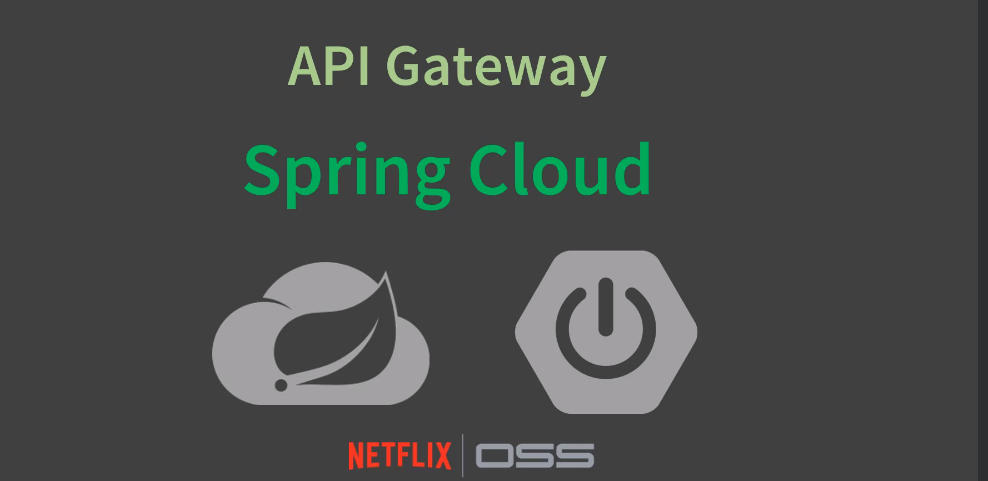
Actividades a desarrollar:

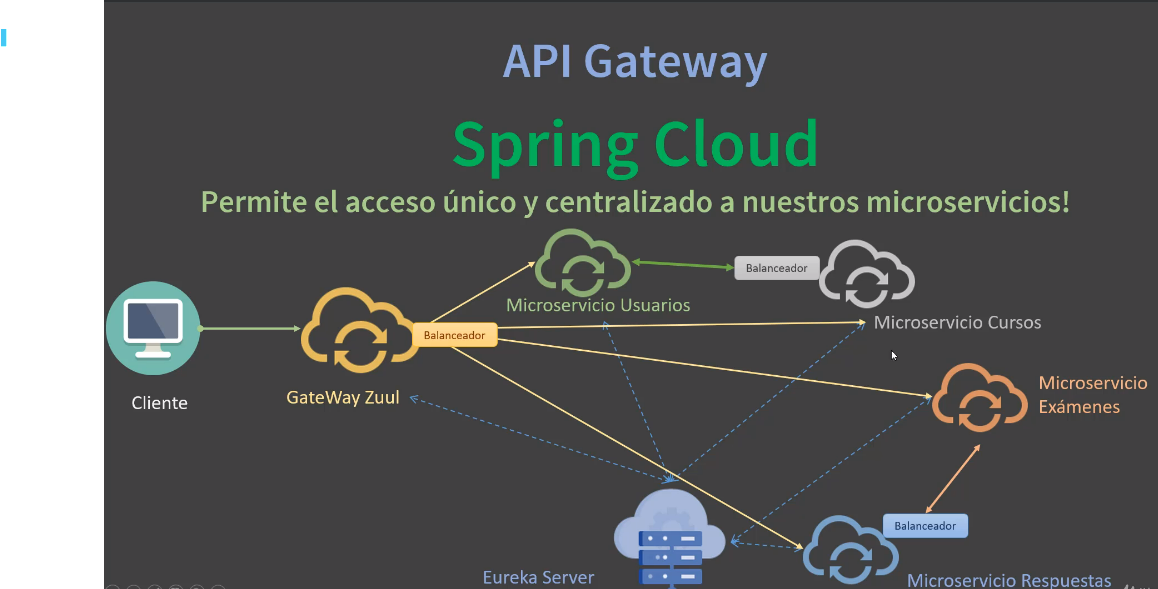
CREACIÓN DE PUERTA DE ENLACE GATEWAY

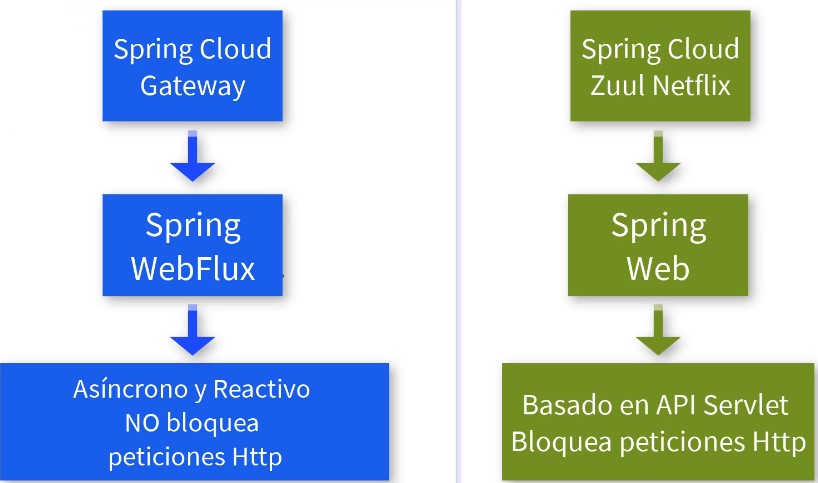


1.1 Buscar y poner en esta parte el concepto de Spring Cloud Gateway

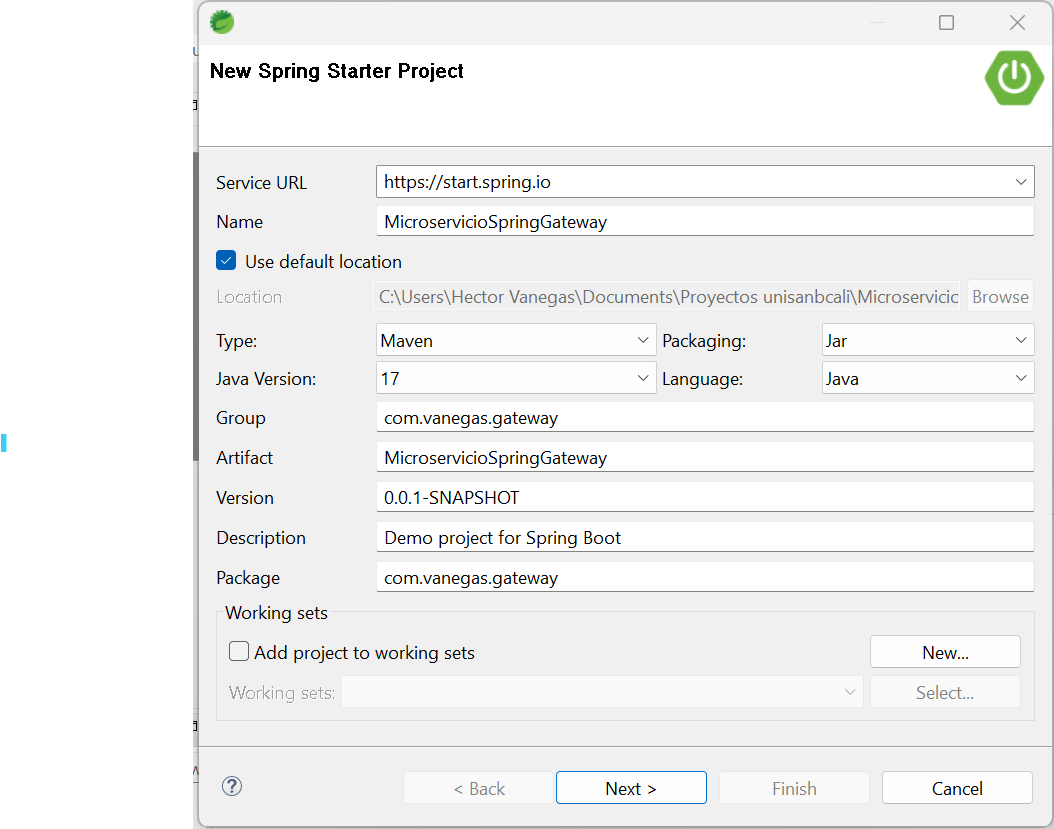
Texto

Descripción generada automáticamente

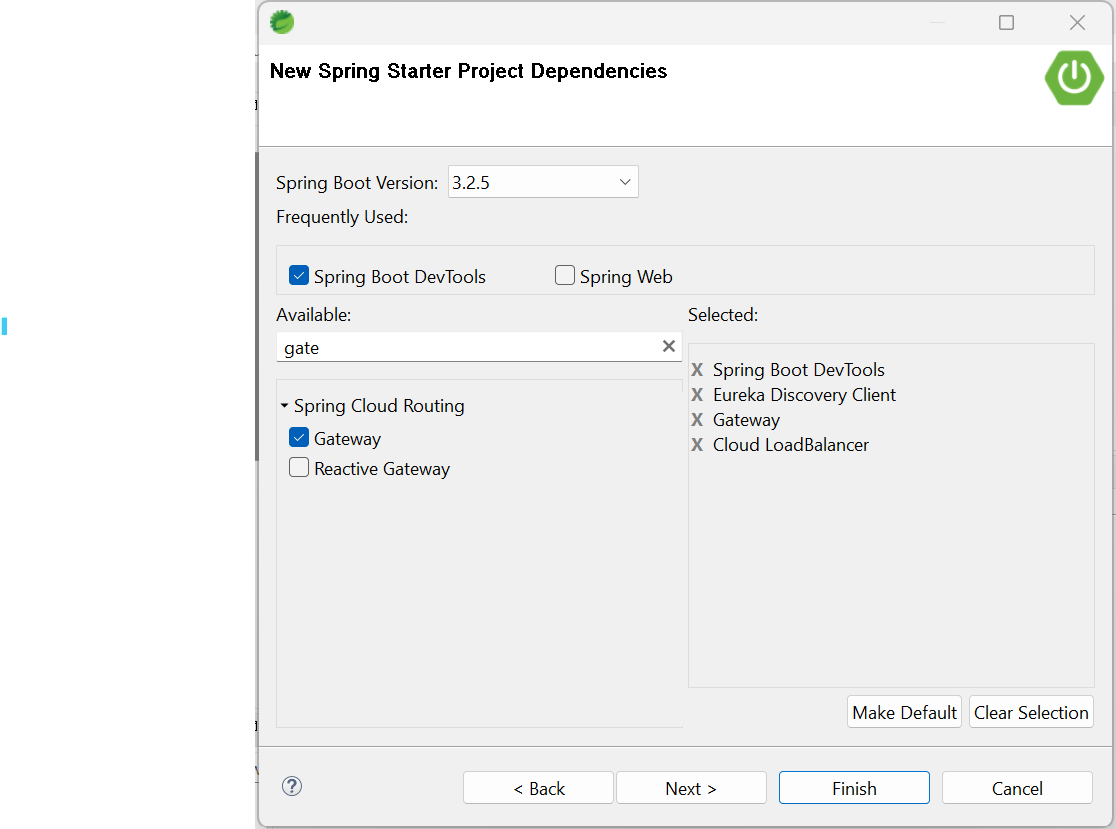




1.2 Se debe crear un nuevo servicio que servirá como apigateway

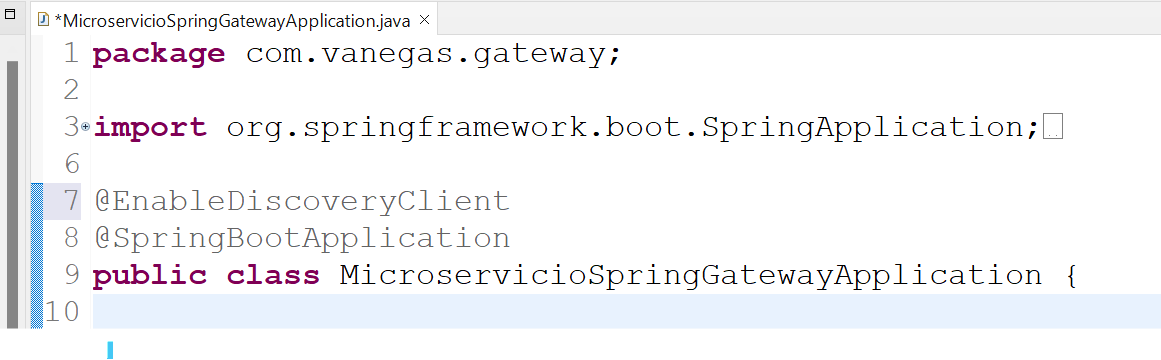


1.3 Las dependencias a usar son las siguientes:

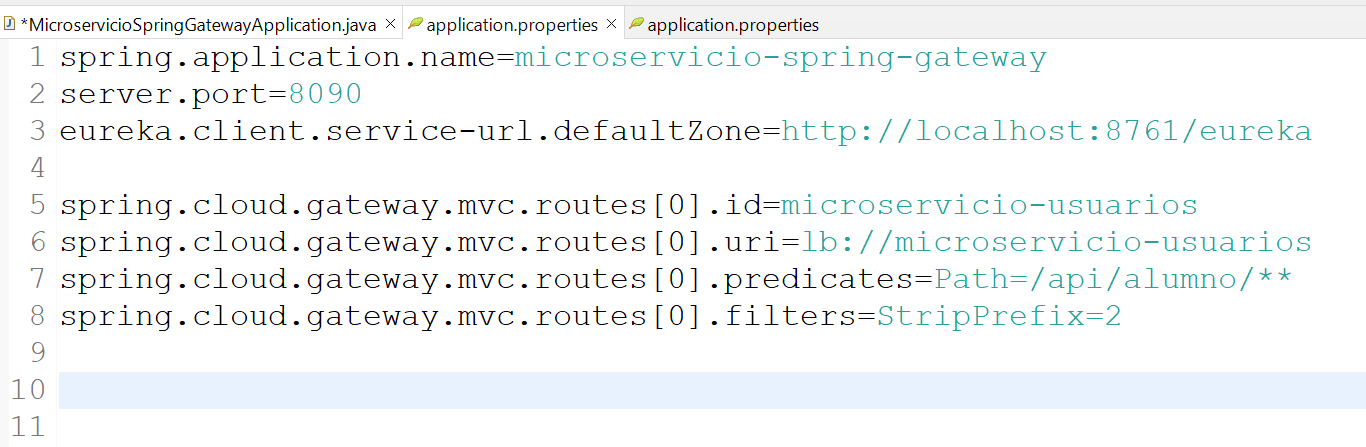


1.4 Luego de haber creado el proyecto en la clase principal colocamos a nivel de la clase la anotación

@EnableDiscoveryClient



1.5 En el archivo de propiedades colocamos las siguientes propiedades relacionadas con Gateway



1.6 con la orientación del profesor colocar en este parte del documento el significado de cada una de las propiedades del anterior archivo

R//

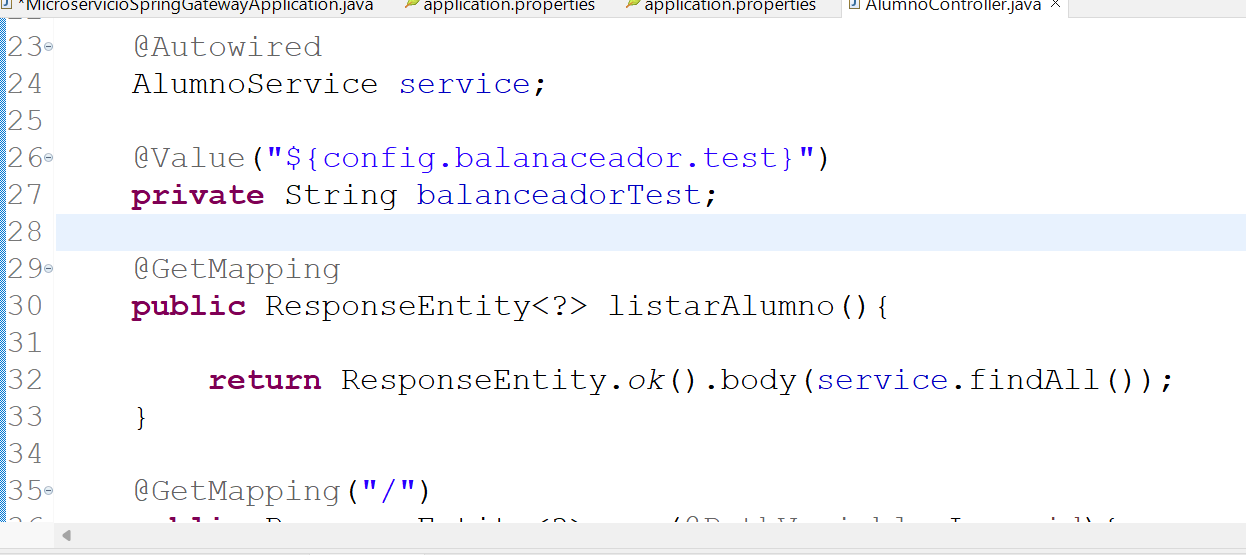
1. **spring.application.name=MicroservicioSpringGateway**: Esta propiedad establece el nombre de la aplicación Spring Boot. Es útil para identificar la aplicación en el entorno de despliegue y para propósitos de registro y monitoreo.
2. **server.port=8090**: Esta propiedad define el puerto en el que la aplicación Spring Boot escuchará las solicitudes entrantes. En este caso, la aplicación estará disponible en el puerto 8090.
3. **eureka.client.service-url.defaultZone=** <http://localhost:8761/eureka>: Esta propiedad configura la URL del servidor de Eureka. Eureka es un servicio de registro y descubrimiento utilizado en entornos de microservicios. Aquí, se está configurando para que la aplicación se registre en el servidor de Eureka ubicado en **http://localhost:8761**.
4. **spring.cloud.gateway.mvc.routes[0].id=microservicio-usuarios**: Esto configura un enrutador en Spring Cloud Gateway. Está definiendo un identificador para la ruta, que es útil para la configuración y el registro.
5. **spring.cloud.gateway.mvc.routes[0].uri=lb://microservicio-usuarios**: Esta propiedad especifica la URI de destino a la que se redirigen las solicitudes que coinciden con esta ruta. En este caso, las solicitudes se enviarán al servicio denominado "microservicio-usuarios".
6. **spring.cloud.gateway.mvc.routes[0].predicates=Path=/api/alumno//\***: Aquí se definen los predicados para la ruta. En este caso, la ruta se activará cuando la URL de la solicitud coincida con el patrón especificado (**/api/alumno/\*\***).
7. **spring.cloud.gateway.mvc.routes[0].filters=StripPrefix=2**: Los filtros se utilizan para modificar las solicitudes o respuestas mientras pasan por la puerta de enlace. En este caso, se está utilizando el filtro **StripPrefix** para quitar los primeros dos segmentos del path de la URL antes de que la solicitud se enruté al servicio de destino. Por ejemplo, si la URL de la solicitud es **/api/alumno/xyz**, después de aplicar este filtro, se convertirá en **/xyz** antes de ser enviada al servicio "microservicio-usuarios".

1.7. Para permitir la comunicación y exposición del servicio usuarios a través del balanceador de carga, colocamos en el archivo de propiedades de ese servicio la siguiente anotación, lo cual corresponde una variable de entorno:

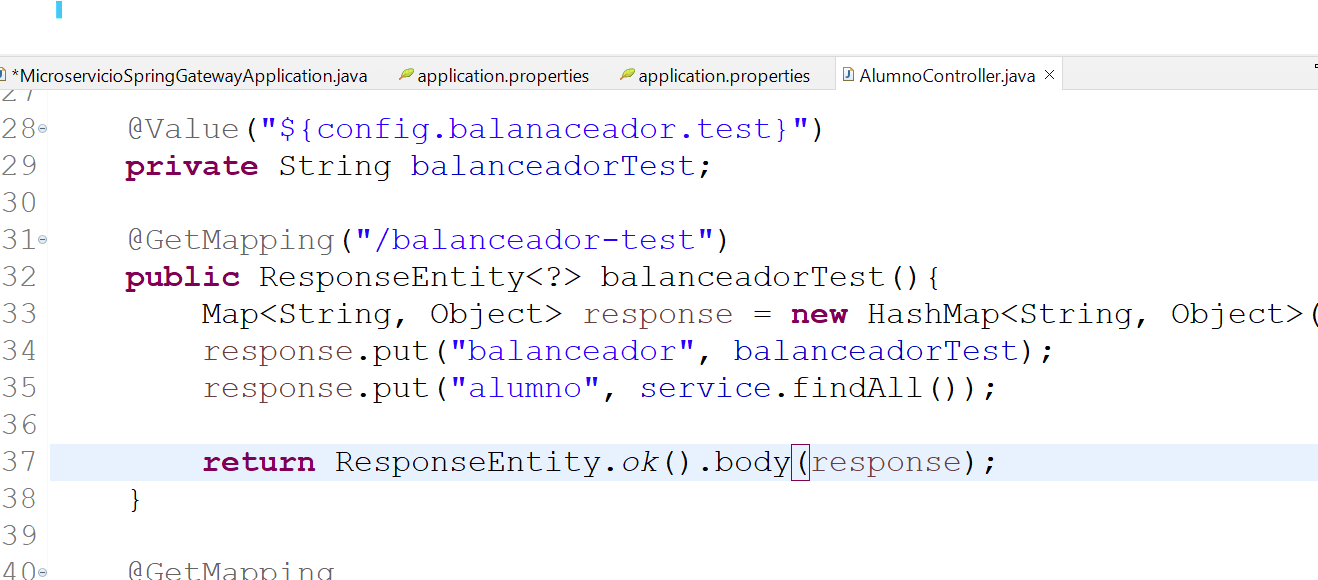
config.balanaceador.test=${BALANCEADOR\_TEST: string por defecto}

Esta variable nos permitirá que instancia esta siendo invocada por el balanceador

1.8 Luego inyectamos esa variable en el controlador



1.9 Ahora creamos un método de prueba dentro del controlador que nos retorne el valor de esa variable de entorno junto a la lista de alumnos dentro de un HasMap , de la siguiente forma



El anterior método nos permite validar el funcionamiento del balanceador de carga.

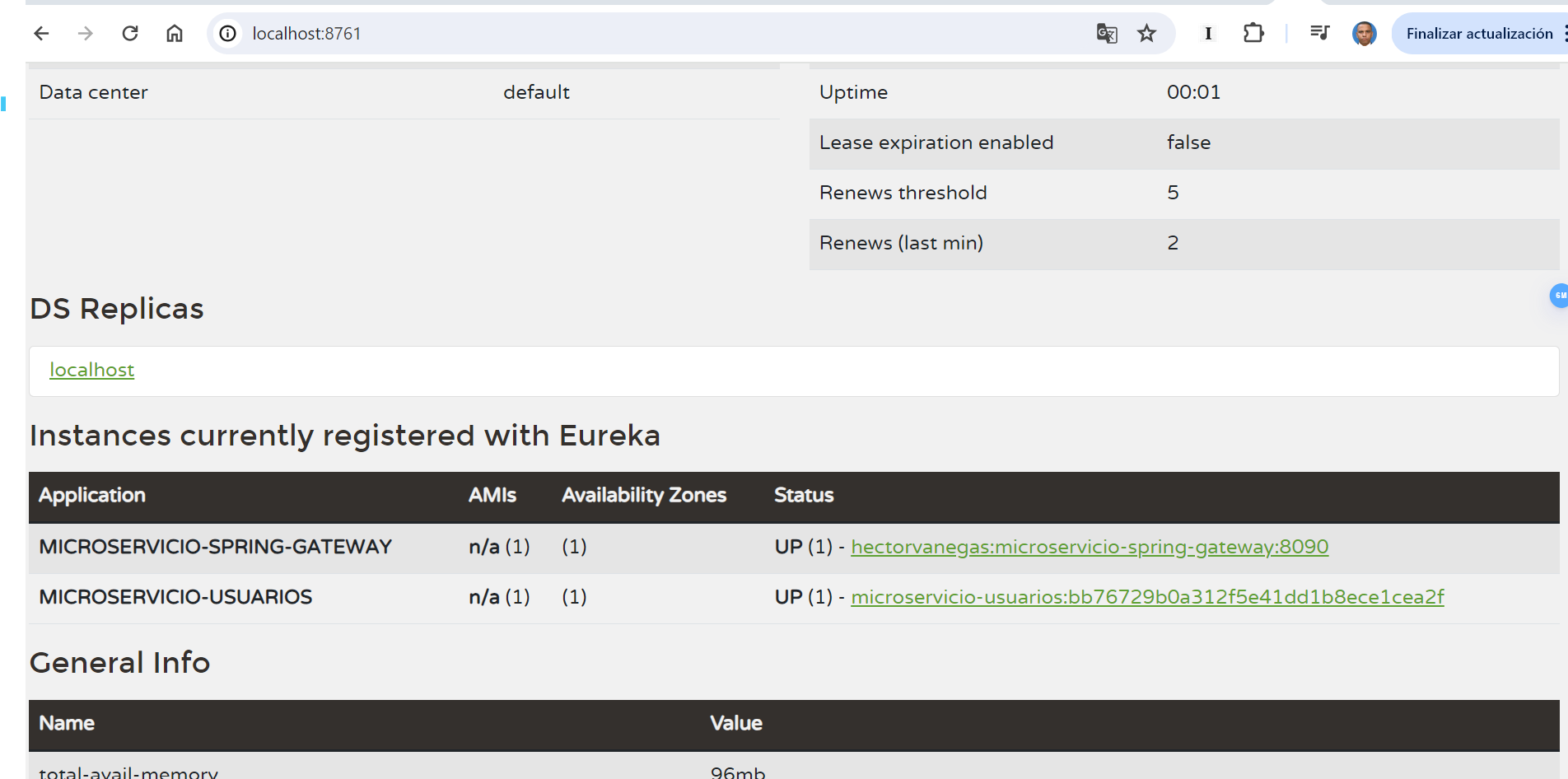
2.0 para probar los servicio levantamos primero a eureka server, luego el microservicio usuarios y por ultimo el Gateway.

Podemos ver como están corriendo los 3 servicios

Aplicación

Descripción generada automáticamente

Y vamos a eureka para ver que realmente estén registrados alli



3. Vamos a crear un componente genérico que nos va a permitir generar reuso de código.

Antes de continuar investiga y ponlo en esta parte de la actividad que son los genéricos en java , sus ventajas y usos.

R// Los genéricos en Java son una característica del lenguaje que permite crear clases, interfaces y métodos que puedan funcionar con diferentes tipos de datos. Esto permite escribir código que sea más flexible, reutilizable y seguro en términos de tipo.

Las principales ventajas de utilizar genéricos en Java son:

1. **Reutilización del código**: Los genéricos permiten escribir código que puede ser reutilizado con diferentes tipos de datos, evitando la necesidad de escribir múltiples versiones del mismo código para diferentes tipos.
2. **Seguridad de tipos (type safety)**: Los genéricos proporcionan seguridad de tipos en tiempo de compilación. Esto significa que los errores de tipo se detectan en tiempo de compilación en lugar de en tiempo de ejecución, lo que ayuda a evitar errores relacionados con tipos de datos en el código.
3. **Abstracción de datos**: Los genéricos permiten escribir código que es independiente del tipo de datos con el que trabaja, lo que proporciona una mayor abstracción y modularidad en el diseño del programa.
4. **Mejor legibilidad del código**: El uso de genéricos puede hacer que el código sea más claro y legible, ya que los tipos de datos se especifican de manera explícita en el código, lo que hace que la intención del programador sea más evidente.

Los genéricos se utilizan en una variedad de situaciones en Java, incluyendo:

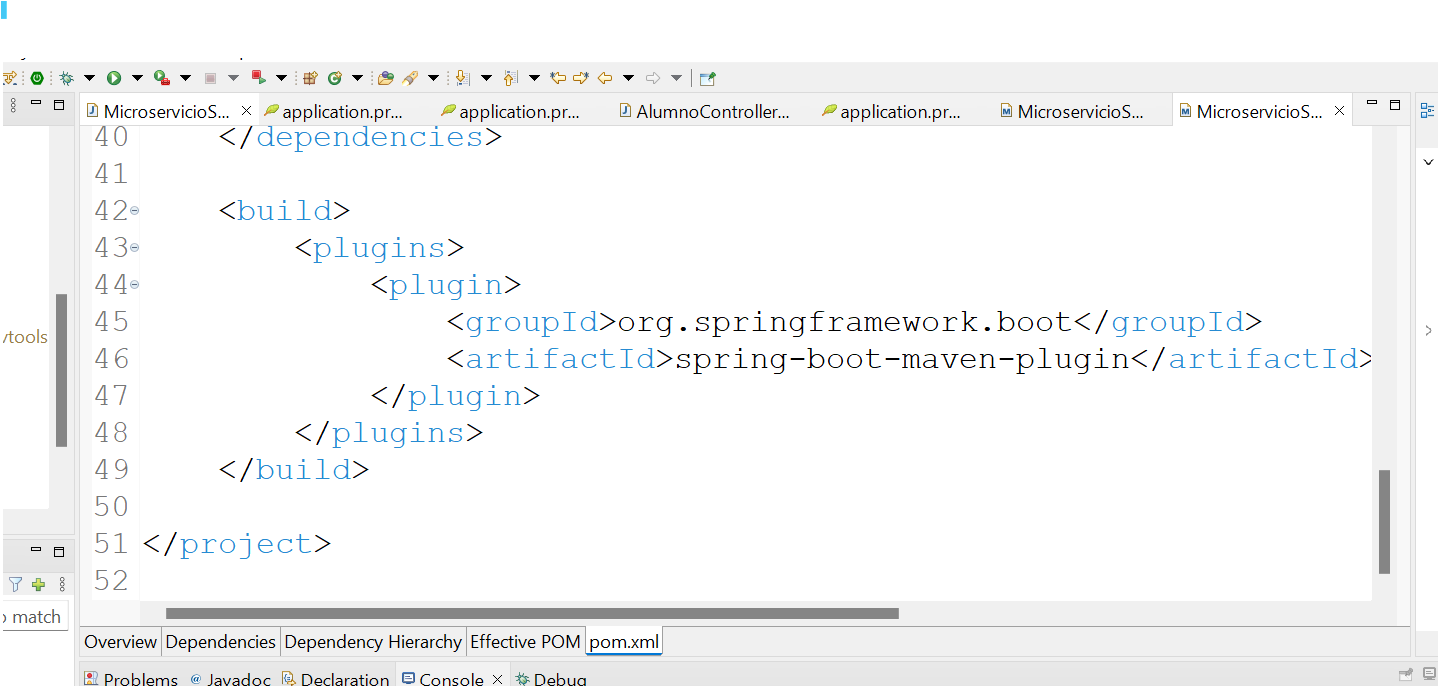
* **Colecciones**: Las colecciones genéricas en Java, como ArrayList, HashMap, etc., permiten almacenar y manipular elementos de cualquier tipo de manera segura. Por ejemplo, **ArrayList<Integer>** puede almacenar solo enteros, **ArrayList<String>** puede almacenar solo cadenas, etc.
* **Clases y métodos**: Se pueden definir clases y métodos genéricos que funcionen con tipos de datos desconocidos en tiempo de compilación. Esto es útil para escribir algoritmos y estructuras de datos que sean independientes del tipo de datos con el que trabajan.
* **Interfaces funcionales y lambdas**: El uso de genéricos en interfaces funcionales y lambdas permite escribir código más genérico y flexible. Por ejemplo, se pueden definir interfaces funcionales genéricas como **Predicate<T>** o **Function<T, R>** que trabajen con diferentes tipos de datos.

3.1 Creamos un nuevo proyecto con las siguientes dependencias

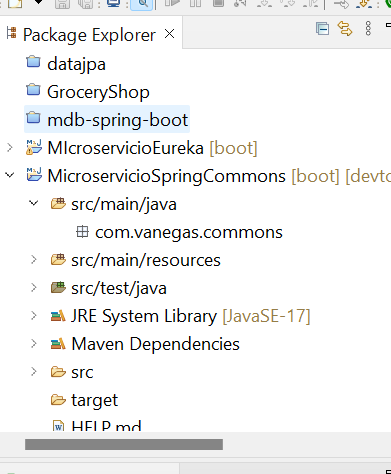
Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

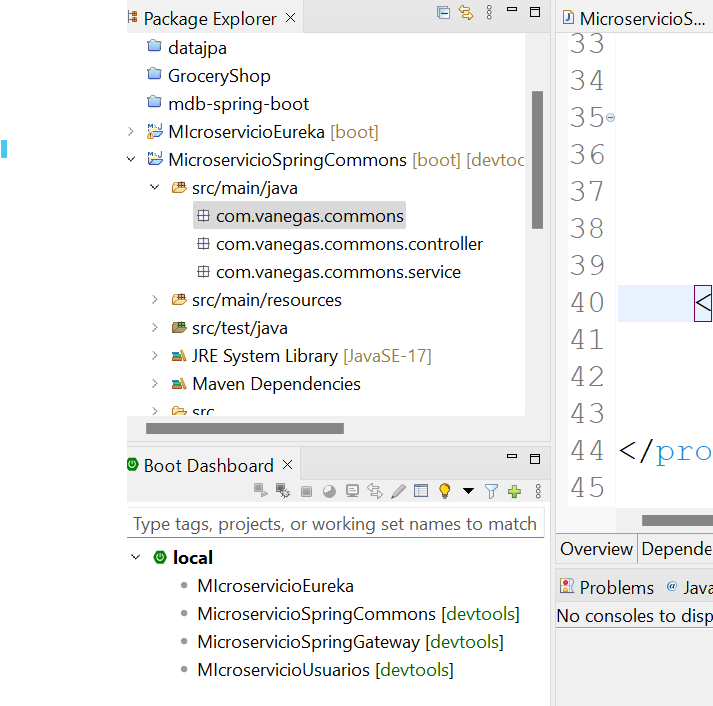
3.2 Luego de creado el proyecto nos dirigimos al pom.xml y eliminamos el plugin de Maven como se ve a continuación:



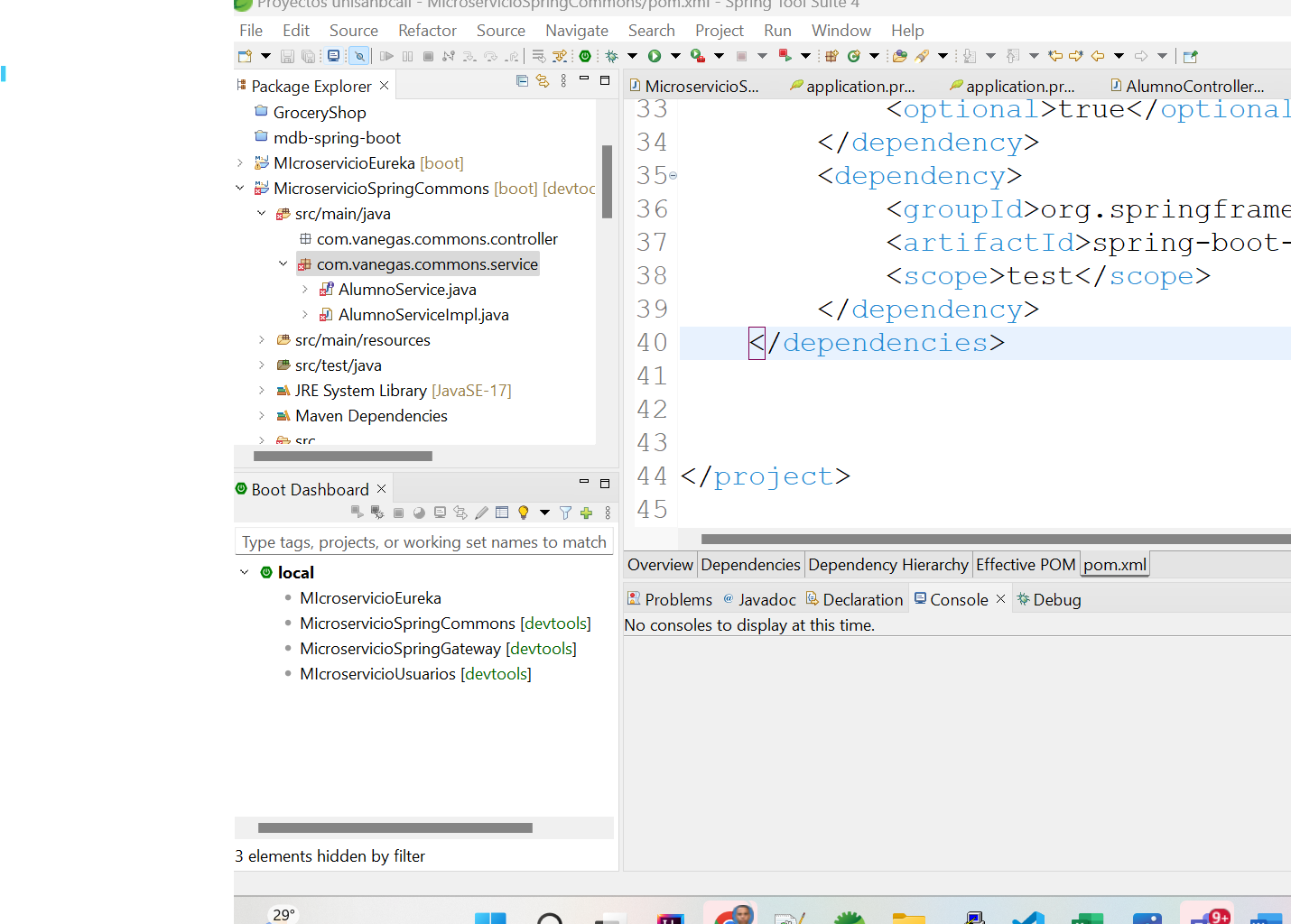
3.3 Luego eliminamos la clase principal ya que lo que se quiere en construir una librería



3.4 Luego creamos dos paquetes uno con el path de service y el otro de controller



3.5 Dentro del paquete service copiamos las clases AlumnoService y AlumnoServiceImpl que tenemos en el proyecto MicroservicioUsuarios

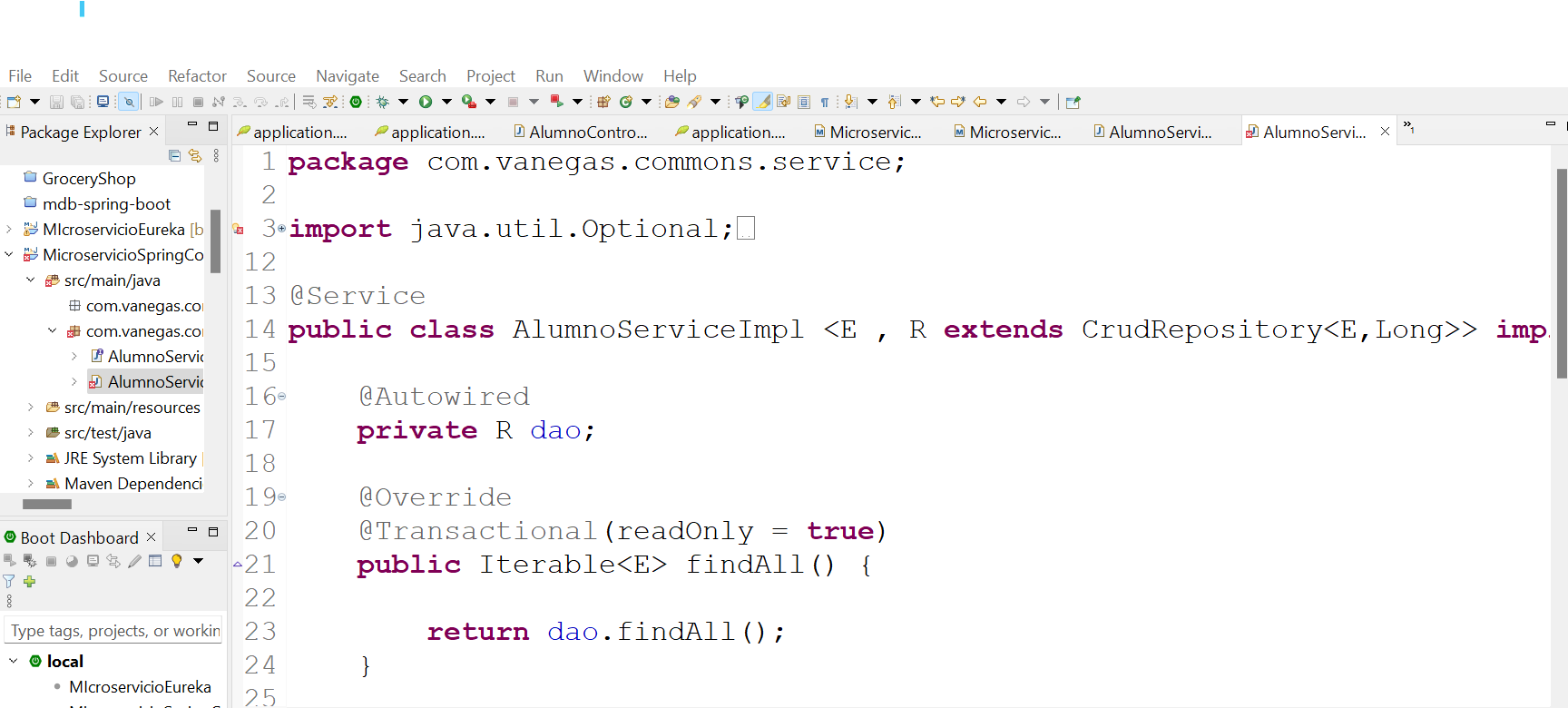


3.6 Vamos a hacer algunos cambios en la interface, quietamos lo que tenga la entidad Alumno y trabajaremos con el api generic de java y quedaría así:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

3.7 ahora modificamos la implementación del service de la siguiente manera



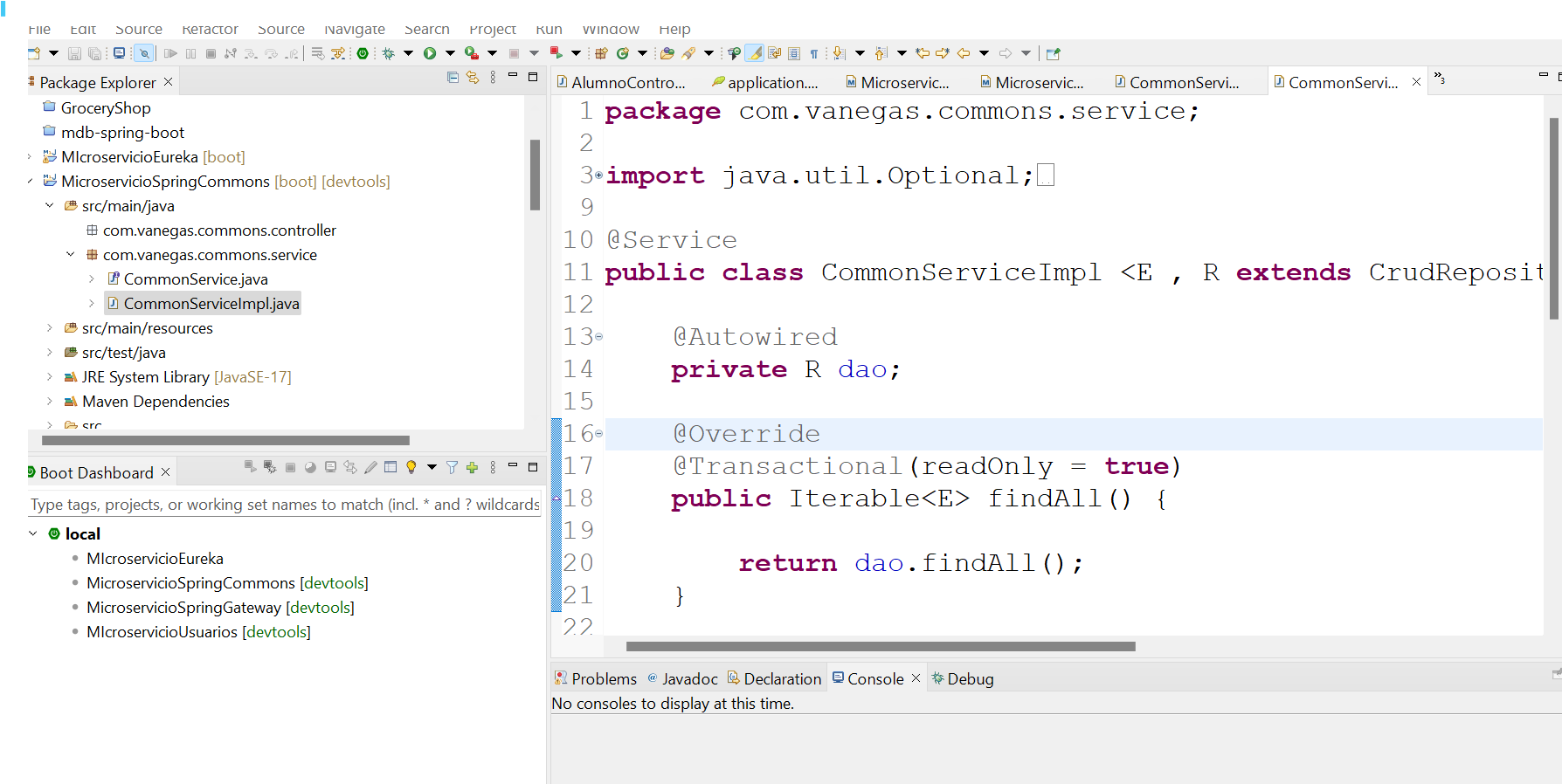
3.8 Cambiamos la inyección del componente repository para que sea genérico , allí se puede ver que estamos usando dos genéricos la E que corresponde a la entidad de la bd y la R que corresponde a la capa de acceso a datos.

Y en todas las partes de esta clase donde se encuentre la clase Alumno lo cambiamos por E y la palabra alumno por entity

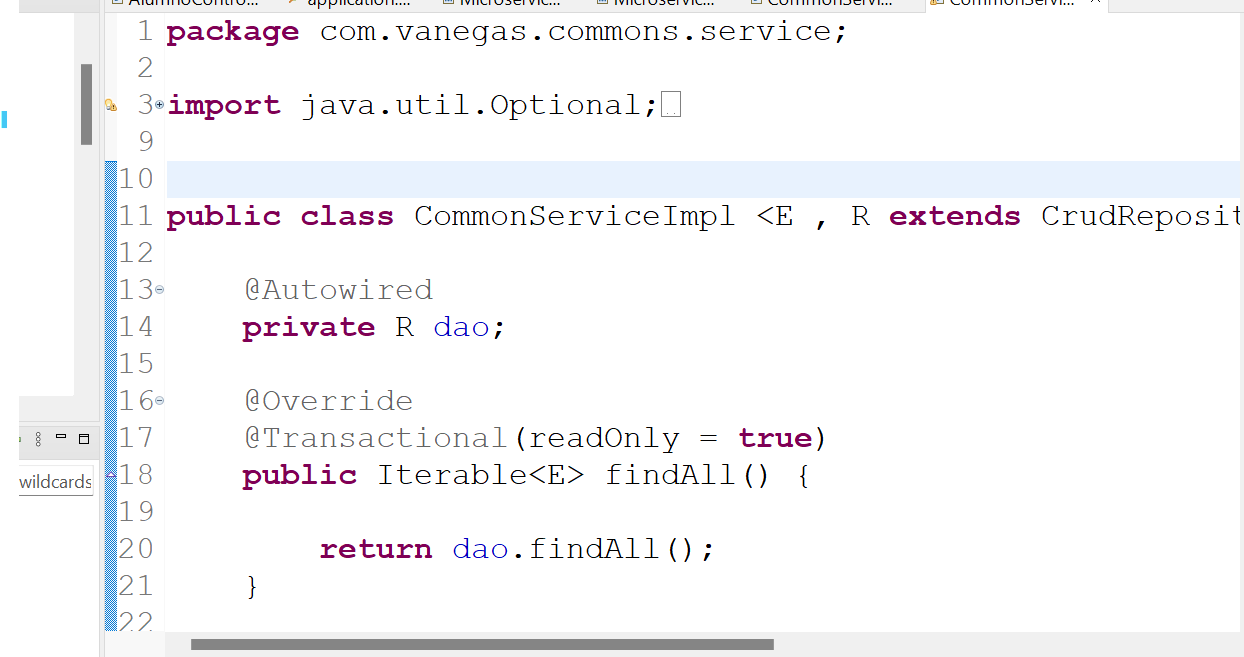
Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

3.9 Y luego por buenas practicas renombramos las clases que hemos modificado con CommonService y CommonServiceImpl



Y como no es un componente que vamos a inyectar en la clase CommonServiceImpl quitamos el decorador @Service



3.10 ahora se tiene que inyectar este servicio en el microservicio en el de usuarios para ello copiamos el

groupId , artifactId y version

Estas dependencias del servicio de Common , deben ser llevadas al servicio de Usuarios como una dependencia y de momento dejamos allí