PEC 3: Rendimiento Web

Enlace a página: https://blissful-blackwell-d8a212.netlify.app/

Enlace a repositorio: https://github.com/ymartinezac/pec1-travel-portal

Tabla comparativa de métricas antes y después de implementación de optimizaciones

Desktop

Título	URL	Tiempo de carga (medio)		Peso total		Peso transferido		Cantidad de recursos	
		Inicial	Final	Inicial	Final	Inicial	Final	Inicial	Final
Inicio	https://blissful-blackwell-d8a212.netlify.app/	7.18	4.812	715.59 KB	606.35 KB	553.05 KB	426.31 KB	28	21
Categoría	https://blissful-blackwell-d8a212.netlify.app/buscar-tours	2.996	2.718	380.40 KB	352.18 KB	226.77 KB	200.13 KB	23	16
Detalle	https://blissful-blackwell-d8a212.netlify.app/tours/3	5.696	5.668	645.18 KB	626.55 KB	483.77 KB	475.25 KB	24	16
Presentación	https://blissful-blackwell-d8a212.netlify.app/presentacion	2.146	1.954	325.79 KB	337.22 KB	163.63 KB	121.67 KB	20	13

Móvil

Título	URL	Tiempo de carga (medio)		Peso total		Peso transferido		Cantidad de recursos	
		Inicial	Final	Inicial	Final	Inicial	Final	Inicial	Final
Inicio	https://blissful-blackwell-d8a212.netlify.app/	4.928	2.83	608.78 KB	210.55 KB	454.85 KB	173.61 KB	29	19
Categoría	https://blissful-blackwell-d8a212.netlify.app/buscar-tours	2.912	2.35	380.40 KB	323.26 KB	226.81 KB	176.22 KB	23	15
Detalle	https://blissful-blackwell-d8a212.netlify.app/tours/3	5.664	2.346	645.18 KB	312.23 KB	491.26 KB	164.49 KB	26	14
Presentación	https://blissful-blackwell-d8a212.netlify.app/presentacion	2.454	1.712	325.79 KB	78.31 KB	171.10 KB	67.30 KB	20	12

A primera vista se puede observar que en todas las métricas mejoraron al realizarse las optimizaciones. Los tiempos de carga (en segundos) disminuyeron, el peso total de la página, el peso transferido y la cantidad de recursos también. Se pueden observar las mejoras más drásticas en las páginas de inicio (desktop y móvil) y la página de detalle (móvil). Estas páginas eran las que contenían más elementos y unos tiempos de carga entre 5 y 7 segundos, por ende, pude implementar mayor número de optimizaciones y tenía un margen más amplio para mejorar. Aun cuando en las otras páginas había menos optimizaciones por hacer, se vieron impactadas positivamente.

Los cambios implementados incluyen la carga de imágenes y componentes de React mediante uso de lazy loading, uso de módulos de javascript y carga asíncrona de hojas de estilos. También he unido las múltiples hojas de

estilos en una sola. Por costumbre de trabajar antes en Angular, en este proyecto estaba manteniendo las hojas de estilo separadas una para cada componente, para facilidad de encontrar los estilos de cada componente. Cuando se realiza el build de Angular, todas estas hojas se combinaban en una sola y así el servidor realiza una sola llamada en lugar de varias. Entiendo que bundlers como Webpack también permiten unir las hojas de estilo, pero en este caso no encontré como hacerlo con Parcel, así que me dispuse a hacerlo manualmente. Además, he creado una hoja de estilos para la versión móvil y así no cargar estilos innecesarios.

Quiero hacer la aclaración de que dejé de lado el formato Tablet para esta PEC por limitaciones de tiempo. Pasé mucho trabajo haciendo el diseño responsive y durante esta PEC he caído en cuenta que se debe a que no añadí el tag `<meta name="viewport">` al index.html. Antes de esto, los breakpoints variaban demasiado a través de dispositivos y se me hizo difícil dar con las dimensiones y tamaños de font que se ajustaran a la variedad de dispositivos disponibles. Encima de esto estuve trabajando hasta hace poco en un portátil con una pantalla de muy alta resolución (casi 4k), y no ha sido hasta que cambié de portátil a uno con pantalla Full HD que he caído en cuenta lo mal que se adaptaba el diseño. Intenté añadir un dispositivo de laptop a Firefox para hacer estas comparaciones, pero no ofrece la opción de añadir la densidad de pixeles, y con las dimensiones que he añadido no me dió resultados muy fidedignos. Esto lo seguiré explorando en futuros proyectos.

```
import React, { lazy } from "react";  7.2K (gzipped: 2.9K)
const Snorkeling = lazy(() => import('../snorkeling/snorkeling'));
const InfoGeneral = lazy(() => import('../info-general/info-general'));
```

Ilustración 1 – Código para hacer lazy loading en componentes

Implementé el lazy loading tanto en imágenes como en componentes de React. Originalmente, tenía un componente "Home" para la página de Inicio que era bastante largo. Dividí el contenido en varios componentes para poder hacer provecho de la carga asíncrona de estos. Implementé el lazy loading de imágenes utilizando la librería react-lazy-load-image-component ya que utilizar la propiedad loading=lazy no funciona en React. Esta librería tiene como ventaja el uso de thresholds para disparar la carga de la imagen cuando esta se encuentre a la distancia en pixeles determinada en el threshold.

```
import { useInView, InView } from 'react-intersection-observer';  4.5K (gzipped: 1.9K)

const Snorkeling = () => {
    const [ref, inView, entry] = useInView({
        threshold: 0,
    });
    return (
        <section ref={ref} className={inView ? "snorkel visible" : "snorkel invisible"} >
                <h1>Sumérgete en nuestros <span>snorkeling tours</span></h1>
                <LazyLoadImage className="snorkel-svg" src={mask} alt="snorkel mask" width="275
```

Ilustración 2 - Código para detectar si componente se encuentra en el viewport

He implementado el lazy loading de manera "forzada" en una imagen de fondo que se añade con la propiedad de CSS background-image. Digo forzada porque no hay manera directa de implementar lazy loading en imágenes cargadas en el CSS, en cambio lo que he hecho es que inicialmente, la propiedad de CSS background-image no carga ninguna imagen, tal como se observa en la Ilustración 3. He utilizado la librería react-intersection-observer, como se puede observar en la Ilustración 2, para detectar cuándo el componente en el que se encuentra esa imagen entre al viewport. Una vez lo hace, se añade al nombre de la clase la palabra 'visible'. Entonces el CSS para la clase 'visible' contiene el background-image con la imagen. De esta manera se difiere la carga de la imagen hasta que sea necesaria.

```
.snorkel{
    display: flex;
    flex-wrap: wrap;
    height: auto;
    color: var(--main-bg-color);
    padding: 0 12vw 0 12vw;
    background-repeat: no-repeat;
    background-size: cover;
    background-image: none;
}

.snorkel.visible {
    background-image: url(./public/images/underwater_medium.avif);
   }
```

Ilustración 3 - CSS para componente cuando está visible

Se puede observar en la ventana de Network de las herramientas de desarrollador que las imágenes con lazy loading no aparecen en la lista de recursos descargados hasta que se aparecen el viewport. Intenté aplicar lazy loading a las imágenes de las tarjetas de los viajes pero están muy cerca del viewport e igual se descargan en la renderización inicial de la página. Aún así, las que sí son afectadas por el lazy loading han hecho que disminuya considerablemente el tamaño de la página de inicio y su vez el tiempo de carga. Las demás páginas no contienen imágenes debajo del viewport inicial.

Mi aplicación, al ser hecha en React, lleva el HTML dentro de los scripts de Javascript mismo, así que en lugar de renderizar el HTML primero y cargar asíncronamente los scripts, lo que he hecho es hacer lazy loading de componentes de React. Funciona muy similar a las imágenes, pero requiere de la utilización de un componente "Suspense" que es el contenido de "fallback" para cuando aún no ha cargado el componente. Un potencial problema de esto es que el contenido de fallback es de un tamaño distinto al del componente principal que hará que se muevan las cosas de sitio si no cargan lo suficientemente rápido. Esto puede afectar la métrica de Cumulative Layout Shift y afectar la experiencia de usuario ya que es desagradable para los usuarios ver el contenido moviéndose inesperadamente.

He realizado carga asíncrona de la hoja de estilos porque ha sido sugerido en una de las recomendaciones de los informes del Page Speed Insights. Para este proyecto hago uso de Adobe Fonts que te crea automáticamente una hoja de estilos y te provee una url para accederla. El problema con Adobe Fonts es que a veces al activar fonts, si no eres cuidadoso, te añade fonts que no vas a utilizar (por ejemplo las versiones itálicas y bold de una familia de font que escojas). Tenía muchos de estos sin utilizar y los he eliminado del proyecto. También Adobe Fonts te permite

implementar el “swap”, es decir que aplica otro Font en lo que carga el principal. Escogí como “fallbcak” fonts del sistema (Verdana, Georgia) que no requieren que cargue un archivo para renderizarlos. Una vez más, el problema de esto es la posibilidad de Layout Shift una vez cargan los archivos de Font. No pude medir con precisión el impacto que estas optimizaciones tuvieron en la página pero puedo asumir que aportó a la mejora.

Además de la carga asíncrona de los fonts, creé dos hojas de estilos, como he mencionado anteriormente. Una contiene los estilos de la versión desktop y la otra los de la versión móvil. De esta manera, la aplicación llama la que sea necesaria de acuerdo con el dispositivo que está accediéndola. Esto es posible mediante el atributo de media donde se determina el breakpoint para que la aplicación tome la decisión. En la Ilustración 4 se puede ver que el breakpoint es el ancho de pantalla de 1100px.

```
<link rel="stylesheet" href="styles.css" media="(min-width:1100px)">
<link rel="stylesheet" href="styles-mobile.css" media="(max-width:1100px)">
```

Ilustración 4 - Media queries para hojas de estilos

Informe Primera Iteración

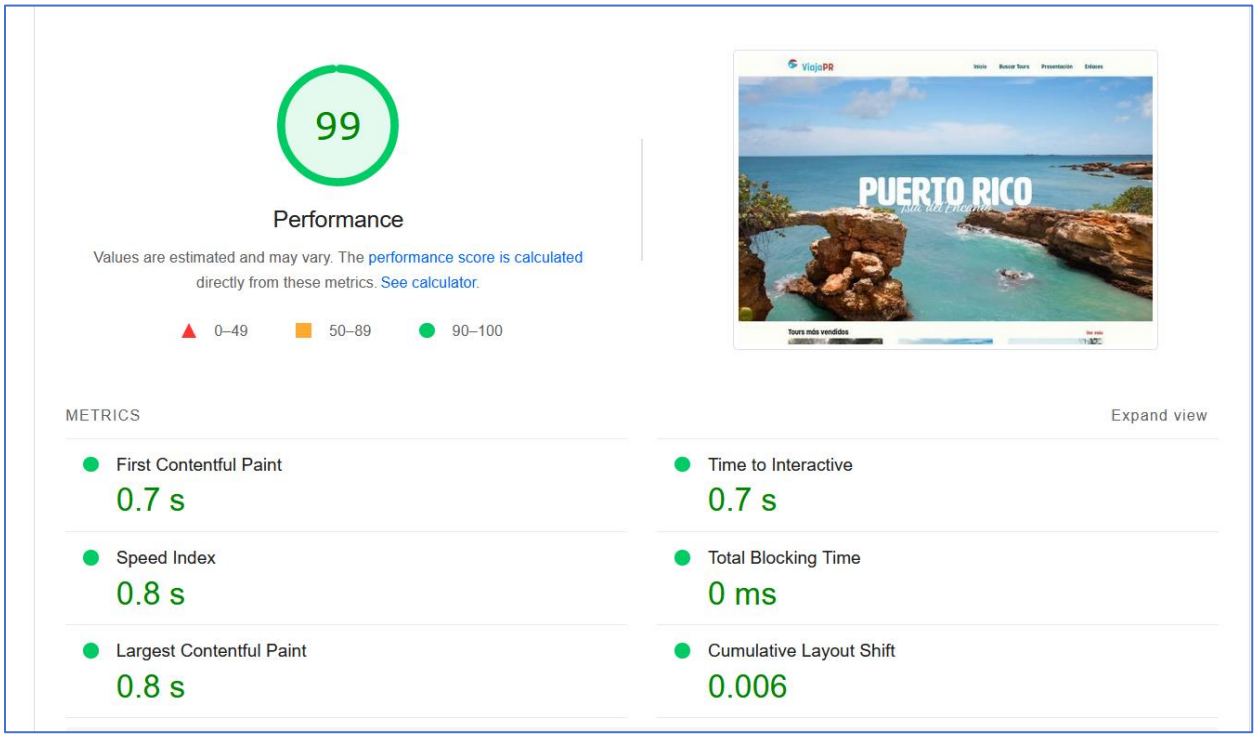

Ilustración 5 - Resultados Página de Inicio (desktop)

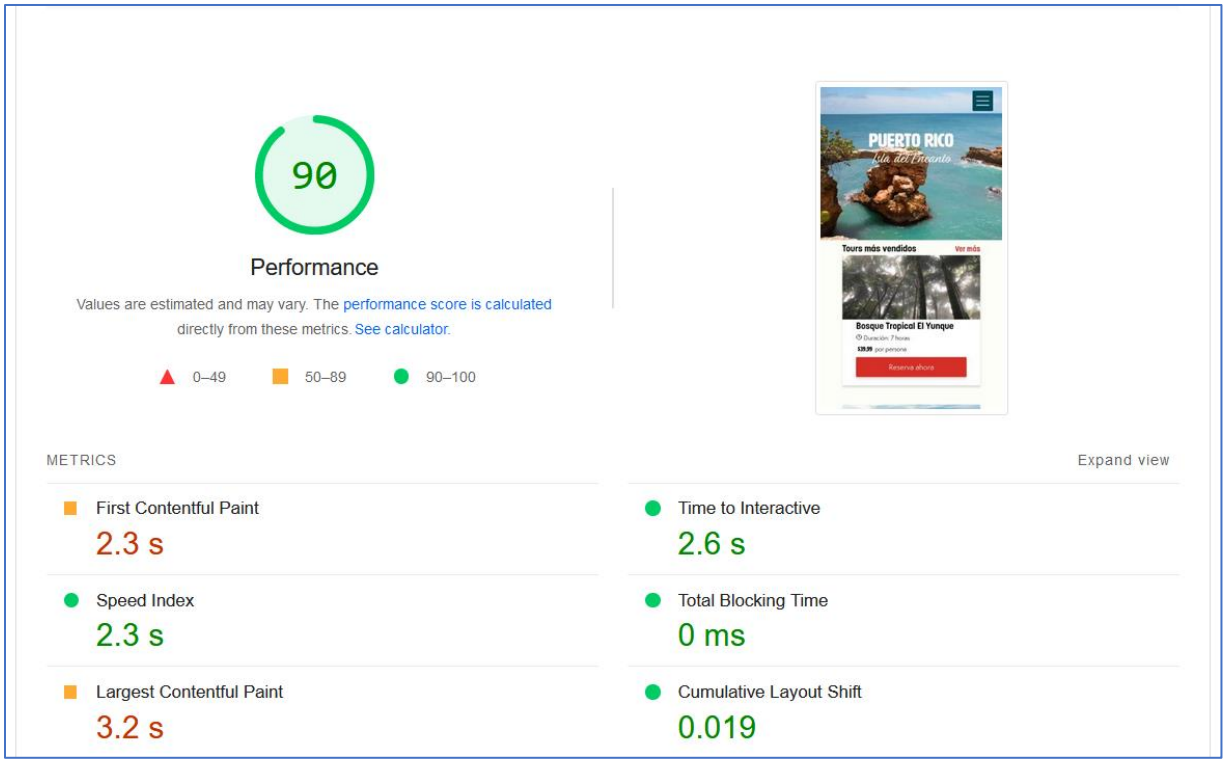

Ilustración 6 - Resultados página de Inicio (móvil)

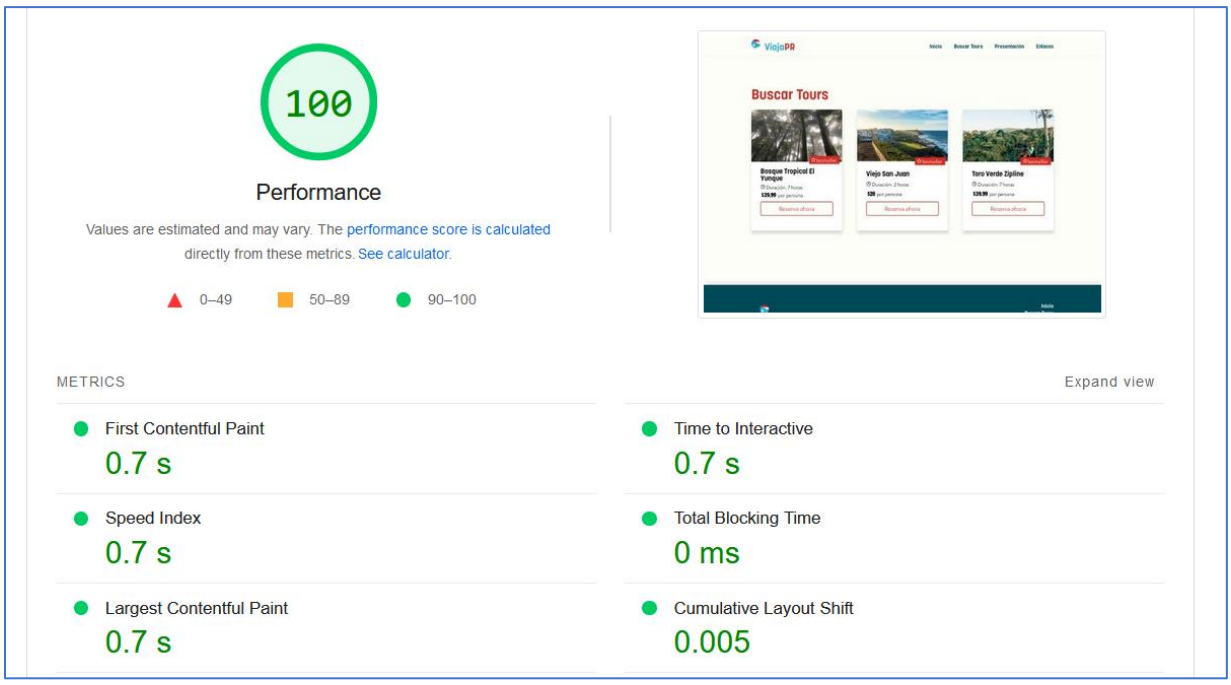

Ilustración 7 - Resultados página de Categoría (desktop)

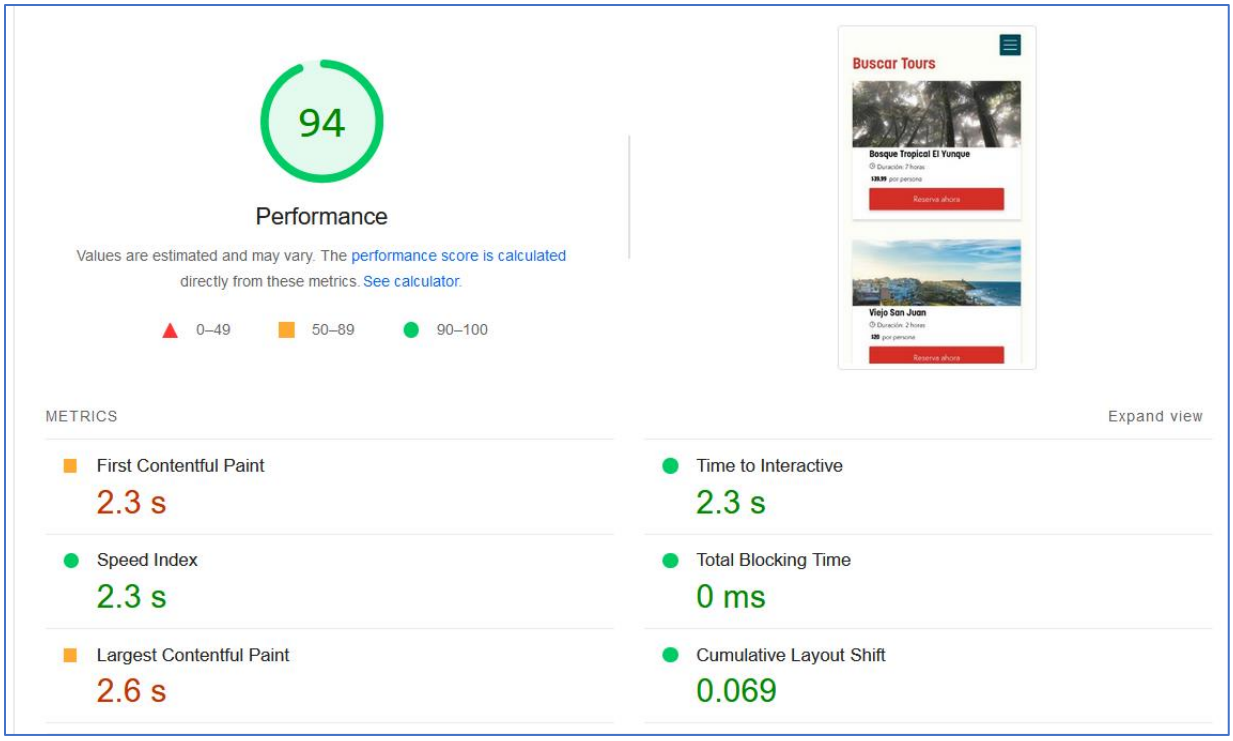

Ilustración 8 - Resultados página categoría (móvil)

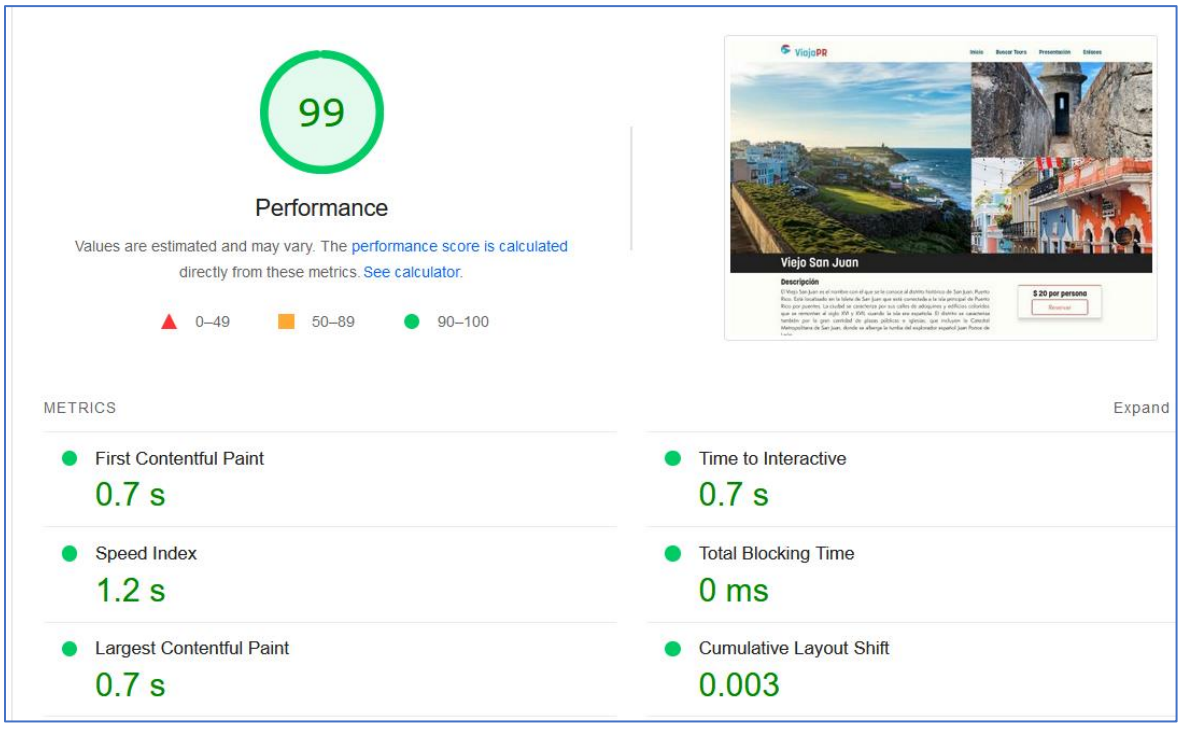

Ilustración 9 - Resultados página detalle (desktop)

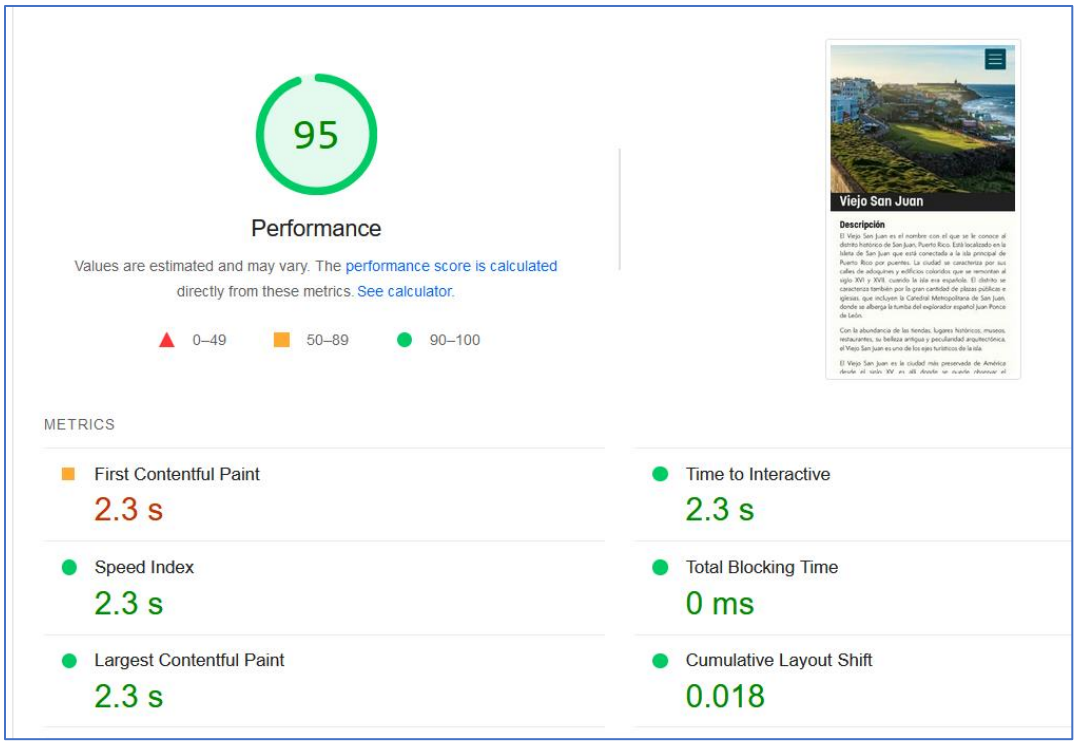

Ilustración 10 - Resultados página detalle (móvil)

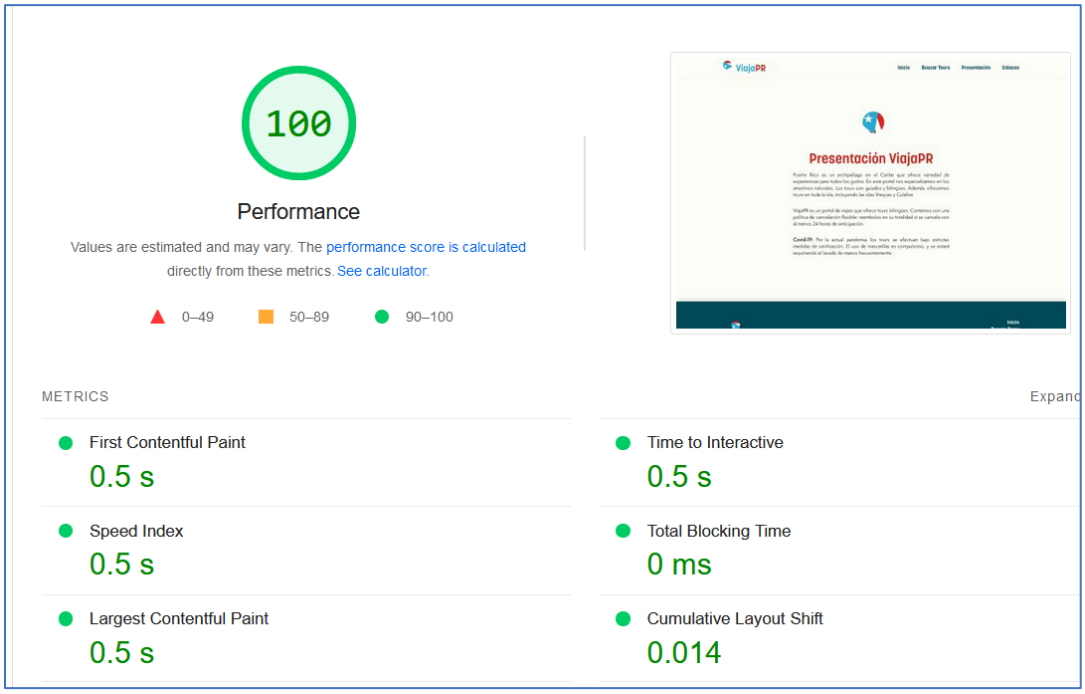

Ilustración 11- Resultados página presentación (desktop)

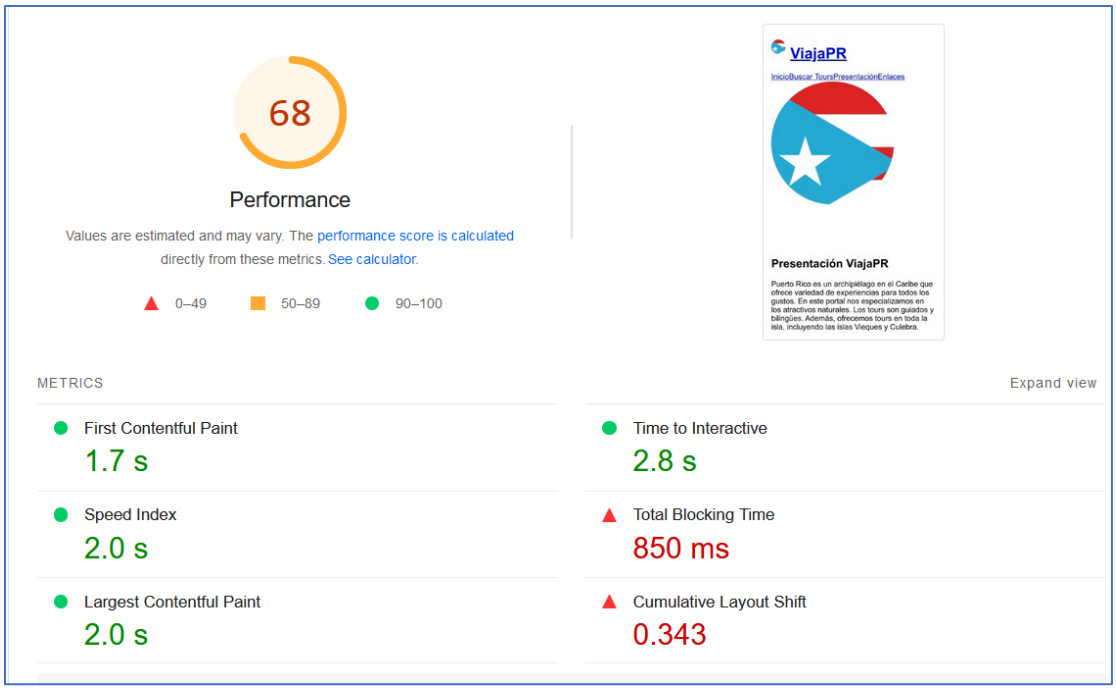

Ilustración 12 - Resultados página presentación (móvil), probablemente con algún error de conexión

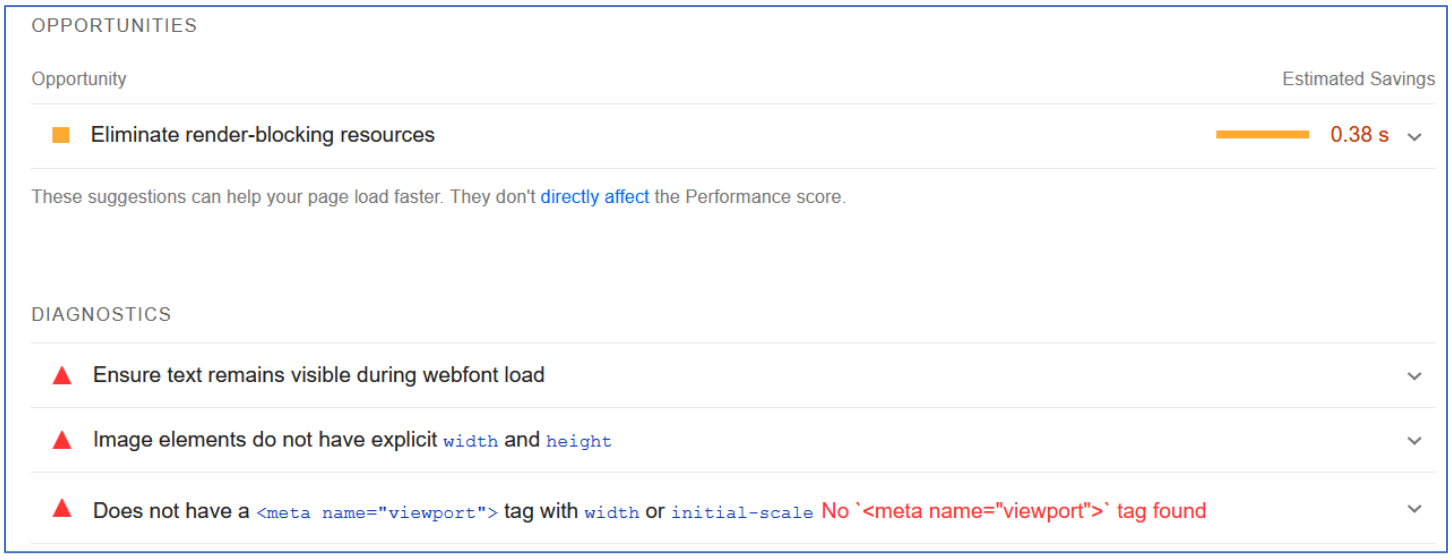

Ilustración 13 - Recomendaciones del informe de la página de Inicio (desktop)

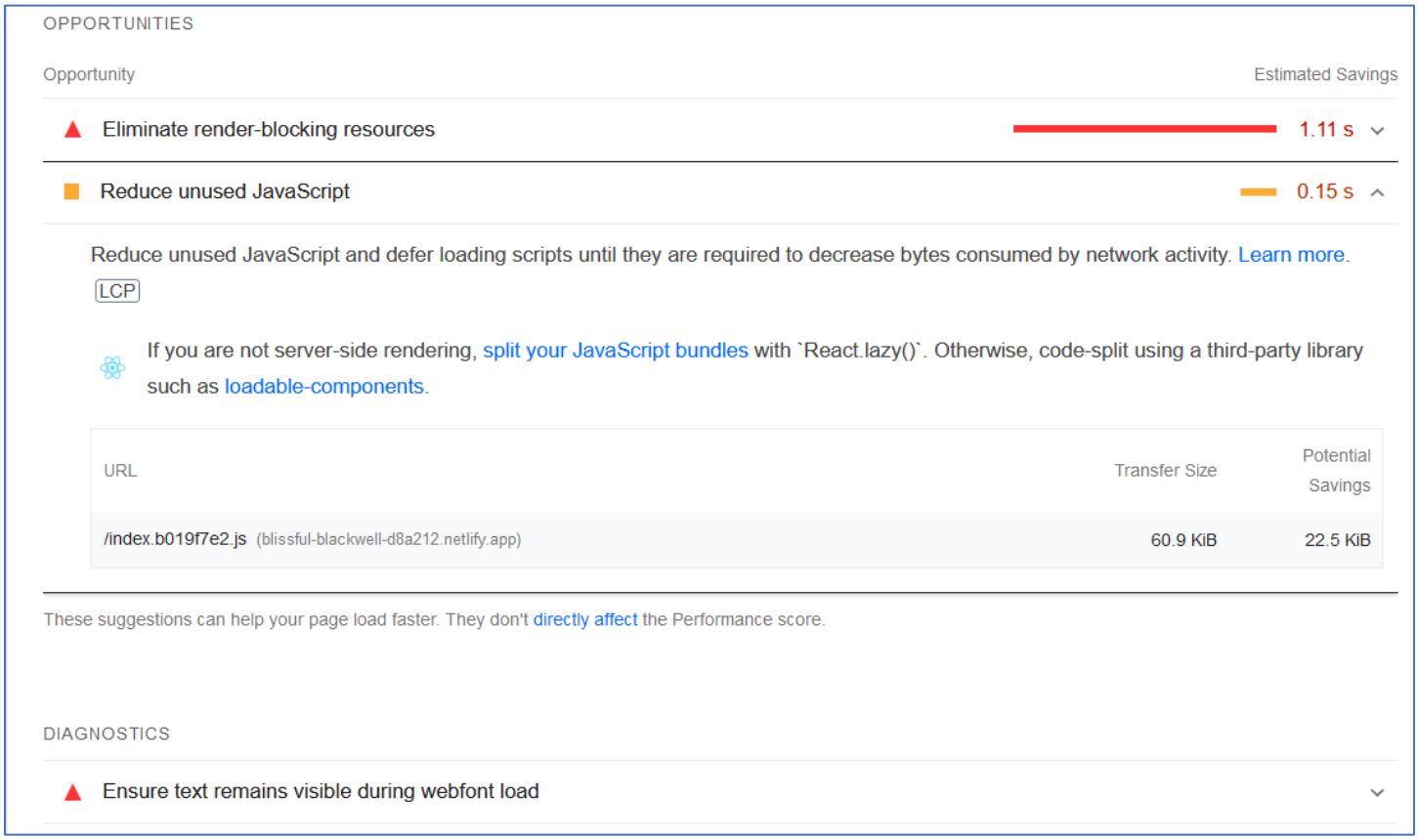

Ilustración 14 - Recomendaciones del informe de la página de Inicio (móvil)

He adjuntado los resultados de todas las páginas, pero las propuestas de mejoras son solo de la página de Inicio (tanto desktop como móvil) ya que en las otras páginas se duplicaban algunas de las recomendaciones de estas. De primera instancia se puede observar que la mayoría de las puntuaciones son muy buenas, todas están por encima de 90. La única excepción es la página de presentación versión móvil, guardé estos resultados sin percatarme de que pareciera que la página nunca cargó correctamente y el test probablemente contiene un error. Las puntuaciones más bajas son las de las versiona móviles, siendo las métricas de First Contentful Paint y Largest Contentful Paint las más afectadas.

La primera recomendación para la versión Desktop es “Eliminate render-blocking resources”. Esta recomendación se resuelve con la implementación de lazy loading de imágenes y scripts como he explicado anteriormente. El informe estima que el ahorro de tiempo es de 0.38 segundos. La segunda recomendación “Ensure text remains visible during webfont load” también la he explicado anteriormente, se trata de hacer uso de la posibilidad de “swap” fonts y hacer que la carga de la hoja de estilos de los fonts sea asíncrona. La tercera recomendación “Image elements do not have explicit `width` and `height`” la he implementado fácilmente en todas las imágenes bajo la etiqueta de <img>. La única limitación fue ponerla en las imágenes pago las etiquetas de <picture> y <source> ya que no contienen estos atributos. Declarar estos atributos sirve para evitar los cambios bruscos de layout en imágenes que son cargadas mediante lazy loading y de las cuales la el navegador no conoce sus dimensiones. Por último, el informe señala

que la página no contiene el tag <meta name="viewport">. Añadí la etiqueta y solucioné los problemas de diseño responsive que tenía debido a la falta de esta.

En la version móvil, la primera recomendación adicional es la de “Reduce unused JavaScript”. Como se puede ver en la Ilustración 14, el informe sugiere hacer lazy loading de los componentes de React, que ha sido lo que he hecho. Otra de las recomendaciones recurrentes ha sido mejorar el First Contentful Paint. Para esto volví a retocar las imágenes que quizás eran muy grandes y no se adaptaban al formato del dispositivo. También, he corregido el problema de los tags de <picture> y <source> que en la página de Detalle continuaba cargando imágenes que no eran visibles. Esto redujo bastante el tiempo de carga de esta página específica.

Informe Segunda Iteración

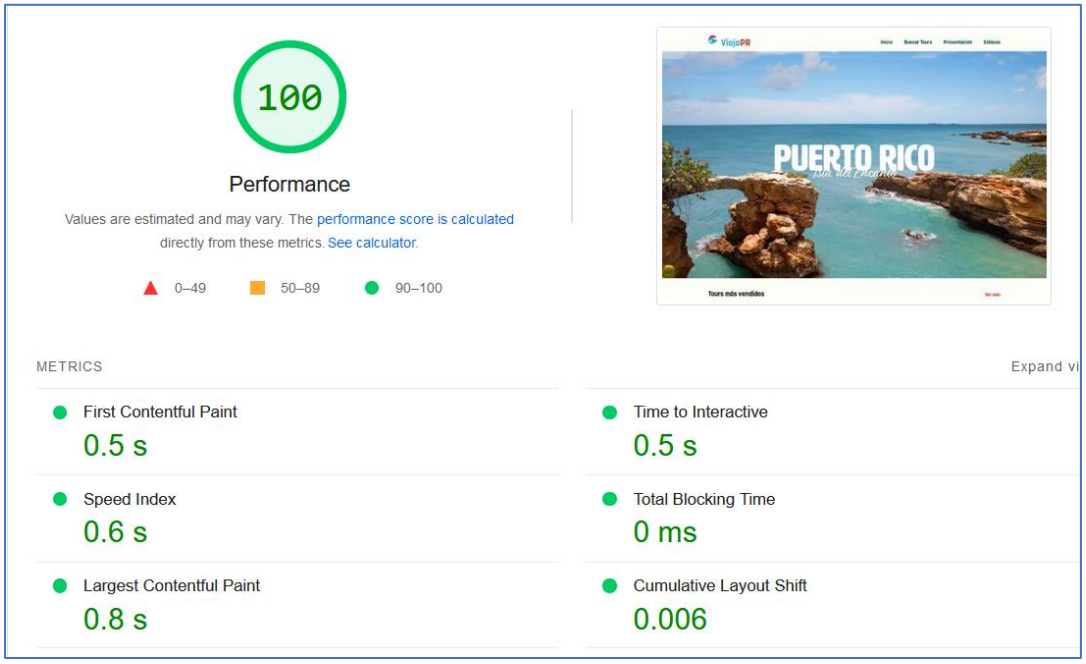

Ilustración 15 - Resultados 2da iteración página de Inicio (desktop)

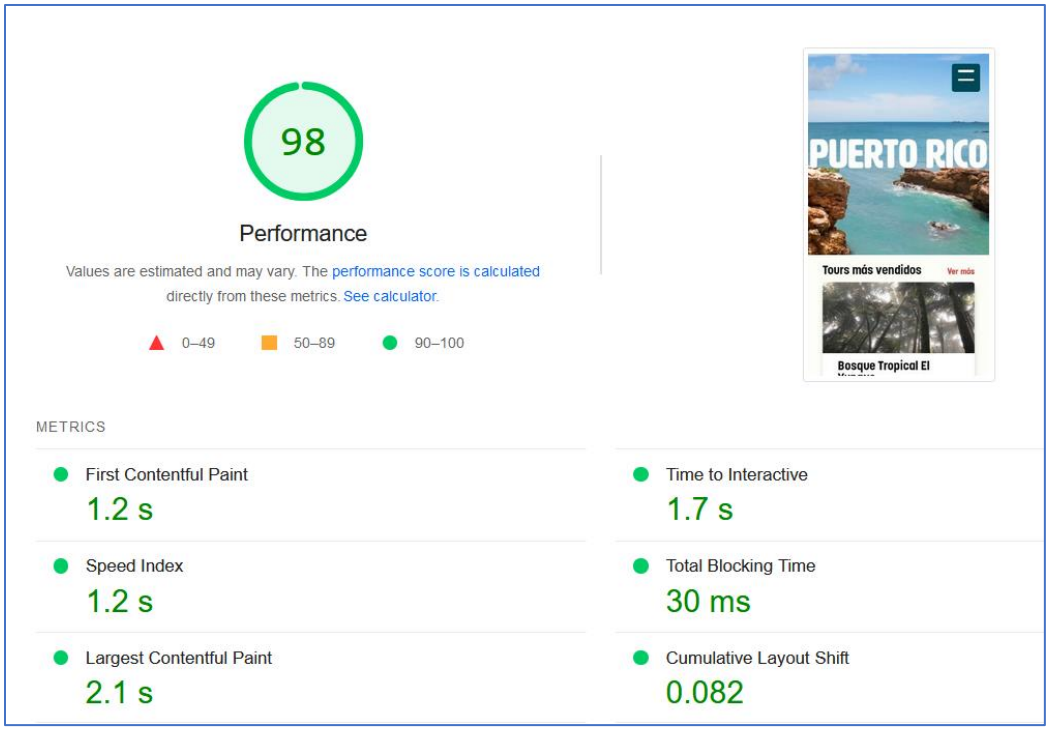

Ilustración 16 - Resultados 2da iteración página de Inicio (móvil)

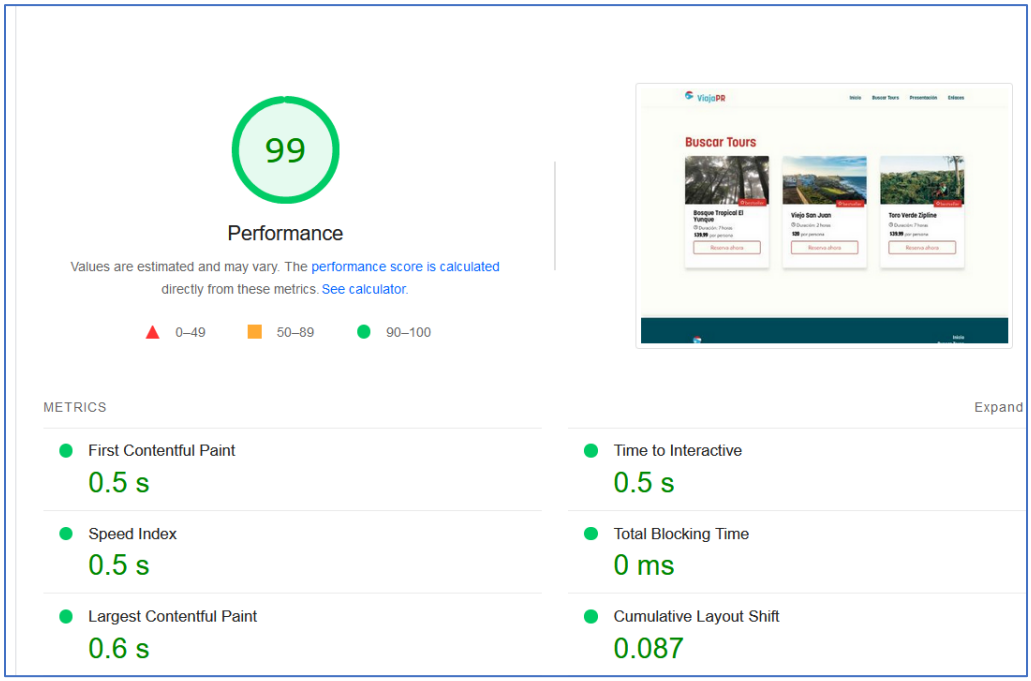

Ilustración 17- Resultados 2da iteración página de Categoría (desktop)

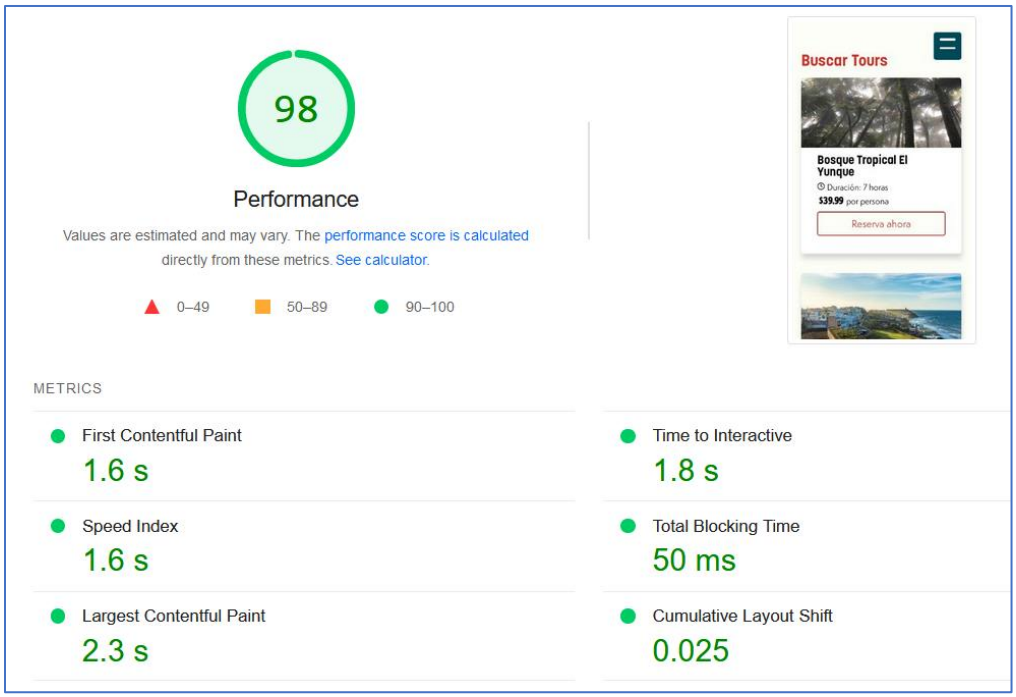

Ilustración 18 - Resultados 2da iteración página de Categoría (móvil)

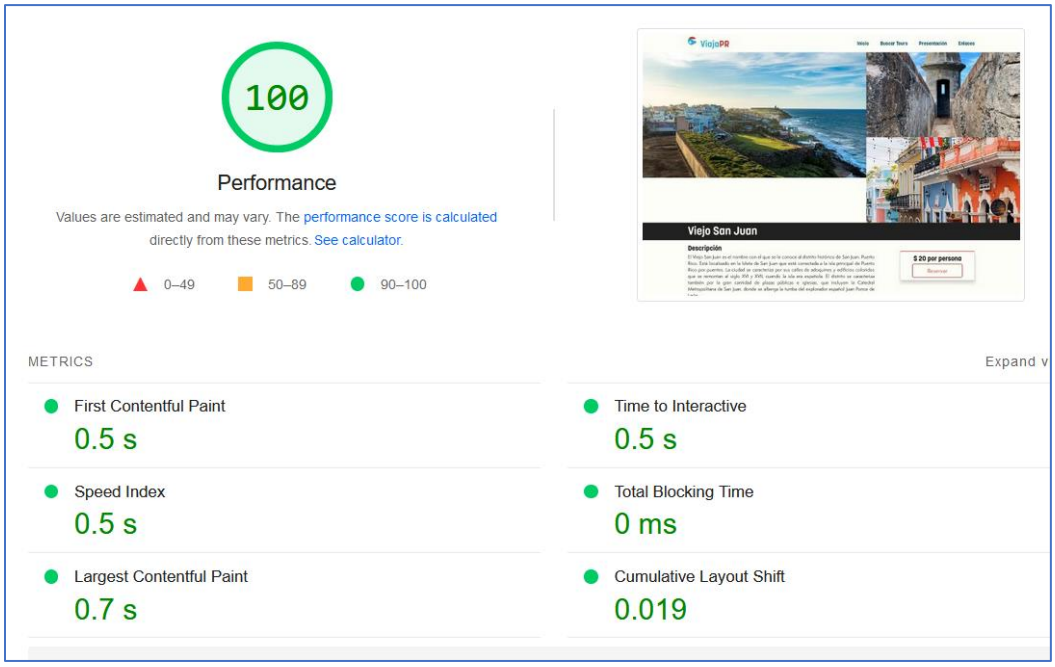

Ilustración 19 - Resultados 2da iteración página de Detalle (desktop)

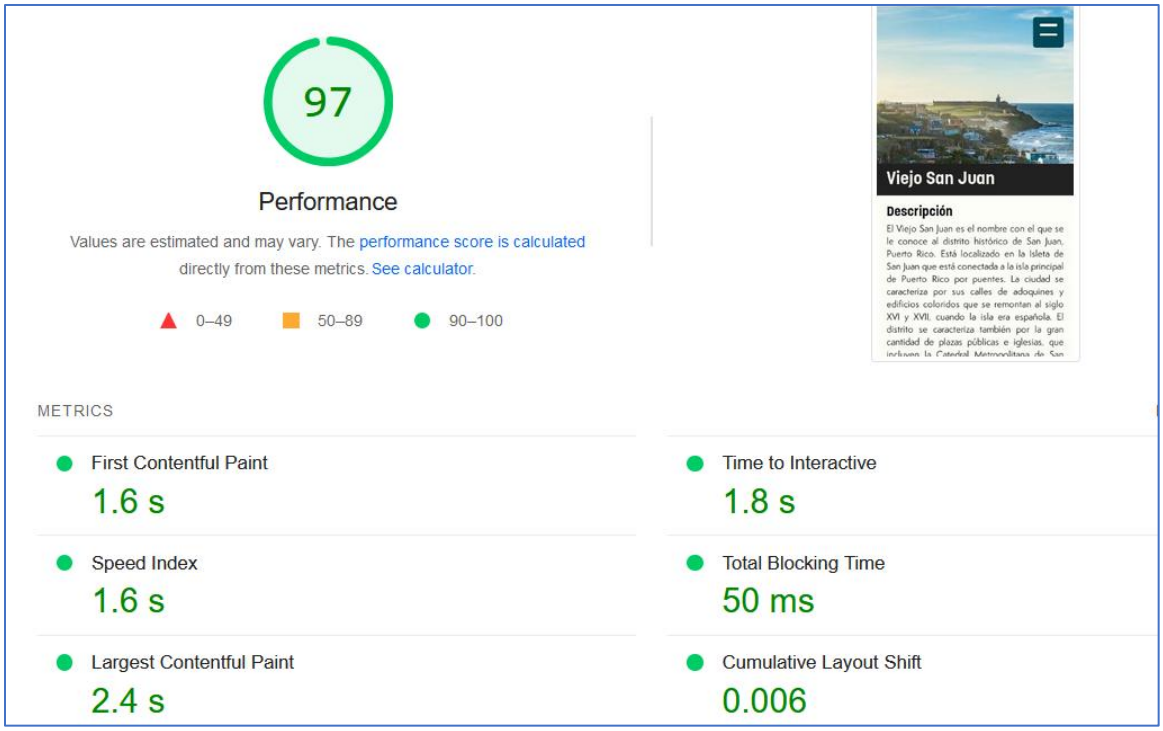

Ilustración 20 - Resultados 2da iteración página de detalle (móvil)

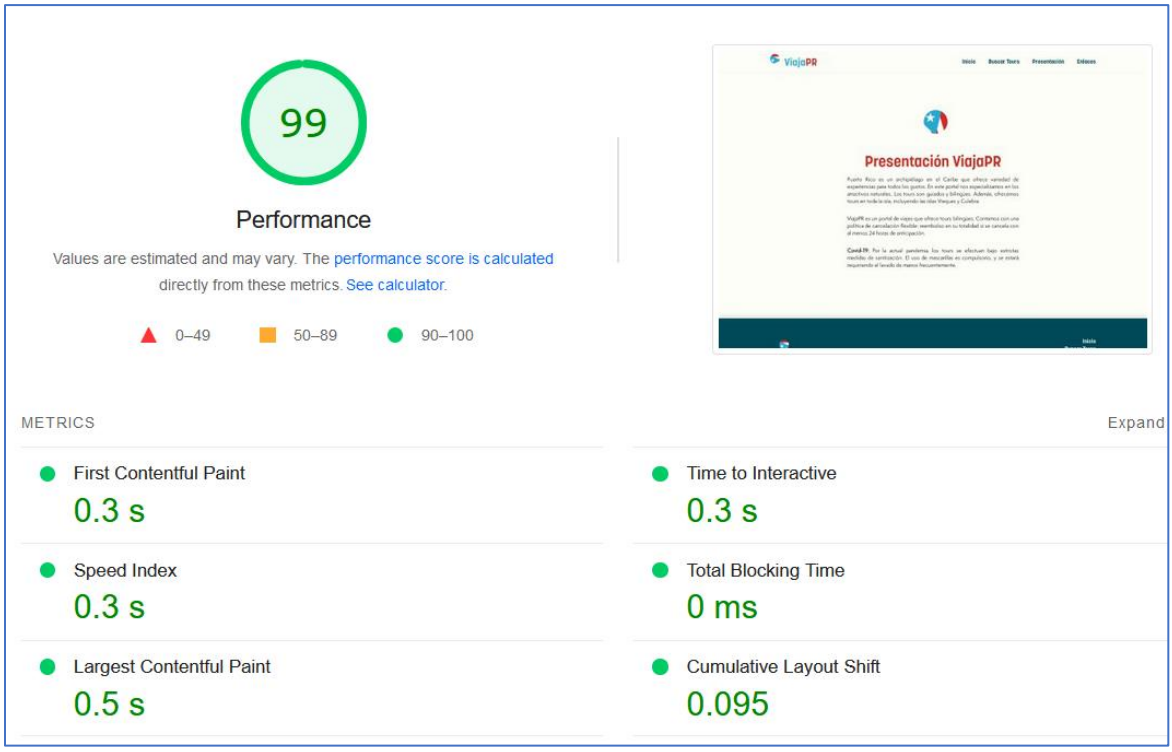

Ilustración 21 - Resultados 2da iteración página de Presentación (desktop)

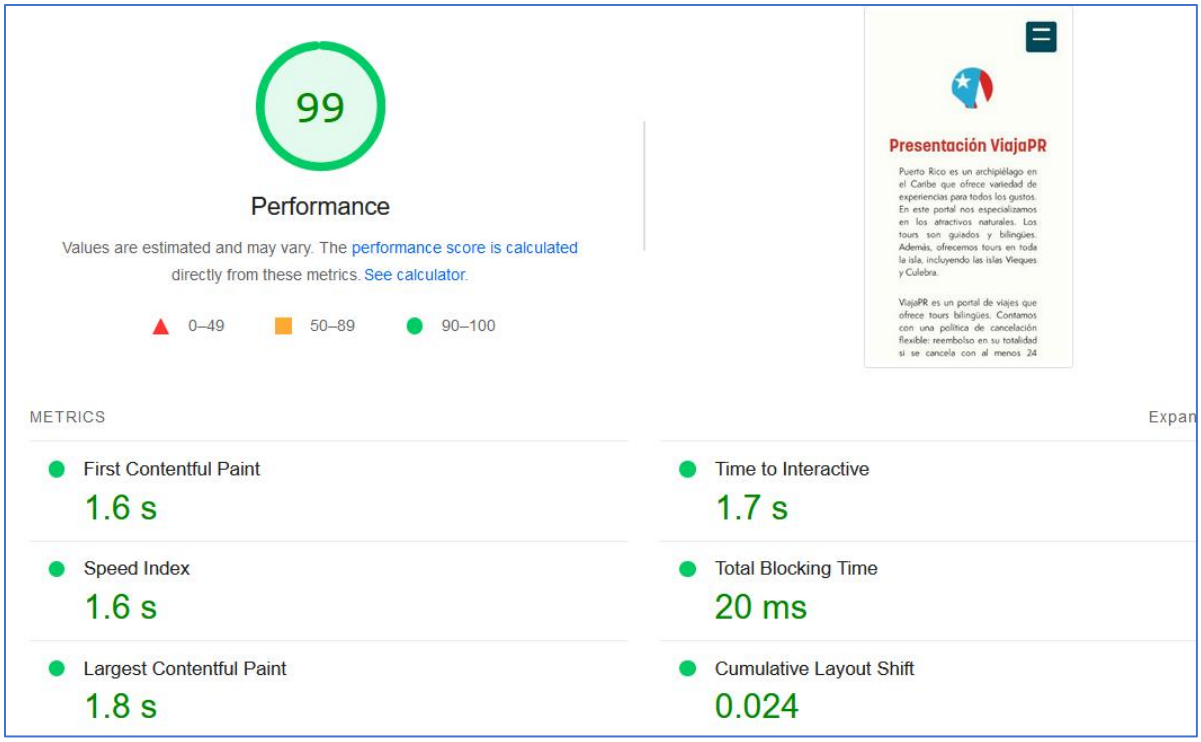

Ilustración 22 - Resultados 2da iteración página de Presentación (móvil)

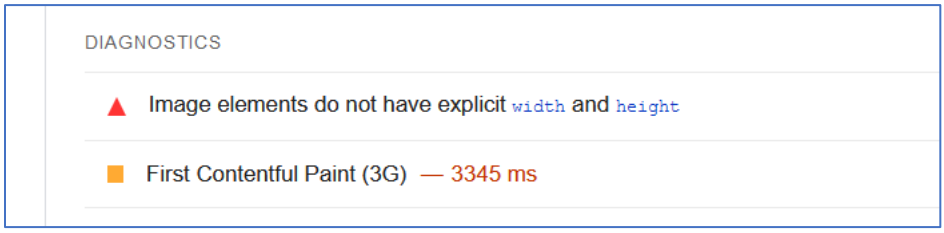

Ilustración 23 - Recomendaciones de 2da iteración

Es evidente que la mayoría de las páginas mejoraron en sus métricas, sobre todo las versiones móviles. Después de las optimizaciones, solo seguía apareciendo las recomendaciones de la Ilustración 23 en algunas de las páginas móviles, en la página desktop de Detalle que utiliza <picture> y <source> también recomendaba el uso de width y height. En el formato móvil la recomendación de la mayoría era disminuir el First Contentful Paint. Creo que la solución obvia sería eliminar o reducir archivos de imágenes más aún.

Las puntuaciones del PSI no reflejan el efecto en el tiempo de carga que tuvieron las optimizaciones. A mi parecer, las optimizaciones fueron muy exitosas en lograr reducir el tiempo de carga, especialmente en las páginas con elementos pesados como imágenes. Sería interesante seguir estudiando posibilidades de mejoras para la página.

Anexo

Respuesta a feedback de PEC anterior:

- Parcel incluye de serie minificadores de imagen (aunque incluimos imagemin, ya lleva uno llamado Sharp) que utiliza un compresor llamado SVGO para minificar SVG. SVGO puede alterar los SVG en algunos casos concretos. Seguramente, esto es lo que rompe tu SVG. Lee más aquí: https://parceljs.org/languages/svg/#minification
    - Para probar si este era el problema, he desactivado las optimizaciones en el build usando el flag –no-optimize de Parcel. El SVG seguía viéndose roto. No he podido encontrar otra posible cause.
- La técnica de dirección de arte es correcta, pero no el propósito. Las imágenes que tienen dirección de arte se ocultan en pantallas pequeñas. No sé si se ha entendido para qué sirve esta técnica
    - Como mencioné anteriormente, tuve problemas con el diseño responsive y estuve utilizando una Laptop de resolución casi 4k. En esta se veían las 3 imágenes correctamente. Cuando utilicé la pantalla Full HD, entonces me caí cuenta que estaba mal implementado.
- No usas la técnica de resolution switching por DPI
- No hay imágenes con técnica de resolution switching por tamaño
    - La verdad creo que no he entendido bien estas técnicas.
- Importas `react-dom` y `fontawesome-svg-core` en tu código, sin embargo, no están en el package.json y da error.
    - Por alguna razón a mi no mde error la falta de estas dependencias en mi package.json. Aún así, he instalado react-dom sin problemas,, pero no me fue posible instalar fontawesome-svg-core`. Este me daba error (el intentar instalar). Quizás estamos utilizando versiones diferentes de Node. El mío es v16.13.1.
- No conozco bien React, pero es mala práctica que en tu código pongas las URLs absolutas en lugar de relativas para llamar a las imágenes. (linea 65 de home.css, entre otras)
    - El problema con React es que si vas a hacer uso de una imagen debes importarla primero. Esto se vuelve difícil cuando estas importando dinámicamente. La documentación de créate-react-app recomienda el uso del public folder (https://create-react-app.dev/docs/using-the-public-folder/#when-to-use-the-public-folder) para esto. Para mí ha sido problemático utilizar imágenes incluso si las importo y no son cargadas dinámicamente porque el relative path sería distinto en folder del production build de todas formas.
    - La verdad me he quedado corta de tiempo para seguir investigando otras maneras de hacer esto correctamente pues estoy segura que debe haber alguna alternativa. Sucede que he trabajado con React anteriormente pero llevaba un tiempo sin utilizarlo y este último año han introducido cambios bastantes drásticos que me ha costado más de la cuenta asimilar. Cuando intento hacer búsqueda de soluciones en internet, la mayoría no se adaptan a la versión actual de React así que complica la cosa.