Casa abierta al tiempo

UNIVERSIDAD AUTÓNOMA METROPOLITANA

UNIDAD CUAJIMALPA

LICENCIATURA EN TECNOLOGÍAS Y SISTEMAS DE INFORMACIÓN.

PROYECTO FINAL

Sistema de Inventario Full Stack

UNIDAD DE ENSEÑANZA

PROGRAMACIÓN WEB DINÁMICO

Dr. Guillermo Monroy

Marin Sanchez Jose Abraham
2233030514

INTRODUCCIÓN

En este reporte documentare todo el proceso de la realización del proyecto final de la UEA Programación Web Dinámico en donde elegí de entre varias opciones el Sistema de Gestión de Inventario Básico , este proyecto es un sistema fullstack y el objetivo principal es tener nuestra página web con secciones de administración de productos, registros, categorías y un dashboard de métricas.

REQUERIMIENTOS

En las especificaciones dadas por el profesor se trabajara en cumplir con los siguientes requisitos funcionales.

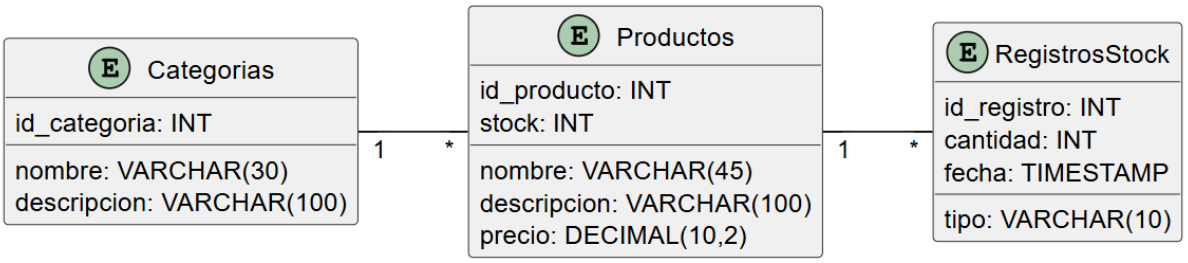
- Creación de CRUD de productos y Categorías.
- Registros de movimientos de stock de entradas y salidas.
- La tabla de productos deberá contar con la implementación de filtros, paginación y búsqueda.
- Creación de un dashboard que muestre métricas simples como pueden ser productos con bajo stock y el total de categorías.
- Documentación con Swagger
- Para la parte del frontend además se hará uso de:
  - Routing.
  - Servicios de comunicación HTTP.
  - Formularios reactivos.
  - Componentes organizados por módulos.

TECNOLOGÍAS

Para realizar este proyecto se usaron las siguientes tecnologías:
FRONTEND : Angular y Angular Material.
BACKEND : Spring Boot 3.5.3, Spring Web , Spring Data JPA.
LENGUAJES: Java, Typescript, Javascript, HTML, CSS.
BASE DE DATOS : MySQL.
OTROS: Swagger, Github(https://github.com/MarinoX/proyecto_web_dinamico)

ARQUITECTURA

Estructura de la Base de Datos:

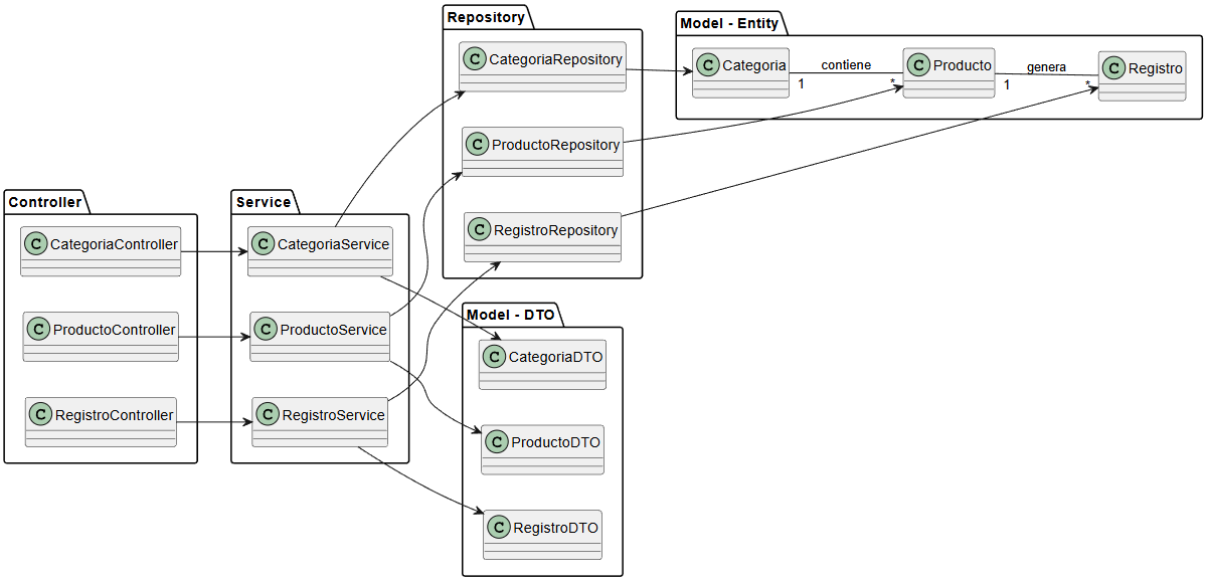


Mi base de datos hecha en SQL está organizada alrededor de la tabla **Categorías** ya que esta funciona como el punto de clasificación principal. Cada categoría tiene un nombre y una descripción, y sirve para agrupar productos de forma ordenada.

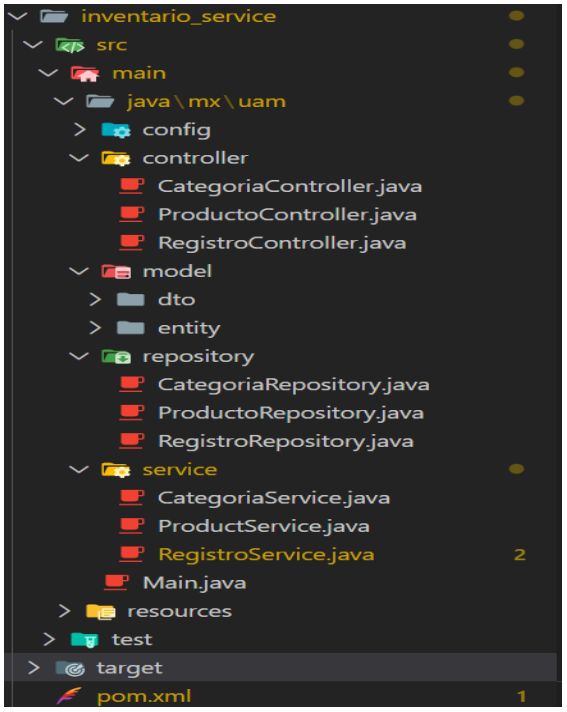
Luego tengo la tabla de **Productos** que es pues la parte principal del inventario. Aquí se almacena todo lo importante: nombre, descripción, precio y el stock actual. Cada producto está ligado a una categoría mediante una relación uno a muchos, lo que ayuda a evitar el duplicar información y mantiene todo bien normalizado.

Finalmente tengo la tabla **RegistrosStock** que guarda el historial de entradas y salidas del inventario. Al hacer un nuevo registro indica si fue una entrada o una salida, la cantidad de stock y la fecha exacta.

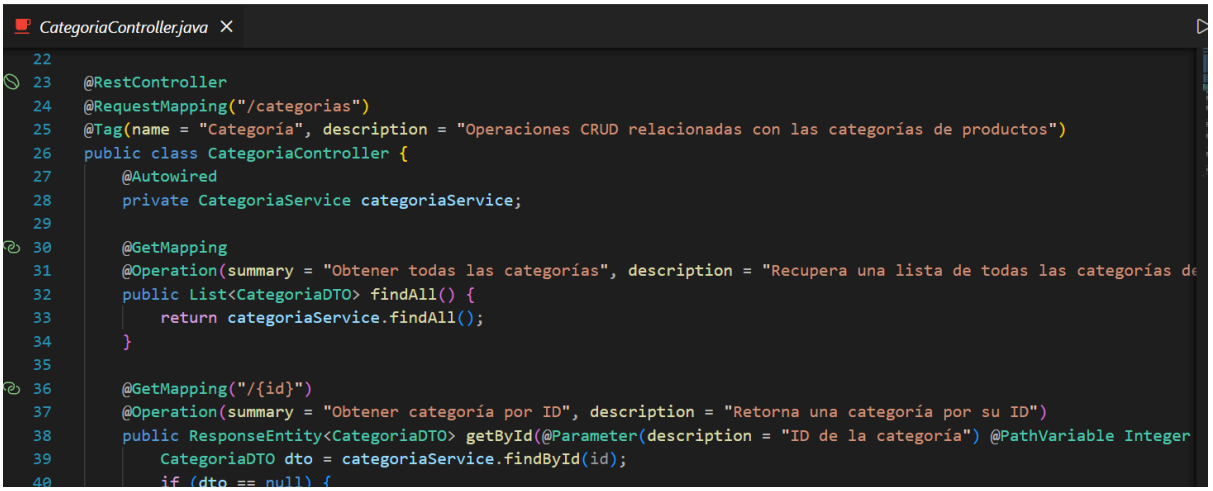
Estructura del Backend:



El backend está estructurado siguiendo el patrón MVC, dividiendo la arquitectura en varias capas: Controllers, Services, Repositories y Models (Entity y DTO).



Los Controllers (CategoriaController, ProductoController, RegistroController) actúan como puntos de entrada que reciben las peticiones HTTP y las dirigen a la capa correspondiente donde cada controller maneja las rutas REST específicas para sus recursos.



Los servicios contienen toda la lógica de negocio de la aplicación ya que aquí es donde ocurren las validaciones, cálculos y transformaciones de datos. Por ejemplo el RegistroService no solo crea registros de stock sino que verifica que haya stock

suficiente antes de permitir una salida y si si se puede realizar el movimiento automáticamente actualiza el stock del producto.

CategoriaService.java

```java
import mx.uam.repository.ProductoRepository;

@Service
public class CategoriaService {
    @Autowired
    private CategoriaRepository categoriarepository;

    @Autowired
    private ProductoRepository productoRepository;

    public List<CategoriaDTO> findAll(){
        return categoriarepository.findAll().stream()
        .map(this::toDTO)
        .collect(Collectors.toList());
    }

    public CategoriaDTO findById(Integer id) {
        if (id == null) {
```

Los repositoriy contienen las queries y operaciones CRUD necesarias y manejan el acceso directo a la base de datos mediante Spring Data JPA. Los modelos están divididos en Entities (Categoria, Producto, Registro) que mapean directamente las tablas de la base de datos y los DTO que son los objetos que se envían y reciben en las peticiones HTTP.

CategoriaRepository.java

```java

import mx.uam.model.entity.Categoria;

public interface CategoriaRepository extends JpaRepository<Categoria, Integer> {

}
```

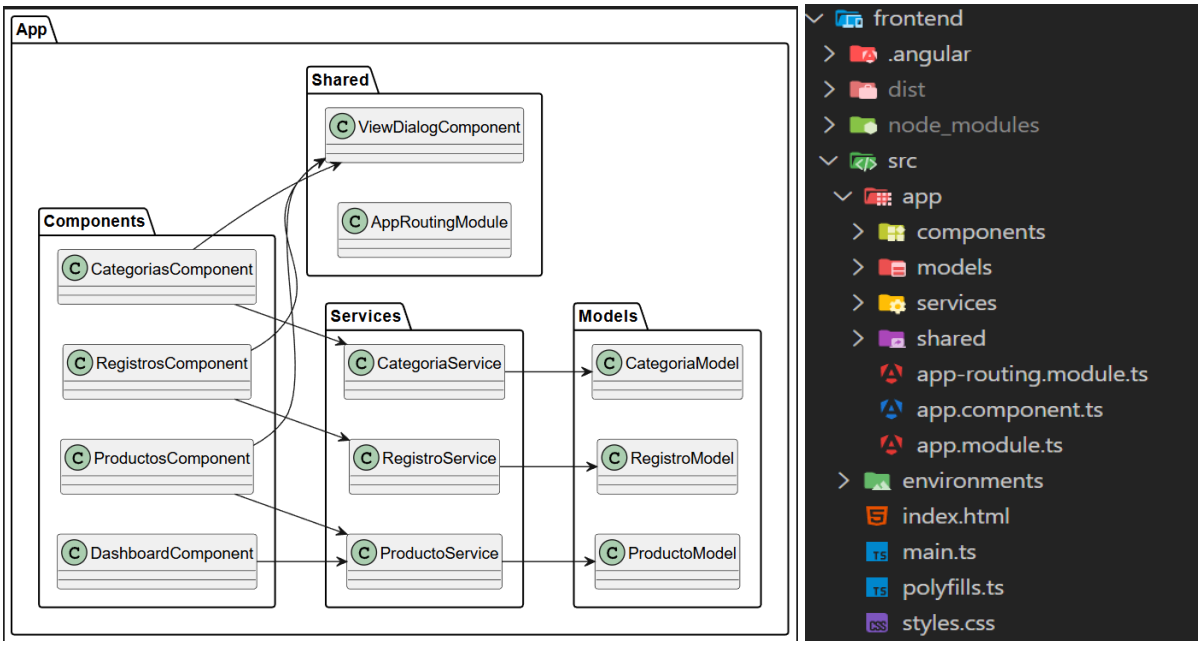
CategoriaDTO.java

```java
package mx.uam.model.dto;

public class CategoriaDTO {
    private Integer id;
    private String nombre;
    private String descripcion;

    public Integer getId() {
        return id;
    }
```

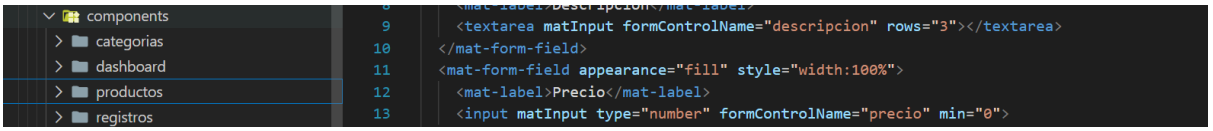
Categoria.java

```java

@Entity
public class Categoria {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String nombre;
    private String descripcion;

    @OneToMany(mappedBy = "categoria", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    private List<Producto> productos;
```

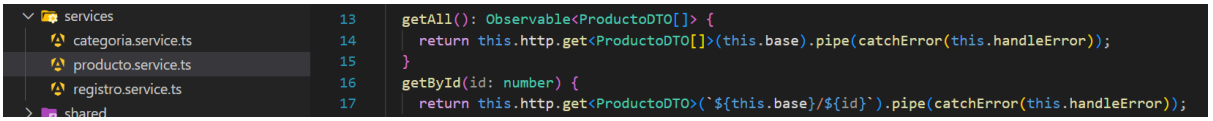
Estructura del Frontend:



El frontend está construido con Angular y sigue una arquitectura modular basada en componentes. La carpeta App actúa como contenedor principal que contiene toda la aplicación, dentro de esta se encuentran varios componentes para cada apartado específico como el DashboardComponent que muestra métricas del inventario, CategoriasComponent que gestiona las categorías, ProductosComponent para la administración de los productos y RegistrosComponent para visualizar y crear registros de entrada y salida de stock, aqui cada componente es responsable de su propia ui y maneja la interacción con el usuario.

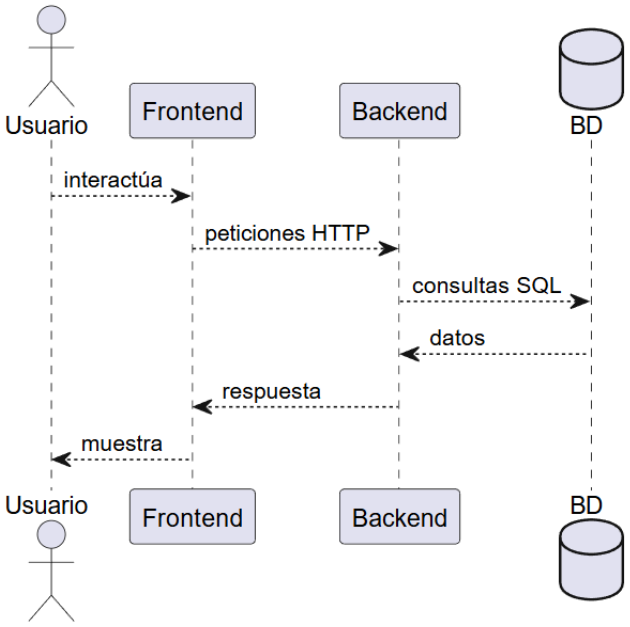


Los Servicios son la conexión entre los componentes y el backend. Los servicios realizan las llamadas HTTP al servidor y transforman los datos recibidos en los Models que son las interfaces que definen la estructura de datos de la aplicación. Haciendo que los componentes no interactúen directamente con el backend sino que solicitan datos a los servicios.



La carpeta Shared contiene componentes y módulos reutilizables en toda la aplicación, como el ViewDialogComponent y el AppRoutingModule que gestiona la navegación entre los diferentes componentes.

Finalmente la interacción de mi sistema full stack sería esta:



EJECUCIÓN DEL PROYECTO

Para la ejecución del sistema es necesario revisar el README de mi repositorio de Github ya que ahí explico paso a paso cómo ejecutarlo dando los pasos desde la base de datos, crear el usuario necesario y ejecutar el backend, que es el que levanta la interfaz y la documentación Swagger.

Resumiendo un poco se necesita tener MySQL configurado, correr los scripts incluidos en la carpeta sql y después iniciar el backend con mvn spring-boot:run o usando el archivo JAR ya compilado. Una vez ejecutado el sistema el frontend se muestra desde http://localhost:8080 y la documentación completa de la API hecha en Swagger se ve desde http://localhost:8080/swagger-ui/index.html

PRUEBAS DE BACKEND MEDIANTE SWAGGER

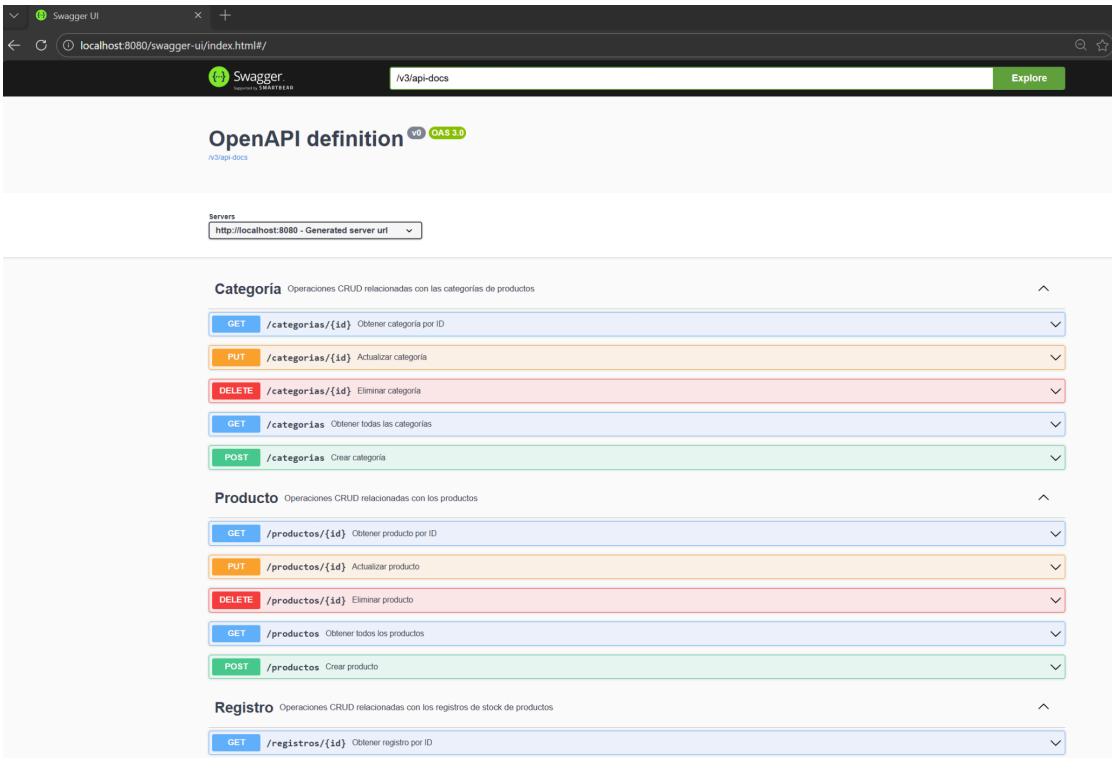
Para la ejecución del proyecto es esencial la lectura del README del repositorio de GITHUB.

Para la documentación de mi backend utilicé Swagger a través de la dependencia el cual genera de forma automática toda la documentación basada en las anotaciones de mis controllers.

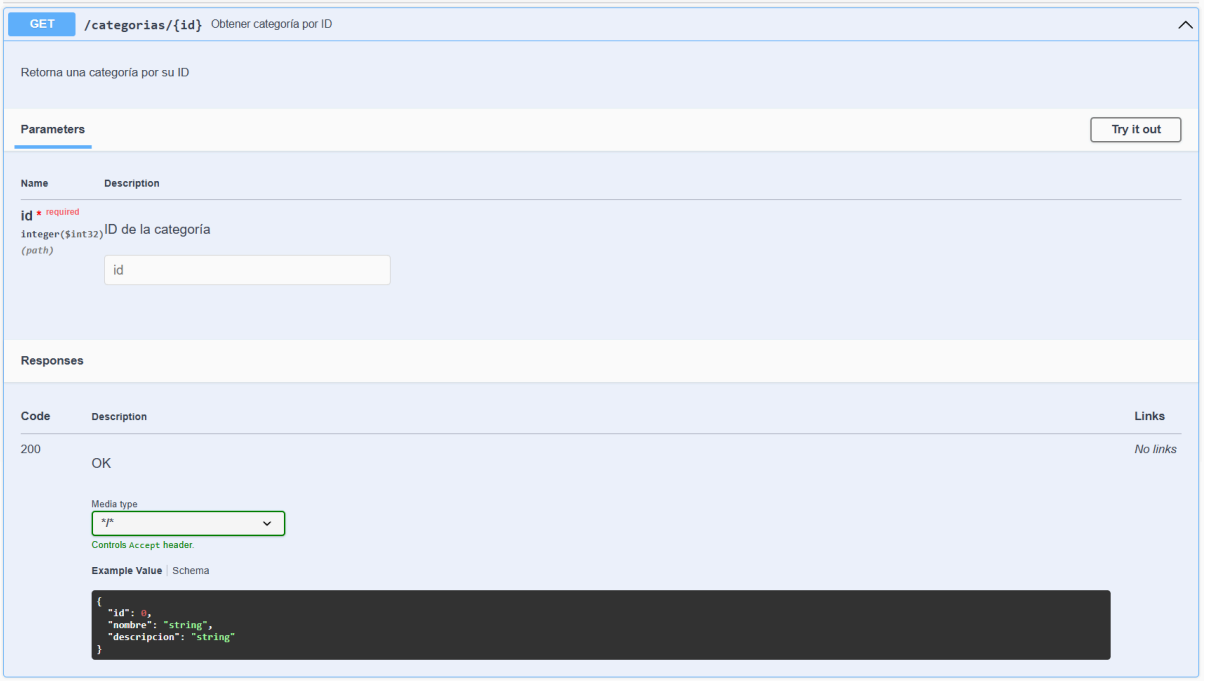
```
@GetMapping
@Operation(summary = "Obtener todas las categorías", description = "Recupera una lista de todas las categorías
public List<CategoriaDTO> findAll() {
    return categoriaService.findAll();
}
```

A la documentación se puede acceder desde el navegador y nos permite visualizar los endpoints, sus parámetros necesarios y nos deja probar las peticiones directamente sin necesidad de usar Postman, CURL o el mismo frontend.

En mi proyecto la documentación se encuentra disponible en la ruta: http://localhost:8080/swagger-ui/index.html

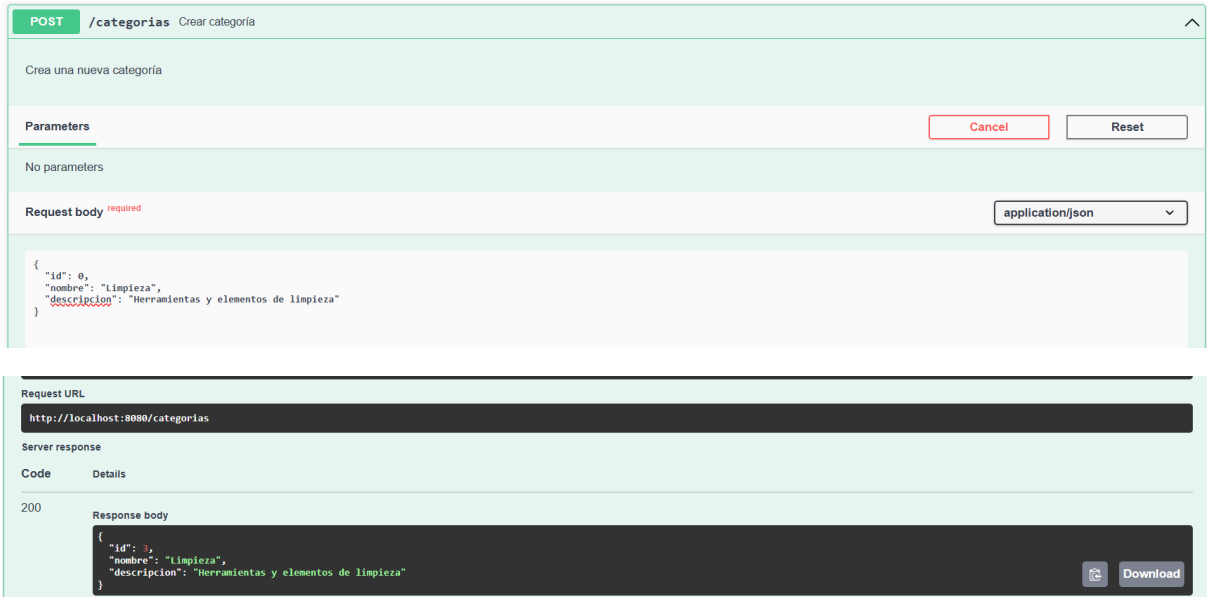


Desde aquí se pueden ver todos los endpoints de Productos, Categorías y Registros de Stock. Cada uno muestra información como: método HTTP, parámetros, respuestas esperadas, códigos de error y ejemplos de uso ,lo que facilita el desarrollo porque se pueden revisar que las rutas funcionan de manera correcta.

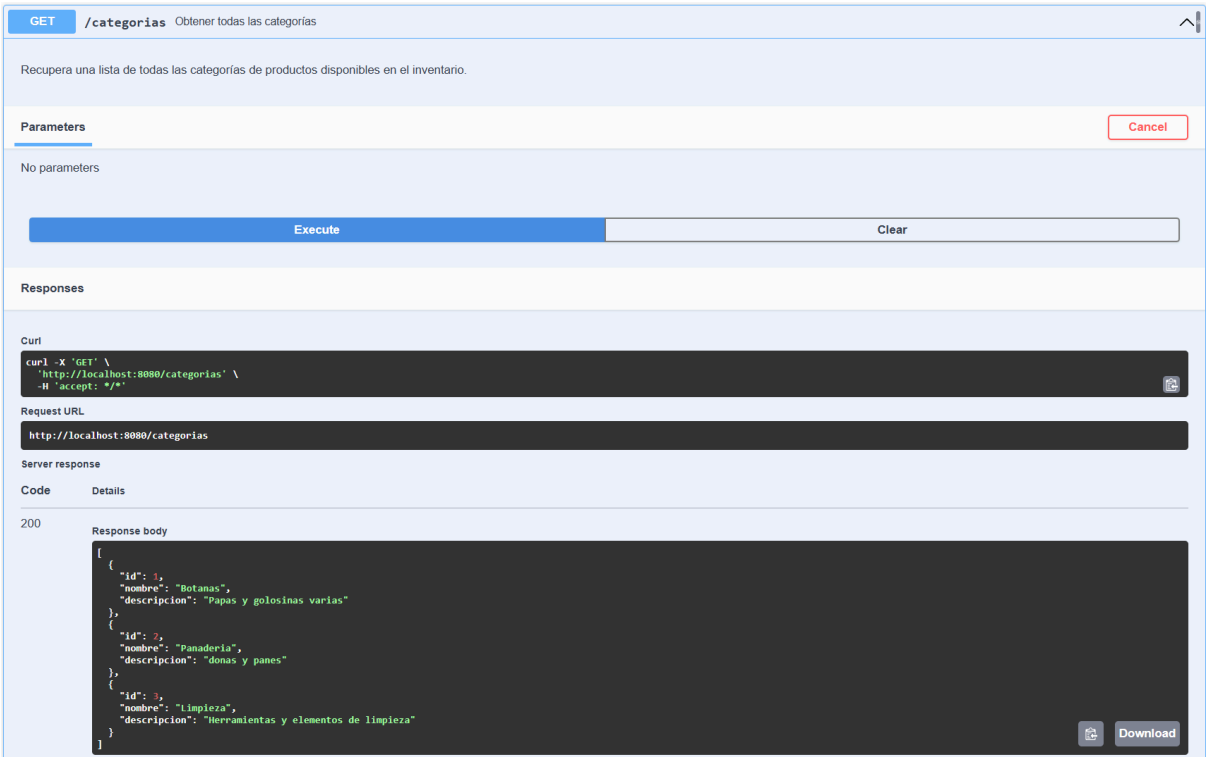


Algunas pruebas para demostrar el uso de la documentación swagger son las siguientes:

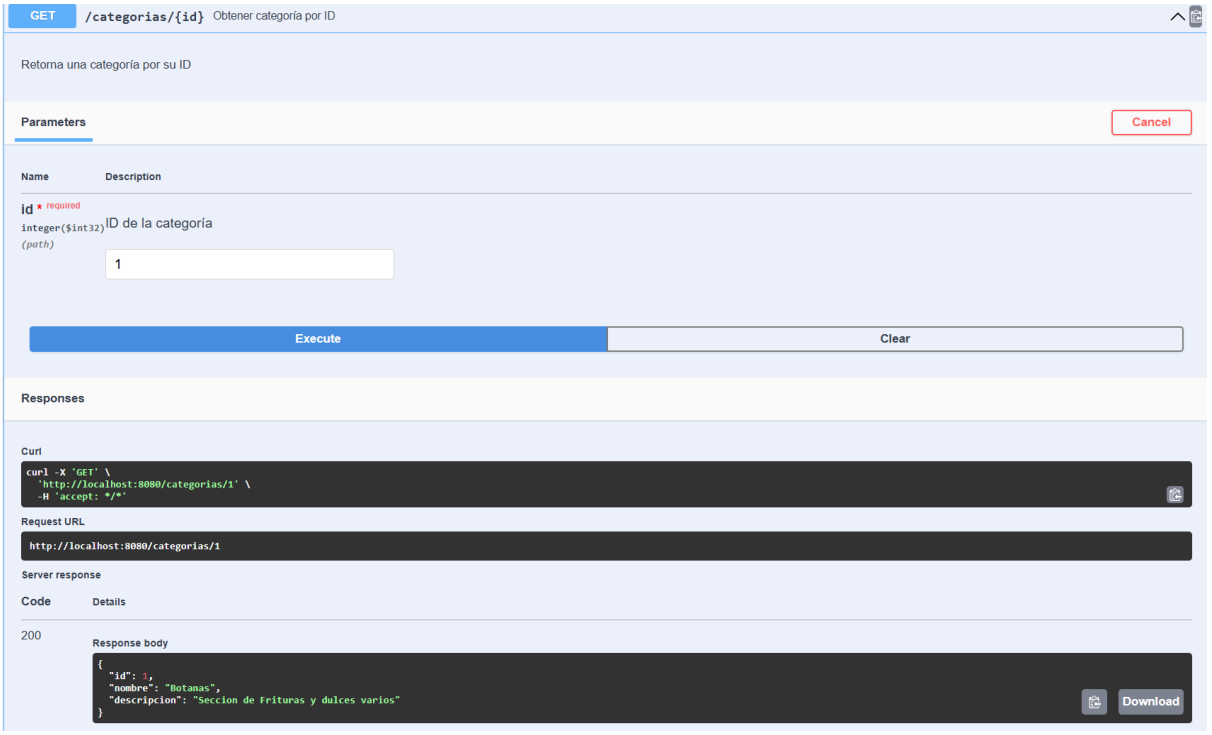
1.- Crear Categoría.



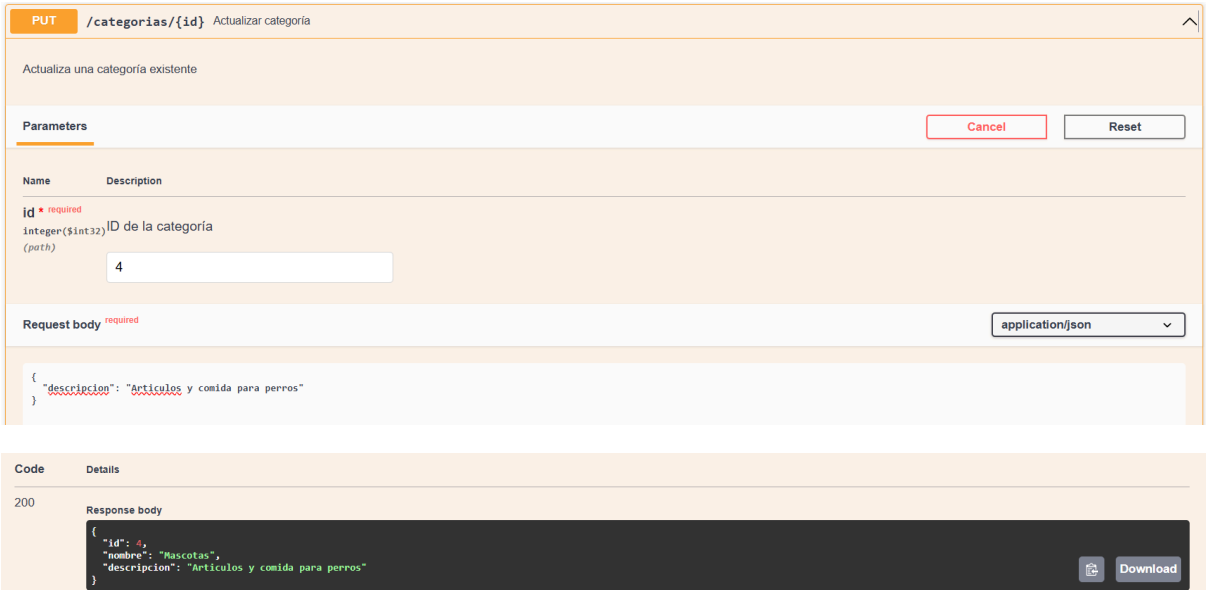
2.- Obtener todas las categorías.



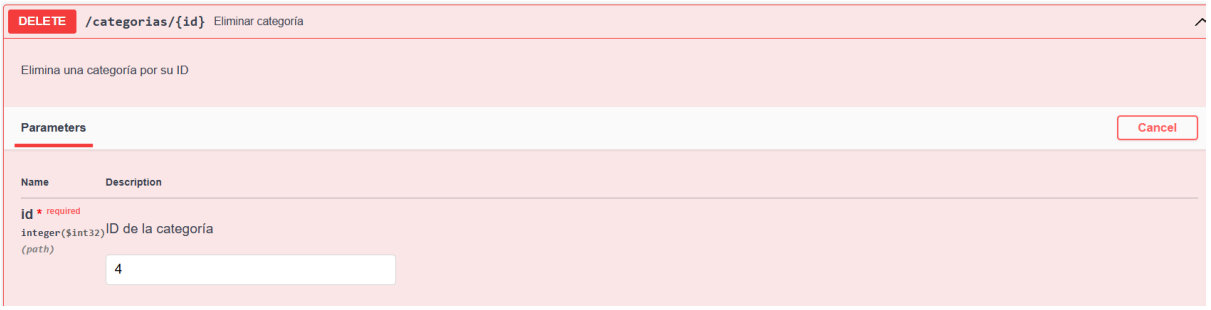
3.- Obtener categoría por id.



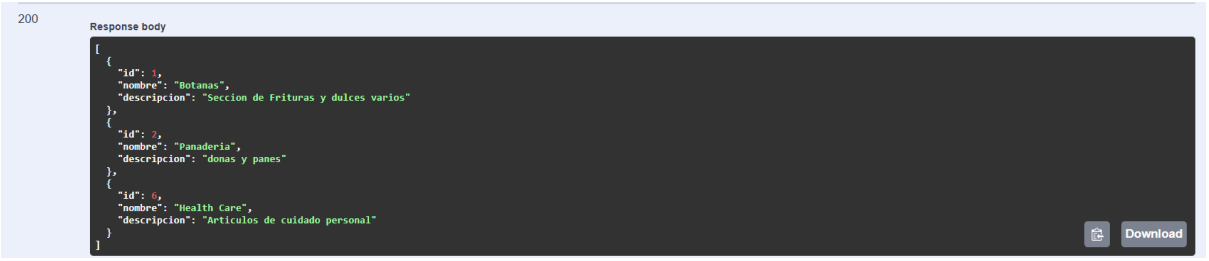
4.- Actualizar Categoría.



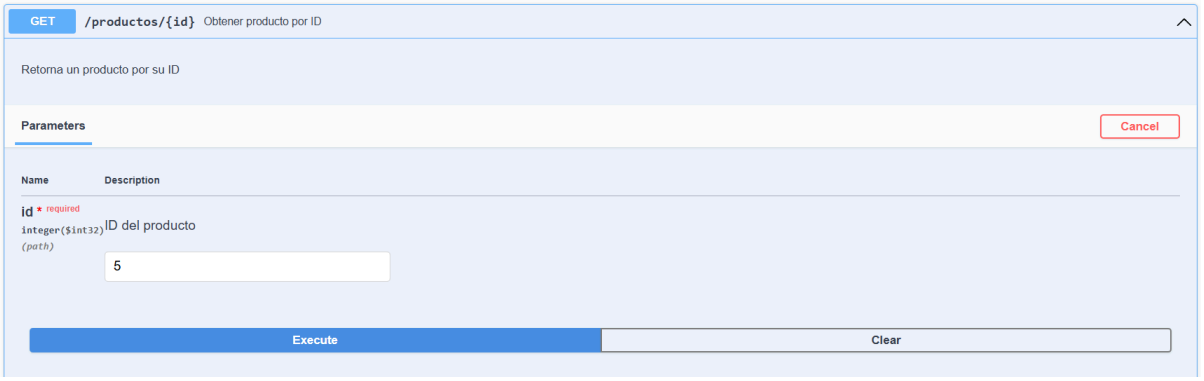
5.- Eliminar Categoría

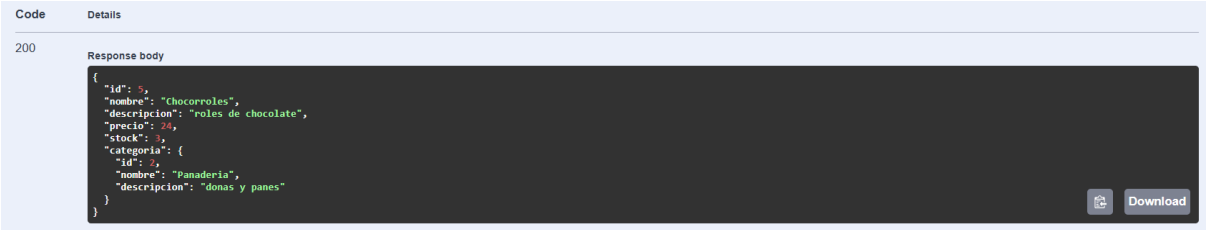


Revisión en obtener todas las categorías para ver que ya no este la categoria.



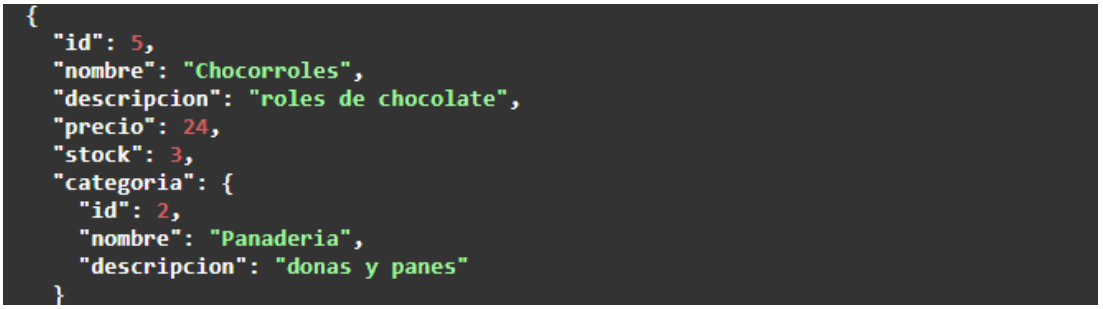
6.- Obtener producto por id.



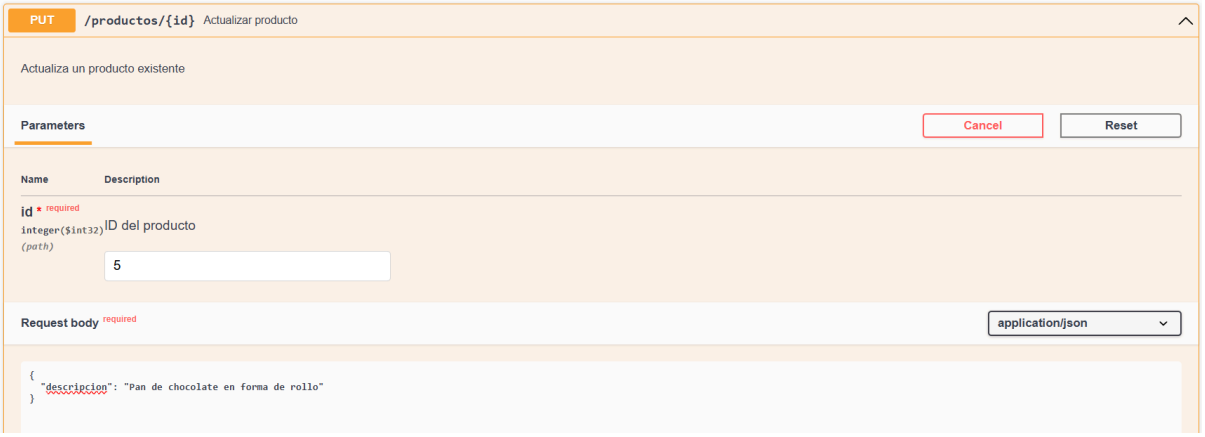


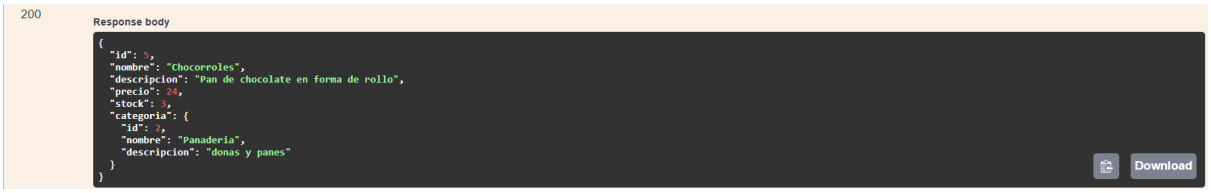
7.- Actualizar producto.

Antes de actualizar:

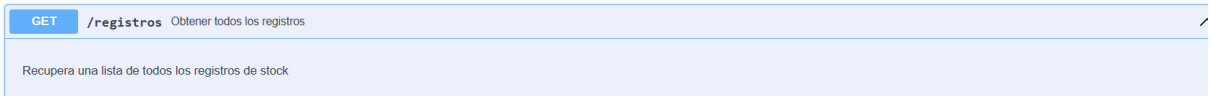


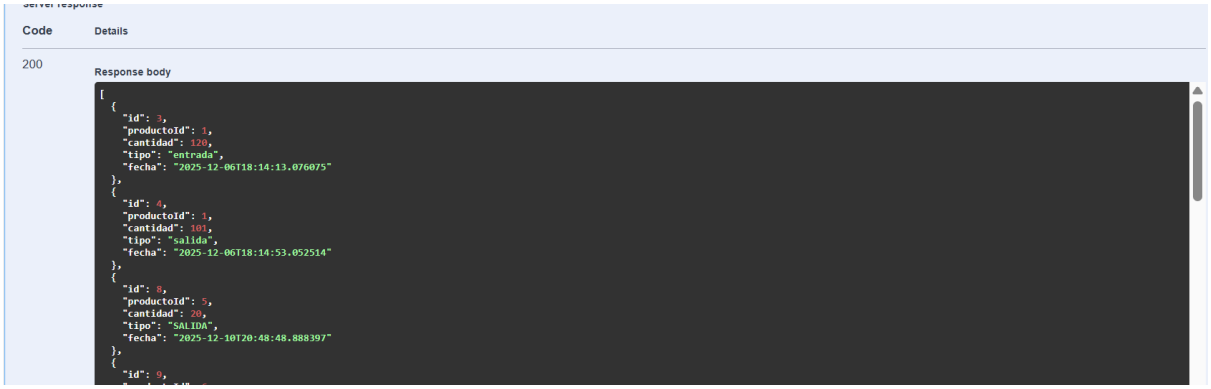
Usando el endpoint de actualizar producto por id



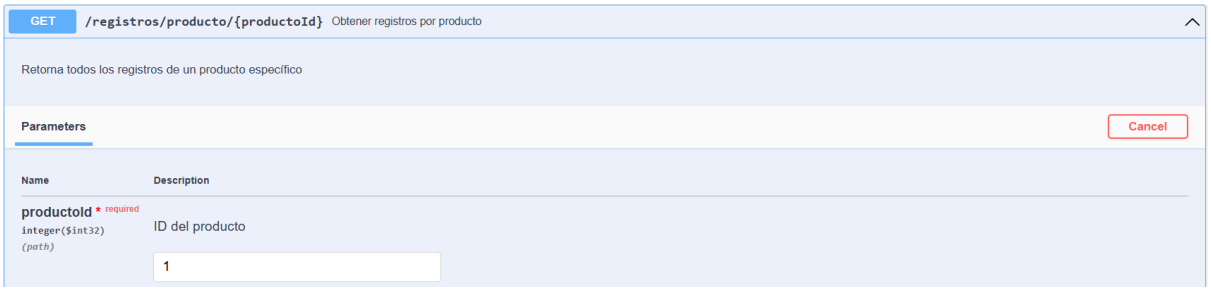


8.- Obtener todos los registros





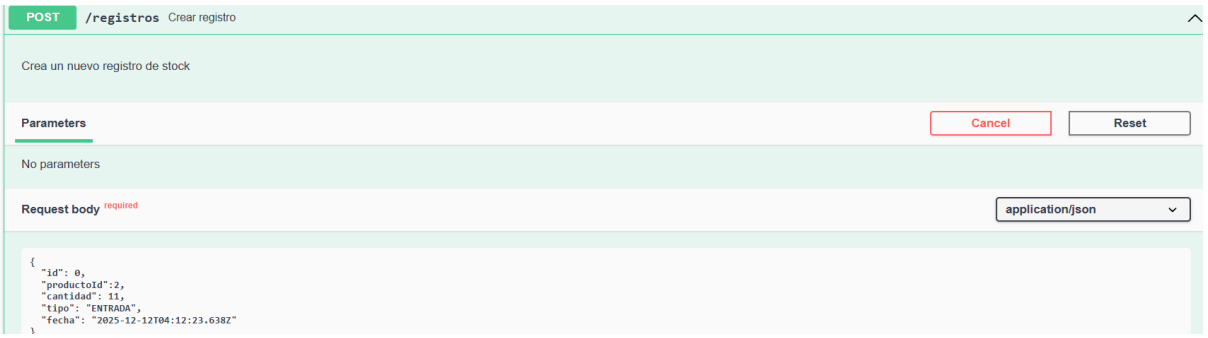
9.- Obtener registro por producto

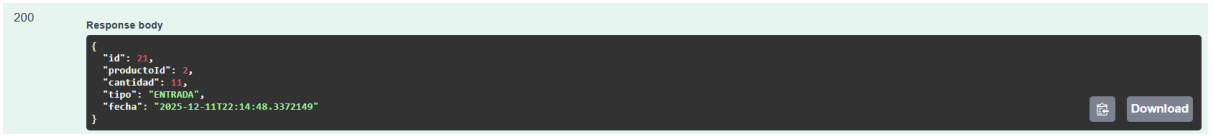


10.- Crear nuevo registro.

Se hará un registro de ingreso y uno de salida, el producto actualmente es el siguiente:

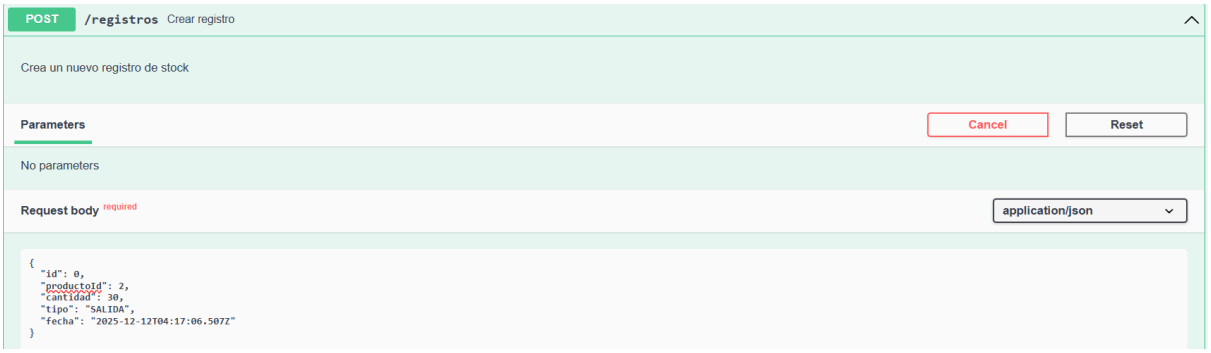
Registro de entrada.





Revisando el producto con el endpoint de obtener producto ha obtenido 11 mas de stock.

Registro de Salida.



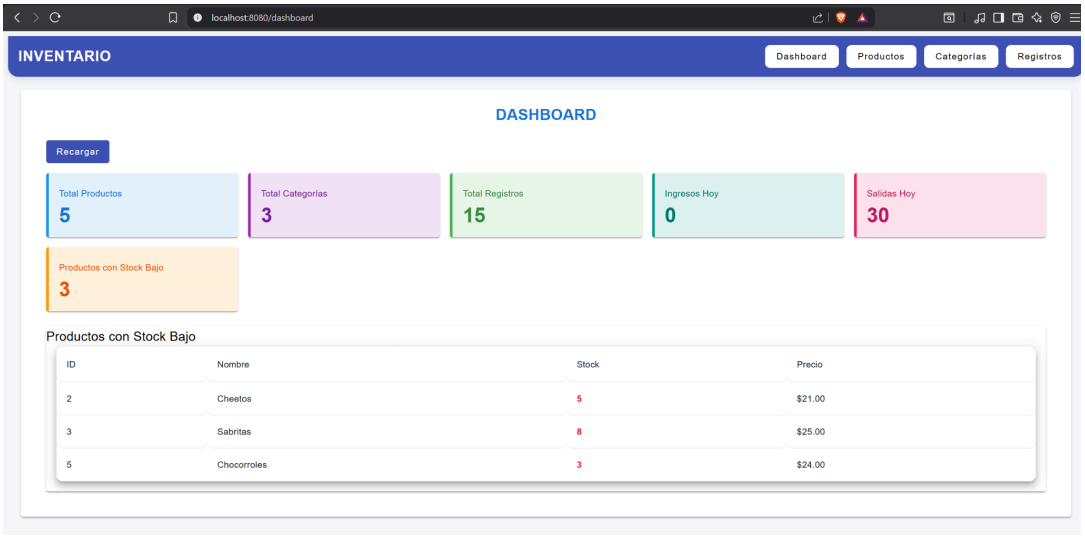
Stock actualizado después de la salida de 30 de stock., ahora solo le quedan 5.

FRONTEND

Al frontend que es la interfaz gráfica donde el usuario podrá utilizar las operaciones CRUD de manera visual se podrá acceder desde http://localhost:8080/ , donde se desplegará la página web.

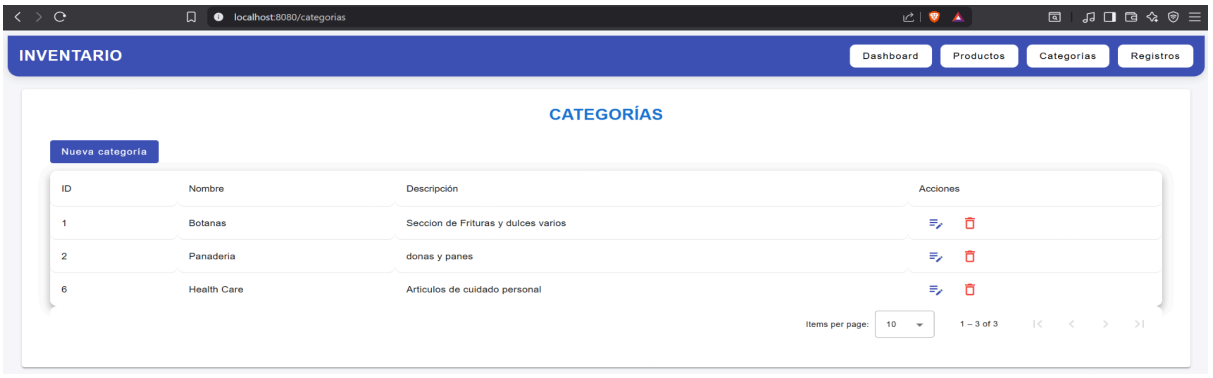
En esta página se encuentran varios apartados como Dashboard que muestra métricas básicas de nuestro inventario, Categorías donde se muestran las categorías que tenemos y nos deja agregar una nueva o modificarla o eliminarla.

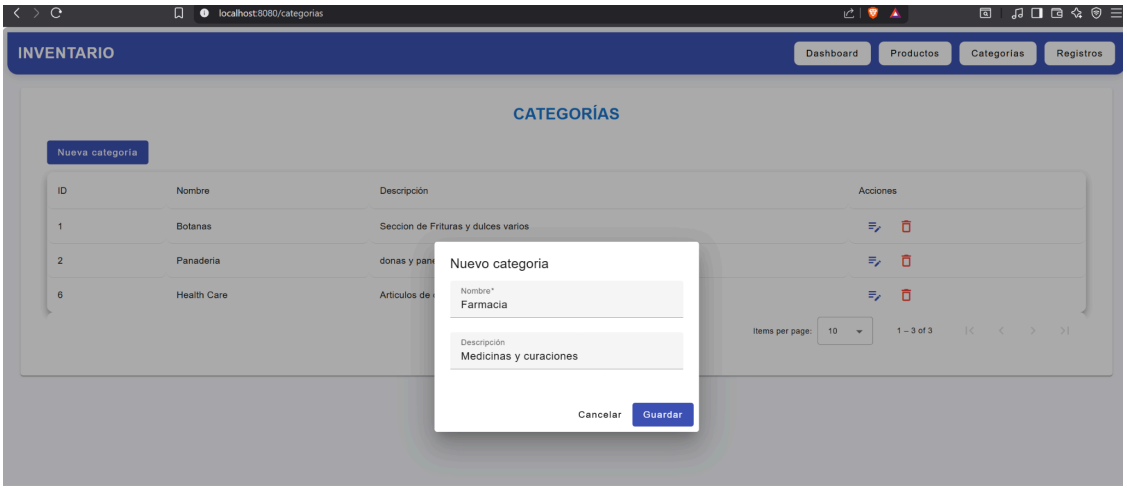
Productos que nos muestra la tabla de productos con varios filtros disponibles y también las operaciones CRUD de manera visual usando formularios, y el registro que nos muestra las entradas y salidas de stock con filtros y paginación.

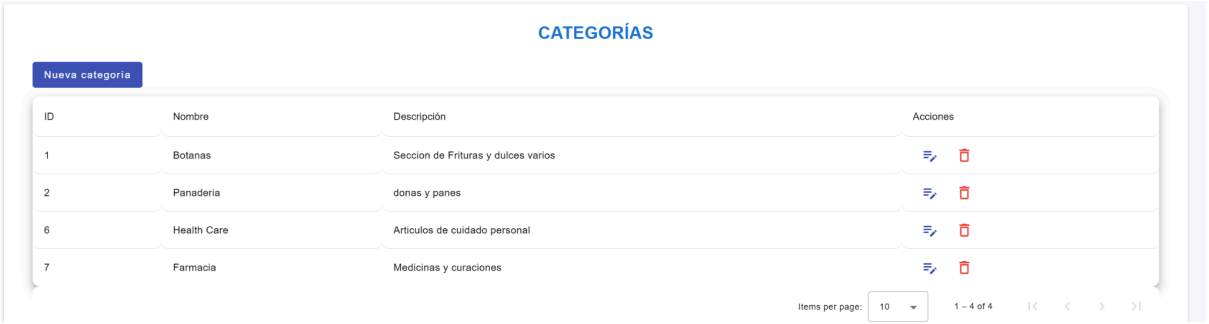


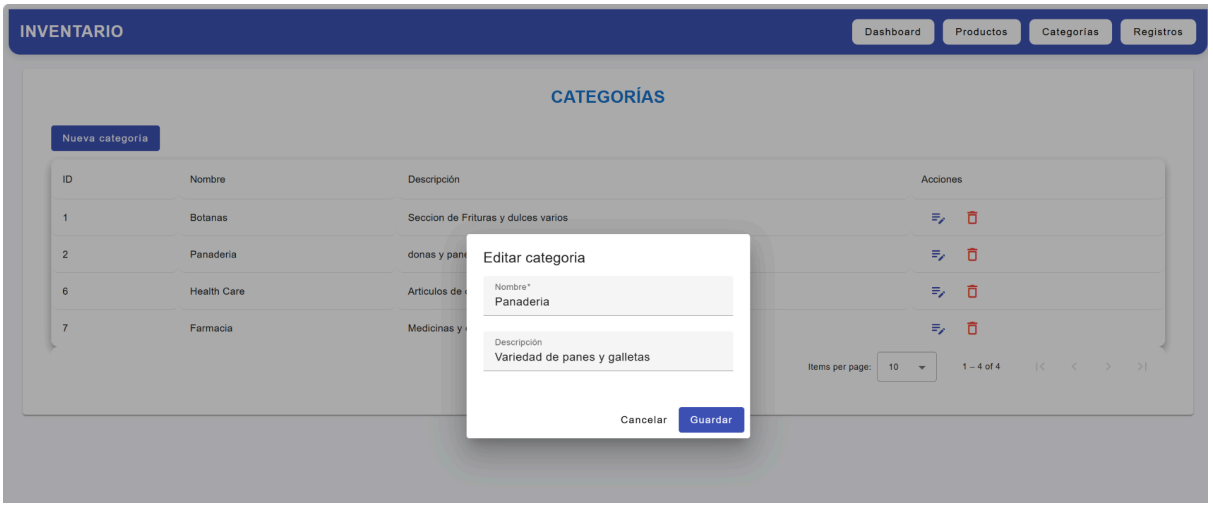
En el dashboard podemos ver una tabla con productos con bajo stock y métricas como el total de productos, categorías, registros , la cantidad de productos con bajo stock y la cantidad de ingresos y salidas de stock del día.

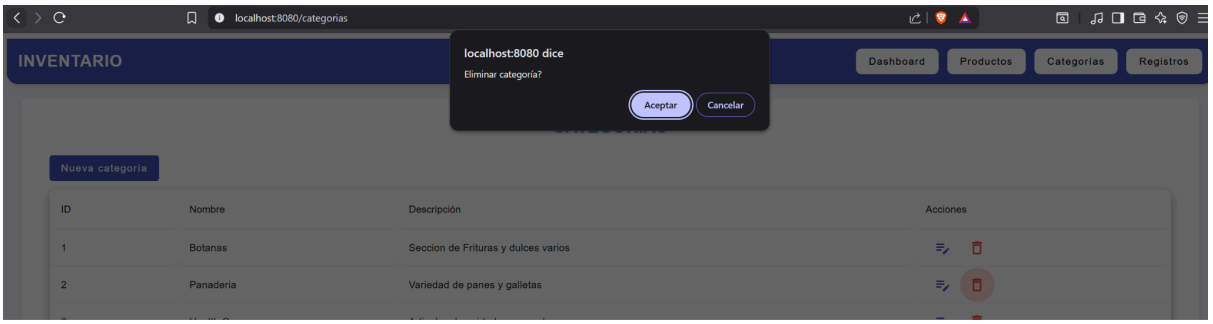
Tenemos el apartado de categorías donde se muestran las categorías que tenemos y nos permite agregar nuevas categorías, editarlas y también eliminarlas (siempre que no tenga productos).







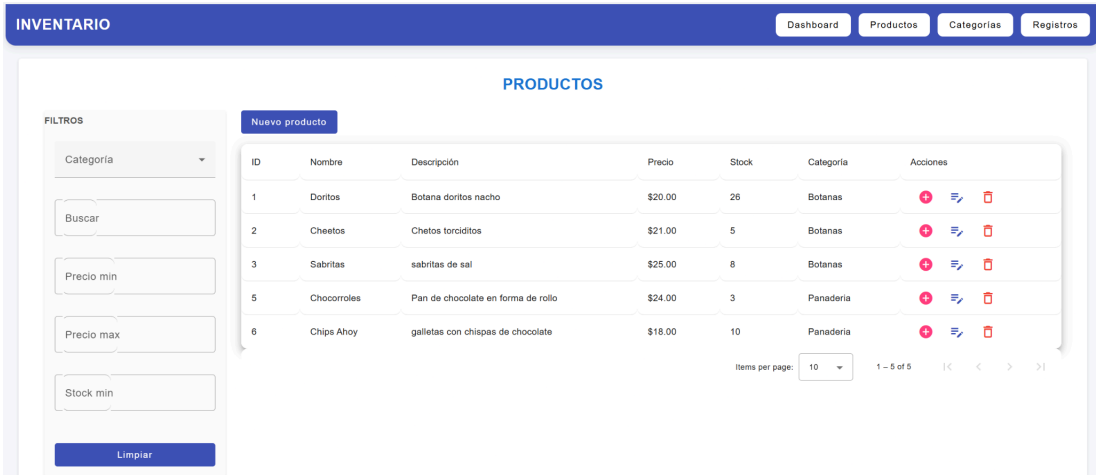


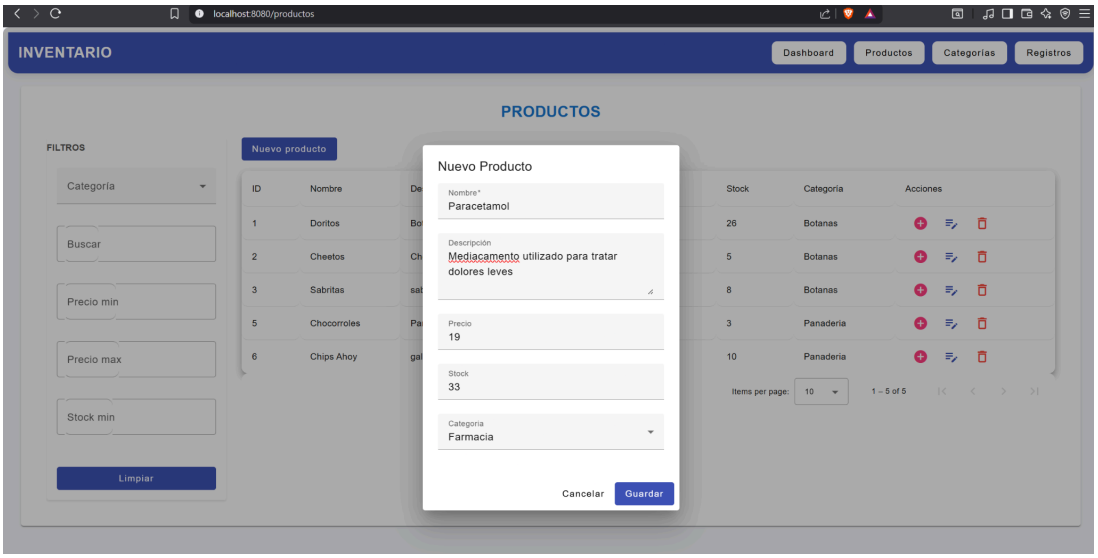


En el apartado de productos podemos encontrar una tabla de todos nuestros productos donde vemos id, nombre, descripción, precio, stock, categoría y un apartado de las acciones disponibles donde se puede agregar o quitar stock (creando a la vez el registro) , editar y eliminar el producto.

Así también tenemos un botón para agregar nuevos productos, y del lado izquierdo tenemos un apartado de filtros donde podemos buscar por nombre. precio minimo o máximo así como por cantidad de stock , además de un botón para limpiar filtros , además de la paginación de la tabla.

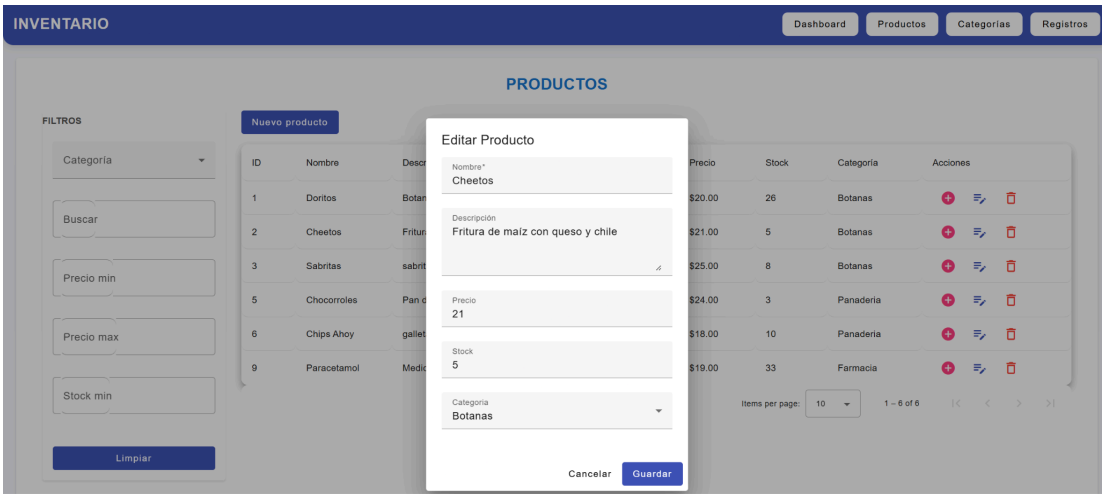
Agregar nuevo producto:





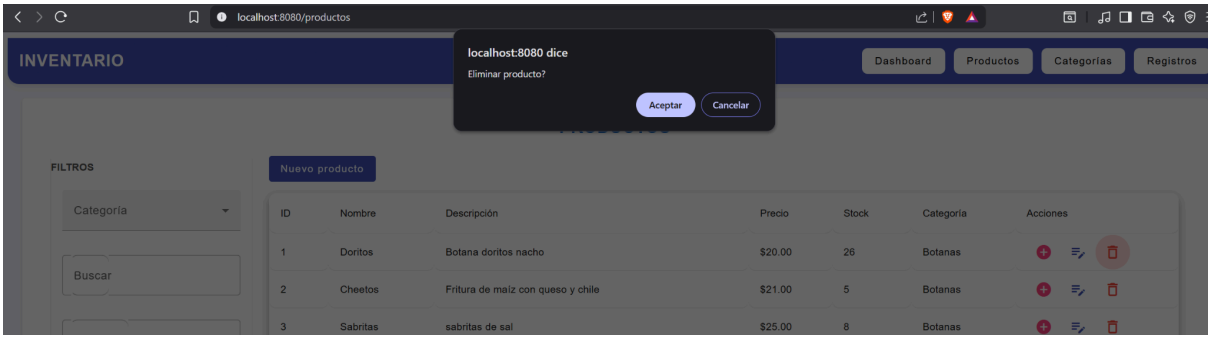
Editar producto:

9	Paracetamol	Medicamento utilizado para tratar dolores leves	$19.00	33	Farmacia	

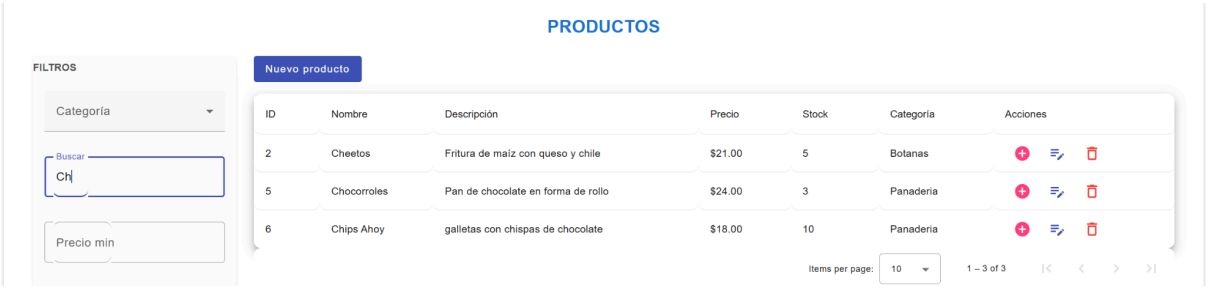


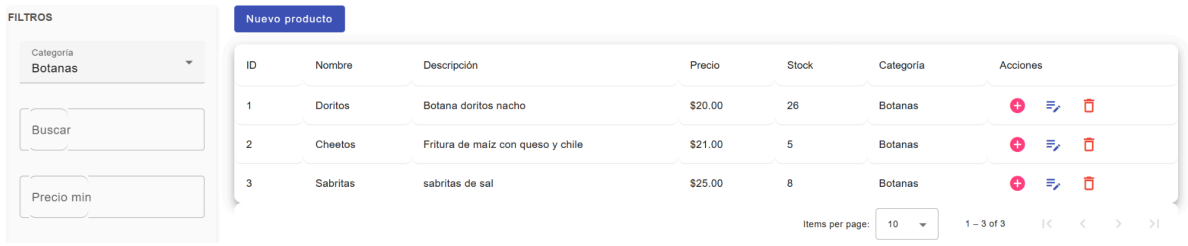
2	Cheetos	Fritura de maíz con queso y chile	$21.00	5	Botanas	

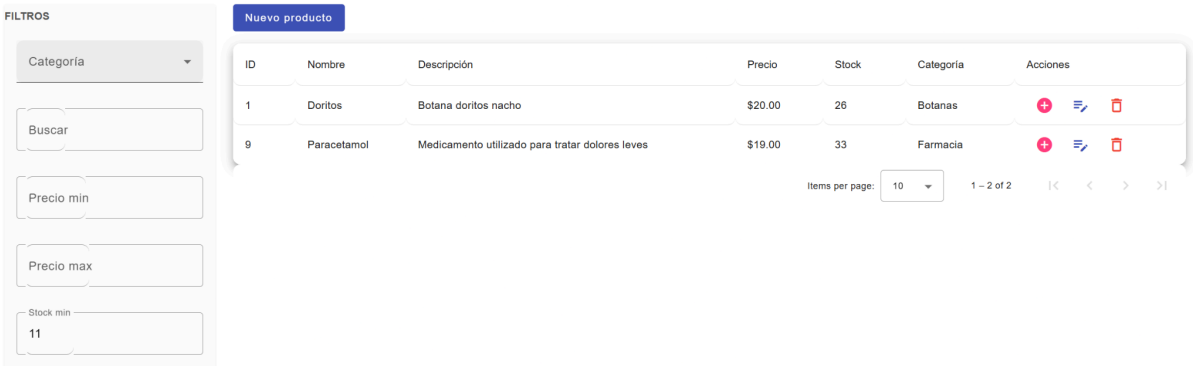
Eliminar Producto:

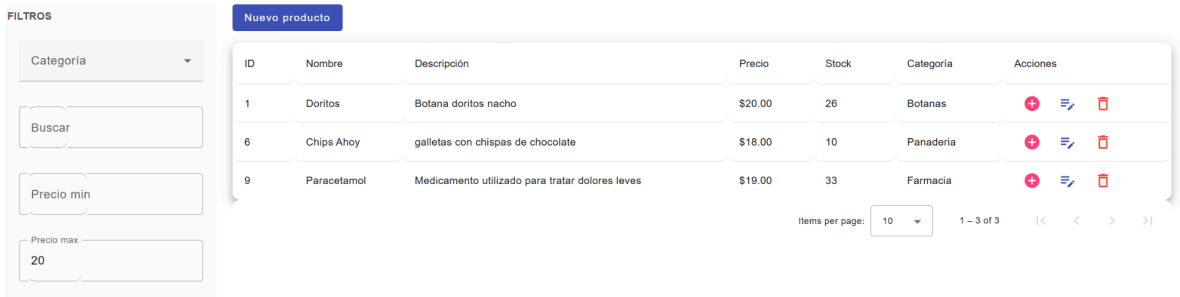


Filtrado de productos:

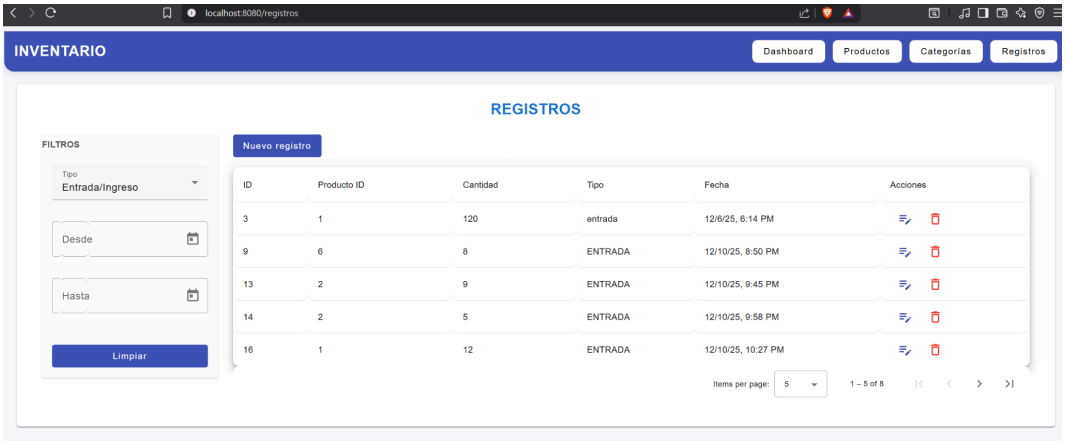


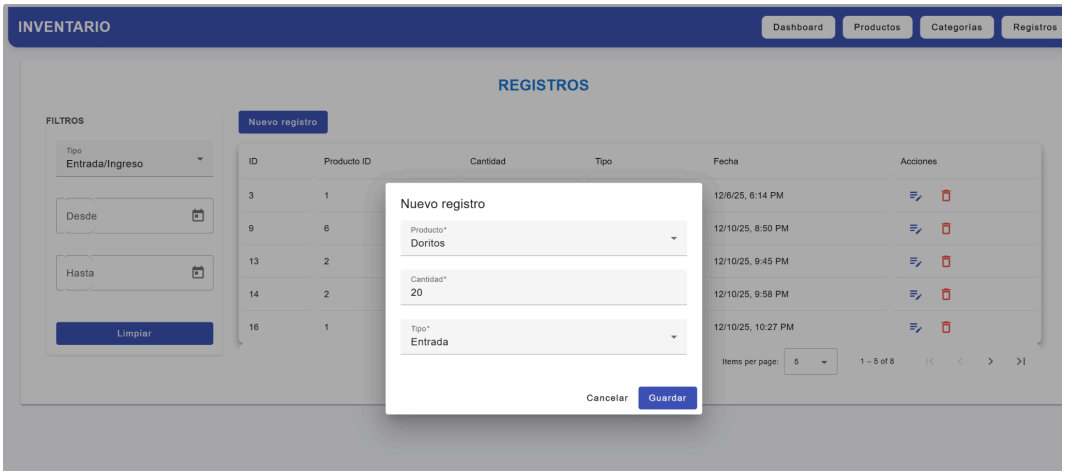


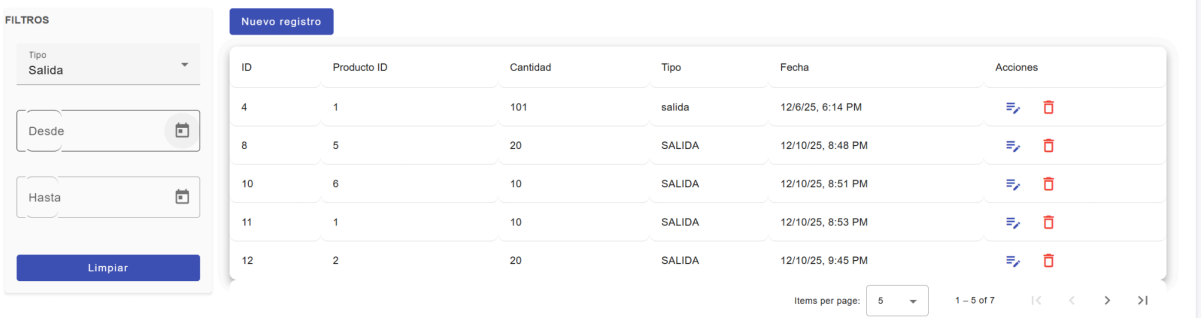


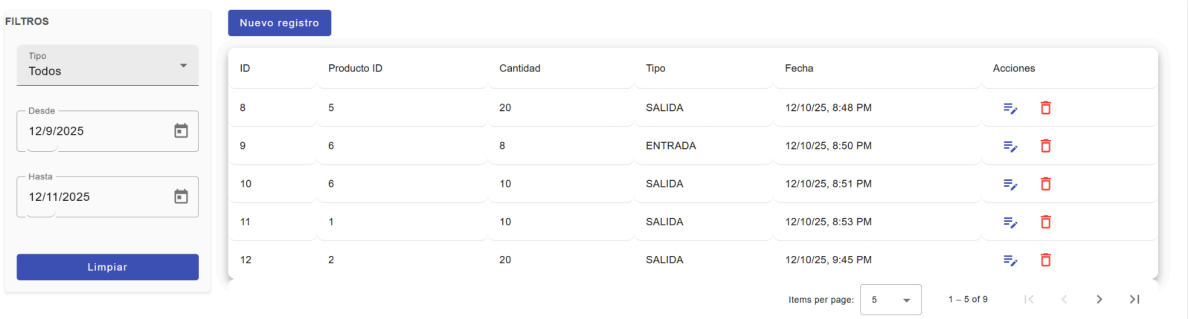


El apartado de registros nos muestra una tabla de los registros donde se ve el tipo de movimiento y la fecha del mismo, se cuentan con las acciones CRUD asi como con un filtrado por fecha y por tipo de movimiento.









CONCLUSIÓN

Este proyecto final de sistema de inventario me sirvió para integrar y aplicar conocimientos de varias UEA's que he cursado a lo largo de la carrera como sistemas distribuidos, bases de datos, web estático y ahora web dinámico.

Además este proyecto me sirvió para explorar más a fondo un perfil full-stack ya que estuve combinando tanto la parte visual e interactiva del frontend como la lógica, reglas de negocio del backend. Trabajar con ambos lados al mismo tiempo me ayudó a entender mejor cómo se comunican, cómo se diseñan flujos limpios entre capas y qué responsabilidades tiene cada sección del sistema. En conclusión fue una experiencia que reforzó mis habilidades y me dio una visión más completa de lo que implica desarrollar software profesional de principio a fin.

URL de proyecto en GITHUB: https://github.com/MarinoX/proyecto_web_dinamico