

Manual Integración de Módulos Full Stack

Descargamos

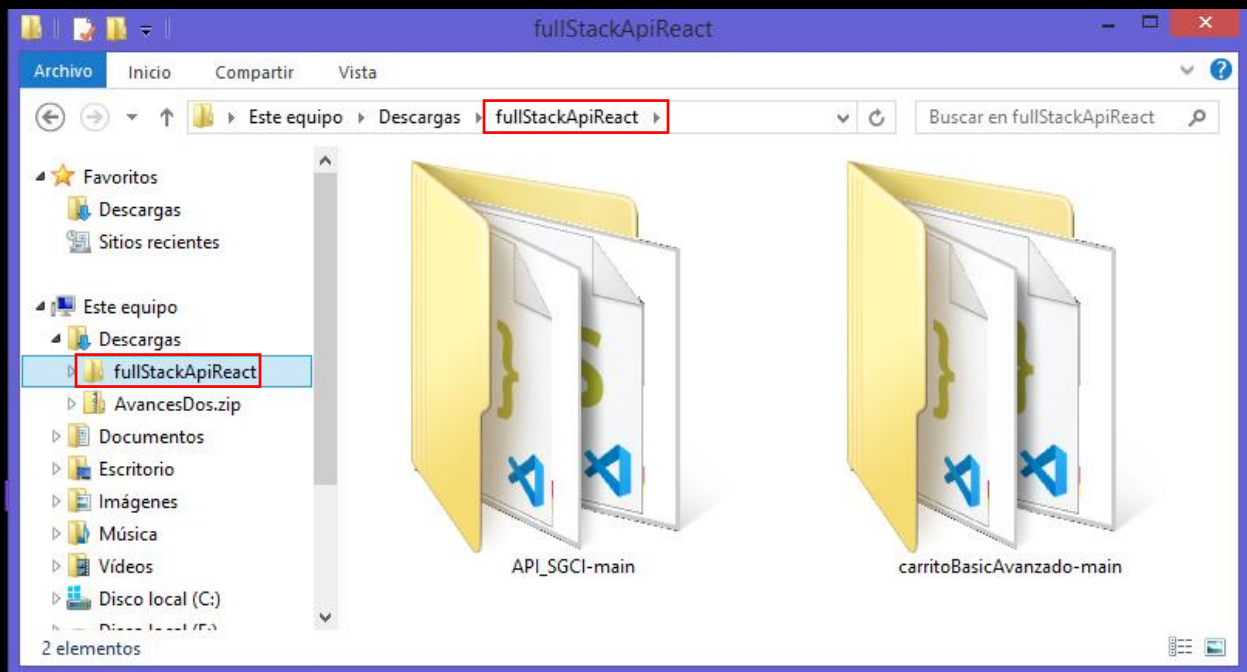
Frontend:

<https://github.com/mellotak/carritoBasicAvanzado/archive/refs/heads/main.zip>

Backend:

https://github.com/mellotak/API_SGCI/archive/refs/heads/main.zip

Extraemos su contenido y colocamos, las dos carpetas dentro de una carpeta llamada **fullStackApiReact**, quedando el resultado así:



Instalamos las dependencias necesarias, para cada carpeta, utilizando el comando **npm install** para cada carpeta, de la siguiente manera:

```
MINGW64:/c/Users/SebasPC/Downloads/fullStackApiReact/API_SGCI-main
SebasPC@Sebastian MINGW64 ~/Downloads/fullStackApiReact/API_SGCI-main (master)
$ npm install

npm warn deprecated shortid@2.2.16: Package no longer supported. Contact Support at https://npmjs.com/support for more info.

added 126 packages, and audited 127 packages in 45s

14 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

```
MINGW64:/c/Users/SebasPC/Downloads/fullStackApiReact/carritoBasicAvanzado-main
SebasPC@Sebastian MINGW64 ~/Downloads/fullStackApiReact/carritoBasicAvanzado-main (master)
$ npm install

npm warn deprecated rimraf@3.0.2: Rimraf versions prior to v4 are no longer supported

changed 60 packages, and audited 318 packages in 37s

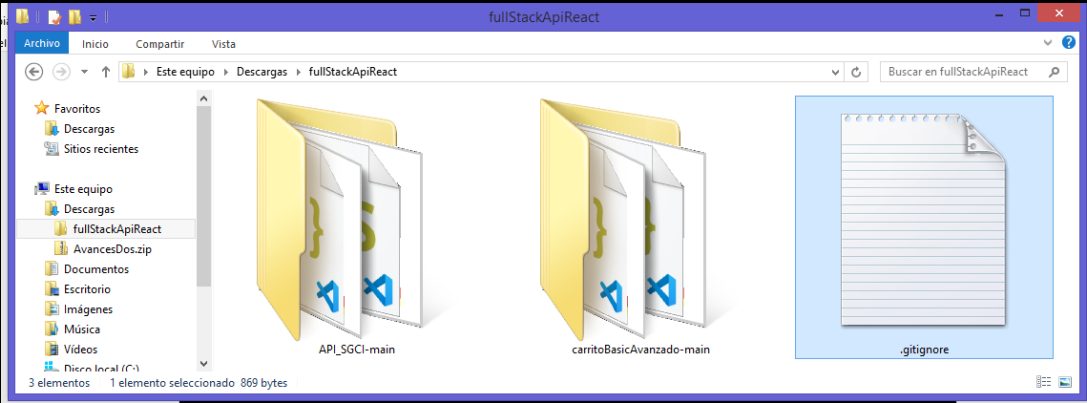
104 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

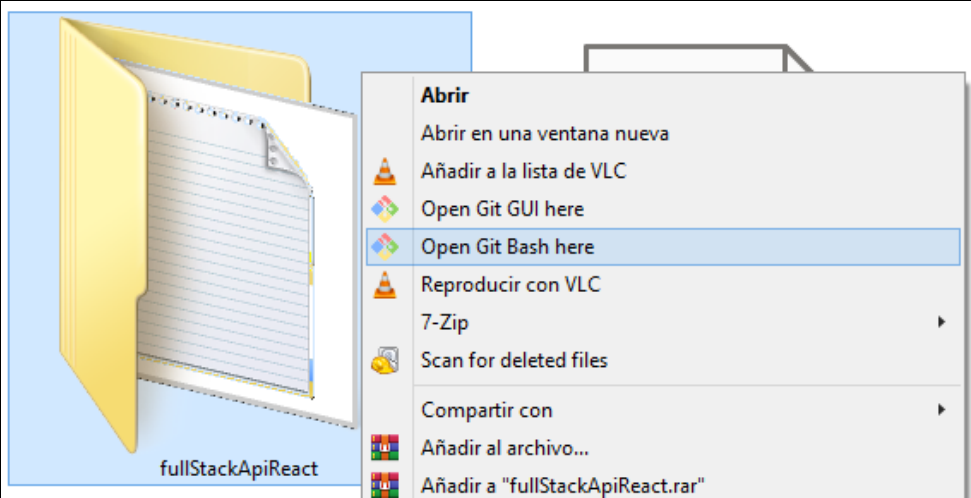
Dentro de la carpeta **fullStackApiReact** vamos a colocar un archivo llamado `.gitignore`, que pueden descargar aquí:

<https://drive.usercontent.google.com/uc?id=1WQmnC-7hbAdtT1hJYmaJ0g-vV-lGoI3v&export=download>

Quedando este resultado:



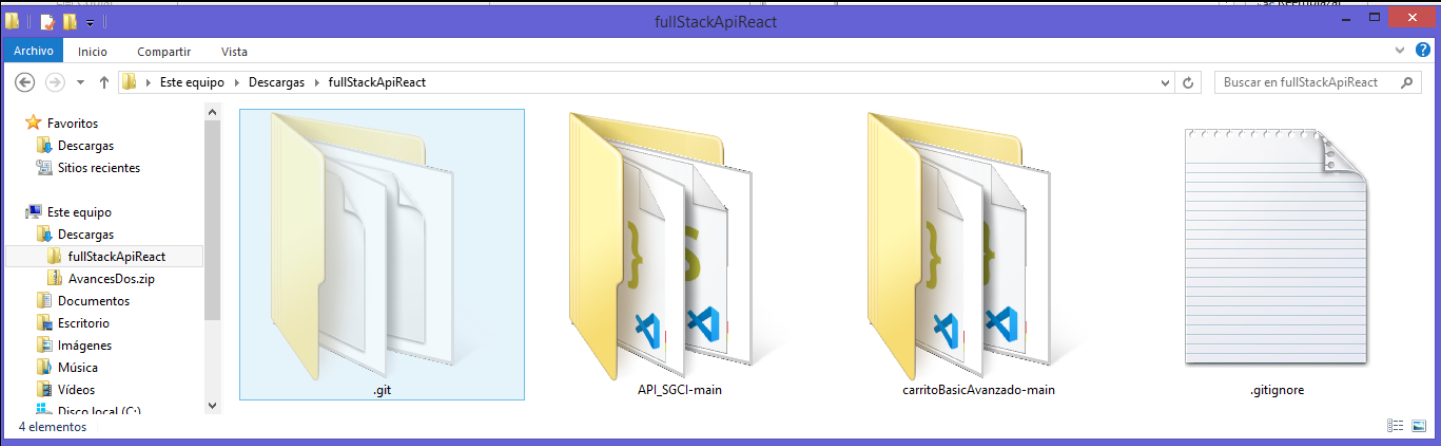
Vamos a inicializar git a nuestra carpeta **fullStackApiReact**



Con el comando: `git init`



Resultado:



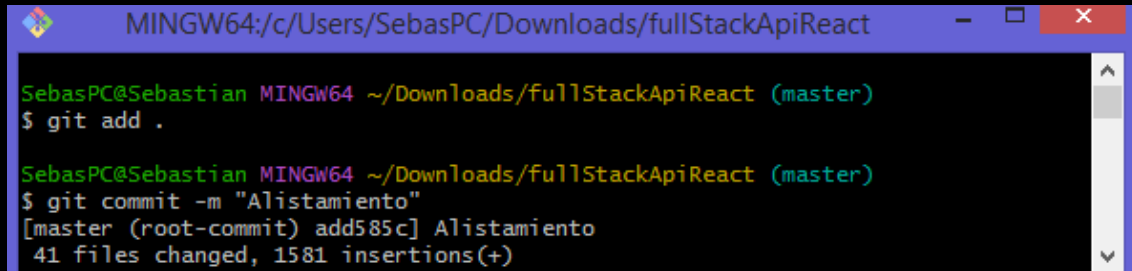
Alistamos con el comando:

```
git add .
```

Y cargamos con el comando:

```
git commit -m "Alistamiento"
```

Resultado:

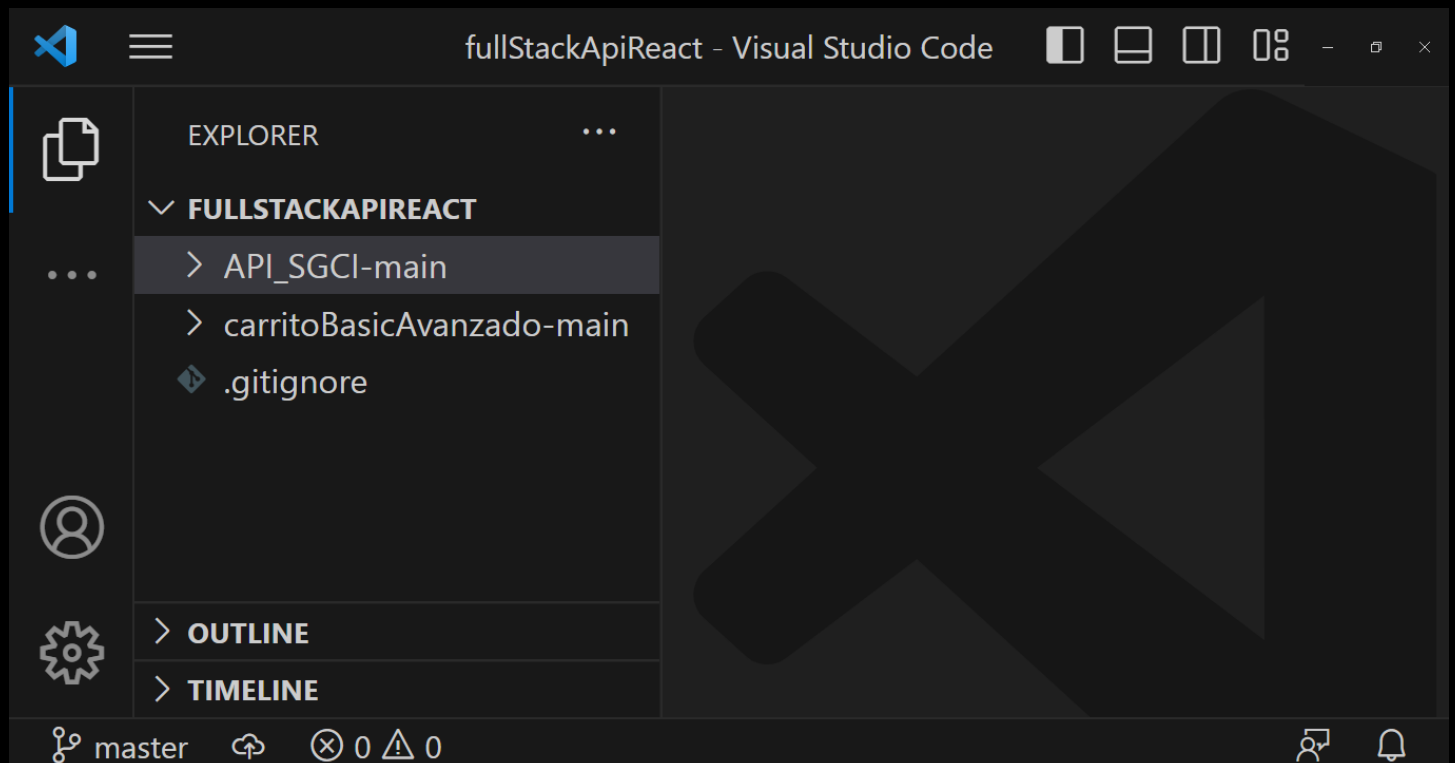
A screenshot of a Windows Command Prompt window titled "MINGW64:/c/Users/SebasPC/Downloads/fullStackApiReact". The prompt shows the user "SebasPC@Sebastian" in the directory "MINGW64 ~/Downloads/fullStackApiReact (master)". The first command entered is "\$ git add .". The second command is "\$ git commit -m 'Alistamiento'". The output of the commit command is "[master (root-commit) add585c] Alistamiento" followed by "41 files changed, 1581 insertions(+)" on the next line. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

```
SebasPC@Sebastian MINGW64 ~/Downloads/fullStackApiReact (master)
$ git add .

SebasPC@Sebastian MINGW64 ~/Downloads/fullStackApiReact (master)
$ git commit -m "Alistamiento"
[master (root-commit) add585c] Alistamiento
41 files changed, 1581 insertions(+)
```

Ahora comencemos, con mostrar productos desde la BD

Abrimos nuestra carpeta **fullStackApiReact** con visual code



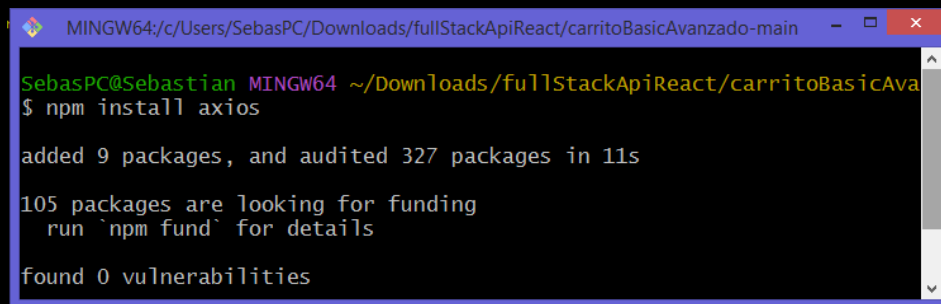
Carpeta FrontEND: carritoBasicAvanzado-main

Para esta sección vamos a utilizar **Axios**, pero: ¿Qué es Axios?

Axios es una librería de JavaScript utilizada para hacer peticiones HTTP desde el navegador y Node.js. Permite a los desarrolladores interactuar con APIs y servicios web de una manera sencilla y eficiente, manejando tanto solicitudes como respuestas de manera asíncrona. Se puede instalar con el comando:

```
npm install axios
```

Resultado

A screenshot of a Windows Command Prompt window. The title bar shows the path 'MINGW64:/c/Users/SebasPC/Downloads/fullStackApiReact/carritoBasicAvanzado-main'. The prompt is 'SebasPC@Sebastian MINGW64 ~/Downloads/fullStackApiReact/carritoBasicAva'. The command entered is '\$ npm install axios'. The output shows 'added 9 packages, and audited 327 packages in 11s', '105 packages are looking for funding', 'run `npm fund` for details', and 'found 0 vulnerabilities'.

```
SebasPC@Sebastian MINGW64 ~/Downloads/fullStackApiReact/carritoBasicAva
$ npm install axios

added 9 packages, and audited 327 packages in 11s

105 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

App.jsx

Vamos a trabajar con el archivo **App.jsx** de la carpeta Carrito, en este link podrás ver las adiciones y los cambios:

LINK: <https://github.com/mellotak/fullStackApiReactADSO/commit/efb7ba566efc00cba77bc34386a22e86e3bf2780#diff-93157c85182eb7c769f77d53d10c8c8ca73118946ace3337b1caffa14c4a575e>

De acuerdo al anterior código, hemos trabajado con los hooks de REACT: **useEffect** y **useLocation**:

useLocation se utiliza dentro del componente para obtener la ubicación actual, y luego **useEffect** utiliza esta información para decidir si debe ejecutar **fetchProducts** (es decir si esta /, carga los productos).

USELOCATION

¿Qué es?

useLocation es un hook proporcionado por **react-router-dom** que te permite acceder al objeto de ubicación (**location**) actual. Este objeto contiene información sobre la URL actual, como la ruta, el estado y la búsqueda (**query**).

USEEFFECT

¿Qué es?

useEffect es un hook de React que te permite ejecutar código en momentos específicos del ciclo de vida de un componente. Puedes pensar en él como una forma de decirle a React "haz esto cuando el componente se monte, se actualice o se desmonte".

También se ha utilizado el código de mapeo para transformar los datos de la API

```
// Función de mapeo para transformar los datos de la API
const mapProductData = (product) => {
  return {
    id: product._id,
    name: product.nombre,
    price: product.precio,
    imagen: product.imagen,
    // Agrega más mapeos según sea necesario
  };
};
```

Propósito

El propósito de esta función de mapeo es estandarizar y simplificar el uso de los datos obtenidos de una API. A menudo, los datos de las API tienen propiedades con nombres que pueden no ser intuitivos o no coincidir con la convención de nombres que utilizas en tu aplicación. Al transformar estos datos, puedes:

- **Mejorar la legibilidad:** Usar nombres de propiedades que sean más claros y concisos.
- **Facilitar el mantenimiento:** Centralizar el mapeo de datos en una función hace que sea más fácil realizar cambios futuros.
- **Asegurar la consistencia:** Garantizar que los datos tienen un formato consistente en toda tu aplicación.

También tenemos la función traerProductos, **fetchProducts**:

```
useEffect(() => {
  const fetchProducts = async () => {
    try {
      const response = await axios.get('http://localhost:5000/api/productos');
      console.log('Productos desde la API:', response.data);
      // Mapeamos los productos antes de establecerlos en el estado
      const mappedProducts = response.data.map(mapProductData);
      setProducts(mappedProducts);
    } catch (error) {
      console.error('Error al obtener los productos', error);
    }
  };

  if (location.pathname === '/') {
    fetchProducts();
  }
}, [location.pathname]);
```

La función **fetchproducts** que está dentro del hook `useEffect` sirve para:

Objetivo General

Obtener y gestionar datos de productos desde una API y actualizar el estado de `products` en el componente `App` cuando la ruta (`pathname`) de la ubicación actual es `'/'`.

Funcionamiento Detallado

1. Definición de la Función Asíncrona `fetchProducts`:

- **Solicitud a la API:** Hace una solicitud GET a la API en `'http://localhost:5000/api/productos'` utilizando `axios`.
- **Manejo de la Respuesta:** Si la solicitud es exitosa, los datos de los productos obtenidos de la API se mapean utilizando `mapProductData`.
- **Actualización del Estado:** Los productos mapeados se establecen en el estado `products` mediante `setProducts(mappedProducts)`.
- **Manejo de Errores:** Si hay un error al obtener los productos, se captura y se registra en la consola.

2. Condición para Ejecutar `fetchProducts`:

- La función `fetchProducts` solo se ejecuta si `location.pathname` es `'/'`. Esto asegura que los productos solo se obtengan cuando el usuario está en la ruta principal.

3. Dependencia del Hook `useEffect`:

- `useEffect` tiene una dependencia en `[location.pathname]`, lo que significa que se ejecutará cada vez que `location.pathname` cambie. Esto asegura que la lógica para obtener productos se ejecute cuando la ruta cambie a `'/'`.

Beneficios

- **Carga Dinámica de Datos:** Los productos se obtienen dinámicamente cuando se necesita, es decir, cuando el usuario está en la ruta principal.
- **Actualización del Estado:** Mantiene el estado `products` actualizado con los datos más recientes de la API.
- **Manejo de Errores:** Proporciona una manera de manejar y registrar errores en la solicitud a la API.

En resumen, esta función dentro del hook `useEffect` es crucial para gestionar la obtención y actualización de los datos de productos de la API de manera eficiente y reactiva, asegurando que la interfaz de usuario tenga los datos más actualizados cuando se necesita.

producto.jsx

Ahora vamos actualizar el componente o archivo llamado **producto.jsx**

Pueden ver la actualización en este

LINK: <https://github.com/mellotak/fullStackApiReactADSO/commit/efb7ba566efc00cba77bc34386a22e86e3bf2780#diff-f4bb30594d6ff9b0242a06153cba1d4ba96a65ac130b8383d0bf5eaea602c8af>

Lo que se actualizo fue colocar la ruta de las imágenes, en una constante y llamarla en la impresión del html.

Nota: En este código, se define una constante imageURL que almacena la ruta de la imagen del producto, generada dinámicamente con una plantilla de cadena de texto. Esta constante se usa en el atributo src del elemento img, lo que facilita la gestión y actualización de las rutas de las imágenes y mejora la claridad y mantenibilidad del código.

Ahora bien, si ejecutamos o arrancamos la API, el Frontend y la BD:

Con los comandos: **npm run dev**, y **mongod**

```
SebasPC@Sebastian MINGW64 ~/Downloads/fullStackApiReact/API_SGCI-main (m
$ npm run dev

> api_sgci@1.0.0 dev
> nodemon server.js

[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
Servidor corriendo en el puerto 5000
MongoDB conectado
```

```
SebasPC@Sebastian MINGW64 ~/Downloads/fullStackApiReact/carritoBasicAvanzado-main
$ npm run dev

> shopan@0.0.0 dev
> vite

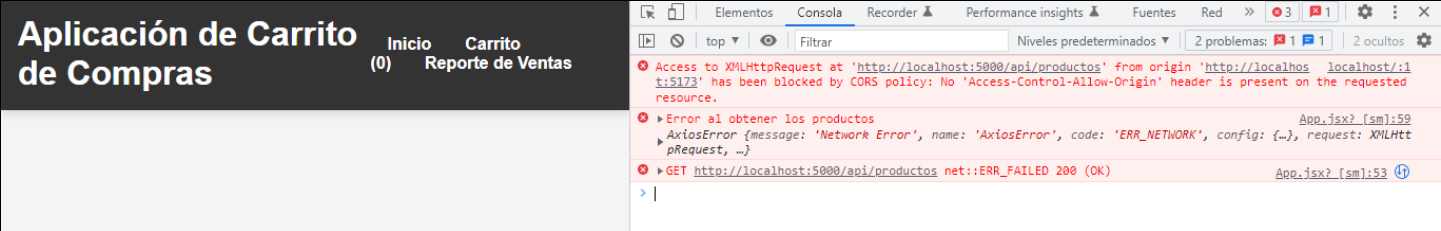
VITE v5.3.1 ready in 736 ms

→ Local:   http://localhost:5173/
→ Network: use --host to expose
```

```
Símbolo del sistema - mongod
Microsoft Windows [Versión 6.3.9600]
(c) 2013 Microsoft Corporation. Todos los derechos reservados.

C:\Users\SebasPC>mongod
2024-07-19T10:37:36.883-0500 I CONTROL [main] Automatically disabling TLS 1.0,
to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'
2024-07-19T10:37:38.145-0500 W ASIO [main] No TransportLayer configured dur
ing NetworkInterface startup
2024-07-19T10:37:38.147-0500 I CONTROL [initandlisten] MongoDB starting : pid=
3640 port=27017 dbpath=C:\data\db\ 64-bit host=Sebastian
2024-07-19T10:37:38.148-0500 I CONTROL [initandlisten] targetMinOS: Windows 7/
Windows Server 2008 R2
2024-07-19T10:37:38.148-0500 I CONTROL [initandlisten] db version v4.2.25
2024-07-19T10:37:38.148-0500 I CONTROL [initandlisten] git version: 41b59c2bfb
5121e66f18cc3ef40055a1b5fb6c2e
2024-07-19T10:37:38.148-0500 I CONTROL [initandlisten] allocator: tcmalloc
2024-07-19T10:37:38.149-0500 I CONTROL [initandlisten] modules: none
2024-07-19T10:37:38.149-0500 I CONTROL [initandlisten] build environment:
2024-07-19T10:37:38.149-0500 I CONTROL [initandlisten] distmod: 2012plus
2024-07-19T10:37:38.149-0500 I CONTROL [initandlisten] distarch: x86_64
2024-07-19T10:37:38.149-0500 I CONTROL [initandlisten] target_arch: x86_64
2024-07-19T10:37:38.150-0500 I CONTROL [initandlisten] options: {}
2024-07-19T10:37:38.153-0500 I STORAGE [initandlisten] Detected data files in
C:\data\db\ created by the 'wiredtiger' storage engine, so setting the active st
```

Tenemos: **El siguiente resultado:**



El error que estás viendo indica que tu aplicación React en `http://localhost:5173` no puede acceder a la API en `http://localhost:5000` debido a una política de CORS que bloquea la solicitud porque el servidor no está enviando el encabezado 'Access-Control-Allow-Origin'. Para solucionar esto, debes configurar el servidor backend para permitir solicitudes desde `http://localhost:5173`

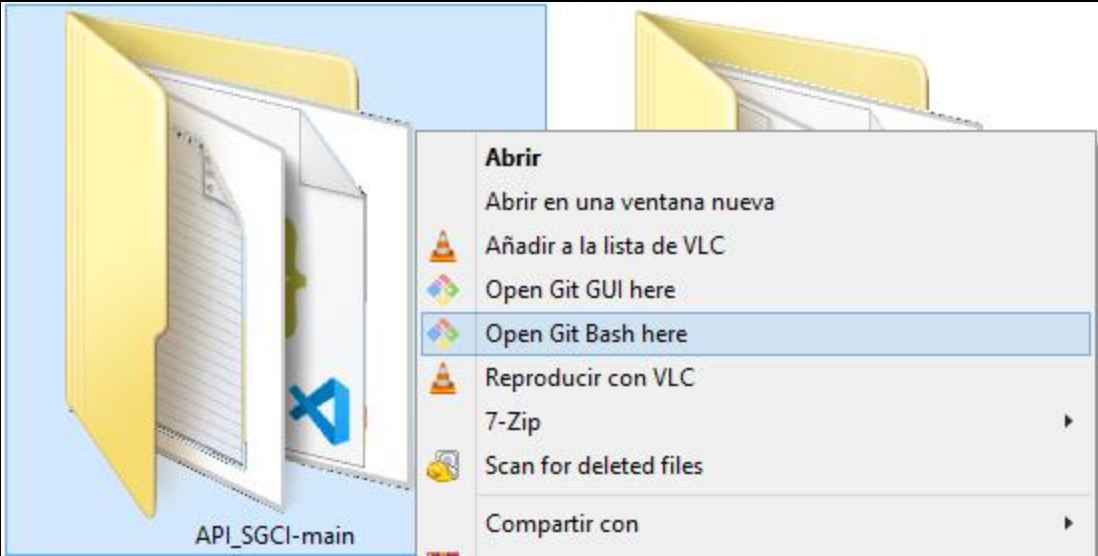
Por consiguiente, debemos hacer una actualización a nuestro backend específicamente al [archivo: server.js](#)

CARPETA BACKEND: API_SGCI-main

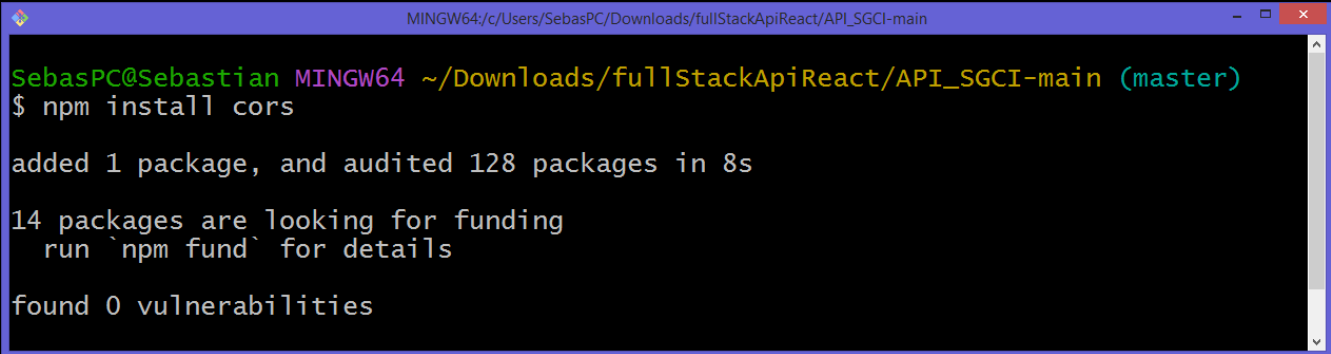
En esta sección trabajaremos con la dependencia `cors` en nuestra backend debemos instalarlo con el comando:

```
npm install cors
```

Recuerda instalarlo en la carpeta correcta:



Resultado



SERVER.JS

Ahora actualizamos nuestro archivo server.js de nuestro backend.

Pueden ver las actualizaciones en este

LINK: <https://github.com/mellotak/fullStackApiReactADSO/commit/efb7ba566efc00cba77bc34386a22e86e3bf2780#diff-2b491b7708de37985651cff01942af0c9d952722ccb4185dc06bedfa0555bebce>

Explicación de algunas actualizaciones:

```
const cors = require('cors'); // Importar cors
const path = require('path'); // Importar path
```

Importamos cors y patch

```
// Configuración de CORS
const corsOptions = {
  origin: 'http://localhost:5173', // Reemplaza con el dominio de tu frontend
  methods: ['GET', 'POST', 'PUT', 'DELETE'], // Métodos permitidos
  allowedHeaders: ['Content-Type', 'Authorization'], // Cabeceras permitidas
  credentials: true // Habilita el manejo de cookies y otros credenciales
};
app.use(cors(corsOptions));
```

CORS (Cross-Origin Resource Sharing):

- Permite que tu servidor acepte solicitudes de otros dominios distintos al del servidor. Esto es útil cuando tienes un frontend y un backend en dominios diferentes. Por ejemplo, si tu frontend está en `http://localhost:5173` y tu backend en `http://localhost:5000`, necesitas configurar CORS para permitir que el frontend interactúe con el backend.
- La configuración especifica los métodos HTTP permitidos (GET, POST, PUT, DELETE), las cabeceras permitidas (Content-Type, Authorization), y habilita el manejo de credenciales como cookies.

```
// Middleware para servir archivos estáticos desde el directorio 'uploads'
app.use('/uploads', express.static(path.join(__dirname, 'uploads')));
```

Middleware para servir archivos estáticos:

- Permite que tu servidor sirva archivos estáticos (imágenes, videos, archivos CSS, etc.) desde un directorio específico en tu servidor. En este caso, los archivos en el directorio uploads estarán accesibles públicamente a través de la ruta /uploads.
- Esto es útil para servir recursos que los clientes necesitan, como imágenes de productos en una tienda en línea.

Ahora bien volvemos arrancar la API, el Frontend y la BD

Con los Comandos: **npm run dev**, y **mongod**

SebasPC@Sebastian MINGW64 ~/Downloads/fullStackApiReact/API_SGCI-main

```
$ npm run dev

> api_sgci@1.0.0 dev
> nodemon server.js

[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
Servidor corriendo en el puerto 5000
MongoDB conectado
```

SebasPC@Sebastian MINGW64 ~/Downloads/fullStackApiReact/carritoBasicAvanzado-main

```
$ npm run dev

> shopan@0.0.0 dev
> vite

VITE v5.3.1 ready in 736 ms

➔ Local:   http://localhost:5173/
➔ Network: use --host to expose
```

Símbolo del sistema - mongod

```
Microsoft Windows [Versión 6.3.9600]
(c) 2013 Microsoft Corporation. Todos los derechos reservados.

C:\Users\SebasPC>mongod
2024-07-19T10:37:36.883-0500 I CONTROL [main] Automatically disabling TLS 1.0,
to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'
2024-07-19T10:37:38.145-0500 W ASIO [main] No TransportLayer configured dur
ing NetworkInterface startup
2024-07-19T10:37:38.147-0500 I CONTROL [initandlisten] MongoDB starting : pid=
3640 port=27017 dbpath=C:\data\db\ 64-bit host=Sebastian
2024-07-19T10:37:38.148-0500 I CONTROL [initandlisten] targetMinOS: Windows 7/
Windows Server 2008 R2
2024-07-19T10:37:38.148-0500 I CONTROL [initandlisten] db version v4.2.25
2024-07-19T10:37:38.148-0500 I CONTROL [initandlisten] git version: 41b59c2bfb
5121e66f18cc3ef40055a1b5fb6c2e
2024-07-19T10:37:38.148-0500 I CONTROL [initandlisten] allocator: tcnalloc
2024-07-19T10:37:38.149-0500 I CONTROL [initandlisten] modules: none
2024-07-19T10:37:38.149-0500 I CONTROL [initandlisten] build environment:
2024-07-19T10:37:38.149-0500 I CONTROL [initandlisten] distmod: 2012plus
2024-07-19T10:37:38.149-0500 I CONTROL [initandlisten] distarch: x86_64
2024-07-19T10:37:38.149-0500 I CONTROL [initandlisten] target_arch: x86_64
2024-07-19T10:37:38.150-0500 I CONTROL [initandlisten] options: {}
2024-07-19T10:37:38.153-0500 I STORAGE [initandlisten] Detected data files in
C:\data\db\ created by the 'wiredTiger' storage engine, so setting the active st
```

Resultado, me esta extrayendo correctamente los datos de la BD y me los imprime correctamente en el navegador.

Aplicación de Carrito de Compras

Inicio Carrito (0) Reporte de Ventas



Portatil

\$1.00

-

1

+

Añadir al Carrito



Computadora Editada

\$1000.00

-

1

+

Añadir al Carrito



Teclado

\$100.00

-

1

+

Añadir al Carrito

Consola del navegador:

Elementos Consola Recorder Performance insights Fuentes Red

top Filtar Niveles predeterminados 1 problema: 1 2 ocultos

Productos desde la API:

Array(3)

0: {_id: '6699edabbc4b8caaa89c609', nombre: 'Portatil', descripcion: 'Excelente resistencia', precio: 1,

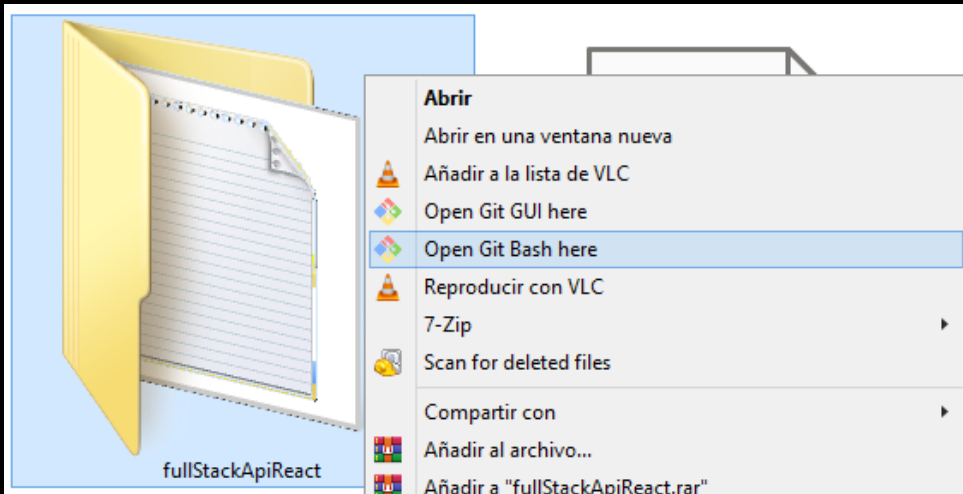
1: {_id: '6699ef2fbc4b8caaa89c60f', nombre: 'Computadora Editada', descripcion: 'Excelente resistencia',

2: {_id: '6699ef86bc4b8caaa89c61a', nombre: 'Teclado', descripcion: 'Excelente resistencia', precio: 100

length: 3

[[Prototype]]: Array(0)

Vamos a crear nuestro siguiente commit, para ello nos aseguramos de abrir correctamente la carpeta: **fullStackApiReact**



Y procedemos alistar y cargar con los comandos

```
git add .
```

```
git commit -m "Mostrar productos desde la API con REACT"
```

Resultado:

```
MINGW64:/c:/Users/SebasPC/Downloads/fullStackApiReact
SebasPC@Sebastian MINGW64 ~/Downloads/fullStackApiReact (master)
$ git add .

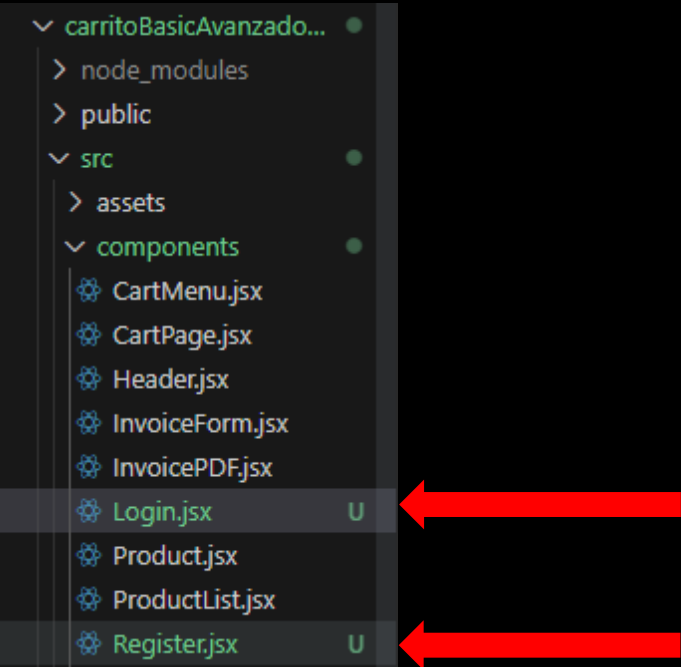
SebasPC@Sebastian MINGW64 ~/Downloads/fullStackApiReact (master)
$ git commit -m "Mostrar productos desde la API con REACT"
[master efb7ba5] Mostrar productos desde la API con REACT
6 files changed, 55 insertions(+), 9 deletions(-)
create mode 100644 API_SGCI-main/Uploads/xWNyqXvXc.jpeg

SebasPC@Sebastian MINGW64 ~/Downloads/fullStackApiReact (master)
$
```

A continuación, en las próximas secciones, nos enfocaremos en la creación de los componentes de Login y Register. Estos componentes permitirán a los usuarios registrarse e iniciar sesión en la plataforma, habilitándolos para realizar sus pedidos. Este proceso de autenticación es fundamental para garantizar que solo los usuarios registrados puedan acceder a las funcionalidades completas de la aplicación, proporcionando una experiencia personalizada y segura para cada cliente.

CARPETA FRONTEND: CARRITOBASICAVANZADO-MAIN

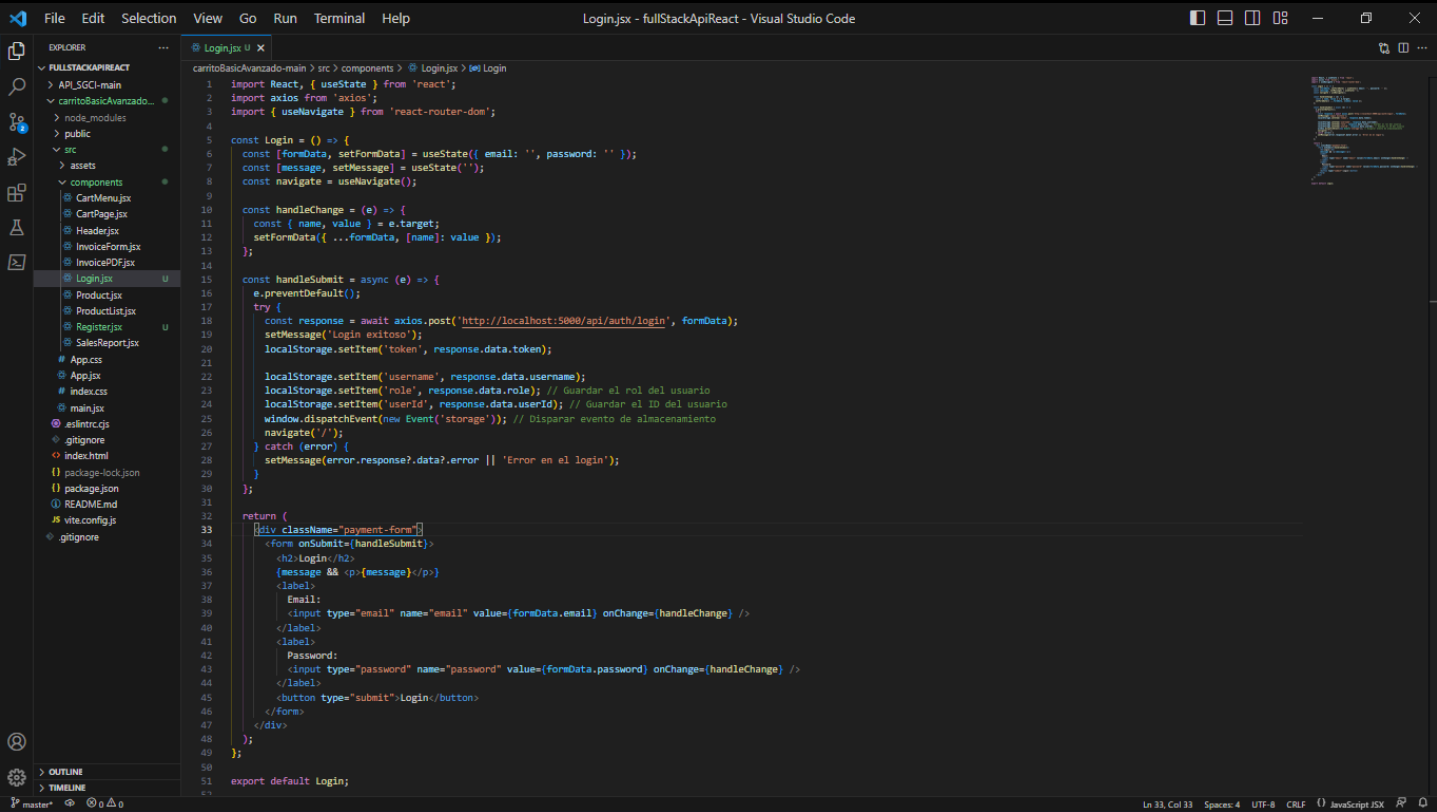
Vamos a construir los componentes: `Login.jsx` y `register.jsx` dentro de la carpeta componentes, así:



Y Colocamos el código correspondiente:

Login.jsx: LINK Código:

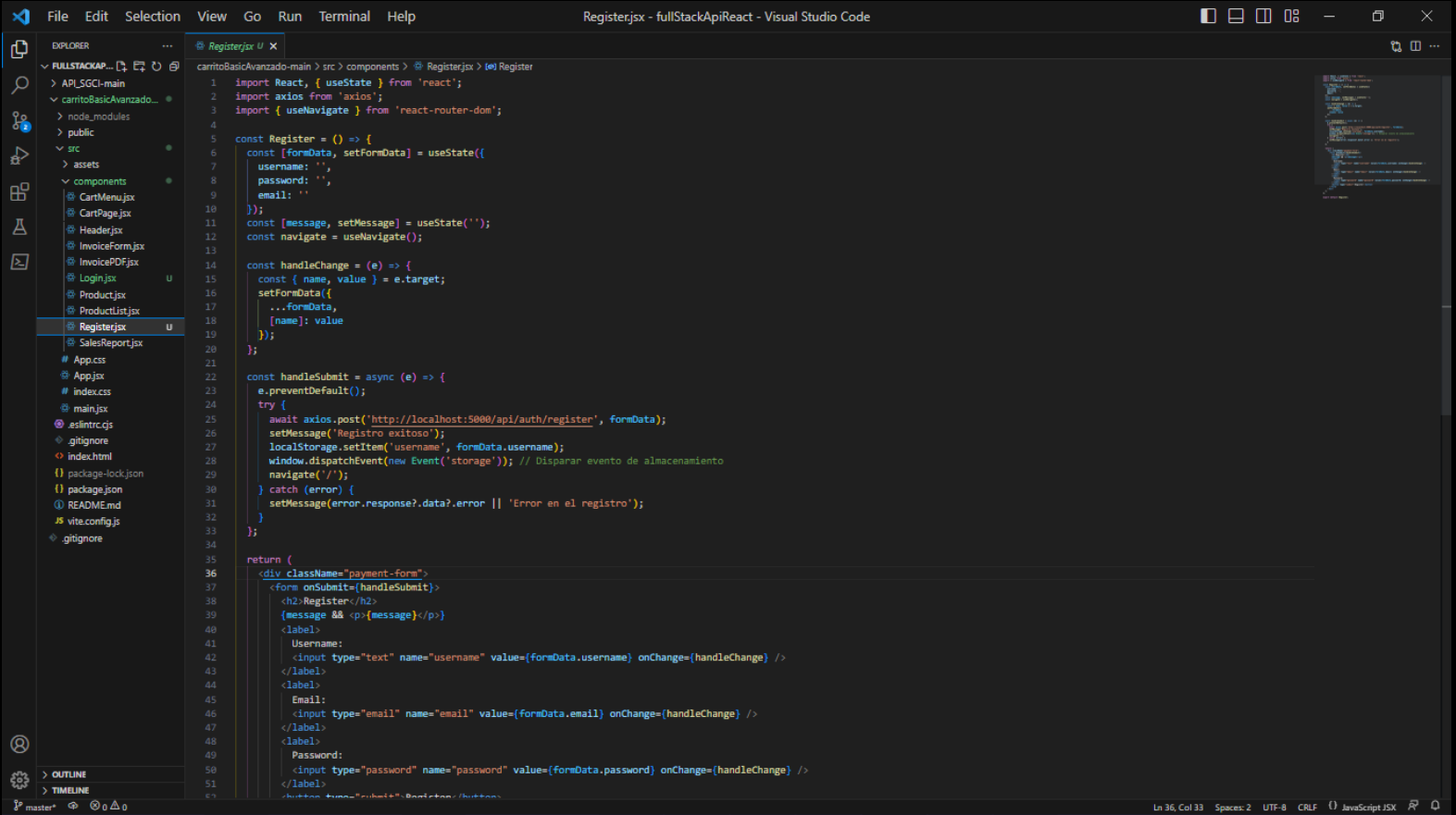
<https://github.com/mellotak/fullStackApiReactADSO/blob/main/carritoBasicAvanzado-main/src/components/Login.jsx>



El componente Login en React permite a los usuarios autenticarse en la plataforma. Utiliza los hooks `useState` y `useNavigate` de React, así como `axios` para hacer solicitudes HTTP. El componente mantiene el estado de los campos de entrada del formulario (correo electrónico y contraseña) y muestra mensajes de error si la autenticación falla. Cuando el usuario envía el formulario, se realiza una solicitud POST a la API de autenticación. Si la autenticación es exitosa, se almacenan en `localStorage` el token de autenticación, el nombre de usuario, el rol y el ID del usuario. Luego, se redirige al usuario a la página principal. En caso de error, se muestra un mensaje adecuado.

Register.jsx: LINK Código:

<https://github.com/mellotak/fullStackApiReactADSO/blob/main/carritoBasicAvanzado-main/src/components/Register.jsx>

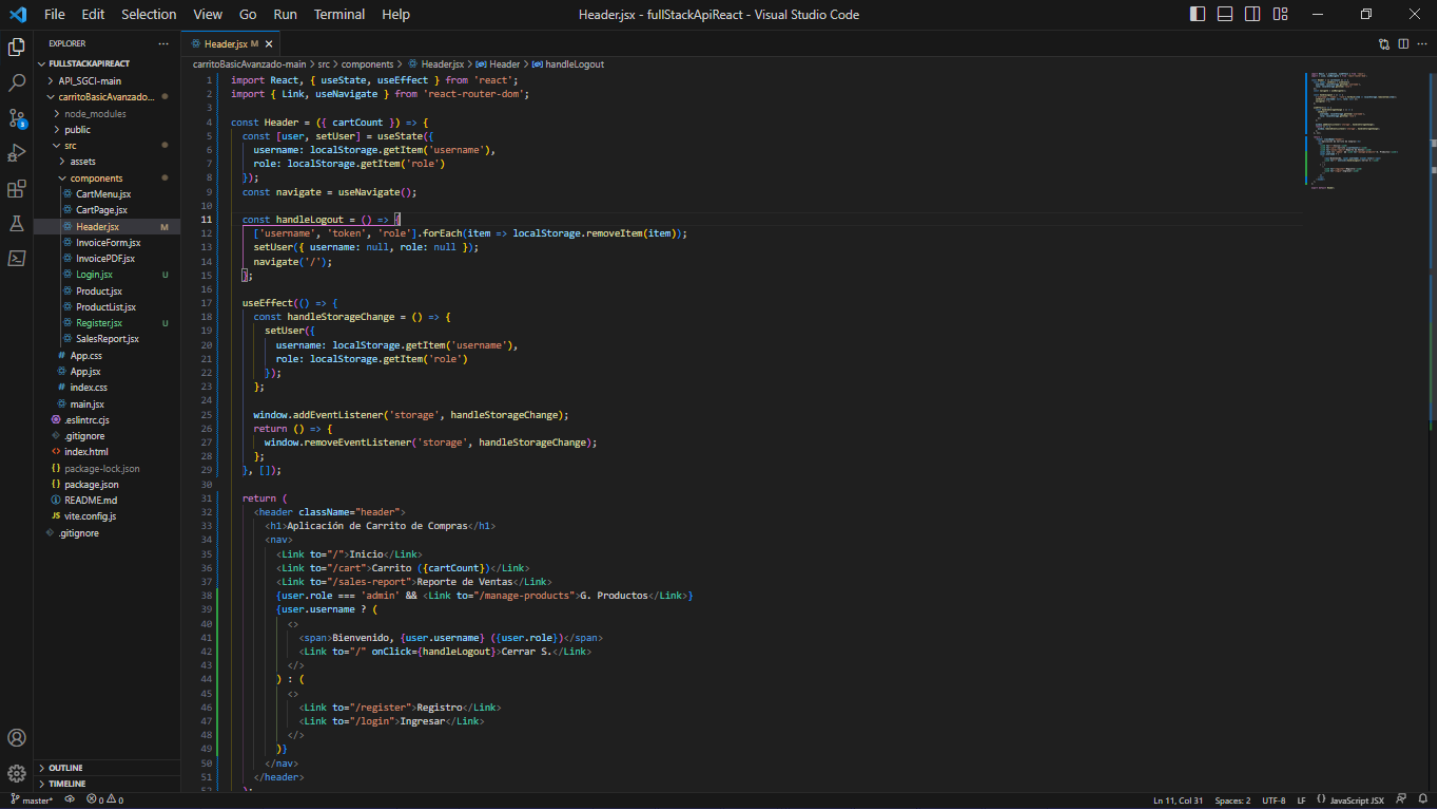


El componente Register en React permite a los usuarios registrarse en la plataforma. Utiliza los hooks useState y useNavigate de React, así como axios para hacer solicitudes HTTP. El componente mantiene el estado de los campos de entrada del formulario (nombre de usuario, correo electrónico y contraseña) y muestra mensajes de error si el registro falla. Cuando el usuario envía el formulario, se realiza una solicitud POST a la API de registro. Si el registro es exitoso, se almacenan en localStorage el nombre de usuario y el token de autenticación. Luego, se dispara un evento de almacenamiento y se redirige al usuario a la página principal. En caso de error, se muestra un mensaje adecuado.

Nota:

A continuación, vamos a reconstruir completamente el **header.jsx** de nuestro Frontend, para ello borrar el código y agregar el nuevo código que encontraran en este link:

Link: <https://github.com/mellotak/fullStackApiReactADSO/blob/main/carritoBasicAvanzado-main/src/components/Header.jsx>



```
1 import React, { useState, useEffect } from 'react';
2 import { Link, useNavigate } from 'react-router-dom';
3
4 const Header = ({ cartCount }) => {
5   const [user, setUser] = useState({
6     username: localStorage.getItem('username'),
7     role: localStorage.getItem('role')
8   });
9   const navigate = useNavigate();
10
11   const handleLogout = () => {
12     ['username', 'token', 'role'].forEach(item => localStorage.removeItem(item));
13     setUser({ username: null, role: null });
14     navigate('/');
15   };
16
17   useEffect(() => {
18     const handleStorageChange = () => {
19       setUser({
20         username: localStorage.getItem('username'),
21         role: localStorage.getItem('role')
22       });
23     };
24
25     window.addEventListener('storage', handleStorageChange);
26     return () => {
27       window.removeEventListener('storage', handleStorageChange);
28     };
29   }, []);
30
31   return (
32     <header className="header">
33       <h1>Aplicación de Carrito de Compras</h1>
34       <nav>
35         <Link to="/">Inicio</Link>
36         <Link to="/cart">Carrito ({cartCount})</Link>
37         <Link to="/sales-report">Reporte de Ventas</Link>
38         {user.role === 'admin' && <Link to="/manage-products">G. Productos</Link>}
39         {user.username ? (
40           <>
41             <span>Bienvenido, {user.username} ({user.role})</span>
42             <Link to="/" onClick={handleLogout}>Cerrar S.</Link>
43           </>
44         ) : (
45           <>
46             <Link to="/register">Registro</Link>
47             <Link to="/login">Ingresar</Link>
48           </>
49         )}
50       </nav>
51     </header>
52   );
53 }
```

Explicación del código header.jsx:

En este componente Header, estamos creando una cabecera para una aplicación de carrito de compras utilizando React. Este componente funcional emplea los hooks `useState` y `useEffect` para gestionar el estado del usuario y su rol, almacenados en el `localStorage`. El estado del usuario se inicializa al cargar el componente y se actualiza dinámicamente cuando ocurren cambios en el almacenamiento local, gracias a un evento escuchador de `storage`. La función `handleLogout` se encarga de eliminar las credenciales del usuario del `localStorage` y redirigirlo a la página principal. La cabecera renderiza enlaces de navegación básicos y condicionales: los enlaces de registro e inicio de sesión se muestran si el usuario no ha iniciado sesión, mientras que el nombre del usuario y un enlace de cierre de sesión aparecen si el usuario está autenticado. Además, si el usuario tiene el rol de administrador, se muestra un enlace adicional para la gestión de productos. La estructura de navegación y los saludos personalizados aseguran una experiencia de usuario coherente y segura dentro de la aplicación.

A continuación, tenemos que **actualizar app.jsx**, para que centralice e integre lo anterior.

Actualización de **APP.JSX**

En el siguiente link pueden ver la actualización de **app.jsx**

LINK: <https://github.com/mellotak/fullStackApiReactADSO/commit/ef7ec042547a395170c4566a4c82e1d1c55339e0#diff-93157c85182eb7c769f77d53d10c8c8ca73118946ace3337b1caffa14c4a575e>

Algunas actualizaciones:

```
import Register from './components/Register';
import Login from './components/Login';
```

Explicación: Estas líneas importan los componentes Register y Login desde sus respectivos archivos. Esto permite que estos componentes se utilicen dentro del archivo app.jsx, facilitando la navegación y la autenticación en la aplicación.

```
const PrivateRoute = ({ children }) => {
  const role = localStorage.getItem('role');
  return role === 'admin' ? children : <div>No tienes acceso a esta página</div>;
};
```

Explicación: PrivateRoute es un componente de ruta protegida que verifica si el usuario tiene el rol de administrador antes de permitir el acceso a ciertos contenidos. Si el usuario no es administrador, muestra un mensaje indicando que no tiene acceso a la página.

```
<Route path="/register" element={<Register />} />
<Route path="/login" element={<Login />} />
```

Explicación: Estas rutas definen las páginas de registro e inicio de sesión dentro de la aplicación. La ruta /register renderiza el componente Register, mientras que la ruta /login renderiza el componente Login, permitiendo a los usuarios registrarse o iniciar sesión en la plataforma.

Nota:

A continuación, vamos hacer una pequeña actualización en nuestro Backend específicamente en el archivo: `authController.js`, que se encuentra en la carpeta `controllers` de nuestra API.

CARPETA backend: API_SGCI-main

SubCarpeta **Controller**, archivo: `authController.js`

Link: <https://github.com/mellotak/fullStackApiReactADSO/commit/ef7ec042547a395170c4566a4c82e1d1c55339e0#diff-3e35483d3b1052556c362f1fa7a2e6148ccec987f3e4968751cb0639699b6b51>

Actualizamos la línea 57, que es esta:

```
res.status(200).json({ mensaje: 'Inicio de sesión exitoso', token });
```

La vamos actualizar por esta más completa:

```
res.status(200).json({ mensaje: 'Inicio de sesión exitoso', token, username: user.username, role: user.role, userId: user._id });
```

Explicación: Con esta actualización, la respuesta del servidor no solo confirma el éxito del inicio de sesión, sino que también proporciona al cliente los datos del usuario necesarios para personalizar la experiencia del usuario y gestionar correctamente los permisos y accesos dentro de la aplicación. Datos que serán muy útiles en nuestro Frontend como mostrar el nombre del usuario y obtener el rol.

Testemos y el resultado fue el correcto:



En la siguiente sección vamos actualizar de nuestro **Frontend** de la carpeta **components** el archivo `InvoiceForm.jsx` para poder guardar los pedidos del cliente.

CARPETA FRONTEND: CARRITOBASICAVANZADO-MAIN

En la subcarpeta **components** en el archivo **InvoiceForm.jsx**
Vamos actualizar el archivo, para ello pueden ver el link:

LINK:

<https://github.com/mellotak/fullStackApiReactADSO/commit/ef7ec042547a395170c4566a4c82e1d1c55339e0#diff-752d7e5526b9852ade94cbea185e73b7b2c82492616d5c4ce7b7d60e432e94a6>

Inicialmente hemos importado axios

```
import axios from 'axios';
```

Luego Hemos actualizado la función **handleSubmit**, borrarla y colocar esta nueva:

```
const handleSubmit = async (e) => {
  e.preventDefault();
  const paymentCode = Math.floor(Math.random() * 1000000);

  const pedido = {
    cliente: localStorage.getItem('userId'),
    pedido: cartItems.map(item => ({
      producto: item.product.id,
      cantidad: item.quantity
    })),
    total: cartItems.reduce((sum, item) => sum + item.product.price * item.quantity, 0),
    paymentCode,
    nombreEnvio: formData.name,
    telefonoEnvio: formData.email, // Suponiendo que el email se usa como teléfono
    direccionEnvio: formData.address,
    barrioEnvio: formData.barrio,
    municipioEnvio: formData.municipio,
    departamentoEnvio: formData.departamento
  };

  console.log(pedido); // Añadir esta línea para verificar el objeto pedido

  try {
    await axios.post('http://localhost:5000/api/pedidos/nuevo', pedido);
    navigate('/invoice-pdf', { state: { ...formData, cartItems, paymentCode } });
  } catch (error) {
    console.error('Error al enviar el pedido', error);
  }
};
```

Explicación: En este fragmento de código, la función **handleSubmit** es una función asíncrona que se ejecuta cuando se envía un formulario. Primero, previene el comportamiento por defecto del formulario. Luego, genera un código de pago aleatorio. A continuación, crea un objeto **pedido** que incluye el ID del cliente almacenado en **localStorage**, una lista de productos con sus cantidades extraídas de **cartItems**, y el total calculado sumando el precio de cada producto multiplicado por su cantidad. También incluye detalles de envío extraídos de **formData**. La función imprime el objeto **pedido** en la consola para verificar su contenido. Luego, intenta enviar una solicitud POST a la API para crear un nuevo pedido y redirige a la página de factura PDF si la solicitud es exitosa. Si ocurre un error durante la solicitud, se captura y se imprime en la consola.

Ahora bien volvemos arrancar la API, el Frontend y la BD

Con los Comandos: **npm run dev**, y **mongod**

```
MINGW64/c/Users/SebasPC/Downloads/fullStackApiReact/API_SGCI-main
SebasPC@Sebastian MINGW64 ~/Downloads/fullStackApiReact/API_SGCI-main (m
$ npm run dev

> api_sgci@1.0.0 dev
> nodemon server.js

[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
Servidor corriendo en el puerto 5000
MongoDB conectado
```

```
MINGW64/c/Users/SebasPC/Downloads/fullStackApiReact/carritoBasicAvanzado-main
SebasPC@Sebastian MINGW64 ~/Downloads/fullStackApiReact/carritoBasicAvanzado-main
$ npm run dev

> shopan@0.0.0 dev
> vite

VITE v5.3.1 ready in 736 ms

➔ Local:   http://localhost:5173/
➔ Network: use --host to expose
```

```
Símbolo del sistema - mongod
Microsoft Windows [Versión 6.3.9600]
(c) 2013 Microsoft Corporation. Todos los derechos reservados.

C:\Users\SebasPC>mongod
2024-07-19T10:37:36.883-0500 I CONTROL [main] Automatically disabling TLS 1.0,
to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'
2024-07-19T10:37:38.145-0500 W ASIO [main] No TransportLayer configured dur
ing NetworkInterface startup
2024-07-19T10:37:38.147-0500 I CONTROL [initandlisten] MongoDB starting : pid=
3640 port=27017 dbpath=C:\data\db\ 64-bit host=Sebastian
2024-07-19T10:37:38.148-0500 I CONTROL [initandlisten] targetMinOS: Windows 7/
Windows Server 2008 R2
2024-07-19T10:37:38.148-0500 I CONTROL [initandlisten] db version v4.2.25
2024-07-19T10:37:38.148-0500 I CONTROL [initandlisten] git version: 41b59c2bfb
5121e66f18cc3ef40055a1b5fb6c2e
2024-07-19T10:37:38.148-0500 I CONTROL [initandlisten] allocator: tcmalloc
2024-07-19T10:37:38.149-0500 I CONTROL [initandlisten] modules: none
2024-07-19T10:37:38.149-0500 I CONTROL [initandlisten] build environment:
2024-07-19T10:37:38.149-0500 I CONTROL [initandlisten] distmod: 2012plus
2024-07-19T10:37:38.149-0500 I CONTROL [initandlisten] distarch: x86_64
2024-07-19T10:37:38.149-0500 I CONTROL [initandlisten] target_arch: x86_64
2024-07-19T10:37:38.150-0500 I CONTROL [initandlisten] options: {}
2024-07-19T10:37:38.153-0500 I STORAGE [initandlisten] Detected data files in
C:\data\db\ created by the 'wiredTiger' storage engine, so setting the active st
```

Resultado, me está guardando correctamente el pedido en la BD:

localhost:27017

10 DBS 15 COLLECTIONS

HOST localhost:27017

CLUSTER Standalone

EDITION MongoDB 4.2.25 Community

My Queries

Databases

Filter your data

LibretaDB

admin

authdb

- pedidos
- productos
- users

bdSGCI

> _MONGOSH

Documentsauthdb.pedidos

authdb.pedidos

DocumentsAggregationsSchemaExplain PlanIndexesValidation

FILTER { field: 'value' }

ADD DATAVIEW

Displaying documents 1 - 9 of 9

departamentoEnvio: "akkas"

createdAt: 2024-07-19T22:37:45.665+00:00

updatedAt: 2024-07-19T22:37:45.665+00:00

__v: 0

_id: ObjectId('669aef9b70c5130e6bb13137')

cliente: ObjectId('66998d23b19d03f1fac666fd')

pedido: Array

total: 2203

estado: "PENDIENTE"

paymentCode: 732778

nombreEnvio: "Edwin Cadena"

telefonoEnvio: "artecarnavaldepasto@gmail.com"

direccionEnvio: "Calle 55-2-5"

barrioEnvio: "Las Cuadras"

municipioEnvio: "San Gil"

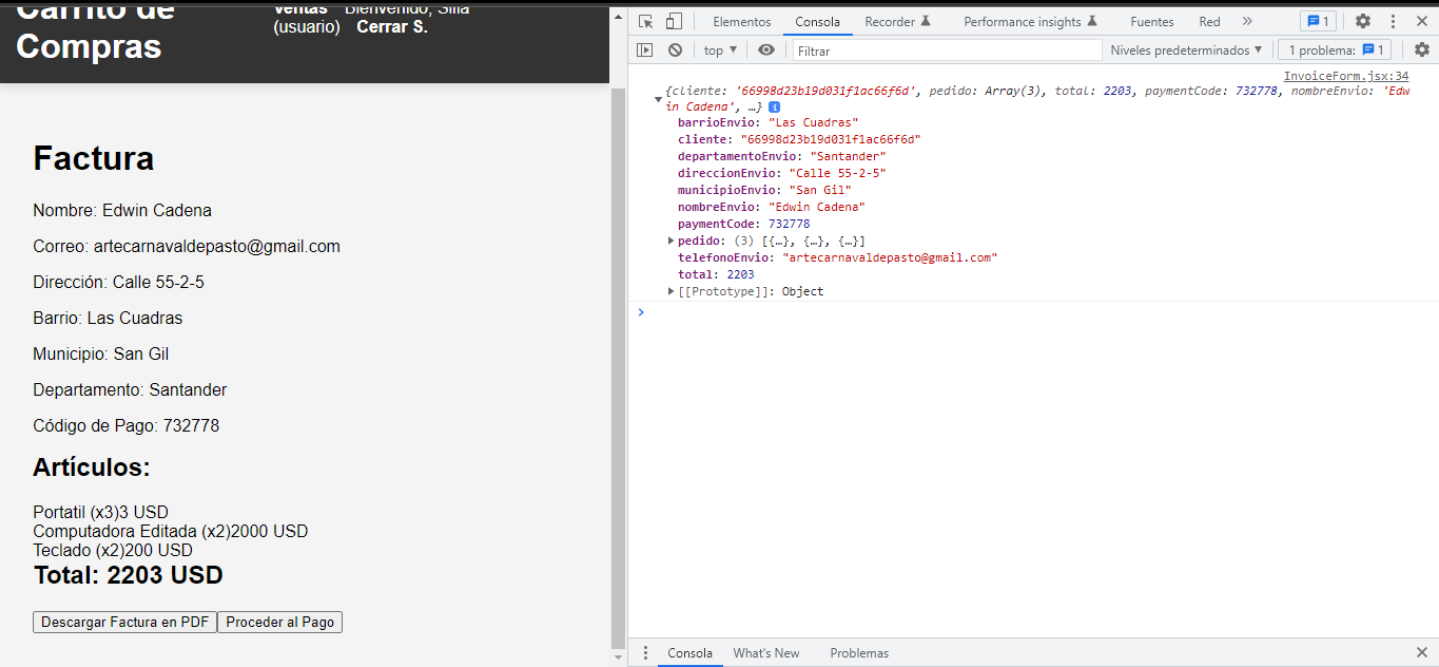
departamentoEnvio: "Santander"

createdAt: 2024-07-19T22:58:35.232+00:00

updatedAt: 2024-07-19T22:58:35.232+00:00

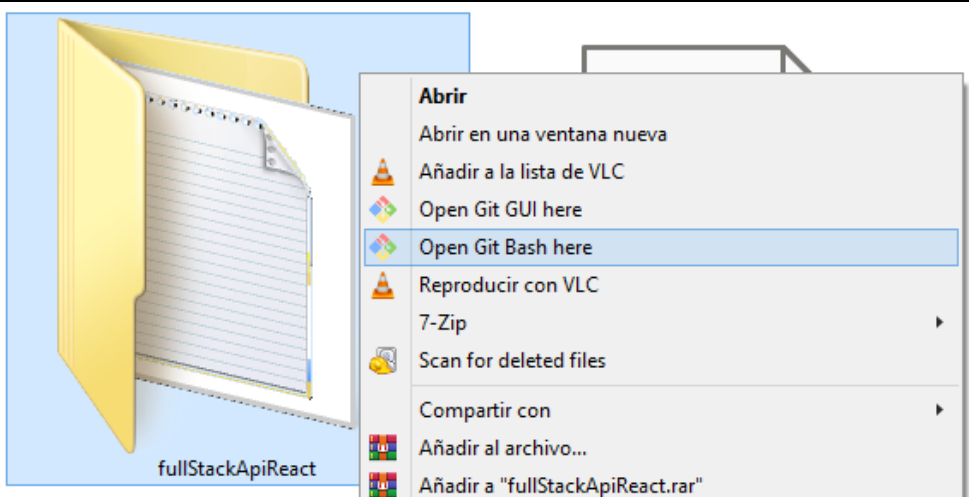
__v: 0

Consola:



Creemos nuestro nuevo commit

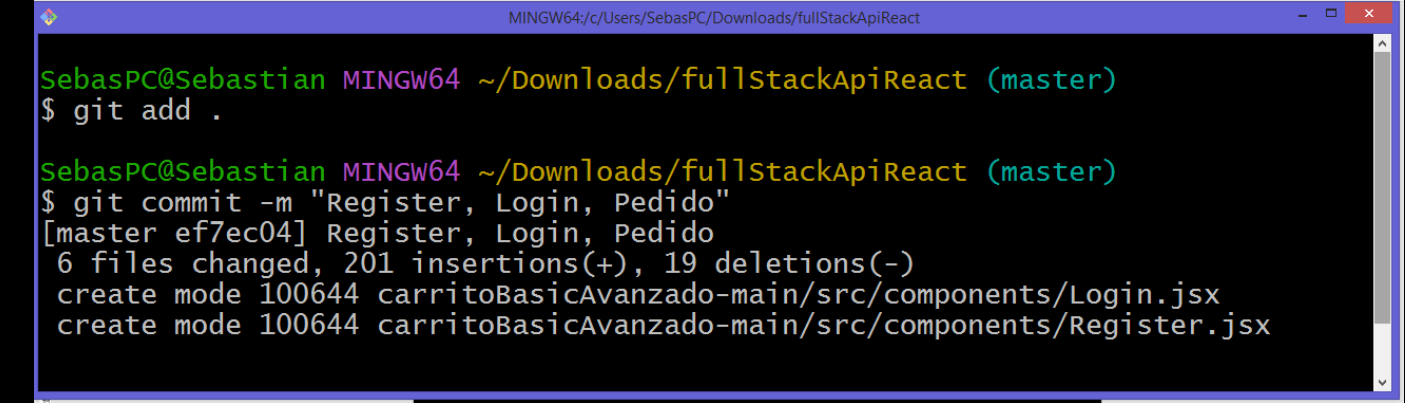
Vamos a crear nuestro siguiente commit, para ello nos aseguramos de abrir correctamente la carpeta: **fullStackApiReact**



Y procedemos alistar y cargar con los comandos

```
git add .  
git commit -m "Register, Login, Pedido"
```

Resultado:



En las siguientes secciones trabajaremos el gestor de productos.

CARPETA FRONTEND: CARRITO BASICA AVANZADO-MAIN

En la subcarpeta `components` vamos a construir los componentes `ManageProducts.jsx` y `ProductModal.jsx`

ManageProducts.jsx

El código lo pueden ver en el siguiente LINK:

LINK:

<https://github.com/mellotak/fullStackApiReactADSO/blob/main/carritoBasicaAvanzado-main/src/components/ManageProducts.jsx>

Explicación del código:

El componente `ManageProducts` de React gestiona productos utilizando hooks y Axios para solicitudes HTTP, e incluye un modal para editar productos. Mantiene estados para la lista de productos, visualización del modal, producto a editar, consulta de búsqueda y mensajes de error y éxito. Al montarse, `useEffect` llama a `fetchProducts` para cargar los productos del servidor. Las funciones `handleDeleteProduct`, `handleEditProduct`, `handleAddProduct` y `handleSearch` permiten eliminar, editar, agregar y buscar productos, respectivamente, actualizando los estados y mostrando mensajes según corresponda.

`fetchProducts` realiza una solicitud GET a la API para obtener la lista de productos y actualiza el estado `products`. `handleDeleteProduct` envía una solicitud DELETE para eliminar un producto por su ID, actualiza la lista llamando a `fetchProducts` y muestra un mensaje de éxito en `successMessage`. `handleEditProduct` establece el producto a editar en `productToEdit` y muestra el modal. `handleAddProduct` limpia `productToEdit` y muestra el modal para agregar un nuevo producto. `handleSearch` busca productos en el servidor, actualiza la lista y, si no se encuentran productos o hay un error, actualiza `errorMessage`.

El componente renderiza un título, un botón para agregar productos, un formulario de búsqueda, mensajes de retroalimentación y una tabla con los productos. La tabla incluye columnas como Nombre, Descripción, Precio, Stock, Imagen y Acciones, con botones para editar o eliminar cada producto llamando a `handleEditProduct` y `handleDeleteProduct`. El componente incluye `ProductModal` para agregar o editar productos, manejando la visibilidad del modal y la actualización de la lista de productos. Este componente ofrece una interfaz completa para la gestión de productos, permitiendo a los usuarios realizar todas las operaciones necesarias desde una única vista, con una experiencia de usuario fluida y mensajes de retroalimentación claros.

ProductModal.jsx

El código lo pueden ver en el siguiente LINK:

LINK: <https://github.com/mellotak/fullStackApiReactADSO/blob/main/carritoBasicAvanzado-main/src/components/ProductModal.jsx>

Explicación del código:

El componente ProductModal de React permite agregar y editar productos utilizando hooks y Axios para manejar el estado y realizar solicitudes HTTP. Mantiene estados para los datos del producto, mensajes de error y éxito. Al montarse, useEffect actualiza los datos del producto con la información del producto a editar, si existe. Las funciones handleInputChange y handleFileChange actualizan los campos de texto y el archivo de imagen, respectivamente. handleSubmit maneja el envío del formulario, previniendo el comportamiento por defecto, creando un objeto FormData con los datos del producto, y realizando una solicitud POST o PUT a la API según corresponda. Si la operación es exitosa, muestra un mensaje de éxito, actualiza la lista de productos, y cierra el modal después de 2 segundos; si hay un error, muestra un mensaje de error. Si showModal es falso, no se renderiza nada; si es verdadero, se muestra un modal con un formulario para ingresar o editar los datos del producto, incluyendo campos para nombre, descripción, precio, stock e imagen, y un botón de envío que cambia su texto entre "Agregar" y "Actualizar" según la acción a realizar.

APP.JSX

Tenemos que actualizar nuestro archivo app.jsx

Ver la actualización en el

LINK:

<https://github.com/mellotak/fullStackApiReactADSO/commit/07fe6a891a3344bcb3844b48bdbd21eba19fe9f5#diff-93157c85182eb7c769f77d53d10c8c8ca73118946ace3337b1caffa14c4a575e>

Para ello importamos el componente:

```
import ManageProducts from './components/ManageProducts';
```

Y creamos la ruta:

```
<Route path="/manage-products" element={<PrivateRoute><ManageProducts /></PrivateRoute>} />
```


Ahora vamos a completar los estilos, para ello en nuestro archivo:
index.css

Puedes ver en el siguiente link desde la sección:

```
/* Estilos para el modalProducto */
```

En el siguiente Link

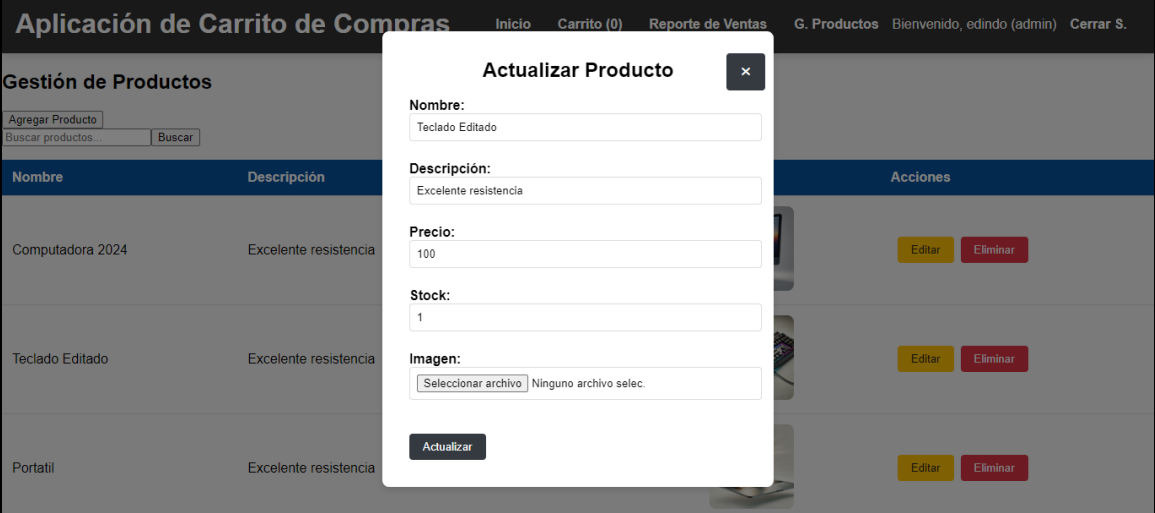
LINK: <https://github.com/mellotak/fullStackApiReactADSO/blob/main/carritoBasicAvanzado-main/src/index.css>

Explicación del código:

El estilo para el modal de producto (modalProducto) incluye una superposición fija con un fondo semi-transparente para centrar el modal y resaltar su contenido. El modal en sí tiene un fondo blanco, bordes redondeados y una sombra suave para destacar sobre el fondo. Dentro del modal, los encabezados están centrados, y los formularios tienen etiquetas y entradas estilizadas para una mejor legibilidad y usabilidad. Los botones del modal tienen un color de fondo oscuro con un efecto de hover para una mejor interacción visual, y hay un botón de cierre posicionado en la esquina superior derecha. Los mensajes de error y éxito se estilizan con colores distintivos (rojo para errores y verde para éxitos) y están centrados y en negrita para mayor visibilidad.

Para la gestión de productos, el contenedor principal tiene un fondo claro, bordes redondeados y una sombra suave. Los encabezados están centrados y los botones tienen colores diferenciados (azul para agregar, verde para buscar, rojo para eliminar y amarillo para editar) con efectos de hover. La tabla de productos se estiliza con bordes colapsados y colores alternativos para encabezados y celdas, mejorando la legibilidad. Las imágenes de los productos tienen bordes redondeados, y los botones de acciones (editar y eliminar) siguen la misma pauta de colores y efectos de hover que el resto de los botones del componente.

Resultado:



Subimos el commit:

```
MINGW64/c/Users/SebasPC/Downloads/fullStackApiReact

SebasPC@Sebastian MINGW64 ~/Downloads/fullStackApiReact (master)
$ git add .

SebasPC@Sebastian MINGW64 ~/Downloads/fullStackApiReact (master)
$ git commit -m "Gestor de productos"
[master 07fe6a8] Gestor de productos
8 files changed, 435 insertions(+)
rename API_SGCI-main/Uploads/{pYEa_w4qE.jpeg => GwkwqPdDl.jpeg} (100%)
create mode 100644 API_SGCI-main/Uploads/INimW3qBG.jpeg
rename API_SGCI-main/Uploads/{xWNyqXvXc.jpeg => QmE_xZ6Tu.jpeg} (100%)
create mode 100644 API_SGCI-main/Uploads/xbjUL7-N1.jpeg
create mode 100644 carritoBasicAvanzado-main/src/components/ManageProducts.jsx
create mode 100644 carritoBasicAvanzado-main/src/components/ProductModal.jsx

SebasPC@Sebastian MINGW64 ~/Downloads/fullStackApiReact (master)
$ |
```

Lo subimos a nuestro repositorio web

```
MINGW64/c/Users/SebasPC/Downloads/fullStackApiReact

SebasPC@Sebastian MINGW64 ~/Downloads/fullStackApiReact (master)
$ git remote add origin https://github.com/mellotak/fullStackApiReactADSO.git

SebasPC@Sebastian MINGW64 ~/Downloads/fullStackApiReact (master)
$ git branch -M main

SebasPC@Sebastian MINGW64 ~/Downloads/fullStackApiReact (main)
$ git push -u origin main
Enumerating objects: 90, done.
Counting objects: 100% (90/90), done.
Delta compression using up to 8 threads
Compressing objects: 100% (87/87), done.
Writing objects: 100% (90/90), 3.41 MiB | 1.09 MiB/s, done.
Total 90 (delta 26), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (26/26), done.
To https://github.com/mellotak/fullStackApiReactADSO.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.

SebasPC@Sebastian MINGW64 ~/Downloads/fullStackApiReact (main)
$ |
```

A todos los commit pueden acceder a este LINK:

<https://github.com/mellotak/fullStackApiReactADSO/commits/main/>