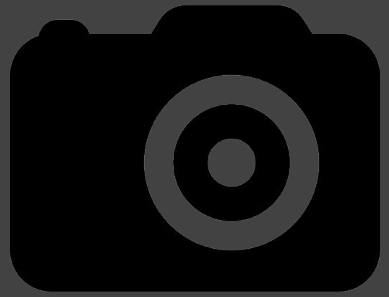
Hedy

Recordá poner a grabar la clase




Hedy
Calilegua

Hedy

Componentes de servidor y componentes de cliente

En Next.js, hay dos tipos principales de componentes:

1) Componentes del lado del servidor (**Server Side Rendering** -SSR-)

2) Componentes del lado del cliente (**Client Side Rendering** -CSR-)

Por defecto, Next.js trabaja con **componentes del lado del servidor (SSR)**
(a menos que se indique lo contrario en el código)

Componentes de servidor y componentes de cliente

Componentes del lado del servidor (SSR):

Los componentes del lado del servidor son renderizados en el servidor antes de enviar la página al navegador.

Son útiles cuando se necesita contenido dinámico basado en datos del servidor, como datos de la base de datos, API externa, o cualquier otra fuente de datos que solo esté disponible en el servidor.

Estos componentes son adecuados para mejorar el SEO, ya que los motores de búsqueda pueden rastrear el contenido generado en el servidor.

Se pueden implementar utilizando la función `getServerSideProps()` o `getStaticProps()` en las páginas o en los componentes.

En Next.js, el trabajo de renderizado se divide en segmentos de ruta para permitir el streaming y el renderizado parcial, y existen tres estrategias de renderizado de servidor diferentes:

- **Static Rendering**
- **Dynamic Rendering**
- **Streaming**

Componentes de servidor y componentes de cliente

Ventajas de la renderización en servidor

- **Obtención de datos (*data fetching*)**: los componentes de servidor permiten trasladar la obtención de datos al servidor, más cerca de la fuente de datos. Esto puede mejorar el rendimiento al reducir el tiempo que se tarda en obtener los datos necesarios para la representación y el número de peticiones que debe realizar el cliente.
- **Seguridad**: Los componentes de servidor le permiten mantener datos y lógica sensibles en el servidor, como tokens y claves API, sin el riesgo de exponerlos al cliente.
- **Almacenamiento en caché**: al renderizar en el servidor, el resultado puede almacenarse en caché y reutilizarse en solicitudes posteriores y entre usuarios. Esto puede mejorar el rendimiento y reducir los costes, ya que se reduce la cantidad de renderizado y búsqueda de datos que se realiza en cada solicitud.

Hedy

Componentes de servidor y componentes de cliente

Ventajas de la renderización en servidor

- **Rendimiento**: Los componentes de servidor ofrecen herramientas adicionales para optimizar el rendimiento a partir de la línea de base. Por ejemplo, mover las piezas no interactivas de la interfaz de usuario a componentes de servidor puede reducir la cantidad de JavaScript necesario en el lado del cliente. Esto es beneficioso para los usuarios con Internet más lenta o dispositivos menos potentes, ya que el navegador tiene menos JavaScript del lado del cliente para descargar, analizar y ejecutar.
- **Carga inicial de la página** y **First Contentful Paint (FCP)**: En el servidor, podemos generar HTML para permitir a los usuarios ver la página inmediatamente, sin esperar a que el cliente descargue, analice y ejecute el JavaScript necesario para renderizar la página.

Componentes de servidor y componentes de cliente

Ventajas de la renderización en servidor

- **Optimización para motores de búsqueda y compartibilidad en redes sociales**: El HTML renderizado puede ser utilizado por bots de motores de búsqueda para indexar sus páginas y bots de redes sociales para generar vistas previas de tarjetas sociales para sus páginas.
- **Streaming**: Los componentes de servidor permiten dividir el trabajo de renderizado en trozos y transmitirlos al cliente a medida que están listos. Esto permite al usuario ver partes de la página antes sin tener que esperar a que toda la página se renderice en el servidor.

Componentes de servidor y componentes de cliente

Estrategias de renderizado de servidor

1. Static Rendering (Default)

Con el renderizado estático, las rutas se renderizan en el momento de la compilación o en segundo plano tras la revalidación de los datos. El resultado se almacena en caché y puede enviarse a una red de distribución de contenidos (CDN). Esta optimización permite compartir el resultado del trabajo de renderizado entre usuarios y peticiones al servidor.

La renderización estática es útil cuando una ruta tiene datos que no están personalizados para el usuario y pueden conocerse en el momento de la compilación, como una entrada de blog estática o una página de producto.

Ventajas: Velocidad, reducción de la carga del servidor y optimizaciones de SEO.

Componentes de servidor y componentes de cliente

Estrategias de renderizado de servidor

2. Dynamic Rendering

Con el renderizado dinámico, las rutas se renderizan para cada usuario en el momento de la solicitud.

El renderizado dinámico es útil cuando una ruta contiene datos personalizados para el usuario o información que sólo puede conocerse en el momento de la solicitud, como las cookies o los parámetros de búsqueda de la URL.

Ventajas: Datos en tiempo real, contenido específico del usuario, información en el momento de la solicitud

Componentes de servidor y componentes de cliente

Componentes del lado del cliente (CSR):

Los componentes del lado del cliente son renderizados en el navegador del usuario.

Son útiles para partes de la interfaz de usuario que no necesitan datos dinámicos del servidor en el momento de la carga inicial, o para interactuar con el estado local del cliente.

Estos componentes pueden mejorar la velocidad de carga de la página, ya que pueden renderizarse después de que la página principal haya sido enviada al navegador.

En Next.js, los componentes del lado del cliente se pueden implementar de la misma manera que en una aplicación React típica, utilizando métodos como useState(), useEffect(), etc.

Componentes de servidor y componentes de cliente

Ventajas de los componentes del lado del cliente (CSR):

- **Interactividad**: Los componentes cliente pueden utilizar estado, efectos y escuchadores de eventos, lo que significa que pueden proporcionar información inmediata al usuario y actualizar la interfaz de usuario.
- **Hooks**: pueden utilizarse todos los hooks de react y crear nuevos hooks.
- **APIs del navegador**: Los componentes cliente tienen acceso a las APIs del navegador, como geolocalización o localStorage.

Para convertir un componente a CSR, basta con agregar la directiva ***"use client"*** al comienzo del archivo.

Componentes de servidor y componentes de cliente

En resumen:

Los componentes del lado del servidor son útiles para la generación de contenido dinámico desde el servidor, mientras que los componentes del lado del cliente son útiles para la interactividad y la manipulación del DOM en el navegador del usuario. En muchos casos, se utilizan ambos en una aplicación Next.js para combinar la velocidad de carga inicial con la interactividad dinámica.

Pre-renderizado de páginas

En versiones anteriores de Next.js, con Pages router, el pre-renderizado de páginas se podía lograr utilizando las funciones **get Static Props** y **getServerSideProps**, dependiendo de si se necesitaban datos estáticos o dinámicos.

getStaticProps: sirve para pre-renderizar páginas estáticas durante el tiempo de compilación. Es útil cuando el contenido de la página no cambia con frecuencia y puede ser generado en tiempo de compilación. Actualmente, con el APP router, un simple fetch() cumple la misma función.

```
// `pages` directory

export async function getStaticProps() {
  const res = await fetch(`https://...`)
  const projects = await res.json()

  return { props: { projects } }
}
```

```
// `app` directory

// This function can be named anything
async function getProjects() {
  const res = await fetch(`https://...`)
  const projects = await res.json()

  return projects
}
```

Pre-renderizado de páginas

getServerSideProps: Esta función se utilizaba para pre-renderizar páginas en cada solicitud del cliente. Es útil cuando necesitás acceder a datos en tiempo de ejecución o cuando el contenido cambia con frecuencia. Esto puede afectar el rendimiento ya que cada solicitud del cliente genera la página de nuevo en el servidor.

Con el *app router*, podemos hacer el data fetching directamente desde un componente de servidor. Esto nos permite enviar menos JavaScript al cliente, manteniendo el HTML renderizado desde el servidor.

Estableciendo la opción de caché a ***no-store***, podemos indicar que los datos obtenidos nunca deben ser almacenados en caché. Esto es similar a getServerSideProps en el directorio pages.

```
// `pages` directory

export async function getServerSideProps() {
  const res = await fetch('https://...')
  const projects = await res.json()

  return { props: { projects } }
}
```

```
// `app` directory

// This function can be named anything
async function getProjects() {
  const res = await fetch('https://...', { cache: 'no-store' })
  const projects = await res.json()

  return projects
}
```


Hedy