Microservicios

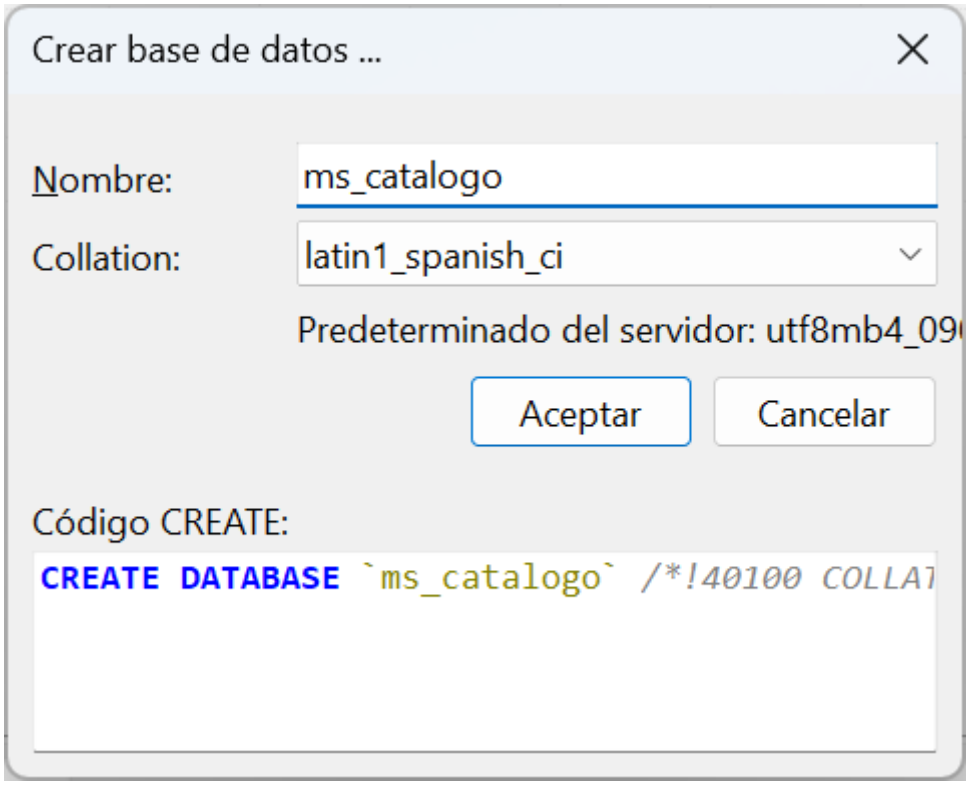
Nelson Jose Luque Mamani

Link: https://github.com/ZethGecko/DAD---01

Preparación de ambiente de desarrollo

1. **Instalación de Laragon**
2. **Instalación y configuración de Git**
3. **Instalación de Java 17 o 21**
4. **Instalación de Intellij idea**
5. **Instalación de HTTPie/Postman**

Creación de la base de datos:

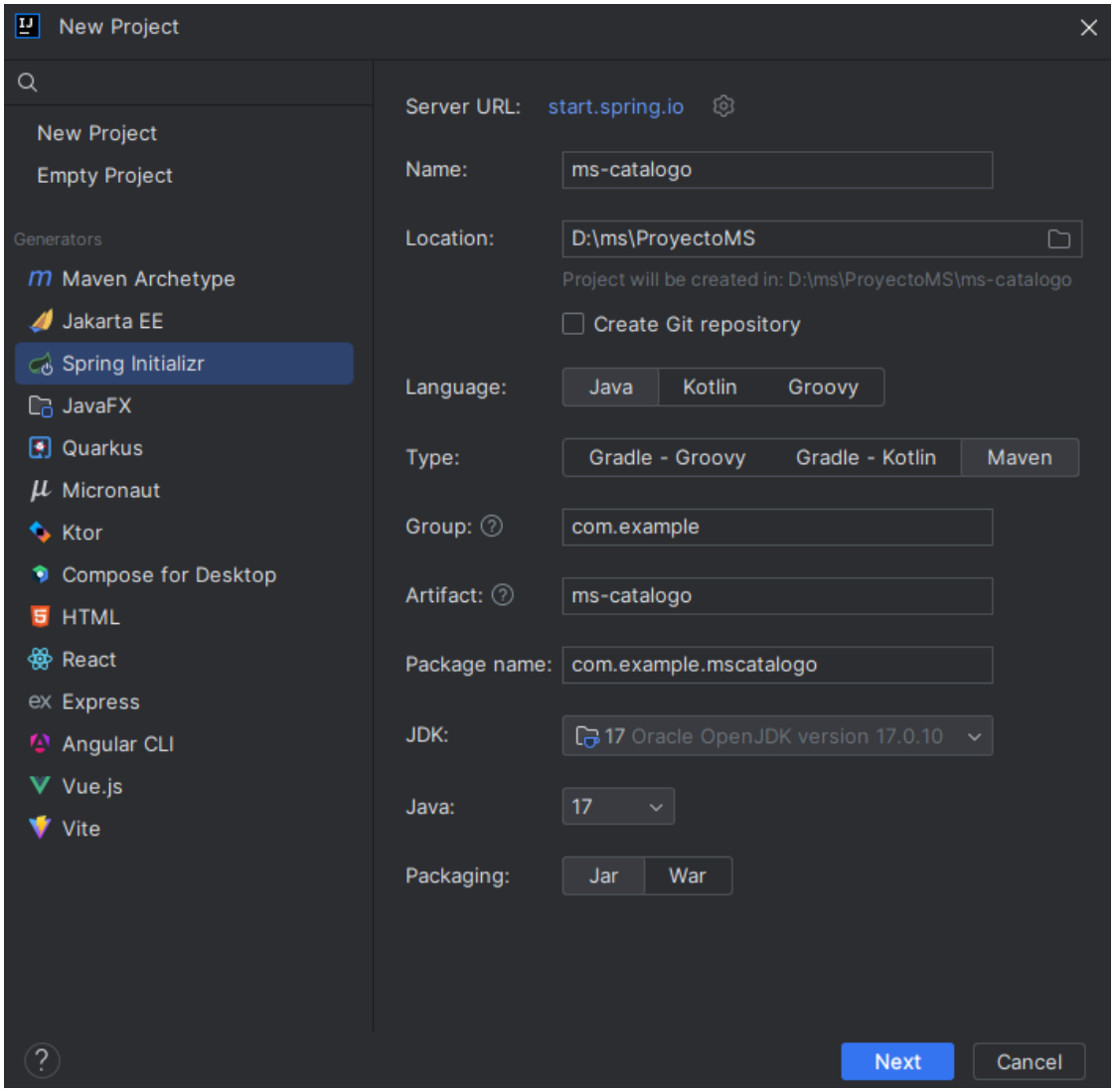


Creación de del proyecto: ms-catalogo

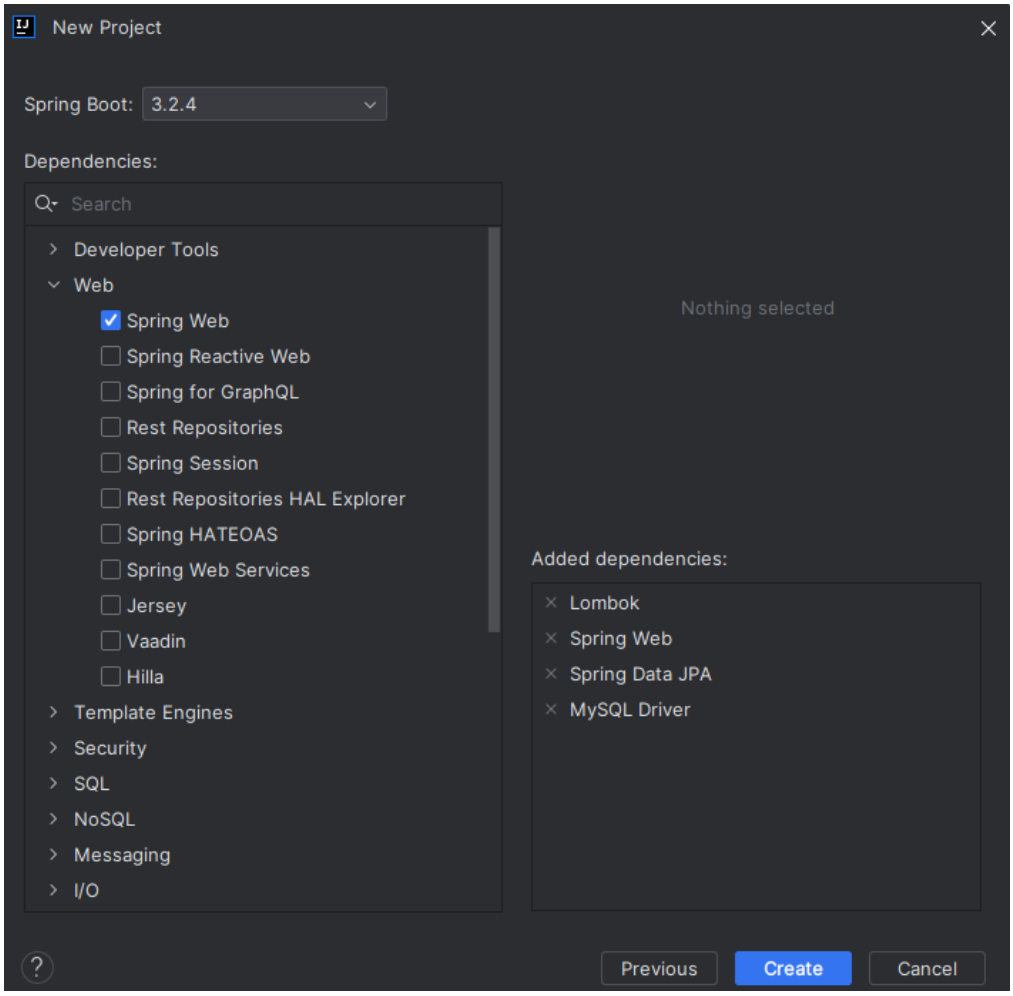
Nombre: ms-catalogo

Localización: D:\ms\ProyectoMS

Tipo: Maven



Seleccionar las librerías necesarias:

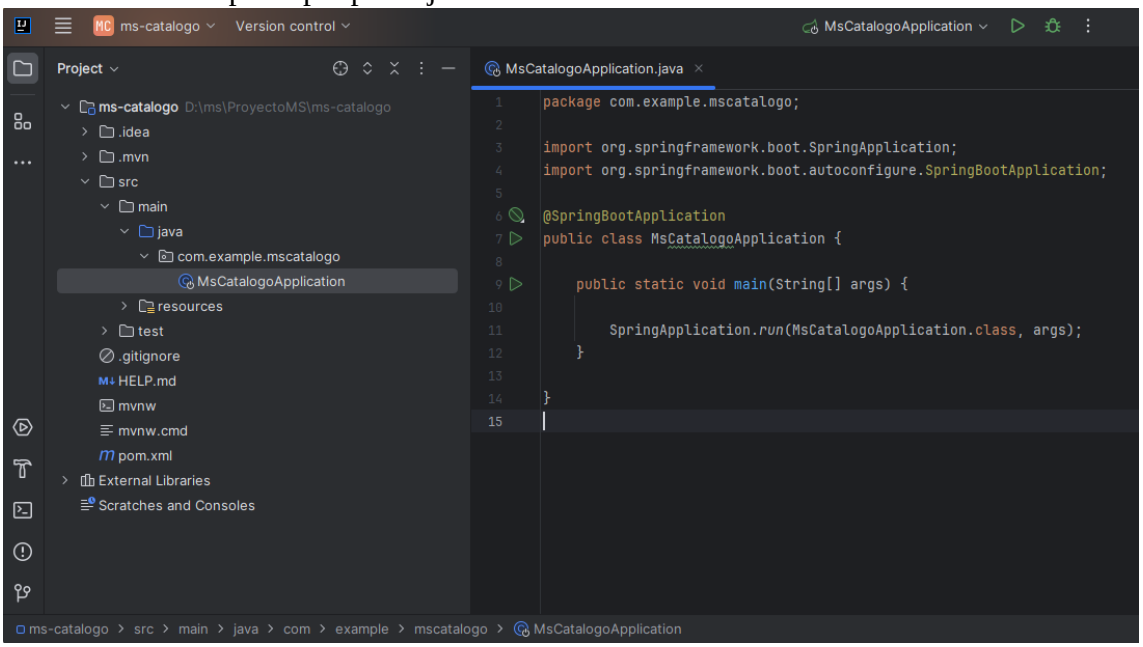


Comprobar las dependencias antes de iniciar en el POM.xml

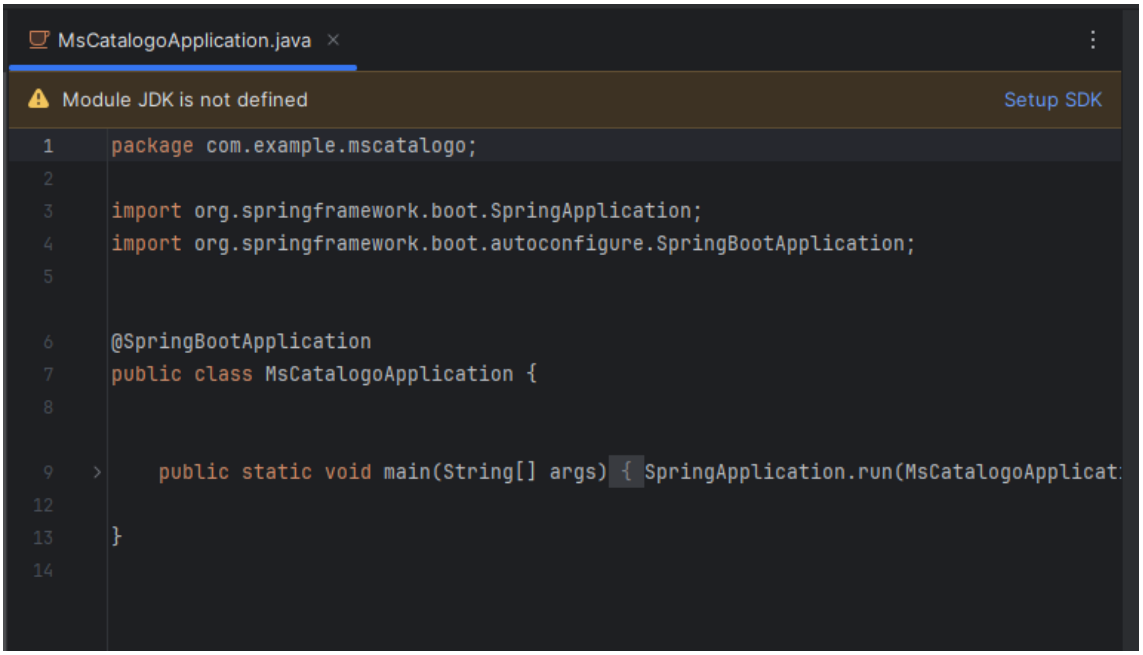
pom.xml

```
        <dependency>
            <groupId>com.mysql</groupId>
            <artifactId>mysql-connector-j</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

Habilitar la clase principal para ejecutarlo:

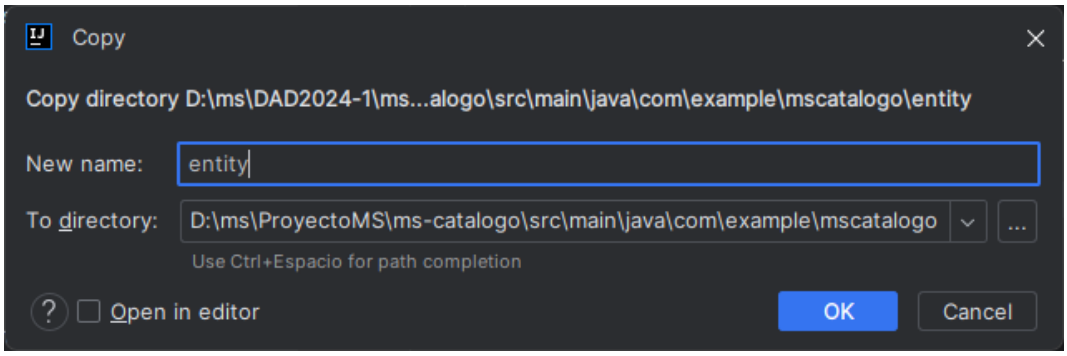


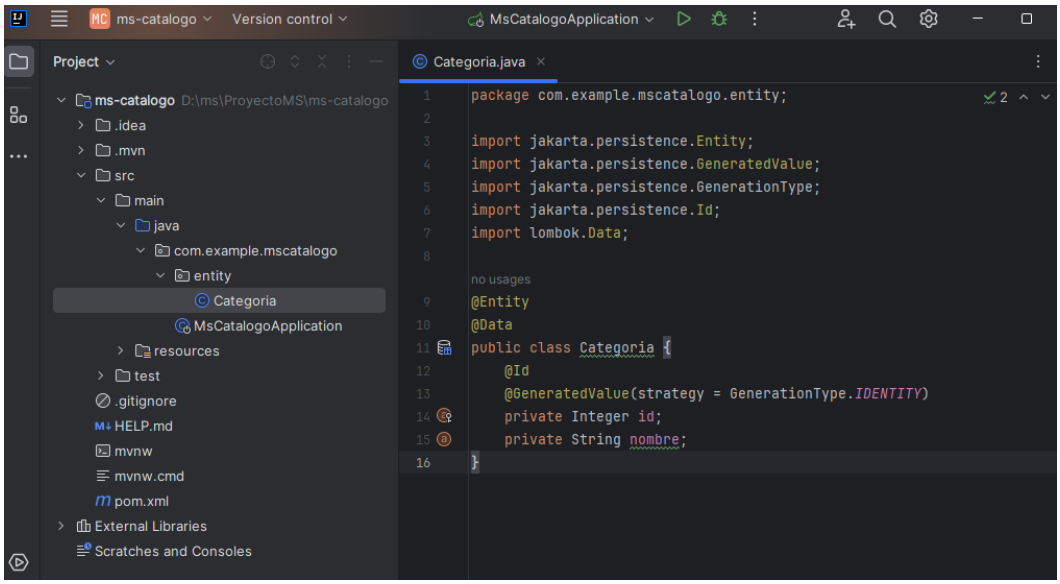
asegurarse de actualizar el IDE



Entity

Realizar la entidad clase java para representar la tabla en la base de datos, en este caso llamada catalogo:



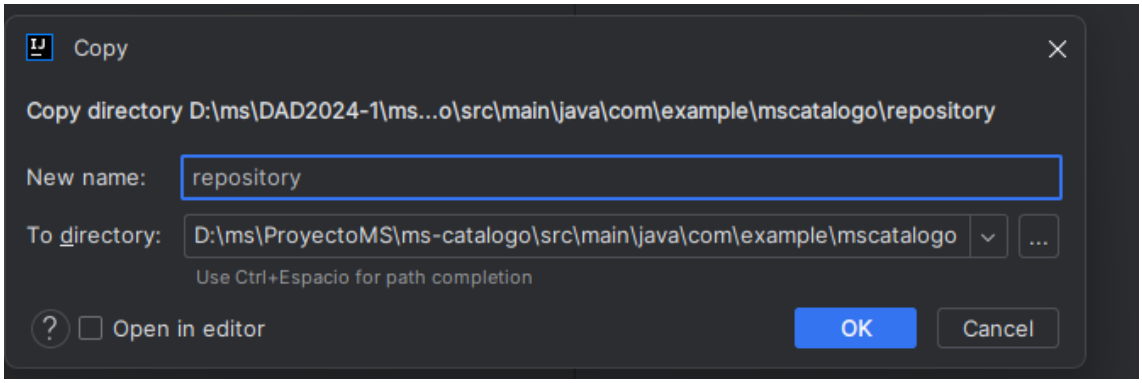



```java
package com.example.mscatalogo.entity;


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;

}
```

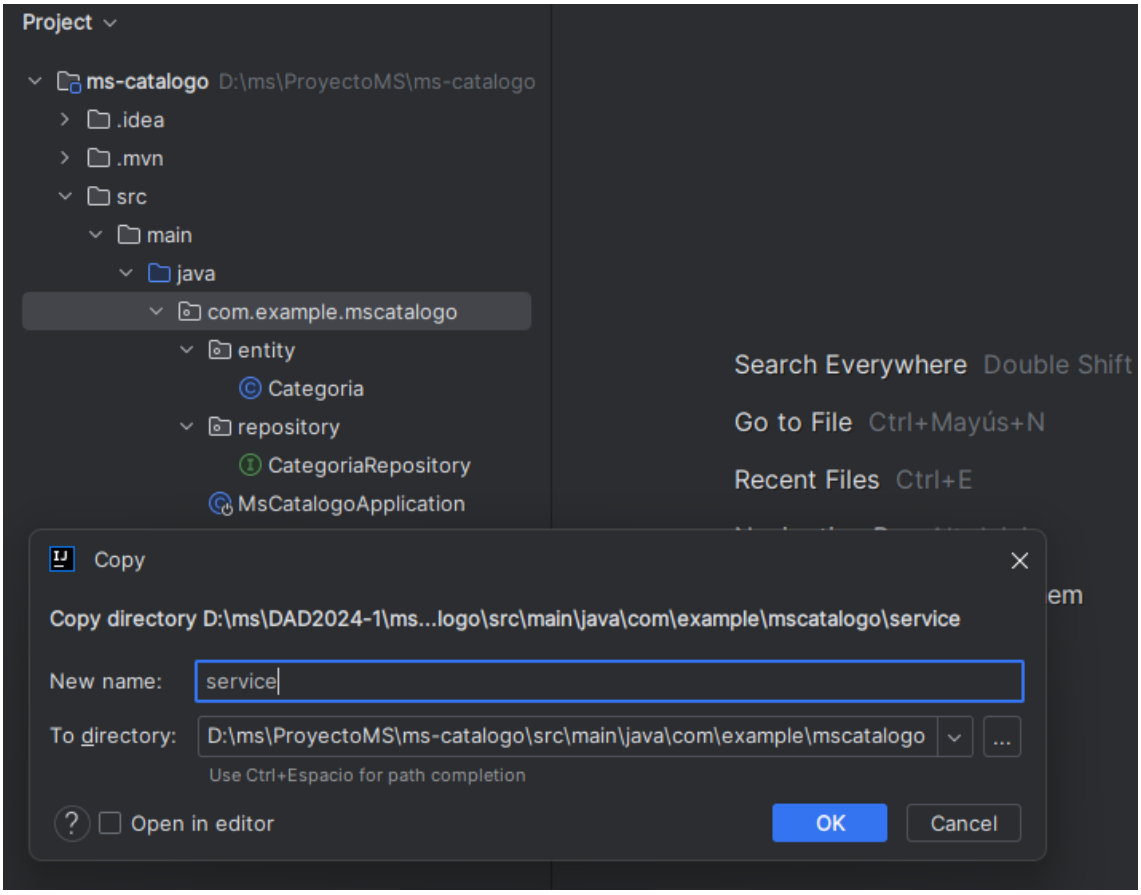
Repositorio:

Una interfaz que funciona como una capa de abstracción sobre la capa de almacenamiento de datos de la aplicación. Su propósito es ofrecer métodos y acciones específicas para interactuar con la base de datos y ejecutar 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:




```java
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);
}
```

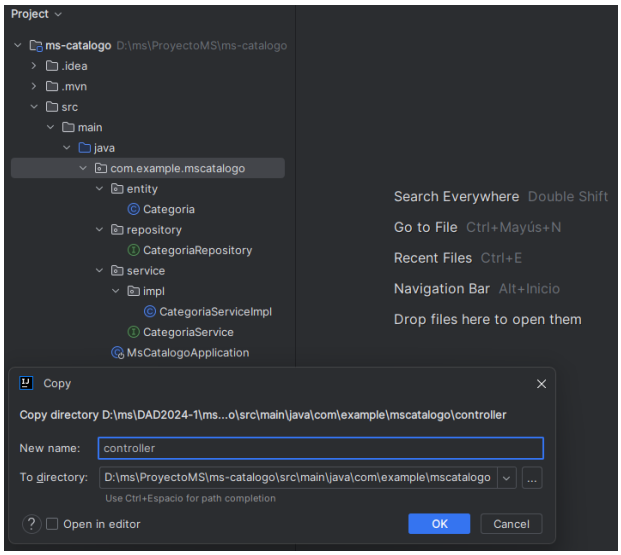
ServiceImpl

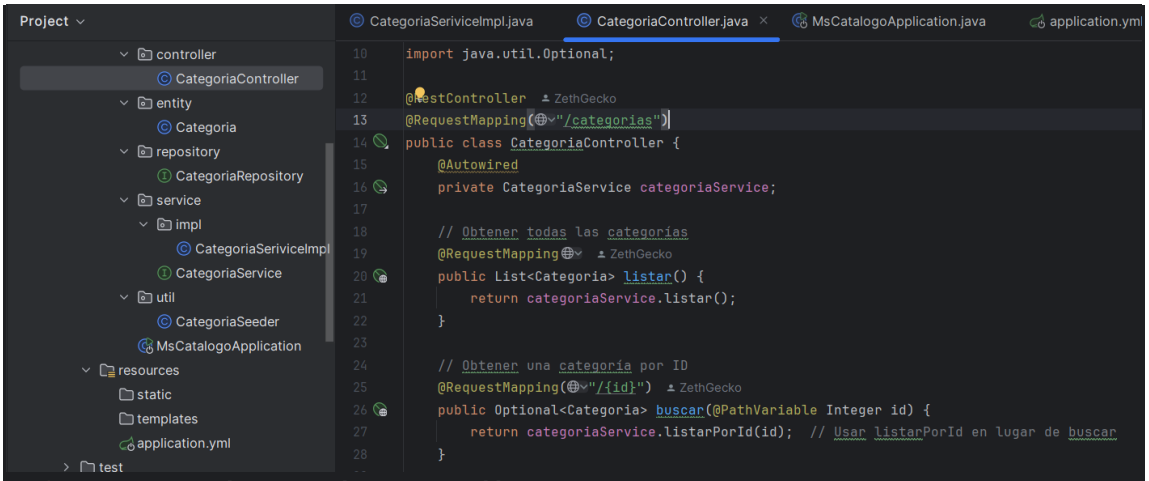

```java
package com.example.mscatalogo.service.impl;


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:

Es una clase que funciona como punto de entrada para las solicitudes HTTP provenientes del cliente (como un navegador web, una aplicación móvil, etc.). Su objetivo es gestionar las peticiones del cliente y devolver la respuesta adecuada, que puede ser en formatos como HTML, JSON, XML, entre otros.



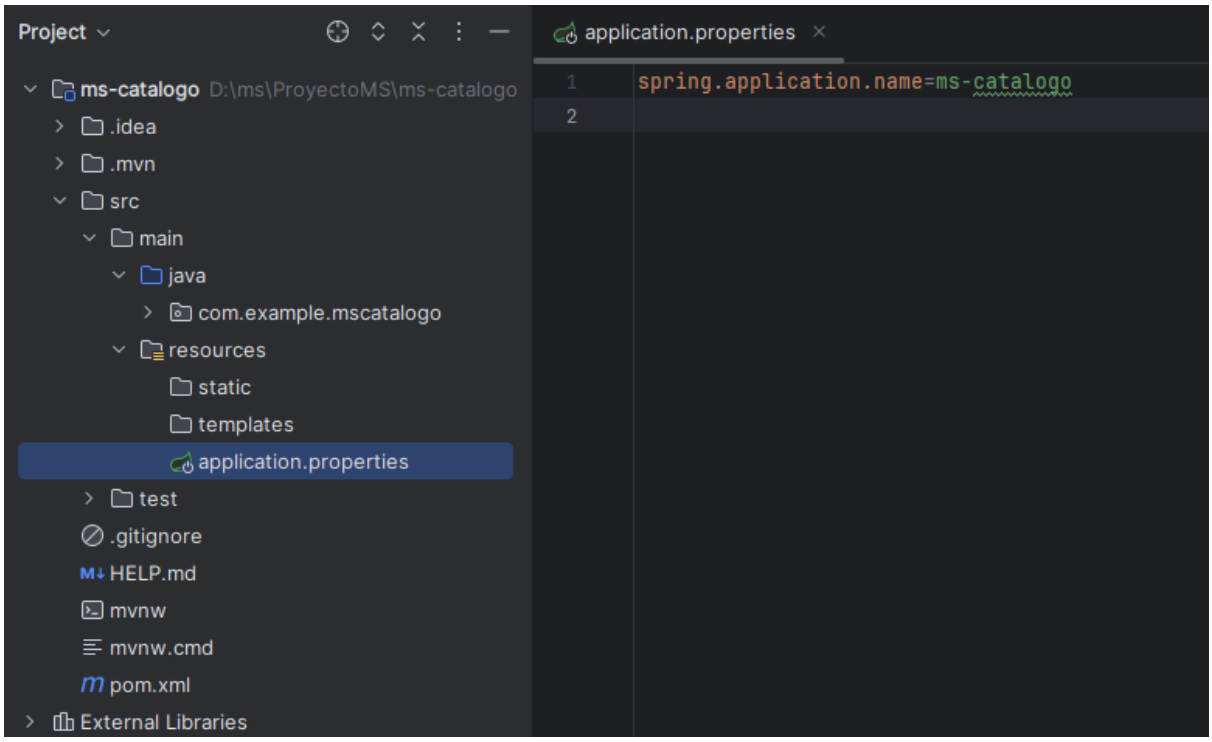



```java
package com.example.mscatalogo.controller;


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()
```

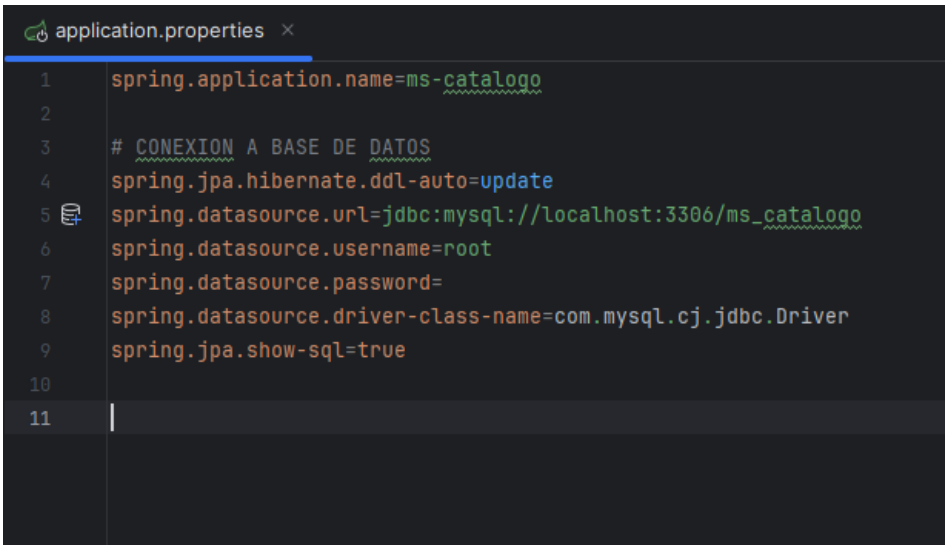
```
    public ResponseEntity<List<Categoria>> list() {
        return ResponseEntity.ok().body(categoriaService.listar());
    }
    @PostMapping()
    public ResponseEntity<Categoria> save(@RequestBody Categoria categoria){
        return ResponseEntity.ok(categoriaService.guardar(categoria));
    }
    @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 la base de datos:

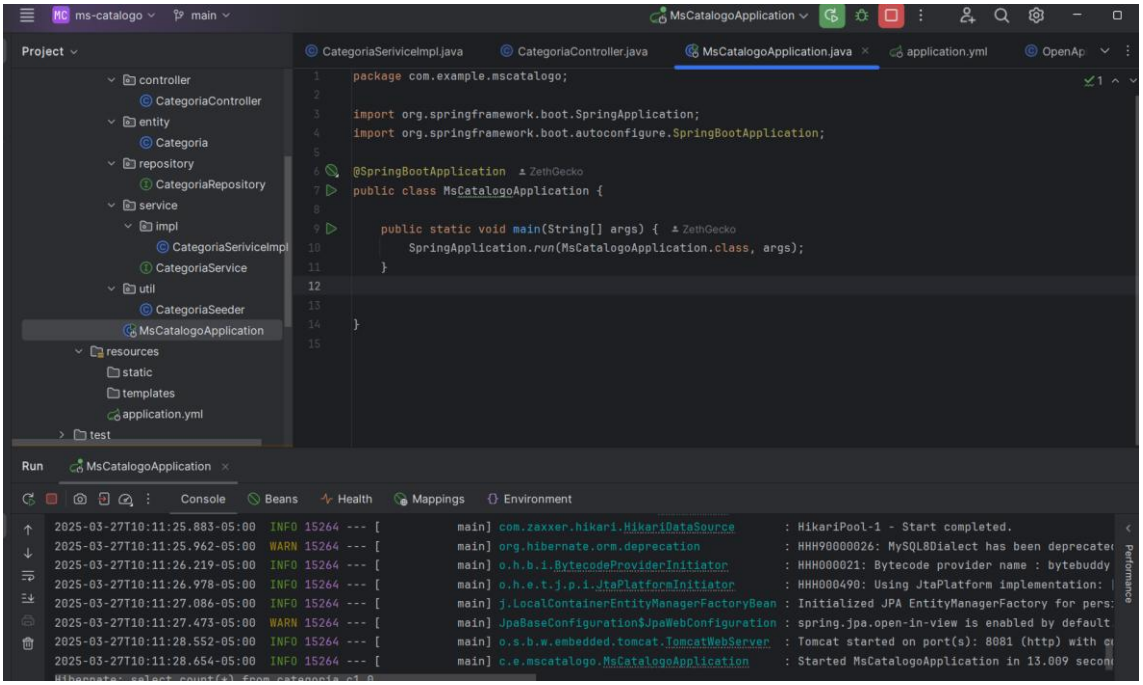
Para conectar y realizar la operación CRUD con MySQL DB con la aplicación Spring Boot, se debe configurar dentro del archivo application.properties de la siguiente manera:



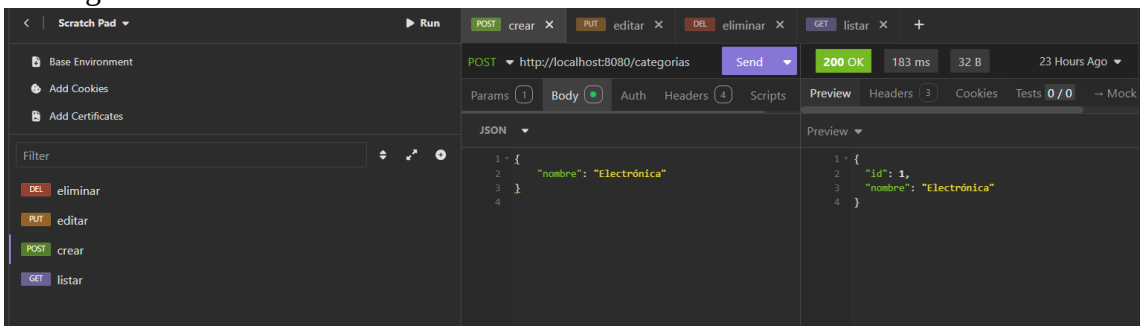
```
spring.application.name=ms-catalogo


# 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
```



Ejecutar:

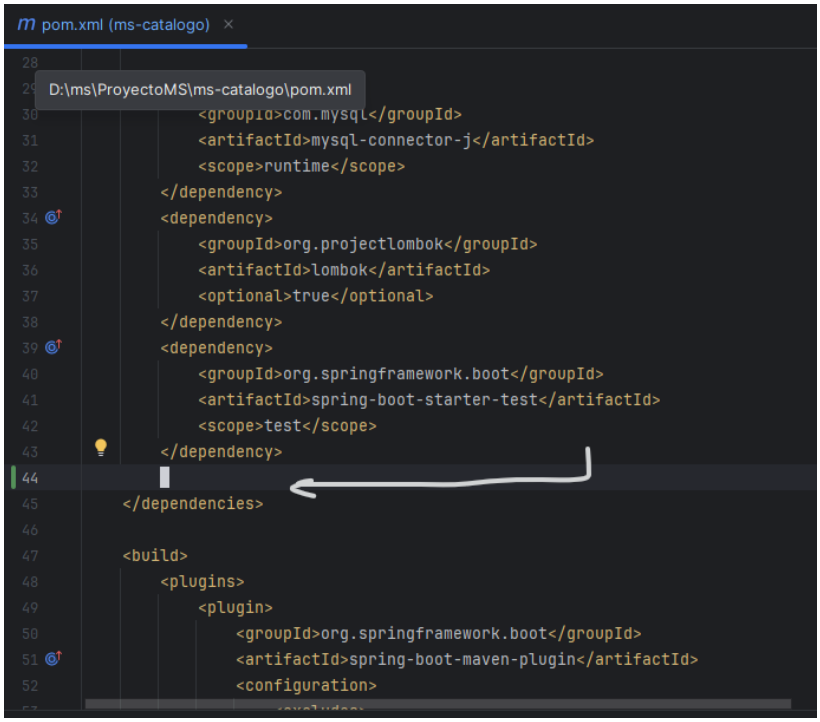


Testing:



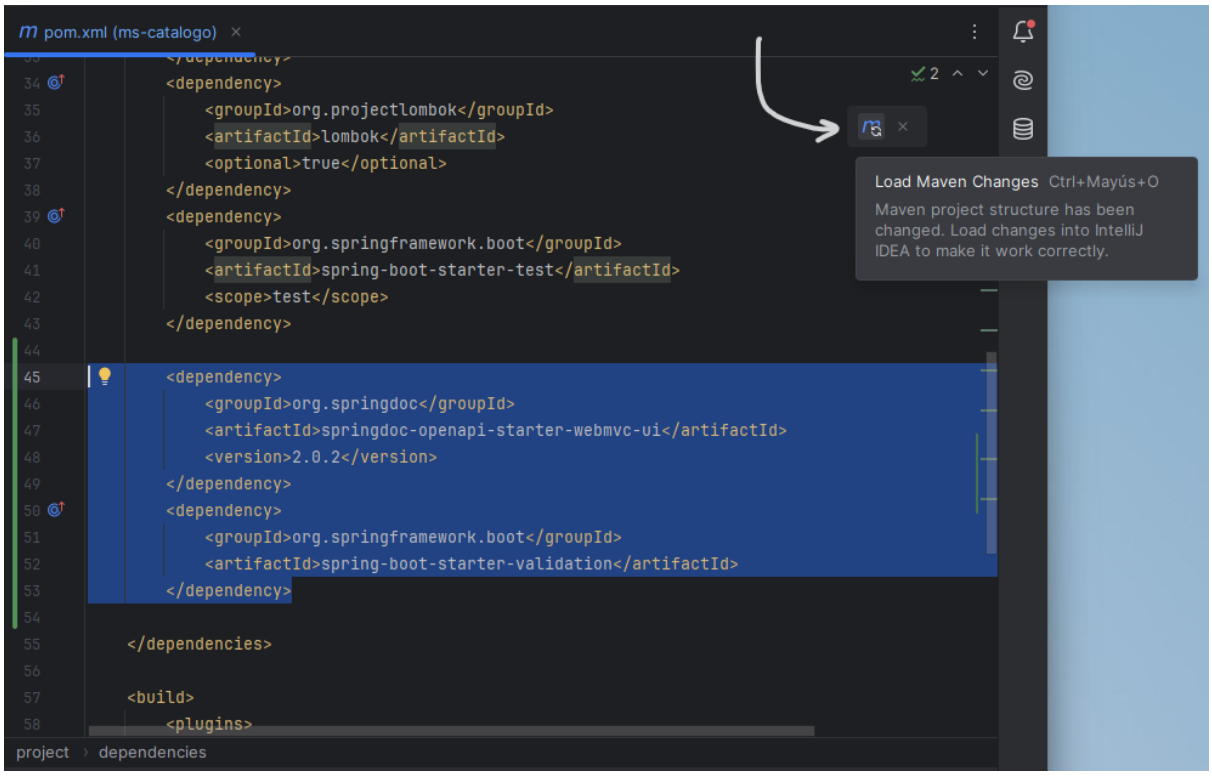
OpenAPI Specification

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

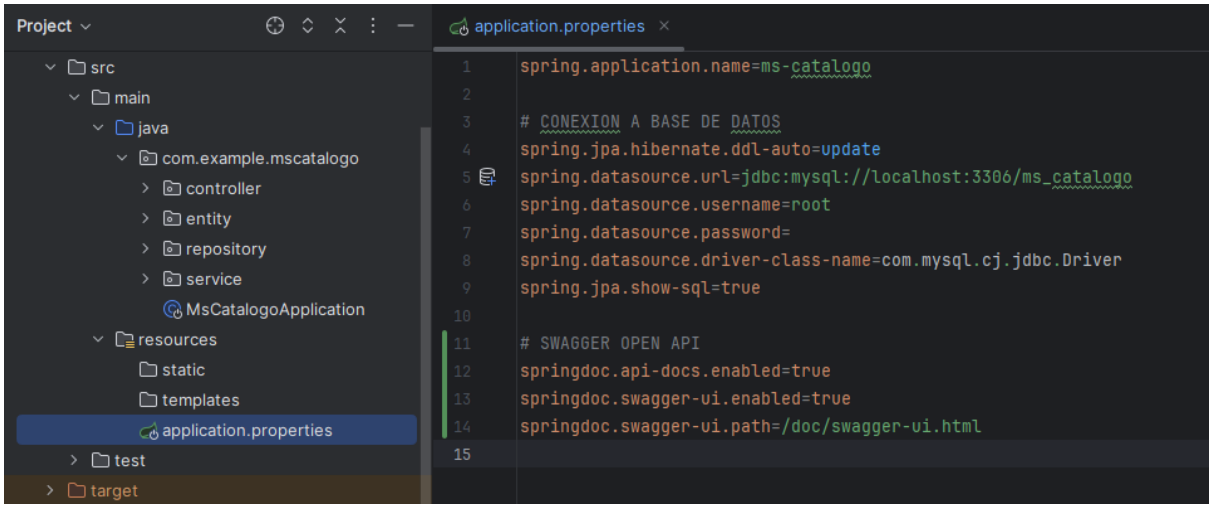



```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
    <version>2.0.2</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

Ahora, debes actualizar la descarga de las librerías

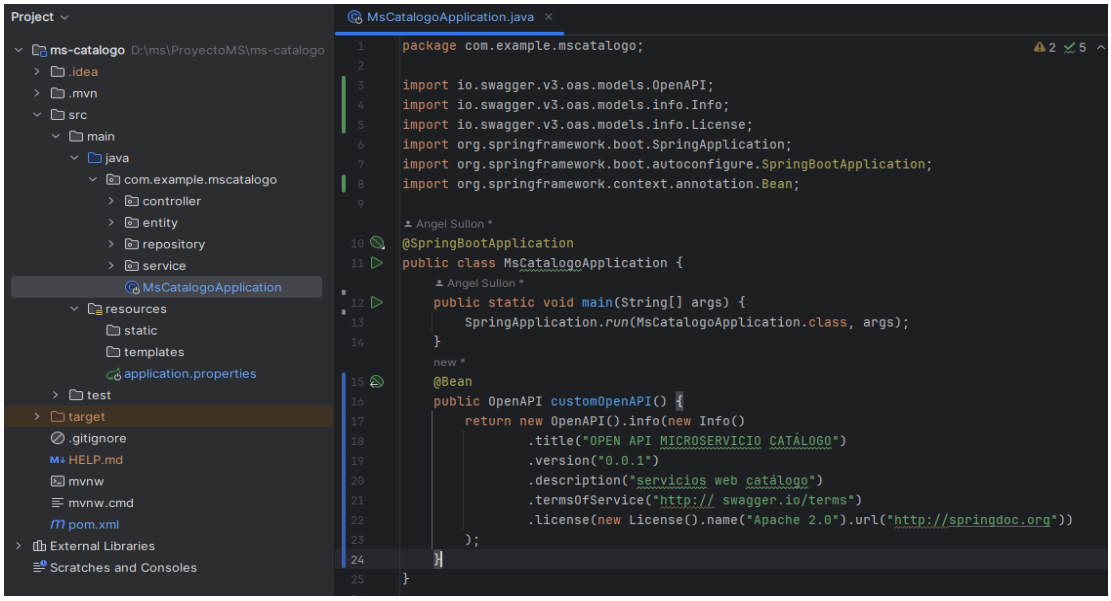


Configurar en: application.properties




```
# SWAGGER OPEN API
springdoc.api-docs.enabled=true
springdoc.swagger-ui.enabled=true
springdoc.swagger-ui.path=/doc/swagger-ui.html
```

Configurar Bean: `MsCatalogoApplication`




```java
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"))

       );

   }

}
```

Swagger

/v3/api-docs

Explore

OPEN API MICROSERVICIO CATÁLOGO 0.0.1 OAS3

/v3/api-docs

servicios web catálogo

Terms of service

Apache 2.0

Servers

http://localhost:8080 - Generated server url

categoria-controller

GET /categoria

PUT /categoria

POST /categoria

GET /categoria/{id}

DELETE /categoria/{id}

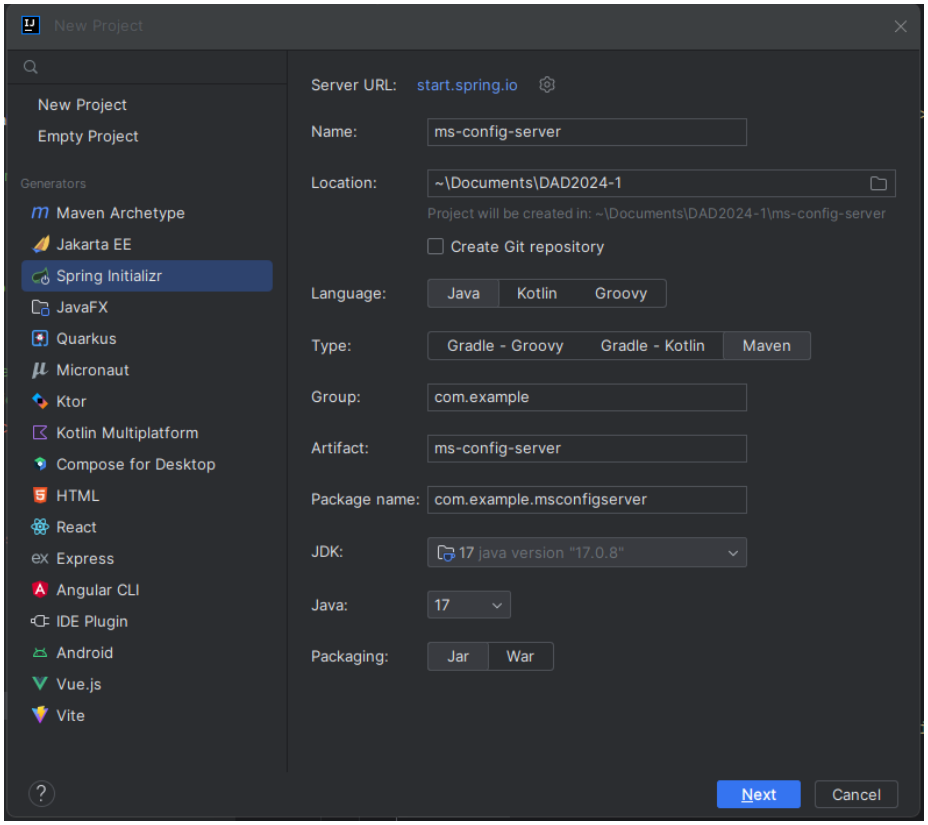
Schemas

Llevar de monolito a microservicios:

Crear el ms-config-serve:

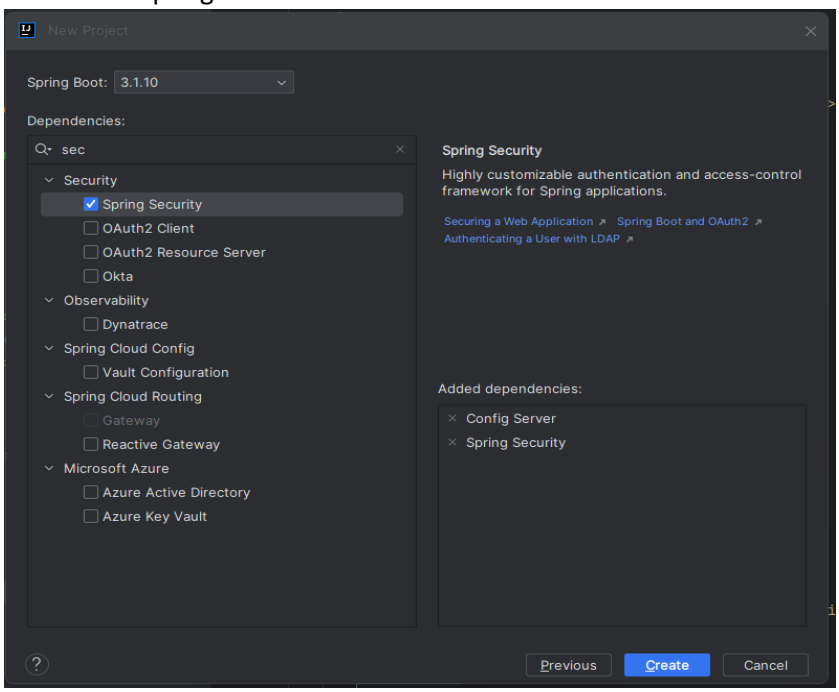
Nombre: ms-config-server

Tipo: Maven



Seleccionar las librerías:

Versión de Spring Boot: 3.1.10

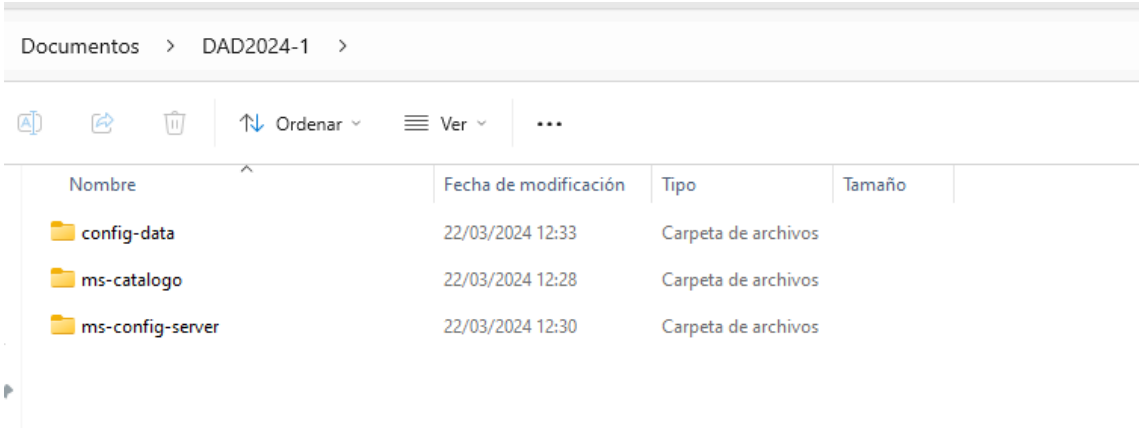


Cambiar la extecion de application.properties en el proyecto config-server por application.yml

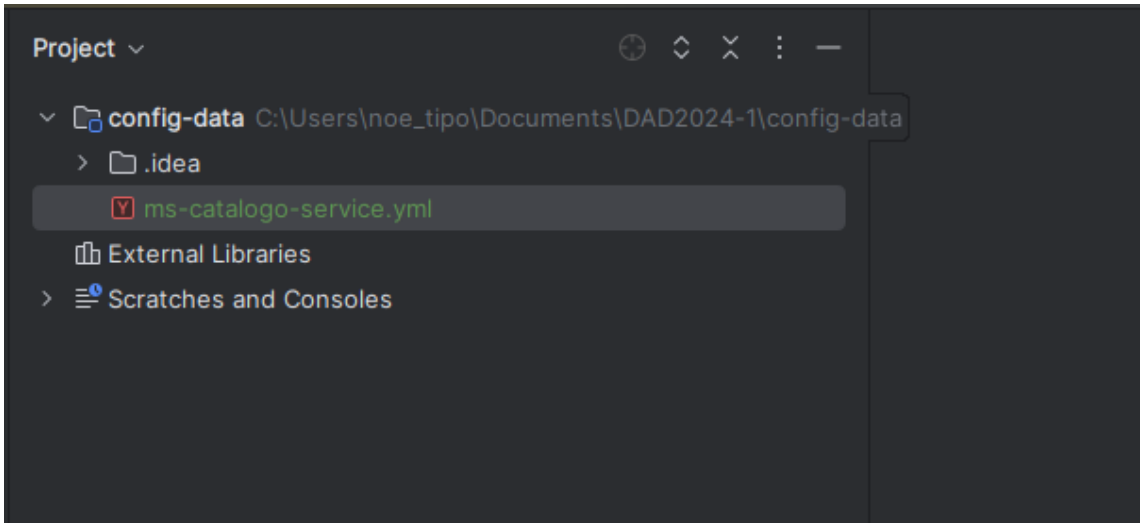
```yaml
server:
  port: 7070

spring:
  application:
    name: config-server
  cloud:
    config:
      server:
        git:
          uri: https://github.com/ZethGecko/DAD---01.git # ruta del
repositorio git
          searchPaths: config-data
          default-label: main

  security:
    user:
      name: root
      password: password
```

Agregar la anotación @EnableConfigServe en **MsConfig Server Application**

En la misma ruta crear una nueva carpeta config-data:



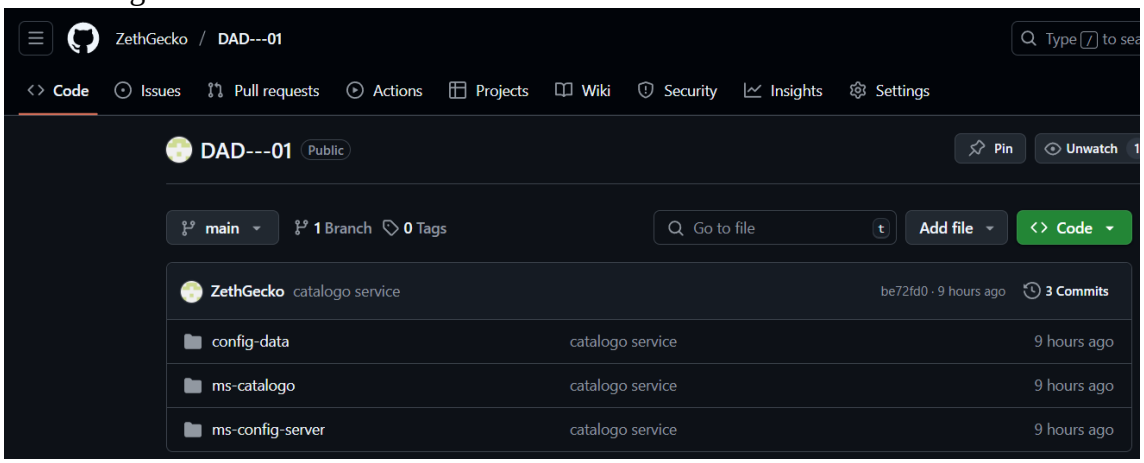
Editar el archivo ms-catalogo-service.yml



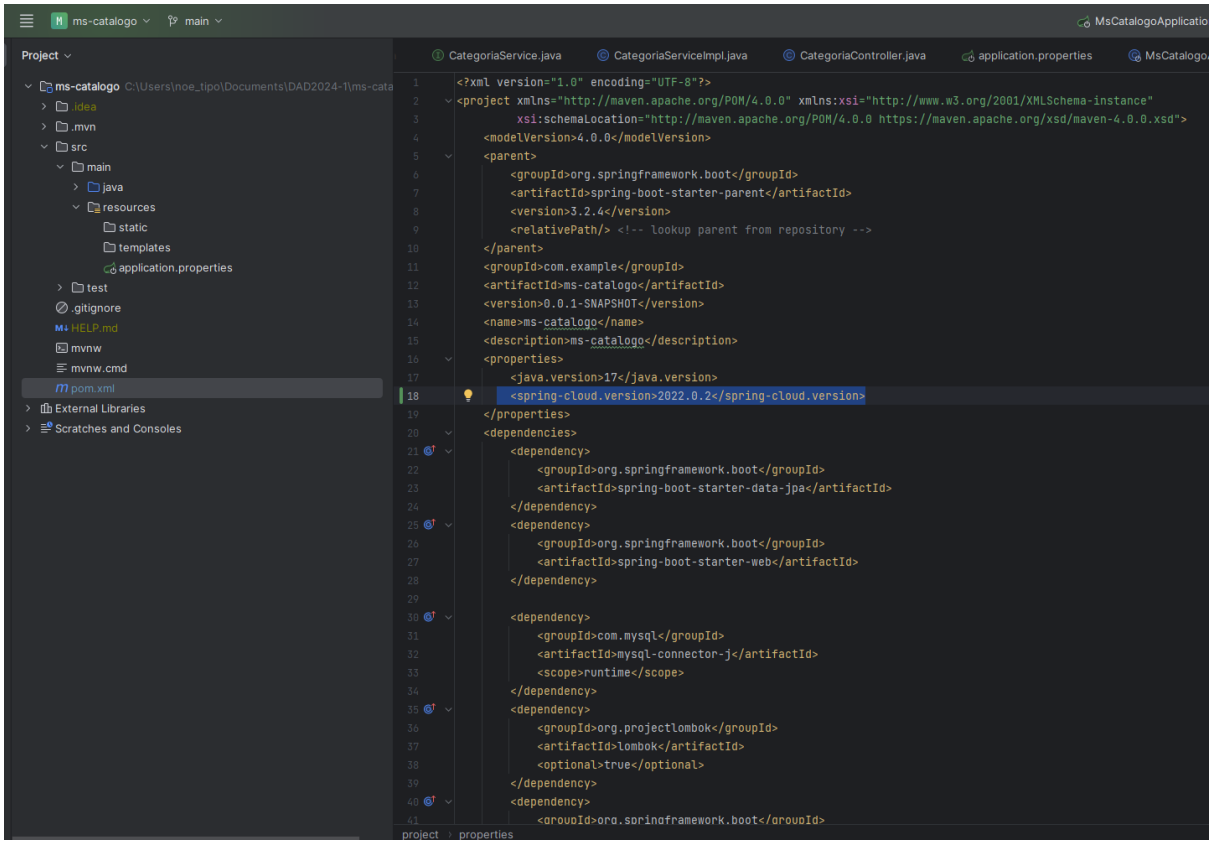
código para copiar y pegar

```
server:
 port: 8081
springdoc:
 api-docs:
   enabled: true
 swagger-ui:
   enabled: true
   path: /doc/swagger-ui.html
```

Subirlo a github todo los archivos:



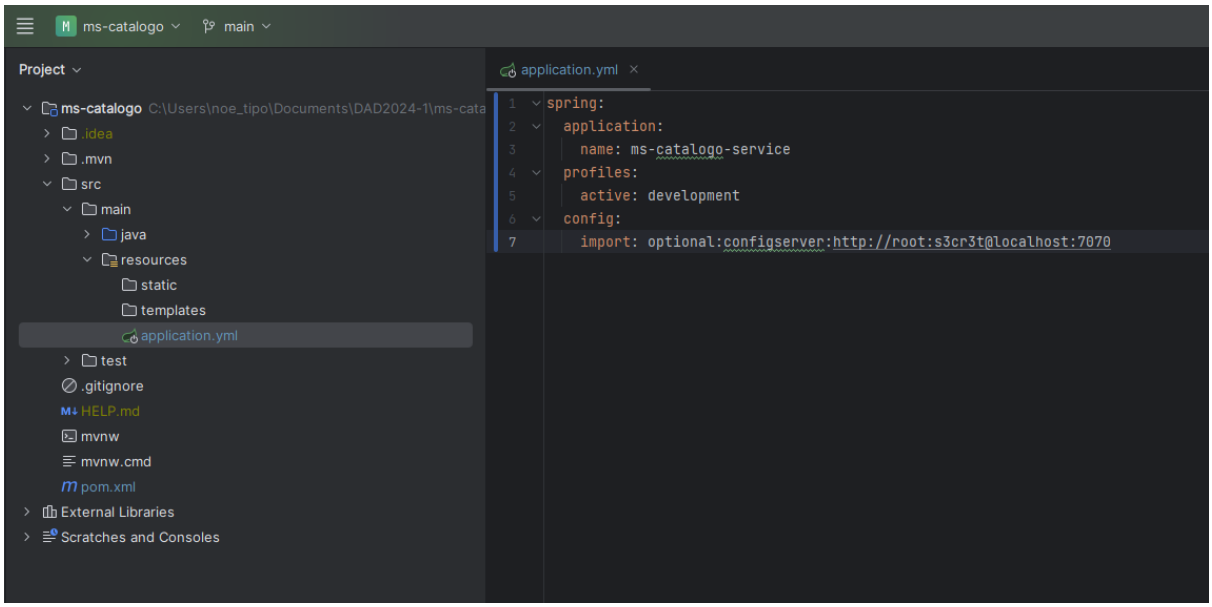
Luego agregar la versión de spring cloud en el proyecto ms-catálogo



código para copiar y pegar

```
<spring-cloud.version>2022.0.2</spring-cloud.version>
```

editar el archivo application.yml del proyecto application.yml



código para copiar y pegar

```
spring:
 application:
   name: ms-catálogo-service
 profiles:
   active: development
 config:
```

resultados finales: