



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.

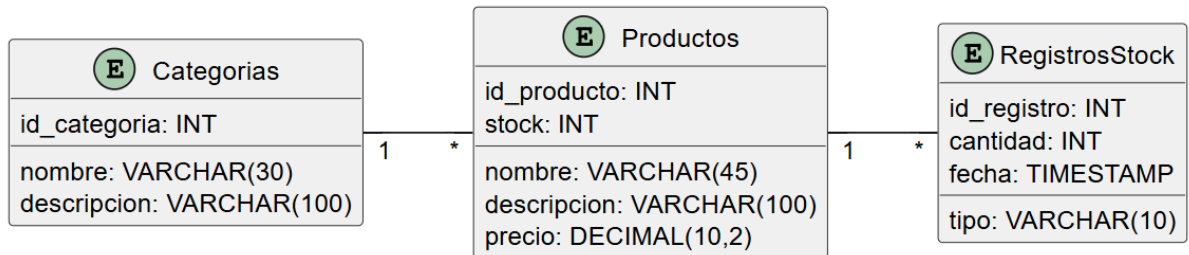
LENGUAJES: Java, Typescript, Javascript, HTML, CSS.

BASE DE DATOS : MySQL.

OTROS: Swagger, Github(https://github.com/MarinoX/proyecto_web_dinamico)

ARQUITECTURA

Estructura de la Base de Datos:

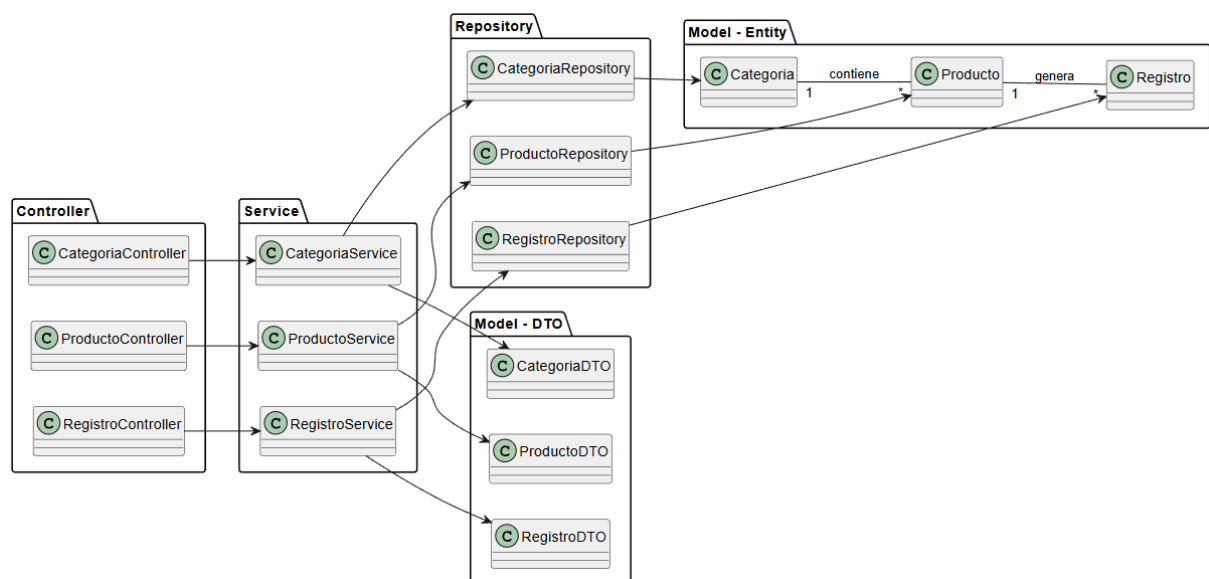


Mi base de datos hecha en SQL está organizada alrededor de la tabla **Categorías** ya que esta funciona como el punto de clasificación principal. Cada categoría tiene un nombre y una descripción, y sirve para agrupar productos de forma ordenada.

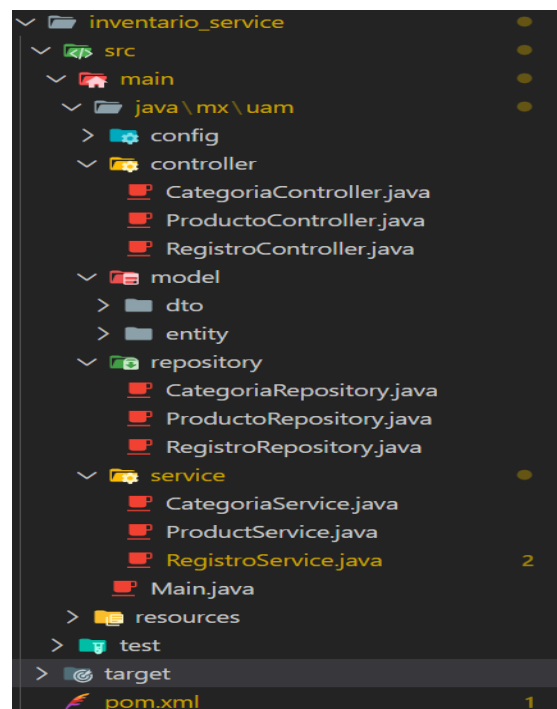
Luego tengo la tabla de **Productos** que es pues la parte principal del inventario. Aquí se almacena todo lo importante: nombre, descripción, precio y el stock actual. Cada producto está ligado a una categoría mediante una relación uno a muchos, lo que ayuda a evitar el duplicar información y mantiene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

```
CategoriaController.java X
22
23 @RestController
24 @RequestMapping("/categorias")
25 @Tag(name = "Categoria", description = "Operaciones CRUD relacionadas con las categorías de productos")
26 public class CategoriaController {
27     @Autowired
28     private CategoriaService categoriaService;
29
30     @GetMapping
31     @Operation(summary = "Obtener todas las categorías", description = "Recupera una lista de todas las categorías de productos")
32     public List<CategoriaDTO> findAll() {
33         return categoriaService.findAll();
34     }
35
36     @GetMapping("/{id}")
37     @Operation(summary = "Obtener categoría por ID", description = "Retorna una categoría por su ID")
38     public ResponseEntity<CategoriaDTO> getById(@Parameter(description = "ID de la categoría") @PathVariable Integer id) {
39         CategoriaDTO dto = categoriaService.findById(id);
40         if (dto == null) {
```

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 se puede realizar el movimiento automáticamente actualiza el stock del producto.

```
CategoriaService.java X
14 import mx.uam.repository.ProductoRepository;
15
16 @Service
17 public class CategoriaService {
18     @Autowired
19     private CategoriaRepository categoriarepository;
20
21     @Autowired
22     private ProductoRepository productoRepository;
23
24     public List<CategoriaDTO> findAll(){
25         return categoriarepository.findAll().stream()
26             .map(this::toDTO)
27             .collect(Collectors.toList());
28     }
29
30     public CategoriaDTO findById(Integer id) {
31         if (id == null) {
```

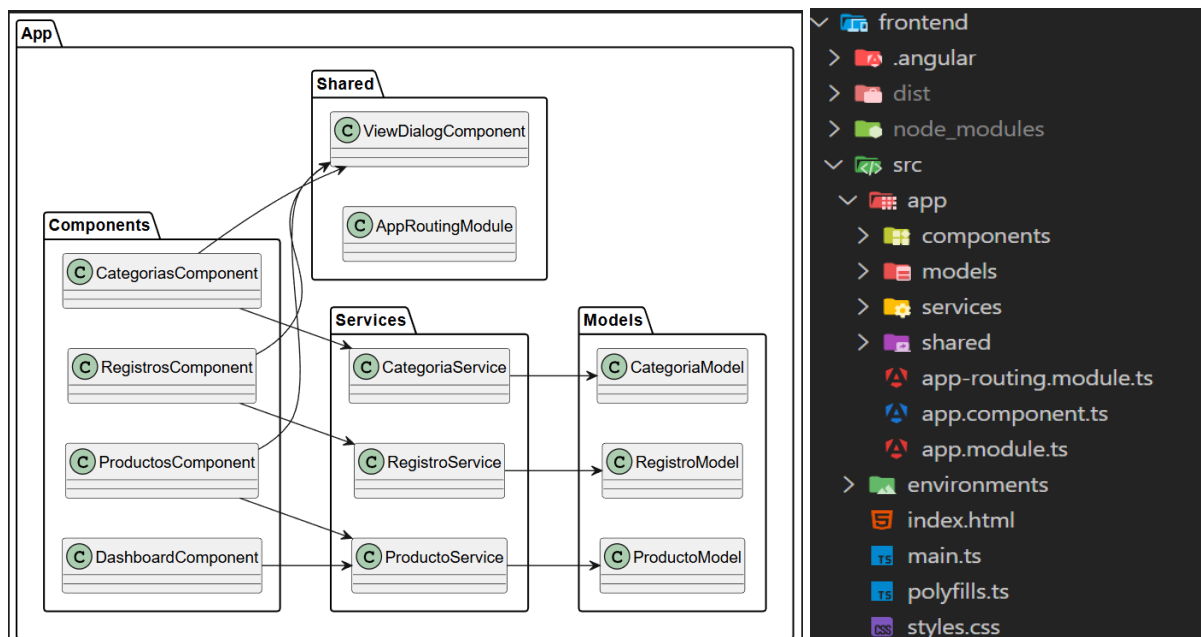
Los repository contienen las queries y operaciones CRUD necesarias y manejan el acceso directo a la base de datos mediante Spring Data JPA. Los modelos están divididos en Entities (Categoria, Producto, Registro) que mapean directamente las tablas de la base de datos y los DTO que son los objetos que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

```
CategoriaRepository.java X
1 import org.springframework.data.jpa.repository.JpaRepository;
2
3
4
5 import mx.uam.model.entity.Categoria;
6
7 public interface CategoriaRepository extends JpaRepository<Categoria, Integer> {
8
9 }
10
```

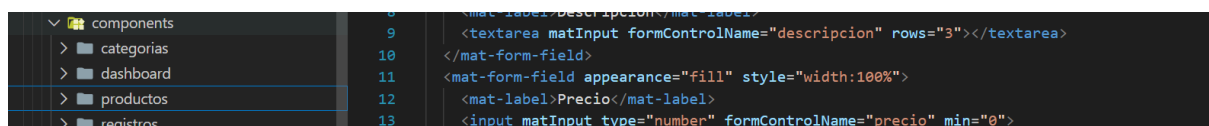
```
CategoriaDTO.java X
1 package mx.uam.model.dto;
2
3 public class CategoriaDTO {
4     private Integer id;
5     private String nombre;
6     private String descripcion;
7
8     public Integer getId() {
9         return id;
10    }
11 }
```

```
Categoria.java X
1 import javax.persistence.Entity;
2
3
4
5 @Entity
6 public class Categoria {
7     @Id
8     @GeneratedValue(strategy = GenerationType.IDENTITY)
9     private Integer id;
10    private String nombre;
11    private String descripcion;
12
13    @OneToMany(mappedBy = "categoria", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
14    private List<Producto> productos;
15 }
```

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, CategoríasComponent que gestiona las categorías, ProductosComponent para la administración de los productos y RegistrosComponent para visualizar y crear registros de entrada y salida de stock, aquí 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.

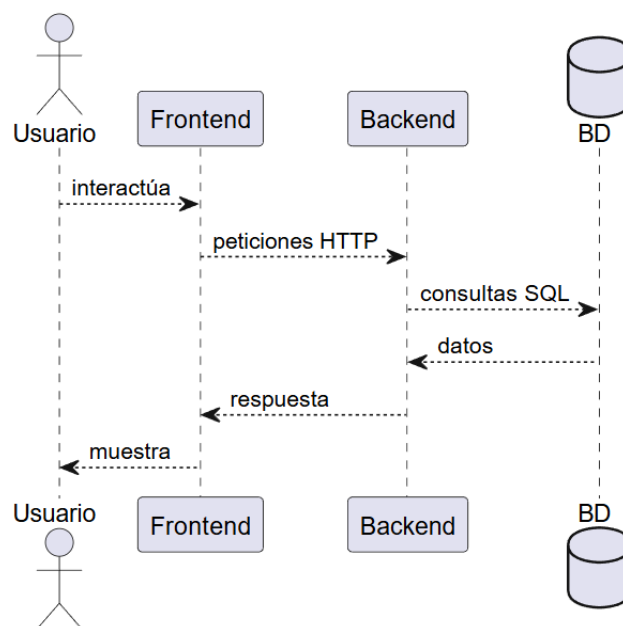
```

13  getAll(): Observable<ProductoDTO[]> {
14      return this.http.get<ProductoDTO[]>(this.base).pipe(catchError(this.handleError));
15  }
16  getById(id: number) {
17      return this.http.get<ProductoDTO>(`${this.base}/${id}`).pipe(catchError(this.handleError));

```

La carpeta Shared contiene componentes y módulos reutilizables en toda la aplicación, como el ViewDialogComponent y el AppRoutingModuleModule 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.

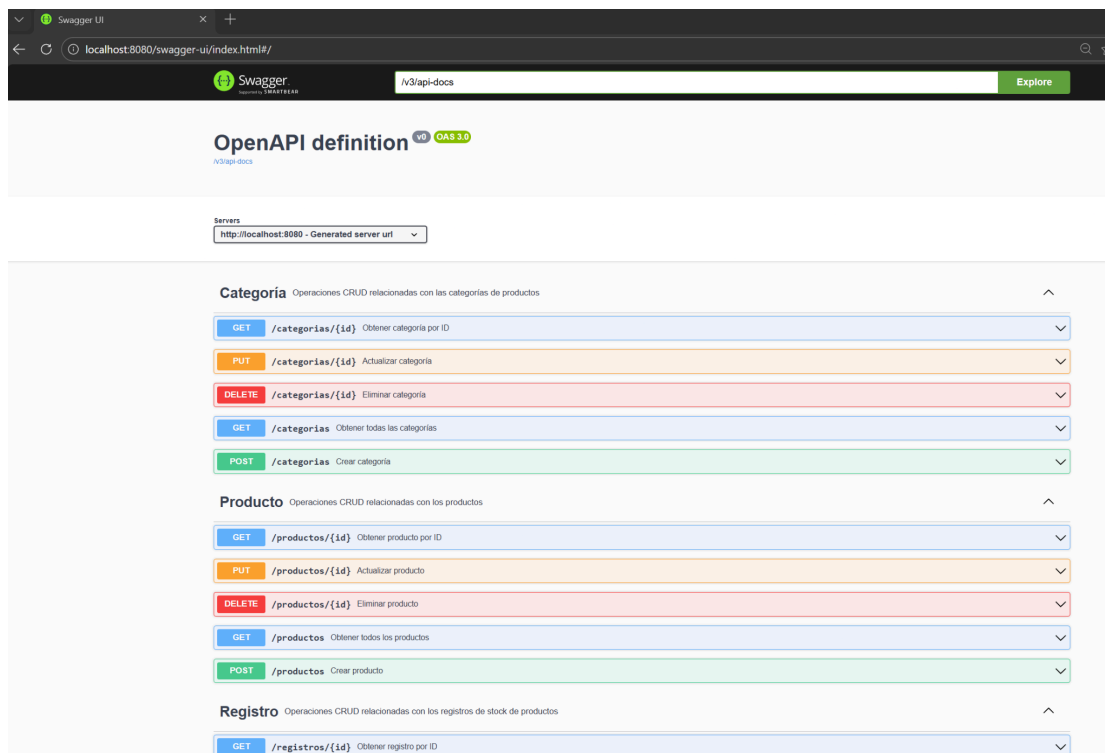
Para la documentación de mi backend utilicé Swagger a través de la dependencia el cual genera de forma automática toda la documentación basada en las anotaciones de mis controllers.

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

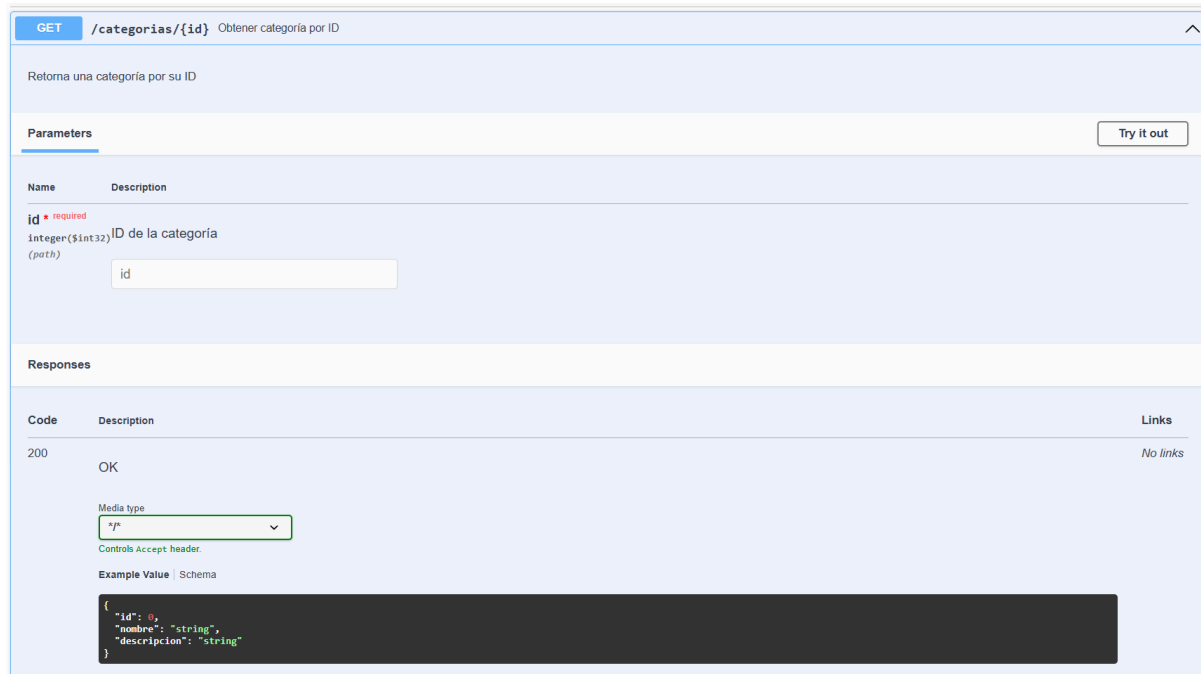
A la documentación se puede acceder desde el navegador y nos permite visualizar los endpoints, sus parámetros necesarios y nos deja probar las peticiones directamente sin necesidad de usar Postman, CURL o el mismo frontend.

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.



Swagger UI for the endpoint `GET /categorias/{id}` titled "Obtener categoría por ID". The description states: "Retorna una categoría por su ID".

Parameters

Name	Description
<code>id</code> * required <code>integer(\$int32)</code> (path)	ID de la categoría

Input field for `id`:

Responses

Code	Description	Links
200	OK	No links

Media type:

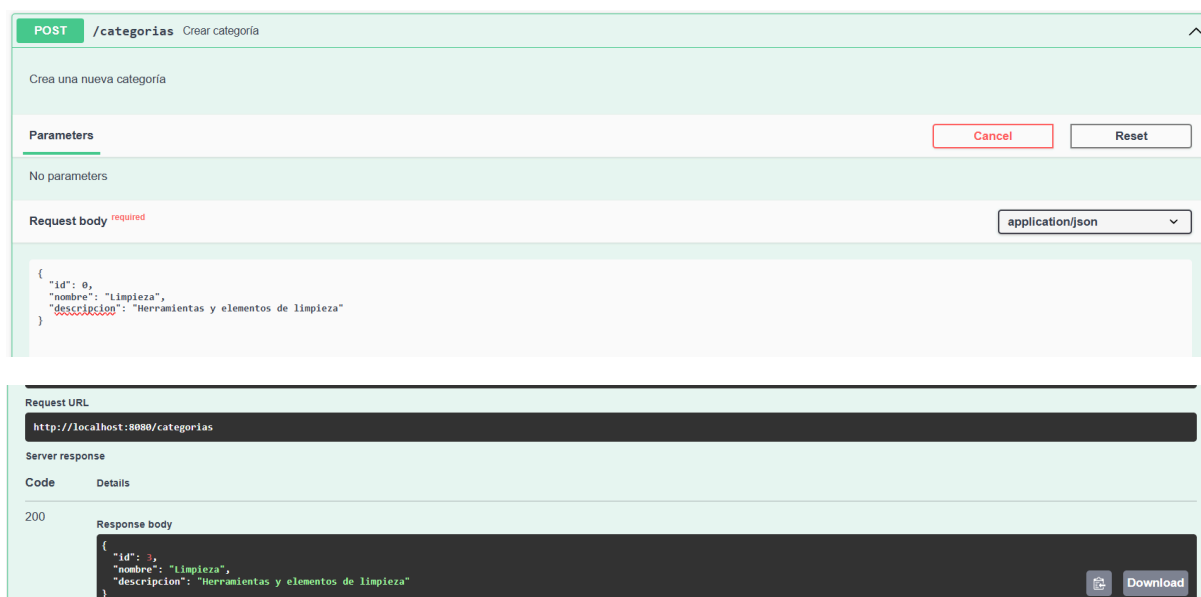
Controls Accept header.

Example Value | Schema

```
{
  "id": 0,
  "nombre": "limpieza",
  "descripcion": "string"
}
```

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

1.- Crear Categoría.



Swagger UI for the endpoint `POST /categorias` titled "Crear categoría". The description states: "Crea una nueva categoría".

Parameters

No parameters

Request body required

Media type:

```
{
  "id": 0,
  "nombre": "limpieza",
  "descripcion": "Herramientas y elementos de limpieza"
}
```

Request URL

`http://localhost:8080/categorias`

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 0, "nombre": "limpieza", "descripcion": "Herramientas y elementos de limpieza" }</pre>

Download

2.- Obtener todas las categorías.

GET

/categorias

Obtener todas las categorías

Recupera una lista de todas las categorías de productos disponibles en el inventario.

Parameters

Cancel

No parameters

Execute

Clear

Responses

Curl

curl -X 'GET' \n'http://localhost:8080/categorias' \n-H 'accept: */*'

Request URL

http://localhost:8080/categorias

Server response

Code

Details

200

Response body

{\n {\n "id": 1,\n "nombre": "Botanas",\n "descripcion": "Papas y golosinas varias"\n },\n {\n "id": 2,\n "nombre": "Panaderia",\n "descripcion": "donas y panes"\n },\n {\n "id": 3,\n "nombre": "Limpieza",\n "descripcion": "Herramientas y elementos de limpieza"\n }\n}

Download

3.- Obtener categoría por id.

GET

/categorias/{id}

Obtener categoría por ID

Retorna una categoría por su ID

Parameters

Cancel

Name	Description
id * required	
integer(\$int32)	ID de la categoría
(path)	
	1

Execute

Clear

Responses

Curl

curl -X 'GET' \n'http://localhost:8080/categorias/1' \n-H 'accept: */*'

Request URL

http://localhost:8080/categorias/1

Server response

Code

Details

200

Response body

{\n "id": 1,\n "nombre": "Botanas",\n "descripcion": "Seccion de Frituras y dulces varios"\n}

Download

4.- Actualizar Categoría.

PUT /categorias/{id} Actualizar categoría

Actualiza una categoría existente

Parameters Cancel Reset

Name	Description
id * required integer(\$int32) (path)	ID de la categoría
<input type="text" value="4"/>	

Request body * required application/json

```
{  "description": "Artículos y comida para perros"}
```

Code **Details**

200

Response body

```
{  "id": 4,  "nombre": "Rascotas",  "descripcion": "Artículos y comida para perros"}
```

Download

5.- Eliminar Categoría

DELETE /categorias/{id} Eliminar categoría

Elimina una categoría por su ID

Parameters Cancel

Name	Description
id * required integer(\$int32) (path)	ID de la categoría
<input type="text" value="4"/>	

Code **Details**

204
Undocumented

Response headers

```
connection: keep-alive
date: Fri, 12 Dec 2025 03:47:24 GMT
keep-alive: timeout=60
vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
```

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

200

Response body

```
[  {    "id": 1,    "nombre": "Botanas",    "descripcion": "Sección de Frituras y dulces varios"  },  {    "id": 2,    "nombre": "Panadería",    "descripcion": "donas y panes"  },  {    "id": 3,    "nombre": "Health Care",    "descripcion": "Artículos de cuidado personal"  }]
```

Download

6.- Obtener producto por id.

GET /productos/{id} Obtener producto por ID

Retorna un producto por su ID

Parameters

Name	Description
id * required	
integer(\$int32)	ID del producto
(path)	

5

Execute Clear

200

Response body

```
{
  "id": 5,
  "nombre": "Chocorroles",
  "descripcion": "roles de chocolate",
  "precio": 24,
  "stock": 3,
  "categoria": {
    "id": 2,
    "nombre": "Panaderia",
    "descripcion": "donas y panes"
  }
}
```

Download

7.- Actualizar producto.

Antes de actualizar:

```
{
  "id": 5,
  "nombre": "Chocorroles",
  "descripcion": "roles de chocolate",
  "precio": 24,
  "stock": 3,
  "categoria": {
    "id": 2,
    "nombre": "Panaderia",
    "descripcion": "donas y panes"
  }
}
```

Usando el endpoint de actualizar producto por id

PUT /productos/{id} Actualizar producto

Actualiza un producto existente

Parameters

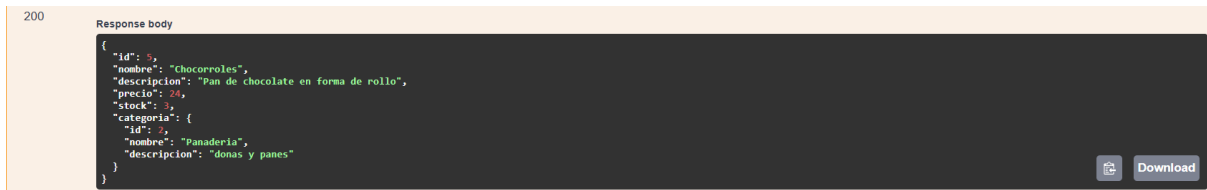
Name	Description
id * required	
integer(\$int32)	ID del producto
(path)	

5

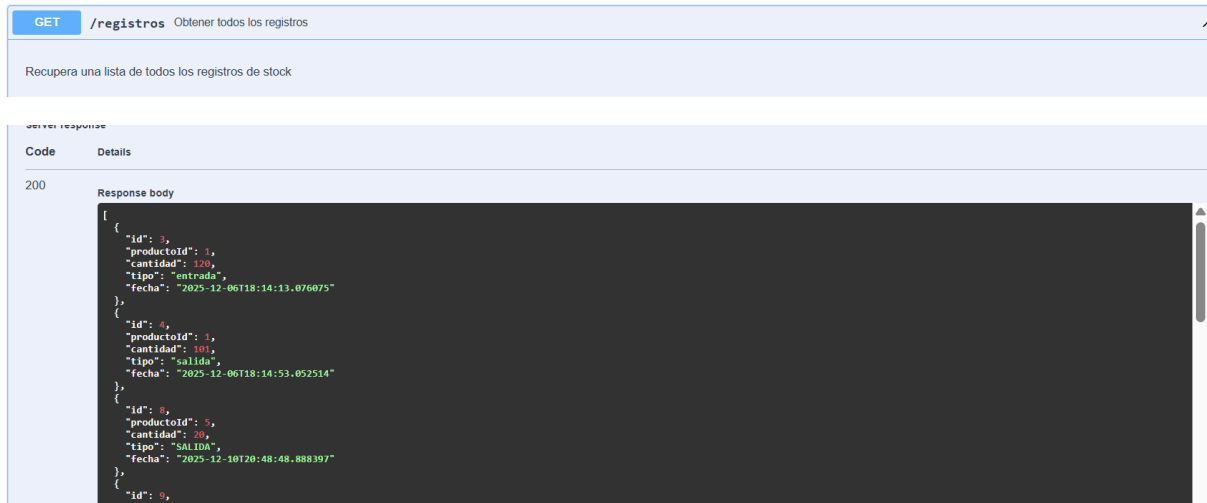
Request body required

application/json

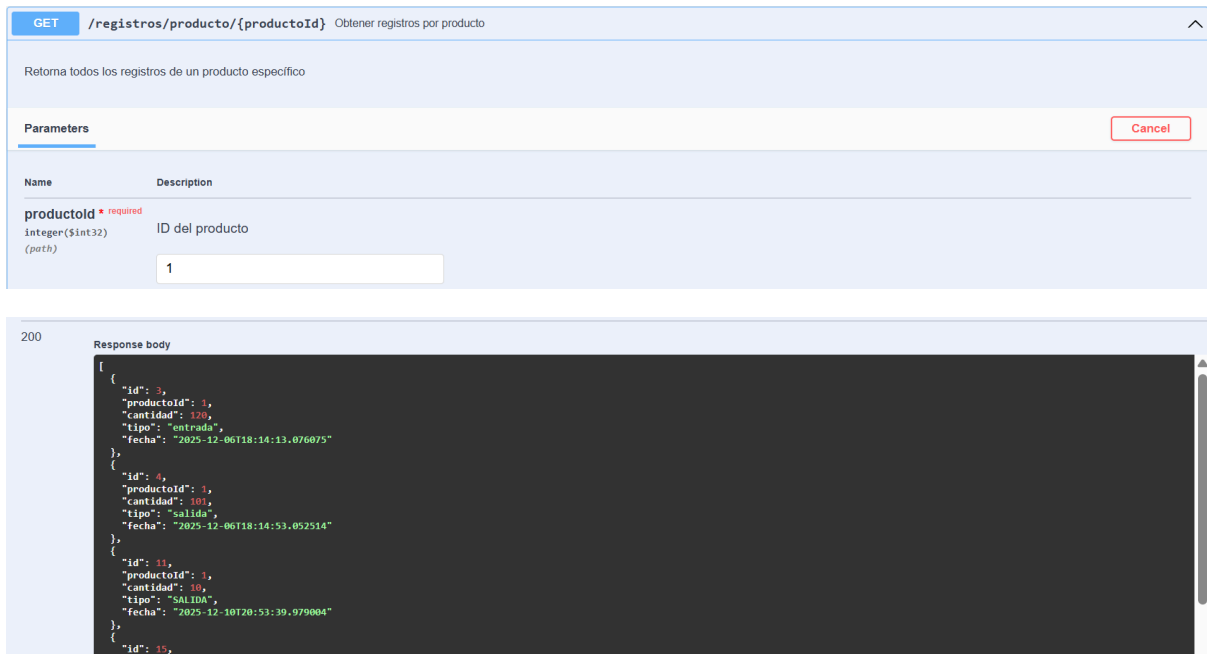
```
{
  "descripcion": "Pan de chocolate en forma de rollo"
}
```



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:

```
200
Response body
{
  "id": 2,
  "nombre": "Cheetos",
  "descripcion": "Chetos torcidos",
  "precio": 22,
  "stock": 24,
  "categoria": {
    "id": 1,
    "nombre": "Botanas",
    "descripcion": "Seccion de Frituras y dulces varios"
  }
}
```

Registro de entrada.

POST /registros Crear registro

Crema un nuevo registro de stock

Parameters Cancel Reset

No parameters

Request body required application/json

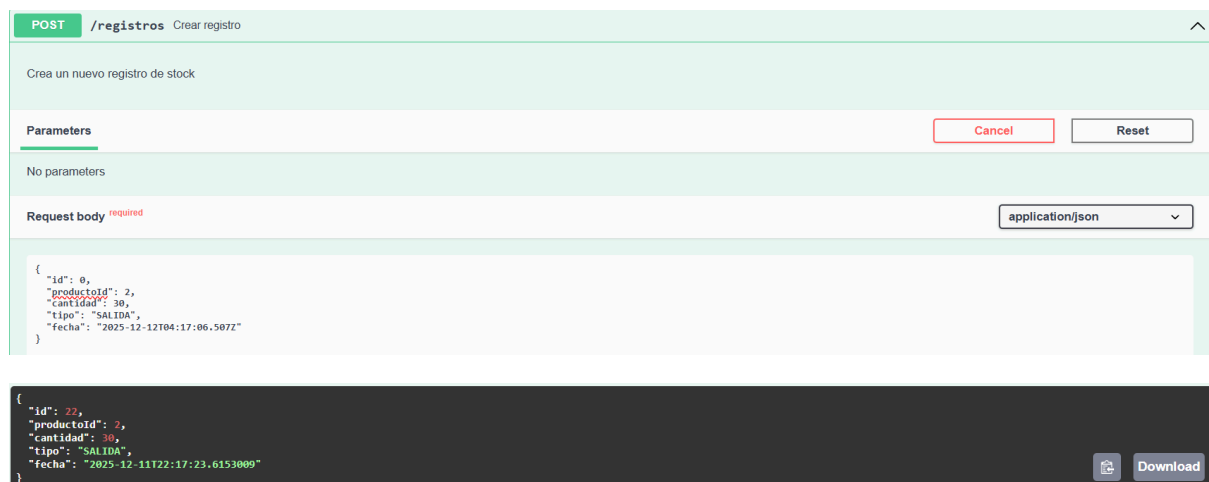
```
{
  "id": 0,
  "productoId": 2,
  "cantidad": 11,
  "tipo": "ENTRADA",
  "fecha": "2025-12-12T04:12:23.638Z"
}
```

```
200
Response body
{
  "id": 21,
  "productoId": 2,
  "cantidad": 11,
  "tipo": "ENTRADA",
  "fecha": "2025-12-11T22:14:48.3372149"
}
```

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

```
{
  "id": 2,
  "nombre": "Cheetos",
  "descripcion": "Chetos torcidos",
  "precio": 22,
  "stock": 35,
  "categoria": {
    "id": 1,
    "nombre": "Botanas",
    "descripcion": "Seccion de Frituras y dulces varios"
  }
}
```

Registro de Salida.



POST /registros Crear registro

Crea un nuevo registro de stock

Parameters Cancel Reset

No parameters

Request body required application/json

```
{
  "id": 0,
  "productId": 2,
  "cantidad": 30,
  "tipo": "SALIDA",
  "fecha": "2025-12-12T04:17:06.507Z"
}
```

```
{
  "id": 22,
  "productId": 2,
  "cantidad": 30,
  "tipo": "SALIDA",
  "fecha": "2025-12-11T22:17:23.6153009"
}
```

Download

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

```
{
  "id": 2,
  "nombre": "Cheetos",
  "descripcion": "chetos torciditos",
  "precio": 21,
  "stock": 5,
  "categoria": {
    "id": 1,
    "nombre": "Botanas",
    "descripcion": "Seccion de Frituras y dulces varios"
  }
}
```

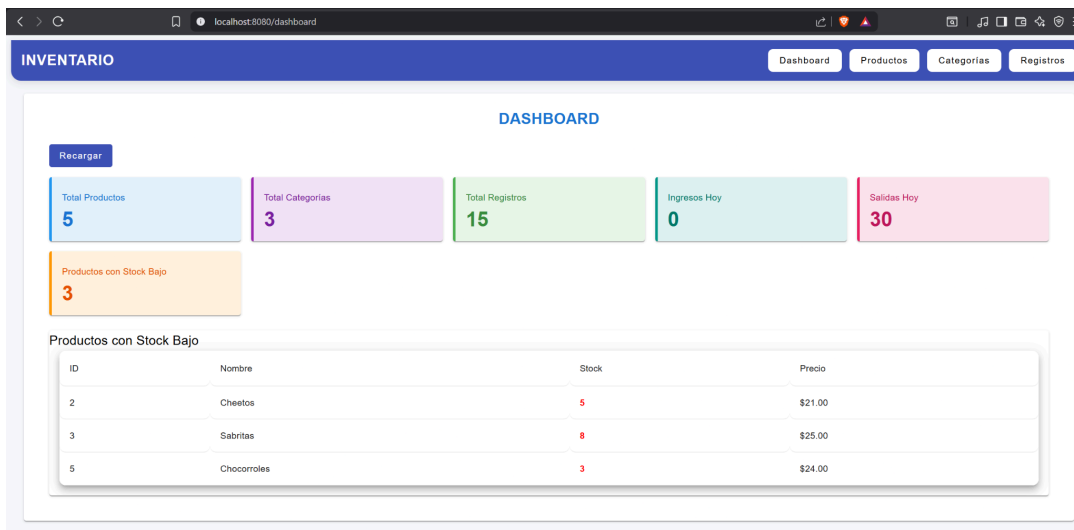
Download

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.

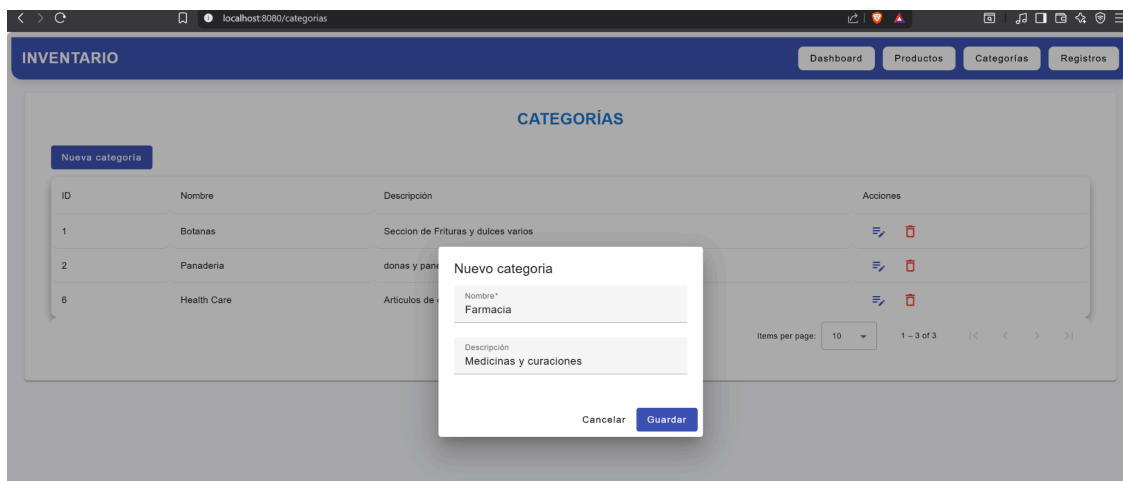
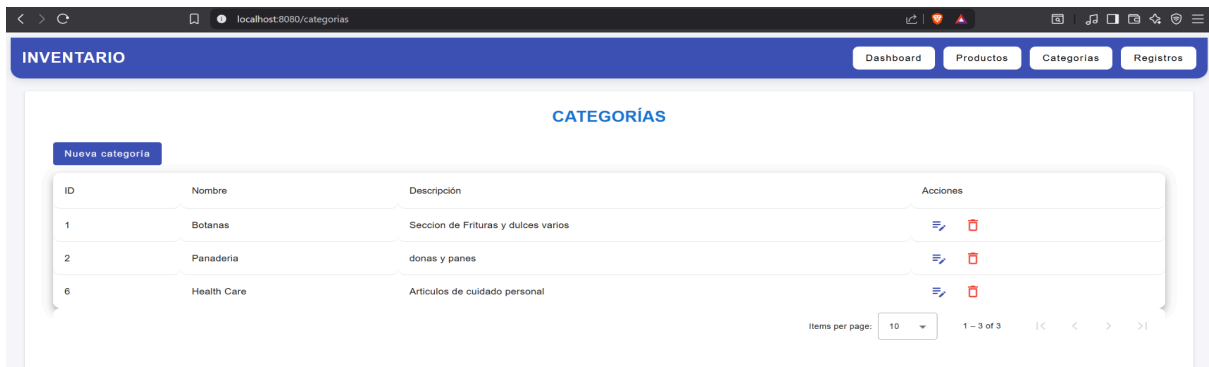
En esta página se encuentran varios apartados como Dashboard que muestra métricas básicas de nuestro inventario, Categorías donde se muestran las categorías que tenemos y nos deja agregar una nueva o modificarla o eliminarla.









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).



CATEGORÍAS			
Nueva categoría			
ID	Nombre	Descripción	Acciones
1	Botanas	Seccion de Frituras y dulces varios	 
2	Panaderia	donas y panes	 
6	Health Care	Articulos de cuidado personal	 
7	Farmacia	Medicinas y curaciones	 









Items per page: 10 1 - 4 of 4

INVENTARIO

Dashboard Productos Categorías Registros

CATEGORÍAS

Nueva categoría

ID	Nombre	Descripción	Acciones
1	Botanas	Seccion de Frituras y dulces varios	 
2	Panaderia	donas y panes	 
6	Health Care	Articulos de cuidado personal	 
7	Farmacia	Medicinas y curaciones	 

Editar categoría

Nombre*

Panaderia

Descripción

Variedad de panes y galletas

Cancelar

Guardar

Items per page: 10 1 - 4 of 4

2 Panaderia Variedad de panes y galletas  

localhost:8080/categorias





INVENTARIO

localhost:8080 dice
Eliminar categoría?

Aceptar Cancelar

Dashboard Productos Categorías Registros

Nueva categoría

ID	Nombre	Descripción	Acciones
1	Botanas	Seccion de Frituras y dulces varios	 
2	Panaderia	Variedad de panes y galletas	 

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:

INVENTARIO

DashboardProductosCategoríasRegistros

PRODUCTOS

FILTROS

Categoría

Buscar

Precio min

Precio max

Stock min

Limpiar

Nuevo producto

ID	Nombre	Descripción	Precio	Stock	Categoría	Acciones
1	Doritos	Botana doritos nacho	\$20.00	26	Botanas	<div>+≡🗑</div>
2	Cheetos	Chetos torciditos	\$21.00	5	Botanas	<div>+≡🗑</div>
3	Sabritas	sabritas de sal	\$25.00	8	Botanas	<div>+≡🗑</div>
5	Chocorrollos	Pan de chocolate en forma de rollo	\$24.00	3	Panadería	<div>+≡🗑</div>
6	Chips Ahoy	galletas con chispas de chocolate	\$18.00	10	Panadería	<div>+≡🗑</div>

Items per page: 101 - 5 of 5<<<>>>

INVENTARIO

DashboardProductosCategoríasRegistros

PRODUCTOS

FILTROS

Categoría

Buscar

Precio min

Precio max

Stock min

Limpiar

Nuevo producto

Nuevo Producto

Nombre*

Paracetamol

Descripción

Medicamento utilizado para tratar dolores leves

Precio

19

Stock

33

Categoría

Farmacia

Cancelar

Guardar

Editar producto:

9ParacetamolMedicamento utilizado para tratar dolores leves\$19.0033Farmacia

+≡🗑

INVENTARIO

DashboardProductosCategoríasRegistros

PRODUCTOS

FILTROS

Categoría

Buscar

Precio min

Precio max

Stock min

Limpiar

Nuevo producto

Editar Producto

Nombre*

Cheetos

Descripción

Fritura de maíz con queso y chile

Precio

21

Stock

5

Categoría

Botanas

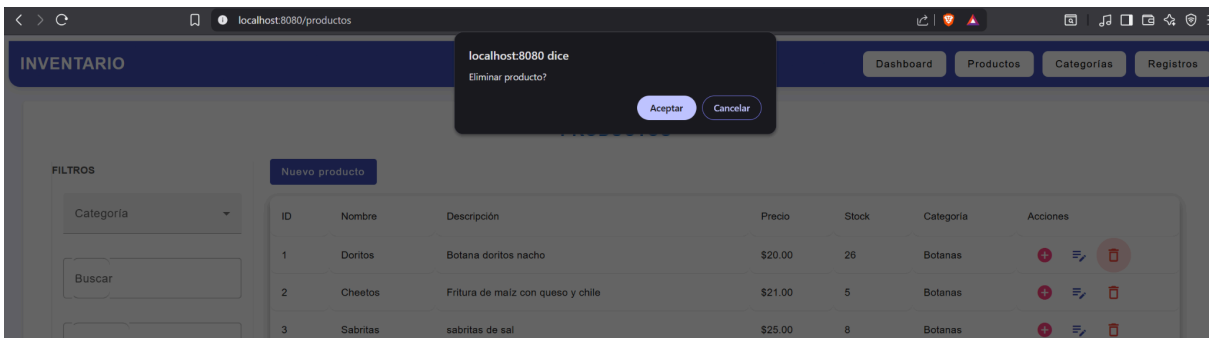
Cancelar

Guardar

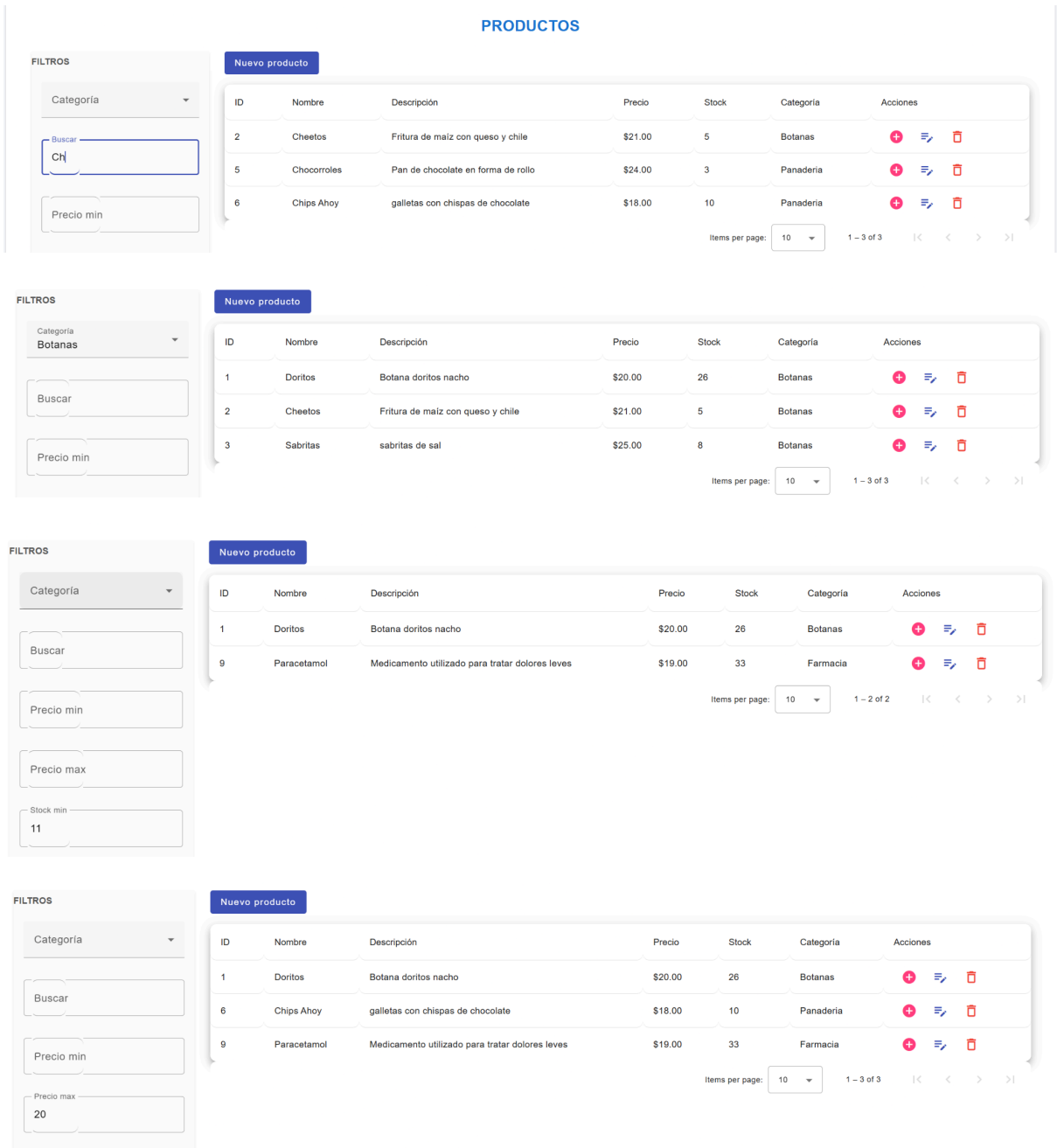
2CheetosFritura de maíz con queso y chile\$21.005Botanas

+≡🗑

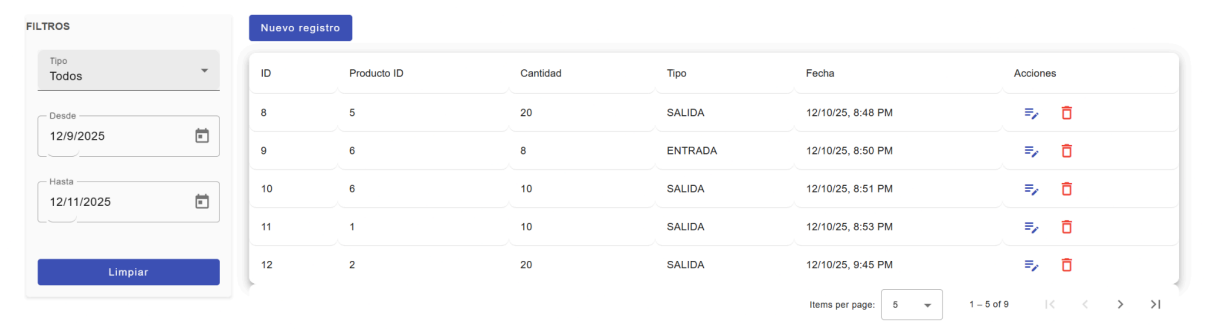
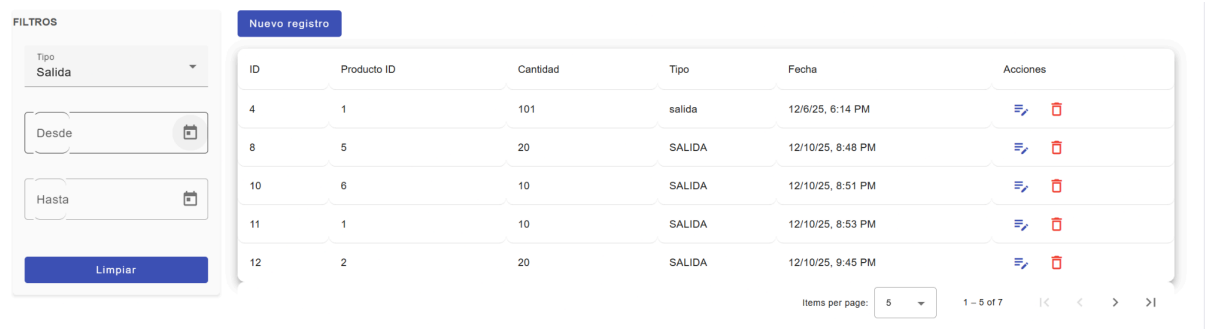
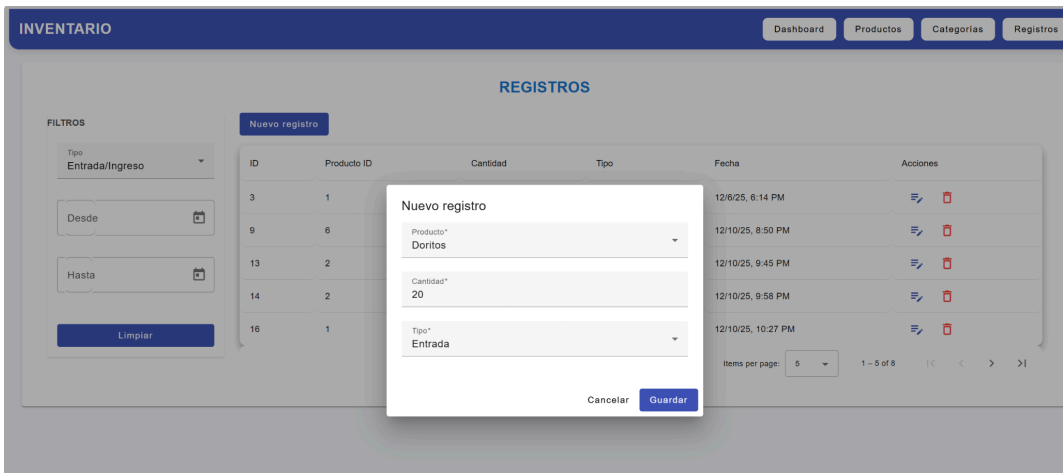
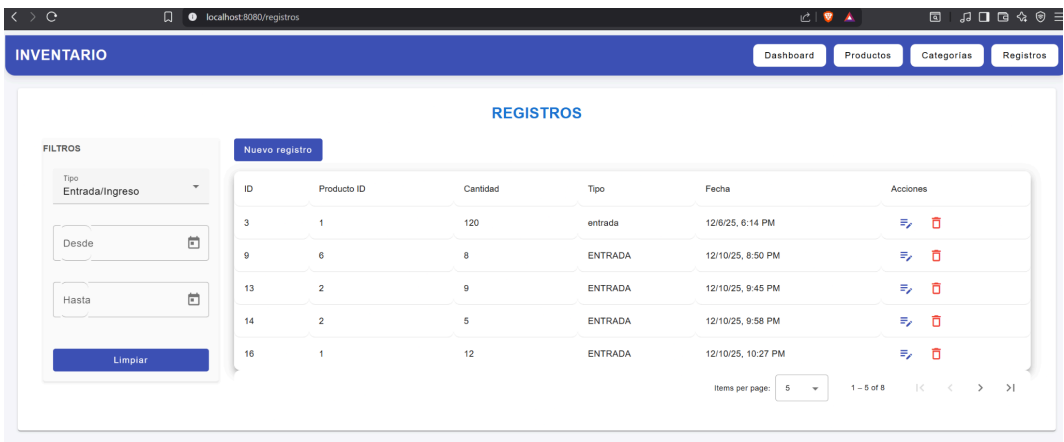
Eliminar Producto:



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 así 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