UNIVERSIDAD POLITÉCNICA SALESIANA
SEDE GUAYAQUIL

TRABAJO DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO DE:
INGENIERO DE SISTEMAS

CARRERA:
INGENIERÍA DE SISTEMAS

TEMA:
“ANÁLISIS COMPARATIVO DE JAMSTACK VS NODE.JS EN EL DESARROLLO DE PÁGINAS Y APLICACIONES WEB”

AUTOR:
Juan Hidalgo Molina y Diana Garcia Castro

TUTOR:
Msg. Vanessa Jurado Vite

Julio 2021
GUAYAQUIL-ECUADOR

DECLARATORIA DE RESPONSABILIDAD

Yo, **GARCIA CASTRO DIANA STEFANIE y HIDALGO MOLINA JUAN IGNACIO**, declaramos que los conceptos y análisis desarrollados y las conclusiones del presente trabajo son de exclusiva responsabilidad de los autores.

Nombre: Diana Stefanie Garcia Castro
CI. 0927771071

Firma del autor

Nombre: Juan Ignacio Hidalgo Molina
CI. 0924424674

Firma:
Nombre y Apellidos (Tutor): Vanessa Jurado Vite
Número de cédula:0923483507

ANÁLISIS COMPARATIVO DE JAMSTACK VS NODE.JS EN EL DESARROLLO DE PÁGINAS Y APLICACIONES WEB

Vanessa Jurado Vite[1], Juan Hidalgo Molina[1] y Diana García Castro[1]

[1] Universidad Politécnica Salesiana, Guayaquil, Ecuador
jhidalgom@est.ups.edu.ec, dgarciacas@est.ups.edu.ec, vjurado@ups.edu.ec

Abstract. Contamos con un sinnúmero de frameworks y tecnologías disponibles para la construcción de un sitio o aplicación web. Establecer claramente las características y requisitos del producto final es de suma importancia, pero, así mismo, determinar donde implementaremos la lógica y el renderizado nos permitirá elegir las arquitecturas más adecuadas para su desarrollo. Claras comparaciones entre estas tecnologías son escasas y superficiales y sin un debido análisis científico. En el presente artículo realizamos una comparativa entre 2 tecnologías y/o arquitecturas de desarrollo web más recientes, JamStack y NodeJS, las ventajas y desventajas de cada una de ellas, y, analizamos la respuesta del servidor y renderizado de páginas web implementadas con ellas, es decir, bajo las modalidades de Pre-Rendering (Sitio Estático) y Renderizado del lado del Servidor (SSR). Construimos el sitio web con el framework NuxtJS, y generamos el mismo de manera estática y SSR. Ambos sitios web, desarrollados en ambas modalidades, son analizadas por la herramienta Lighthouse. Las métricas de rendimiento han demostrado que, en efecto, el sitio desarrollado con JamStack, demuestra una mayor velocidad de carga y renderizado.

Keywords: Web, NodeJS, VueJS, NuxtJS, JamStack, Prerendering, Server Side Rendering.

1 Introducción

Con el pasar de los años, el incremento en el número de herramientas y tecnologías para el desarrollo y diseño web se ha mostrado cómo ha evolucionado la WEB, no solo del lado del desarrollador, sino también del cliente y usuario final, quien, cada día, exige en una aplicación o página web, una mayor respuesta, características y procesamiento de lo que se esperaba al respecto de esta hace unos años atrás.

Según [1] hasta hace 2 décadas atrás, el desarrollador WEB debía de conocer tan solo 3 simples tecnologías, HTML, CSS y JavaScript, suficientes para poder diseñar e implementar un documento electrónico [2], que era como se denominaba antes a una página web.

Mientras los nuevos usos para la web emergían vertiginosamente, el desarrollo de esta se volvió más extenso, complejo y exigente de cara al cliente final, quien requería

en una página web, nuevas funciones y capacidades inherentes a aplicaciones de escritorio. Para el desarrollador codificar cada característica una y otra vez tomaba una gran cantidad de tiempo y debía hacerse con cada nuevo proyecto [1]. Es así como nacen los diferentes frameworks de desarrollo web, los cuales permiten entregar un proyecto en menos tiempo, con un código más limpio y claro, mediante técnicas cuya eficacia ya han sido comprobadas [3].

En el 2006 nace JQuery, simplificando un gran número de tareas en una menor cantidad de líneas de código. Al año siguiente le siguen los frameworks CSS, destinados a simplificar y a permitir el uso de estilos no soportados en todos los navegadores de la época. Para el 2010 nacen los primeros frameworks orientados al FrontEnd, los cuales permiten desarrollar Aplicaciones WEB de una sola página (SPA) y mantener sincronizadas diferentes partes de una misma aplicación WEB [1], siendo algunos de estos, los más reconocidos, Angular, Vue y React. Sin embargo, un año antes, nace NodeJS, un framework orientado al BackEnd, cambiando el panorama del desarrollo web del lado del servidor y nominando a JavaScript como el ganador indiscutible en el desarrollo de toda la WEB, transformándolo lentamente en un lenguaje más de programación y borrando aquella percepción de lenguaje de scripting que tenía anteriormente [4].

En el 2016 fue presentado Nuxt.Js, un framework de desarrollo de aplicaciones y sitios web que permite crear SPA's y sitios web de una manera no solo ágil, sino también, generarla de manera estática, o de tal manera que sea renderizada en el servidor.

Son ambos tipos de sitios web los que se van a analizar en este artículo. Sitio estático que será implementado y servido mediante la arquitectura JamStack y, por último, renderizado en el servidor a ser implementado y servido bajo NodeJS.

Esto, con el fin de contribuir con datos que permitan visualizar la diferencia entre ambas arquitecturas y tecnologías, JamStack y NodeJS, y poder establecer y profundizar entre las ventajas y desventajas de estas.

2 Marco Teórico

2.1 Escenario General

Usualmente, se ha venido construyendo aplicaciones y páginas web de manera tradicional. Se ha usado frameworks de desarrollo web como Wordpress o stacks como XAMP o WAMP, e incluso, frameworks más tradicionales como ASP.NET. Pero también utilizado una gran cantidad de esfuerzo y tiempo configurando servidores web, puertos, seguridades, escalabilidad y varias cuestiones más, inherentes a un marco de trabajo que se ha venido usando desde hace mucho tiempo atrás. Son varias las tecnologías que han aparecido desde entonces, con aras de simplificar gran cantidad de tareas, tiempo, e incluso a ahorrar una gran cantidad de configuraciones pertinentes a la seguridad y operaciones (DevOps), que un programador web estaba obligado a dominar.

2.2 JamStack

JamStack es una arquitectura de desarrollo web, que en realidad viene a ser una nueva manera de construir sitios y aplicaciones web de manera no tradicional. JamStack entrega mayor rendimiento, mayor seguridad y un menor costo de escalado [5]. Esto lo logra de la siguiente manera:

- El renderizado ya ha sido previamente realizado al momento del despliegue, logrando así una mejor respuesta.
- La página es entregada por un servidor CDN (Content Delivery Network), con lo cual no solo la respuesta del servidor es mejor, sino que también, los temas como escalabilidad y seguridad son administrados por el proveedor CDN.
- La seguridad es mucho mejor que en otros escenarios. Al haber comunicación con el servidor solo mediante API's, el riesgo es mínimo y las oportunidades que algún hacker pueda aprovechar son escasas.
- El costo de despliegue es mínimo pues se trata con archivos estáticos que deberán ser provistos de la misma manera, estáticamente. Existen servicios de hosting estáticos de muy bajo costo como Netlify e incluso gratuitos como Firebase.
- Para el desarrollador, basta con dominar HTML, CSS Y JavaScript. Liberando al mismo de varios temas que corresponden a otros roles como los de seguridad o escalabilidad.

2.3 NodeJS

NodeJS es un entorno en tiempo de ejecución, multiplataforma, basado en el motor V8 de Chrome. Creado con el fin de poder utilizar JavaScript del lado del servidor, pero no exclusivamente, permite crear aplicaciones de red altamente escalables, como servidores web [6].

Luego de que NodeJS fuera presentado, nace el administrador de paquetes NPM (Node Packet Manager), el cual permitió que el ecosistema de NodeJS creciera cada vez más. Herramientas de código para NodeJS que incluían funcionalidades del lado del cliente, servidor y más, eran ahora más fáciles de distribuir y usar entre miles de desarrolladores, quienes no tenían que reprogramar las mismas funcionales requeridas una y otra vez [4].

Esto hizo que fuese más fácil para los desarrolladores crear prototipos más rápidamente del lado del cliente y del servidor con el manejo de un solo lenguaje, JavaScript, y permitiendo a los mismos realizar una fluida transición al campo del desarrollador FULLSTACK, a diferencia de aquellos que necesitaban dominar lenguajes de programación como PHP o Python [4].

NodeJS prácticamente revolucionó el BackEnd y con ello puso al alcance de cada vez más desarrolladores el tan anhelado rol de desarrollador FULLSTACK con el dominio de un solo lenguaje, JavaScript.

2.4 Nuxt.js

Nuxt.js es un framework de desarrollo que trabaja sobre Vue.js, permite realizar proyectos que necesiten un prototipo rápido, debido a que, la naturaleza modular de este permite, según los requerimientos y el crecimiento del proyecto, usar solamente las librerías y herramientas que sean necesarias [7].

Se puede utilizar Nuxt.js y generar la solución final de 3 modos diferentes, Aplicación o Página Web Estática (Static Page), Aplicación de una sola página (SPA) o de Aplicación de Servidor (SSR).

Generalmente, se ha decidido usar Nuxt.js debido a que sus características permiten ofrecer el mismo proyecto en varios modos, cuenta con una estructura de carpetas ya predefinida y debidamente ordenada; el enrutamiento es mucho más sencillo que si se usara solamente Vue.js y viene cargado de configuraciones listas para la producción y construcción de código.

2.5 CDN

Una Red de Distribución de Contenidos (Content Delivery Network) se refiere a un grupo de servidores distribuidos geográficamente y que trabajan en conjunto para proveer de contenidos de manera más rápida y eficiente a través de internet [8]

El punto fuerte de un CDN es la implementación de Cache a nivel de frontera, lo cual mejora enormemente la distribución de los archivos necesarios en la carga de una página o sitio web. Esto lleva a muchos sitios web modernos a distribuir sus recursos y contenidos a través de un CDN y no de un servidor tradicional.

2.6 Lighthouse

Google Lighthouse es una herramienta gratuita capaz de realizar auditorías automatizadas a un sitio o aplicación web, generar un reporte en el rendimiento, accesibilidad, SEO y más [9].

2.7 Métricas de Rendimiento Consideradas

SEO

Search Engine Optimization es el conjunto de estrategias o técnicas que es utilizada para optimizar el sitio web y así los motores de búsqueda puedan encontrar y colocar al mismo en una buena posición. Uno de los aspectos que más se valoran en la calificación SEO es la experiencia del usuario, es decir, la experiencia y usabilidad de la página y que tan fácil es interactuar con ella [10].

Analizar la capacidad SEO del sitio web es de suma importancia, pues establece la eficiencia con la que los motores de búsqueda y de indexado son capaces de analizar la página web e incluir la misma en sus resultados.

Métricas establecidas en Google web.dev

El rendimiento y la velocidad de carga de páginas web es fuertemente considerado por el equipo de Google, y según Philip Walton [11] la verdad acerca del rendimiento es un concepto relativo, y que puede variar entre diferentes redes, calidad de equipos, y la percepción del usuario de la carga del contenido y la interacción con el mismo. Una vez definidas los tipos de métricas relevantes a como el usuario percibe el rendimiento de la web se ha establecido los siguientes parámetros:

- **Perceived load speed** o que tan rápida puede una página cargar y renderizar los elementos en la pantalla.
- **Load Responsiveness** o que tan rápido puede una web cargar y ejecutar cualquier JavaScript requerido para habilitar la interacción con el usuario.
- **Runtime Responsiveness** o como luego de la carga, la página responde a la interacción del usuario.
- **Visual Stability** o si los elementos se mueven de manera inconsistente o interfieren con el usuario.
- **Smoothness** o como las animaciones o transiciones renderizan de manera consistente.

Estos parámetros demuestran que, en efecto, ninguna métrica es suficiente para capturar el rendimiento de una página web. Es así como Google establece las siguientes métricas como las más adecuadas para medir el rendimiento de una página web.

First Contentful Paint (FCP)
Marca el tiempo en el que el primer texto o imagen es pintada. Mide cuanto le toma al navegador web renderizar el primer elemento del DOM luego de que el usuario navega a la página [12].

Total Blocking Time (TBT)
Marca la cantidad de tiempo que la página se mantiene bloqueada y sin respuesta a la interacción del usuario [13].

Largest Contentful Paint (LCP)
Mide el tiempo de renderizado de la más extensa porción de texto o imagen dentro del Viewport. En otras palabras, marca la velocidad de carga percibida por el usuario [14].

Time to Interactive (TTI)
Tiempo que toma para que la página sea totalmente interactiva. Es importante pues hay sitios que optimizan la visibilidad a expensas de la interactividad, creando una frustrante experiencia de usuario [15].

2.8 Renderizado del lado del Servidor (SSR)

La gran ventaja del renderizado del lado del Servidor es que se puede sacar ventaja del servidor. Este tiene mayor capacidad de procesamiento, potencialmente una mejor conexión a internet que permite realizar llamadas a API's de manera más eficiente, procesar las respuestas, añadirlas al código HTML y generar el código HTML final a ser enviado a petición del cliente. [16]

Todo este proceso conlleva un pequeño retraso, pero el hecho de que el servidor realice todo este trabajo lo hace más fácil de ignorar.

Y no podemos olvidar las capacidades SEO (Search engine Optimization) inherentes al SSR. Debido a que la respuesta HTML enviada ya contiene todos los datos dentro de sí, el navegador no necesita compilar ninguna de las porciones que son analizadas por los bots de indexado usados por los motores de búsqueda.

Define, por lo tanto, una mejor oportunidad de SEO y de mantener la carga baja en el navegador cliente en términos de la ejecución de código.

2.9 Pre Rendering

El pre renderizado usa lo mejor de ambos mundos, el renderizado del lado del cliente y del lado del servidor. Ciertas porciones del código HTML son generadas al momento de compilación en el entorno de desarrollo. El HTML es enviado al cliente al momento de ser requerido, pero no es creado en ese momento de ser solicitado como en el renderizado del lado del servidor, es creado antes de tiempo, en la implementación del servidor. Por lo tanto, cuando el cliente solicita la página, el servidor no hace prácticamente nada, solo envía el HTML requerido [16]

El HTML en el navegador, una vez recibido, será rehidratado mediante código JavaScript que ejecuta el navegador y dará la interactividad necesaria al sitio web.

Este es el modo que ejecuta la arquitectura JamStack. El cliente solicita archivos estáticos, entre ellos el HTML previamente compilado, que son distribuidos a través de servidores CDN a nivel mundial.

3 Método

Se desarrolló el sitio web para la empresa Duport, un Operador Logístico que realiza actividades de exportación y de comercio exterior para varios clientes. La empresa tiene su sede en la ciudad de Guayaquil.

El sitio web fue desarrollado con el framework de desarrollo web Nuxt.JS en ambos modos, SSR y Pre Rendering, lo cual nos permite usar ambas arquitecturas y/o herramientas, JamStack y NodeJS, con el fin de establecer, cuál de las 2 ofrece un mejor rendimiento y respuesta.

JamStack es implementado mediante el uso de páginas estáticas, lo cual se genera con una pequeña configuración dentro de las opciones de Nuxt.JS (archivo nuxt.config.js) previo a la compilación del código. En la figura 1 se puede visualizar como en

este caso, la variable target, que por defecto viene establecida como “server” (SSR), será cambiada a “static”, con lo cual se generará un sitio estático que se implementa en el hosting estático conocido como Firebase.

Se utiliza Node.JS para el Renderizado del lado del Servidor (SSR), para esto, la misma variable target, debe permanecer con su valor por defecto “server”, o en su caso, sin ser especificada. Se usa un VPS (Virtual Private Server) en el servicio en la nube llamado Linode para la implementación del sitio web con el servidor Node.Js.

```
/* (Pre-Rendering). El valor
predeterminado es 'server' para SSR */
export default {
  target: 'static'
}
```

Fig. 1. Archivo de Configuración nuxt.config.js.

El modo estático es compilado mediante el comando “npm run generate”, el cual se encarga de construir y determinar cada ruta del sitio web como un archivo HTML diferente. Los archivos estáticos como imágenes, hojas de estilo e incluso archivos javascript usados en la interactividad del sitio, también son empaquetados y generados de manera estática. El producto final se encuentra en la carpeta dist del directorio de Nuxt.JS. Solo basta con subir todos los archivos contenidos en esta carpeta al hosting de Firebase mediante la interfaz de comandos provista por el mismo como lo pueden observar en la Figura 2.

```
PS C:\Users\nacho\Documents\duportWeb> npm run gene
rate

> duport@1.0.0 generate C:\Users\nacho\Documents\du
portWeb
> nuxt generate


 WARN  When using nuxt generate, you should set tar
get: 'static' in your nuxt.config
       👉 Learn more about it on https://go.nuxtjs.
dev/static-target


 WARN  mode option is deprecated. You can safely re
move it from nuxt.config

i Production build
i Bundling for server and client side
i Target: static
√ Builder initialized
√ Nuxt files generated
```

Fig. 2. Comando npm run generate (static).

El modo SSR es compilado mediante el comando “npm run build” como se puede visualizar en la Figura 3, es el cual se encarga de construir y empaquetar los archivos a ser distribuidos por NodeJs. Así mismo, genera el archivo del servidor y una configuración básica del mismo. De igual manera, todos los archivos compilados se encuentran en la carpeta “dist” del directorio de Nuxt.JS. El servidor NodeJS es instalado y configurado en el VPS de Linode, y los archivos compilados son subidos mediante una herramienta de transferencia de archivos (sFTP) al mismo.

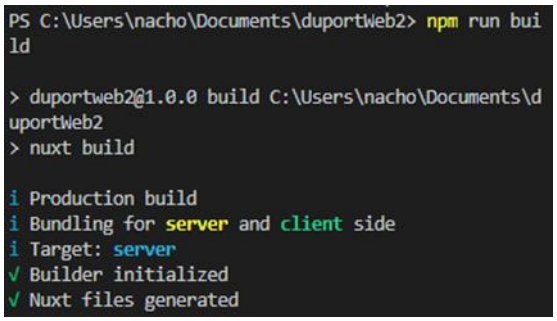

Fig. 3. Comando npm run build (SSR).

Una vez implementados ambos modos, se accede al sitio web final de la siguiente manera:

JamStack: https://duportecuador.com

NodeJS: http://173.255.241.159:50004/

La implementación en NodeJS no cuenta con un dominio propio debido a la no disponibilidad de este y obedece a la razón de ser usada solo para los fines investigativos del presente artículo.

La herramienta Lighthouse de Google ha sido usada para realizar las pruebas de rendimiento y respuesta de ambas soluciones. Esta herramienta ha sido instalada en el navegador como una extensión al mismo.

3.1 Métricas Aplicadas

La auditoría de rendimiento es realizada con la herramienta Lighthouse, incluida en el panel de Herramientas para el Desarrollador del navegador Google Chrome o como extensión para Google Chrome.

Usamos aquellas métricas de rendimiento pertinentes al renderizado final de la página web construida y analizada y que Lighthouse considera son las métricas más importantes. Estas son: FCP, TBT, LCP TTI.

4 Resultados

Mediante el uso de la herramienta Lighthouse y en base a las métricas de rendimiento establecidas previamente, se han obtenido los siguientes resultados.

Table 1. Resultados Métricas de Rendimiento Lighthouse

	JamStack (estático)	**NodeJS (SSR)**
First Contentful Paint	2.1 s	2.4 s
Largest Contentful Paint	2.9 s	8.3 s
Time to Interactive	7.7 s	7.6 s
Total Blocking Time	380 ms	330 ms

Los resultados expuestos en la tabla sugieren que la implementación realizada en JamStack es mas eficiente que aquella implementada en NodeJS. La tabla ha sido construida con los datos obtenidos en Lighthouse. A continuación, se ofrecen las capturas concernientes a las pruebas con Lighthouse.

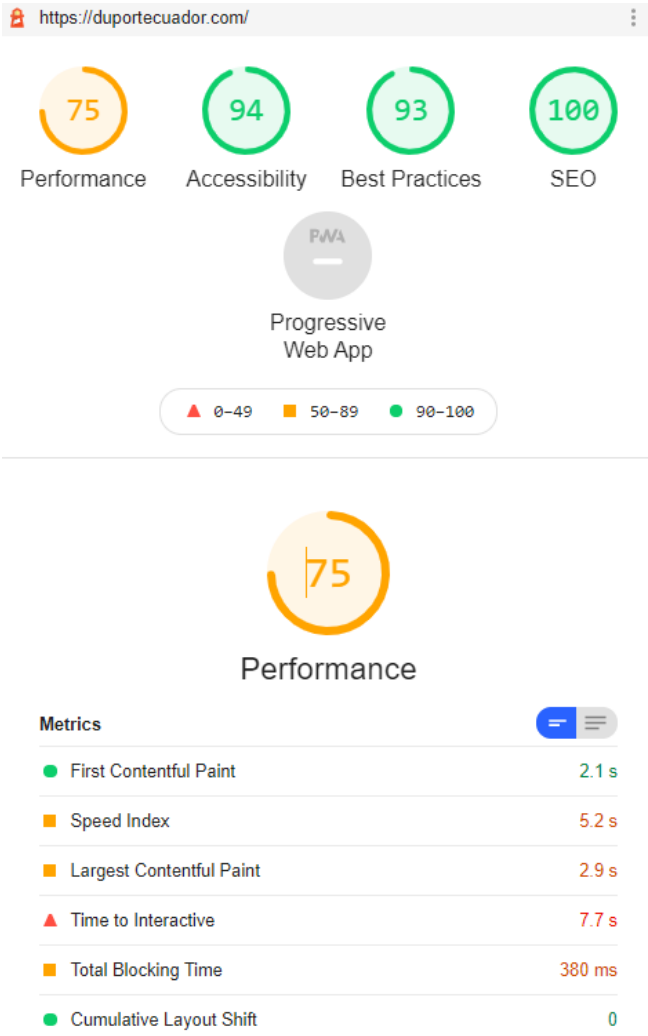

Fig. 4. Resultados Lighthouse JamStack (Estático en Firebase).

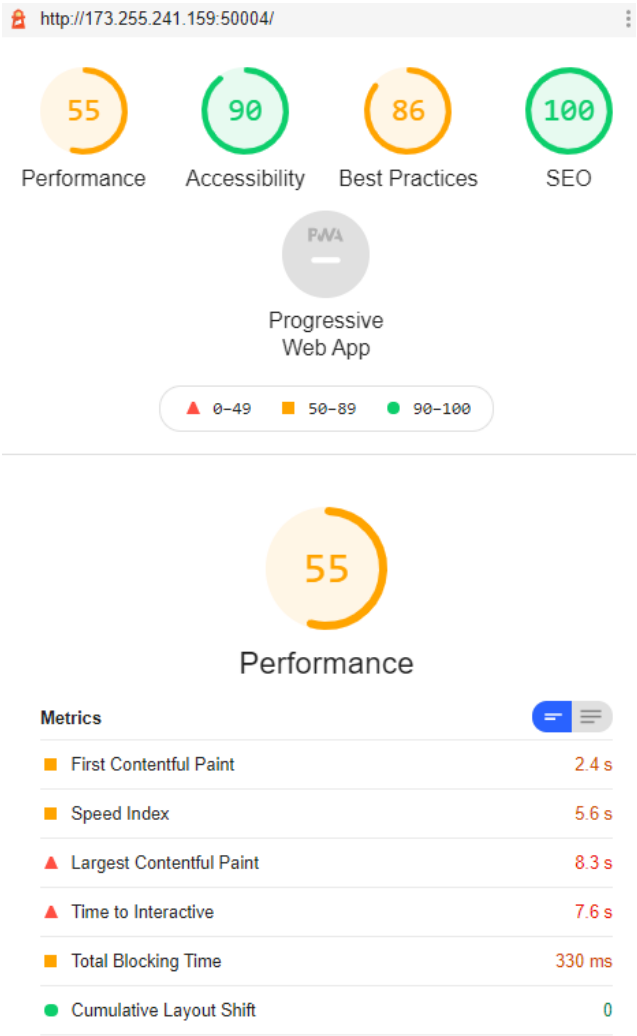

Fig. 5. Resultados Lighthouse NodeJS (SSR en Linode VPS)

Se debe hacer notar que Lighthouse expone también una métrica llamada Speed Index. Esta métrica no es usada en el presente análisis pues depende mucho de las capacidades internas de renderizado de cada navegador, lo cual, se considera indiferente a la respuesta del servidor.

5 Conclusiones

Ambas soluciones, sugieren, según las métricas de Lighthouse, un rendimiento por encima de lo aceptable. Sin embargo, la calificación otorgada a la implementación en

JamStack, sugiere que, pese a contener exactamente el mismo código, imágenes, hojas de estilo y otros recursos, la respuesta a la solicitud del sitio web, y el renderizado toma menos tiempo que con la implementación en NodeJS.

El FCP es ligeramente menor en la implementación con JamStack, lo que sugiere que los tiempos de respuesta a la solicitud inicial son casi iguales en ambas implementaciones.

El TBT es mayor en la implementación con JamStack, lo que demuestra que, efectivamente cierta porción del código JavaScript es procesado ligeramente y de manera casi imperceptible (al orden de los milisegundos) más rápidamente en el Servidor mediante SSR. La naturaleza de JamStack pone del lado del cliente la rehidratación del sitio web, provocando que ciertas porciones del código JavaScript sean procesadas con las capacidades de procesamiento del cliente, muy diferentes a las de un servidor.

El LCP nos proporciona justamente con la idea que sugiere que, en efecto, la implementación por JamStack responde de manera más rápida. Se comprueba que efectivamente, el pre renderizado provee de manera directa, y sin ser procesado en cada solicitud, con todas aquellas porciones de HTML requeridas por el cliente. Los diferentes recursos del sitio web son entregados al cliente de una manera mucho más rápida debido a la calidad de servidores CDN, inherentes a la arquitectura JamStack, que usa Firebase y cualquier otro proveedor como es el caso de Netlify y muchos más.

El TTI es al parecer, casi el mismo en ambas soluciones, lo que demuestra que la porción del código JavaScript que se encarga de dotar de interactividad a la página y otorgar la inmediata respuesta al cliente, se procesa en el cliente y no en el servidor.

A pesar de que los escenarios en el desarrollo web y los requerimientos del cliente son muy diversos, nuevas tecnologías como JamStack, intentan no solo simplificar el proceso de desarrollo sino superar en varios ámbitos a otras arquitecturas más tradicionales e incluso a otras relativamente nuevas como NodeJS. Estos ámbitos, como el de la velocidad, seguridad y disponibilidad, se han convertido en los pilares sobre los que JamStack se erige por encima de otras soluciones. Y si bien es cierto que no todas las implementaciones son iguales, ni todos los casos totalmente aplicables a JamStack, ha quedado claro que, a pesar de su corta edad, cumple con lo prometido.

La disponibilidad con JamStack es genial. Su distribución mediante CDN's que se establecen a nivel mundial, asegura que el sitio web esté siempre disponible logrando rutas más cercanas al cliente final. La sobrecarga en el BackEnd es mucho menor pues se implementan API's cuya función no será más que la de proveer de datos al sitio web. La seguridad completa el ciclo, sin un servidor dedicado, la cantidad de posibles vulnerabilidades a ser explotadas es casi nula.

Sin embargo, se cree que NodeJS y el SSR, seguirán reinando en aquellas implementaciones que requieran mayor cantidad de procesamiento en aras de liberar del mismo al cliente. Escenarios como la prestación de servicios a dispositivos móviles o de baja gama, son perfectos para implementaciones con SSR. Y si bien una de las ventajas del SSR era un mejor rendimiento SEO, no se debe olvidar que JamStack logra alcanzarlo en igual medida.

6 Trabajos Futuros

Se considera que, hay una amplia oportunidad de estudiar otros ámbitos entre ambas soluciones. Aquellos como el de la seguridad y disponibilidad, que han quedado por fuera de este estudio, pueden proveer con aún más indicadores que nos permitan escoger entre cualquiera de estas 2 tecnologías o incluso diferenciar y delimitar casos más específicos para el desarrollo entre una y otra arquitectura.

7 Referencias

[1] L. Plitnichenko, «Web Development Evolution from the 2000s' to 2020,» 2020. [En línea]. Available: https://jellyfish.tech/web-development-evolution-from-2000s-to-2020/.

[2] Librosweb.es, «Breve Historia de CSS,» [En línea]. Available: http://dis.um.es/~lopezquesada/documentos/IES_1213/LMSGI/curso/UT5/librosweb css/www.librosweb.es/css/capitulo1/breve_historia_de_css.html.

[3] G. Muente, «Guía completa del Framework: qué es, cuáles tipos existen y por qué es importante en Internet,» RockContent, 08 01 2020. [En línea]. Available: https://rockcontent.com/es/blog/framework/.

[4] H. McMurdy, «A Brief History of JavaScript & Node.js,» Medium, 05 06 2020. [En línea]. Available: https://medium.com/javascript-in-plain-english/a-brief-history-of-javascript-9289a4d344d2.

[5] O. Ekwuno, «Why you should be using JAMstack,» LogRocket, 20 06 2019. [En línea]. Available: https://blog.logrocket.com/why-you-should-be-using-jamstack/.

[6] «NodeJS,» Wikipedia, [En línea]. Available: https://es.wikipedia.org/wiki/Node.js.

[7] P. J. Hontanilla, «OpenWebinars,» 12 12 2019. [En línea]. Available: https://openwebinars.net/blog/que-es-nuxtjs-framework-de-vuejs/.

[8] «What is a CDN?,» Cloudflare, [En línea]. Available: https://www.cloudflare.com/learning/cdn/what-is-a-cdn/.

[9] A. Cunnane, «How using Google Lighthouse can help publishers to improve their websites,» https://bookmachine.org/, 11 05 2020. [En línea]. Available: https://bookmachine.org/2020/05/11/how-using-google-lighthouse-can-help-publishers-to-improve-their-websites/.

[10] «¿Qué es el SEO y por qué lo necesito?,» 40defiebre.com, [En línea]. Available: https://www.40defiebre.com/guia-seo/que-es-seo-por-que-necesito.

[11] P. Walton, «web.dev,» Google, 08 11 2019. [En línea]. Available: https://web.dev/user-centric-performance-metrics/.

[12] «First Contentful Paint,» web.dev, 02 05 2019. [En línea]. Available: https://web.dev/first-contentful-paint/.

[13] «Total Blocking Time,» web.dev, 02 10 2019. [En línea]. Available: https://web.dev/lighthouse-total-blocking-time/.

[14] «Largest Contentful Paint,» web.dev, 08 08 2019. [En línea]. Available: https://web.dev/lcp/.

[15] «Time to Interactive,» web.dev, 02 05 2019. [En línea]. Available: https://web.dev/interactive/.

[16] T. Piros, «fullstacktraining.com,» 18 09 2019. [En línea]. Available: https://blog.fullstacktraining.com/the-three-rendering-musketeers-server-render-client-render-and-pre-render/.