Next.js clase 1 primer vistazo.

Historia:
Next.js se creó en 2016, venía a solucionar cómo ejecutar código en el servidor se dieron cuenta que un componente se renderizaba en el servidor era mas rapido mas agil para crear una web.

Vino a cubrir una necesidad en el entorno de react.js. Vino a poner orden y brindar soluciones para proyectos en react.js.

Lo vemos como el framework que vino a completar a react.js.

Next.js es una herramienta que nos permite construir sitios web, que se ejecutan tanto en el servidor como en el cliente.

La documentación de next.js lo podemos encontrar en el siguiente enlace:

https://nextjs.org/

Es el framework de react para la web. Nos aporta estas ventajas:

- Nos permite tener un router de manera global el cual no tenemos que configurar, viene de manera nativa.
- Tiene componentes que ya vienen creados y optimizados siguiendo ciertos estándares de la web.

En su web podemos ver los diferentes clientes que usan nextjs.

Actualmente nextjs está respaldado por vercel, cuando tenemos un proyecto de nextjs hacer deploy a vercel ya que los servidores de nextjs están vercel.

Vamos a ver las características que tiene nextjs:

- Client and Server Rendering

Render de componentes tanto del lado del cliente como del lado del servidor, nos encontramos con una plataforma híbrida. Podemos decidir qué componentes se van a renderizar en el servidor y cuáles lo harán en el cliente. Para trabajos pesados usaremos el servidor y para las cosas livianas usaremos el cliente.

- Nested Routing

Es la capacidad para tener rutas de navegación dentro de nuestro proyecto, por default, viene de serie. Rutas dinámicas. Ya viene listo para las rutas y preconfigurado.

- Data Fetching

Hace referencia a que podemos elegir cualquier librería de terceros para hacer fetch a recursos en la web, como lo es nativo del cliente fetch o la librería axios. Pero con nextjs ya no es necesario instalar otras librerías.

- Built-in Optimizations

Básicamente nextjs tiene dos componentes ya propios que vienen optimizados para poder brindar una mejor experiencia a los usuarios. Podemos ver el ejemplo de las imágenes.
En la documentación de nextjs lo vemos.
https://nextjs.org/docs/basic-features/image-optimization

La idea es poder cumplir los estándares de la web y las web vitals de google.
https://web.dev/i18n/es/vitals/

Son métricas que nos miden el rendimiento de nuestra web. También nos dirían en qué debemos mejorar.

- Typescript Support

Nextjs tiene soporte también para typescript. Es muy importante abrir este abanico de opciones. En el proceso de instalación nos dirá si deseamos o no usarlo.

- API Routes

Puede ser nextjs un framework fullstack, podemos definir en un mismo proyecto el frontend y el backend, dependerá del desarrollo de nuestro proyecto. Usa el ambiente del servidor como nodejs. Podemos crear nuestra api. Podemos crear apis que se ejecuten en nodejs o edge que es otro runtime de js. Por defecto viene con nodejs.

- Middleware

Es básicamente es aquella función o método que está en medio antes que se finalice un proceso pero en el backend.

- Node.Js and Edge Runtimes

Para el caso de Nextjs tenemos dos entornos de desarrollo, el default que es node y en modo de beta tenemos Edge.

- CSS Support

Tenemos soporte para css puro, también se puede implementar SASS, styled component y estilos en línea.

Showcase.

Ahora vamos a ver en la documentación de Nextjs los ejemplos de aplicaciones desarrolladas con Nextjs.

https://nextjs.org/

- Página de netflix jobs.
- TikTok
- Twitch
- Hulu
- Notion, etc.

Nextjs está en crecimiento continuo, es ya el presente y el futuro. Solo con ver el respaldo de Vercel. Detrás hay una gran comunidad.

Ahora la gran pregunta, ¿Porque Nextjs?

Porque es un framework fullstack y cumple muy bien para el desarrollo de proyectos web. Deberíamos usarlo si nuestro proyecto necesita del servidor, sino para eso ya tenemos Reactjs. Pero por la gran cantidad que mueven hoy las web Nextjs es lo que debemos implementar. A continuación vemos algunas de las cosas que responden a nuestra pregunta:

- Zero Config

Desde que iniciamos nuestra instalación de nuestra aplicación ya nos viene los router configurados y demás funcionalidades.

- SSR, CSR y SSG

Server Site Rendering, Client Site Rendering, Static Site Generation.

- Automatic Code Splitting

Solo carga la parte del bundle que el cliente necesita. Lo cual hace paginas mas rapidas.

- Better User Experience

Lleva una mejor experiencia a nivel de usuarios.

- SEO

Muy amigable con las arañas de google para indexar la información.

- Great community support

Una comunidad en crecimiento. No tan grande como la de React.js pero sí creciendo mucho.

Ahora vamos a ver una pequeña comparativa de React.js vs Next.js.

Yardstick	Next.js	React
Performance	Web apps built with Next.js are very performant thanks to SSR and SSG	No code splitting really causes poor performance in React apps
State of Education	Can be hard to learn if there is no prior knowledge of React	Easy
Configuration	Almost everything is configurable	Strict
Talent Pool	Narrow	Broad
Community	Small with a good amount of resources	Broad with lots of resources
Documentation	Well written	Well written
Development Cost	Low	Low

Feature	Server-side rendering, static site generation, automatic routing, build size optimization, fast refresh/reload	React is pretty much extensible and some of these features can be enabled
SEO	More SEO friendly	Slightly SEO friendly
Third-Party API	It is possible to have third-party API using API routes	React is mainly focused on building UI
Crossplatform Applications	Next.js is mainly meant for the web	React native is based on the React.js library
Typescript	Supports	Supports
Image Optimization	Makes pictures adaptable in small viewports	It isn't built but can be achieved using third-party libraries
Offline Support	Supports	Supports
Dynamic Route	Most SSR frameworks support dynamic routes	Routes

Rendering en la web.

En la web podemos encontrar diferentes tipos de rendering, uno de ellos es el render en el servidor y podemos introducir estos tres datos qie se toman de una web para poder analizar qué es lo que nos conviene, cuando se utilizan los diferentes tipos de rendering:

- TTFB: Time to First Byte

El tiempo al primer byte.

- FCP: First Contentful Paint

El primer contenido que se muestra en la página al cargar.

- TTI: Time To Interactive

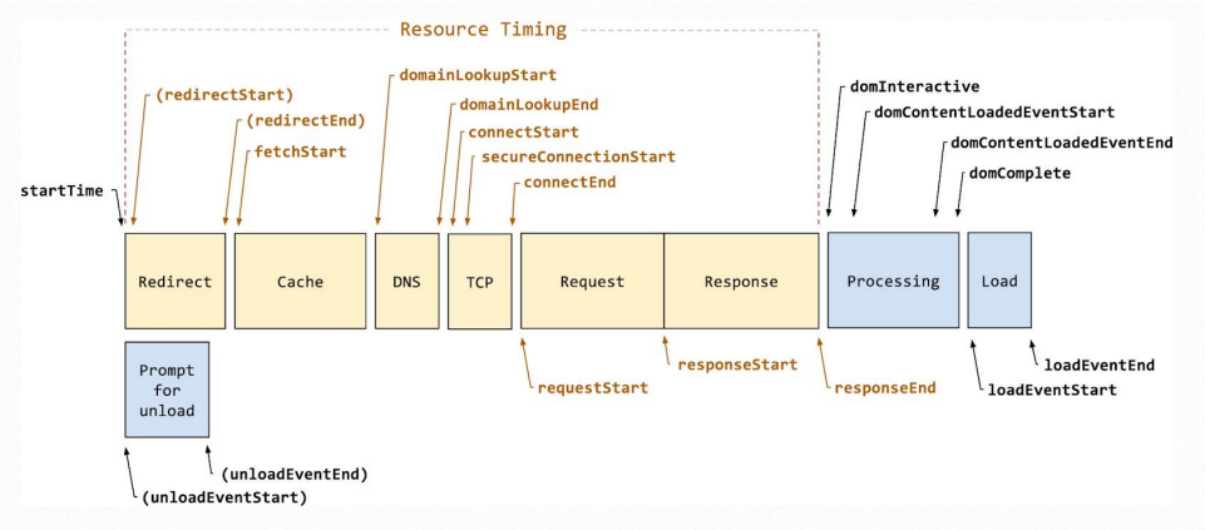
El tiempo que transcurre desde la primera carga hasta que se puede usar, que el usuario podría ya interactuar con ella.

Esto que vimos anteriormente son métricas de web vitals. Lo podemos ver en su página. Nos permiten medir la rapidez de un sitio web.

https://web.dev/i18n/es/vitals/

Esto define para google lo que es un sitio web rápido.
Vemos el primero: TTFB: Imagen de webvitals.

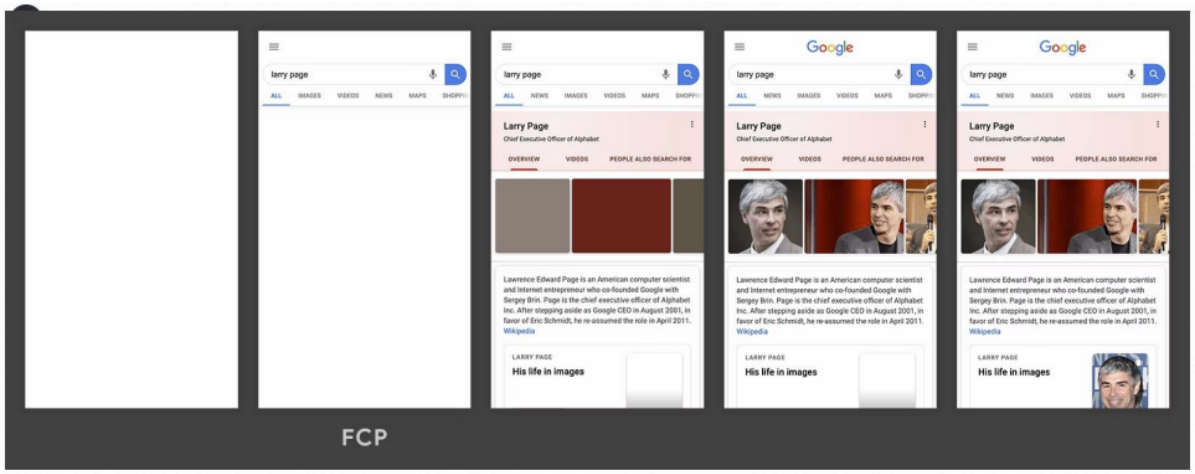
Aquí vemos el tiempo en milisegundos que debe tardar una web para llegar a ese primer byte de información. Como vemos por debajo de **800 ms** está OK, entre **1800 ms** y **800 ms** necesita mejoras y ya por encima de **1800 ms** ya es pobre y el sitio necesita mejoras urgentes en optimización y experiencia de usuario.
A continuación vemos cómo se determina este tiempo.



Una herramienta que nos aporta google para analizar una web es **PageSpeed Insights**, vemos el enlace a continuación:

https://pagespeed.web.dev/

Ahora vamos con la segunda métrica.
Vemos la segunda: FPC: First ContentFul Paint:



Es el primer contenido que se pinta en la pantalla. Como lo podemos ver en la imagen anterior.

Vemos como empieza con una página en blanco, y luego vemos la segunda carga, el primer componente del buscador, ahí es cuando se mide el tiempo. Vemos que por debajo de 1.8 seg está bien entre 3 seg y 1.8 seg necesita mejoras, por encima de 3 seg mal. Estas son las recomendaciones de performance que nos dice Google.

Ahora vamos a ver ya conociendo los conceptos anteriores lo que es renderizar en el servidor, renderizar en el cliente y páginas estáticas.

Server Side Rendering (SSR):

Genera el sitio web (HTML) sin JS, en el servidor en respuesta a la navegación. Se lo envía al cliente para que este lo **HIDRATE**. Esto evita más peticiones cuando estamos recibiendo contenido en el cliente.

Es como si fuera una página estática, pero no lo es, necesitará de JS para hacerla interactiva. Solo en la primera carga es estática. Por lo tanto podemos decir que cada página que se renderiza en el servidor con SSR en automático utiliza static site generation (SSG). Es una combinación híbrida.

HYDRATION: hidratar.

Es una técnica en la cual el cliente convierte una página HTML enviada por el servidor en un sitio web completamente interactivo para el usuario, agregando los **event handlers** a cada uno de los elementos.

Nosotros como usuario hacemos una petición al servidor, este se encarga de procesar toda esta información de la petición precarga la página y la manda como HTML y el cliente muestra esa información en lo que empieza a hidratar y darle vida a todos los elementos. Es decir agregar React.js la cual hará su magia.

La primera carga con SSR es más rápida cuando usamos hidratación.

Ventajas del SSR:

- FCP es más rápido.
- El servidor procesa y envía lo necesario, evitando todo el bundling al cliente. Esto hace que el TTI sea más rápido.
- Sitios amigables con dispositivos que tienen CPUs no tan potentes o dispositivos antiguos.
- Es más fácil que los navegadores indexen tu sitio web. Esto ayuda a posicionar tu página en los buscadores.

Desventajas del SSR:

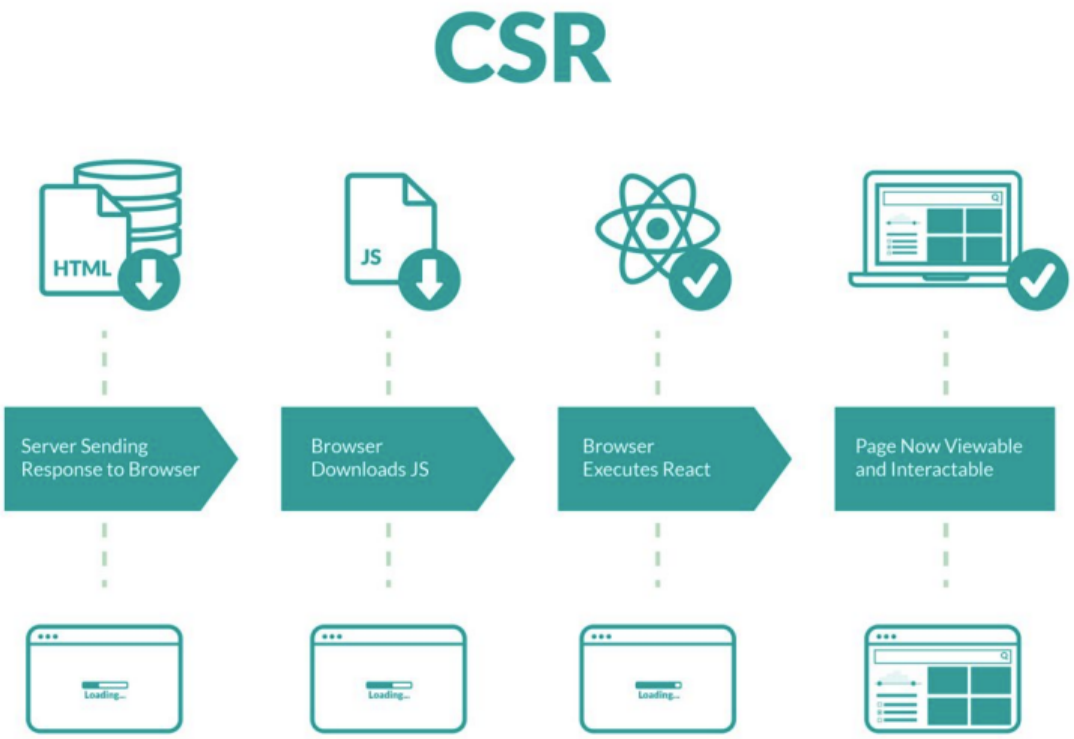
- Generar páginas en el servidor necesita tiempo. Por lo tanto produce un TTFB más lento.
- Incrementa el costo de los servidores.
- Problemas de compatibilidad con dependencias de terceros.

Herramientas para hacer SSR:

- **Next.js. De react.js**
- **Nuxt.js. De vue.js**
- **Universal. De angular.js**

Client Side Rendering (CSR):

Esta técnica renderiza las páginas directamente en el browser utilizando Javascript. Data fetching, routing son manejados dentro del browser.



Ventajas del CSR:

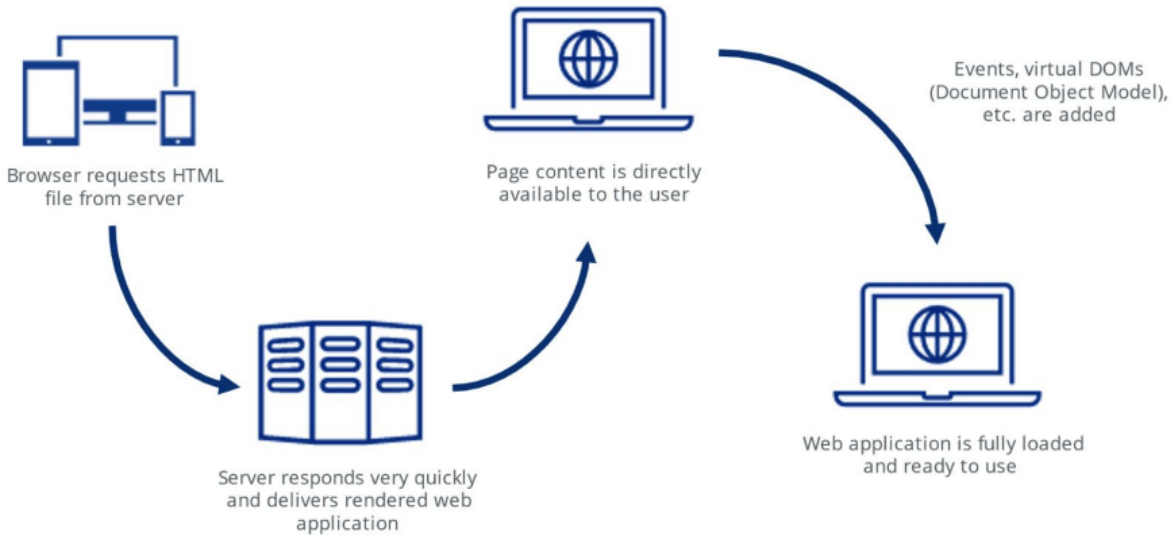
- Suelen tener costos más bajos de hosting.
- La navegación es más fluida, debido a que todo se encuentra listo en el cliente, no es necesario hacer una petición al servidor.
- El despliegue es más fácil.

Desventajas del CSR:

- El bundle tiende a crecer conforme el código se incrementa, lo cual nos obliga en un futuro a hacer code splitting manual para evitar performance issues.
- SEO pobre.
- La experiencia en dispositivos no tan potentes puede ser una pesadilla

Static Site Generation (SSG):

Es una técnica que sucede al momento de hacer el build de una aplicación. Ofrece un FCP y TTI más rápido, asumiendo que la cantidad de Javascript es limitada. Cada página produce un html diferente.



Ventajas del SSG:

- Sitios web más rápidos.
- Igual que el SSR, nos brinda el beneficio del SEO.
- Menos vulnerables a ataques (hacking).

Desventajas del SSG:

- Si el javascript y/o HTML crecen, los builds son más lentos.
- Si cambias cualquier cosa, hay que hacer un re-deploy. Ya que el contenido es estático