La revolución del rendimiento en las páginas web

Kamo Sylla Diaby

Resumen– Las aplicaciones web cada vez son más populares entre los internautas debido a su compatibilidad con cualquier sistema operativo y que solo se requiere de un navegador para acceder al servicio. Uno de los aspectos que ha fomentado este crecimiento es ofrecer una mayor usabilidad y mejor experiencia al usuario a través del diseño responsive. El hecho de que las aplicaciones web se puedan adaptar a cualquier dispositivo es un factor clave para que el usuario pueda hacer un buen uso del servicio. Pero ya no se limita simplemente al diseño de la aplicación, sino a su rendimiento, es muy importante que el usuario pueda navegar por una página web con unos tiempos de respuesta óptimos debido a nuestro estilo de vida reactivo y sensible al tiempo. Aquí es donde entra en juego la tecnología Web Performance Optimization (WPO), su importancia es tal que ya se ha convertido en una métrica que utiliza Google a la hora del posicionamiento SEO [1], una web con un bajo rendimiento es penalizada a la hora de posicionarla en los resultados de búsqueda, siendo un cuello de botella a la hora de conseguir un gran volumen de tráfico orgánico.

Palabras clave– WPO, SEO, Caché, Responsive, HTTP, Cliente-Servidor, CDN.

Abstract– Web applications are increasingly popular among Internet users due to their compatibility with any operating system and that only one browser is required to access the service. One of the aspects that has fostered this growth is to offer greater usability and better user experience through responsive design. The fact that web applications can be adapted to any device is a key factor for the user to make good use of the service. But it is no longer limited simply to the design of the application, but to its performance, it is very important that the user can browse a website with optimal response times due to our reactive and time-sensitive lifestyle. This is where Web Performance Optimization (WPO) technology comes into play, its importance is such that it has already become a metric used by Google when it comes to SEO positioning [1], a web with a low performance is penalized when positioning it in search results, being a bottleneck when it comes to getting a large volume of organic traffic.

Keywords– WPO, SEO, Cache, Responsive, HTTP, Client-Server, CDN.

1 Introducción - Contexto del trabajo

El hecho de que WPO se aplica sobre el mundo web, es necesario tener conocimientos sobre los diferentes personajes que interactúan en este escenario, con el fin de tener una visión global del porqué y como ciertas optimizaciones logran un mayor impacto que otras en el rendimiento. Para poder abordar el problema es necesario conocer la arquitectura que hay debajo de las aplicaciones web y qué elementos la conforman, con tal de buscar qué aspectos optimizar.

• E-mail de contacto: 1493829@uab.cat
• Mención realizada: Ingeniería de Computadores
• Trabajo tutorizado por: Ramon Grau Sala (Departamento de Arquitectura de Computadores y Sistemas Operativos)
• Curso 2021/2022

1.1. Arquitectura cliente-servidor

La arquitectura cliente-servidor es uno de los modelos más utilizados en muchos de los servicios y protocolos que ofrece Internet [2]. En este escenario hay dos personajes claramente diferenciados. El cliente, que podría ser cualquier usuario que se conecte a través de un dispositivo a la red y accede a un servicio como puede ser una red social. Y el servidor, el cual normalmente suele tener un hardware más potente para poder dar respuesta en un tiempo óptimo a todos los usuarios que solicitan el servicio, además de un software específico para poder ofrecerlo, como puede ser un servidor web de Apache o Nginx.

1.2. Protocolo HTTP

En esta arquitectura, la comunicación se realiza a través de los conocidos mensajes HTTP [3] que permiten al cliente solicitar los diferentes recursos al servidor, con el fin de renderizarlos y mostrarlos al usuario. Gracias a este protocolo y a su estandarización, el servidor es capaz de interpretar la información solicitada por el cliente con el fin de ofrecerle una respuesta a través de este mismo protocolo. Estos mensajes son la base de todo porque lo que se busca en WPO son las mejores maneras de que estos se envíen de manera óptima y comprimida por la red para lograr una menor latencia.

2 Objetivos

Según WPO Stats [4], en Amazon 0,1 segundos de retraso implican una pérdida del 1 % de los ingresos. En Google 0,4 segundos de retraso causan una caída del 0,59 % de las búsquedas por usuario; 0,5 segundos más en cargar implica un 25 % menos de búsquedas. En Facebook 0,5 segundos más lento provoca una caída de tráfico del 3 %; 1 segundo provoca una caída del 6 %.

Son datos que han influenciado a la hora de trazar mi objetivo, el cual consiste en trasladar los conocimientos y metodologías de la ingeniería del rendimiento adquiridas durante estos años en la facultad, al mundo web.

Es un mundo complejo debido a que se requiere analizar los diferentes puntos de vista de los clientes, es posible que para un cliente A al servidor web al cual quiere solicitar un recurso o servicio se encuentre geográficamente en la misma ubicación, pero para un cliente B se encuentre en una ubicación mucho más alejada, o bien es posible que el cliente A acceda al servicio desde una red cableada y que el cliente B acceda desde una conexión 4G.

Todo son factores que es necesario a tener en cuenta a la hora de realizar las diferentes optimizaciones, la mejor manera de realizar este estudio y documentar qué tecnologías actuales se utilizan en el mundo WPO, qué impacto tienen sobre el rendimiento y si provocan conflictos en las páginas web, como puede ser la compresión o combinación de los archivos, es llevarlo a la práctica. Siendo este un documento que refleja el trabajo efectuado durante estos meses, el cual servirá a los diferentes ingenieros como una base para acelerar sus páginas webs, ofrecer una mejor experiencia al usuario y lograr un mejor posicionamiento en los buscadores.

3 Estado del arte

La tecnología WPO está logrando popularidad estos últimos años, por lo que no hay un gran repertorio de investigaciones realizadas, más bien existen pequeños manuales sobre como aplicar las diferentes técnicas de optimización. El problema de estas, es que en muchas ocasiones se basan en aplicar las técnicas, sin transmitir el porqué de estas al lector, en este documento se pretende que el lector tenga un mayor ángulo visual de que elementos conforman estas y porque ciertas optimizaciones tienen mayor impacto que otras, de manera que proactivamente pueda sacar sus conclusiones y sus mejoras que puedan nutrir a otros ingenieros formando un ciclo de aprendizaje.

4 Metodología

Con el fin de alcanzar los objetivos, me he basado en la metodología **SCRUM** en el sentido de que cada semana he realizado una entrega parcial del producto final. Entre estas prácticas de SCRUM me he centrado en el Product Backlog y Sprint Backlog, ya que los conceptos de Scrum Master, Product Owner, Scrum Team no tienen mucho sentido en un proyecto de fin de curso, donde soy el único integrante del equipo. Mi lista de tecnologías a estudiar vendrían siendo mi Product Backlog, y el subconjunto semanal de estas tecnologías que implemento vendrían siendo mi Sprint Backlog, es decir, aquellas tecnologías que acabaré documentando en mi informe esa semana, elaborando cada vez más un documento incremental.

Con el fin de llevarlo a la práctica, he instalado dos servidores, uno virtual y otro dedicado a los cuales apunto a través de un dominio ficticio, en el caso del virtual accedo a este a través de `http://tfe.virtual` y en el caso del dedicado accedo a este a través del `htpp://tfe.dedicado`. En estos dos servidores he creado una página web replicada (Figura 1) la cual cuenta con diferentes recursos como imágenes, scripts, fuentes, en la cual realizo las optimizaciones (si es posible) de la tecnología que forma parte de mi Product Backlog de esa semana.

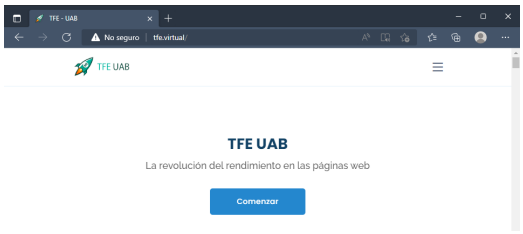

Fig. 1: Página web

Trello, es un administrador de proyectos que es válido para metodología Agile y SCRUM, por lo que me ha parecido ideal para tener un control de mi product backlog. Mi tablero lo defino en tres espacios. Véase la figura (2).

- **Lista de tareas**: En primer lugar, el listado de tareas que conforma el Product Backlog de las diferentes fases.
- **En proceso**: Es un subconjunto del Product Backlog de la lista de tareas, que hace referencia al Sprint Backlog, es decir, aquello en lo que estoy trabajando en esta semana en concreto de mi planificación.
- **Hecho**: Son todas las tecnologías que ya he estudiado, implementado y redactado.

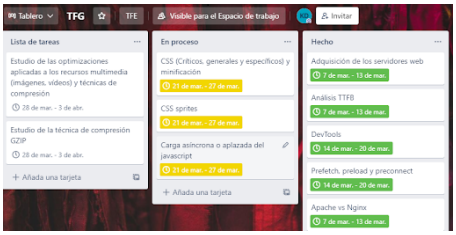

Fig. 2: Tareas del tablero Trello

5 Servidores dedicados y virtuales

Una de las bases y aspectos más críticos sobre la que se rige WPO, es la selección de un correcto modelo de alojamiento, de manera que podamos ofrecer un servicio, en un tiempo de respuesta óptimo para un gran número de clientes. Uno de los modelos más habituales y económicos es la adquisición de un **hosting compartido** [5] donde almacenamos nuestras aplicaciones web, las cuales residen físicamente con aplicaciones de otros clientes compartiendo el software y el hardware. Este tipo de modelo se basa en la virtualización del hosting. En una misma máquina donde tenemos instalado un servidor web, es posible a través de los archivos de configuración especificar unas directivas para ofrecer una aplicación web u otra según el nombre del dominio, siendo esta la base de la virtualización.

Un ejemplo de esto serían las siguientes líneas de código, las cuales especifican dos alojamientos virtuales de Apache, donde el alojamiento `http://tfe.dedicado` servirá los archivos del directorio C:/xampp/htdocs/tfe.dedicado y el alojamiento `http://tfe.virtual` servirá los archivos del directorio C:/xampp/htdocs/tfe.virtual.

```
<VirtualHost *:80>
   DocumentRoot
       "C:/xampp/htdocs/tfe.dedicado"
   ServerName tfe.dedicado
</VirtualHost>
<VirtualHost *:80>
   DocumentRoot
       "C:/xampp/htdocs/tfe.virtual"
   ServerName tfe.virtual
</VirtualHost>
```

A la hora de ofrecer un buen rendimiento, esta opción no es la más adecuada debido a que un exceso de consumo por parte de otros usuarios de otras aplicaciones con las que compartes el servidor, pueden influir negativamente en el rendimiento de tu página web.

Los **servidores virtuales** [5] bajan un nivel más bajo, ya no virtualizamos a nivel de hosting, sino a nivel de una máquina física, disponemos de una máquina virtual en la cual nosotros tenemos el total control y contamos con recursos propios, a pesar de que la máquina es compartida con otras máquinas virtuales. Véase la figura (3).

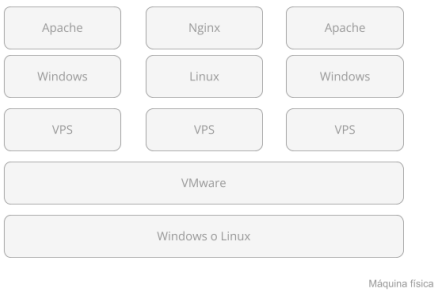

Fig. 3: Arquitectura servidor virtual

El último modelo es el **servidor dedicado** [5] en el cual disponemos en exclusiva de una máquina física. Sin embargo, no siempre un servidor dedicado ofrece un mejor rendimiento que los servidores virtuales, ya que esto depende del servicio contratado. Para este proyecto nos interesa trabajar de base con un servidor virtual o bien dedicado, puesto que el alojamiento compartido ya de entrada no proporciona un buen punto de partida para la implementación de las técnicas WPO.

En este proyecto he trabajado con un servidor virtual y otro dedicado los cuales cuentan con recursos muy similares, el virtual sirve como centro de datos del alojamiento con el dominio ficticio `http://tfe.virtual`. En el caso del dedicado he utilizado un alojamiento con el dominio ficticio `http://tfe.dedicado`.

6 (Time to First Byte) TTFB

La métrica que suele estar relacionada con el rendimiento de un servidor respecto a otro es el Time To First Byte (TTFB), es decir, cuanto debe esperar el navegador hasta recibir el primer byte de información. Como mayor sea este, peor puede ser el rendimiento de las aplicaciones web. Esta métrica está fuertemente relacionada con las prestaciones de la máquina. Según **Google PageSpeed Insights** [6] el tiempo recomendable es de menos de 200 ms. Entre 300 y 500 ms es un tiempo estándar y más de 600 ms ya requiere una revisión por parte del servidor.

Hay diferentes opiniones sobre la importancia del TTFB, según Cloudflare [7], a pesar de que en muchas ocasiones TTFB se utiliza como una medida de la rapidez del servidor, en una prueba que realizaron, instalaron un servidor que primero responde en el encabezado una “H” y después responde con el resto de cabeceras, cuando hicieron la prueba el TTFB era el tiempo desde que se enviaba la solicitud hasta que la “H” era recibida, es decir que no era el primer byte de datos sino el primer byte de la cabecera HTTP. Esto son cosas muy diferentes debido a que los encabezados de respuesta se pueden generar muy rápidamente, pero son los datos lo que afectará a que el usuario llegue a ver la página.

Por otro lado, según Ilya Grigorik (Ingeniero de rendimiento web de Google) [7] la latencia de red es uno de los principales cuellos de botella del rendimiento en la web. Ya que este requiere de una búsqueda Domain Name System (DNS), TCP, dos viajes de ida y vuelta para negociar el túnel TLS y la solicitud y respuesta HTTP, para obtener el primer byte del documento HTML, no el primer byte de la cabecera.

Realizando un primer experimento en los dos alojamientos podemos inferir el TTFB de nuestro servidor virtual y dedicado. Para ponernos en el peor caso en lo que es el experimento es recomendable utilizar un escenario donde no consideremos cache y utilicemos una conexión 3G. Los TTFB de los dos servidores son muy similares por las prestaciones tan similares que ofrecen y además de ser un servicio ofrecido por un mismo proveedor donde las condiciones de red y la ubicación geográfica es prácticamente la misma. Lo interesante es analizar que el TTFB se encuentra dentro de los rangos de un tiempo estándar aproximadamente. Cabe destacar que un usuario que emplee una conexión 4G o WI-FI, obtendría un TTFB bastante inferior.

Fig. 4: TTFB del servidor dedicado

Fig. 5: TTFB del servidor virtual

Esta métrica se calcula en 3 pasos:

- **Tiempo de solicitud**: El tiempo que tarda en enviarse la solicitud HTTP.
- **Procesamiento de la solicitud**: El tiempo que el servidor web tarda en procesar la solicitud.
- **Tiempo de respuesta al usuario**: El tiempo que tarda en responder el servidor con el primer byte de datos.

6.1. Tiempo de solicitud

Es la primera comunicación que realiza el cliente hacia al servidor para solicitarle un recurso [7]. Podemos ponernos en la situación de un cartero que sale de nuestro hogar con una carta que le entregamos y nosotros esperamos una respuesta por parte del receptor. Es posible que tardemos en recibir la respuesta porque el receptor se encuentra en otra ciudad o bien que el cartero debe primero buscar la dirección exacta del receptor (Tiempo de búsqueda DNS) o bien que el cartero vaya en bicicleta en lugar de un vehículo motorizado (Velocidad de internet).

6.2. Procesamiento de la solicitud

Una vez el servidor recibe la petición es necesario que procese dicha solicitud, por ejemplo que deba buscar un archivo PHP o que realice un gran número de consultas a una base de datos lenta y los recursos del servidor sean ineficientes en cuanto al disco I/O y memoria, provocando una lentitud en este segundo factor.

6.3. Tiempo de respuesta al usuario

Este último es el tiempo que tarda el servidor en responder con los datos solicitados, es decir, el momento en el que el navegador (cliente) recibe el primer byte. Quitando DNS este también puede verse afectado por los mismos factores que el tiempo de solicitud, como una velocidad de red lenta por parte del servidor o del usuario.

7 HTTP/1.1 y HTTP/2

Desde 1999 hasta 2015 la versión 1.1 del protocolo HTTP es la versión que se utilizaba para el estándar de comunicación en la World Wide Web, sin embargo, en 2015 aparece HTTP/2.0 y recientemente HTTP/3.0 la cual no acaba de ser compatible con todas los navegadores por lo que nos centraremos en estas dos primeras versiones.

HTTP/2 llegó en un punto en el que se buscaba más velocidad a la hora de servir los contenidos, reduciendo la latencia sobre todo en plataformas móviles. Este, está basado en el protocolo SPDY, desarrollado principalmente en Google con la intención de reducir la carga de las páginas web con aspectos como la compresión, la multiplexación y la priorización.

Este mejora notablemente en cuanto a rendimiento a su antecesor, las claves de esta se encuentran en:

- **Formato binario**: HTTP/2 utiliza una capa de encuadre binario para encapsular los mensajes [8], mientras que en HTTP/1.1 mantiene las solicitudes en formato texto, siendo el primero más compacto y fácil de interpretar.
- **Multiplexación**: Esta es una de las mayores mejoras, en HTTP/1.1 se abre una conexión TCP para cada elemento que se requiera cargar en la web, en cambio, en HTTP/2 se abre una única conexión TCP y se envía a través de este único canal [9], sin tener que estar abriendo y cerrando conexiones.

  Cabe destacar que en el primer protocolo no se podía enviar una solicitud hasta haber recibido la anterior, es decir, es un proceso en serie (head-of-line blocking), sin embargo, con HTTP/2 se puede enviar múltiples solicitudes al mismo tiempo y recibir las respuestas en paralelo. Véase la figura (6).

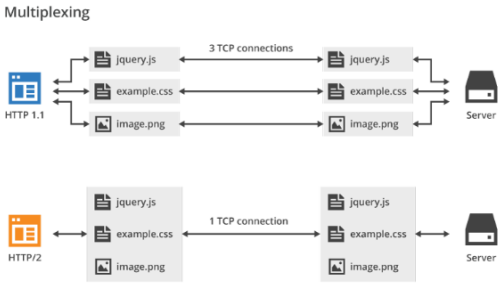

Fig. 6: Multiplexado en HTTP/2

- **Server Push**: HTTP/2 permite enviar información a la caché sin que se haya realizado la petición, el servidor predice los datos que se podrían necesitar [9].
- **Compresión de cabeceras**: En HTTP/2 las cabeceras se comprimen con HPACK para transmitir lo mismo en un volumen más reducido de datos. Esto no tiene un gran impacto en el rendimiento, pero todo influye [9].

8 Nginx y Apache

Nginx y Apache son los dos servidores más populares de Internet, más de un 50 % de los sitios web ejecutan estos dos servidores web [10]. Nginx surgió para abordar las limitaciones de rendimiento de los servidores Apache. Este se basa en una arquitectura basada en eventos, a diferencia de Apache de enfoque a proceso y subproceso o por conexión.

El servidor web Apache tiene una arquitectura centrada en los procesos, para cada petición este genera un hilo en el servidor, generando en ocasiones un consumo de CPU y RAM innecesario. En aplicaciones web con un gran número de clientes, esto se multiplica y desencadena una sobrecarga en el servidor.

Nginx, por otro lado, cuenta con un proceso máster que es el encargado de realizar las tareas críticas de bajo nivel, luego existen otros procesos secundarios que se encargan de otras tareas como:

- Carga de caché.
- Administración de caché.
- Workers, encargados de administrar las conexiones de red, I/O a disco y comunicación con otros servicios.

Apache siempre ha sido el líder del mercado, pero el rendimiento en las páginas se han vuelto algo tan importante que Nginx en 2022 ya ha superado a este primero en cuanto a usuarios utilizando el servicio, como se muestra en la siguiente gráfica (7):

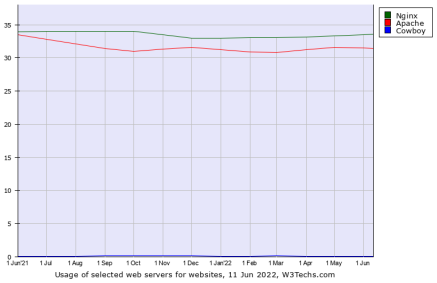

Fig. 7: Apache frente a Nginx

9 DevTools

DevTools es una herramienta intuitiva utilizada para analizar el impacto que tienen las diferentes optimizaciones que se presentan a lo largo del documento.

Los navegadores más populares ofrecen una serie de herramientas que permiten desde depurar los scripts, analizar el Document Object Model (DOM), hasta perfilar todos los aspectos del rendimiento de nuestra página web desde nuestro ordenador, teniendo la posibilidad de cambiar el tipo de conexión para ofrecer un escenario más similar al que pudiera tener un cliente de nuestra página web.

A través de la opción **herramientas para desarrolladores** de nuestro navegador podemos acceder al conjunto de utilidades que nos ofrece esta, pero en concreto para realizar análisis de WPO, el apartado de **Red** (Figura 8). Este es el más adecuado para poder analizar qué tiempos consumen los recursos

Las opciones más interesantes son:

- (1) Modificar las características de la conexión
  - (a) Deshabilitar la caché
  - (b) Modificar el tipo de conexión
- (2) Filtrado de los recursos
- (3) Listado de recursos
- (4) Waterfall

- **Modificar las características de la conexión**: En este apartado es posible modificar el tipo de conexión y deshabilitar la memoria caché.

  Por defecto el tipo de conexión es sin limitación, pero al encontrarme en una red cableada los tiempos de

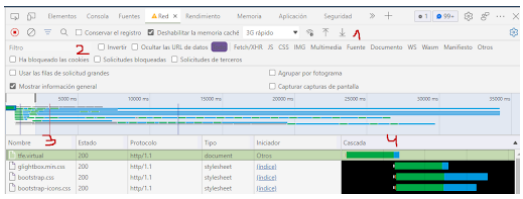

Fig. 8: Dev Tools

respuesta son rápidos, el problema es que no puedo esperar que todos los usuarios se conecten a través de una red rápida a mi página, por lo que es mejor ponerse en un caso medio o en el peor caso.

- **Filtrado de los recursos**: Esta sección es muy útil, cuando queremos analizar los tiempos de un subconjunto de los recursos como pueden ser las imágenes.
- **Listado de recursos**: Esta sección contiene todas las peticiones que se realizan en nuestra página web, es posible añadir más información aparte del estado de la petición, el protocolo, el tipo, el tamaño, etc.
- **Waterfall**: Ofrece una visión gráfica de los tiempos de los recursos asignando un color a los diferentes estados por los que pasa, como por ejemplo el estado verde está asociado al TTFB como observamos en el análisis de las prestaciones del servidor. Esta visión gráfica rápidamente permite comprender que en el caso de la petición a la raíz de la web el mayor tiempo se invierte en el time to first byte (Figura 9). Lo que es la propia descarga de todo el documento es la región azul.

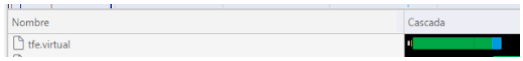

Fig. 9: Waterfall

10 Cascading Style Sheets (CSS)

Cuando solicitamos una página web a través de la barra de búsqueda del navegador, a bajo nivel estamos enviando un mensaje a través de la red al servidor que contiene el recurso, el cual nos responde normalmente con un documento HTML o PHP que el navegador es capaz de interpretar. Durante esa fase de interpretación va solicitando los recursos que requiere para poder renderizar la página web de manera que sea humanamente legible. Precisamente en esta fase de solicitud de recursos hay algunos estilos CSS que tienen más prioridad que otros en función del peso que tengan a la hora de renderizar las partes más básicas de la página web.

Estas prioridades establecen 3 tipos de archivos CSS. Con el objetivo de cargar menor cantidad de código según el lugar donde nos encontremos.

Críticos: Son aquellos estilos básicos para crear la estructura de la página web. Son los de máxima prioridad y son los primeros que se deberían de cargar [11].

Una buena práctica de esto podría ser un CSS que solo contenga el esqueleto de la página sin aplicar los colores, transformaciones, hovers, etc. O bien establecer como CSS crítico aquello que es visible de primeras en nuestra pantalla.

DevTools, también nos ofrece una herramienta en el apartado de Cobertura que de manera muy visual nos indica qué porcentaje de un recurso CSS se está utilizando en nuestra página web.

En el ejemplo de página utilizada, tenemos varias librerías como Bootstrap, Remixicon que contienen mucha reglas CSS, pero al cargar nuestra página principal realmente se utilizan muy pocas de estas, en cambio, en nuestro style.css la parte utilizada es superior a la parte no utilizada por lo que realmente extraer la parte crítica y cargar la parte no utilizada de manera diferida no generará un gran impacto en la visualización de la página. Véase la figura (10).

Fig. 10: Apartado de cobertura

Existen herramientas como **Puppeteer** que permiten extraer la parte crítica de nuestro CSS, así, cargar esa parte con una alta prioridad mientras la parte media se carga de manera diferida. Esto nos ayuda a mejorar la métrica de First Contentful Paint (FCP) la cual marca el tiempo en el que el primer texto o imagen es pintado.

Generales: Son aquellos comunes para todas las páginas, que no conforman el esqueleto como tal, estos podemos cargarlos con un sistema de preload de manera diferida.

```
<link rel="preload"
   href="general.css"
   as="style"
   onload="this.onload=null;
   this.rel='stylesheet'">

<noscript><link rel="stylesheet"
    href="general.css"></noscript>
```

Específicas: Son aquellas utilizadas en páginas concretas, por ejemplo es muy habitual que todo un E-commerce siga una línea concreta de diseño, pero en el checkout se utilice un diseño totalmente diferente, también se podría carga a través de preload de manera diferida.

10.1. Minificación

Es habitual que utilicemos prácticas donde añadamos comentarios a nuestros estilos CSS y tabulemos para una buena legibilidad del código. Pero, cuando ya hemos finalizado, es un navegador el que debe interpretar esta información, por lo que no es necesario que esté humanamente legible, la minificación permite reducir el tamaño de estos archivos, para reducir la latencia de la petición. Existen multitud de herramientas como **CSS Minify** que realizan esta práctica.

Antes de minificar:

```
/* Estilos para los h1 */
#hero h1{
   color: #3dc125;
   font-size: 1rem;
}
/* Estilos para los botones */
.btn-get-started{
   background: #3dc125;
}
```

Después de minificar:

```
#hero h1{color: #3dc125;font-size:
    1rem;}.btn-get-started{background:
    #3dc125;}
```

En el apartado de **Cobertura** podemos observar como disminuye el total de bytes.

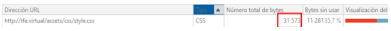

Fig. 11: Antes de minificar

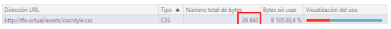

Fig. 12: Después de minificar

Esta práctica también puede aplicarse a los scripts.

10.2. CSS Sprites

Estando en HTTP/1.1 comentamos que la solicitud de cada recurso se hacía abriendo una nueva conexión, la mayoría de optimizaciones se centran en reducir estas peticiones o bien el tamaño de los recursos solicitados en estas. Una de las técnicas que podemos utilizar para reducir el número de peticiones es utilizar sprites a través de CSS.

Un sprite es una colección de imágenes colocadas en una sola imagen. Es decir, en lugar de hacer diversas peticiones para las diferentes imágenes, cargar todas las imágenes con una sola petición. W3Schools muestra un ejemplo del uso de este donde tenemos 3 imágenes colocadas sobre un único sprite [12].

Fig. 13: Sprite

Si quisiéramos mostrar únicamente la imagen de la casa, utilizaremos la propiedad background de CSS para indicar que parte del sprite queremos mostrar.

```
#home {
  width: 46px;
  height: 44px;
  background: url(img_navsprites.gif) 0 0;
}
```

Este código mostraría únicamente la casa, sin embargo, si quisiéramos mostrar la flecha derecha, recorreríamos los 46px/47px del ancho de la casa en el sprite.

```
#prev {
  width: 43px;
  height: 44px;
  background: url(img_navsprites.gif) -47px
      0;
}
```

Este código mostraría únicamente la flecha previa. Esto nos ahorra 3 peticiones frente a 1, cuando utilizamos muchas más imágenes podríamos ahorrar mucho más. Lo más habitual es emplearlo para imágenes pequeñas como logos, íconos, etc.

11 Carga asíncrona o aplazada de JavaScript

Los recursos JavaScript también son de aquellos críticos que bloquean el renderizado de la página web, es decir, que cuando el navegador está procesando un documento HTML (Construyendo la DOM) y se encuentra con:

```
<script>
   alert("TFE UAB");
</script>
```

Debe ejecutar el script al momento, por eso lo más habitual es que los scripts se encuentren abajo del documento HTML y decirle al navegador que los trate de manera diferida o asíncrona. Si no hacemos ese tratamiento cuando llegue al final del documento, la ejecución de ese script seguirá afectando a la carga de la DOM por mucho que esté al final, sin embargo, al añadir defer o async le diremos al navegador que no espere por el script para construir la DOM, sino, cargamos este en segundo plano y lo ejecutamos cuando la petición haya finalizado.

En la figura 14 observamos que si ejecutamos los scripts de manera síncrona, el hilo principal debe esperar a que la solicitud se procese y después ejecutarla, sin embargo al ejecutarlo de manera asíncrona permitimos que en el instante de la petición se siga procesando la DOM y una vez se ha descargado se pare la construcción de la DOM para ejecutar el script. El hecho de no bloquear la DOM mientras se procesa la solicitud es lo que permite reducir el tiempo [13].

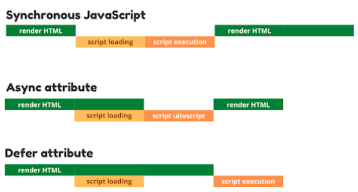

Fig. 14: Async y Defer

Este sería un ejemplo de la carga de manera diferida del script main.

```
<script defer
    src="assets/js/main.js"></script>
```

12 Compresión recursos multimedia

Cuando hablábamos de los sprites, comentamos que es una técnica para reducir el número de peticiones, pero, que también existen técnicas como la minificación para reducir el tamaño de los archivos CSS y JavaScript. En el caso de las imágenes también es importante reducir su tamaño utilizando la compresión. Estos recursos suelen ser los habituales cuellos de botella. Imaginemos que tenemos un blog donde tenemos diversos editores, y cada editor incluye imágenes de medidas que no corresponden a lo que se muestra y tamaños grandes, esto provoca que la página se vea realmente afectada.

En nuestro ejemplo me he puesto en la piel de un editor que no tiene conocimientos de qué impacto tiene subir imágenes sin comprimir en una página web y he subido 4 imágenes en un slider de **1,4 MB** cada una, en cuanto a una página web es una barbaridad tener imágenes de este tamaño, para hacernos una idea la media de páginas web sirven páginas enteras de **2165.5 KB** (2 MB aproximadamente).

Para una conexión 3G podríamos estar más de 39 segundos esperando a que la imagen se terminara de descargar. Véase la figura (15).

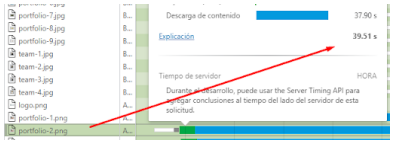

Fig. 15: Tiempo de carga

A través de tinyPNG, tinyJPG podemos comprimir estas imágenes para mejorar el rendimiento, esto ha permitido reducir el tiempo de descarga a la mitad (Figura 16). Cabe destacar que estamos simulando una red móvil, pero estamos utilizando imágenes del tamaño de escritorio, por lo que no es solo comprimir, sino reducir las dimensiones de la imagen a una pantalla móvil, de esa manera se reducirá aún más esa latencia. Además, no estamos considerando caché.

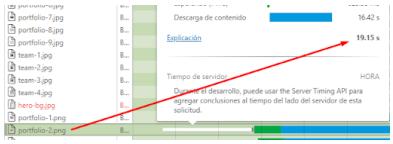

Fig. 16: Tiempo de carga

13 GZIP

GZIP, es una técnica de compresión muy popular utilizada en los servidores web, este lo que hace es comprimir ciertos archivos antes de enviarlos al navegador web. Es empleado para mejorar el ancho de banda. Plataformas como Netflix han conseguido un ahorro del 43 % en el ancho de banda por tener activada la compresión GZIP [14].

Es una implementación muy sencilla para los beneficios que aporta. Hay algunos servidores que tiene el módulo deflate activado, este es muy parecido a GZIP con la diferencia de que posteriormente añade un checksum y un encabezado de datos.

Si tenemos el módulo de GZIP activado en nuestro servidor, simplemente es añadir las directivas en nuestro archivo de configuración, en el caso de Apache este archivo es el **.htaccess**.

```
<ifModule mod_gzip.c>
mod_gzip_on Yes
...
</ifModule>
```

En el caso de Nginx es necesario modificar el archivo **nginx.conf**.

```
gzip on;
...
```

Ciertos proveedores están incluyendo en sus servidores, **Brotli**, este es un algoritmo de compresión desarrollado por Google pensando como sucesor del método GZIP. La idea de este es utilizar un diccionario de palabras clave y frases comunes en el lado del cliente y el servidor. Según CertSimple [15]:

- Los archivos JavaScript comprimidos con Brotli son un **14 %** más pequeños que GZIP.
- Los archivos HTML son un **21 %** más pequeños que GZIP.
- Los archivos CSS son un **17 %** más pequeños que GZIP.

A pesar de todo GZIP, sigue siendo la opción más popular.

14 Memorias Caché

Uno de los conceptos clave de WPO es el uso de las memorias caché, esta es la técnica que más nos ayuda a obtener un gran rendimiento en nuestra página web. Este se basa en la idea de almacenar datos de manera temporal a los que se accede con frecuencia.

Cuando un usuario realiza una petición al servidor, hay un tiempo como vimos en el TTFB para procesar dicha solicitud, en ocasiones, se requiere hacer mucho trabajo de compilado de PHP, consultas a base de datos, etc. La idea es una vez procesada esta información, a medida que se envía la información al cliente, almacenar esta copia estática en memoria, de manera que la próxima vez que el usuario quiera solicitar el recurso se sirva directamente desde una memoria, que evita que efectuemos todo el proceso de compilado.

Si observamos todo esto pasa desde el lado del servidor, pero los navegadores actuales también permiten utilizar la caché desde el lado del cliente. Por lo que tenemos 2 grandes mecanismos para almacenar caché, desde el lado del cliente y desde el lado del servidor. Haciendo un uso adecuado de este, podemos obtener tiempos notables [16].

14.1. Caché del lado del cliente

Los navegadores modernos permiten utilizar caché desde el lado del cliente, es decir, en la memoria del navegador utilizando el propio dispositivo del usuario [16]. Esto permite obtener páginas con latencias realmente bajas, cuando empleamos este tipo de caché estamos sirviendo la página desde nuestro propio navegador, a diferencia del caché del lado del servidor que a pesar de ser una web estática la petición al servidor se ha de realizar de todos modos.

Si nos ponemos en un caso de conexión 3G y con la memoria caché deshabilitada, nuestra página podría tardar alrededor de 40 segundos en cargar; sin embargo, al habilitar la caché con 2 segundos es suficiente para cargar la página.

Existen 2 tipos de caché, **memory cache** y **disk cache**. En la primera, los recursos son cargados y almacenados desde la RAM (Figura 17). La segunda es persistente, los recursos son cargados y almacenados en el disco (Figura 18).

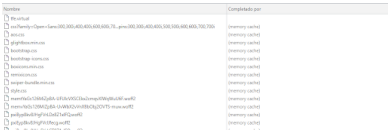

Fig. 17: Caché en RAM

Si cerramos la pestaña, automáticamente esa caché pasará de memory cache a disk cache, para poder recuperar esa información en un futuro. Si volvemos a realizar el proceso observaremos que ahora los recursos han sido cargados desde disk cache.

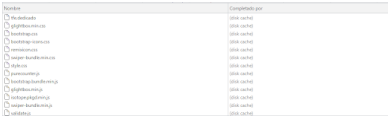

Fig. 18: Caché en disco

Esta caché es más bien utilizada para almacenar recursos estáticos, como HTML, CSS, imágenes, JavaScript, etc. Aunque existe otra caché, la JavaScript, que permite ejecutar scripts dentro del navegador que realizan cambios a medida que el usuario interactúa con el sitio web.

Existen unas series de cabeceras HTTP que podemos utilizar para "controlar" el caché del navegador [3].

- **Expires**: Crea una fecha de expiración para el contenido almacenado en caché del navegador.
- **Pragma (no-cache)**: Permite establecer que cierto contenido nunca sea cacheado.

  Al añadir

  ```
  <meta http-equiv="pragma"
      content="no-cache" />
  ```

  en el **index.html** evitamos que esta página sea cacheada por el navegador. Véase la figura (17).
- **Cache-control**: En el encabezado de control de caché solicita si la respuesta del usuario debe almacenarse en caché. Si la respuesta es Sí, control caché responde cuánto tiempo. Si la respuesta es No, no se almacena en caché.

14.2. Caché del lado del servidor

En este tipo de caché nosotros somos los que implementamos las políticas para servir la caché. De manera básica el servidor ya implementa una política para comparar la instantánea estática de la página con el recurso original para

saber si ha cambiado y entregar uno u otro, pero nosotros también podemos aplicar nuestras políticas de cacheado, lo más habitual es utilizar plug-ins que permitan decidir que aspectos cachear o no.

Es un aspecto muy importante, ya que cuando servimos contenido dinámico, no es tan sencillo y podemos proporcionar datos incoherentes al usuario, por ejemplo cuando añadimos un producto a la cesta, nos dirigimos al carrito y nos encontramos que se ha servido una instantánea cuando se debería de servir el carrito con el nuevo producto añadido, bien se pueden aplicar políticas para decidir que cachear y que no. De cierta manera estamos añadiendo inteligencia a la caché para aplicar una serie de reglas para invalidar esta.

Existen varios términos para nombrar a las cachés del lado del servidor.

- **Almacenamiento en caché móvil**: Es un almacenamiento para aplicaciones y dispositivos móviles.
- **Almacenamiento en caché de usuarios**: Una caché dedicada para cada usuario, por ejemplo, una caché para los usuarios que han iniciado sesión y otro para lo usuarios que no han iniciado sesión.
- **Opcode**: Son cachés utilizadas para servir PHP compilado con menor latencia.
- **Micro caché**: Este almacena en caché una copia estática generada dinámicamente durante un periodo de tiempo muy corto (Utilizado en páginas muy dinámicas).
- **Content Delivery Network (CDN)**: Consiste en el uso servidores para servir contenido más cerca de los usuarios finales del cual se hablará posteriormente en este documento.

Por otro lado, tenemos **Object cache**. Esencialmente, son servidores de base de datos como MYSQL diseñados para almacenar datos de forma persistente y servir contenido más rápido, ya que los datos se almacenan en la RAM. Cada vez que un usuario realice una solicitud a la base de datos, la solicitud la servirá a través de la memoria caché.

Si observamos, cuando la memoria caché es del lado del cliente, solo dicho cliente puede sacar provecho a este, pero desde el lado del servidor, si un visitante 1 solicita una página y acaba almacenada en caché y un visitante 2 solicita esta misma página, el servidor puede reutilizar la caché que generó con la petición del visitante 1 para el visitante 2.

15 Content Delivery Network (CDN)

Cuando buscamos escalar nuestro servicio ofreciendo nuestro producto a un mayor público y ubicado en diferentes ubicaciones geográficas, es de gran importancia servir los recursos según las características de dichos clientes. Para ello se hace uso de las CDN o red de distribución de contenido, las cuales se encargan de servir recursos estáticos desde la localización más cercana a tu público. Véase la figura (19).

Consiste en un grupo de servidores distribuidos geográficamente que trabajan juntos para ofrecer una entrega rápida, si tenemos un E-commerce relacionado con la moda y queremos empezar a comercializar nuestros productos en Latinoamérica, el hecho de no usar una CDN implica que los clientes de dicha ubicación tengan que realizar las peticiones a nuestro propio servidor el cual se encuentra en una ubicación mucho más alejada de la que nos encontramos nosotros, y aún peor es que solo es un servidor el que debe procesar todas estas peticiones.

Una solución a esto, sería contratar un servicio de CDN para que los usuarios de Latinoamérica se les sirva los recursos desde un servidor físicamente en Argentina, por ejemplo, y que en caso de que el recurso no se encuentra se comunique con el servidor de origen, esto permite reducir notablemente el ancho de banda.

Fig. 19: CDN

16 Proxies

El hecho de utilizar memorias cachés de lado del servidor, permite que el navegador sirva la página web propiamente de la lectura de esta sin necesidad de realizar la petición al servidor, tanto este procedimiento, como el uso de CDN permiten reducir la sobrecarga en el servidor, un aspecto muy importante para que el servidor pueda atender a las diferentes peticiones en unos tiempos óptimos.

Otro aspecto que ayuda a reducir la sobrecarga de un único servidor sirviendo el contenido, es el uso de proxies, existen diversos tipos, pero en lo que repercute al rendimiento web hablamos de los proxies inversos. Este, no es más que un servidor intermediario que habla en nombre de otro, por lo que cuando un usuario realice una petición a nuestro servidor, esta será atendida por el servidor proxy [17]. Véase la figura (20).

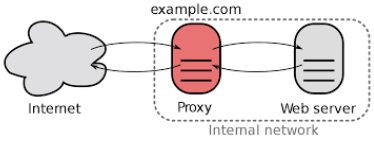

Fig. 20: Proxy inverso

La idea del uso de proxies inversos en una arquitectura web es ofrecer una capa de seguridad protegiendo al servidor origen contra infracciones, proporcionar caché, debido a que si hay en caché el recurso solicitado en la proxy ya no se tendrá que solicitar el recurso al servidor de origen y un aspecto muy interesante como es el balanceo de carga. La proxy puede aplicar políticas para en el caso de tener más de un servidor ofreciendo los datos, determine cuál de estos está más sobrecargado para delegar las peticiones al otro servidor.

17 Impactos de los diferentes tipos de conexión

Hasta hace unos años la mayor parte de la actividad web se efectuaba desde un ordenador con una conexión rápida, buenos recursos, que provocaba menos esfuerzos en las optimizaciones web, pero el mundo móvil ha significado una revolución.

Los dispositivos móviles tienen menos recursos, además son menos potentes que un ordenador de escritorio, no podemos esperar que el dispositivo móvil haga nuestro trabajo, cuando comentamos el tema de la compresión de las imágenes comentamos que no basta con comprimir estas, sino ofrecer en la versión móvil, imágenes más adecuadas para evitar que el dispositivo tenga que hacer el trabajo de reducir el tamaño, invirtiendo tiempo y batería.

Por otro lado, tenemos un tipo de conexión 3G y 4G, la cuales no solo son más lentas que una red cableada o Wi-FI, sino que cuestan dinero si no tienes planes ilimitados. Una web lenta puede provocar una mala experiencia por parte del usuario y que al notar el consumo excesivo de sus datos abandone la página.

Para comprobar experimentalmente a qué reto se enfrenta nuestra página web podemos analizar la velocidad de carga con diferentes perfiles, por defecto DevTools únicamente incluye 3G y 3G rápido, pero tenemos la posibilidad de añadir nuevos perfiles, desde Configuración:Limitación, en mi caso he añadido una conexión 4G y Wi-FI.

En la siguiente tabla se muestra los tiempos que le tardará en cargar la página, la primera vez, a un usuario que se conecte desde los diferentes tipos de conexión, sin tener en cuenta caché.

Diferentes tipos de conexión	
Tipo de conexión	**Tiempo de carga**
WI-FI	1,34 segundos
4G/LTE Regular	7,59 segundos
3G Rápido	19,61 segundos
3G Lento	60 segundos

18 Resultados

Los resultados son los mostrados en los diferentes apartados específicos, ya que es más útil analizar el impacto de las tecnologías por separado, a modo de resumen en la siguiente gráfica (Figura 21) expreso de las tecnologías mostradas, cuáles tienen un mayor impacto, de una manera muy visual y simplificada.

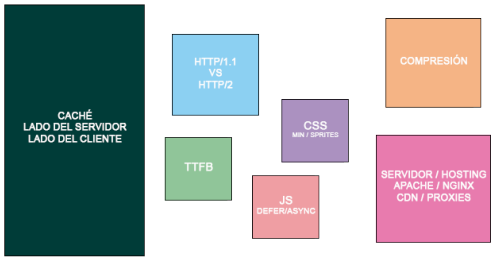

Fig. 21: Resultados

19 Conclusiones

En muchas ocasiones nos preguntamos ciertos conocimientos que adquirimos en la universidad para qué nos servirán, y a la hora de realizar proyectos como este, es cuando nos damos cuenta de la importancia de estar nutridos de conocimientos de distintas áreas, ya que en un futuro nos podemos encontrar en medio de proyectos donde requiramos conocimientos de todo tipo.

Me encuentro muy satisfecho de poner en un punto común mis conocimientos de Redes, donde me ha permitido tener una mayor visión de como interactúan los diferentes nodos en Internet como ocurre en las páginas webs, mis conocimientos de STW que me ha permitido trabajar sobre la aplicación web, mis conocimientos de arquitecturas de computadores que me han permitido ver que métricas son más interesantes y siempre adentrarme hasta encontrar el porqué de las cosas.

Este documento refleja las tecnologías más actuales y más interesantes a la hora de optimizar aplicaciones, siendo una receta para todos aquellos ingenieros que quieren aplicar estas técnicas. En el caso de seguir con el proyecto, utilizaría este documento para desarrollar una aplicación WPO para las plataformas más populares, que incluyera un panel intuitivo donde los usuarios menos experimentados puedan hacer optimizaciones de rendimiento en sus páginas webs.

Agradecimientos

Muchas gracias a Ramon Grau Sala por aceptar el reto y todos los consejos y mejoras recibidos por su parte durante el proyecto.

Referencias

[1] S. Koller. (2021) ¿qué es el wpo y cómo mejorar este factor clave para el posicionamiento seo? [Online]. Available: https://expeditedsecurity.com/blog/nginx-brotli/

[2] Cio-Wiki. (2022) Client server architecture. [Online]. Available: https://cio-wiki.org/wiki/Client_Server_Architecture

[3] Mozilla. (2022) Generalidades del protocolo http. [Online]. Available: https://developer.mozilla.org/es/docs/Web/HTTP/Overview

[4] W. stats. (2022) Wpo stats. [Online]. Available: https://wpostats.com/

[5] amnislabs. (2019) ¿qué servidor necesito? compartido, vps, dedicado o cloud. [Online]. Available: https://www.amnislabs.com/blog/que-servidor-necesito-compartido-vps-dedicado-o-cloud/

[6] S. Baidya. (2022) How to reduce ttfb in google page speed insights. [Online]. Available: https://www.cloudzat.com/reduce-ttfb-google-page-speed/

[7] Kinsta. (2022) How to reduce ttfb to improve wordpress page load times. [Online]. Available: https://kinsta.com/blog/ttfb/

[8] ——. (2021) What is http/2 – the ultimate guide. [Online]. Available: https://kinsta.com/learn/what-is-http2/

[9] R. Hodson. (2015) Http/2 for web developers. [Online]. Available: https://blog.cloudflare.com/http-2-for-web-developers/

[10] A. Krishnan. (2022) Nginx vs apache: Head to head comparison. [Online]. Available: https://hackr.io/blog/nginx-vs-apache

[11] J. Casares. (2022) Web performance. [Online]. Available: https://www.webperformance.es/codigo-css/

[12] W3Schools. (2022) Css image sprites. [Online]. Available: https://www.w3schools.com/css/css_image_sprites.asp

[13] F. Copes. (2018) Efficiently load javascript with defer and async. [Online]. Available: https://flaviocopes.com/javascript-async-defer/

[14] U. E. N. Bill Scott Director. (2008) Improving netflix performance. [Online]. Available: https://cdn.oreillystatic.com/en/assets/1/event/7/Improving%20Netflix%20Performance%20Presentation.pdf

[15] M. MacCana. (2021) 'you can't use brotli for dynamic content'. [Online]. Available: https://expeditedsecurity.com/blog/nginx-brotli/

[16] A. Joseph. (2021) Server-side caching vs. client-side caching. [Online]. Available: https://edgemesh.com

[17] CloudFlare. (2022) ¿qué es un proxy inverso? [Online]. Available: https://www.cloudflare.com/es-es/learning/cdn/glossary/reverse-proxy/