PERFREVIEWS

Informe de auditoría de rendimiento web · Junio 2026

Motor de reservas NOW

Catorce incidencias de rendimiento documentadas sobre el flujo de reserva, con evidencia medida, causa raíz y soluciones priorizadas por impacto y esfuerzo.

14	1	23	6,2 s
incidencias documentadas	causa raíz que multiplica al resto	gráficos sobre datos medidos	de cascada en /reserva/crear

CLIENTE

Vantia Hotels & Resorts

ALCANCE

`apps/now-web` · `/reserva/crear`

ANALISTA

Joan León · PerfReviews

MERCADO PRIORITARIO

Latinoamérica · locale `/es-mx/`

Informe de muestra. Corresponde a una auditoría real, anonimizada. Las cifras, los hallazgos y las recomendaciones son los originales; el nombre del cliente, el sector y los proveedores de terceros se han ficcionado. Los gráficos son reconstrucciones vectoriales de los datos de captura, no capturas de DevTools: el detalle de cada fichero de evidencia se conserva en las tablas de cada capítulo.

ÍNDICE

Qué contiene este informe

PRELIMINARES

INCIDENCIAS — UN CAPÍTULO POR CADA UNA

CIERRE

SECCIÓN A

Cómo leer este informe

Cada incidencia es un capítulo autocontenido. Los preliminares recogen todo lo que si no habría que repetir catorce veces: el entorno de medición, los umbrales de Core Web Vitals y las convenciones de evidencia.

Anatomía de un capítulo

Todos los capítulos abren con el número, el título, un resumen del hallazgo en una línea y una franja de metadatos: prioridad, alcance, métricas afectadas, esfuerzo y equipo responsable. Justo debajo va la cifra medida más importante de esa incidencia. El cuerpo sigue siempre el mismo orden:

Sección	Qué contiene
Problema	Qué ocurre en la aplicación y por qué, con la ruta de código responsable cuando se ha identificado.
Impacto en Core Web Vitals	Qué métricas se ven afectadas y por qué mecanismo. Las definiciones están en la Sección C.
Impacto medido	Cifras tomadas de una captura real. Todo lo que no está medido se etiqueta como estimación.
Solución recomendada	Cambios concretos, ordenados por retorno sobre esfuerzo. El código es ilustrativo, no un parche listo para aplicar.
Evidencia	Qué ficheros de captura existen, cuáles faltan, y los gráficos derivados de ellos.
Mejora esperada	Ahorro proyectado por optimización, más el multiplicador del mercado remoto. Estimaciones salvo indicación contraria.
Referencias	Fuentes externas que respaldan el diagnóstico y la solución.

Convenciones

Prioridad

La prioridad combina impacto y esfuerzo, no gravedad a secas. Una incidencia de impacto muy alto que exige trabajo de infraestructura entre equipos solo adelanta a una victoria rápida cuando su efecto multiplicador lo justifica — que es exactamente el caso de una de las catorce.

Los colores siguen la misma rampa que los umbrales de Core Web Vitals: rojo, ámbar y verde. La etiqueta siempre acompaña al color, de modo que el documento se lee igual en blanco y negro o con una deficiencia de visión cromática.

Nivel	Significado	Incidencias
P0	Atacar primero — impacto muy alto, amplifica todo lo demás	PERF-005
P1	Victorias rápidas de alto impacto	PERF-001, PERF-003, PERF-004, PERF-006, PERF-007, PERF-011, PERF-013 Resuelta

Nivel	Significado	Incidencias
P2	Mejoras estratégicas	PERF-002, PERF-008, PERF-009, PERF-014
P3	Acabado e infraestructura	PERF-010, PERF-012

Etiquetas de métrica

Las métricas se agrupan por lo que describen:

`LCP` `FCP` `TTFB` carga · `TBT` `INP` `TTI` interactividad y respuesta · `CLS` estabilidad visual

Estado de la evidencia

Recopilada la captura existe en el directorio de evidencia. **Pendiente** falta por capturar; la afirmación que la rodea se apoya en la evidencia que sí está, y se señala como tal.

Los gráficos

Los gráficos de este informe son **reconstrucciones vectoriales de los datos de captura**, no capturas de pantalla de DevTools. Cada uno indica en su pie el fichero de evidencia del que sale. Ninguna cifra se ha redondeado ni suavizado para que el gráfico quede mejor.

Las cifras

Los valores medidos se dan tal cual. Las proyecciones usan la palabra *estimado* o un rango, y nombran siempre la línea base de la que extrapolan. Donde un valor es una cota inferior respecto al usuario real — que es prácticamente en todo el documento — el capítulo lo dice.

SECCIÓN B

Matriz de prioridades y triaje

Impacto frente a esfuerzo para las catorce incidencias. La zona sombreada es el cuadrante de alto impacto y bajo esfuerzo: el trabajo que conviene planificar primero.

IMPACTO ↑

	Bajo	Bajo-medio	Medio	Alto	Muy alto
Muy alto					PERF-005
Alto			PERF-007, PERF-011	PERF-001	
Medio-alto	PERF-013	PERF-006	PERF-009		
Medio	PERF-003, PERF-004		PERF-012, PERF-014		
Medio-bajo	PERF-008	PERF-010	PERF-002		

ESFUERZO →

P0 Atacar primero · P1 Victoria rápida · P2 Estratégica · P3 Acabado · Sombreado: alto impacto por poco esfuerzo

P0 — Atacar primero

ID	Título	Alcance	Métricas	Esfuerzo	Responsable
PERF-005	Backend alojado en EE. UU. y caché de CDN anulada por las cabeceras del origen	Todas las rutas	LCP, TTI, INP	Muy alto	Infra + Backend

MULTIPLICADOR DE TODO LO DEMÁS

Cada una de las otras incidencias de red de este informe queda amplificada por el trayecto de ida y vuelta hasta EE. UU., y otra vez por la distancia hasta el mercado latinoamericano. Corregir PERF-005 no elimina las trece restantes, pero cambia el tamaño de todas sus cifras.

P1 — Victorias rápidas de alto impacto

ID	Título	Alcance	Métricas	Esfuerzo	Responsable
PERF-013 Resuelta	Cargar el SDK de mapas bajo demanda	`/reserva/crear`	TBT TTI	Bajo	Frontend
PERF-006	Aplazar el JavaScript no crítico	Todas	FCP LCP TBT	Bajo-medio	Frontend + Plataforma
PERF-003	Preconectar con los dominios externos	Todas	LCP FCP	Bajo	Frontend + Plataforma
PERF-004	Precargar las fuentes	Todas	LCP CLS	Bajo	Frontend + Plataforma
PERF-011	Deduplicar las llamadas a la API	`/reserva/crear`	TTI INP	Medio	Frontend
PERF-007	Consolidar y aplazar los scripts de terceros	Todas	FCP LCP TBT INP	Medio	Frontend + Analítica
PERF-001	Mejorar el encadenamiento de peticiones del formulario	`/reserva/crear`	LCP TTI	Alto	Frontend + Backend

P2 — Mejoras estratégicas

ID	Título	Alcance	Métricas	Esfuerzo	Responsable
PERF-008	Precargar los chunks críticos de la aplicación	`/reserva/crear`	LCP TTI	Bajo	Frontend + Plataforma
PERF-009	Speculation Rules para la navegación de confirmación	`/crear` → `/confirmada`	LCP TTI (página siguiente)	Medio	Frontend
PERF-014	Optimizar el sprite SVG de iconos	Todas	LCP TBT	Medio	Frontend + Diseño
PERF-002	Cargar solo el CSS necesario	Todas	FCP LCP	Medio	Frontend

P3 — Acabado e infraestructura

ID	Título	Alcance	Métricas	Esfuerzo	Responsable
PERF-010	`content-visibility` en las secciones fuera de pantalla	`/reserva/crear`	TBT INP	Bajo-medio	Frontend
PERF-012	HTTP/2 103 Early Hints	Todas	FCP LCP	Medio	Infra + Plataforma

SECCIÓN C

Referencia de Core Web Vitals

Umbrales a los que se refiere cada capítulo. Las métricas de campo son las que experimenta el usuario; las de laboratorio son aproximaciones que se usan durante el diagnóstico.

Métrica	Nombre completo	Umbral bueno	Qué mide
LCP	Largest Contentful Paint	≤ 2,5 s	Carga — cuándo aparece el contenido principal
FCP	First Contentful Paint	≤ 1,8 s	Carga — cuándo aparece el primer píxel
TTFB	Time to First Byte	≤ 800 ms	Tiempo de respuesta de servidor y red
TBT	Total Blocking Time	≤ 200 ms	Aproximación a la interactividad — solo laboratorio
INP	Interaction to Next Paint	≤ 200 ms	Respuesta a todas las interacciones
TTI	Time to Interactive	≤ 3,8 s	Interactividad plena — obsoleta, se sigue midiendo aquí
CLS	Cumulative Layout Shift	≤ 0,1	Estabilidad visual

SECCIÓN D

Entorno de medición y convenciones de evidencia

Aplica a todas las capturas del informe salvo que un capítulo diga lo contrario. Leerlo una vez aquí ahorra leerlo catorce.

Condiciones de la línea base

Variable	Valor
Entorno	Producción, sesión autenticada
Ubicación del analista	España (UE) — unos 120 ms de RTT hasta el origen alojado en EE. UU.
Dispositivo	MacBook Pro M3
Conexión	Fibra simétrica ~600 Mbps, ~10 ms de latencia local
Limitación artificial	Ninguna, salvo en PERF-010, donde se aplica un factor 4× de CPU y se indica
Navegador	Chrome estable. Chrome Canary tiene un fallo en DevTools que oculta las llamadas reales y solo muestra los *preflight* `OPTIONS`: no sirve para estas capturas
Locale objetivo	`/es-mx/reserva/crear` — el mercado latinoamericano, el más afectado

ESTAS CIFRAS SON UNA COTA INFERIOR

Un MacBook sobre fibra europea es prácticamente el mejor caso que esta aplicación va a ver nunca. El público que importa — Latinoamérica, en dispositivos modestos y redes móviles — se mueve en RTT reales de 300 a 500 ms. Cada salto serie que quede se multiplica por un factor estimado de **2 a 4×** respecto a esta línea base. Léase cada valor medido como el suelo, no como el usuario típico.

Huecos conocidos de la medición

- **WebPageTest está bloqueado por el CDN**, así que no hay datos sintéticos de laboratorio de terceros para estas rutas.
- **El entorno de desarrollo no usa el bundle de producción**, de modo que los hallazgos sobre tamaño de bundle y scripts bloqueantes solo son reproducibles contra producción.
- **No existe todavía ninguna captura tomada desde Latinoamérica.** Todas las cifras de ese mercado son extrapolaciones desde la línea base europea, y así se etiquetan.

CÓMO CONTRIBUIR CON UNA CAPTURA DESDE EL MERCADO REMOTO

Una captura tomada desde México, Colombia o Chile es la aportación más valiosa que puede recibir este análisis. Se usa la misma URL `/es-mx/reserva/crear`, se sigue el `CAPTURE-GUIDE.md` del directorio de evidencia correspondiente y se dejan los ficheros junto a los existentes con sufijo `-mx` o `-latam` — por ejemplo `PERF-001-no-cache-mx.har`.

Estructura del directorio de evidencia

Cada incidencia guarda su evidencia en su propio subdirectorio. No todos los ficheros existen para todas: la tabla de evidencia de cada capítulo indica el conjunto realmente recopilado.

```
docs/performance/issues/evidence/
└── PERF-XXX/
    ├── CAPTURE-GUIDE.md                 instrucciones de captura paso a paso
    ├── PERF-XXX-no-cache.har            captura de red en frío
    ├── PERF-XXX-with-cache.har          captura de red en templado
    ├── PERF-XXX-no-cache-trace.json.gz  traza de rendimiento en frío
    ├── PERF-XXX-with-cache-trace.json.gz traza de rendimiento en templado
    ├── PERF-XXX-*.png                   capturas de cascada y flame chart
    └── PERF-XXX-recording.mp4           grabación (solo PERF-001, -002, -010)
```

UNA INCOHERENCIA QUE CONVIENE RESOLVER

En el material de partida aparecen dos cifras distintas para el trayecto UE→EE. UU.: unos 120 ms en las notas de evidencia de cada incidencia y unos 250 ms en el resumen de PERF-005. Se han conservado ambas tal como se registraron, sin reconciliarlas. Antes de usar estos números para construir un caso de negocio, conviene confirmar una y corregir la otra.

PARTE DOS

Las incidencias

Catorce capítulos, uno por incidencia, en orden de identificador. El capítulo *n* es `PERF-0nn`.

01 PERF-001 P1 · Victoria rápida

Mejorar el encadenamiento de peticiones del formulario de reserva

Treinta y dos peticiones de esquema en una sola carga, diecisiete de ellas redundantes, y todas esperando detrás de una llamada de arranque que por sí sola tarda 3,9 segundos.

ALCANCE	MÉTRICAS	ESFUERZO	RESPONSABLE	ESTADO
`/reserva/crear`	LCP TTI	Alto	Frontend + Backend	Abierta

6 173 ms de ventana entre la primera petición de esquema y la última respuesta, con caché templada, desde España hacia `/es-mx/reserva/crear`. Desde el inicio de la navegación hasta el último esquema: **~8 537 ms**.

Problema

El formulario de reserva carga su estructura mediante una cascada secuencial de peticiones. Cada una tiene que completarse antes de que arranque la siguiente:

1. Descarga del HTML → arranque de Angular → comprobación de autenticación
2. Autenticación resuelta → `GET /reservas/schemas/default`
3. Esquema de arranque resuelto → `POST /reservas/schemas/guest-details`
4. Datos del huésped resueltos → `POST /reservas/schemas/rate-details`
5. Tarifas resueltas → `POST /reservas/schemas/v2/extra-services`
6. …y así para cada sección del formulario

La cascada no es puramente serie: en cuanto `/schemas/default` responde, el resto de esquemas salen en oleadas paralelas. Pero todas las oleadas están detrás de esa primera respuesta, y esa primera respuesta tarda 3 871 ms.

Impacto en Core Web Vitals

Métrica	Efecto
LCP	Retrasada — el esqueleto del formulario, que es el candidato a Largest Contentful Paint, no puede renderizarse hasta que hay datos de esquema.
TTI	Muy degradada — las peticiones en serie mantienen ocupado el hilo principal y el formulario sin responder.

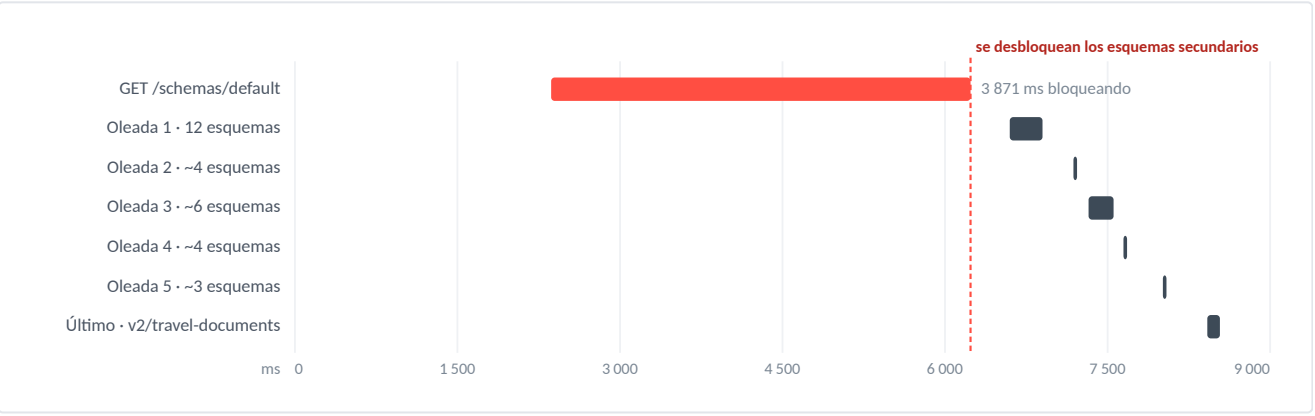
Cada ida y vuelta adicional se traslada directamente a LCP y TTI. En **PERF-005** está el motivo de que cada una de esas idas y vueltas cueste lo que cuesta.

Impacto medido

Capturado con `devtools-snippet-perf001.js` contra producción, caché templada, España (UE) → `/es-mx/reserva/crear`.

32	15	17	3 871 ms
peticiones de esquema	endpoints únicos	refetches redundantes	solo /schemas/default

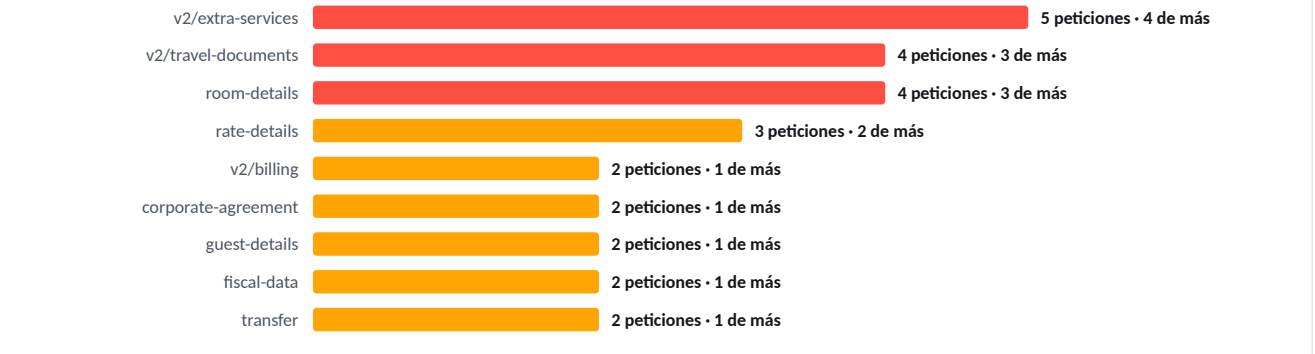
Métrica	Valor
Ventana total, primer → último esquema	6 173 ms
Suma de las duraciones individuales	10 911 ms
Solapamiento aprovechado por el paralelismo	4 738 ms
Inicio de navegación → último esquema	~8 537 ms



Estructura de la cascada. La barra roja es la puerta: mientras `/schemas/default` no responde, ninguna de las cinco oleadas siguientes puede empezar. Ocupa el 43 % de la ventana completa. *Reconstruido a partir de PERF-001-Highlight-schema-requests.json.*

Peticiones duplicadas

El snippet destapó un hallazgo que la cascada por sí sola no muestra: 15 endpoints únicos generan 32 peticiones HTTP dentro de una única carga de página.



Refetches por endpoint. Diecisiete de las treinta y dos peticiones vuelven a pedir un esquema ya cargado en la misma sesión de página. *Reconstruido a partir de PERF-001-Highlight-schema-requests.json.*

Solución recomendada

1 · Identificar los esquemas invariantes y pedirlos en paralelo

Los esquemas que siempre hacen falta con independencia del tipo de reserva — datos del huésped, habitación, tarifas — pueden pedirse junto al esquema de arranque en lugar de después.

```
// 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 de esquemas al pasar por encima del enlace

Calentar los esquemas cuando el usuario pasa el ratón sobre el enlace de «Nueva reserva», con una estrategia de prefetch propia o una llamada directa de `HttpClient`.

3 · Fusionar arranque y esquema por defecto

Evaluar si el backend puede devolver los esquemas más habituales dentro de `/reservas/schemas/default` y eliminar así las idas y vueltas separadas.

4 · Deduplicar

Diecisiete de las treinta y dos peticiones repiten un esquema ya cargado. Añadir `shareReplay(1)` por URL en cada servicio de esquema, o un `Map<url, Observable>` compartido a nivel de interceptor, para que cada endpoint único se pida como mucho una vez por carga. Ver **PERF-011**, que aborda el mismo tipo de problema a nivel de interacción.

Evidencia

CONDICIONES DE CAPTURA

Producción autenticada, España (UE) → `/es-mx/reserva/crear`, MacBook Pro M3 sobre fibra, sin limitación artificial. Condiciones completas en la **Sección D**. Los valores son una cota inferior para el mercado latinoamericano: cada salto serie que quede se multiplica por un factor estimado de 2 a 4× con RTT de 300–500 ms.

Fichero	Estado	Notas
`PERF-001-no-cache-waterfall.png`	Recopilada	Cascada completa con descarga de assets
`PERF-001-with-cache-waterfall.png`	Recopilada	Cascada solo de API, aislando la cadena de esquemas
`PERF-001-recording.mp4`	Recopilada	Grabación del retardo con caché templada
`PERF-001-Highlight-schema-requests.png`	Recopilada	Salida por consola del snippet

Fichero	Estado	Notas
`PERF-001-Highlight-schema-requests.json`	**Recopilada**	Salida estructurada — 32 peticiones, 6 173 ms
`PERF-001-no-cache.har · with-cache.har`	**Recopilada**	—
`PERF-001-*-trace.json.gz`	**Recopilada**	Trazas en frío y en templado

Mejora esperada

Sobre los datos medidos. El ahorro de las dos primeras filas se solapa: el objetivo combinado no es su suma.

Optimización	Ahorro estimado
Eliminar los 17 refetches redundantes	~400–800 ms
Paralelizar los esquemas secundarios con `default`	hasta ~3 871 ms
Fusionar `default` y esquemas comunes en el servidor	hasta ~3 871 ms
Objetivo combinado realista	**1 500–3 500 ms**

Para usuarios en Latinoamérica con RTT de 300–500 ms, cada salto serie restante se multiplica por un factor estimado de 2 a 4× respecto a la línea base europea.

Referencias

- → **web.dev — LCP** · definición de la métrica principal; entender qué retrasa LCP antes de optimizarla
- → **web.dev — Optimize LCP** · guía completa, con una sección dedicada a eliminar retardos de carga de recursos
- → **web.dev — Critical rendering path** · por qué cada salto de red adicional multiplica la latencia percibida
- → **High Performance Browser Networking — HTTP/2** (Ilya Grigorik) · capítulo canónico sobre multiplexado y bloqueo de cabecera de línea; explica por qué HTTP/2 no elimina las cascadas de nivel de aplicación
- → **RxJS — forkJoin** · ejecutar observables independientes en paralelo y esperar a todos
- → **Chrome DevTools — cascada de red** · medir tiempos y visualizar la cadena de dependencias

02 PERF-002 P2 · Estratégica

Cargar solo el CSS necesario

Una única hoja de 1,25 MB arrastra todas las secciones de la aplicación, y la hoja del widget de chat bloquea el render 108 ms estando sin usar al 99,3 %.

ALCANCE	MÉTRICAS	ESFUERZO	RESPONSABLE	ESTADO
Todas las rutas foco en `/reserva/crear`	FCP LCP	Medio	Frontend	Abierta

99,3 % de `assistant-launcher.css` no se usa en el render inicial — y aun así cuesta una ida y vuelta bloqueante de 108 ms en cada carga, incluso estando en caché.

Problema

Todos los estilos de la aplicación se empaquetan en un único punto de entrada `main.scss` que importa cada sección de forma incondicional:

Hoja de estilos	¿Crítica?	Motivo
`body.scss`	Crítica	Layout base
`card.scss`	Crítica	Contenedor de tarjeta
`compact-form-grid.scss`	Crítica	Rejilla del formulario
`form.scss`	Crítica	Controles de formulario
`navigation.scss`	Crítica	Barra de navegación
`skeleton.scss`	Crítica	Esqueleto de carga
`utils.scss · z-index.scss`	Crítica	Utilidades compartidas y apilado
`guided-tour.scss`	No crítica	Solo tras interacción
`voucher.scss`	No crítica	Solo después de confirmar
`table.scss`	No crítica	Vistas secundarias
`toast.scss`	Discutible	Rara vez necesaria en el primer pintado

El CSS de secciones que no se ven en el render inicial — facturación, servicios extra, documentación, voucher, tour guiado — se parsea y aplica igualmente, añadiendo recálculo de estilo innecesario. El sprite de iconos (861 KB descomprimidos, ver **PERF-014**) y los estilos de terceros también se cargan de forma global.

Impacto en Core Web Vitals

Métrica	Efecto
FCP	Retrasada — el CSS bloqueante debe parsearse por completo antes de que el navegador pinte nada.
LCP	Retrasada — el esqueleto del formulario no puede renderizarse hasta que se aplica el CSS.

El CSS bloquea el render por diseño. Cada kilobyte extra en la ruta crítica retrasa FCP directamente.

Impacto medido

Capturado con `devtools-snippet-perf002.js`, producción, caché templada, España (UE) → `/es-mx/reserva/crear`.

Cobertura de CSS. La pestaña Coverage revela cuatro ficheros, dos más de los que detecta el snippet: `design-system-core.css` entra por un `@import` dentro del bundle principal, lo que lo hace invisible al filtro de `<link>`. *Reconstruido a partir de PERF-002-coverage.png y PERF-002-CSS_Stylesheet_Audit.json.*

Posición de las hojas bloqueantes. La del launcher del asistente arranca a los 2 673 ms y cuesta 108 ms reales de red. *Reconstruido a partir de PERF-002-CSS_Stylesheet_Audit.json.*

LOS 108 MS SON UN COSTE REAL

Aunque el fichero está en caché, el navegador hace igualmente un `GET` condicional y recibe un `304 Not Modified`. El render queda bloqueado mientras espera esa respuesta. **Un fichero cacheado no es un fichero gratis.**

Solución recomendada

1 · Extraer el CSS crítico

Identificar los estilos necesarios para renderizar el esqueleto visible sin desplazar — rejilla, barra de navegación, contenedor de tarjeta, base de los campos — e incorporarlos en línea en un bloque `<style>` del `index.html` del shell, o usar `inlineStyleLanguage` de Angular con un extractor de CSS crítico.

2 · Cargar en diferido los estilos globales no críticos

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

3 · Auditar y eliminar utilidades globales sin uso

Pasar PurgeCSS, o `@angular-builders/custom-webpack` con una pasada de *tree-shaking* de CSS, contra el uso real en las plantillas.

4 · Mover los estilos específicos de ruta a los componentes

Facturación, documentación y datos de la reserva deberían llevar su SCSS a nivel de componente. Conviene verificar que no queda duplicación en el bundle global.

5 · Aplazar la hoja del asistente

Es de terceros, bloquea el render, se dispara a los 2 673 ms, cuesta 108 ms por carga incluso cacheada y no se usa al 99,3 % en el render inicial. Debe cargarse de forma asíncrona cuando el formulario ya es interactivo, o solo al abrir el chat.

```
private loadAssistantStyles(): void {
  const link = document.createElement('link');
  link.rel = 'stylesheet';
  // URL tomada del <link> que inyecta el script del asistente
  link.href = 'https://chat.assistant-vendor.com/...assistant-launcher.css';
  document.head.appendChild(link);
}
```

Evidencia

CONDICIONES DE CAPTURA

Producción autenticada, España (UE) → `/es-mx/reserva/crear` , MacBook Pro M3 sobre fibra, sin limitación. Condiciones completas en la **Sección D**.

Fichero	Estado	Notas
`PERF-002-coverage.png`	**Recopilada**	Pestaña Coverage con el uso real de CSS
`PERF-002-CSS_Stylesheet_Audit.png / .json`	**Recopilada**	Salida del snippet, en consola y estructurada
`PERF-002-recording.mp4`	**Recopilada**	Grabación del retardo de render
`PERF-002.har`	**Recopilada**	Cascada con la posición bloqueante y los tamaños
`PERF-002-trace.json.gz`	**Recopilada**	Traza con entradas `Parse Stylesheet` y `Recalculate Style`

Mejora esperada

Optimización	Ahorro estimado
Aplazar `assistant-launcher.css` elimina la ida y vuelta de 108 ms	~100–120 ms
Diferir los estilos globales no críticos	~50–150 ms de FCP
Eliminar o partir el `@import` de `design-system-core.css`	~100–300 ms de parseo
Objetivo combinado realista	**200–500 ms de FCP**

Para usuarios en Latinoamérica con RTT de 300–500 ms, solo la ida y vuelta de `assistant-launcher.css` puede costar entre 300 y 500 ms si falla la caché.

Referencias

- → **web.dev — FCP** · el CSS bloquea el render por defecto y retrasa FCP directamente
- → **web.dev — Eliminate render-blocking resources** · cómo bloquea el CSS y qué estrategias reducen su impacto
- → **web.dev — Defer non-critical CSS** · cargar CSS no crítico sin bloquear la ruta de render
- → **web.dev — Extract critical CSS** · incorporar en línea el CSS visible para eliminar la ida y vuelta
- → **Chrome DevTools — pestaña Coverage** · medir el porcentaje de CSS y JS sin usar por carga
- → **PurgeCSS** · *tree-shaking* de CSS según el uso real en plantillas
- → **Angular — Component styles** · estilos con ámbito de componente para evitar CSS global innecesario

03 PERF-003 P1 · Victoria rápida

Preconectar con los dominios externos

Durante la carga se contacta con cuatro orígenes externos y ninguno tiene una pista de conexión. El navegador paga DNS, TCP y TLS completos justo en el momento en que menos se lo puede permitir.

ALCANCE	MÉTRICAS	ESFUERZO	RESPONSABLE	ESTADO
Todas las rutas	FCP LCP	Bajo	Frontend + Plataforma	Abierta

0 pistas de `preconnect` en el documento. La auditoría de *resource hints* encontró 2 `preload` y 1 `prefetch`, los tres inyectados por terceros — y ni un solo `preconnect` ni `dns-prefetch`.

Problema

La página pide recursos a varios orígenes externos sin ninguna pista previa de conexión. El navegador descubre esos orígenes solo después de parsear HTML y CSS o de ejecutar JavaScript, y en ese momento tiene que pagar el coste completo de conexión antes de recibir un solo byte de contenido.

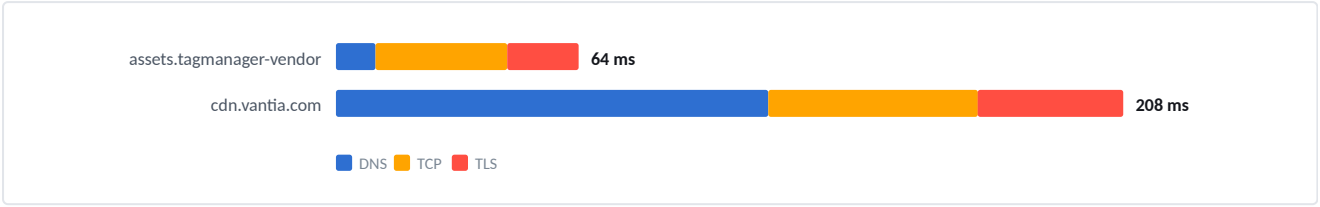
Origen	Función	Se descubre en
`maps.geo-vendor.com`	SDK de mapas y autocompletado	~3,5 s — al ejecutar JS
`tiles.geo-vendor.com`	Teselas y assets del mapa	Después de cargar el SDK
`assets.tagmanager-vendor.com`	Gestor de etiquetas	Parseo del HTML
`cdn.vantia.com`	Assets corporativos: fuentes, cabecera y pie	Parseo del HTML
`chat.assistant-vendor.com`	Asistente virtual	Carga de página

La resolución DNS más el *handshake* TCP más la negociación TLS cuestan entre 100 y 300 ms por origen desde España, y se pagan íntegros durante la carga.

Impacto en Core Web Vitals

Métrica	Efecto
FCP	Retrasada — scripts y estilos de orígenes externos bloquean o demoran el render.
LCP	Retrasada — los recursos externos descubiertos tarde alargan la ruta crítica.

Impacto medido



Coste de conexión por origen. Solo `cdn.vantia.com` gasta 208 ms en el *handshake* antes siquiera de enviar la petición. Es tiempo que ningún código de aplicación puede recuperar. *Reconstruido a partir de PERF-003-tagmanager-timing.png y PERF-003-cdn-timing.png.*

Pistas de recursos presentes en el documento. Las únicas tres que existen las inyectan scripts de terceros. La aplicación no aporta ninguna. *Reconstruido a partir de PERF-003-resource-hints-audit.png.*

Solución recomendada

1 · Pistas estáticas en el `<head>` del shell

```
<!-- Preconexiones críticas: abrir la conexión antes de que dispare el JS -->
<link rel="preconnect" href="https://assets.tagmanager-vendor.com">
<link rel="preconnect" href="https://cdn.vantia.com" crossorigin>
```

2 · Pista programática para el SDK de mapas, solo con intención

El mapa no debe preconectarse al inicializar: en **PERF-013** el problema es precisamente su carga anticipada. La pista se añade cuando el usuario muestra intención pasando el ratón por el campo de dirección.

```
// 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);
    }
  });
}
```

Se llama en el `mouseenter` del campo de dirección: para cuando se añade la etiqueta del script, la conexión ya está caliente.

Evidencia

CONDICIONES DE CAPTURA

Producción autenticada, España (UE), MacBook Pro M3 sobre fibra, sin limitación. Condiciones completas en la **Sección D**. Con RTT de 300–500 ms, el coste por origen es de 2 a 4× mayor.

Fichero	Estado	Notas
`PERF-003-cold-waterfall.png`	**Recopilada**	Cascada en frío con las columnas de DNS y conexión
`PERF-003-cold.har`	**Recopilada**	HAR en frío con el detalle por origen
`PERF-003-tagmanager-timing.png`	**Recopilada**	Panel Timing del gestor de etiquetas
`PERF-003-cdn-timing.png`	**Recopilada**	Panel Timing de `cdn.vantia.com`
`PERF-003-resource-hints-audit.png`	**Recopilada**	Confirmación de `preconnect: 0`
`PERF-003-maps-timing.png`	**Pendiente**	Panel Timing del SDK de mapas (opcional)
`PERF-003-trace.json`	**Pendiente**	Traza con el coste DNS/TCP/TLS en la línea temporal

Mejora esperada

Cada *handshake* en frío que se elimina ahorra entre 100 y 300 ms sobre fibra desde España, con unos 120 ms de RTT hasta orígenes alojados en EE. UU. Con tres o cuatro orígenes externos, un ahorro total de **300 a 900 ms** en la primera carga es realista.

Para el mercado latinoamericano con RTT de 300–500 ms, el coste por origen es de 2 a 4× superior, así que la mejora crece en proporción: entre 600 ms y 1,6 s por carga en frío.

Referencias

- → **web.dev — FCP** · la métrica más afectada por la latencia de conexión externa
- → **Ilya Grigorik — Eliminating Roundtrips with Preconnect** · artículo de referencia del autor de *High Performance Browser Networking*; explica las capas DNS, TCP y TLS y cómo `preconnect` elimina cada una
- → **web.dev — Preconnect to required origins** · cuándo usar `preconnect` en lugar de `dns-prefetch`
- → **MDN — Link types: preconnect** · especificación del atributo
- → **Chrome DevTools — Network request timing** · medir DNS Lookup, Initial Connection y SSL por petición
- → **web.dev — Resource hints** · guía completa de `preconnect`, `dns-prefetch`, `preload`, `modulepreload` y `prefetch`

04 PERF-004 P1 · Victoria rápida

Precargar las fuentes

La primera fuente no se descubre hasta 4 713 ms después de la navegación, porque el navegador tiene que descargar y parsear el bundle de CSS entero antes de enterarse de que existe.

ALCANCE	MÉTRICAS	ESFUERZO	RESPONSABLE	ESTADO
Todas las rutas las fuentes son globales	LCP CLS	Bajo	Frontend + Plataforma	Abierta

4 713 ms hasta que arranca la primera petición de fuente. El propio snippet lo dice sin rodeos: con un `<link rel="preload" as="font">` correcto esto debería ser 0 ms. La ventana completa de carga de fuentes abarca 3 542 ms.

Problema

Las fuentes corporativas se cargan como *web fonts* referenciadas por reglas `@font-face` en el CSS. El navegador no puede descubrir la URL de una fuente hasta que ha:

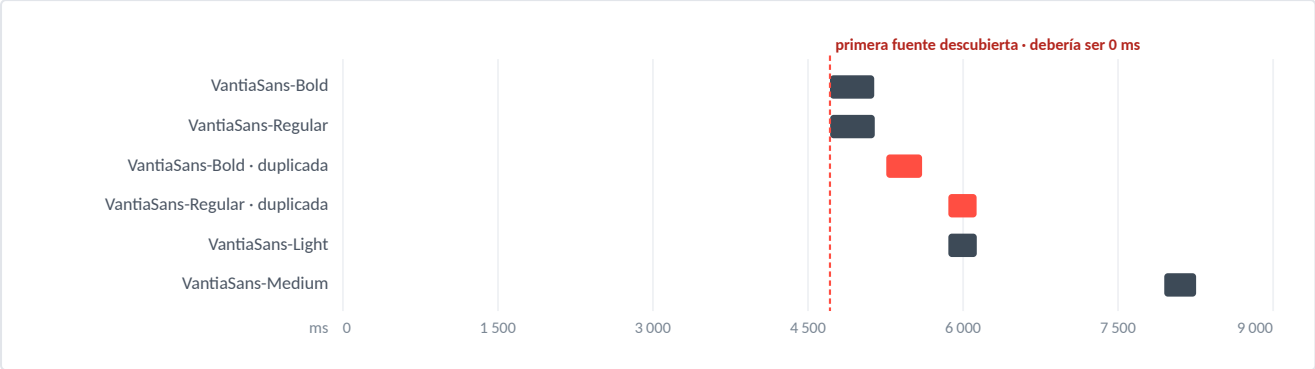
1. Descargado y parseado el bundle completo de CSS
2. Encontrado un elemento que necesita esa fuente
3. Emparejado la regla `@font-face`

Esta cascada de tres pasos produce entre 300 y 600 ms de texto invisible o con fuente de respaldo antes de que se pinte la real. FOUT y FOIT son dos síntomas visibles del mismo retardo de descubrimiento.

Impacto en Core Web Vitals

Métrica	Efecto
LCP	Retrasada — el elemento LCP, normalmente un titular o una etiqueta del formulario, no puede pintarse con la fuente correcta hasta que el fichero llega.
CLS	En riesgo — el intercambio de la fuente de respaldo por la definitiva desplaza el layout si las métricas tipográficas difieren.

Impacto medido



Línea temporal de carga de fuentes. Seis peticiones. Las dos barras rojas son duplicados: `Bold` y `Regular` se piden dos veces cada una.
Reconstruido a partir de PERF-004-no-cache-font-duplicates.png y la salida del snippet.

4 713 *ms*	**8 255 *ms***	**3 542 *ms***	**2**
primera fuente descubierta	todas las fuentes listas	ventana de carga	fuentes pedidas dos veces

SEGUNDO HALLAZGO — PETICIONES DUPLICADAS

`VantiaSans-Bold.woff2` y `VantiaSans-Regular.woff2` se piden dos veces cada una. Cada duplicado gasta ancho de banda y añade carga a la ruta crítica. Conviene investigarlo junto con el trabajo de *preload*: precargar una fuente que después se pide dos veces no arregla la doble referencia de fondo.

Solución recomendada

1 · Precargar las variantes críticas en el `<head>`

```
<!-- Solo las fuentes de la ruta crítica de render -->
<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>
```

Solo las que se usan sobre la línea de flotación. Precargar todos los pesos y estilos desperdicia ancho de banda y compite con recursos más críticos.

2 · Añadir `font-display: swap` a cada regla

```
@font-face {
  font-family: 'VantiaSans';
  src: url('../fonts/VantiaSans-Regular.woff2') format('woff2');
  font-display: swap; // el texto es visible durante la carga
}
```

3 · Resolver los duplicados

Rastrear qué dos declaraciones referencian los mismos ficheros `Bold` y `Regular`, y consolidarlas.

Evidencia

CONDICIONES DE CAPTURA

Producción autenticada, España (UE), MacBook Pro M3 sobre fibra, sin limitación. Condiciones completas en la **Sección D**.

Fichero	Estado	Notas
`PERF-004-no-cache.har · with-cache.har`	**Recopilada**	En templado las fuentes vienen de disco, pero el retardo de descubrimiento persiste
`PERF-004-*-trace.json.gz`	**Recopilada**	Confirman que el hueco no es un problema de caché
`PERF-004-no-cache-font-detail.png`	**Recopilada**	Detalle de la petición: «Initiated by `main-[hash].css`»
`PERF-004-no-cache-font-duplicates.png`	**Recopilada**	Filtro de fuentes: `Bold` y `Regular` ×2
`PERF-004-*-waterfall.png`	**Pendiente**	Cascadas en frío y templado

Mejora esperada

Precargar dos o tres ficheros críticos elimina la cascada de descubrimiento y puede mejorar LCP entre **200 y 400 ms** en la primera visita sin caché, medido desde la línea base España/fibra. Las visitas siguientes no se ven afectadas, porque las fuentes ya vienen de caché.

Con RTT de 300–500 ms el retardo de descubrimiento es proporcionalmente mayor, lo que sitúa la mejora estimada entre 2 y 4× por encima: unos 400 a 800 ms por carga en frío.

Referencias

- → **web.dev — LCP** · el candidato a LCP suele ser texto que depende de la fuente personalizada
- → **Zach Leatherman — Comprehensive Guide to Web Font Loading Strategies** · la referencia definitiva del ecosistema; más de once estrategias con sus compromisos, FOUT, FOIT y el patrón Critical FOFT
- → **web.dev — Web font best practices** · `preload`, `font-display`, autoalojamiento frente a CDN y su impacto en Core Web Vitals
- → **web.dev / MDN — font-display** · todos los valores del descriptor y cuándo usar cada uno
- → **CSS Font Loading API** · detectar cuándo una fuente está lista sin sondeo ni dependencia del CSS

05 PERF-005 P0 · Atacar primero

TTFB alto y caché de CDN desactivada por el origen

El CDN está delante del backend y no cachea nada, porque el origen envía `no-store` en cada respuesta. Cincuenta y cinco llamadas por sesión, 342 ms de media, todas hasta el origen.

ALCANCE	MÉTRICAS	ESFUERZO	RESPONSABLE	ESTADO
Todas las rutas con API	LCP TTI INP	Muy alto	Infra + Backend	Abierta

18,8 s de tiempo de red acumulado en una única sesión: 55 llamadas a la API, todas con fallo de caché en el CDN. Servidas desde el *edge* deberían sumar entre 0,5 y 1 s.

LA CAUSA RAÍZ QUE MULTIPLICA

Esta es la incidencia que encarece a todas las demás. Incluso después de que PERF-001 elimine el encadenamiento en serie, cada llamada individual seguirá arrastrando entre 235 y 342 ms de sobrecoste. Arreglar el encadenamiento reduce el número de idas y vueltas; arreglar esto reduce el precio de cada una de las que queden.

Problema

El backend de reservas está desplegado en la nube corporativa, en una región de EE. UU., detrás del CDN corporativo. Pero el CDN funciona de hecho como un simple paso a través: cada petición es un fallo de caché, porque el origen envía `Cache-Control: no-cache, no-store, max-age=0` en todas sus respuestas. Todas las peticiones llegan a un origen que no está cerca de los usuarios.

La cabecera `server-timing` de cada respuesta de esquema lo declara sin ambigüedad:

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

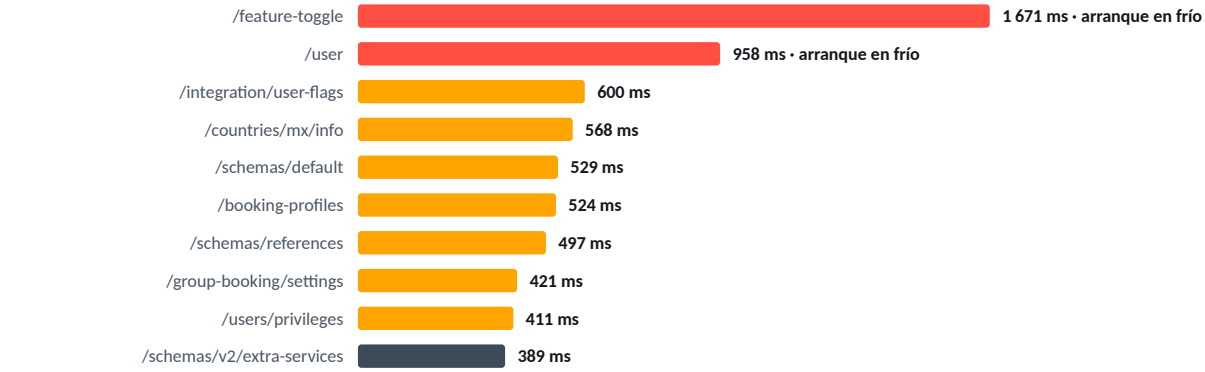
Mediciones confirmadas

Dos capturas, Chrome estable, España, 1 de junio de 2026.

Métrica	HAR templado, 29 llamadas	Snippet, sesión completa, 55	Servido desde el *edge*
Estado de caché en CDN	MISS en todas	—	HIT
Duración, peticiones templadas	210–345 ms media ~235 ms	173–957 ms media ~342 ms	~10–20 ms
Arranque en frío · `schemas/default`	4 328 ms de TTFB	—	~10–20 ms

Métrica	HAR templado, 29 llamadas	Snippet, sesión completa, 55	Servido desde el *edge*
Arranque en frío · `/feature-toggle`	—	1 671 ms	~10–20 ms
Arranque en frío · `/user`	—	958 ms	~10–20 ms
Proceso en el *edge*, templado	103–136 ms	—	—
Proceso en el origen, templado	70–90 ms	—	—
Llamadas totales	29	55	—
Duración acumulada	—	18,8 s	~0,5–1 s
Sobrecoste frente a un asset estático del CDN	**—**	**~211 ms por llamada**	**~0 ms**

El HAR se capturó en una única navegación a /reserva/crear. El snippet se ejecutó sobre una sesión completa con interacciones de formulario, que es lo que destapa el volumen real de llamadas y los arranques en frío más allá de schemas/default.



Las diez llamadas más lentas de la sesión. Las tres primeras son arranques en frío. La media de las 55 llamadas es de 342 ms; el máximo, 1 671 ms. *Reconstruido a partir de PERF-005-TTFB-01.png y -02.png.*

Dónde se va el tiempo de espera. En una petición templada el origen ya aporta 77 ms sobre 353. En el arranque en frío, 691 ms de 989 son puro tiempo de origen. *Reconstruido a partir de PERF-005-cdn-cache-miss-headers.png y PERF-005-cold-start-timing.png.*

Causas raíz

1 · Las cabeceras del propio origen desactivan la caché

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

`no-store` le indica al CDN que no cachee nunca la respuesta. La infraestructura de CDN ya está desplegada y pagada: lo que impide que sea efectiva son las cabeceras.

2 · Penalización de arranque en frío en `schemas/default`

La primera petición de esquema de cada sesión tarda 4 328 ms de TTFB, de los cuales 3 667 ms se gastan en el origen. Las siguientes bajan a unos 75 ms. Esa firma es coherente con una inicialización perezosa: pool de conexiones, calentamiento de la máquina virtual o un cálculo bajo demanda.

3 · El origen no está cerca de los usuarios

Incluso en templado, 210–345 ms de TTFB indican distancia geográfica real entre el *edge* y el origen. Con 29 llamadas en la carga del formulario, eso se acumula deprisa.

> **NOTA SOBRE UN HUECO PREVIO DE INVESTIGACIÓN**
>
> Esta incidencia estuvo aparcada como «pendiente de investigar» porque DevTools de Chrome Canary solo registraba las peticiones *preflight* `OPTIONS` y ocultaba las llamadas reales. Chrome estable las captura todas. Todos los HAR adjuntos se tomaron con Chrome estable.

Solución recomendada

Opciones por retorno sobre esfuerzo.

Opción 1 · Quitar `no-store` de las respuestas de esquema

Máximo retorno, mínimo esfuerzo. La infraestructura ya está; el único bloqueo es el `Cache-Control` del origen.

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

O bien, para que cachee el CDN pero no el navegador:

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

Las respuestas de esquema son en gran medida deterministas por cuenta y rol. Un TTL corto con una clave de caché que incluya el identificador de cuenta sería seguro y efectivo, y llevaría el TTFB de ~235 ms a ~10–20 ms en las respuestas cacheadas.

Opción 2 · Corregir el arranque en frío

Los 3,67 s de origen en la primera llamada deben investigarse desde backend. Causas probables: inicialización perezosa del servicio, expiración del pool de conexiones por inactividad o un primer cálculo costoso. Soluciones: conexiones *keep-alive*, inicialización anticipada en el despliegue o un *ping* de calentamiento programado.

Opción 3 · Gateway regional

Desplegar una pasarela ligera cerca del mercado que termine TLS regionalmente y haga de proxy contra el origen. Reduce el tramo transatlántico con independencia de la política de caché.

Opción 4 · Despliegue regional completo

Desplegar el backend en una región próxima al mercado. Máxima reducción de latencia; requiere coordinación con backend, infraestructura y los equipos de residencia de datos y cumplimiento.

Opción 5 · Agrupar peticiones en el *edge*

Usar funciones en el *edge* del CDN para agrupar varias peticiones de esquema en una sola ida y vuelta al origen, reduciendo el número de trayectos aunque el origen siga donde está.

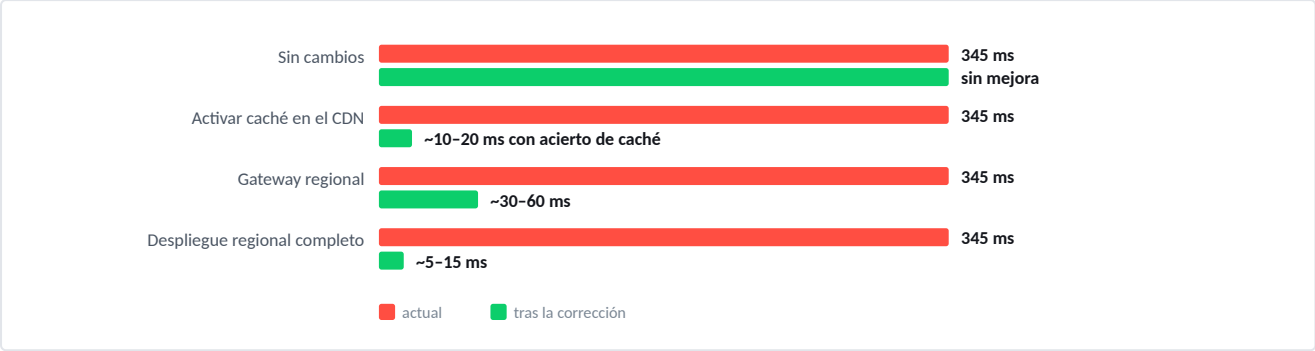
Evidencia

CONDICIONES DE CAPTURA

Producción autenticada, España (UE), MacBook Pro M3 sobre fibra, sin limitación, **solo con Chrome estable**. Condiciones completas en la **Sección D**.

Fichero	Estado	Notas
`PERF-005-warm-cache.har`	Recopilada	29 llamadas, `cdn-cache: MISS`, TTFB 210–345 ms
`PERF-005-with-cache.har`	Recopilada	Segunda captura templada; confirma el patrón entre sesiones
`PERF-005-no-cache.har`	Recopilada	En frío; TTFB de `schemas/default` ~4 328 ms
`PERF-005-cdn-cache-miss-headers.png`	Recopilada	`cdn-cache; desc=MISS` junto a `Cache-Control: no-store`
`PERF-005-cold-start-timing.png`	Recopilada	Panel Timing con la barra de TTFB en ~4 300 ms
`PERF-005-TTFB-01.png · -02.png`	Recopilada	Salida del snippet: TTFB por endpoint
`PERF-005-network-ttfb.png`	Pendiente	Panel de red con la columna TTFB legible
`PERF-005-us-baseline.har`	Pendiente	Línea base desde EE. UU. para medir el delta de RTT
`PERF-005-trace.json`	Pendiente	Traza con el TTFB por llamada y el pico de arranque en frío

Mejora esperada



TTFB por llamada según la opción elegida. Activar la caché del CDN es la corrección de mayor retorno y menor esfuerzo de todo el informe: la infraestructura ya existe y solo hay que cambiar las cabeceras. *Proyecciones sobre la línea base medida de 210–345 ms.*

Corrección	TTFB por llamada	Arranque en frío
Línea base actual España/fibra	210–345 ms	4 328 ms
Activar caché en el CDN	~10–20 ms con acierto	eliminado en los endpoints cacheados
Solo corregir el arranque en frío	210–345 ms	~75 ms
Gateway regional	~30–60 ms	~30–60 ms
Despliegue regional completo	~5–15 ms	~5–15 ms

Las cifras base reflejan condiciones de España sobre fibra. Para el mercado latinoamericano con RTT de 300–500 ms, el TTFB templado se estima entre 500 y 900 ms por llamada, lo que empeora bastante el coste acumulado de sesión (55 llamadas × ~700 ms de media) y hace la corrección de caché proporcionalmente más valiosa.

Referencias

- **web.dev — Optimize TTFB** · estrategia de CDN, caché de respuestas y Early Hints
- **web.dev — LCP** · cada milisegundo de TTFB retrasa LCP en la misma proporción
- **web.dev — INP** · las interacciones que disparan llamadas en tiempo real arrastran también la latencia geográfica
- **Cabecera HTTP** `server-timing` · desglose de CDN y origen visible directamente en el panel de red
- **High Performance Browser Networking — HTTP/2** · por qué la latencia domina incluso con multiplexado
- **Documentación de tu CDN sobre caché de respuestas HTTP** · cómo respeta y sobrescribe las cabeceras `Cache-Control`

06 PERF-006 P1 · Victoria rápida

Aplazar el JavaScript no crítico

Tres scripts del `<head>` del shell no llevan `defer` ni `async`. Ninguno hace falta para pintar el esqueleto del formulario, y los tres detienen el parseo del HTML a los 463 ms.

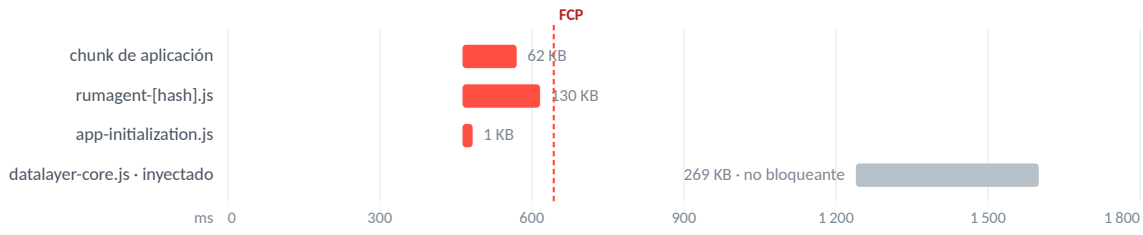
ALCANCE	MÉTRICAS	ESFUERZO	RESPONSABLE	ESTADO
Todas las rutas shell de plataforma	FCP LCP TBT	Bajo-medio	Frontend + Plataforma	Abierta

193 KB de JavaScript bloqueante — unos 378 KB descomprimidos — repartidos en tres scripts estáticos del `<head>`, los tres arrancando a los 463 ms y bloqueando el FCP de los 643 ms.

Problema

Varios ficheros JavaScript de tamaño considerable se cargan de forma síncrona en el `<head>`, bloqueando el parseo y el render.

Script	Tamaño en red	Comportamiento
`chunk de aplicación [hash]`	62 KB 185 KB descomprimidos	Bloquea el render — estático, sin `defer` / `async`
`rumagent-[hash].js agente de RUM`	130 KB	Bloquea el render — estático, sin `defer` / `async`
`app-initialization.production-[hash].js`	1 KB	Bloquea el render — estático, sin `defer` / `async`



Scripts bloqueantes frente al FCP. Los tres arrancan simultáneamente a los 463 ms; el FCP no llega hasta los 643. La barra gris es el *data layer*, que el navegador no clasifica como bloqueante pero que igualmente ocupa el hilo principal. *Reconstruido a partir de PERF-006-network-blocking.png y PERF-006-trace-fcp-blocked.png.*

LOS SCRIPTS INYECTADOS SON UN PROBLEMA APARTE

`datalayer-core.js` (~900 KB) y el script de cabecera y pie (~904 KB) los inyecta Angular en tiempo de ejecución. El navegador les asigna prioridad de red baja y no los clasifica como bloqueantes: `renderBlockingStatus` es `"non-blocking"`. Aun así se ejecutan en el hilo principal durante el arranque y contribuyen al TBT, así que retrasar su inyección a tiempo ocioso (corrección 2) sigue aplicando.

Impacto en Core Web Vitals

Métrica	Efecto
FCP	Muy bloqueada — el parseo del HTML se detiene en cada `<script>` síncrono hasta que se descarga, parsea y ejecuta.
LCP	Retrasada — el navegador no puede pintar hasta que terminan todos los scripts bloqueantes.
TBT	Alto — los scripts grandes generan tareas largas en el hilo principal que bloquean la interactividad.

Impacto medido

Métrica	Medido	Condiciones
JS bloqueante transferido	193 KB	Caché en frío, España, fibra, sin limitación
JS bloqueante descomprimido	~378 KB	Caché en frío, España, fibra, sin limitación
Scripts bloqueantes en `<head>`	3	Estáticos, sin `defer` / `async`
Inicio de descarga	~463 ms	Caché en frío
FCP	~643 ms	Caché en frío
Inicio de inyección del *data layer*	~1 239 ms	Caché en frío

Solución recomendada

1 · Añadir `defer` a todos los scripts estáticos del `<head>`

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

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

2 · Retrasar la inyección del *data layer*

```
// 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)
    );
    // lógica de inyección existente...
  }
}
```

3 · Cuándo `async` y cuándo `defer`

Atributo	Comportamiento	Para qué
`async`	Descarga en paralelo y ejecuta en cuanto está listo; puede interrumpir el parseo	Scripts totalmente independientes, como analítica
`defer`	Descarga en paralelo y ejecuta al terminar el parseo, en orden de documento	Scripts que dependen del DOM, como el de cabecera y pie

Evidencia

CONDICIONES DE CAPTURA

Producción autenticada, España (UE), MacBook Pro M3 sobre fibra, sin limitación. Condiciones completas en la **Sección D**.

Fichero	Estado	Notas
`PERF-006-no-cache.har · with-cache.har`	Recopilada	En templado el parseo y la evaluación siguen retrasando el render
`PERF-006-*-trace.json.gz`	Recopilada	Bloques «Evaluate Script» antes del marcador de FCP
`PERF-006-network-blocking.png`	Recopilada	Los tres scripts bloqueantes con prioridad alta
`PERF-006-trace-fcp-blocked.png`	Recopilada	FCP retrasado por tareas de evaluación
`PERF-006-coverage.png`	Recopilada	Porcentaje sin usar de cada script bloqueante

Mejora esperada

Añadir `defer` a los tres scripts estáticos elimina la fase de bloqueo del parseo a los 463 ms. Sobre fibra de escritorio en Europa el FCP ya está en ~643 ms, así que el margen en condiciones óptimas es modesto — pero el beneficio se multiplica en conexiones lentas y con RTT de 300–500 ms.

Retrasar la inyección del *data layer* (269 KB, arrancando a los 1 239 ms) a tiempo ocioso reduce además el TBT.

Referencias

- → **web.dev — FCP** · cada script bloqueante retrasa el primer píxel en pantalla
- → **V8 — The cost of JavaScript in 2019** (Addy Osmani) · referencia imprescindible del equipo de V8: parseo, compilación y ejecución son tres fases caras distintas; 300 KB de JS no equivalen a 300 KB de imagen
- → **web.dev — Eliminate render-blocking resources** · identificar qué bloquea el FCP y sacarlo de la ruta crítica
- → **MDN — `<script>` defer / async** · garantías de orden de ejecución y momento respecto al parseo
- → **MDN — requestIdleCallback** · planificar trabajo no crítico en los periodos ociosos del navegador

07 PERF-007 P1 · Victoria rápida

Consolidar y aplazar los scripts de terceros

Analítica y seguimiento son la mayor fuente evitable de coste de red y CPU de la aplicación — y el widget de chat sigue haciendo *polling* durante ocho minutos para usuarios que nunca lo abren.

ALCANCE	MÉTRICAS	ESFUERZO	RESPONSABLE	ESTADO
Todas las rutas	FCP LCP TBT INP	Medio	Frontend + Analítica	Abierta

5,35 MB descomprimidos — 828 KB transferidos — de código de terceros en una carga en frío, sin contar el tracker de personalización ni las respuestas de los *beacons*. En más de 28 peticiones.

Problema

Gestor de etiquetas

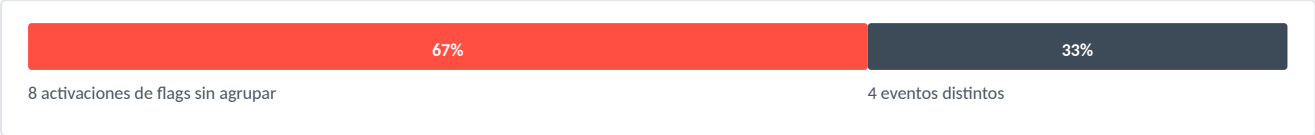
- 466 KB comprimidos / 2,25 MB descomprimidos en 13 peticiones: un bundle principal y doce ficheros de extensiones
- Encolado a los 4,06 s del ciclo de vida de la página — lo inyecta el shell dinámicamente, no es una etiqueta `<script>` descubierta por el parser
- Coste de conexión en frío al CDN del proveedor: DNS 10 ms + TCP 33 ms + SSL 18 ms = 61 ms de *handshake* antes de transferir un solo byte
- `EvaluateScript` del bundle principal: una tarea larga de 51,3 ms en el hilo principal
- No hace falta hasta que la página es interactiva

Asistente virtual (widget de chat)

- 362 KB comprimidos / 3,1 MB descomprimidos en más de 15 peticiones en la primera carga, sin contar llamadas a la API
- `chat-launch.js` y `chat-loader.min.js` se cargan **dos veces cada uno**: una desde el contexto de la página y otra desde un *iframe* parseado con prioridad alta. Es duplicación estructural, causada por el patrón de inicialización anticipada.
- La actividad de fondo continúa toda la sesión — se observó *polling* durante más de 8 minutos — aunque el usuario nunca abra el chat
- Solo hace falta cuando el usuario abre el chat explícitamente; se carga en todas las visitas

Beacons de analítica

Doce *beacons* por carga de página hacia el endpoint de recogida. Cada URL lleva un sufijo aleatorio único, que es el mecanismo estándar de la plataforma para evitar la caché, no un identificador por evento.



Composición de los doce beacons. Ocho corresponden a activaciones de *feature flags* distintas; cuatro son eventos genuinamente diferentes. *Reconstruido a partir de PERF-007-beacon-burst.png.*

EL PROBLEMA NO SON DATOS DUPLICADOS

Ocho de los doce comparten el mismo tipo de evento de activación, pero cada uno lleva un identificador de *flag* y una variante distintos: son ocho *feature flags* activas en `/reserva/crear`, no ocho copias del mismo evento. El problema es que se envían como **ocho peticiones HTTP en serie dentro de 430 ms donde bastaría una agrupada**. La plataforma de analítica soporta agrupación de eventos; la de experimentación dispara un *beacon* por activación sin agrupar nada.

Los cuatro restantes sí son eventos distintos: vista de página (×2), formulario listo, y un clic sobre la recepción de esquema de tarifas.

Tracker de personalización — recién identificado

Se carga un tracker adicional desde el CDN de un proveedor de personalización. Su alcance e impacto no están medidos todavía. Debería entrar en la auditoría de carga diferida junto con el gestor de etiquetas.

Impacto medido

Peso descomprimido por proveedor. Caché en frío, España, fibra, sin limitación. El total medido, excluyendo personalización y beacons, es de 828 KB transferidos y 5 350 KB descomprimidos en más de 28 peticiones. *Reconstruido a partir de PERF-007-no-cache.har y las trazas asociadas.*

Solución recomendada

Gestor de etiquetas — cargar tras la interactividad

```
// Sustituir el script anticipado por una inyección posterior al render
afterNextRender(() => {
  const script = document.createElement('script');
  script.src = '//assets.tagmanager-vendor.com/container-xxx.min.js';
  script.async = true;
  document.head.appendChild(script);
});
```

Asistente — cargar solo con interacción del usuario

```
// 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 {
  // inyectar aquí el script del asistente, y solo ahora
}
```

Analítica — agrupar las activaciones de *flags*

```
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 {
  // enviar un único beacon con todas las activaciones pendientes
  this.flushScheduled = false;
  this.pendingActivations.length = 0;
}
```

Personalización — auditar y aplazar

Determinar si el script de personalización hace falta para el render inicial. Si no, aplicarle el mismo patrón de `afterNextRender()`.

Evidencia

CONDICIONES DE CAPTURA

Producción autenticada, España (UE), MacBook Pro M3 sobre fibra, sin limitación. Condiciones completas en la **Sección D**.

Fichero	Estado	Notas
`PERF-007-no-cache.har · with-cache.har`	**Recopilada**	—
`PERF-007-*-trace.json.gz`	**Recopilada**	Confirman el `EvaluateScript` de 51,3 ms y la duplicación por *iframe*
`PERF-007-tracking-waterfall-tagmanager.png`	**Recopilada**	13 peticiones, 466 KB / 2 294 KB
`PERF-007-tracking-waterfall-assistant.png`	**Recopilada**	Duplicación por *iframe* visible
`PERF-007-beacon-burst.png`	**Recopilada**	Los 12 beacons y el panel de cabeceras

Fichero	Estado	Notas
`PERF-007-tagmanager-timing.png`	**Pendiente**	Detalle: encolado 4,06 s, *handshake* 61 ms, total 138 ms
`PERF-007-assistant-zoom.png`	**Pendiente**	Misma captura ampliada a 0–15 s

Mejora esperada

- **Aplazar el gestor de etiquetas** saca 466 KB (2,25 MB descomprimidos) y una tarea de 51,3 ms de la ruta crítica, y elimina los 61 ms de *handshake* en frío para quien llega por primera vez.
- **Corregir la inicialización del asistente** elimina la duplicación por *iframe*, quita más de 362 KB de la carga inicial para el 80 % largo de usuarios que nunca abren el chat, y detiene los ocho minutos de *polling* de fondo.
- **Agrupar los beacons** elimina siete de las doce peticiones de analítica por carga.
- **Auditar la personalización** puede aportar ahorro adicional una vez se mida su alcance.

Referencias

→ **web.dev — INP** · los scripts de terceros son la causa principal de un INP malo: compiten con los manejadores de interacción en el hilo principal

→ **Harry Roberts — Identifying, Auditing, and Discussing Third Parties** · marco de trabajo para auditar, clasificar y negociar el coste de terceros con las partes implicadas

→ **web.dev — Best practices for tags and tag managers** · cuándo cargarlos y cómo medir su impacto real en Core Web Vitals

→ **web.dev — Third-party JavaScript** · impacto, diagnóstico y mitigación: fachada, carga diferida, Partytown

→ **Chrome DevTools — Identify slow third-party scripts** · aislar el coste por origen desde el panel Performance

→ **Angular — afterNextRender** · el hook correcto para aplazar trabajo hasta después del primer ciclo de render

08 PERF-008 P2 · Estratégica

Precargar los chunks críticos de la aplicación

Dos chunks que suman 2,8 MB son necesarios para pintar el formulario, y el navegador no se entera de que existen hasta que ha ejecutado el bundle principal. Dos huecos, 1 550 ms de espera pura.

ALCANCE	MÉTRICAS	ESFUERZO	RESPONSABLE	ESTADO
`/reserva/crear`	LCP TTI	Bajo	Frontend + Plataforma	Abierta

~1 550 ms de sobrecoste de descubrimiento en serie sobre fibra — 720 ms antes de la tercera oleada y 830 antes de la cuarta — durante los cuales la red está ociosa esperando a que el JavaScript nombre el siguiente fichero.

Problema

Dos chunks grandes con carga perezosa hacen falta para renderizar el formulario, pero solo se descubren después de que se ejecute el bundle inicial.

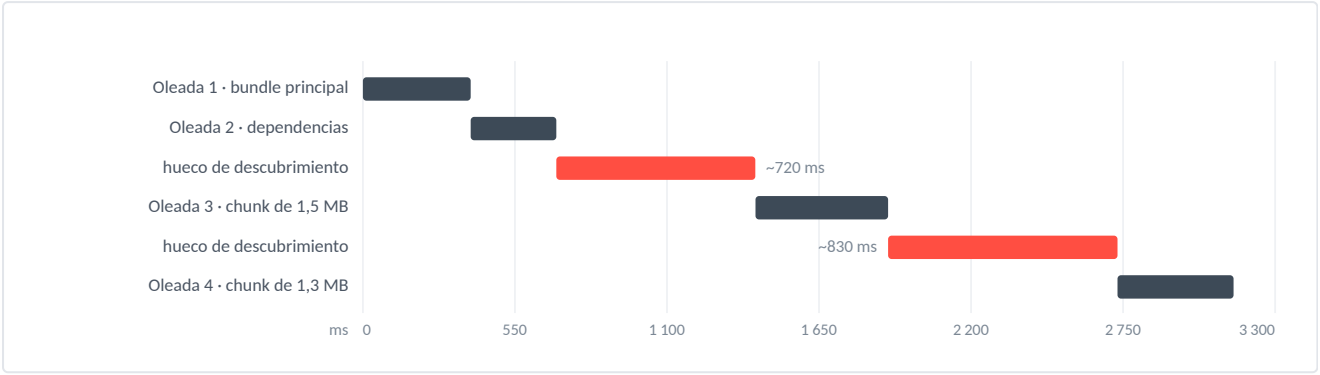
Chunk	Tamaño	Momento de descubrimiento
`chunk-[hash-a].js`	~1,5 MB ~396 KB comprimido	Tras ejecutar el bundle principal
`chunk-[hash-b].js`	~1,3 MB ~262 KB comprimido	Tras ejecutar el bundle principal

Como los chunks se descubren mediante `import()` dinámicos, el navegador no puede empezar a pedirlos hasta que corre el router de Angular, que a su vez depende de que el bundle principal esté parseado y ejecutado. El resultado es una dependencia en serie donde 2,8 MB de código crítico esperan detrás del bundle.

Impacto en Core Web Vitals

Métrica	Efecto
LCP	Retrasada — el formulario no se pinta hasta que el chunk de 1,5 MB está descargado y parseado.
TTI	Retrasada — Angular no puede activar la ruta hasta que los dos chunks están listos.

Impacto medido



Oleadas de carga y huecos de descubrimiento. Las dos barras rojas no son descargas: son tiempo en que la red está parada porque el navegador todavía no sabe qué pedir. *Reconstruido a partir de PERF-008-key-chunks-waterfall.png y PERF-008-no-cache.har.*

Solución recomendada

Añadir pistas de `modulepreload` en el `<head>` del shell:

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

LOS NOMBRES DE CHUNK LLEVAN HASH DE CONTENIDO

Cambian en cada build, así que una pista escrita a mano se queda obsoleta en silencio. Hay que mantenerla sincronizada por alguno de los tres mecanismos siguientes.

1 · Generación en tiempo de build

```
// scripts/inject-modulepreload.js (dentro del pipeline de build)
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}">`);
// inyectar en la plantilla del shell...
```

2 · Chunks con nombre

El build ya tiene `namedChunks: true`. Basta con mapear los nombres a los ficheros con hash en un manifiesto del artefacto de build.

3 · Coordinación con el equipo de plataforma

Si el shell lo genera la plataforma corporativa, las pistas de `modulepreload` deben incluirse en su propio pipeline.

Evidencia

CONDICIONES DE CAPTURA

Producción autenticada, España (UE), MacBook Pro M3 sobre fibra, sin limitación. Condiciones completas en la **Sección D**.

Fichero	Estado	Notas
`PERF-008-no-cache.har · with-cache.har`	**Recopilada**	Cuatro oleadas en serie; los dos chunks grandes en la tercera y la cuarta
`PERF-008-*-trace.json.gz`	**Recopilada**	El bundle principal termina y hay un hueco visible antes de los chunks
`PERF-008-key-chunks-waterfall.png`	**Recopilada**	Cascada con las oleadas y los dos huecos
`devtools-snippet-perf008.js`	**Recopilada**	Snippet reutilizable

Mejora esperada

Los dos huecos medidos suman ~1 550 ms sobre fibra. Precargar ambos chunks en paralelo con el bundle principal puede eliminarlos, con una mejora de TTI de **500 a 1 500 ms**.

Con RTT de 300–500 ms cada ida y vuelta de descubrimiento cuesta bastante más, y la mejora se estima entre 2 y 4× superior. Los valores de España/fibra son una cota inferior.

Referencias

- **web.dev — LCP** · los chunks críticos en la ruta de render retrasan LCP directamente
- **web.dev — Module preload** · `rel=modulepreload` frente a `rel=preload` : diferencias semánticas y prelectura del grafo de módulos
- **MDN —** `<link rel="modulepreload">` · especificación: `as` , `crossorigin` e integración con módulos ES
- **Angular — Lazy loading feature modules** · cómo genera Angular los chunks con `loadChildren` y su efecto en el grafo
- **Angular CLI —** `--stats-json` · identificar los chunks más grandes y su árbol de dependencias

09 PERF-009 P2 · Estratégica

Speculation Rules para la navegación de confirmación

El usuario ha invertido entre dos y cinco minutos rellenando el formulario. El clic final — el que confirma la reserva — le cuesta después entre uno y tres segundos más de espera.

ALCANCE	MÉTRICAS	ESFUERZO	RESPONSABLE	ESTADO
`/reserva/crear` → `/reserva/confirmada`	LCP TTI en la página siguiente	Medio	Frontend	Abierta

1–3 s de retardo de navegación en el momento de mayor intención de todo el embudo, medido sobre fibra con caché templada. Estimado en 3–6 s para el mercado latinoamericano.

CORRECCIÓN SOBRE EL FLUJO DE LA INTERFAZ

En el modo cómodo, pulsar «Ver resumen» abre una capa modal dentro de `/reserva/crear` sin cambiar la URL. La navegación real ocurre al pulsar el botón de confirmar dentro de esa capa, que lleva a `/reserva/confirmada`. En el modo compacto, «Confirmar reserva» navega directamente. **No existe ninguna ruta intermedia de revisión**, aunque borradores anteriores de este seguimiento hablaban de una.

Problema

Completado el formulario, el usuario navega a `/reserva/confirmada`. En ese momento el navegador tiene que:

1. Pedir el HTML y los assets de la página de confirmación
2. Descargar el chunk perezoso del contenedor de confirmación
3. Hacer las llamadas a la API que confirman la reserva

No hay ninguna estrategia de *prefetch* para esta ruta. El retardo cae justo donde la frustración sale más cara: el usuario ya ha invertido minutos de trabajo y está comprometiéndose con una compra.

Impacto en Core Web Vitals

Métrica	Efecto en `/reserva/confirmada`
LCP	Deja de ser espera — los assets ya están precargados antes de navegar.
TTI	Reducida — el chunk específico de confirmación ya está en caché.

Impacto medido

Métrica	Valor
Retardo de navegación, crear → confirmada	1–3 s
LCP en `/reserva/confirmada`	**Traza pendiente**
TTI en `/reserva/confirmada`	**Traza pendiente**

Solución recomendada

Inyectar las reglas de especulación cuando el usuario ha demostrado progreso real en el formulario — por ejemplo, al validar el primer paso — y no en la carga de página.

```
// En comfortable-navigation.component.ts — desde viewSummary()
// o desde la suscripción a showViewSummaryButton$
private injectSpeculationRules(): void {
  if (!HTMLScriptElement.supports?.('speculationrules')) return;         // detección
  if (document.querySelector('script[type="speculationrules"]')) return; // ya inyectada

  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);
}
```

Para navegadores sin soporte — Safari y Firefox — se recurre a un *prefetch* estándar:

```
<!-- Alternativa para navegadores sin Speculation Rules -->
<link rel="prefetch" href="/booking/es-mx/reserva/confirmada">
```

Nivel de *eagerness*	Comportamiento
`immediate`	Precarga o prerrenderiza de inmediato
`moderate`	Cuando el enlace es visible en el viewport — el valor por defecto y la elección correcta aquí
`conservative`	Solo al pasar el ratón o pulsar

Con `moderate` se evita gastar ancho de banda en usuarios que abandonan el formulario pronto.

Evidencia

CONDICIONES DE CAPTURA

Producción autenticada, España (UE), MacBook Pro M3 sobre fibra, sin limitación. Condiciones completas en la **Sección D**.

Fichero	Estado	Notas
`PERF-009-baseline.har`	**Recopilada**	Caché templada, sin reglas activas. Todos los recursos de la página de confirmación arrancan de cero al navegar.
`devtools-snippet-perf009.js`	**Recopilada**	Snippet reutilizable
`PERF-009-baseline-trace.json`	**Pendiente**	Línea temporal completa de la navegación; el coste de LCP y TTI que la corrección eliminaría
`PERF-009-nav-waterfall.png`	**Pendiente**	Cascada con los recursos arrancando de cero
`PERF-009-spec-rules-panel.png`	**Pendiente**	Panel Speculation Rules vacío, confirmando que no hay reglas

ESTE CAPÍTULO TODAVÍA NO TIENE GRÁFICOS DE MEDICIÓN

A diferencia del resto del informe, PERF-009 se apoya en un único HAR. La línea base de 1–3 s es real, pero gruesa: falta la traza que la descompondría en tiempo de chunk frente a tiempo de API. Las cifras de mejora que siguen son proyecciones, no mediciones.

Mejora esperada

Navegación proyectada tras la corrección. Estimaciones sobre la línea base medida de 1–3 s, no mediciones. *Base: PERF-009-baseline.har.*

Para el mercado latinoamericano con RTT de 300–500 ms, la línea base se estima entre 3 y 6 s y la mejora proyectada escala en la misma proporción.

Referencias

- **Chrome Developers — Prerender pages for instant navigations** · guía completa: niveles de *eagerness*, restricciones del prerenderizado y comportamiento en páginas autenticadas
- **MDN — Speculation Rules API** · especificación: sintaxis JSON, campos `prefetch` y `prerender`, *eagerness* y selectores `where`
- **web.dev — `<link rel="prefetch">`** · la alternativa compatible con Safari y Firefox
- **Chrome DevTools — Debug Speculation Rules** · estado de las reglas, errores y páginas prerrenderizadas activas

10 PERF-010 P3 · Acabado

Aplicar `content-visibility` a las secciones fuera de pantalla

Desplegar un menú de la barra lateral repinta el formulario entero. El navegador calcula el layout de 225 de los 242 nodos del documento en una sola pasada, porque nada del formulario está contenido.

ALCANCE	MÉTRICAS	ESFUERZO	RESPONSABLE	ESTADO
`/reserva/crear` vista compacta	TBT INP	Bajo-medio	Frontend	Abierta

225 / 242 nodos del documento entran en una única pasada de layout, con `#document` como raíz. Frames consecutivos de 133,5 ms y 108,4 ms — ocho veces el umbral de 16,6 ms.

Problema

La vista compacta del formulario renderiza todas las secciones en el DOM a la vez, aunque solo una sea visible en cada momento. El panel de resumen renderiza además el estado completo de todos los pasos ya completados.

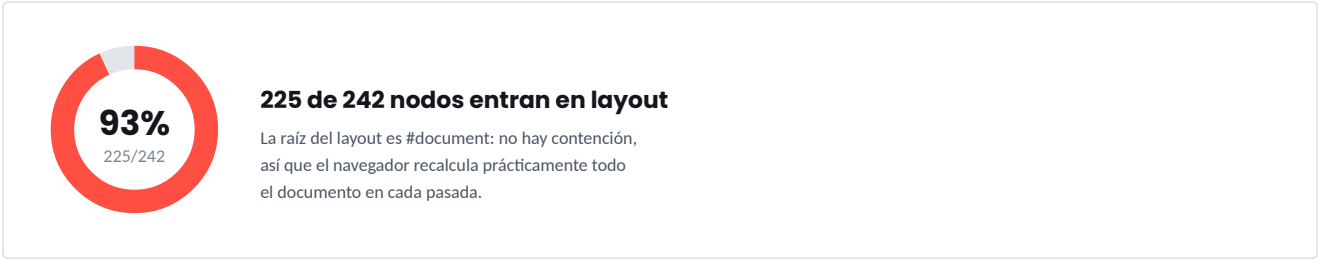
El navegador hace layout, cálculo de estilo y pintado de todos los nodos del DOM con independencia de su visibilidad. Con quince o más secciones montadas — facturación, servicios extra, datos del huésped, documentación y el resto — eso es mucho trabajo de render innecesario. La configuración de pasos define más de treinta componentes, y todos pueden existir en el DOM durante las transiciones.

Es un problema que se nota sobre todo en dispositivos modestos, donde el cuello de botella es el render y no la red.

Impacto en Core Web Vitals

Métrica	Efecto
TBT	Se reduce con la corrección — menos trabajo de layout y pintado significa tareas más cortas en el hilo principal.
INP	Mejora con la corrección — menos elementos renderizados significa respuesta más rápida al desplazamiento y a la entrada de datos.

Impacto medido



Proporción de nodos que entran en cada pasada de layout. El panel de resumen de DevTools para la tarea seleccionada indica 3,41 ms de duración, raíz `#document` y, como primera invalidación, la detección de cambios de Angular. *Reconstruido a partir de PERF-010-flame-chart.png.*

Duración de frames frente al umbral de fluidez. Dos frames consecutivos muy por encima de los 16,6 ms que marcan los 60 fps. *Reconstruido a partir de la pista Frames de PERF-010-flame-chart.png.*

EL SÍNTOMA MÁS REVELADOR

Con el resaltado de pintado activado en DevTools, desplegar el submenú de la barra lateral provoca destellos de repintado sobre el contenido principal del formulario: campos que no tienen ninguna relación con la interacción. Ocurre porque las secciones no tienen contención de pintado: cualquier cambio del DOM que el navegador no pueda acotar fuerza un repintado del documento entero. `content-visibility: auto` aplica implícitamente `contain: layout style paint`, que confina los repintados al límite del elemento.

Solución recomendada

Opción A · CSS `content-visibility`, sin tocar JavaScript

```
// En los estilos de sección del formulario compacto
.booking-form-section {
  content-visibility: auto;
  contain-intrinsic-size: auto 300px;  // altura aproximada, evita CLS
}
```

Opción B · `@defer (on viewport)` de Angular — recomendada

Difiere además la inicialización del componente y su detección de cambios, no solo el layout y el pintado.

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

@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>
}
```

Evidencia

CONDICIONES DE CAPTURA — ATENCIÓN A LA DESVIACIÓN

Producción autenticada, España (UE), MacBook Pro M3 sobre fibra. Sin limitación de red. **Se aplica un factor 4× de CPU donde se indica**, porque en un problema de render el cuello de botella representativo es la CPU y no la red. Línea base completa en la **Sección D**.

Fichero	Estado	Notas
`PERF-010-flame-chart.png`	**Recopilada**	Tarea de layout con 225 nodos sucios
`PERF-010-paint-flashing.png`	**Recopilada**	Repintado provocado por una interacción ajena
`PERF-010-recording.mp4`	**Recopilada**	Grabación del repintado
`PERF-010-baseline-trace.json`	**Pendiente**	Duración total de Layout, Recalculate Style y Paint, con y sin la corrección
`PERF-010.har`	**Pendiente**	Prioridad baja: es un problema de render, no de red

Mejora esperada

Aplicando `content-visibility: auto` o `@defer (on viewport)`, las secciones fuera de pantalla quedan excluidas de la pasada de layout y el recuento de nodos sucios baja a los 40–60 que están realmente en el viewport. Mejora esperada: **reducción del 40 al 60 % en tiempo de layout y pintado**. En dispositivos modestos, entre 200 y 500 ms menos de Time to Interactive y un desplazamiento apreciablemente más suave.

El multiplicador esperado para el mercado remoto está entre 2 y 4×, y aquí lo impulsa el hardware más limitado, no la red.

Referencias

→ **web.dev — INP** · menos trabajo de layout y pintado significa respuesta más rápida a la interacción

- → **web.dev — content-visibility: the new CSS property that boosts your rendering performance** (Una Kravets y Vladimir Lebedev) · el artículo original, con mediciones de hasta 7× de reducción en tiempo de render
- → **MDN — content-visibility** · valores `auto`, `hidden` y `visible`, y su relación con la contención CSS
- → **MDN — contain-intrinsic-size** · reservar espacio para evitar CLS cuando `content-visibility: auto` omite el layout
- → **Angular — bloques @defer** · disparadores `on viewport`, `on idle`, `on interaction` y bloques `@placeholder`, `@loading` y `@error`

11 PERF-011 P1 · Victoria rápida

Deduplicar las llamadas a la API

Cambiar un solo campo del formulario lleva la sesión de 32 llamadas a 57. Diecisiete de ellas son idas y vueltas sobrantes a endpoints que ya habían respondido.

ALCANCE	MÉTRICAS	ESFUERZO	RESPONSABLE	ESTADO
`/reserva/crear`	TTI INP	Medio	Frontend	Abierta

~21,8 s de tiempo de red estimado como desperdiciado tras cambiar un único campo, frente a ~2,4 s en la carga de página. Línea base España/fibra; en el mercado remoto se desperdicia entre 2 y 4× más.

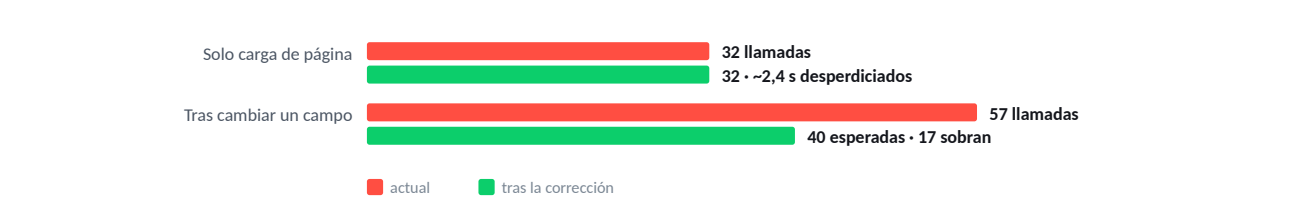
SE COMPONE CON PERF-005

Cada llamada duplicada cuesta una ida y vuelta completa hasta el origen. Deduplicar elimina la llamada; corregir PERF-005 abarata las que queden. Las dos merecen la pena, y se multiplican entre sí.

Problema

El análisis de red muestra que varios endpoints se llaman muchas más veces de las necesarias.

Llamadas por endpoint tras cambiar un campo. Frente a las una o dos que cabría esperar en cada caso. *Reconstruido a partir de PERF-011-call-summary_after.json.*



Volumen total de llamadas. En la carga de página ya hay exceso; una sola interacción lo agrava. *Reconstruido a partir de PERF-011-call-summary.json y _after.json.*

Cada llamada duplicada desperdicia una ida y vuelta de 150 a 250 ms sobre la línea base España/fibra, CPU de backend para una petición idéntica, y entre 50 y 200 KB de ancho de banda del usuario por respuesta.

Impacto en Core Web Vitals

Métrica	Efecto
`TTI`	Degradada — las llamadas redundantes compiten por el ancho de banda de la conexión HTTP/2 y retrasan la disponibilidad de los esquemas.
`INP`	Degradada — las llamadas que dispara el cambio de un campo se lanzan varias veces y producen tirones visibles.

Impacto medido

Métrica	Carga de página	Tras un cambio de campo
Llamadas totales	32	57
Exceso sobre las ~40 esperadas	—	+17
Endpoints con duplicados	4	11
Tiempo de red desperdiciado (estimado)	~2,4 s	~21,8 s
Peor caso	`v2/extra-services` ×5, media 324 ms	`v2/extra-services` ×7, media 670 ms

Solución recomendada

Nivel 1 · `debounceTime` y `distinctUntilChanged`

```
formValueChanges$.pipe(
  debounceTime(300),
  distinctUntilChanged((a, b) => isEqual(a, b)),          // ignora payloads idénticos
  switchMap((payload) => this.schemaService.fetch(payload)), // cancela las que vuelan
)
```

`switchMap` es el operador crítico: cancela las peticiones en vuelo cuando llega un disparo nuevo, lo que evita que una respuesta antigua aterrice después de una reciente.

Nivel 2 · Deduplicar peticiones en vuelo

```
// libs/shared/schemas/data-access — capa de deduplicación
@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>;
  }
}
```

Como clave, un hash del payload de la reserva.

Nivel 3 · Cachear respuestas deterministas

Para los endpoints cuya respuesta es determinista dado el mismo payload de entrada, cachear en cliente con el hash del payload como clave, e invalidar al reiniciar el estado de la reserva.

Evidencia

CONDICIONES DE CAPTURA

Producción autenticada, España (UE), MacBook Pro M3 sobre fibra, sin limitación. Condiciones completas en la **Sección D**. Con RTT de 300–500 ms cada duplicado cuesta de 2 a 4× más.

Fichero	Estado	Notas
`PERF-011.har`	Recopilada	Caché templada, todas las llamadas de una carga completa
`PERF-011-trace.json.gz`	Recopilada	Bloques XHR repetidos: no hay cancelación ni deduplicación activa
`PERF-011-duplicate-calls.png`	Recopilada	Listado completo, ordenado por nombre
`PERF-011-endpoint-detail.png`	Recopilada	Detalle del peor caso, con una petición cancelada en vuelo
`PERF-011-call-summary.png / .txt`	Recopilada	Salida del snippet
`PERF-011-call-summary.json · _after.json`	Recopilada	Línea base y post-interacción

Mejora esperada

Tras una única interacción se observan 57 llamadas donde se esperan unas 40: 17 idas y vueltas sobrantes. A entre 150 y 670 ms cada una sobre la línea base España/fibra, eliminarlas reduce el tiempo de red desperdiciado estimado de **~21,8 s a prácticamente cero**.

Con RTT de 300–500 ms la ganancia escala en la misma proporción: la corrección vale bastante más para el mercado prioritario de lo que sugieren las cifras europeas.

Referencias

- → **web.dev — INP** · las llamadas duplicadas que dispara la interacción degradan INP directamente
- → **RxJS — switchMap** · cancela el observable anterior cuando llega una emisión nueva; el antídoto contra las cascadas reactivas
- → **RxJS — debounceTime** · suprime las emisiones intermedias durante un intervalo; imprescindible en cambios de formulario
- → **RxJS — share** · multidifusión de un observable frío para que varios suscriptores compartan una sola ejecución
- → **RxJS — distinctUntilChanged** · evita emisiones cuando el valor no ha cambiado; complemento necesario de `debounceTime`
- → **Angular — HttpClient** · los interceptores como punto centralizado para caché y deduplicación

12 PERF-012 P3 · Infraestructura

Activar HTTP/2 103 Early Hints

Durante los primeros 463 ms de una carga en frío el navegador ha enviado su petición y no ha recibido nada. No puede descargar ni un recurso. Early Hints convierte esa ventana muerta en tiempo de descarga.

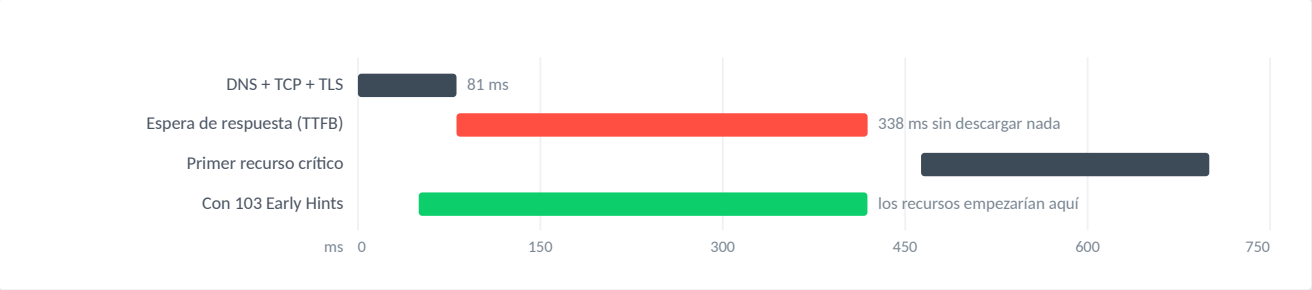
ALCANCE	MÉTRICAS	ESFUERZO	RESPONSABLE	ESTADO
Todas las rutas servidas por el shell	FCP LCP	Medio	Infra + Plataforma	Abierta

~413 ms de tiempo ocioso recuperable por carga en frío: una ventana observada de 463 ms menos unos 50 ms estimados de retardo de la respuesta 103.

Problema

El TTFB del motor de reservas ronda los 250 ms, y en la captura de abajo se midió en 338 ms. Durante toda esa ventana el navegador ha enviado su petición y no ha recibido nada: no puede empezar a pedir ningún recurso porque todavía no sabe cuáles son.

HTTP/2 103 Early Hints permite al servidor enviar una respuesta preliminar con cabeceras `Link` de *preload antes* de que la respuesta HTML completa esté lista. El navegador puede actuar sobre esas pistas de inmediato.



La ventana ociosa, y lo que ocuparía Early Hints. El *handshake* completo son 81 ms; la espera de respuesta, 338. La barra verde marca dónde podrían empezar a descargarse los recursos críticos. *Reconstruido a partir de PERF-012-ttfb-timing.png y PERF-012-ttfb-summary.png.*

Impacto en Core Web Vitals

Métrica	Efecto
FCP	Mejora — el CSS y el JS críticos empiezan a descargarse durante el tiempo de proceso del servidor.
LCP	Mejora — los recursos críticos llegan antes y desbloquean el render.

La técnica rinde justo cuando el TTFB es alto, que según **PERF-005** es aquí la condición normal.

Impacto medido

Conexión en frío — 2 de junio de 2026, España → `/es-mx/`

Fase	Duración
Resolución DNS	20,11 ms
Conexión inicial	38,98 ms
SSL	21,90 ms
Petición enviada	0,91 ms
Espera de respuesta — TTFB	338,11 ms
Descarga de contenido	2,18 ms
Total	**402,58 ms**

Los 338 ms de TTFB están por encima de la línea base de ~250 ms de PERF-005, algo coherente con la variabilidad del servidor. DNS, TCP y TLS juntos suman unos 81 ms: cuatro veces menos que el TTFB por sí solo, lo que convierte la ventana ociosa en el coste dominante con diferencia.

Conexión templada — reutilización de HTTP/2

Fase	Duración
Encolado y bloqueo	1,57 ms
Petición enviada	0,23 ms
Espera de respuesta — TTFB	228,40 ms
Descarga de contenido	1,89 ms
Total	**232,09 ms**

El TTFB templado es menor porque no hay coste de conexión, pero siguen siendo 228 ms durante los cuales no puede descargarse nada.

Ausencia de pistas confirmada en las dos capas

- **Capa HTTP** — la sección de cabeceras de respuesta no contiene ninguna cabecera `Link:`, de modo que Early Hints no está en funcionamiento.
- **Capa HTML** — el snippet de *resource hints* confirma que no hay `<link rel="preload">` ni `<link rel="preconnect">` propios de la aplicación en el DOM. El primer recurso crítico arranca a los 463 ms, frente a un `responseStart` del HTML de 403 ms.

Solución recomendada

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;
}
```

En el CDN

La mayoría de CDN modernos ofrecen Early Hints como una opción de un solo clic, e incluso infieren las cabeceras `Link` a partir de la respuesta HTML anterior. Los assets concretos a precargar deben salir del `stats.json` del build: los chunks iniciales de CSS y JS que hacen falta en todas las cargas.

Evidencia

CONDICIONES DE CAPTURA

Producción autenticada, España (UE), MacBook Pro M3 sobre fibra, sin limitación. Condiciones completas en la **Sección D**. En el mercado latinoamericano el TTFB se estima aún mayor, entre 400 y 600 ms, lo que hace la técnica proporcionalmente más rentable.

Fichero	Estado	Notas
`PERF-012-no-cache.har`	Recopilada	TTFB del documento ≈ 338 ms; todo el CSS y JS arrancan después
`PERF-012-no-cache-trace.json.gz`	Recopilada	Hueco de TTFB antes de cualquier barra de descarga
`PERF-012-ttfb-timing.png · -warm.png`	Recopilada	Conexión en frío y templada
`PERF-012-response-headers-01.png · -02.png`	Recopilada	Sin cabecera `Link:`
`PERF-012-resource-hints.png / .txt`	Recopilada	Sin *preload* ni *preconnect* propios en el DOM
`PERF-012-ttfb-summary.png`	Recopilada	Confirmación numérica de la ventana ociosa
`PERF-012-ttfb-summary.txt`	Pendiente	Versión en texto de la salida del snippet

Mejora esperada

103 Early Hints puede recuperar hasta **~413 ms** de tiempo ocioso por carga en frío. Los recursos críticos empezarían a descargarse durante la espera de TTFB en lugar de después.

Referencias

- **web.dev — FCP** · Early Hints adelanta el inicio de descarga de los assets dentro de la ventana ociosa
- **Chrome Developers — HTTP 103 Early Hints** · guía completa con ejemplos, impacto real y compatibilidad de servidores
- **RFC 8297 — An HTTP Status Code for Indicating Hints** · especificación: semántica del 103, campos `Link` admitidos y comportamiento esperado del cliente
- **Documentación de Early Hints de tu CDN** · activación e inferencia automática de cabeceras `Link`
- **web.dev — Optimize TTFB** · Early Hints dentro del trabajo global sobre TTFB: CDN, *streaming* de HTML y caché en el *edge*

13 PERF-013 Resuelta

Cargar el SDK de mapas bajo demanda

666 KB de SDK de mapas se cargaban a los 4,7 segundos de cada visita, sin ningún mapa en pantalla y sin ninguna interacción previa. Había dos disparadores independientes; hubo que corregir los dos.

ALCANCE	MÉTRICAS	ESFUERZO	RESPONSABLE	ESTADO
`/reserva/crear` campos de dirección	TBT TTI	Bajo	Frontend	Resuelta

666 KB de JavaScript pedidos a los 4 716 ms sin una sola interacción previa — 186 KB comprimidos, 84 ms de parseo fuera del hilo principal y 20 ms de evaluación dentro, en cada visita, también con caché templada.

Encuadre estratégico

PRIMERO PREGUNTAR SI EL RECURSO HACE FALTA

Antes de recurrir a microoptimizaciones — *preconnect*, *preload*, compresión — conviene preguntarse si el recurso pinta algo en esta vista. Aquí no: no se renderiza ningún mapa en `/reserva/crear`; el SDK solo hace falta en el momento en que alguien enfoca un campo de dirección y empieza a escribir; y quien escribe la dirección a mano no lo necesita nunca.

La carga bajo demanda y el patrón de fachada son la primera medida correcta para cualquier script de terceros pesado que no esté en la ruta crítica. *Preconnect* es una mejora secundaria, útil solo *después* de arreglar la carga anticipada: mientras el script se carga incondicionalmente no aporta absolutamente nada.

Problema

Secuencia de carga del SDK de mapas. La última fila es la clave: no hay ninguna interacción del usuario en toda la secuencia. *Reconstruido a partir de PERF-013-no-cache-trace.json.gz.*

El SDK solo hace falta cuando el usuario enfoca un campo de dirección y escribe (autocompletado), cuando selecciona una sugerencia (geocodificación para extraer los componentes de la dirección), o cuando consulta el mapa de la propiedad, que es una ruta distinta y ajena a `/reserva/crear`.

Causa raíz — dos disparadores, no uno

Disparador 1, principal · suscripción en el constructor

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

```
private readonly placesLibraryStatus =
  toSignal(this.mapsApiService.placesLibraryStatus$, {
    requireSync: true,   // ← suscripción síncrona al construir
  });
```

`placesLibraryStatus$` deriva de `api$`, que inyecta la etiqueta `<script>` del SDK en el DOM:

```
Se renderiza el formulario
  → se construye AddressInputComponent
    → toSignal(placesLibraryStatus$, { requireSync: true })
      → placesLibraryStatus$ → api$
        → se inyecta <script src="maps.geo-vendor.com/...">
          → petición de red a los 4 716 ms   ← confirmado en la traza
```

Es incondicional: se dispara toque o no toque el usuario un campo de dirección.

Disparador 2, secundario · `preloadLib()` tras inicializar la vista

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

La memoización de `api$` hace que el script se inyecte una sola vez, así que el disparador 1 se lleva la petición. Pero `preloadLib()` la provocaría igualmente si se eliminara el primero. Había que corregir los dos.

Solución aplicada

Corrección 1 · No suscribirse al estado del SDK en el constructor

El componente usaba `placesLibraryStatus$` solo para decidir qué interfaz mostrar: campo de búsqueda si el SDK ha cargado, campo manual si ha fallado o sigue cargando. Ese patrón fuerza una carga del SDK únicamente para pintar la interfaz.

Se sustituye por un valor optimista: mostrar el campo de búsqueda por defecto, asumir que el SDK estará disponible, y caer al campo manual solo cuando falle de verdad. El esqueleto de carga sobra: al usuario no le aporta nada saber que el SDK se está inicializando antes de tocar el campo.

```
// Antes — dispara la carga del SDK al construir:
private readonly placesLibraryStatus =
  toSignal(this.mapsApiService.placesLibraryStatus$, { requireSync: true });

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

// Después — nada se carga hasta que hay interacción.
private readonly mapsLoadFailed = signal(false);

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

// Lo llama AddressSearchInputComponent cuando el SDK falla:
onMapsLoadFailed(): void {
  this.mapsLoadFailed.set(true);
}
```

Corrección 2 · Mover `preloadLib()` al evento de foco

```
// Después: el SDK carga solo cuando el usuario muestra intención
private handleTextInputFocus(): Subscription {
  return fromEvent(this.textInput, 'focus').subscribe(() => {
    this.autocompleteApiService.preloadLib();    // ← al primer foco
    if (this.currentPredictions().length > 0 || this.showLastAutocompleteOption()) {
      this.showDropdown();
    }
  });
}
```

`preloadLib()` ya tiene una guarda interna, así que llamarlo en cada foco es seguro: el script se inyecta una sola vez.

Mejora opcional · *preconnect* al pasar el ratón

Eliminada la carga anticipada, un `preconnect` al dominio del SDK en el `mouseenter` del contenedor de dirección le da al navegador ventaja sobre el *handshake* antes de que el usuario llegue a enfocar el campo. Importa sobre todo donde el RTT es alto.

Evidencia

CONDICIONES DE CAPTURA

Producción autenticada, España (UE), MacBook Pro M3 sobre fibra, sin limitación. Condiciones completas en la **Sección D**.

Fichero	Estado	Notas
`PERF-013-no-cache.har · with-cache.har`	**Recopilada**	El SDK carga de forma anticipada incluso con todo lo demás en caché
`PERF-013-*-trace.json.gz`	**Recopilada**	Sin eventos `mousedown`, `click`, `focus` ni `keydown` antes de la petición
`PERF-013-maps-waterfall.png`	**Recopilada**	—
`PERF-013-maps-flamechart.png`	**Recopilada**	—
`devtools-snippet-perf013.js`	**Recopilada**	Snippet reutilizable

Mejora obtenida

Escenario	Antes	Después
El usuario nunca toca un campo de dirección	666 KB descargados, parseados y ejecutados	**0 KB** — el SDK no llega a cargarse
El usuario enfoca un campo de dirección	666 KB a los 4 716 ms	666 KB al primer foco, en diferido
Usuario en el mercado remoto	Varios segundos de descarga antes siquiera de parsear	Totalmente diferido; con *preconnect* al pasar el ratón, el coste de RTT se reduce aún más

La línea base es España (UE) sobre fibra: el mejor caso posible. Con RTT de 300–500 ms y dispositivos modestos, las cifras del «antes» eran de 2 a 4× peores.

Referencias

- → **V8 — The cost of JavaScript in 2019** (Addy Osmani) · el coste de parseo, compilación y ejecución más allá del tamaño de descarga; el dato clave para justificar la carga diferida ante los responsables
- → **web.dev — Third-party JavaScript** · impacto, diagnóstico y mitigación: fachada, carga diferida, Partytown
- → **web.dev — Facade pattern for third-party embeds** · sustituir un tercero pesado por un marcador ligero hasta que hay interacción
- → **Documentación del SDK de mapas — importación dinámica de librerías** · cargar bajo demanda solo las librerías necesarias
- → **MDN — evento focus** · propagación, `focusin` como alternativa delegable y diferencias con `click`

14 PERF-014 P2 · Estratégica

Optimizar el sprite SVG de iconos

El sprite del sistema de diseño lleva 694 iconos. La aplicación usa 76. Los otros 618 se envían a todos los usuarios en todas las cargas y no se renderizan ni una sola vez.

ALCANCE	MÉTRICAS	ESFUERZO	RESPONSABLE	ESTADO
Todas las rutas el sprite es global	LCP TBT	Medio	Frontend + Diseño	Abierta

11 % del sprite se usa: 76 iconos de 694. El fichero pesa 861 KB descomprimidos, 292 KB en red, y empieza a descargarse a los 3 077 ms, justo dentro de la ventana de LCP.

Problema

`design-system-icons.svg` es el sprite de iconos del sistema de diseño corporativo y se carga como asset estático en todas las páginas. Está referenciado en la configuración de assets del proyecto:

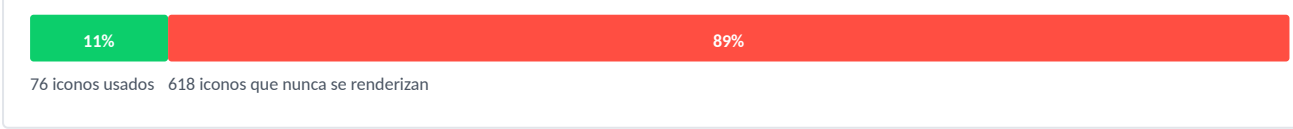
```
{
  "glob": "design-system-icons.svg",
  "input": "node_modules/@vantia/design-system",
  "output": "/assets"
}
```

Este asset monolítico bloquea el hito de render de los dos segundos y contribuye al Total Blocking Time.

Impacto en Core Web Vitals

Métrica	Efecto
LCP	Bloqueada — el sprite compite con recursos críticos justo en la marca de los dos segundos.
TBT	Aumenta — la descarga y el parseo de un asset grande alimentan tareas largas del hilo principal.

Auditoría de uso de iconos



Proporción real de uso del sprite. De los 76 iconos usados en toda la aplicación, solo 26 aparecen en `/reserva/crear`. *Reconstruido a partir de PERF-014-icon-audit.txt y PERF-014-used-icons.txt.*

Métrica	Valor
Iconos totales en el sprite	694
Iconos usados en toda la aplicación	76
Iconos visibles en `/reserva/crear`	26
Ratio de uso	11 %
Tamaño en red	292 KB comprimidos
Tamaño descomprimido	861 KB
Momento de inicio de descarga	3 077 ms
Duración de la descarga	244 ms

Solución recomendada

Paso 1 · Auditar el uso — hecho

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

Resultado: 76 iconos únicos en todas las plantillas, de 694 disponibles.

Paso 2 · Generar un sprite a medida — preferido

```
// scripts/build-icon-sprite.js — dentro del pipeline de build
const svgstore = require('svgstore');
const usedIcons = require('./used-icons.json');   // salida del paso 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());
```

Después se sustituye el *glob* de assets por el sprite generado.

Paso 3 · Incorporar en línea los iconos visibles al cargar

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

Paso 4 · Carga diferida — alternativa si el sprite a medida no es 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);
});
```

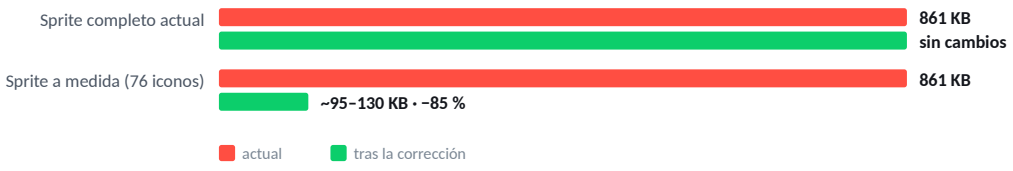
Evidencia

CONDICIONES DE CAPTURA

Producción autenticada, España (UE), MacBook Pro M3 sobre fibra, sin limitación. Condiciones completas en la **Sección D**.

Fichero	Estado	Notas
`PERF-014-no-cache.har · with-cache.har`	Recopilada	Arranque a los 3 077 ms, 292 KB en red / 861 KB descomprimidos
`PERF-014-*-trace.json.gz`	Recopilada	Barra de descarga visible en la ventana de 3–3,3 s
`PERF-014-sprite-waterfall.png`	Recopilada	El sprite junto a los recursos críticos concurrentes
`PERF-014-sprite-flamechart.png`	Recopilada	Barra ancha de descarga solapando la ventana de LCP
`PERF-014-icon-audit.txt · used-icons.txt`	Recopilada	Auditoría completa y los 76 iconos referenciados

Mejora esperada



Peso del sprite antes y después. Un sprite que contenga solo los 76 iconos usados se estima entre 95 y 130 KB comprimidos: alrededor de un 85 % menos. *Estimación sobre la proporción medida de uso.*

Sobre la línea base España/fibra el sprite arranca a los 3 077 ms y tarda 244 ms, completándose hacia los 3 321. Con RTT de 300–500 ms, la ventana de descarga de 861 KB se estima entre 2 y 3× más larga, lo que añade dos o tres segundos para el mercado prioritario.

- **Sprite a medida (pasos 1–2):** elimina el bloqueo de render de los dos segundos sobre fibra, y rinde proporcionalmente más en conexiones limitadas.

- **Iconos en línea (paso 3):** aparecen de inmediato, con coste de red cero.
- **Carga diferida (paso 4):** aplaza los 861 KB hasta que la página es interactiva.

Referencias

→ **web.dev — LCP** · el sprite compite con los recursos de la ruta crítica y retrasa LCP directamente

→ **SVGO** · el optimizador de SVG más extendido; reduce el tamaño de cada icono antes de componer el sprite

→ **svgstore** · generar sprites concatenando iconos como elementos `<symbol>`

→ **svg-sprite** · alternativa más completa: genera sprites en varios formatos (inline, CSS, view, defs)

→ **MDN — elemento SVG `<use>`** · referencia `href`, comportamiento del shadow DOM y restricciones de origen cruzado

SECCIÓN E

Dónde este informe es débil

Dicho sin rodeos, para que nadie tenga que descubrirlo en la reunión.

Incidencia	Hueco
Todas	No existe ninguna captura tomada desde el mercado latinoamericano. Todas las cifras de ese mercado son extrapolaciones desde la línea base europea.
PERF-003	Faltan el panel Timing del SDK de mapas y la traza de rendimiento.
PERF-004	Faltan las capturas de cascada en frío y en templado.
PERF-005	No hay línea base desde EE. UU., así que el delta de RTT está inferido, no medido. Faltan también la captura de red con la columna TTFB y la traza.
PERF-007	Faltan el detalle de tiempos del gestor de etiquetas y la cascada ampliada del asistente. El alcance del tracker de personalización está sin medir.
PERF-009	Se apoya en un único HAR. Sin traza, sin capturas y sin desglose del 1–3 s en tiempo de chunk frente a tiempo de API.
PERF-010	Faltan la traza de línea base y la comparativa posterior a la corrección.
PERF-012	Falta la versión en texto del resumen de TTFB.

UNA INCOHERENCIA QUE CONVIENE RESOLVER

En el material de partida aparecen dos cifras distintas para el trayecto UE→EE. UU.: unos 120 ms en las notas de evidencia de cada incidencia y unos 250 ms en el resumen de PERF-005. Ambas se han conservado tal como se registraron. Conviene confirmar una y corregir la otra antes de usar estos números para construir un caso de negocio.

Qué haría falta para cerrar los huecos

- **Una captura desde el mercado prioritario.** Es la aportación de mayor valor. Convierte catorce extrapolaciones en catorce mediciones.
- **Una línea base desde EE. UU.** Zanja la discusión sobre cuánto del TTFB es distancia y cuánto es proceso de origen.
- **Monitorización continua.** Este informe es una fotografía. Sin datos de campo y sin tests de regresión, dentro de tres meses no habrá forma de saber si las correcciones aguantaron.

SOBRE ESTE DOCUMENTO

Informe de muestra elaborado por **PerfReviews**. Corresponde a una auditoría real, anonimizada: las cifras, los hallazgos y las recomendaciones son los originales; el cliente, el sector y los proveedores de terceros se han ficcionado. Los gráficos son reconstrucciones vectoriales de los datos de captura.

perf.reviews · mail@perf.reviews