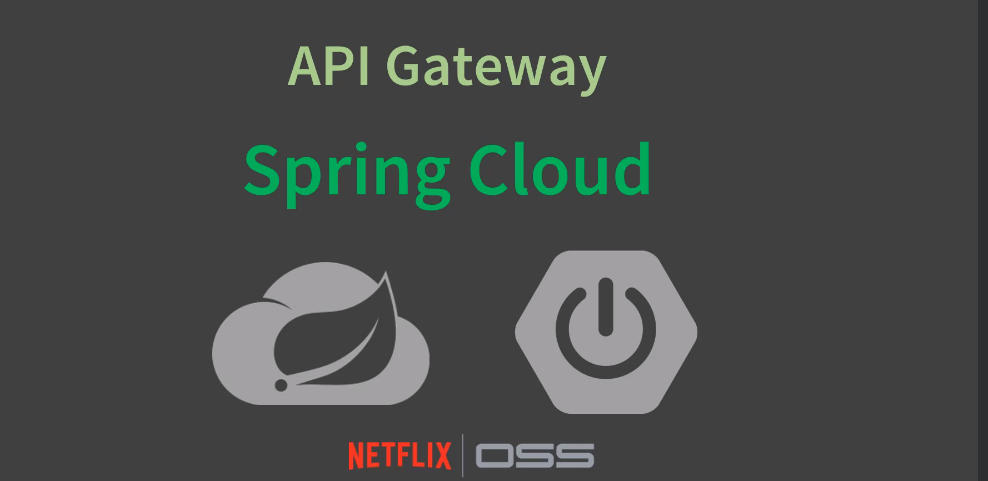
Actividades a desarrollar:

CREACIÓN DE PUERTA DE ENLACE GATEWAY

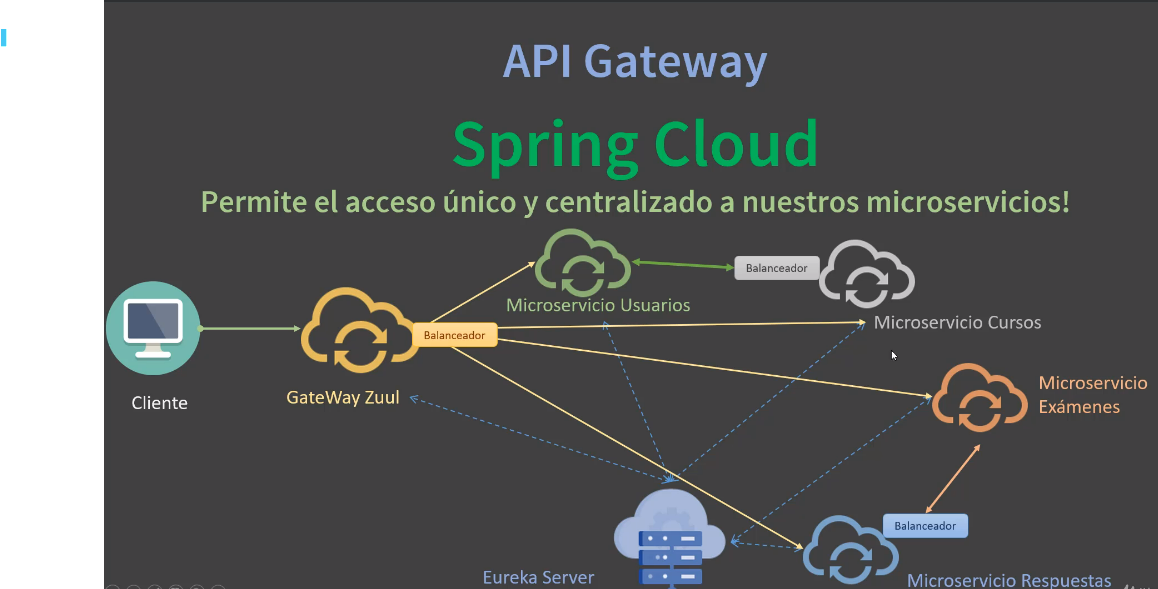


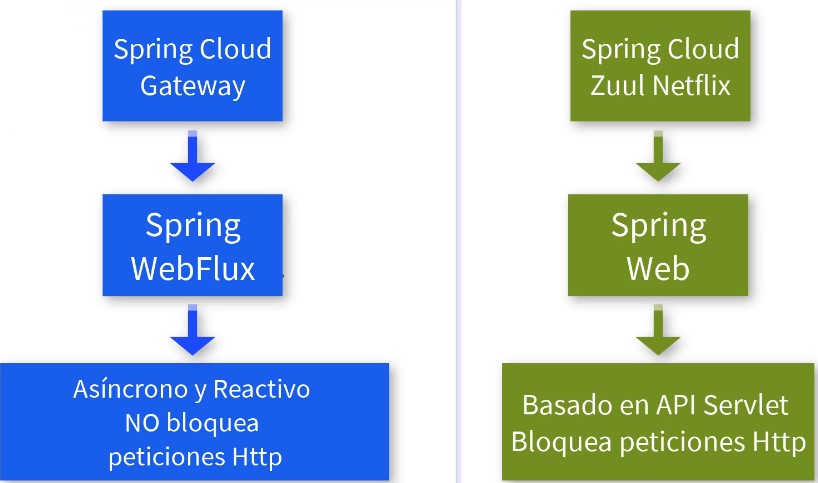
* 1. Buscar y poner en esta parte el concepto de Spring Cloud Gateway

R// Spring Cloud Gateway es un proyecto que se usa para gestionar y redirigir de manera reactiva las solicitudes entrantes a servicios en una arquitectura basada en microservicios. Proporciona funciones como predicados y filtros que ayudan a determinar cómo deben procesarse las solicitudes, permitiendo tareas como agregar o modificar encabezados, balanceo de carga, redirección de rutas, etc

Texto

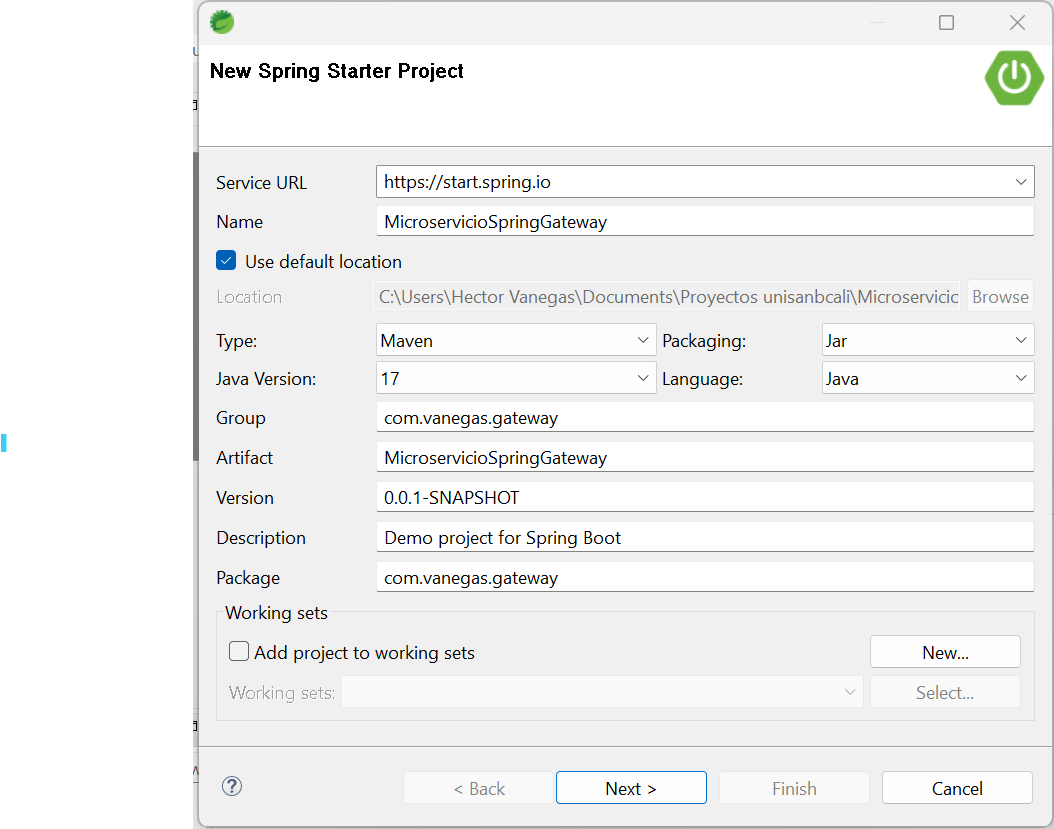
Descripción generada automáticamente



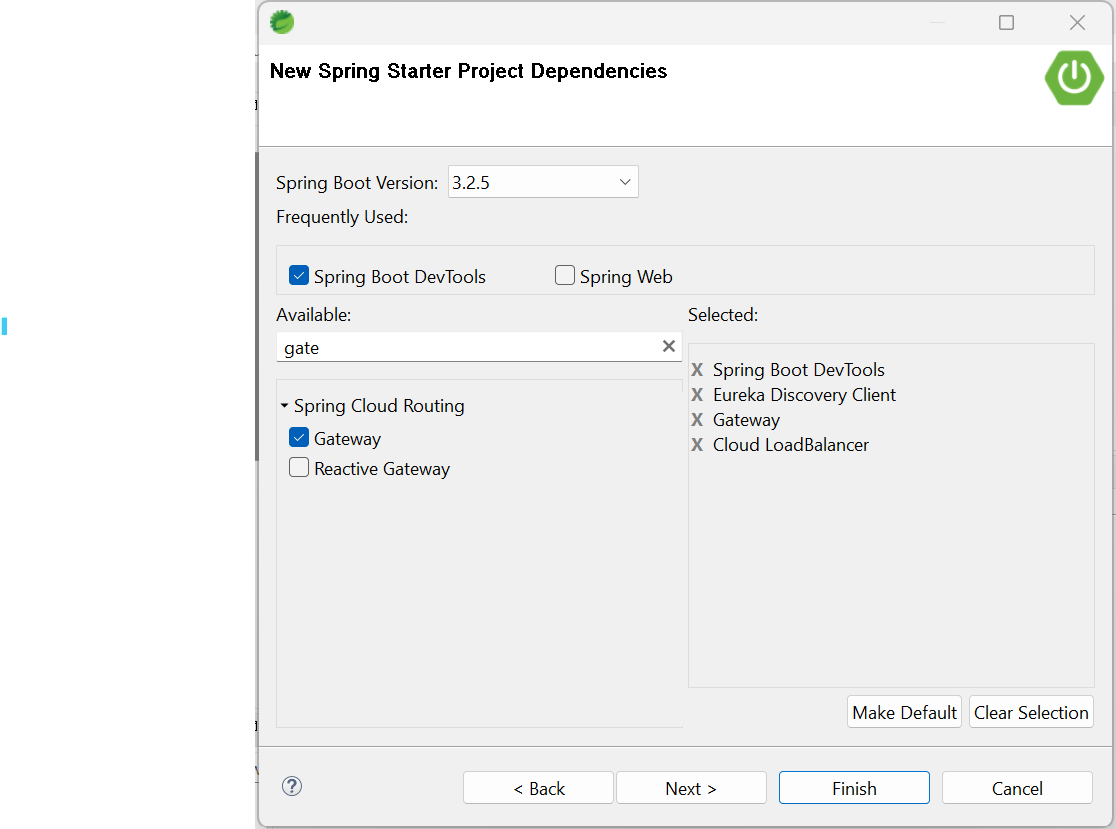


1.2 Se debe crear un nuevo servicio que servirá como apigateway

1. **CREACIÓN DE 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

Texto

Descripción generada automáticamente

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

Texto

Descripción generada automáticamente

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

R//

*”spring.cloud.gateway.mvc.routes[0].id=Usuario”:* Define un identificador único para la ruta en la configuración del gateway. En este caso, el ID es "Usuario", lo que sirve para identificar la ruta de forma específica y permite organizar o gestionar varias rutas.

*”spring.cloud.gateway.mvc.routes[0].uri=lb://Usuario”:* Indica la URI (Uniform Resource Identifier) del servicio al cual se redirigirá la solicitud. El prefijo lb:// especifica que la URI debe usar un balanceador de carga (load balancer), buscando servicios registrados bajo el nombre "Usuario" en el servicio de descubrimiento como Eureka.

*”spring.cloud.gateway.mvc.routes[0].predicates=Path=/api/alumno/\*\*”:* Define una condición (predicado) que se debe cumplir para que la solicitud sea redirigida. Aquí, la condición es que el camino de la solicitud comience con /api/alumno/, redirigiendo todas las solicitudes que coincidan con esta ruta.

*”spring.cloud.gateway.mvc.routes[0].filters=StripPrefix=2”:* Especifica un filtro que elimina partes del prefijo de la ruta antes de redirigirla al servicio de destino. StripPrefix=2 elimina los dos primeros segmentos del camino de la solicitud, simplificando la URL antes de llegar al servicio objetivo.

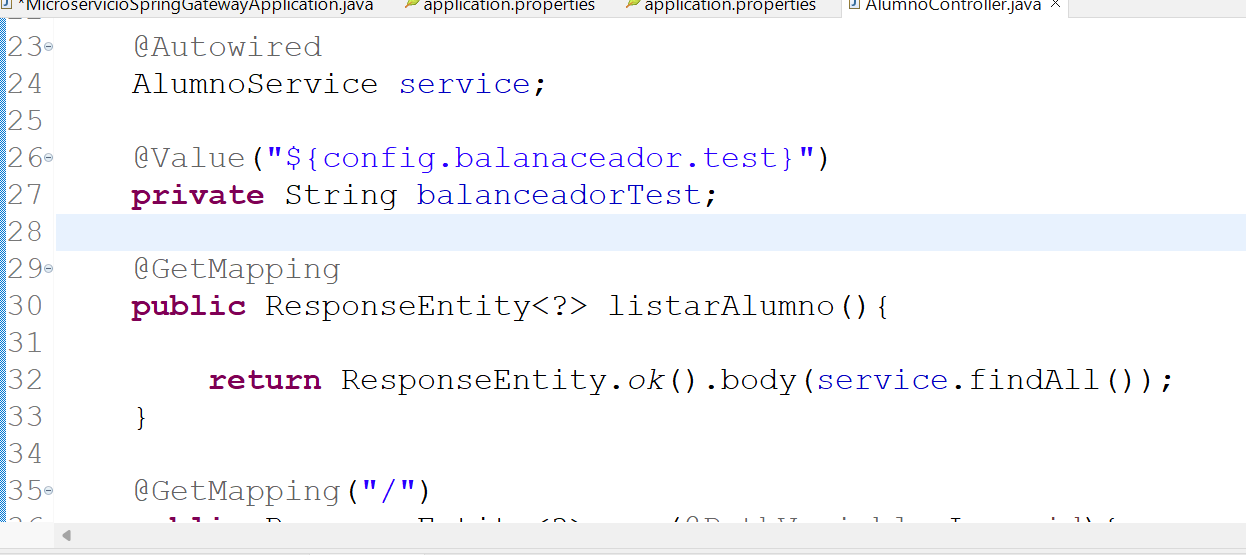
Todas estas propiedades permiten a Spring Cloud Gateway gestionar y modificar las solicitudes entrantes de manera efectiva, facilitando la interacción entre microservicios.

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:

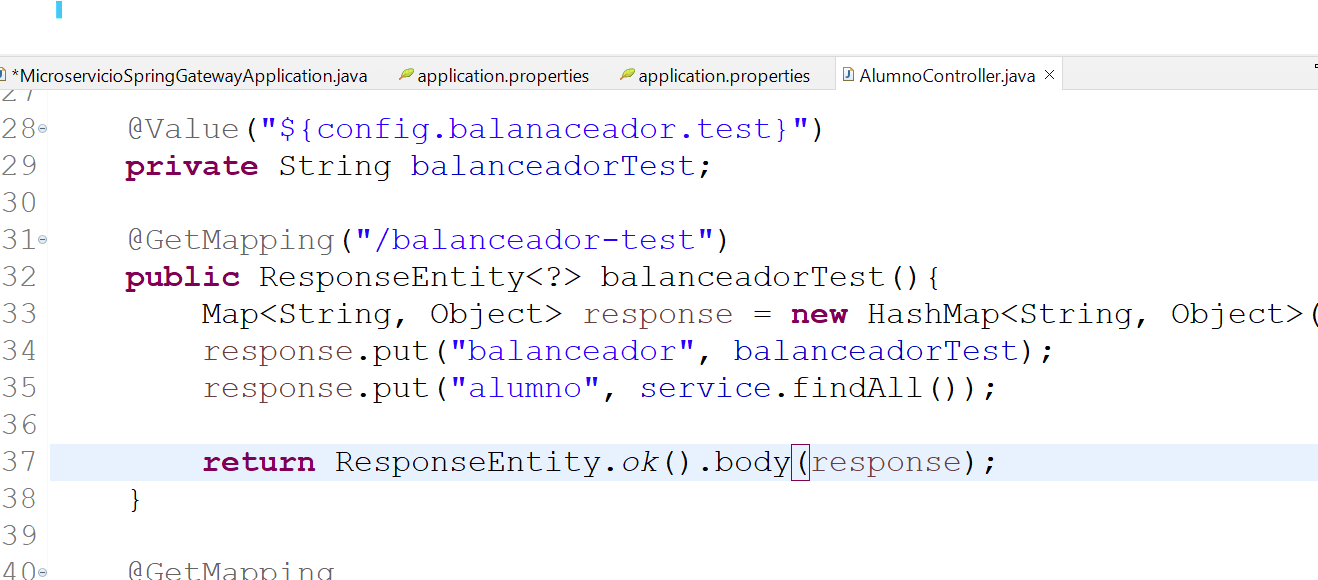
config.balanaceador.test=${BALANCEADOR\_TEST: string por defecto}

Esta variable nos permitirá validar que instancia está 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 servicios levantamos primero a eureka server, luego el microservicio usuarios y por último el Gateway.

Podemos ver cómo están corriendo los 3 servicios

Aplicación

Descripción generada automáticamente

Y vamos a eureka para ver que realmente estén registrados allí

Una captura de pantalla de una red social

Descripción generada automáticamente

**3. CREACIÓN DE LIBRERÍA**

3. Vamos a crear un componente genérico que nos va a permitir generar reúso 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//

En Java, los genéricos permiten definir clases, interfaces y métodos que operan con tipos de datos específicos mientras mantienen la flexibilidad de trabajar con múltiples tipos sin duplicar el código. Los genéricos ayudan a que el código sea más robusto, legible y seguro, además de facilitar el reúso del mismo.

**Ventajas de los Genéricos**

1. Seguridad en tiempo de compilación: Los genéricos permiten detectar errores en tiempo de compilación, lo que reduce los errores de tipo en el código.
2. Reutilización de código: Se pueden definir estructuras de datos y algoritmos que funcionen con cualquier tipo de datos, lo que evita duplicación de código.
3. Legibilidad y mantenibilidad: El uso de genéricos hace que el código sea más claro y fácil de entender, ya que especifica el tipo de dato que maneja.
4. Eficiencia: Los genéricos eliminan la necesidad de realizar conversiones innecesarias, mejorando el rendimiento del programa.

**Usos Comunes de los Genéricos**

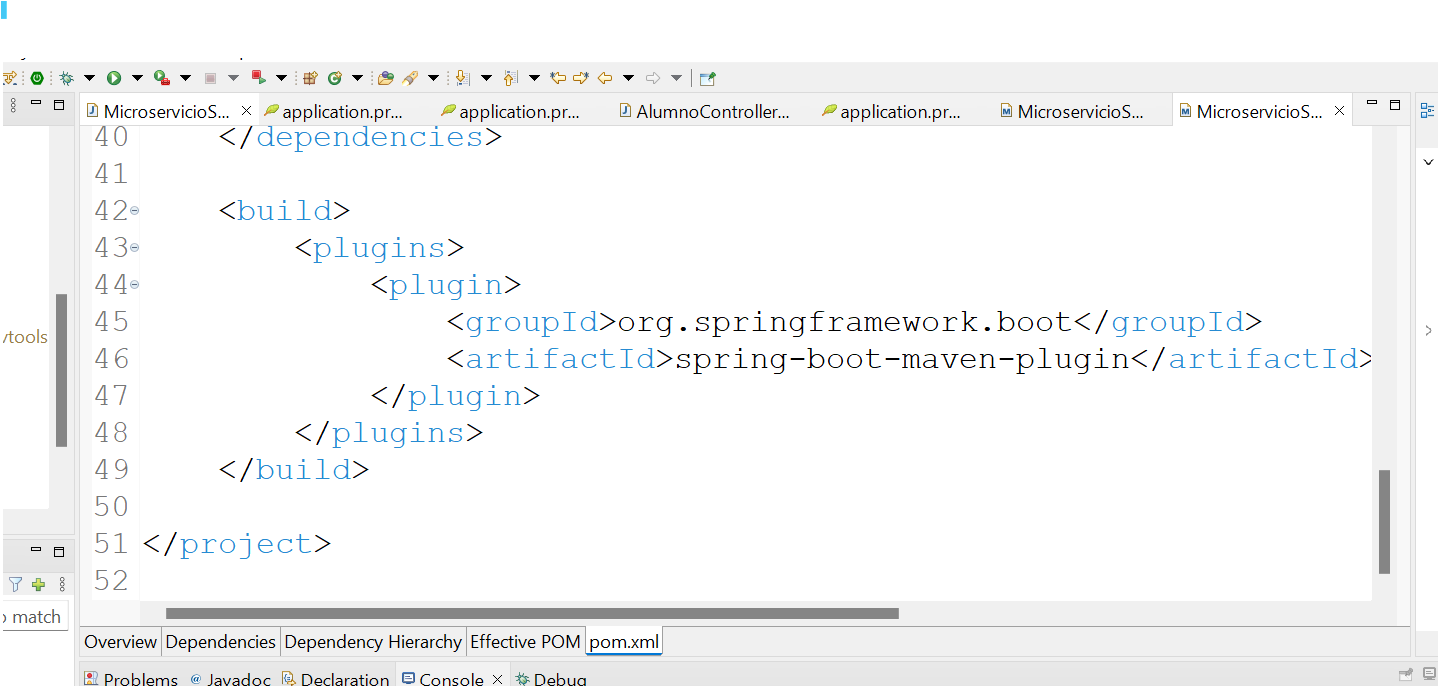
1. Colecciones: Las colecciones en Java (como List, Set y Map) son genéricas, lo que permite especificar el tipo de elementos que contienen.
2. Métodos Genéricos: Permiten que un método sea escrito de forma que funcione con cualquier tipo de dato.
3. Interfaces y Clases Genéricas: Permiten definir interfaces y clases que trabajen con tipos de datos desconocidos en el momento de la creación.

3.1 Creamos un nuevo proyecto con las siguientes dependencias

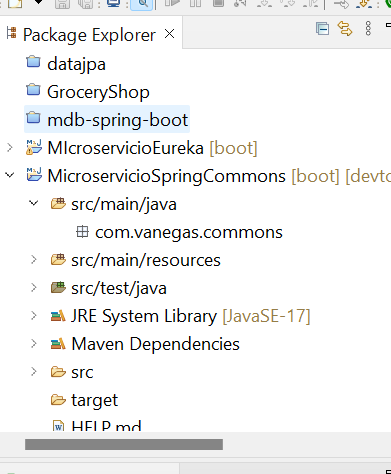
Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

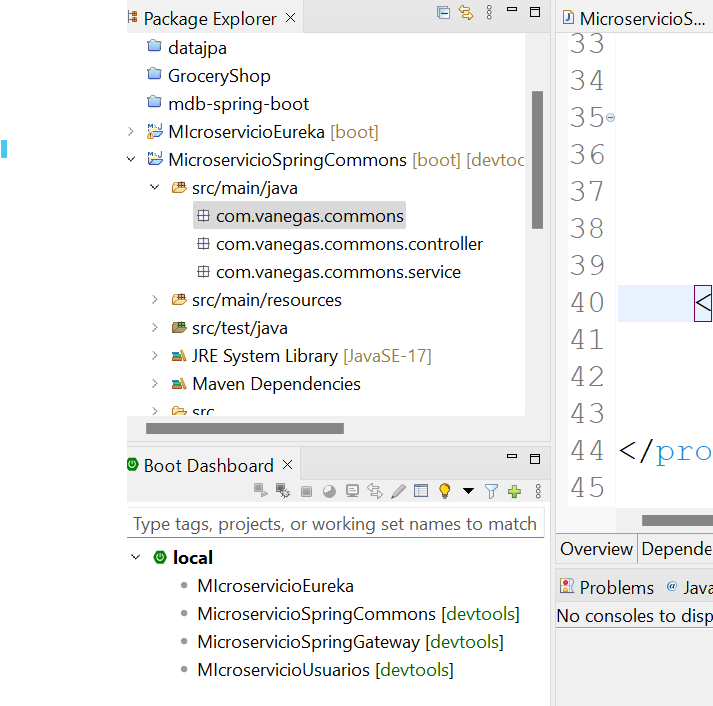
3.2 Luego de creado el proyecto nos dirigimos al pom.xml y eliminamos el plugin de Maven como se ve a continuación:



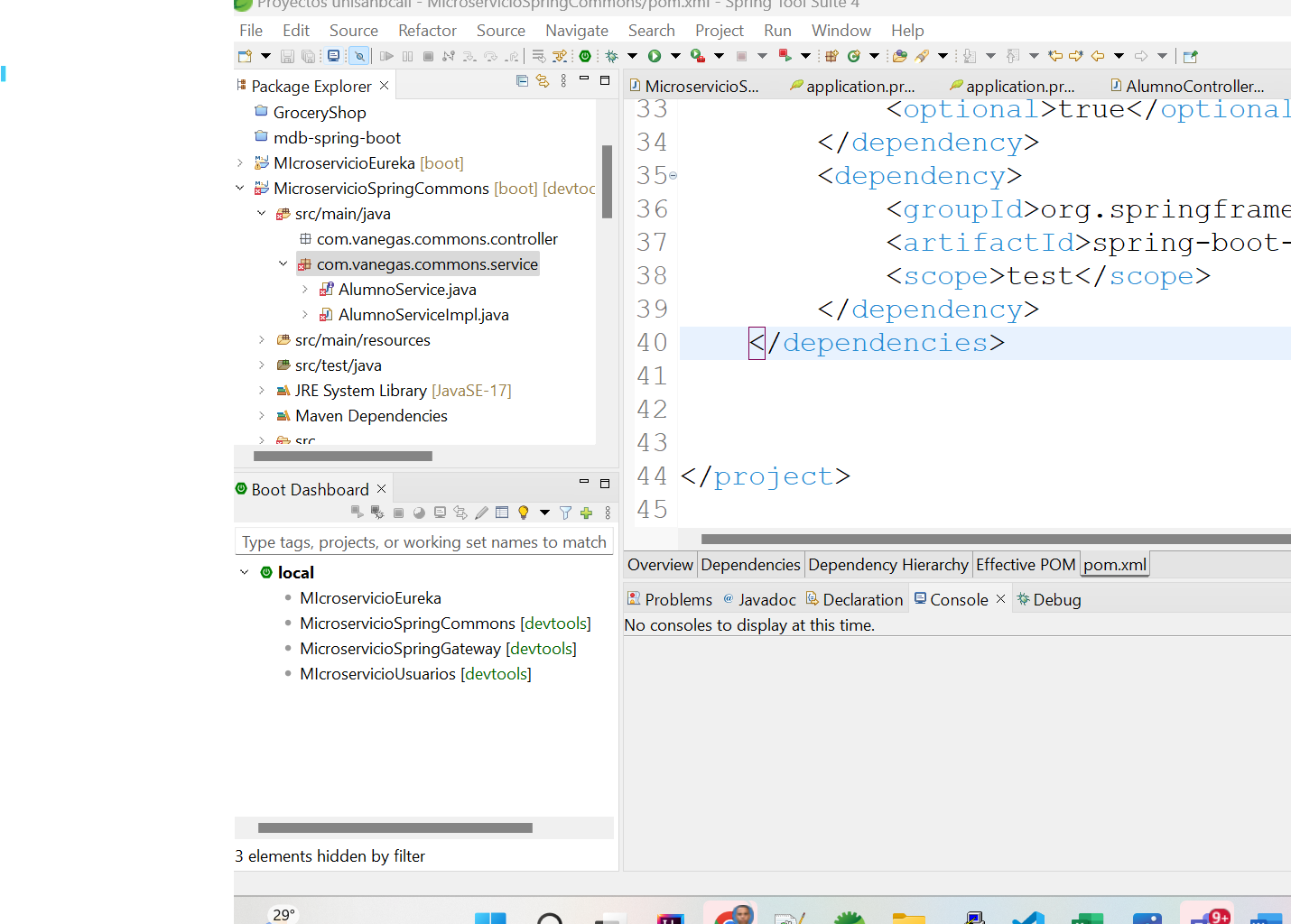
3.3 Luego eliminamos la clase principal ya que lo que se quiere en construir una librería



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

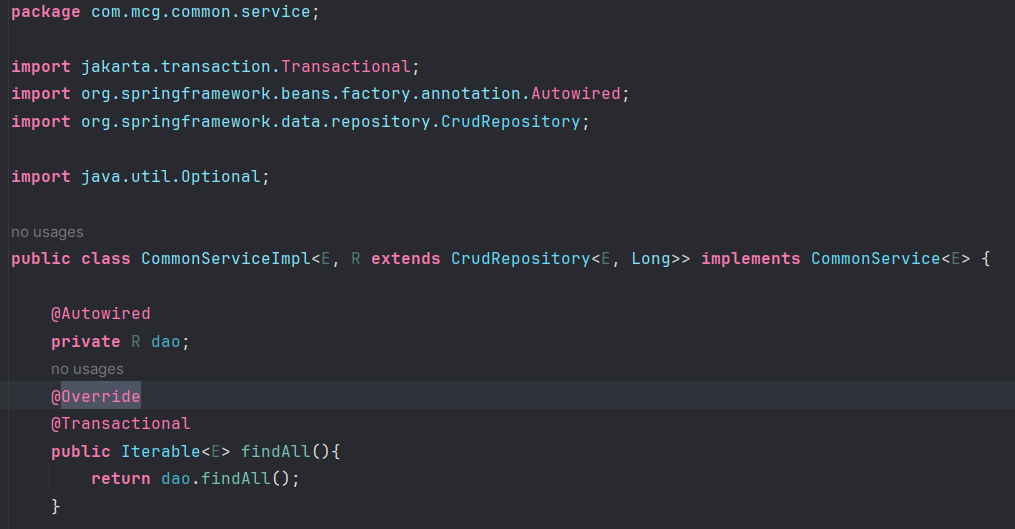


3.6 Vamos a hacer algunos cambios en la interface, quitamos lo que tenga la entidad Alumno y trabajaremos con el api generic de java y quedaría así:

Texto

Descripción generada automáticamente

3.7 ahora modificamos la implementación del service de la siguiente manera



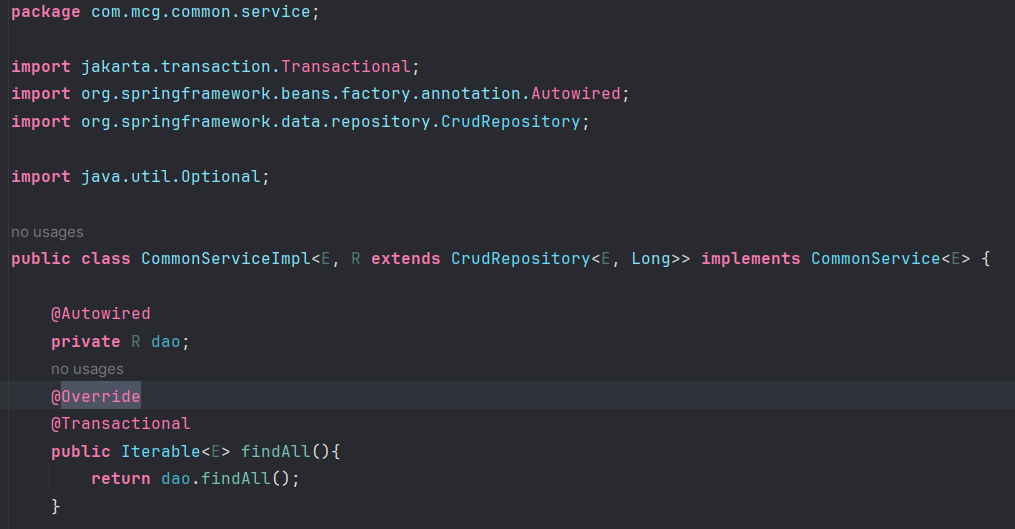
3.8 Cambiamos la inyección del componente repository para que sea genérico, allí se puede ver que estamos usando dos genéricos la E que corresponde a la entidad de la bd y la R que corresponde a la capa de acceso a datos.

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

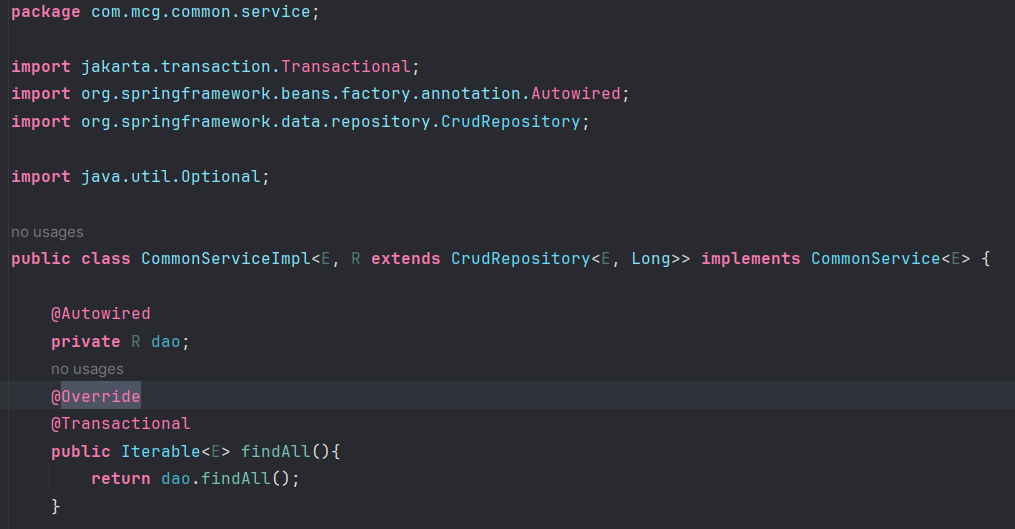
Interfaz de usuario gráfica, Aplicación

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