

PROYECTO EN NEXT JS

Guía paso a paso

Danna Camila Guerra Rincon

Julián David Quintero Rivera

¿Qué es NEXT JS?

Next.js es un **framework** de JavaScript basado en **React** que permite la creación de aplicaciones web de alto rendimiento con **renderizado del lado del servidor** (SSR) y **generación de sitios estáticos**. Combina las mejores características de React con funcionalidades adicionales que facilitan la creación de sitios web modernos, dinámicos y eficientes.

Características clave de Next.js:

1. **Renderizado del lado del servidor (SSR)**: Genera páginas HTML en el servidor antes de enviarlas al navegador, mejorando la velocidad de carga inicial.
2. **Rutas automáticas**: Next.js crea rutas automáticamente basadas en la estructura de los archivos dentro de la carpeta pages, lo que simplifica la navegación.
3. **API integrada**: Puedes crear una API directamente en la misma aplicación de Next.js, sin necesidad de configurar un servidor aparte.
4. **Soporte para CSS y Sass**: Te permite escribir estilos CSS directamente en los componentes o importar archivos CSS/Sass globales.

¿Para qué se utiliza Next.js?

Next.js es ideal para crear aplicaciones y sitios web que necesiten ser rápidos, escalables y optimizados para motores de búsqueda (SEO). Es utilizado en una variedad de casos como:

- **Aplicaciones web dinámicas**.
- **Blogs y sitios web de contenido** con generación estática.
- **E-commerce**: Las tiendas en línea pueden aprovechar el SSR para mostrar productos de manera más rápida.
- **Dashboards y paneles de control**: Por su capacidad de actualizar datos en tiempo real y su fácil integración con APIs.

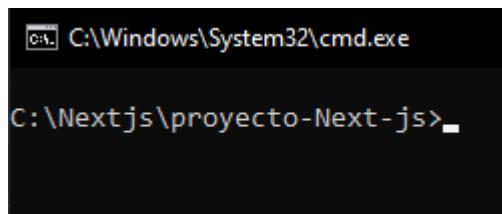
CAPITULO 1

1. Creación de un proyecto en NEXT.JS

Para crear un proyecto con Next.js, utiliza el siguiente comando en la terminal

```
{ npx create-next-app@latest }
```

Antes de ejecutar el comando, asegúrate de estar en la ubicación correcta donde deseas crear el proyecto, ya que la ubicación puede variar según las preferencias de cada uno.



```
C:\Windows\System32\cmd.exe
C:\Nextjs\proyecto-Next-js>
```

Una vez decidido el lugar, puedes proceder con la creación del proyecto.

```
C:\Nextjs\proyecto-Next-js>npx create-next-app@latest
```

Configuración Inicial

Después de ejecutar el comando, Next.js te guiará a través de una configuración básica. Las opciones que elijas dependerán de cómo quieras estructurar tu entorno de desarrollo. A continuación, algunas recomendaciones a tener en cuenta:

1. **Nombre del proyecto**: Debe escribirse en minúsculas.
2. **Uso de TypeScript (TS)**: Decide si lo vas a usar.
3. **Habilitar ESLint**: Te ayudará a detectar errores de código de manera automática.
4. **No usaremos Tailwind CSS**: Aunque es una herramienta poderosa, para este proyecto nos enfocaremos en estilos básicos.

Puedes seguir las selecciones que aparecen en la imagen de referencia para simplificar este proceso.

```
C:\Nextjs\proyecto-Next-js>npx create-next-app@latest
✓ What is your project named? ... proyecto-next-js
✓ Would you like to use TypeScript? ... No / Yes
✓ Would you like to use ESLint? ... No / Yes
✓ Would you like to use Tailwind CSS? ... No / Yes
✓ Would you like your code inside a `src/` directory? ... No / Yes
✓ Would you like to use App Router? (recommended) ... No / Yes
✓ Would you like to use Turbopack for next dev? ... No / Yes
✓ Would you like to customize the import alias (@/* by default)? ... No / Yes
Creating a new Next.js app in C:\Nextjs\proyecto-Next-js\proyecto-next-js.

Using npm.

Initializing project with template: app

Installing dependencies:
- react
- react-dom
- next

Installing devDependencies:
- typescript
- @types/node
- @types/react
- @types/react-dom
- eslint
- eslint-config-next
```

Uso de npm

Vamos a trabajar con el manejador de paquetes npm. Cuando finalices la creación del proyecto, navega hasta el directorio recién creado y ábrelo con tu editor de código preferido; en este caso, utilizaremos **Visual Studio Code (VSCode)**.

```
C:\Nextjs\proyecto-Next-js> cd proyecto-next-js
C:\Nextjs\proyecto-Next-js\proyecto-next-js>
```

2. Dependencias

Al crear el proyecto, Next.js incluye algunas dependencias por defecto.

```
● ● ●  
1 "dependencies": {  
2   "react": "19.0.0-rc-65a56d0e-20241020",  
3   "react-dom": "19.0.0-rc-65a56d0e-20241020",  
4   "next": "15.0.0"  
5 },  
6 "devDependencies": {  
7   "typescript": "^5",  
8   "@types/node": "^20",  
9   "@types/react": "^18",  
10  "@types/react-dom": "^18",  
11  "eslint": "^8",  
12  "eslint-config-next": "15.0.0"
```

Sin embargo, deberás instalar las siguientes librerías adicionales para tu entorno de desarrollo

```
● ● ●  
1 "dependencies": {  
2   "@emotion/react": "^11.13.3",  
3   "@emotion/styled": "^11.13.0",  
4   "@mui/icons-material": "^6.1.4",  
5   "@mui/material": "^6.1.4",  
6   "next": "15.0.0",  
7   "react": "^18.3.1",  
8   "react-dom": "^18.3.1",  
9   "sass": "^1.80.3",  
10  "sweetalert2": "^11.14.4"  
11 } ,
```

npm install @mui/material @emotion/react @emotion/styled

npm install @mui/icons-material

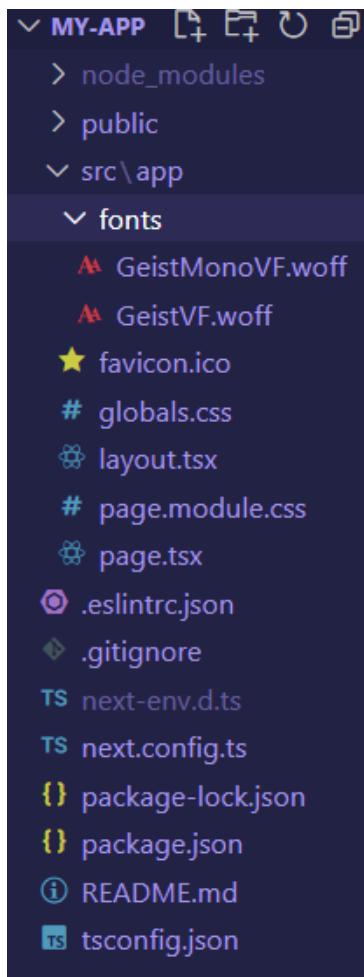
npm install react@18 react-dom@18 (Opcional)

npm install sweetalert2

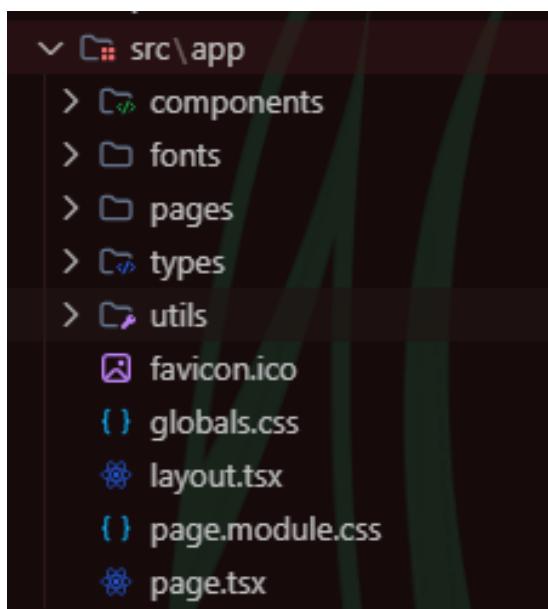
npm install sass

IMPORTANTE: El comando que vamos a utilizar para ejecutar el programa será **{npm run dev }**

3. Estructura del proyecto



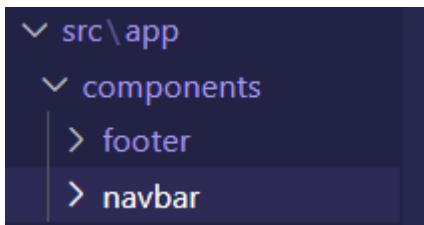
Al comenzar, verás que el proyecto tiene una estructura básica predefinida. A partir de esta, podrás agregar las carpetas y archivos que necesites para tu proyecto.



CAPITULO 2

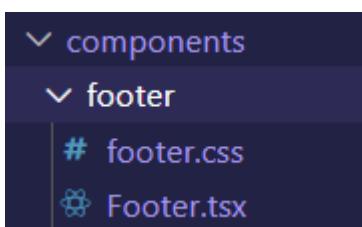
1. Creación del contenido de la carpeta componentes

1.1 Comenzamos creando dos carpetas dentro de la de componentes las cuales vamos a llamar footer y navbar



2. Creación del contenido de la carpeta footer

2.1 Ahora vamos a crear estos dos archivos en footer

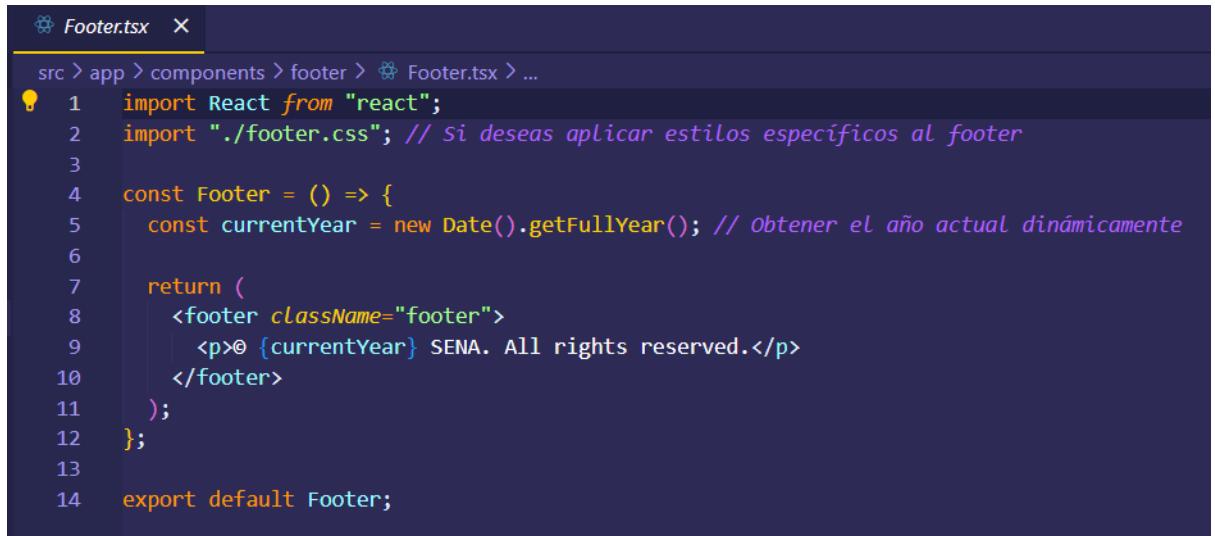


2.2 En footer.css vamos a colocar los siguientes estilos

```
# footer.css ●
src > app > components > footer > # footer.css > ...
1  .footer {
2    background-color: #whitesmoke; /* o el color que prefieras */
3    color: #black;
4    text-align: center;
5    padding: 10px 0;
6    position: fixed;
7    bottom: 0;
8    width: 100%;
9    box-shadow: 0px 2px 10px #rgba(0, 0, 0, 0.1);
10   }
11 |
```

En el archivo **footer.css**, puedes añadir los estilos que prefieras. Te animo a ser creativo con el diseño. Si prefieres utilizar el código que te proporciono, está bien, pero si quieras personalizarlo, ¡adelante!

2.3 En Footer.tsx vamos a poner la siguiente configuración



```
Footer.tsx X
src > app > components > footer > Footer.tsx > ...
1 import React from "react";
2 import "./footer.css"; // Si deseas aplicar estilos específicos al footer
3
4 const Footer = () => {
5   const currentYear = new Date().getFullYear(); // obtener el año actual dinámicamente
6
7   return (
8     <footer className="footer">
9       <p>© {currentYear} SENA. All rights reserved.</p>
10    </footer>
11  );
12};
13
14 export default Footer;
```

3. Creación del contenido de la carpeta navbar

3.1 Ahora vamos a crear estos dos archivos en navbar



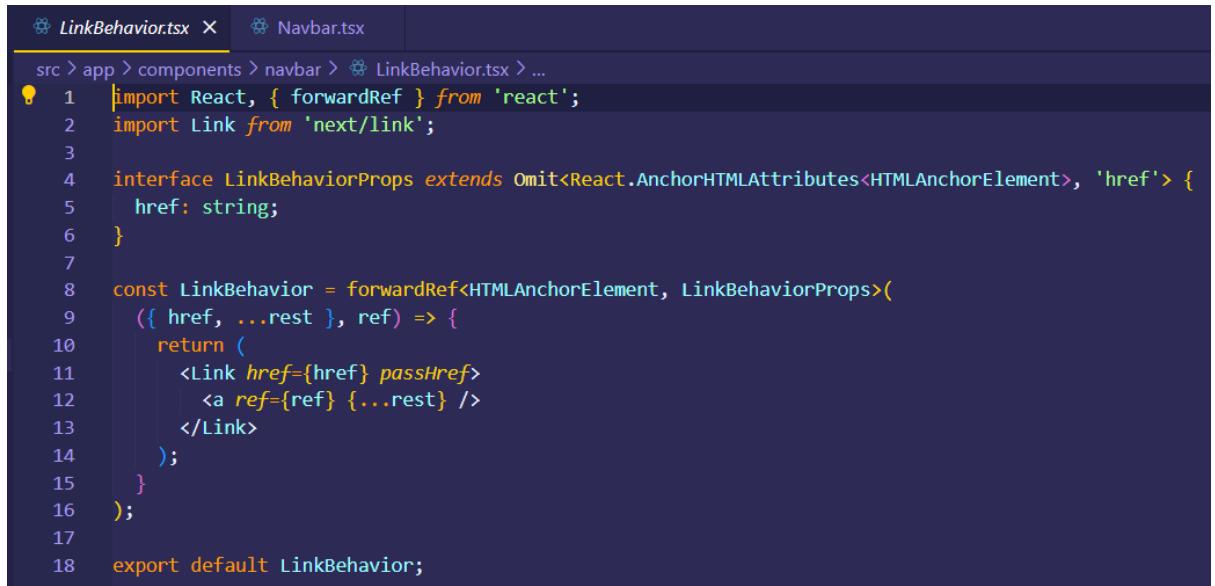
3.2 El archivo Navbar.tsx nos permitirá al usuario acceder a diferentes secciones del sistema, como "Home", "Proveedores", "Clientes", y "Productos".

```
● ● ●
1 'use client';
2 import React, { useState } from "react";
3 import {
4   AppBar,
5   Toolbar,
6   Typography,
7   IconButton,
8   Drawer,
9   List,
10  ListItem,
11  ListItemText,
12  Box,
13  Divider,
14 } from "@mui/material";
15 import MenuIcon from "@mui/icons-material/Menu";
16 import { styled } from "@mui/system";
17 import Link from "next/link";
18
19 // Estilos personalizados para los elementos del sidebar
20 const SidebarContainer = styled(Box)({
21   width: 250,
22   background: "linear-gradient(135deg, #6a11cb 0%, #2575fc 100%)", // Gradiente similar al de los otros componentes
23   height: "100vh",
24   color: "#fff",
25   padding: "20px",
26   transition: "all 0.3s ease-in-out",
27 });
28
29 const SidebarHeader = styled(Typography)({
30   fontSize: "1.5rem",
31   fontWeight: "bold",
32   textAlign: "center",
33   marginBottom: "1rem",
34   color: "#fff", // Cambiado a blanco para mejor contraste
35 });
36
37 const SidebarItem = styled(ListItem)({
38   padding: "15px 10px",
39   margin: "10px 0",
40   borderRadius: "10px", // Bordes más redondeados
41   transition: "background-color 0.3s ease, transform 0.3s ease",
42   "&:hover": {
43     backgroundColor: "rgba(255, 255, 255, 0.1)", // Fondo semi-transparente al pasar el ratón
44     transform: "translateY(-3px)", // Efecto de elevación sutil
45   },
46 });
47
48 const SidebarDivider = styled(Divider)({
49   backgroundColor: "rgba(255, 255, 255, 0.2)", // Divisor semi-transparente
50   margin: "10px 0",
51 });
52
```

```
● ● ●
1  const Navbar = () => {
2    const [isOpen, setIsOpen] = useState(false);
3
4    const toggleDrawer = (open: boolean) => (event: React.KeyboardEvent | React.MouseEvent) => {
5      if (event.type === "keydown" && ((event as React.KeyboardEvent).key === "Tab" || (event as React.KeyboardEvent).key === "Shift")) {
6        return;
7      }
8      setIsOpen(open);
9    };
10
11  const sidebarContent = (
12    <SidebarContainer role="presentation" onClick={toggleDrawer(false)} onKeyDown={toggleDrawer(false)}>
13      <SidebarHeader variant="h6">Navegación</SidebarHeader>
14      <List>
15        <SidebarItem>
16          <Link href="/pages" passHref style={{ color: "inherit", textDecoration: "none", width: "100%" }}>
17            <ListItemText primary="Home" />
18          </Link>
19        </SidebarItem>
20        <SidebarItem>
21          <Link href="/pages/proveedores" passHref style={{ color: "inherit", textDecoration: "none", width: "100%" }}>
22            <ListItemText primary="Proveedores" />
23          </Link>
24        </SidebarItem>
25        <SidebarItem>
26          <Link href="/pages/clientes" passHref style={{ color: "inherit", textDecoration: "none", width: "100%" }}>
27            <ListItemText primary="Clientes" />
28          </Link>
29        </SidebarItem>
30        <SidebarItem>
31          <Link href="/pages/productos" passHref style={{ color: "inherit", textDecoration: "none", width: "100%" }}>
32            <ListItemText primary="Productos" />
33          </Link>
34        </SidebarItem>
35      </List>
36      <SidebarDivider />
37      <Typography variant="caption" sx={{ textAlign: "center", display: "block", marginTop: "1rem", opacity: 0.7 }}>
38        Sistema de Gestión © 2024
39      </Typography>
40    </SidebarContainer>
41  );
42
43  return (
44    <>
45      <AppBar position="fixed" sx={{ background: "transparent", boxShadow: "none" }}>
46        <Toolbar>
47          <IconButton edge="start" color="inherit" aria-label="menu" onClick={toggleDrawer(true)}>
48            <MenuIcon sx={{ color: "#fff" }} />
49          </IconButton>
50        </Toolbar>
51      </AppBar>
52
53      <Drawer anchor="left" open={isOpen} onClose={toggleDrawer(false)}>
54        {sidebarContent}
55      </Drawer>
56    </>
57  );
58};
59
60 export default Navbar;
```

Aquí te mostramos cómo luce el código del **Navbar** y el **sidebar**, que incluye estilos personalizados con Material UI y el sistema de **styled components**.

3.3 En LinkBehavior.jsx vamos a poner la siguiente configuración



```
LinkBehavior.tsx X Navbar.tsx
src > app > components > navbar > LinkBehavior.tsx > ...
1 import React, { forwardRef } from 'react';
2 import Link from 'next/link';
3
4 interface LinkBehaviorProps extends Omit<React.AnchorHTMLAttributes<HTMLAnchorElement>, 'href'> {
5   href: string;
6 }
7
8 const LinkBehavior = forwardRef<HTMLAnchorElement, LinkBehaviorProps>(
9   ({ href, ...rest }, ref) => {
10     return (
11       <Link href={href} passHref>
12         <a ref={ref} {...rest} />
13       </Link>
14     );
15   }
16 );
17
18 export default LinkBehavior;
```

IMPORTANTE

Utilizamos 'use client'; en Next.js para indicar que el componente se ejecuta en el navegador. Esto es necesario cuando usamos funcionalidades del cliente, como el manejo de estados o eventos de interfaz, que no pueden ejecutarse en el servidor. Entonces si no lo llegas a poner nos dará un **error 404**

CAPITULO 3

1. Creación del contenido de types

- 1.1 En la carpeta Types vamos a poner tres archivos para las tres entidades que tenemos en nuestro proyecto, Clientes.type.ts , Producto.type.ts , Proveedor.type.ts.

```
▽ src\app
  > components
  > fonts
  > pages
  ▽ types
    TS Clientes.type.ts
    TS Producto.type.ts
    TS Proveedor.type.ts
```

- 1.2 Lo que contendrá estos archivos será lo que en el back-end definimos como interfaces.

1.2.1 Así se verá el de Producto

```
1 // src/types/productos.ts
2
3 import { Clientes } from "./Clientes.type";
4 import { Proveedores } from "./Proveedor.type";
5
6 export interface Productos {
7   _id: string;
8   nombre_producto: string;
9   cantidad: number;
10  precio: number;
11  proveedor: Proveedores[]; // Ajusta según tus necesidades
12  cliente: Clientes[]; // Ajusta según tus necesidades
13  activo?: boolean; // Para saber si el producto está activo
14 }
15
```

1.2.2 Así se verá el de Proveedor

```
● ● ●  
1 export interface Proveedores{  
2     id_proveedor: string;  
3     nombre_proveedor: string;  
4     email_proveedor: string;  
5     celular_proveedor: string;  
6     activo_proveedor?: boolean;  
7 }
```

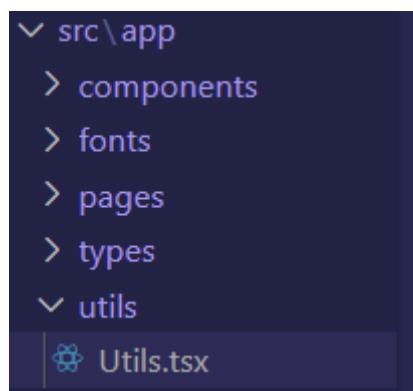
1.2.3 Así se verá el de Clientes

```
● ● ●  
1 export interface Clientes {  
2  
3     id_cliente: string;  
4     nombre_cliente: string;  
5     email_cliente: string;  
6     celular_cliente: string;  
7     activo_cliente?: boolean;  
8 }
```

CAPITULO 4

1. Creación del contenido de la carpeta utils

1.1 Esta carpeta contiene lo que antes llamábamos las validaciones, entonces aquí verás las validaciones más comunes y ya las que quieras poner adicionales.



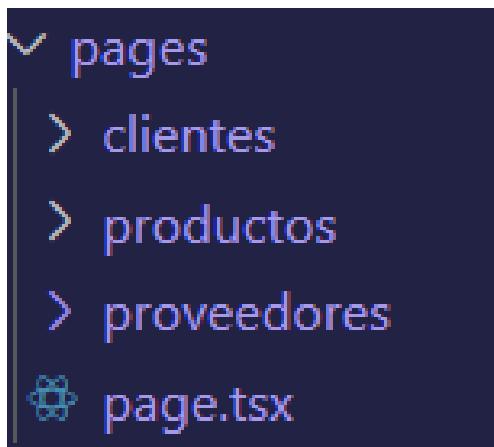
1.2 Vamos a crear el archivo Utils.tsx y en este colocamos las validaciones que necesitamos.

```
1 function normalizeText(input: string): string {
2   // Valida que el input sea un string
3   if (typeof input !== "string") {
4     throw new TypeError("Input must be a string");
5   }
6
7   // Define los caracteres que se reemplazarán y sus reemplazos
8   const from = "éíóúáÃÍÓÜAEIOU";
9   const to = "eiouaaEIOUaeiou";
10
11  // Crea un mapa para hacer las sustituciones
12  const mapping = new Map<string, string>();
13  for (let i = 0; i < from.length; i++) {
14    mapping.set(from[i], to[i]);
15  }
16
17  // Normaliza el texto eliminando espacios, caracteres especiales y convirtiendo a minúsculas
18  const result = input
19    .replace(/\s+/g, "") // Elimina todos los espacios
20    .split("") // Convierte el string en un array de caracteres
21    .map((char) => mapping.get(char) || char) // Reemplaza los caracteres o los deja intactos si no están en el mapa
22    .join("") // Junta los caracteres de nuevo en un string
23    .toLowerCase(); // Convierte todo el texto a minúsculas
24
25  return result;
26}
27
28 export default normalizeText;
29
```

CAPITULO 5

1. Creación del contenido de la carpeta pages

Esta carpeta contiene ya las páginas de cada entidad de nuestro proyecto.



IMPORTANTE

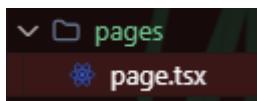
La navegación en **Next.js** se gestiona a través de un archivo page.tsx dentro de cada carpeta. Este archivo actúa como punto de entrada para la ruta que representa.

¿Cómo funciona la navegación en Next.js?

- En **Next.js**, la estructura de carpetas dentro de pages/ define las rutas de la aplicación.
- Para cada página que queramos mostrar, necesitamos un archivo page.tsx dentro de la carpeta correspondiente.

Ejemplo básico:

1. Dentro de la carpeta pages/, creamos el archivo page.tsx.



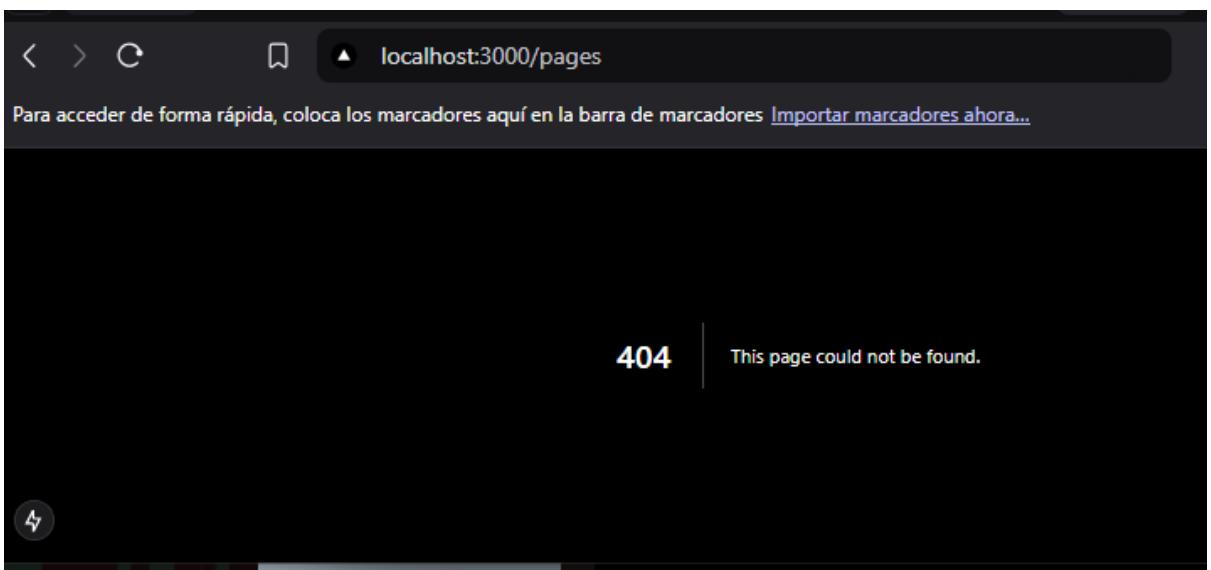
Este archivo debe contener una función que se exporta por defecto. Esta función es lo que Next.js mostrará cuando visites esa ruta.



```
1 export default function pagePrincipal(){
2     return <h1>Hoooolaaa</h1>
3 }
```

Probar la navegación:

1. Ejecutamos el proyecto con **npm run dev**, que abre el servidor en el puerto 3000 por defecto.
2. Al ir a <http://localhost:3000>, veremos lo que el archivo page.tsx de la carpeta principal contiene.
3. Si no tiene ese page.tsx va a salir esto.



Rutas anidadas:

Para crear subpáginas, simplemente anidamos carpetas dentro de pages/ y colocamos un archivo page.tsx en cada una.

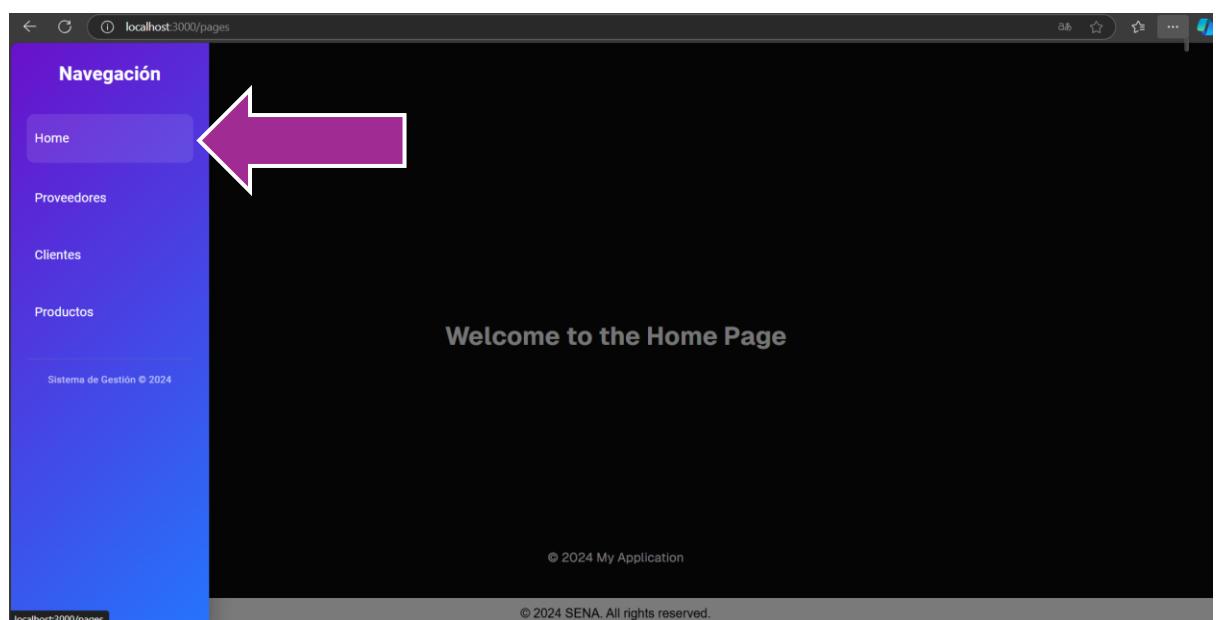
Ya sabiendo esto sigamos en la creación de los pages de cada entidad.

2. Creación del Home

2.1 La codificación del home la colocaremos en dos archivos, uno de ellos será en el page.tsx y el page de app.



La razón es que en el page.tsx de la carpeta pages nos dejara navegar en la barra de navegación de nuestro sitio web.



En este page.tsx de la carpeta pages colocaremos el siguiente código para home.

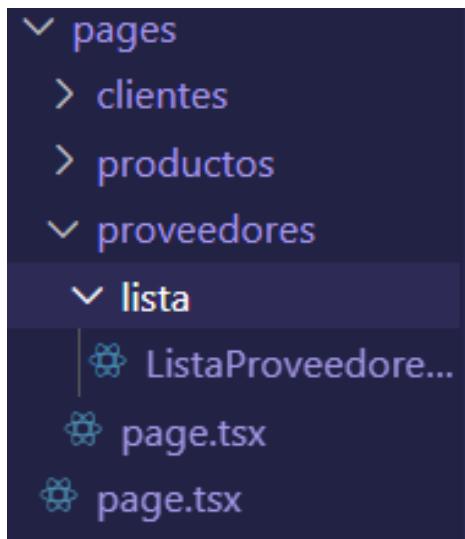
```
● ● ●
1  'use client'
2  import React from "react";
3  import type { AppProps } from "next/app"; // Importamos el tipo adecuado de Next.js
4  import ProveedorLista from "./proveedores/lista/ListaProveedores";
5  import styles from "../page.module.css";
6
7
8  export default function page() {
9    return (
10      <div className={styles.page}>
11
12        {/* Aquí colocamos el Navbar para que aparezca en La página de Home */}
13
14
15
16        <main className={styles.main}>
17          <h1>Welcome to the Home Page</h1>
18          {/* Agrega el contenido de tu página aquí */}
19        </main>
20
21        <footer className={styles.footer}>
22          <p>© 2024 My Application</p>
23        </footer>
24      </div>
25    );
26  }
27
28
```

Y para el page.tsx de app será este

```
● ● ●
1
2  import Image from "next/image";
3  import styles from "../page.module.css";
4
5
6
7  export default function Home() {
8    return (
9      <div className={styles.page}>
10
11        {/* Aquí colocamos el Navbar para que aparezca en La página de Home */}
12
13
14
15        <main className={styles.main}>
16          <h1>Welcome to the Home Page</h1>
17          {/* Agrega el contenido de tu página aquí */}
18        </main>
19
20        <footer className={styles.footer}>
21          <p>© 2024 My Application</p>
22        </footer>
23      </div>
24    );
25  }
26
```

3. Creación del contenido del page de proveedores

3.1 En nuestra carpeta de pages vamos a crear la carpeta de proveedores y en esta hacemos otra carpeta la cual llamaremos lista, y esta tendrá un archivo llamado ListaProveedores.jsx. Recuerda colocar el page.tsx el cual nos va permitir la navegación en esta página.



3.2 En el page.jsx de proveedores vamos a poner la siguiente configuración.

```
● ● ●  
1 import ProveedorLista from "./lista/ListaProveedores"  
2 export default function ProveedoresPage()  
3 {  
4     return (  
5         <div>  
6             <ProveedorLista/>  
7         </div>  
8     )  
9  
10    }  
11 }
```

3.3 En el archivo ListaProveedores.jsx vamos a colocar la siguiente codificación.

```
● ● ●  
1  'use client';  
2  import React, { useState, useEffect } from "react";  
3  import {  
4    Container,  
5    Grid,  
6    Card,  
7    CardContent,  
8    Typography,  
9    Button,  
10   Dialog,  
11   DialogTitle,  
12   DialogContent,  
13   DialogActions,  
14   TextField,  
15   Snackbar,  
16   Alert,  
17 } from "@mui/material";  
18 import { styled } from "@mui/system";
```

Estas importaciones permiten crear una interfaz interactiva en React:

- **'use client'**: Habilita la ejecución del código en el cliente en Next.js.
- **React, useState, useEffect**: Controlan el estado y los efectos del ciclo de vida del componente.
- **Material UI Components**:
 - Container, Grid, Card, CardContent: Estructuran y organizan el diseño.
 - Typography: Muestra texto con estilos predefinidos.
 - Button, Dialog, DialogTitle, DialogContent, DialogActions: Manejan botones y diálogos modales.
 - TextField: Campo de entrada de texto.
 - Snackbar, Alert: Muestran notificaciones.
- **styled**: Permite crear componentes personalizados con estilos.

3.4 Siguiente de las importaciones definimos los estilos que queremos que tenga nuestra pagina. Te recuerdo que puedes hacer un archivo aparte de css si quieres que todo predefinido y no repetir código.

```
1 // Estilo para las tarjetas dinámicas
2 const StyledCard = styled(Card)(({ theme }) => ({
3   transition: "transform 0.3s ease, box-shadow 0.3s ease",
4   "&:hover": {
5     transform: "translateY(-8px)", // Movimiento suave al hover
6   },
7   width: "100%", // Ancho completo en pantallas pequeñas
8   marginBottom: theme.spacing(3),
9   borderRadius: "20px", // Bordes más redondeados
10  backgroundColor: "#f9f9f9", // Fondo suave para las tarjetas
11  padding: theme.spacing(3), // Espaciado interno
12  [theme.breakpoints.up("md")]: {
13    width: "75%", // Ancho más grande en pantallas medianas y grandes
14    margin: "auto", // Centrado horizontal en pantallas más grandes
15  },
16));
17
18 // Estilo para los botones de acción personalizados
19 const ActionButton = styled(Button)(({ theme }) => ({
20   margin: theme.spacing(1),
21   padding: theme.spacing(1.5),
22   fontSize: "0.875rem",
23   fontWeight: "bold",
24   textTransform: "none",
25   borderRadius: "10px",
26   transition: "background-color 0.3s ease",
27 }));
28
29 const ActivateButton = styled(ActionButton)({
30   backgroundColor: "#4caf50",
31   color: "#fff",
32   "&:hover": {
33     backgroundColor: "#388e3c",
34   },
35 });
36
37 const DeactivateButton = styled(ActionButton)({
38   backgroundColor: "#ff9800",
39   color: "#fff",
40   "&:hover": {
41     backgroundColor: "#f57c00",
42   },
43 });
44
45 const DeleteButton = styled(ActionButton)({
46   backgroundColor: "#f44336",
47   color: "#fff",
48   "&:hover": {
49     backgroundColor: "#d32f2f",
50   },
51 });
52
53 const UpdateButton = styled(ActionButton)({
54   backgroundColor: "#2196f3",
55   color: "#fff",
56   "&:hover": {
57     backgroundColor: "#1976d2",
58   },
59 });
```

Los estilos aplicados crean una interfaz moderna e interactiva:

- **StyledCard**: Define tarjetas con una suave animación al hacer hover, bordes redondeados, y un fondo claro. En pantallas grandes, se centra y ajusta su ancho al 75%.
- **Botones de acción (ActionButton, ActivateButton, DeactivateButton, DeleteButton, UpdateButton)**: Los botones personalizados tienen colores distintivos y un cambio de color suave al hacer hover. Cada botón tiene bordes redondeados, un tamaño de fuente específico, y un diseño sin capitalizar el texto para un estilo más limpio y profesional.

3.5 Ahora vamos a colocar el componente que se encarga de gestionar una lista de proveedores, permitiendo realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) a través de una interfaz de usuario.

1. Declaración del Componente y Estado Local

```
● ● ●

1 const ProveedorLista = () => {
2   const [proveedores, setProveedores] = useState([]);
3   const [openModal, setOpenModal] = useState(false);
4   const [formValues, setFormValues] = useState({
5     nombre_proveedor: "",
6     email_proveedor: "",
7     celular_proveedor: "",
8     activo_proveedor: true,
9   });
10  const [selectedProveedor, setSelectedProveedor] = useState<any>(null);
11  const [openSnackbar, setOpenSnackbar] = useState(false);
12  const [errorMessage, setErrorMessage] = useState("");
13  const [actionType, setActionType] = useState<'create' | 'update'>('create');
14
```

Tipos de Estado:

- **proveedores**: Un array de objetos de tipo Proveedor que almacena la lista de proveedores.
- **openModal**: Un booleano que controla la visibilidad del modal.
- **formValues**: Un objeto de tipo FormValues que contiene los valores del formulario.
- **selectedProveedor**: Puede ser un objeto de tipo Proveedor o null, utilizado para almacenar el proveedor seleccionado.
- **openSnackbar**: Un booleano que indica si el snackbar debe estar visible.

- **errorMessage**: Una cadena que almacena el mensaje de error a mostrar en el snackbar.
- **actionType**: Un string que indica si la acción actual es 'create' o 'update'.

2. Cargar Proveedores

```
● ● ●  
1 useEffect(() => {  
2   fetchProveedores();  
3 }, []);
```

- **useEffect**: Se ejecuta una vez al montar el componente, llamando a la función fetchProveedores para obtener la lista de proveedores.

3. Función para Obtener Proveedores

```
● ● ●  
1 const fetchProveedores = async () => {  
2   try {  
3     const respuesta = await fetch('http://localhost:2000/api/proveedores');  
4     if (!respuesta.ok) throw new Error('Error al obtener todos los proveedores');  
5     const data = await respuesta.json();  
6     setProveedores(data);  
7   } catch (error) {  
8     console.error('Error al obtener los proveedores: ', error);  
9     setErrorMessage('Error al obtener los proveedores');  
10    setOpenSnackbar(true);  
11  }  
12};
```

- **fetchProveedores**: Realiza una solicitud a la API para obtener la lista de proveedores. Se define que data es de tipo Proveedor[] (array de Proveedores). En caso de error, se muestra un mensaje en el snackbar.

4. Manejo de Modal para Crear/Actualizar Proveedor

```
● ● ●  
1  const handleOpenModal = (proveedor: any = null) => {  
2      setSelectedProveedor(proveedor);  
3      if (proveedor) {  
4          setFormValues({  
5              nombre_proveedor: proveedor.nombre_proveedor,  
6              email_proveedor: proveedor.email_proveedor,  
7              celular_proveedor: proveedor.celular_proveedor,  
8              activo_proveedor: proveedor.activo_proveedor,  
9          });  
10         set ActionType('update');  
11     } else {  
12         setFormValues({  
13             nombre_proveedor: "",  
14             email_proveedor: "",  
15             celular_proveedor: "",  
16             activo_proveedor: true,  
17         });  
18         set ActionType('create');  
19     }  
20     setOpenModal(true);  
21 };
```

- **handleOpenModal:** Abre el modal para crear o editar un proveedor. Si se pasa un proveedor, se cargan sus datos en el formulario y se establece el tipo de acción a 'update'. Si no, se preparan los valores para un nuevo proveedor.

5. Cerrar Modal

```
1 const handleCloseModal = () => {
2     setOpenModal(false);
3     setSelectedProveedor(null);
4 };
```

- **handleCloseModal**: Cierra el modal y reinicia el proveedor seleccionado.

6. Manejo de Cambios en el Formulario

```
1 const handleInputChange = (e: any) => {
2     const { name, value } = e.target;
3     setFormValues({
4         ...formValues,
5         [name]: value,
6     });
7 }
8
```

- **handleInputChange**: Actualiza los valores del formulario cuando el usuario cambia algún campo. Aquí, el evento es de tipo React.ChangeEvent<HTMLInputElement> para asegurar que se manejen correctamente los cambios en los inputs.

7. Guardar Proveedor

```
1  const handleSaveProveedor = async () => {
2    try {
3      let url;
4      let method;
5      if (actionType === 'create') {
6        url = 'http://localhost:2000/api/proveedores';
7        method = 'POST';
8      } else if (actionType === 'update') {
9        url = `http://localhost:2000/api/proveedores/update/${selectedProveedor._id}`;
10       method = 'PUT';
11     }
12     const response = await fetch(`${url}`, {
13       method: method,
14       headers: {
15         "Content-Type": "application/json",
16       },
17       body: JSON.stringify(formValues),
18     });
19     if (!response.ok) {
20       const errorMessage = await response.json();
21       throw new Error(errorMessage.error || 'Error al guardar el proveedor');
22     }
23     fetchProveedores();
24     handleCloseModal();
25   } catch (error) {
26     console.error("Error al guardar el proveedor:", error);
27     setErrorMessage("Error al guardar el proveedor");
28     setOpenSnackbar(true);
29   }
30 };
```

handleSaveProveedor: Dependiendo del tipo de acción, envía una solicitud a la API para crear o actualizar un proveedor. Los tipos de url y method se definen como string. En caso de error, se muestra un mensaje en el snackbar.

8. Activar y Desactivar Proveedor

```
● ● ●  
1 const handleActivate = async (id: string) => {  
2   try {  
3     await fetch(`http://localhost:2000/api/proveedores/active/${id}`, { method: 'PUT' });  
4     fetchProveedores();  
5   } catch (error) {  
6     console.error("Error al activar el proveedor:", error);  
7   }  
8 };  
9  
10 const handleDeactivate = async (id: string) => {  
11   try {  
12     await fetch(`http://localhost:2000/api/proveedores/deactive/${id}`, { method: 'PUT' });  
13     fetchProveedores();  
14   } catch (error) {  
15     console.error("Error al desactivar el proveedor:", error);  
16   }  
17 };
```

- **handleActivate** y **handleDeactivate**: Activan y desactivan un proveedor, respectivamente, mediante una solicitud PUT a la API.

9. Eliminar Proveedor

```
● ● ●  
1 const handleDelete = async (id: string) => {  
2   try {  
3     await fetch(`http://localhost:2000/api/proveedores/delete/${id}`, { method: 'DELETE' });  
4     fetchProveedores();  
5   } catch (error) {  
6     console.error("Error al eliminar el proveedor:", error);  
7   }  
8 };  
9
```

- **handleDelete**: Elimina un proveedor mediante una solicitud DELETE a la API.

10. Cerrar Snackbar

```
1 const handleCloseSnackbar = () => {  
2     setOpenSnackbar(false);  
3     setErrorMessage("");  
4 };
```

- **handleCloseSnackbar**: Cierra el snackbar y reinicia el mensaje de error.

3.6 Ahora en el mismo archivo vamos a hacer el return .

1. Contenedor Principal

```
1 return (  
2     <Container maxWidth="lg" style={{ marginTop: "100px" }}>
```

- **Container**: Un componente de Material-UI que ayuda a establecer un ancho máximo y a centrar el contenido.
- **maxWidth="lg"**: Establece un ancho máximo grande para el contenedor.
- **style={{ marginTop: "100px" }}**: Aplica un margen superior de 100 píxeles al contenedor.

2. Sección con Estilo

```
● ● ●  
1 <section style={{  
2   background: "linear-gradient(135deg, #6a11cb 0%, #2575fc 100%)", // Gradiente de colores  
3   padding: "20px", // Espaciado interno  
4   borderRadius: "10px", // Bordes redondeados  
5   color: "#fff" // Color del texto  
6 }}>
```

- **background**: Define un fondo con un gradiente de dos colores.
- **padding**: Aplica un espaciado interno de 20 píxeles.
- **borderRadius**: Redondea los bordes de la sección.
- **color**: Establece el color del texto a blanco.

3. Botón para Crear Proveedor

```
● ● ●  
1 <Button  
2   variant="contained"  
3   color="primary"  
4   onClick={() => handleOpenModal()}  
5   style={{ marginBottom: "30px" }} // Espaciado adicional  
6 >  
7   Crear Proveedor  
8 </Button>
```

- **variant="contained"**: Estilo del botón que lo hace resaltar.
- **color="primary"**: Aplica el color primario del tema al botón.
- **onClick**: Llama a handleOpenModal para abrir el modal.
- **style**: Aplica un margen inferior de 30 píxeles.

4. Lista de Proveedores

```
1 <Grid container spacing={4}>
2   {proveedores.map((proveedor: any) => (
3     <Grid item xs={12} md={6} key={proveedor._id}>
4       <StyledCard>
5         <CardContent>
6           <Typography variant="h6" component="div" gutterBottom>
7             {proveedor.nombre_proveedor}
8           </Typography>
9           <Typography variant="body2" color="textSecondary">
10            Email: {proveedor.email_proveedor}
11           </Typography>
12           <Typography variant="body2" color="textSecondary">
13            Celular: {proveedor.celular_proveedor}
14           </Typography>
15           <Typography
16             variant="body2"
17             style={{
18               color: proveedor.activo_proveedor ? "#4caf50" : "#f44336", // Verde si está activo, rojo si está inactivo
19               fontWeight: "bold"
20             }}
21           >
22             {proveedor.activo_proveedor ? "Activo" : "Desactivado"}
23           </Typography>
24           <UpdateButton onClick={() => handleOpenModal(proveedor)}>
25             Actualizar
26           </UpdateButton>
27           {proveedor.activo_proveedor ?
28             <DeactivateButton onClick={() => handleDeactivate(proveedor._id)}>
29               Desactivar
30             </DeactivateButton>
31           ) :
32             <ActivateButton onClick={() => handleActivate(proveedor._id)}>
33               Activar
34             </ActivateButton>
35           )
36           <DeleteButton onClick={() => handleDelete(proveedor._id)}>
37             Eliminar
38           </DeleteButton>
39         </CardContent>
40       </StyledCard>
41     </Grid>
42   ))
43 </Grid>
44 </section>
```

- **Grid container:** Crea un contenedor de cuadrícula que organiza los elementos de manera responsiva.
- **spacing={4}:** Establece el espacio entre los elementos de la cuadrícula.
- **proveedores.map(...):** Itera sobre el array de proveedores para crear una tarjeta para cada uno.
- **key={proveedor._id}:** Proporciona una clave única para cada elemento de la lista.
- **StyledCard:** Un componente que estiliza la tarjeta.
- **CardContent:** Contenedor para el contenido de la tarjeta.
- **Typography:** Usado para mostrar texto con diferentes estilos.
- **color:** Cambia el color del estado del proveedor basado en si está activo o no.

- **Botones:** Cada proveedor tiene botones para actualizar, activar/desactivar y eliminar.

5. Modal para Crear/Actualizar Proveedor



```

1 <Dialog open={openModal} onClose={handleCloseModal}>
2   <DialogTitle>{actionType === 'create' ? "Crear Proveedor" : "Actualizar Proveedor"}</DialogTitle>
3   <DialogContent>
4     <TextField
5       fullWidth
6       margin="normal"
7       label="Nombre"
8       name="nombre_proveedor"
9       value={formValues.nombre_proveedor}
10      onChange={handleInputChange}>
11    />
12    <TextField
13      fullWidth
14      margin="normal"
15      label="Email"
16      name="email_proveedor"
17      value={formValues.email_proveedor}
18      onChange={handleInputChange}>
19    />
20    <TextField
21      fullWidth
22      margin="normal"
23      label="Celular"
24      name="celular_proveedor"
25      value={formValues.celular_proveedor}
26      onChange={handleInputChange}>
27    />
28  </DialogContent>
29  <DialogActions>
30    <Button onClick={handleCloseModal} color="secondary">Cancelar</Button>
31    <Button onClick={handleSaveProveedor} color="primary">
32      {actionType === 'create' ? "Crear" : "Actualizar"}</Button>
33  </DialogActions>
34</Dialog>
35
36

```

- **Dialog:** Componente que representa el modal.
- **open:** Controla la visibilidad del modal.
- **onClose:** Función que se llama al cerrar el modal.
- **DialogTitle:** Muestra el título del modal dependiendo de la acción (crear o actualizar).
- **DialogContent:** Contiene los campos del formulario.
- **TextField:** Campos de entrada para el nombre, email y celular.
- **DialogActions:** Contiene los botones de acción del modal (cancelar y crear/actualizar).

5. Snackbar para Mensajes

```
1  /* Snackbar para mensajes */
2  <Snackbar open={openSnackbar} autoHideDuration={3000} onClose={handleCloseSnackbar}>
3      <Alert severity={errorMessage ? "error" : "success"} onClose={handleCloseSnackbar}>
4          {errorMessage || "Operación exitosa"}
5      </Alert>
6  </Snackbar>
7 </Container>
8 );
9 };
10
11 export default ProveedorLista;
```

- **Snackbar:** Componente que muestra mensajes temporales.
- **open:** Controla la visibilidad del snackbar.
- **autoHideDuration={3000}:** Duración en milisegundos antes de que el snackbar se oculte automáticamente.
- **onClose:** Función que se llama al cerrar el snackbar.
- **Alert:** Muestra un mensaje de alerta con un nivel de severidad (éxito o error).
- **errorMessage:** Muestra un mensaje de error si existe; de lo contrario, muestra un mensaje de éxito.

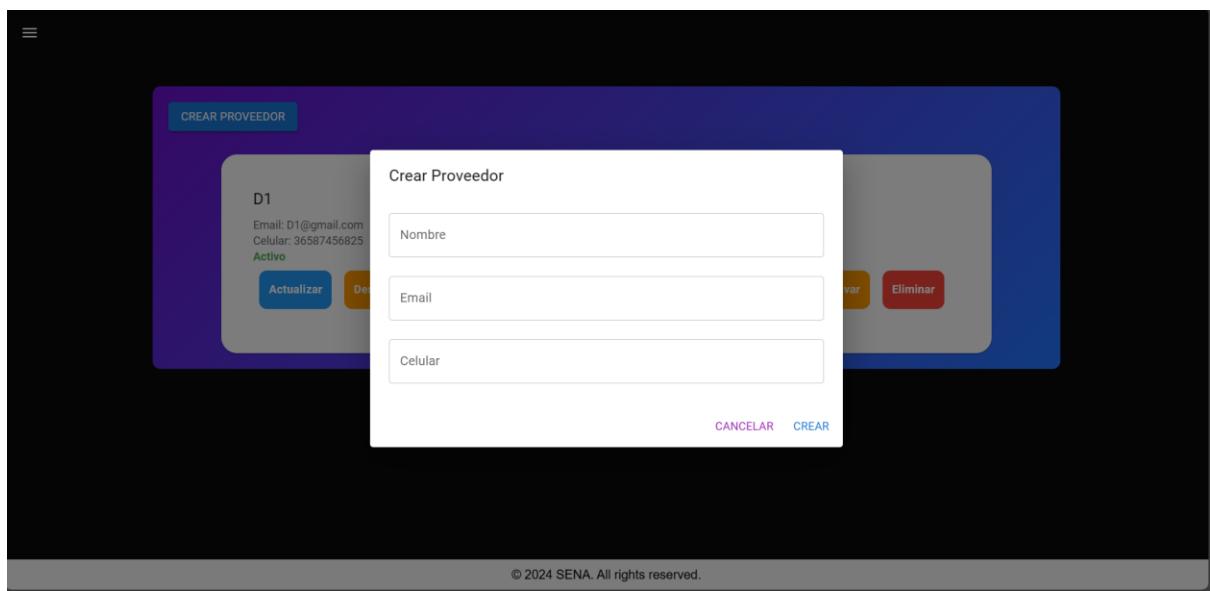
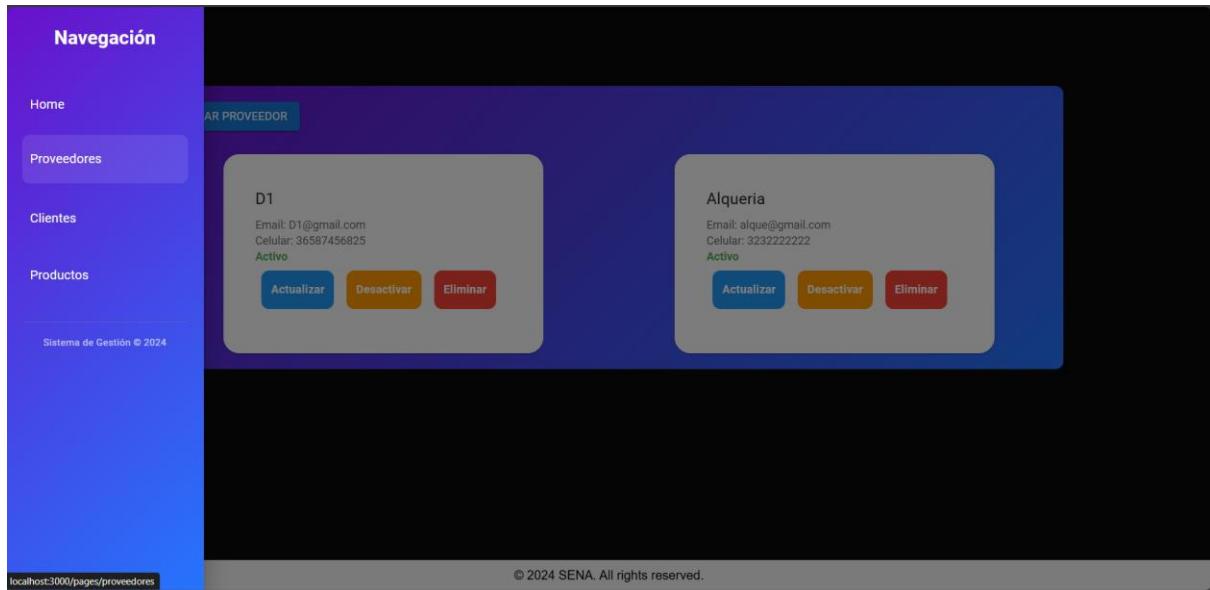
3.7 Al final se tiene que ver así el componente al final.

```
● ● ●
1 'use client';
2 import React, { useState, useEffect } from "react";
3 import {
4   Container,
5   Grid,
6   Card,
7   CardContent,
8   Typography,
9   Button,
10  Dialog,
11  DialogTitle,
12 DialogContent,
13  DialogActions,
14  Textfield,
15  Snackbar,
16  Alert,
17 } from "@mui/material";
18 import { styled } from "@mui/system";
19
20 // Estilo para las tarjetas dinámicas
21 const StyledCard = styled(Card)(({ theme }) => ({
22   transition: "transform 0.3s ease, box-shadow 0.3s ease",
23   "&:hover": {
24     transform: "translateY(-8px)", // Movimiento suave al hover
25   },
26   width: "100%", // Ancho completo en pantallas pequeñas
27   marginBottom: theme.spacing(3),
28   borderRadius: "20px", // Bordes más redondeados
29   backgroundColor: "#f9f9f9", // Fondo suave para las tarjetas
30   padding: theme.spacing(3), // Espaciado interno
31   [theme.breakpoints.up("md")]: {
32     width: "55%", // Ancho más grande en pantallas medianas y grandes
33     margin: "auto", // Centrado horizontal en pantallas más grandes
34   },
35 }));
36
37 // Estilo para los botones de acción personalizados
38 const ActionButton = styled(Button)(({ theme }) => ({
39   margin: theme.spacing(1),
40   padding: theme.spacing(1.5),
41   fontSize: "0.875rem",
42   fontWeight: "bold",
43   textTransform: "none",
44   borderRadius: "10px",
45   transition: "background-color 0.3s ease",
46 }));
47
48 const ActivateButton = styled(ActionButton)(({
49   backgroundColor: "#4caf50",
50   color: "#fff",
51   "&:hover": {
52     backgroundColor: "#388e3c",
53   },
54 }));
55
56 const DeactivateButton = styled(ActionButton)(({
57   backgroundColor: "#ff9800",
58   color: "#fff",
59   "&:hover": {
60     backgroundColor: "#f57c00",
61   },
62 }));
63
64 const DeleteButton = styled(ActionButton)(({
65   backgroundColor: "#f44336",
66   color: "#fff",
67   "&:hover": {
68     backgroundColor: "#d32f2f",
69   },
70 }));
71
72 const UpdateButton = styled(ActionButton)(({
73   backgroundColor: "#2196f3",
74   color: "#fff",
75   "&:hover": {
76     backgroundColor: "#1976d2",
77   },
78 }));
79
80 const ProveedorLista = () => {
81   const [proveedores, setProveedores] = useState([]);
82   const [openModal, setOpenModal] = useState(false);
83   const [formValues, setFormValues] = useState({
84     nombre_proveedor: "",
85     email_proveedor: "",
86     celular_proveedor: "",
87     activo_proveedor: true,
88   });
89   const [selectedProveedor, setSelectedProveedor] = useState(null);
90   const [openSnackbar, setOpenSnackbar] = useState(false);
91   const [errorMessage, setErrorMessage] = useState("");
92   const [actionType, setActionType] = useState('create' | 'update' | 'create');
93
94   useEffect(() => {
95     fetchProveedores();
96   }, []);
97
98   const fetchProveedores = async () => {
99     try {
100       const respuesta = await fetch('http://localhost:2000/api/proveedores');
101       if (!respuesta.ok) throw new Error('Error al obtener todos los proveedores');
102       const data = await respuesta.json();
103       setProveedores(data);
104     } catch (error) {
105       console.error('Error al obtener los proveedores: ', error);
106       setErrorMessage('Error al obtener los proveedores');
107       setOpenSnackbar(true);
108     }
109   };
}
```

```
 1 const handleOpenModal = (proveedor: any = null) => {
 2   setSelectedProveedor(proveedor);
 3   if (proveedor) {
 4     setFormValues({
 5       nombre_proveedor: proveedor.nombre_proveedor,
 6       email_proveedor: proveedor.email_proveedor,
 7       celular_proveedor: proveedor.celular_proveedor,
 8       activo_proveedor: proveedor.activo_proveedor,
 9     });
10   setActionType('update');
11 } else {
12   setFormValues({
13   nombre_proveedor: '',
14   email_proveedor: '',
15   celular_proveedor: '',
16   activo_proveedor: true,
17 });
18 setActionType('create');
19 }
20 setOpenModal(true);
21 };
22
23 const handleCloseModal = () => {
24   setOpenModal(false);
25   setSelectedProveedor(null);
26 };
27
28 const handleInputChange = (e: any) => {
29   const { name, value } = e.target;
30   setFormValues({
31     ...formValues,
32     [name]: value,
33   });
34 };
35
36 const handleSaveProveedor = async () => {
37   try {
38     let url;
39     let method;
40     if (actionType === 'create') {
41       url = 'http://localhost:2000/api/proveedores';
42       method = 'POST';
43     } else if (actionType === 'update') {
44       url = 'http://localhost:2000/api/proveedores/update/${selectedProveedor._id}';
45       method = 'PUT';
46     }
47     const response = await fetch(`${url}`, {
48       method: method,
49       headers: {
50         'Content-Type': 'application/json',
51       },
52       body: JSON.stringify(formValues),
53     });
54     if (!response.ok) {
55       const errorData = await response.json();
56       throw new Error(errorData.error || 'Error al guardar el proveedor');
57     }
58     fetchProveedores();
59     handleCloseModal();
60   } catch (error) {
61     console.error("Error al guardar el proveedor:", error);
62     setErrorMessage("Error al guardar el proveedor");
63     setOpenSnackbar(true);
64   }
65 };
66
67 const handleActivate = async (id: string) => {
68   try {
69     await fetch(`http://localhost:2000/api/proveedores/active/${id}`, { method: 'PUT' });
70     fetchProveedores();
71   } catch (error) {
72     console.error("Error al activar el proveedor:", error);
73   }
74 };
75
76 const handleDeactivate = async (id: string) => {
77   try {
78     await fetch(`http://localhost:2000/api/proveedores/deactive/${id}`, { method: 'PUT' });
79     fetchProveedores();
80   } catch (error) {
81     console.error("Error al desactivar el proveedor:", error);
82   }
83 };
84
85 const handleDelete = async (id: string) => {
86   try {
87     await fetch(`http://localhost:2000/api/proveedores/delete/${id}`, { method: 'DELETE' });
88     fetchProveedores();
89   } catch (error) {
90     console.error("Error al eliminar el proveedor:", error);
91   }
92 };
93
94 const handleCloseSnackbar = () => {
95   setOpenSnackbar(false);
96   setErrorMessage("");
97 };
98
```

```
1  return (
2    <Container maxWidth="lg" style={{ marginTop: "100px" }}>
3      <Section style={{
4        background: "linear-gradient(135deg, #6a11cb 0%, #2575fc 100%)", // Gradiente de colores
5        padding: "20px", // Espaciado interno
6        borderRadius: "10px", // Bordes redondeados
7        color: "#fff" // Color del texto
8      }}>
9        <Button
10          variant="contained"
11          color="primary"
12          onClick={() => handleOpenModal()}
13          style={{ marginBottom: "30px" }} // Espaciado adicional
14        >
15          Crear Proveedor
16        </Button>
17        <Grid container spacing={4}>
18          {proveedores.map((proveedor: any) => (
19            <Grid item xs={12} md={6} key={proveedor._id}>
20              <StyledCard>
21                <CardContent>
22                  <Typography variant="h6" component="div" gutterBottom>
23                    {proveedor.nombre_proveedor}
24                  </Typography>
25                  <Typography variant="body2" color="textSecondary">
26                    Email: {proveedor.email_proveedor}
27                  </Typography>
28                  <Typography variant="body2" color="textSecondary">
29                    Celular: {proveedor.celular_proveedor}
30                  </Typography>
31                  <Typography
32                    variant="body2"
33                    style={{
34                      color: proveedor.activo_proveedor ? "#4caf50" : "#f44336", // Verde si est&aacute; activo, roja si est&aacute; inactivo
35                      fontWeight: "bold"
36                    }}
37                  >
38                    {proveedor.activo_proveedor ? "Activo" : "Desactivado"}
39                  </Typography>
40                  <UpdateButton onClick={() => handleOpenModal(proveedor)}>
41                    Actualizar
42                  </UpdateButton>
43                  {proveedor.activo_proveedor ? (
44                    <DeactivateButton onClick={() => handleDeactivate(proveedor._id)}>
45                      Desactivar
46                    </DeactivateButton>
47                  ) : (
48                    <ActivateButton onClick={() => handleActivate(proveedor._id)}>
49                      Activar
50                    </ActivateButton>
51                  )}
52                  <DeleteButton onClick={() => handleDelete(proveedor._id)}>
53                    Eliminar
54                  </DeleteButton>
55                <CardContent>
56              <StyledCard>
57            </Grid>
58          )));
59        </Grid>
60      </Section>
61
62      <Dialog open={openModal} onClose={handleCloseModal}>
63        <DialogTitle>{actionType === 'create' ? 'Crear Proveedor' : "Actualizar Proveedor"}</DialogTitle>
64        <DialogContent>
65          <TextField
66            fullWidth
67            margin="normal"
68            label="Nombre"
69            name="nombre_proveedor"
70            value={formValues.nombre_proveedor}
71            onChange={handleInputChange}
72          />
73          <TextField
74            fullWidth
75            margin="normal"
76            label="Email"
77            name="email_proveedor"
78            value={formValues.email_proveedor}
79            onChange={handleInputChange}
80          />
81          <TextField
82            fullWidth
83            margin="normal"
84            label="Celular"
85            name="celular_proveedor"
86            value={formValues.celular_proveedor}
87            onChange={handleInputChange}
88          />
89        </DialogContent>
90        <DialogActions>
91          <Button onClick={handleCloseModal} color="secondary">Cancelar</Button>
92          <Button onClick={handleSaveProveedor} color="primary">
93            {actionType === 'create' ? "Crear" : "Actualizar"}
94          </Button>
95        </DialogActions>
96      </Dialog>
97
98      {/* Snackbar para mensajes */}
99      <Snackbar open={openSnackbar} autoHideDuration={3000} onClose={handleCloseSnackbar}>
100        <Alert severity={errorMessage ? "error" : "success"} onClose={handleCloseSnackbar}>
101          {errorMessage || "Operaci&on exitosa"}
102        </Alert>
103      </Snackbar>
104    </Container>
105  );
106 };
107
108 export default ProveedorLista;
```

3.8 Se tiene que ver así nuestra interfaz



4. Actividad: Implementación de la Página de Clientes

Objetivo: Crear la página de clientes para visualizar, agregar, editar y eliminar clientes en la aplicación.

Instrucciones:

1. Planificación:

- Reflexiona sobre cómo debe ser la interfaz de usuario de la página de clientes. ¿Qué información es esencial mostrar para cada cliente? (Ejemplo: Nombre, Correo, Teléfono, etc.)

2. Creación del Componente:

- Crea un nuevo archivo para el componente de clientes en tu proyecto.
- Asegúrate de que el componente esté correctamente estructurado para manejar las diferentes funcionalidades.

3. Definición del Estado:

- Piensa en las variables que necesitarás para almacenar la lista de clientes y el estado del formulario de edición.
- Considera cómo manejarás los errores y los mensajes de éxito al realizar acciones.

4. Desarrollo de Funciones:

- Define claramente las funciones necesarias para:
 - Obtener la lista de clientes.
 - Agregar un nuevo cliente.
 - Editar un cliente existente.
 - Eliminar un cliente.

5. Interacción con el Usuario:

- Planifica cómo los usuarios interactuarán con la página. ¿Cómo se abrirá el formulario para agregar o editar un cliente?

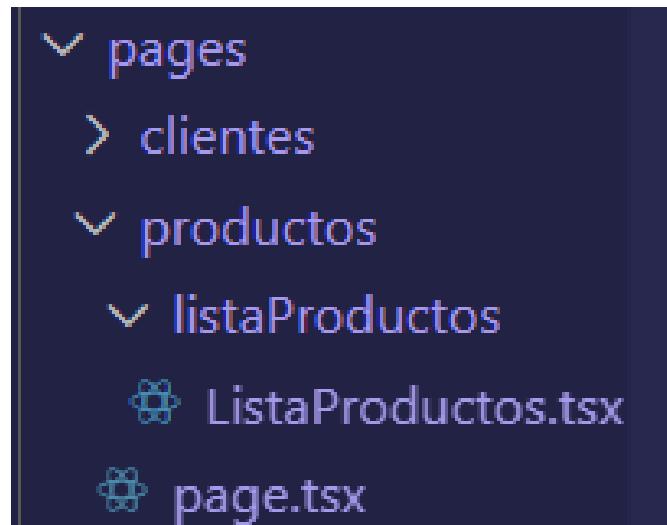
- Determina cómo se mostrarán los mensajes de confirmación o error al usuario.

Entregables:

- Un componente de clientes completamente funcional.
- Un breve informe que explique el diseño y las decisiones tomadas durante el desarrollo de la página.

5. Creación del contenido de producto

5.1 En nuestra carpeta de pages vamos a crear la carpeta de productos y en esta hacemos otra carpeta la cual llamaremos listaProductos y esta tendrá un archivo llamado ListaProductos.jsx. Recuerda colocar el page.tsx el cual nos va a permitir la navegación en esta página.



5.2 En el page.jsx de clientes vamos a poner la siguiente configuración.

```
1
2 import ProductosLista from "./listaProductos/ListaProductos";
3
4 export default function ProductosPage() {
5     // Aquí puedes definir un productId para pasar al componente
6
7
8     return (
9         <div>
10            /* Renderiza el componente de gestión de proveedores y clientes */
11            <ProductosLista />
12            </div>
13        );
14    }
15}
16
```

5.3 En el archivo ListaProductos.jsx vamos a colocar la siguiente codificación.

```
● ● ●  
1  'use client';  
2  
3  import React, { useEffect, useState } from 'react';  
4  import {  
5    Container,  
6    Typography,  
7    Button,  
8    Snackbar,  
9    Alert,  
10   Table,  
11   TableBody,  
12   TableCell,  
13   TableContainer,  
14   TableHead,  
15   TableRow,  
16   Paper,  
17   Modal,  
18   TextField,  
19   Box,  
20   MenuItem,  
21   Select,  
22   FormControl,  
23  InputLabel,  
24  
25 } from '@mui/material';  
26 import { styled } from '@mui/system';  
27 import { Productos } from '@/app/types/Producto.type';  
28 import { Proveedores } from '@/app/types/Proveedor.type';  
29 import { Clientes } from '@/app/types/Clientes.type';  
30 import AddIcon from '@mui/icons-material/Add';  
31 import EditIcon from '@mui/icons-material/Edit';  
32 import DeleteIcon from '@mui/icons-material/Delete';  
33 import PersonAddIcon from '@mui/icons-material/PersonAdd';  
34 import BusinessIcon from '@mui/icons-material/Business';  
35 import ToggleOffIcon from '@mui/icons-material/ToggleOff';  
36 import ToggleOnIcon from '@mui/icons-material/ToggleOn';  
37 import Tooltip from '@mui/material/Tooltip';  
38 import Checkbox from '@mui/material/Checkbox';  
39 import { SelectChangeEvent } from '@mui/material/Select';  
40 import Swal from 'sweetalert2';  
41 import 'sweetalert2/src/sweetalert2.scss';  
42
```

Este código importa varios componentes de la librería @mui/material, entre otros paquetes y librerías, para construir una interfaz de usuario en React. El bloque use client al inicio indica que este código debe ejecutarse en el cliente. A continuación, se importan componentes de Material UI como Button, Table, Modal, y otros para crear elementos de interfaz como tablas, botones, formularios y notificaciones. También se incluyen íconos (AddIcon, EditIcon, etc.) y la biblioteca SweetAlert2 para mostrar alertas estilizadas.

5.4 Siguiente de las importaciones definimos los estilos que queremos que tenga nuestra página. Te recuerdo que puedes hacer un archivo aparte de ccs si quieras que todo esté predefinido y no repetir código.



```
1 // Estilos para botones y tabla
2 const ActionButton = styled(Button)(({ theme }) => ({
3   margin: theme.spacing(0.5),
4   padding: theme.spacing(1),
5   fontSize: '0.875rem',
6   fontWeight: 'bold',
7   textTransform: 'none',
8   borderRadius: '10px',
9 }));
```

Este código define un estilo personalizado para un modal. La constante style incluye propiedades para centrar el modal en la pantalla (top, left, y transform), ajustar su tamaño (width), color de fondo (bgcolor), borde (border), sombra (boxShadow), y espacio interno (p para padding). Esto hace que el modal se muestre centrado y con un diseño destacado sobre el fondo.

5.5A hora vamos a colocar el componente que se encarga de gestionar una lista de proveedores, permitiendo realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) a través de una interfaz de usuario.

Componente Principal



Definición: Define un componente funcional en React llamado ProductoLista, que utiliza TypeScript para las props.

Estados



Definición: Define varios estados usando useState, incluyendo la lista de productos, el nuevo producto a crear, mensajes de error y controles para los modales.

Funciones para Obtener Datos

```
● ● ●

1 const obtenerClientes = async () => {
2   const response = await fetch('http://localhost:2000/api/clientes');
3   const data = await response.json();
4   setClientes(data);
5 };
6
7 const obtenerProveedores = async () => {
8   const response = await fetch('http://localhost:2000/api/proveedores');
9   const data = await response.json();
10  setProveedores(data);
11 };
12
13
14 const obtenerProductos = async () => {
15   const response = await fetch('http://localhost:2000/api/productos');
16   const data = await response.json();
17   setProductos(data);
18 };
```

Definición: Estas funciones asíncronas obtienen datos de la API para los clientes, proveedores y productos, y actualizan el estado correspondiente.

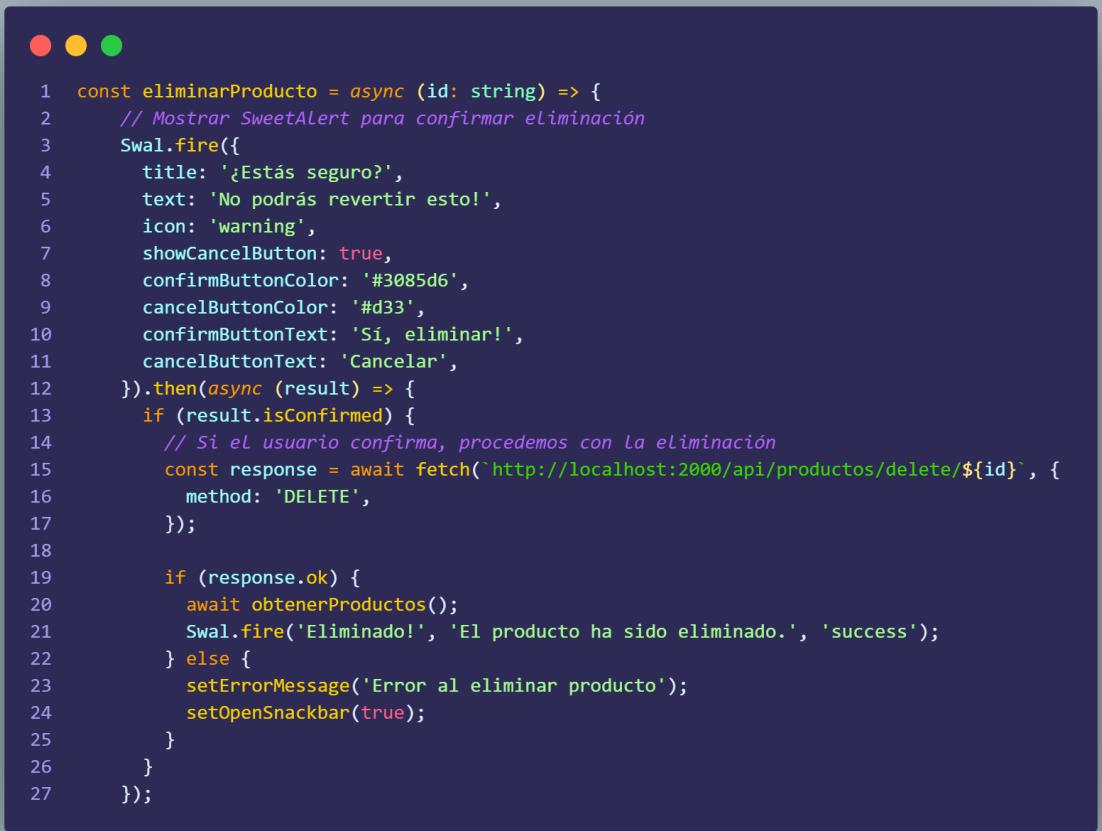
Crear Producto

```
● ● ●

1 const crearProducto = async (data: Omit<Productos, '_id'>) => {
2   const response = await fetch('http://localhost:2000/api/productos', {
3     method: 'POST',
4     headers: {
5       'Content-Type': 'application/json',
6     },
7     body: JSON.stringify(data),
8   });
9
10  if (response.ok) {
11    setNuevoProducto({ nombre_producto: '', cantidad: 0, precio: 0, proveedor: [], cliente: [], activo: true });
12    setOpenModal(false);
13    await obtenerProductos();
14  } else {
15    const errorData = await response.json();
16    setErrorMessage(errorData.message || 'Error al crear producto');
17    setOpenSnackbar(true);
18  }
19};
```

Definición: Crea un nuevo producto enviando una solicitud POST a la API y maneja la respuesta para actualizar el estado o mostrar un error.

Eliminar Producto



```
1 const eliminarProducto = async (id: string) => {
2     // Mostrar SweetAlert para confirmar eliminación
3     Swal.fire({
4         title: '¿Estás seguro?',
5         text: 'No podrás revertir esto!',
6         icon: 'warning',
7         showCancelButton: true,
8         confirmButtonColor: '#3085d6',
9         cancelButtonColor: '#d33',
10        confirmButtonText: 'Sí, eliminar!',
11        cancelButtonText: 'Cancelar',
12    }).then(async (result) => {
13        if (result.isConfirmed) {
14            // Si el usuario confirma, procedemos con la eliminación
15            const response = await fetch(`http://localhost:2000/api/productos/delete/${id}`, {
16                method: 'DELETE',
17            });
18
19            if (response.ok) {
20                await obtenerProductos();
21                Swal.fire('Eliminado!', 'El producto ha sido eliminado.', 'success');
22            } else {
23                setErrorMessage('Error al eliminar producto');
24                setOpenSnackbar(true);
25            }
26        }
27    });
}
```

Definición: Muestra un cuadro de alerta para confirmar la eliminación y, si se confirma, envía una solicitud DELETE a la API.

Manejo de Cambios en Selecciones

```
● ● ●  
1 const handleClienteChange = (event: SelectChangeEvent<string[]>) => {  
2   const {  
3     target: { value },  
4   } = event;  
5   setClientesSeleccionados(value as string[]); // Asegúrate de que sea un array de strings  
6 };  
7  
8 const handleProveedorChange = (event: SelectChangeEvent<string[]>) => {  
9   const {  
10    target: { value },  
11  } = event;  
12  setProveedoresSeleccionados(value as string[]); // Asegúrate de que sea un array de strings  
13};
```

Definición: Manejan los cambios en los campos de selección para los clientes y proveedores, actualizando el estado de los seleccionados.

Efectos

```
● ● ●  
1 useEffect(() => {  
2   obtenerProductos();  
3 }, []);  
4  
5 useEffect(() => {  
6   if (openModal) {  
7     obtenerClientes();  
8     obtenerProveedores();  
9   }  
10 }, [openModal]);
```

Definición: Efectos que se ejecutan al montar el componente para obtener productos, y al abrir el modal para obtener clientes y proveedores.

Manejo de Snackbar

```
1 const handleCloseSnackbar = () => {
2     setOpenSnackbar(false);
3     setErrorMessage('');
4 };
```

Definición: Cierra el snackbar y resetea el mensaje de error.

Control del Modal

```
1 const handleOpenModal = () => {
2     setOpenModal(true);
3 };
4
5 const handleCloseModal = () => {
6     setOpenModal(false);
7     setNuevoProducto({ nombre_producto: '', cantidad: 0, precio: 0, proveedor: [], cliente: [], activo: true });
8     setProveedoresSeleccionados ([]);
9     setCiudadesSeleccionadas ([]);
10};
```

Definición: Abre y cierra el modal, reiniciando los estados de los productos y selecciones al cerrar.

Manejo de Cambios en Entradas

```
● ● ●  
1 const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {  
2   const { name, value } = e.target;  
3   setNuevoProducto(prev => ({  
4     ...prev,  
5     [name]: name === 'nombre_producto' ? value.toUpperCase() : name === 'cantidad' || name === 'precio' ? Number(value) : value,  
6   }));  
7 };
```

Definición: Actualiza el estado del nuevo producto al cambiar los valores en los campos de entrada.

Apertura del Modal de Edición

```
● ● ●  
1 const handleOpenEditModal = (producto: Productos) => {  
2   setProductoEditado(producto);  
3   setOpenEditModal(true);  
4 };  
5  
6 const handleCloseEditModal = () => {  
7   setOpenEditModal(false);  
8   setProductoEditado(null);  
9 };
```

Definición: Maneja la apertura y cierre del modal para editar un producto seleccionado.

Actualizar Producto

```
● ● ●  
1 const actualizarProducto = async (id: string, data: Omit<Productos, '_id'>) => {  
2   const response = await fetch(`http://localhost:2000/api/productos/update/${id}`, {  
3     method: 'PUT',  
4     headers: {  
5       'Content-Type': 'application/json',  
6     },  
7     body: JSON.stringify(data),  
8   });  
9  
10  if (response.ok) {  
11    await obtenerProductos();  
12    handleCloseEditModal();  
13  } else {  
14    const errorMessage = await response.json();  
15    setErrorMessage(errorMessage.message || 'Error al actualizar producto');  
16    setOpenSnackbar(true);  
17  }  
18};
```

Definición: Envía una solicitud PUT para actualizar un producto específico y maneja la respuesta.

Manejo de Cambios en Edición

```
● ● ●  
1 const handleEditChange = (e: React.ChangeEvent<HTMLInputElement>) => {  
2   const { name, value } = e.target;  
3   if (productoEditado) {  
4     setProductoEditado((prev) => ({  
5       ...prev,  
6       [name]: name === 'nombre_producto' ? value.toUpperCase() : name === 'cantidad' || name === 'precio' ? Number(value) : value,  
7     }));  
8   }  
9};
```

Definición: Actualiza los campos del producto que se está editando al cambiar los valores en las entradas.

Envío del Formulario

```
1 const handleSubmit = (event: React.FormEvent<HTMLFormElement>) => {
2   event.preventDefault();
3
4   const clientesSeleccionadosDetalles = clientesSeleccionados.map(id => obtenerClientePorId(id));
5   const proveedoresSeleccionadosDetalles = proveedoresSeleccionados.map(id => obtenerProveedorPorId(id));
6
7   const productoData = {
8     ...nuevoProducto,
9     cliente: clientesSeleccionadosDetalles,
10    proveedor : proveedoresSeleccionadosDetalles,
11  };
12
13  crearProducto(productoData);
14};
15
```

Definición: Maneja el envío del formulario, creando un objeto con los detalles del nuevo producto y llamando a la función para crear el producto.

Obtener Cliente y Proveedor por ID

```
1 const obtenerClientePorId = (id: string): Clientes => {
2   return clientes.find(cliente => cliente.id_cliente === id) as Clientes;
3 };
4
5 const obtenerProveedorPorId = (id: string): Proveedores => {
6   return proveedores.find(proveedor => proveedor.id_proveedor === id) as Proveedores;
7 };
```

Definición: Busca un cliente o proveedor en los arrays de clientes y proveedores usando su ID.

Cambiar Estado Activo

```
1 const toggleActivo = async (id: string, activo: boolean) => {
2   const response = await fetch(`http://localhost:2000/api/productos/${activo ? 'deactive' : 'active'}/${id}`, {
3     method: 'PUT',
4   });
5
6   if (response.ok) {
7     await obtenerProductos();
8   } else {
9     setErrorMessage('Error al cambiar estado del producto');
10    setOpenSnackbar(true);
11  }
12};
13
```

Definición: Cambia el estado activo de un producto específico enviando una solicitud PATCH a la API.

5.6 Ahora en el mismo archivo vamos a agregar el return.

Contenedor Principal

```
1 return (
2   <Container
3     maxWidth="lg"
4     style={{
5       marginTop: '20px',
6       height: '90vh',
7       display: 'flex',
8       justifyContent: 'center',
9       alignItems: 'center',
10      flexDirection: 'column',
11      background: 'linear-gradient(135deg, #6a11cb 0%, #2575fc 100%)',
12      paddingBottom: '20px',
13    }}
14 >
```

Definición: Este contenedor centraliza y estiliza la sección de la lista de productos, utilizando un fondo con un degradado y asegurando que ocupe un 90% de la altura de la vista.

Título

```
1  <section style={{ width: '100%', position: 'relative' }}>
2    <Typography variant="h4" style={{ marginBottom: '30px', textAlign: 'center', color: '#fff' }}>
3      Lista de Productos
4    </Typography>
```

Definición: Muestra el título "Lista de Productos" en un formato destacado y centrado, con un color de texto blanco.

Botón para Agregar Producto

```
1  /* Botón para abrir el modal de agregar producto */
2  <Button
3    variant="contained"
4    onClick={handleOpenModal}
5    style={{
6      position: 'absolute',
7      top: '20px',
8      right: '20px',
9      backgroundColor: '#6a11cb',
10     color: '#fff',
11   }}
12   >
13     <AddIcon /> Agregar Producto
14   </Button>
```

Definición: Este botón, al hacer clic, abre un modal para agregar un nuevo producto. Está estilizado con un color de fondo que se alinea con el tema general.

Tabla de Productos

```
1 <TableContainer component={Paper}>
2   <Table>
3     <TableHead>
4       <TableRow>
5         <TableCell style={{ color: '#6a11cb' }}>Nombre del Producto</TableCell>
6         <TableCell style={{ color: '#6a11cb' }}>Cantidad</TableCell>
7         <TableCell style={{ color: '#6a11cb' }}>Precio</TableCell>
8         <TableCell style={{ color: '#6a11cb' }}>Estado</TableCell>
9         <TableCell style={{ color: '#6a11cb' }}>Clientes Asociados</TableCell>
10        <TableCell style={{ color: '#6a11cb' }}>Proveedores Asociados </TableCell>
11        <TableCell style={{ color: '#6a11cb' }}>Acciones</TableCell>
12      </TableRow>
13    </TableHead>
14    <TableBody>
15      {productos.map((producto) => (
16        <TableRow key={producto._id}>
17          <TableCell>{producto.nombre_producto}</TableCell>
18          <TableCell>{producto.cantidad}</TableCell>
19          <TableCell>${producto.precio}</TableCell>
20          <TableCell>
21            <span style={{ color: producto.activo ? 'green' : 'red', fontWeight: 'bold' }}>
22              {producto.activo ? 'Activo' : 'Desactivado'}
23            </span>
24          </TableCell>
25
26          <TableCell>
27            {Array.isArray(producto.cliente) && producto.cliente.length > 0 ? (
28              producto.cliente.map((cli) => <div key={cli.id_cliente}>{cli.nombre_cliente}</div>)
29            ) : (
30              'Sin clientes'
31            )}
32          </TableCell>
33
34          <TableCell>
35            {Array.isArray(producto.proveedor) && producto.proveedor.length > 0 ? (
36              producto.proveedor.map((prov) => <div key={prov.id_proveedor}>{prov.nombre_proveedor}</div>)
37            ) : (
38              'Sin proveedores'
39            )}
40          </TableCell>
41
42          <TableCell>
43            <Tooltip title="Editar producto">
44              <Button onClick={() => handleOpenEditModal(producto)}>
45                <EditIcon />
46              </Button>
47            </Tooltip>
48            <Tooltip title={producto.activo ?? false ? 'Desactivar' : 'Activar'}>
49              <Button
50                onClick={() => toggleActivo(producto._id, producto.activo?? false)}
51                style={{
52                  backgroundColor: producto.activo ? 'green' : 'red',
53                  color: '#fff',
54                }}
55              >
56                {producto.activo ? <ToggleOnIcon /> : <ToggleOffIcon />}
57              </Button>
58            </Tooltip>
59            <Tooltip title="Eliminar producto">
60              <Button onClick={() => eliminarProducto(producto._id)}>
61                <DeleteIcon />
62              </Button>
63            </Tooltip>
64
65          </TableCell>
66        </TableRow>
67      ))}
68    </TableBody>
69  </Table>
70</TableContainer>
```

Definición: Esta tabla muestra una lista de productos con columnas para nombre, cantidad, precio, estado, clientes y proveedores. Los datos se generan dinámicamente a partir del arreglo productos.

Modales para Agregar y Editar Producto

Modal para Agregar Producto

```
2  /* Modal para agregar producto */
3  <Modal open={openModal} onClose={handleCloseModal}>
4    <Box sx={{ ...style, backgroundColor: 'white', color: 'black', padding: '20px', borderRadius: '8px' }}>
5      <Typography variant="h6" component="h2" sx={{ marginBottom: '20px', fontWeight: 'bold' }}>
6        Agregar Producto
7      </Typography>
8
9      {/* Botón de salir */}
10     <Button onClick={handleCloseModal} sx={{ position: 'absolute', top: '10px', right: '10px' }}>
11       Salir
12     </Button>
13
14     <form onSubmit={handleSubmit}>
15       {/* Campos de entrada */}
16       <TextField
17         name="nombre_producto"
18         label="Nombre del Producto"
19         value={nuevoProducto.nombre_producto}
20         onChange={handleChange}
21         fullWidth
22         margin="normal"
23         InputLabelProps={{ style: { color: 'black' } }}
24       />
25       <TextField
26         name="cantidad"
27         label="Cantidad"
28         type="number"
29         value={nuevoProducto.cantidad}
30         onChange={handleChange}
31         fullWidth
32         margin="normal"
33         InputLabelProps={{ style: { color: 'black' } }}
34       />
35       <TextField
36         name="precio"
37         label="Precio"
38         type="number"
39         value={nuevoProducto.precio}
40         onChange={handleChange}
41         fullWidth
42         margin="normal"
43         InputLabelProps={{ style: { color: 'black' } }}
44       />
45
46       {/* Selección de clientes */}
47       <FormControl fullWidth margin="normal">
48         <InputLabel style={{ color: 'black' }}>Clientes</InputLabel>
49         <Select
50           multiple
51           value={clientesSeleccionados}
52           onChange={handleClienteChange}
53           renderValue={(selected) =>
54             selected.map((id) => obtenerClientePorId(id)?.nombre_cliente).join(', ')
55           }
56           sx={{ color: 'black' }}
57         >
58           {clientes.map((cliente) => (
59             <MenuItem key={cliente.id_cliente} value={cliente.id_cliente}>
60               {cliente.nombre_cliente}
61             </MenuItem>
62           ))}
63         </Select>
64       </FormControl>
65
66       {/* Selección de proveedores */}
67       <FormControl fullWidth margin="normal">
68         <InputLabel style={{ color: 'black' }}>Proveedores</InputLabel>
69         <Select
70           multiple
71           value={proveedoresSeleccionados}
72           onChange={handleProveedorChange}
73           renderValue={(selected) =>
74             selected.map((id) => obtenerProveedorPorId(id)?.nombre_proveedor).join(', ')
75           }
76           sx={{ color: 'black' }}
77         >
78           {proveedores.map((proveedor) => (
79             <MenuItem key={proveedor.id_proveedor} value={proveedor.id_proveedor}>
80               {proveedor.nombre_proveedor}
81             </MenuItem>
82           )))
83         </Select>
84       </FormControl>
85
86       {/* Botón para guardar */}
87       <Button variant="contained" type="submit" fullWidth sx={{ marginTop: '20px' }}>
88         Guardar Producto
89       </Button>
90     </form>
91   </Box>
92 </Modal>
```

Definición: Este modal permite agregar un nuevo producto, con campos de entrada para el nombre, cantidad y precio, así como selección de clientes y proveedores. Al enviar el formulario, se ejecuta handleSubmit.

Modal para Editar Producto



```
1  /* Modal para editar producto */
2      <Modal open={openEditModal} onClose={handleCloseEditModal}>
3          <Box sx={style}>
4              <Typography variant="h6" component="h2" style={{ marginBottom: '16px', color: 'purple' }}>
5                  Editar Producto
6              </Typography>
7              {productoEditado && (
8                  <form onSubmit={(e) => { e.preventDefault(); actualizarProducto(productoEditado._id, productoEditado); }}>
9                      <TextField
10                         Label="Nombre del Producto"
11                         name="nombre_producto"
12                         value={productoEditado.nombre_producto}
13                         onChange={handleEditChange}
14                         required
15                         fullWidth
16                         style={{ marginBottom: '16px' }}
17                     />
18                     <TextField
19                         Label="Cantidad"
20                         name="cantidad"
21                         type="number"
22                         value={productoEditado.cantidad}
23                         onChange={handleEditChange}
24                         required
25                         fullWidth
26                         style={{ marginBottom: '16px' }}
27                     />
28                     <TextField
29                         Label="Precio"
30                         name="precio"
31                         type="number"
32                         value={productoEditado.precio}
33                         onChange={handleEditChange}
34                         required
35                         fullWidth
36                         style={{ marginBottom: '16px' }}
37                     />
38                     <Button type="submit" variant="contained" color="primary" style={{ marginRight: '8px' }}>
39                         Actualizar
40                     </Button>
41                     <Button variant="outlined" style={{ backgroundColor: 'red', color: 'white' }} onClick={handleCloseEditModal}>
42                         Salir
43                     </Button>
44                 </form>
45             )}
46         </Box>
47     </Modal>
```

Definición: Este modal permite editar un producto existente. Se rellena con los datos del productoEditado y al enviar el formulario, se llama a actualizarProducto.

Snackbar para Mensajes de Error

```
1  /* Snackbar para mensajes de error */
2  <Snackbar open={openSnackbar} autoHideDuration={6000} onClose={handleCloseSnackbar}>
3      <Alert onClose={handleCloseSnackbar} severity="error" sx={{ width: '100%' }}>
4          {errorMessage}
5      </Alert>
6  </Snackbar>
7  </section>
8  </Container>
9 );
10 };
11
12 export default ProductoLista;
```

Definición: Este componente muestra un mensaje de error si ocurre un problema durante las operaciones, como al agregar o actualizar productos. Se oculta automáticamente después de 6 segundos.

5.7 Por ultimo así se ve así nuestro código

```
1 'use client';
2
3 import React, { useEffect, useState } from 'react';
4 import {
5   Container,
6   Typography,
7   Button,
8   Snackbar,
9   Alert,
10  Table,
11  TableBody,
12  TableCell,
13  TableContainer,
14  TableHead,
15  TableRow,
16  Paper,
17  Modal,
18  TextField,
19  Box,
20  MenuItem,
21  Select,
22  FormControl,
23  InputLabel,
24
25 } from '@mui/material';
26 import { styled } from '@mui/system';
27 import { Productos } from '@app/types/Producto.type';
28 import { Proveedores } from '@app/types/Proveedor.type';
29 import { Clientes } from '@app/types/Clientes.type';
30 import AddIcon from '@mui/icons-material/Add';
31 import EditIcon from '@mui/icons-material/Edit';
32 import DeleteIcon from '@mui/icons-material/Delete';
33 import ToggleOffIcon from '@mui/icons-material/ToggleOff';
34 import ToggleOnIcon from '@mui/icons-material/ToggleOn';
35 import Tooltip from '@mui/material/Tooltip';
36 import { SelectChangeEvent } from '@mui/material>Select';
37 import Swal from 'sweetalert2';
38 import 'sweetalert2/src/sweetalert2.scss';
39
40
41 // Estilo del modal
42 const style = {
43   position: 'absolute' as 'absolute',
44   top: '50%',
45   left: '50%',
46   transform: 'translate(-50%, -50%)',
47   width: 400,
48   bgcolor: 'background.paper',
49   border: '2px solid #000',
50   boxShadow: 24,
51   p: 4,
52 };
53
54
55
56 const ProductoLista: React.FC = () => {
57   const [productos, setProductos] = useState<Productos[]>([]);
58   const [nuevoProducto, setNuevoProducto] = useState<Omit<Productos, '_id'>>({
59     nombre_producto: '',
60     cantidad: 0,
61     precio: 0,
62     proveedor: [],
63     cliente: [],
64     activo: true,
65   });
66   const [openSnackbar, setOpenSnackbar] = useState(false);
67   const [errorMessage, setErrorMessage] = useState('');
68   const [openModal, setOpenModal] = useState(false);
69   const [productoEditado, setProductoEditado] = useState<Productos | null>(null);
70   const [openEditModal, setOpenEditModal] = useState(false);
71   const [clientes, setClientes] = useState<Clientes[]>([]);
72   const [clientesSeleccionados, setClientesSeleccionados] = useState<string[]>([]);
73   const [proveedores, setProveedores] = useState<Proveedores[]>([]);
74   const [proveedoresSeleccionados, setProveedoresSeleccionados] = useState<string[]>([]);
75   const [producto, setProducto] = useState<Productos | null>(null);
76
77
78   const obtenerClientes = async () => {
79     const response = await fetch('http://localhost:2000/api/clientes');
80     const data = await response.json();
81     setClientes(data);
82   };
83
84   const obtenerProveedores = async () => {
85     const response = await fetch('http://localhost:2000/api/proveedores');
86     const data = await response.json();
87     setProveedores(data);
88   };
89
90
91   const obtenerProductos = async () => {
92     const response = await fetch('http://localhost:2000/api/productos');
93     const data = await response.json();
94     setProductos(data);
95   };
}
```

```
1  const crearProducto = async (data: Omit<Productos, '_id'>) => {
2    const response = await fetch('http://localhost:2000/api/productos', {
3      method: 'POST',
4      headers: {
5        'Content-Type': 'application/json',
6      },
7      body: JSON.stringify(data),
8    });
9
10   if (response.ok) {
11     setNuevoProducto({ nombre_producto: '', cantidad: 0, precio: 0, proveedor: [], cliente: [], activo: true });
12     setOpenModal(false);
13     await obtenerProductos();
14   } else {
15     const errorData = await response.json();
16     setErrorMessage(errorData.message || 'Error al crear producto');
17     setOpenSnackbar(true);
18   }
19 };
20
21 const eliminarProducto = async (id: string) => {
22   // Mostrar SweetAlert para confirmar eliminación
23   Swal.fire({
24     title: '¿Estás seguro?',
25     text: 'No podrás revertir esto!',
26     icon: 'warning',
27     showCancelButton: true,
28     confirmButtonColor: '#3885d6',
29     cancelButtonColor: '#d33',
30     confirmButtonText: 'Sí, eliminar!',
31     cancelButtonText: 'Cancelar',
32   }).then(async (result) => {
33     if (result.isConfirmed) {
34       // Si el usuario confirma, procedemos con la eliminación
35       const response = await fetch(`http://localhost:2000/api/productos/delete/${id}`, {
36         method: 'DELETE',
37       });
38
39     if (response.ok) {
40       await obtenerProductos();
41       Swal.fire('Eliminado!', 'El producto ha sido eliminado.', 'success');
42     } else {
43       setErrorMessage('Error al eliminar producto');
44       setOpenSnackbar(true);
45     }
46   });
47 });
48 };
49
50 const handleClienteChange = (event: SelectChangeEvent<string[]>) => {
51   const {
52     target: { value },
53   } = event;
54   setClientesSeleccionados(value as string[]); // Asegúrate de que sea un array de strings
55 };
56
57 const handleProveedorChange = (event: SelectChangeEvent<string[]>) => {
58   const {
59     target: { value },
60   } = event;
61   setProveedoresSeleccionados(value as string[]); // Asegúrate de que sea un array de strings
62 };
63
64 useEffect(() => {
65   obtenerProductos();
66 }, []);
67
68 useEffect(() => {
69   if (openModal) {
70     obtenerClientes();
71     obtenerProveedores();
72   }
73 }, [openModal]);
74
75
76 const handleCloseSnackbar = () => {
77   setOpenSnackbar(false);
78   setErrorMessage('');
79 };
80
81 const handleOpenModal = () => {
82   setOpenModal(true);
83 };
84
85 const handleCloseModal = () => {
86   setOpenModal(false);
87   setNuevoProducto({ nombre_producto: '', cantidad: 0, precio: 0, proveedor: [], cliente: [], activo: true });
88   setProveedoresSeleccionados([]);
89   setClientesSeleccionados([]);
90 };
91
92 const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
93   const { name, value } = e.target;
94   setNuevoProducto(prev => ({
95     ...prev,
96     [name]: name === 'nombre_producto' ? value.toUpperCase() : name === 'cantidad' || name === 'precio' ? Number(value) : value,
97   }));
98 };
99
100 const handleOpenEditModal = (producto: Productos) => {
101   setProductoEditado(producto);
102   setOpenEditModal(true);
103 };
104
```

```
1 const handleCloseEditModal = () => {
2     setOpenEditModal(false);
3     setProductoEditado(null);
4 };
5
6 const actualizarProducto = async (id: string, data: Omit<Productos, '_id'>) => {
7     const response = await fetch( http://localhost:2000/api/productos/update/\${id} , {
8         method: 'PUT',
9         headers: {
10             'Content-Type': 'application/json',
11         },
12         body: JSON.stringify(data),
13     });
14
15     if (response.ok) {
16         await obtenerProductos();
17         handleCloseEditModal();
18     } else {
19         const errorMessage = await response.json();
20         setErrorMessage(errorMessage.message || 'Error al actualizar producto');
21         setOpenSnackbar(true);
22     }
23 };
24
25 const handleEditChange = (e: React.ChangeEvent<HTMLInputElement>) => {
26     const { name, value } = e.target;
27     if (productoEditado) {
28         setProductoEditado((prev) => ({
29             ...prev,
30             [name]: name === 'nombre_producto' ? value.toUpperCase() : name === 'cantidad' || name === 'precio' ? Number(value) : value,
31         }));
32     }
33 };
34
35 const handleSubmit = (event: React.FormEvent<HTMLFormElement>) => {
36     event.preventDefault();
37
38     const clientesSeleccionadosDetalles = clientesSeleccionados.map(id => obtenerClientePorId(id));
39     const proveedoresSeleccionadosDetalles = proveedoresSeleccionados.map(id => obtenerProveedorPorId(id));
40
41     const productoData = {
42         ...nuevoProducto,
43         cliente: clientesSeleccionadosDetalles,
44         proveedor: proveedoresSeleccionadosDetalles,
45     };
46
47     crearProducto(productoData);
48 };
49
50 const obtenerClientePorId = (id: string): Clientes => {
51     return clientes.find(cliente => cliente.id_cliente === id) as Clientes;
52 };
53
54 const obtenerProveedorPorId = (id: string): Proveedores => {
55     return proveedores.find(proveedor => proveedor.id_proveedor === id) as Proveedores;
56 };
57
58 const toggleActivo = async (id: string, activo: boolean) => {
59     const response = await fetch(' http://localhost:2000/api/productos/\${activo ? 'deactive' : 'active'}/\${id} ', {
60         method: 'PUT',
61     });
62
63     if (response.ok) {
64         await obtenerProductos();
65     } else {
66         setErrorMessage('Error al cambiar estado del producto');
67         setOpenSnackbar(true);
68     }
69 };
70 }
```

```
1  return (
2    <Container
3      maxWidth="lg"
4      style={{
5        marginTop: '20px',
6        height: '90vh',
7        display: 'flex',
8        justifyContent: 'center',
9        alignItems: 'center',
10       flexDirection: 'column',
11       background: 'linear-gradient(135deg, #6a11cb 0%, #2575fc 100%)',
12       paddingBottom: '20px',
13     }}
14   >
15   <section style={{ width: '100%', position: 'relative' }}>
16     <Typography variant="h4" style={{ marginBottom: '30px', textAlign: 'center', color: '#fff' }}>
17       Lista de Productos
18     </Typography>
19
20     {/* Botón para abrir el modal de agregar producto */}
21     <Button
22       variant="contained"
23       onClick={handleOpenModal}
24       style={{
25         position: 'absolute',
26         top: '20px',
27         right: '20px',
28         backgroundColor: '#6a11cb',
29         color: '#fff',
30       }}
31   >
32     <AddIcon /> Agregar Producto
33   </Button>
34
35   <TableContainer component={Paper}>
36     <Table>
37       <TableHead>
38         <TableRow>
39           <TableCell style={{ color: '#6a11cb' }}>Nombre del Producto</TableCell>
40           <TableCell style={{ color: '#6a11cb' }}>Cantidad</TableCell>
41           <TableCell style={{ color: '#6a11cb' }}>Precio</TableCell>
42           <TableCell style={{ color: '#6a11cb' }}>Estado</TableCell>
43           <TableCell style={{ color: '#6a11cb' }}>Clientes Asociados</TableCell>
44           <TableCell style={{ color: '#6a11cb' }}>Proveedores Asociados </TableCell>
45           <TableCell style={{ color: '#6a11cb' }}>Acciones</TableCell>
46         </TableRow>
47       </TableHead>
48       <TableBody>
49         {productos.map((producto) => (
50           <TableRow key={producto._id}>
51             <TableCell>{producto.nombre_producto}</TableCell>
52             <TableCell>{producto.cantidad}</TableCell>
53             <TableCell>{producto.precio}</TableCell>
54             <TableCell>
55               <span style={{ color: producto.activo ? 'green' : 'red', fontWeight: 'bold' }}>
56                 {producto.activo ? 'Activo' : 'Desactivado'}
57               </span>
58             </TableCell>
59
60             <TableCell>
61               {Array.isArray(producto.cliente) && producto.cliente.length > 0 ? (
62                 producto.cliente.map((cli) => <div key={cli.id_cliente}>{cli.nombre_cliente}</div>
63               ) : (
64                 'Sin clientes'
65               )}
66             </TableCell>
67
68             <TableCell>
69               {Array.isArray(producto.proveedor) && producto.proveedor.length > 0 ? (
70                 producto.proveedor.map((prov) => <div key={prov.id_proveedor}>{prov.nombre_proveedor}</div>
71               ) : (
72                 'Sin proveedores'
73               )}
74             </TableCell>
75
76             <TableCell>
77               <Tooltip title="Editar producto">
78                 <Button onClick={() => handleOpenEditModal(producto)}>
79                   <EditIcon />
80                 </Button>
81               </Tooltip>
82               <Tooltip title={producto.activo ?? false ? 'Desactivar' : 'Activar'}>
83                 <Button
84                   onClick={() => toggleActivo(producto._id, producto.activo?? false)}
85                   style={{
86                     backgroundColor: producto.activo ? 'green' : 'red',
87                     color: '#fff',
88                   }}
89                 >
90                   {producto.activo ? <ToggleOnIcon /> : <ToggleOffIcon />}
91                 </Button>
92               </Tooltip>
93               <Tooltip title="Eliminar producto">
94                 <Button onClick={() => eliminarProducto(producto._id)}>
95                   <DeleteIcon />
96                 </Button>
97               </Tooltip>
98
99             </TableCell>
100           </TableRow>
101         ))}
102       </TableBody>
103     </Table>
104   </TableContainer>
```

```

1  /* Modal para agregar producto */
2  <Modal open={openModal} onClose={handleCloseModal}>
3      <Box sx={{ ...style, backgroundColor: 'white', color: 'black', padding: '20px', borderRadius: '8px' }}>
4          <Typography variant="h6" component="h2" sx={{ marginBottom: '20px', fontWeight: 'bold' }}>
5              Agregar Producto
6          </Typography>
7
8          /* Botón de salir */
9          <Button onClick={handleCloseModal} sx={{ position: 'absolute', top: '10px', right: '10px' }}>
10             Salir
11         </Button>
12
13         <form onSubmit={handleSubmit}>
14             {/* Campos de entrada */}
15             <TextField
16                 name="nombre_producto"
17                 label="Nombre del Producto"
18                 value={nuevoProducto.nombre_producto}
19                 onChange={handleChange}
20                 fullWidth
21                 margin="normal"
22                 InputLabelProps={{ style: { color: 'black' } }}
23             />
24             <TextField
25                 name="cantidad"
26                 label="Cantidad"
27                 type="number"
28                 value={nuevoProducto.cantidad}
29                 onChange={handleChange}
30                 fullWidth
31                 margin="normal"
32                 InputLabelProps={{ style: { color: 'black' } }}
33             />
34             <TextField
35                 name="precio"
36                 label="Precio"
37                 type="number"
38                 value={nuevoProducto.precio}
39                 onChange={handleChange}
40                 fullWidth
41                 margin="normal"
42                 InputLabelProps={{ style: { color: 'black' } }}
43             />
44
45             /* Selección de clientes */
46             <FormControl fullWidth margin="normal">
47                 <InputLabel style={{ color: 'black' }}>Clientes</InputLabel>
48                 <Select
49                     multiple
50                     value={clientesSeleccionados}
51                     onChange={handleClienteChange}
52                     renderValue={(selected) =>
53                         selected.map((id) => obtenerClientePorId(id)?.nombre_cliente).join(', ')
54                     }
55                     sx={{ color: 'black' }}
56                 >
57                     {clientes.map((cliente) => (
58                         <MenuItem key={cliente.id_cliente} value={cliente.id_cliente}>
59                             {cliente.nombre_cliente}
60                         </MenuItem>
61                     )))
62                 </Select>
63             </FormControl>
64
65             /* Selección de proveedores */
66             <FormControl fullWidth margin="normal">
67                 <InputLabel style={{ color: 'black' }}>Proveedores</InputLabel>
68                 <Select
69                     multiple
70                     value={proveedoresSeleccionados}
71                     onChange={handleProveedorChange}
72                     renderValue={(selected) =>
73                         selected.map((id) => obtenerProveedorPorId(id)?.nombre_proveedor).join(', ')
74                     }
75                     sx={{ color: 'black' }}
76                 >
77                     {proveedores.map((proveedor) => (
78                         <MenuItem key={proveedor.id_proveedor} value={proveedor.id_proveedor}>
79                             {proveedor.nombre_proveedor}
80                         </MenuItem>
81                     )))
82                 </Select>
83             </FormControl>
84
85             /* Botón para guardar */
86             <Button variant="contained" type="submit" fullWidth sx={{ marginTop: '20px' }}>
87                 Guardar Producto
88             </Button>
89         </form>
90     </Box>
91 </Modal>

```

```
1  /* Modal para editar producto */
2  <Modal open={openEditModal} onClose={handleCloseEditModal}>
3    <Box sx={style}>
4      <Typography variant="h6" component="h2" style={{ marginBottom: '16px', color: 'purple' }}>
5        Editar Producto
6      </Typography>
7      {productoEditado && (
8        <form onSubmit={(e) => { e.preventDefault(); actualizarProducto(productoEditado._id, productoEditado); }}>
9          <TextField
10            label="Nombre del Producto"
11            name="nombre_producto"
12            value={productoEditado.nombre_producto}
13            onChange={handleEditChange}
14            required
15            fullWidth
16            style={{ marginBottom: '16px' }}}
17          />
18          <TextField
19            label="Cantidad"
20            name="cantidad"
21            type="number"
22            value={productoEditado.cantidad}
23            onChange={handleEditChange}
24            required
25            fullWidth
26            style={{ marginBottom: '16px' }}}
27          />
28          <TextField
29            label="Precio"
30            name="precio"
31            type="number"
32            value={productoEditado.precio}
33            onChange={handleEditChange}
34            required
35            fullWidth
36            style={{ marginBottom: '16px' }}}
37          />
38          <Button type="submit" variant="contained" color="primary" style={{ marginRight: '8px' }}>
39            Actualizar
40          </Button>
41          <Button variant="outlined" style={{ backgroundColor: 'red', color: 'white' }} onClick={handleCloseEditModal}>
42            Salir
43          </Button>
44        </form>
45      )}
46    </Box>
47  </Modal>
48
49  /* Snackbar para mensajes de error */
50  <Snackbar open={openSnackbar} autoHideDuration={6000} onClose={handleCloseSnackbar}>
51    <Alert onClose={handleCloseSnackbar} severity="error" sx={{ width: '100%' }}>
52      {errorMessage}
53    </Alert>
54  </Snackbar>
55  </section>
56  </Container>
57 );
58 };
59
60 export default ProductoLista;
```

5.8 Así se ve la interfaz final.

The screenshot shows a mobile application interface titled "Lista de Productos". At the top right is a purple button labeled "+ AGREGAR PRODUCTO". Below it is a table with columns: Nombre del Producto, Cantidad, Precio, Estado, Clientes Asociados, Proveedores Asociados, and Acciones. The table contains six rows of data. Each row includes edit, activate/deactivate, and delete icons in a blue box. The footer of the screen displays the copyright notice "© 2024 SENA. All rights reserved."

Nombre del Producto	Cantidad	Precio	Estado	Clientes Asociados	Proveedores Asociados	Acciones
CRMM	52	\$52	Desactivado	Sin clientes	Sin proveedores	
MAR	565	\$6565	Activo	Sin clientes	Sin proveedores	
YOUNGURT	5656	\$565	Activo	Sin clientes	Sin proveedores	
DONA	52	\$52	Activo	Sin clientes	Sin proveedores	
HUEVOS	63	\$5000	Activo	Camila	Sin proveedores	
HAHSAHAH	5665	\$56566	Activo	Camila	D1	

The screenshot shows a mobile application interface titled "Lista de Productos". A modal window titled "Agregar Producto" is open in the center. It contains fields for Nombre del Producto, Cantidad (with value 0), Precio (with value 0), Clientes (a dropdown menu), and Proveedores (a dropdown menu). At the bottom of the modal is a blue button labeled "GUARDAR PRODUCTO". In the top right corner of the modal is a "SALIR" button. The background shows the same product list as the previous screenshot. The footer of the screen displays the copyright notice "© 2024 SENA. All rights reserved."