Documentación Técnica

1. Introducción

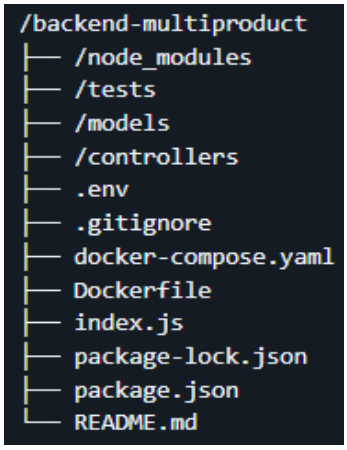
Este proyecto es una **API RESTful** diseñada para gestionar productos. El objetivo principal de la API es proporcionar una interfaz sencilla y flexible para realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) sobre productos de una tienda o inventario. La API permite a los usuarios interactuar con los productos de forma eficiente, realizar consultas sobre los productos disponibles, y actualizar o eliminar información de productos cuando sea necesario.

La aplicación está construida utilizando **Node.js** y **Express.js**, lo que permite un desarrollo rápido y eficiente de APIs en un entorno escalable. Los datos de los productos se almacenan en una base de datos **Postgres**, asegurando la persistencia de los datos de manera confiable.

Tecnologías utilizadas:

- **Node.js**: Utilizado para crear el servidor backend, permitiendo un entorno de ejecución eficiente para JavaScript en el servidor.
- **Express.js**: Framework minimalista que facilita la creación y gestión de rutas y middleware en la API.
- **Sequelize ORM**: Herramienta para interactuar con la base de datos PostgreSQL de manera abstracta, facilitando la creación y ejecución de consultas SQL de forma segura y eficiente.
- **PostgreSQL**: Sistema de gestión de bases de datos relacional que almacena la información de los productos en una única tabla Products.
- **Mocha**: Framework de pruebas utilizado para realizar pruebas automatizadas en la API, asegurando que las funcionalidades trabajen correctamente y de manera estable.
- **Postman**: Herramienta utilizada para realizar pruebas manuales sobre las rutas de la API, facilitando la validación y el testeo de la interfaz.
- **Swagger**: Herramienta que se utiliza para generar y visualizar la documentación interactiva de la API. Permite a los desarrolladores ver todas las rutas disponibles, los parámetros esperados, las respuestas posibles, y realizar pruebas directamente desde la interfaz de Swagger UI.
- **Git**: Sistema de control de versiones utilizado para gestionar el código fuente y realizar el seguimiento de los cambios a lo largo del desarrollo. **.gitignore** se usa para evitar que ciertos archivos no deseados sean añadidos al repositorio (por ejemplo, dependencias, configuraciones locales, etc.).

Arquitectura del Proyecto



Explicación de las carpetas y archivos

1. src/ (Código fuente del proyecto)

Este directorio contiene el código principal de la API, organizado en varias carpetas para una estructura limpia y modular.

a. controllers/

- **productController.js**: Aquí es donde se definen las funciones que manejan las solicitudes HTTP relacionadas con los productos. Cada función (o controlador) se encarga de recibir la solicitud, interactuar con el servicio correspondiente, y devolver la respuesta adecuada. Por ejemplo, si se hace una solicitud para crear un producto, el controlador gestionará el flujo de la solicitud y enviará la respuesta correspondiente.

b. models/

- **product.js**: Aquí se define el modelo de los productos utilizando Sequelize, un ORM (Object-Relational Mapping). Este modelo describe cómo se estructura la tabla de productos en la base de datos y contiene los atributos que los productos deben tener, como nombre, precio, cantidad, etc.

c. routes/

- **productRoutes.js**: Este archivo define todas las rutas relacionadas con los productos. Es el lugar donde se especifica qué controlador debe manejar qué tipo de solicitud HTTP. Las rutas de la API, como GET /api/products, POST /api/products, PUT /api/products/:id,

y DELETE /api/products/:id se definen aquí, asociándose con las funciones correspondientes del controlador.

d. services/

- **productService.js**: Este archivo contiene la lógica de negocio que interactúa directamente con el modelo y realiza las operaciones CRUD en la base de datos. Mientras que los controladores gestionan las solicitudes y respuestas HTTP, los servicios se encargan de realizar las operaciones en la base de datos, como crear, leer, actualizar o eliminar productos.

e. config/

- **db.js**: Este archivo contiene la configuración de la conexión a la base de datos. Aquí se especifica cómo conectar la API a la base de datos PostgreSQL (host, puerto, base de datos, usuario, contraseña, etc.). También puede contener configuraciones relacionadas con Sequelize, como la definición de la base de datos y las opciones de conexión.

2. test/ (Archivos de prueba)

Este directorio contiene las pruebas automatizadas para la API. Las pruebas son importantes para asegurarse de que las funcionalidades de la API están funcionando correctamente y para detectar posibles errores de manera temprana.

a. product.test.js

Este archivo contiene las pruebas unitarias de la API. Utiliza Mocha y, probablemente, Chai o alguna otra librería de aserciones para hacer las verificaciones. Las pruebas pueden incluir verificar que la API responde correctamente con el código de estado adecuado (por ejemplo, 201 para un producto creado), que las rutas devuelvan los resultados correctos, que los errores se gestionen correctamente, etc.

3. Archivos de Configuración y Setup

a. Dockerfile

El Dockerfile es un archivo que contiene las instrucciones necesarias para construir la imagen de Docker del proyecto. Dentro de este archivo se definen los pasos para crear el contenedor de la aplicación, como la instalación de dependencias, la configuración del entorno, la exposición de puertos y la ejecución de la API.

En este archivo, se pueden encontrar instrucciones como:

- FROM node:14: Especifica la imagen base (en este caso, Node.js).

- COPY . .: Copia los archivos del proyecto al contenedor.
- RUN npm install: Instala las dependencias del proyecto dentro del contenedor.
- CMD ["npm", "start"]: Define el comando por defecto que se ejecutará al iniciar el contenedor.

b. docker-compose.yml

Este archivo se utiliza para definir y ejecutar múltiples contenedores Docker. Si el proyecto depende de varios servicios (como una base de datos PostgreSQL o un contenedor de pruebas), el docker-compose.yml proporciona una forma sencilla de configurarlos y ponerlos en funcionamiento.

En este archivo se definen servicios como:

- **app**: El contenedor que ejecuta la API.
- **db**: El contenedor que ejecuta PostgreSQL.

Se especifican puertos, volúmenes y redes para facilitar la comunicación entre los contenedores.

c. .gitignore

El archivo .gitignore le indica a Git qué archivos o directorios no debe incluir en el control de versiones. Esto es útil para evitar que archivos sensibles, temporales o innecesarios se suban al repositorio, como:

- node_modules/: Carpeta de dependencias de Node.js, que se puede regenerar con npm install.
- *.log: Archivos de registro generados por la aplicación.
- .env: Archivos de configuración con credenciales sensibles.

d. package.json

Este archivo contiene información sobre el proyecto, como su nombre, versión, dependencias y scripts. En este archivo se especifican las dependencias necesarias para el funcionamiento del proyecto (como express, sequelize, etc.) y los scripts para iniciar el servidor, ejecutar pruebas, y otras tareas de desarrollo.

Ejemplo de algunas secciones en package.json:

- **dependencies**: Paquetes que la aplicación necesita para funcionar (por ejemplo, express, sequelize).
- **scripts**: Comandos predefinidos para ejecutar tareas como iniciar el servidor (npm start), ejecutar pruebas (npm test), etc.

e. package-lock.json

Este archivo es generado automáticamente por npm cuando se instalan dependencias. Contiene un registro detallado de las versiones exactas de las dependencias instaladas, lo que asegura que el proyecto tenga las mismas versiones de dependencias en todas las instalaciones y entornos.

4. README.md

El archivo README.md contiene la documentación básica del proyecto. Proporciona una descripción general de la API, instrucciones de instalación, uso y ejemplos de solicitudes. Es útil para cualquier persona que quiera comprender rápidamente cómo funciona el proyecto y cómo utilizarlo.

Base de Datos

- **Base de datos**: En esta sección describe la base de datos que se utiliza, la estructura de la base de datos (en este caso, una sola tabla), y cómo se conecta el proyecto con ella.
- **Diagrama de la base de datos** (si es necesario).

Ejemplo: Este proyecto utiliza una base de datos **MySQL** que contiene una tabla llamada **Products**. A continuación, se describen las columnas de esta tabla:

Tabla Products:

id	integer
nombre*	string
descripcion	string
precio*	number($float)
cantidad*	integer
categoria	string
imagen_url	string

Instalación

1. Clona este repositorio:
git clone https://github.com/cewhizzar/backend-multiproduct
cd backend-multiproduct

2. Instala las dependecias:
npm install

3. Configura tu base de datos en un archivo .env:

```
APP_PORT=3000
DB_USER=postgres
DB_HOST=localhost
DB_DATABASE=multiproductDB
DB_PASSWORD=tu_contraseña
DB_PORT=5432
DB_DIALECT=postgres
```

Uso

1. Para iniciar el servidor:

```
npm run dev
```

El servidor se ejecutará en **http://localhost:3000.**

2. La API estará disponible en los siguientes endpoints:

```
POST /api/products: Crear un nuevo producto.
GET /api/products: Obtener todos los productos.
GET /api/products/:id: Obtener un producto por su ID.
PUT /api/products/:id: Actualizar un producto por su ID.
DELETE /api/products/:id: Eliminar un producto por su ID.
```

Tests

1. Para ejecutar los tests:

```
npm run test
```

Ejecución en Docker:

Si se desea ejecutar la aplicación usando Docker, sigue estos pasos:

1. **Construye la imagen Docker:**
   docker build -t backend-multiproduct
2. **Levanta los contenedores con Docker Compose**:
   docker-compose up

Dependencias

```
"dependencies": {
  "axios": "^1.2.2",
  "dotenv": "^16.4.7",
  "express": "^4.18.2",
  "form-data": "^4.0.0",
  "mocha": "^10.2.0",
  "nodemon": "^2.0.20",
  "nyc": "^15.1.0",
  "pg": "^8.13.1",
  "pg-hstore": "^2.3.4",
  "sequelize": "^6.37.5",
  "swagger-ui-express": "^5.0.1",
  "unit.js": "^2.1.1",
  "yamljs": "^0.3.0"
},
"devDependencies": {
  "assert": "^2.1.0",
  "supertest": "^7.0.0"
}
```

1. **axios**: Cliente HTTP para realizar solicitudes a APIs, soporta Promesas.
2. **dotenv**: Carga variables de entorno desde un archivo .env a process.env.
3. **express**: Framework minimalista para crear servidores y APIs web.
4. **form-data**: Permite manejar y enviar datos de formularios en solicitudes HTTP.
5. **mocha**: Framework para pruebas unitarias y de integración.
6. **nodemon**: Reinicia automáticamente el servidor al detectar cambios en los archivos.
7. **nyc**: Generador de cobertura de código para medir qué partes del código se ejecutan durante las pruebas.
8. **pg**: Librería oficial de PostgreSQL para interactuar con bases de datos.
9. **pg-hstore**: Serializa/deserializa datos en formato JSON para trabajar con el tipo hstore en PostgreSQL.
10. **sequelize**: ORM para bases de datos relacionales como PostgreSQL, MySQL, SQLite, etc.
11. **swagger-ui-express**: Permite integrar y visualizar documentación de API generada con Swagger en aplicaciones Express.
12. **unit.js**: Biblioteca para escribir pruebas unitarias y de integración.
13. **yamljs**: Manejo de archivos YAML, útil para configuraciones y documentación.
14. **assert**: Herramienta para realizar aserciones en pruebas, parte del núcleo de Node.js.
15. **supertest**: Herramienta para realizar pruebas de solicitudes HTTP en aplicaciones Node.js.