Nombre de la práctica	App Full Stack productos			No.	1
Asignatura:	Programación web	Carrera:	INGENIERÍA EN SISTEMAS COMPUTACIONALES-3602	Duración de la práctica (Hrs)	2 horas

NOMBRE DEL ALUMNO: Francisco David Colin Lira
GRUPO: 3602

I. Competencia(s) específica(s):

Desarrolla aplicaciones web dinámicas del lado cliente y del servidor, considerando la conectividad a orígenes de datos, la interconectividad entre aplicaciones y cómputo en la nube.

II. Lugar de realización de la práctica (laboratorio, taller, aula u otro):

- **Aula**
- **Casa**

III. Material empleado:

- Equipo de computo

IV. Desarrollo de la práctica:

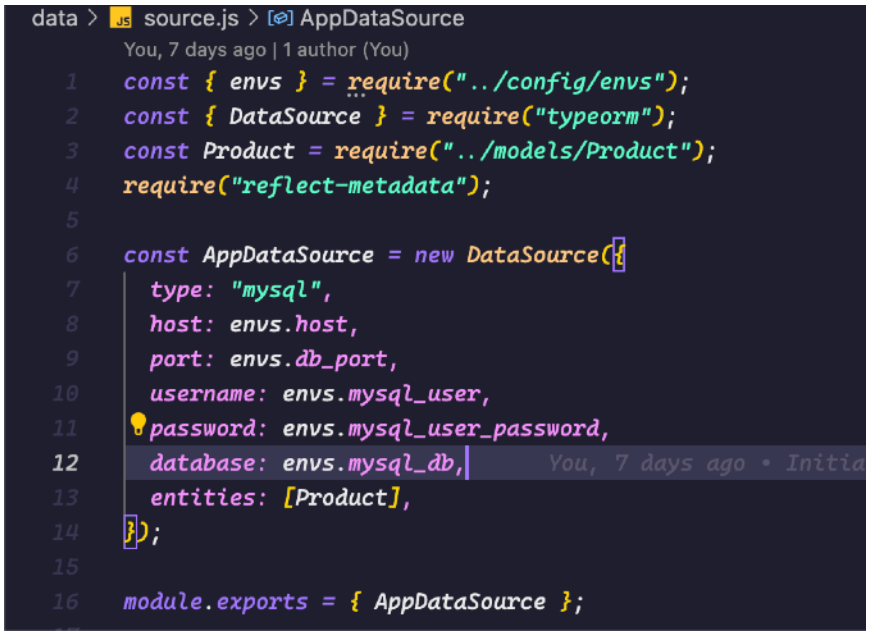
En nuestra aplicacion es cliente servidor, asi que empecemos por el servidor que me parece buen punto de partida, en nuestro servidor esta echo con express en node js, usamos typeORM con js simple, sin typescript para poder manipular de una manera mas sencilla la base datos, se creo una API REST que se expone en nuestro puerto 3000 con el endpoint de products, donde se realizan las operaciones de un CRUD (Crear, Leer, Actualizar, Eliminar).

1.- definimos nuestro modelo de productos, este es como se vería la tabla en nuestra base de datos:

models > Product.js > ...
You, 7 days ago | 1 author (You)

```
const { EntitySchema } = require("typeorm");

module.exports = new EntitySchema({
  name: "Product",
  tableName: "productos",
  columns: {
    id: {
      primary: true,
      type: "int",
      generated: true,
    },
    nombre: {
      type: "varchar",
      length: 255,
      nullable: false,
    },
    precio: {
      type: "decimal",
      precision: 10,
      scale: 2,
    },
    stock: {
      type: "int",
    },
  },
});
```

2.- Definimos nuestro dataSource, este es útil para nuestro repositorio, contiene información de la base de datos, la conexión, y que modelos o entidades existen.



3.- vamos a nuestro repositorio y colocamos nuestro dataSource que definimos y le colocamos que modelo usara este repositorio.

4.- Ahora tenemos que definir nuestro service, que es el en encargado de interactuar con nuestro repositorio y también aquí se debería incluir la lógica del negocio o de lo que contemple, en este caso productos, aquí solo estamos extrayendo información de la base de datos pero desde nuestro repositorio.

services > product.service.js > ProductService > getAll

You, 7 days ago | 1 author (You)

```
const { ProductRepository } = require("../respositories/product.repository");

You, 7 days ago | 1 author (You)
class ProductService {
  async getAll() {      You, 7 days ago • Initialize repository with application f
    return await ProductRepository.find();
  }

  async getOneById(id) {
    return await ProductRepository.findOneBy({ id });
  }

  async create(product) {
    const newProduct = ProductRepository.create(product);
    return await ProductRepository.save(newProduct);
  }

  async update(id, data) {
    await ProductRepository.update(id, data);
    return await ProductRepository.findOneBy({ id });
  }

  async delete(id) {
    return await ProductRepository.delete({ id });
  }
}

module.exports = { ProductService: new ProductService() };

```

5.- Ahora definimos nuestro controller que este indica las acciones que existen a el recurso, en este caso productos, solo debería tener y poder controlar las peticiones y respuestas, no debería contener lógica del negocio, aunque en este caso esta un poco mezclado con algunas cosas del service.

```
You, 7 days ago | 1 author (You)
const { ProductService } = require("../services/product.service");      You, 7 days
const logger = require("../utils/logger");
const { createProductDto, updateProductDto } = require("../utils/validateData");

You, 7 days ago | 1 author (You)
class ProductController {
  static async getAll(req, res) {
    try {
      const products = await ProductService.getAll();
      return res.status(200).json(products);
    } catch (error) {
      logger.error("Error al leer los productos", error?.message);
      return res.status(500).json({
        message:
          "Ocurrio un error al leer los datos" ||
          "Internal server error: " + error?.message,
      });
    }
  }

  static async getOneById(req, res) {
    try {
      const { id } = req.params;
      if (!id || isNaN(id)) {
        return res.status(400).json({
          message: "Valid Id is required, expected number",
        });
      }
      const product = await ProductService.getOneById(id);
      if (!product) {
        return res.status(404).json({
          message: "Product not found",
        });
      }
      return res.status(302).json(product); // 302: Found
      // return res.status(200).json(product); // 200: OK
    } catch (error) {
      logger.error("Error al leer el producto", error?.message);
      return res.status(500).json({
        message:
          "Ocurrio un error al leer los datos" ||
          "Internal server error: " + error?.message,
      });
    }
  }
}
```

```
  static async create(req, res) {
    try {
      const { nombre, precio, stock } = req.body;
      const data = { nombre, precio, stock };

      if (!createProductDto(data)) {
        return res.status(400).json({
          message:
            "Data not provided correctly. Please provide valid 'nombre', 'precio' and 'stock'",
        });
      }

      const product = await ProductService.create(data);
      if (!product) {
        return res.status(404).json({
          message: "Product not found",
        });
      }
      return res.status(201).send();
    } catch (error) {
      logger.error("Error al crear el producto", error?.message);
      return res.status(500).json({
        message:
          "Ocurrio un error al leer los datos" ||
          "Internal server error: " + error?.message,
      });
    }
  }
```

```
  static async update(req, res) {
    try {
      const { id } = req.params;
      const { nombre, precio, stock } = req.body;
      const data = {};

      nombre ? (data.nombre = nombre) : null;
      precio ? (data.precio = precio) : null;
      stock ? (data.stock = stock) : null;

      logger.info("Datos recibidos", data);

      if (!id || isNaN(id)) {
        return res.status(400).json({
          message: "Valid Id is required, expected number",
        });
      }
      if (!updateProductDto(data)) {
        return res.status(400).json({
          message:
            "Data not provided correctly. Please provide valid 'nombre', 'precio' and 'stock'",
        });
      }
      const product = await ProductService.update(id, data);
      if (!product) {
        return res.status(404).json({
          message: "Product not found",
        });
      }
      logger.info("Producto actualizado correctamente", product);
      return res.status(201).json(product);
    } catch (error) {
      logger.error("Error al actualizar el producto", error?.message);
      return res.status(500).json({
        message:
          "Ocurrio un error al leer los datos" ||
          "Internal server error: " + error?.message,
      });
```

```
  static async delete(req, res) {
    try {
      const { id } = req.params;
      if (!id || isNaN(id)) {
        return res.status(400).json({
          message: "Valid Id is required, expected number",
        });
      }
      const product = await ProductService.delete(id);
      if (!product) {
        return res.status(404).json({
          message: "Product not found",
        });
      }
      return res.status(204).send();
    } catch (error) {
      logger.error("Error al eliminar el producto", error?.message);
      return res.status(500).json({
        message:
          "Ocurrio un error al leer los datos" ||
          "Internal server error: " + error?.message,
      });
    }
  }
}

module.exports = { ProductController };
```

6.- Ahora tenemos que definir las rutas para que podamos después exponer nuestro controller y que maneje las acciones de nuestro recurso, en este caso yo decidí mapear en un array de objetos cada ruta, método y acción que se puede hacer y luego iterar en este array y crear las rutas de manera mas sencilla.

routes > products.routes.js > ...

You, 7 days ago | 1 author (You)

```
1  const { ProductController } = require("../controllers/products.controller");
2  const express = require("express");
3  const router = express.Router();
4  const logger = require("../utils/logger");
5
6  const routes = {
7    products: [
8      { method: "get", path: "/", action: ProductController.getAll },
9      { method: "get", path: "/:id", action: ProductController.getOneById },
10     { method: "post", path: "/", action: ProductController.create },
11     { method: "patch", path: "/:id", action: ProductController.update },
12     { method: "delete", path: "/:id", action: ProductController.delete },
13   ],
14 };
15
16 routes.products.forEach((route) => {
17   const { method, path, action } = route;
18   router[method](path, action);
19   // Add log to mapped routes of products
20   logger.info(`Mapped route: ${method.toUpperCase()} products${path}`);
21 });
22
23 module.exports = router;
```

7.- Ahora solo tenemos que crear nuestro punto de inicio de nuestra aplicación o API REST y mandar a llamar a nuestras rutas con nuestro recurso que queremos exponer, ademas de iniciar la conexión con la base de datos.

index.js > ...

```
const cors = require("cors");
const { envs } = require("./config/envs");
const { initDB } = require("./config/initDB");
// const saludarRoutes = require("./greet/routes/greet.routes");
// const despedirRoutes = require("./despedir/routes/despedir.routes");
const productRoutes = require("./routes/products.routes");
const { AppDataSource } = require("./data/source");
const logger = require("./utils/logger");

const app = express();

// Middlewares
app.use(express.json());
app.use(cors());

async function start() {
  try {
    await AppDataSource.initialize();
    await initDB();
    logger.info("Conexión a la base de datos exitosa");
  } catch (error) {
    logger.error("Error al conectar a la base de datos ", error);
  }
}
// Rutas
// app.use("/saludar", saludarRoutes);
// app.use("/despedir", despedirRoutes);
app.use("/api/v1/products", productRoutes);      You, 7 days ago • Initialize rep

app.listen(envs.port, async () => {
  // console.log({ envs });
  await start();
  logger.success(`Server running on port ${envs.port}`);
});
```

Ahora que ya esta nuestra API rest vamos con nuestro frontend, en este se realizó una app responsiva, eso quiere decir que se puede usar tanto en movil como una computadora o tableta, se realizó el diseño con html y tailwindcss, se cargaron los datos de manera dinámica gracias a js, haciendo llamadas a nuestra API REST y manejando acciones del DOM.

1.- Se creo nuestra app en html con un formulario para insertar datos:

```
You, 6 days ago | 1 author (You)
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Proyecto 1</title>
    <link rel="icon" href="public/bimbo-seeklogo.svg">
    <!— <script src="https://unpkg.com/@tailwindcss/browser@4"></script> —>
    <link rel="stylesheet" href="output.css">
</head>
<body ... >
    <header ... >
        <img src="public/bimbo-seeklogo.svg" alt="Bimbo" ... >
        <!— <h1 ... >Bimbo 🧸</h1> —>
    </header>

    <main ... >
        <form action="" ... id="form">
            <h2 ... >Productos</h2>
            <div ... >
                <label for="nombre">Nombre</label>
                <input type="text" name="" id="nombre" placeholder="Donitas ... " ... >
            </div>
            <div ... >
                <label for="precio">Precio</label>
                <input type="text"  min="0" id="precio" placeholder="23.80" ... >
            </div>
            <div ... >
                <label for="stock">Stock</label>
                <input type="number" min="1" id="stock" placeholder="5" ... >
            </div>
            <button type="submit" ... id="btnSubmit" name="add">Agregar Producto</button>
            <button type="reset" ... id="btnClear">Limpiar campos</button>
        </form>
```

2.- se creo un lista des-ordenada en la cual previamente se están creándooslos elementos o cards que contienen información de lo que tenemos en la base de datos.

```
        <section ≈ ... >
            <h2 ≈ ... >Lista de Productos</h2>
            <ul ≈ ...  id="contenedor-cards">
                <li ≈ ... >⋯
                </li>
            </ul>      You, 7 days ago • Initialize repository with application fullstac…
        </section>

    </main>
    <script src="./index.js"></script>
</body>
</html>
```

3.- Ahora vayamos al codigo js, donde estamos haciendo las peticiones a la API REST y las demas acciones, lo primero que hacemos es escuchar el documento y cuando este cargado todo completamente ya podémoste recuperar algunos elementos como los son el formulario, botones y el contenedor de las cards donde previamente vamos a insertar elementos.

```
document.addEventListener("DOMContentLoaded", () => {
  const contenedorCards = document.getElementById("contenedor-cards");
  const formData = document.getElementById("form");
  const btnSubmit = document.getElementById("btnSubmit");
  const btnCancelOrClean = document.getElementById("btnClear");

  let productoEditandoId = null;

  const HOST = "http://localhost:3000/api/v1";
```

4.- Ahora tenemos un evento de escucha al formulario cuando se haga submit, aquí extraemos los datos de los campos, después revisamos que accionarial tiene el formulario y realizamos la acción.

```
formData.addEventListener("submit", async (e) => {
  e.preventDefault();

  const nombre = document.getElementById("nombre").value;
  const precio = parseFloat(document.getElementById("precio").value);
  const stock = parseInt(document.getElementById("stock").value);
  const data = { nombre, precio, stock };

  if (formData.getAttribute("data-mode") === "edit") {
    await editEvent(data);
    return;
  }
  await addEvent(data);
  return;
});
```

5.- Ahora tenemos la función que se llama obtener productosAPI, la cual realiza una petición jet a nuestra API y extrae todos los datos actuales y los inserta en nuestro contenedor-cards

```
const obtenerProductosAPI = async () => {
  const response = await fetch(`${HOST}/products`);
  if (!response.ok) return alert("Ocurrió un error al obtener productos");

  const products = await response.json();
  contenedorCards.innerHTML = "";

  products.forEach((product) => {
    createCardProduct(product);
  });
};
```

6.- Ahora tenemos la función de limpiar o cancelar los datos, este solo realiza un reset al formulario coloca atributos para las acciones.

```
btnCancelOrClean.addEventListener("click", () => {
  const attributeMode = btnCancelOrClean.getAttribute("data-mode");
  if (attributeMode === "edit") {
    resetFormulario();
    return;
  }
});
```

7.- Ahora tenemos la función de crearProduct, el cual crea el producto y lo inserta en el nuestro contenedor, con la información del producto.

```
window.createCardProduct = (product) => {
  contenedorCards.innerHTML += `
  <li id="${product.id}" ...>
    <div ...>
        <h3 ...>${product.nombre}</h3>
        <div ...>
            <span ...>Stock: </span>
            <p ...> ${product.stock}</p>
        </div>
    </div>

    <div ...>
        <div ...>
            <p> $${product.precio}</p>
        </div>

        <div ...>
            <button onclick="editarProducto(${product.id}, '${product.nombre}', ${product.precio}, ${product.stock})" ...>
              <img src="./public/icon-edit.svg" ...></img>
              Editar
            </button>
            <button onclick="eliminar(${product.id})" ...>
              <img src="./public/icon-delete.svg" ...></img>
              Eliminar
            </button>
        </div>
    </div>
  </li>`;
};
```

8.- tenemos la acción de eliminar, limpia el elemento del contenedor y la función que lo elimina de la base de datos.

```
window.editarProducto = (id, nombre, precio, stock) => {
  document.getElementById("nombre").value = nombre;
  document.getElementById("precio").value = precio;
  document.getElementById("stock").value = stock;

  productoEditandoId = id;
  formData.setAttribute("data-mode", "edit");
  btnSubmit.textContent = "Actualizar Producto";
  btnSubmit.classList.remove("bg-teal-300", "hover:bg-teal-400");
  btnSubmit.classList.add("bg-amber-300", "hover:bg-amber-400");
  btnCancelOrClean.setAttribute("data-mode", "edit");
  btnCancelOrClean.textContent = "Cancelar";
};

window.eliminar = async (id) => {
  if (!confirm("¿Estás seguro de eliminar este producto?")) return;

  const response = await fetch(`${HOST}/products/${id}`, {
    method: "DELETE",
  });

  if (response.ok) {
    alert("Producto eliminado con éxito");
    document.getElementById(id).remove();
  } else {
    alert("No se pudo eliminar el producto");
  }
};
```

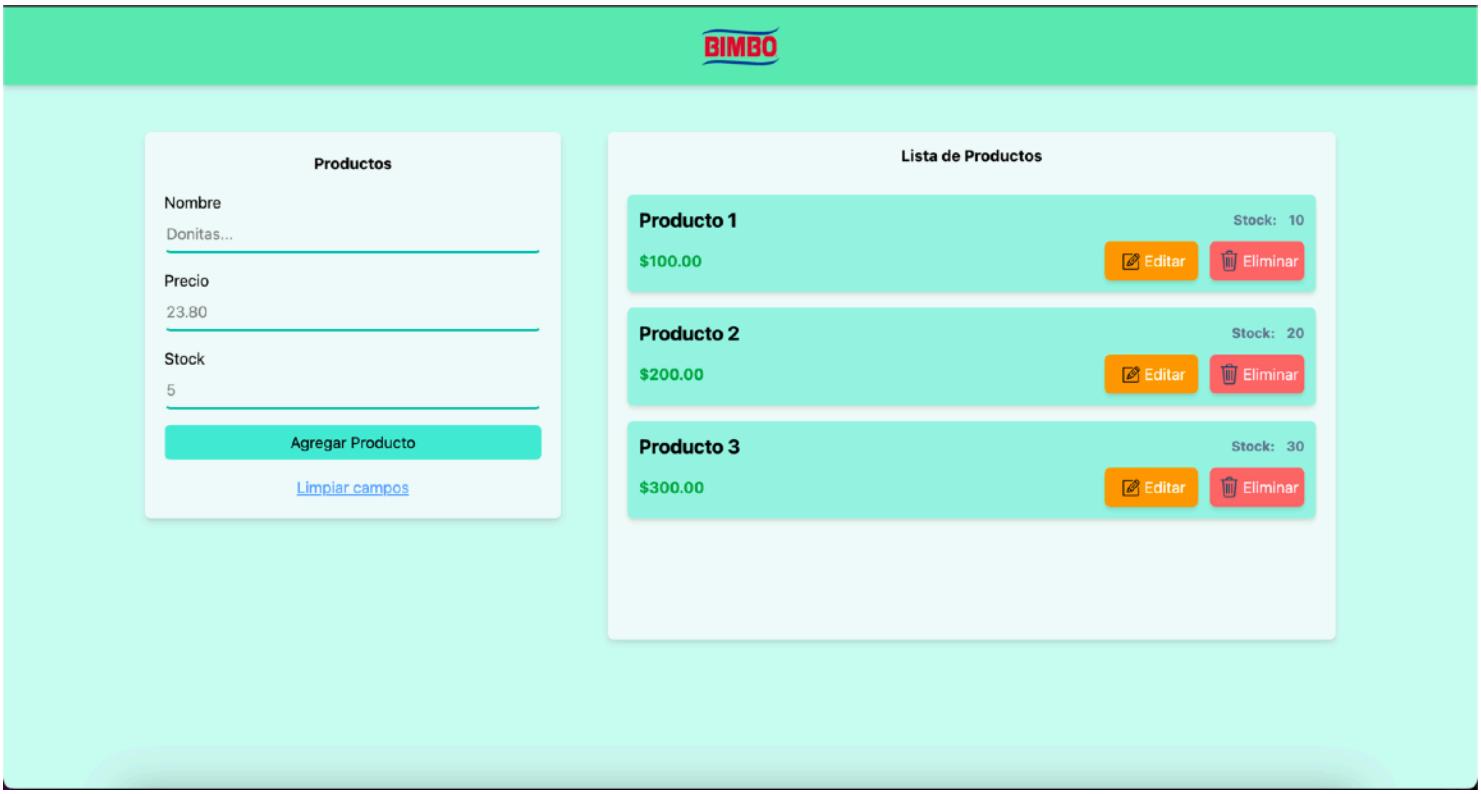
9.- las funciones de editar el producto, donde se cambian estilos en el formulario y de como se hace la petición a nuestra API.

```
window.editEvent = async (data) => {
  const response = await fetch(`${HOST}/products/${productoEditandoId}`, {
    method: "PATCH",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(data),
  });

  if (response.ok) {
    alert("Producto actualizado correctamente");
    resetFormulario();
    obtenerProductosAPI();
    return;
  } else {
    alert("Error al actualizar el producto");
    return;
  }
};

window.addEvent = async (data) => {
  const response = await fetch(`${HOST}/products`, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(data),
  });

  if (response.ok) {      You, 7 days ago • Refactor product form handling and update butto…
    alert("Producto agregado correctamente");
    resetFormulario();
    obtenerProductosAPI();
    return;
  } else {
    alert("Error al agregar el producto");
    return;
  }
};
```

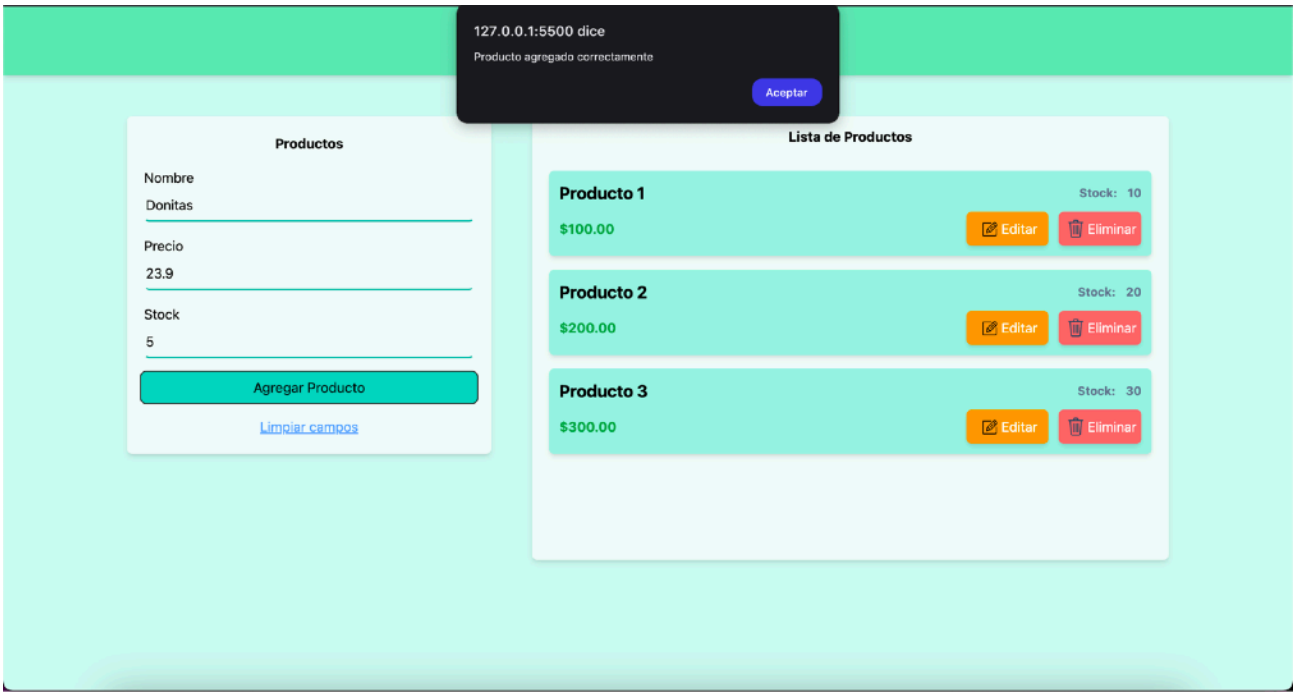
10.- Por ultimo nuestras funciones de resetFormulario que limpia los campos y regresa los estilos, ademas que cuando carga el DOM, ejecutamos la función de obtenerProductosAPI para que se muestren en el documento.

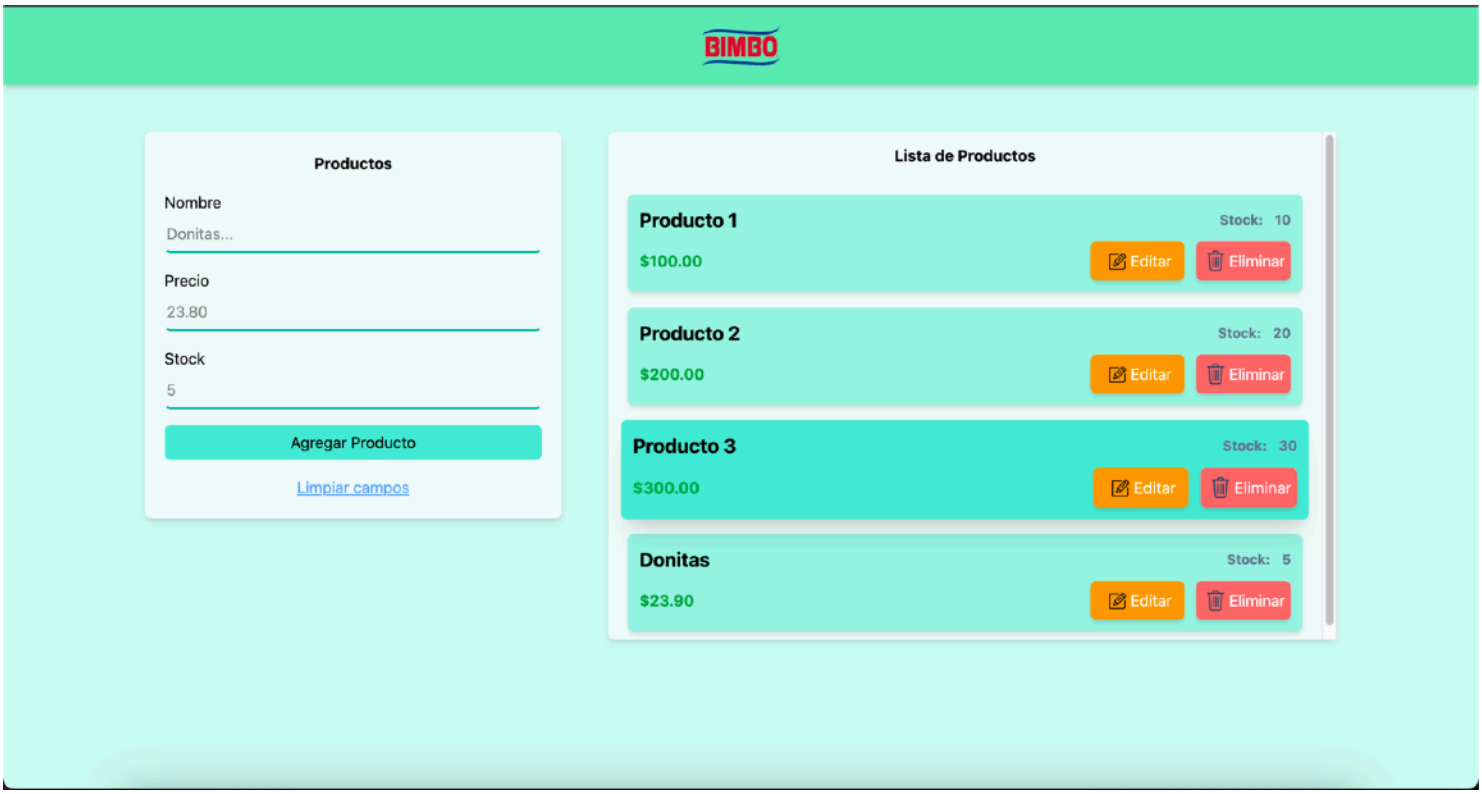
```
  const resetFormulario = () => {
    formData.reset();

    formData.setAttribute("data-mode", "add");
    btnCancelOrClean.setAttribute("data-mode", "clean");

    productoEditandoId = null;
    btnCancelOrClean.textContent = "Limpiar campos";
    btnSubmit.textContent = "Agregar Producto";
    btnSubmit.classList.remove("bg-amber-300", "hover:bg-amber-400");
    btnSubmit.classList.add("bg-teal-300", "hover:bg-teal-400");

    document.activeElement.blur();
    document.body.focus();
  };

  obtenerProductosAPI();
});
```

Y Aquí tenemos el como funciona ya nuestra aplicación.
1.- Cuando carga la aplicación.

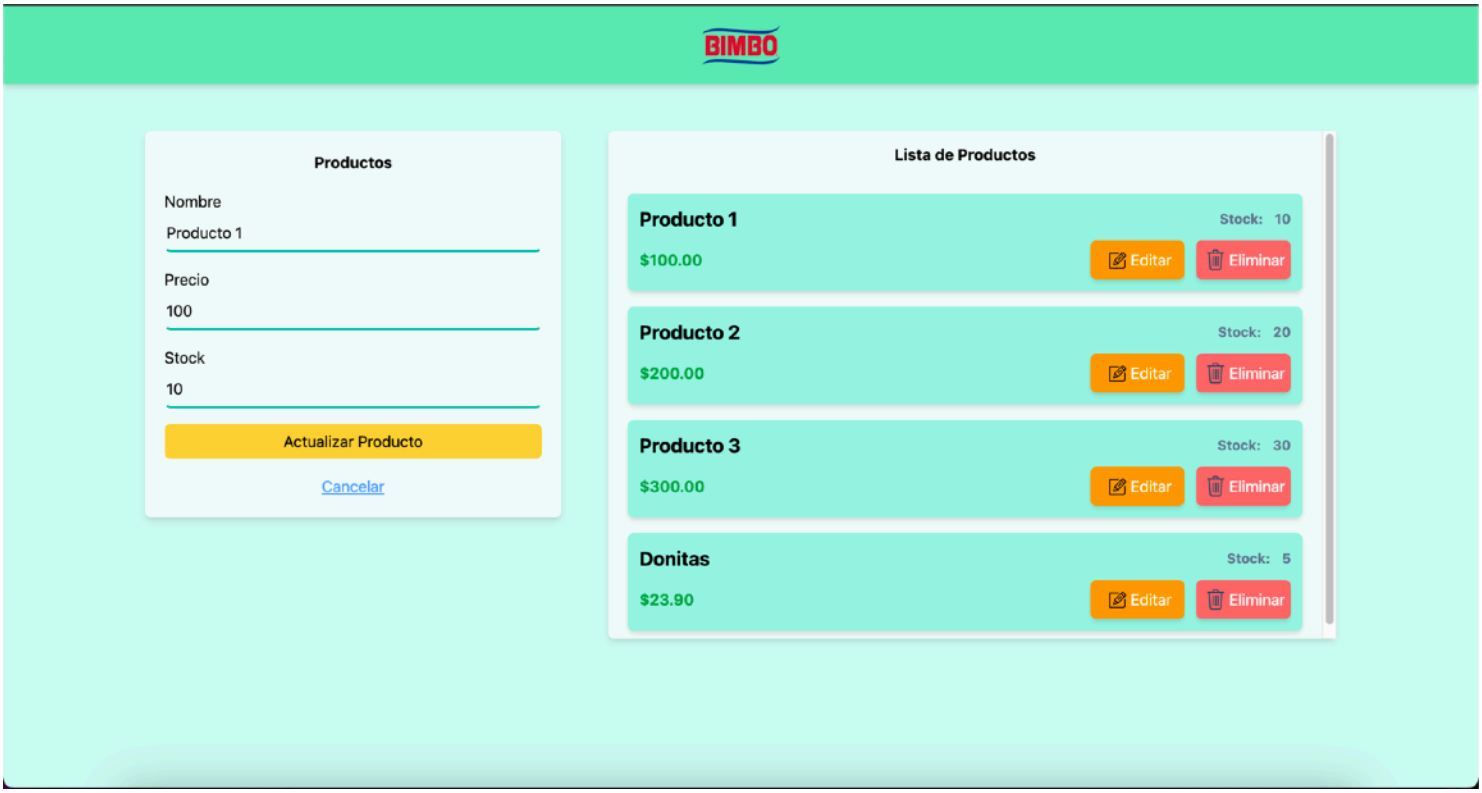


2.- Cuando agregamos un producto.

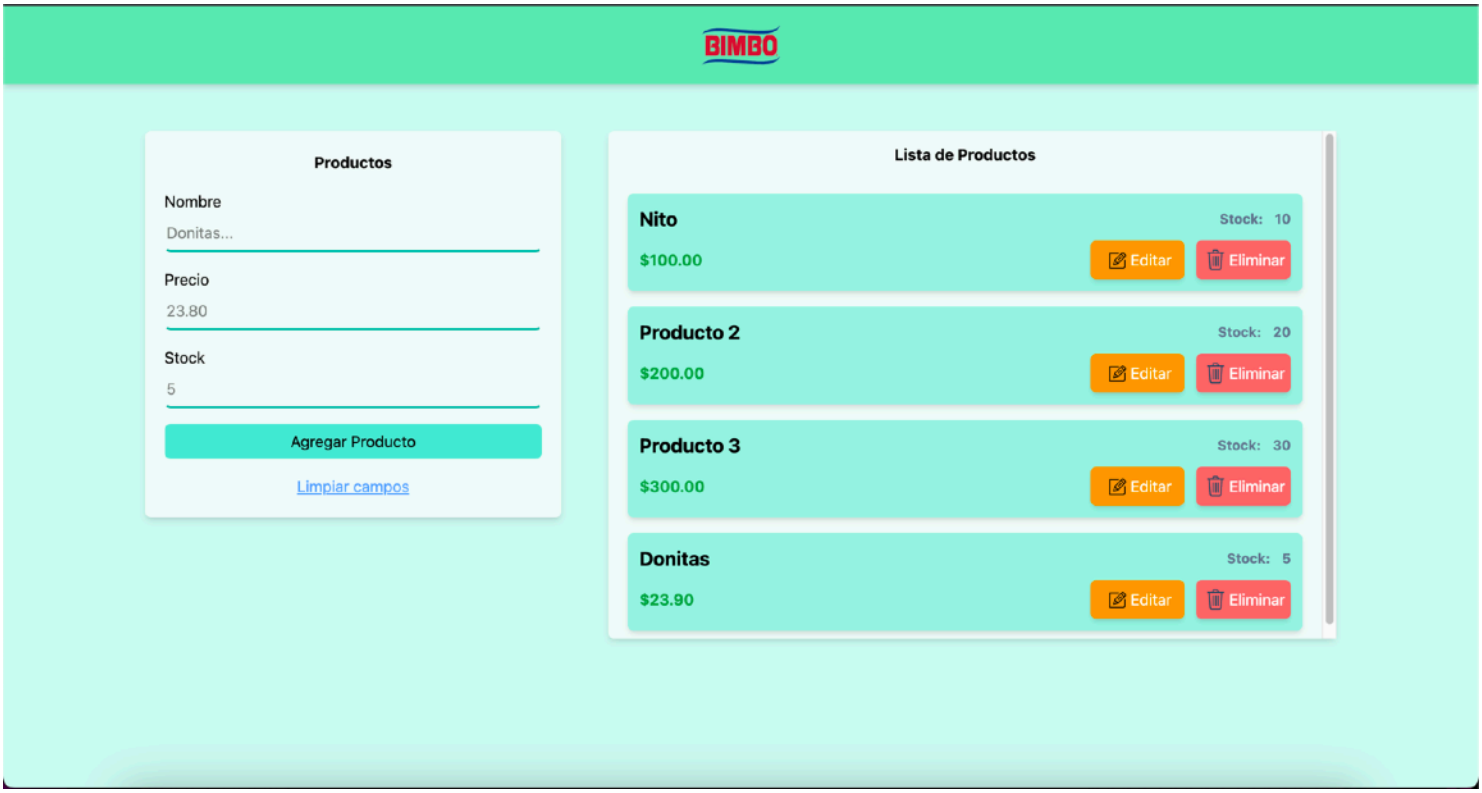




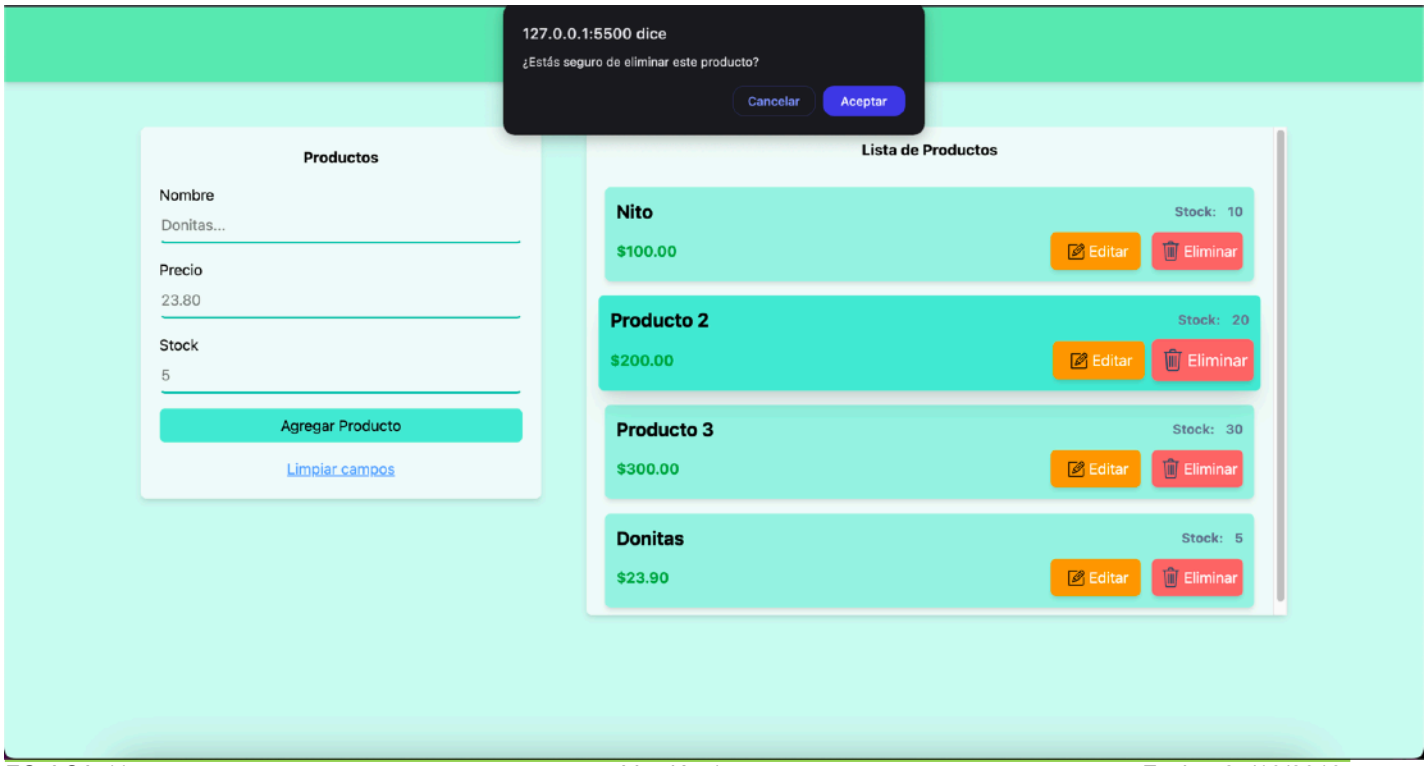
3.- Cuando editamos un producto:



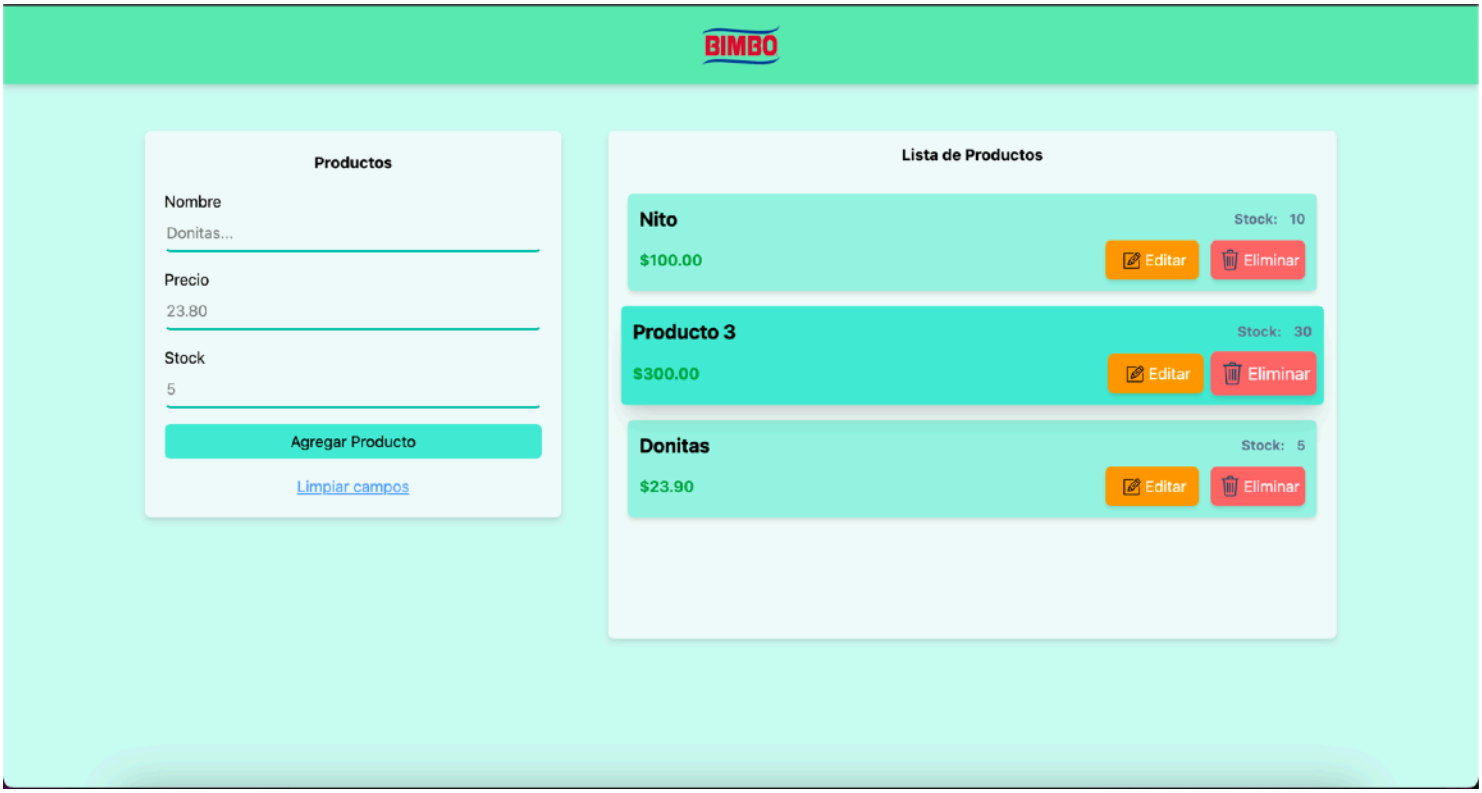
Aqui el producto ya actualizado



4.- Eliminando un producto



Aqui ya eliminado el producto.



V. Conclusiones:

En esta práctica aprendi mas de lo que imagine, en parte como se usa typeORM con js simple sin typescript aunque no me gusto que no se crea la tabla, eso le veo la única desventaja de esta forma con js, ya que con typescript si crea la tabla cuando la definimos en los modelos o entidades, pero de igual forma me pareció bastante buena esta practica, ya que tienes que tener buenos conceptos de esto si no te puede llegar a ser algo confuso, y del diseño pues me gusto como se ve, se mira bastante bien todo la verdad, me gusto mucho esta practica.