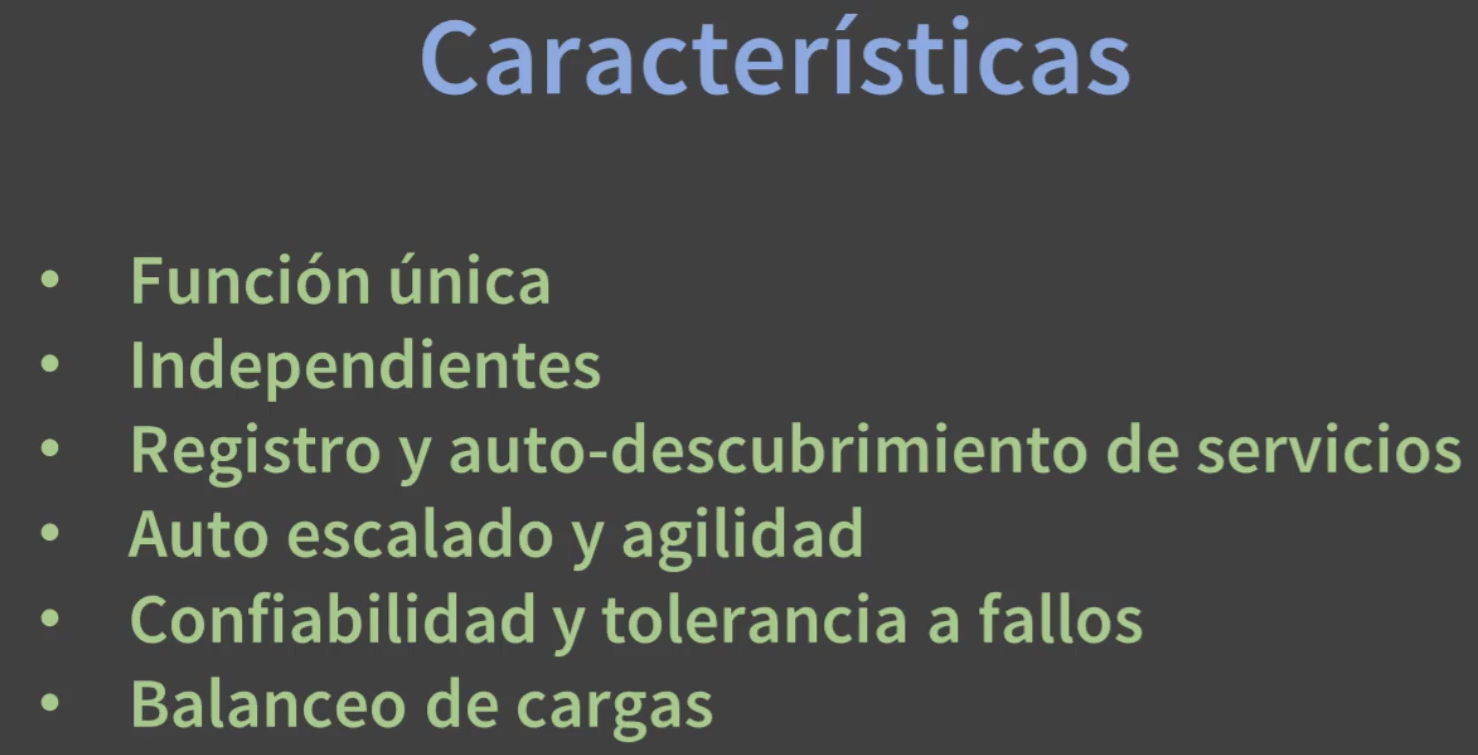
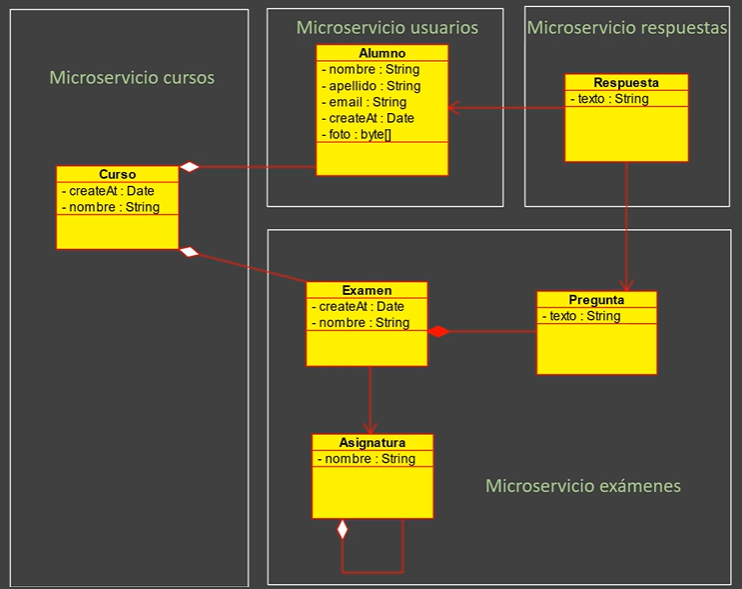


Servidor de nombre Eureka

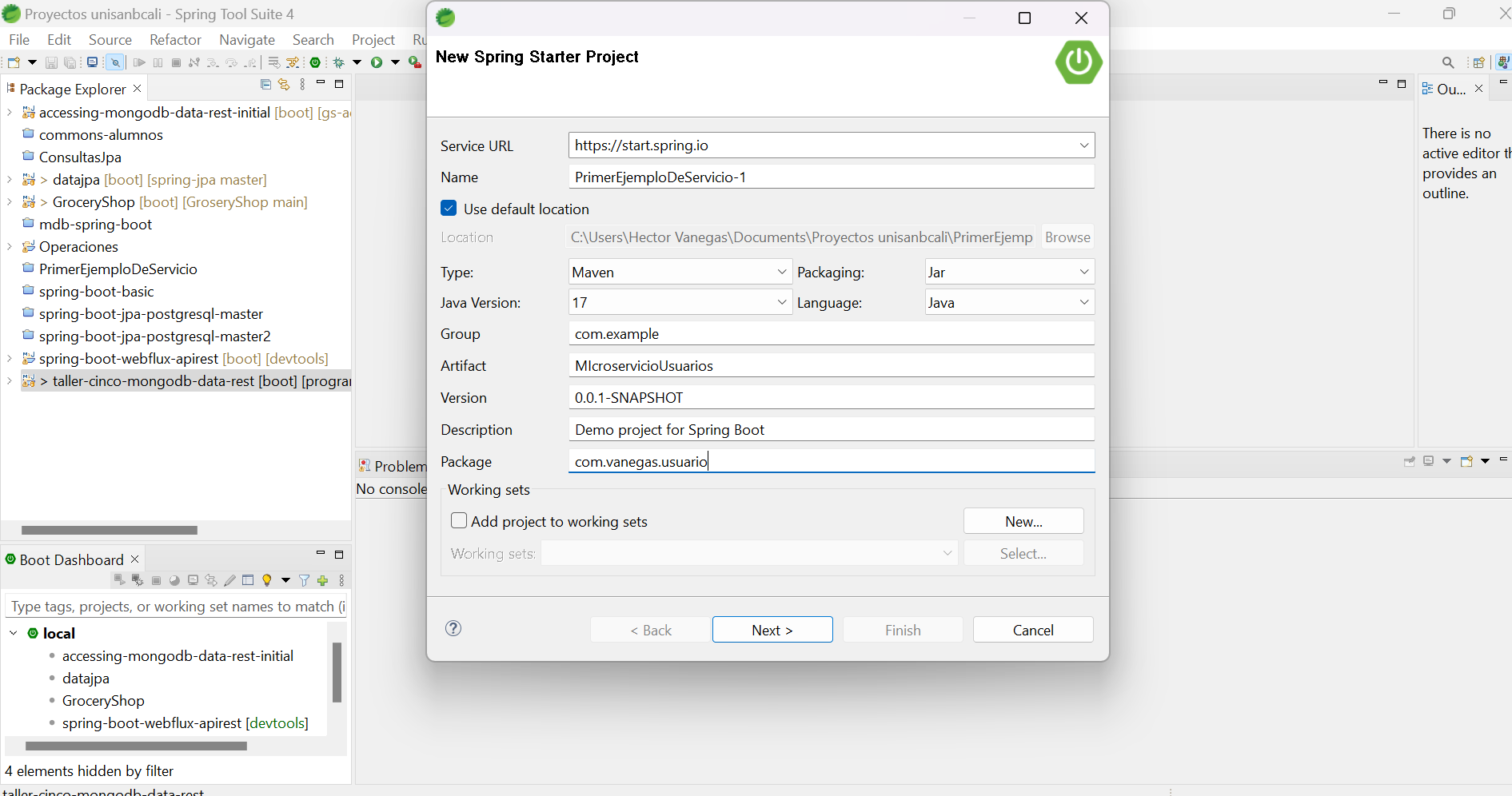


Llegaremos a construir este diagrama



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

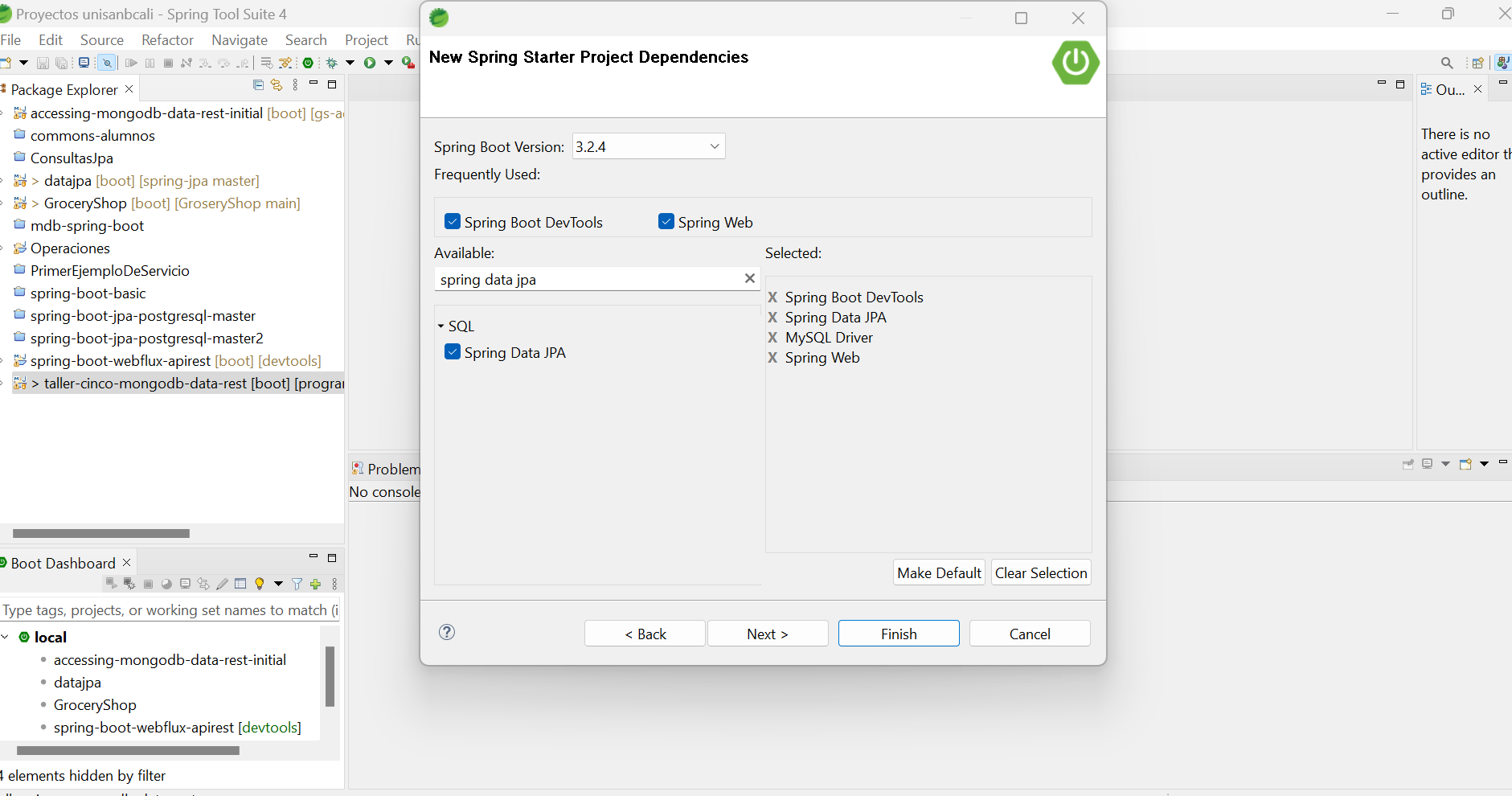
Spring Dev Tools

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

R//

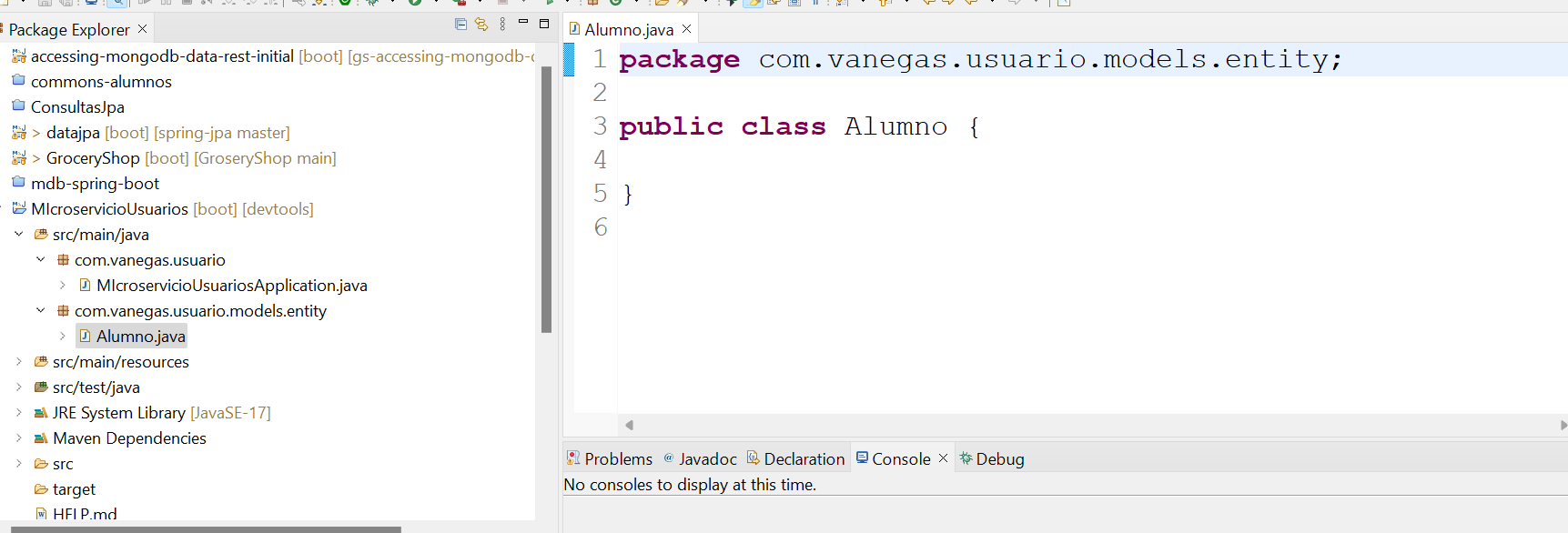
Spring Web: Esta dependencia permite construir aplicaciones web y servicios REST en Java utilizando Spring Boot. Incluye herramientas para manejar solicitudes HTTP, realizar enrutamiento, configurar controladores y más.

Spring Dev Tools: Spring DevTools (Herramientas de Desarrollo de Spring) es una dependencia que facilita el proceso de desarrollo en Spring Boot, especialmente en etapas tempranas.

Spring Data JPA: Spring Data JPA proporciona una capa de abstracción sobre JPA (Java Persistence API), simplificando la implementación de acceso a datos en bases de datos relacionales. Está construido sobre la API de JPA, pero proporciona un enfoque de acceso a datos basado en repositorios y elimina gran parte del código repetitivo.

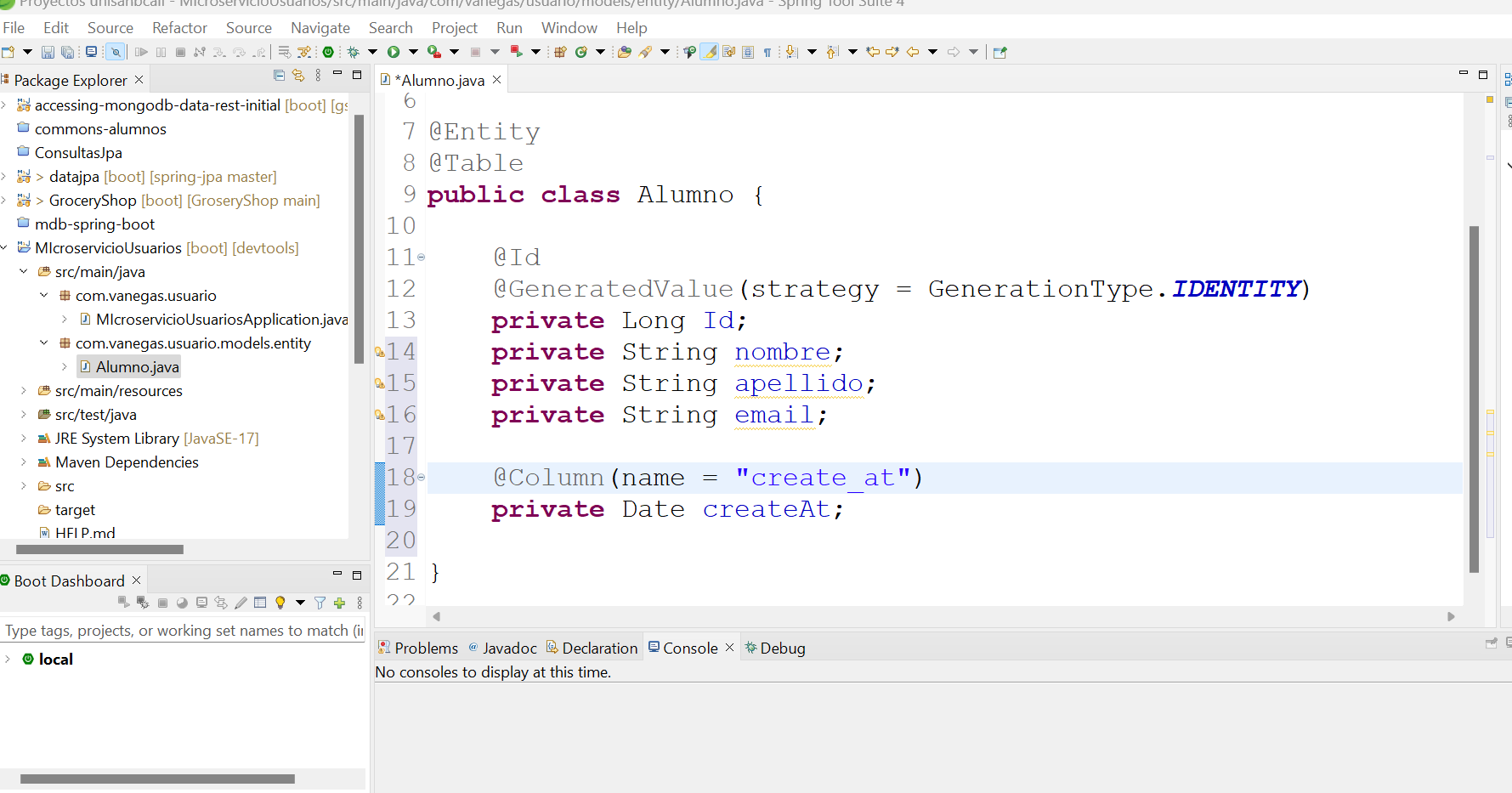
MySQL Driver: Esta dependencia incluye el controlador JDBC para MySQL, que permite la conexión y comunicación de una aplicación Spring Boot con una base de datos MySQL.

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

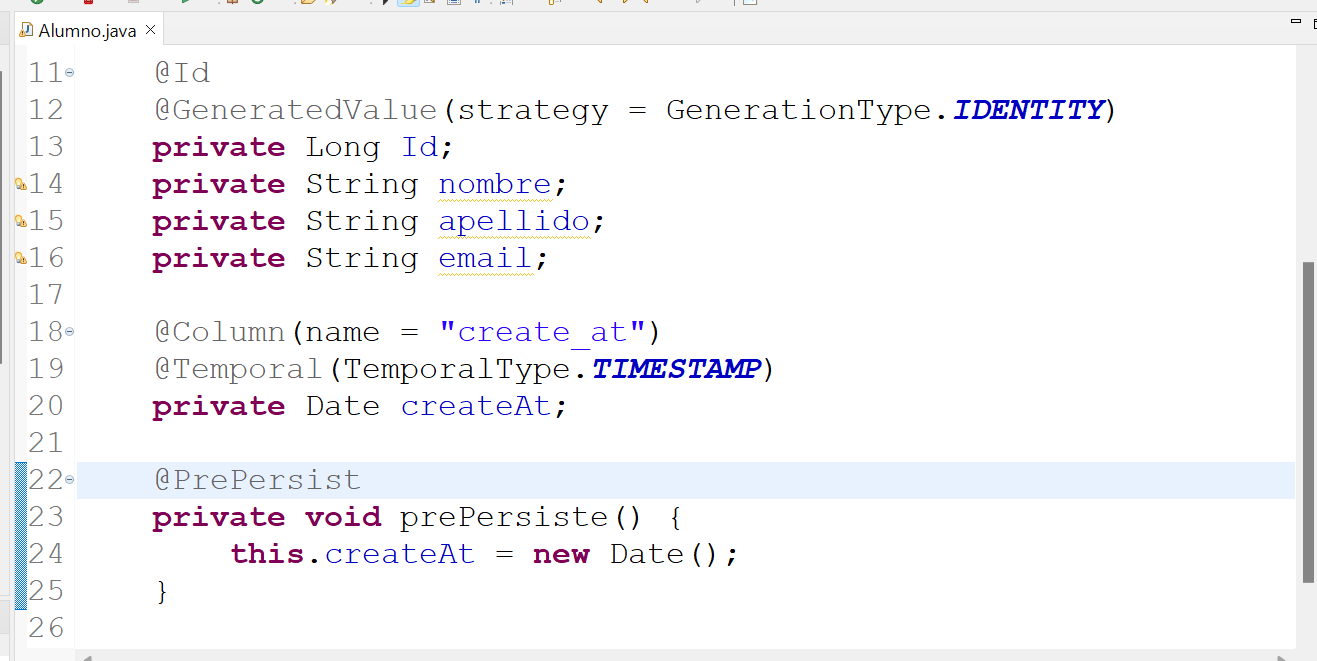


1. Colocaremos a nivel de clase los decoradores @Entity y @Table importándolo de la siguiente librería: jakarta.persistence - hecho
2. Agregamos los siguientes atributos de clase:

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: Indica que el campo de la clase es la clave primaria de la entidad en la base de datos. Es un requisito en JPA para identificar de forma única cada registro de la entidad.

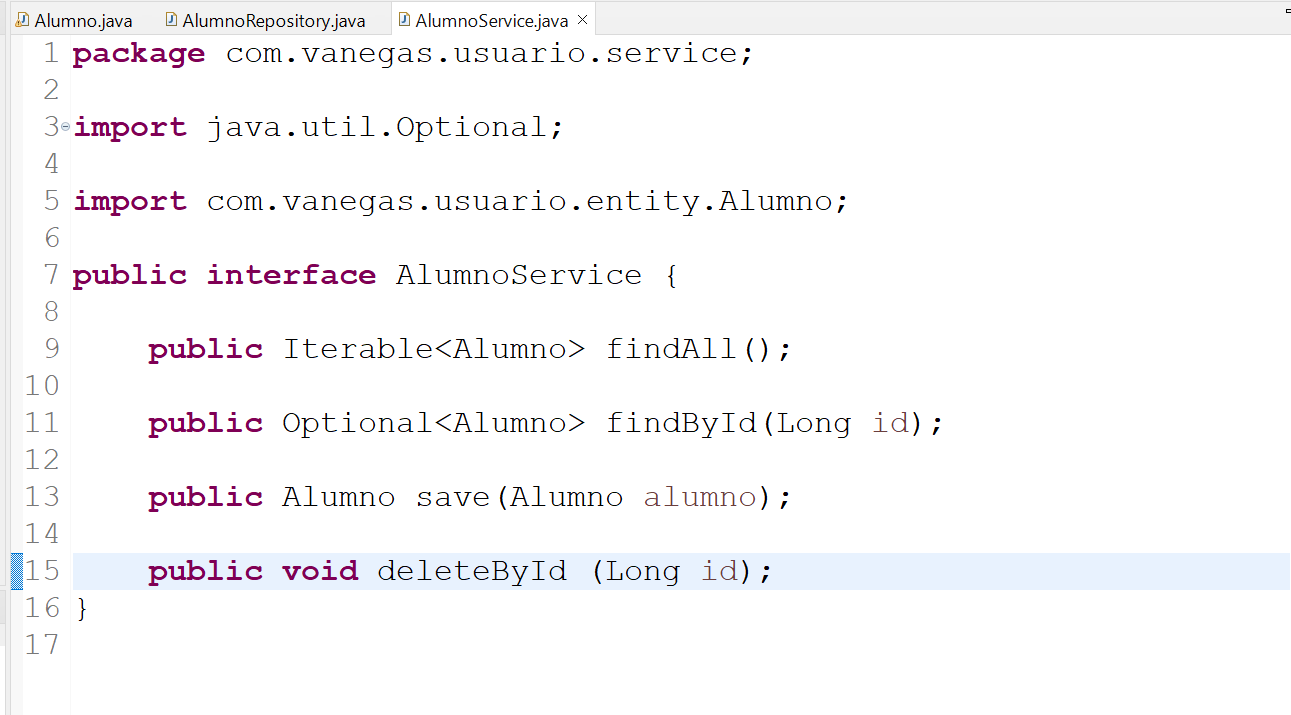
@GeneratedValue: Especifica que el valor de la clave primaria será generado automáticamente. Puede configurarse para que el valor sea generado de diferentes formas (por ejemplo, autoincremento en la base de datos).

@Column: Define el nombre de la columna en la base de datos que se mapea a este campo de la clase. Permite configurar propiedades como el nombre, tipo de datos y si el campo es obligatorio o no.

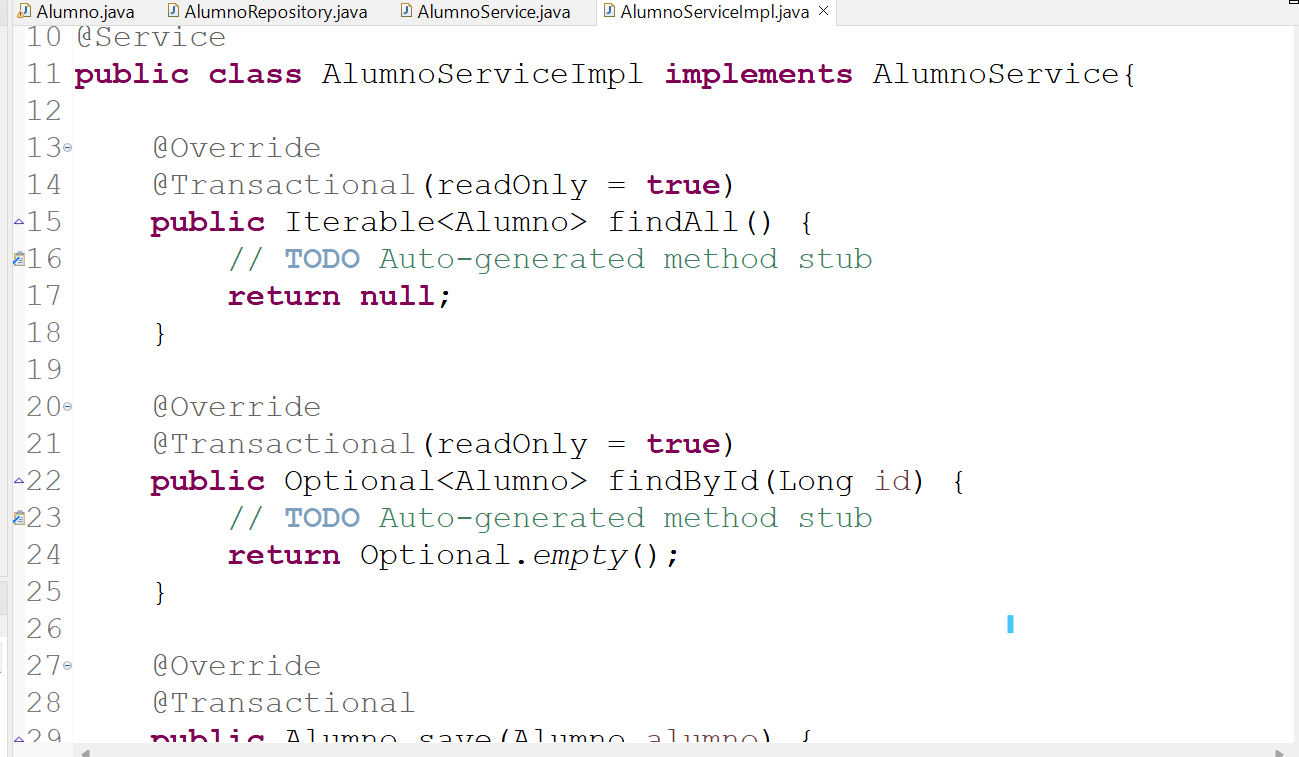
@Temporal: Indica el tipo de dato temporal (como Date) a usar en la columna de la base de datos. En JPA, se usa para especificar si el campo debe almacenar solo la fecha (TemporalType.DATE), solo la hora (TemporalType.TIME), o fecha y hora completa (TemporalType.TIMESTAMP).

@PrePersist: Marca un método de ciclo de vida de JPA que se ejecuta antes de que una nueva entidad se guarde en la base de datos. Este método es útil para realizar operaciones previas, como inicializar valores o validar datos antes de que se haga la inserción en la base de datos.

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 - hecho
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 - hecho
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:



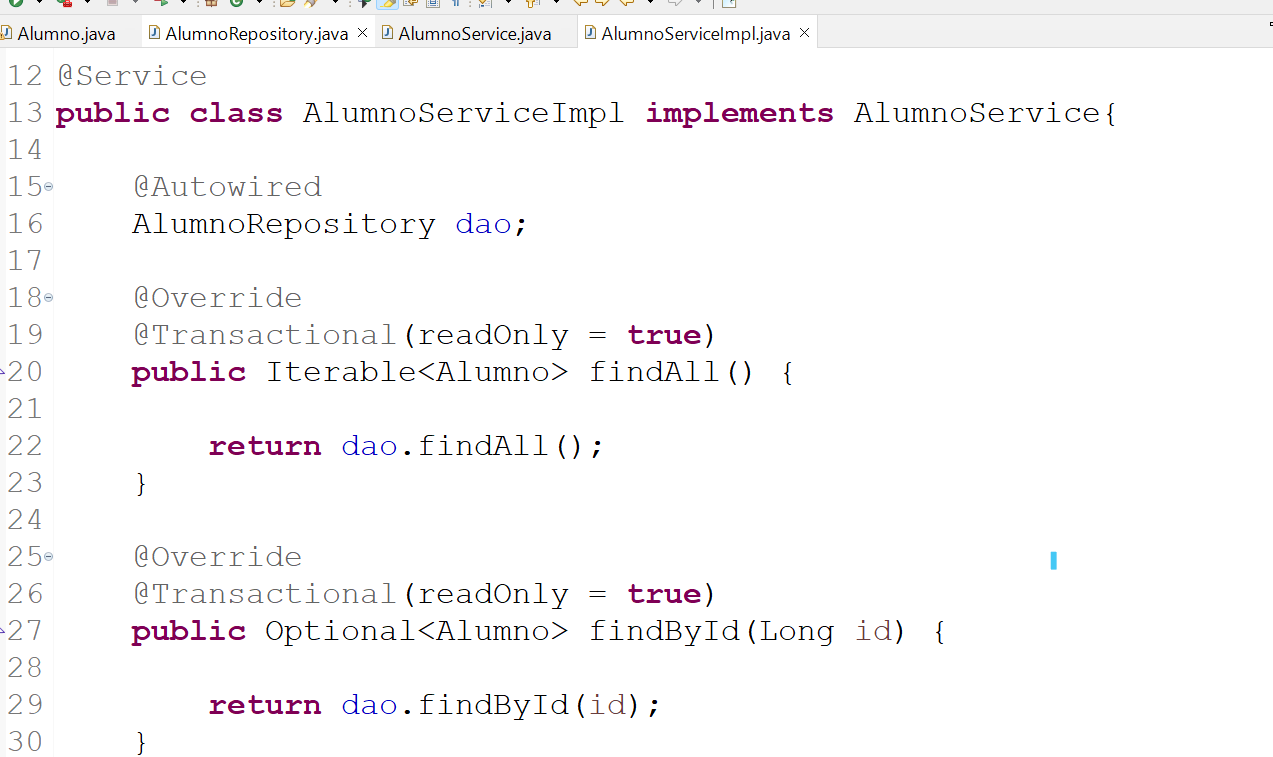
Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

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

@Transactional: Indica que el método o clase en la que se aplica debe ejecutarse dentro de una transacción de base de datos. En Spring, este decorador asegura que todas las operaciones dentro de su ámbito se realicen de manera atómica, es decir, todas deben completarse con éxito para que se confirmen (commit) los cambios en la base de datos. Si ocurre una excepción no controlada, se revierte (rollback) la transacción, deshaciendo cualquier cambio parcial realizado en la base de datos durante la ejecución del método. Este decorador es útil para mantener la integridad de los datos y evitar inconsistencias en operaciones que implican múltiples acciones sobre la base de datos.

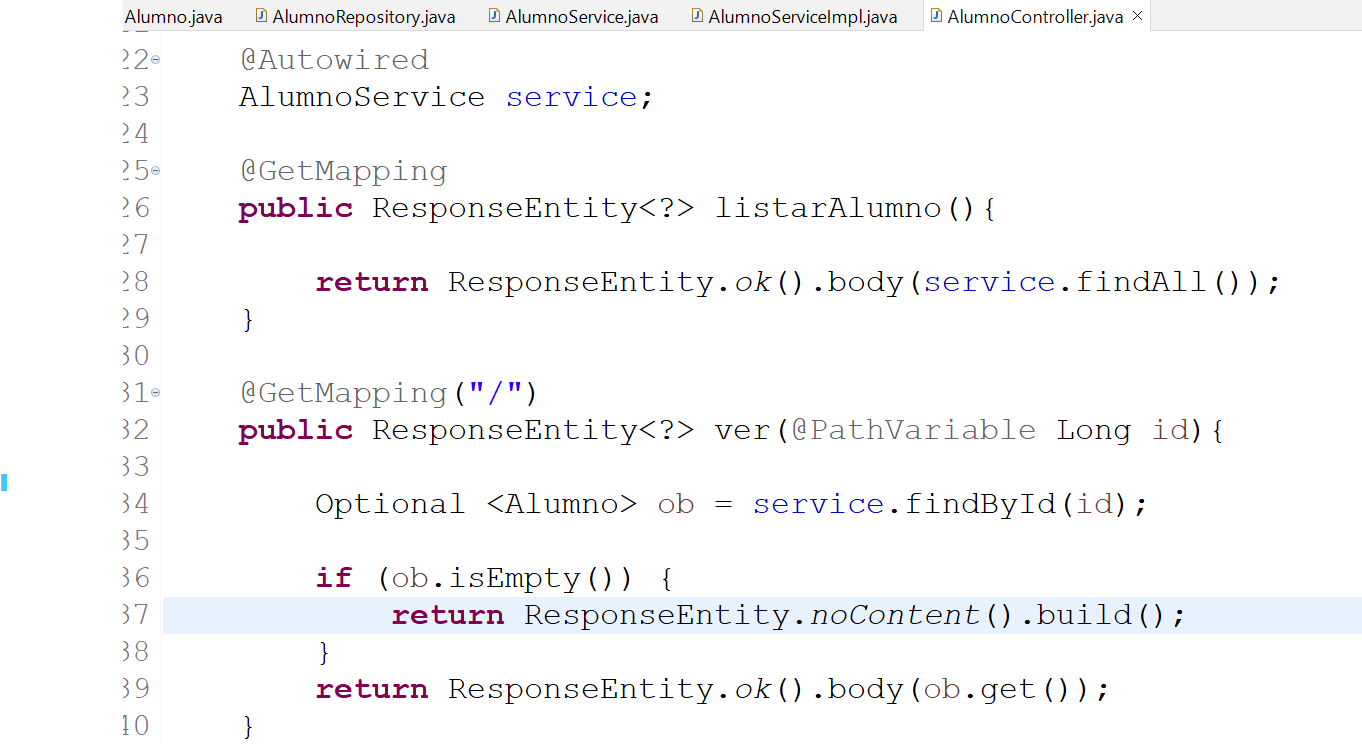
1. Inyectamos en esta clase la interface AlumnoRepository e implementamos cada uno de los métodos como se muestra en la siguiente imagen:



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. - hecho
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<?>: es una clase de Spring utilizada para representar una respuesta HTTP completa, incluyendo el cuerpo de la respuesta, los encabezados y el código de estado HTTP. <?> indica que es un tipo genérico que puede contener cualquier tipo de objeto como cuerpo de la respuesta.
3. Optional: es una clase de Java (introducida en Java 8) que actúa como un contenedor para valores que podrían estar presentes o no. Es una forma segura de manejar posibles valores nulos, reduciendo el riesgo de excepciones NullPointerException.
4. @PathVariable: es una anotación de Spring utilizada para extraer variables de la URL en los métodos de controlador. Permite acceder a partes específicas de la ruta para utilizarlas como argumentos del método.

Y los otros métodos son :

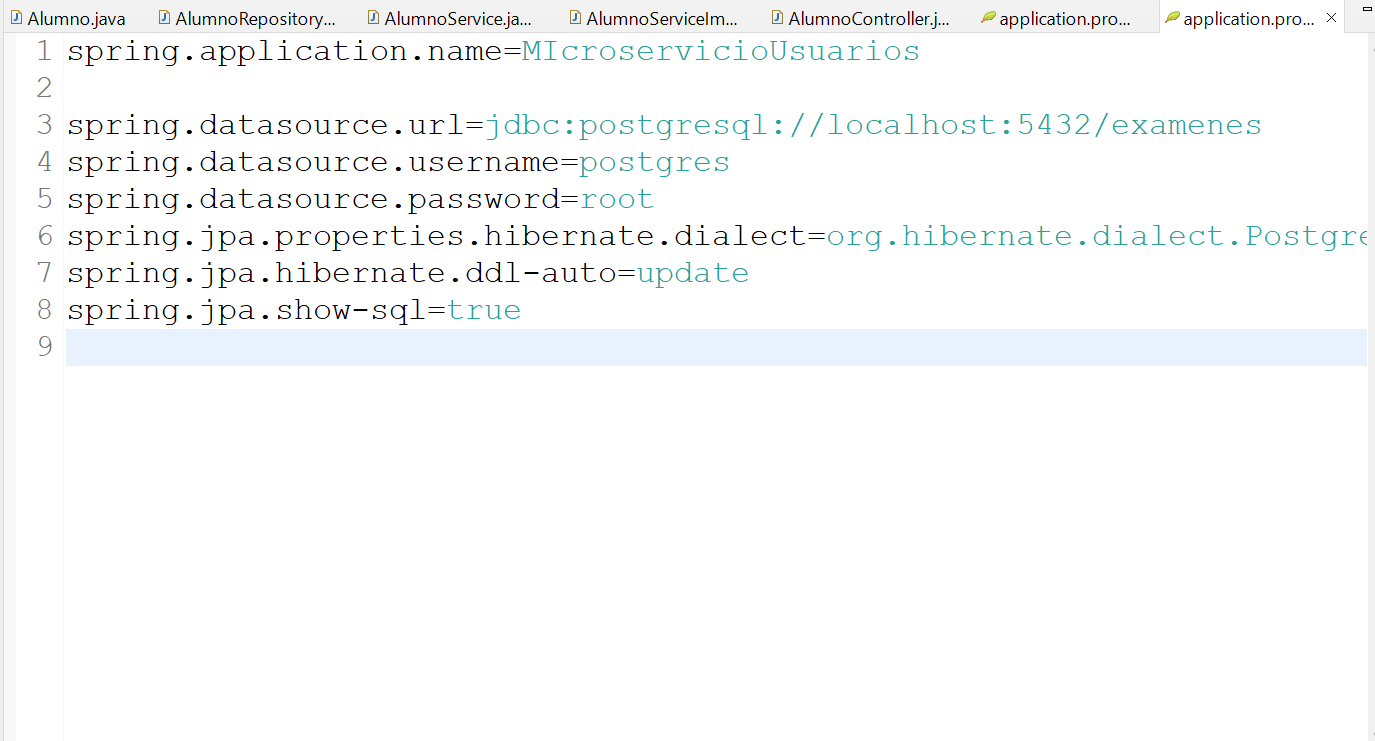
Imagen que contiene Texto

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:

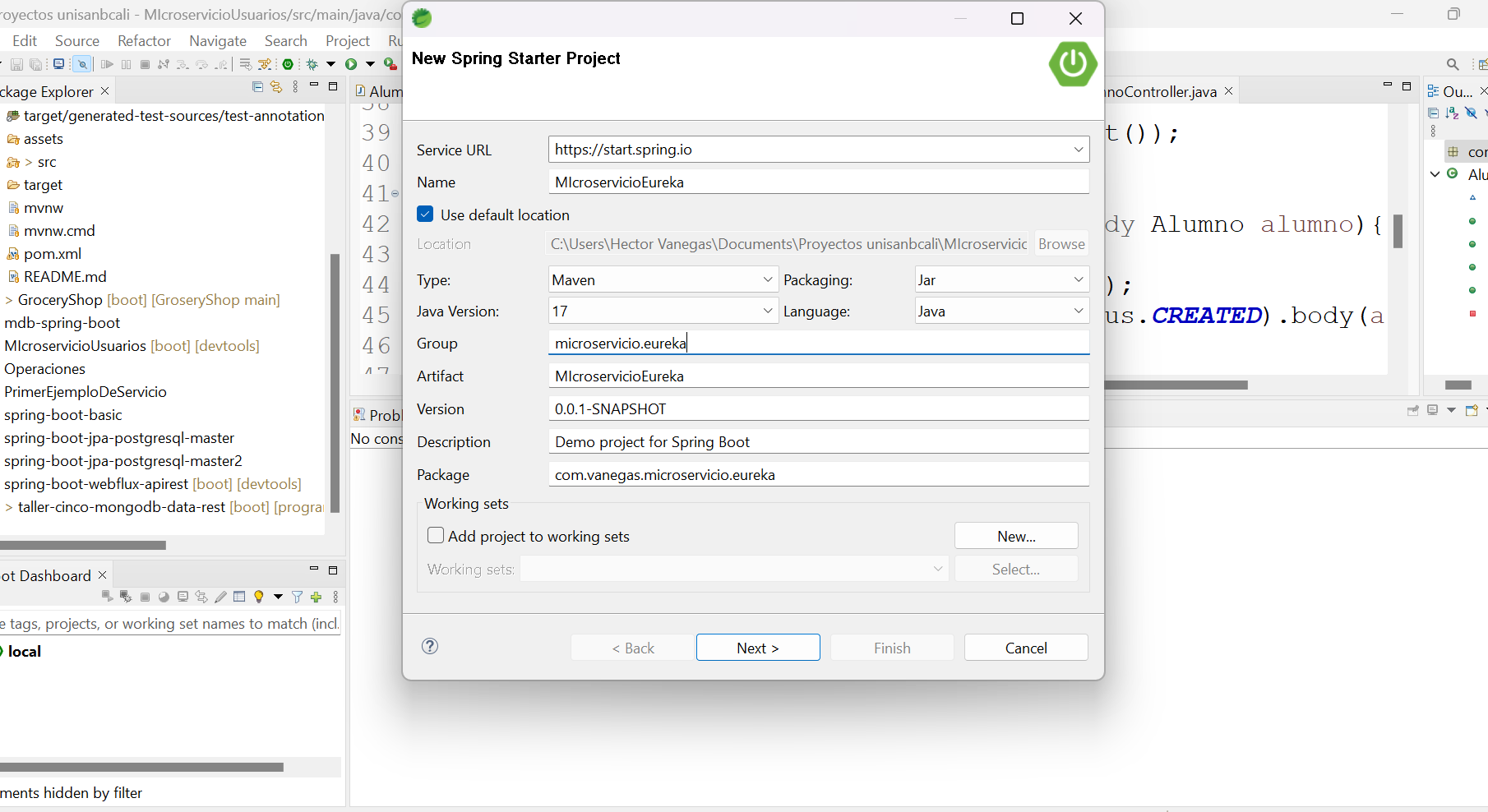


**CREACIÓN SERVIDOR EUREKA**

1. Investigar y colocar en este parte del documento lo que es el servidor EUREKA como parte de Spring Cloud

R// Es un servicio de descubrimiento proporcionado por Spring Cloud como parte de Netflix OSS. Funciona como un servidor de registro, permitiendo que los microservicios se registren y se descubran entre sí automáticamente. En un entorno de microservicios, donde las instancias pueden escalar dinámicamente, el servidor Eureka mantiene una lista actualizada de todas las instancias de servicios disponibles, permitiendo que cada servicio localice a otros servicios por su nombre sin necesidad de conocer las direcciones IP exactas.

1. 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

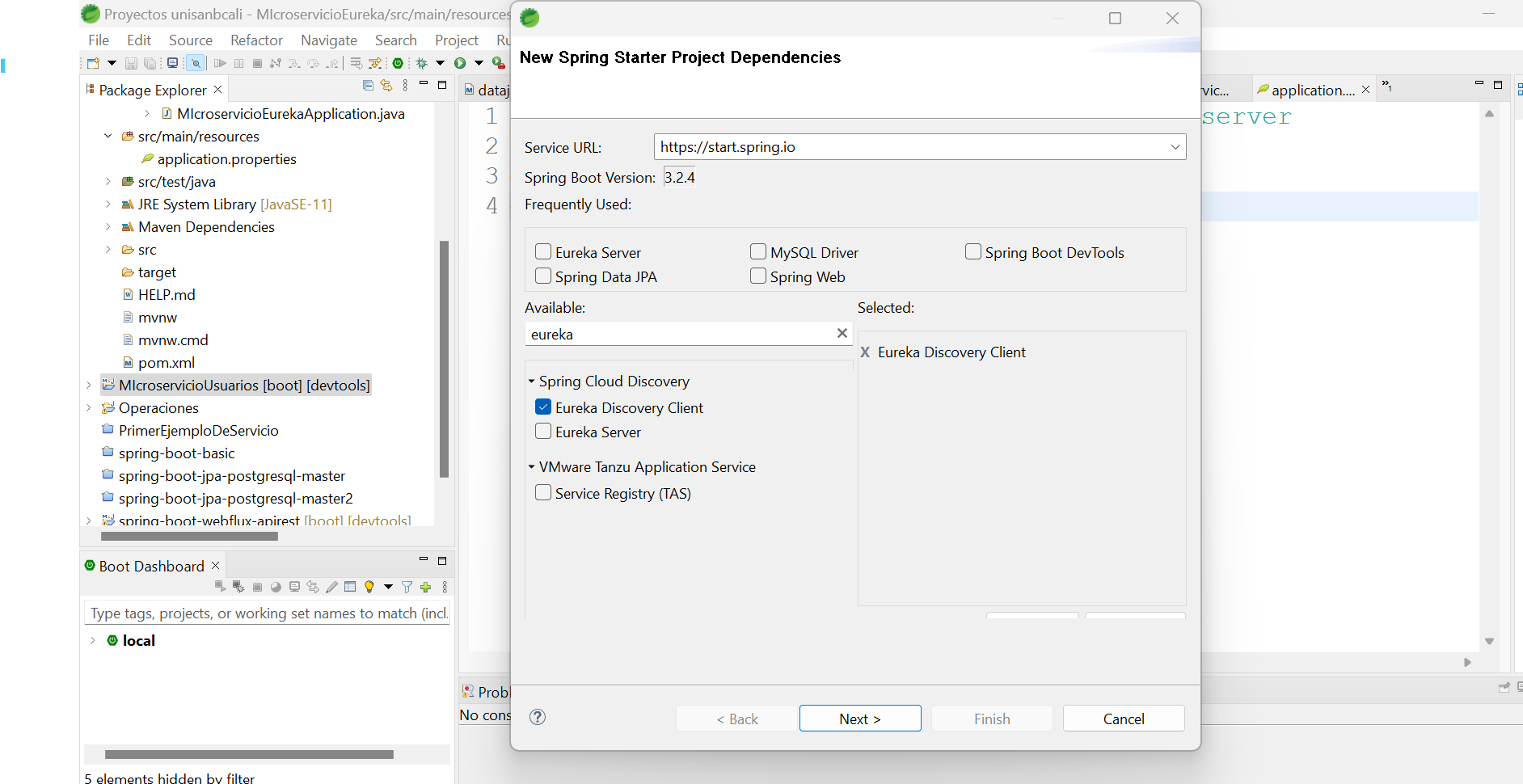
Y eso es todo de momento con el servicio de Eureka

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

Evidencia Mariana Cruz

Una captura de pantalla de una red social

Descripción generada automáticamente