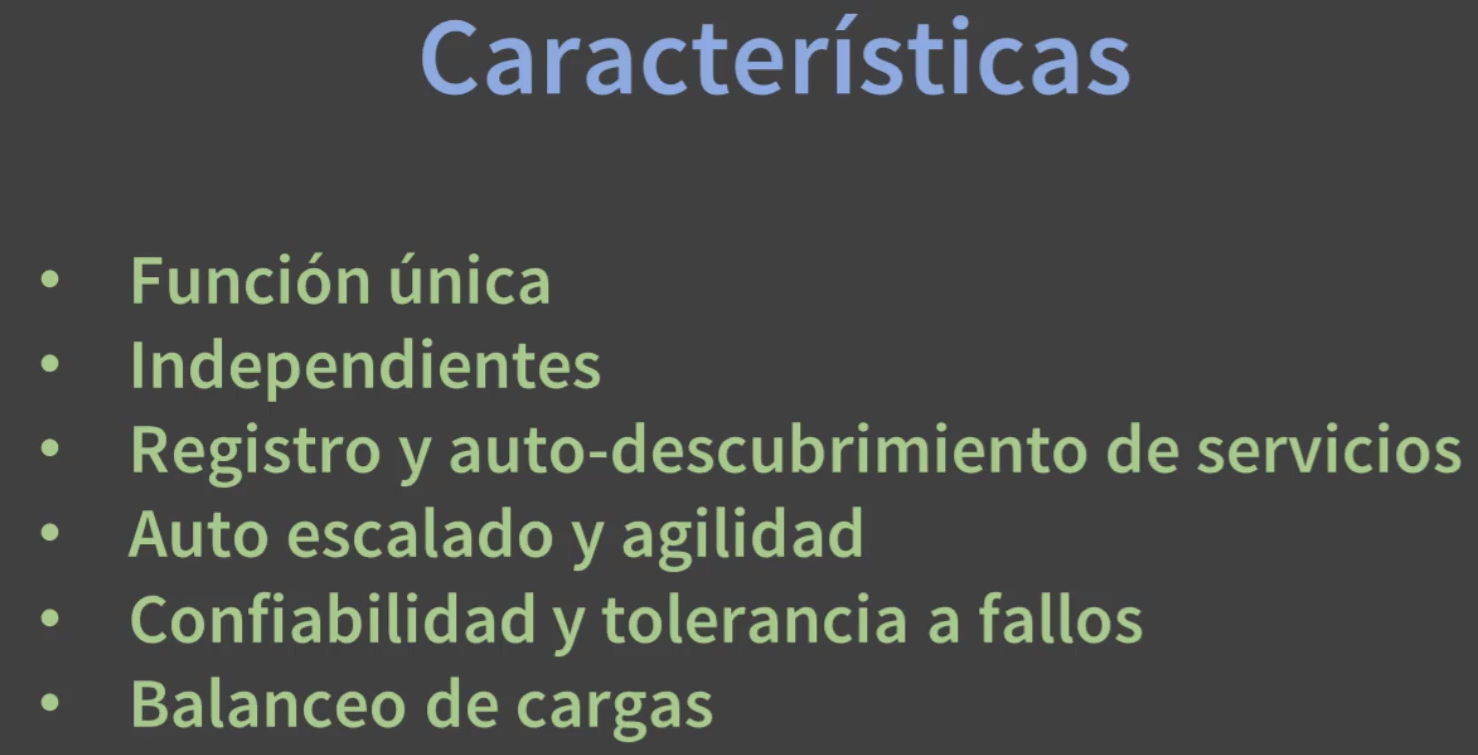
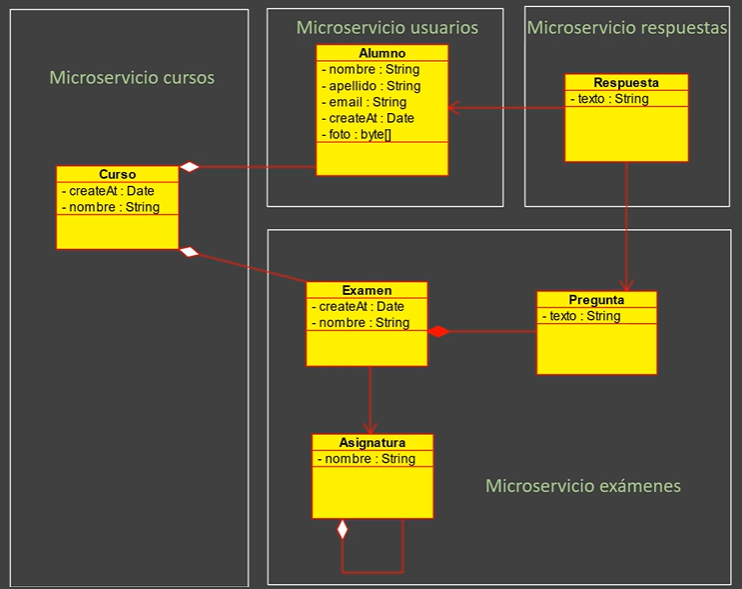


Servidor de nombre Eureka

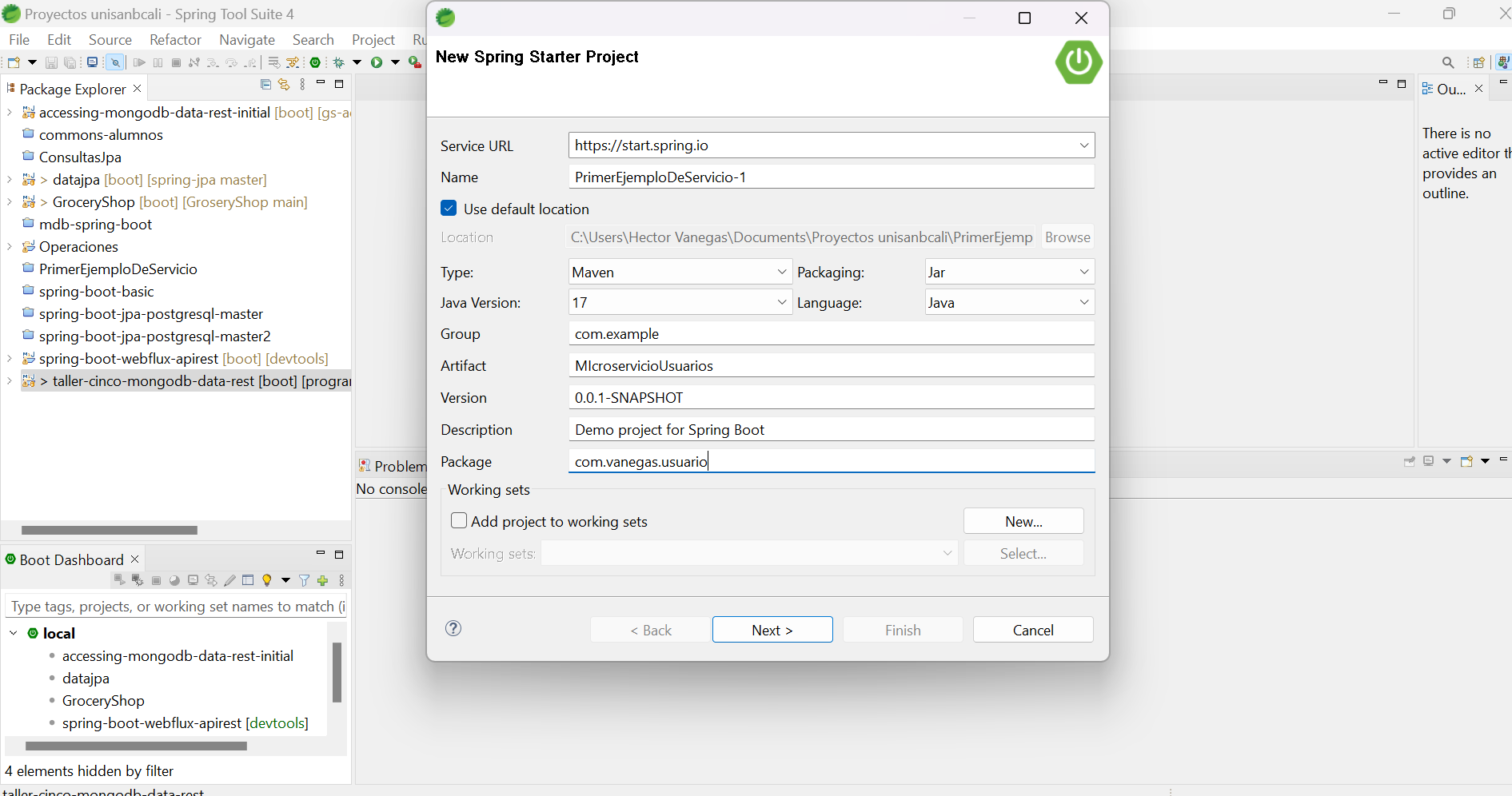


Llegaremos a construir este diagrama



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

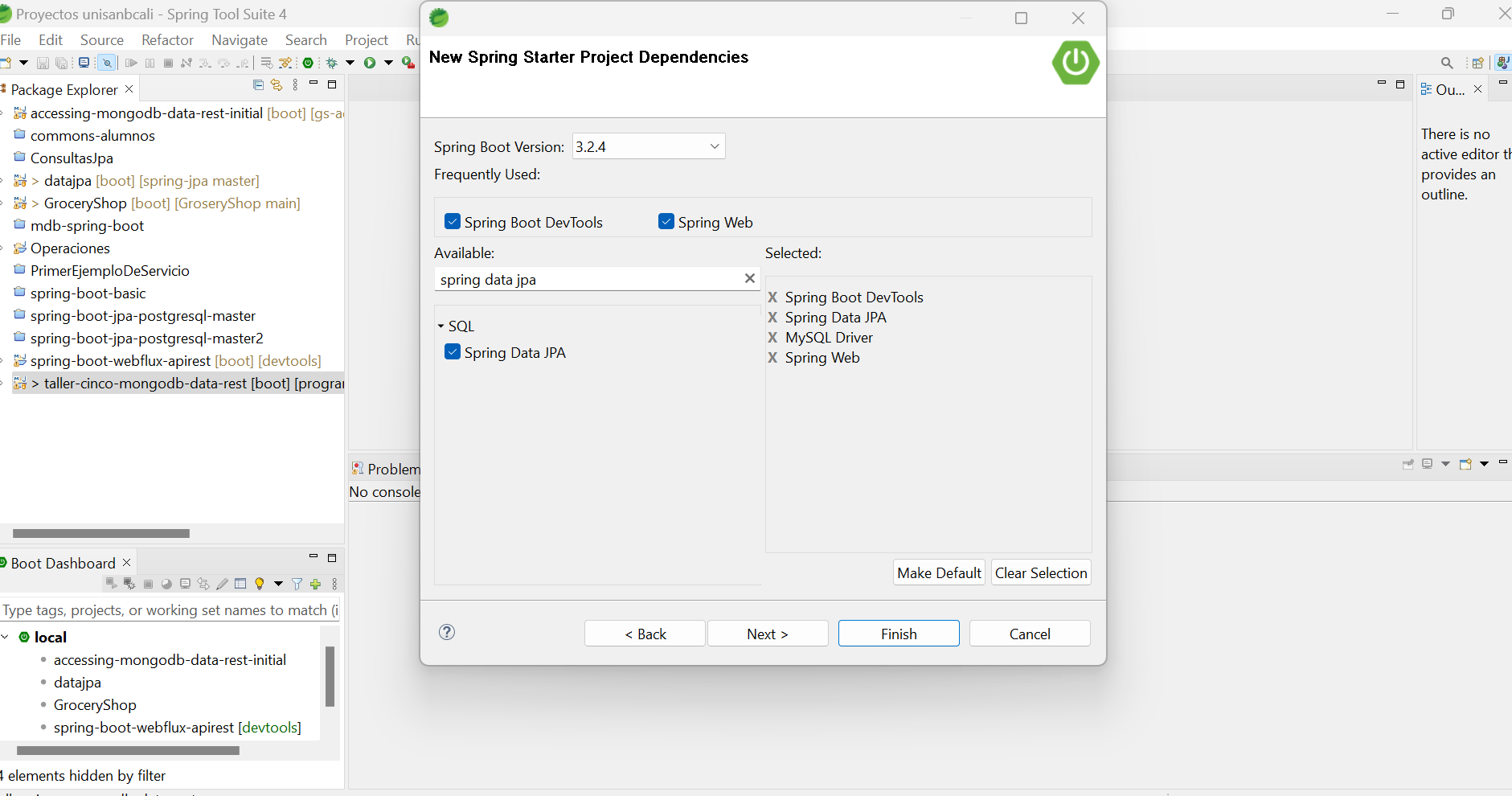
Spring Dev Tools

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

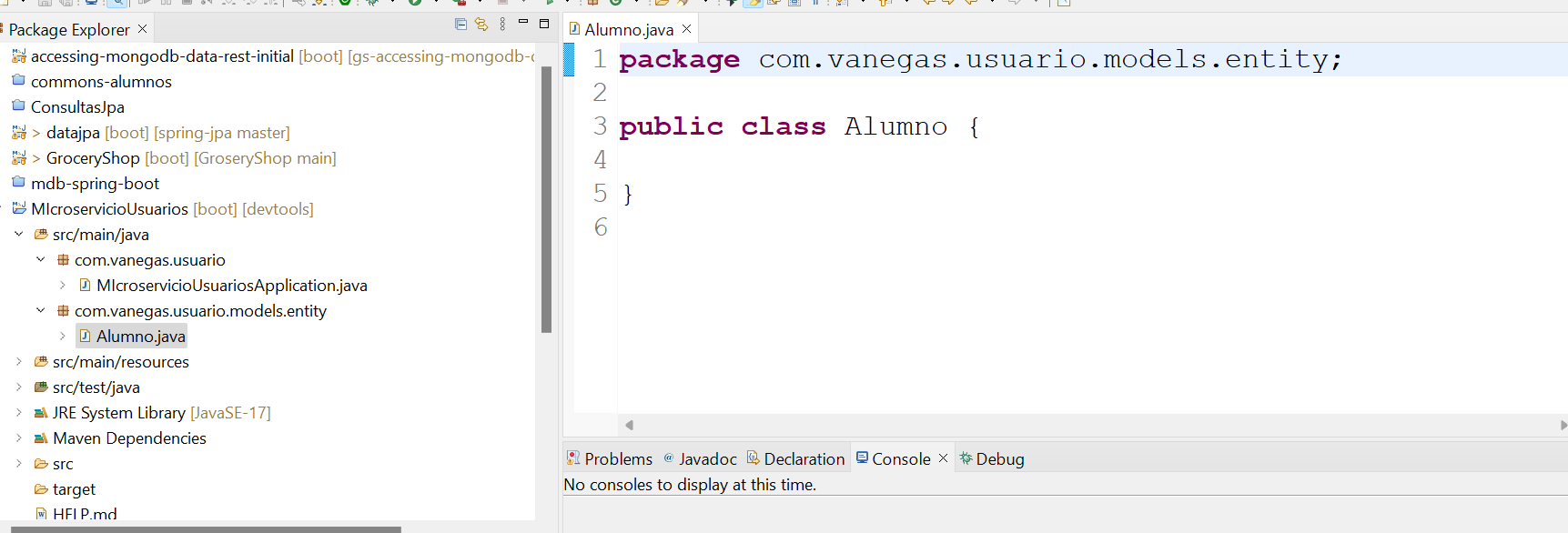
**Spring Web:** Esta dependencia proporciona soporte para crear aplicaciones web utilizando el framework Spring. Incluye funcionalidades como controladores, manejo de peticiones HTTP, vistas, entre otros.

**Spring Dev Tools:** Spring Dev Tools es una colección de herramientas que facilitan el desarrollo de aplicaciones Spring Boot. Incluye capacidades de reinicio automático de la aplicación cuando se detectan cambios en el código fuente, lo que agiliza el ciclo de desarrollo al evitar la necesidad de reiniciar manualmente el servidor.

**Spring Data JPA:** Spring Data JPA es una capa de abstracción que simplifica el acceso a bases de datos relacionales desde aplicaciones Spring

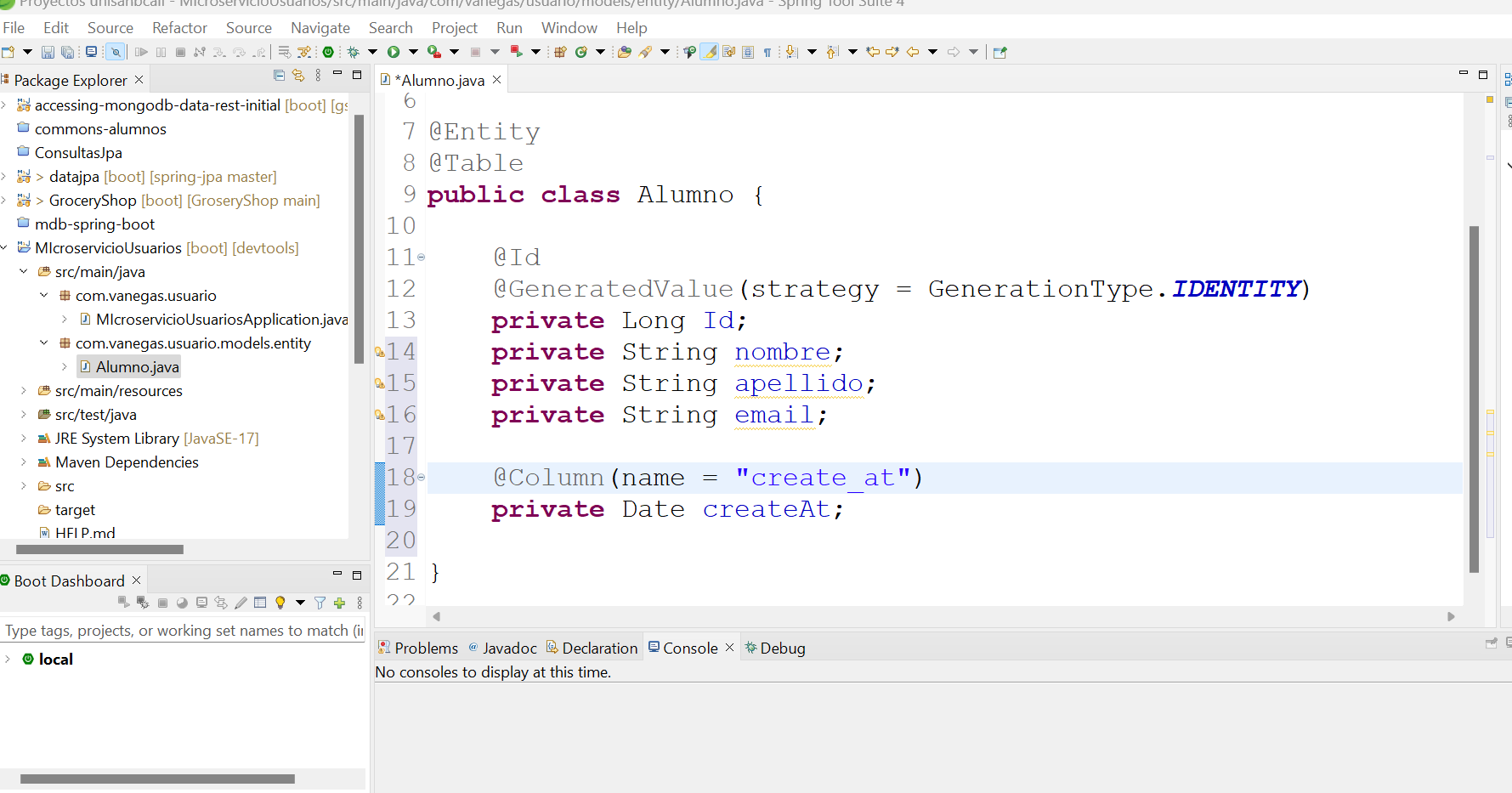
**PostgresDriver:** Esta referencia parece ser la dependencia del controlador JDBC (Java Database Connectivity) para PostgreSQL. JDBC es un estándar de Java que define cómo se pueden acceder a las bases de datos desde aplicaciones Java.

1. 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.
2. Dentro del paquete anterior se debe construir una clase concreta con el nombre de Alumno.

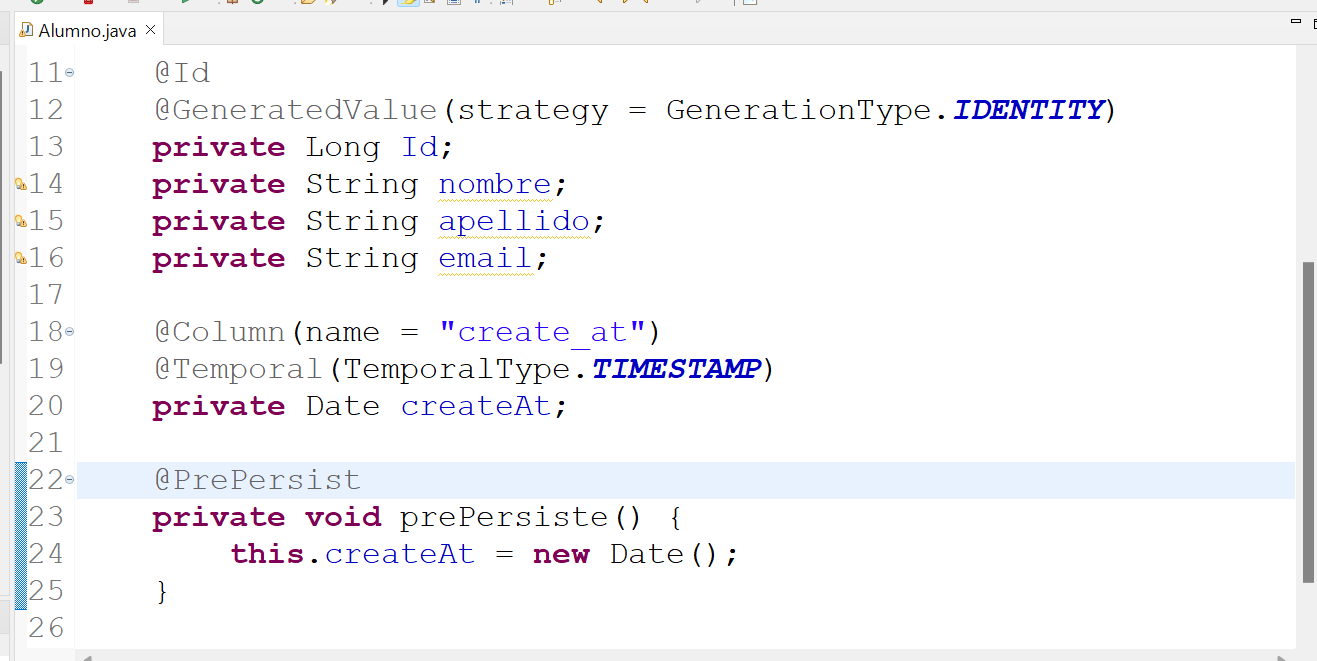


1. Colocaremos a nivel de clase los decoradores @Entity y @Table importándolo de la siguiente librería: jakarta.persistence
2. Agregamos los siguientes atributos de clase:

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.

**@Id:** Esta anotación se utiliza para marcar una propiedad de una clase como la clave primaria de una entidad en JPA.

**@GeneratedValue:** Se utiliza en combinación con @Id para especificar cómo se generará el valor de la clave primaria

**@Column:** Esta anotación se utiliza para especificar el mapeo entre el atributo de una entidad y la columna de la base de datos.

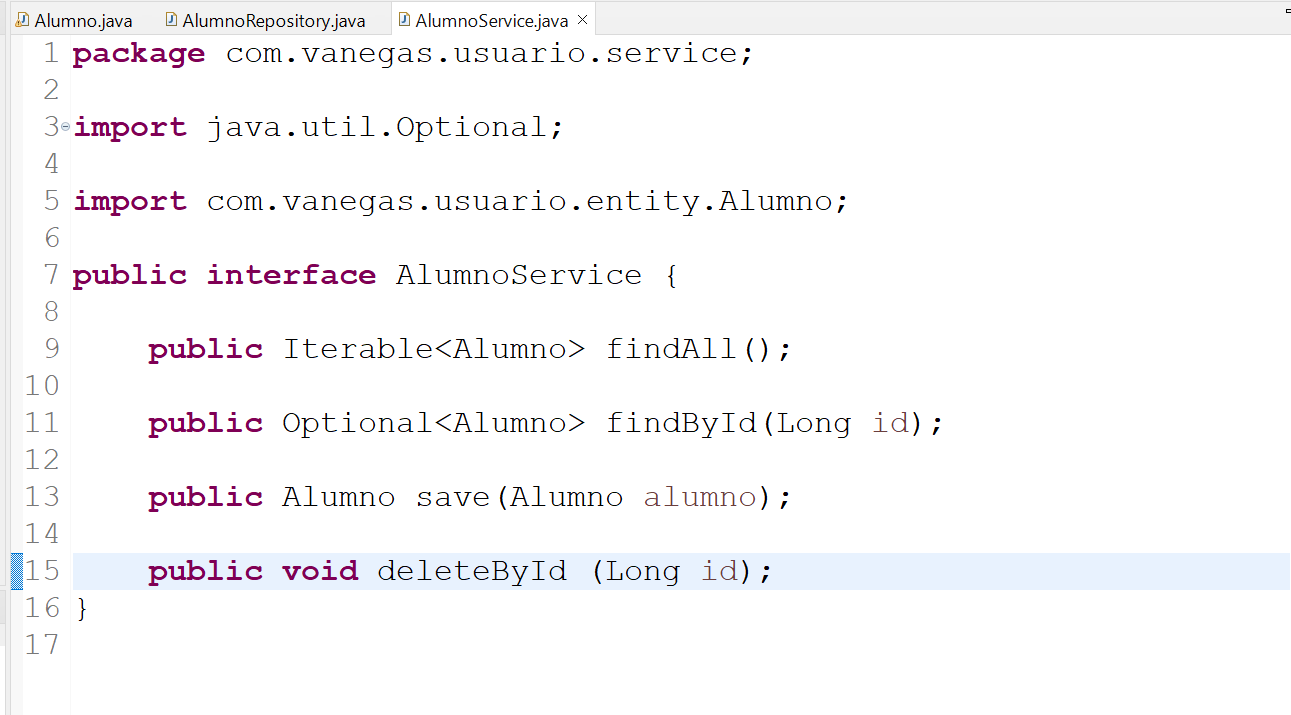
**@Temporal:** Se utiliza para especificar el tipo de mapeo temporal de un atributo de fecha o hora en una entidad. Puede ser utilizado con los tipos de datos java.util.Date, java.util.Calendar y java.time.LocalDate, java.time.LocalDateTime, java.time.LocalTime en JPA para indicar cómo se debe persistir el atributo temporalmente en la base de datos.

**@PrePersist:** Esta anotación se utiliza para marcar un método que se debe ejecutar justo antes de que se persista una entidad en la base de datos.

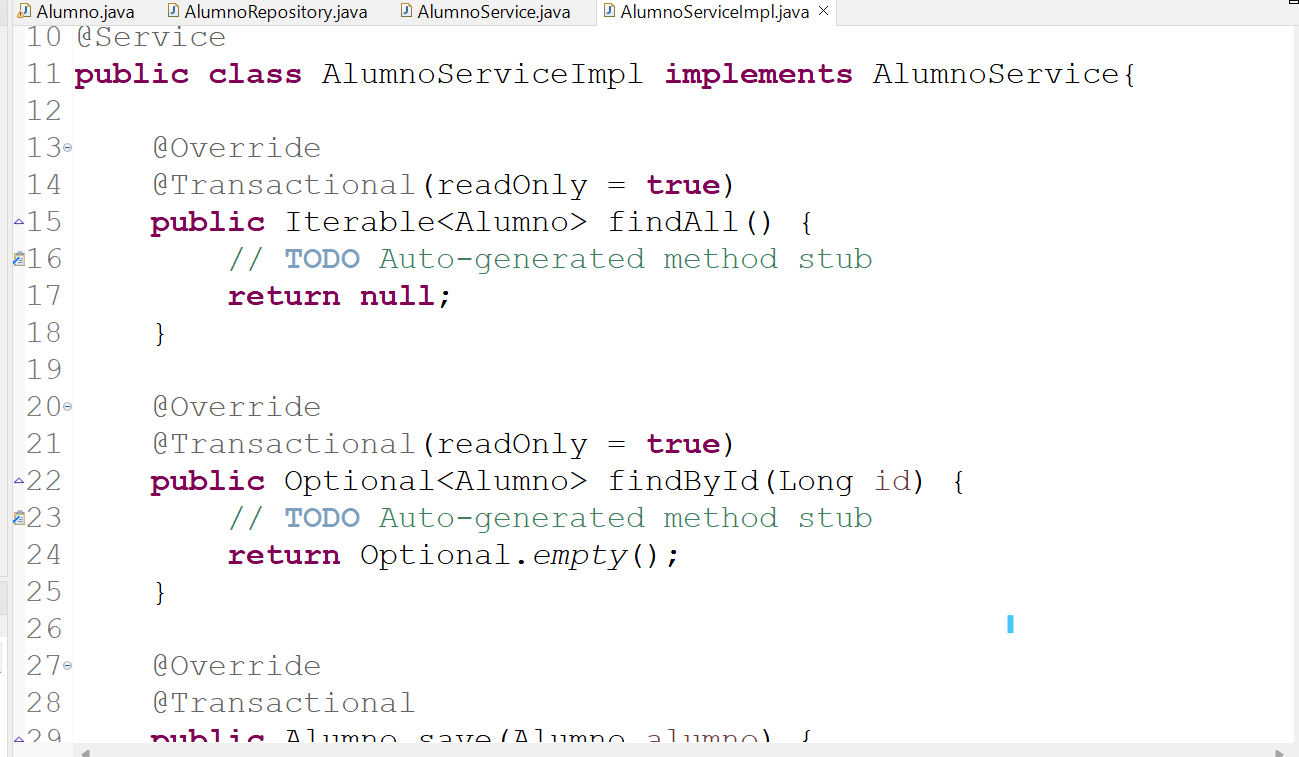
**@Entity:** Se utiliza para marcar una clase como una entidad JPA, lo que indica que la clase debe ser mapeada a una tabla en la base de datos

**@Table:** Esta anotación se utiliza para especificar el nombre de la tabla de base de datos a la que se debe asignar una entidad.

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:



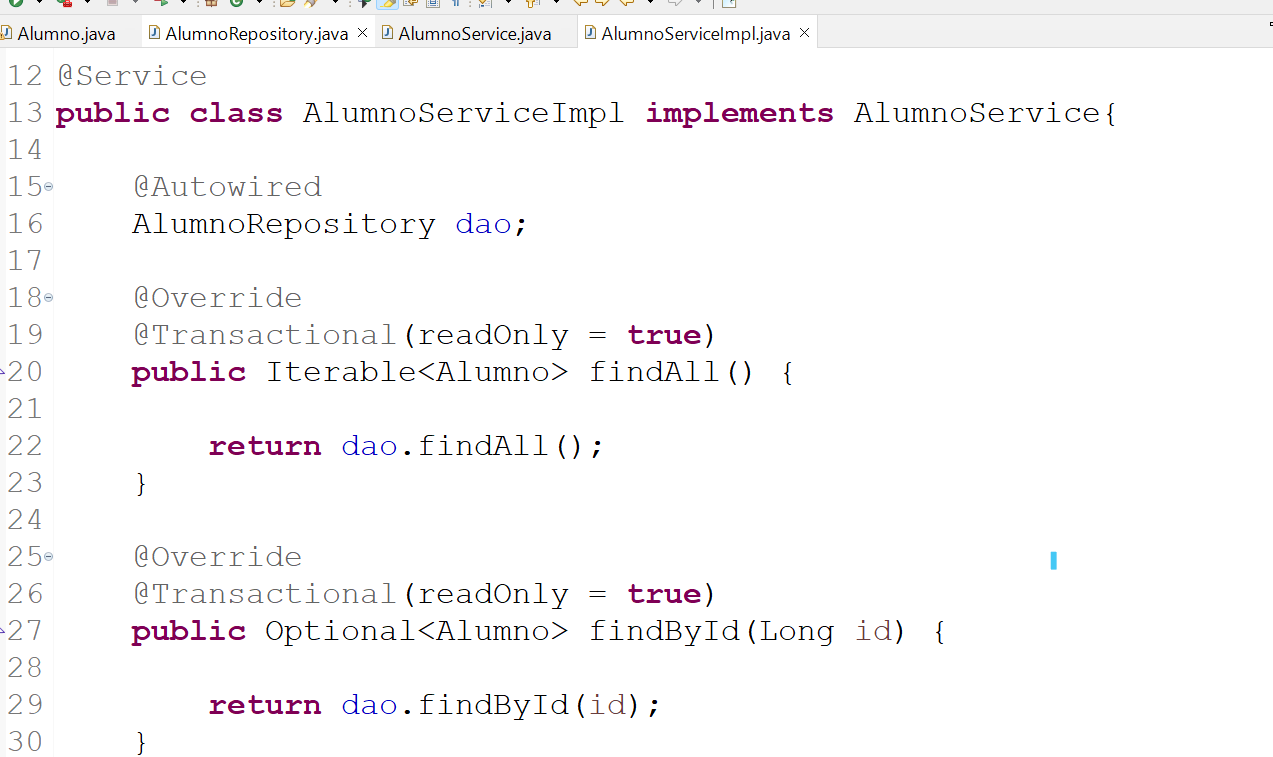
Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

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

El decorador @Transactional es una anotación en Spring que se utiliza para indicar que un método o una clase debe ser gestionado por transacciones. Esta anotación se utiliza al trabajar con bases de datos relacionales, donde es fundamental mantener la consistencia de los datos a través de transacciones.

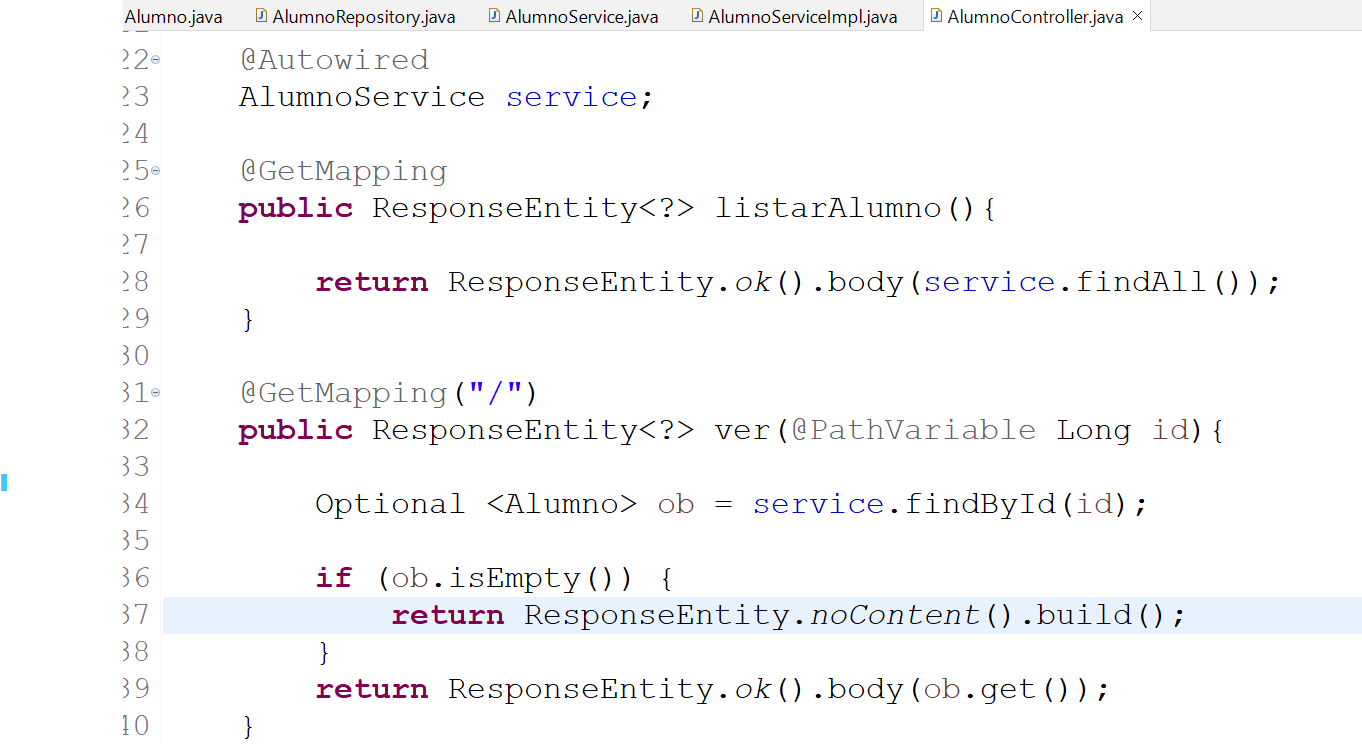
1. Inyectamos en esta clase la interface AlumnoRepository e implementamos cada uno de los métodos como se muestra en la siguiente imagen:



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 :

ResponseEntity<?>

Optional

@PathVariable

**ResponseEntity<?>:** ResponseEntity es una clase proporcionada por Spring Framework que representa toda la respuesta HTTP: código de estado, encabezados y cuerpo. Permite controlar directamente todos los aspectos de la respuesta devuelta desde un controlador de Spring. El genérico <?> significa que puede retornar cualquier clase de objeto.

**Optional:** es una clase que se utiliza para representar un valor opcional. En el contexto de Spring Boot, Optional se utiliza principalmente para evitar el manejo de valores nulos de manera explícita.

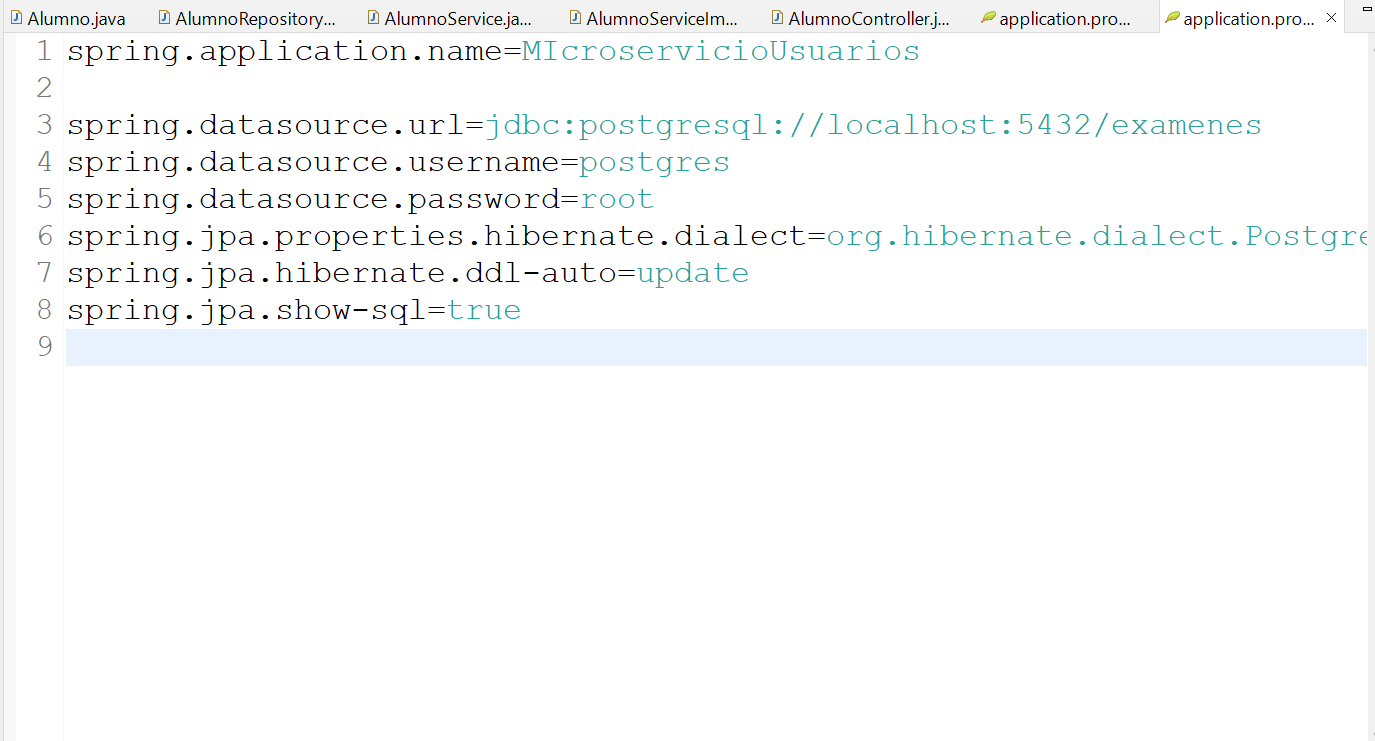
**@PathVariable** es una anotación que se utiliza en métodos de controladores para mapear variables de la URL a parámetros de método en un controlador de Spring. Esto permite que los valores proporcionados en la URL se pasen como argumentos al método del controlador. Por ejemplo, si tienes una URL como /usuarios/{id}, donde {id} es una variable en la URL, puedes usar @PathVariable para vincular el valor de {id} a un parámetro en el método del controlador.

Y los otros métodos son :

Imagen que contiene Texto

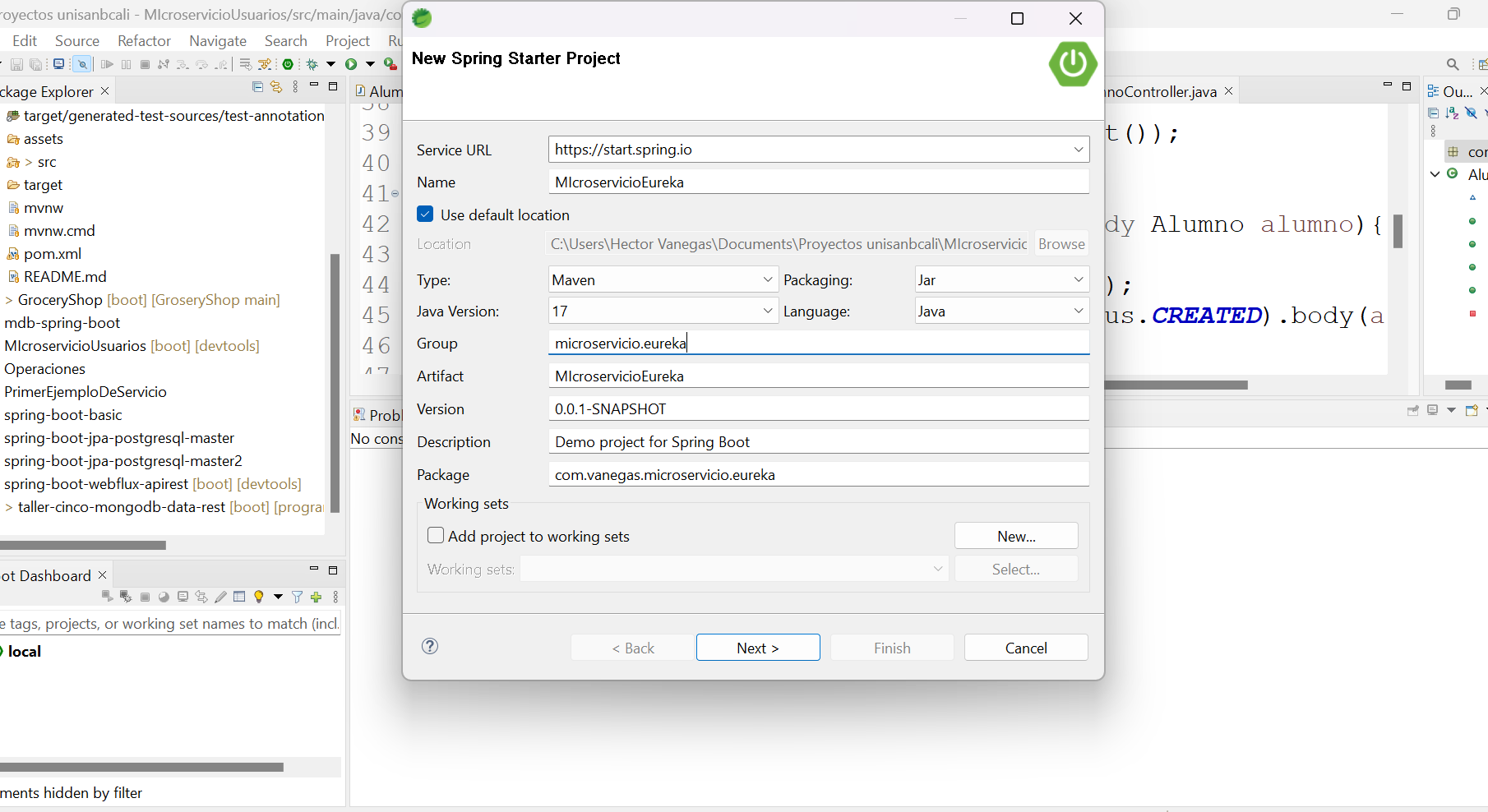
Descripción generada automáticamente

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>f

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

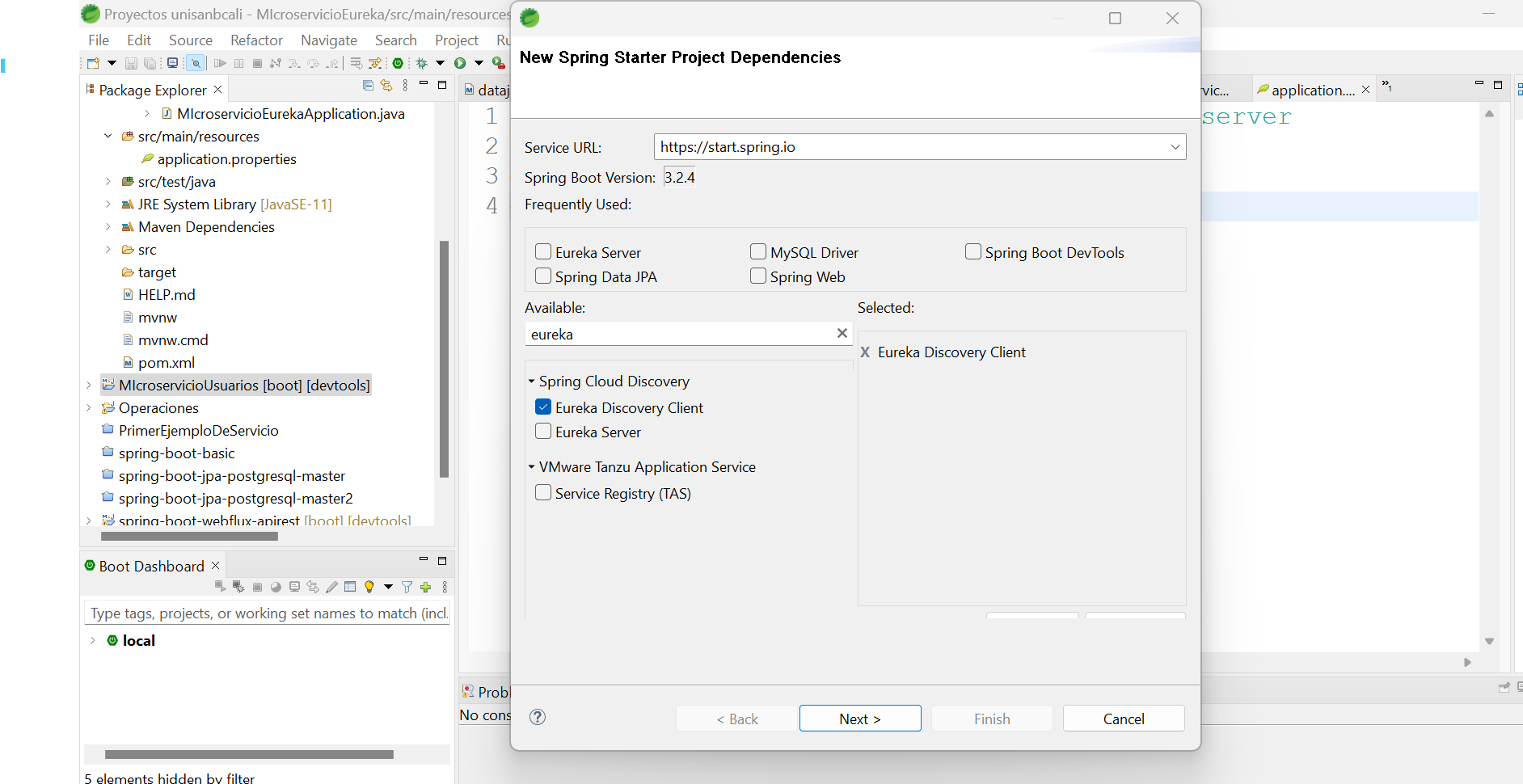
Y eso es todo de momento con el servicio de Eureka

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