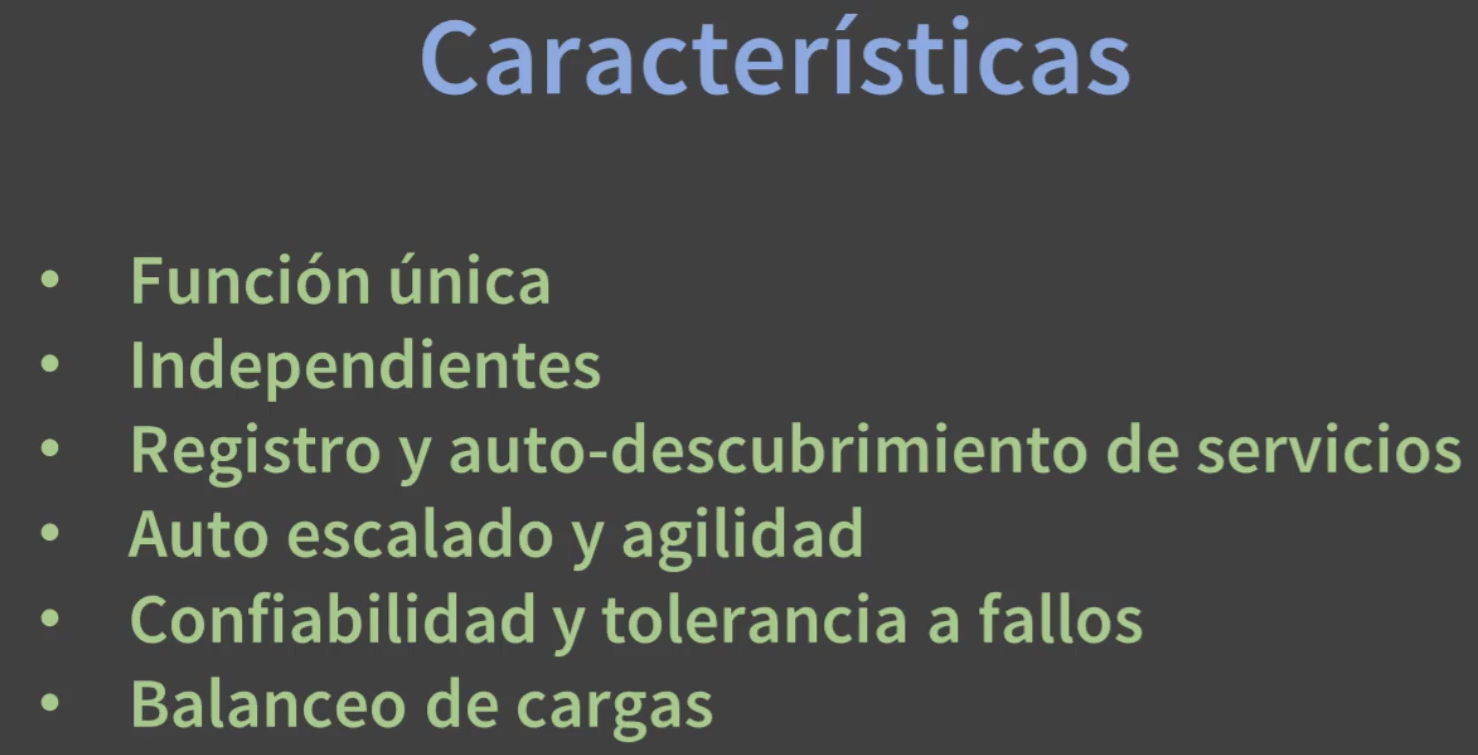
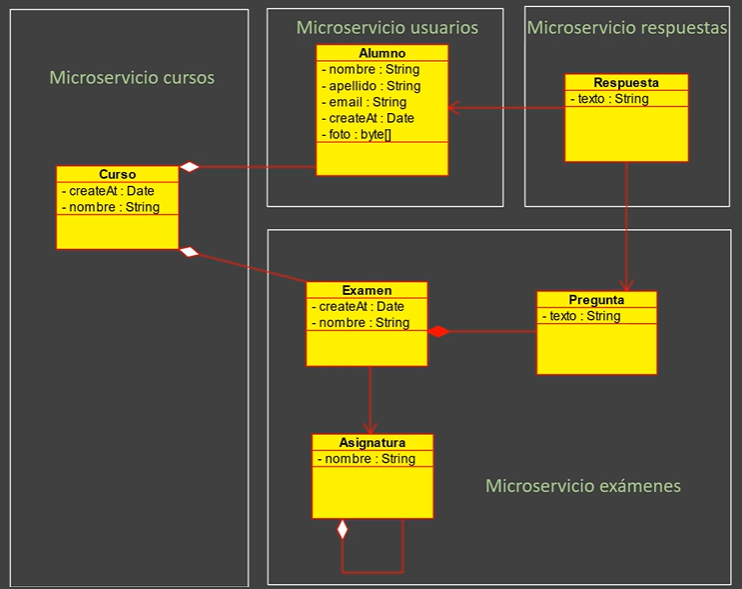


Servidor de nombre Eureka

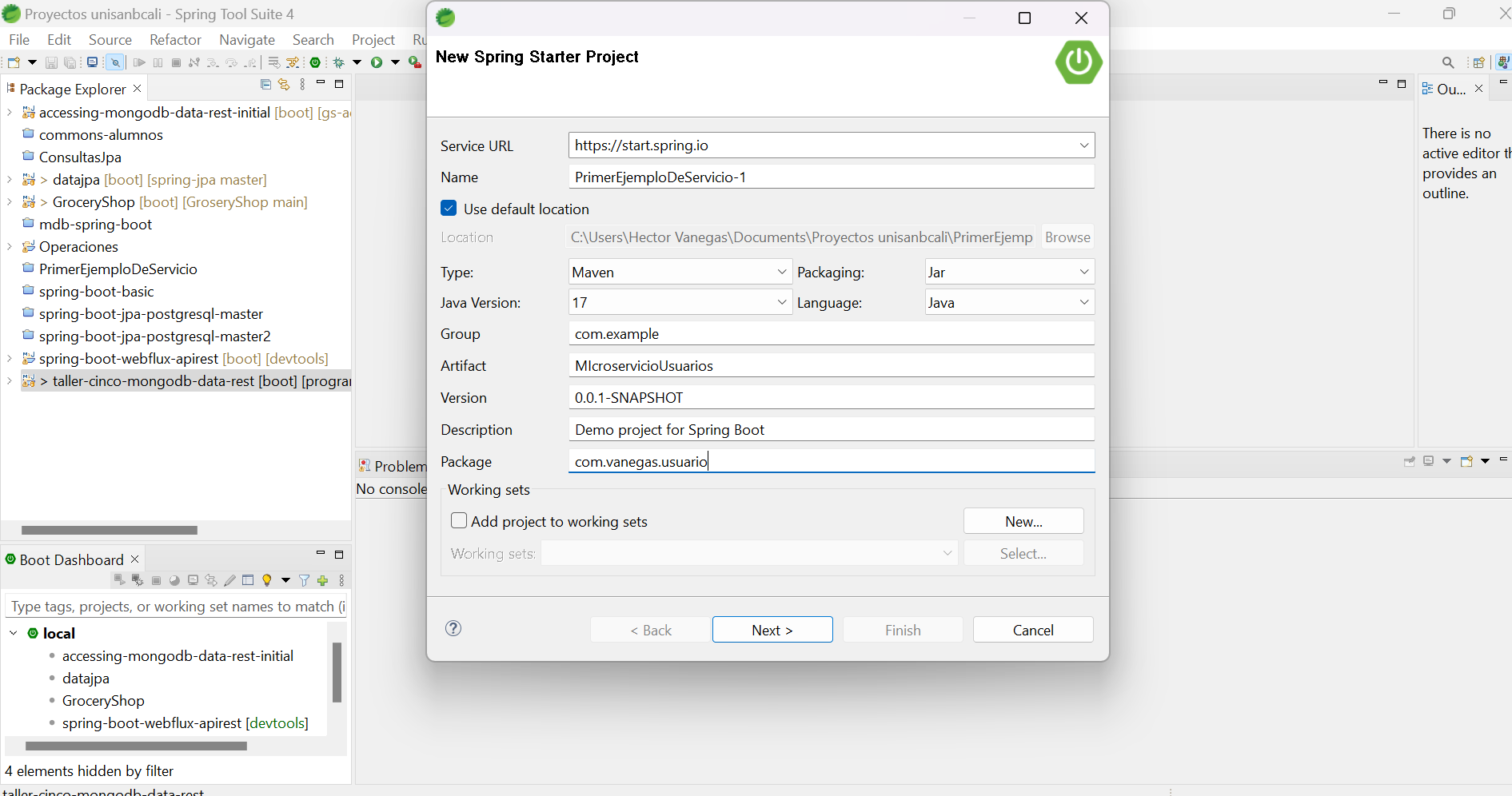


Llegaremos a construir este diagrama



Actividades a desarrollar:

1. **CREACIÓN DE SERVICIO USUARIOS**
2. Creamos el primer microservicio con el nombre de Usuarios y lo generamos a través de spring.io con las siguientes dependencias:



Spring web

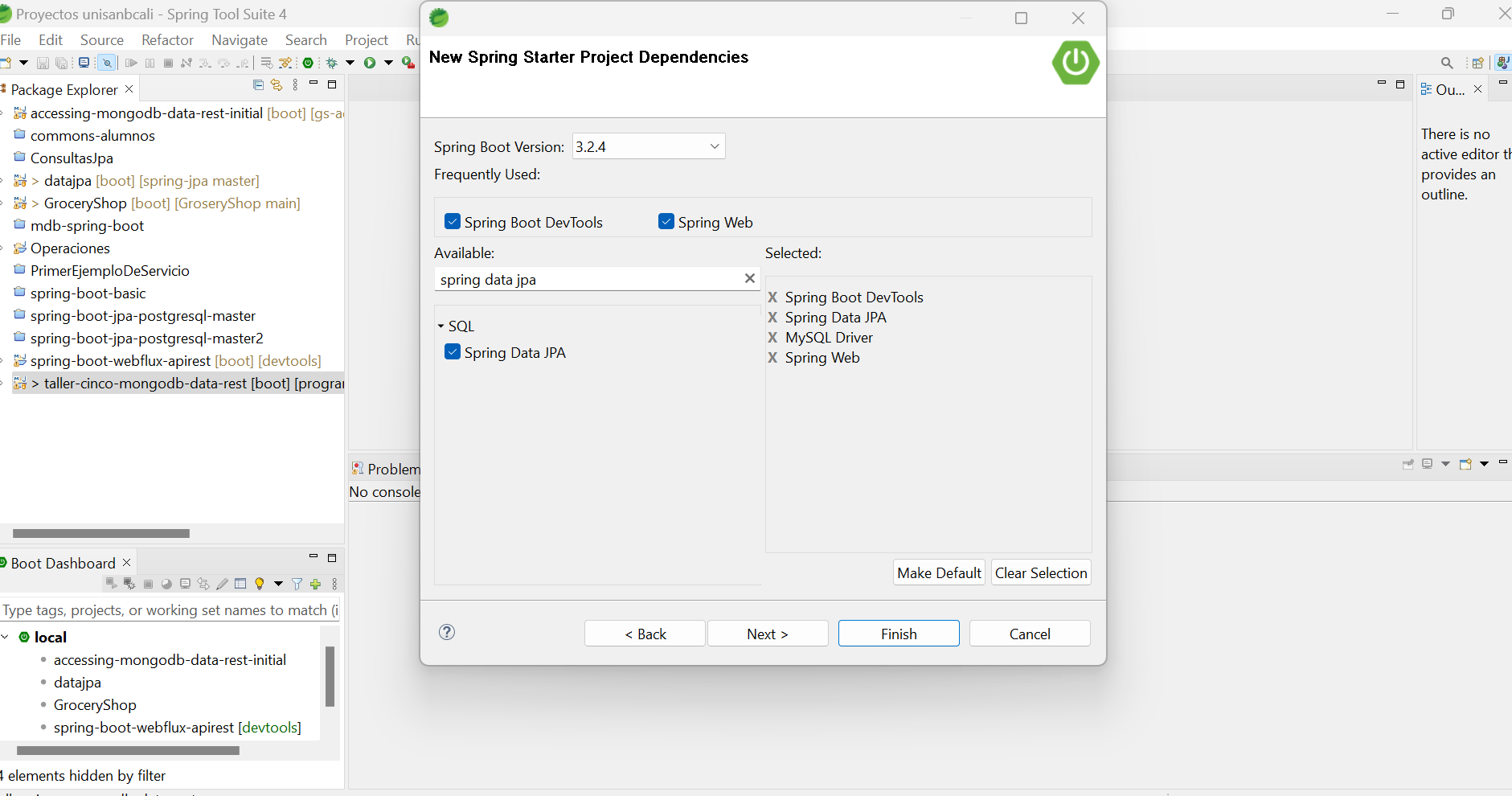
Spring Dev Tools

Spring Data Jpa

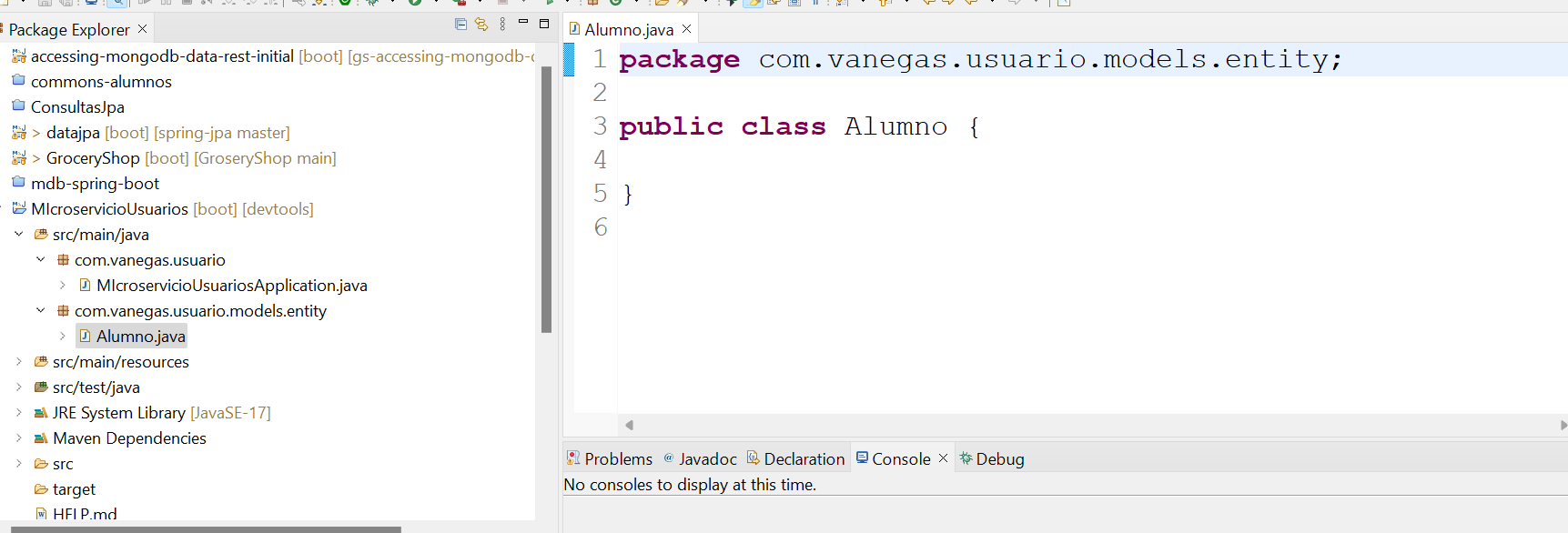
MySql Driver

Y que debe tener como paquete principal el siguiente nombre:

com.co.(su apellido).usuarios

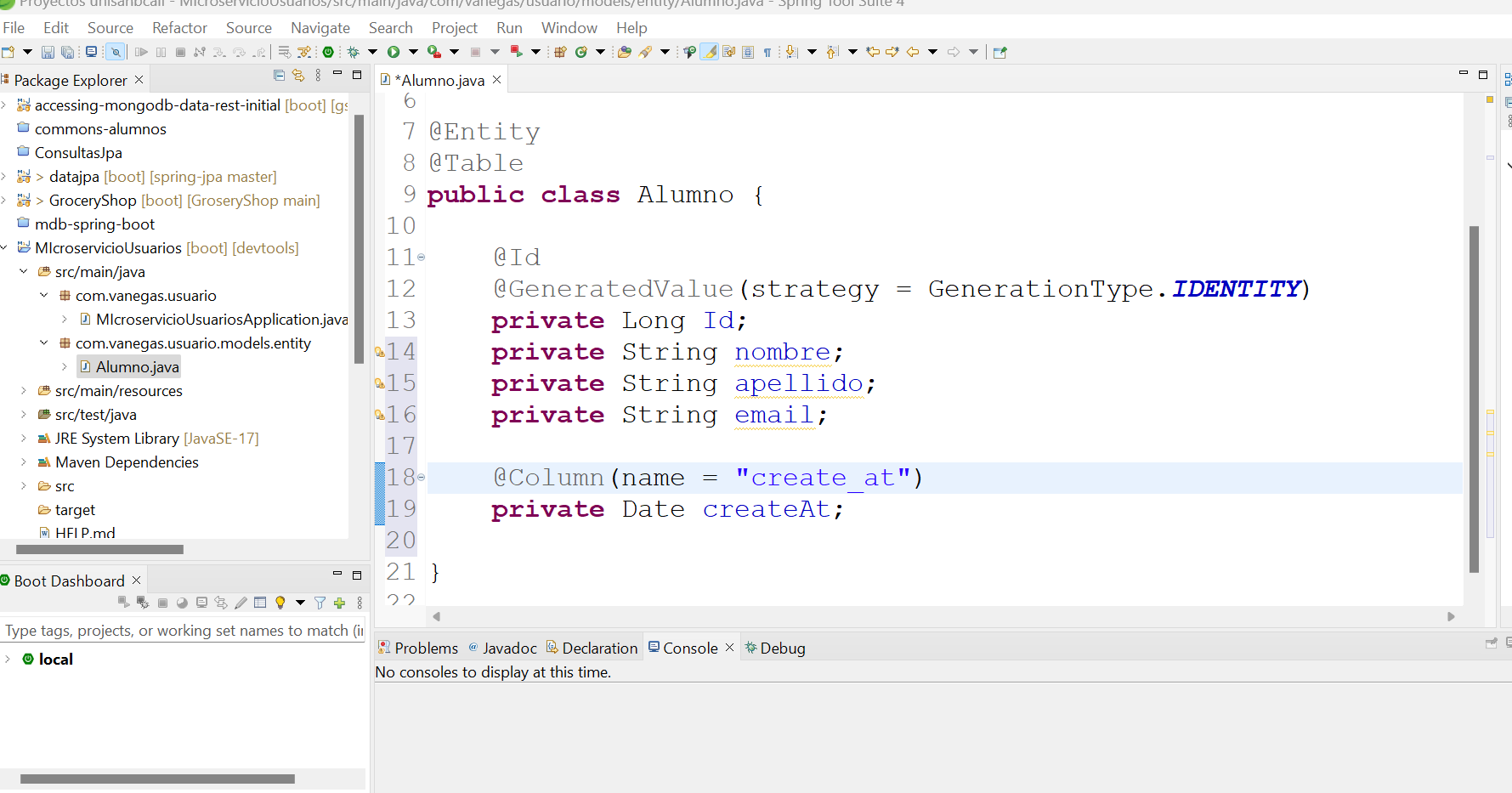


1. Dentro de este mismo documento deberá investigar y colocar el significado de cada una de las dependencias de spring descritas anteriormente
2. En el proyecto anterior crearemos el crud de la clase Alumno, para ello se creará primero un paquete a partir del paquete principal con el nombre de com.co.(su apellido).usuarios.entity.
3. Dentro del paquete anterior se debe construir una clase concreta con el nombre de Alumno.

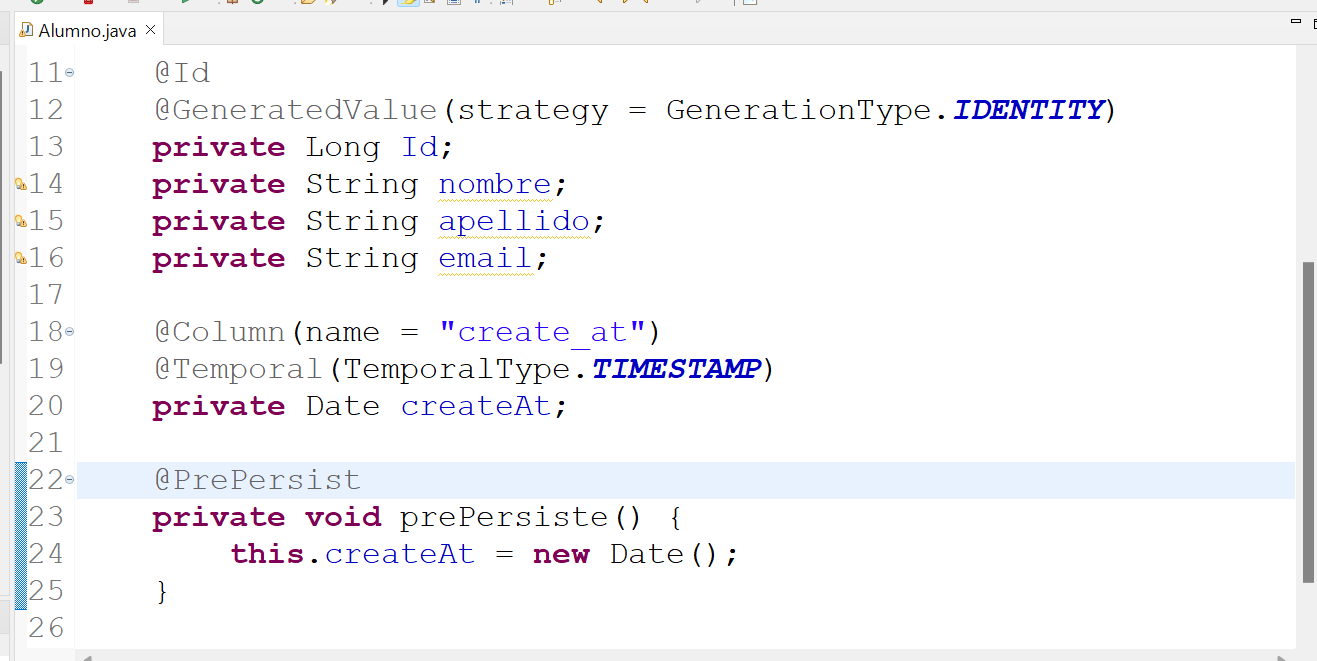


1. Colocaremos a nivel de clase los decoradores @Entity y @Table importándolo de la siguiente librería: jakarta.persistence
2. Agregamos los siguientes atributos de clase:

Id, nombre apellidos, email, fecha de creación tal como se ve en la siguiente imagen:



1. Debe crear un método que cada vez que guarde un nuevo alumno se almacene la fecha de creación de ese alumno de la siguiente forma:



1. Debe investigar y colocar en este documento el significado de cada uno de

los decoradores que se encuentra en esta clase.

R// @Id

@GeneratedValue

@Column

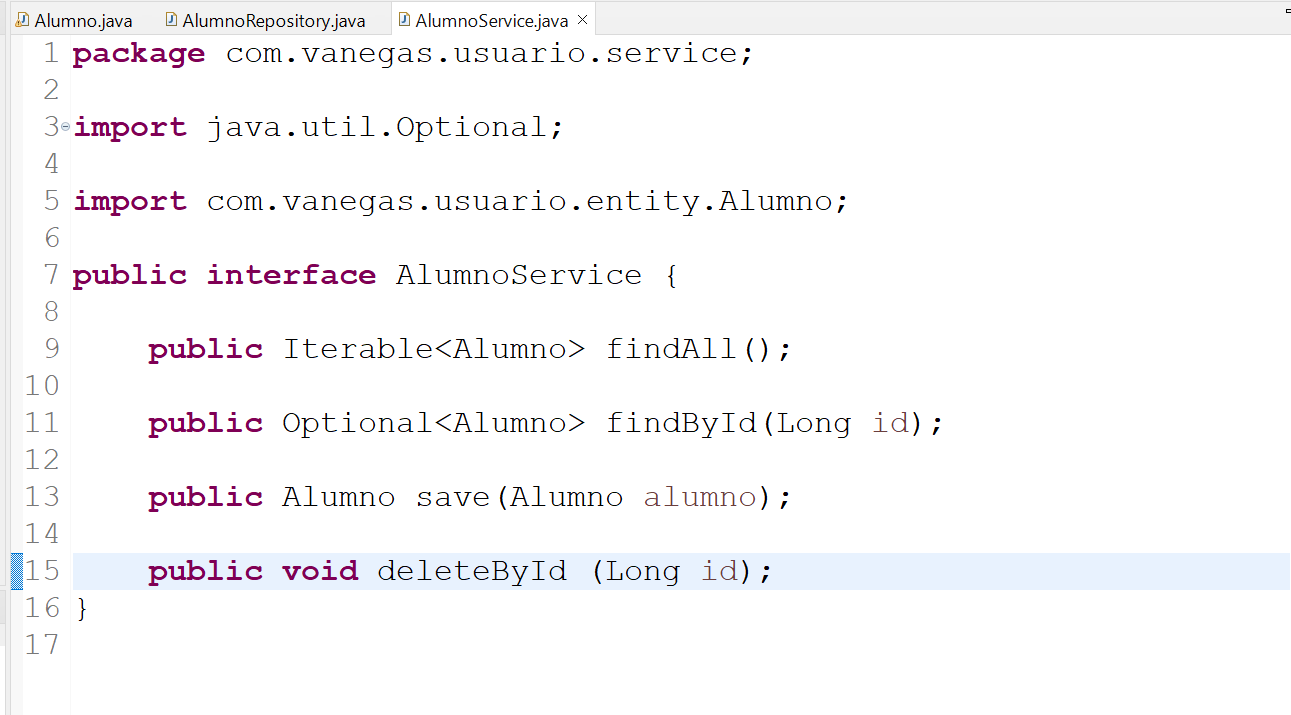
@Temporal

@PrePersist

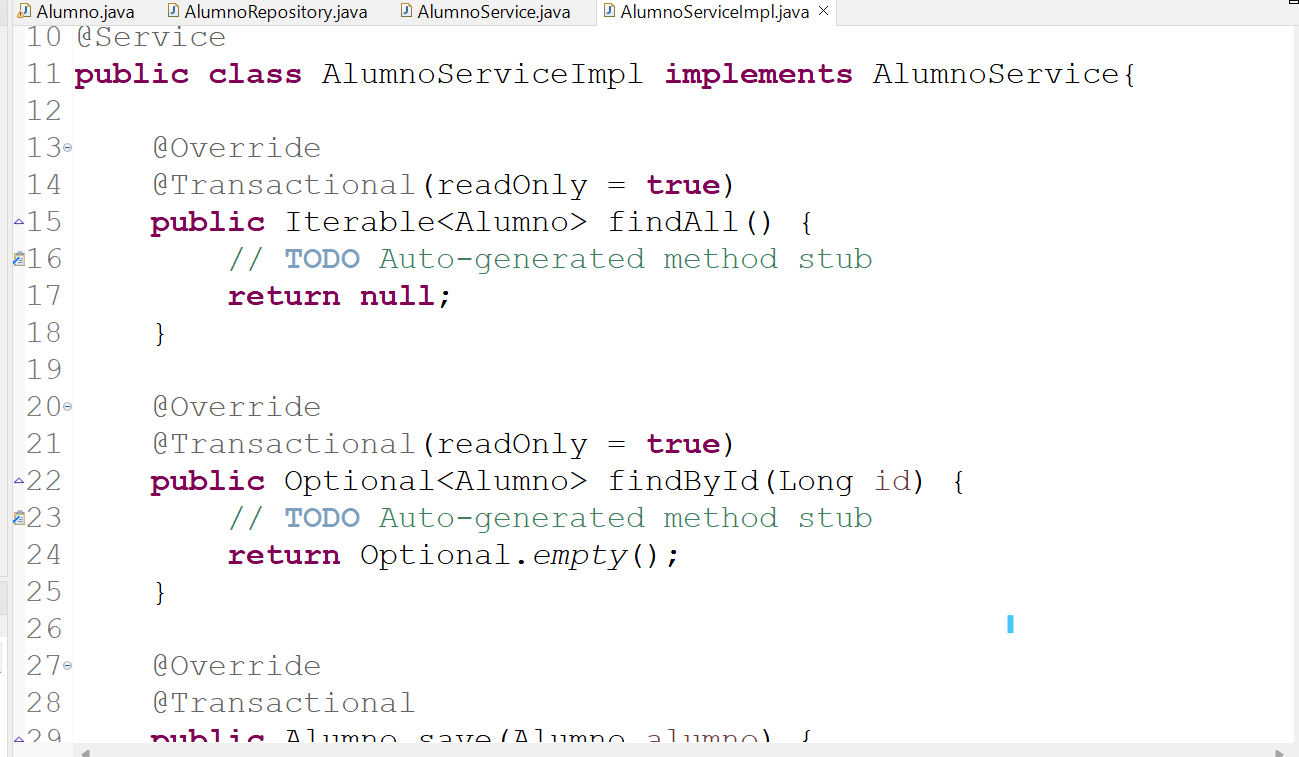
1. **@Id**: Este decorador se utiliza para marcar una propiedad de una clase como la clave primaria de una entidad en una base de datos. Es parte del estándar de mapeo objeto-relacional (ORM) y se utiliza comúnmente con JPA (Java Persistence API) para identificar la clave primaria de una entidad.
2. **@GeneratedValue**: Este decorador se utiliza en conjunto con el decorador @Id para especificar cómo se generará el valor de la clave primaria para una entidad. Por lo general, se utiliza para campos de clave primaria que son valores generados automáticamente por la base de datos, como identidades incrementales o secuencias. Puedes especificar diferentes estrategias de generación, como AUTO, IDENTITY, SEQUENCE, etc.
3. **@Column**: Este decorador se utiliza para especificar las propiedades de una columna de base de datos que se asignará a un campo de una entidad. Puedes especificar diferentes atributos de columna, como el nombre de la columna, la longitud máxima, si es nulo o no, etc.
4. **@Temporal**: Este decorador se utiliza para especificar cómo se debe mapear un campo de tipo de fecha y hora (por ejemplo, java.util.Date o java.time.LocalDate) a una columna de base de datos. Puedes utilizar este decorador para indicar si el campo representa solo una fecha, una hora o una marca de tiempo.
5. **@PrePersist**: Este decorador se utiliza para marcar un método en una clase de entidad que debe ejecutarse antes de que la entidad sea persistida en la base de datos. Puedes utilizar este método para realizar tareas de preparación o validación antes de que se guarde la entidad.

Estos decoradores son parte de la potente funcionalidad de Spring Boot para el desarrollo de aplicaciones Java, especialmente en el contexto de la persistencia de datos y el mapeo objeto-relacional.

1. Agregar los métodos getter and setter, además del constructor con todos los campos y el constructor por defecto vacío y sobreescribir el método ToString
2. Luego creamos un nuevo paquete con el nombre com.co.(su apellido).usuarios.repository dentro de éste creamos una interfaz con el nombre AlumnoRepository que extienda de CrudRepository
3. Luego creamos otro paquete con el nombre com.co.(su apellido).usuarios.service y dentro de este una interfaz con el nombre AlumnoService y dentro de ella los siguientes métodos abstractos:



1. Luego creamos dentro de ese mismo paquete una clase concreta que implemente la interfaz y por ende que implemente los métodos concretos de esa interfaz, esa clase debe estar con el decorador service, además a nivel de cada método colocaremos el decorador Transactional indicando si es no de lectura o escritura tal como se ve en la siguiente imagen:



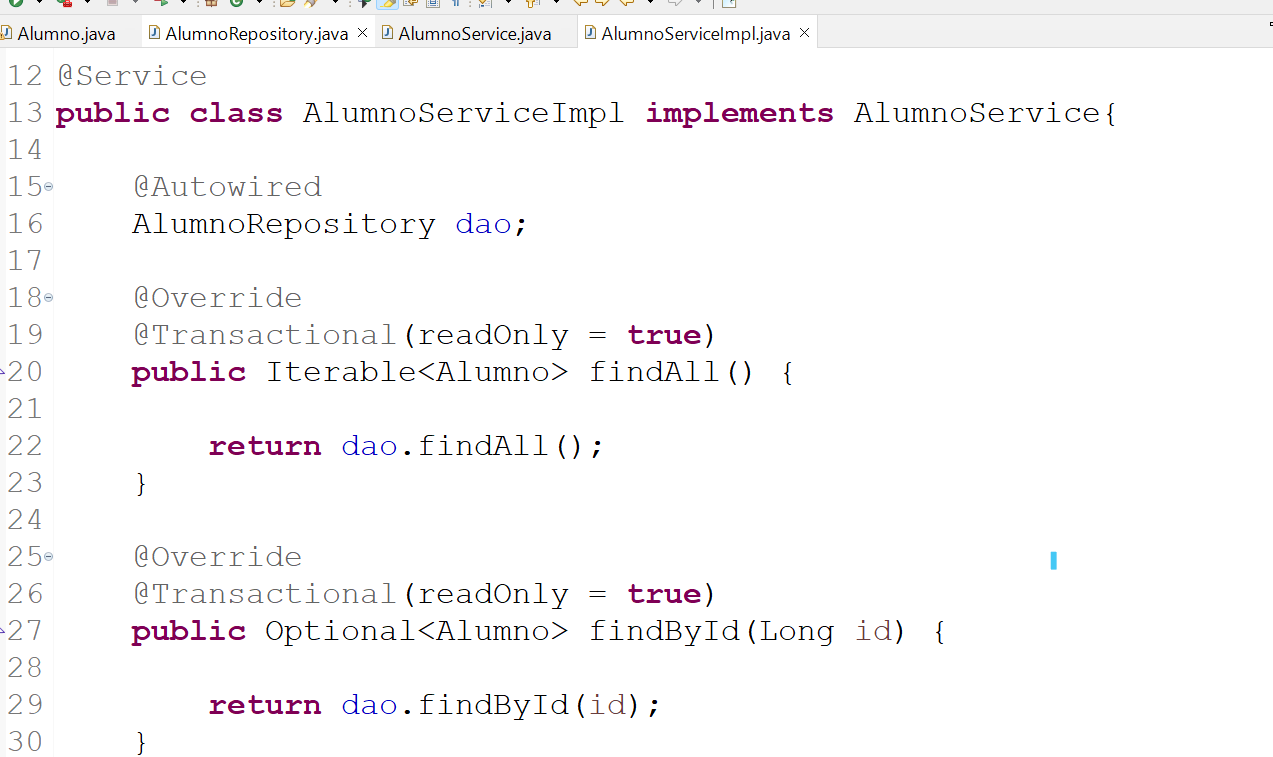
Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

1. Investigar y colocar dentro del este documento el concepto del decorador Transaction.

R// En el contexto de Java Spring Boot, el decorador **@Transactional** se utiliza para indicar que un método debe ejecutarse dentro de una transacción. Una transacción es una unidad de trabajo que debe ejecutarse de manera completa y coherente, lo que significa que todas las operaciones dentro de la transacción se realizan correctamente o ninguna se realiza en absoluto.

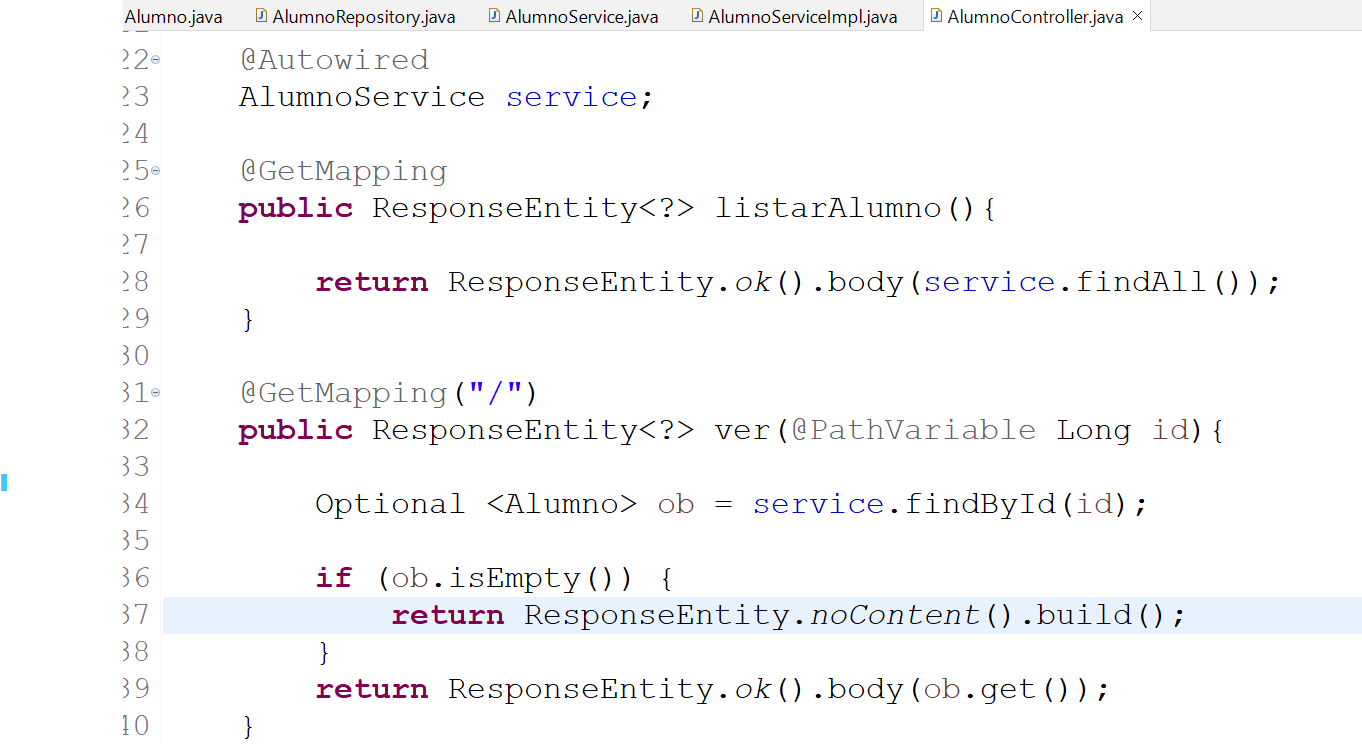
1. Inyectamos en esta clase la interface AlumnoRepository e implementamos cada uno de los métodos como se muestra en la siguiente imagen:



Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

1. Creamos ahora el paquete controller y dentro de él una clase concreta con el nombre AlumnoController y decorada a nivel de la clase con la etiqueta @RestController.
2. Dentro del controlador inyectamos la interface AlumnoService y allí implementamos los métodos que se encuentran en la siguiente imagen



1. Definir que significa los siguientes conceptos en Spring Boot :
2. ResponseEntity<?>
3. Optional
4. @PathVariable

Y los otros métodos son :

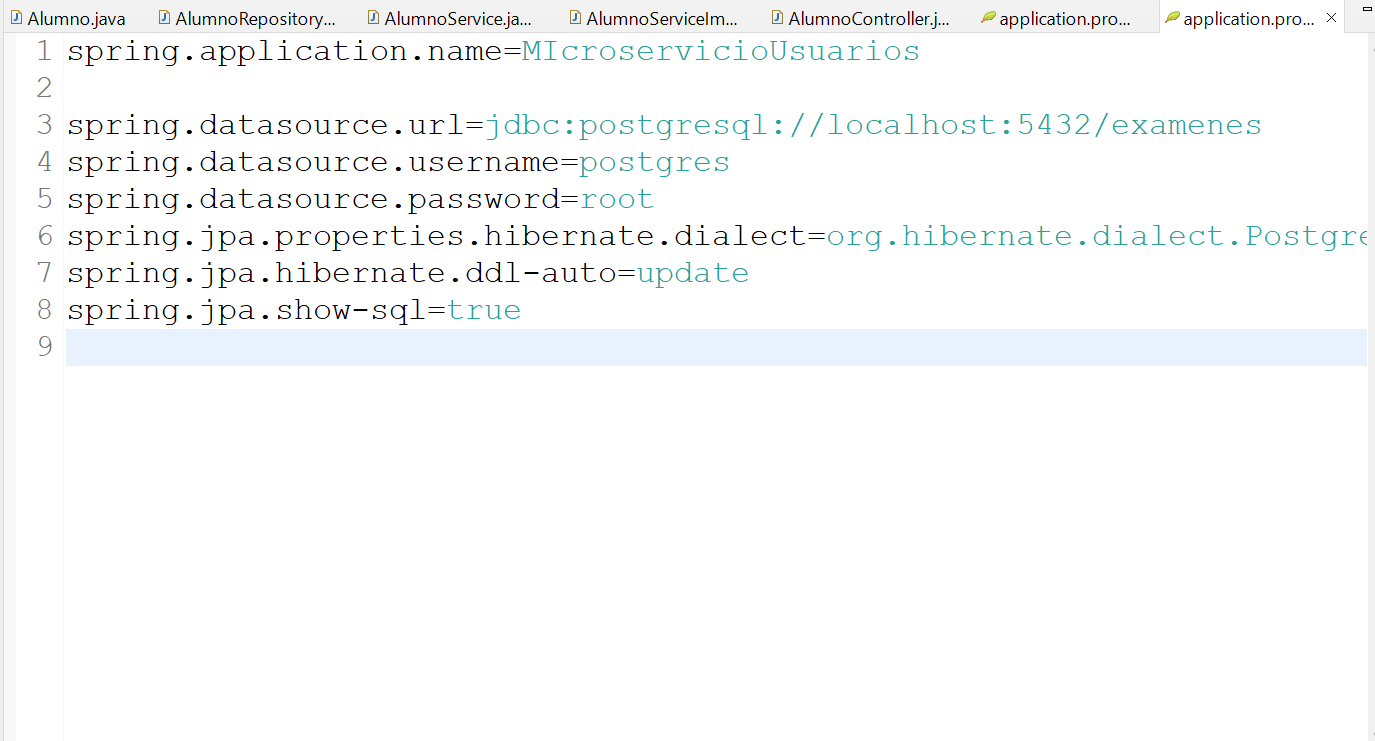
Imagen que contiene Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

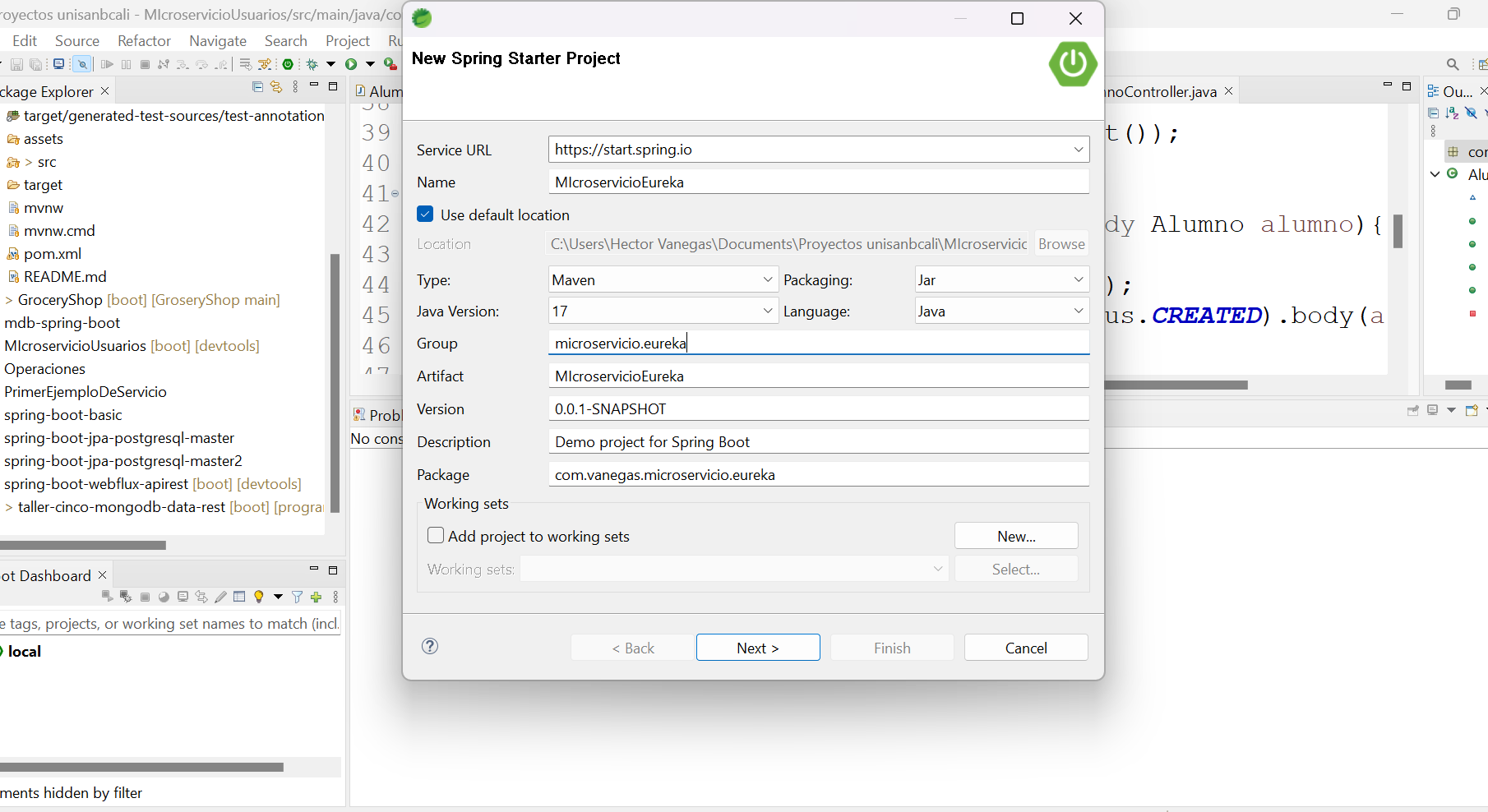
Descripción generada automáticamente

1. Crear el string de conexión a la bd de su preferencia, en mi caso es postgres y el string de conexión es el siguiente:



**CREACIÓN SERVIDOR EUREKA**

1. Investigar y colocar en este parte del documento lo que es el servidor EUREKA como parte de Spring Cloud
2. Se debe crea un nuevo proyecto con Spring.io tal y como hicimos el servicio anterior



Con las siguientes dependencias:

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

</dependency>

<dependency>

<groupId>org.glassfish.jaxb </groupId>

<artifactId>jaxb-runtime</artifactId>

</dependency>

1. Una vez creado el proyecto coloco el decorador de Eureka a nivel de la clase principal del proyecto como se muestra a continuación:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

1. En el archivo de propiedades colocamos la siguiente configuración:

Interfaz de usuario gráfica, Texto, Aplicación, Word

Descripción generada automáticamente

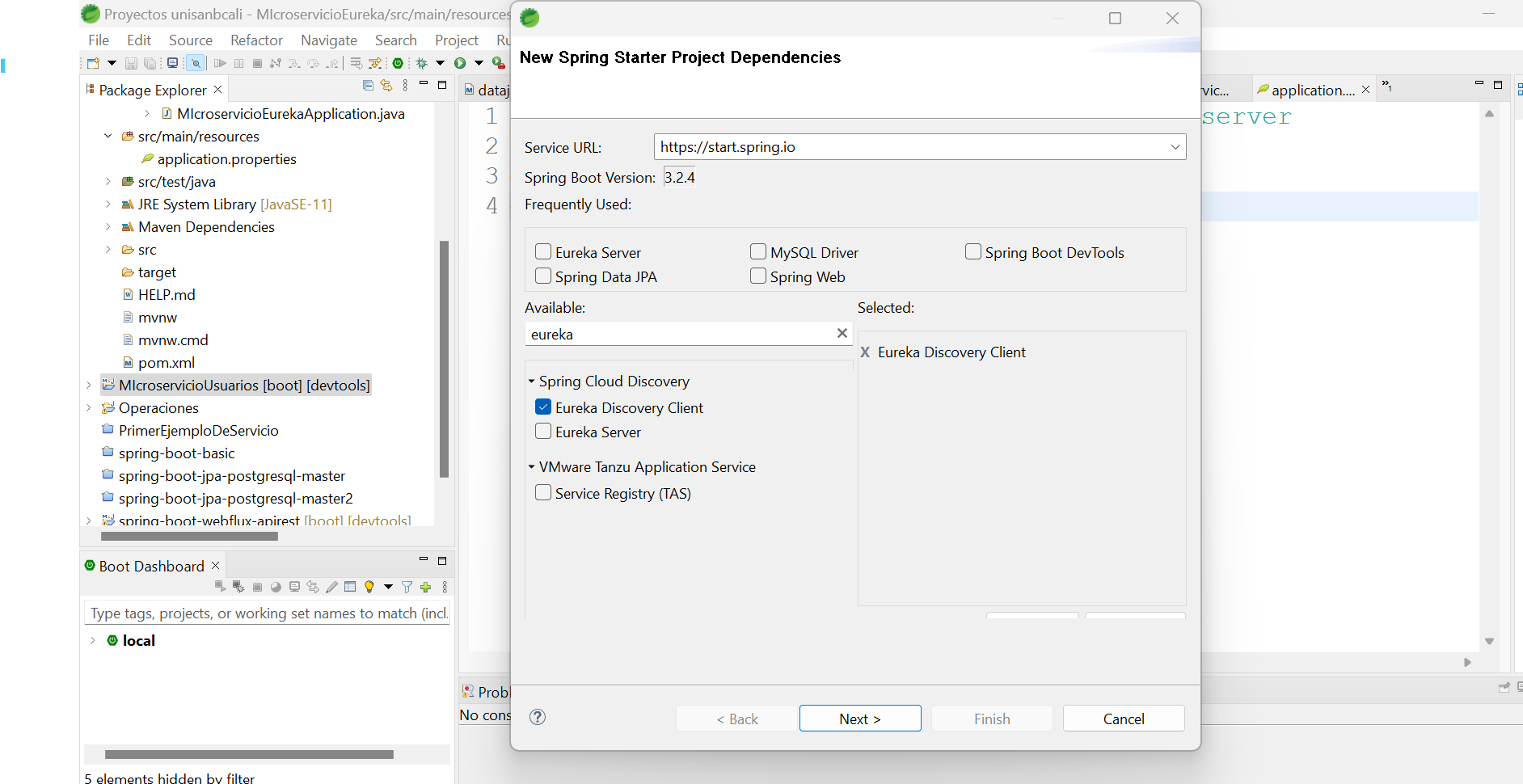
Y eso es todo de momento con el servicio de Eureka

1. Volvemos al servicio de Usuarios para agregar la dependencia de eureka cliente, se la siguiente forma

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

En la imagen anterior queremos editar las dependencias y agregamos la dependencia de EurekaClient



Luego se actualiza el pom de ese proyecto.

1. Se agrega el decorador de eureka cliente en la clase principal como se ve en la siguiente imagen

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

1. Le damos un nombre a este servicio a través del archivo de propiedades y le colocamos un puerto RAMDOM para escalar, como se ve a continuación:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

1. Se debe levantar primero el servicio de Eureka y luego el servicio de Usuarios
2. Probar que todo esta bien, primero debe salir el log en el ms de eureka el siguiente log:

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja

Luego abrimos el log del servicio de Usuarios y debe ver que esta enviadno señal al servidor de nombres cada 30 segundos como se ve en la imagen

Texto

Descripción generada automáticamente con confianza baja

Y por ultimo vamos al navegador y colocamos la siguiente url

<http://localhost:8761>

y deberá mostrar lo siguiente:

Una captura de pantalla de una red social

Descripción generada automáticamente

Si has llegado hasta aquí , acabas de completar el primer reto de esta actividad , donde has creado un servicio y lo has registrado en un servidor de nombre de Spring Cloud