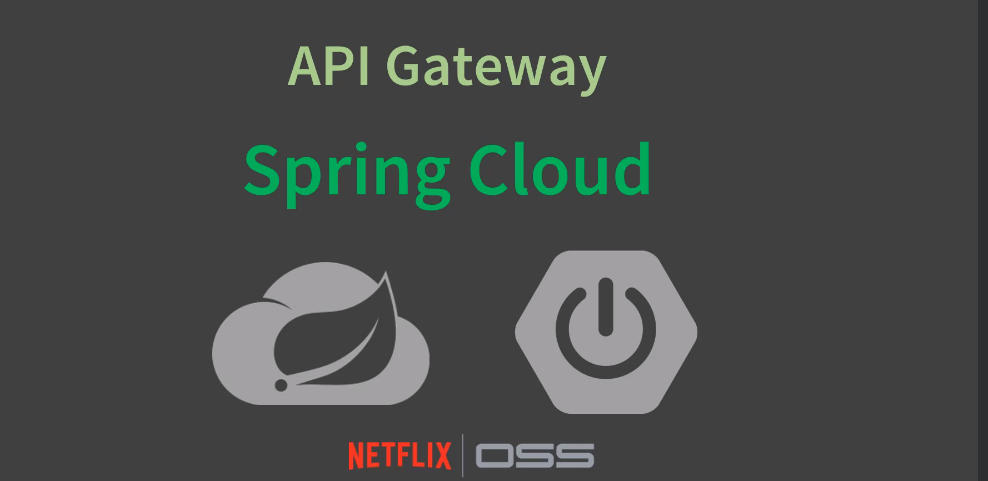
Actividades a desarrollar:

Desarrollado por Jurgen Sanclemente

CREACIÓN DE PUERTA DE ENLACE GATEWAY

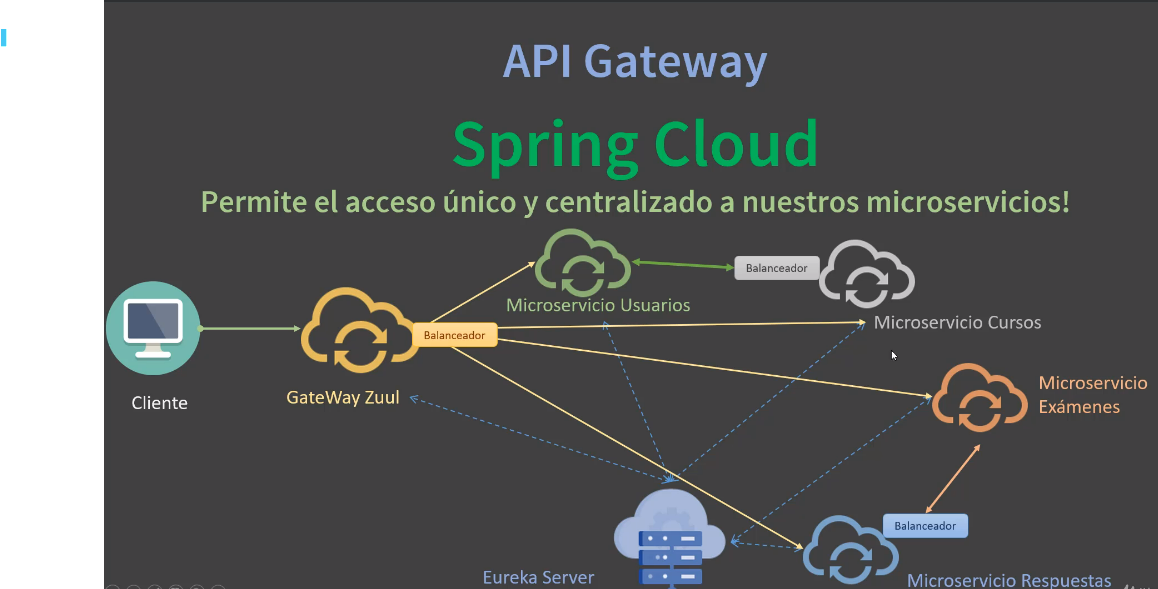


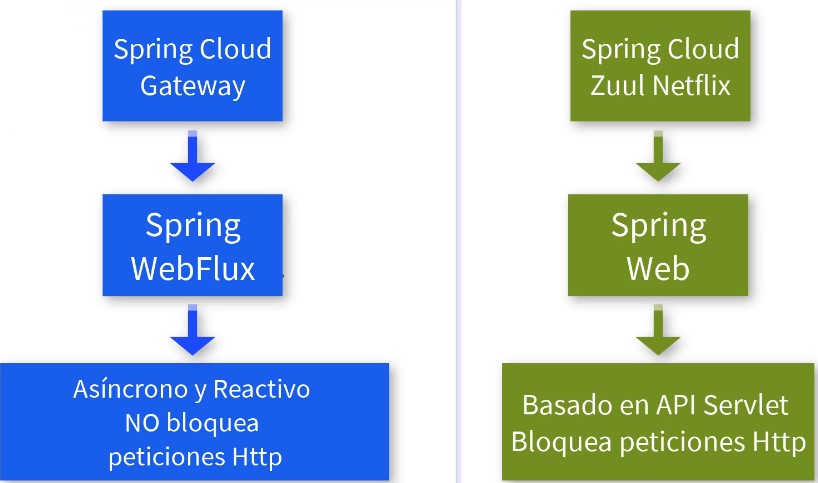
* 1. Buscar y poner en esta parte el concepto de Spring Cloud Gateway

Es un proyecto de spring que proporciona una forma de enrutamiento y filtrado de soluciones para aplicaciones basadas en la nube. Ofrece características como: Enrutamiento dinámico (Cambiar rutas sin reiniciar aplicación ), Filtrado de soluciones ( Modificar solicitudes HTTP entrantes o salientes), Integración con servicios de descubrimiento como Eureka, Balanceo de carga.

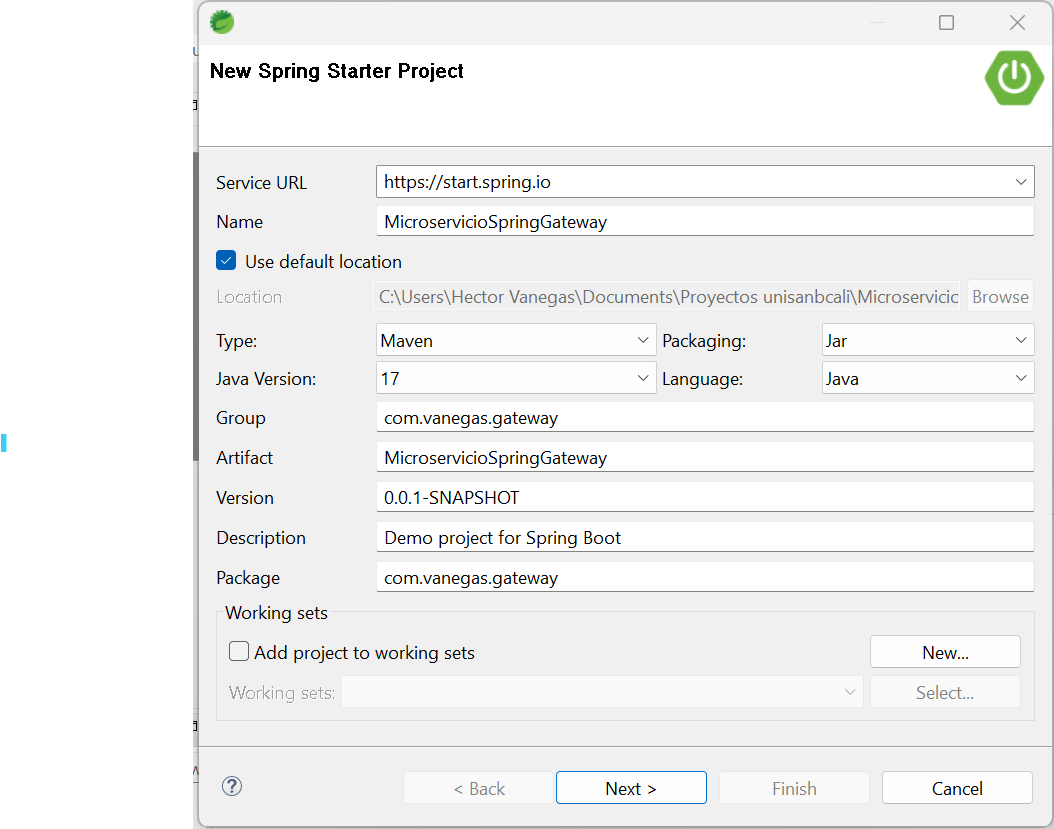
Texto

Descripción generada automáticamente

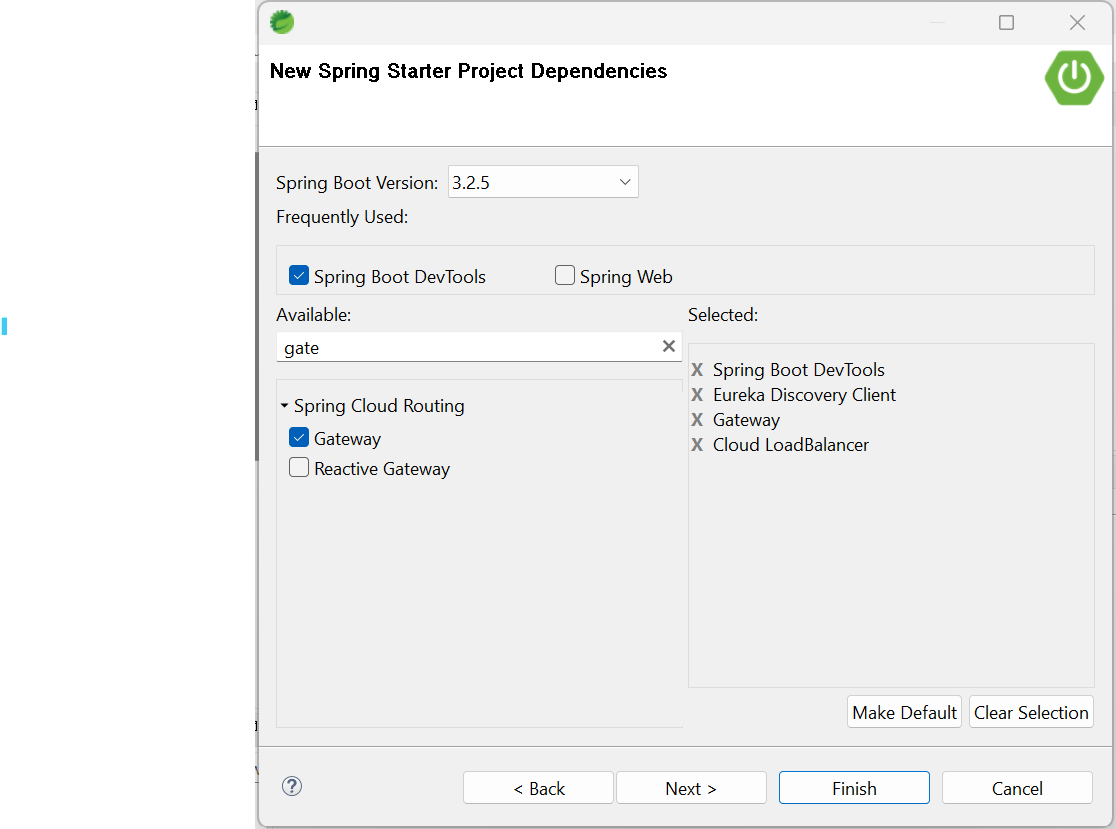




1.2 Se debe crear un nuevo servicio que servirá como apigateway

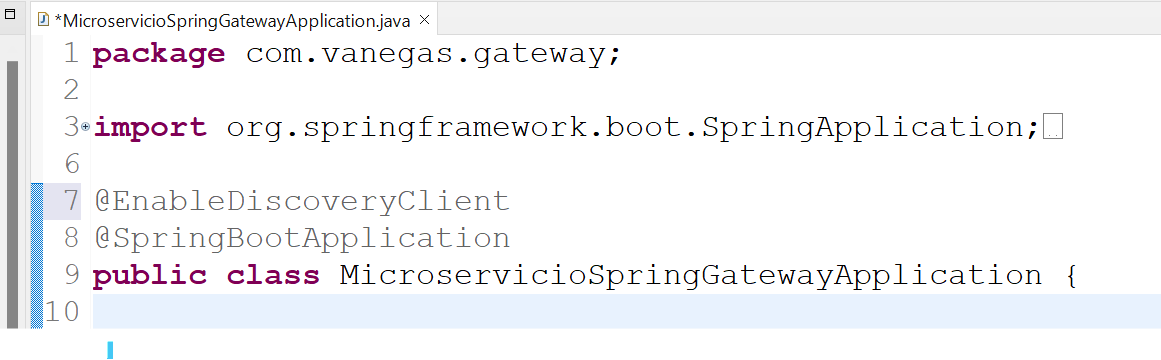


1.3 Las dependencias a usar son las siguientes:

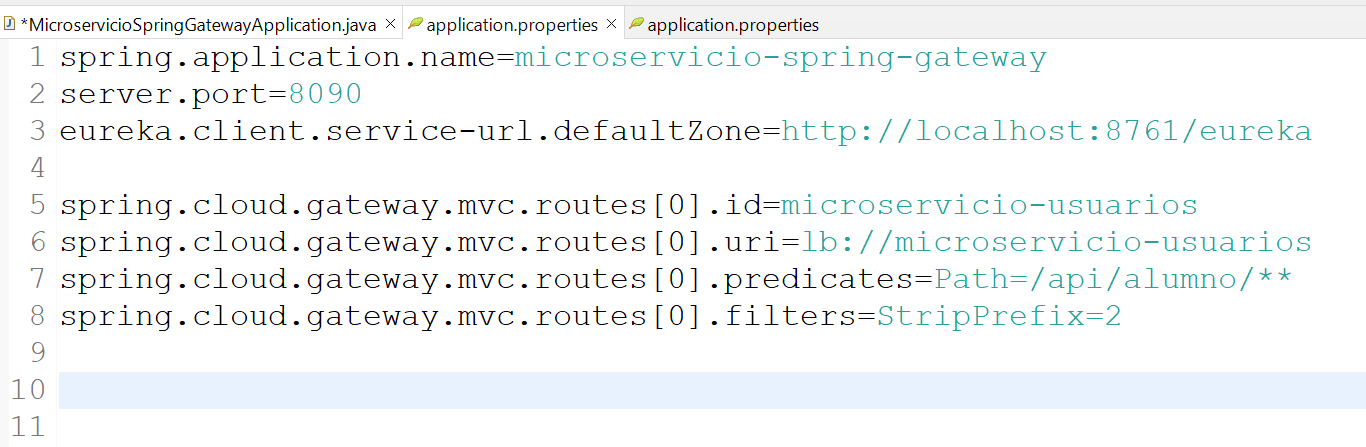


1.4 Luego de haber creado el proyecto en la clase principal colocamos a nivel de la clase la anotación

@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

**eureka.client.service-url.defaultZone**: Esta propiedad se utiliza para configurar la ubicación del servidor de Eureka en tu aplicación. Eureka es un servicio de descubrimiento que permite a las aplicaciones cliente encontrar y comunicarse entre sí sin necesidad de conocer las direcciones exactas de red.

spring.cloud.gateway.routes[0].id: Esta propiedad define un identificador único para una ruta en Spring Cloud Gateway.

spring.cloud.gateway.routes[0].uri: Aquí se especifica la URI de destino a la que se redirigirán las solicitudes que coincidan con esta ruta en particular

spring.cloud.gateway.routes[0].predicates: Los predicados definen las condiciones bajo las cuales una ruta específica debe aplicarse.

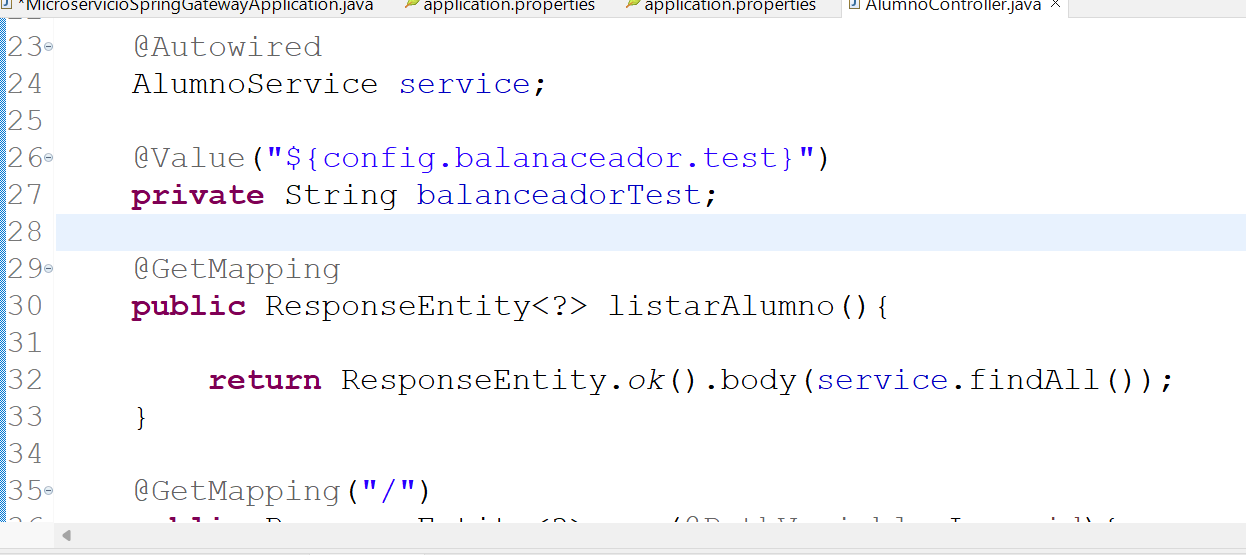
spring.cloud.gateway.routes[0].filters: Los filtros son transformaciones o acciones que se aplican a las solicitudes o respuestas antes o después de que se enrutaron a su destino.

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:

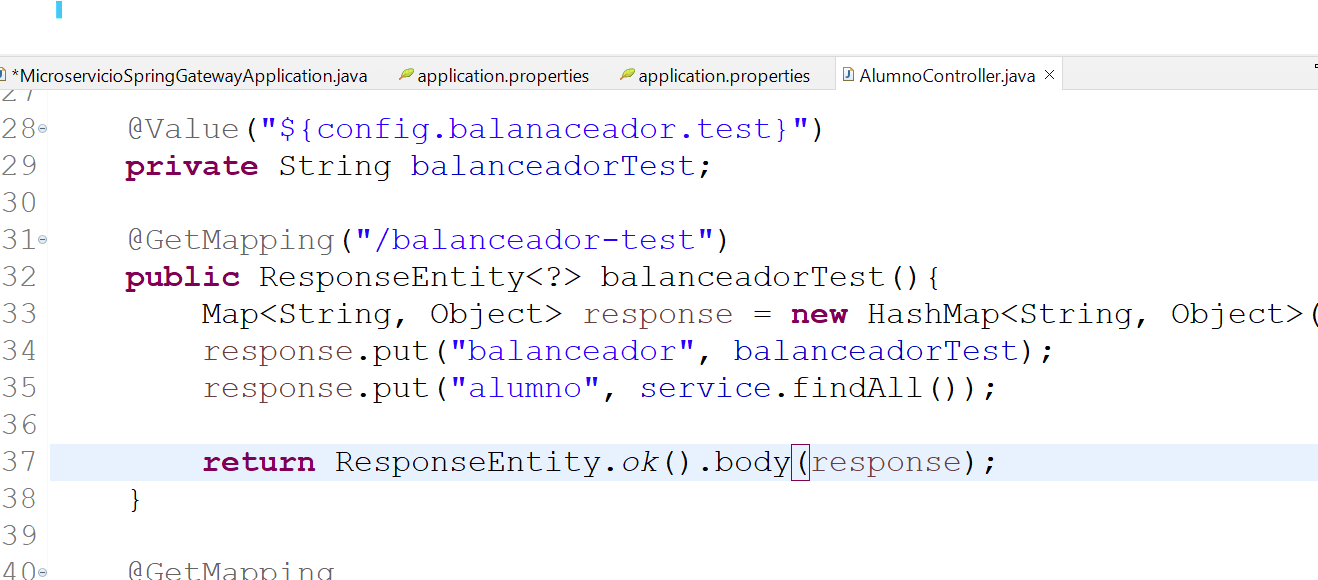
config.balanaceador.test=${BALANCEADOR\_TEST: string por defecto}

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.

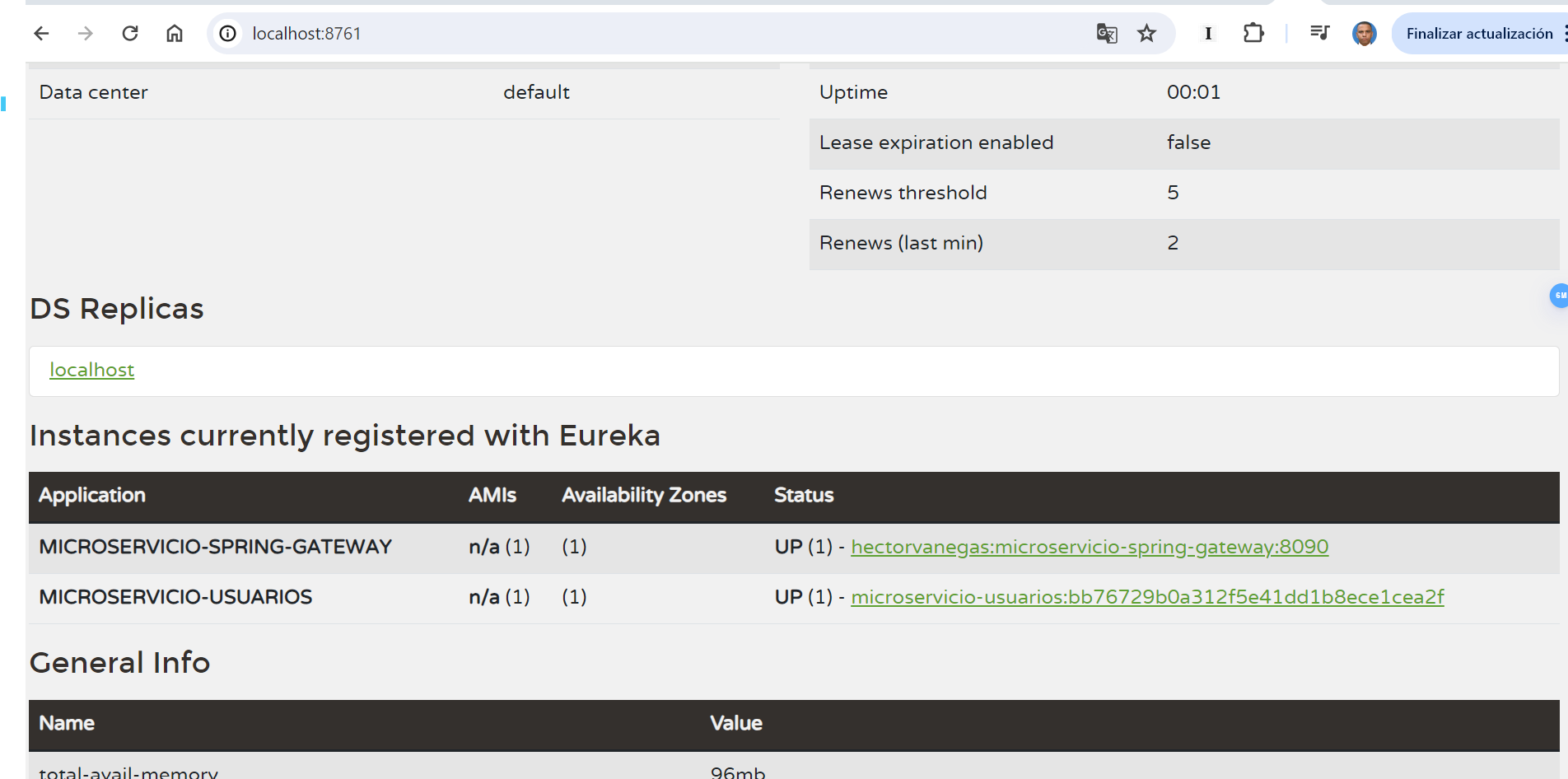
2.0 para probar los servicio levantamos primero a eureka server, luego el microservicio usuarios y por ultimo el Gateway.

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.

Los genéricos son una característica que permite escribir clases y métodos que puedan trabajar con diferentes tipos de datos sin tener que realizar conversiones. Permiten parametrizar tipos. Un ejemplo:

Sin genéricos:

List strings = new ArrayList();

Strings.add(“Hola”);

String saludo = (String) strings.get(0);

En el ejemplo anterior, vemos que necesitamos un parseo, para que el dato sea un string, con genérico no hay necesidad de esto:

List<String> strings = new ArrayList();

Strings.add(“Hola”);

String saludo = strings.get(0);

El generico se implementó en la clase List, y esto permite que esta última, pueda recibir cualquier tipo de dato y trabajarlo desde una lista.

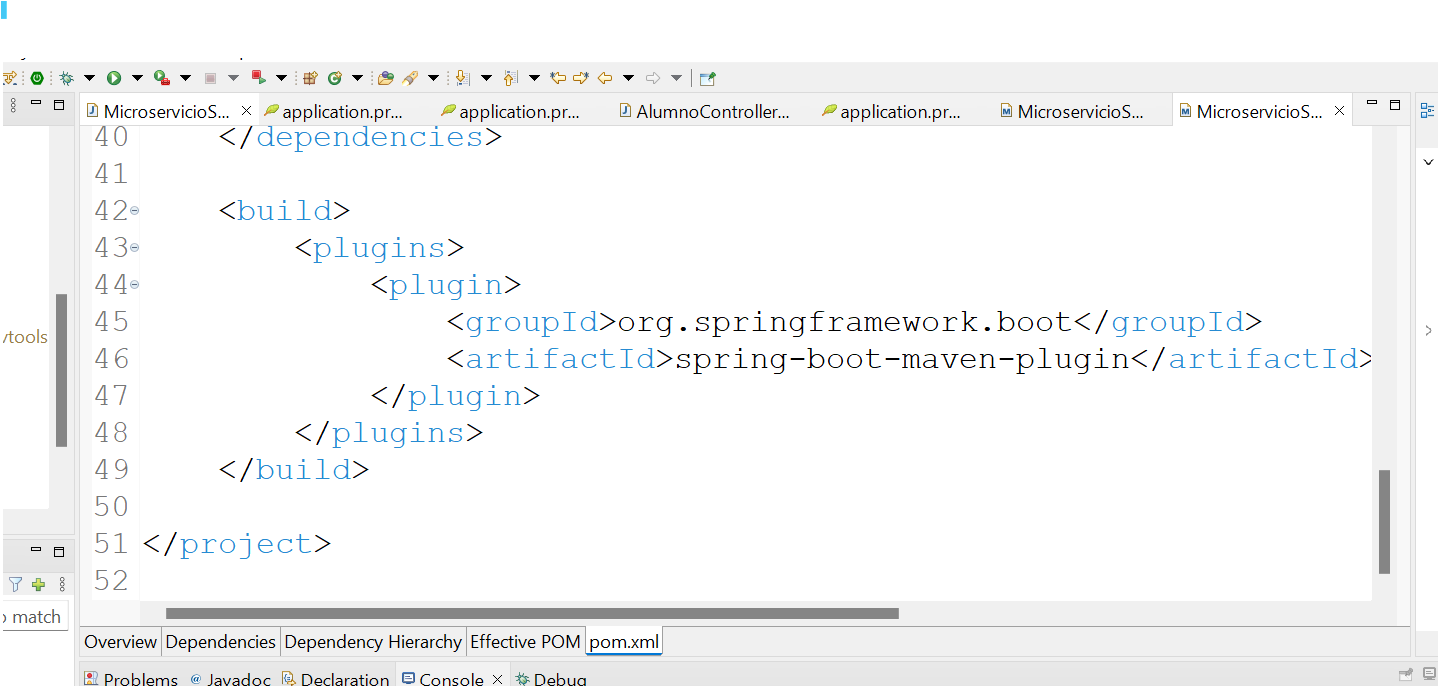
Las ventajas son: Seguridad de tipos, reutilización de código, abstracción mas fuerte, mejora del rendimiento.

3.1 Creamos un nuevo proyecto con las siguientes dependencias

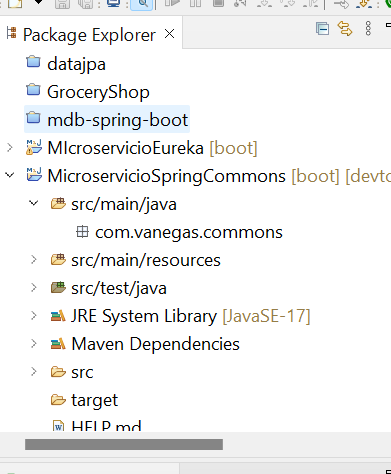
Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

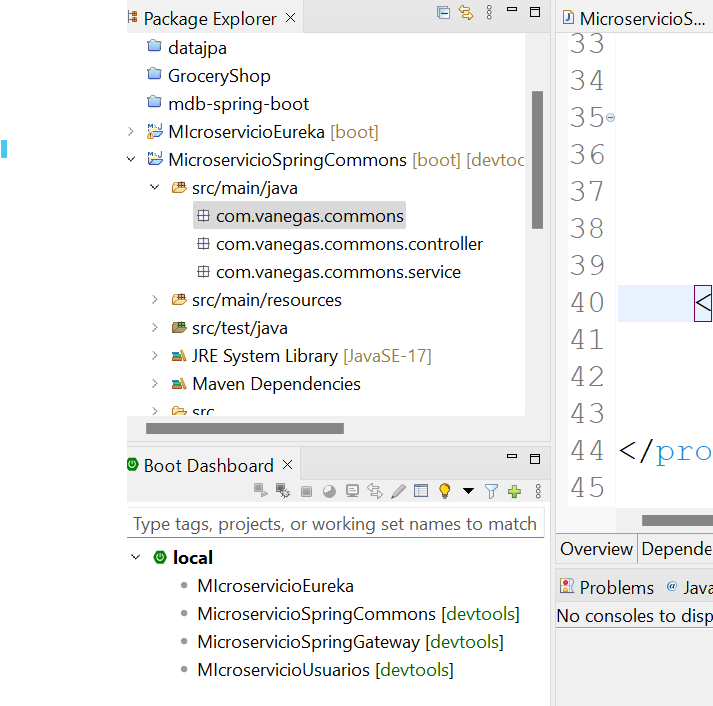
3.2 Luego de creado el proyecto nos dirigimos al pom.xml y eliminamos el plugin de Maven como se ve a continuación:



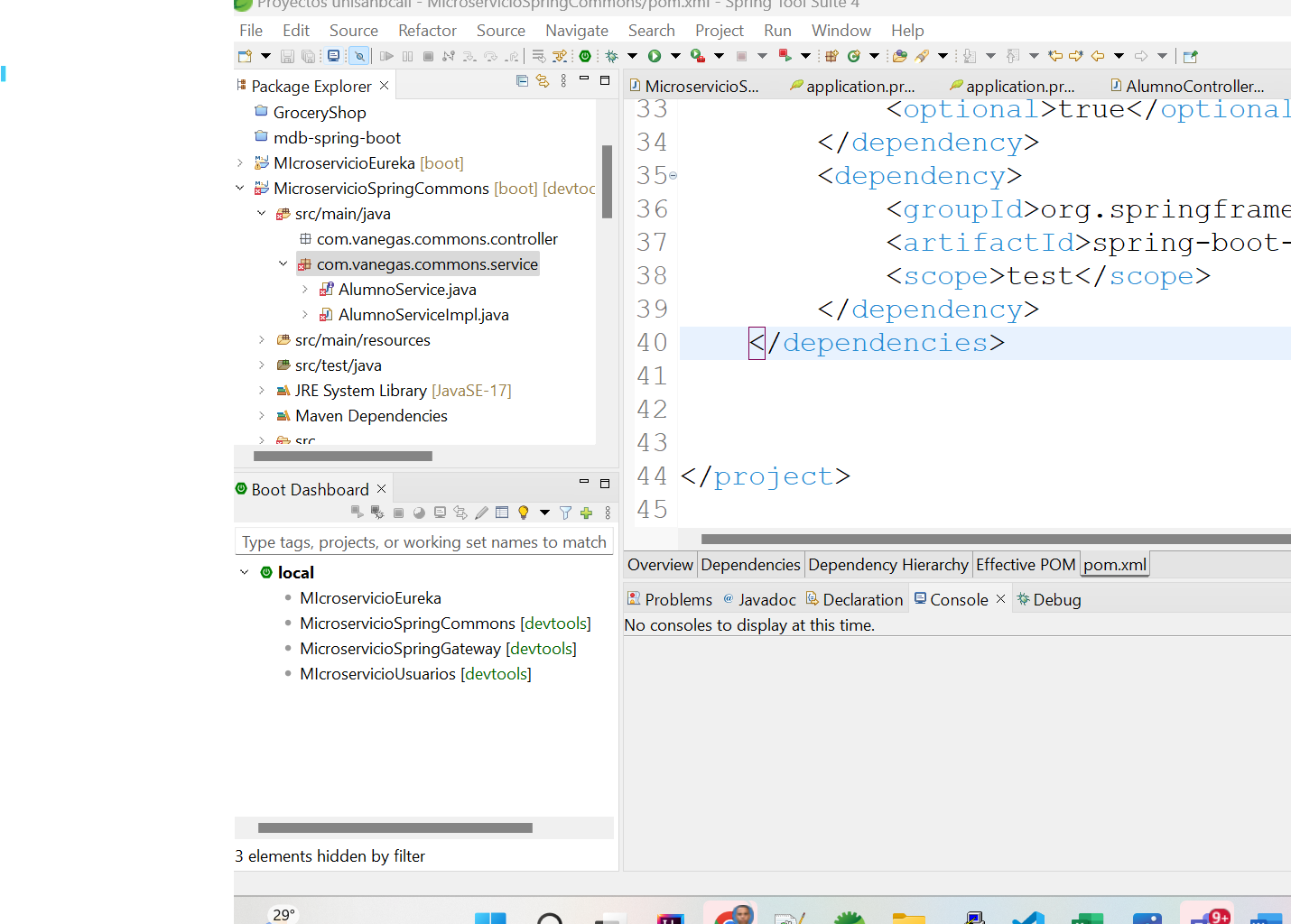
3.3 Luego eliminamos la clase principal ya que lo que se quiere en construir una librería



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

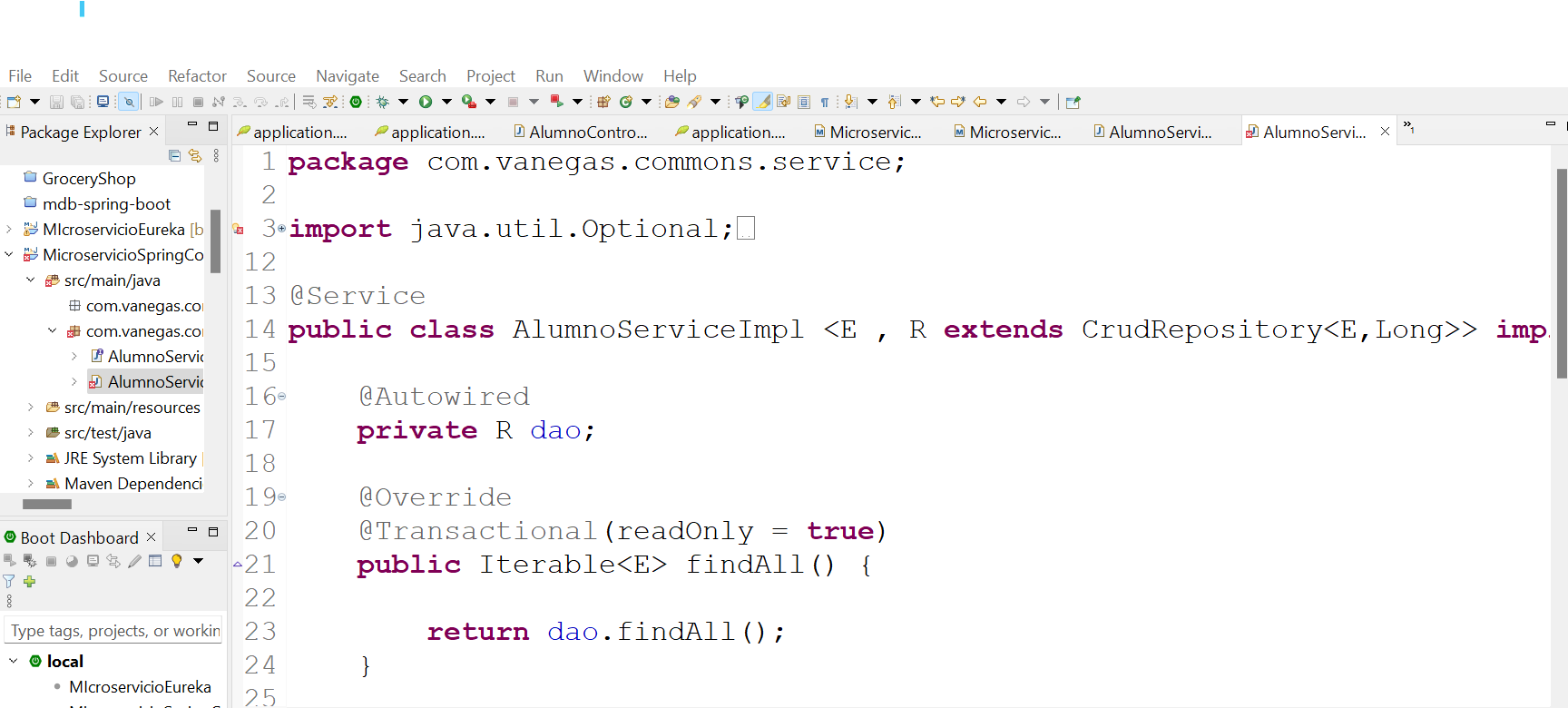


3.6 Vamos a hacer algunos cambios en la interface, quietamos lo que tenga la entidad Alumno y trabajaremos con el api generic de java y quedaría así:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

3.7 ahora modificamos la implementación del service de la siguiente manera



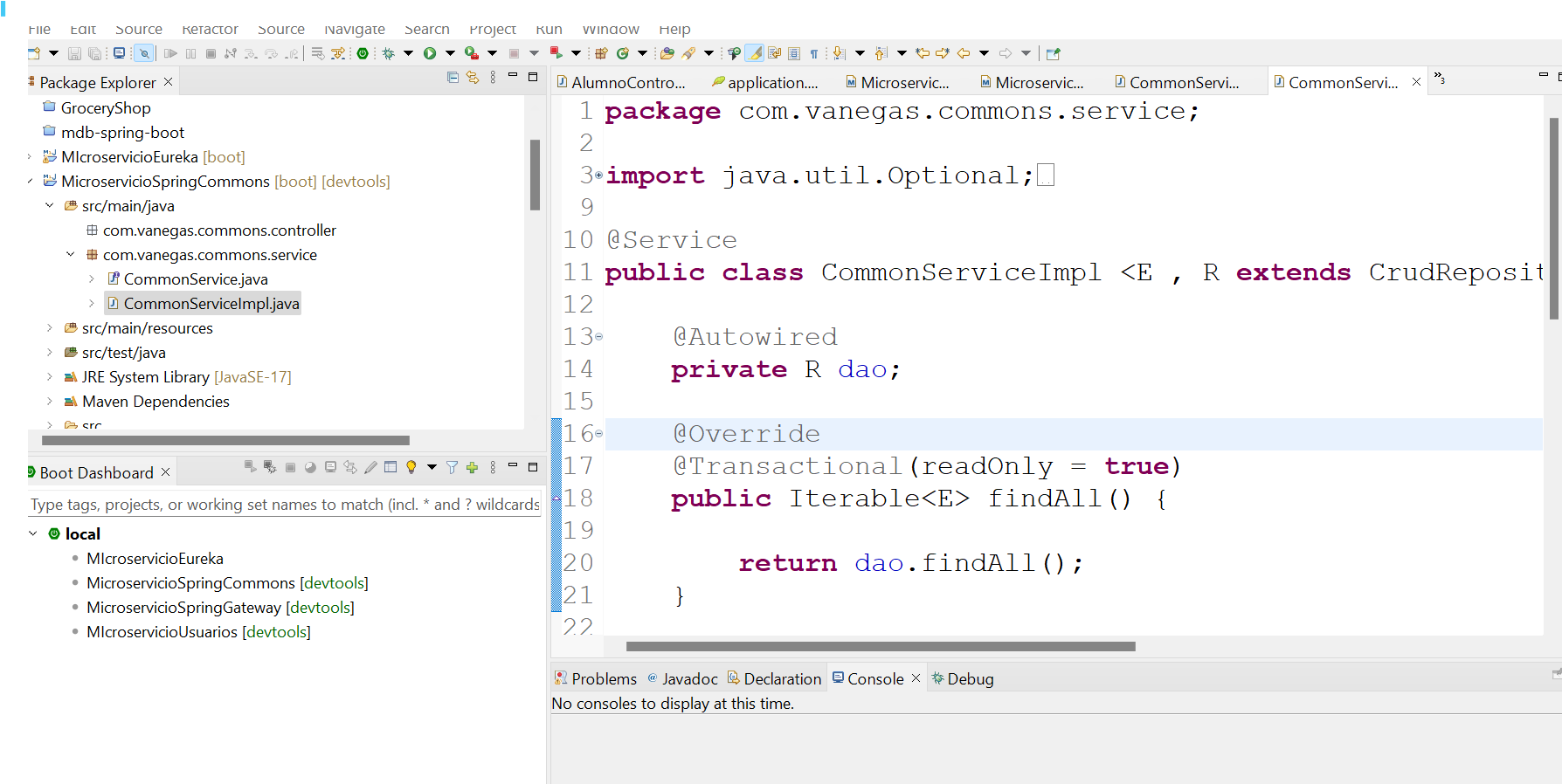
3.8 Cambiamos la inyección del componente repository para que sea genérico , allí se puede ver que estamos usando dos genéricos la E que corresponde a la entidad de la bd y la R que corresponde a la capa de acceso a datos.

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

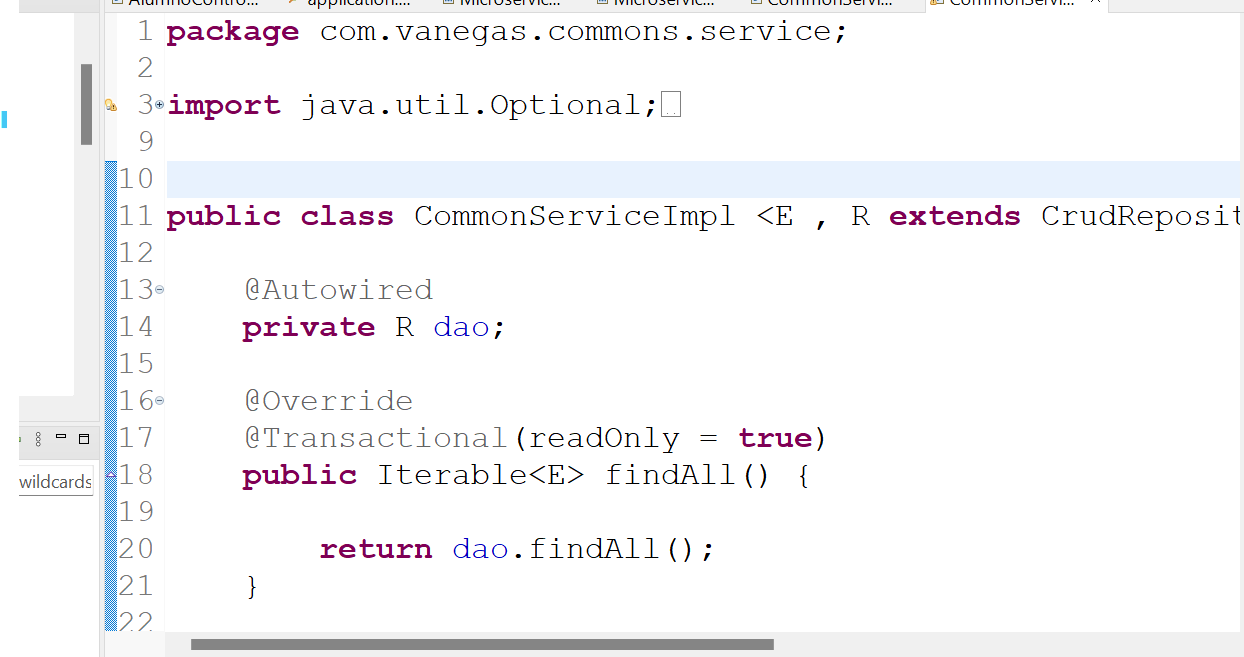
Interfaz de usuario gráfica, Aplicación

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