DAYI

# Server-Side Rendering

El Server-side rendering (o renderizado en la parte servidor) se basa en la posibilidad de poder renderizar el HTML de nuestros componentes en cadenas de texto en la parte servidor, vez de la parte cliente. Estas cadenas serán las respuestas que nuestros servidores de NodeJS devolverán a las peticiones principales de nuestra Web. En vez de funciones que manipulan DOM en el navegador, delegamos este renderizado a una fase anterior en el servidor.

 ¿Que es Server Side Rendering?

Es una técnica utilizada para representar una aplicación de página única (SPA) en el servidor. Si, el servidor envía una página totalmente representada al cliente y posteriormente la SPA tomará el control de la interacción y renderizado al terminar de descargar e interpretar su código JavaScript.

# ****¿Cómo es esto?****

El cliente realiza una petición al servidor y éste, consulta los datos necesarios y genera una página HTML enriquecida con los datos como respuesta, pero, cada vez que se navega o se cambia de ruta, es necesario realizar todo este proceso nuevamente.

Con la aparición de algunos frameworks y bibliotecas JavaScript modernas, conocimos las aplicaciones totalmente generadas por JavaScript y el concepto de SPA (Single Page Aplication = aplicación de página única), las cuales se renderizan del lado del cliente, son dinámicas y muy rápidas.

De este modo, el cliente realiza un petición al servidor y este devuelve un solo archivo HTML sin ningún contenido hasta que el navegador obtenga e interprete todo el JavaScript.

JOSS

# ¿Qué impacto en desarrollo me puede suponer usar SSR?

Tenemos que tener claro que hacer uso de SSR no va a ser útil para todos y que su uso tiene que estar muy justificado en nuestra aplicación, pues podemos tener 3 imprevistos con los que no habíamos contado:

1. Contar con SSR nos va a suponer tener **un aumento en gastos por infraestructura**. Nuestras aplicaciones SPA, cuando se encuentran empaquetadas, se comportan como estáticos que pueden ser servidos desde un CDN que tengamos contratado. Si necesitamos SSR, mínimo necesitamos una máquina que ejecute NodeJS para que se realice el sistema de renderizado en servidor.
2. Delegar este renderizado a servidor nos**va a suponer una mayor carga en recursos**. Lo que quizá provoque que nuestro sistema escale peor y que necesitemos más CPU o memoria en casos de un uso intensivo del sistema.
3. Algunas librerías de cliente que vayamos a usar, puede que no se lleven bien con el SSR y que**tengamos que adaptarlas para que funcionen con este sistema**. O incluso puede que ya tengamos una aplicación en vue servida como una SPA, que esté haciendo uso de hooks del ciclo de vida del componente, y que en SSR no vayan a funcionar por su propia naturaleza, teniendo que hacer adaptaciones.

# Como funciona

El problema de primera carga

Vaaaleee te compro lo del SEO, pero... ahora ¿cómo me vas a decir que la primera carga en una aplicación moderna puede ser más lenta que con lenguajes de generación de markup en servidor tipo Razor o JSP? Veamos el porqué, cuando montamos una página con una tecnología de generación de markup en servidor, todo se genera desde recursos locales (o al menos servidores que tenemos a mano):

Diagrama

Descripción generada automáticamente

Cuando tenemos que servir por primera vez una página desde una tecnología moderna (React, Angular, Vue...), tenemos primero que cargar el HTML + JavaScript, y después dar otro salto para pedir los datos que nos hacen falta y ya pintar la página.

Imagen que contiene Diagrama

Descripción generada automáticamente

Es algo así como si fueras a casa de tus padres y en vez saquearles la nevera de primeras, esperaras a volver a tu piso para pedirles que te envíen "los tupper" por mensajero.

¿No sería mejor que esos datos ya vinieran precargados? Eso es, veo que vas por buen camino... en esto consiste “server side rendering".

¿Por qué es tan importante la diferencia de segundos?

Si lo aplicamos a nosotros mismos, a que más de una vez somos impaciente cuando navegamos por internet o incluso en mitad de la carga nos arrepentimos de comprar algo, una página que tarde más en cargar que la competencia se traduce en una página que vende menos, una página que sea un poco lenta en cargar se traduce en clientes que se arrepienten de haber clickado y se meten a otra cosa es decir... tu empresa pierde dinero y eso hace daño.

MOI

API REST

En principio podemos hacer un fetch para pedir datos a esas API's tanto desde servidor como de cliente... hasta aquí parece que todo bien, pero nos podemos topar con un problema serio... LA SEGURIDAD: cuando estamos autenticados, tenemos cookies y headers que arrastran por ejemplos tokens de sesión, ¿Qué podemos plantear?

OPCION 1

En el lado servidor no hay problemas en setear las cookies que hagan falta para poder hacer peticiones, en el lado cliente por seguridad el navegador web puede limitar el envío de esas cookies a una API REST que no esté en el mismo dominio (o no queremos liarnos la manta a la cabeza arrastrándolas y configurando servidores). En este caso lo que podemos hacer es que el propio servidor web de front end haga de "palanca", es decir tenga un proxy que a peticiones del mismo dominio (por ejemplo las que tengan como prefijo "/api" en la ruta), las redirija al servidor que toque, y esa respuesta se la envíe al cliente (digamos que es como si engañaramos al navegador).

Diagrama

Descripción generada automáticamente

**Opción 2:** Si usamos una API moderna podemos tener bien configurado CORS y arrastrar las cookies y demás, en este caso, la solución es mucho más directa, no nos hace falta servidor de proxy, ya que tanto cliente como servidor acceden a un servidor tercero de API's.

Diagrama

Descripción generada automáticamente

1. Cómo sería la carga de una primera página (el servidor genera y sirve HTML / JS).

Diagrama

Descripción generada automáticamente

2. Una vez que la primera página esta generada, nunca más volvemos a generar para esa sesión una página en servidor, ya se genera en cliente (y el cliente se encarga de cargar los datos vía llamadas AJAX), dependiendo de la implementación atacaremos un proxy server o directamente el servidor de API.

Diagrama

Descripción generada automáticamente