

# UNIVERSIDAD AUTONOMA DE SINALOA

## FACULTAD DE INFORMATICA DE CULIACÁN

### LICENCIATURA EN INFORMÁTICA DE CULIACÁN.

**ASIGNATURA: DESARROLLO WEB DEL LADO DEL SERVIDOR.**

**GRUPO: 2-1.**

**DOCENTE: JOSÉ MANUEL CAZAREZ ALDERETE.**

**NOMBRE: PEREIDA VERDUGO JESÚS ELEAN.**

**FECHA: 29/ MAYO/ 2025.**



#### Actividad final: CRUD completo



José Manuel Cazarez Alderete • 22 may (Última modificación: 22 may)

100 puntos

Fecha de entrega: Mañana, 23:59

Crear una aplicación web (front y back) que implemente un CRUD completo, es decir, que tenga las operaciones de crear, leer, actualizar y eliminar.

Consideraciones:

- Deberá utilizar una base de datos (ejemplo: sqlite).
- El código deberá estar en un repositorio público de github.
- Anexar pantallas (screenshots) explicando la funcionalidad del código. Por ejemplo: las rutas y los componentes visuales.
- Anexar pantallas (screenshots) mostrando como quedó la aplicación terminada.

**FECHA DE ENTREGA: viernes 30 de mayo.**

NOTA: las tecnologías a utilizar serán react (front) y node express (back).

## Creación de la Base De Datos en SQLite.

```
CREATE TABLE "TablaProductos" (  
  "IdProducto" INTEGER,  
  "Nombre" TEXT,  
  "Marca" TEXT,  
  "Proveedor" TEXT,  
  "Precio" REAL,  
  "Descuento" REAL,  
  "Unidad" INTEGER,  
  PRIMARY KEY("IdProducto" AUTOINCREMENT));
```

### Estructura del Proyecto

### Usar Terminal y Usas los

### Comandos en el directorio

#### Backend:

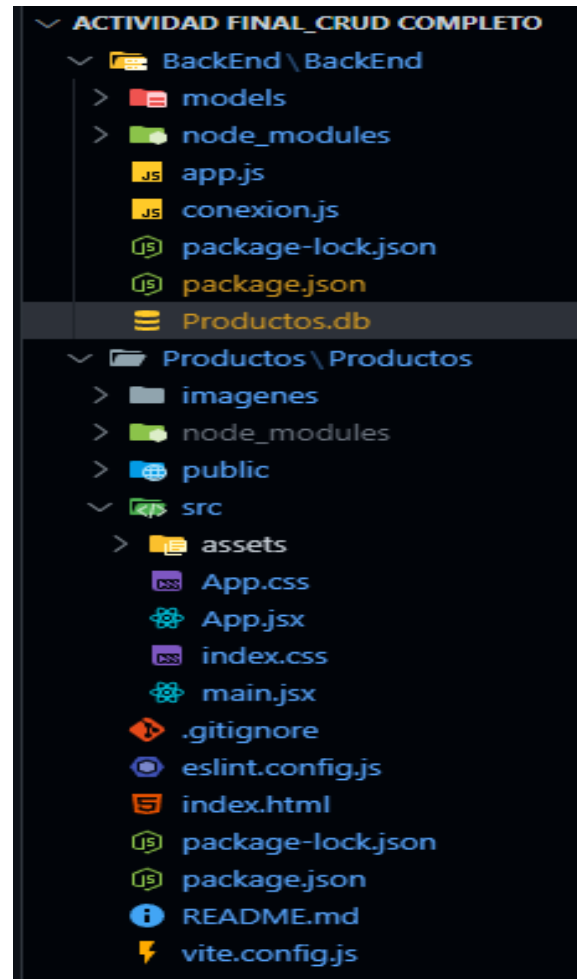
npm init

npm i express sequelize sqlite3

npm cors

#### DIR Productos:

npm create vite@latest



npm install

**Este código configura Sequelize para usar una base de datos SQLite almacenada en Productos.db y la exporta para su uso en el proyecto.**

```
const Sequelize = require('sequelize');
const sequelize = new Sequelize(
  {
    dialect: 'sqlite',
    storage: './Productos.db',
  }
);

module.exports = sequelize;
```

**Este código define un modelo de base de datos llamado Producto usando Sequelize. Está conectado a la instancia de Sequelize importada desde ../conexion y mapea los datos de la tabla TablaProductos. Cada producto tiene atributos como IdProducto (clave primaria con auto-incremento), Nombre, Marca, Proveedor, Precio, Descuento y Unidad. Además, desactiva los timestamps (createdAt y updatedAt) para evitar registros automáticos de fecha. Finalmente, exporta el modelo para su uso en otras partes del proyecto.**

```
const { DataTypes } = require('sequelize');
const sequelize = require('../conexion');

const Producto = sequelize.define('Producto', {
  IdProducto: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
  Nombre: { type: DataTypes.STRING },
  Marca: { type: DataTypes.STRING },
  Proveedor: { type: DataTypes.STRING },
  Precio: { type: DataTypes.FLOAT },
  Descuento: { type: DataTypes.FLOAT },
  Unidad: { type: DataTypes.INTEGER },
}, {
  {
    tableName: 'TablaProductos',
    timestamps: false,
  }
});

module.exports = Producto;
```

Este código define una API REST en Node.js usando Express y Sequelize para gestionar productos en una base de datos. Se inicializa Express, se define el puerto 3000 y se añade body-parser para procesar datos JSON. También se habilita CORS para permitir solicitudes de diferentes orígenes.

### Operaciones CRUD:

**Create:** Agrega nuevos productos con un POST a /productos.

**Read:** Recupera la lista de productos con un GET a /productos.

**Update:** Modifica un producto existente mediante PUT en /productos/:IdProducto.

```
const express = require('express');
const bodyParser = require('body-parser');
const productos = require('./models/productos');
const puerto = 3000;

const app = express();
app.use(bodyParser.json());

app.listen(puerto, () => {
  console.log('Servicio iniciado en el puerto', puerto);
});

const cors = require("cors");
app.use(cors());

// Create
app.post('/productos', async (req, res) => {
  try {
    const { Nombre, Marca, Proveedor, Precio, Descuento, Unidad } = req.body;
    const data = await productos.create({ Nombre, Marca, Proveedor, Precio, Descuento, Unidad });
    res.status(201).send(data);
  } catch (error) {
    console.error(error);
    res.status(500).send({ mensaje: 'Error al crear producto', error });
  }
});

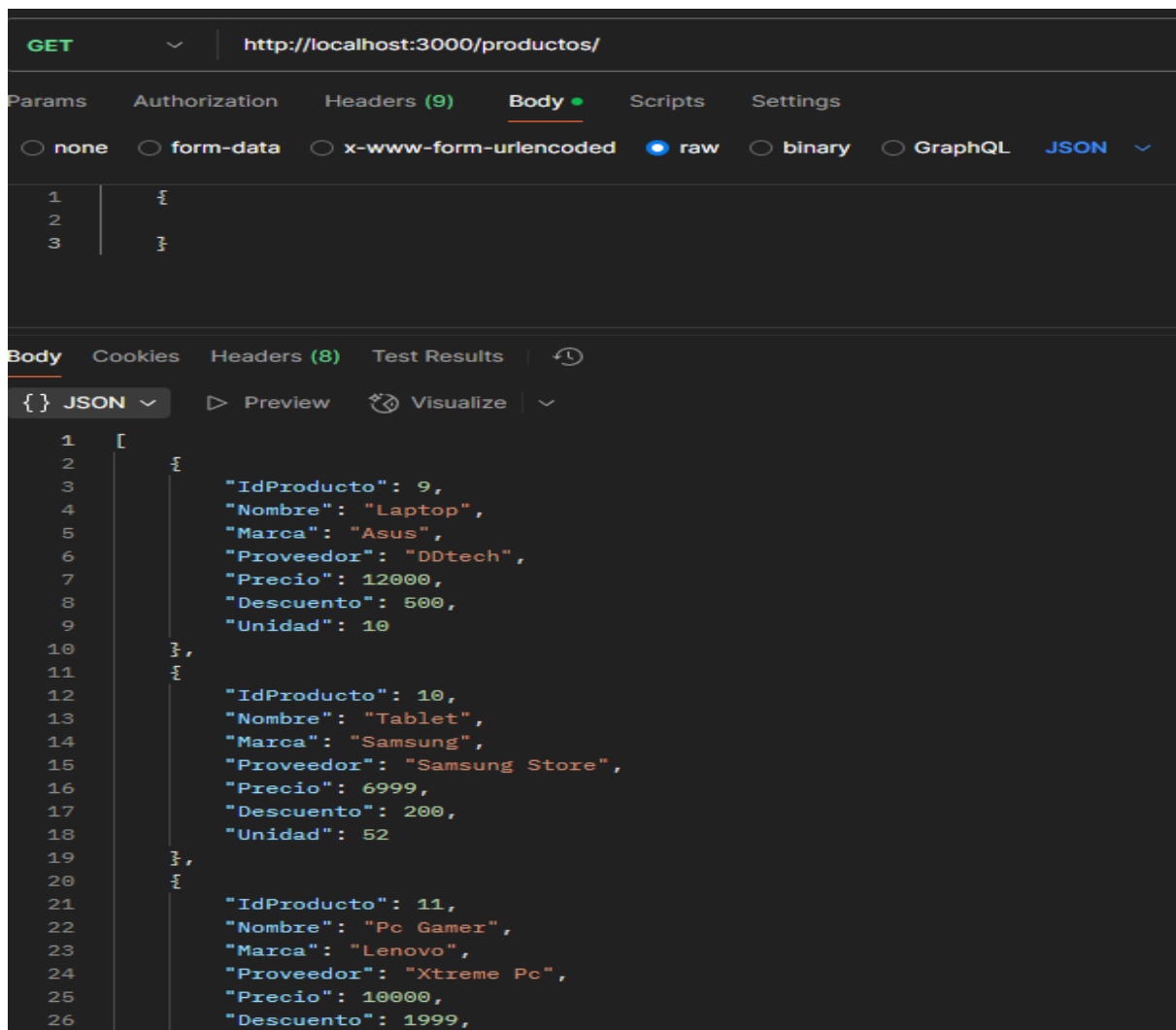
// Read
app.get('/productos', async (req, res) => {
  try {
    const data = await productos.findAll();
    res.status(200).send(data);
  } catch (error) {
    console.error(error);
    res.status(500).send({ mensaje: 'Error al obtener productos', error });
  }
});
```

```
// Update
app.put('/productos/:IdProducto', async (req, res) => {
  try {
    const { Nombre, Marca, Proveedor, Precio, Descuento, Unidad } = req.body;
    const { IdProducto } = req.params;
    const data = await productos.update(
      { Nombre, Marca, Proveedor, Precio, Descuento, Unidad },
      { where: { IdProducto } }
    );
    res.status(200).send(data);
  } catch (error) {
    console.error(error);
    res.status(500).send({ mensaje: 'Error al actualizar producto', error });
  }
});

// Delete
app.delete('/productos/:IdProducto', async (req, res) => {
  try {
    const { IdProducto } = req.params;
    const data = await productos.destroy({ where: { IdProducto } });
    res.status(200).send({ mensaje: 'Producto eliminado', resultado: data });
  } catch (error) {
    console.error(error);
    res.status(500).send({ mensaje: 'Error al eliminar producto', error });
  }
});
```

## FUNCIONES CRUD MEDIANTE POSTMAN

### GET.



The screenshot shows a Postman interface with a GET request to `http://localhost:3000/productos/`. The request is configured with the 'Body' tab selected and 'raw' format. The response is displayed in the 'Body' tab, showing a JSON array of three product objects.

**Request:**

- Method: GET
- URL: `http://localhost:3000/productos/`
- Body: raw (empty)

**Response:**

```
[
  {
    "IdProducto": 9,
    "Nombre": "Laptop",
    "Marca": "Asus",
    "Proveedor": "DDtech",
    "Precio": 12000,
    "Descuento": 500,
    "Unidad": 10
  },
  {
    "IdProducto": 10,
    "Nombre": "Tablet",
    "Marca": "Samsung",
    "Proveedor": "Samsung Store",
    "Precio": 6999,
    "Descuento": 200,
    "Unidad": 52
  },
  {
    "IdProducto": 11,
    "Nombre": "Pc Gamer",
    "Marca": "Lenovo",
    "Proveedor": "Xtreme Pc",
    "Precio": 10000,
    "Descuento": 1999,
    "Unidad": 1
  }
]
```

```
[
  {
    "IdProducto": 9,
    "Nombre": "Laptop",
    "Marca": "Asus",
    "Proveedor": "DDtech",
    "Precio": 12000,
    "Descuento": 500,
    "Unidad": 10
  },
  {
    "IdProducto": 10,
    "Nombre": "Tablet",
    "Marca": "Samsung",
    "Proveedor": "Samsung Store",
    "Precio": 6999,
    "Descuento": 200,
    "Unidad": 52
  },
  {
    "IdProducto": 11,
    "Nombre": "Pc Gamer",
    "Marca": "Lenovo",
    "Proveedor": "Xtreme Pc",
    "Precio": 10000,
    "Descuento": 1999,
    "Unidad": 14
  }
]
```

## POST.

```
1  {
2      "Nombre": "RTX 5090",
3      "Marca": "MSI",
4      "Proveedor": "DDtech",
5      "Precio": 50000,
6      "Descuento": 100,
7      "Unidad": 2
8  }
```

Body Cookies Headers (8) Test Results

{ } JSON Preview Visualize

```
1  {
2      "IdProducto": 12,
3      "Nombre": "RTX 5090",
4      "Marca": "MSI",
5      "Proveedor": "DDtech",
6      "Precio": 50000,
7      "Descuento": 100,
8      "Unidad": 2
9  }
```

```
[
  {
    "IdProducto": 9,
    "Nombre": "Laptop",
    "Marca": "Asus",
    "Proveedor": "DDtech",
    "Precio": 12000,
    "Descuento": 500,
    "Unidad": 10
  },
  {
    "IdProducto": 10,
    "Nombre": "Tablet",
    "Marca": "Samsung",
    "Proveedor": "Samsung Store",
    "Precio": 6999,
    "Descuento": 200,
    "Unidad": 52
  },
  {
    "IdProducto": 11,
    "Nombre": "Pc Gamer",
    "Marca": "Lenovo",
    "Proveedor": "Xtreme Pc",
    "Precio": 10000,
    "Descuento": 1999,
    "Unidad": 14
  },
  {
    "IdProducto": 12,
    "Nombre": "RTX 5090",
    "Marca": "MSI",
    "Proveedor": "DDtech",
    "Precio": 50000,
    "Descuento": 100,
    "Unidad": 2
  }
]
```

## PUT.

The screenshot shows a REST client interface with the following details:

- Method:** PUT
- URL:** http://localhost:3000/productos/10
- Body Type:** none (selected)
- Body Content:**

```
1 {
2   "Nombre": "Laptop Gamer",
3   "Marca": "MSI",
4   "Proveedor": "Walmart",
5   "Precio": 19999,
6   "Descuento": 100,
7   "Unidad": 10
8 }
```
- Response Body:**

```
1 [
2   1
3 ]
```

```
[
  {
    "IdProducto": 9,
    "Nombre": "Laptop",
    "Marca": "Asus",
    "Proveedor": "DDtech",
    "Precio": 12000,
    "Descuento": 500,
    "Unidad": 10
  },
  {
    "IdProducto": 10,
    "Nombre": "Laptop Gamer",
    "Marca": "MSI",
    "Proveedor": "Walmart",
    "Precio": 19999,
    "Descuento": 100,
    "Unidad": 10
  },
  {
    "IdProducto": 11,
    "Nombre": "Pc Gamer",
    "Marca": "Lenovo",
    "Proveedor": "Xtreme Pc",
    "Precio": 10000,
    "Descuento": 1999,
    "Unidad": 14
  },
  {
    "IdProducto": 12,
    "Nombre": "RTX 5090",
    "Marca": "MSI",
    "Proveedor": "DDtech",
    "Precio": 50000,
    "Descuento": 100,
    "Unidad": 2
  }
]
```

## DELETE.

The screenshot shows a REST client interface with the following details:

- Method:** DELETE
- URL:** http://localhost:3000/productos/10
- Body Type:** none (selected)
- Body Content:**

```
1 {
2   "Nombre": "Laptop Gamer",
3   "Marca": "MSI",
4   "Proveedor": "Walmart",
5   "Precio": 19999,
6   "Descuento": 100,
7   "Unidad": 10
8 }
```
- Response Body:**

```
1 {
2   "mensaje": "Producto eliminado",
3   "resultado": 1
}
```

**Este código crea una aplicación en React para gestionar un inventario de productos.**

**Estado y formularios: Usa useState para almacenar la lista de productos y los datos del formulario.**

**Operaciones con la API: Se comunica con la API REST en <http://localhost:3000/productos> para realizar operaciones CRUD:**

- **GET (obtenerProductos):** Recupera la lista de productos.
- **POST (agregarProducto):** Agrega un nuevo producto.
- **PUT (modificarProducto):** Modifica un producto existente.
- **DELETE (eliminarProducto):** Elimina un producto.

**Interfaz: Muestra una lista de productos y permite la gestión de datos a través de un formulario con inputs y botones.**

**Efectos: Usa useEffect para cargar automáticamente los productos cuando la app se inicia.**



```
import React, { useState, useEffect } from "react";
import "./App.css";

const App = () => {
  const [productos, setProductos] = useState([]);
  const [formData, setFormData] = useState({
    IdProducto: "",
    Nombre: "",
    Marca: "",
    Proveedor: "",
    Precio: "",
    Descuento: "",
    Unidad: "",
  });

  // Obtener productos
  const obtenerProductos = async () => {
    const res = await fetch("http://localhost:3000/productos");
    const data = await res.json();
    setProductos(data);
  };

  // Manejar cambios en el formulario
  const manejarCambio = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  // Agregar producto
  const agregarProducto = async () => {
    await fetch("http://localhost:3000/productos", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(formData),
    });
    obtenerProductos();
  };

  // Modificar producto
  const modificarProducto = async () => {
    await fetch(`http://localhost:3000/productos/${formData.IdProducto}`, {
      method: "PUT",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(formData),
    });
    obtenerProductos();
  };
};
```

```

// Eliminar producto
const eliminarProducto = async () => {
  await fetch(`http://localhost:3000/productos/${formData.IdProducto}`, {
    method: "DELETE",
  });
  obtenerProductos();
};

useEffect(() => {
  obtenerProductos();
}, []);

return (
  <div>
    <div className="navbar">SUPER PC GAMING STORE</div>

    <div className="container">
      <h2>REGISTRO DE INVENTARIO DE LA TIENDA</h2>

      <div className="input-container">
        <input type="number" name="IdProducto" placeholder="ID" onChange={manejarCambio} />
        <input type="text" name="Nombre" placeholder="Nombre" onChange={manejarCambio} />
        <input type="text" name="Marca" placeholder="Marca" onChange={manejarCambio} />
        <input type="text" name="Proveedor" placeholder="Proveedor" onChange={manejarCambio} />
        <input type="number" name="Precio" placeholder="Precio" onChange={manejarCambio} />
        <input type="number" name="Descuento" placeholder="Descuento" onChange={manejarCambio} />
        <input type="number" name="Unidad" placeholder="Unidad" onChange={manejarCambio} />
      </div>

      <div className="button-container">
        <button onClick={agregarProducto}>Agregar</button>
        <button onClick={modificarProducto}>Modificar</button>
        <button onClick={eliminarProducto}>Eliminar</button>
        <button onClick={obtenerProductos}>Consultar</button>
      </div>
    </div>
  </div>
);

```

```

    <h2>Lista de Productos</h2>
    <div className="product-list">
      {productos.map((producto) => (
        <div key={producto.IdProducto} className="product-card">
          
          <h3>{producto.Nombre}</h3>
          <p>Marca: {producto.Marca}</p>
          <p>Proveedor: {producto.Proveedor}</p>
          <p>Precio: ${producto.Precio}</p>
        </div>
      ))}
    </div>
  </div>
);

```

