

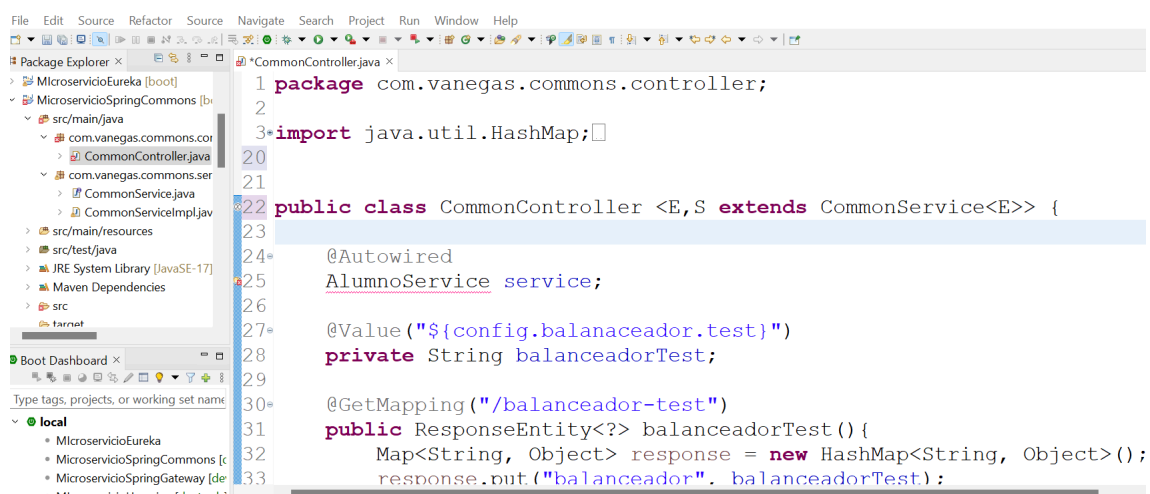
Samuel Santiago Falla Alfaro - 90180

Actividades a desarrollar para culminar el parcial dos:

1. Crear controlador genérico en la librería MicroservicioCommon

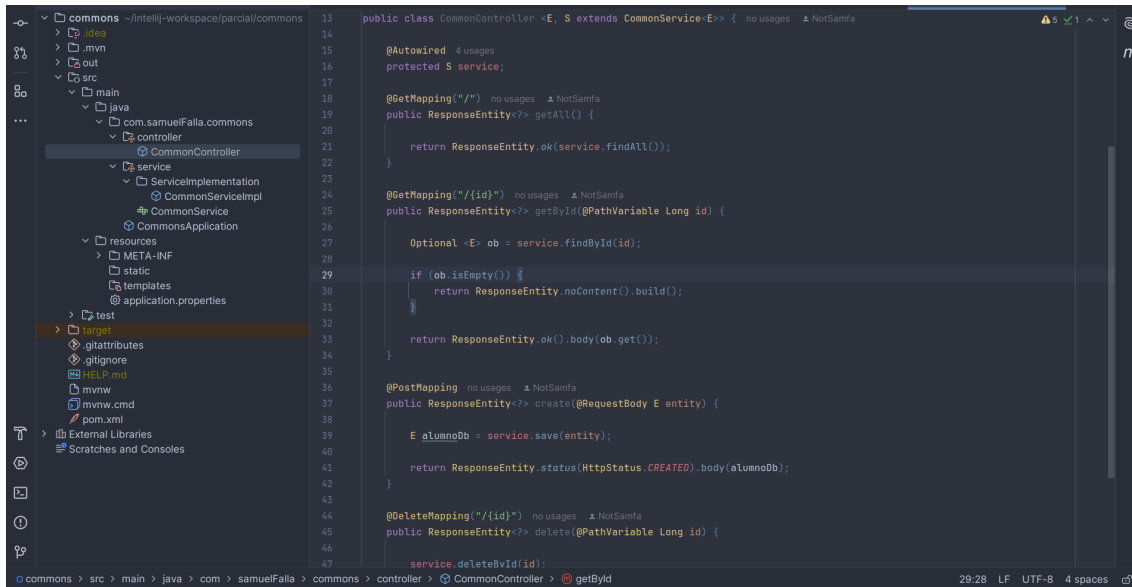
Se debe crear el controlador genérico dentro de la librería que se ha estado construyendo para ello debe hacer lo siguiente.

- Se debe copiar la clase AlumnoController que se tiene en el servicio MicroservicioUsuarios dentro del paquete controller que se encuentra en el servicio MicroservicioSpringCommons, pero antes debe renombrarla con el nombre CommonController
- Se debe depurar la clase que se ha copiado, para ello debe eliminar los import que ya no son necesarios y que están sacando error, luego se debe eliminar el decorador @RestController, ya que va hacer un controlador genérico que se va a reutilizar y no hará parte propiamente del contexto de Spring Boot ya que es una librería.
- Todos los métodos excepto el Put se reusarán, por eso debemos eliminar ese método de este controlador genérico, el cual tiene el nombre de editar
- Debemos convertir esta clase como un genérico para ello a nivel de la clase y dentro de los rombos colocamos dos genéricos el primero con la letra E correspondiente a la entidad y el segundo con la letra S correspondiente al servicio el cual hereda de la clase genérica service, separados con una coma



```
1 package com.vanegas.commons.controller;
2
3 import java.util.HashMap;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22 public class CommonController <E,S extends CommonService<E>> {
23
24     @Autowired
25     AlumnoService service;
26
27     @Value("${config.balanaceador.test}")
28     private String balanceadorTest;
29
30     @GetMapping("/balanceador-test")
31     public ResponseEntity<?> balanceadorTest() {
32         Map<String, Object> response = new HashMap<String, Object>();
33         response.put("balanceador". balanceadorTest);
```

En toda parte de esta clase donde se encuentre el nombre de la entidad Alumno se debe reemplazar con el genérico E, el cual es la entidad y la inyección que se hace de la clase AlumnoService se reemplaza por el genérico de servicio el cual esta representado por la letra S

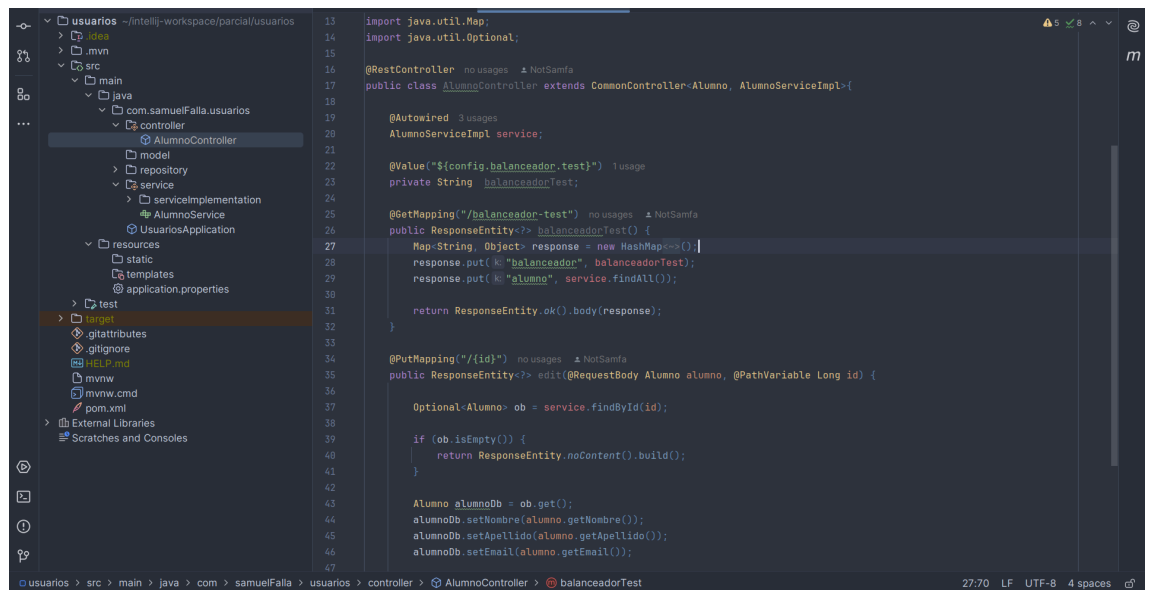


- e. En todos los lugares donde se encuentra el objeto alumno se debe reemplazar por la palabra entity ya que ese objeto ahora será un genérico
- f. Y como esta clase se heredar , entonces sus atributos no pueden ser ni privados ni p blico y menos default , deber n ser protected y por ultimo eliminamos los import no usados.

2. Refactorizar la clase controller en el servicio Usuarios

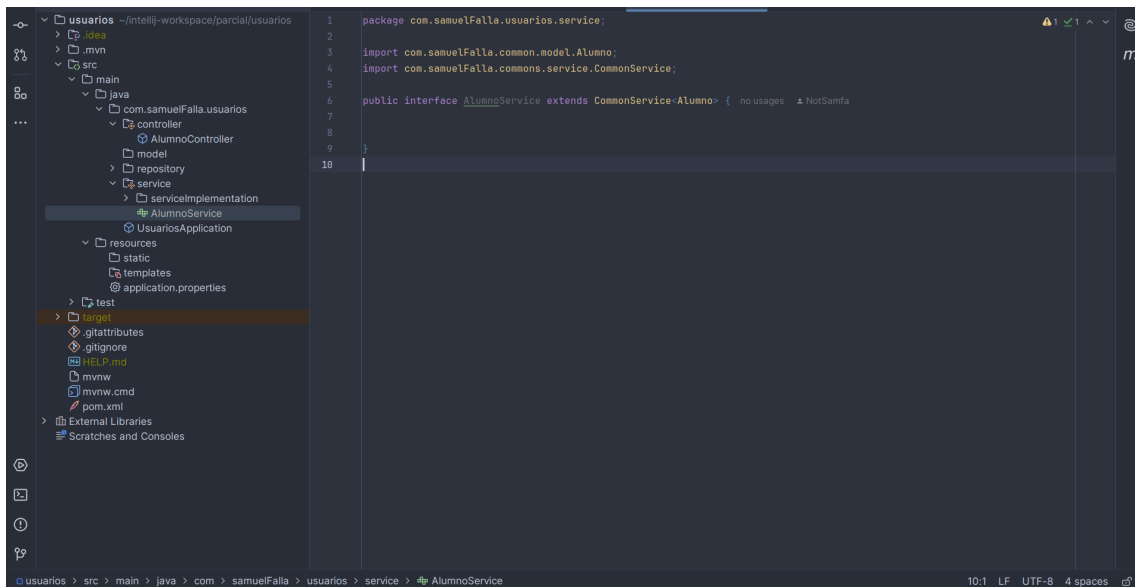
- a. Eliminar todos los m todos que se encuentran en la clase AlumnoController del servicio Usuarios excepto los m todos Put el cual esta con el nombre de editar y el m todo de comprobaci n del balanceador con el nombre de balanceadorTest(), ya que esos m todos son los  nicos que no se reusar n, ya que, ser n distintos en cada microservicio
- b. Eliminamos todos los import innecesarios y se debe extender de la clase CommonController que construimos en el punto anterior dejando como par metro entre rombos la entidad concreta que en este caso es Alumno y como segundo par metro la clase AlumnoService.

- c. Ahora se quita la clase `AlumnoService` la cual se encuentra inyectada con el decorador `@Autowired` ya que esa inyección se hace en la clase de servicio genérico que creamos en la segunda parte de esta actividad.



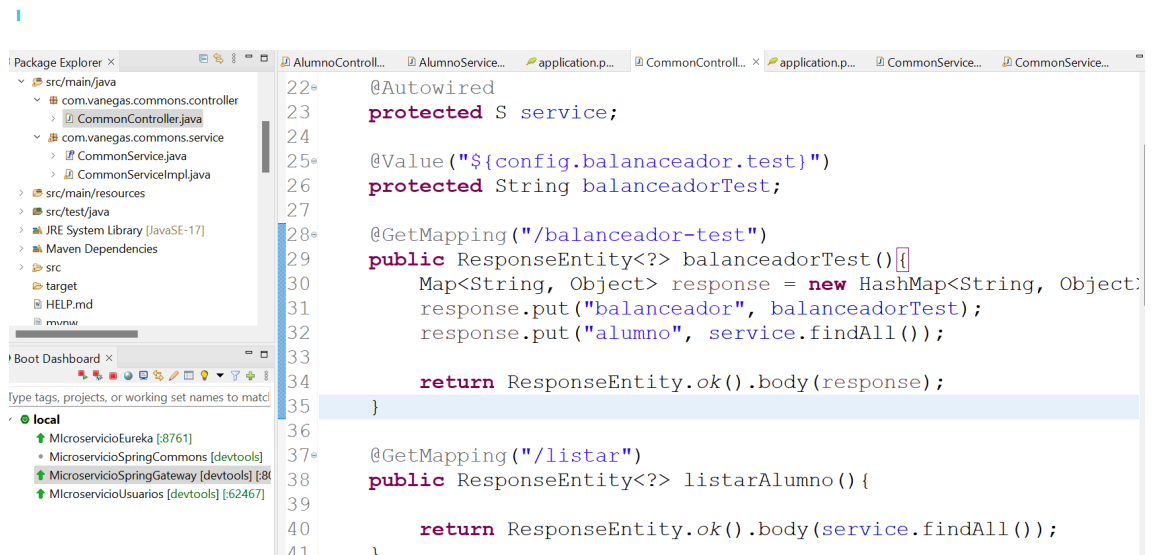
```
13 import java.util.Map;
14 import java.util.Optional;
15
16 @RestController no usages 1 NotSamfa
17 public class AlumnoController extends CommonController-Alumno, AlumnoServiceImpl-{
18
19     @Autowired 1 usages
20     AlumnoServiceImpl service;
21
22     @Value("${config.balanceador.test}") 1 usage
23     private String balanceadorTest;
24
25     @GetMapping("/balanceador-test") no usages 1 NotSamfa
26     public ResponseEntity<?> balanceadorTest() {
27         Map<String, Object> response = new HashMap<>();
28         response.put(k "balanceador", balanceadorTest);
29         response.put(k "alumno", service.findAll());
30
31         return ResponseEntity.ok().body(response);
32     }
33
34     @PutMapping("/{id}") no usages 1 NotSamfa
35     public ResponseEntity<?> edit(@RequestBody Alumno alumno, @PathVariable Long id) {
36
37         Optional<Alumno> ob = service.findById(id);
38
39         if (ob.isEmpty()) {
40             return ResponseEntity.noContent().build();
41         }
42
43         Alumno alumnoDb = ob.get();
44         alumnoDb.setNombre(alumno.getNombre());
45         alumnoDb.setApellido(alumno.getApellido());
46         alumnoDb.setEmail(alumno.getEmail());
47     }
```

- d. Ahí vas a tener un error , entonces nos vamos a la clase `AlumnoService` y quitamos todos los métodos que allí se encuentran excepto los métodos `deleteById` y `save` y colocamos que esa clase herede de `CommonService` dejando como parámetro la clase `alumno`



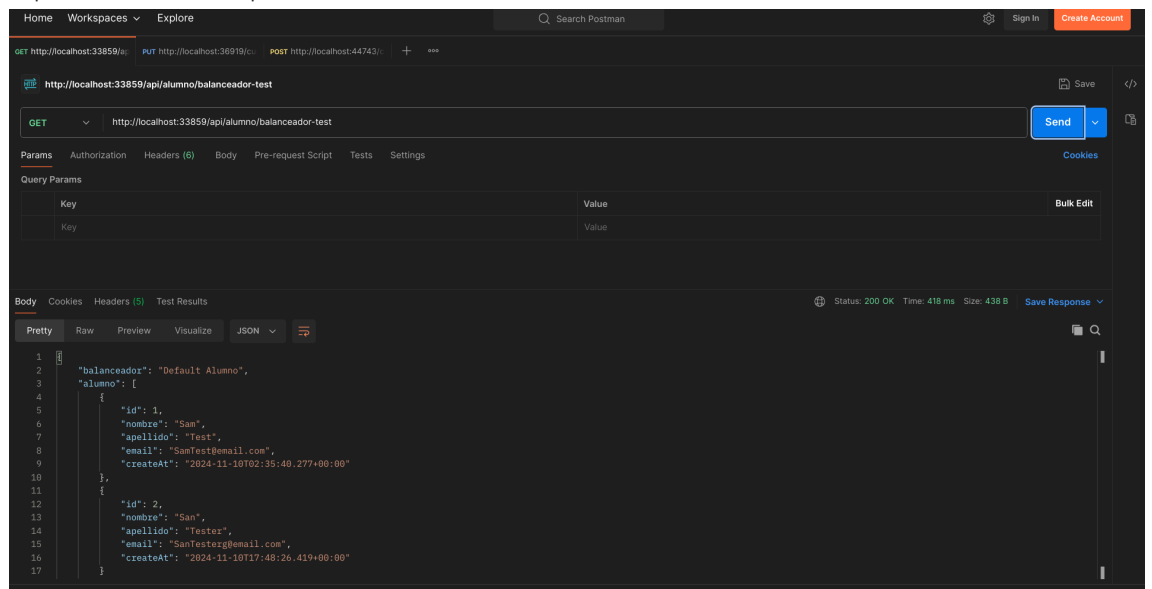
```
1 package com.samuelFalla.usuarios.service;
2
3 import com.samuelFalla.common.model.Alumno;
4 import com.samuelFalla.commons.service.CommonService;
5
6 public interface AlumnoService extends CommonService-Alumno { no usages 1 NotSamfa
7
8 }
9
10
```

- e. Colocamos en el controlador que se encuentra en el servicio Commons los paths necesarios en cada uno de los métodos expuestos como se ve a continuación:



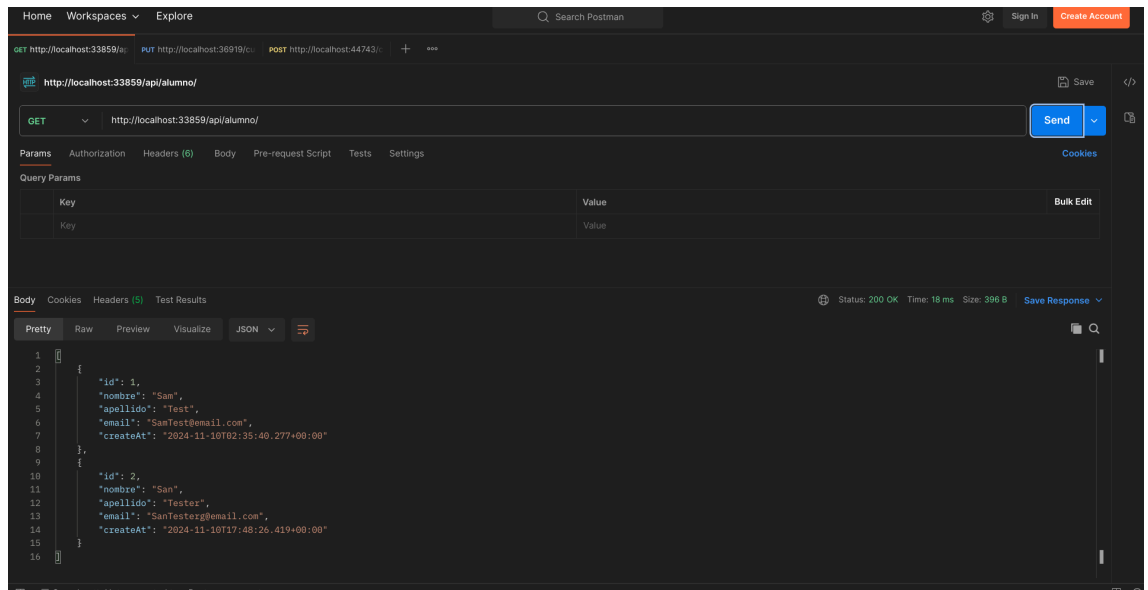
3. Ahora vamos a probar hasta aquí los servicios, para ello levantamos el servicio de Eureka, luego el servicio de usuarios y por último el Gateway

<http://localhost:8090/api/alumno/balanceador-test>

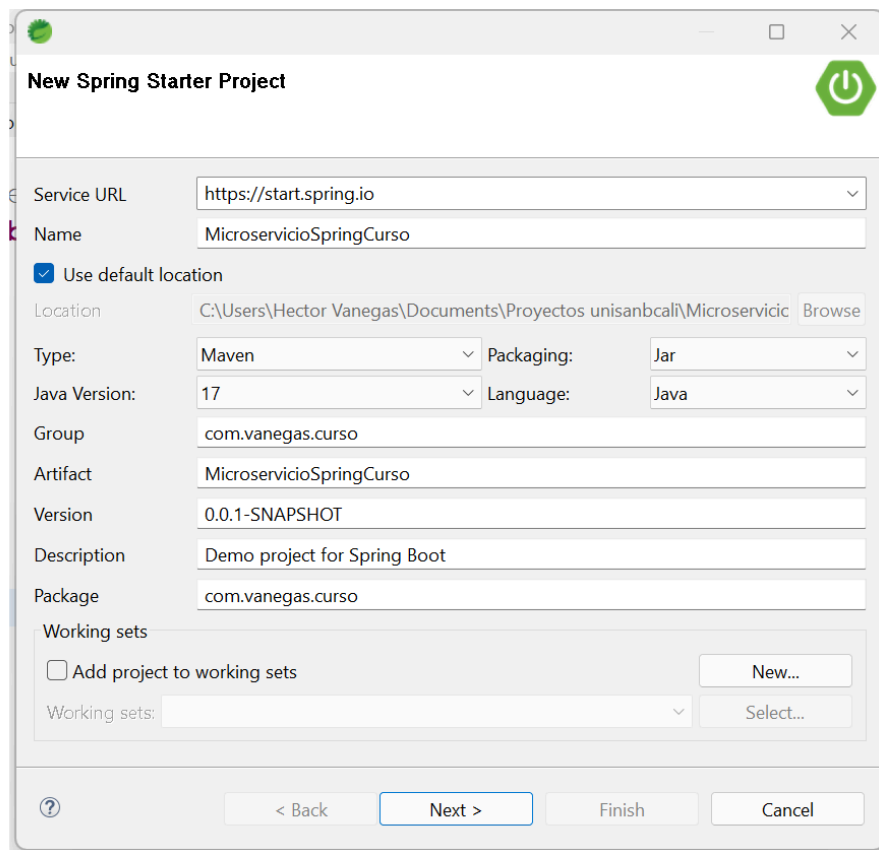


<http://localhost:8090/api/alumno/listar>

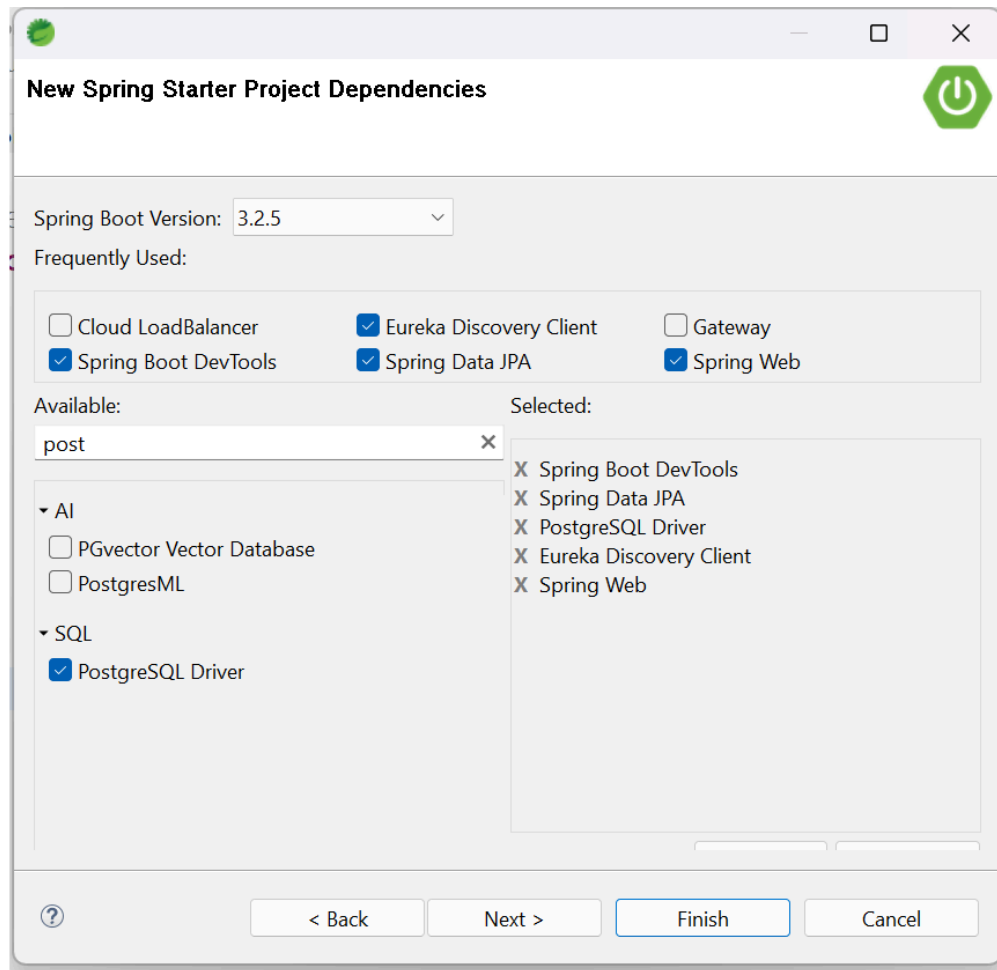
y así probar los otros métodos que se encuentran en el controlador genérico dentro del servicio Commons



4. Crear el servicio Curso



Con estas dependencias



1. Se debe inyectar la dependencia del servicio Commons en el pom de este servicio tal y como se hizo con el servicio Usuario

```
<dependency>
```

```
<groupId>com.vanegas.commons</groupId>
```

```
<artifactId>MicroservicioSpringCommons</artifactId>
```

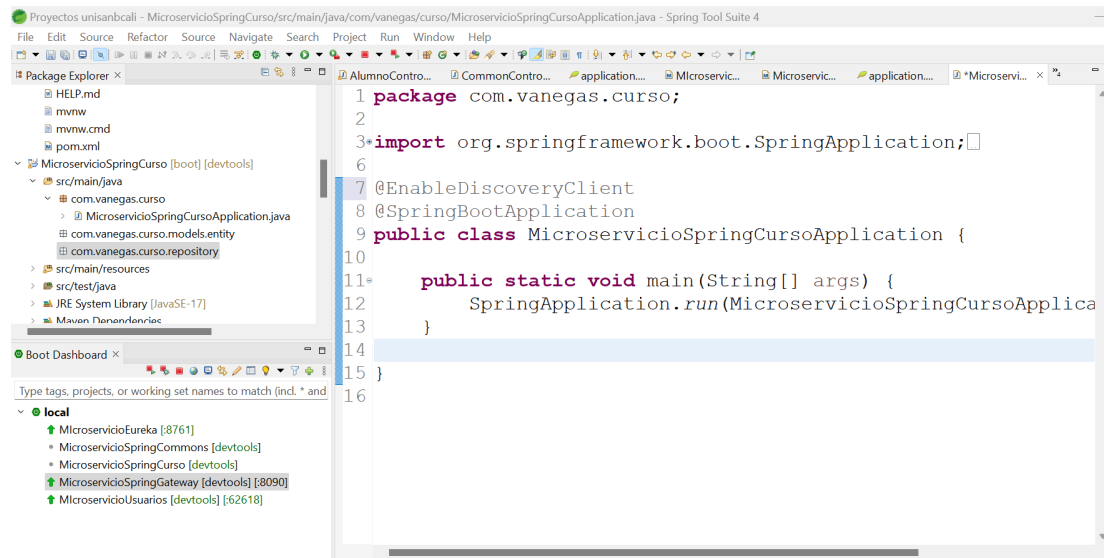
```
<version>0.0.1-SNAPSHOT</version>
```

```
</dependency>
```

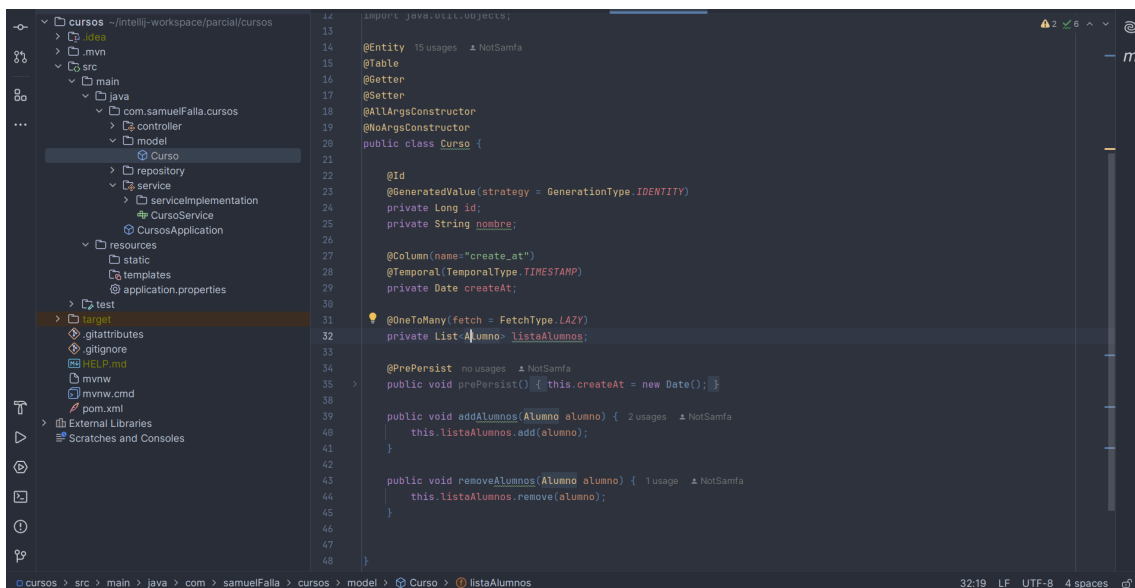
2. Luego copiamos todo el contenido del archivo de configuraciones del servicio Usuarios en este nuevo servicio y allí solo cambiaremos el nombre del servicio todo lo demás se deja igual
3. En la clase principal colocamos el decorador

@EnableDiscoveryClient

4. Creamos el paquete models.entity y repository



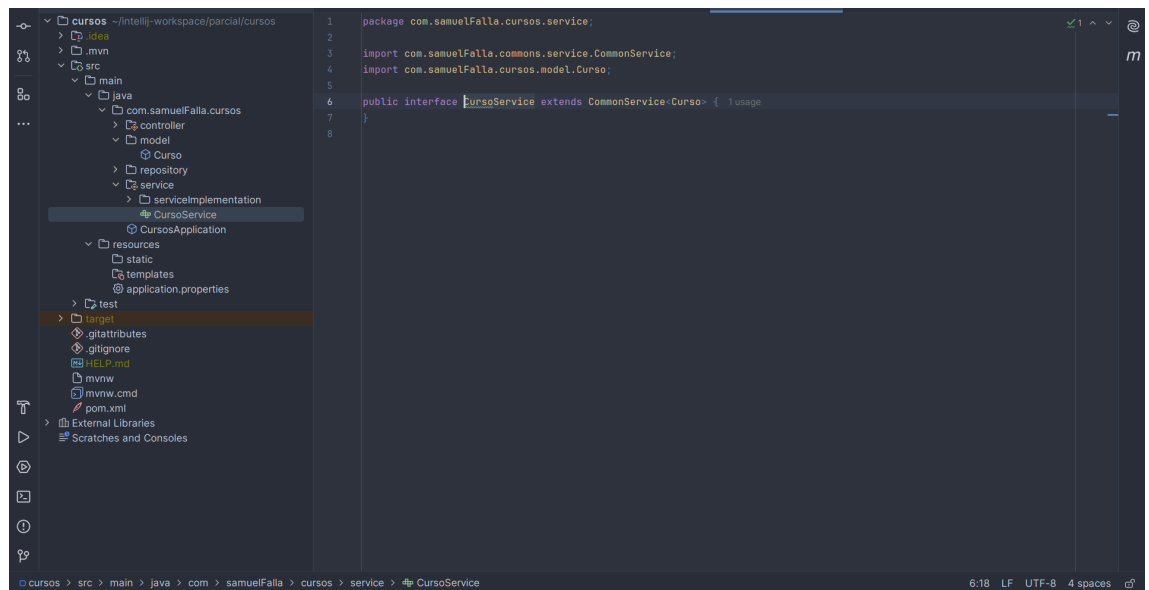
5. Creamos una clase POJO con el nombre de Curso dentro del paquete models.entity con la siguiente estructura:



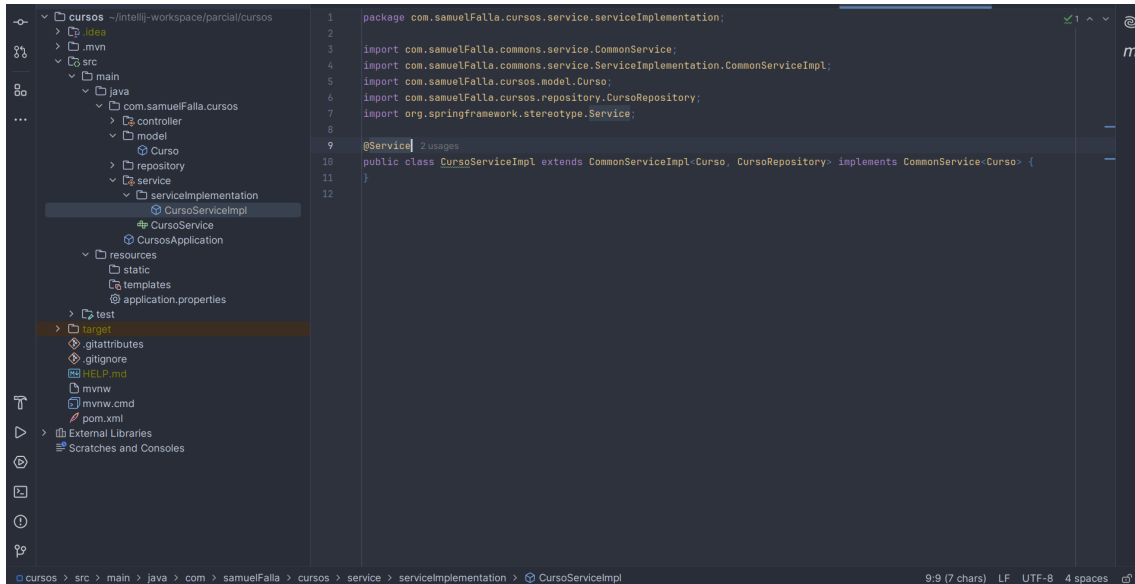
6. Creamos dentro del paquete repository la interface CursoRepository la cual debe heredar de CrudRepository y como parámetros de esa herencia debe estar la clase Curso y el tipo de dato Long.

7. Ahora creamos los paquetes controller y services

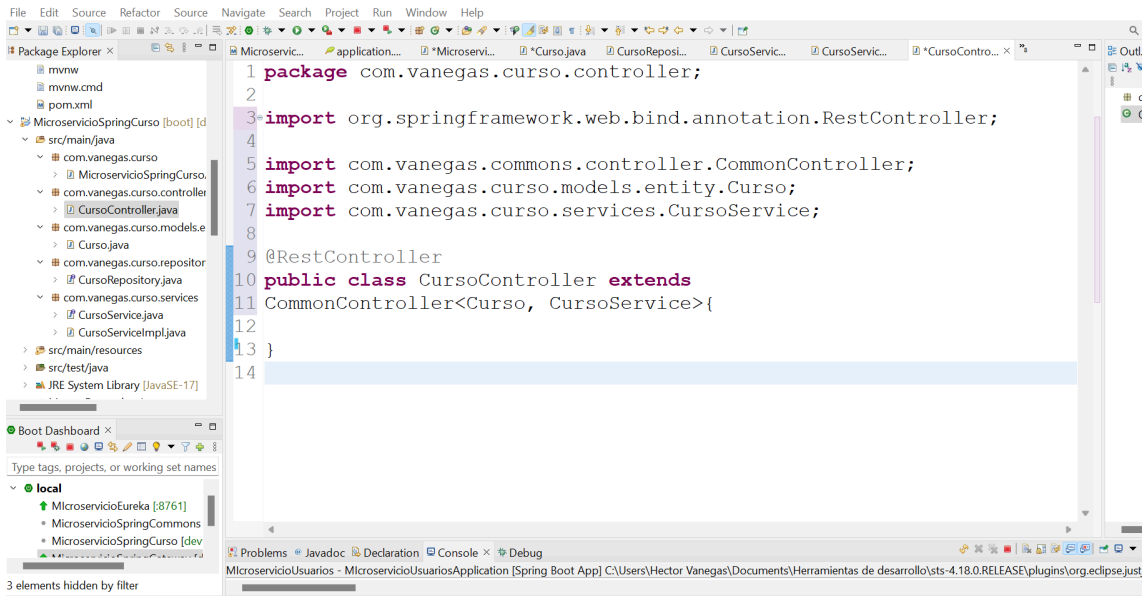
8. Dentro del paquete service se debe crear la clase CursoService la cual deberá heredar del genérico que se encuentra dentro del servicio common y decoramos la clase con la etiqueta @Service



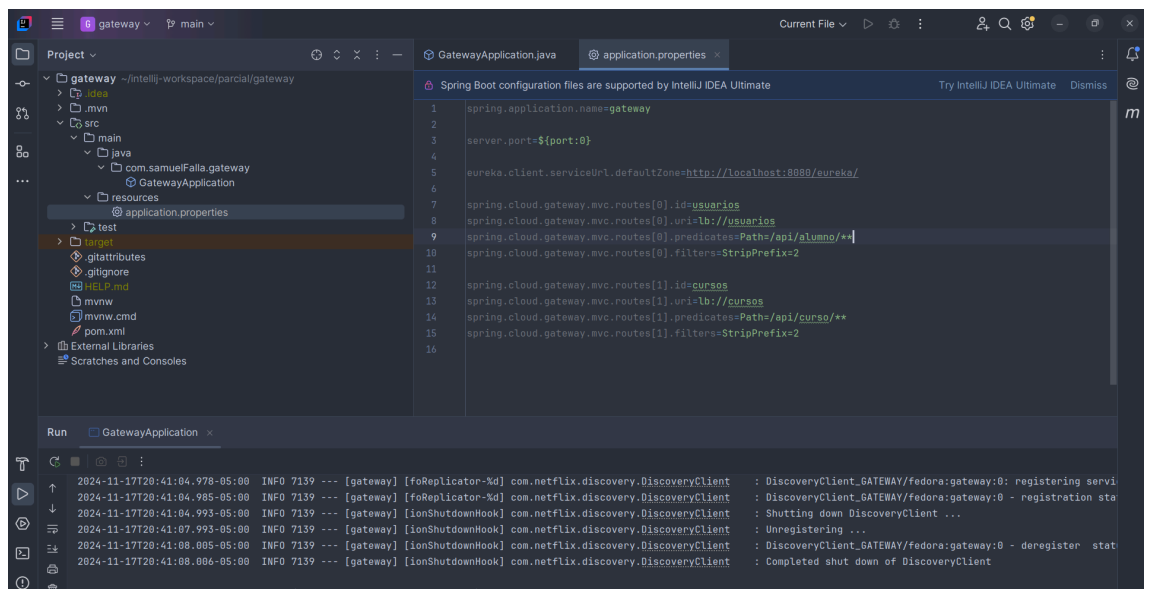
9. Ahora se debe crear la implementación de esa interface dentro del mismo paquete



10. quitamos los import
11. Ahora creamos el controller de la manera mas sencilla como esta

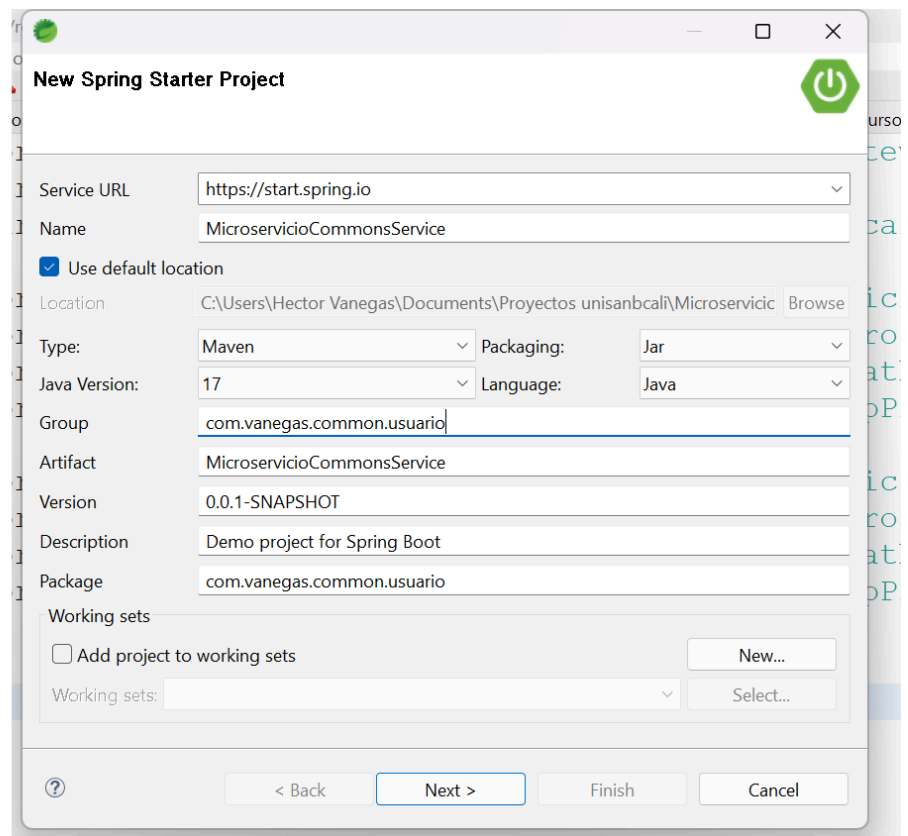


12. Copiamos el único método que quedó en el controlador del servicio Usuario , en este caso nos referimos al método Put y lo ajustamos a este servicio, ya que este es el único método que no se va a reutilizar
13. Ahora inscribimos el servicio Curso dentro del servicio Gateway colocando en el archivo de propiedades de este ultimo las siguientes líneas de código



5. Crear el servicio Common para los entities

1. Ahora se debe crear otro proyecto que tiene como propósito crear otra librería que permita reusar los componentes entity en todos los servicios



New Spring Starter Project

Service URL:

Name:

☒ Use default location

Location:

Type: Packaging:

Java Version: Language:

Group:

Artifact:

Version:

Description:

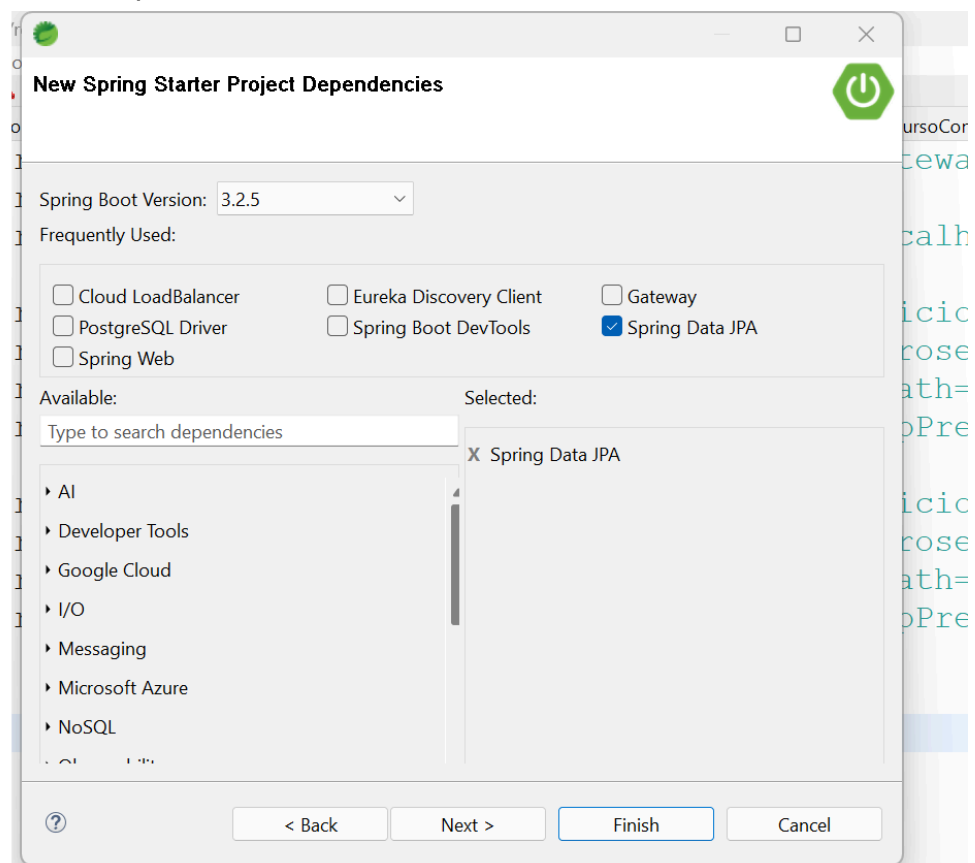
Package:

Working sets

☐ Add project to working sets

Working sets:

Con las siguientes dependencias



New Spring Starter Project Dependencies

Spring Boot Version:

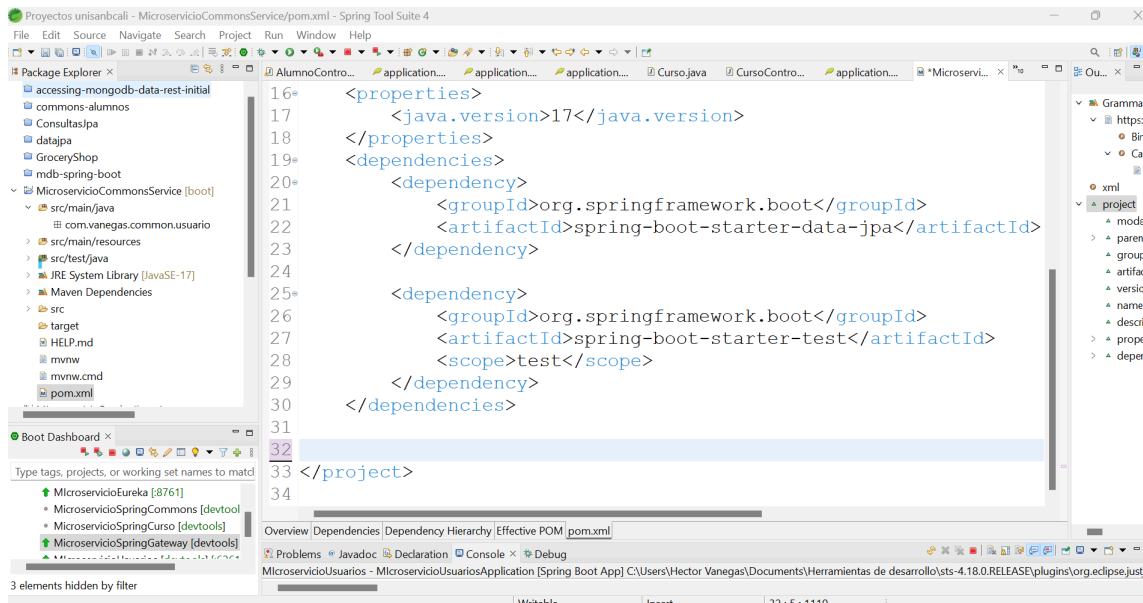
Frequently Used:

<input type="checkbox"/> Cloud LoadBalancer	<input type="checkbox"/> Eureka Discovery Client	<input type="checkbox"/> Gateway
<input type="checkbox"/> PostgreSQL Driver	<input type="checkbox"/> Spring Boot DevTools	<input checked="" type="checkbox"/> Spring Data JPA
<input type="checkbox"/> Spring Web		

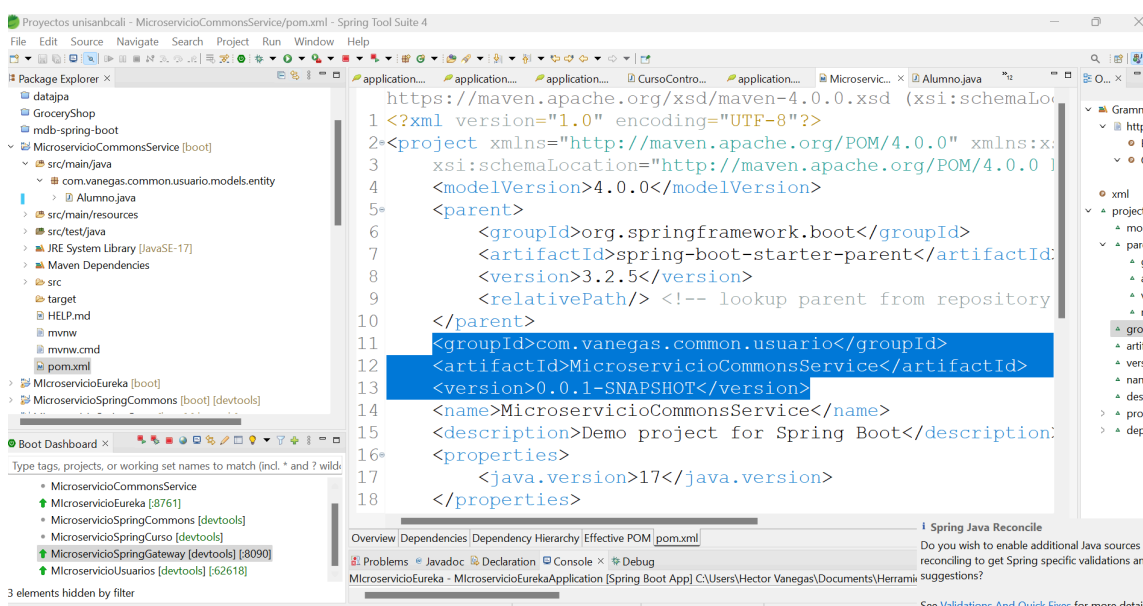
Available:

Selected: ☒ Spring Data JPA

2. Como es una librería debemos quitar la clase principal y en el pom eliminamos el plugins de Maven



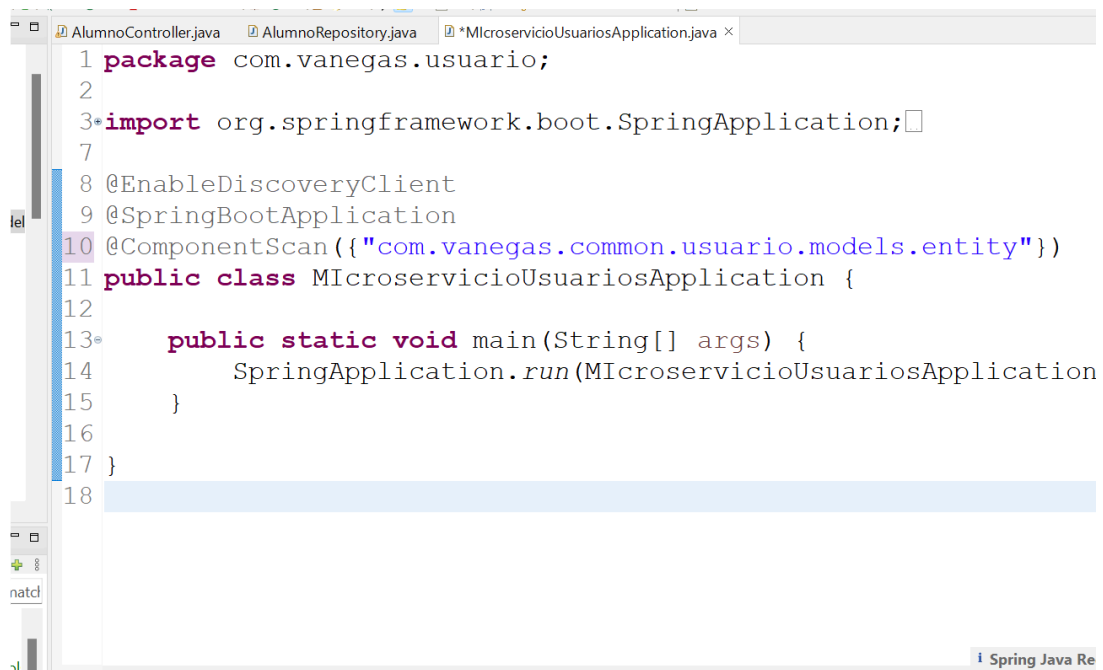
3. creamos el paquete models.entity y dentro de ese paquete movemos la clase Alumno del servicio usuarios y eliminamos el paquete donde se encontraba la clase dentro del servicio Usuarios. Es normal que saque errores en el servicio usuarios , en el siguiente paso lo solucionaremos
4. Copiamos las etiquetas groupId, artifactId y versión del servicio CommonService como dependencia en el servicio Usuarios



Y hacen un Maven update para que automáticamente se actualice las dependencias

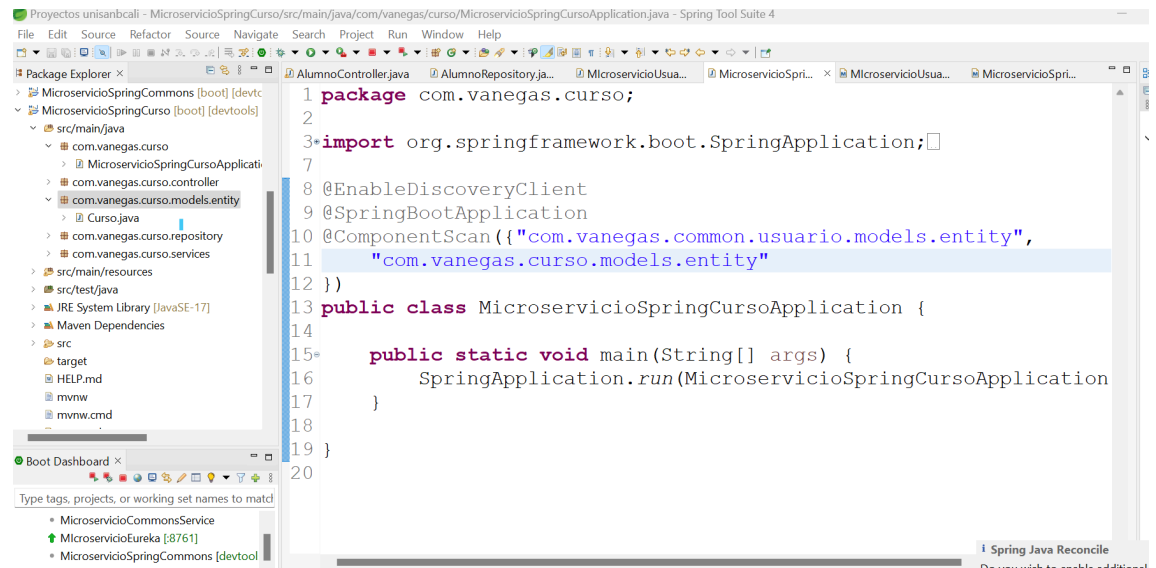
5. Para evitar inconvenientes de forma explícita le indicamos a spring boot que en el contexto incluya este nuevo componente para ello dentro de la clase principal del servicio Usuarios colocamos la siguiente anotación:

@ComponentScan para que esta mapea las clases que se encuentra dentro del paquete de entidades que creamos en la librería

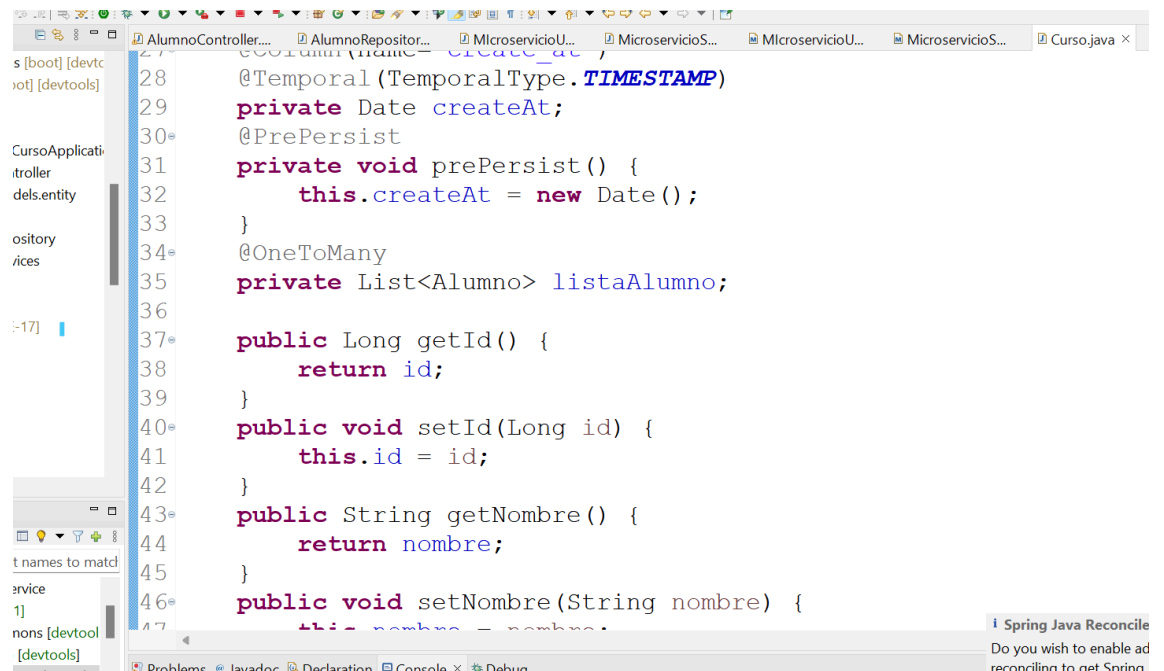


```
1 package com.vanegas.usuario;
2
3 import org.springframework.boot.SpringApplication;
4
5 @EnableDiscoveryClient
6 @SpringBootApplication
7 @ComponentScan({"com.vanegas.common.usuario.models.entity"})
8 public class MicroservicioUsuariosApplication {
9
10     public static void main(String[] args) {
11         SpringApplication.run(MicroservicioUsuariosApplication.class, args);
12     }
13 }
14
15
16
17
18
```

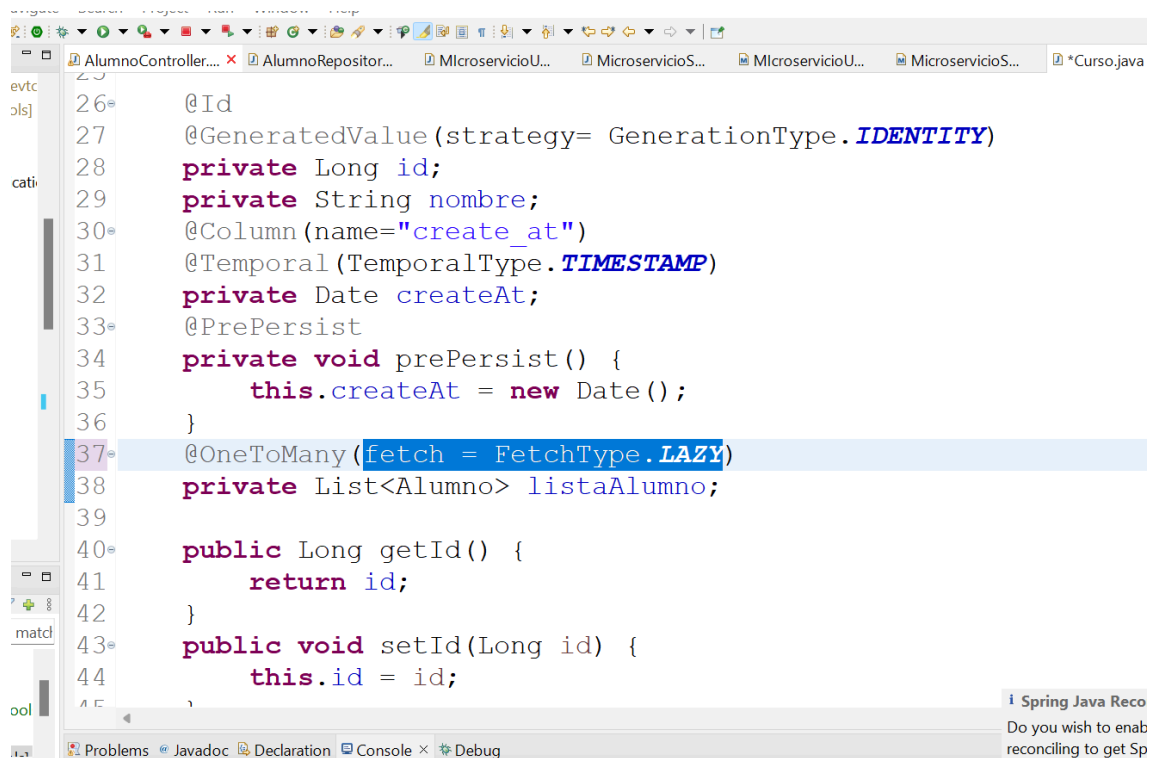
6. Ahora ejecutamos los pasos 4 y 5 dentro del servicio Curso y como agregamos otro arreglo dentro del component scan



7. Creamos relación entre Curso y Alumno para ello dentro del servicio Curso mas exactamente en el POJO o clase Curso agregamos un nuevo atributo en este caso sería una lista de tipo Alumno haciendo los import necesarios y agregamos los métodos getter y setter de este nuevo atributo y justo encima de este nuevo atributo colocamos el decorador que permite establecer la relación entre estos



8. Para que se haga mejor la relación colocamos la siguiente anotación `fetch = FetchType.LAZY`



```
26 @Id
27 @GeneratedValue(strategy= GenerationType.IDENTITY)
28 private Long id;
29 private String nombre;
30 @Column(name="create_at")
31 @Temporal(TemporalType.TIMESTAMP)
32 private Date createAt;
33 @PrePersist
34 private void prePersist() {
35     this.createAt = new Date();
36 }
37 @OneToMany(fetch = FetchType.LAZY)
38 private List<Alumno> listaAlumno;
39
40 public Long getId() {
41     return id;
42 }
43 public void setId(Long id) {
44     this.id = id;
45 }
```

9. En este punto investigar y colocar por que se usa ese argumento dentro de esa relación (fetch = FetchType.LAZY)

(fetch = FetchType.LAZY): Indica a Hibernate solo realizar fetch de las entidades relacionadas cuando sea haga uso de resta por primera vez o bajo demanda..

Referencia:

<https://thorben-janssen.com/entity-mappings-introduction-jpa-fetchtypes/>

10. Ahora vamos a construir un método dentro de esta clase el cual permitirá agregar alumnos a la listas que acabamos de crear

```

54     }
55     public void setCreateAt(Date createAt) {
56         this.createAt = createAt;
57     }
58     public List<Alumno> getListAlumno() {
59         return listaAlumno;
60     }
61     public void setListaAlumno(List<Alumno> listaAlumno) {
62         this.listaAlumno = listaAlumno;
63     }
64
65     public void setAlumnos(Alumno alumno) {
66         this.listaAlumno.add(alumno);
67     }
68
69
70
71
72 }
73

```

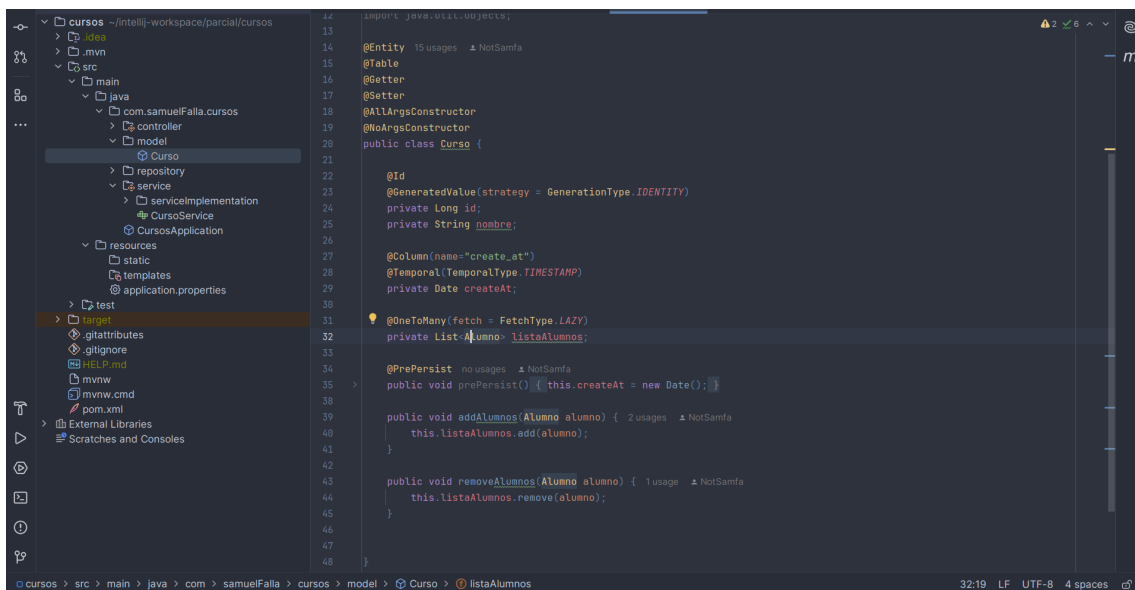
11. Hasta aquí esa lista es null, por tal motivo debemos inicializar dentro del constructor

```

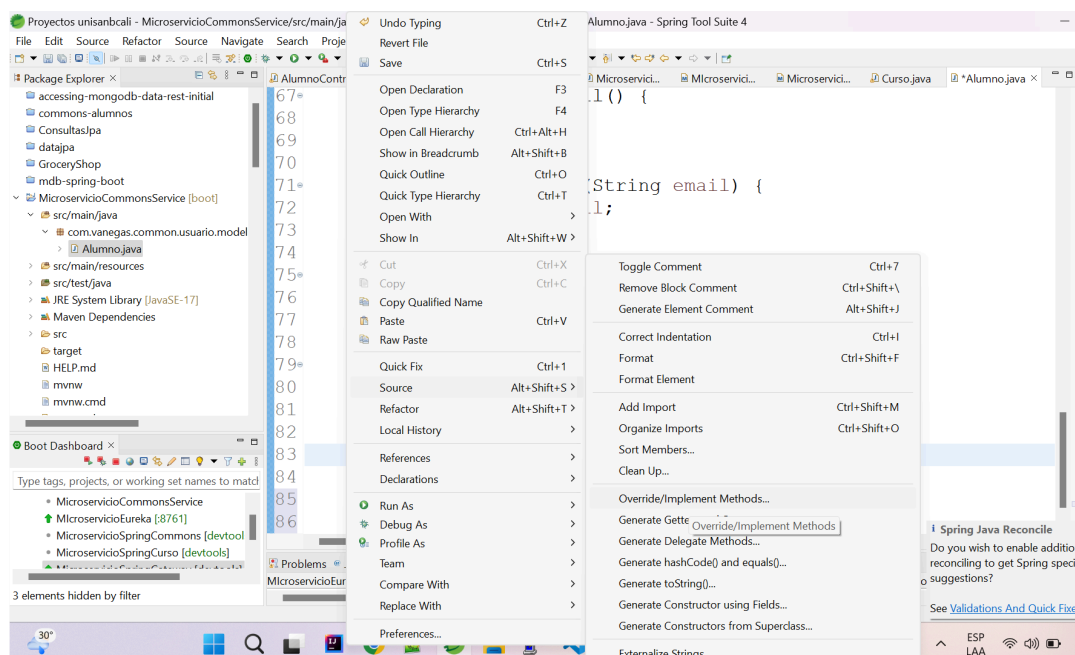
36         this.createAt = new Date();
37     }
38     @OneToMany(fetch = FetchType.LAZY)
39     private List<Alumno> listaAlumno;
40
41
42
43     public Curso() {
44         this.listaAlumno = new ArrayList<>();
45     }
46     public Long getId() {
47         return id;
48     }
49     public void setId(Long id) {
50         this.id = id;
51     }

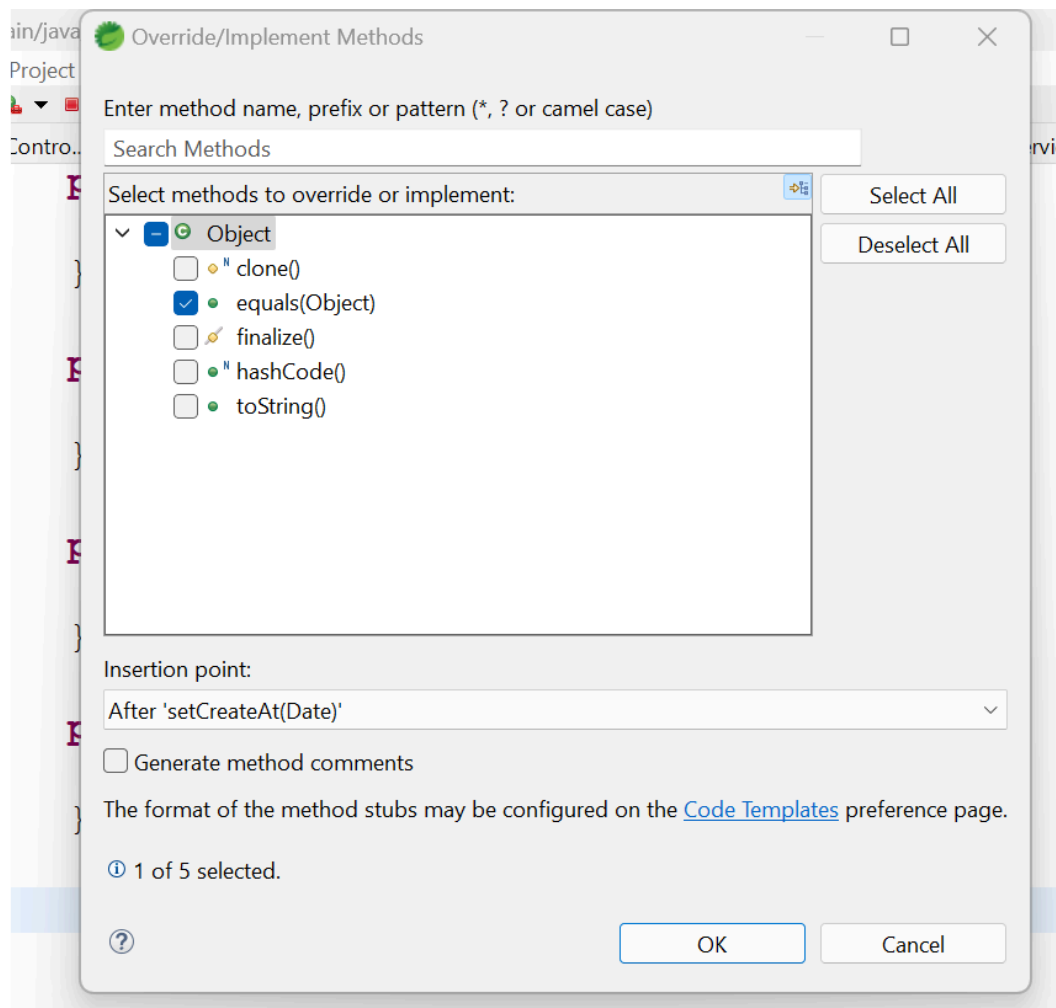
```

12. Creamos otro método que permita eliminar un alumno, este es poco mas completo, primero en la clase Curso creamos el método de la siguiente forma



13. Pero como les dije no es sencillo así como esta no funciona , así que toca ir a la clase Alumno del servicio Common y garantizar esa eliminación a través de un método previa validación con el siguiente fragmento de código :

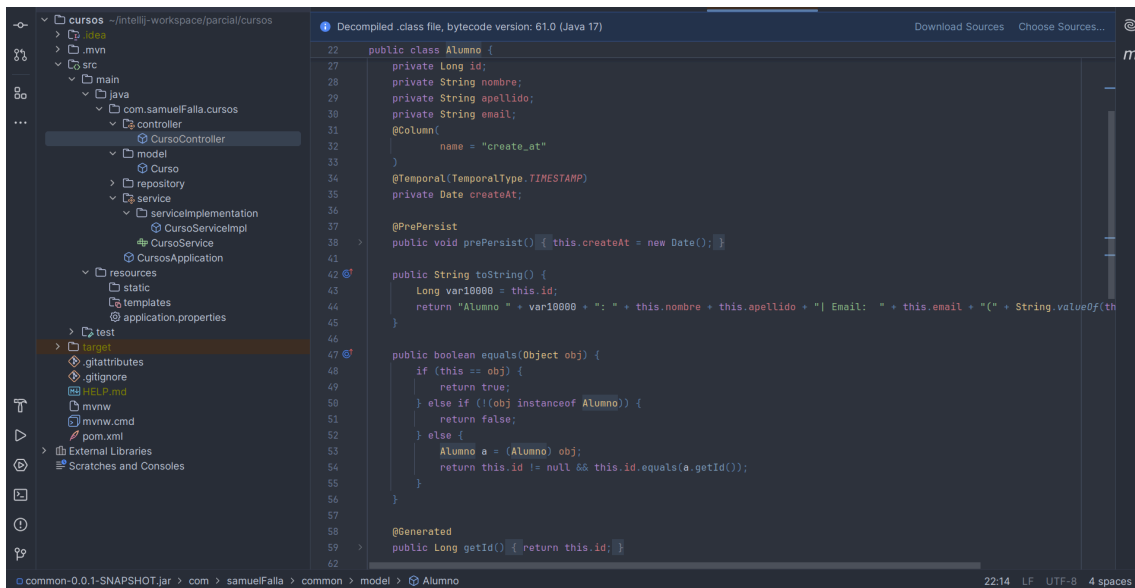




```
74  
75 public Date getCreateAt() {  
76     return createAt;  
77 }  
78  
79 public void setCreateAt(Date createAt) {  
80     this.createAt = createAt;  
81 }  
82  
83  
84 @Override  
85 public boolean equals(Object obj) {  
86     // TODO Auto-generated method stub  
87     return super.equals(obj);  
88 }  
89  
90  
91  
92  
93
```

Spring Java Reconcili

En este método que acabamos de crear vamos a hacer una validación antes de eliminar el Alumno del Curso



6. Eliminar y asignar Alumnos al Curso

1. En el controlador del servicio Curso agregamos los siguientes métodos

Este método para asignar un alumno al curso

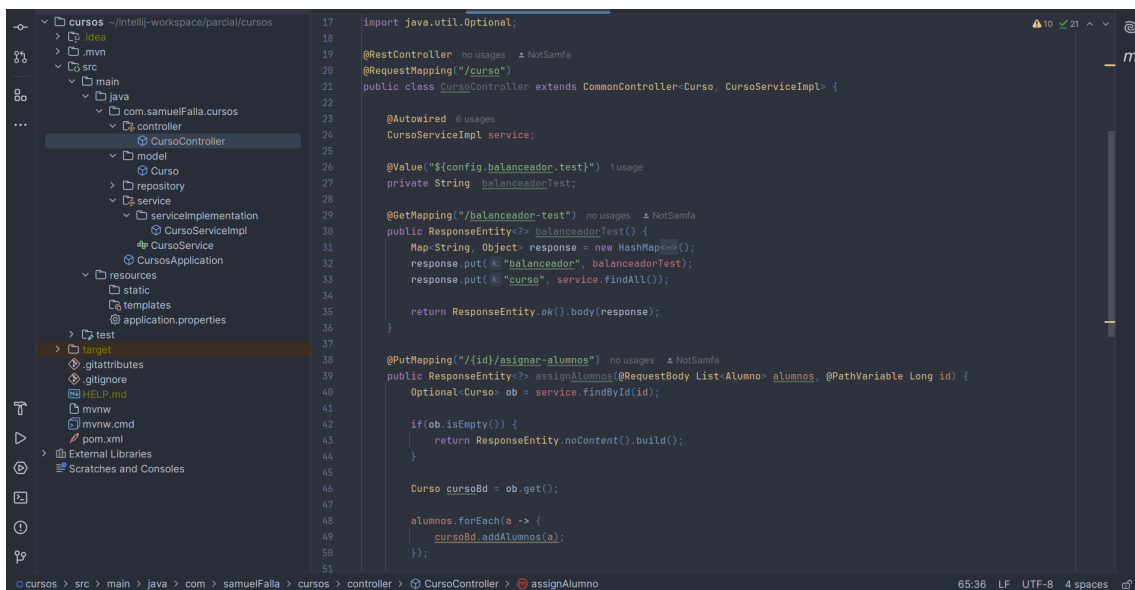
```
@PostMapping("/{id}/asignar-alumno")
public ResponseEntity<?> asignarAlumno(@RequestBody List<Alumno> alumno,
    @PathVariable Long id) {

    Optional<Curso> ob = service.findById(id);

    if (ob.isEmpty()) {
        return ResponseEntity.noContent().build();
    }
    Curso cursoBd = ob.get();
    alumno.forEach(a -> {
        cursoBd.addAlumnos(a);
    });

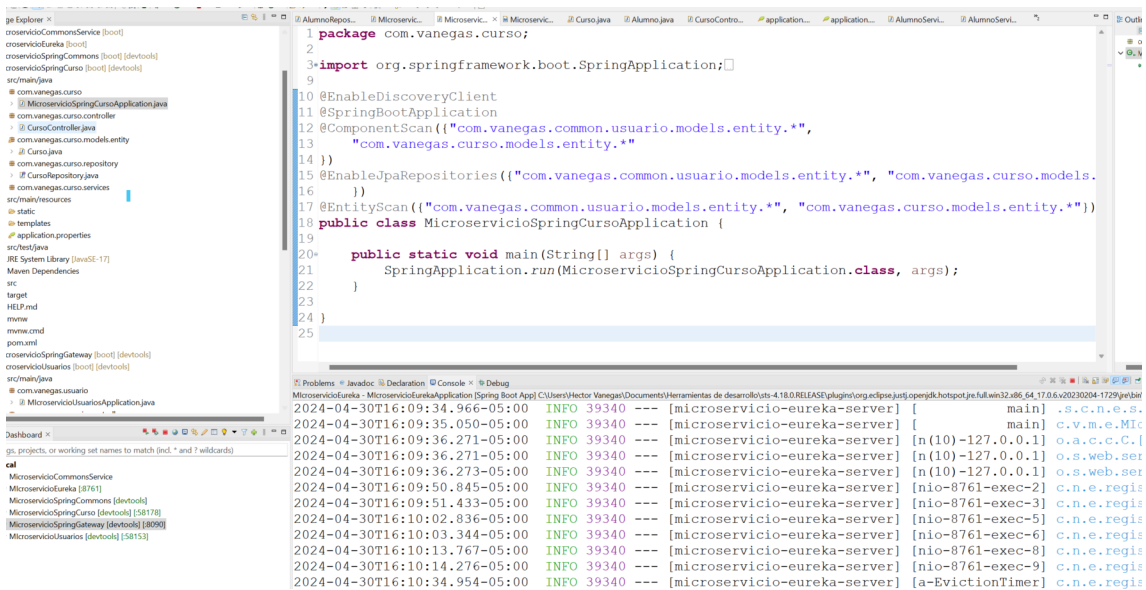
    return ResponseEntity.status(HttpStatus.CREATED).body(service.save(cur
}
```

Y este método sirve para eliminar un alumno del curso

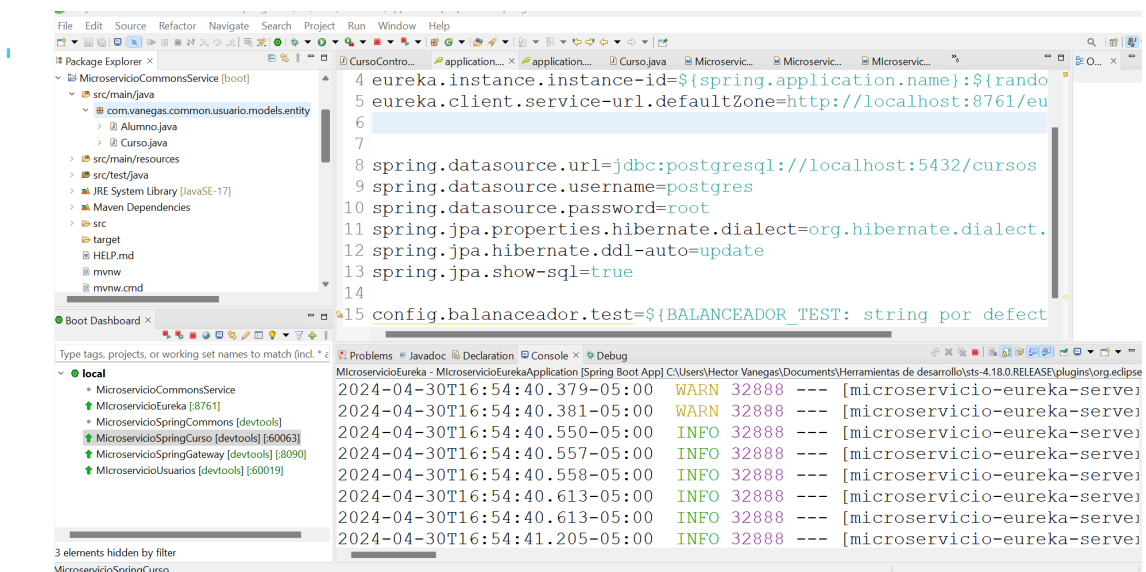


Si se tiene algún problema a la hora de levantar los servicios de Usuarios y Curso ,se debe agregar estas anotaciones a nivel de la clase principal

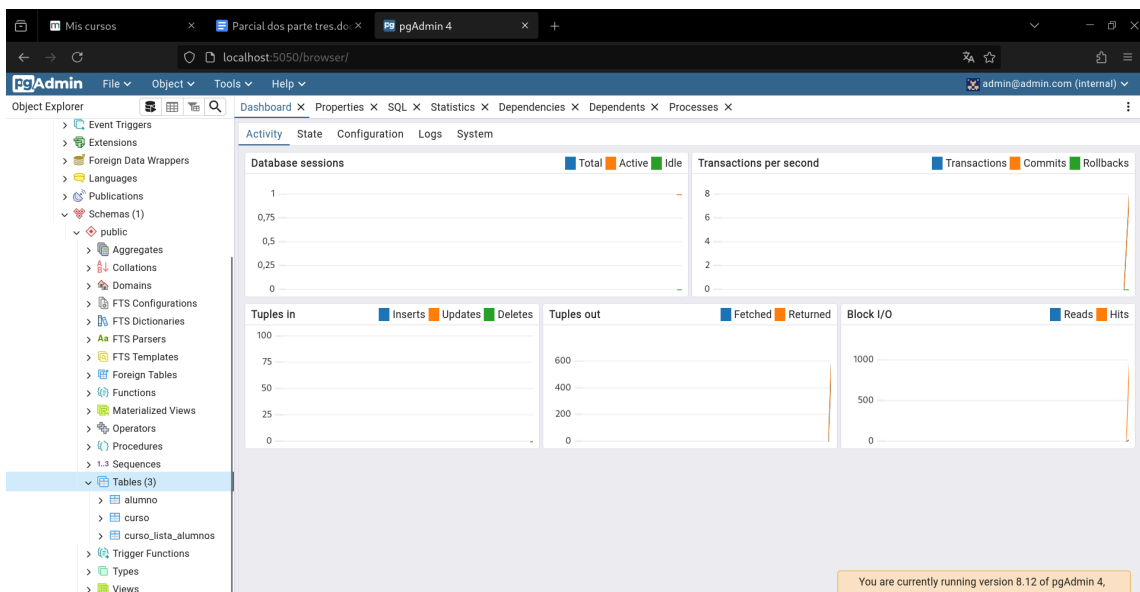
```
@EnableDiscoveryClient
@SpringBootApplication
@ComponentScan({"com.vanegas.common.usuario.
models.entity.*",
    "com.vanegas.curso.models.entity.*"
})
@EnableJpaRepositories({"com.vanegas.common.
usuario.models.entity.*",
    "com.vanegas.curso.models.entity.*"
})
@EntityScan({"com.vanegas.common.usuario.mod
els.entity.*",
    "com.vanegas.curso.models.entity.*"})
```



Si tenemos problemas al crear la tabla de cursos procedemos a mover la Curso del microservicio Curso al servicio Common justo en el mismo paquete donde se encuentra la clase Alumno



La estructura de tablas que se crean el BD es



Probamos los servicios con postman