NEXT.JS V12

Índice

Inicializar el proyecto

Podés crear un nuevo proyecto usando:

```
npx create-next-app@latest
# o
yarn create next-app
# o
pnpm create next-app
```

Esto va a crear una plantilla común y corriente de Next.js, si querés usar typescript debés agregar `--typescript` luego del `create-next-app` y eso se vería algo así:

```
npx create-next-appp@latest --typescript
# o
yarn create next-app --typescript
# o
pnpm create next-app
```

Podés ver que además de npm/npx se podría usar yarn o pnpm que son package managers más nuevos que suelen tener un mejor rendimiento.

Es muy recomendable investigar sobre create-t3-app si estás usando nextAuth, tRPC, Typescript, o Tailwindcss con Next.js. Ver más sobre create-t3-app

Ver más

Rutas

Next.js usa un sistema de archivos como rutas que significa que cualquier archivo dentro del directorio `pages` se considerará una ruta nueva. A diferencia de React sin Next.js, el cual necesita de un Router dedicado.

Sub rutas

Para crear sub rutas como `/blog/post/123` debes crear un archivo dentro de una carpeta con el nombre que quieras para la sub ruta. Por ejemplo: si creas una carpeta blog con una carpeta post adentro y luego un archivo llamado `123.js` la ruta anterior funcionará.

Ver más

Rutas índice

El servidor automáticamente convierte cualquier archivo index.js a su índice. Es decir, si tenés `/blog/index.js` eso sería equivalente a `/blog` en el buscador.

Ver más

Rutas dinámicas

Las rutas dinámicas en Next.js se hacen poniendo el nombre del archivo entre `[]`. Por ejemplo: `/blog/post/[id].js` esto va a crear una ruta dinámica después de `/post`. Para acceder a la info ingresada a la ruta dinámica podes usar el parámetro dentro de `getInitialProps(ctx)/getServerSideProps(ctx)`. Tambien podés usar el `useRouter()` hook. Si se ingresa a `/blog/post/123` este código pondría en display `Post: 123`.

```
import { useRouter } from "next/router";

const Post = () => {
    const router = useRouter();
    const { id } = router.query;
    // se usa id porque ese es el nombre de la ruta dinamica([id].js).

    return <p>Post: {id}</p>;
};

export default Post;
```

Esto también se puede usar para estructurar queries ingresadas en la URL. Por ejemplo si se ingresa a `/blog/post/123?foo=bar` `router.query` va a ser igual a:

```
{ "foo": "bar", "id": "123" }
```

En el caso de que las rutas necesiten más de una ruta dinámica podés tener rutas dinámicas dentro de otras usando `[]` como nombre de la carpeta. Por ejemplo `/blog/[postName]/[id].js`. El hook `useRouter` devolverá algo así:

```
{ "postName": "...", "id": "..." }
```

Ver más

Next Link

next/link es una buena opción para reemplazar los elementos `<a>` comunes, ya que precarga las páginas a las que linkeas mejorando el rendimiento cuando el usuario sigue cualquier link. Un ejemplo del next/link en uso sería:

```
import Link from "next/link";

function Home() {
    return (
        <ul>
            <li>
                <Link href="/">Home</Link>
            </li>
            <li>
                <Link href="/about">About</Link>
            </li>
```

```
            <li>
                <Link href="/blog/posts">Blog</Link>
            </li>
        </ul>
    );
}
```

Ver más

Rutas del frontend

Para crear una nueva ruta en el frontend lo único que debes crear es un nuevo archivo .jsx, .tsx, .js, o .ts si estás usando typescript en la carpeta `/pages` y mientras que no esté dentro de la carpeta `/pages/api` ese archivo ahora va será una ruta del frontend.

Por ejemplo si creas un archivo en `/pages/juan.jsx` que exporta un código así:

```
export default function Juan() {
    return <h1>JUAN</h1>;
}
```

Ver más

SSR vs SSG

¿Cuáles son las principales diferencias entre SSG y SSR y para qué sirven?

SSG genera el HTML cuando buildeas el proyecto y luego renderiza el HTML "compilado". Esto puede llegar a ser útil para cosas como blogs en las que el SEO es importante y el rendimiento es importante y el contenido no cambia demasiado.

SSR en cambio genera una página de HTML nueva por cada request que recibe la página. Esto sería útil para algo como una página como twitch en la que los streams activos cambian constantemente

Ver más

SSG sin data

Por default next.js elige SSG como método de rendereo eso sea si tu página hace algún requesta a una página externa o no.

En el caso de querer usar SSG y importar data de algo como una API ver getStaticProps

Ver más

GetStaticProps

GetStaticProps es un método de hacer request desde el servidor antes de que cargue la página si siempre es igual la data que va a recibir o si no cambia seguido. Esto se suele usar con SSG ya que como indica el nombre GetStaticProps no va a cambiar el resultado de esta función a menos que se haga un build nuevo.

Esta función se puede usar en cualquier pagina que sea del frontend de esta manera:

```
export default function Blog({ posts }) {
    // Render posts ...
}

// This function gets called at build time
export async function getStaticProps() {
    // Call an external API endpoint to get posts
    const res = await fetch("https:// .../posts");
    const posts = await res.json();

    // By returning { props: { posts } }, the Blog component
    // will receive `posts` as a prop at build time
    return {
        props: {
```

```
            posts,
        },
    };
}
```

Ver más

SSR

Como vimos anteriormente SSR genera el HTML para cada pagina apenas recibe el request. Lo que tiene de bueno esto es que el HTML que devuelve la pagina se sigue generando en el servidor que nos da la posibilidad de hacer SEO con esto

GetServerSideProps

La función más conocida de SSR es getServerSideProps en la que se puede hacer un request a un servidor externo antes de que cargue la página. GetServerSideProps es similar a getStaticProps la única diferencia siendo que getServerSideProps se corre en cada request y getStaticProps se corre únicamente una vez en el build y luego nunca mas. Un Ejemplo de getServerSideProps en acción es:

```
export async function getServerSideProps(ctx) {
    // como pueden ver la función recibe un parámetro, en este caso ctx
    // ctx va a tener info del request que luego podemos usar por ejemplo si es una ruta dinámica
    // el ctx nos va a dejar agarrar la parte que está cambiando y usarla para lo que necesitemos

    const res = await fetch(`https://...data/${ctx.query.id}`);
    // aqui usamos ctx.query.id porque estamos asumiendo que la ruta dinámica es */[id].js
    const data = await res.json();

    return { props: { data } };
}
```

Ver más

ISR

¿Qué es ISR? ISR es corto para incremental static regeneration y nos permite darle un "tiempo de expiración" a las páginas que usan cosas como getStaticProps para que puedan actualizar la data que está ahí ya que sabemos que getStaticPaths solo se corre cuando se buildea y luego nunca más. Para poder hacer que la función getStaticProps se revalide cada X tiempo debes hacerlo así:

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

    return {
        props: {
            posts,
        },
        revalidate: X, // osea se va a revalidar cada X segundos
        // esto causa que se revaliden los datos de toda la página cada como máximo X segundos.
    };
}
```

Ver más

Fetch de datos en el cliente

Para fetchear datos del cliente similar a como se suele hacer en react debes hacerlo dentro de un useEffect de la siguiente manera:

```
import { useState, useEffect } from 'react'
```

```
function Page() {
  const [data, setData] = useState(null)
  const [isLoading, setLoading] = useState(false)


  useEffect(() => {
    setLoading(true)
    fetch('/api/profile-data')
      .then((res) => res.json())
      .then((data) => {
        setData(data)
        setLoading(false)
      })
  }, [])


  if (isLoading) return Loading ...
  if (!data) return No profile data


  return (
    <div>
      <h1>{data.name}</h1>
      {data.bio}
    </div>
  )
}
```

Ver más

Rutas del backend

Next.js es un lenguaje full stack entonces eso significa que muchas de las cosas que normalmente hacemos con node.js se puede hacer en las rutas /api de una manera serverless.

si usas `next export` estas rutas no se exportarán, `next export` es solo para el frontend aka cliente.

Ver más

Ruta simple

para crear una ruta del backend en next.js común y corriente lo único que debes hacer es crear un archivo en /pages/api/. Por ejemplo:

```
export default function handler(req, res) {
    res.status(200).json({ name: "John Doe" });
}
```

esta ruta va devolver un objeto de JSON cuando se le haga un request, sea GET, POST, PUT, DELETE o el que sea.

Ver más

Tipos de requests

Para poder separar por tipos de requests, osea GET, POST PUT, DELETE y mas debes usar un if por ejemplo:

```
export default function handler(req, res) {
    if (req.method === "POST") {
        // Procesar el request POST
    } else {
        // Procesar cualquier otro tipo de request
    }
}
```

Ver más

Tipos de respuestas

res como en express.js tiene muchas posibles respuestas algunas son:

- `res.status(código)` - es una función que devuelve un estado, por ejemplo el código 200 que dice que está todo bien o el 404 que dice que no se encontró la página.
- `res.json(JSON)` - es una función que devuelve `JSON`
- `res.send(body)` - es una función que puede devolver `strings`, `JSON`, o un `Buffer`.
- `res.redirect([status], ruta)` - es una función que redirecciona al usuario como dice el nombre.

Ver más

Dándole un código de status a una respuesta

Esto es útil para debugear luego en el frontend ya que muchas veces es más fácil fijarse si la respuesta fue un 200, 300, 201 o algún otro tipo de respuesta en vez de fijarse el JSON.

esto se puede hacer de tal manera:

```
export default function handler() {
    res.status(200).json({ nombre: "DAN" });
}
```

Ver más

Asegurando tipos en el backend

para mejor seguridad de tipos no es recomendado extender los objetos req/res en vez usa funciones para trabajar con ellos.

utils/cookies.ts

```
// utils/cookies.ts

import { serialize, CookieSerializeOptions } from "cookie";
import { NextApiResponse } from "next";

/**
 * This sets `cookie` using the `res` object
 */

export const setCookie = (
    res: NextApiResponse,
    name: string,
    value: unknown,
    options: CookieSerializeOptions = {}
) => {
    const stringValue =
        typeof value === "object"
            ? "j:" + JSON.stringify(value)
            : String(value);

    if (typeof options.maxAge === "number") {
        options.expires = new Date(Date.now() + options.maxAge * 1000);
    }

    res.setHeader("Set-Cookie", serialize(name, stringValue, options));
};
```

pages/api/cookies.ts

```
// pages/api/cookies.ts

import { NextApiRequest, NextApiResponse } from "next";
import { setCookie } from "../../utils/cookies";
```

```
const handler = (req: NextApiRequest, res: NextApiResponse) => {
    // Llamando a nuestra función pura, usando el objeto `res` va a crear el encabezado `set-cookie`.
    // Agregar el encabezado `set-cookie` en el dominio principal y va a expirar después de 30 días.
    setCookie(res, "Next.js", "api-middleware!", {
        path: "/",
        maxAge: 2592000,
    });
    // devolver el encabezado `set-cookie` para que podamos verlo en el buscador
    res.end(res.getHeader("Set-Cookie"));
};

export default handler;
```

Un paquete que también puede ayudar mucho con seguridad de tipos y legibilidad de código es tRPC.

Ver más

Self-Hosting ISR

Self-Hosting lo único que significa es que no lo haces en vercel en este caso donde todo te es más fácil. Para hostear en cualquier otro lugar que no sea vercel debes subir la carpeta .next que es el build y ya estás listo.

Ver más

Deploy

Podes usar `next build` para generar una versión optimizada de tu aplicación para producción. Esta versión suele consistir de:

- HTML para las páginas que usan `getStaticProps` o Optimización Automática Estática
- CSS estilos globales o estilos para componentes en específicos
- JS para pre-renderizar contenido dinámico del servidor de Next.js
- JS para interactividad en el cliente por React.js

la salida estará en la carpeta `.next`

- `.next/static/chunks/pages` - Cada archivo de JS en esta carpeta se relaciona con una página con ese mismo nombre. Por ejemplo si tienes `.next/static/chunks/pages/about.js` este archivo sería el que se carga cuando entras a /about.
- `.next/static/media` - Las imágenes importadas estáticamente usando `next/image` son hasheadas y copiadas aquí.
- `.next/static/css` - CSS global/especifico para todas las pagina de la app.
- `.next/server/pages` - Todo lo prerenderizado del servidor se guarda aca con la extensión `.nft.json` y estos archivos muestran todos los caminos que dependen de cierta pagina.
- `.next/cache` - El cache de build, imágenes, respuestas y páginas de Next.js. Esto ayuda al rendimiento de la página y de los builds.

Ver más

Vercel

Si usas vercel podes subir tu código a GitHub y luego importarlo directamente desde vercel donde ellos se ocupan de los builds y lo demás.