El proyecto esta conformado por dos entidades gestionadas en diferentes microservicios, donde WasteManagerEntity almacena además una instancia de WasteManagerAddressEntity

El microservicio “Microservicio\_WasteManager” se comunica con “Microservicio\_WasteManagerAddress” , mediante el uso de Feing Client, no se utilizo Rest Template para evitar manipular manualmente el host y puerto del servidor

Para la comunicación por medio de Feing Client , se creo en “Microservicio\_WasteManager” el paquete “com.example.microserviciowastemanager.feignclients” donde se implementó la interfaz “WasteManagerAddressClient” con los métodos del microservicio “Microservicio\_WasteManagerAddress” a utilizar. Con esto se pudo utilizar un instancia de esta interfaz inyectada en la clase que cumple la función de service de este microservicio

Con el objetivo de exteriorizar y centralizar la configuración de los microservicios en un solo lugar se utilizo un servidor de comunicación con Git “config-service” con una carpeta de configuración: “config-data” en la cual se encuentran los detalles principales del perfil dev de los diferentes microservicios

El servidor va a tomar del repositorio git la configuración de la carpeta mencionada, y en específico de la rama “perfil-dev” que es la seleccionada

Para los archivos de configuración se utilizó .yaml en vez de .properties porque ofrece un nivel mayor de organización y la documentación oficial se encuentra en este formato

Se utilizo a Spring Cloud Starter Bootstrap para ayudar al funcionamiento del Config server y por consecuencia mis archivos de configuración de cada proyecto ahora deben de llamarse bootstrap.yaml

Para el registro y localización de microservicios, para el balanceo de carga y la tolerancia a fallos se realizó una implementación de un servidor Eureka “NamingService”, este solo necesita conocer el identificador de un microservicio ("/waste" y "/address" ) para dar con la url correcta

Eureka proporciona un dashboard que permite ver los microservicios existentes actualmente en el registro “http://localhost:8761/”

Para manipular las url de los microservicios se utilizó una puerta de enlace (API Gateway) “GatewayService”, lo que nos permite acceder a los diferentes microservicios desde la misma ip

Esta configuración se encuentra en el “config-data/ gatewayservice-dev.yaml” para el perfil actual “dev” donde dentro de este archivo en los “predicates:” se agregan los "/waste" y "/address" mencionados anteriormente

Para evitar que el sistema se caiga si falla un componente, en este caso el microservicio “Microservicio\_WasteManagerAddress”, se utilizó una implementación del patrón de diseño Circuit Breaker usando Resilence4j

Se utiliza en la clase controladora del microservicio “Microservicio\_WasteManager”

en la función fallBackGetAddres enlazada a los métodos POST y PUT, por medio de la anotación

@CircuitBreaker(name = "microserviciowastemanageraddresCB",fallbackMethod = "fallBackGetAddres")

También es necesario para su funcionamiento la configuración adicional agregada a microserviciowastemanager-dev.yaml de “resilience4j.circuitbreaker” donde se declara la instancia "microserviciowastemanageraddresCB"

Para poder mantener un rastreo de todas las ejecuciones de nuestro proyecto se utilizó a Zipkin mediante la ejecución de su .jar “zipkin-server-2.24.3-exec.jar”

El rastreo se une al proyecto mediante una pequeña configuración agregada a los archivos “microserviciowastemanager-dev.yaml” y “microserviciowastemanageraddres-dev.yaml”, en específico las líneas:

spring:

  application:

    zipkin: http://localhost:9411

luego se puede acceder a la información recopilada desde esta misma dirección

Para desplegar el proyecto:

1. Ejecutar el .jar de Zipkin mediante el comando “java -jar zipkin-server-2.24.3-exec.jar”
2. Ejecutar el proyecto: config-service
3. Ejecutar el proyecto: NamingService
4. Ejecutar el proyecto: GatewayService
5. Ejecutar el proyecto: Microservicio\_WasteManagerAddress
6. Ejecutar el proyecto: Microservicio\_WasteManager

Se incluye además una colección Postman para probar la api

Repositorio del proyecto:

<https://github.com/critBus/prueba_microservicios/tree/perfil-dev>

Descarga de zipkin:

https://search.maven.org/remote\_content?g=io.zipkin&a=zipkin-server&v=LATEST&c=exec