

Ejercicio Práctico 3.

1. Crear una Api resFull

nombre: api-productos

2. Pasos a Seguir

2.1 model (agregar los modelo ejemplo Categoria.java *clase)

2.2 dao --> CrudRepository (extends CrudRepository *Interface)

2.3 response ----> ResponseRest .java(clase)

(solo se realizar una vez, es el HashMap array clave valor de metada de la respuesta de la api)

2.3.1 response --> CategoriaResponse (devuelve list<Categoria> crear getter y setter *clase)

2.3.2 response --> CategoriaResponseRest(extends ResponseRest *porque contiene la estructura de metada
plantilla y crear una variable private CategoriaResponse e instanciamos el objeto

(private CategoriaResponse categoriaResponse = new CategoriaResponse())

*porque devuelve una lista de categorias se crean los getter y setter)

2.4 service

2.4.1 service--> ICategoriaService (la capa service sirve para acceder a los datos *Interface)

se crean los metodos para el crud

a- public CategoriaResponseRest buscarCategorias();

2.4.2 service--> CategoriaServiceImpl ("clase de implementación" contiene @Service, implements ICategoriaService
y agregar el metodo a implementar.)

*** se agrega la logica de negocio.. por cada metodo del crud.

@Autowired ---> inyección de dependencia

2.5 cotroller--> CategoriaRestController: (controlador de punto de entrada para manejo de nuestra api-rest)

notación @RestController: se inyecta como un bean

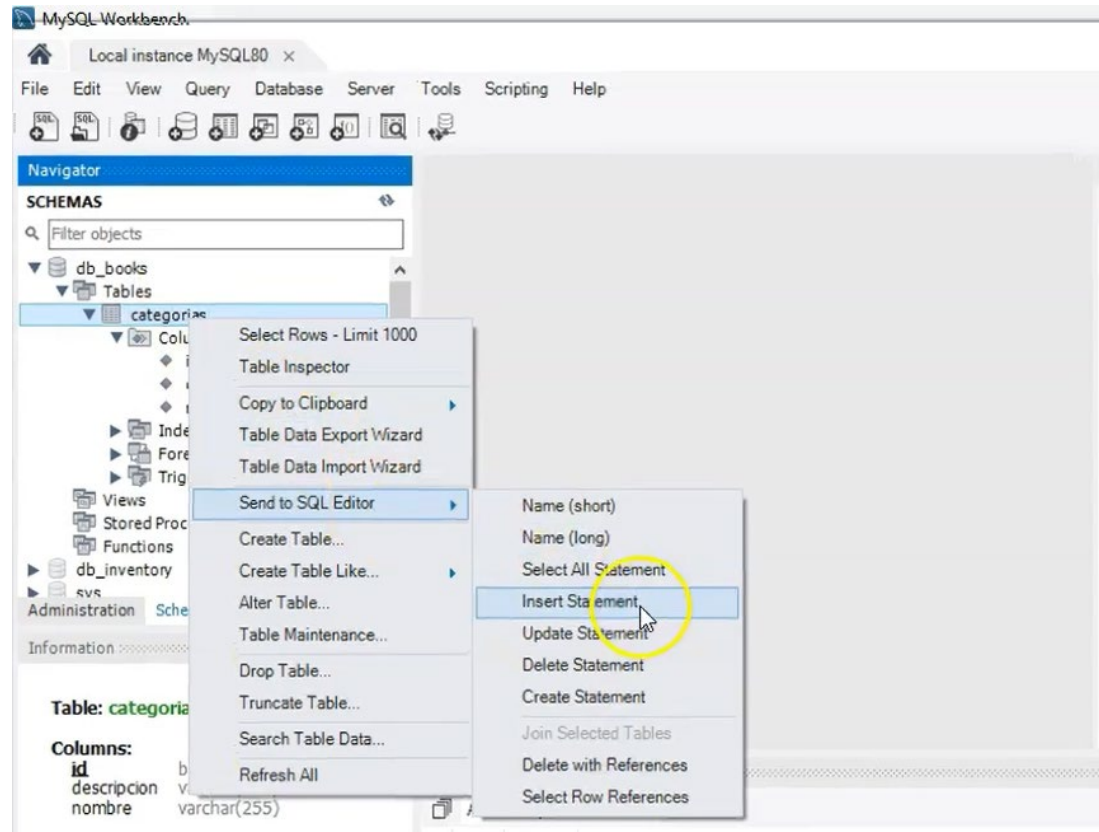
notación @RequestMapping: indica el el mapeo/rutas de la api. se agrega el prefijo. ("/v1")

@Autowired inyeccion

a). crear los metodos del crud. @GetMapping("/categorias") , @PostMapping , @PutMapping,@DeleteMapping
manejo de las rutas. GET, PUT, POST, DELETE

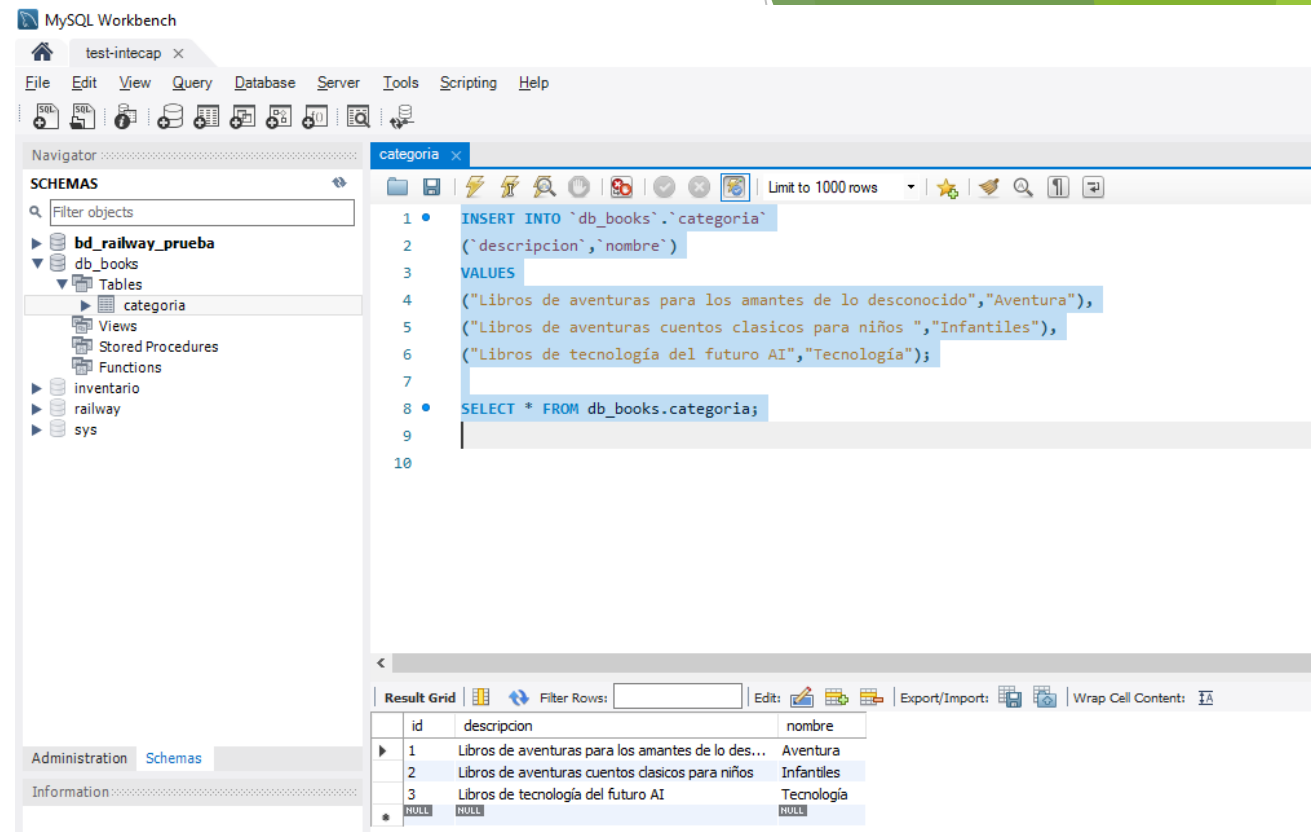
3. probar su api-rest-full en postman.

4. ENVIAR EN PDF SU INFORME DE SU API-PRODUCTOS (PROCEDIMIENTO COMPLETO PASO A PASO DE LA CONSTRUCCIÓN DEL MISMO..



```
INSERT INTO `db_books`.`categoria`
(`descripcion`, `nombre`)
VALUES
('Libros de aventuras para los amantes de lo desconocido', 'Aventura'),
('Libros de aventuras cuentos clasicos para niños', 'Infantiles'),
('Libros de tecnología del futuro AI', 'Tecnología');

SELECT * FROM `db_books`.`categoria`;
```



Uso de clase HttpStatus

<https://developer.mozilla.org/es/docs/Web/HTTP/Status>

En el contexto del Spring Framework en Java, la clase HttpStatus se utiliza para representar códigos de estado HTTP y sus descripciones asociadas. Esto es útil cuando estás construyendo aplicaciones web utilizando Spring y necesitas enviar respuestas HTTP con códigos de estado apropiados.

La clase HttpStatus proporciona una serie de constantes estáticas que representan los códigos de estado HTTP más comunes, junto con sus descripciones. Algunos ejemplos de códigos de estado incluidos en HttpStatus son:

- OK: Representa el código de estado 200 OK, que indica que la solicitud fue exitosa.
- NOT_FOUND: Representa el código de estado 404 Not Found, que indica que el recurso solicitado no se encontró en el servidor.
- BAD_REQUEST: Representa el código de estado 400 Bad Request, que indica que la solicitud no se pudo entender o procesar por el servidor debido a un formato incorrecto.
- INTERNAL_SERVER_ERROR: Representa el código de estado 500 Internal Server Error, que indica un error interno en el servidor.

```
apuntes.txt • dato.java •
dato.java
1  import org.springframework.http.HttpStatus;
2  import org.springframework.http.ResponseEntity;
3  import org.springframework.web.bind.annotation.GetMapping;
4  import org.springframework.web.bind.annotation.RequestMapping;
5  import org.springframework.web.bind.annotation.RestController;
6
7  @RestController
8  @RequestMapping("/api")
9  public class MyController {
10
11      @GetMapping("/success")
12      public ResponseEntity<String> success() {
13          return new ResponseEntity<>("Operación exitosa", HttpStatus.OK);
14      }
15
16      @GetMapping("/not-found")
17      public ResponseEntity<String> notFound() {
18          return new ResponseEntity<>("Recurso no encontrado", HttpStatus.NOT_FOUND);
19      }
20
21      @GetMapping("/error")
22      public ResponseEntity<String> error() {
23          return new ResponseEntity<>("Error interno del servidor", HttpStatus.INTERNAL_SERVER_ERROR);
24      }
25  }
```

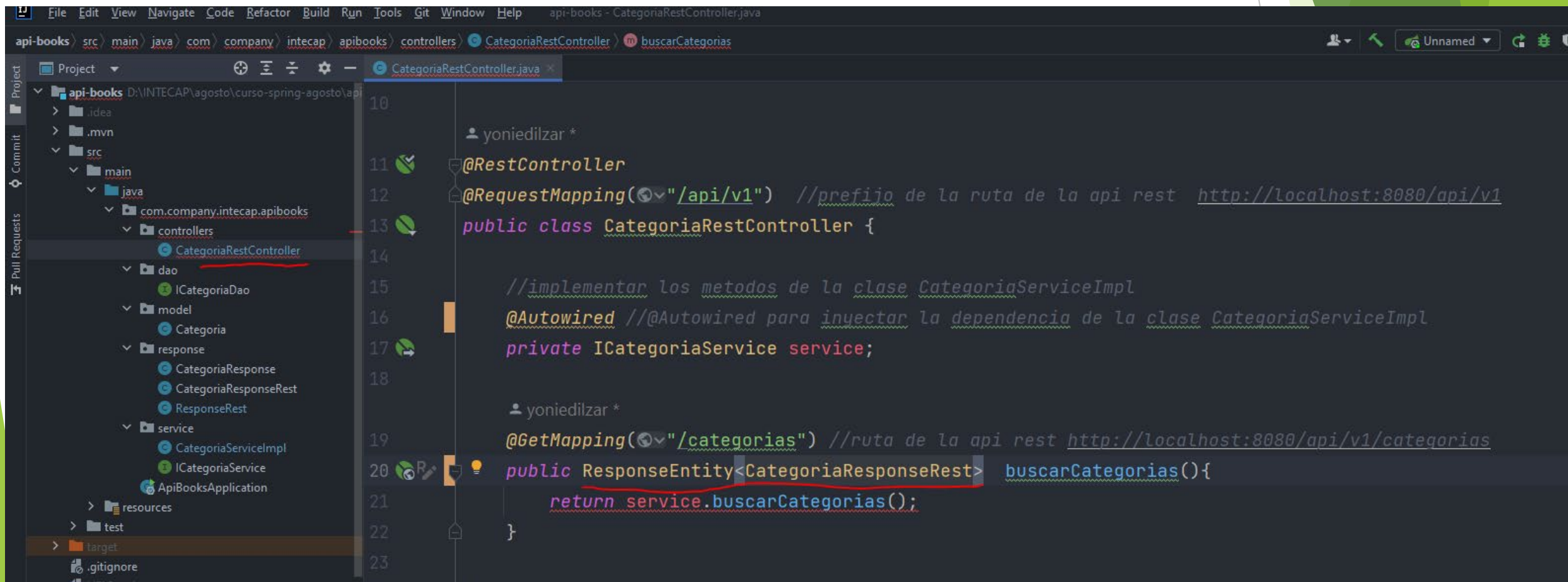
En este ejemplo, se utilizan las constantes `HttpStatus.OK`, `HttpStatus.NOT_FOUND` y `HttpStatus.INTERNAL_SERVER_ERROR` para enviar respuestas con diferentes códigos de estado y mensajes correspondientes.

Recuerda que esta es solo una introducción básica al uso de `HttpStatus` en Spring. En aplicaciones más complejas, también puedes personalizar respuestas más detalladas utilizando las clases `ResponseEntity` y `ResponseExceptionHandler`.

Uso de clase HttpStatus

Aplicar HttpStatus al servicio de categoría.

1. Controllers -> CategoriaRestController

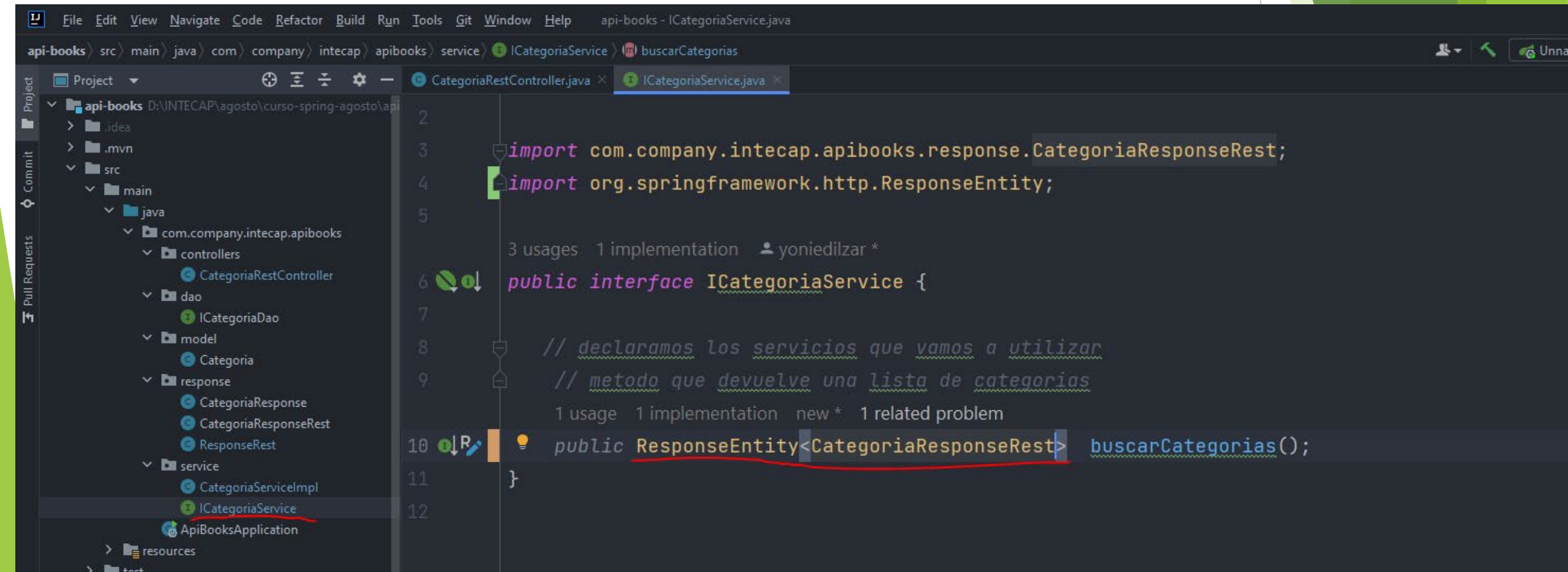


```
10
11 yoniedilzar *
12 @RestController
13 @RequestMapping("/api/v1") //prefijo de la ruta de la api rest http://localhost:8080/api/v1
14 public class CategoriaRestController {
15     //implementar los metodos de la clase CategoriaServiceImpl
16     @Autowired //@Autowired para inyectar la dependencia de la clase CategoriaServiceImpl
17     private ICategoriaService service;
18
19     yoniedilzar *
20     @GetMapping("/categorias") //ruta de la api rest http://localhost:8080/api/v1/categorias
21     public ResponseEntity<CategoriaResponseRest> buscarCategorias(){
22         return service.buscarCategorias();
23     }
24 }
```


Uso de clase HttpStatus

Aplicar HttpStatus al servicio de categoría.

2. Service -> ICategoriaService



The screenshot shows an IDE with the following components:

- Project Explorer (Left):** Displays the project structure for 'api-books'. The 'service' package contains 'ICategoriaService' and 'CategoriaServiceImpl'. The 'response' package contains 'CategoriaResponse', 'CategoriaResponseRest', and 'ResponseRest'.
- Editor (Right):** Shows the code for 'ICategoriaService.java'.
 - Line 3: `import com.company.intecap.apibooks.response.CategoriaResponseRest;`
 - Line 4: `import org.springframework.http.ResponseEntity;`
 - Line 6: `public interface ICategoriaService {`
 - Line 8: `// declaramos los servicios que vamos a utilizar`
 - Line 9: `// metodo que devuelve una lista de categorias`
 - Line 10: `public ResponseEntity<CategoriaResponseRest> buscarCategorias();` (The `ResponseEntity` is underlined in red).
 - Line 11: `}`

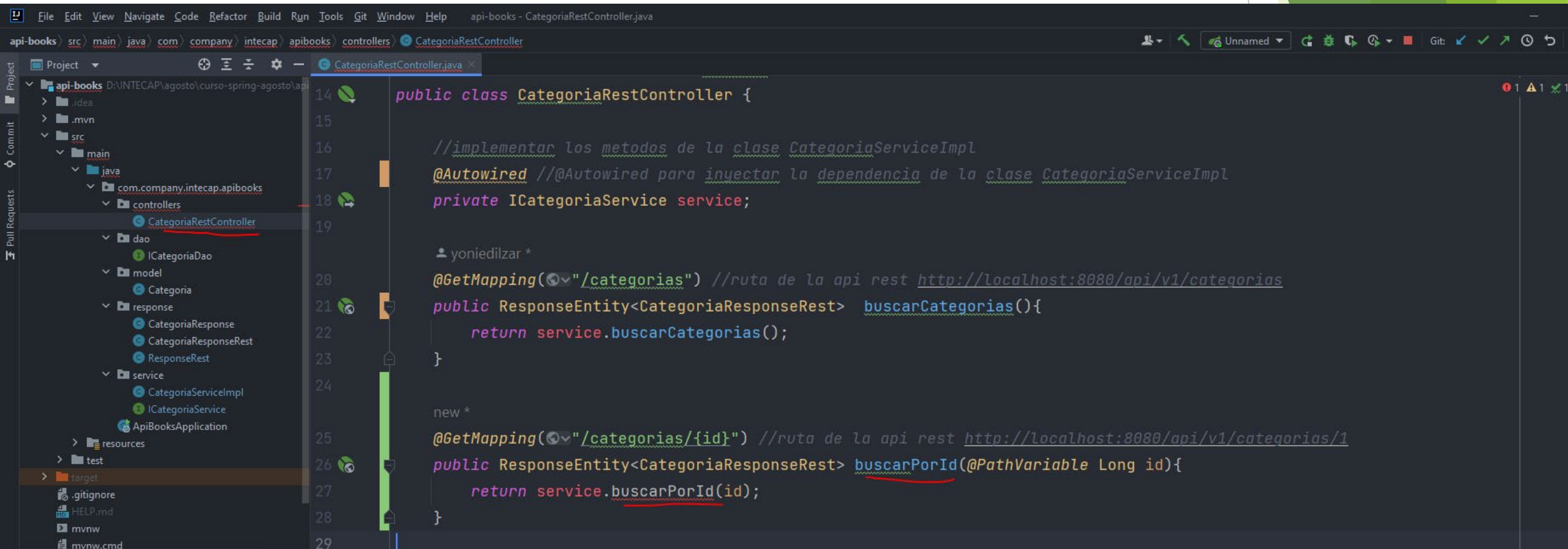
Aplicar `HttpStatus` al servicio de categoría.

3. Service -> CategoriaServiceImpl

```
api-books - CategoriaServiceImpl.java
File Edit View Navigate Code Refactor Build Run Tools Git Window Help
api-books > src > main > java > com > company > intecap > apibooks > service > CategoriaServiceImpl > buscarCategorias
Project
api-books
  .idea
  .mvn
  src
    main
      java
        com.compan.y.intecap.apibooks
          controllers
            CategoriaRestController
          dao
            ICategoriaDao
          model
            Categoria
          response
            CategoriaResponse
            CategoriaResponseRest
Pull Requests
1 usage yoniedilzar *
23 private ICategoriaDao categoriaDao;
24
25
26 @Override
27 @Transactional(readOnly = true) //para que sea de solo lectura (package import org.springframework.transaction.annotation.Transactional)
28 public ResponseEntity<CategoriaResponseRest> buscarCategorias() {
29     //aqui se implementa la logica de negocio
30
31     log.info("inicio metodo buscarCategorias()");
32
33     try {
34         List<Categoria> categoria = (List<Categoria>) categoriaDao.findAll(); //lista de categorias que se obtiene del metodo findAll
35         //va a la base de datos a traves del metodo findAll de la Intefaz crudRepository y trae todas las categorias de la tabla
36         response.getCategoriaResponse().setCategorias(categoria); //obtiene la lista de categorias y lo setea a setCategoriaResponse
37         response.setMetadata( tipo: "Respuesta ok", codigo: "200", date: "Respuesta exitosa");
38     } catch (Exception e) {
39         response.setMetadata( tipo: "Respuesta no ok", codigo: "-1", date: "Error al consultar categorias");
40         log.error("Error al consultar categorias {}", e.getMessage());
41         e.printStackTrace(); //envia el error a la consola e.getMessage() para mostrar el mensaje de error
42         return new ResponseEntity<CategoriaResponseRest>(response, HttpStatus.INTERNAL_SERVER_ERROR);
43     }
44
45     return new ResponseEntity<CategoriaResponseRest>(response, HttpStatus.OK);
46 }
47
48
49
50
```

Desarrollo CRUD de API Rest Categoría

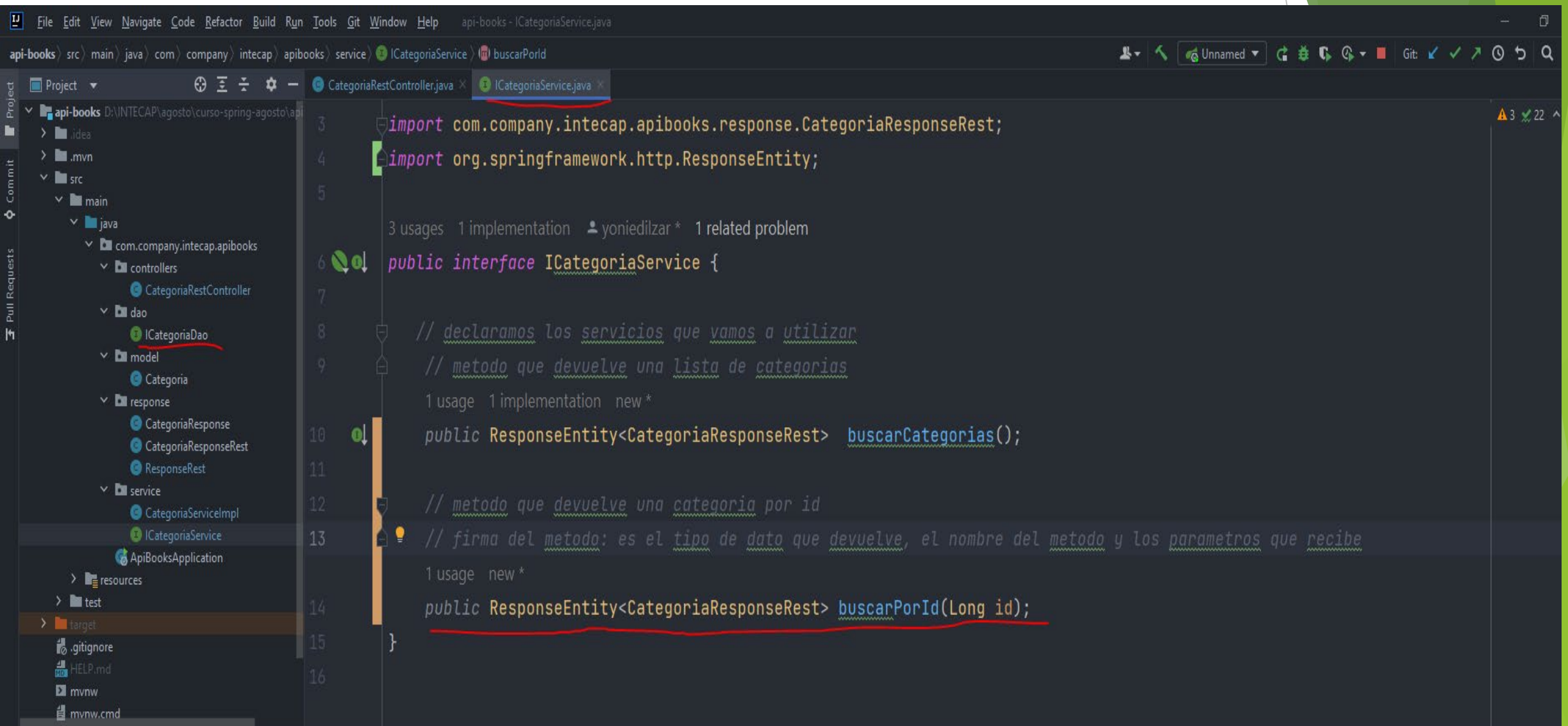
API buscar por id - Uso de PathVariable



The screenshot shows an IDE with the following components:

- Project Explorer (Left):** Displays the project structure for 'api-books'. The path is `src/main/java/com/company/intecap/apibooks/controllers`. The file `CategoriaRestController` is selected and highlighted.
- Code Editor (Center):** Shows the implementation of `CategoriaRestController`. The code includes:
 - Line 14: `public class CategoriaRestController {`
 - Line 16: `//implementar los metodos de la clase CategoriaServiceImpl`
 - Line 17: `@Autowired // @Autowired para inyectar la dependencia de la clase CategoriaServiceImpl`
 - Line 18: `private ICategoriaService service;`
 - Line 20: `@GetMapping("/categorias") // ruta de la api rest http://localhost:8080/api/v1/categorias`
 - Line 21: `public ResponseEntity<CategoriaResponseRest> buscarCategorias(){`
 - Line 22: `return service.buscarCategorias();`
 - Line 23: `}`
 - Line 25: `@GetMapping("/{id}") // ruta de la api rest http://localhost:8080/api/v1/categorias/1`
 - Line 26: `public ResponseEntity<CategoriaResponseRest> buscarPorId(@PathVariable Long id){`
 - Line 27: `return service.buscarPorId(id);`
 - Line 28: `}`
 - Line 29: `}`
- Run/Debug Console (Right):** Shows a status bar with a red error icon and a warning icon.

Creamos el controlador buscarPorId(Long id)



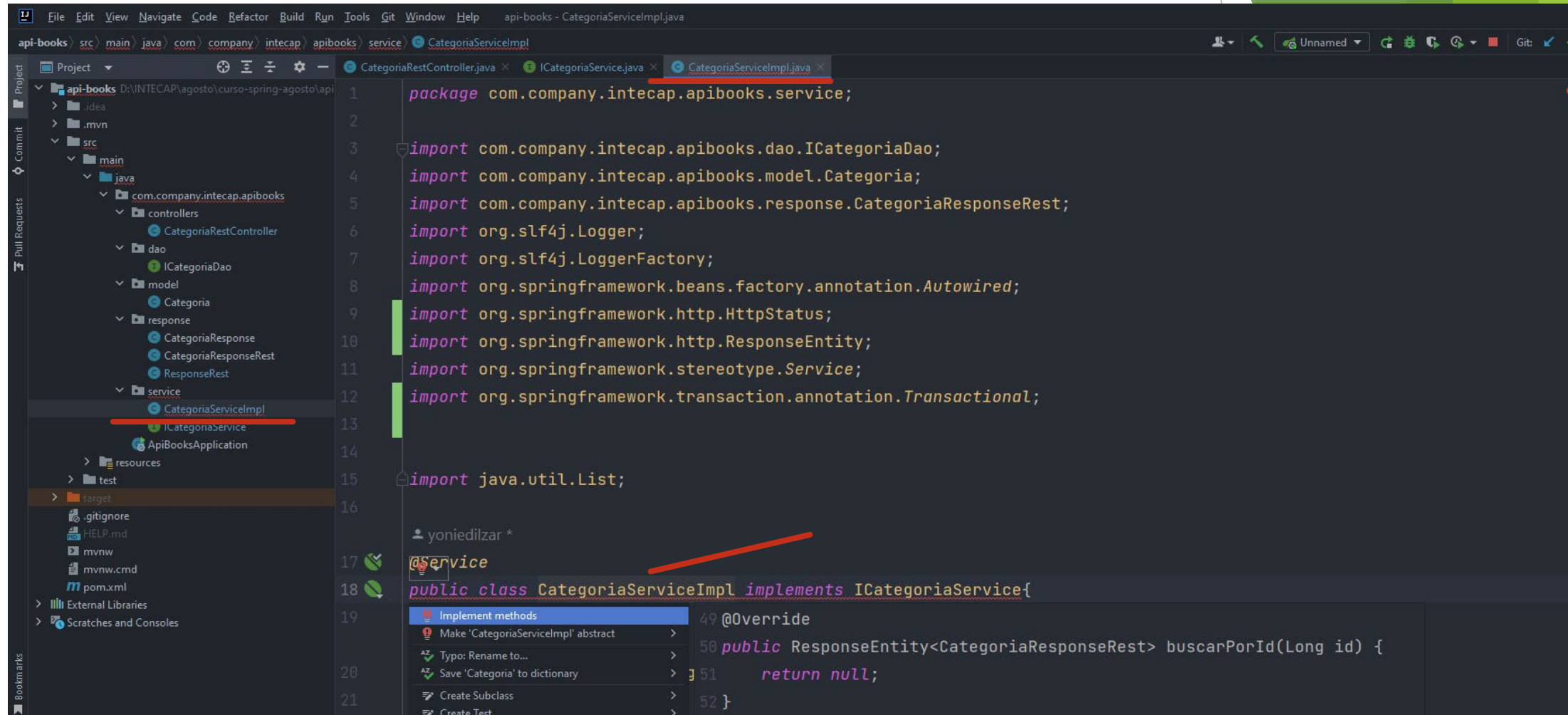
The screenshot shows an IDE with the following components:

- Project Explorer (Left):** Displays the project structure. The path `com.company.intecap.apibooks.service` is expanded, showing `ICategoriaService` highlighted.
- Editor (Center):** Shows the `ICategoriaService.java` file. The code is as follows:

```
3 import com.company.intecap.apibooks.response.CategoriaResponseRest;
4 import org.springframework.http.ResponseEntity;
5
6 public interface ICategoriaService {
7
8     // declaramos los servicios que vamos a utilizar
9     // metodo que devuelve una lista de categorias
10    public ResponseEntity<CategoriaResponseRest> buscarCategorias();
11
12    // metodo que devuelve una categoria por id
13    // firma del metodo: es el tipo de dato que devuelve, el nombre del metodo y los parametros que recibe
14    public ResponseEntity<CategoriaResponseRest> buscarPorId(Long id);
15 }
```

The `buscarPorId` method signature is underlined in red. A tooltip for `buscarCategorias` is visible, showing "3 usages 1 implementation yoniedilzar * 1 related problem".
- Right Margin:** Shows a warning icon and the number "3".

En service → Creamos la firma del método y lo implementamos – de buscarPorId



The screenshot shows an IDE with the following components:

- Project Explorer (Left):** Displays the project structure. The path `com.company.intecap.apibooks.service.CategoriaServiceImpl` is highlighted.
- Editor (Center):** Shows the `CategoriaServiceImpl.java` file. The code includes package declarations, imports for `ICategoriaDao`, `Categoria`, `CategoriaResponseRest`, `Logger`, `LoggerFactory`, `Autowired`, `HttpStatus`, `ResponseEntity`, `Service`, `Transactional`, and `List`. The `@Service` annotation is highlighted with a red arrow. The class `CategoriaServiceImpl` implements `ICategoriaService`. The `buscarPorId` method is implemented to return `null`.
- Context Menu (Bottom):** A right-click context menu is open over the `@Service` annotation, showing options like "Implement methods", "Make 'CategoriaServiceImpl' abstract", "Typo: Rename to...", "Save 'Categoria' to dictionary", "Create Subclass", and "Create Test".

```
1 package com.company.intecap.apibooks.service;
2
3 import com.company.intecap.apibooks.dao.ICategoriaDao;
4 import com.company.intecap.apibooks.model.Categoria;
5 import com.company.intecap.apibooks.response.CategoriaResponseRest;
6 import org.slf4j.Logger;
7 import org.slf4j.LoggerFactory;
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.http.HttpStatus;
10 import org.springframework.http.ResponseEntity;
11 import org.springframework.stereotype.Service;
12 import org.springframework.transaction.annotation.Transactional;
13
14
15 import java.util.List;
16
17 @Service
18 public class CategoriaServiceImpl implements ICategoriaService{
19
20     @Override
21     public ResponseEntity<CategoriaResponseRest> buscarPorId(Long id) {
22         return null;
23     }
24 }
```

Creamos la lógica de Implementación. Utilizando crudRepository

The screenshot shows an IDE with the following components:

- Project Structure (Left):** The project is named 'api-books'. The file 'CategoriaServiceImpl.java' is located in the path: `src/main/java/com/company/intecap/apibooks/service/`.
- Code Editor (Right):** The file 'CategoriaServiceImpl.java' is open. The code is as follows:

```
52  @Override
53  @Transactional(readOnly = true)
54  public ResponseEntity<CategoriaResponseRest> buscarPorId(Long id) {
55      log.info("inicio metodo buscarPorId()");
56      CategoriaResponseRest response = new CategoriaResponseRest(); //instancia de la clase CategoriaResponseRest
57      List<Categoria> list = new ArrayList<>(); //lista de categorias
58
59      try {
60          Optional<Categoria> categoria = categoriaDao.findById(id); //busca una categoria por id
61          if (categoria.isPresent()){ //si la categoria existe
62              list.add(categoria.get()); //agrega la categoria a la lista
63              response.getCategoriaResponse().setCategorias(list); //obtiene la lista de categorias y lo setea a setCategorias
64          }else{
65              log.error("Error al consultar categoria {}", id);
66              response.setMetadata( tipo: "Respuesta no ok", codigo: "-1", date: "Categoria no encontrada");
67              return new ResponseEntity<CategoriaResponseRest>(response, HttpStatus.NOT_FOUND);
68          }
69      }catch (Exception e){
70          log.error("Error al consultar categorias {}", e.getMessage());
71          response.setMetadata( tipo: "Respuesta no ok", codigo: "-1", date: "Error al consultar categorias");
72          return new ResponseEntity<CategoriaResponseRest>(response, HttpStatus.INTERNAL_SERVER_ERROR);
73      }
74      response.setMetadata( tipo: "Respuesta ok", codigo: "200", date: "Respuesta exitosa");
75      return new ResponseEntity<CategoriaResponseRest>(response, HttpStatus.OK);
76  }
77  }
```


API guardarCategoria-creamos el controlador

The screenshot shows an IDE window with the following components:

- Project Explorer (Left):** Shows the project structure. The path is `ap [api-books] > src > main > java > com > company > intecap > apibooks > controllers > CategoriaRestController`. The file `CategoriaRestController` is selected.
- Editor (Center):** Displays the code for `CategoriaRestController.java`. The code includes:
 - Line 18: `@GetMapping("/categorias")` with a comment `//ruta de la api rest http://localhost:8080/api/v1/categorias`.
 - Line 19: `public ResponseEntity<CategoriaResponseRest> buscarCategorias() { return service.buscarCategorias(); }`
 - Line 23: `@GetMapping("/categorias/{id}")` with a comment `//ruta de la api rest http://localhost:8080/api/v1/categorias/1`.
 - Line 24: `public ResponseEntity<CategoriaResponseRest> buscarPorId(@PathVariable Long id){`
 - Line 25: `return service.buscarPorId(id);`
 - Line 26: `}`
 - Line 27: `@PostMapping("/categorias")` with a comment `//ruta de la api rest http://localhost:8080/api/v1/categorias`.
 - Line 28: `public ResponseEntity<CategoriaResponseRest> crear(@RequestBody Categoria request){` with a comment `//RequestBody para recibir los datos en for`.
 - Line 29: `return service.crear(request);`
 - Line 30: `}`
 - Line 31: `}`
 - Line 32: `}`
- Run Console (Bottom):** Shows the command `Run: ApiBooksApplication`.

API guardarCategoria-Creamos la firma

The screenshot shows an IDE window with the file `ICategoriaService.java` open. The project structure on the left includes `ap [api-books]` with subdirectories `src/main/java/com/company/intecap/apibooks/service`. The code in the editor defines a public interface `ICategoriaService` with three methods: `buscarCategorias()`, `buscarPorId(Long id)`, and `crear(Categoria categoria)`. Each method is preceded by a comment in Spanish and a usage note. The IDE interface includes a menu bar, a toolbar, and a run console at the bottom.

```
1 public interface ICategoriaService {
2
3     // declaramos los servicios que vamos a utilizar
4     // metodo que devuelve una lista de categorias
5     1 usage 1 implementation yoniedilzar
6     public ResponseEntity<CategoriaResponseRest> buscarCategorias();
7
8     // metodo que devuelve una categoria por id
9     1 usage 1 implementation yoniedilzar
10    public ResponseEntity<CategoriaResponseRest> buscarPorId(Long id);
11
12    // metodo que crea una categoria
13    1 usage 1 implementation yoniedilzar
14    public ResponseEntity<CategoriaResponseRest> crear(Categoria categoria);
15
16 }
17
```

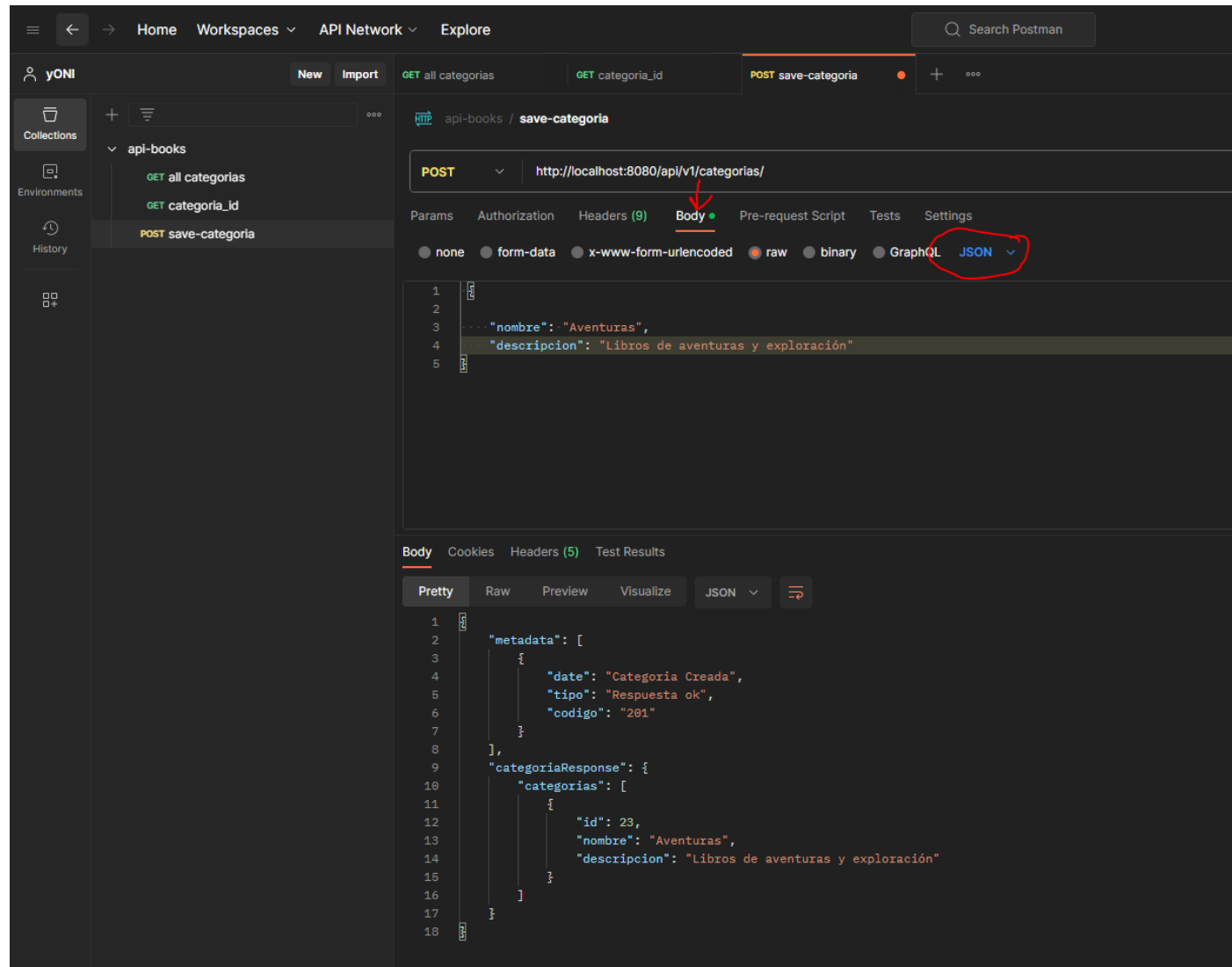
API guardarCategoria-Implementamos – Lógica de negocio

The screenshot shows an IDE with the following components:

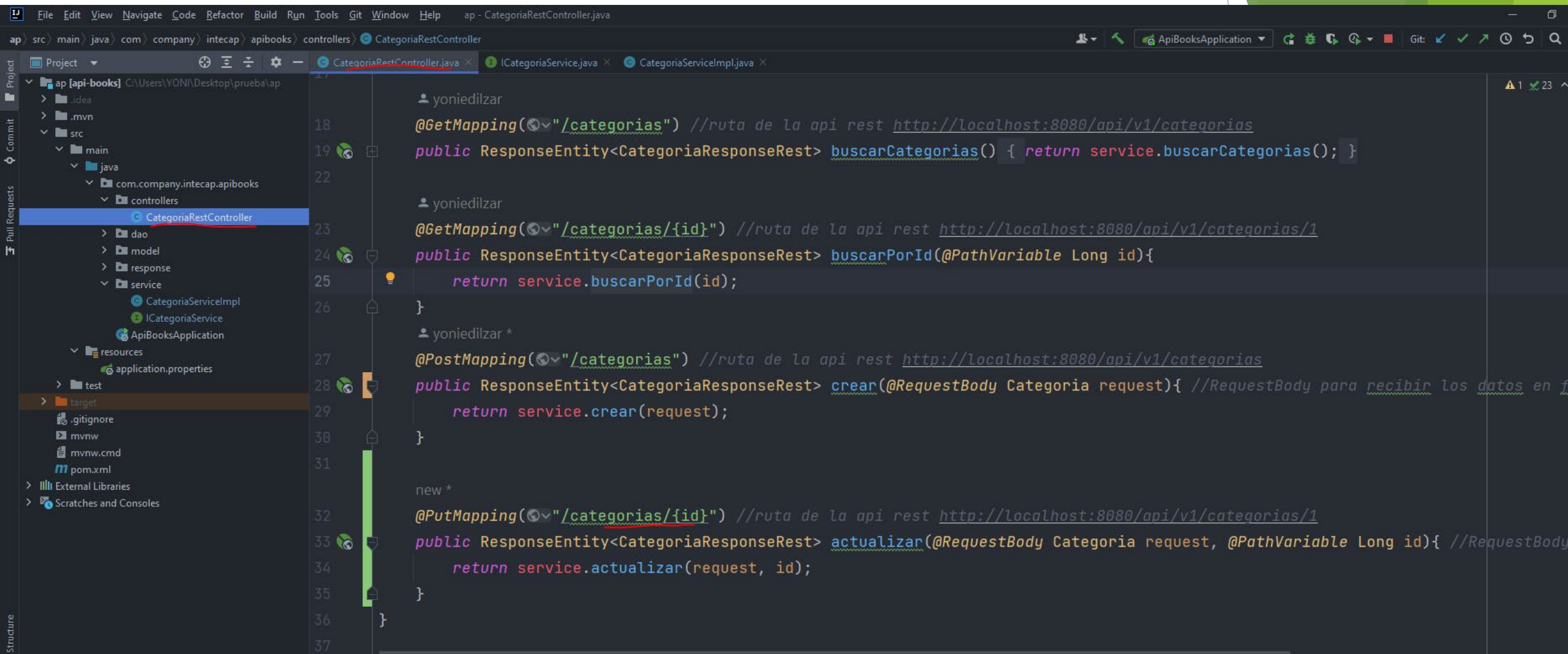
- Project Explorer (Left):** Shows the project structure with folders like `src/main/java/com/company/intecap/apibooks/service`. The file `CategoriaServiceImpl.java` is selected.
- Code Editor (Center):** Displays the implementation of the `crear` method in `CategoriaServiceImpl`. The code is annotated with comments in Spanish explaining the logic.
- Run/Console (Bottom):** Shows the application running with the `ApiBooksApplication` context.

```
80  @Override
81  @Transactional //Si algo sale mal un rollback: deshace la transaccion y Si todo sale bien un commit: guarda la transaccion
82  public ResponseEntity<CategoriaResponseRest> crear(Categoria categoria) {
83      log.info("inicio metodo crear() Categoria");
84      CategoriaResponseRest response = new CategoriaResponseRest(); //instancia de la clase CategoriaResponseRest
85      List<Categoria> list = new ArrayList<>(); //lista de categorias
86
87      try{
88          Categoria categoriaGuardada = categoriaDao.save(categoria); //guarda la categoria en la base de datos save() es un metodo
89          if(categoriaGuardada != null){ //si la categoria se guardo correctamente
90              list.add(categoriaGuardada); //agrega la categoria a la lista
91              response.getCategoriaResponse().setCategorias(list); //setea la lista de categorias a la clase CategoriaResponseRest
92          }else{
93              log.error("Error al guardar categoria {}", categoria.toString());
94              response.setMetadata( tipo: "Respuesta no ok", codigo: "400", date: "Categoria no guardada");
95              return new ResponseEntity<CategoriaResponseRest>(response, HttpStatus.BAD_REQUEST); //error 400
96          }
97      }catch (Exception e){
98          log.error("Error al crear categoria {}", e.getMessage());
99          response.setMetadata( tipo: "Respuesta no ok", codigo: "500", date: "Error al crear categoria");
100         return new ResponseEntity<CategoriaResponseRest>(response, HttpStatus.INTERNAL_SERVER_ERROR); //error 500
101     }
102     response.setMetadata( tipo: "Respuesta ok", codigo: "201", date: "Categoria Creada");
103     return new ResponseEntity<CategoriaResponseRest>(response, HttpStatus.OK); //respuesta exitosa
104 }
105 }
```

API guardarCategoria-Probamos con Postman



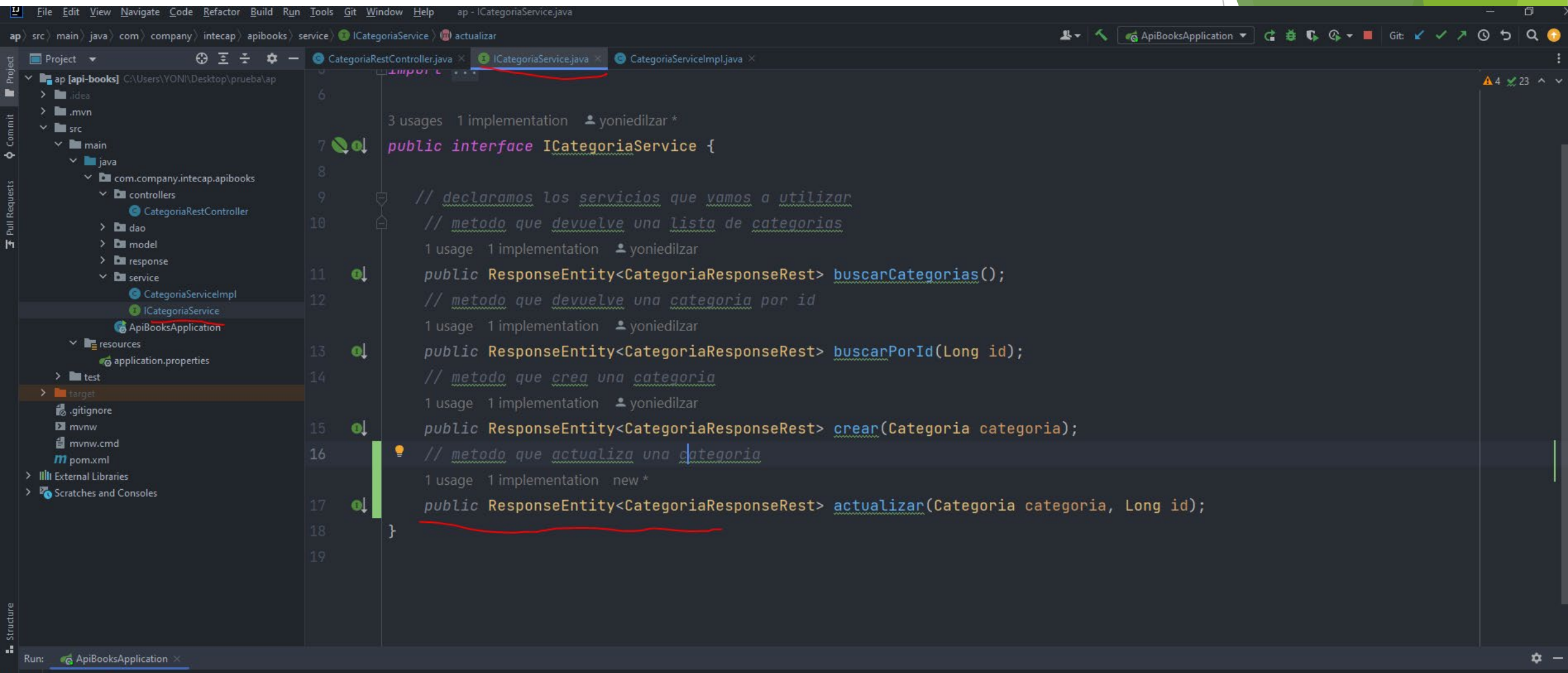
API ActualizarCategoria-Controllador



The screenshot shows an IDE window with the file `CategoriaRestController.java` open. The project structure on the left includes `ap [api-books]`, `src`, `main`, `java`, `com.company.intecap.apibooks`, `controllers`, `CategoriaRestController`, `dao`, `model`, `response`, `service`, `CategoriaServiceImpl`, `ICategoriaService`, `ApiBooksApplication`, `resources`, `application.properties`, `test`, `target`, `.gitignore`, `mvnw`, `mvnw.cmd`, `pom.xml`, `External Libraries`, and `Scratches and Consoles`.

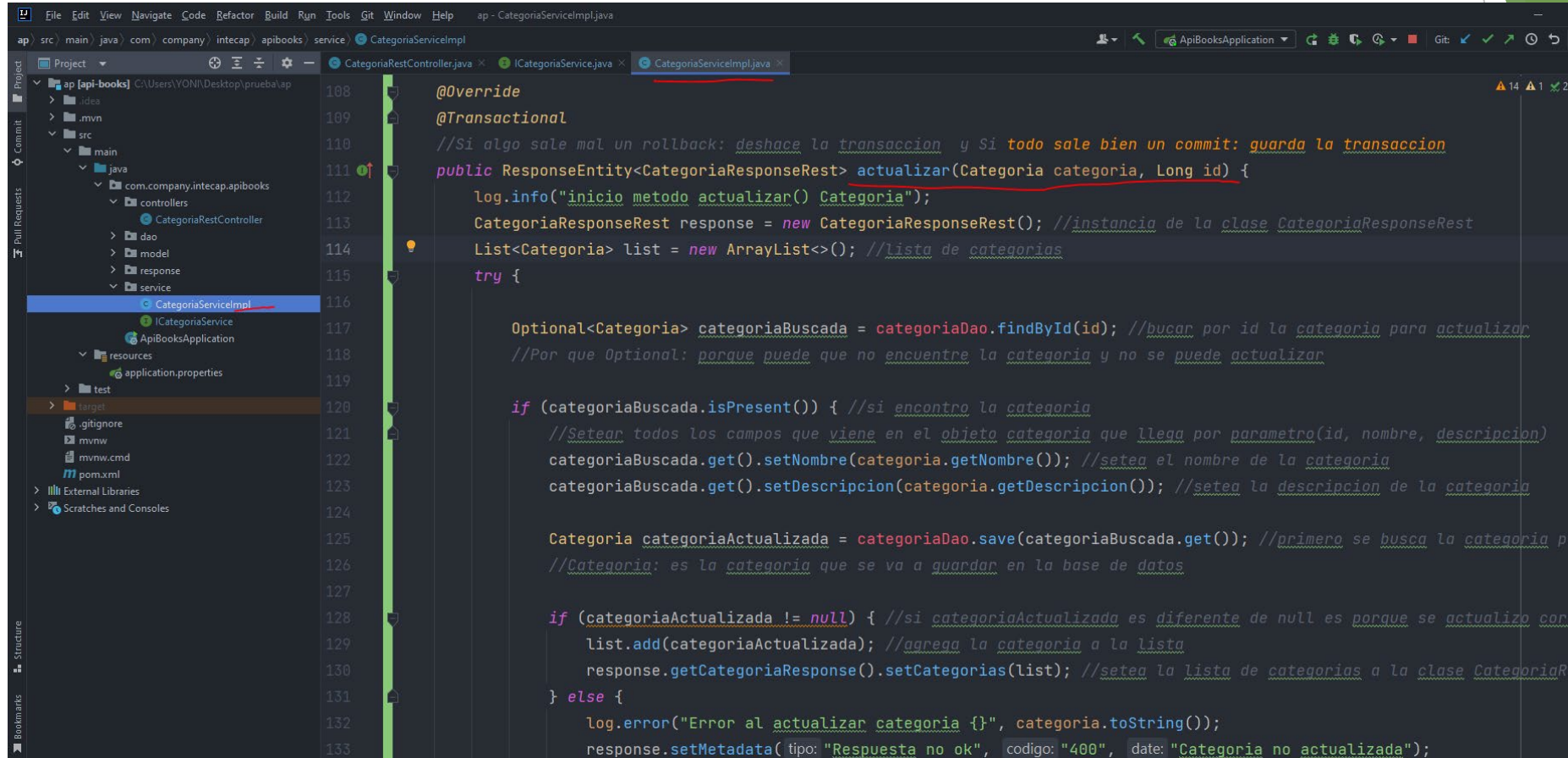
```
18  yoniedilzar
19  @GetMapping("/categorias") //ruta de la api rest http://localhost:8080/api/v1/categorias
20  public ResponseEntity<CategoriaResponseRest> buscarCategorias() { return service.buscarCategorias(); }
21
22  yoniedilzar
23  @GetMapping("/categorias/{id}") //ruta de la api rest http://localhost:8080/api/v1/categorias/1
24  public ResponseEntity<CategoriaResponseRest> buscarPorId(@PathVariable Long id){
25      return service.buscarPorId(id);
26  }
27
28  yoniedilzar *
29  @PostMapping("/categorias") //ruta de la api rest http://localhost:8080/api/v1/categorias
30  public ResponseEntity<CategoriaResponseRest> crear(@RequestBody Categoria request){ //RequestBody para recibir los datos en f
31      return service.crear(request);
32  }
33
34  new *
35  @PutMapping("/categorias/{id}") //ruta de la api rest http://localhost:8080/api/v1/categorias/1
36  public ResponseEntity<CategoriaResponseRest> actualizar(@RequestBody Categoria request, @PathVariable Long id){ //RequestBody
37      return service.actualizar(request, id);
38  }
39  }
```


API ActualizarCategoria-Firma del metodo



Ctrl + Alt + L Ordenar código

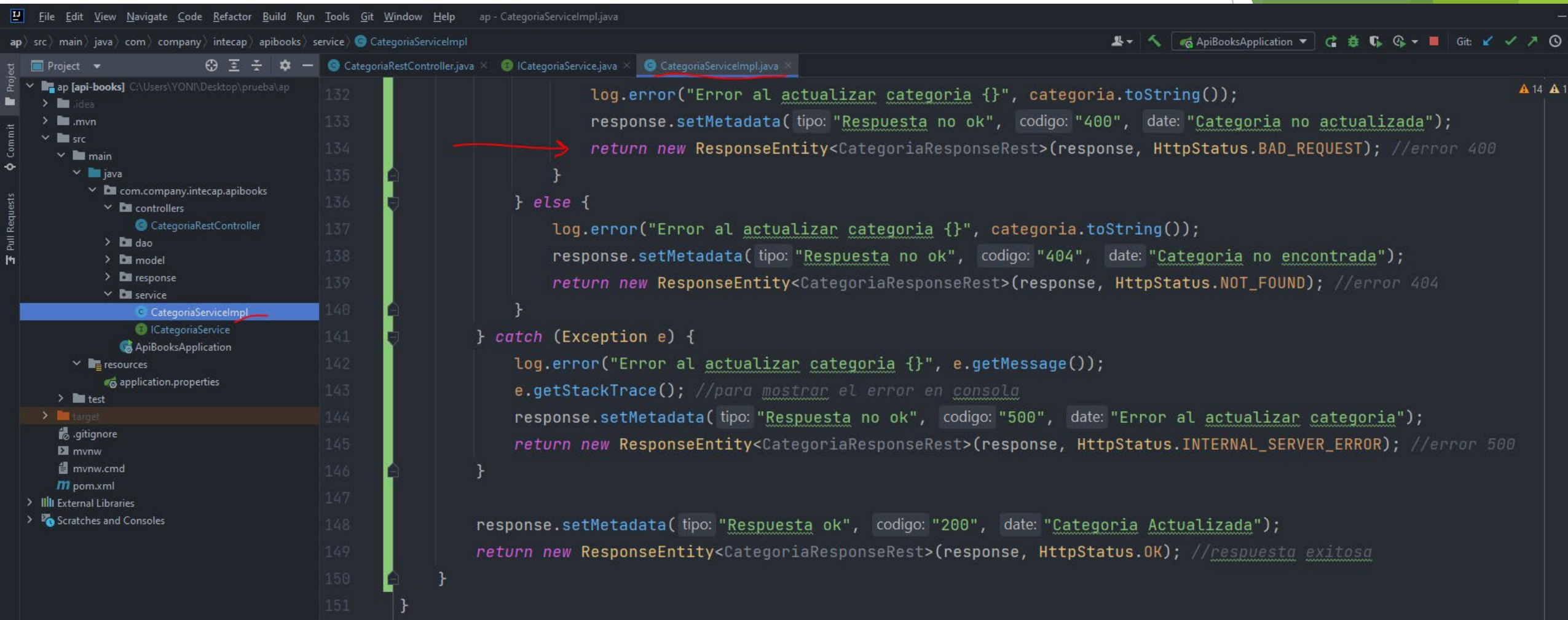
API Actualizar Categoría-Lógica de Negocio



```
108 @Override
109 @Transactional
110 //Si algo sale mal un rollback: deshace la transaccion y Si todo sale bien un commit: guarda la transaccion
111 public ResponseEntity<CategoriaResponseRest> actualizar(Categoria categoria, Long id) {
112     log.info("inicio metodo actualizar() Categoria");
113     CategoriaResponseRest response = new CategoriaResponseRest(); //instancia de la clase CategoriaResponseRest
114     List<Categoria> list = new ArrayList<>(); //lista de categorias
115     try {
116
117         Optional<Categoria> categoriaBuscada = categoriaDao.findById(id); //buscar por id la categoria para actualizar
118         //Por que Optional: porque puede que no encuentre la categoria y no se puede actualizar
119
120         if (categoriaBuscada.isPresent()) { //si encontro la categoria
121             //Setear todos los campos que viene en el objeto categoria que llega por parametro(id, nombre, descripcion)
122             categoriaBuscada.get().setNombre(categoria.getNombre()); //setea el nombre de la categoria
123             categoriaBuscada.get().setDescripcion(categoria.getDescripcion()); //setea la descripcion de la categoria
124
125             Categoria categoriaActualizada = categoriaDao.save(categoriaBuscada.get()); //primero se busca la categoria por
126             //Categoria: es la categoria que se va a guardar en la base de datos
127
128             if (categoriaActualizada != null) { //si categoriaActualizada es diferente de null es porque se actualiza correctamente
129                 list.add(categoriaActualizada); //agrega la categoria a la lista
130                 response.getCategoriaResponse().setCategorias(list); //setea la lista de categorias a la clase CategoriaResponseRest
131             } else {
132                 log.error("Error al actualizar categoria {}", categoria.toString());
133                 response.setMetadata(tipo: "Respuesta no ok", codigo: "400", date: "Categoria no actualizada");
```

Ctrl + Alt + L Ordenar código

API Actualizar Categoría-Lógica de Negocio



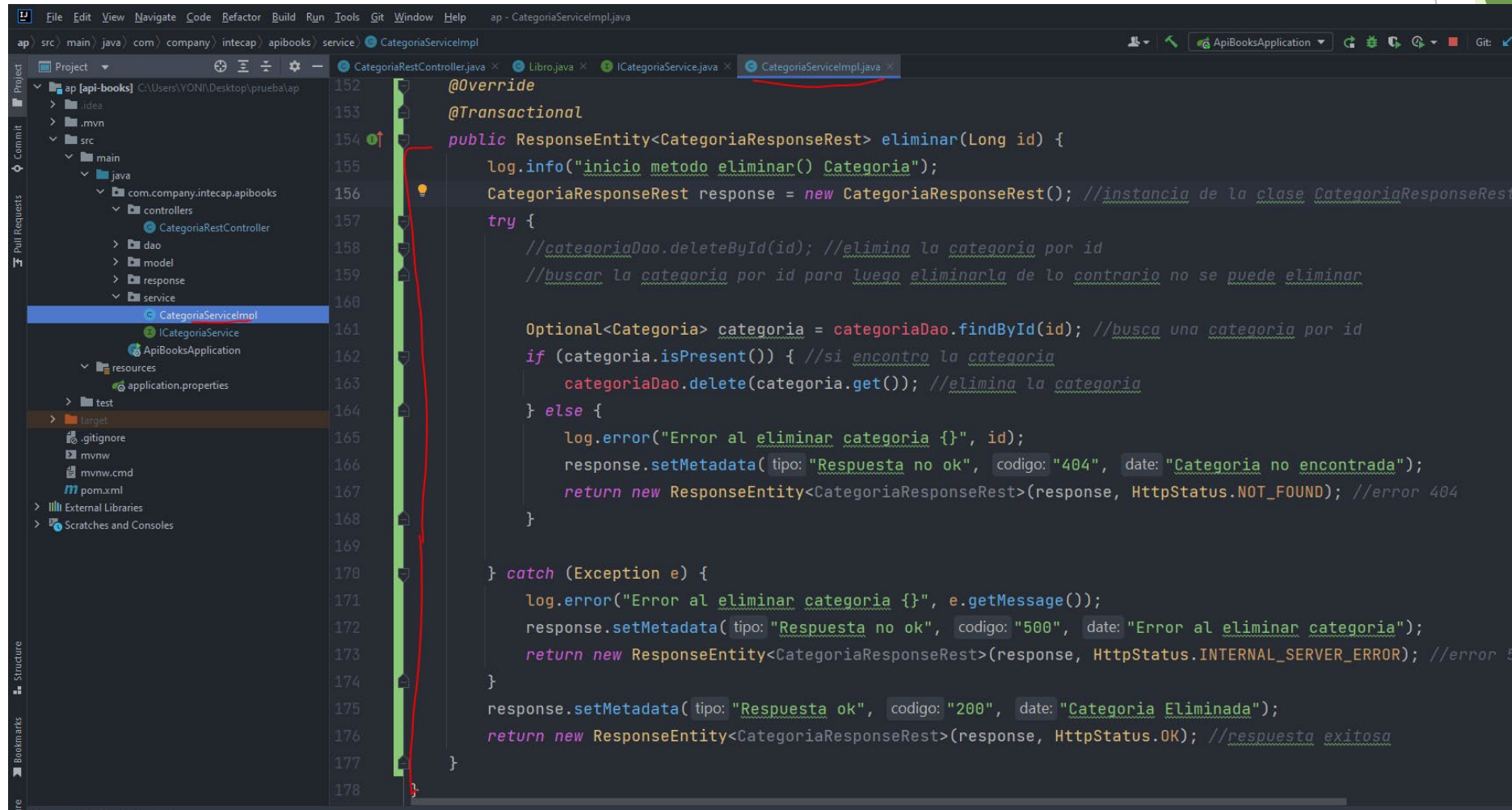
```
132     log.error("Error al actualizar categoria {}", categoria.toString());
133     response.setMetadata( tipo: "Respuesta no ok", codigo: "400", date: "Categoria no actualizada");
134     return new ResponseEntity<CategoriaResponseRest>(response, HttpStatus.BAD_REQUEST); //error 400
135 }
136 } else {
137     log.error("Error al actualizar categoria {}", categoria.toString());
138     response.setMetadata( tipo: "Respuesta no ok", codigo: "404", date: "Categoria no encontrada");
139     return new ResponseEntity<CategoriaResponseRest>(response, HttpStatus.NOT_FOUND); //error 404
140 }
141 } catch (Exception e) {
142     log.error("Error al actualizar categoria {}", e.getMessage());
143     e.printStackTrace(); //para mostrar el error en consola
144     response.setMetadata( tipo: "Respuesta no ok", codigo: "500", date: "Error al actualizar categoria");
145     return new ResponseEntity<CategoriaResponseRest>(response, HttpStatus.INTERNAL_SERVER_ERROR); //error 500
146 }
147
148 response.setMetadata( tipo: "Respuesta ok", codigo: "200", date: "Categoria Actualizada");
149 return new ResponseEntity<CategoriaResponseRest>(response, HttpStatus.OK); //respuesta exitosa
150 }
151 }
```


API EliminarCategoria-

1-controller

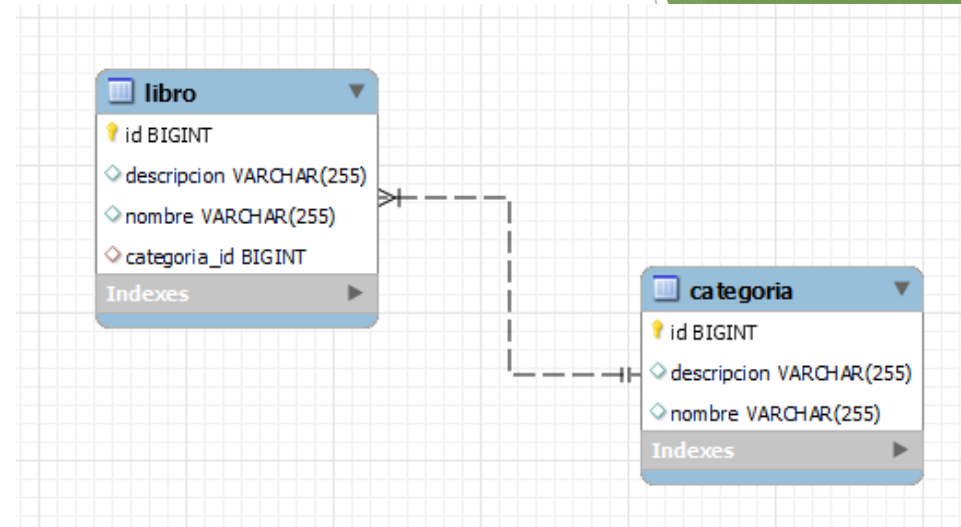
2-Interface(firma del metodo)

3-Lógica de Negocio



```
152 @Override
153 @Transactional
154 public ResponseEntity<CategoriaResponseRest> eliminar(Long id) {
155     log.info("inicio metodo eliminar() Categoria");
156     CategoriaResponseRest response = new CategoriaResponseRest(); //instancia de la clase CategoriaResponseRest
157     try {
158         //categoriaDao.deleteById(id); //elimina la categoria por id
159         //buscar la categoria por id para luego eliminarla de lo contrario no se puede eliminar
160
161         Optional<Categoria> categoria = categoriaDao.findById(id); //busca una categoria por id
162         if (categoria.isPresent()) { //si encontro la categoria
163             categoriaDao.delete(categoria.get()); //elimina la categoria
164         } else {
165             log.error("Error al eliminar categoria {}", id);
166             response.setMetadata( tipo: "Respuesta no ok", codigo: "404", date: "Categoria no encontrada");
167             return new ResponseEntity<CategoriaResponseRest>(response, HttpStatus.NOT_FOUND); //error 404
168         }
169     } catch (Exception e) {
170         log.error("Error al eliminar categoria {}", e.getMessage());
171         response.setMetadata( tipo: "Respuesta no ok", codigo: "500", date: "Error al eliminar categoria");
172         return new ResponseEntity<CategoriaResponseRest>(response, HttpStatus.INTERNAL_SERVER_ERROR); //error 5
173     }
174
175     response.setMetadata( tipo: "Respuesta ok", codigo: "200", date: "Categoria Eliminada");
176     return new ResponseEntity<CategoriaResponseRest>(response, HttpStatus.OK); //respuesta exitosa
177 }
178 }
```


•Uso de anotaciones de persistencia ManyToOne.



- La anotación @ManyToOne es una anotación de persistencia en el contexto de JPA (Java Persistence API) que se utiliza para establecer una relación de muchos a uno entre dos entidades en una base de datos relacional. Esta anotación se utiliza en el mundo de los ORM (Object-Relational Mapping) para mapear relaciones complejas entre objetos Java y tablas de bases de datos.
- Cuando utilizas la anotación @ManyToOne, estás indicando que una entidad tiene una relación con otra entidad en la que múltiples instancias de la primera entidad pueden estar relacionadas con una sola instancia de la segunda entidad. Por ejemplo, supongamos que tienes dos entidades: Libro y categoría. Una categoría puede tener muchos libros, pero un libro solo puede pertenecer a una categoría.

1.model

2. response -> LibroResponse.java

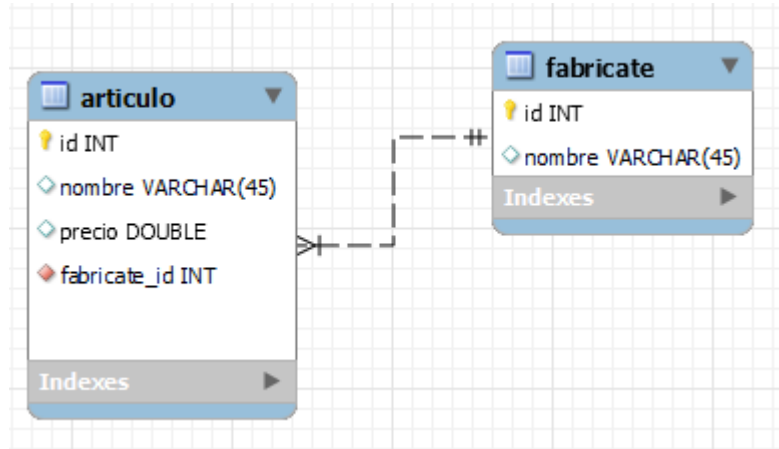
3. dao->ILibroDato (crear la Interfaz de crudRepository)

4. Service->ILibroService(crear las firmas de los métodos)

4.1 *public* ResponseEntity<LibroResponseRest> buscarLibros();

5. Service -> LibroServiceImpl

- Uso de anotaciones de persistencia ManyToOne.



Tarea: Realizar una Api RestFull

Base de datos

<https://es.slideshare.net/jsrfsmontemayor/ejercicio-sql-tienda-informatica-1>

GitHub donde se puede guiar

HomeWorkspacesAPI NetworkExplore

Search Postman

InviteSettingsNotificationsUpgrade

yONI

NewImport

GET find-all-categoriasGET find-id-categoriaPOST save-categoriaPUT update-categoriaDEL delete-categoriaGET find-all-libro

No Environment

Collections

+api-books-categoria

GET find-all-categoriasGET find-id-categoriaPOST save-categoriaPUT update-categoriaDEL delete-categoria

api-books-libro

GET find-all-libro

api-books-libro / find-all-libro

GEThttp://localhost:8080/api/v1/libros

Send

ParamsAuthorizationHeaders (7)BodyPre-request ScriptTestsSettings

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

BodyCookiesHeaders (5)Test Results

Status: 200 OKTime: 10 msSize: 458 BSave as Example

PrettyRawPreviewVisualizeJSON

```
1 {
2   "metadata": [
3     {
4       "date": "Respuesta exitosa",
5       "tipo": "Respuesta ok",
6       "codigo": "200"
7     }
8   ],
9   "libroResponse": {
10    "libros": [
11      {
12        "id": 1,
13        "nombre": "Barbuchin",
14        "descripcion": "es un libro clasico de fabulas y cuentos",
15        "categoria": {
16          "id": 2,
17          "nombre": "Infantiles",
18          "descripcion": "Libros de cuentos infantiles clásicos"
19        }
20      }
21    ]
22  }
23 }
```