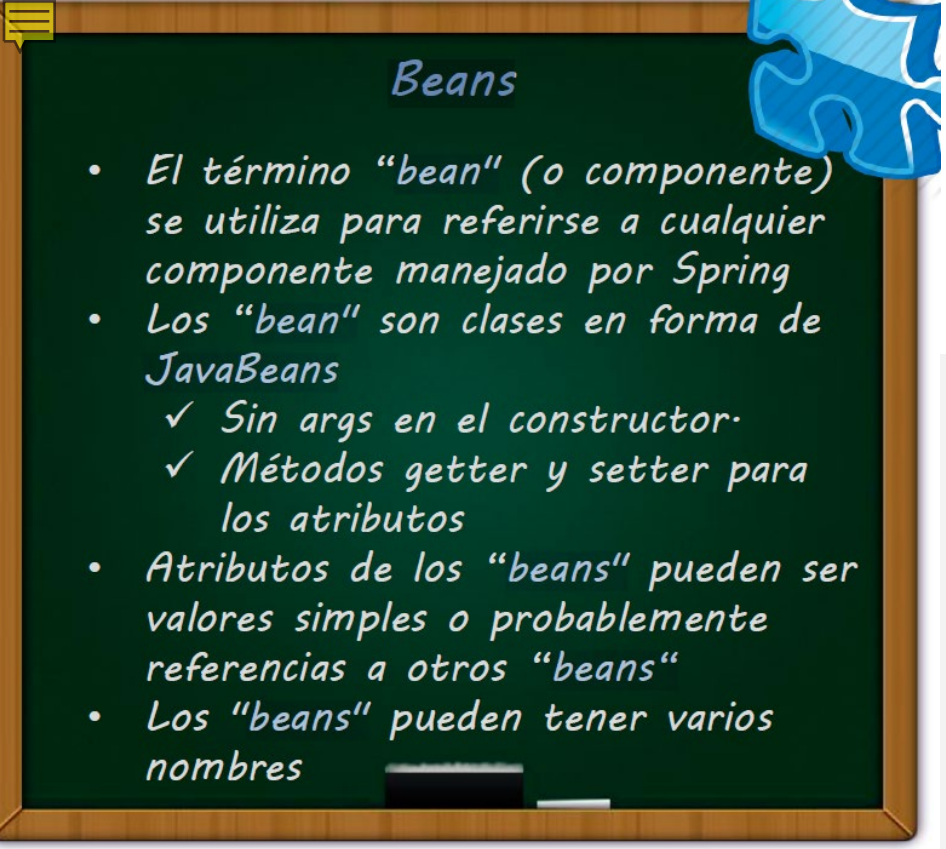
- Uso de Persistencia de datos.
- Uso de Interfaz CrudRepository
- Creación de estructura de respuesta de Api



Ejemplo

Con un ejemplo se ve bien sencillo, supongamos que queremos un programa que va a almacenar información sobre personas. Lo primero por lo que podemos empezar es creando una clase Persona, por ejemplo:

```java
public class Persona implements Serializable {
    private int id;
    private String nombre;
    private String apellido1;
    private String apellido2;
    private String direccion;
    private String email;
    private String telefono;

    public Persona() {

    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
```

....

Características de los Beans

Se trata de una clase **seriarizable** que contiene unas variables que van a almacenar la información que necesitamos. Éstas **variables deben ser privadas**. Y para acceder a éstas variables deben implementarse unos metodos **get** (para obtener el valor) y **set** (para establecer el valor). Éstos métodos deben ser **públicos**, y debe tener también un **constructor público**.

Por ejemplo el método *getTelefono*() devuelve el valor del teléfono almacenado en la clase. Y con el método *setTelefono*(número) guardamos el valor.

SERIALIZACIÓN

En Java, la palabra “serialización” se refiere al proceso de convertir un objeto en una secuencia de bytes que puede ser almacenada en un archivo o transmitida a través de una red. Un objeto que se puede serializar se dice que es “serializable”.

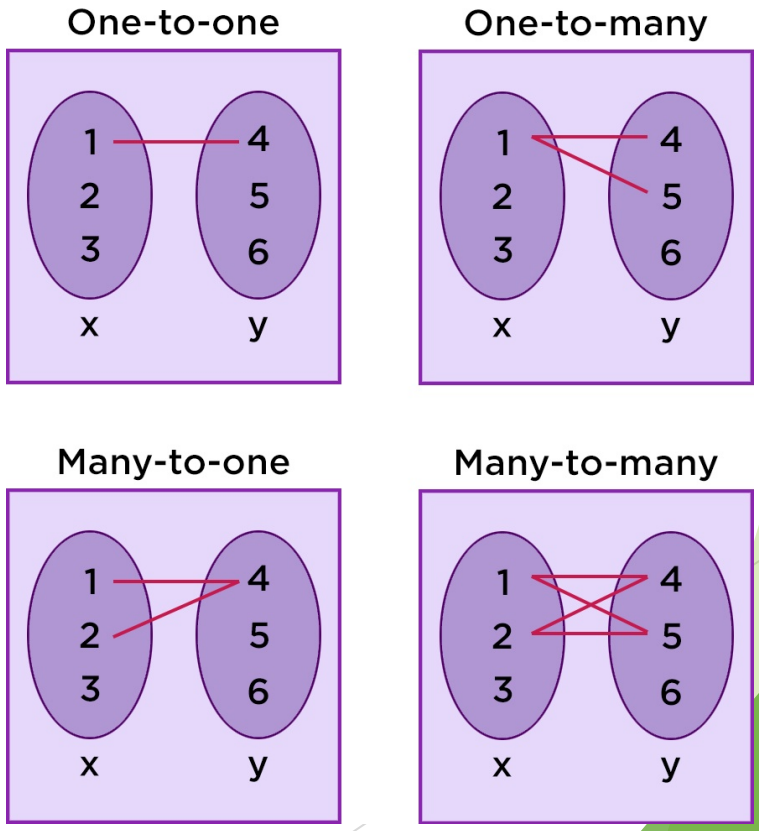
Un campo, clase o método que es “serializable” significa que puede ser convertido en una secuencia de bytes y luego reconstruido como un objeto nuevo. Esto es útil cuando se necesita guardar el estado de un objeto en disco o transmitirlo a través de una red.

https://www.blog.facilcloud.com/noticias/jpa-vs-hibernate/
https://www.baeldung.com/learn-jpa-hibernate

JPA es una especificación creada para acceder y hacer el mapeo entre los objetos y las bases de datos. Por otro lado, Hibernate es un framework que permite implementar esta especificación
----podemos cambiar de forma transparente … mysql, postgres

Instructor.java | Course.java

```
2
3  import javax.persistence.Entity;
4  import javax.persistence.GeneratedValue;
5  import javax.persistence.Id;
6  import javax.persistence.ManyToOne;
7
8  import com.fasterxml.jackson.annotation.JsonIgnore;
9
10 @Entity
11 public class Course {
12
13     @JsonIgnore
14     @ManyToOne
15     private Instructor instructor;
16
17     @Id
18     @GeneratedValue
19     private Long id;
20
21     Course() { // jpa only
22     }
```

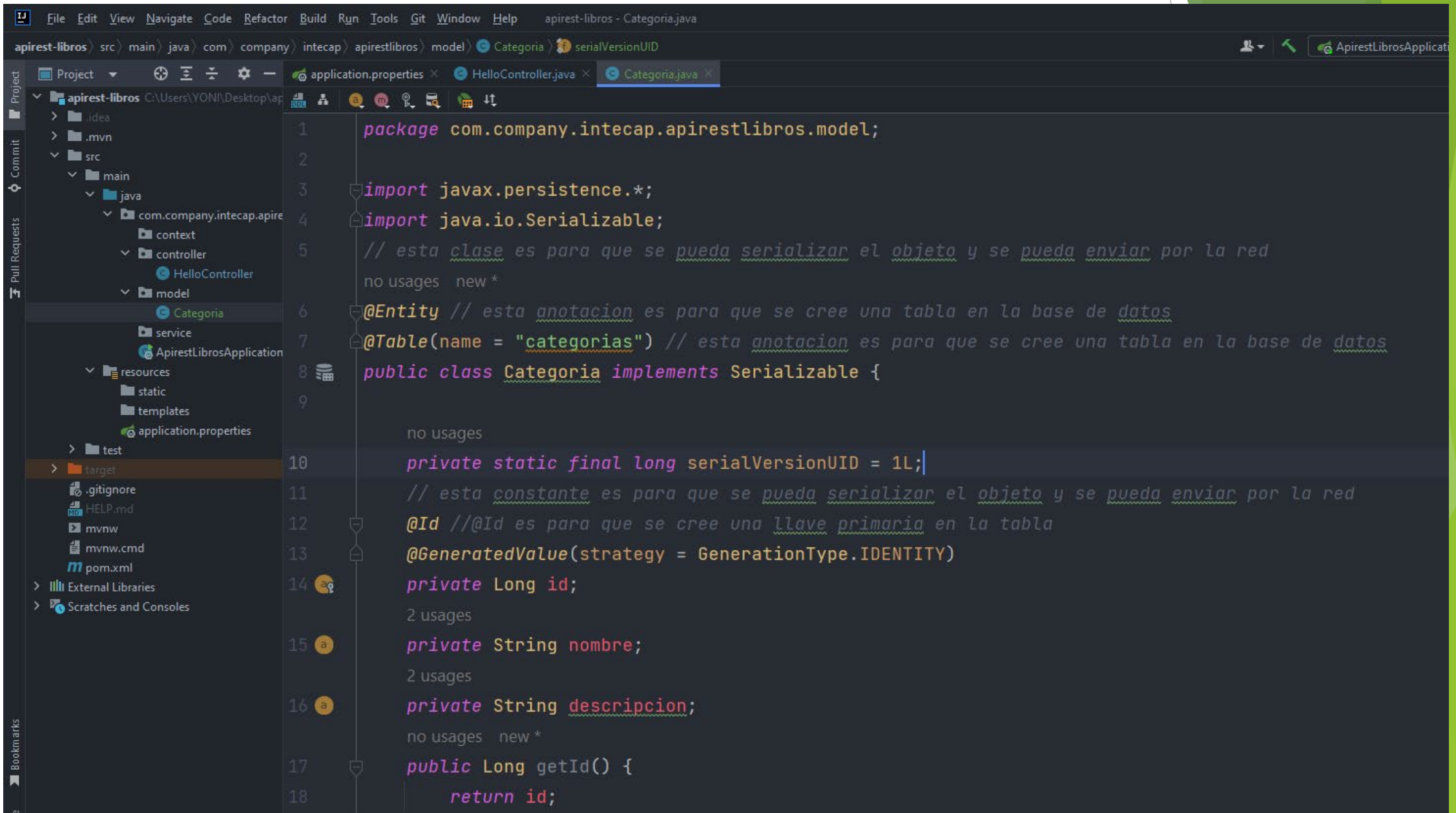


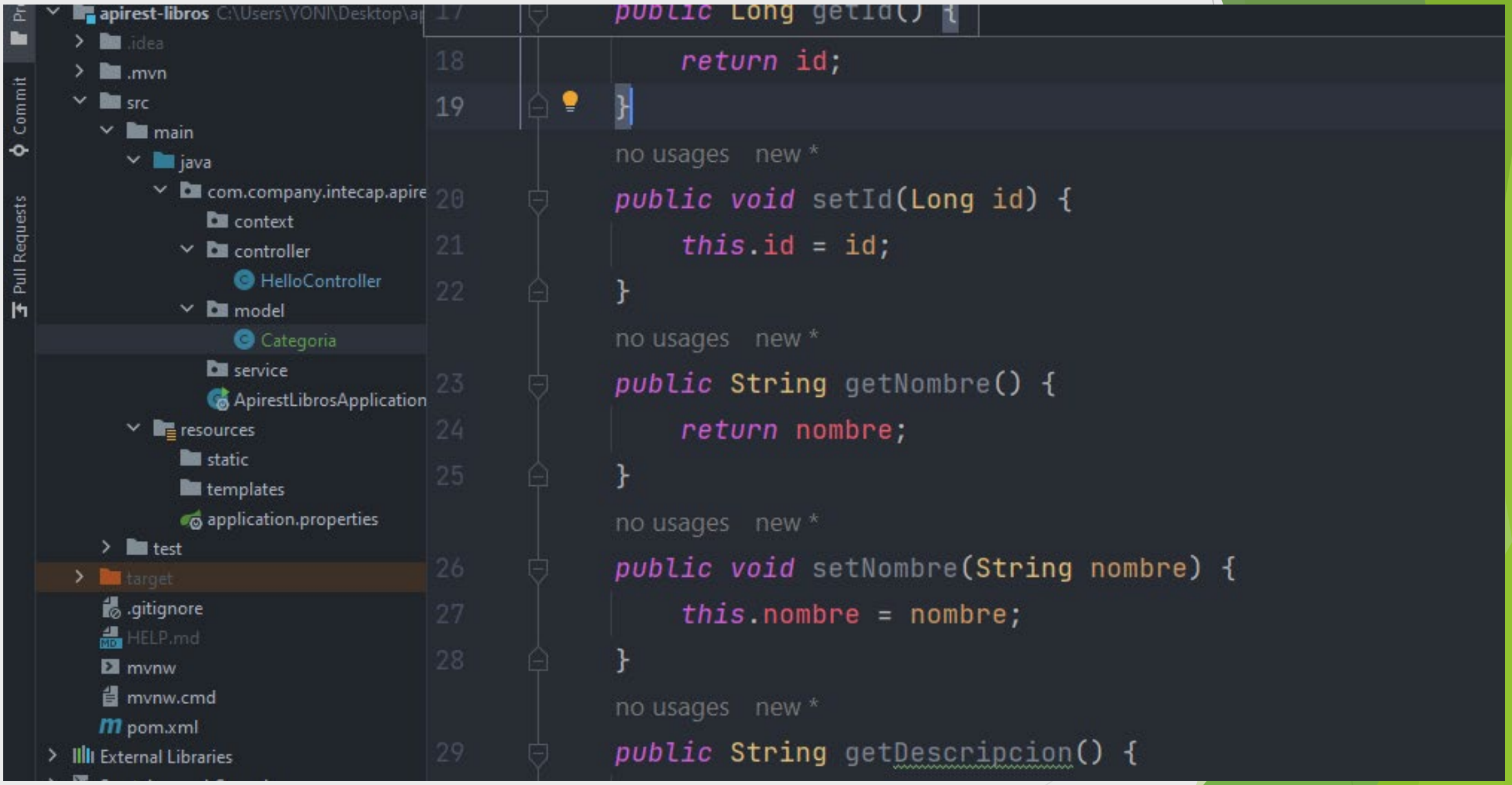
CREACIÓN DE PAQUETES.

→ com/company/intecap/apirestlibros.model

El paquete model, guardara nuestra clase modelo, clases pojo de nuestra aplicación.

```
@Table(name = "categoria") // esta anotacion es para que se cree una tabla en la base de datos
```




apirest-libros C:\Users\YONI\Desktop\a
.idea
.mvn
src
main
java
com.company.intecap.apire
context
controller
HelloController
model
Categoria
service
ApirestLibrosApplication
resources
static
templates
application.properties
test
target
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
External Libraries
Commit
Pull Requests
public Long getId() {
return id;
}
no usages new *
public void setId(Long id) {
this.id = id;
}
no usages new *
public String getNombre() {
return nombre;
}
no usages new *
public void setNombre(String nombre) {
this.nombre = nombre;
}
no usages new *
public String getDescripcion() {

¿Qué es una interfaz DAO?

En software, un objeto de acceso a datos (en inglés: data access object, abreviado DAO) es un componente de software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos, tales como una Base de datos o un archivo.

DAO es un estándar que tiene como finalidad centralizar la responsabilidad de acceso a la capa de datos, el modelo. DAO viene de Data Access Object y esta capa es la que realiza la conexión con la fuente de información. Sea una base de datos SQL, NoSQL, cache, archivo...

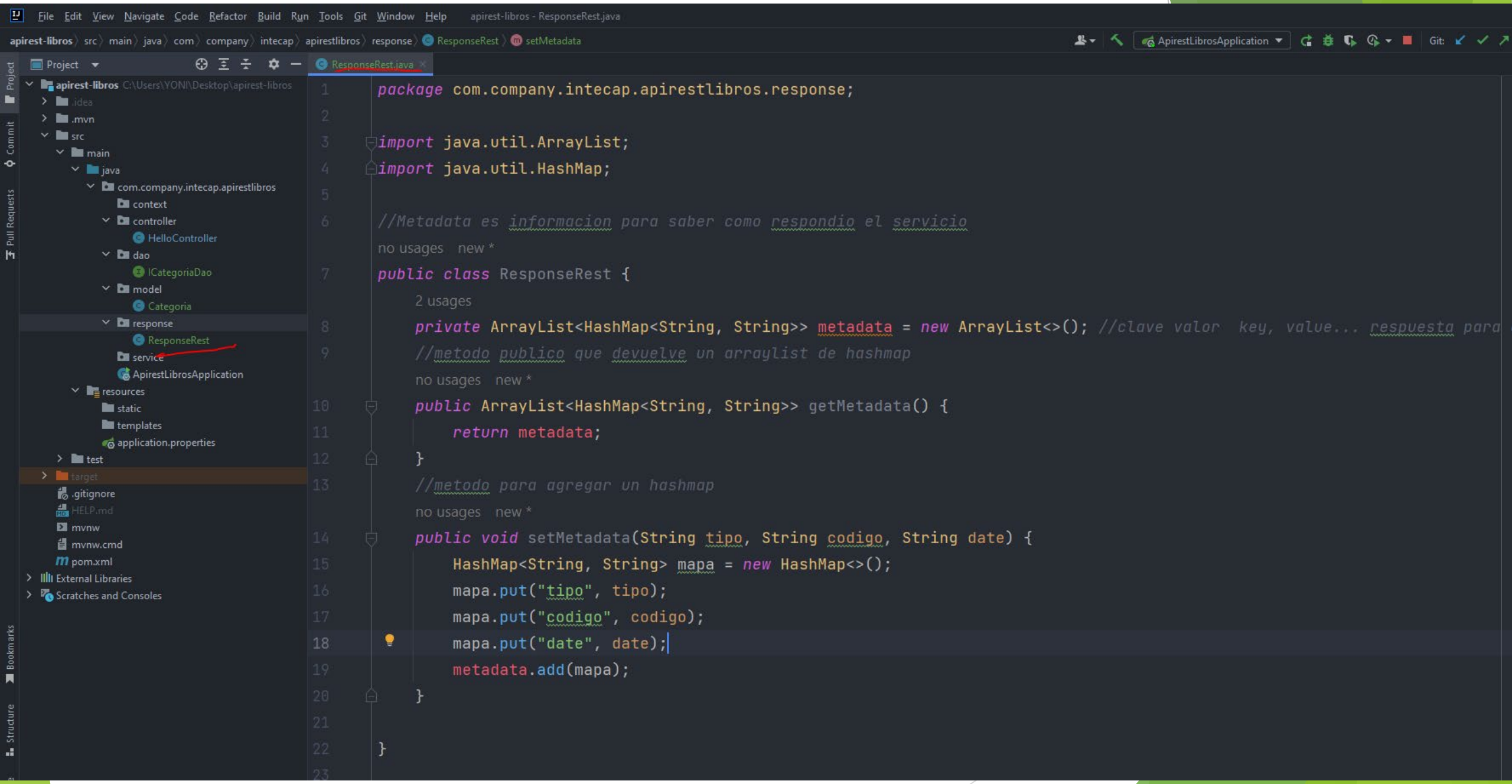
Se suele centralizar, en estas clases DAO, la responsabilidad sobre una sola entidad. Entonces si en tu aplicación tienes una entidad de Usuario, una de Producto y una para definir la Compra. Vamos a tener tres clases DAO, UsuarioDAO, ProductoDAO y CompraDAO. Cada una con su responsabilidad y lógica para acceder los datos sobre una entidad.

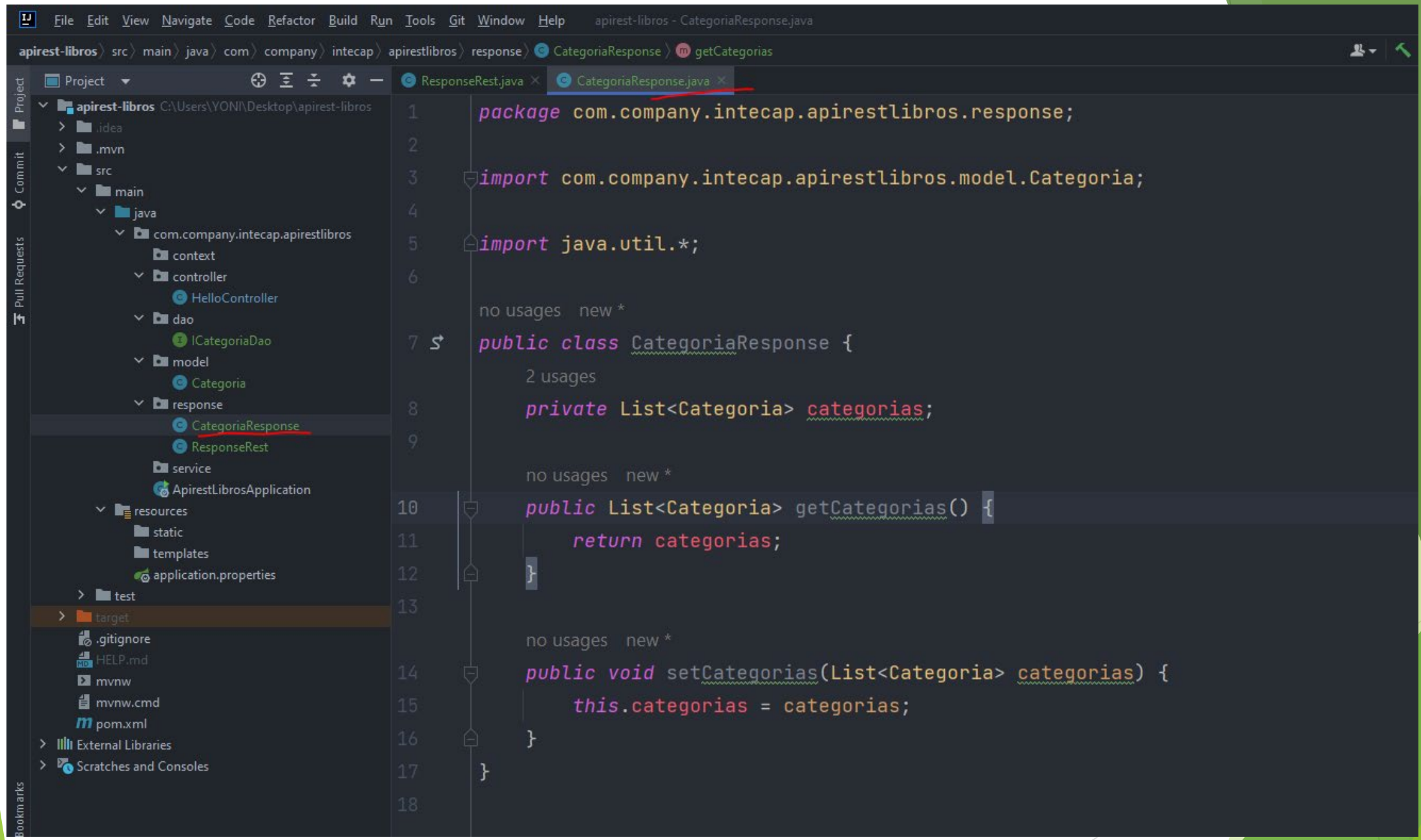
¿Qué es CrudRepository?

CrudRepository es una interfaz Spring Data para operaciones CRUD genéricas en un repositorio de un tipo específico . Proporciona varios métodos listos para usar para interactuar con una base de datos

https://stackoverflow.com/questions/14014086/what-is-difference-between-crudrepository-and-jparepository-interfaces-in-spring

Una interfaz en Java es una colección de métodos abstractos y propiedades constantes. En las interfaces se especifica qué se debe hacer pero no su implementación. Serán las clases que implementen estas interfaces las que describan la lógica del comportamiento de los métodos.

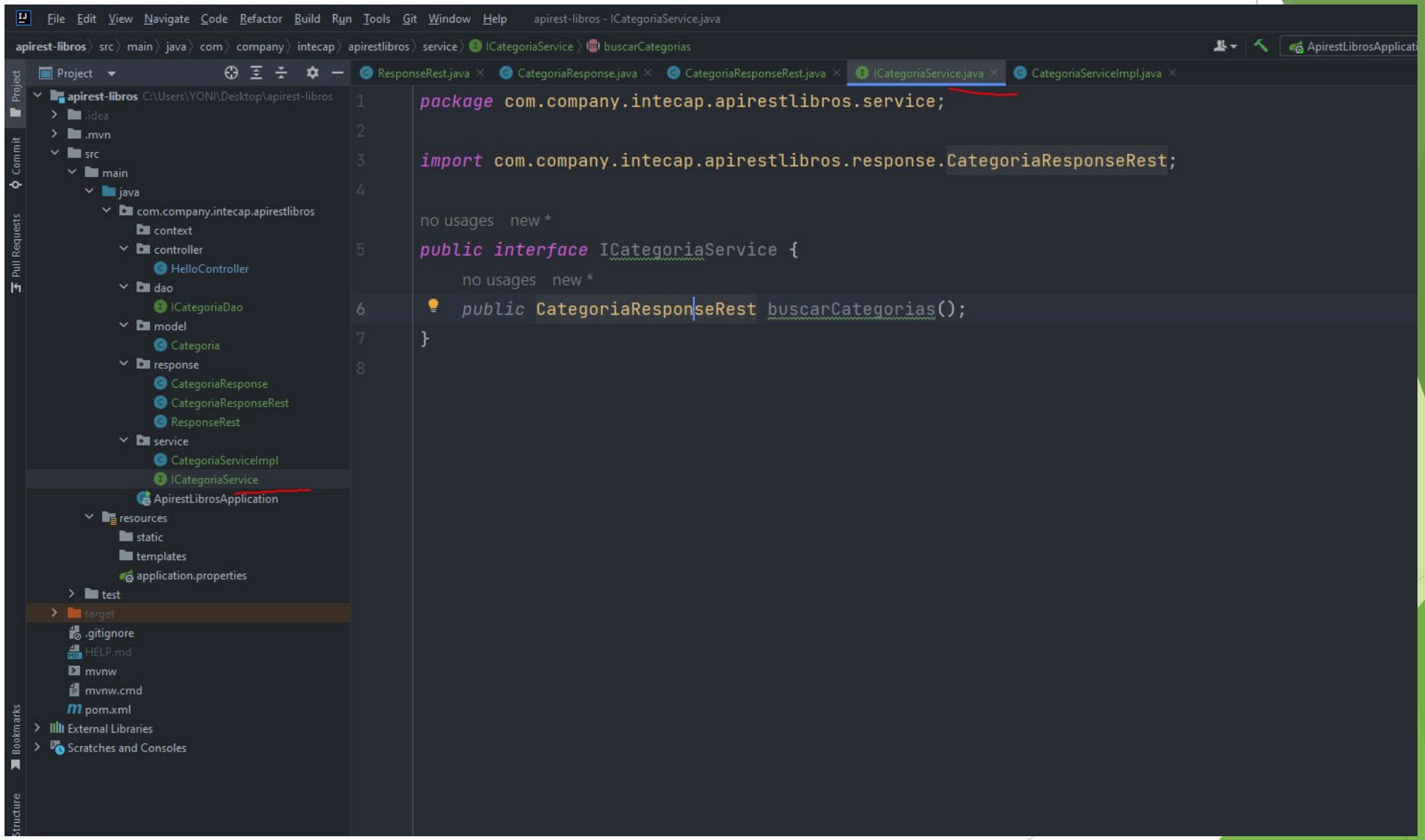

File Edit View Navigate Code Refactor Build Run Tools Git Window Help
apirest-libros - ResponseRest.java
apirest-libros src main java com company intecap apirestlibros response ResponseRest setMetadata
ApirestLibrosApplication
Git:
Project
ResponseRest.java
apirest-libros C:\Users\YONI\Desktop\apirest-libros
.idea
.mvn
src
main
java
com.company.intecap.apirestlibros
context
controller
HelloController
dao
ICategoriaDao
model
Categoria
response
ResponseRest
service
ApirestLibrosApplication
resources
static
templates
application.properties
test
target
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
External Libraries
Scratches and Consoles
package com.company.intecap.apirestlibros.response;
import java.util.ArrayList;
import java.util.HashMap;
//Metadata es informacion para saber como respondio el servicio
no usages new *
public class ResponseRest {
2 usages
private ArrayList<HashMap<String, String>> metadata = new ArrayList<>(); //clave valor key, value... respuesta para
//metodo publico que devuelve un arraylist de hashmap
no usages new *
public ArrayList<HashMap<String, String>> getMetadata() {
return metadata;
}
//metodo para agregar un hashmap
no usages new *
public void setMetadata(String tipo, String codigo, String date) {
HashMap<String, String> mapa = new HashMap<>();
mapa.put("tipo", tipo);
mapa.put("codigo", codigo);
mapa.put("date", date);
metadata.add(mapa);
}
}

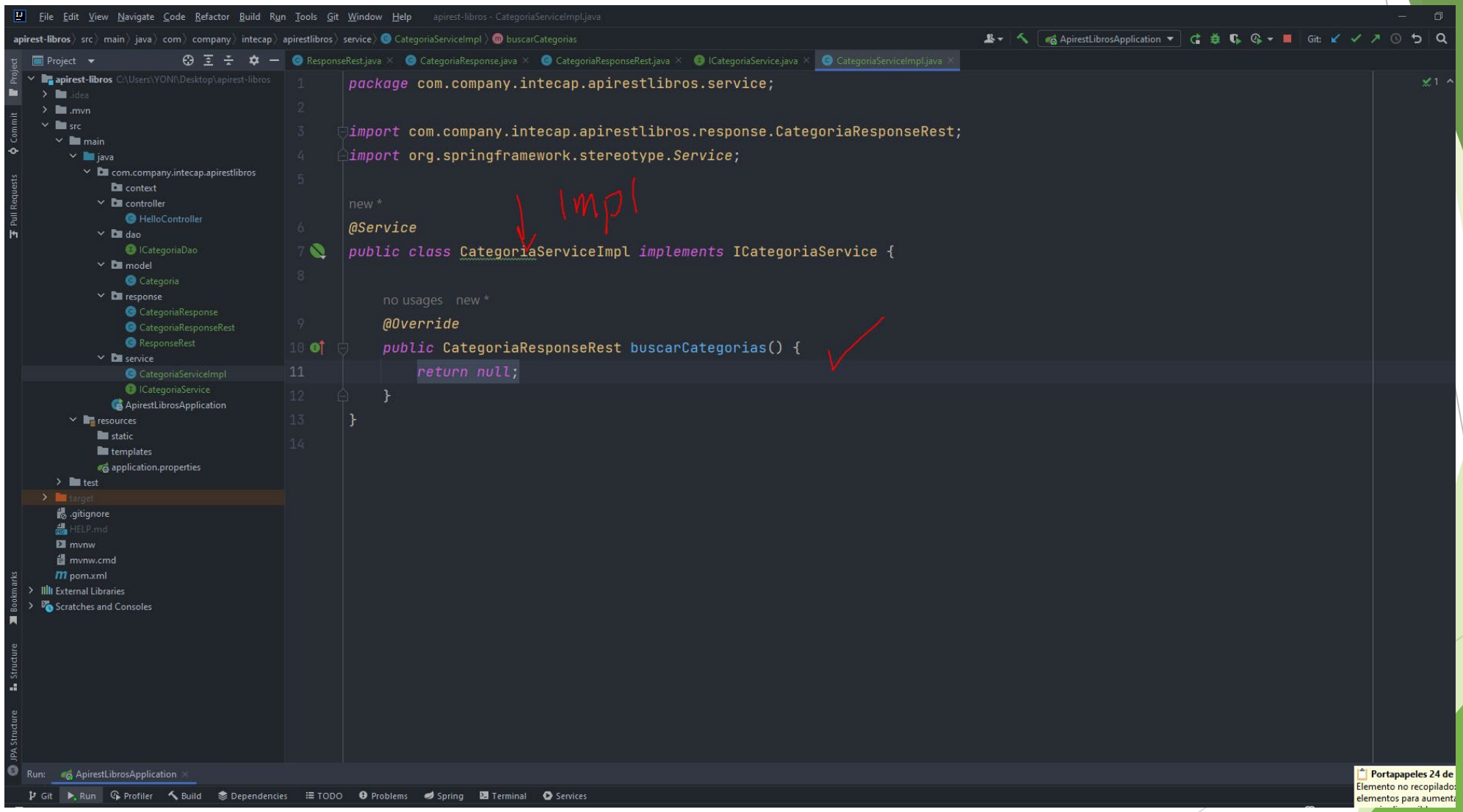


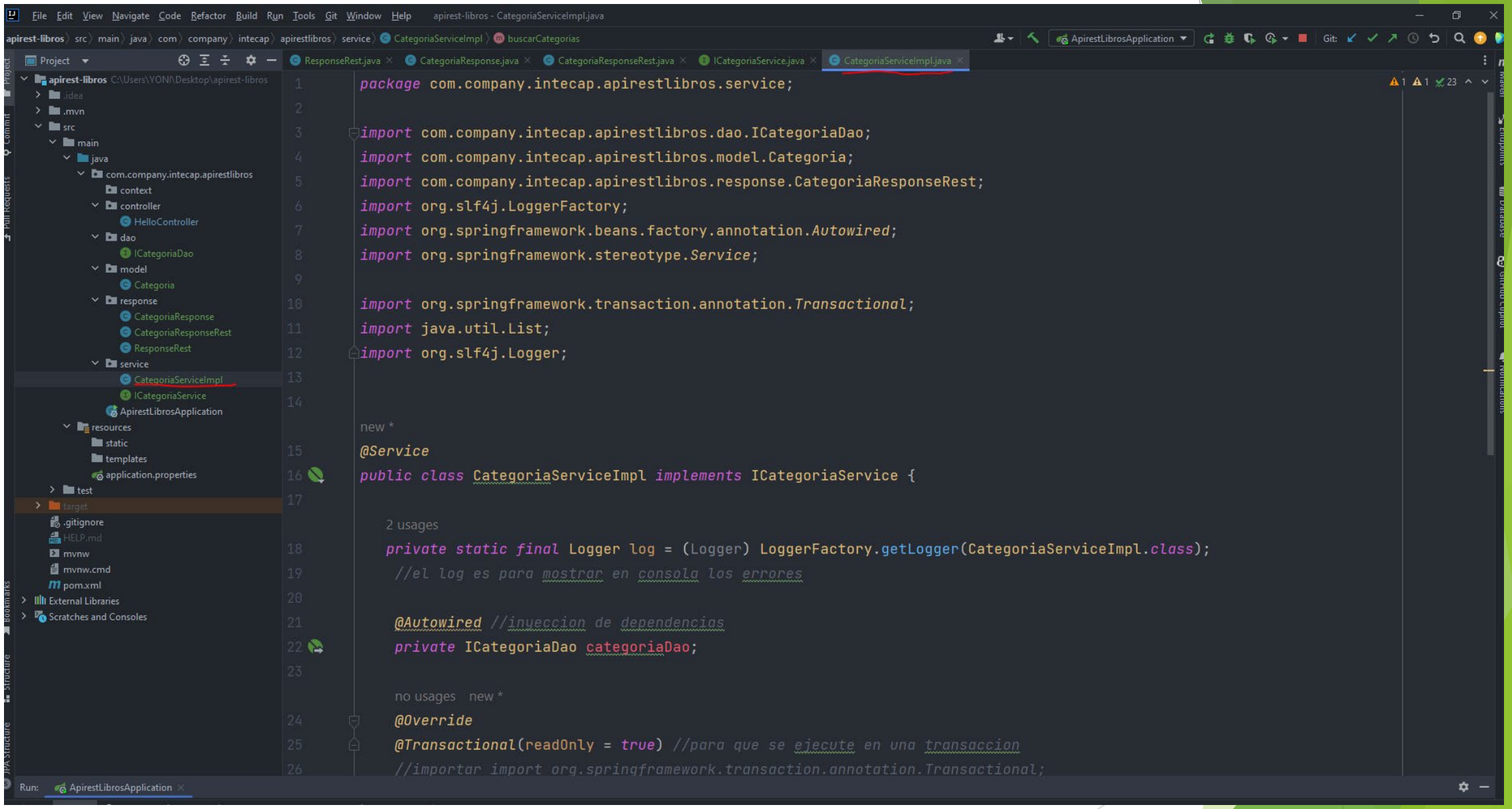
Este método se utiliza para las repuesta, para los métodos del crud

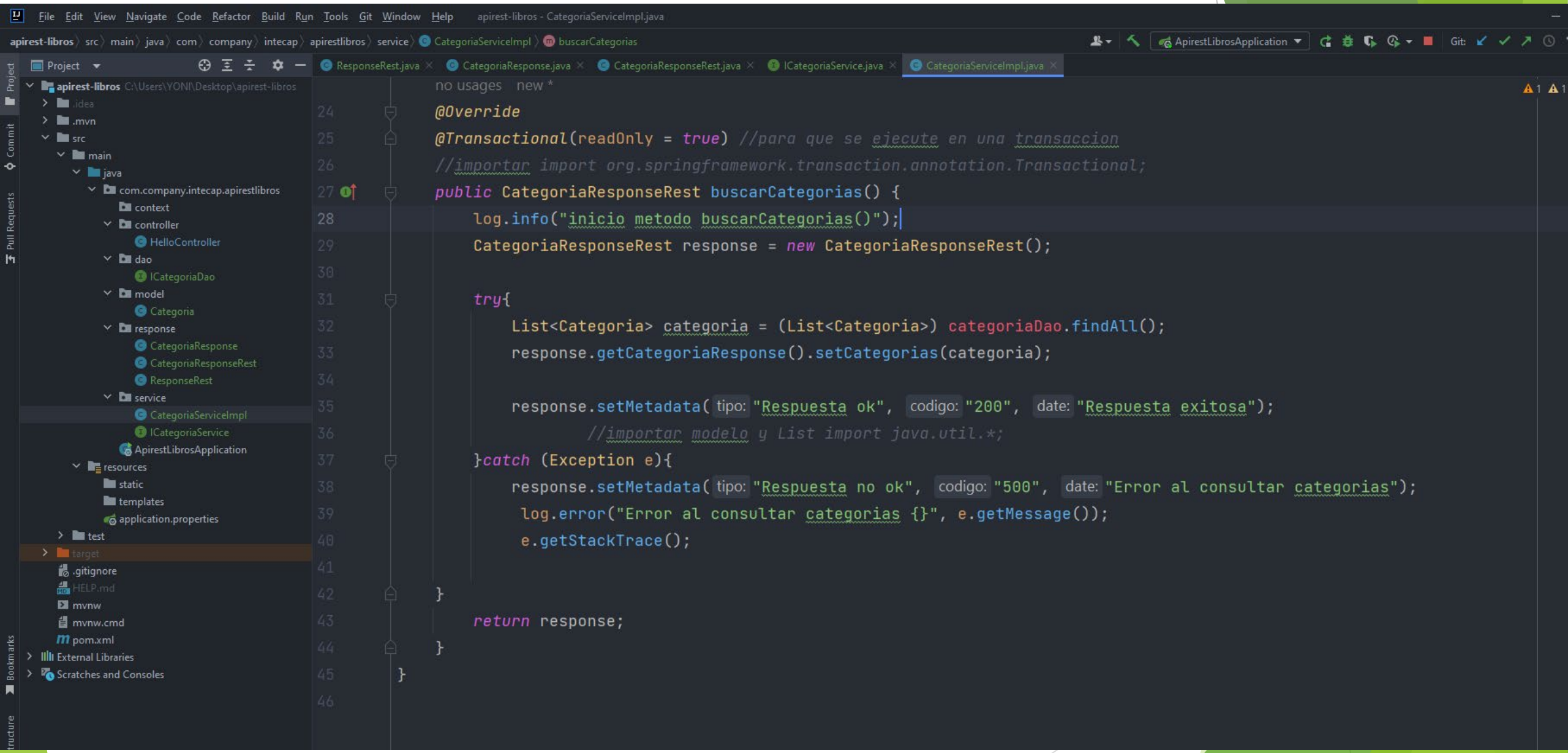
- Creación primera API Rest del proyecto.
- Uso de etiquetas @Autowired, @Service, @RestController.
- Uso de log en nuestro proyecto.
- Uso de método GET.
- Uso de Postman para probar API Rest.
- Manejo de errores con HttpStatus.
-

Interfaz para acceder a los Datos.. Services..




File Edit View Navigate Code Refactor Build Run Tools Git Window Help
apirest-libros - CategoriaServiceImpl.java
apirest-libros src main java com company intecap apirestlibros service CategoriaServiceImpl buscarCategorias
ApirestLibrosApplication
Git:
Project
apirest-libros C:\Users\YONI\Desktop\apirest-libros
.idea
.mvn
src
main
java
com.company.intecap.apirestlibros
context
controller
HelloController
dao
ICategoriaDao
model
Categoria
response
CategoriaResponse
CategoriaResponseRest
ResponseRest
service
CategoriaServiceImpl
ICategoriaService
ApirestLibrosApplication
resources
static
templates
application.properties
test
target
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
External Libraries
Scratches and Consoles
ResponseRest.java
CategoriaResponse.java
CategoriaResponseRest.java
ICategoriaService.java
CategoriaServiceImpl.java
package com.company.intecap.apirestlibros.service;
import com.company.intecap.apirestlibros.response.CategoriaResponseRest;
import org.springframework.stereotype.Service;
new *
@Service
public class CategoriaServiceImpl implements ICategoriaService {
no usages new *
@Override
public CategoriaResponseRest buscarCategorias() {
return null;
}
}
Impl
Run: ApirestLibrosApplication
Git Run Profiler Build Dependencies TODO Problems Spring Terminal Services
Portapapeles 24 de
Elemento no recopilado
elementos para aumenta


File Edit View Navigate Code Refactor Build Run Tools Git Window Help
apirest-libros - CategoriaServiceImpl.java
apirest-libros src main java com company intecap apirestlibros service CategoriaServiceImpl buscarCategorias
ApirestLibrosApplication
Git:
Project
apirest-libros C:\Users\YONI\Desktop\apirest-libros
.idea
.mvn
src
main
java
com.company.intecap.apirestlibros
context
controller
HelloController
dao
ICategoriaDao
model
Categoria
response
CategoriaResponse
CategoriaResponseRest
ResponseRest
service
CategoriaServiceImpl
ICategoriaService
ApirestLibrosApplication
resources
static
templates
application.properties
test
target
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
External Libraries
Scratches and Consoles
ResponseRest.java
CategoriaResponse.java
CategoriaResponseRest.java
ICategoriaService.java
CategoriaServiceImpl.java
1 1 23
package com.company.intecap.apirestlibros.service;
import com.company.intecap.apirestlibros.dao.ICategoriaDao;
import com.company.intecap.apirestlibros.model.Categoria;
import com.company.intecap.apirestlibros.response.CategoriaResponseRest;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import java.util.List;
import org.slf4j.Logger;
new *
@Service
public class CategoriaServiceImpl implements ICategoriaService {
2 usages
private static final Logger log = (Logger) LoggerFactory.getLogger(CategoriaServiceImpl.class);
//el log es para mostrar en consola los errores
@Autowired //inyeccion de dependencias
private ICategoriaDao categoriaDao;
no usages new *
@Override
@Transactional(readOnly = true) //para que se ejecute en una transaccion
//importar import org.springframework.transaction.annotation.Transactional;
Run: ApirestLibrosApplication


File Edit View Navigate Code Refactor Build Run Tools Git Window Help
apirest-libros - CategoriaServiceImpl.java
apirest-libros src main java com company intecap apirestlibros service CategoriaServiceImpl buscarCategorias
ApirestLibrosApplication
Git:
Project
ResponseRest.java
CategoriaResponse.java
CategoriaResponseRest.java
ICategoriaService.java
CategoriaServiceImpl.java
apirest-libros C:\Users\YONI\Desktop\apirest-libros
.idea
.mvn
src
main
java
com.company.intecap.apirestlibros
context
controller
HelloController
dao
ICategoriaDao
model
Categoria
response
CategoriaResponse
CategoriaResponseRest
ResponseRest
service
CategoriaServiceImpl
ICategoriaService
ApirestLibrosApplication
resources
static
templates
application.properties
test
target
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
External Libraries
Scratches and Consoles
no usages new *
@Override
@Transactional(readOnly = true) //para que se ejecute en una transaccion
//importar import org.springframework.transaction.annotation.Transactional;
public CategoriaResponseRest buscarCategorias() {
log.info("inicio metodo buscarCategorias()");
CategoriaResponseRest response = new CategoriaResponseRest();
try{
List<Categoria> categoria = (List<Categoria>) categoriaDao.findAll();
response.getCategoriaResponse().setCategorias(categoria);
response.setMetadata(tipo: "Respuesta ok", codigo: "200", date: "Respuesta exitosa");
//importar modelo y List import java.util.*;
}catch (Exception e){
response.setMetadata(tipo: "Respuesta no ok", codigo: "500", date: "Error al consultar categorias");
log.error("Error al consultar categorias {}", e.getMessage());
e.getStackTrace();
}
return response;
}
}