

[Aplicación Terminada](#)

[Creación y configuración del proyecto](#)

[Listado de Productos](#)

[Categorías y búsqueda](#)

[Carrito](#)

[Login](#)

[Detalle de Producto](#)

[Add remove productos de carrito](#)

Aplicación Terminada

DEMO <https://eagle-wear-app.vercel.app>

REPO: <https://github.com/sena-soft/eagle-wear-app>

Inicio de sesión, la aplicación se configuró para que solo al iniciar sesión se tenga acceso a los productos, se puede usar cualquier usuario válido de la api

username: mor_2314

password: 83r5^_



Iniciar Sesión

Usuario

Password

Iniciar Sesión

No tienes cuenta? [Crear cuenta gratis](#)

Listado de Productos

Eagle Wear

Bienvenido mor_2314 (2) Cerrar Sesión

Electronics Jewelry Men's Clothing Women's Clothing All

Q X

Lista de Productos



Fjallraven - Foldsack No. 1 \$109.95
Backpack, Fits 15 Laptops
men's clothing
★ ★ ★ ★ ★ (120)
[Agregar](#)



Mens Casual Premium Slim \$22.3
Fit T-Shirts
men's clothing
★ ★ ★ ★ ★ (259)
[Agregar](#)



Mens Cotton Jacket \$55.99
men's clothing
★ ★ ★ ★ ★ (500)
[Agregar](#)



Mens Casual Slim Fit \$15.99
men's clothing
★ ★ ★ ★ ★ (430)
[Agregar](#)

Carrito

Eagle Wear

Bienvenido mor_2314 (2) Cerrar Sesión

Electronics Jewelry Men's Clothing Women's Clothing All

X

Lista de Productos



Fjallraven - Foldsack No. 1 \$109.95
Backpack, Fits 15 Laptops
men's clothing
★ ★ ★ ★ ★ (120)
[Agregar](#)



Mens Casual Premium Slim \$22.3
Fit T-Shirts
men's clothing
★ ★ ★ ★ ★ (259)
[Agregar](#)



Mens Cotton Jacket \$55.99
men's clothing
★ ★ ★ ★ ★ (500)
[Agregar](#)



WD 2TB Elements Portable \$64
External Hard Drive - USB 3.0
electronics
Cantidad: 1
[Borrar](#)

Mi carrito X

	Fjallraven - Foldsack No. 1 \$109.95 Backpack, Fits 15 Laptops men's clothing Cantidad: 2 Borrar
	WD 2TB Elements Portable \$64 External Hard Drive - USB 3.0 electronics Cantidad: 1 Borrar

Subtotal \$283.9
Los impuestos se calcularán al finalizar la compra

[Finalizar compra](#)

[Continuar comprando →](#)

Detalle de producto

Bienvenido mor_2314 (2) Cerrar Sesión

Regresar

Fjallraven - Foldsack No. 1 Backpack, Fits 15 Laptops

Your perfect pack for everyday use and walks in the forest. Stash your laptop (up to 15 inches) in the padded sleeve, your everyday

\$109.95

☆☆☆☆ (120)

Agregar al Carrito

Creación y configuración del proyecto

Una vez leído el requerimiento con sus especificaciones tome la decisión de desarrollar la app con React y Typescript, para iniciar el proyecto lo hago con [Vite](#).

Antes de crear al proyecto y para tener un overview general entramos a la documentación de la api que vamos a usar <https://fakestoreapi.com/docs> para identificar los servicios y particularidades de estos

Para este proyecto damos por hecho que ya tenemos el entorno de desarrollo de NodeJS, procedemos a crear el proyecto con el comando `npm create vite@latest`
Agregamos el nombre del proyecto

```
○ sena@MacBook-Pro-de-Juan portfolio % npm create vite@latest
? Project name: > eagle-wear-app
```

seleccionamos el framework en este caso React

```
? Select a framework: > - Use arrow-keys. Return to submit.  
  Vanilla  
  Vue  
>  React  
  Preact  
  Lit  
  Svelte  
  Solid  
  Qwik  
  Others
```

Elegimos como vamos a escribir código, en este caso Typescript

```
✓ project name: ... eagle-wear-app  
✓ Select a framework: > React  
? Select a variant: > - Use arrow-keys. Return to submit.  
>  TypeScript  
  TypeScript + SWC  
  JavaScript  
  JavaScript + SWC  
  Remix ↗
```

Seguimos los pasos que nos indica

```
Scaffolding project in /Users/sena/dev/portfolio/eagle-wear-app...
```

```
Done. Now run:
```

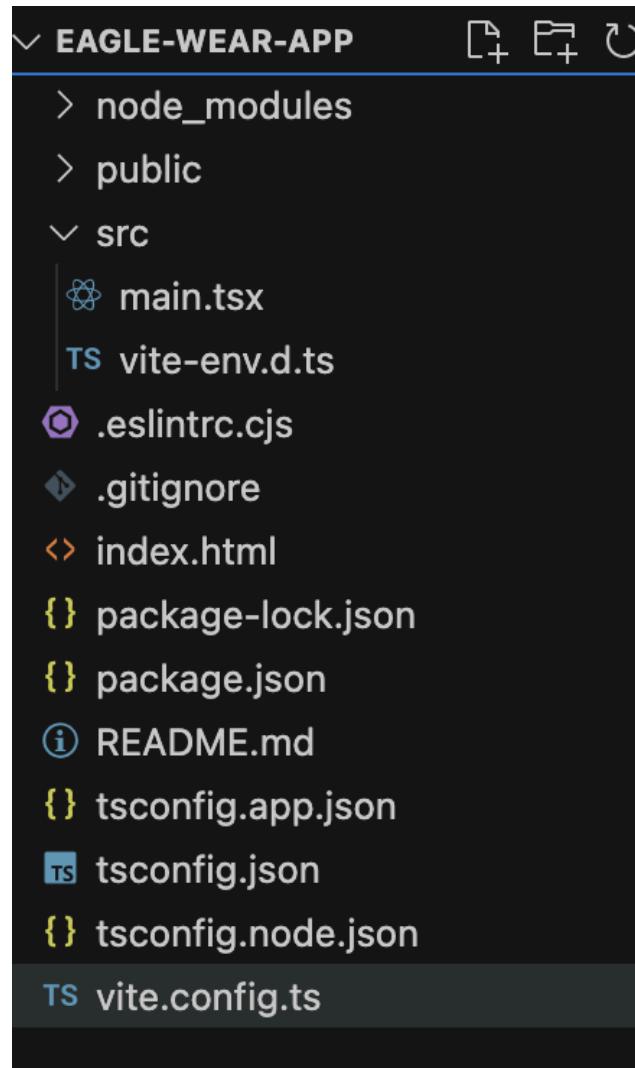
```
cd eagle-wear-app  
npm install  
npm run dev
```

```
sena@MacBook-Pro-de-Juan portfolio % █
```

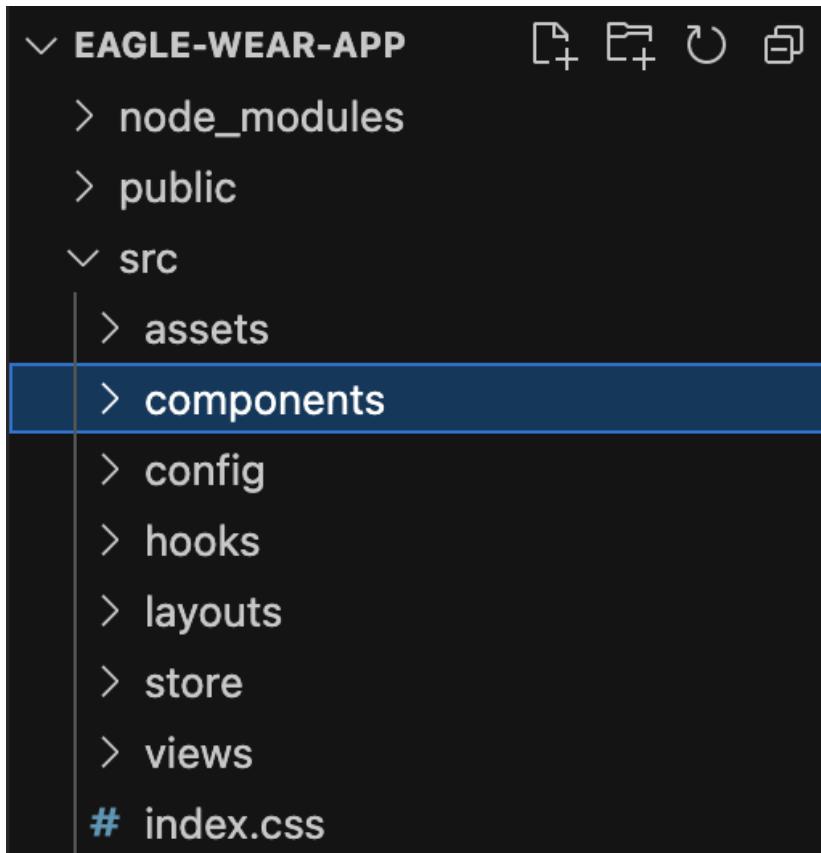
ln

Accedemos a la carpeta del proyecto, después instalamos las dependencias y levantamos el ambiente local.

Ahora limpiamos el proyecto, eliminamos los archivos que no necesitamos y quedaría algo como lo siguiente



Ahora creamos la estructura de carpetas para el proyecto, tratando siempre de que sea escalable, entendible y fácil de mantener.



La estructura de carpetas que definimos para este proyecto es la siguiente

- Assets: para los archivos estáticos, en especial imágenes
- components: contendrá los componentes React para construir la app
- config: configuraciones generales de la app
- hooks: si necesitamos crear un hook personalizado
- layouts: servirá para contener el header y footer de app
- store: contendrá el state global de la app, en este caso vamos a usar redux toolkit
- view: nos servirá para contener las vistas de la aplicación, serán los contenedores donde uniendo componentes se producirán las interfaces

El siguiente paso es agregar/installar las dependencias al proyecto que ya identificamos que vamos a usar las cuales son

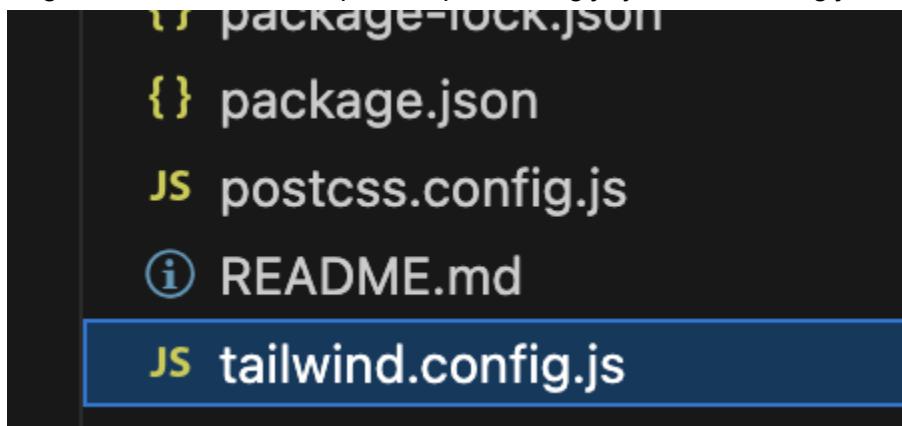
- Axios: con esta librería vamos a homologar las peticiones http. comando `npm install axios @types/axios`
- Redux: para este caso utilizaremos redux toolkit `npm install @reduxjs/toolkit react-redux`
- react router dom: usaremos la nueva versión de react router dom `npm install react-router-dom`
- Toastify: utilizaremos esta librería para crear alertas de una forma sencilla y visualmente `npm install react-toastify`

- Ahora instalamos tailwind que nos servirá para los estilos de la app `npm install -D tailwindcss postcss autoprefixer`

por el momento son los paquetes que tenemos identificados que son necesarios, en este punto aprovechamos para configurar tailwind para esto usamos el comando

```
npx tailwindcss init -p
```

se generan dos archivos que son post.config.js y tailwind.config.js



el archivo tailwind.config.js lo vamos a modificar para agregar la configuración

```
JS tailwind.config.js > ...
1  /** @type {import('tailwindcss').Config} */
2  export default {
3    content: [
4      "./index.html",
5      "./src/**/*.{js, jsx, ts, tsx}",
6    ],
7    theme: {
8      extend: {},
9    },
10   plugins: [
11     ],
12   },
13 }
14
```

Habíamos borrado el archivo index.css pero lo necesitamos para agregar las directivas necesarias para usar tailwind como se muestra en la imagen.

The screenshot shows a file tree on the left and a code editor on the right. The file tree includes node_modules, public, and a SRC folder containing assets, components, config, hooks, layouts, store, and views. The index.css file is selected in the tree, highlighted with a blue bar at the bottom. The code editor shows the following content:

```
src > # index.css
1 @tailwind base;
2 @tailwind components;
3 @tailwind utilities;
```

agregamos unas clases al index.html

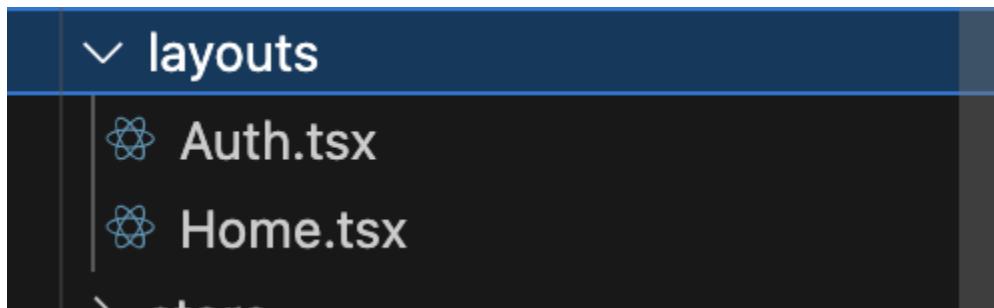
The screenshot shows the index.html file content:

```
<!doctype html>
<html lang="en" class="h-full bg-white">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Vite + React + TS</title>
  </head>
  <body class="h-full">
    <div id="root"></div>
    <script type="module" src="/src/main.tsx"></script>
  </body>
</html>
```

ahora empezamos con la configuración del router, para esto, dentro de src a la altura del archivo main.tsx agregamos el archivo router.tsx a continuación se muestra el archivo

```
src > ⚛ router.tsx > ...
1  import { createBrowserRouter } from "react-router-dom";
2  import Layout from "./layouts/Home";
3  import AuthLayout from "./layouts/Auth";
4
5  export const router = createBrowserRouter([
6   {
7     path: "/",
8     element: <AuthLayout />,
9     children: [...]
10  ],
11  {
12     path: "/products",
13     element: <Layout />,
14     children: [...]
15  },
16  ]);
17
```

como teníamos visualizado vamos a usar dos layouts uno para el login que llamamos auth y otro para listar los productos que lo llamaremos home, entonces creamos ambos componentes dentro de la carpeta layout



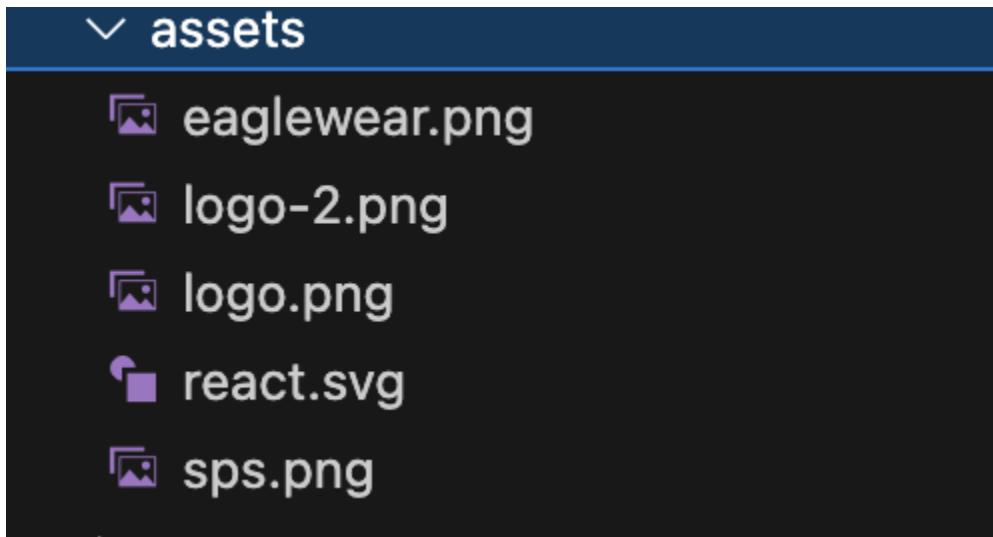
vamos a crear el Layout home

```

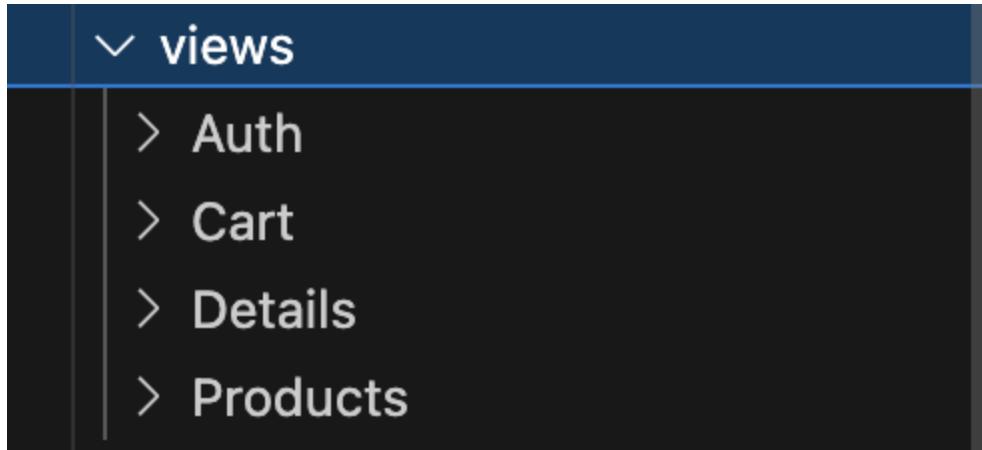
    return [
      <>
        <header className="bg-slate-800 fixed top-0 left-0 w-full flex gap-3 items-center justify-between z-50 shadow-md">
          <Link
            to="/products"
            className="rounded-md inline-flex pr-2 items-center text-sm font-semibold text-slate-500 shadow-sm hover:opacity-60">
            <img
              alt="Company logo"
              src={LogoImage}
              className="mx-auto h-20 w-auto"
            />
            <h1 className="hidden sm:block font-extrabold text-2xl text-red-500"> Eagle Wear </h1>
          </Link>
        </div> ...
      </div>
    </header>
    <Cart isOpen={cartOpen} onClose={handleCloseCart} />
    <ToastContainer />
    <main className="mt mx-auto max-w-6xl mt-10 p-12 bg-slate-100 shadow">
      <Outlet />
    </main>
    <Footer/>
  </>
];

```

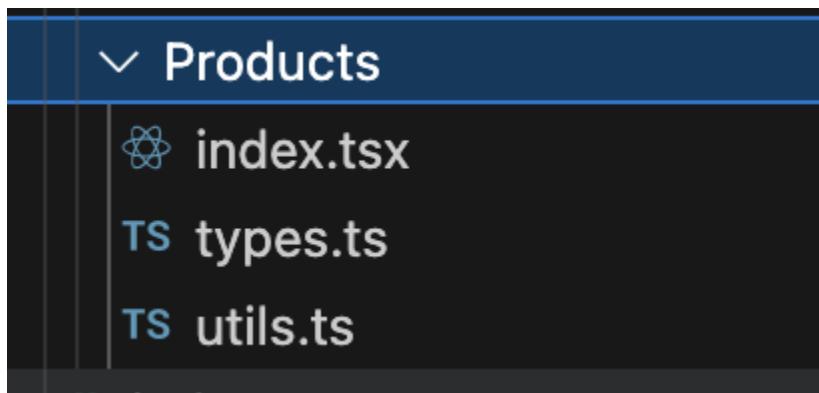
necesito las imágenes proporcionadas, entonces al ser en pdf hago una captura de pantalla de cada una y las guardo dentro del proyecto en src/assets, para quitar el fondo de la imagen usamos canva



de la misma forma creamos el layout para auth
ahora creamos las vistas que ya tenemos identificadas, para esto crearemos una carpeta por vista y dentro puede tener el componente que estará dentro de index.tsx, types.ts para agregar las interfaces y en caso de ser necesario util.ts para utilidades como generar requests



las vistas que tenemos identificadas con Auth/login, Cart, Detalles y Productos iniciamos por productos, esta es la estructura



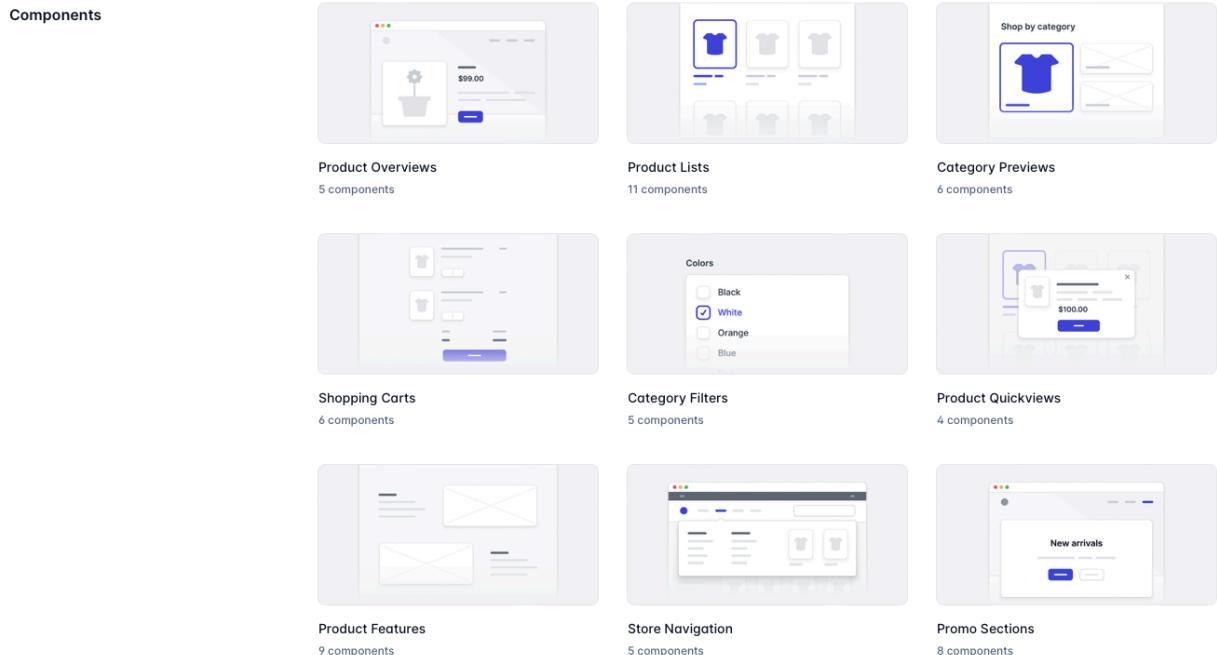
La misma estructura de archivos vamos a usar para los componentes

Listado de Productos

Ahora vamos a maquetar la vista de productos y nos ayudamos de tailwind, dentro de la documentación oficial <https://tailwindui.com/components> existen componentes ya creados que se pueden usar, algunos son gratuitos y otros de pago, únicamente usaremos los gratuitos

Ecommerce

Checkout forms, shopping carts, product views — everything you need to build your next ecommerce front-end.

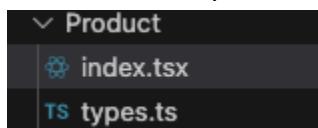


Hay una sección que se llama ecommerce y nos basamos en product list para la creación de la lista de productos.

como es una lista ocupamos un componente product para cada elemento de la lista y poder recorrer la lista y mapear la data que responde el servicio

GET <https://fakestoreapi.com/products>

Dentro de la carpeta components creamos el componente con la estructura



index.tsx será el functional component de Product y types tendrá los tipos que va a utilizar en el componente

Definimos la estructura de Product

```
export interface Products {
  id: number;
  title: string;
  price: number;
  category: string;
  description: string;
  image: string;
  rating: Rating;
}
```

para hacer esto usamos postman para ver la respuesta del servicio get Products

The screenshot shows the Postman interface. At the top, the URL is set to `https://fakestoreapi.com/products/`. Below the URL, there is a dropdown menu showing `GET` and a button labeled `Send`. Under the URL, there are tabs for `Params`, `Authorization`, `Headers (7)`, `Body`, `Pre-request Script`, `Tests`, and `Settings`. The `Headers (7)` tab is selected. In the `Query Params` section, there is a table with one row containing a key and a value. The `Body` tab is selected, showing a JSON response with 14 numbered lines. The response body is:

```
1 [ ] 
2   { 
3     "id": 1,
4     "title": "Fjallraven - Foldsack No. 1 Backpack, Fits 15 Laptops",
5     "price": 109.95,
6     "description": "Your perfect pack for everyday use and walks in the forest. Stash your laptop (up to 15 inches) in the padded sleeve, your everyday",
7     "category": "men's clothing",
8     "image": "https://fakestoreapi.com/img/81fPKd-2AYL._AC_SL1500_.jpg",
9     "rating": {
10       "rate": 3.9,
11       "count": 120
12     }
13   },
14 }
```

At the bottom right of the Postman interface, there is a status bar showing `Status: 200 OK Time: 1270 ms Size: 11.06 KB`.

Para empezar a hacer los requests vamos a configurar una instancia de axios



```
src > config > TS axiosConfig.ts > ...
1 import axios from "axios";
2
3 const API_URL = "https://fakestoreapi.com";
4
5 const axiosInstance = axios.create({
6   baseURL: API_URL,
7   headers: {
8     "Content-Type": "application/json",
9   },
10 });
11
12 export default axiosInstance;
13 |
```

Usaremos esta instancia de axios para hacer todas las peticiones

Dentro de la vista de Products que contiene la lista de productos vamos a crear la función que hace la petición para recuperar los products entonces este código lo agregamos dentro de utils.ts

```
src > views > Products > TS utils.ts > ...
1 import axios from "../../config/axiosConfig";
2 import { Products } from "./types";
3
4 export const getAllProducts = async (): Promise<Products[]> => {
5   const response = await axios.get<Products[]>('/products');
6   return response.data;
7 }
```

Dentro de la vista products ejecutamos el request en el useEffect

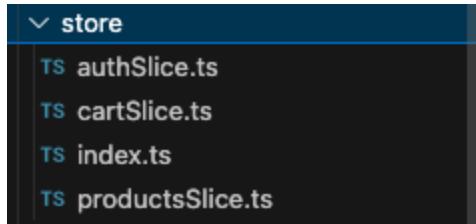
```
export const ProductsView = () => {

  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await getAllProducts();
        console.log(response);

      } catch (err) {
        console.log(err);
      }
    };
    fetchData();
  }, []);
  return (

```

En este punto necesitamos configurar redux para guardar los productos



Para este proyecto dividimos nuestro store en tres slice para contener los datos de carrito, auth y la lista de productos, de una vez creamos todos los archivos pero solo vamos a configurar productSlice

Dentro de index.ts creamos el store

```

src > store > ts productsSlice.ts > ...
1 import { createSlice, PayloadAction } from '@reduxjs/toolkit';
2 import { Products } from '../views/Products/types';
3
4 export interface ProductsState {
5   data: Products[];
6   products: Products[];
7   loading: boolean;
8 }
9
10 const initialState: ProductsState = {
11   data: [],
12   products: [],
13   loading: false,
14 };
15
16 const usersSlice = createSlice({
17   name: 'products',
18   initialState,
19   reducers: {
20     setData(state, action: PayloadAction<Products[]>) {
21       state.data = action.payload;
22       state.products = action.payload;
23     },
24     setLoading(state, action: PayloadAction<boolean>) {
25       state.loading = action.payload;
26     },
27   },
28 });
29
30 export const {
31   setData,
32   setLoading,
33 } = usersSlice.actions;
34
35 export default usersSlice.reducer;
36
37 |

```

Vamos a tener un state compuesto por data que es el listado de productos devueltos por el api, products que también es el listado de productos devueltos por el api pero que nos servirá para el filtrado o búsqueda de productos y loading qué es un booleano para mostrar un loader mientras se recupera información de la api

Ahora vamos a crear el store

```

src > store > ts index.ts > ...
1  import { configureStore } from '@reduxjs/toolkit';
2  import productsReducer from './productsSlice';
3
4  const store = configureStore({
5    reducer: {
6      products: productsReducer,
7    }
8  });
9
10 export type AppDispatch = typeof store.dispatch;
11 export type RootState = ReturnType<typeof store.getState>;
12
13 export default store;
14

```

Ya podemos guardar los datos en el store de productos y avanzamos con la vista de productos

```

11  export const ProductsView = () => {
12    const products = useSelector((state: RootState) => state.products.products);
13    const isLoading = useSelector((state: RootState) => state.products.loading);
14    const dispatch: AppDispatch = useDispatch();
15    useEffect(() => {
16      const fetchData = async () => {
17        dispatch(setLoading(true));
18        try {
19          const response = await getAllProducts();
20          dispatch(setData(response));
21          dispatch(setLoading(false));
22        } catch (err) {
23          console.log(err);
24          dispatch(setLoading(false));
25        }
26      };
27      fetchData();
28    }, [dispatch]);
29    return (
30      <div className="■bg-white">
31        <Categories />
32        <div className="mx-auto max-w-2xl px-4 py-16 sm:px-6 sm:py-10 lg:max-w-7xl lg:px-8">
33          {isLoading ? (
34            <Loading>Cargando Productos</Loading>
35          ) : (
36            <>
37              <h2 className="text-2xl font-bold tracking-tight □text-gray-900">
38                Lista de Productos
39              </h2>
40
41              <div className="mt-6 grid grid-cols-1 gap-x-6 gap-y-10 sm:grid-cols-2 lg:grid-cols-4 xl:gap-x-8">
42                {products.map((product) => (
43                  <Product key={product.id} product={product} />
44                )));
45              </div>
46            </>
47          )
48        </div>
49      </div>
50    );
51  };
52

```

Dentro de la petición que ya hacemos en useEffect hacemos dispatch de loading porque vamos a configurar un loader con ese dato y además hacemos dispatch de la data regresada por el servicio y procedemos a bindear la lista de productos

Creamos un componente llamado Loading para que el usuario tenga feedback de que pasa mientras se cargan los datos

```
src > components > Loading > index.tsx > [e] Loading
1  import { PropsWithChildren } from 'react'
2
3  const Loading = ({children}: PropsWithChildren) => {
4    return (
5      <div className='flex justify-center text-slate-400 font-semibold'>
6        <div>{children}</div>
7      </div>
8    )
9  }
10
11 export default Loading
```

Categorías y búsqueda

Creamos un componente categorías que lo vamos a usar en el listado de productos, para esto vemos que el api regresa categorías y productos por categoría

Dentro de components creamos el componente categories que va a listar las categorías devueltas por el api

```
src > components > Categories > index.tsx > Categories
19  export const Categories = () => {
62    return (
63      <nav>
64        <div className="mx-auto flex max-w-7xl items-center justify-between p-2 sm:p-4 md:p-8 overflow-hidden">
65          <div className="flex flex-wrap gap-x-3">
66            {categories.map((category) => (
67              <Button
68                key={category}
69                onClick={() => handleChangeCategory(category)}
70                className={`text-sm leading-6 capitalize ${{
71                  categoryActive === category
72                    ? "text-red-500 font-extrabold "
73                    : "text-slate-500 font-semibold hover:text-slate-500"
74                }}`}
75              >
76                {category}
77              </Button>
78            ))}
79            <Button
80              onClick={() => handleChangeCategory("All")}
81              className={`text-sm leading-6 capitalize ${{
82                  categoryActive === "All"
83                    ? "text-red-500 font-extrabold "
84                    : "text-slate-500 font-semibold hover:text-slate-500"
85                }}`}
86            >
87              All
88            </Button>
89          </div>
90        </div>
91      </div>
92    )
93  )
```

Creamos una variable de estado para capturar la categoría seleccionada

```

src > components > Categories > index.tsx > Categories > categories
17 import { MagnifyingGlassIcon, XMarkIcon } from "@heroicons/react/24/outline";
18
19 export const Categories = () => {
20   const dispatch: AppDispatch = useDispatch();
21
22   const [categories, setCategories] = useState<string>([]);
23   const [categoryActive, setCategoryActive] = useState<string>("All");
24   const [filter, setFilter] = useState<string>("");
25
26   useEffect(() => {
27     const fetchCategories = async () => {
28       try {
29         const response = await getCategories();
30         setCategories(response);
31         console.log(response);
32       } catch (error) {
33         console.log(error);
34       }
35     };
36
37     fetchCategories();
38   }, []);
39   const handleChangeCategory = (category: string) => {
40     setCategoryActive(category);
41     setFilter("");
42     fetchProductsCategory(category);
43   };
44   const fetchProductsCategory = async (category: string) => {
45     dispatch(setLoading(true));
46     try {
47       const response = await getProductByCategory(category);
48       dispatch(setData(response));
49       dispatch(setLoading(false));
50     } catch (err) {
51       console.log(err);
52       dispatch(setLoading(false));
53     }
54   };
55   const handleFilterProducts = () => {
56     dispatch(setFilteredProducts(filter));
57   };
58   const handleCancelFilterProducts = () => {
59     setFilter("");
60     dispatch(setFilteredProducts(""));
61   };

```

Capturamos la acción de click sobre cada categoría y a su vez hacemos un request de productos por categoría

```
src > views > Products > ts utils.ts > ...
1  import axios from "../../config/axiosConfig";
2  import { Products } from "./types";
3
4  export const getAllProducts = async (): Promise<Products[]> => {
5    const response = await axios.get<Products[]>('/products');
6    return response.data;
7  }
8
9  export const getProductByCategory = async (category: string): Promise<Products[]> => {
10    if (category === 'All') {
11      return await getAllProducts();
12    }
13    const response = await axios.get<Products[]>('/products/category/${category}`);
14    return response.data;
15  }
16
```

En el caso de que la categoría seleccionada el All lo que hacemos es devolver getAllProducts de lo contrario hacemos el fetch de products/category

Y se guarda en el store de esa forma se actualizan los elementos que se muestran en el listado de productos

También agregamos una barra de búsqueda, en este caso va a ser trabajada desde el Front, hará una búsqueda en el store, para esto creamos un nuevo reducer setFilteredProducts

```

src > store > ts productsSlice.ts > usersSlice > reduces > setFilteredProducts
1 import { createSlice, PayloadAction } from '@reduxjs/toolkit';
2 import { Products } from '../views/Products/types';
3
4 export interface ProductsState {
5   data: Products[];
6   products: Products[];
7   loading: boolean;
8 }
9
10 const initialState: ProductsState = {
11   data: [],
12   products: [],
13   loading: false,
14 };
15
16 const usersSlice = createSlice({
17   name: 'products',
18   initialState,
19   reducers: {
20     setData(state, action: PayloadAction<Products[]>) {
21       state.data = action.payload;
22       state.products = action.payload;
23     },
24     setLoading(state, action: PayloadAction<boolean>) {
25       state.loading = action.payload;
26     },
27     setFilteredProducts(state, action: PayloadAction<string>) {
28       const filtered = state.data.filter( product => product.title.toLowerCase().includes(action.payload.toLowerCase()) );
29       state.products = filtered;
30       state.loading = false;
31     }
32   },
33 });
34
35 export const {
36   setData,
37   setLoading,
38   setFilteredProducts
39 } = usersSlice.actions;
40
41 export default usersSlice.reducer;
42

```

Lo que hace es filtrar en el store por coincidencia de nombre

Carrito

Para la vista de carrito vamos a usar un componente de tailwind

<https://tailwindui.com/components/e-commerce/components/shopping-carts>

Que me parece muy bien visualmente

Para esto me copio el código desde tailwind dentro de la vista cart simplemente para ver si funciona y cómo implementarlo en el proyecto y tengo un error al intentar usar el componente

cart de tailwind

```
2 import { useState } from 'react'
3 import { Dialog, DialogBackdrop, DialogPanel, DialogTitle } from '@headlessui/react'
4 import { XMarkIcon } from '@heroicons/react/24/outline'
5
6 const products = [
7   {
8     id: 1,
9     name: 'Throwback Hip Bag',
10    href: '#',
11    color: 'Salmon',
12    price: '$90.00',
13    quantity: 1,
14    imageSrc: 'https://tailwindui.com/img/ecommerce-images/shopping-cart-page-04-product-01.jpg',
15    imageAlt: 'Salmon orange fabric pouch with match zipper, gray zipper pull, and adjustable hip belt.',
16  },
17 },
18 {
19   id: 2,
20   name: 'Medium Stuff Satchel',
21   href: '#',
22   color: 'Blue'
```

PROBLEMS 9 OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE Filter (e.g. text, **/*.ts, !**/node_modules/**)

index.tsx src/views/Cart 2

- Cannot find module '@headlessui/react' or its corresponding type declarations. ts(2307) [Ln 4, Col 66]
- Cannot find module '@heroicons/react/24/outline' or its corresponding type declarations. ts(2307) [Ln 5, Col 27]

Busco los dos modulos que estan causando problemas y así entender para que se usan
En el caso de heroiconos es iconografía y no estamos usando ninguna, nos sirve para las
acciones entonces la vamos a agregar, seguimos las instrucciones de la documentación
<https://github.com/tailwindlabs/heroicons>

React

First, install `@heroicons/react` from npm:

```
npm install @heroicons/react
```

Now each icon can be imported individually as a React component:

```
import { BeakerIcon } from '@heroicons/react/24/solid'

function MyComponent() {
  return (
    <div>
      <BeakerIcon className="size-6 text-blue-500" />
      <p>...</p>
    </div>
  )
}


```

En el caso de headlessui es un set de UI components que nosotros podríamos crear desde cero solo usando tailwind pero por efectos de tiempo los utilizaremos

<https://headlessui.com>

Sequimos la documentacion para instalarlo <https://github.com/tailwindlabs/headlessui>

Ahora ya no tenemos problemas en el componente

```
1 import { useState } from 'react'
2 import { Dialog, DialogBackdrop, DialogPanel, DialogTitle } from '@headlessui/react'
3 import { XMarkIcon } from '@heroicons/react/24/outline'

4 const products = [
5   {
6     id: 1,
7     name: 'Throwback Hip Bag',
8     href: '#',
9     color: 'Salmon',
10    
```

Para integrarlo dentro del layout de home que previamente creamos agregamos un botón para acceder al carrito

```
1 > layouts > Home.tsx > [o] Layout
2
3
4 const Layout = () => {
5   return []
6   <>
7     <header className="bg-slate-800">
8       <div className="mx-auto max-w-6xl py-10">
9         <h1 className="text-4xl font-extrabold text-white">Titulo</h1>
10        <Button className="inline-flex items-center gap-2 rounded-md bg-gray-700 py-1.5 px-3 text-sm/6 font-semibold">
11          carrito
12        </Button>
13      </div>
14    </header>
15
16    <main className="mt mx-auto max-w-6xl p-10 bg-slate-200 shadow">
17      <Outlet />
18    </main>
19  </>
20  [];
21}
22
23 export default Layout;
```

Desde aquí controlamos cuando se abre el carrito entonces ocupamos una variable de estado

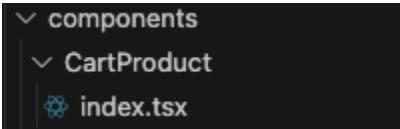
Ahora lo integramos en el layout

```
src > layouts > Home.tsx > [o] Layout
  5
  6  const Layout = () => {
  7    |   const [cartOpen, setCartOpen] = useState(false);
  8    |   const handleOpenCart = () => {
  9    |     setCartOpen(true);
 10   }
 11   const handleCloseCart = () => {
 12     |     setCartOpen(false);
 13   }
 14   return (
 15     <>
 16       <header className="bg-slate-800">
 17         <div className="mx-auto max-w-6xl py-10">
 18           <h1 className="text-4xl font-extrabold text-white">Titulo</h1>
 19           <Button
 20             |   className="inline-flex items-center gap-2 rounded-md bg-gray-700 py-1.5 px-3 text-sm/6 font-semibold text-white"
 21             |   onClick={handleOpenCart}
 22           >
 23             |   carrito
 24           </Button>
 25         </div>
 26       </header>
 27       <Cart
 28         |   isOpen={cartOpen}
 29         |   onClose={handleCloseCart}
 30       />
 31       <main className="mt mx-auto max-w-6xl p-10 bg-slate-200 shadow">
 32         <Outlet />
 33       </main>
 34     </>
 35   );
 36 }
 37
 38 export default Layout;
 39
```

dentro del la vista de cart queda algo como esto

```
src > views > Cart > ✎ index.tsx > [o] Cart
28 ]
29
30 export const Cart = ({isOpen, onClose} : CartProps) => {
31
32     return (
33         <Dialog open={isOpen} onClose={onClose} className="relative z-10">
34             <DialogBackdrop
35                 transition
36                 className="fixed inset-0 bg-gray-500 bg-opacity-75 transition-opacity duration-500 ease-in-out data-[close]"
37             />
38
39             <div className="fixed inset-0 overflow-hidden">
40                 <div className="absolute inset-0 overflow-hidden">
41                     <div className="pointer-events-none fixed inset-y-0 right-0 flex max-w-full pl-10">
42                         <DialogPanel
43                             transition
44                             className="pointer-events-auto w-screen max-w-md transform transition duration-500 ease-in-out data-[close]"
45                         >
46                             <div className="flex h-full flex-col overflow-y-scroll bg-white shadow-xl">
47                                 <div className="flex-1 overflow-y-auto px-4 py-6 sm:px-6">
48                                     <div className="flex items-start justify-between">
49                                         <DialogTitle className="text-lg font-medium text-gray-900">Shopping cart</DialogTitle>
50                                         <div className="ml-3 flex h-7 items-center">
51                                             <button
52                                                 type="button"
53                                                 onClick={onClose}
54                                                 className="relative -m-2 p-2 text-gray-400 hover:text-gray-500"
55                                             >
56                                                 <span className="absolute -inset-0.5" />
57                                                 <span className="sr-only">Close panel</span>
58                                                 <XMarkIcon aria-hidden="true" className="h-6 w-6" />
59                                             </button>
60                                         </div>
61                                     </div>
62                                 </div>
63                             </div>
64                         </div>
65                     </div>
66                 </div>
67             </div>
68         </Dialog>
69     )
70 }
```

Funciona correctamente, a partir de aquí lo que hago es crear un componente para los items del carrito



Configuramos el slice para cart

```
src > store > TS cartSlice.ts > [+] initialState > ↴ data
1  import { createSlice, PayloadAction } from '@reduxjs/toolkit';
2  import { CartProduct, UserCartProps } from '../views/Cart/types';
3
4  export interface CartState {
5    data: UserCartProps;
6    loading: boolean;
7  }
8  const initialState: CartState = {
9    data: {
10      id: 0,
11      userId: 0,
12      date: '',
13      products: []
14    },
15    loading: false,
16  };
17
18
19  const usersSlice = createSlice({
20    name: 'cart',
21    initialState,
22    reducers: {
23      setCart(state, action: PayloadAction<UserCartProps>) {
24        state.data = action.payload;
25      },
26      setLoading(state, action: PayloadAction<boolean>) {
27        state.loading = action.payload;
28      },
29      setNewProductCount(state, action: PayloadAction<CartProduct>) {
30        state.products = action.payload;
31      }
32    }
33  );
34
```

Dentro de la vista de cart, creamos el archivo utils para hacer la petición y recuperar el carrito lo guardamos en el store y lo mostramos

```
src > views > Cart > index.tsx > [e] Cart
17  export const Cart = ({ isOpen, onClose }: CartProps) => {
66
67      <div className="mt-8">
68          <div className="flow-root">
69              {
70                  products.length>0 ? (
71                      <ul
72                          role="list"
73                          className="!-my-6 divide-y divide-gray-200"
74                      >
75                          {products.map((product) => (
76                              <CartProduct key={product.id} product={product} />
77                          )))
78                      </ul>
79                  ) : <p className="text-center">No hay productos en el carrito</p>
80              }
81
82          </div>
83      </div>
84  </div>
85
```

hasta ahora recuperamos el cart dentro de la view cart pero no podemos continuar con este flujo ya que dentro de layout estamos mostrando la cantidad de productos en el carrito Entonces al montarse el componente layout vamos a hacer el request y guardar los datos de carrito en el store

dentro de Home Layout

```
src > layouts > Home.tsx > Layout
18  const Layout = () => {
23
24    const { products } = useSelector((state: RootState) => state.cart.data);
25    useEffect(() => {
26      const getCart = async () => {
27        try {
28          const response = await getUserCart(idUser);
29          console.log(response[0]);
30          dispatch(setCart(response[0]));
31        } catch (error) {
32          console.log(error);
33        }
34      };
35      getCart();
36    }, []);
37    const handleOpenCart = () => {
38      setCartOpen(true);
39    };
40    const handleCloseCart = () => {
41      setCartOpen(false);
42    };
43    const handleLogout = () => {
44      dispatch(setLogout(""));
45    };

```

Así podemos mostrar la cantidad de items en el carrito desde la vista principal, al ya estar guardada la información en el store, en la vista de cart sólo accedemos a ella para bindear el carrito

Al revisar el servicio GET carrito por usuario <https://fakestoreapi.com/carts/5>

```
1  [
2      "id": 5,
3      "userId": 3,
4      "date": "2020-03-01T00:00:00.000Z",
5      "products": [
6          {
7              "productId": 7,
8              "quantity": 1
9          },
10         {
11             "productId": 8,
12             "quantity": 1
13         }
14     ],
15     "__v": 0
16 ]
```

Solo tenemos el id del producto y vamos a mostrar precio, imagen, nombre, etc en la interface
Analizando lo que podemos hacer es que por cada elemento del carrito hacer un request GET product por id <https://fakestoreapi.com/products/1> que nos da la informacion detallada del producto

```
1   [
2     {
3       "id": 1,
4       "title": "Fjallraven - Foldsack No. 1 Backpack, Fits 15 Laptops",
5       "price": 109.95,
6       "description": "Your perfect pack for everyday use and walks in the forest. Stash your
7                     everyday",
8       "category": "men's clothing",
9       "image": "https://fakestoreapi.com/img/81fPKd-2AYL._AC_SL1500_.jpg",
10      "rating": {
11        "rate": 3.9,
12        "count": 120
13      }
14    ]
15  ]
```

entonces creamos un hook que nos ayude con esto, dentro de hooks

```

src > hooks > ✨ useGetCartProducts.tsx > [!] useFetchCartProducts
  1  import { useState, useEffect } from 'react';
  2  import { CartProduct } from '../views/Cart/types';
  3  import { Products } from '../views/Products/types';
  4  import { getProductByID } from '../views/Products/utils';
  5
  6  !
  7  const useFetchCartProducts = (cartProducts: CartProduct[]) => {
  8    const [products, setProducts] = useState<Products[]>([]);
  9    const [loading, setLoading] = useState<boolean>(true);
 10    const [error, setError] = useState<string | null>(null);
 11
 12    useEffect(() => {
 13      const fetchProductData = async () => {
 14        try {
 15          const productsDetails = cartProducts.map(async (cartProduct) => {
 16            const response = await getProductByID(cartProduct.productId);
 17            return {...response, quantity: cartProduct.quantity};
 18          });
 19
 20          const productsData = await Promise.all(productsDetails);
 21          setProducts(productsData);
 22        } catch (err) {
 23          setError('Failed to fetch product data');
 24        } finally {
 25          setLoading(false);
 26        }
 27      };
 28
 29      fetchProductData();
 30    }, [cartProducts]);
 31
 32    return { products, loading, error };
 33  };
 34
 35  export default useFetchCartProducts;
 36

```

lo implementamos en la vista cart

```

src > views > Cart > ✨ index.tsx > [!] Cart > [!] onSubmit
  16
  17  export const Cart = ({ isOpen, onClose }: CartProps) => {
  18    const dispatch: AppDispatch = useDispatch();
  19
  20    const { products: cartProducts } = useSelector(
  21      (state: RootState) => state.cart.data
  22    );
  23    const { products } = useFetchCartProducts(cartProducts);
  24
  25

```

De esa forma tenemos acceso a toda la información del producto y la cantidad de cada uno

Para mostrar el subtotal debemos hacer cálculos multiplicando la cantidad por el precio del producto, para esto podemos crear otro hook, pero en realidad ya tenemos la lista de productos con cantidad y precio que nos devuelve el hook que acabamos de hacer, entonces solo ocupamos hacer la suma de (precio x cantidad), para esto dentro de utils de cart creamos una función getSubtotal

```
src > views > Cart > ts utils.ts > ...
1  import axios from "....../config/axiosConfig";
2  import { Products } from "../Products/types";
3  import { UserCartProps } from "./types";
4
5
6  export const getUserCart = async (idUser: number): Promise<UserCartProps[]> => {
7    const response = await axios.get<UserCartProps[]>(`/carts/user/${idUser}`);
8    return response.data;
9  };
10 export const getSubtotal = (cartProducts: Products[]): number => {
11   let subtotal = 0;
12   cartProducts.forEach(product => {
13     subtotal += (product.price * product.quantity);
14   });
15   return subtotal;
16 }
17
18
19
```

En la vista cart solo bindeamos la función que devuelve el dato y le pasamos products que es la lista de productos devuelta por el hook creado

```
<div className="border-t border-gray-200 px-4 py-6 sm:px-6">
  <div className="flex justify-between text-base font-medium text-gray-900">
    <p>Subtotal</p>
    <p>${getSubtotal(products)}</p>
  </div>
  <p className="mt-0.5 text-sm text-gray-500">
    Los impuestos se calcularán el finalizar la compra
  </p>
</div>
```

Hasta este punto estamos haciendo la recuperación de carrito con un id de usuario hardcoded, entonces vamos a trabajar con el login de usuario

Creamos un layout para auth, pensando que posteriormente se use también para registro de usuarios

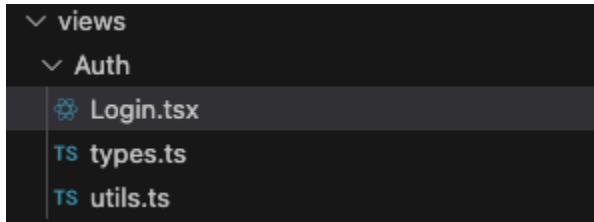
```

src > layouts > Auth.tsx
12  const AuthLayout = () => {
13
14  return [
15    <div className="flex min-h-full flex-1 flex-col justify-center px-6 py-12 lg:px-8">
16      <div className="sm:mx-auto sm:w-full sm:max-w-sm">
17        <img
18          alt="Company logo"
19          src={LogoImage}
20          className="mx-auto h-20 w-auto"
21        />
22
23        <h2 className="mt-10 text-center text-2xl font-bold leading-9 tracking-tight text-gray-900">
24          Inicio de Sesión
25        </h2>
26        <div className="mt-10 sm:mx-auto sm:w-full sm:max-w-sm">
27          <Outlet />
28          <ToastContainer />
29        </div>
30
31        </div>
32      </div>
33    ];
34
35  export default AuthLayout;
36
37
38
39
40
41
42
43
44
45
46
47
48
49

```

Login

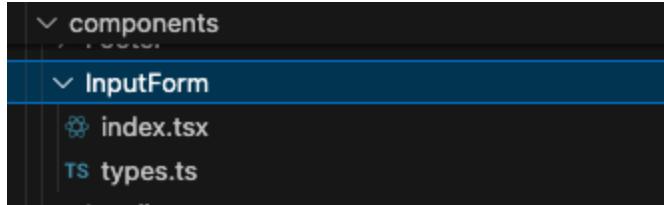
Vamos a trabajar la vista de Login



Ocupamos de un formulario de login y vamos a usar una vista que nos da tailwind
<https://tailwindui.com/components/application-ui/forms/sign-in-forms>

Depuramos el código cambiamos la imagen cambiamos textos y quitamos cosas que no nos sirven

Teniendo en cuenta que es un aplicación que sea escalable y se puedan registrar usuarios, agregar productos, editar, etc. vamos a crear un elemento InputForm Personalizado, con la idea de que lo podamos utilizar en cualquier formulario



Vamos a utilizar la librería react-hook-form para el manejo de formularios

<https://react-hook-form.com/>

Así quedaría el componente

```
src > components > InputForm > index.tsx > ...
1  import ErrorMessage from "../ErrorMessage";
2  import { InputFormProps } from "./types";
3
4  export const InputForm = ({ ...
5    label,
6    name,
7    register,
8    errors,
9    type = "text",
10   placeholder,
11   validationRules,
12 }: InputFormProps) => {
13   return (
14     <div className="mb-4">
15       <label
16         className="block text-sm font-medium leading-6 text-gray-900"
17         htmlFor={name}
18       >
19         {label}
20       </label>
21       <input
22         className="block w-full rounded-md border-0 py-1.5 text-gray-900 shadow-sm ring-1 ring-inset ring-gray-300 placeholder:pl-2"
23         type={type}
24         id={name}
25         placeholder={placeholder}
26         {...register(name, validationRules)}
27       />
28       {errors && <ErrorMessage>{errors.message}</ErrorMessage>}
29     </div>
30   );
31 }
32
```

En la vista de Login agregamos los input necesarios

```
src > views > Auth > Login.tsx > Login > onSubmit
12  export const Login = () => {
13    ,
14    return (
15      <>
16      <form onSubmit={handleSubmit(onSubmit)} className="space-y-6">
17        <InputForm
18          label="Usuario"
19          name="username"
20          register={register}
21          errors={errors.username}
22          placeholder="Usuario"
23          validationRules={{ required: 'Usuario es requerido' }}
24        />
25
26        <InputForm
27          label="Password"
28          name="password"
29          register={register}
30          errors={errors.password}
31          placeholder="Password"
32          type="password"
33          validationRules={{ required: 'Password es requerido' }}
34        />
35
36        <div>
37          <button
38            type="submit"
39            disabled={isLoading}
40            className="flex w-full justify-center rounded-md bg-indigo-600 px-3 py-1.5 text-sm font-semibold leading-6"
41          >
42            Iniciar Sesión
43          </button>
44        </div>
45      </form>
46
47      <p className="mt-10 text-center text-sm text-gray-500">
48        No tienes cuenta?{' '}
49        <a href="#" className="font-semibold leading-6 text-indigo-600 hover:text-indigo-500">
50          Crear cuenta gratis
51        </a>
52      </p>
53    </>
54  )
55 }
```

```
src > views > Auth >  Login.tsx >  Login
11
12  export const Login = () => {
13    const dispatch: AppDispatch = useDispatch();
14    const navigate = useNavigate();
15
16    const isLoading = useSelector((state: RootState) => state.auth.loading);
17
18    const {
19      register,
20      handleSubmit,
21      formState: { errors },
22    } = useForm<LoginFormProps>({
23      defaultValues: {
24        username: '',
25        password: ''
26      },
27      mode: "onChange",
28    });
29    const onSubmit = async (data: LoginFormProps) => {
30      try {
31        dispatch(setLoading(true));
32        const response = await authenticate(data);
33        console.log(response);
34
35        dispatch(setLogin({user: data.username, token: response.token, idUser: getIdUserByToken(response.token ?? '')}));
36        navigate('/products', { replace: true });
37
38      } catch (error: any) {
39        dispatch(setLoading(false));
40        console.log(error.code);
41        if (error.code === 'ERR_BAD_REQUEST') {
42          toast.error("Credenciales incorrectas!");
43
44        }else
45          toast.error("Ha ocurrido un error intentelo nuevamente");
46
47      }
48    };

```

Configuramos el store para guardar los datos de la sesión

```

src > store > ts authSlice.ts > [o] authSlice > [f] reducers > [o] setLogout > [f] idUser
  1 import { createSlice, PayloadAction } from '@reduxjs/toolkit';
  2 import { User } from '../views/Auth/types';
  3
  4 export interface AuthState {
  5   user: User;
  6   isAuthenticated: boolean;
  7   loading: boolean;
  8 }
  9
 10 const initialState: AuthState = {
 11   loading: false,
 12   isAuthenticated: false,
 13   user: {
 14     user: '',
 15     token: null,
 16     idUser: 0
 17   }
 18 };
 19
 20 const authSlice = createSlice({
 21   name: 'auth',
 22   initialState,
 23   reducers: {
 24     setLogin(state, action: PayloadAction<User>) {
 25       state.user = action.payload;
 26       state.isAuthenticated = true;
 27       localStorage.setItem('token', action.payload.token ?? '');
 28     },
 29     setLoading(state, action: PayloadAction<boolean>) {
 30       state.loading = action.payload;
 31     },
 32     setLogout(state, action: PayloadAction<string>) {
 33       state.user = { user: action.payload, token: null, idUser: 0 };
 34       state.isAuthenticated = false;
 35       localStorage.removeItem('token');
 36     },
 37   }
 38 });
 39
 40 export const {
 41   setLoading,
 42   setLogin,
 43   setLogout
 44 } = authSlice.actions;
 45
 46 export default authSlice.reducer;
 47

```

Pensando en que se pueda guardar la sesión a pesar de cerrar la pestaña del navegador si el usuario no ha hecho logout guardamos el token en storage

Todo perfecto el usuario puede hacer login pero para recuperar el carrito ocupamos el id de usuario y no lo tengo, lo que podemos hacer es revisar si viene en el token

En <https://jwt.io/> veo que en efecto tengo el id en el token y puedo usarlo

Encoded PASTE A TOKEN HERE

yJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJsb2dpbiJ9.eyJpc3MiOiJsb2dpbiJ9.eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjoiZm9yLTIzMTUiLCJpYXQiOjE2MjA3MTA2MDAsImhdCI6MTcyMDcxMDI2MH0.GuFo0ghXKBBrQTLE3cRaOpJAS8-IMXJelg0WkTEhSRQk

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE
{} (empty)
PAYLOAD: DATA
{ "sub": 2, "user": "mor_2314", "iat": 1720710260 }
VERIFY SIGNATURE
HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), your-256-bit-secret) <input type="checkbox"/> secret base64 encoded

Para decodificar el token usaremos la librería jwt-decode, la instalamos para poder usarla <https://www.npmjs.com/package/jwt-decode>

```
src > views > Auth > Login.tsx > Login
12  export const Login = () => {
13    ...
29    const onSubmit = async (data: LoginFormProps) => {
30      try {
31        dispatch(setLoading(true));
32        const response = await authenticate(data);
33        console.log(response);
34
35        dispatch(setLogin({user: data.username, token: response.token, idUser: getIdUserByToken(response.token ?? '')}));
36        navigate('/products', { replace: true });
37
38      } catch (error: any) {
39        dispatch(setLoading(false));
40        console.log(error.code);
41        if (error.code === 'ERR_BAD_REQUEST') {
42          toast.error("Credenciales incorrectas!");
43
44        } else
45          toast.error("Ha ocurrido un error intentelo nuevamente");
46
47      }
48    };

```

De esta forma queda la recuperación y el guardado de los datos del usuario

En utils tenemos tanto la petición de login como la decodificación para recuperar el id de usuario

```
src > views > Auth > ts utils.ts > ...
1  import axios from "../../config/axiosConfig";
2  import { DecodedToken, User } from "./types";
3  import { LoginFormProps } from "./types";
4  import { jwtDecode } from "jwt-decode";
5
6  export const authenticate = async (user: LoginFormProps): Promise<User> => {
7    const response = await axios.post<User>('/auth/login', user);
8    return response.data;
9  };
10
11 export const getIdUserByToken = (token: string): number => {
12   try {
13     const decodedToken: DecodedToken = jwtDecode(token);
14     return decodedToken.sub;
15   } catch (error) {
16     return 0;
17   }
18};
```

Ya tenemos el token ahora procedemos a usarlo en las peticiones, para esto podemos agregarlo en cada una de las peticiones que hacemos con axios, pero no es la mejor forma, lo que haremos es un interceptor que agregue a las peticiones el token en caso de existir

Regresamos al archivo donde configuramos la instancia de axios y agregamos el interceptor

```
src > config > TS axiosConfig.ts > ...
1 import axios from "axios";
2
3 const API_URL = "https://fakestoreapi.com";
4
5 const axiosInstance = axios.create({
6   baseURL: API_URL,
7   headers: {
8     "Content-Type": "application/json",
9   },
10 });
11 axiosInstance.interceptors.request.use(
12   config => {
13     const token = localStorage.getItem('token');
14
15     if (token) {
16       config.headers.Authorization = `Bearer ${token}`;
17     }
18
19     return config;
20   },
21   error => {
22     return Promise.reject(error);
23   }
24 );
25
26 export default axiosInstance;
27
```

Para este punto ya tenemos nuestra routing completo

```

src > router.tsx > ...
1  import { createBrowserRouter } from "react-router-dom";
2  import DetailsView from "./views/Details";
3  import Layout from "./layouts/Home";
4  import AuthLayout from "./layouts/Auth";
5  import { Login } from "./views/Auth/Login";
6  import ProtectedRoutes from "./protectedRoutes";
7  import { ProductsView } from "./views/Products";
8
9  export const router = createBrowserRouter([
10    {
11      path: "/",
12      element: <AuthLayout />,
13      children: [
14        {
15          index: true,
16          element: <Login />,
17        },
18        {
19          path: '/sing-up',
20          element: <Login />,
21        },
22      ],
23    },
24    {
25      path: "/products",
26      element: (
27        <ProtectedRoutes>
28          <Layout />
29        </ProtectedRoutes>
30      ),
31      children: [
32        {
33          index: true,
34          element: <ProductsView />,
35        },
36        {
37          path: "/products/:productId",
38          element: <DetailsView />,
39        }
40      ],
41    },
42  ]);

```

Pero cualquier usuario puede acceder a cualquier ruta, dentro del requerimiento dice que el usuario debe iniciar sesión, entonces vamos a agregar una validación extra para proteger las rutas.

Crearemos un componente ProtectedRoutes

```
src > 📂 protectedRoutes.tsx > ...
1 import { useSelector } from 'react-redux';
2 import { Navigate } from 'react-router-dom';
3 import { RootState } from './store';
4
5 interface ProtectedRouteProps {
6   children: JSX.Element;
7 }
8
9 const ProtectedRoutes = ({ children }: ProtectedRouteProps) => {
10   const isAuthenticated = useSelector((state: RootState) => state.auth.isAuthenticated);
11   if (!isAuthenticated) {
12     return <Navigate to="/" />;
13   }
14
15   return children;
16 };
17
18 export default ProtectedRoutes;
```

De esa forma solo los usuarios que inician sesión pueden acceder a la lista de productos y carrito

No recordaba como hacerlo entonces al buscar en internet me encuentro con <https://github.com/remix-run/react-router/issues/10637> me baso en esa información para agregarlo

Ahora tengo dos errores de los que me di cuenta uno que el usuario puede regresar a login y otro que no se guarda la sesión y tengo que estar constantemente iniciando sesión al guardar cambios

Para solucionar esto dentro del layout de auth

```
src > layouts > Auth.tsx > [AuthLayout]
1 import { Outlet, useNavigate } from "react-router-dom";
2 import LogoImage from '../assets/logo-2.png'
3
4 import { ToastContainer } from "react-toastify";
5 import "react-toastify/dist/ReactToastify.css";
6 import { useEffect } from "react";
7 import { getIdUserByToken, getUserByToken } from "../views/Auth/utils";
8 import { setLogin } from "../store/authSlice";
9 import { useDispatch } from "react-redux";
10 import { AppDispatch } from "../store";
11
12 const AuthLayout = () => {
13   const dispatch: AppDispatch = useDispatch();
14   const navigate = useNavigate();
15
16   const token = localStorage.getItem('token');
17
18   useEffect(() => {
19     if (token) {
20       dispatch(setLogin({user: getUserByToken(token), token: token, idUser: getIdUserByToken(token)}));
21       navigate('/products', { replace: true });
22     }
23   }, [token])
```

Primero revisar si existe token una vez que existe token, si existe voy a redireccionar a /products pero adicional necesito guardar el usuario en sesión, entonces creo otra función para recuperar el nombre de usuario del token y se guarda en el store para poder usarlo

```
src > views > Auth > ts utils.ts > ...
1  import axios from "../../config/axiosConfig";
2  import { DecodedToken, User } from "./types";
3  import { LoginFormProps } from "./types";
4  import { jwtDecode } from "jwt-decode";
5
6  export const authenticate = async (user: LoginFormProps): Promise<User> => {
7    const response = await axios.post<User>('/auth/login', user);
8    return response.data;
9  };
10
11 export const getIdUserByToken = (token: string): number => {
12  try {
13    const decodedToken: DecodedToken = jwtDecode(token);
14    return decodedToken.sub;
15  } catch (error) {
16    return 0;
17  }
18};
19 export const getUserByToken = (token: string): string => {
20  try {
21    const decodedToken: DecodedToken = jwtDecode(token);
22    return decodedToken.user;
23  } catch (error) {
24    return '';
25  }
26};
```

Detalle de Producto

Quiero agregar una pantalla con el detalle del producto, para esto en vistas creamos details, en el router ya está configurada una vista detalle

```

src > views > Details > index.tsx > [?] DetailsView
13
14  const DetailsView = () => {
15    const { productId } = useParams();
16    const [product, setProduct] = useState<Products>();
17    const dispatch: AppDispatch = useDispatch();
18    const isLoading = useSelector((state: RootState) => state.products.loading);
19
20    useEffect(() => {
21      const fetchData = async () => {
22        dispatch(setLoading(true));
23        try {
24          const response = await getProductByID(+productId!);
25          setProduct(response);
26          dispatch(setLoading(false));
27        } catch (err) {
28          dispatch(setLoading(false));
29          toast.error("Ha ocurrido un error al obtener el producto");
30        }
31      };
32
33      fetchData();
34    }, [productId]);
35
36    const handleAddProductCart = () => {
37      dispatch(setNewProductCart({ productId: +productId!, quantity: 1 }));
38      toast.success("Producto Agregado al carrito");
39    };
40
41    return (
42      <>
43        <div>
44          <Link
45            to="/products"
46            className="rounded-md text-sm font-semibold p-2 text-slate-500 shadow-sm hover:bg-slate-300"
47          >
48            Regresar
49          </Link>
50        </div>
51        {isLoading ? (
52          <Loading>Cargando Datos...</Loading>
53        ) : (
54          <>
55            <div className="mx-auto mt-6">
56              <div className="overflow-hidden rounded-lg">
57                <img alt={product?.title} src={product?.image} width={300} />
58              </div>
59            </div>

```

Capturamos el id del producto desde la ruta y lo usamos para hacer un request y recuperar el detalle del producto, no ocupamos guardarlo de forma global en el store, solo lo guardaremos en una variable de estado, de forma local

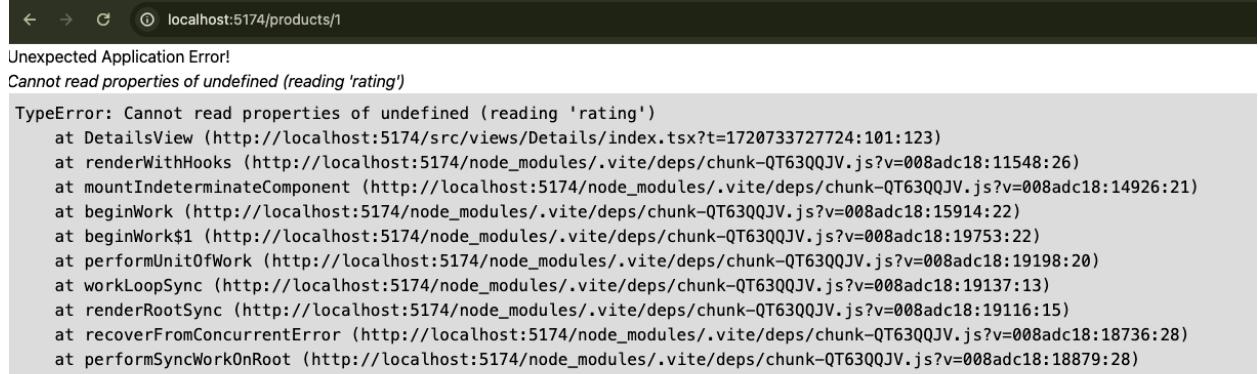
Para darle una mejor experiencia de usuario vamos a crear un componente rating que muestre la calificación de los usuarios.

revisando la respuesta del api tenemos los datos de rate y count

```
src > components > RateProduct > index.tsx > ...
1  import { StarIcon } from "@heroicons/react/24/outline";
2  import { RatingProps } from "./types";
3
4
5
6  const StarRating = ({ rating }: RatingProps) => {
7    const { rate, count } = rating;
8    const fullStars = Math.round(rate);
9    const emptyStars = 5 - fullStars;
10
11   return (
12     <div className="flex items-center space-x-1">
13       {[...Array(fullStars)].map(_, index) => (
14         <StarIcon key={`full${index}`} className="w-5 h-5 text-yellow-500" />
15       ))}
16       {[...Array(emptyStars)].map(_, index) => (
17         <StarIcon key={`empty${index}`} className="w-5 h-5 text-gray-300" />
18       )}
19       <span className="ml-2 text-gray-600">{count}</span>
20     </div>
21   );
22 };
23
24 export default StarRating;
25
```

Este componente de rating lo usamos tanto en el detalle de producto como en el listado de productos

al entrar a detalles tenemos el problema que rating property aún no existe



A screenshot of a browser window showing a TypeScript error in the developer tools' console tab. The URL is `localhost:5174/products/1`. The error message is: `Unexpected Application Error!` followed by `Cannot read properties of undefined (reading 'rating')`. Below the error, a stack trace is shown:

```
TypeError: Cannot read properties of undefined (reading 'rating')
  at DetailsView (http://localhost:5174/src/views/Details/index.tsx?t=1720733727724:101:123)
  at renderWithHooks (http://localhost:5174/node_modules/.vite/deps/chunk-QT63QQJV.js?v=008adc18:11548:26)
  at mountIndeterminateComponent (http://localhost:5174/node_modules/.vite/deps/chunk-QT63QQJV.js?v=008adc18:14926:21)
  at beginWork (http://localhost:5174/node_modules/.vite/deps/chunk-QT63QQJV.js?v=008adc18:15914:22)
  at beginWork$1 (http://localhost:5174/node_modules/.vite/deps/chunk-QT63QQJV.js?v=008adc18:19753:22)
  at performUnitOfWork (http://localhost:5174/node_modules/.vite/deps/chunk-QT63QQJV.js?v=008adc18:19198:20)
  at workLoopSync (http://localhost:5174/node_modules/.vite/deps/chunk-QT63QQJV.js?v=008adc18:19137:13)
  at renderRootSync (http://localhost:5174/node_modules/.vite/deps/chunk-QT63QQJV.js?v=008adc18:19116:15)
  at recoverFromConcurrentError (http://localhost:5174/node_modules/.vite/deps/chunk-QT63QQJV.js?v=008adc18:18736:28)
  at performSyncWorkOnRoot (http://localhost:5174/node_modules/.vite/deps/chunk-QT63QQJV.js?v=008adc18:18879:28)
```

Hey developer!

You can provide a way better UX than this when your app throws errors by providing your own `ErrorBoundary` or `errorElement` prop on your route.

para evitar esto hacemos una condición para que solo se cargue el componente cuando exista product que es la variable de estado que contiene todos los datos

```
<div className="mt-6">
| {product && <StarRating rating={product!.rating} />}
</div>
```

Add remove productos de carrito

Agregamos la funcionalidad para agregar y quitar productos del carrito, solo de forma local en nuestro store

creamos dos nuevos reducers

```

src > store > ts cartSlice.ts > [o] initialState > ↴ data
  9  const initialState: CartState = {
10    data: {
11      user: '',
12      date: '',
13      products: [],
14    },
15    loading: false,
16  };
17}
18
19const usersSlice = createSlice({
20  name: 'cart',
21  initialState,
22  reducers: {
23    setCart(state, action: PayloadAction<UserCartProps>) {
24      state.data = action.payload;
25    },
26    setLoading(state, action: PayloadAction<boolean>) {
27      state.loading = action.payload;
28    },
29    setNewProductCart(state, action: PayloadAction<CartProduct>) {
30      const cart = {...state.data};
31      const exists = cart.products.find(product => product.productId === action.payload.productId);
32      if (exists) {
33        cart.products.forEach(element => {
34          if (element.productId === action.payload.productId) {
35            element.quantity += action.payload.quantity
36          }
37        });
38      } else {
39        cart.products = [
40          ...cart.products,
41          action.payload
42        ]
43      }
44      state.data = cart;
45    },
46    setRemoveProductCart(state, action: PayloadAction<number>) {
47      const cart = {...state.data};
48      cart.products = cart.products.filter(product => product.productId !== action.payload);
49      state.data = cart;
50    },
51  },
52})

```

agregamos un botón de eliminar con su funcionalidad en el componente CartProduct

```

src > components > CartProduct > index.tsx > CartProduct
  1 import { TrashIcon } from "@heroicons/react/24/outline";
  2 import { ProductProps } from "../Product/types";
  3 import { useDispatch } from "react-redux";
  4 import { AppDispatch } from "../../store";
  5 import { toast } from "react-toastify";
  6 import { setRemoveProductCart } from "../../store/cartSlice";
  7
  8 export const CartProduct = ({product}: ProductProps) => {
  9   const dispatch: AppDispatch = useDispatch();
 10
 11   const handleRemoveProductCart =() => {
 12     dispatch(setRemoveProductCart(product.id));
 13     toast.success('Producto eliminado del carrito');
 14   }
 15   return (
 16     <li key={product.id} className="flex py-6">
 17       <div className="h-24 w-24 flex-shrink-0 overflow-hidden rounded-md border border-gray-200">
 18         <img
 19           alt={product.title}
 20           src={product.image}
 21           className="h-full w-full object-cover object-center"
 22         />
 23       </div>
 24
 25       <div className="ml-4 flex flex-1 flex-col">
 26         <div>
 27           <div className="flex justify-between text-base font-medium text-gray-900">
 28             <h3>{product.title}</h3>
 29             <p className="ml-4">${product.price}</p>
 30           </div>
 31           <p className="mt-1 text-sm text-gray-500">{product.category}</p>
 32         </div>
 33         <div className="flex flex-1 items-end justify-between text-sm">
 34           <p className="text-gray-500">Cantidad: {product.quantity}</p>
 35
 36           <div className="flex">
 37             <button
 38               type="button"
 39               onClick={handleRemoveProductCart}
 40               className="font-medium text-indigo-600 hover:text-indigo-500"
 41             >
 42               <TrashIcon aria-hidden="true" className="h-6 w-6" />
 43             </button>
 44           </div>
 45         </div>
 46       </div>
 47     </li>
 48   );
 49 };
 50

```

dentro del listado de productos, dentro del componente Product creamos un botón para agregar con su funcionalidad

```
src > components > Product > index.tsx > [o] Product
  1 import { Link } from "react-router-dom";
  2 import StarRating from "../RateProduct";
  3 import { ProductProps } from "./types";
  4 import { useDispatch } from "react-redux";
  5 import { AppDispatch } from "../../store";
  6 import { toast } from "react-toastify";
  7 import { setNewProductCart } from "../../store/cartSlice";
  8
  9 export const Product = ({ product }: ProductProps) => {
10   const dispatch: AppDispatch = useDispatch();
11
12   const handleAddProductCart = () => {
13     dispatch(setNewProductCart({ productId: product.id, quantity: 1 }));
14     toast.success("Producto Agregado al carrito");
15   };
16   return (
17     <div className="group relative">
18       <div className="aspect-h-1 aspect-w-1 w-full overflow-hidden rounded-md bg-gray-200 lg:aspect-none group-hover:opacity-55 lg:h-80">
19         <img
20           alt={product.title}
21           src={product.image}
22           className="h-full w-full object-cover object-center lg:h-full lg:w-full"
23         />
24       </div>
25       <div className="mt-4 flex justify-between">
26         <div>
27           <Link to={`/products/${product.id}`}>
28             <h3 className="text-sm text-gray-700">{product.title}</h3>
29           </Link>
30           <p className="mt-1 text-sm text-gray-500">{product.category}</p>
31         </div>
32         <p className="text-sm font-medium text-gray-900">${product.price}</p>
33       </div>
34       <StarRating rating={product.rating} />
35       <button
36         onClick={handleAddProductCart}
37         className="w-full font-medium text-indigo-600 hover:text-indigo-500"
38       >
39         Agregar
40       </button>
41     </div>
42   );
43 }
44
```