Práctica

Api Rest / ORM

Para esta práctica se creará una API Rest donde se usará el patrón MVC (Modelo Vista controlador) y el ORM Sequelize.

Estructura del proyecto (MVC)

my-api/

```
│
├── config/          # Configuración de la base de datos
│   └── db.config.js
│
├── controllers/     # Lógica de negocio
│   └── product.controller.js
│
├── models/          # Definición de modelos
│   └── product.model.js
│   └── index.js
│
├── routes/          # Rutas de la API
│   └── product.routes.js
│
├── app.js           # Archivo principal de Express
├── .env             # Variables de entorno
├── package.json
```

Inicializar el proyecto

mkdir my-api && cd my-api

npm init -y

npm install express sequelize mysql2 dotenv

npm install --save-dev nodemon

Crear archivo .env: Los archivos env sirven para almacenar variables de entorno, estas variables sirven para almacenar datos que pueden ser usadas en manera global en el proyecto, para este caso se almacenan datos de la conexión a la BD.

DB_HOST= srv1851.hstgr.io

DB_USER= u911718531_senati

DB_PASSWORD= S3nati123

DB_NAME= u911718531_moviles20251

DB_PORT=3306

PORT=3000

Configurar la base de datos (config/db.config.js)

```
require("dotenv").config();

module.exports = {
  HOST: process.env.DB_HOST,
  USER: process.env.DB_USER,
  PASSWORD: process.env.DB_PASSWORD,
  DB: process.env.DB_NAME,
  PORT: process.env.DB_PORT,
  dialect: "mysql",
};
```

Conectar Sequelize (models/index.js)

```
const { Sequelize, DataTypes } = require("sequelize");
const dbConfig = require("../config/db.config");


const sequelize = new Sequelize(
  dbConfig.DB,
  dbConfig.USER,
  dbConfig.PASSWORD,
  {
    host: dbConfig.HOST,
    dialect: dbConfig.dialect,
  }
);
```

```
const db = {};
db.Sequelize = Sequelize;
db.sequelize = sequelize;

// Importar modelos
db.products = require("./product.model")(sequelize, DataTypes);

module.exports = db;
```

Crear modelo (models/product.model.js)

```
module.exports = (sequelize, DataTypes) => {
  const Product = sequelize.define("Product", {
    name: {
      type: DataTypes.STRING,
      allowNull: false,
    },
    price: {
      type: DataTypes.DECIMAL(10, 2),
      allowNull: false,
    },
    stock: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
  });

  return Product;
};
```

Crear controlador (controllers/product.controller.js)

```
const db = require("../models");
const Product = db.products;

exports.create = async (req, res) => {
 try {
  const product = await Product.create(req.body);
  res.json(product);
 } catch (err) {
  res.status(500).json({ message: err.message });
 }
};

exports.findAll = async (req, res) => {
 try {
  const products = await Product.findAll();
  res.json(products);
 } catch (err) {
  res.status(500).json({ message: err.message });
 }
};

exports.findOne = async (req, res) => {
 try {
  const product = await Product.findByPk(req.params.id);
  product ? res.json(product) : res.status(404).send("Not found");
 } catch (err) {
  res.status(500).json({ message: err.message });
 }
};
```

```
exports.update = async (req, res) => {
 try {
  const rows = await Product.update(req.body, {
   where: { id: req.params.id },
  });
  res.json({ updated: rows[0] });
 } catch (err) {
  res.status(500).json({ message: err.message });
 }
};

exports.delete = async (req, res) => {
 try {
  const rows = await Product.destroy({ where: { id: req.params.id } });
  res.json({ deleted: rows });
 } catch (err) {
  res.status(500).json({ message: err.message });
 }
};
```

Crear rutas (routes/product.routes.js)

```
const express = require("express");
const router = express.Router();
const products = require("../controllers/product.controller");

router.post("/", products.create);
router.get("/", products.findAll);
router.get("/:id", products.findOne);
router.put("/:id", products.update);
router.delete("/:id", products.delete);

module.exports = router;
```

Configurar la app (app.js)

```
const express = require("express");
require("dotenv").config();

const app = express();
const db = require("./models");

app.use(express.json());

// Rutas
const productRoutes = require("./routes/product.routes");
app.use("/api/products", productRoutes);

// Iniciar servidor y sincronizar DB
const PORT = process.env.PORT || 3000;
db.sequelize.sync().then(() => {
  app.listen(PORT, () => {
    console.log(`Servidor corriendo en puerto ${PORT}`);
  });
});
```

Script de desarrollo (package.json)

Agrega en la sección de scripts:

```
"scripts": {
  "start": "node app.js",
  "dev": "nodemon app.js"
}
```

Ejecución de tu API

Vamos a ejecutar cada endpoint de la API:

1. Crear un producto

Método: POST
URL: http://localhost:3000/api/products
Headers: Content-Type: application/json

Body (raw / JSON):

```
{
  "name": "Mouse inalámbrico",
  "price": 49.99,
  "stock": 100
}
```

2. Obtener todos los productos

Método: GET
URL: http://localhost:3000/api/products

No necesita body ni headers especiales.

3. Obtener un producto por ID

Método: GET
URL: http://localhost:3000/api/products/1

Reemplaza 1 por el ID del producto que quieras consultar.

4. Actualizar un producto por ID

Método: PUT
URL: http://localhost:3000/api/products/1
Headers: Content-Type: application/json

Body

```
{
  "name": "Mouse óptico",
  "price": 39.99,
  "stock": 80
}
```

5. Eliminar un producto por ID

Método: DELETE
URL: http://localhost:3000/api/products/1

Esto eliminará el producto con ID 1.

ACTIVIDAD: Crear al menos 02 nuevos endpoints, capturar pantalla y subir a GitHub.

CREACIÓN DE FRONTEND EN REACT

En esta guía usaremos React para crear una interfaz de usuario para consumir la API creada.

Estructura de carpetas sugerida: Puedes usar esta u otra estructura.

src/

├── components/

│ ├── ProductForm.tsx

│ └── ProductList.tsx

├── services/

│ └── productService.ts

├── types/

│ └── Product.ts

├── App.tsx

└── main.tsx

Definir tipos con TypeScript: Los tipos en TypeScript son una forma de decirle al lenguaje qué tipo de datos se espera que tenga una variable, función, objeto, etc.

Sirven para:

- Evitar errores antes de que el programa se ejecute.
- Aclarar el propósito del código.
- Tener ayuda del autocompletado y validación en el editor.

```
export interface Product {
  id?: number;
  name: string;
  price: number;
  stock: number;
}
```

Servicio para consumir la API

Ruta: **src/services/productService.ts**

```
import { Product } from "../types/Product";
```

```
const API_URL = "http://localhost:3000/api/products";

export const getAllProducts = async (): Promise<Product[]> => {
  const res = await fetch(API_URL);
  return await res.json();
};

export const getProductById = async (id: number): Promise<Product> => {
  const res = await fetch(`${API_URL}/${id}`);
  return await res.json();
};

export const createProduct = async (product: Product): Promise<Product> => {
  const res = await fetch(API_URL, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(product),
  });
  return await res.json();
};

export const updateProduct = async (id: number, product: Product): Promise<Product> =>
{
  const res = await fetch(`${API_URL}/${id}`, {
    method: "PUT",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(product),
  });
  return await res.json();
};
```

```
export const deleteProduct = async (id: number): Promise<void> => {
  await fetch(`${API_URL}/${id}`, { method: "DELETE" });
};
```

Formulario de producto

Ruta: src/components/ProductForm.tsx

```
import { useState, useEffect, ChangeEvent, FormEvent } from "react";
import { Product } from "../types/Product";

interface Props {
  onSubmit: (product: Product) => void;
  selectedProduct: Product | null;
}

export default function ProductForm({ onSubmit, selectedProduct }: Props) {
  const [form, setForm] = useState<Product>({ name: "", price: 0, stock: 0 });

  useEffect(() => {
    if (selectedProduct) {
      setForm(selectedProduct);
    }
  }, [selectedProduct]);

  const handleChange = (e: ChangeEvent<HTMLInputElement>) => {
    const { name, value } = e.target;
    setForm({ ...form, [name]: name === "name" ? value : parseFloat(value) });
  };

  const handleSubmit = (e: FormEvent) => {
    e.preventDefault();
    onSubmit(form);
```

```
  setForm({ name: "", price: 0, stock: 0 });
 };

 return (
  <form onSubmit={handleSubmit}>
   <input name="name" value={form.name} onChange={handleChange}
placeholder="Nombre" />
   <input name="price" value={form.price} onChange={handleChange}
placeholder="Precio" type="number" />
   <input name="stock" value={form.stock} onChange={handleChange}
placeholder="Stock" type="number" />
   <button type="submit">{form.id ? "Actualizar" : "Crear"}</button>
  </form>
 );
}
```

Lista de productos

Ruta: src/components/ProductList.tsx

```
import { Product } from "../types/Product";

interface Props {
 products: Product[];
 onDelete: (id: number) => void;
 onEdit: (product: Product) => void;
}

export default function ProductList({ products, onDelete, onEdit }: Props) {
 return (
  <ul>
   {products.map((product) => (
    <li key={product.id}>
     {product.name} - ${product.price} - Stock: {product.stock}
```

```
        <button onClick={() => onEdit(product)}>Editar</button>
        <button onClick={() => onDelete(product.id!)}>Eliminar</button>
      </li>
    ))}
  </ul>
 );
}
```

Lógica principal en App.tsx

Ruta: src/App.tsx

```
import { useEffect, useState } from "react";
import ProductForm from "./components/ProductForm";
import ProductList from "./components/ProductList";
import { Product } from "./types/Product";
import {
 getAllProducts,
 createProduct,
 updateProduct,
 deleteProduct,
} from "./services/productService";

function App() {
 const [products, setProducts] = useState<Product[]>([]);
 const [selectedProduct, setSelectedProduct] = useState<Product | null>(null);

 const loadProducts = async () => {
  const data = await getAllProducts();
  setProducts(data);
 };

 const handleCreateOrUpdate = async (product: Product) => {
  if (product.id) {
```

```
    await updateProduct(product.id, product);
   } else {
    await createProduct(product);
   }
   setSelectedProduct(null);
   await loadProducts();
  };

  const handleDelete = async (id: number) => {
   await deleteProduct(id);
   await loadProducts();
  };

  const handleEdit = (product: Product) => {
   setSelectedProduct(product);
  };

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

  return (
   <div>
    <h1>Gestión de Productos</h1>
    <ProductForm onSubmit={handleCreateOrUpdate}
selectedProduct={selectedProduct} />
    <ProductList products={products} onDelete={handleDelete} onEdit={handleEdit} />
   </div>
  );
}
```

```
export default App;
```