

# 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.

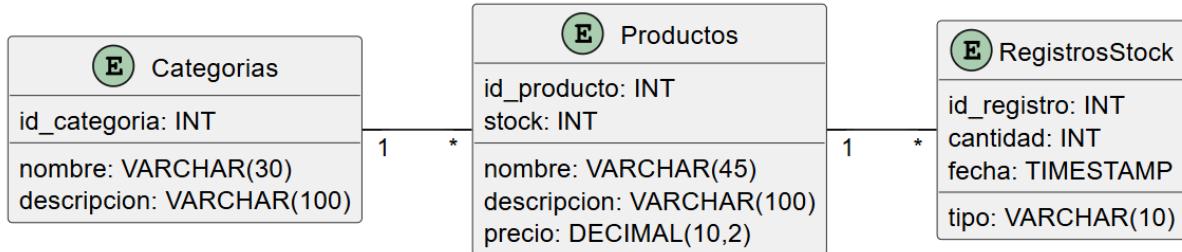
**LENGUAJES:** Java, Typescript, Javascript, HTML, CSS.

**BASE DE DATOS :** MySQL.

**OTROS:** Swagger, Github([https://github.com/MarinoX/proyecto\\_web\\_dinamico](https://github.com/MarinoX/proyecto_web_dinamico) )

# ARQUITECTURA

## Estructura de la Base de Datos:

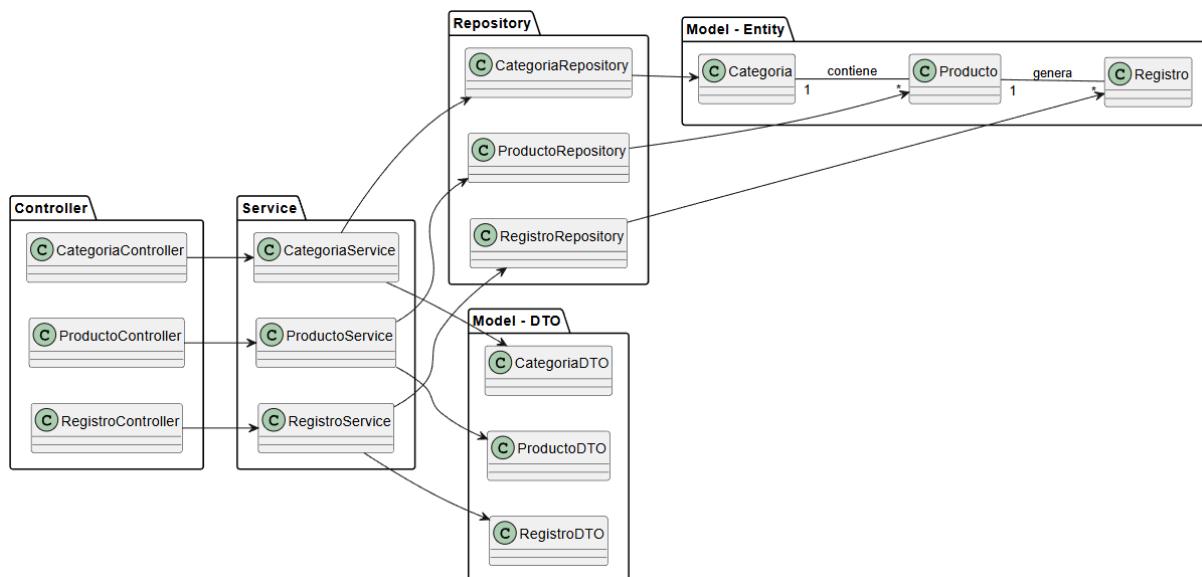


Mi base de datos hecha en SQL está organizada alrededor de la tabla **Categorías** ya que esta funciona como el punto de clasificación principal. Cada categoría tiene un nombre y una descripción, y sirve para agrupar productos de forma ordenada.

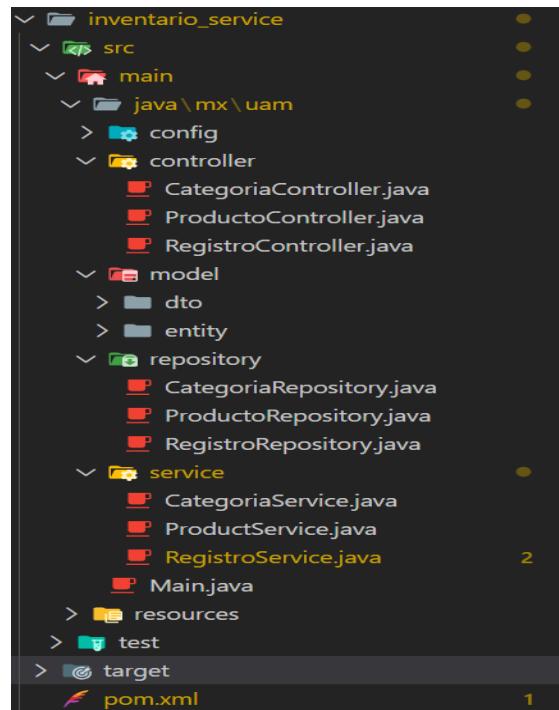
Luego tengo la tabla de **Productos** que es pues la parte principal del inventario. Aquí se almacena todo lo importante: nombre, descripción, precio y el stock actual. Cada producto está ligado a una categoría mediante una relación uno a muchos, lo que ayuda a evitar el duplicar información y mantiene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

```
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
39          id) {
40          CategoriaDTO dto = categoriaService.findById(id);
41          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 si se puede realizar el movimiento automáticamente actualiza el stock del producto.

```
甫 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) {
```

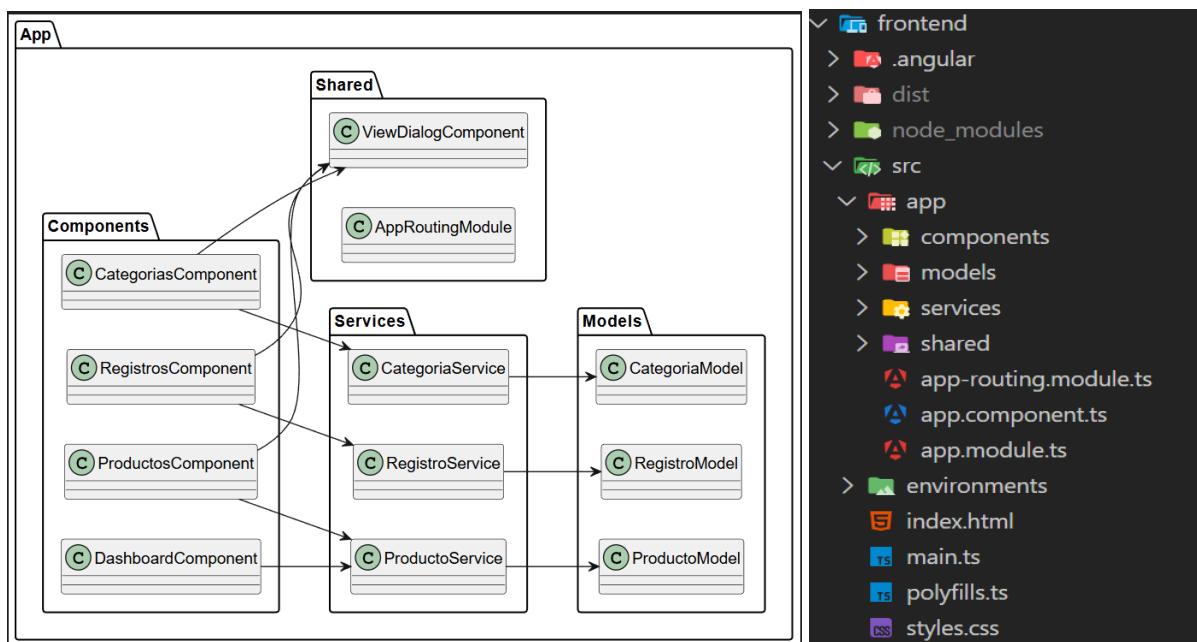
Los repository contienen las queries y operaciones CRUD necesarias y manejan el acceso directo a la base de datos mediante Spring Data JPA. Los modelos están divididos en Entities (Categoria, Producto, Registro) que mapean directamente las tablas de la base de datos y los DTO que son los objetos que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

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

```
甫 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}
```

```
甫 Categoria.java X
11 import jakarta.persistence.Entity;
12
13 @Entity
14 public class Categoria {
15     @Id
16     @GeneratedValue(strategy = GenerationType.IDENTITY)
17     private Integer id;
18     private String nombre;
19     private String descripcion;
20
21     @OneToMany(mappedBy = "categoria", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
22     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, aquí cada componente es responsable de su propia ui y maneja la interacción con el usuario.

```
components
├── categorias
├── dashboard
├── productos
└── registros
```

8	<mat-label>Descripción</mat-label>
9	<textarea matInput formControlName="descripcion" rows="3"></textarea>
10	</mat-form-field>
11	<mat-form-field appearance="fill" style="width:100%">
12	<mat-label>Precio</mat-label>
13	<input matInput type="number" formControlName="precio" min="0">

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.

```

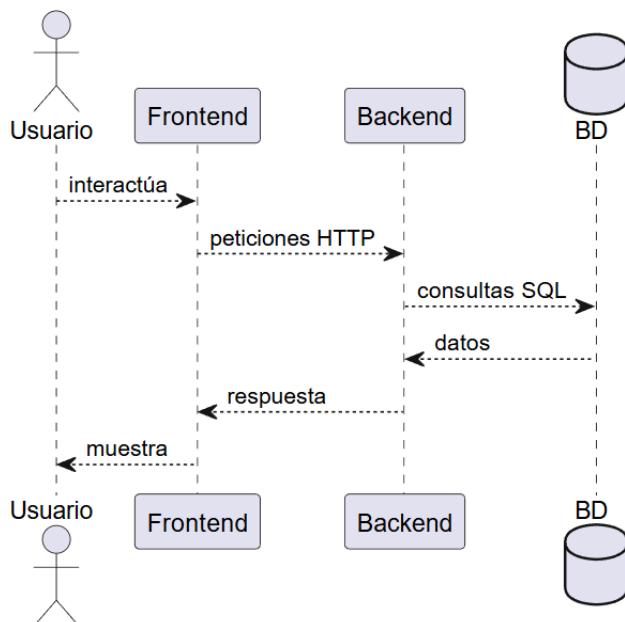
    <services>
      <category.service.ts>
      <product.service.ts>
      <registro.service.ts>
    </services>
  
```

```

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 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>

The screenshot shows the Swagger UI interface running at <http://localhost:8080/swagger-ui/index.html>. The top navigation bar includes the Swagger logo, the URL, and an 'Explore' button. Below the header, the title 'OpenAPI definition' is displayed along with the 'OAS 3.0' badge. A dropdown menu labeled 'Servers' shows the URL 'http://localhost:8080 - Generated server url'. The main content area is organized into sections: 'Categoría' (Operations CRUD related to categories), 'Producto' (Operations CRUD related to products), and 'Registro' (Operations CRUD related to product stock registrations). Each section lists methods (GET, PUT, DELETE, POST) and their corresponding URLs. The 'Create category' and 'Create product' operations are highlighted in green, while others are in blue, orange, or red.

Categoría	
GET	/categorias/{id} Obtener categoría por ID
PUT	/categorias/{id} Actualizar categoría
DELETE	/categorias/{id} Eliminar categoría
GET	/categorias Obtener todas las categorías
POST	/categorias Crear categoría

Producto	
GET	/productos/{id} Obtener producto por ID
PUT	/productos/{id} Actualizar producto
DELETE	/productos/{id} Eliminar producto
GET	/productos Obtener todos los productos
POST	/productos Crear producto

Registro	
GET	/registros/{id} Obtener registro por ID

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.

**GET /categorias/{id}** Obtener categoría por ID

Retorna una categoría por su ID

**Parameters**

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

**Responses**

Code	Description	Links
200	OK	No links

Media type: application/json

Example Value: { "id": 0, "nombre": "string", "descripcion": "string" }

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

## 1.- Crear Categoría.

**POST /categorias** Crear categoría

Crea una nueva categoría

**Parameters**

No parameters

**Request body** required

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

**Request URL**  
http://localhost:8080/categorias

**Server response**

Code	Details
200	Response body

```
{ "id": 3, "nombre": "Limpieza", "descripcion": "Herramientas y elementos de limpieza" }
```

Download

## 2.- Obtener todas las categorías.

The screenshot shows a REST API documentation interface for the `/categorias` endpoint. The top bar indicates a `GET` request and the URL `/categorias`. Below this, a description states: "Recupera una lista de todas las categorías de productos disponibles en el inventario." The "Parameters" section shows "No parameters". At the bottom, there are "Execute" and "Clear" buttons. The "Responses" section contains curl command examples and a request URL. Under "Server response", a table shows the code and details for a 200 status. The response body is a JSON array of three categories:

```
[{"id": 1, "nombre": "Botanas", "descripcion": "Papas y golosinas varias"}, {"id": 2, "nombre": "Panaderia", "descripcion": "doras y panes"}, {"id": 3, "nombre": "Limpieza", "descripcion": "Herramientas y elementos de limpieza"}]
```

Buttons for "Copy" and "Download" are available for the response body.

## 3.- Obtener categoría por id.

The screenshot shows a REST API documentation interface for the `/categorias/{id}` endpoint. The top bar indicates a `GET` request and the URL `/categorias/{id}`. Below this, a description states: "Retorna una categoría por su ID". The "Parameters" section includes a required parameter `id` (integer) with a description "ID de la categoría". A value "1" is entered in the input field. At the bottom, there are "Execute" and "Clear" buttons. The "Responses" section contains curl command examples and a request URL. Under "Server response", a table shows the code and details for a 200 status. The response body is a JSON object for category ID 1:

```
{"id": 1, "nombre": "Botanas", "descripcion": "Sección de Frituras y dulces varios"}
```

Buttons for "Copy" and "Download" are available for the response body.

#### 4.- Actualizar Categoría.

PUT /categorias/{id} Actualizar categoría

Actualiza una categoría existente

**Parameters**

Name	Description
<b>id</b> * required	integer(\$int32) ID de la categoría (path)
	4

**Request body** required application/json

```
{ "descripcion": "Articulos y comida para perros" }
```

**Code** Details

Code	Details
200	<b>Response body</b> <pre>{ "id": 4, "nombre": "Mascotas", "descripcion": "Articulos y comida para perros" }</pre> <span>Copy</span> <span>Download</span>

#### 5.- Eliminar Categoría

DELETE /categorias/{id} Eliminar categoría

Elimina una categoría por su ID

**Parameters**

Name	Description
<b>id</b> * required	integer(\$int32) ID de la categoría (path)
	4

**Code** Details

Code	Details
204	<b>Response headers</b> <pre>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</pre>

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

200 Response body

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

Copy Download

## 6.- Obtener producto por id.

GET /productos/{id} Obtener producto por ID

Retorna un producto por su ID

Parameters

Name	Description
<b>id</b> * required	integer(\$int32) ID del producto (path)

5

Execute Clear

Code Details

200 Response body

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

Copy 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": "danas y panes" } }
```

Usando el endpoint de actualizar producto por id

PUT /productos/{id} Actualizar producto

Actualiza un producto existente

Parameters

Name	Description
<b>id</b> * required	integer(\$int32) ID del producto (path)

5

Request body required

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

application/json

Cancel Reset

200

Response body

```
{
  "id": 5,
  "nombre": "Chocorroles",
  "descripcion": "Pan de chocolate en forma de rollo",
  "precio": 24,
  "stock": 100,
  "categoria": {
    "id": 2,
    "nombre": "Panaderia",
    "descripcion": "doras y panes"
  }
}
```

[Copy](#) [Download](#)

## 8.- Obtener todos los registros

GET /registros Obtener todos los registros

Recupera una lista de todos los registros de stock

Server response

Code	Details
200	<p>Response body</p> <pre>[   {     "id": 3,     "productId": 1,     "cantidad": 120,     "tipo": "entrada",     "fecha": "2025-12-06T18:14:13.076075"   },   {     "id": 4,     "productId": 1,     "cantidad": 101,     "tipo": "salida",     "fecha": "2025-12-06T18:14:53.052514"   },   {     "id": 8,     "productId": 5,     "cantidad": 20,     "tipo": "SALIDA",     "fecha": "2025-12-10T20:48:48.888397"   },   {     "id": 9,     "productId": 5,     "cantidad": 10,     "tipo": "SALIDA",     "fecha": "2025-12-10T20:48:48.888397"   } ]</pre>

## 9.- Obtener registro por producto

GET /registros/producto/{productId} Obtener registros por producto

Retorna todos los registros de un producto específico

Parameters

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

[Cancel](#)

200

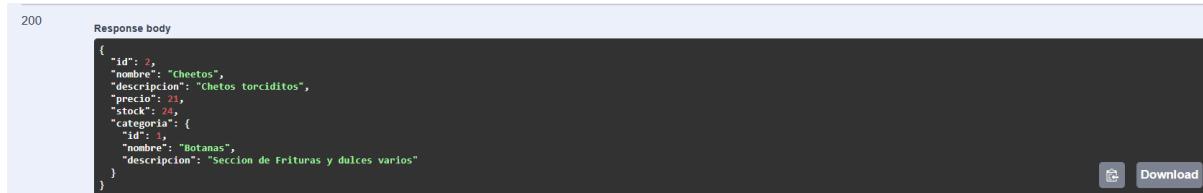
Response body

```
[
  {
    "id": 3,
    "productId": 1,
    "cantidad": 120,
    "tipo": "entrada",
    "fecha": "2025-12-06T18:14:13.076075"
  },
  {
    "id": 4,
    "productId": 1,
    "cantidad": 101,
    "tipo": "salida",
    "fecha": "2025-12-06T18:14:53.052514"
  },
  {
    "id": 8,
    "productId": 5,
    "cantidad": 20,
    "tipo": "SALIDA",
    "fecha": "2025-12-10T20:48:48.888397"
  },
  {
    "id": 9,
    "productId": 5,
    "cantidad": 10,
    "tipo": "SALIDA",
    "fecha": "2025-12-10T20:48:48.888397"
  }
]
```

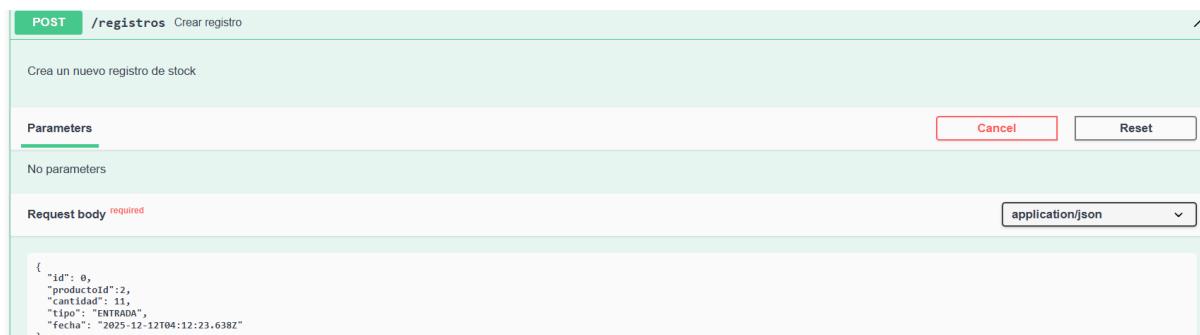
## 10.- Crear nuevo registro.

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

```
200 Response body
{
  "id": 2,
  "nombre": "Cheetos",
  "descripcion": "Cheetos torcidos",
  "precio": 21,
  "stock": 24,
  "categoria": {
    "id": 1,
    "nombre": "Botanas",
    "descripcion": "Sección de Frituras y dulces varios"
  }
}
```



## Registro de entrada.



```
POST /registros Crear registro

Crea un nuevo registro de stock

Parameters
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": "2025-12-11T22:14:48.337Z149"
}
```



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

```
{
  "id": 2,
  "nombre": "Cheetos",
  "descripcion": "Cheetos torcidos",
  "precio": 21,
  "stock": 35,
  "categoria": {
    "id": 1,
    "nombre": "Botanas",
    "descripcion": "Sección de Frituras y dulces varios"
  }
}
```



## Registro de Salida.

The screenshot shows a POST request to the endpoint `/registros` with the sub-action `Crear registro`. The interface is a form for creating a new stock entry. It includes fields for parameters (none listed), a request body (marked as required) with application/json content type, and a preview of the JSON data being sent:

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

Below the preview, there is another JSON object:

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

Buttons for `Cancel`, `Reset`, and `Download` are visible at the bottom right.

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

The screenshot shows a GET request to the endpoint `/productos`. The response is a JSON object containing product details:

```
{
  "id": 2,
  "nombre": "Cheetos",
  "descripcion": "Chetos torcidos",
  "precio": 21,
  "stock": 5,
  "categoria": {
    "id": 1,
    "nombre": "Botanas",
    "descripcion": "Sección de Frituras y dulces varios"
  }
}
```

Buttons for `Copy` and `Download` are visible at the bottom right.

## 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.

The screenshot shows the 'Dashboard' section of the application. At the top, there are five colored boxes displaying summary data: 'Total Productos' (5), 'Total Categorías' (3), 'Total Registros' (15), 'Ingresos Hoy' (0), and 'Salidas Hoy' (30). Below these, a yellow box highlights 'Productos con Stock Bajo' (3). A table titled 'Productos con Stock Bajo' lists three items: Cheetos (Stock 5, Precio \$21.00), Sabritas (Stock 8, Precio \$25.00), and Chocorroles (Stock 3, Precio \$24.00).

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).

The screenshot shows the 'CATEGORÍAS' (Categories) page. It displays a table with three rows: 'Botanas' (Descripción: Sección de Frituras y dulces varios), 'Panadería' (Descripción: donas y panes), and 'Health Care' (Descripción: Artículos de cuidado personal). Each row has edit and delete icons in the 'Acciones' column. At the bottom, there are pagination controls for 'Items per page' (10) and '1 – 3 of 3'.

The screenshot shows the 'CATEGORÍAS' page with a modal window open for creating a new category. The modal has fields for 'Nombre\*' (Farmacia) and 'Descripción' (Medicinas y curaciones). At the bottom of the modal are 'Cancelar' and 'Guardar' buttons. The background table remains the same as in the previous screenshot.

CATEGORÍAS				
<a href="#">Nueva categoría</a>				
ID	Nombre	Descripción	Acciones	
1	Botanas	Sección de Frituras y dulces varios	<a href="#"></a>	<a href="#"></a>
2	Panadería	dones y panes	<a href="#"></a>	<a href="#"></a>
6	Health Care	Artículos de cuidado personal	<a href="#"></a>	<a href="#"></a>
7	Farmacia	Medicinas y curaciones	<a href="#"></a>	<a href="#"></a>

Items per page: 10 | < < > >| 1 – 4 of 4

**INVENTARIO**

[Dashboard](#) [Productos](#) [Categorías](#) [Registros](#)

CATEGORÍAS

[Nueva categoría](#)

ID	Nombre	Descripción	Acciones	
1	Botanas	Sección de Frituras y dulces varios	<a href="#"></a>	<a href="#"></a>
2	Panadería	dones y panes	<a href="#"></a>	<a href="#"></a>
6	Health Care	Artículos de cuidado personal	<a href="#"></a>	<a href="#"></a>
7	Farmacia	Medicinas y curaciones	<a href="#"></a>	<a href="#"></a>

**Editar categoría**

Nombre\*  
Panadería

Descripción  
Variedad de panes y galletas

Cancelar **Guardar**

Items per page: 10 | < < > >| 1 – 4 of 4

2 Panadería Variedad de panes y galletas

[localhost:8080/categorias](#)

**INVENTARIO**

[Dashboard](#) [Productos](#) [Categorías](#) [Registros](#)

CATEGORÍAS

[Nueva categoría](#)

ID	Nombre	Descripción	Acciones	
1	Botanas	Sección de Frituras y dulces varios	<a href="#"></a>	<a href="#"></a>
2	Panadería	Variedad de panes y galletas	<a href="#"></a>	<a href="#"></a>

localhost:8080 dice

Eliminar categoría?

**Aceptar** **Cancelar**

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 mínimo 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**

**PRODUCTOS**

**FILTROS**

**Nuevo producto**

ID	Nombre	Descripción	Precio	Stock	Categoría	Acciones
1	Doritos	Botana doritos nacho	\$20.00	26	Botanas	
2	Cheetos	Cheetos torcidos	\$21.00	5	Botanas	
3	Sabritas	sabritas de sal	\$25.00	8	Botanas	
5	Chocorrolles	Pan de chocolate en forma de rollo	\$24.00	3	Panaderia	
6	Chips Ahoy	galletas con chispas de chocolate	\$18.00	10	Panaderia	

Items per page: 10 | 1 - 5 of 5 | < < > >

**INVENTARIO**

**PRODUCTOS**

**FILTROS**

**Nuevo producto**

**Nuevo Producto**

Nombre\*: Paracetamol

Descripción: Medicamento utilizado para tratar dolores leves

Precio: 19

Stock: 33

Categoría: Farmacia

**Stock** **Categoría** **Acciones**

26	Botanas	
5	Botanas	
8	Botanas	
3	Panaderia	
10	Panaderia	

Items per page: 10 | 1 - 5 of 5 | < < > >

## Editar producto:

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

**INVENTARIO**

**PRODUCTOS**

**FILTROS**

**Nuevo producto**

**Editar Producto**

Nombre\*: Cheetos

Descripción: Fritura de maíz con queso y chile

Precio: 21

Stock: 5

Categoría: Botanas

**Precio** **Stock** **Categoría** **Acciones**

\$20.00	26	Botanas	
\$21.00	5	Botanas	
\$25.00	8	Botanas	
\$24.00	3	Panaderia	
\$18.00	10	Panaderia	
\$19.00	33	Farmacia	

Items per page: 10 | 1 - 6 of 6 | < < > >

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

## Eliminar Producto:

The screenshot shows a confirmation dialog box titled "localhost:8080 dice" with the message "Eliminar producto?". There are two buttons: "Aceptar" (Accept) and "Cancelar" (Cancel). The background shows a table of products with columns: ID, Nombre, Descripción, Precio, Stock, Categoría, and Acciones. The first three rows are Doritos, Cheetos, and Sabritas, all belonging to the "Botanas" category.

## Filtrado de productos:

The screenshots show three different filtering scenarios for a product list:

- Filter by Category:** The first screenshot shows a filter set to "Botanas". The results show three products: Cheetos, Chocorrolles, and Chips Ahoy, all in the "Panadería" category.
- Filter by Category and Search:** The second screenshot shows a filter set to "Botanas" and a search term "Ch". The results show three products: Cheetos, Chocorrolles, and Chips Ahoy, all in the "Panadería" category.
- Advanced Filtering:** The third and fourth screenshots show more complex filters. The third screenshot has filters for Category ("Botanas"), Search ("Buscar"), Price min ("Precio min"), Price max ("Precio max"), and Stock min ("Stock min"). The results show Doritos and Paracetamol. The fourth screenshot has filters for Category ("Botanas"), Search ("Buscar"), Price min ("Precio min"), Price max ("Precio max"), and Stock min ("Stock min"). The results show Doritos, Chips Ahoy, and Paracetamol.

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.

The screenshots illustrate the 'REGISTROS' (Registers) page of an inventory management system. The top navigation bar includes 'Dashboard', 'Productos', 'Categorías', and 'Registros'. The main header shows 'INVENTARIO' and 'REGISTROS'.

**Screenshot 1: Initial State**

- Filtros (Filters): Tipo Entrada/Ingreso, Desde (From), Hasta (To), Limpiar (Clear).
- Table Headers: ID, Producto ID, Cantidad, Tipo, Fecha, Acciones.
- Table Data:
 

ID	Producto ID	Cantidad	Tipo	Fecha	Acciones
3	1	120	entrada	12/6/25, 6:14 PM	
9	6	8	ENTRADA	12/10/25, 8:50 PM	
13	2	9	ENTRADA	12/10/25, 9:45 PM	
14	2	5	ENTRADA	12/10/25, 9:58 PM	
16	1	12	ENTRADA	12/10/25, 10:27 PM	
- Pagination: Items per page: 5, 1 – 5 of 8.

**Screenshot 2: Creating a New Entry (Entrada)**

- Filtros (Filters): Tipo Entrada/Ingreso, Desde, Hasta, Limpiar.
- Table Headers: ID, Producto ID, Cantidad, Tipo, Fecha, Acciones.
- Table Data (partial view):
 

ID	Producto ID	Cantidad	Tipo	Fecha	Acciones
3	1				
9	6				
13	2				
14	2				
16	1				
- New registration modal:
  - Nuevo registro
  - Producto\*: Doritos
  - Cantidad\*: 20
  - Tipo\*: Entrada
  - Guardar button
- Pagination: Items per page: 5, 1 – 5 of 8.

**Screenshot 3: Creating a New Entry (Salida)**

- Filtros (Filters): Tipo Salida, Desde, Hasta, Limpiar.
- Table Headers: ID, Producto ID, Cantidad, Tipo, Fecha, Acciones.
- Table Data:
 

ID	Producto ID	Cantidad	Tipo	Fecha	Acciones
4	1	101	salida	12/6/25, 6:14 PM	
8	5	20	SALIDA	12/10/25, 8:48 PM	
10	6	10	SALIDA	12/10/25, 8:51 PM	
11	1	10	SALIDA	12/10/25, 8:53 PM	
12	2	20	SALIDA	12/10/25, 9:45 PM	
- Pagination: Items per page: 5, 1 – 5 of 7.

**Screenshot 4: Advanced Filtering**

- Filtros (Filters): Tipo Todos, Desde 12/9/2025, Hasta 12/11/2025, Limpiar.
- Table Headers: ID, Producto ID, Cantidad, Tipo, Fecha, Acciones.
- Table Data:
 

ID	Producto ID	Cantidad	Tipo	Fecha	Acciones
8	5	20	SALIDA	12/10/25, 8:48 PM	
9	6	8	ENTRADA	12/10/25, 8:50 PM	
10	6	10	SALIDA	12/10/25, 8:51 PM	
11	1	10	SALIDA	12/10/25, 8:53 PM	
12	2	20	SALIDA	12/10/25, 9:45 PM	
- Pagination: Items per page: 5, 1 – 5 of 9.

## 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](https://github.com/MarinoX/proyecto_web_dinamico)