirmed at both layers

- **HTTP layer** — the response headers contain no `Link:` header, so Early Hints is not in operation.
- **HTML layer** — the resource hints snippet confirms there is no application-owned `<link rel="preload">` or `<link rel="preconnect">` in the DOM. The first critical resource starts at 463 ms, against an HTML `responseStart` of 403 ms.

Recommended fix

Nginx

```
location /reserva/crear {
  add_header Link "</now-web/main.js>; rel=preload; as=script";
  add_header Link "</now-web/styles.css>; rel=preload; as=style";
  return 103;
  proxy_pass http://upstream;
}
```

At the CDN

Most modern CDNs offer Early Hints as a one-click option, and some even infer the `Link` headers from the previous HTML response. The specific assets to preload should come from the build's `stats.json`: the initial CSS and JS chunks needed on every load.

Evidence

CAPTURE CONDITIONS

Authenticated production, Spain (EU), MacBook Pro M3 on fibre, no throttling. Full conditions in **Section D**. In the Latin American market TTFB is estimated higher still, between 400 and 600 ms, which makes the technique proportionally more worthwhile.

File	Status	Notes
`PERF-012-no-cache.har`	Collected	Document TTFB ≈ 338 ms; all CSS and JS start after it
`PERF-012-no-cache-trace.json.gz`	Collected	TTFB gap before any download bar
`PERF-012-ttfb-timing.png · -warm.png`	Collected	Cold and warm connections
`PERF-012-response-headers-01.png · -02.png`	Collected	No `Link:` header
`PERF-012-resource-hints.png / .txt`	Collected	No application-owned preload or preconnect in the DOM
`PERF-012-ttfb-summary.png`	Collected	Numeric confirmation of the idle window
`PERF-012-ttfb-summary.txt`	Pending	Text version of the snippet output

Expected improvement

103 Early Hints can recover up to **~413 ms** of idle time per cold load. Critical resources would begin downloading during the TTFB wait rather than after it.

References

- → **web.dev — FCP** · Early Hints brings asset download start forward into the idle window
- → **Chrome Developers — HTTP 103 Early Hints** · full guide with examples, real-world impact and server support
- → **RFC 8297 — An HTTP Status Code for Indicating Hints** · specification: 103 semantics, supported `Link` fields and expected client behaviour
- → **Your CDN's Early Hints documentation** · enabling it and automatic `Link` header inference
- → **web.dev — Optimize TTFB** · Early Hints within the broader TTFB work: CDN, HTML streaming and edge caching

13 PERF-013 Fixed

Load the maps SDK on demand

666 KB of maps SDK loaded 4.7 seconds into every visit, with no map on screen and no prior interaction. There were two independent triggers; both had to be fixed.

SCOPE	METRICS	EFFORT	OWNER	STATUS
`/reserva/crear` address fields	TBT TTI	Low	Frontend	Fixed

666 KB of JavaScript requested at 4,716 ms with no prior interaction — 186 KB compressed, 84 ms of off-main-thread parsing and 20 ms of on-thread evaluation, on every visit, warm cache included.

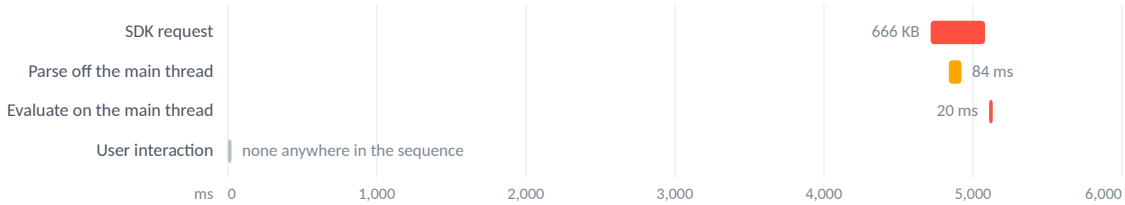
Strategic framing

ASK WHETHER THE RESOURCE IS NEEDED AT ALL FIRST

Before reaching for micro-optimisations — preconnect, preload, compression — it is worth asking whether the resource does anything in this view. Here it does not: no map renders on `/reserva/crear`; the SDK is only needed the moment someone focuses an address field and starts typing; and anyone who types their address by hand never needs it.

On-demand loading and the facade pattern are the correct first move for any heavy third party off the critical path. Preconnect is a secondary improvement, useful only *after* the eager loading is fixed: while the script loads unconditionally it contributes nothing at all.

Problem



Maps SDK load sequence. The last row is the point: there is no user interaction anywhere in the sequence. *Reconstructed from PERF-013-no-cache-trace.json.gz.*

The SDK is only needed when the user focuses an address field and types (autocomplete), when they select a suggestion (geocoding to extract the address components), or when they view the property map, which is a different route entirely and unrelated to `/reserva/crear`.

Root cause — two triggers, not one

Trigger 1, primary · constructor subscription

`libs/booking/feature-create/src/components/address-input/address-input.component.ts:58`

```
private readonly placesLibraryStatus =
  toSignal(this.mapsApiService.placesLibraryStatus$, {
    requireSync: true,   // ← synchronous subscription at construction
  });
```

`placesLibraryStatus$` derives from `api$`, which injects the SDK's `<script>` tag into the DOM:

```
Form renders
  → AddressInputComponent constructed
    → toSignal(placesLibraryStatus$, { requireSync: true })
      → placesLibraryStatus$ → api$
        → <script src="maps.geo-vendor.com/..."> injected
          → network request at 4,716 ms   ← confirmed in the trace
```

It is unconditional: it fires whether or not the user ever touches an address field.

Trigger 2, secondary · `preloadLib()` after view init

```
constructor() {
  effect(() => {
    if (!this.viewInitialized()) return;
    if (this.autocompleteEnabled()) {
      this.autocompleteApiService.preloadLib();  // ← fires on AfterViewInit
    }
  });
}
```

Because `api$` is memoised, the script is injected only once, so trigger 1 wins the request. But `preloadLib()` would cause it anyway if the first were removed. Both had to be fixed.

Fix applied

Fix 1 · Do not subscribe to SDK status in the constructor

The component used `placesLibraryStatus$` only to decide which interface to show: search field if the SDK loaded, manual field if it failed or was still loading. That pattern forces an SDK load purely to paint the interface.

Replace it with an optimistic value: show the search field by default, assume the SDK will be available, and fall back to the manual field only on a real failure. The loading skeleton is unnecessary: the user gains nothing from knowing the SDK is initialising before they touch the field.

```
// Before — triggers the SDK load at construction:
private readonly placesLibraryStatus =
  toSignal(this.mapsApiService.placesLibraryStatus$, { requireSync: true });

protected readonly showManualAddressInput = computed(
  () => this.placesLibraryStatus().failed
     || this.overrideShowManualAddressInput()
     || this.hasSomeAddressFieldFilledIn(),
);

// After — nothing loads until there is interaction.
private readonly mapsLoadFailed = signal(false);

protected readonly showManualAddressInput = computed(
  () => this.mapsLoadFailed()
     || this.overrideShowManualAddressInput()
     || this.hasSomeAddressFieldFilledIn(),
);
protected readonly showSearchAddressInput = computed(
  () => !this.showManualAddressInput(),
);

// Called by AddressSearchInputComponent when the SDK fails:
onMapsLoadFailed(): void {
  this.mapsLoadFailed.set(true);
}
```

Fix 2 · Move `preloadLib()` to the focus event

```
// After: the SDK loads only when the user shows intent
private handleTextInputFocus(): Subscription {
  return fromEvent(this.textInput, 'focus').subscribe(() => {
    this.autocompleteApiService.preloadLib();    // ← on first focus
    if (this.currentPredictions().length > 0 || this.showLastAutocompleteOption()) {
      this.showDropdown();
    }
  });
}
```

`preloadLib()` already has an internal guard, so calling it on every focus is safe: the script is injected only once.

Optional improvement · preconnect on hover

With the eager load removed, a `preconnect` to the SDK domain on the address container's `mouseenter` gives the browser a head start on the handshake before the user reaches the field. This matters most where RTT is high.

Evidence

CAPTURE CONDITIONS

Authenticated production, Spain (EU), MacBook Pro M3 on fibre, no throttling. Full conditions in **Section D**.

File	Status	Notes
`PERF-013-no-cache.har · with-cache.har`	**Collected**	The SDK loads eagerly even with everything else cached
`PERF-013-*-trace.json.gz`	**Collected**	No `mousedown`, `click`, `focus` or `keydown` events before the request
`PERF-013-maps-waterfall.png`	**Collected**	—
`PERF-013-maps-flamechart.png`	**Collected**	—
`devtools-snippet-perf013.js`	**Collected**	Reusable snippet

Improvement achieved

Scenario	Before	After
User never touches an address field	666 KB downloaded, parsed and executed	**0 KB** — the SDK never loads
User focuses an address field	666 KB at 4,716 ms	666 KB on first focus, deferred
User in the remote market	Several seconds of download before parsing even begins	Fully deferred; with preconnect on hover, the RTT cost drops further still

The baseline is Spain (EU) on fibre: the best possible case. At 300–500 ms RTT and on modest devices, the "before" figures were 2 to 4× worse.

References

- → **V8 — The cost of JavaScript in 2019** (Addy Osmani) · the cost of parsing, compiling and executing beyond download size; the key data point for justifying lazy loading to stakeholders
- → **web.dev — Third-party JavaScript** · impact, diagnosis and mitigation: facades, lazy loading, Partytown
- → **web.dev — Facade pattern for third-party embeds** · replacing a heavy third party with a lightweight placeholder until interaction
- → **Maps SDK documentation — dynamic library import** · loading only the libraries you need, on demand
- → **MDN — focus event** · propagation, `focusin` as the delegable alternative, and differences from `click`

14 PERF-014 P2 · Strategic

Optimise the SVG icon sprite

The design system sprite carries 694 icons. The application uses 76. The other 618 are shipped to every user on every load and never render once.

SCOPE	METRICS	EFFORT	OWNER	STATUS
All routes the sprite is global	LCP TBT	Medium	Frontend + Design	Open

11% of the sprite is used: 76 icons out of 694. The file weighs 861 KB uncompressed, 292 KB over the wire, and starts downloading at 3,077 ms — squarely inside the LCP window.

Problem

`design-system-icons.svg` is the corporate design system's icon sprite and loads as a static asset on every page. It is referenced in the project's asset configuration:

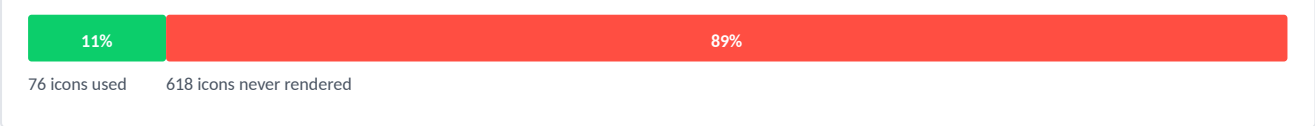
```
{
  "glob": "design-system-icons.svg",
  "input": "node_modules/@vantia/design-system",
  "output": "/assets"
}
```

This monolithic asset blocks the two-second render milestone and contributes to Total Blocking Time.

Core Web Vitals impact

Metric	Effect
LCP	Blocked — the sprite competes with critical resources right at the two-second mark.
TBT	Increased — downloading and parsing a large asset feeds long main-thread tasks.

Icon usage audit



Real usage share of the sprite. Of the 76 icons used across the whole application, only 26 appear on `/reserva/crear`. *Reconstructed from PERF-014-icon-audit.txt and PERF-014-used-icons.txt.*

Metric	Value
Total icons in the sprite	694
Icons used across the application	76
Icons visible on `/reserva/crear`	26
Usage ratio	11%
Wire size	292 KB compressed
Uncompressed size	861 KB
Download start	3,077 ms
Download duration	244 ms

Recommended fix

Step 1 · Audit usage — done

```
grep -r 'icon="[^"]*"' apps/now-web libs --include="*.html" | \
  grep -o 'icon="[^"]*"' | sort -u
```

Result: 76 unique icons across all templates, out of 694 available.

Step 2 · Generate a custom sprite — preferred

```
// scripts/build-icon-sprite.js — inside the build pipeline
const svgstore = require('svgstore');
const usedIcons = require('./used-icons.json');   // output of step 1

const sprite = svgstore();
usedIcons.forEach(iconId => {
  sprite.add(
    iconId,
    fs.readFileSync(`node_modules/@vantia/design-system/icons/${iconId}.svg`)
  );
});
fs.writeFileSync('dist/assets/now-icon-sprite.svg', sprite.toString());
```

Then swap the asset glob for the generated sprite.

Step 3 · Inline the icons visible on load

```
<!-- Instead of <vds-icon icon="status_check_small"> -->
<svg aria-hidden="true" width="16" height="16" focusable="false">
  <use href="#status_check_small"></use>
</svg>
```

Step 4 · Lazy load — fallback if a custom sprite is not viable

```
afterNextRender(() => {
  const link = document.createElement('link');
  link.rel = 'preload';
  link.as = 'image';
  link.type = 'image/svg+xml';
  link.href = '/assets/design-system-icons.svg';
  link.setAttribute('fetchpriority', 'low');
  document.head.appendChild(link);
});
```

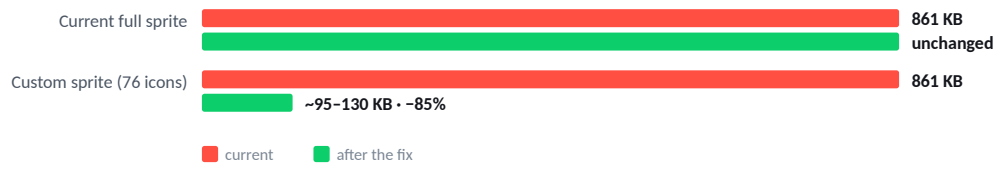
Evidence

CAPTURE CONDITIONS

Authenticated production, Spain (EU), MacBook Pro M3 on fibre, no throttling. Full conditions in **Section D**.

File	Status	Notes
`PERF-014-no-cache.har · with-cache.har`	Collected	Starts at 3,077 ms, 292 KB wire / 861 KB uncompressed
`PERF-014-*-trace.json.gz`	Collected	Download bar visible in the 3–3.3 s window
`PERF-014-sprite-waterfall.png`	Collected	The sprite alongside concurrent critical resources
`PERF-014-sprite-flamechart.png`	Collected	Wide download bar overlapping the LCP window
`PERF-014-icon-audit.txt · used-icons.txt`	Collected	Full audit and the 76 referenced icons

Expected improvement



Sprite weight before and after. A sprite containing only the 76 used icons is estimated at 95 to 130 KB compressed: around 85% smaller. *Estimate based on the measured usage ratio.*

On the Spain/fibre baseline the sprite starts at 3,077 ms and takes 244 ms, completing around 3,321. At 300–500 ms RTT, the download window for 861 KB is estimated 2 to 3× longer, adding two or three seconds for the priority market.

- **Custom sprite (steps 1–2):** removes the two-second render block on fibre, and pays off proportionally more on constrained connections.

- **Inline icons (step 3):** appear immediately, at zero network cost.
- **Lazy loading (step 4):** defers the 861 KB until the page is interactive.

References

→ **web.dev — LCP** · the sprite competes with critical path resources and delays LCP directly

→ **SVGO** · the most widely used SVG optimiser; shrinks each icon before the sprite is composed

→ **svgstore** · generating sprites by concatenating icons as `<symbol>` elements

→ **svg-sprite** · a more complete alternative: generates sprites in several modes (inline, CSS, view, defs)

→ **MDN — SVG `<use>` element** · `href` referencing, shadow DOM behaviour and cross-origin restrictions

SECTION E

Where this report is weak

Stated plainly, so nobody has to discover it in the meeting.

Issue	Gap
All	No capture has been taken from the Latin American market. Every figure for that market is an extrapolation from the European baseline.
PERF-003	The maps SDK Timing panel and the performance trace are missing.
PERF-004	The cold and warm waterfall screenshots are missing.
PERF-005	There is no US baseline, so the RTT delta is inferred rather than measured. The network capture with the TTFB column and the trace are also missing.
PERF-007	The tag manager timing detail and the zoomed assistant waterfall are missing. The personalisation tracker's scope is unmeasured.
PERF-009	Rests on a single HAR. No trace, no screenshots, and no breakdown of the 1–3 s into chunk time versus API time.
PERF-010	The baseline trace and the post-fix comparison are missing.
PERF-012	The text version of the TTFB summary is missing.

AN INCONSISTENCY WORTH RESOLVING

The source material carries two different figures for the EU→US round trip: roughly 120 ms in the per-issue evidence notes, and roughly 250 ms in the PERF-005 summary. Both have been preserved exactly as recorded. Confirm one and correct the other before using these numbers to build a business case.

What it would take to close the gaps

- **One capture from the priority market.** This is the highest-value contribution available. It turns fourteen extrapolations into fourteen measurements.
- **A US baseline.** It settles the argument about how much of the TTFB is distance and how much is origin processing.
- **Continuous monitoring.** This report is a snapshot. Without field data and regression tests, there will be no way to know in three months whether the fixes held.

ABOUT THIS DOCUMENT

Sample report produced by **PerfReviews**. This is a real audit, anonymised: the figures, findings and recommendations are the originals; the client, the sector and the third-party vendors are fictional. The charts are vector reconstructions of the capture data.

perf.reviews · mail@perf.reviews