UNIVERSIDAD AUTÓNOMA DE YUCATÁN

FACULTAD DE MATEMÁTICAS UADY

LICENCIATURA EN INGENIERÍA DE SOFTWARE

OPTIMIZACIÓN DE APLICACIÓN WEB

ENTREGA FINAL

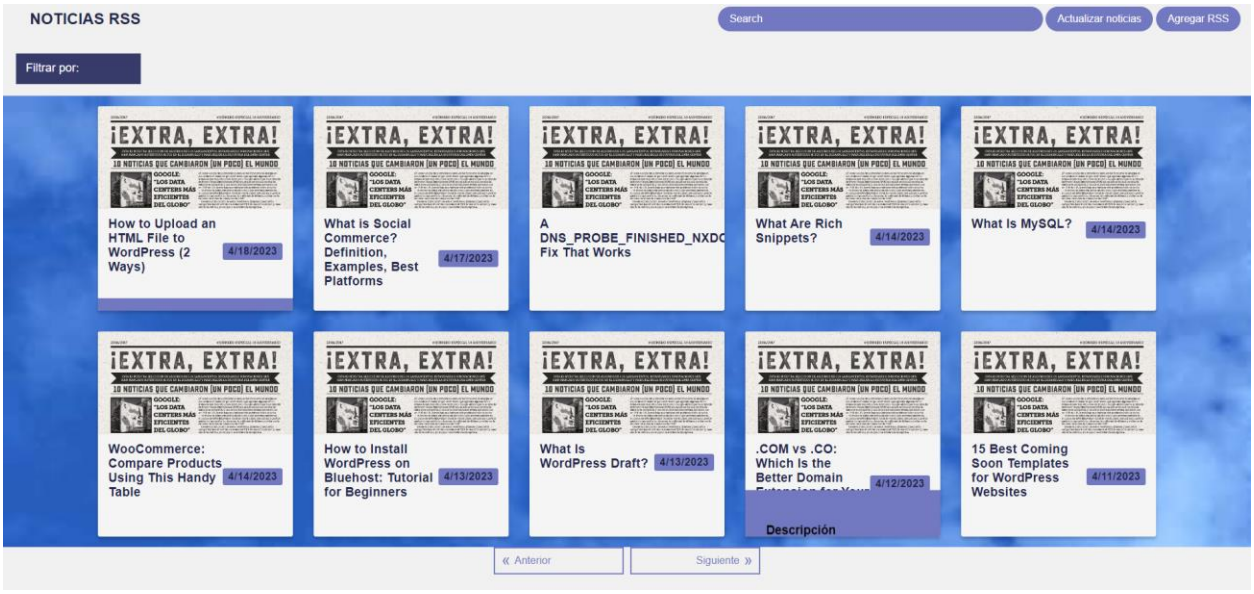
INTEGRANTES:

18 DE MAYO DE 2023

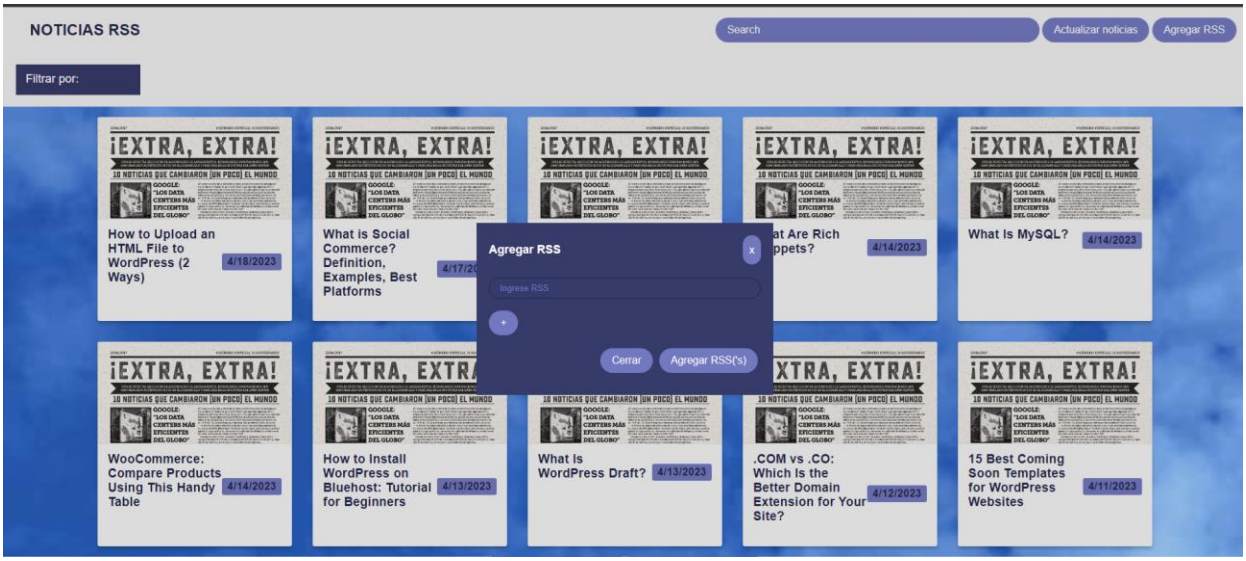
1. **Introducción.**

El siguiente proyecto es un lector de RSS, en el cual el usuario puede ingresar sus RSS favoritos y ver las noticas destacables del mismo.

Pantalla principal de la aplicación:



Pantalla donde se agrega el RSS:



Presenta un menú en el cual se puede filtrar por categorías y de igual forma un buscador

El objetivo de realizar el programa es ver con datos y que nosotros cómo desarrolladores podamos apreciar la diferencia entre una aplicación web optimizada y una que no lo esté.

Podemos ver que la optimización de aplicaciones web es de suma importancia, ya que mejora significativamente la experiencia del usuario y el rendimiento general de la aplicación. Al optimizar, se reducen los tiempos de carga y respuesta, lo que aumenta la eficiencia y la satisfacción del usuario. Además, una aplicación web optimizada tiene mayor probabilidad de aparecer en los primeros resultados de búsqueda, lo que aumenta su visibilidad y alcance. Al implementar técnicas de optimización, como la compresión de archivos, la reducción del tamaño de las imágenes y el uso eficiente de la caché, se logra una mayor velocidad y menor consumo de recursos, lo que beneficia tanto a los usuarios como a los propietarios de la aplicación.

2. **Arquitectura y tecnologías de la aplicación.**

El proyecto fue realizado bajo la arquitectura SPA (Single Page Application), para esto elaboramos una aplicación con la lógica de FrontEnd y BackEnd separadas, siendo estas comunicados por un medio de transferencia de datos, en este caso JSON.

Las tecnologías que se utilizaron para la elaboración del proyecto son:

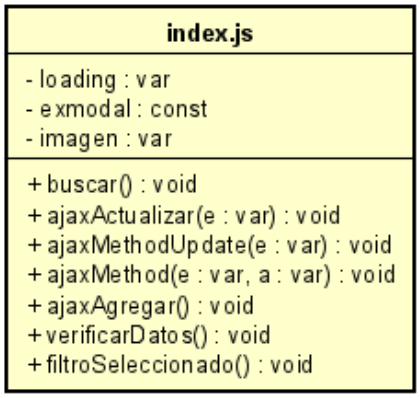
FrontEnd:

a) JavaScript: Es un lenguaje de programación que se ejecuta en el navegador del cliente y permite agregar interactividad y funcionalidad a las páginas web.
b) CSS: Es un lenguaje de diseño utilizado para controlar la apariencia y el formato de los elementos en una página web. Permite definir el estilo, el diseño y la presentación visual

de los elementos HTML, como el color, la tipografía, el tamaño, la posición y otros aspectos visuales.

c) HTML: Es el lenguaje de marcado utilizado para crear la estructura y el contenido de una página web.

A continuación, se presenta el diagrama UML que representa el FrontEnd:

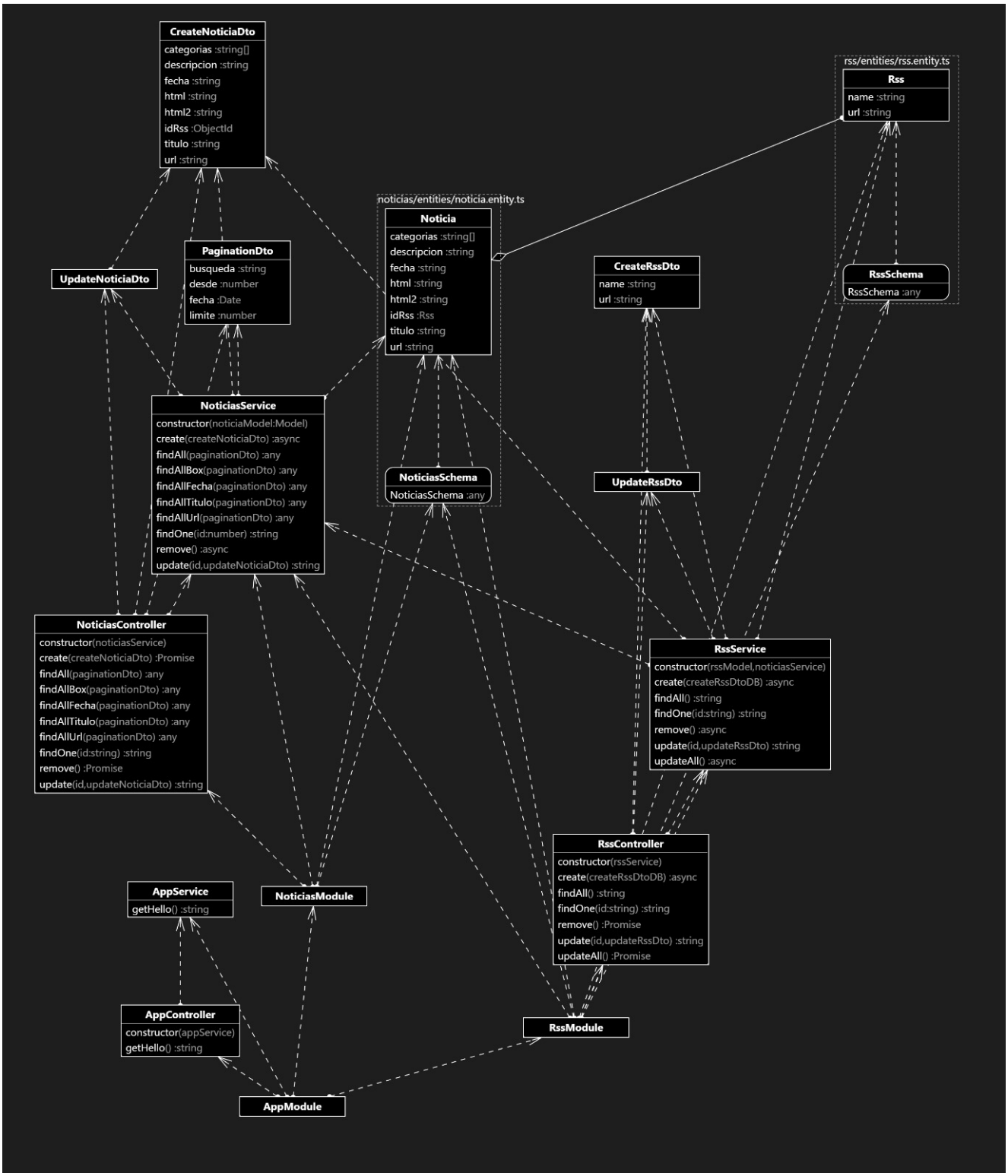


Para el lado del BackEnd estas son las tecnologías utilizadas:

a) NestJs: Es un framework de desarrollo de aplicaciones backend en Node.js, basado en TypeScript. Está diseñado para crear aplicaciones escalables y modularizadas siguiendo el patrón de arquitectura MVC (Modelo-Vista-Controlador). Nest.js combina elementos de otros frameworks populares, como Angular, para proporcionar una estructura organizada y fácil de mantener.

b) MongoDB: es una base de datos NoSQL (No solamente SQL) que se caracteriza por ser orientada a documentos. A diferencia de las bases de datos relacionales tradicionales, MongoDB almacena los datos en documentos flexibles en formato BSON (Binary JSON), lo que permite una estructura dinámica y escalable. MongoDB es altamente escalable y distribuible, lo que lo hace adecuado para aplicaciones con grandes volúmenes de datos y altas demandas de rendimiento.

A continuación, se presenta el diagrama UML que representa en BackEnd:


CreateNoticiaDto
categorias :string[]
descripcion :string
fecha :string
html :string
html2 :string
idRss :ObjectId
titulo :string
url :string
rss/entities/rss.entity.ts
Rss
name :string
url :string
noticias/entities/noticia.entity.ts
Noticia
categorias :string[]
descripcion :string
fecha :string
html :string
html2 :string
idRss :Rss
titulo :string
url :string
PaginationDto
busqueda :string
desde :number
fecha :Date
limite :number
UpdateNoticiaDto
CreateRssDto
name :string
url :string
RssSchema
RssSchema :any
NoticiasService
constructor(noticiaModel:Model)
create(createNoticiaDto) :async
findAll(paginationDto) :any
findAllBox(paginationDto) :any
findAllFecha(paginationDto) :any
findAllTitulo(paginationDto) :any
findAllUrl(paginationDto) :any
findOne(id:number) :string
remove() :async
update(id,updateNoticiaDto) :string
NoticiasSchema
NoticiasSchema :any
UpdateRssDto
NoticiasController
constructor(noticiasService)
create(createNoticiaDto) :Promise
findAll(paginationDto) :any
findAllBox(paginationDto) :any
findAllFecha(paginationDto) :any
findAllTitulo(paginationDto) :any
findAllUrl(paginationDto) :any
findOne(id:string) :string
remove() :Promise
update(id,updateNoticiaDto) :string
RssService
constructor(rssModel,noticiasService)
create(createRssDtoDB) :async
findAll() :string
findOne(id:string) :string
remove() :async
update(id,updateRssDto) :string
updateAll() :async
RssController
constructor(rssService)
create(createRssDtoDB) :async
findAll() :string
findOne(id:string) :string
remove() :Promise
update(id,updateRssDto) :string
updateAll() :Promise
AppService
getHello() :string
NoticiasModule
AppController
constructor(appService)
getHello() :string
RssModule
AppModule

3. Evaluación diagnóstica del desempeño inicial de la aplicación.

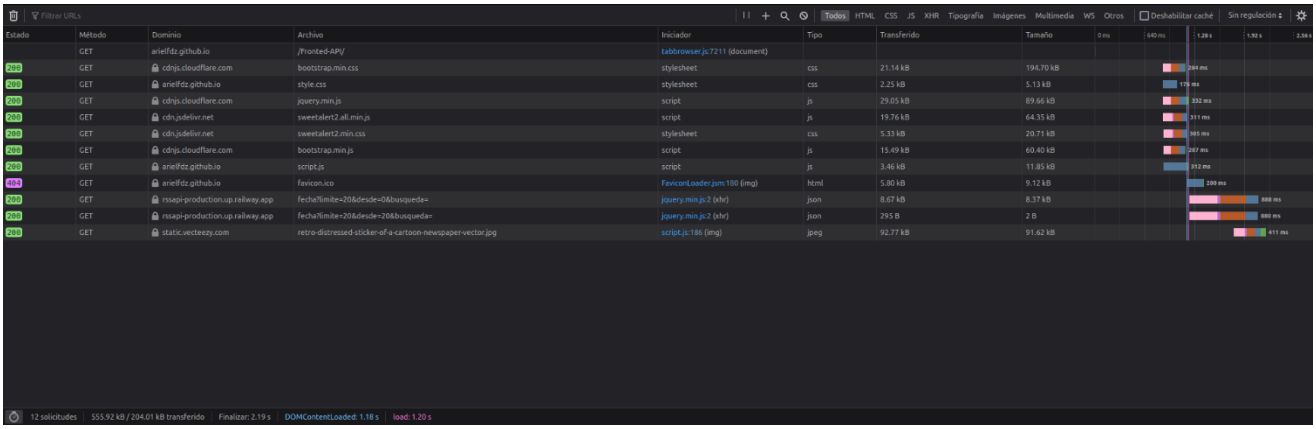
La aplicación fue realizada pensado en una sola ventana por lo que solamente haremos la comparación de esta.

Primero evaluaremos la aplicación sin realizar ninguna optimización, veremos la carga de la web por primera vez y con cache

Carga de la página por primera vez:

	Volumen de transferencia	Tiempo de transferencia
Página principal	555.92kb/204.01kb	1.2s

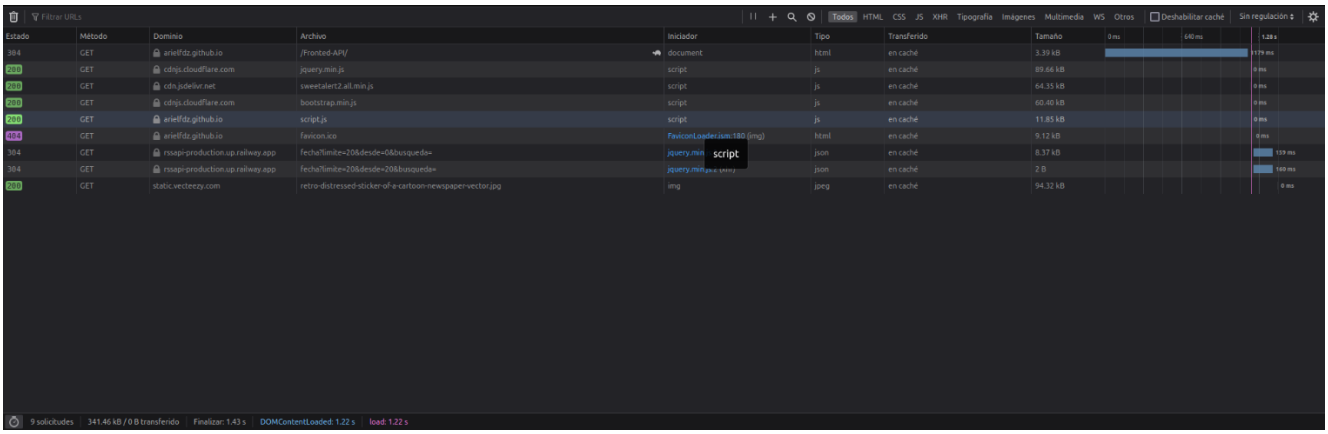
Transferencia de datos



Carga de la página con cache:

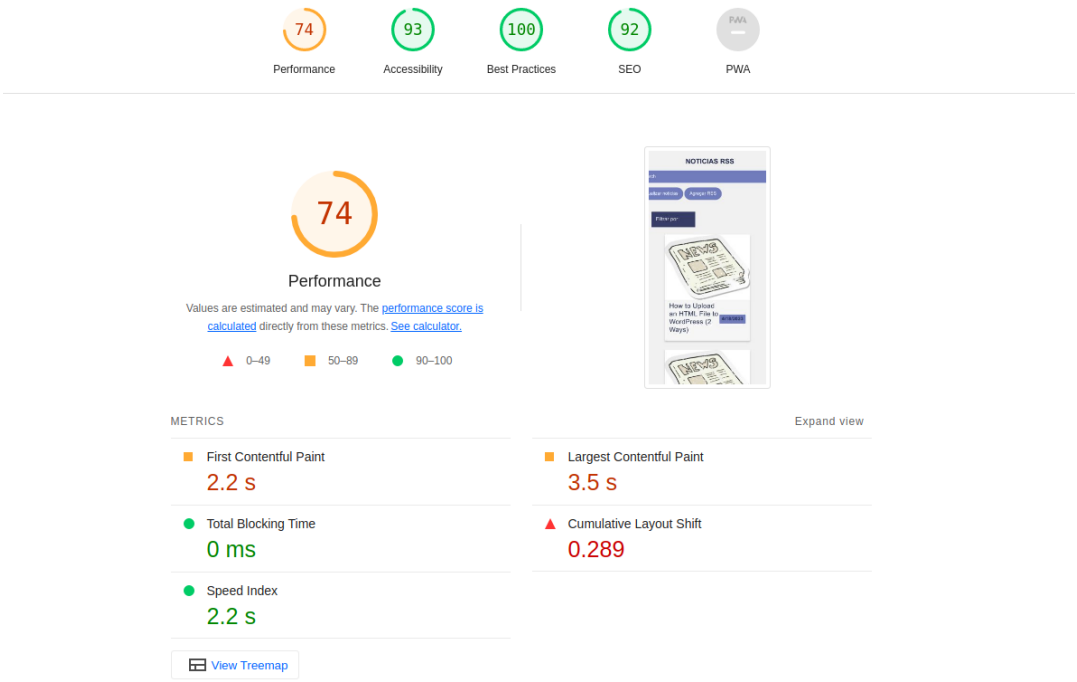
	Volumen de transferencia	Tiempo de transferencia
Página principal	341.46kb/0kb	1.22s

Transferencia de datos:

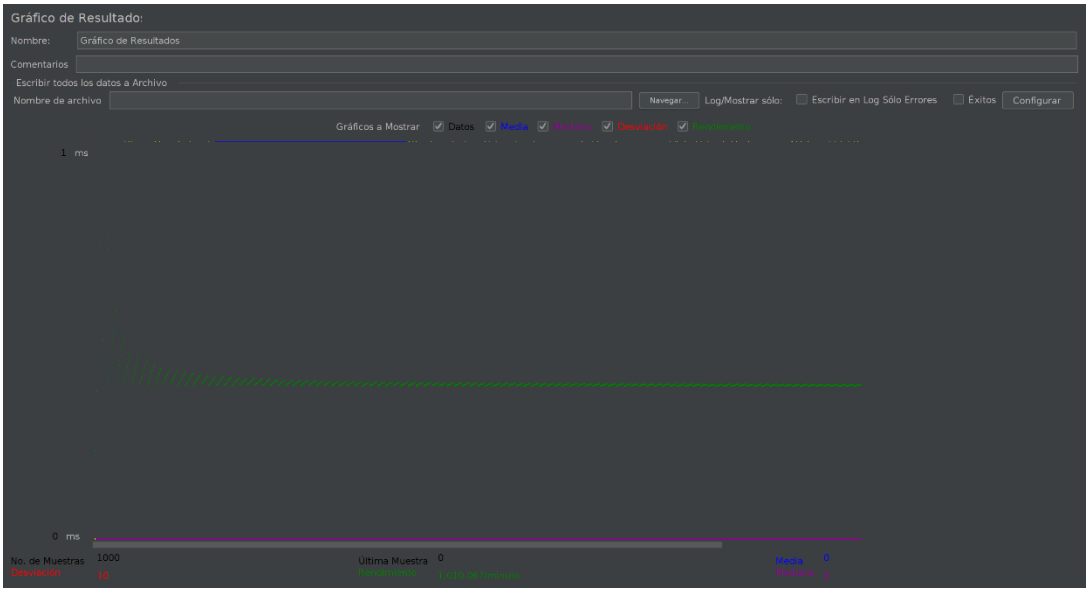


Datos generales

Lighthouse



Jmeter 100 usuarios un minuto



Podemos apreciar que la aplicación por el simple hecho de estar en la cache reduce un 39% de transferencia de información entre el cliente y el servidor. De igual forma basándonos en los otros resultados vemos que mínimo existen 2s de retraso en la carga al momento de mostrar el contenido

4. **Propuestas de mejora aplicadas.**

Las mejoras que se aplicaron para mejorar la optimización de la aplicación web son:

FrontEnd:

1) Minificación: Es el proceso de reducir el tamaño de un archivo de código eliminando espacios en blanco, comentarios y caracteres innecesarios, con el objetivo de mejorar el rendimiento y reducir los tiempos de carga en aplicaciones web.
2) CDN: Se cambiaron CDN de Bootstrap que alentaban la carga de la aplicación web

Backend:

1) Caché: Se implemento la caché del lado del servidor utilizando librerías cómo nestjs/cache-manager cache-manager
2) ORM: Se utilizan ORM por lo cual las consultas están optimizadas evitando escribir sentencias las cuales pueden presentar lentitud en consultas si son realizadas incorrectamente.

5. **Contraste de resultados.**

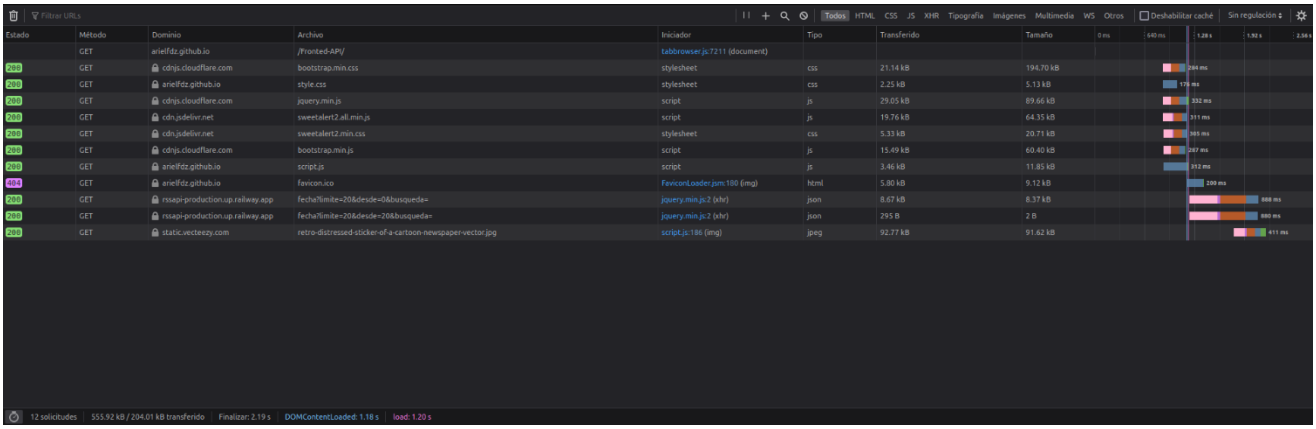
Carga de la página por primera vez

	Volumen de transferencia	Tiempo de transferencia
Página principal	555.92kb/204.01kb	1.2s
Página principal optimizada	452.98kb/323.46kb	673ms

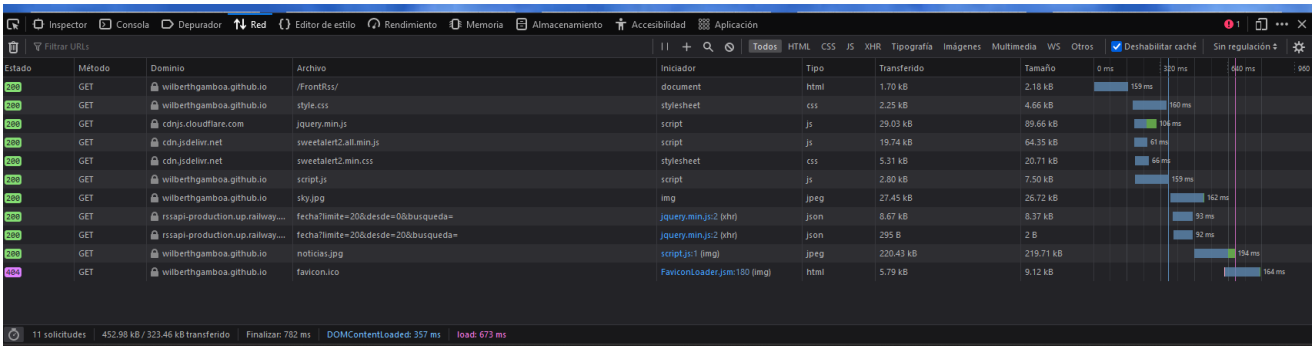
Analizando tenemos que después realizar tareas de optimización se presenta una mejora del 11% al momento de trasferir los datos, mientras que el tiempo de transferencia se ve beneficiado en un 44%

Transferencia de datos sin caché:

Página principal sin optimizar



Página principal optimizada



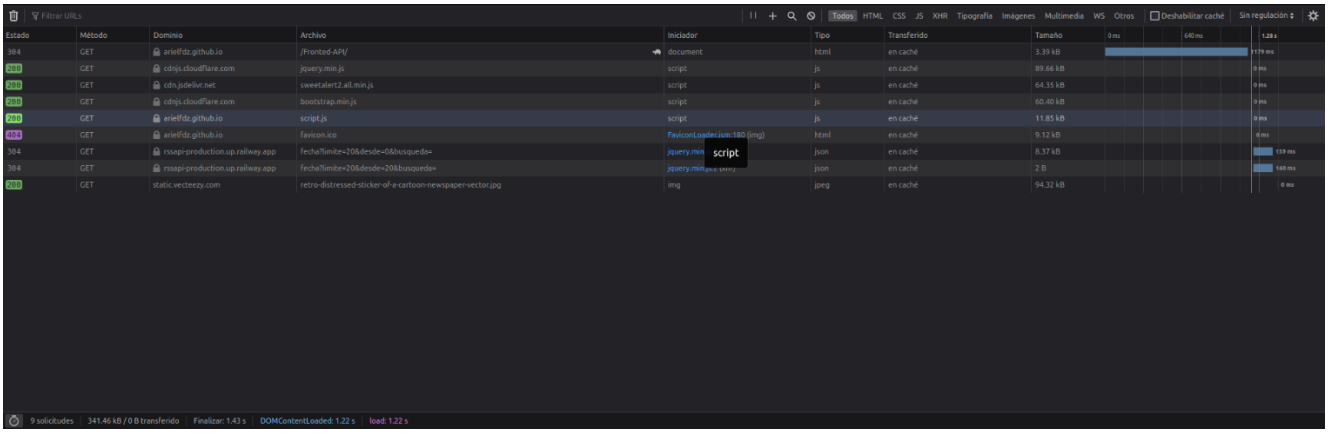
Carga de la página con cache:

	Volumen de transferencia	Tiempo de transferencia
Página principal	341.46kb/0kb	1.22s
Página principal optimizada	239.8kb/0kb	191ms

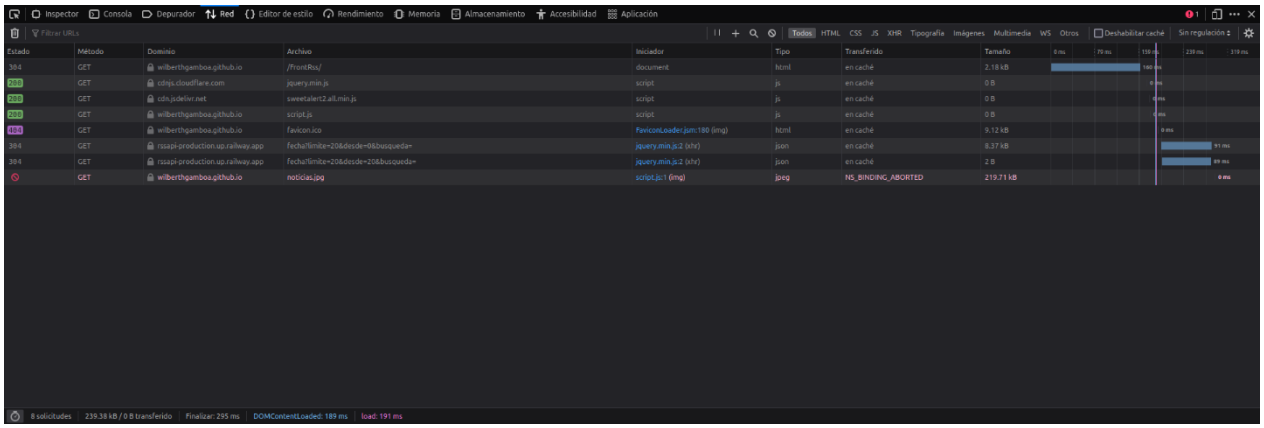
Analizando tenemos que después realizar tareas de optimización teniendo la caché, se presenta una mejora del 30% al momento de trasferir los datos, mientras que el tiempo de transferencia se ve beneficiado en un 85%

Transferencia de datos con caché:

Página principal

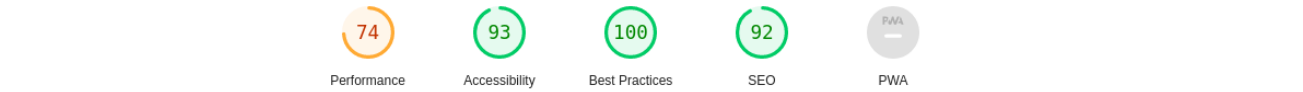


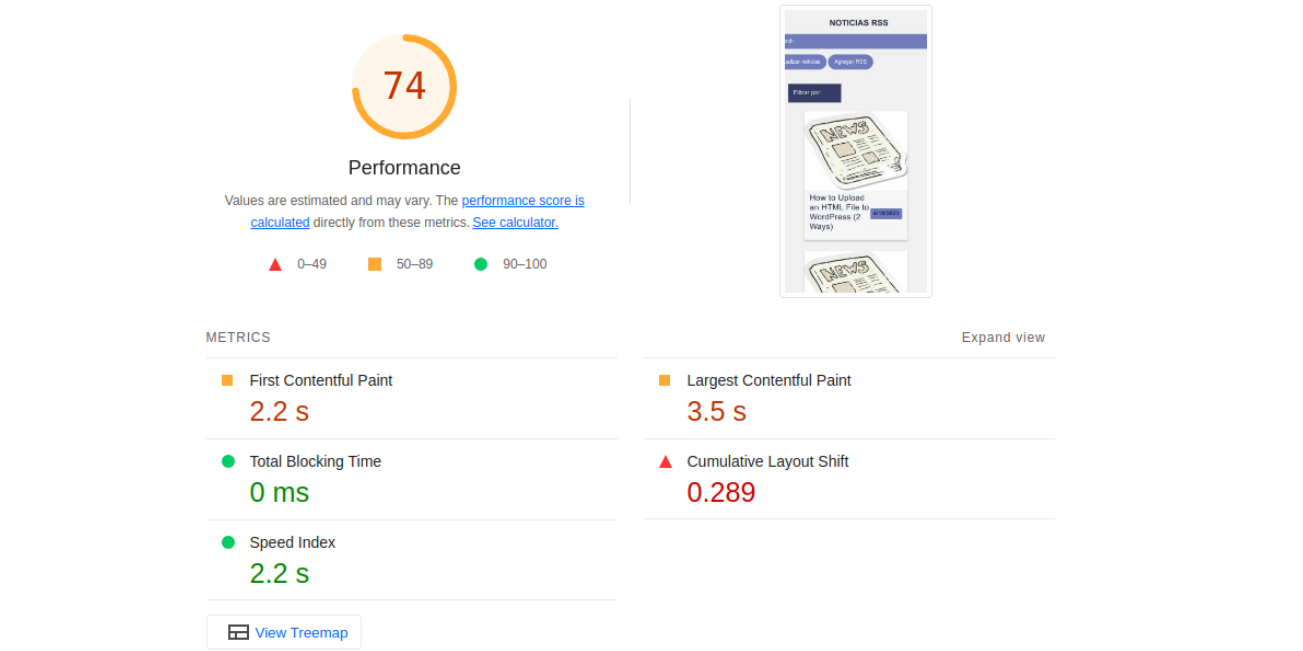
Página principal optimizada:



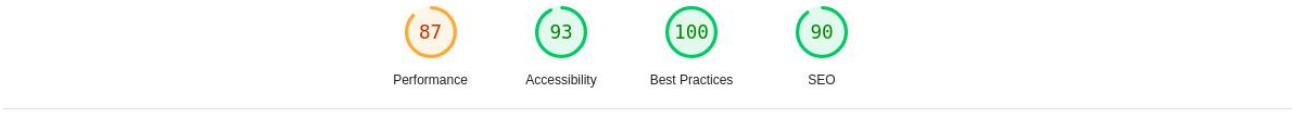
Datos generales

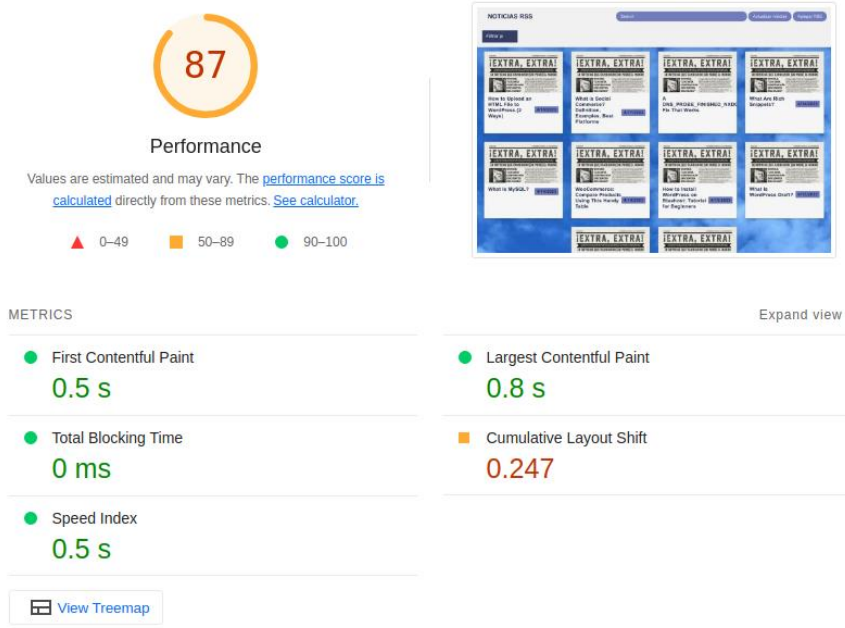
Lighthouse página sin optimizar

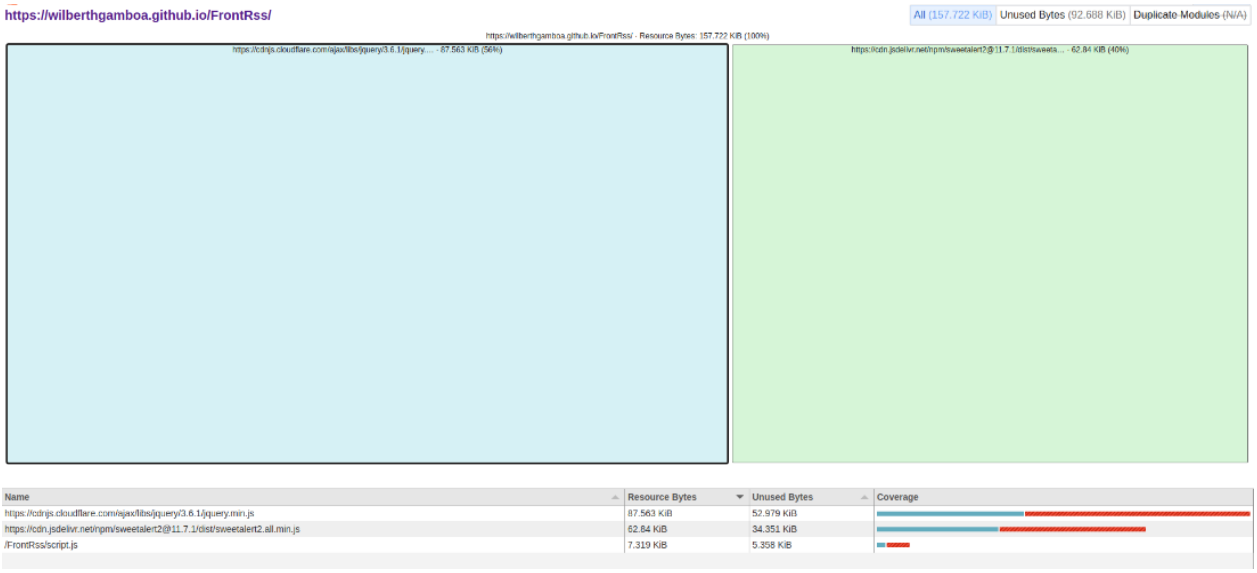




Lighthouse página optimizada

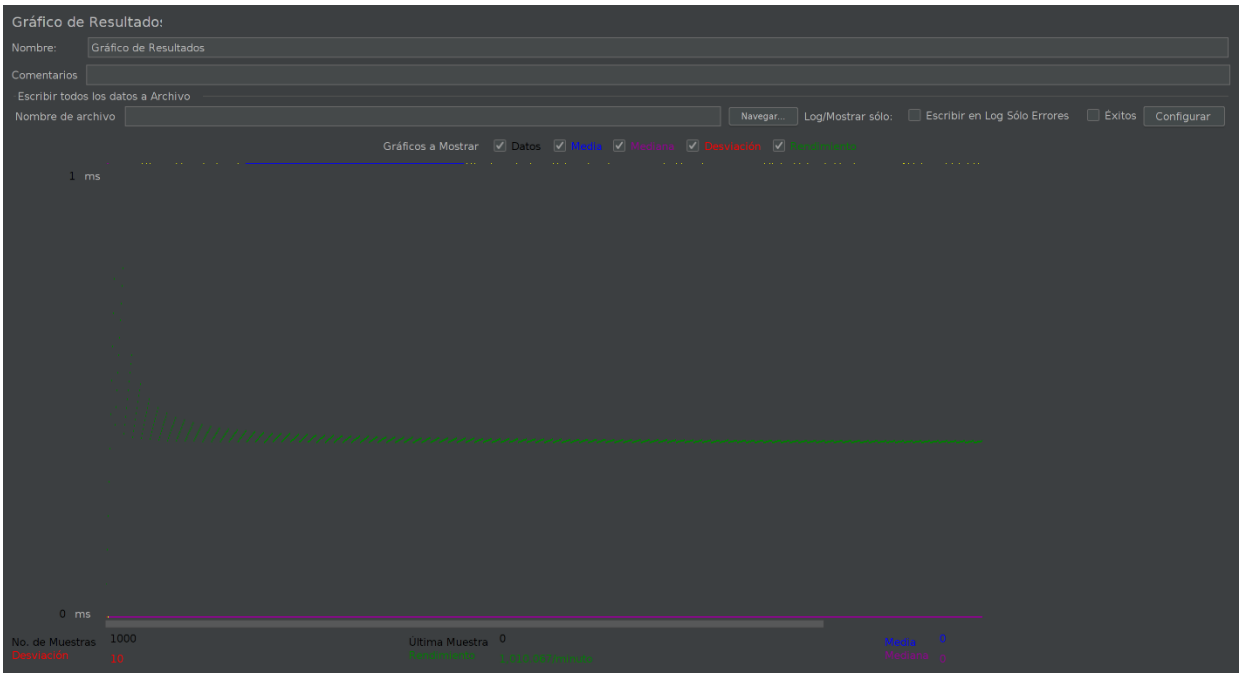




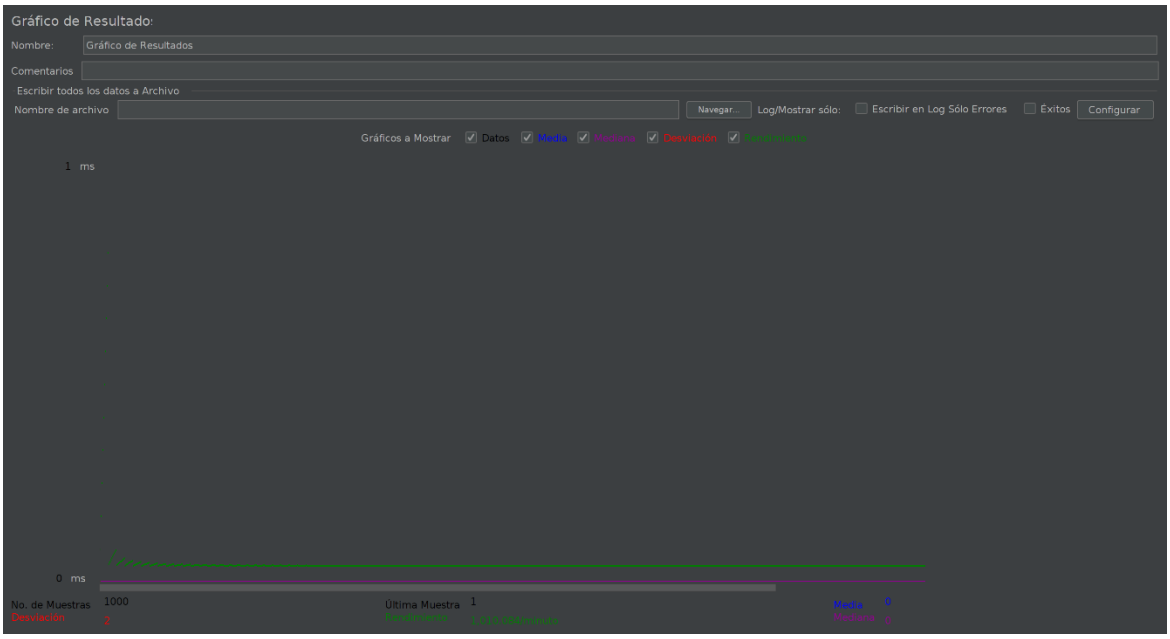


Jmeter 100 usuarios un minuto

Página sin optimizar:



Página optimizada:



Conclusiones

El documento abarca la explicación de la aplicación web desarrollada, se procura no realizar ninguna tarea de optimización debido a que este es el objetivo de la materia, ya que algunos frameworks presentan optimizaciones, es destacable que, aunque ya vienen optimizados, todavía se puede seguir realizando modificaciones en el código cómo en las configuraciones para optimizar mucho más las aplicaciones que se desarrollen.

Se aprecia que después de aplicar la optimización existe una mejora en el rendimiento de la aplicación, esto es visible en el programa de Google Lighthouse, que tomaría cómo punto de partida principal para ver de una forma sencilla si nuestra aplicación está realmente funcionando de forma adecuada, recordemos que Google es uno de los principales buscadores, de éste depende que nuestra aplicación web sea un éxito o un rotundo fracaso, dada la herramienta mencionada podemos ver que el rendimiento mejoró 2s.

Lo importante es recordar la optimización es un tema importante, esta aplicación aunque fue “pequeña” se aprecia que existen un de un 10% a 15% de mejora, ahora supongamos que una aplicación mucho más grande esa mejora de 10kb puede significar 10mb en otra aplicación, por lo cual no debemos subestimar la optimización en nuestras aplicaciones.

Referencias

ITBLOGSOGETI (2014) Single Page Applications. IT:Blog. Recuperado de: https://itblogsogeti.com/2014/06/10/single-page-applications-roberto-bermejo-sogeti/

Mozilla. (s. f.). JavaScript. Recuperado de https://developer.mozilla.org/en-US/docs/Web/JavaScript

Mozilla. (s. f.). HTML: HyperText Markup Language. Recuperado de https://developer.mozilla.org/en-US/docs/Web/HTML