**SPRING BOOT**

**FACULTAD: INGENIERÍA Y CIENCIAS BÁSICAS**

**SYLLABUS: PROGRAMACIÓN AVANZADA**

**SEDE: MOCOA**

**DOCENTE: FREDY JACANAMIJOY**

**SEMESTRE: 7**

**FECHA:**

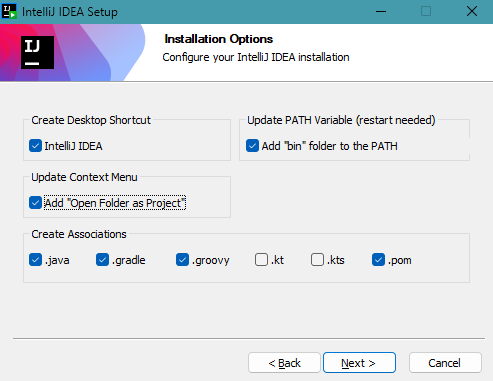
3204414573 – [fredy.jacanamijoy@itp.edu.co](mailto:fredy.jacanamijoy@itp.edu.co)

**Recomendaciones para iniciar a trabajar con spring boot**

* Tener instalado el JDK de java con la versión 17 o cualquiera que sea superior, excepto la última porque la compatibilidad con spring boot aún suele estar en desarrollo.
* Usar **Intellij IDEA** (Comunity edition) preferiblemente.

Link de descarga: <https://www.jetbrains.com/es-es/idea/download/?section=windows>

En la instalación preferiblemente maneja la siguiente configuración.

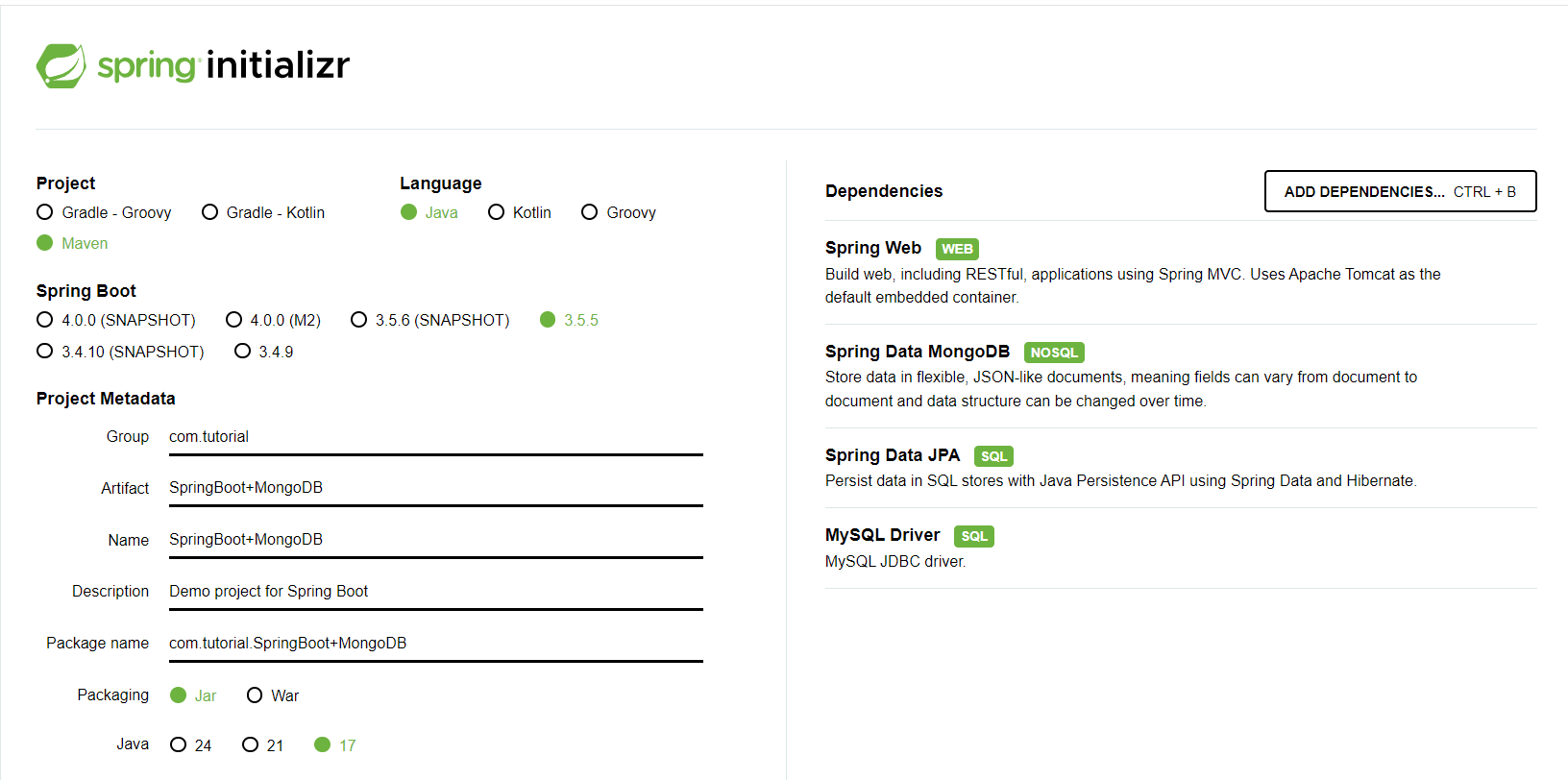




* Se puede usar visual studio code, con extensiones apropiadas u otros IDE tales como Net Beans o Eclipse, pero requieren muchas configuraciones.

**Pasos iniciales:**

* Nos dirigimos al sitio: <https://start.spring.io/> y preparamos el paquete de descarga de acuerdo a la siguiente imagen:

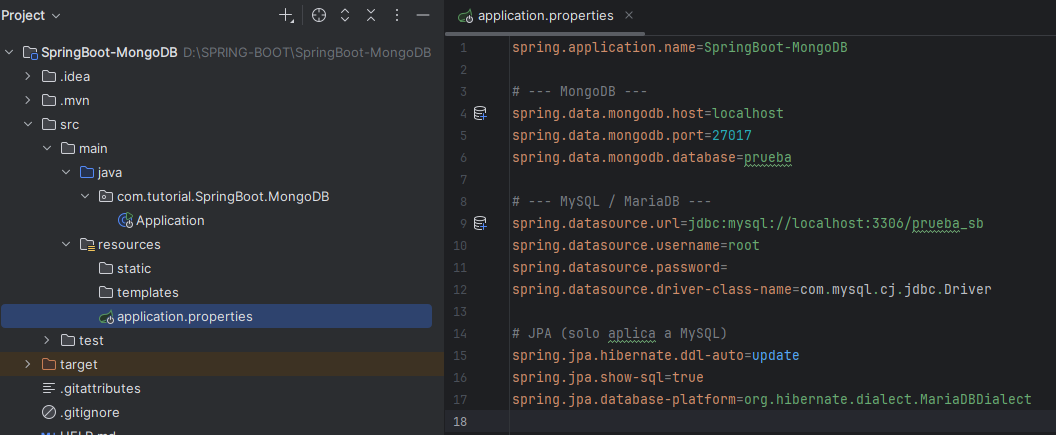


* Le damos en el botón generate, para que descargar un proyecto base:

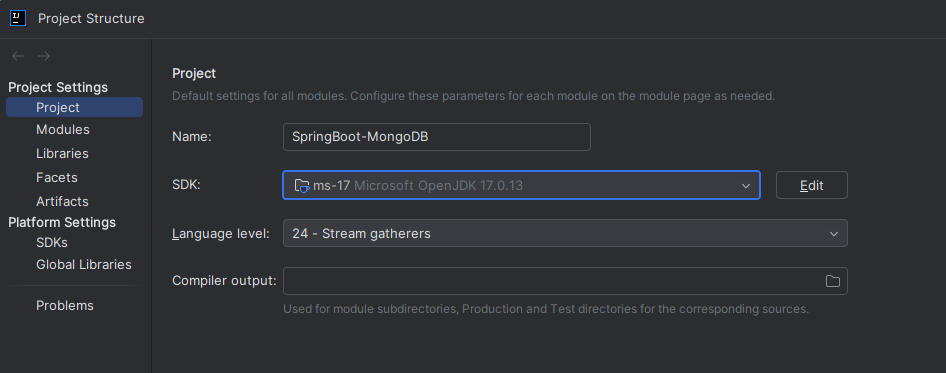


* El paquete comprimido lo ubicamos en una ruta deseada de trabajo.

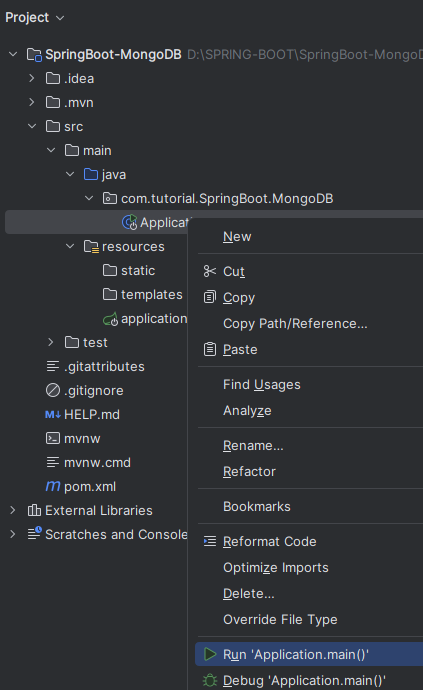
Con el explorador de archivos de Intellij Idea, abrimos el proyecto en la ruta que escogimos y en la ruta que describe la imagen ubicamos el archivo **applications.properties** y escribimos la configuración descrita en la imagen (Suponemos que existe en mongo existe una BD de nombre prueba).



Usa la combinación de teclas: **Ctrl + Alt + Shift + S** para generar la siguiente ventana, en la cual SDK: debe tener seleccionada la versión de java que tienes instalada. Para saber qué versión se tiene instalada, en cmd o powershell ejecuta el comando **java -version**

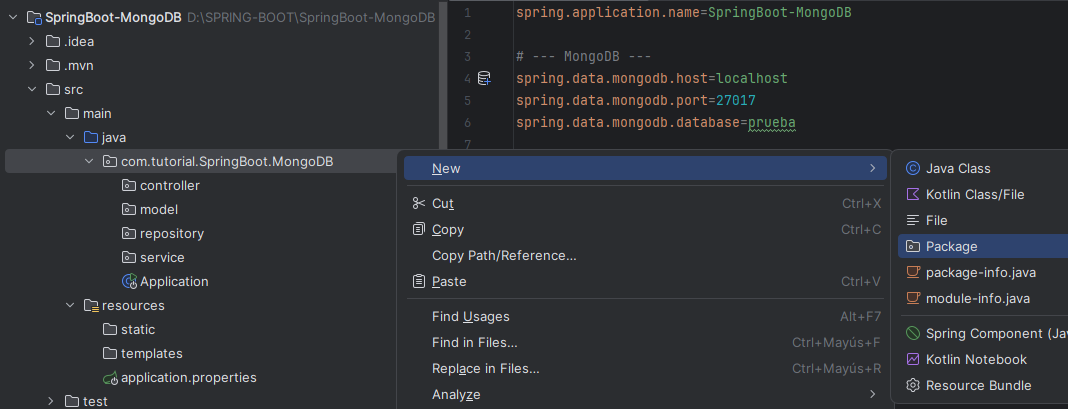


Dale click derecho en el archivo **Applications**, ubicado en la ruta que señala la imagen.

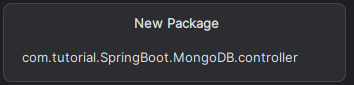


Si no da ningún error, se spring boot se ha conectado correctamente a mongo y a mysql

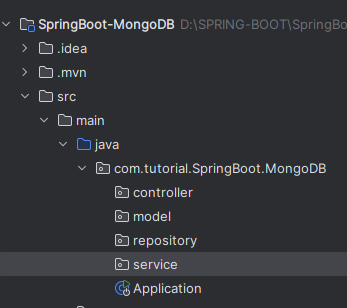
Dentro de la ruta señalada en al imagen creamos 4 paquetes (Package) o carpetas con los siguientes nombres: **controller**, **model**, **repository** y **service.**



Después de darle en la opción **Package** de la anterior magen, aparecerá una sección para dar el nombre a la nueva carpeta: en este caso a continuación se se ha nombrado **controller**.



La correcta creación de los paquetes debe reflejarse así:

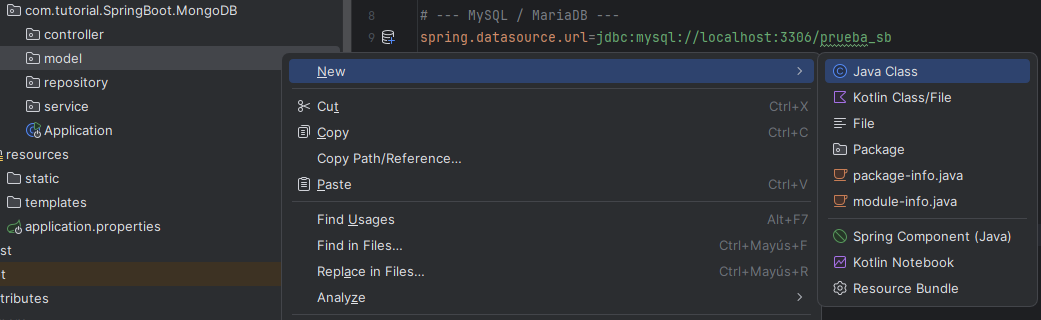


# CREACIÓN DE ENTIDADES

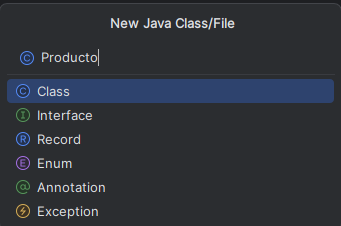
Las **entidades** están asociadas a las **colecciones** en mongo o a **tablas** en mysql. Cada entidad se gestiona mediante una **clase** y al conjunto de todas las clases se le llama **modelo de dominio.** Cada entidad representa el ser de un objeto con sus atributos, los cuales son los campos ya sea dentro de una tabla(mysql) o una colección(mongo). A su vez cada objeto es gestionado y configurado con una clase, la cual es como el molde o plantilla de este, por lo cual se conectará de forma coherente con la base de datos independientemente de si es relacional o no relacional.

Si en mysql tenemos por ejemplo una tabla llamada persona, se debe crear la clase persona. Así mismo si en mongo tenemos una colección llamada categoría, también se debe crear una clase llamada categoría. Las clases se las crea dentro de la carpeta o paquete(Package) llamado model.

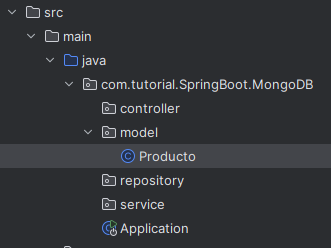
**Creación de una entidad llamada “Producto”**

Se inicia la creación de una clase que llevará el nombre de la entidad en este caso “Producto”. Para ello en el paquete model damos click derecho en model y seleccionamos Java Class.

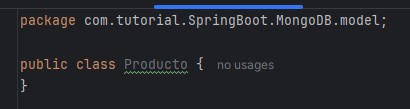
La clase se llamará producto, toda clase se nombra con la primera letra en mayúscula.



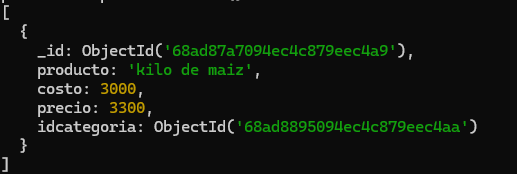
Se debe evidenciar el nuevo archivo dentro de la carpeta model así:



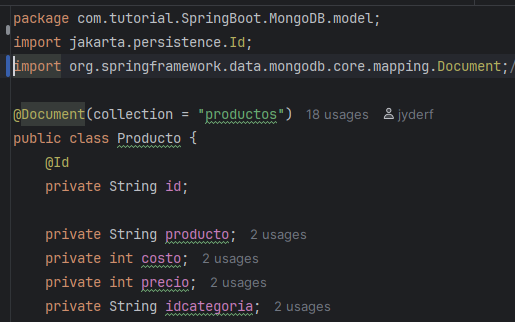
El contenido del archivo estará como se muestra a continuación:



Se debe tener en cuenta los campos de la colección en mongo: en este caso la colección tiene los campos, \_id, producto, costo, precio y idcategoria.



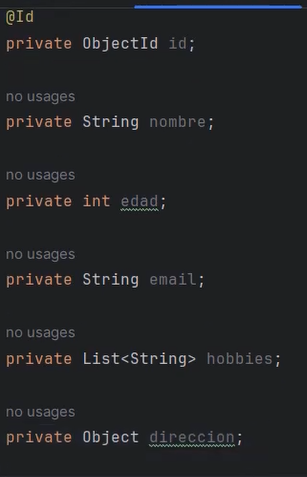
Basándose en los campos anteriormente descritos, la clase quedaría configurada de la siguiente forma:



No olvidar poner la importación indicada y borrar lo que no se vaya a emplear.

En el caso de que un campo sea un array o algo embebido que es propio de mongo, a la derecha vemos un ejemplo de ello:

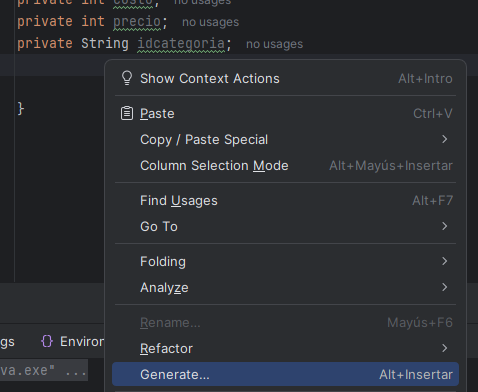
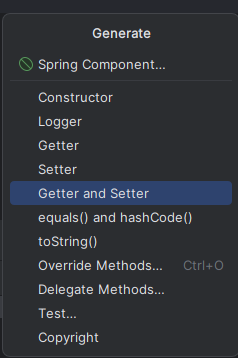
El campo llamado **hobbies**, es un array por ello la variable es tipo lista. Más abajo el campo dirección dentro de mongo está almacenando algo embebido, por ello es de tipo objeto.



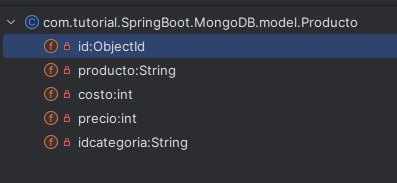
En mongo estaría de la siguiente forma:



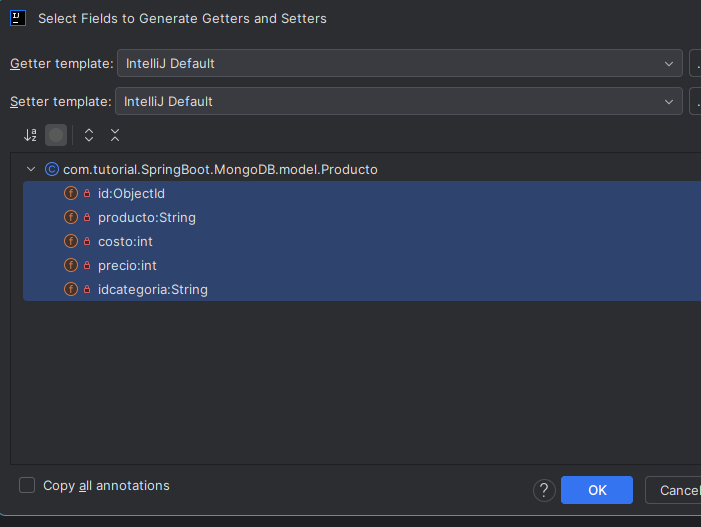
Le damos click derecho sobre una parte de la clase dentro de las llaves y le damos la opción Generate, para crear los **getters y setters.**



Por defecto suele estar señalado solo uno:

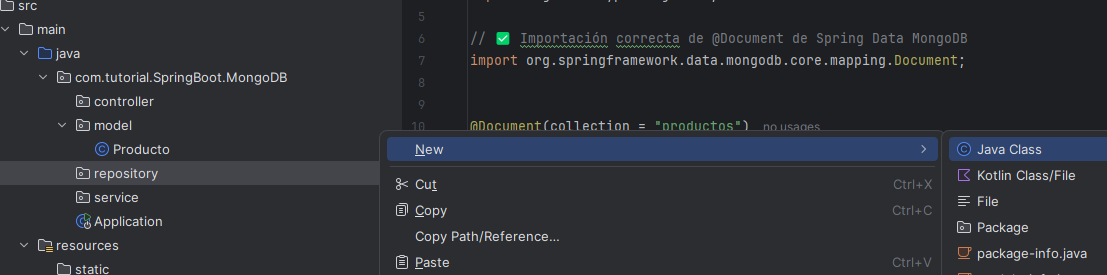


Se debe seleccionar todos con shift y flecha hacia abajo. Luego de damos en ok.

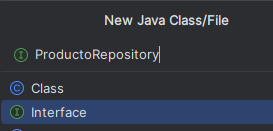


# CREACIÓN DE REPOSITORIOS

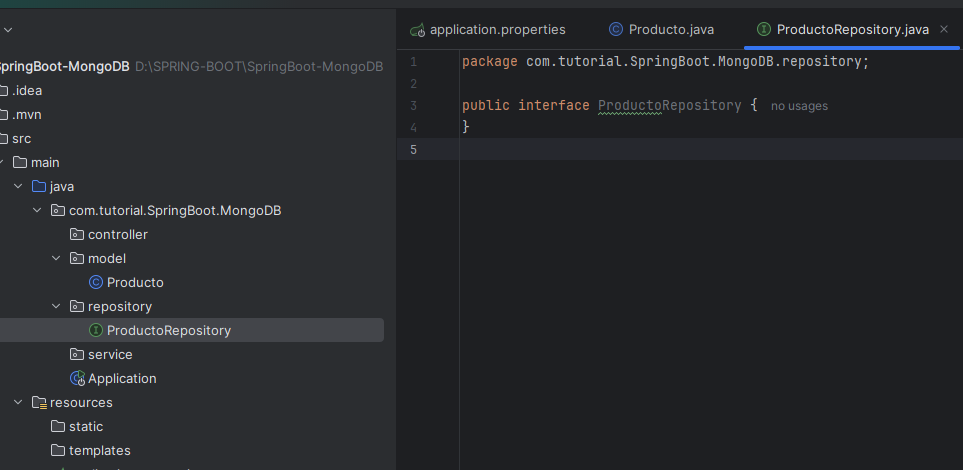
Dentro de la carpeta **repository** crearemos un archivo llamado **ProductoRepository.**



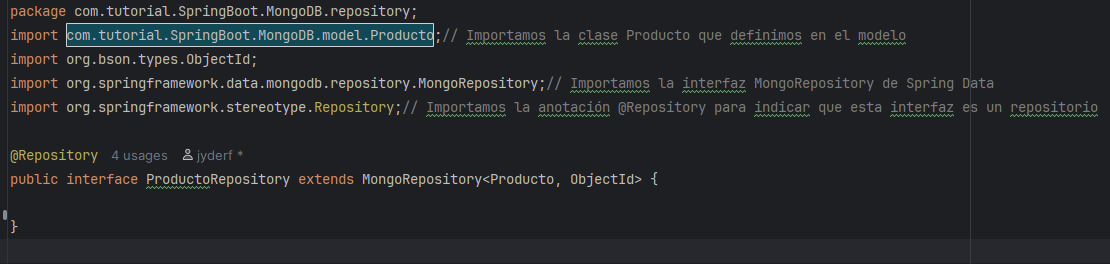
Nos aseguramos que tenga el nombre **ProductoRepository** y que tenga la opción **Interface**



Inicialmente el archivo estará así:

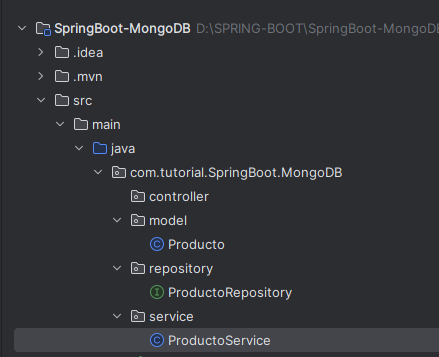


La configuración quedará así:

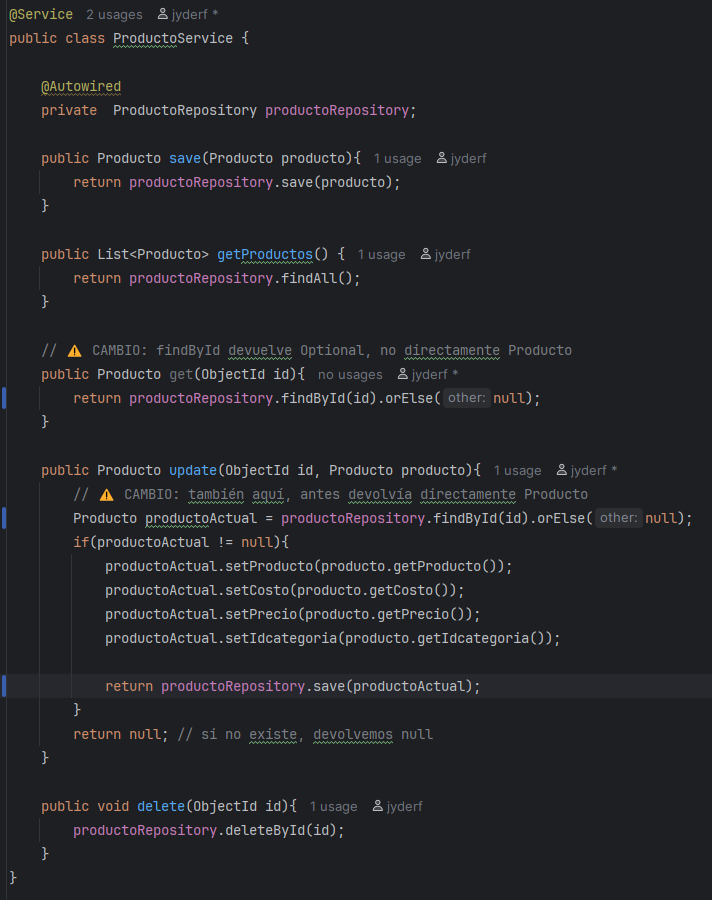


# CREACIÓN DE SERVICIOS

Teniendo en cuenta lo configurado antes, dentro de la carpeta **service,** creamos un archivo llamado **ProductoService,** el cual es de tipo javaClass. Dentro de la ruta se vería más o menos así:



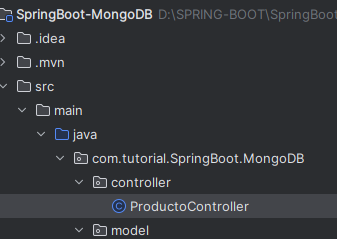
La codificación final del archivo quedaría de la siguiente manera:



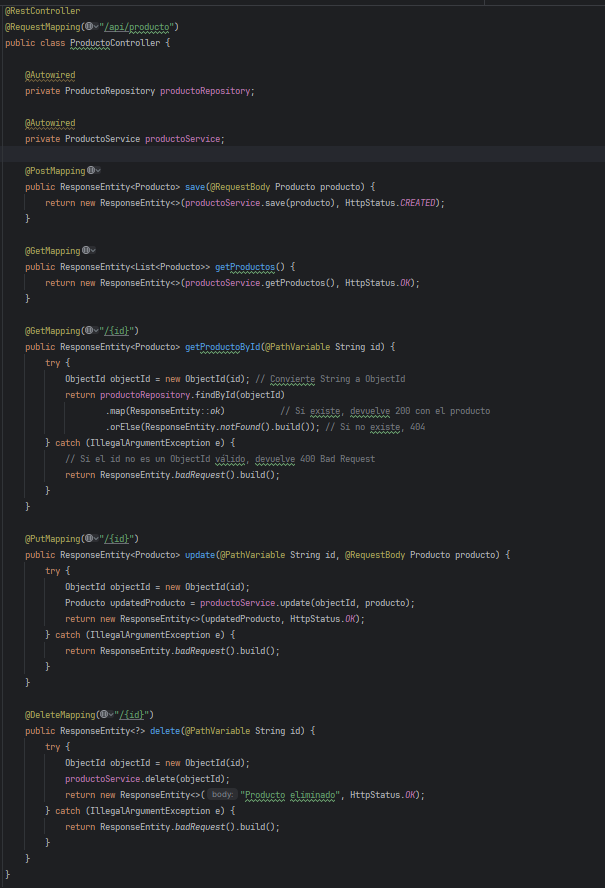
# CREACIÓN DE CONTROLADORES

Dentro de la carpeta controller creamos un archivo de tipo JavaClass, llamado ProductoController

Así:



El archivo finalizado debe quedar así:



Ahora nuestro backend está listo para ser consumido por un frontend y se puede hacer las respectivas pruebas con Postman, Insomnia, nest u otro software para ello.

Aquí los usuarios interactúan con la aplicación a través de URLs:

* GET /productos → lista todos los productos
* GET /productos/{id} → trae un producto por su ID
* POST /productos → guarda un nuevo producto
* DELETE /productos/{id} → elimina un producto

**Flujo de instancias y ejecución**

1. El **usuario** envía una petición HTTP (GET, POST, etc.).
2. El **Controller** recibe la petición y llama al **Service**.
3. El **Service** procesa la lógica y llama al **Repository**.
4. El **Repository** habla con la **Base de Datos**.
5. El resultado sube de nuevo al **Controller**, que lo devuelve al **usuario**.

**¿Qué programar primero?**

1. **El modelo (Entidad Producto)** → define qué quieres guardar.
2. **El repositorio** → define cómo vas a guardar esos datos.
3. **El servicio** → escribe la lógica (guardar, listar, eliminar, etc.).
4. **El controlador** → expone esa lógica al mundo (API REST).
5. Finalmente, **la clase Application** ya está lista para ejecutar todo.

# CÓDIGOS FINALES DE CADA SCRIPT

**ProductoController.java**

package com.tutorial.SpringBoot.MongoDB.controller;

import com.tutorial.SpringBoot.MongoDB.model.Producto;

import com.tutorial.SpringBoot.MongoDB.repository.ProductoRepository;

import com.tutorial.SpringBoot.MongoDB.service.ProductoService;

import org.bson.types.ObjectId;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

import java.util.List;

@RestController

@RequestMapping("/api/producto")

public class ProductoController {

    @Autowired

    private ProductoRepository productoRepository;

    @Autowired

    private ProductoService productoService;

    @PostMapping

    public ResponseEntity<Producto> save(@RequestBody Producto producto) {

        return new ResponseEntity<>(productoService.save(producto), HttpStatus.CREATED);

    }

    @GetMapping

    public ResponseEntity<List<Producto>> getProductos() {

        return new ResponseEntity<>(productoService.getProductos(), HttpStatus.OK);

    }

    @GetMapping("/{id}")

    public ResponseEntity<Producto> getProductoById(@PathVariable String id) {

        try {

            ObjectId objectId = new ObjectId(id); // Convierte String a ObjectId

            return productoRepository.findById(objectId)

                    .map(ResponseEntity::ok)// Si existe, devuelve 200 con el producto

                    .orElse(ResponseEntity.notFound().build()); // Si no existe, 404

        } catch (IllegalArgumentException e) {

            // Si el id no es un ObjectId válido, devuelve 400 Bad Request

            return ResponseEntity.badRequest().build();

        }

    }

    @PutMapping("/{id}")

    public ResponseEntity<Producto> update(@PathVariable String id, @RequestBody Producto producto) {

        try {

            ObjectId objectId = new ObjectId(id);

            Producto updatedProducto = productoService.update(objectId, producto);

            return new ResponseEntity<>(updatedProducto, HttpStatus.OK);

        } catch (IllegalArgumentException e) {

            return ResponseEntity.badRequest().build();

        }

    }

    @DeleteMapping("/{id}")

    public ResponseEntity<?> delete(@PathVariable String id) {

        try {

            ObjectId objectId = new ObjectId(id);

            productoService.delete(objectId);

            return new ResponseEntity<>("Producto eliminado", HttpStatus.OK);

        } catch (IllegalArgumentException e) {

            return ResponseEntity.badRequest().build();

        }

    }

}

package com.tutorial.SpringBoot.MongoDB.model;

**Producto.java**

import jakarta.persistence.Id;

import org.springframework.data.mongodb.core.mapping.Document;// ✅ Importación correcta de @Document de Spring Data MongoDB

@Document(collection = "productos")

public class Producto {

    @Id

    private String id;

    private String producto;

    private int costo;

    private int precio;

    private String idcategoria;

    public String getId() {

        return id;

    }

    public void setId(String id) {

        this.id = id;

    }

    public String getProducto() {

        return producto;

    }

    public void setProducto(String producto) {

        this.producto = producto;

    }

    public int getCosto() {

        return costo;

    }

    public void setCosto(int costo) {

        this.costo = costo;

    }

    public int getPrecio() {

        return precio;

    }

    public void setPrecio(int precio) {

        this.precio = precio;

    }

    public String getIdcategoria() {

        return idcategoria;

    }

    public void setIdcategoria(String idcategoria) {

        this.idcategoria = idcategoria;

    }

}

**Producto.Repository.java**

package com.tutorial.SpringBoot.MongoDB.repository;

// Importamos la clase Producto que definimos en el modelo

import com.tutorial.SpringBoot.MongoDB.model.Producto;

import org.bson.types.ObjectId;

// Importamos la interfaz MongoRepository de Spring Data

import org.springframework.data.mongodb.repository.MongoRepository;

// Importamos la anotación @Repository para indicar que esta interfaz es un repositorio

import org.springframework.stereotype.Repository;

@Repository

public interface ProductoRepository extends MongoRepository<Producto, ObjectId> {

}

**ProductoService.java**

package com.tutorial.SpringBoot.MongoDB.service;

import com.tutorial.SpringBoot.MongoDB.model.Producto;

import com.tutorial.SpringBoot.MongoDB.repository.ProductoRepository;

import org.bson.types.ObjectId;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class ProductoService {

    @Autowired

    private  ProductoRepository productoRepository;

    public Producto save(Producto producto){

        return productoRepository.save(producto);

    }

    public List<Producto> getProductos() {

        return productoRepository.findAll();

    }

    public Producto get(ObjectId id){

        return productoRepository.findById(id).orElse(null);

    }

    public Producto update(ObjectId id, Producto producto){

        Producto productoActual = productoRepository.findById(id).orElse(null);

        if(productoActual != null){

            productoActual.setProducto(producto.getProducto());

            productoActual.setCosto(producto.getCosto());

            productoActual.setPrecio(producto.getPrecio());

            productoActual.setIdcategoria(producto.getIdcategoria());

            return productoRepository.save(productoActual);

        }

        return null;

    }

    public void delete(ObjectId id){

        productoRepository.deleteById(id);

    }

}