

**FACULTAD DE INGENIERÍA Y ARQUITECTURA**

**Escuela Profesional de Ingeniería de Sistemas**

**Curso: Desarrollo de Aplicaciones Distribuidas**

**Guía práctica dirigida de la sesión 01**

Desarrolla un CRUD con Spring Boot

**Docente:**

Contenido

[**Objetivo 3**](#_6rn3ghsdcdsz)

[**Preparación de ambiente de desarrollo 3**](#_89wmf0rvf18m)

[**Crear una base de datos en MySQL 4**](#_owrvzrwx8sly)

[**Crear el proyecto: ms-catalogo 5**](#_91e3vh2hmy7y)

[**CRUD 9**](#_kns37e6adj2r)

[Entity 9](#_bzdupo7v3w2q)

[Repository 11](#_xmej4d6cz1v6)

[Service 12](#_x8238hl6j55)

[ServiceImpl 13](#_mts3haqy1zpp)

[Controller 14](#_4rsiol28x39x)

[Conexión a base datos 15](#_5uttc9dip2ct)

[**Ejecuta 16**](#_y55mp3wx3gir)

[**Testing 17**](#_9l9hu72cjb15)

[**OpenAPI Specification 18**](#_5x210unw1xu)

# Objetivo

El objetivo de esta práctica es desarrollar un CRUD API Rest con Spring Boot como arquitectura base

# Preparación de ambiente de desarrollo

1. **Instalación de Laragon**

Ir a la dirección<https://docs.google.com/document/d/1vPqsINtLVzLEpgudRhP46bd7pEkH556PZH2tibCtcPk/edit?usp=drive_link> y siga las instrucciones.

1. **Instalación y configuración de Git**

Ir a la dirección<https://docs.google.com/document/d/1jcjL-QctAPZ79HT-k03iHnQCZVWNDthFa6HQmO7aKdI/edit?usp=drive_link> y siga las instrucciones.

1. **Instalación de Java**

Ir a la dirección<https://docs.google.com/document/d/16haaTdjjEzYLc9bcftVuwtaBjJ-ukLEzeXpjQlyTAqw/edit?usp=drive_link> y siga las instrucciones.

1. **Instalación de Intellij idea**

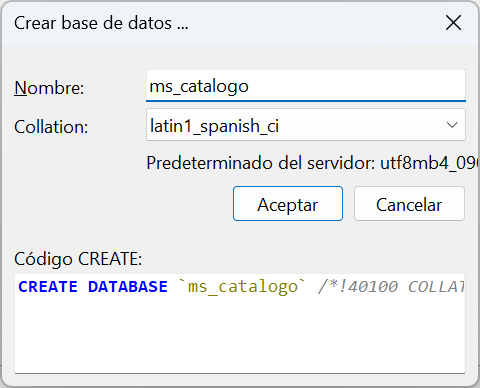
Ir a la dirección<https://docs.google.com/document/d/11CgUbCC0_ijWNtnoSSB38qbt1IIcZWTM6Q_quGBBdTE/edit?usp=drive_link> y siga las instrucciones.

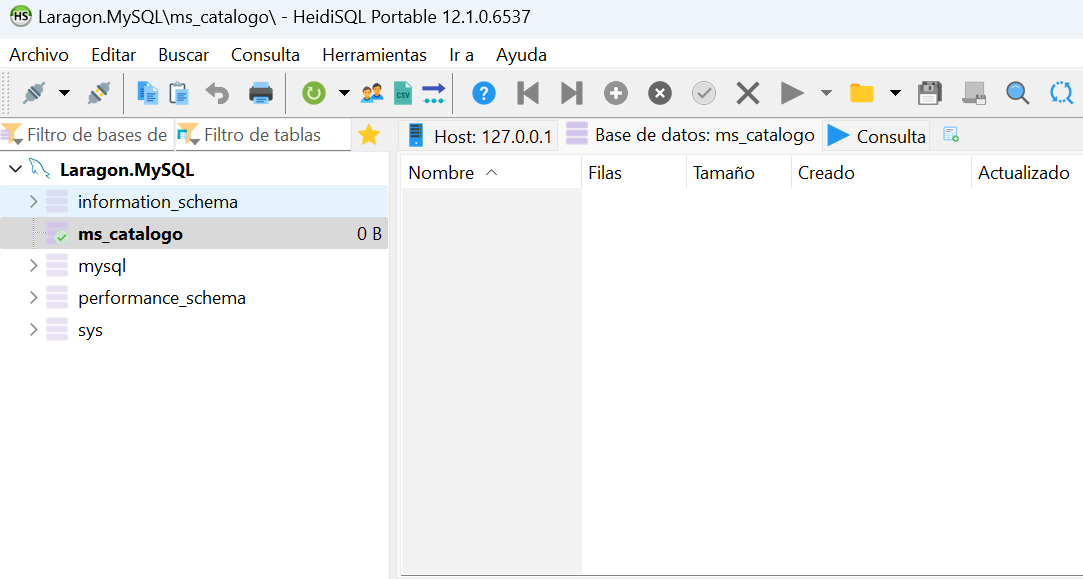
1. **Instalación de HTTPie/Postman**

Ir a la dirección<https://docs.google.com/document/d/1TPEv46MDftcefTuuwwnwIWHxYsxmhgJKKoXvicOdyFA/edit?usp=sharing> y siga las instrucciones.

# Crear una base de datos en MySQL

En Laragon, crear la base de datos ***ms\_catalogo***, si necesita ayuda, revise el manual de instalación de Laragon





# Crear el proyecto: ms-catalogo

# 

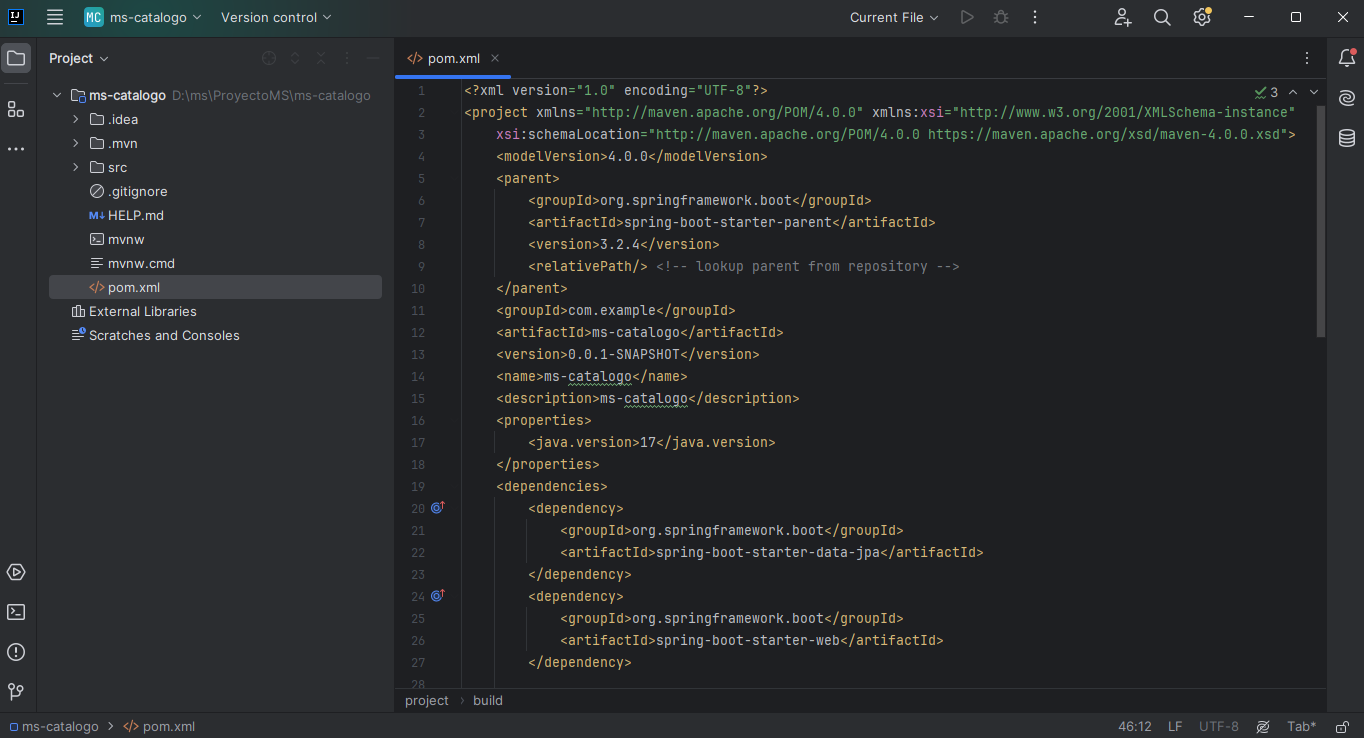
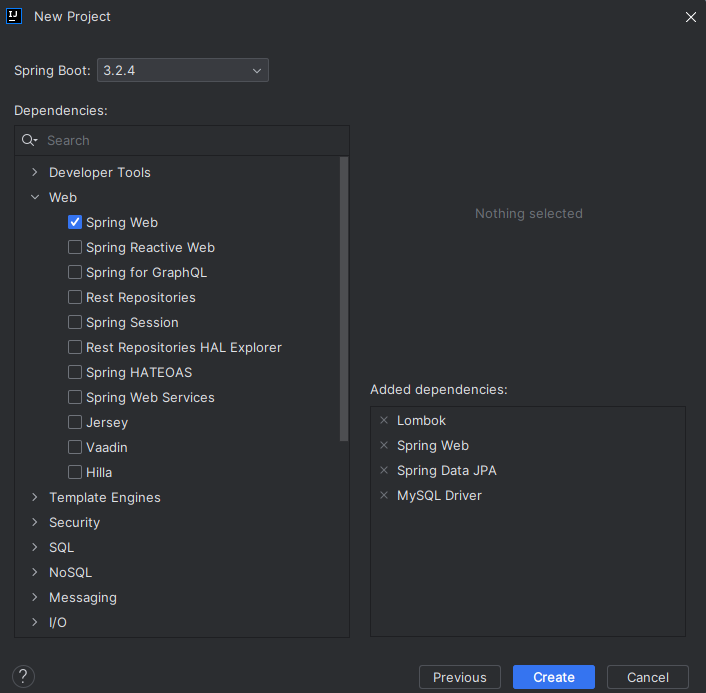
Name: ms-catalogo

Location: D:\ms\ProyectoMS

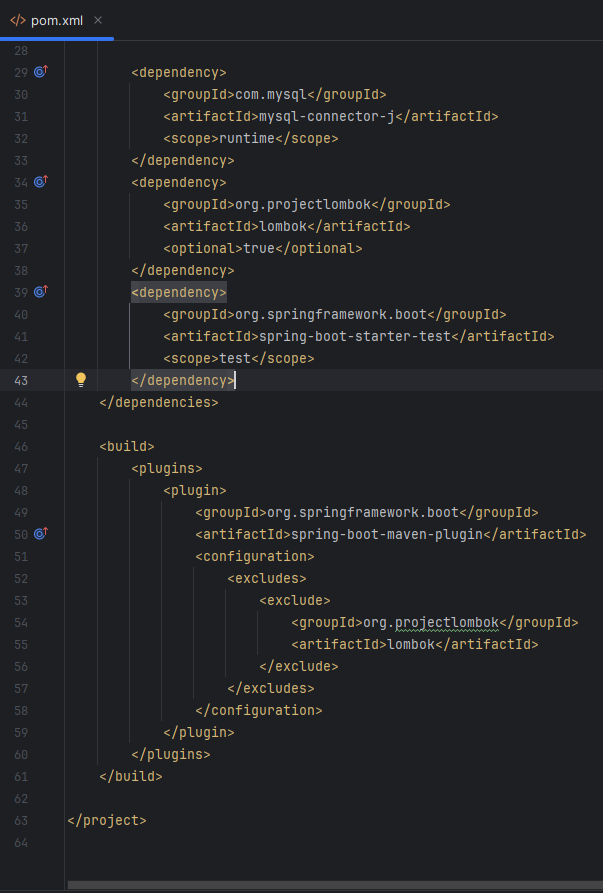
Type: Maven

Elija la versión para Spring boot: 3.1.3, luego puedes cambiar de versión

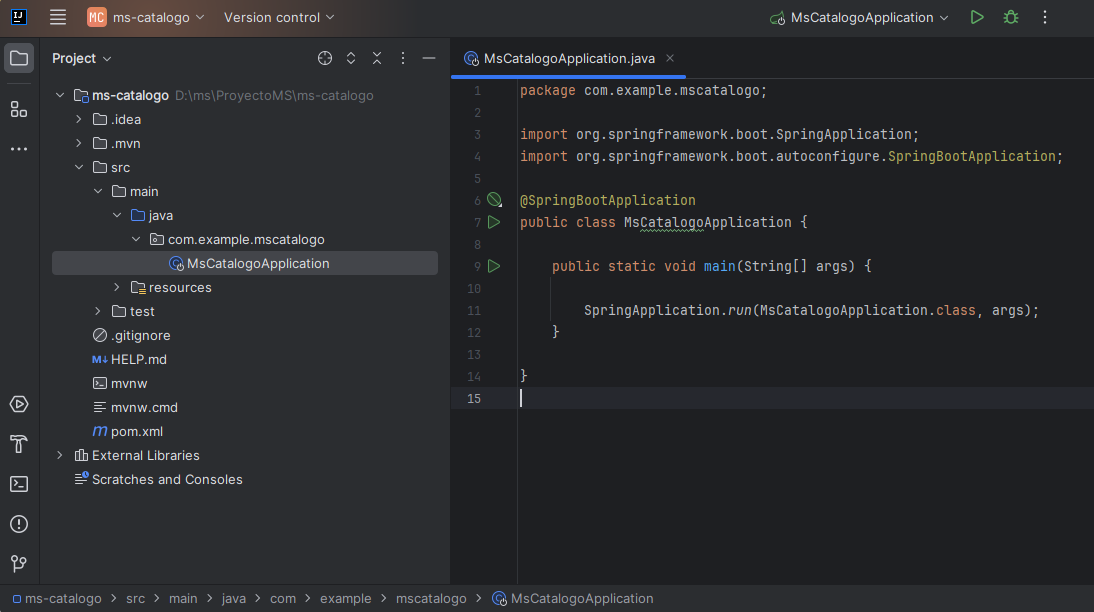
Selecciones las librerías para desarrollar un CRUD API Rest



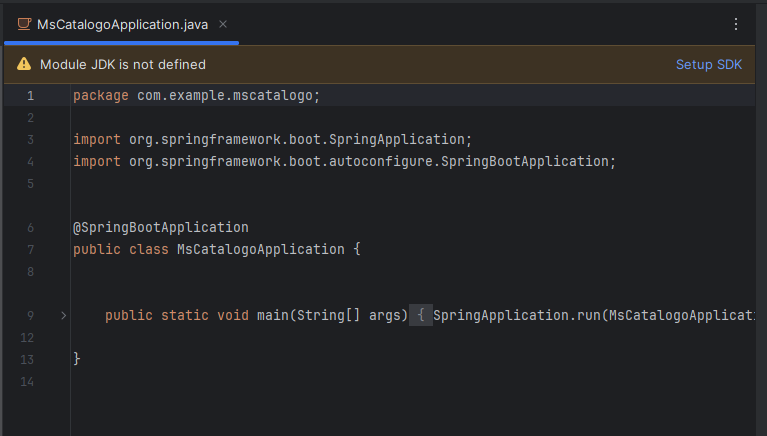
Recuerde estas dependencias iniciales



Clase principal listo para ejecutar

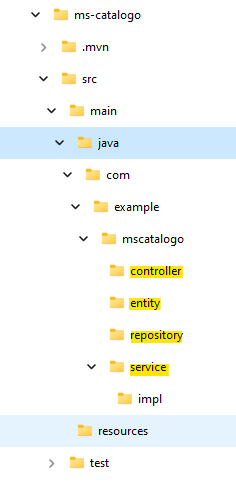


NOTA: Si te aparece esta ventana.



Debes actualizar tu IDE o descarga las librerías con maven

# CRUD



## Entity

Una entidad (Entity) es una clase Java que representa una tabla en una base de datos relacional. Esta clase se mapea directamente a una tabla específica en la base de datos, y cada instancia de la clase representa una fila en esa tabla.

package com.example.mscatalogo.entity;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import lombok.Data;

@Entity

@Data

public class Categoria {

@Id

@GeneratedValue(strategy = GenerationType.*IDENTITY*)

private Integer id;

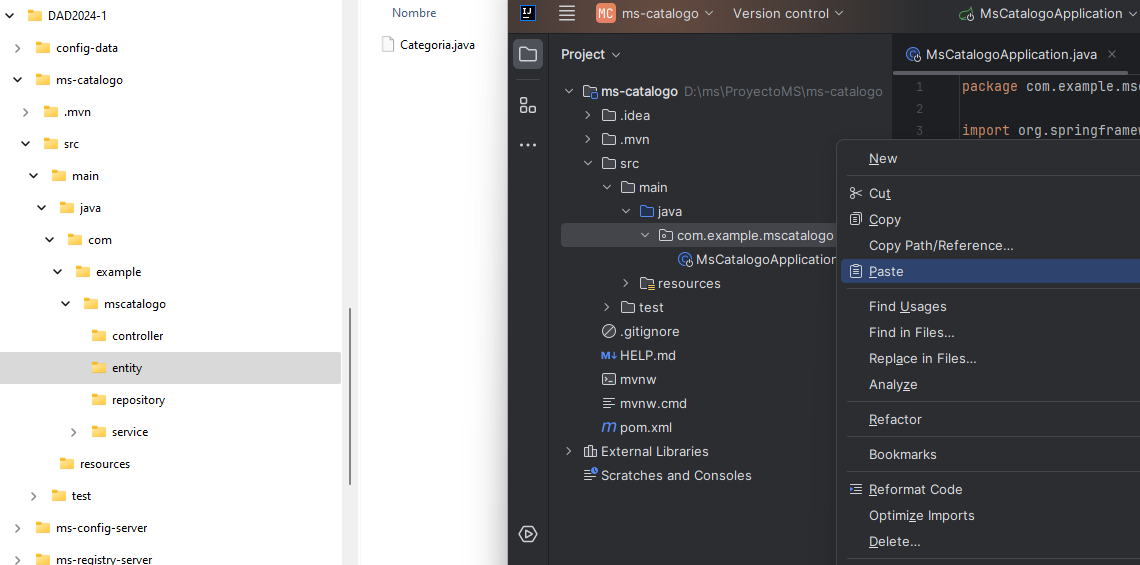
private String nombre;

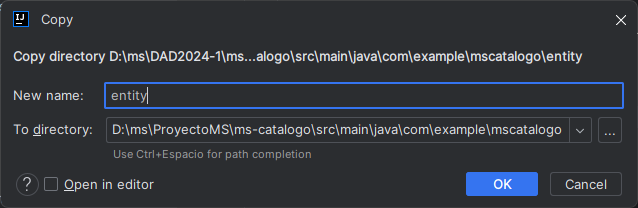
}

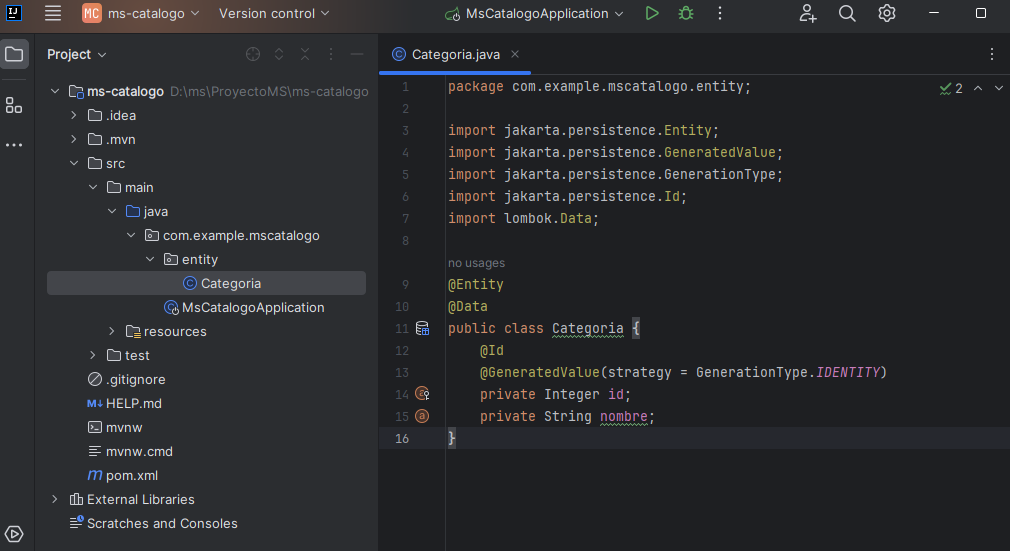
Descargue el código para luego copiar y pegar de aquí <https://github.com/noetipo/DAD2024-1/tree/main/ms-catalogo>

git clone https://github.com/noetipo/DAD2024-1.git

New: Ahora también de <https://github.com/ms-upeu/ProyectoMS/tree/01-CRUD-api-rest>







## Repository

Interfaz que actúa como una capa de abstracción sobre la capa de persistencia de la aplicación. Está diseñado para proporcionar métodos y operaciones específicas para interactuar con la base de datos y realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en las entidades.

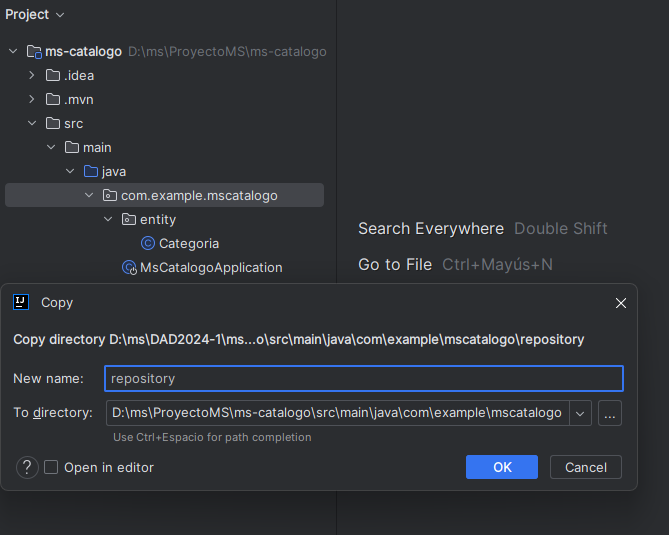
package com.example.mscatalogo.repository;

import com.example.mscatalogo.entity.Categoria;

import org.springframework.data.jpa.repository.JpaRepository;

public interface CategoriaRepository extends JpaRepository<Categoria, Integer> {

}



## Service

Capa intermedia que contiene la lógica de negocio de la aplicación. Su principal objetivo es encapsular la lógica y la funcionalidad relacionada con las reglas de negocio para que sea reutilizable y separada de las capas de controladores y persistencia.

Los servicios se utilizan para coordinar las operaciones entre los controladores (interfaz de usuario) y los repositorios (capa de persistencia). Son una parte importante de la arquitectura de la aplicación ya que promueven modularidad, la reutilización de código y el mantenimiento sostenible.

package com.example.mscatalogo.service;

import com.example.mscatalogo.entity.Categoria;

import java.util.List;

import java.util.Optional;

public interface CategoriaService {

public List<Categoria> listar();

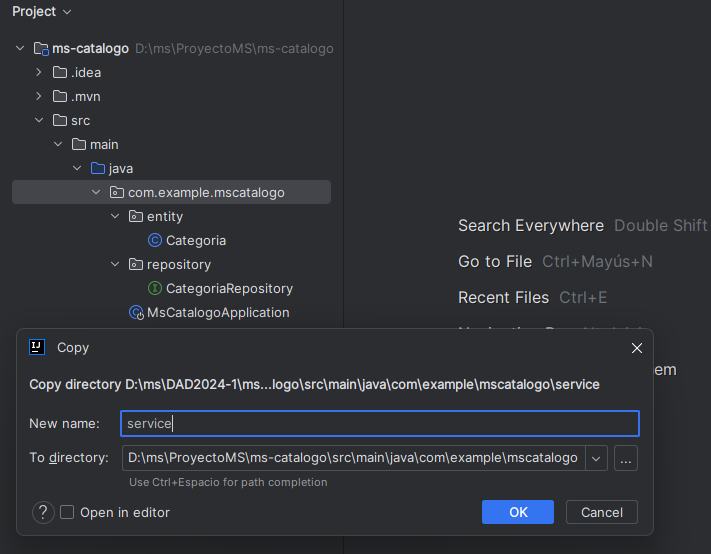
public Categoria guardar(Categoria categoria);

public Categoria actualizar(Categoria categoria);

public Optional<Categoria> listarPorId(Integer id);

public void eliminarPorId(Integer id);

}



### ServiceImpl

package com.example.mscatalogo.service.impl;

import com.example.mscatalogo.entity.Categoria;

import com.example.mscatalogo.repository.CategoriaRepository;

import com.example.mscatalogo.service.CategoriaService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Optional;

@Service

public class CategoriaServiceImpl implements CategoriaService {

@Autowired

private CategoriaRepository categoriaRepository;

@Override

public List<Categoria> listar() {

return categoriaRepository.findAll();

}

@Override

public Categoria guardar(Categoria categoria) {

return categoriaRepository.save(categoria);

}

@Override

public Categoria actualizar(Categoria categoria) {

return categoriaRepository.save(categoria);

}

@Override

public Optional<Categoria> listarPorId(Integer id) {

return categoriaRepository.findById(id);

}

@Override

public void eliminarPorId(Integer id) {

categoriaRepository.deleteById(id);

}

}

## Controller

Clase que actúa como punto de entrada para las peticiones HTTP recibidas desde el cliente (navegador web, aplicación móvil, etc.). Está diseñado para manejar las solicitudes del cliente y responder con la representación apropiada de los datos, ya sea HTML, JSON, XML u otros formatos.

package com.example.mscatalogo.controller;

import com.example.mscatalogo.entity.Categoria;

import com.example.mscatalogo.service.CategoriaService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/categoria")

public class CategoriaController {

@Autowired

private CategoriaService categoriaService;

@GetMapping()

public ResponseEntity<List<Categoria>> list() {

return ResponseEntity.*ok*().body(categoriaService.listar());

}

@PostMapping()

public ResponseEntity<Categoria> save(@RequestBody Categoria categoria){

return ResponseEntity.*ok*(categoriaService.guardar(categoria));

}

@PutMapping()

public ResponseEntity<Categoria> update(@RequestBody Categoria categoria){

return ResponseEntity.*ok*(categoriaService.actualizar(categoria));

}

@GetMapping("/{id}")

public ResponseEntity<Categoria> listById(@PathVariable(required = true) Integer id){

return ResponseEntity.*ok*().body(categoriaService.listarPorId(id).get());

}

@DeleteMapping("/{id}")

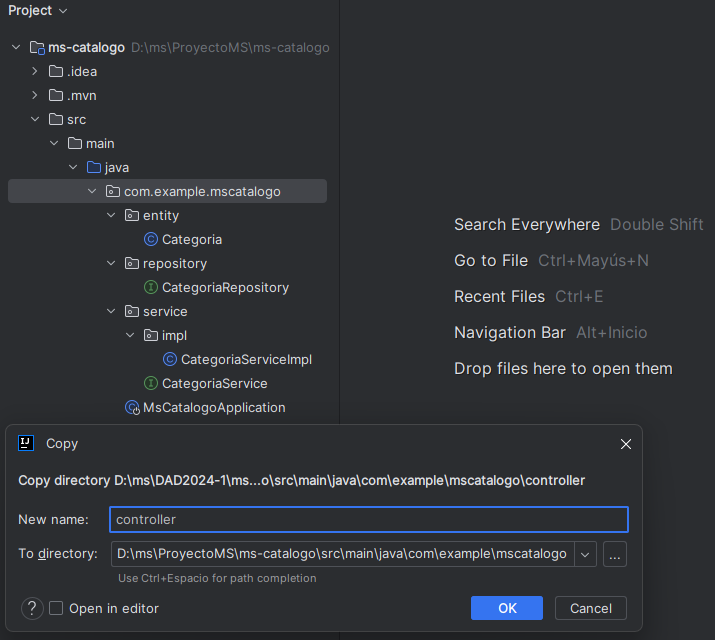
public String deleteById(@PathVariable(required = true) Integer id){

categoriaService.eliminarPorId(id);

return "Eliminacion Correcta";

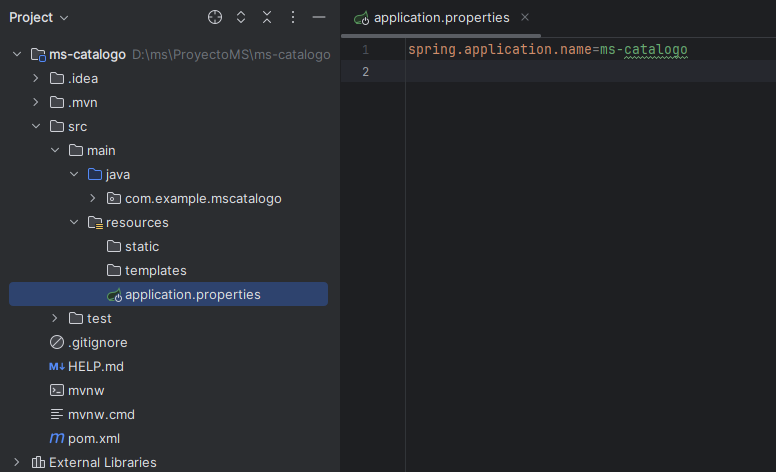
}

}



## Conexión a base datos

Para conectar y realizar la operación CRUD con MySQL DB con la aplicación Spring Boot, se debe configurar dentro del archivo [application.properties](https://www.geeksforgeeks.org/spring-boot-application-properties/?ref=rp) de la siguiente manera:



spring.application.name=ms-catalogo

# CONEXION A BASE DE DATOS

spring.jpa.hibernate.ddl-auto=update

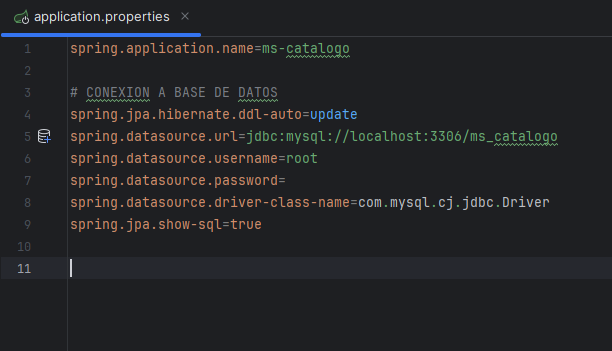
spring.datasource.url=jdbc:mysql://localhost:3306/ms\_catalogo

spring.datasource.username=root

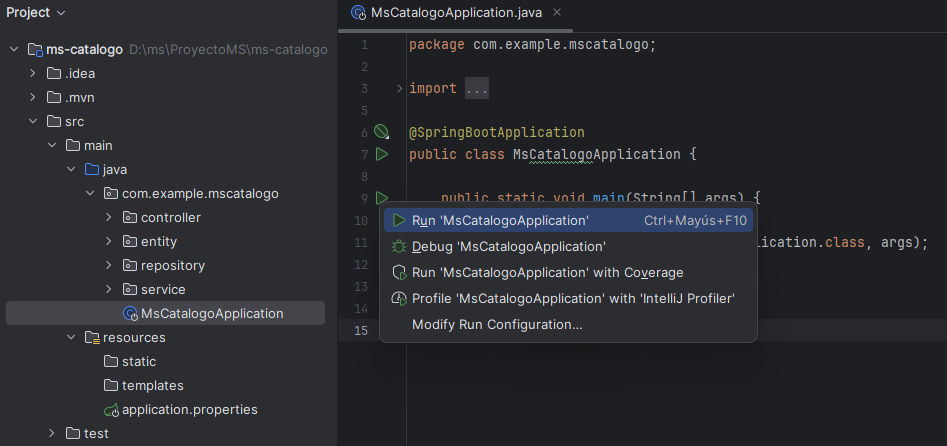
spring.datasource.password=

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

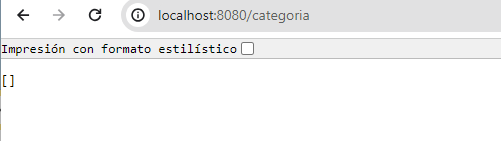
spring.jpa.show-sql=true



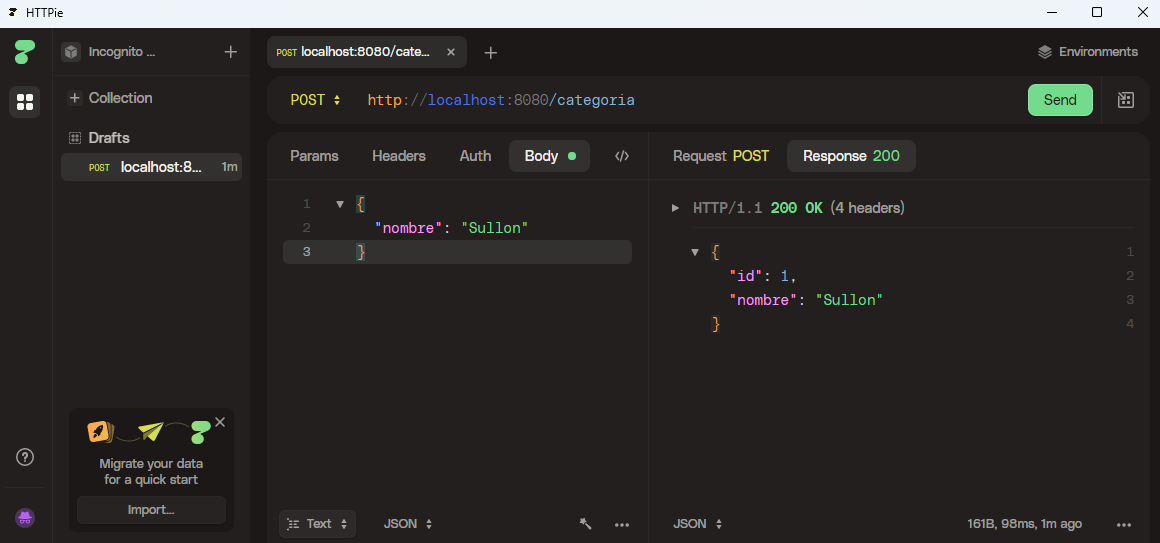
# Ejecuta



<http://localhost:8080/categoria>



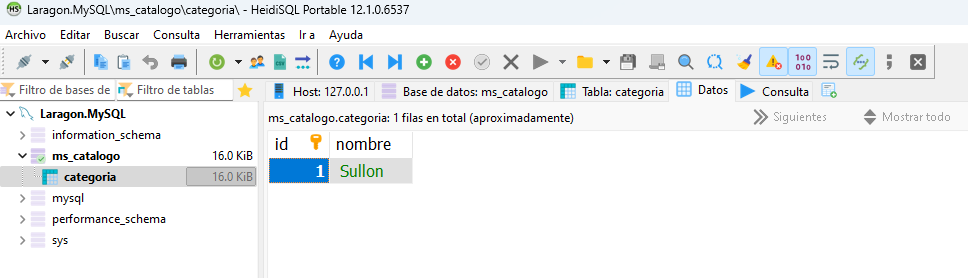
# Testing

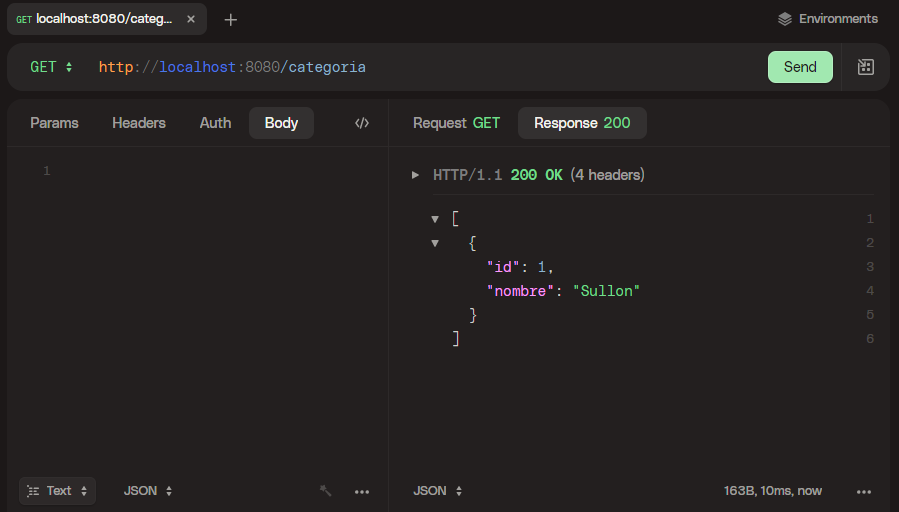


{

"nombre": "Sullon"

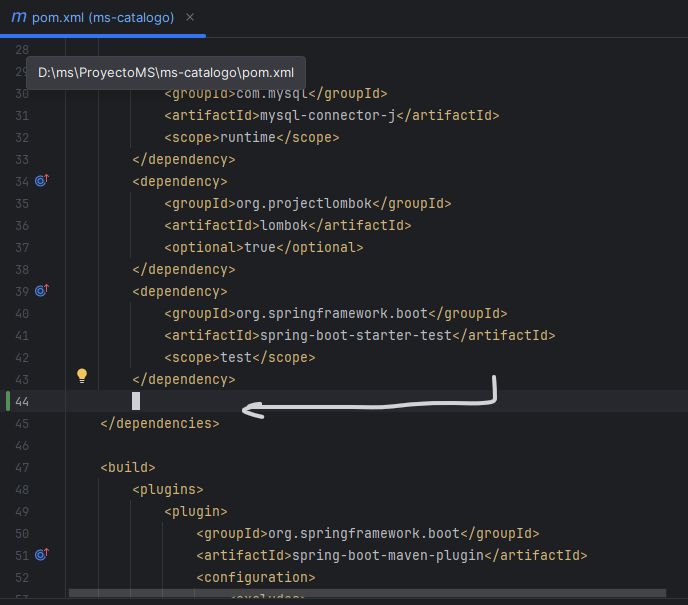
}





# OpenAPI Specification

Agregar dependencias en maven pom.xml (si usas maven)



<dependency>

<groupId>org.springdoc</groupId>

<artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>

<version>2.0.2</version>

</dependency>

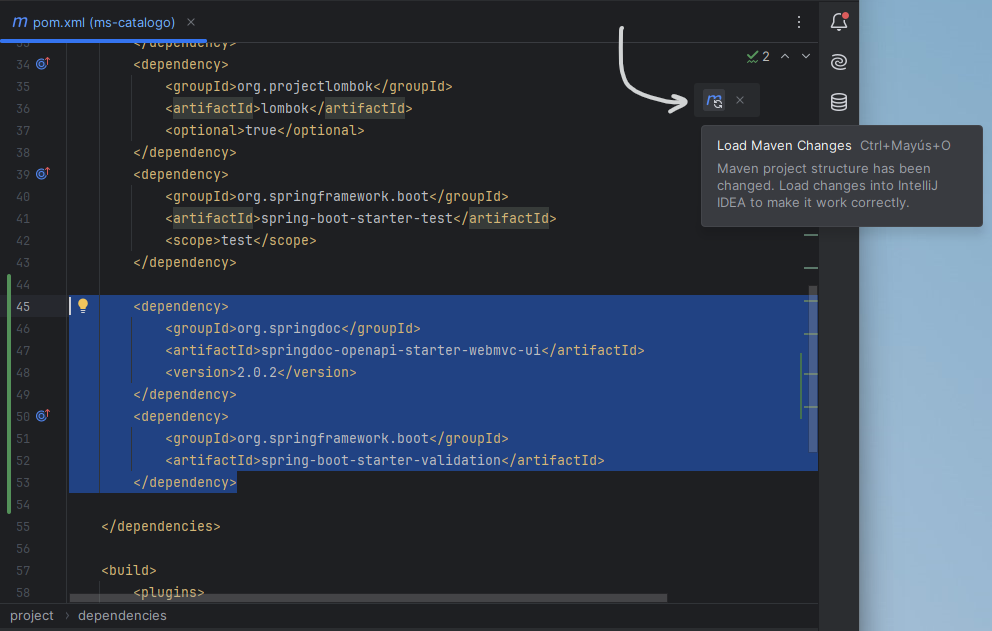
<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

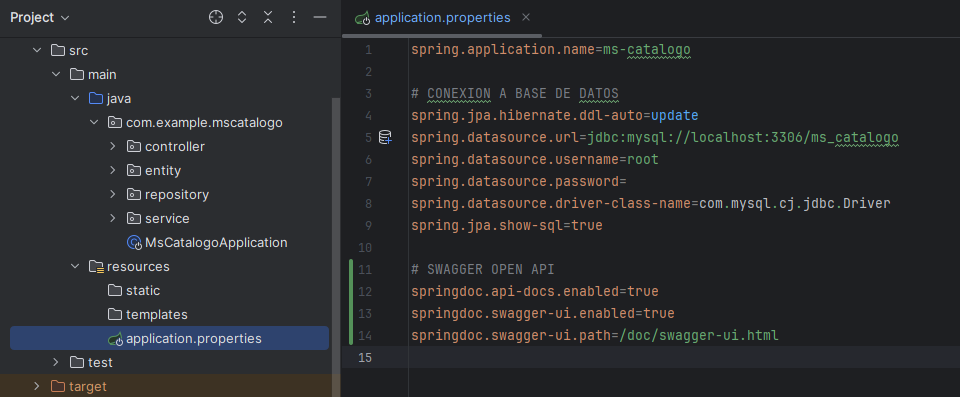
</dependency>

Ahora, debes actualizar la descarga de las librerías



Luego:

Configurar en: application.properties



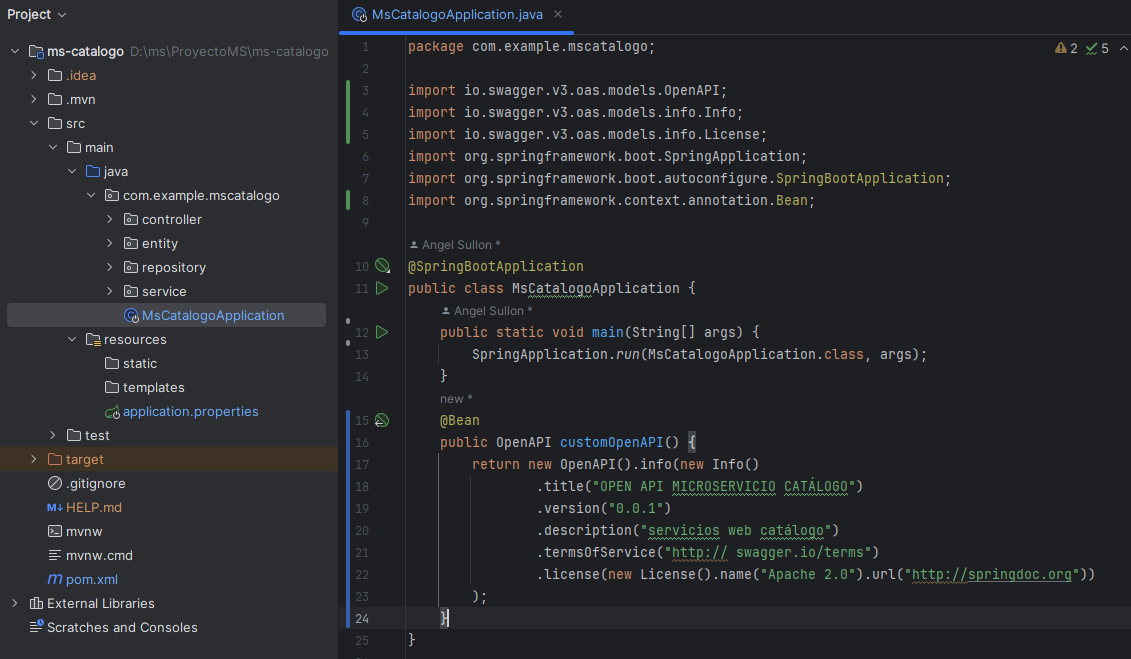
# SWAGGER OPEN API

springdoc.api-docs.enabled=true

springdoc.swagger-ui.enabled=true

springdoc.swagger-ui.path=/doc/swagger-ui.html

Configurar Bean: MsCatalogoApplication



package com.example.mscatalogo;

import io.swagger.v3.oas.models.OpenAPI;

import io.swagger.v3.oas.models.info.Info;

import io.swagger.v3.oas.models.info.License;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Bean;

@SpringBootApplication

public class MsCatalogoApplication {

public static void main(String[] args) {

SpringApplication.*run*(MsCatalogoApplication.class, args);

}

@Bean

public OpenAPI customOpenAPI() {

return new OpenAPI().info(new Info()

.title("OPEN API MICROSERVICIO CATÁLOGO")

.version("0.0.1")

.description("servicios web catálogo")

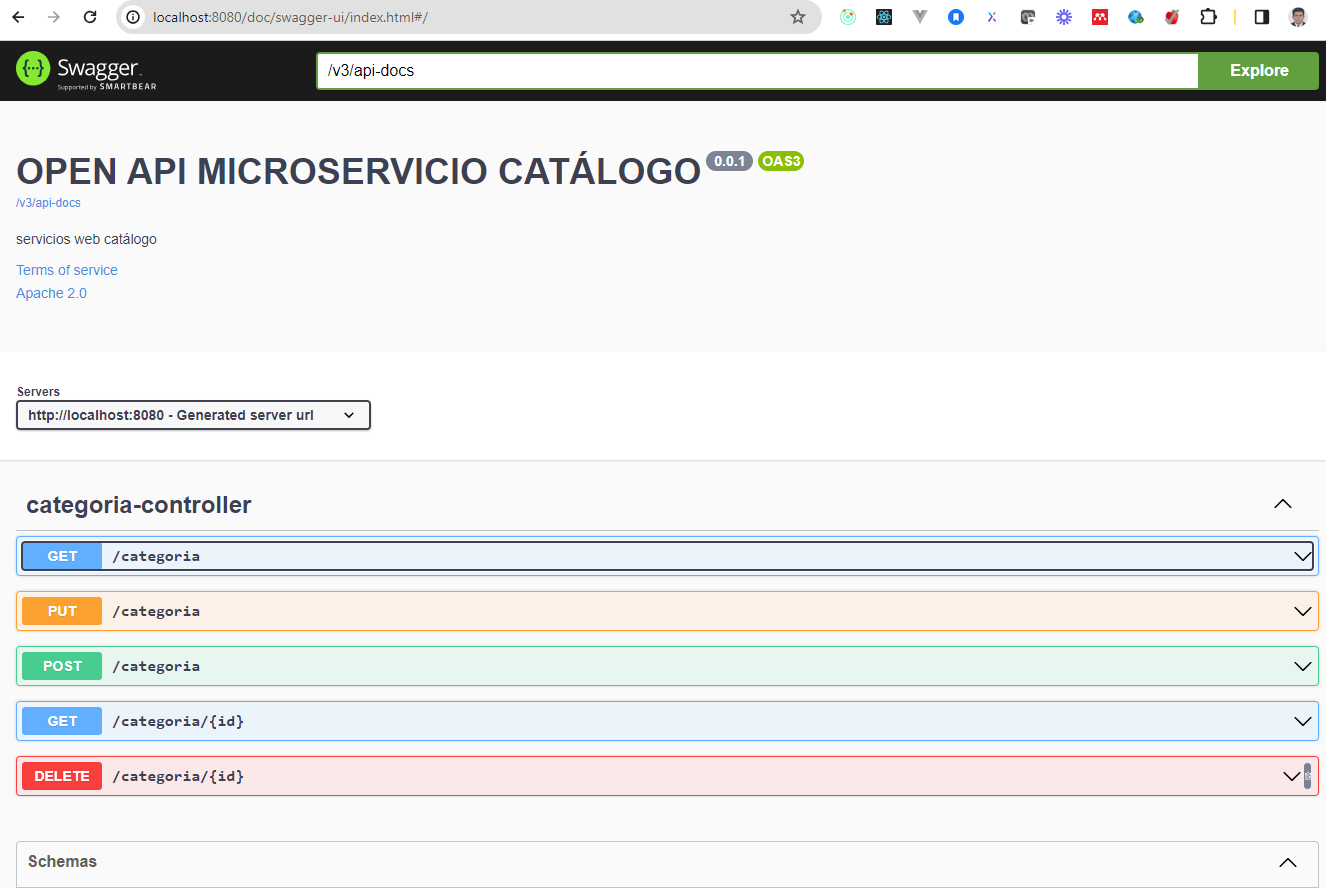
.termsOfService("http:// swagger.io/terms")

.license(new License().name("Apache 2.0").url("http://springdoc.org"))

);

}

}

<http://localhost:8080/doc/swagger-ui/index.html>

Ver código en:

<https://github.com/ms-upeu/ProyectoMS/tree/02-doc-con-swagger>

