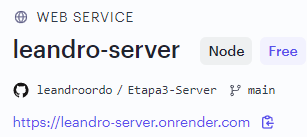
# Etapa 3

## Servidor

Este proyecto fue deployado en Render en esta dirección:

https://leandro-server.onrender.com



Se integra con Github por lo tanto, cada vez que se hace commit en la rama master, se realiza el deploy automáticamente.

## Variables de entorno

Se definieron dos variables de entorno:

|  |  |
| --- | --- |
| Variable | Valor |
| API\_URL | <https://leandro-server.onrender.com/api/> |
| MONGO | mongodb+srv://leandroo:VlJfYh7xm28cvBvu@leandrocluster0.bwkhgnl.mongodb.net/jugueteria?retryWrites=true&w=majority&appName=LeandroCluster0 |

* API\_URL: dirección donde está deployado el backend
* MONGO: string de conexión a la base de datos MongoDB

# NextJS

La aplicación cliente fue creada con el framework NextJS de React. El objetivo es aprender el uso del framework recomendado por React y evitar el uso de un framework ya [deprecado](https://medium.com/@peterdtitan/create-react-app-is-dead-what-next-f05aec3dd2d5)

Es un framework para React para hacer aplicaciones web escalables. Soporta **server side rendering** (SSR) y sitios estáticos

## Ventajas

1. Performance: al utilizar rendering en el servidor, se reduce el tiempo de carga de la página. Además, NextJS divide automáticamente el código generado y solo carga la porción de código necesaria para cada página, a diferencia de React clásico que, por default, descarga la aplicación completa al comienzo.
2. SEO: al realizar rendering del lado del servidor, se optimiza la visibilidad en los buscadores.
3. Backend serverless: Permite crear una api básica para leer y escribir datos
4. Ruteo basado en paths de archivos: Mapea URLs a archivos en la carpeta del proyecto. Hace innecesario utilizar Router Dom
5. Pre-render al hacer build: Se renderiza el sitio o partes de él al hacer el build en vez de renderizarlas dinámicamente en tiempo de ejecución, en el cliente o en el servidor.
6. Seguridad: los tokens, api keys, string de conexión, claves, etc. se mantienen del lado del servidor y no se pasan al cliente.

# Otras librerías usadas

* Typescript
* Redux
* React icons
* React toastify
* Zod

# Layout

NextJS realiza el layout de cada componente por medio de dos archivos:

layout.tsx

page.tsx

* El archivo layout.tsx define la estructura de la página y contiene la definición del header y footer por ejemplo. Contiene el diseño que se repite en todas las páginas de esa ruta y de sus rutas hijas. Se utiliza {children} para indicar dónde se renderizarán los componentes de cada página en particular.
* El archivo page.tsx define el contenido de una página específica. Los componentes allí definidos se renderizan “dentro” del archivo layout.txt

# Server Side Rendering

La mayoría de los componentes del proyecto se renderizan en el servidor. Este es el método por default de NextJS. También se indica al comienzo del módulo con la directiva

"use server"

Aquellos componentes que manejan estado del lado del cliente o que requieran alguna librería del lado del cliente (como por ejemplo mostrar popups o toasts) deben renderizarse del lado del cliente. Es por eso que los formularios, el state provider, el popup del carrito y el header se renderizan del lado del cliente por medio de la directiva

"use client"

Además, un componente del lado del cliente puede llamar a una acción que se ejecuta del lado del servidor. Para ello, la función debe indicar “use server” en su primera línea. Por ejemplo, la siguiente función:

const addToCartAction = async () => {

    "use server";

    return await addToCart({

      id,

      name,

      description,

      price,

      photo,

    });

  };

# Typescript

Todos los componentes y funciones del proyecto están hechos con typescript. Las extensiones de los componentes es tsx (en vez de jsx) y de las librerías de funciones es ts (en vez de js)

# Localstate

Debido al uso de Redux no utilicé Localstate. Además, para poder utilizar localstate se requiere que el componente se renderice del lado del cliente, con lo cual se pierden las ventajas del SSR. Para persistir el estado se aconseja implementar la grabación del carrito en la base de datos. Para poder hacer esto se necesita previamente identificar a cada cliente, o sea implementar la parte de seguridad y log in, lo cual no es un requerimiento del ejercicio.

# Validación

Utilicé validación básica de HTML 5 en los formularios del lado del cliente y con la librería Zod realizo validación del lado del servidor.

const addProductSchema = z.object({

  name: z

    .string({

      invalid\_type\_error: "El nombre no es válido",

      description: "Nombre del producto",

      required\_error: "El nombre del producto es obligatorio",

    })

    .min(3, { message: "El nombre del producto es muy corto" })

    .max(100, { message: "El nombre del producto es muy largo" }),

# Redux

El store de Redux en NextJS está del lado del servidor mientras que el StoreProvider está del lado del cliente. Es decir que el estado de la aplicación se mantiene del lado del servidor.