Communication b/w 2 RESTful APIs :-

1. RestTemplate –hard-code url

ResponseEntity<CurrencyConversion> responseEntity = **new** RestTemplate().getForEntity(

"http://localhost:8000/currency-exchange/from/{from}/to/{to}", CurrencyConversion.**class**, uriVariables);

CurrencyConversion currencyConversion = responseEntity.getBody();

1. Feign-client :- to use feign need to add dependency

Step 1)

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-openfeign</artifactId>

</dependency>

Step 2)annotate springbootapp class with @EnableFeignClients

And Step 3)

Now, create a separate interface(not class) and annotate with @FeignClient

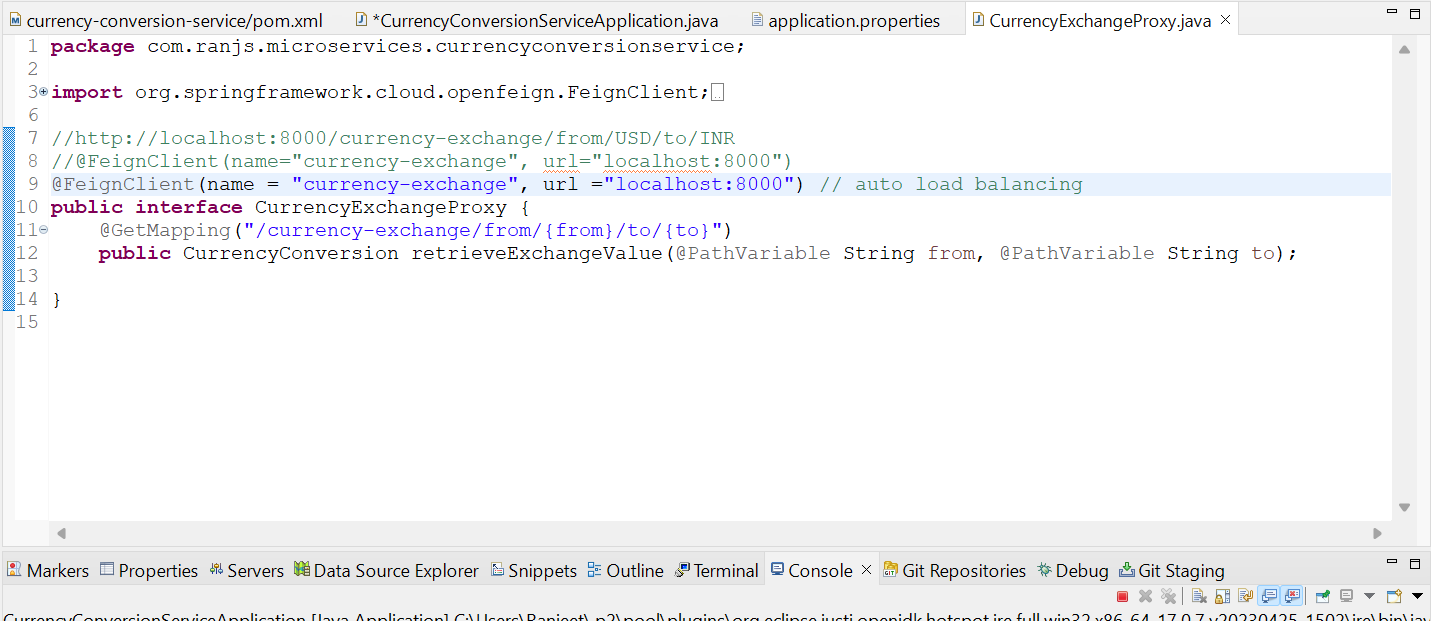
@FeignClient(name = "currency-exchange") // auto load balancing

**public** **interface** CurrencyExchangeProxy {

@GetMapping("/currency-exchange/from/{from}/to/{to}")

**public** CurrencyConversion retrieveExchangeValue(@PathVariable String from, @PathVariable String to);

}

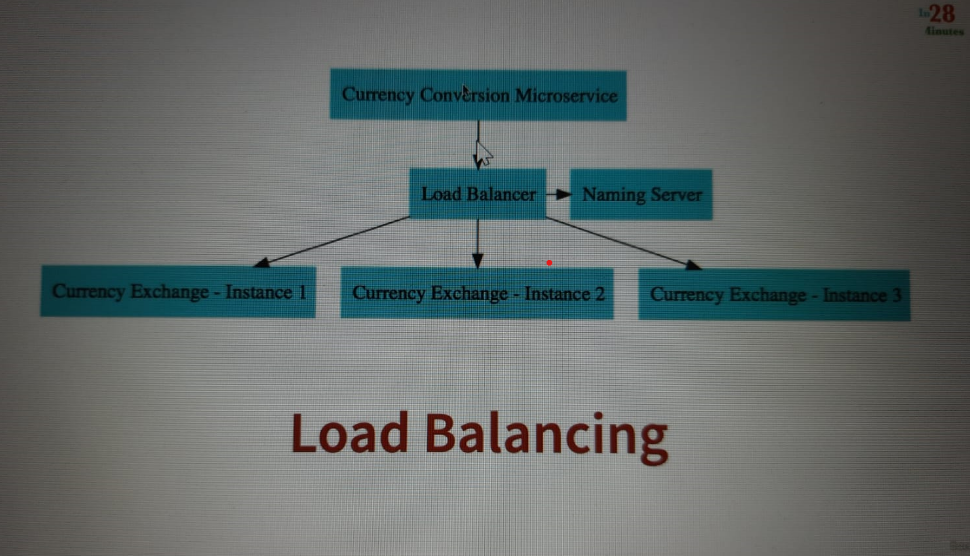


Feign also helps to do load balancing auto.

**Here, we r still hard-coding localhost:8000.**

To balance load lets we have **created 3 instances** of currency-exchange-service on port 8000, 8001 & 8002.

For that we need to go for service-registry so that all the instances of all the services will be registered on **service-registry**.



So, create a separate service called naming-server

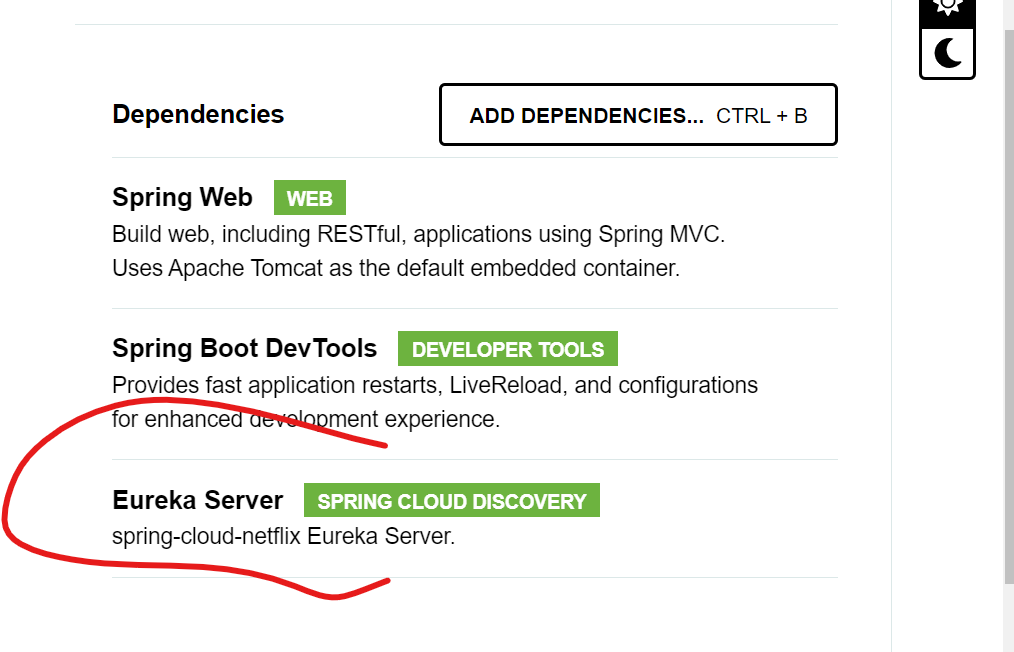
Add eureka server dependency: -

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

</dependency>



Add @EnableEurekaServer

In springbootapp class

And add in app.properties

spring.application.name=naming-server

server.port=8761

##for not allowing this service to be registered with eureka

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

start app it will run at 8761 porn no

need to add below eureka client dependency in service which we want to register with naming server:-

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

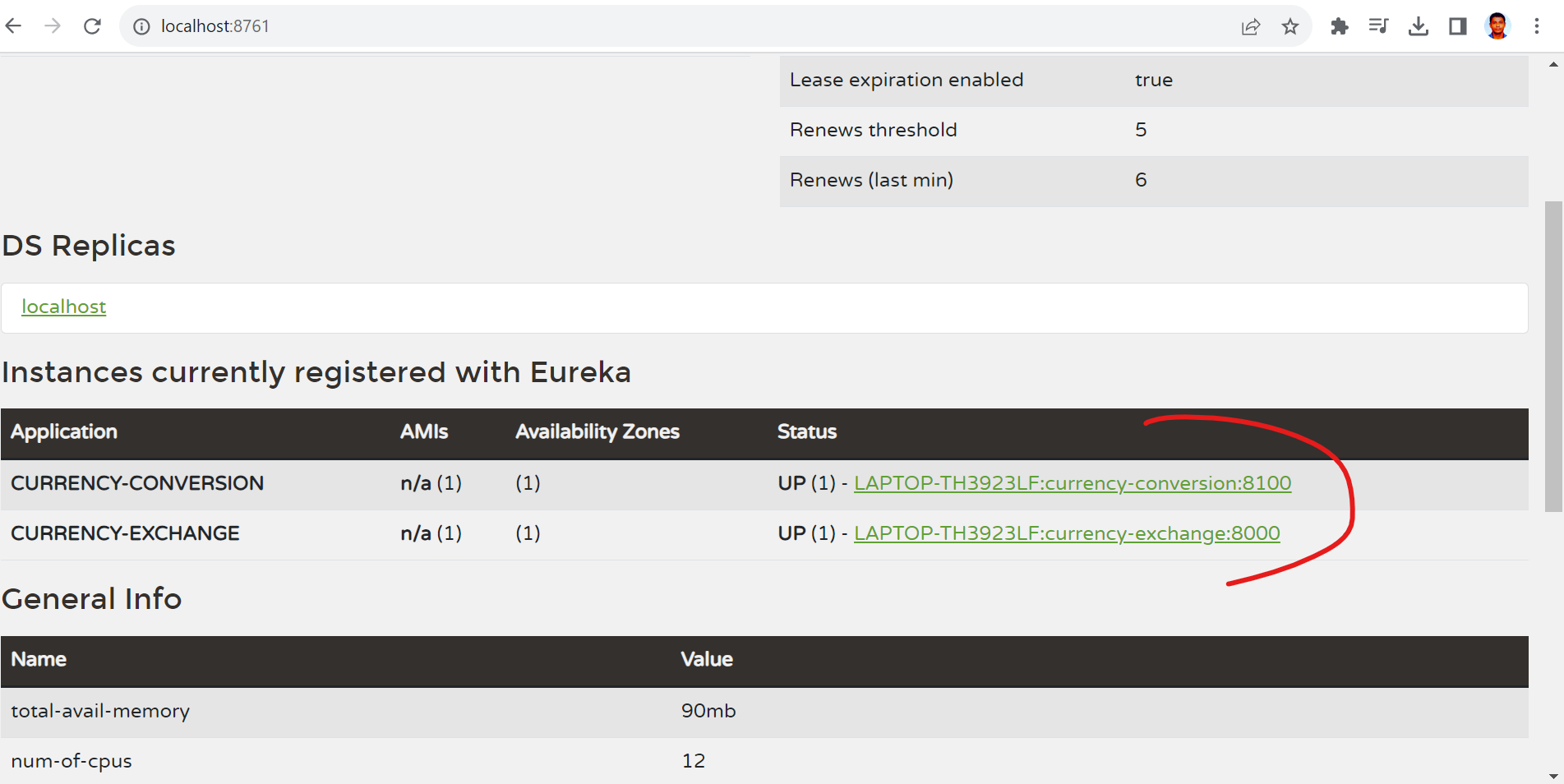
</dependency>

It will auto registred with eureka.

But for more csutomization we can add some more in app properties: -

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/

we can check <http://localhost:8761/>

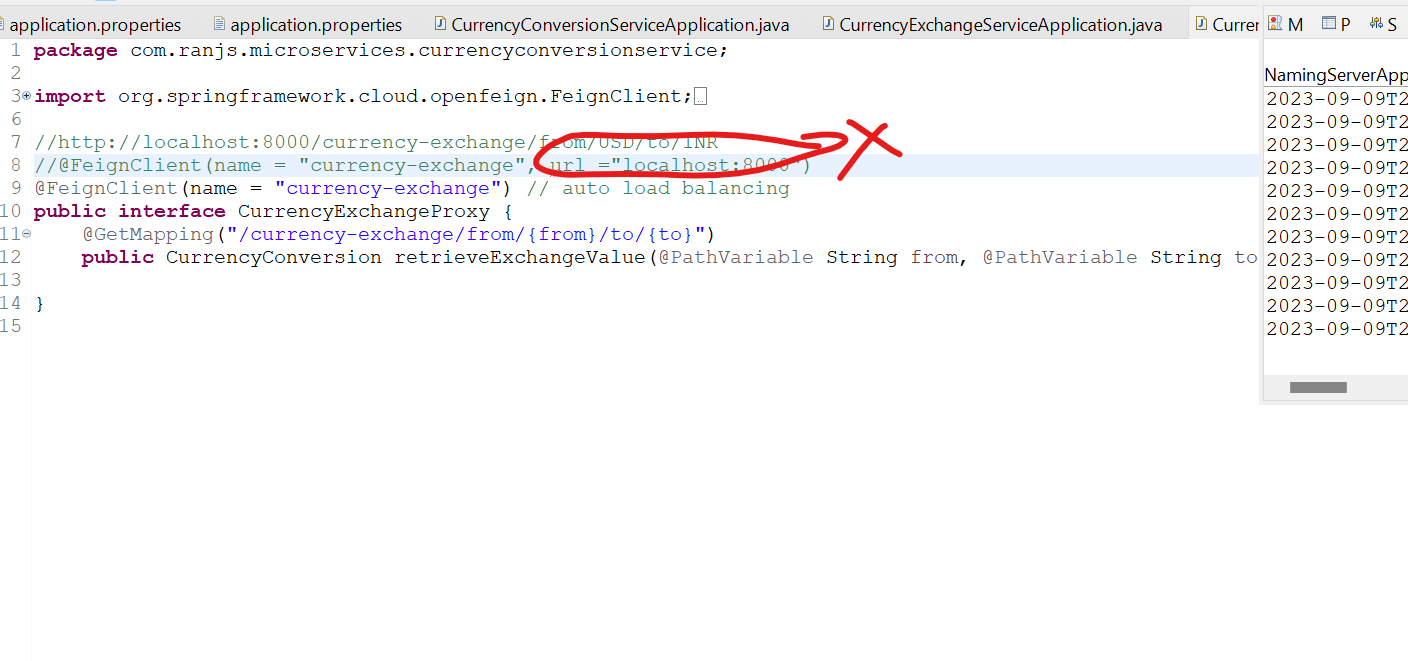


@EnableDiscoveryClient

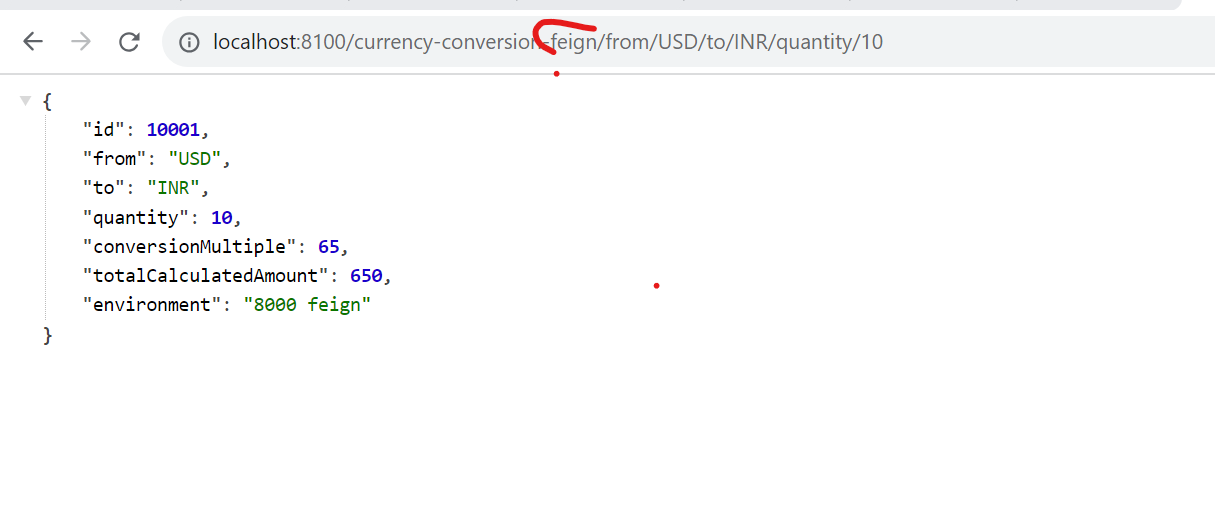
In service which we want to register is optional

Load balancing b/w multiples instances of a target service: -

For that we need to remove url from @FeignClient



Now, feign will auto balance the load b/w multiple instances of a target service namely currency-exchange-service whose instances are running on 8000,8001, & 8002 port no.



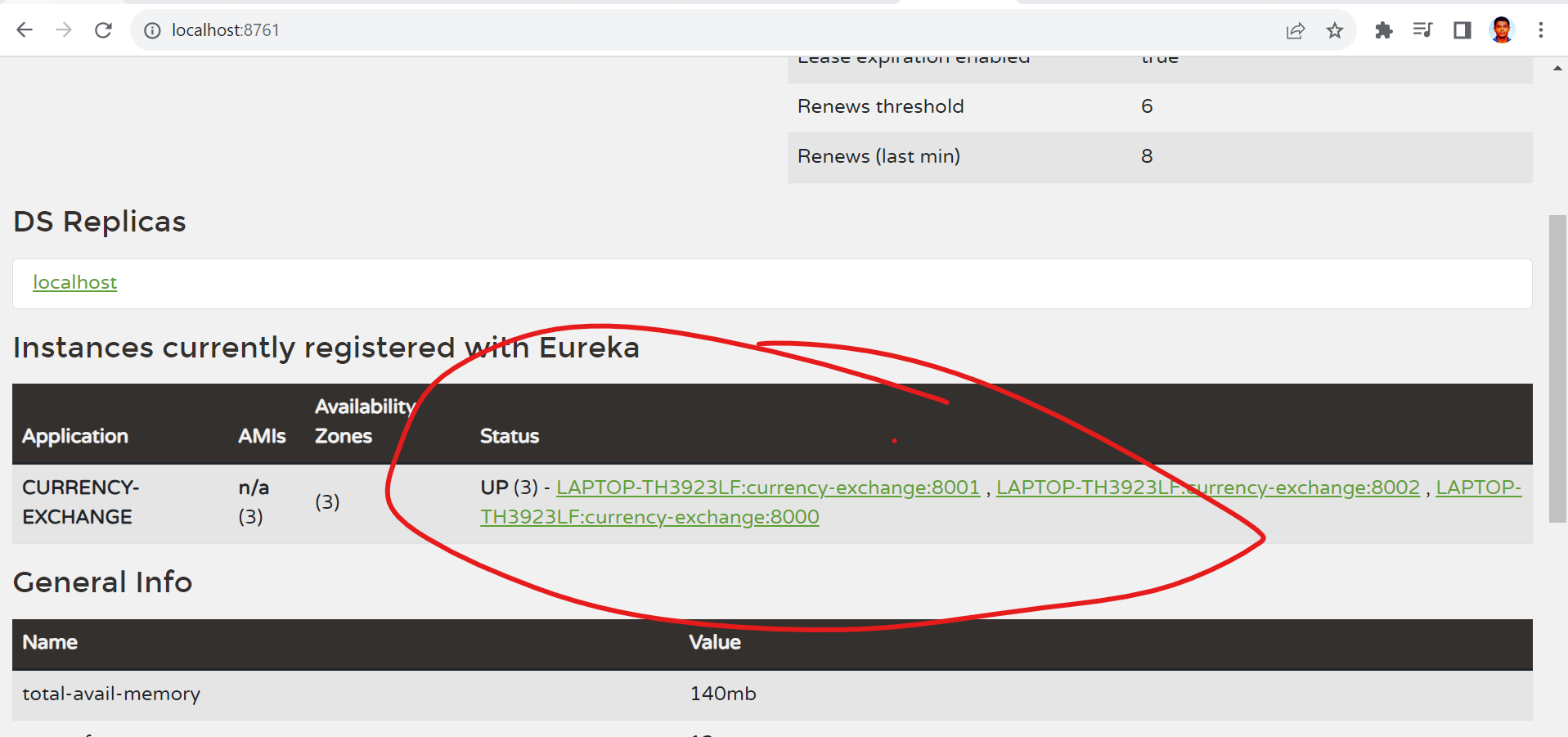
Now, let’s **create instances of currency-exchange**

@EnableDiscoveryClient

In springbootapp class

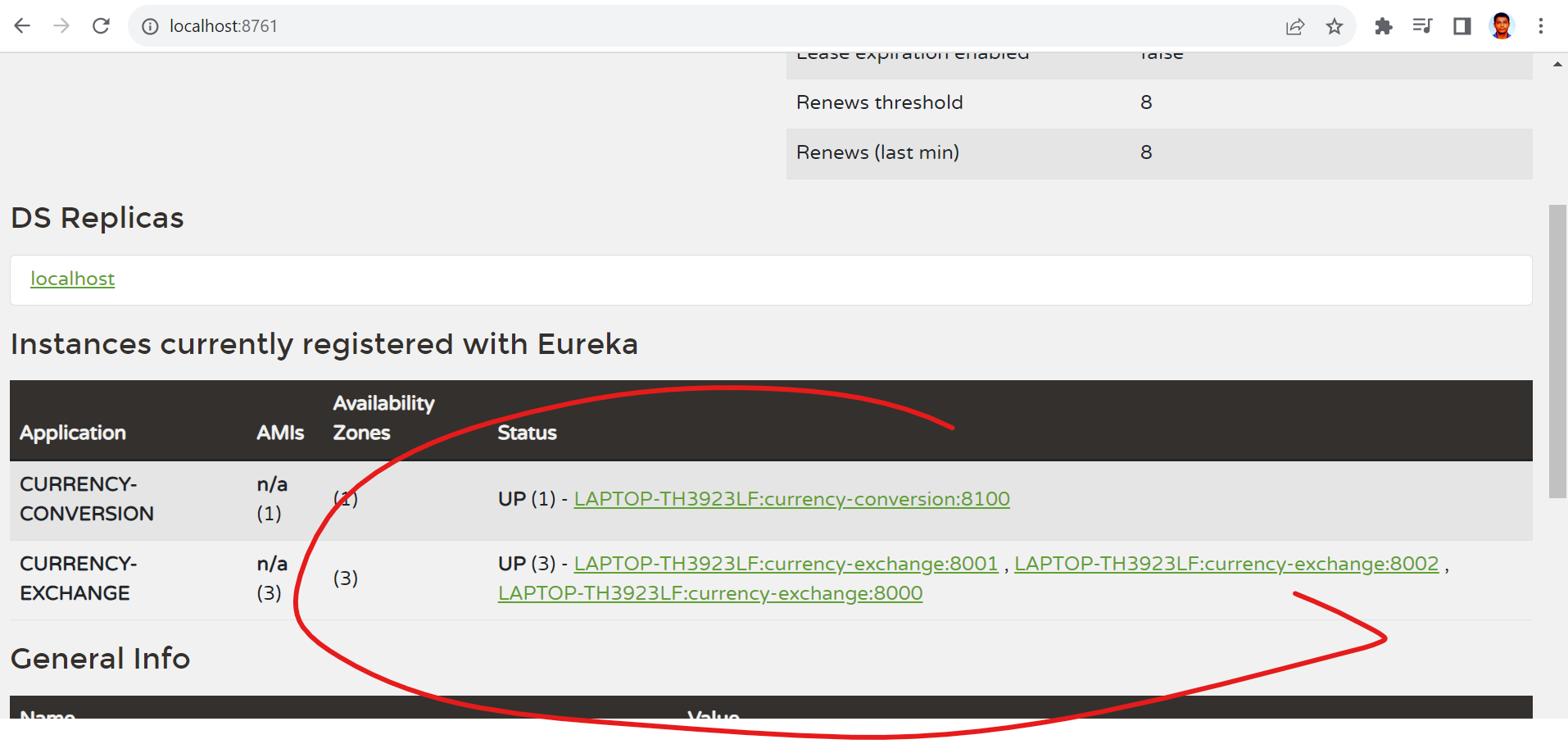
Right click on service -> run as config-> duplicate ->

-Dserver.port=8001 in vm args -> run



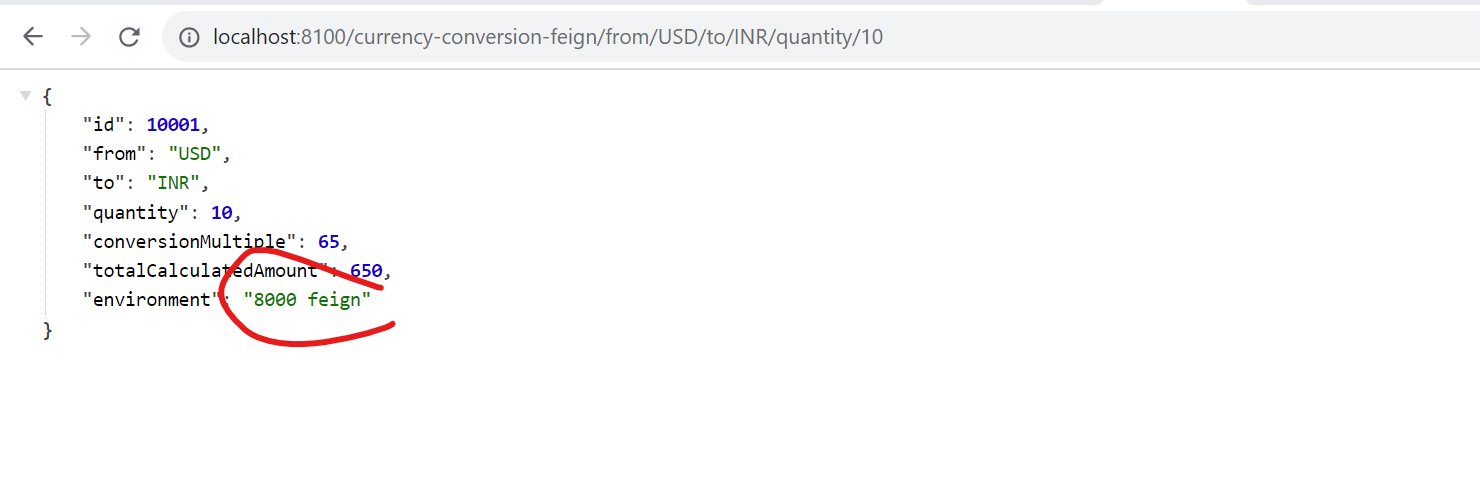
Three instances created.

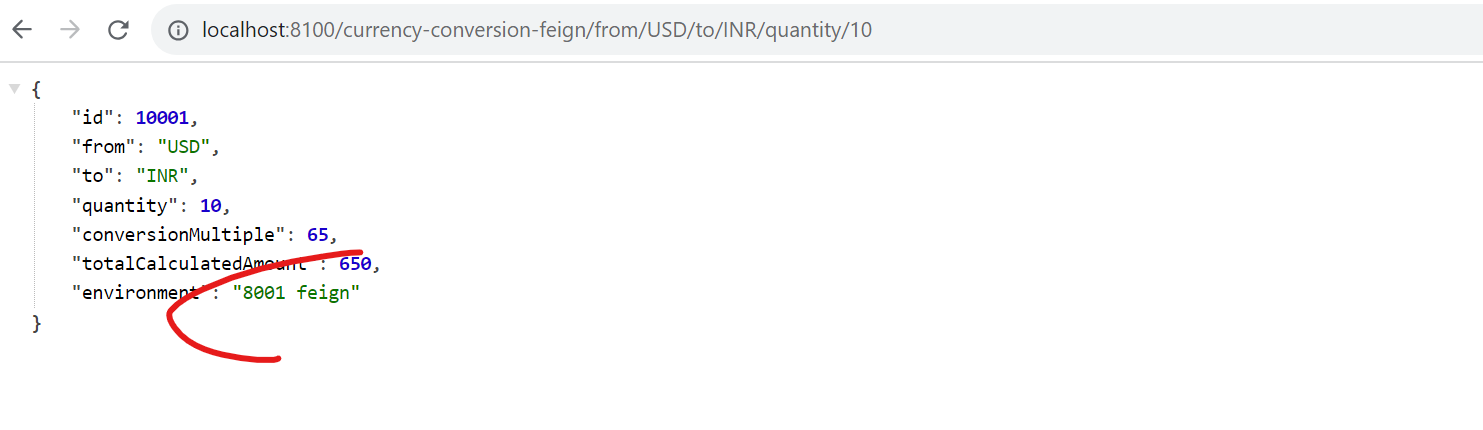
Both services got registred with eureka naming register



Now hit <http://localhost:8100/currency-conversion-feign/from/USD/to/INR/quantity/10>

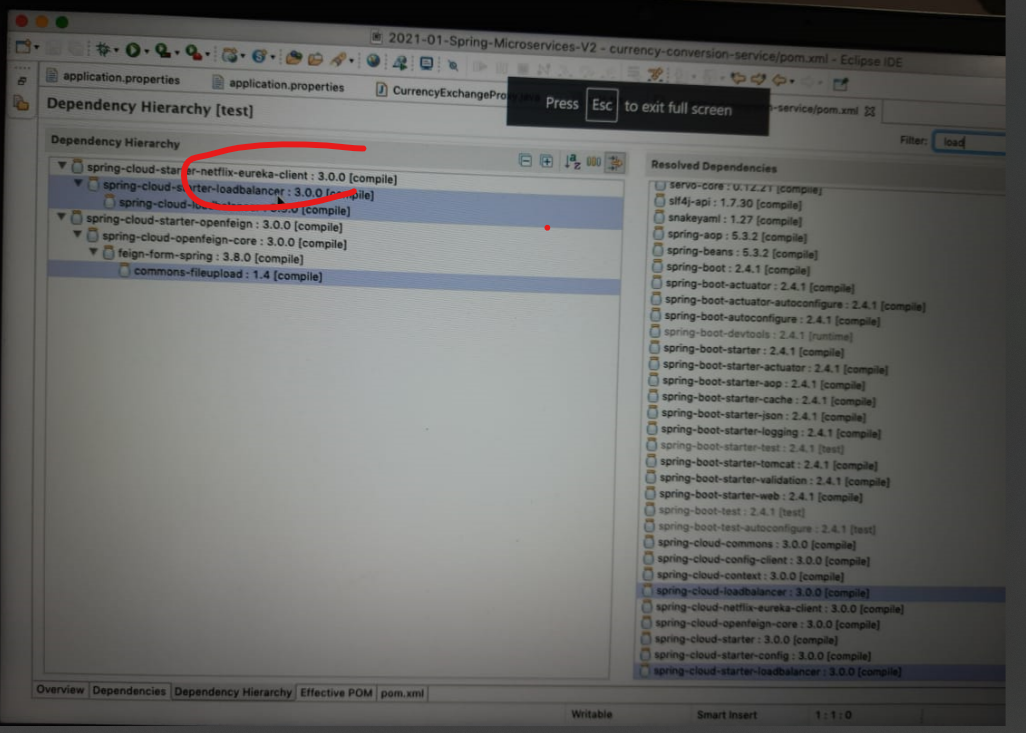
Same url but response from diff instances on diff times and load balancing is taken care by **Feign** itself







Spring-cloud-load-balancer has been brought up to class path by Netflix-eureka-client . This is **client side load balancing**.



Spring cloud API GateWay:-

 Spring Cloud Gateway aims to provide a simple, yet effective way to route to APIs and provide cross cutting concerns to them such as: security, monitoring/metrics, and resiliency.

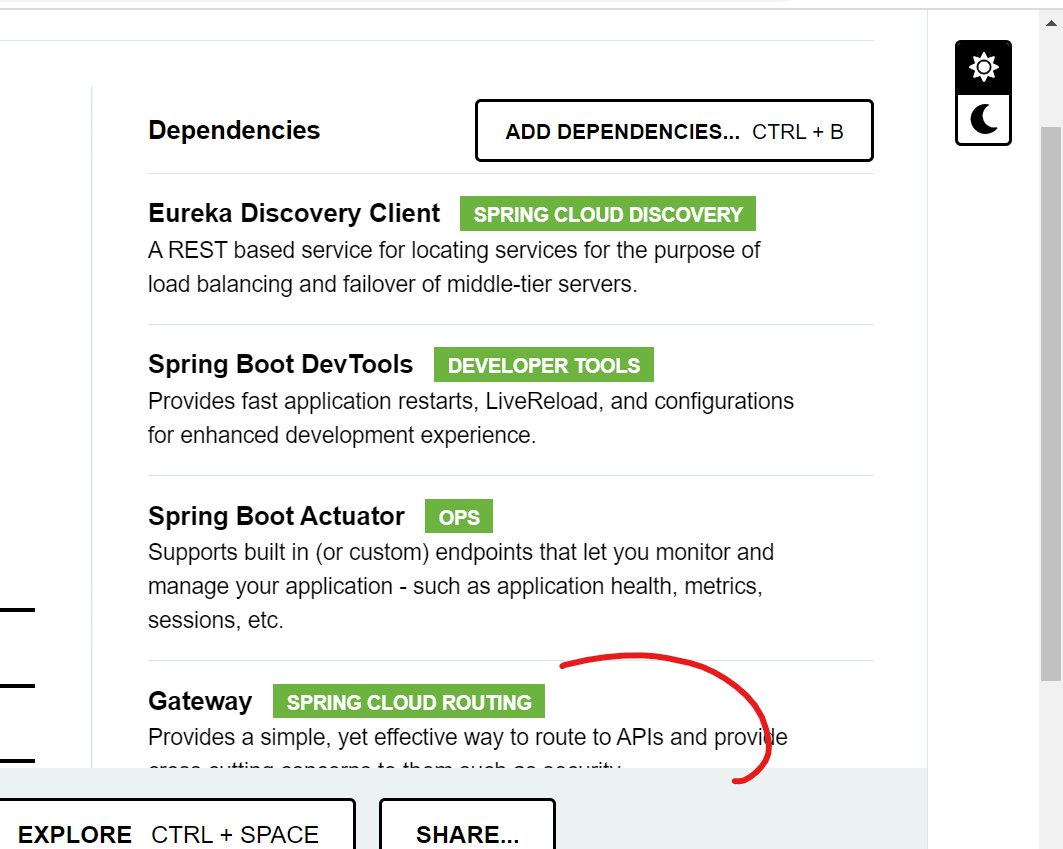
As microservices arch has 100s of microservices and some or many have lot of common features like authentication, Authorization, logging, Rate Limiting.

Spring API gateway provides solutions for all these.

In older version of spring cloud we use Zuul but now its not supported by Netflix.

So, now we use spring cloud api gateway.

Now create an new service for api gateway.



Registered api gateway with eureka server

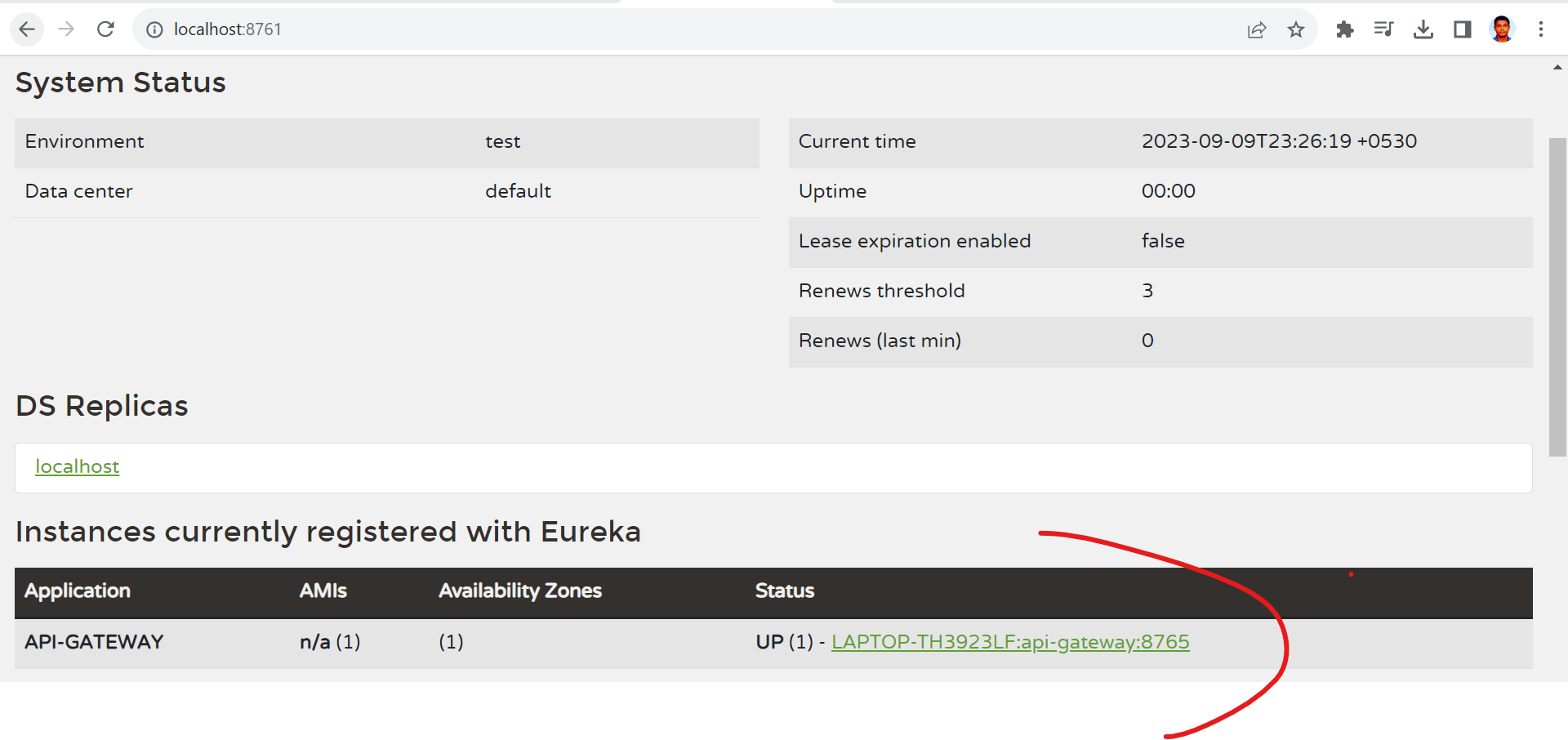
By adding

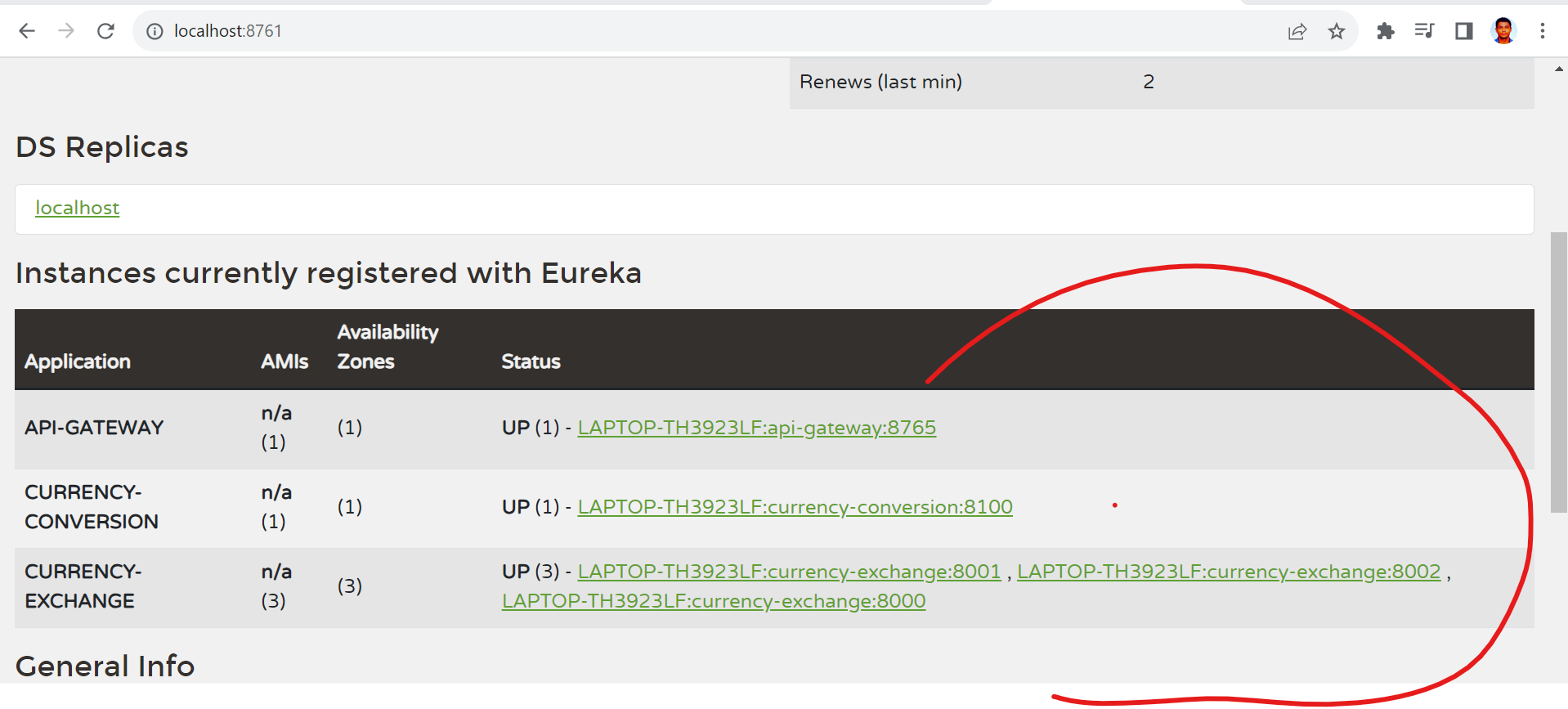
<dependency>

<groupId>org.springframework.cloud</groupId>

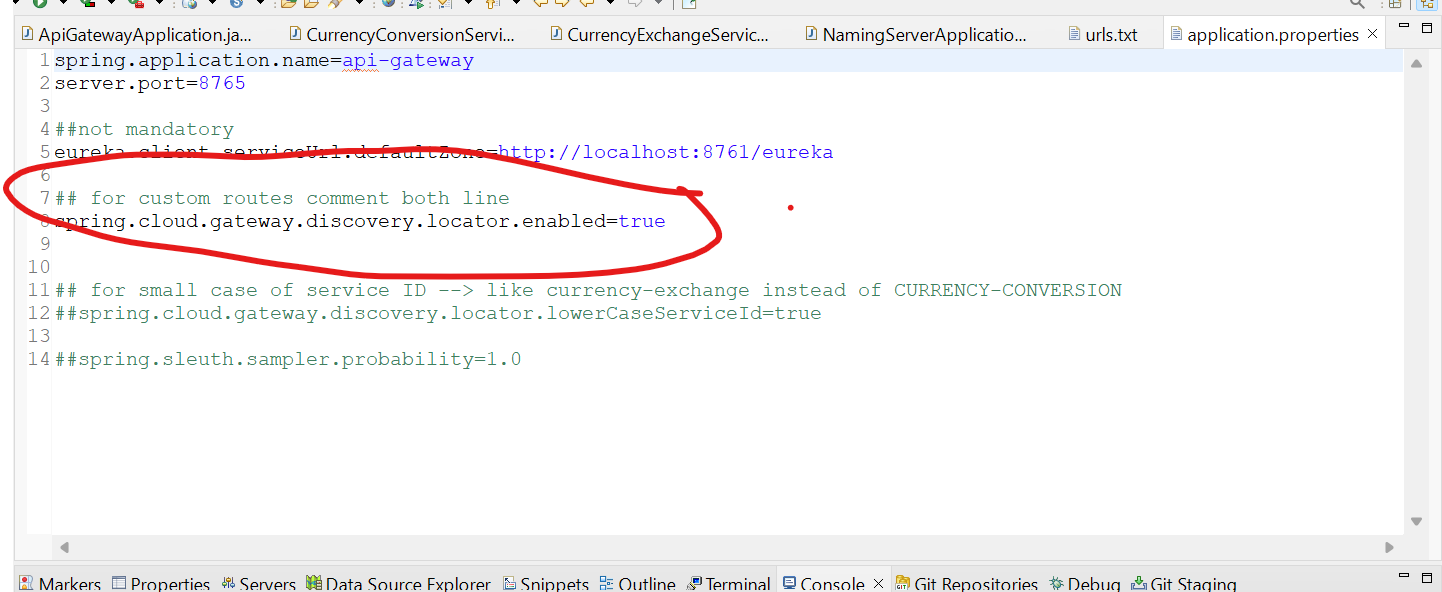
<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>





Api gateway service will run at8765 port no,



spring.cloud.gateway.discovery.locator.enabled=true

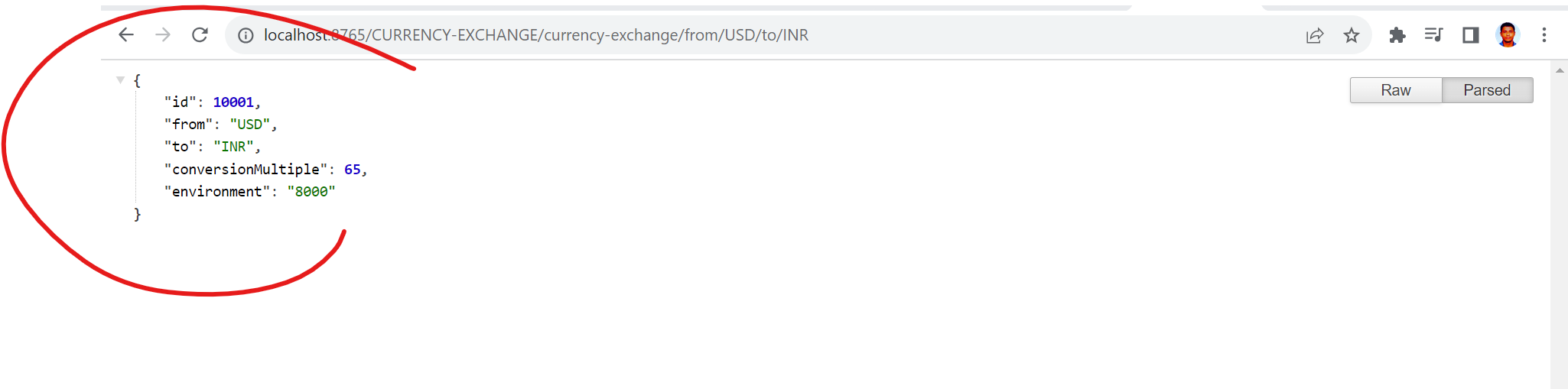
this is mandatory to utilize the url using api gateway.

<http://localhost:8765/CURRENCY-EXCHANGE/currency-exchange/from/USD/to/INR>

8765 –port no api gateway

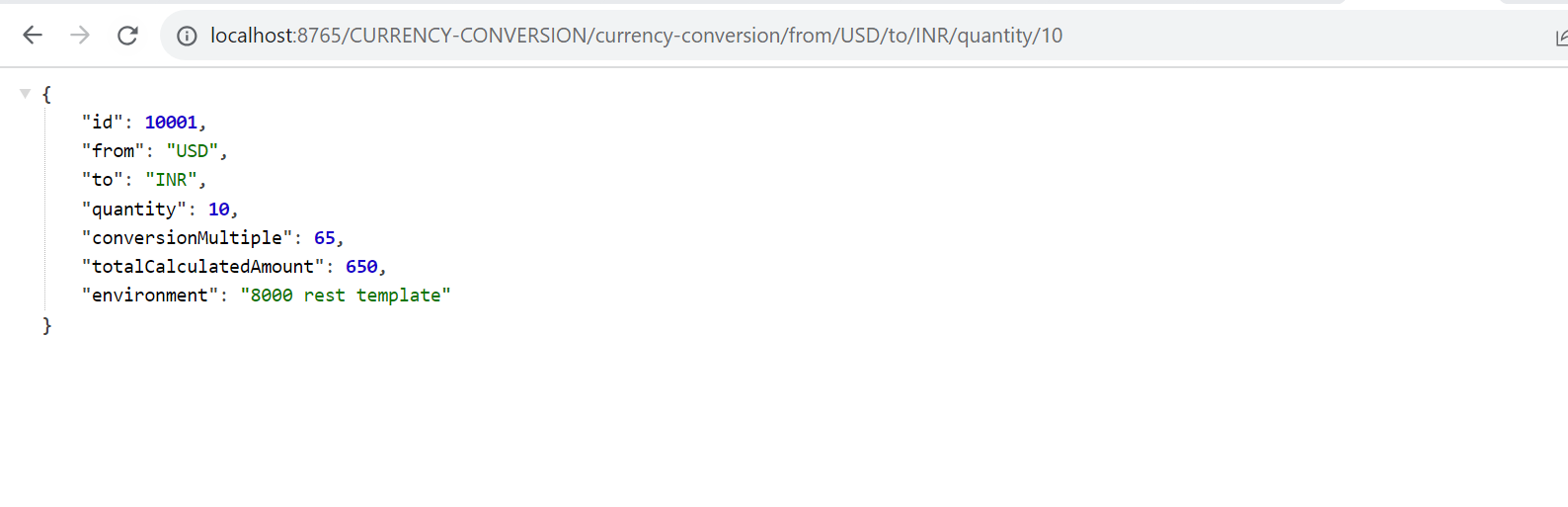
**CURRENCY-EXCHANGE – exchange service registered with eureka naming server with this name.**

currency-exchange/from/USD/to/INR – this is end-point.



http://localhost:8765/CURRENCY-CONVERSION/currency-conversion/from/USD/to/INR/quantity/10

we can access currency-conversion endpoint as well using api gateway.

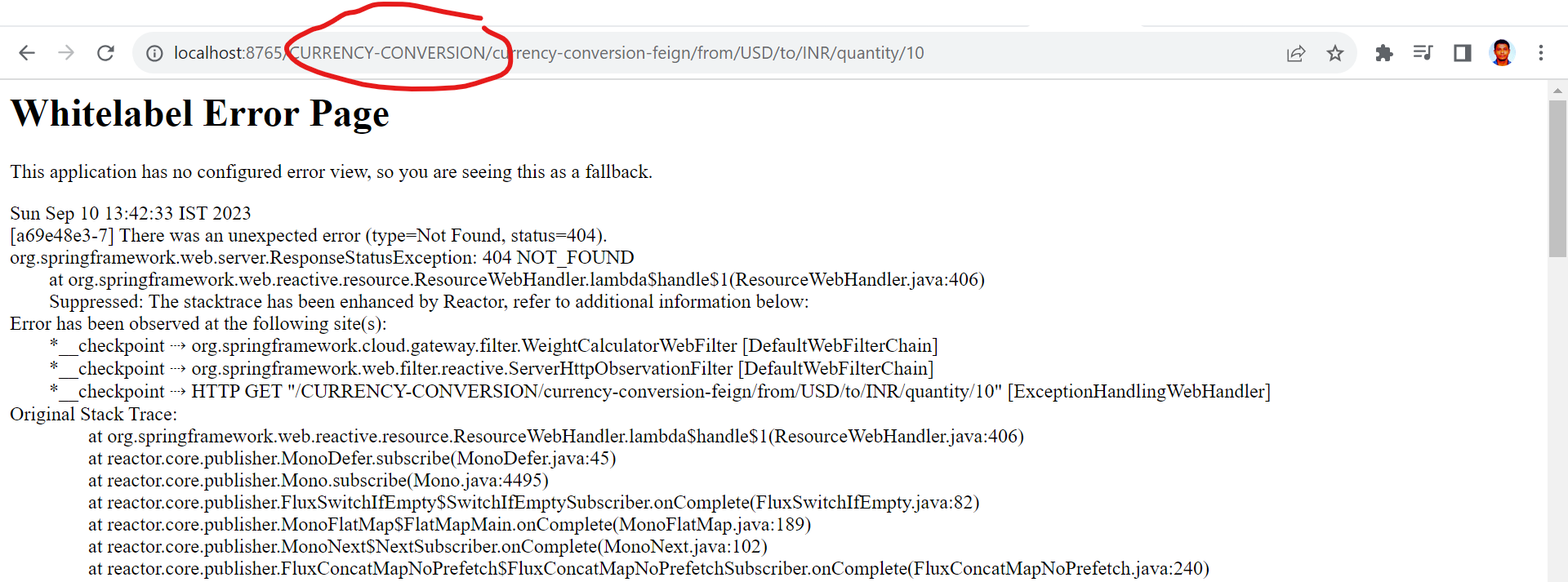


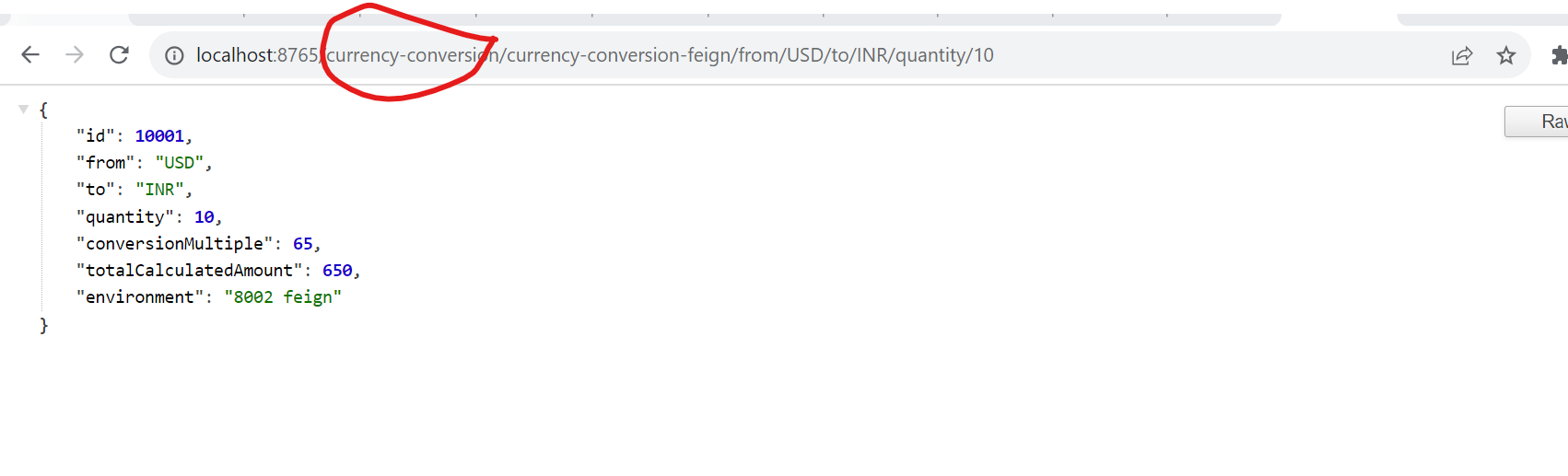
Note:- spring.cloud.gateway.discovery.locator.lowerCaseServiceId=true

In app properties of gateway so that we can ignore case sensitive.

http://localhost:8765/currency-conversion/currency-conversion/from/USD/to/INR/quantity/10

now, upper case won’t work and lower case work well.





**Now, create customized routes: -**

//for custom routes-->

@Configuration

**public** **class** ApiGatewayConfiguration {

@Bean

**public** RouteLocator gatewayRouter(RouteLocatorBuilder builder) {

**return** builder.routes()

.route(p -> p.path("/get")

.filters(f -> f.addRequestHeader("MyHeader", "MyURI").addRequestParameter("Param", "MyValue"))

.uri("http://httpbin.org:80"))

.route(p -> p.path("/currency-exchange/\*\*").uri("lb://currency-exchange"))

.route(p -> p.path("/currency-conversion/\*\*").uri("lb://currency-conversion"))

.route(p -> p.path("/currency-conversion-feign/\*\*").uri("lb://currency-conversion")).route(

p -> p.path("/currency-conversion-new/\*\*")

.filters(f -> f.rewritePath("/currency-conversion-new/(?<segment>.\*)",

"/currency-conversion-feign/${segment}"))

.uri("lb://currency-conversion"))

.build();

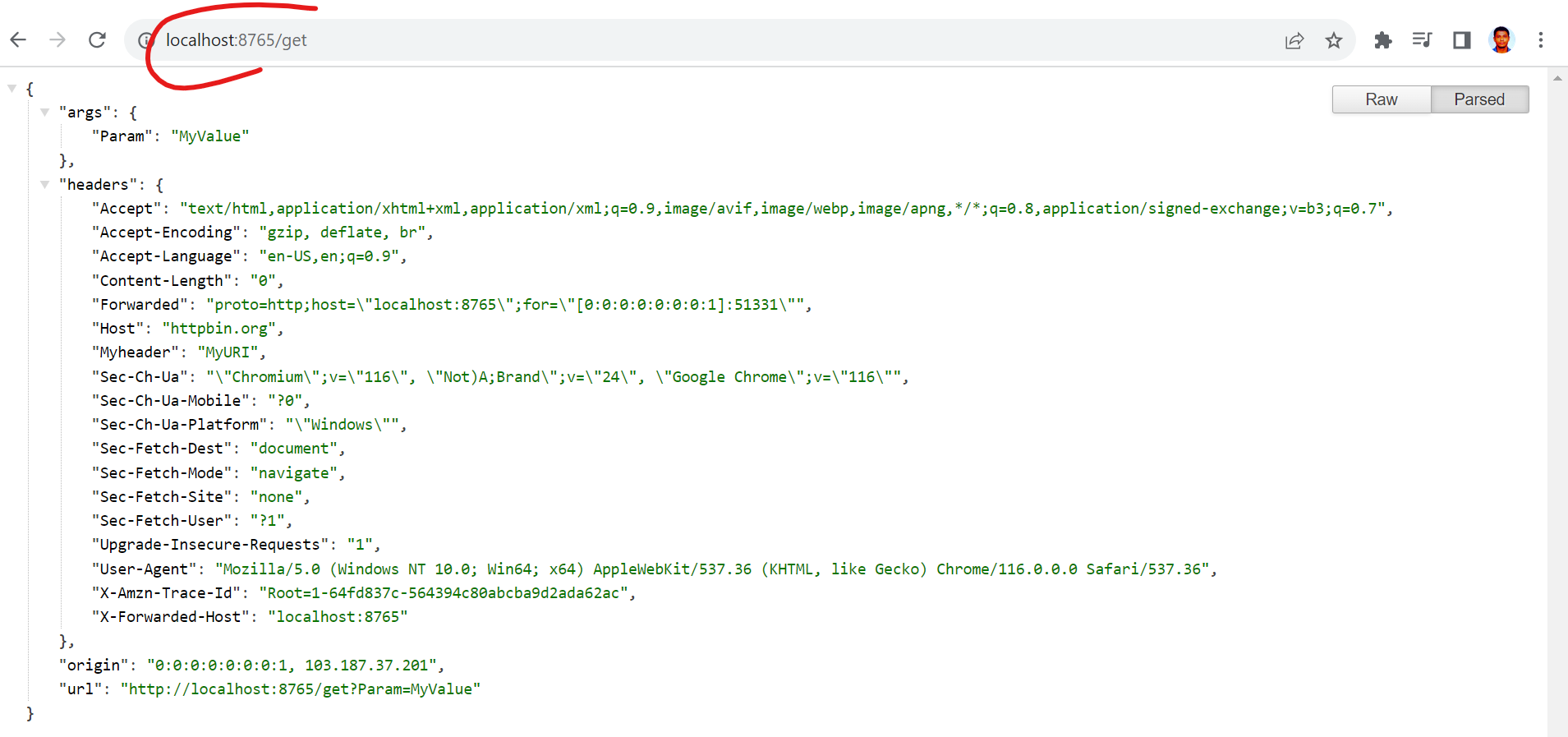
}

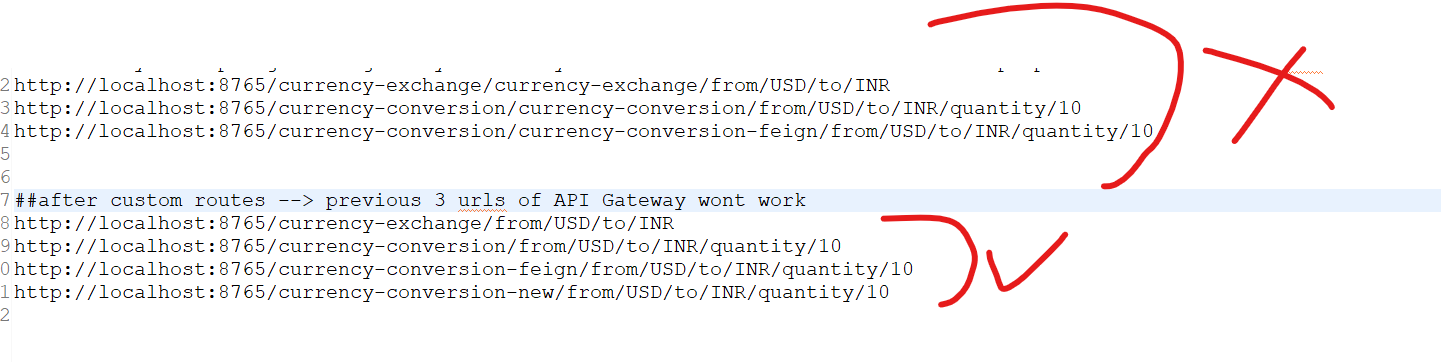
}

a/c to our implementation if we will get hit

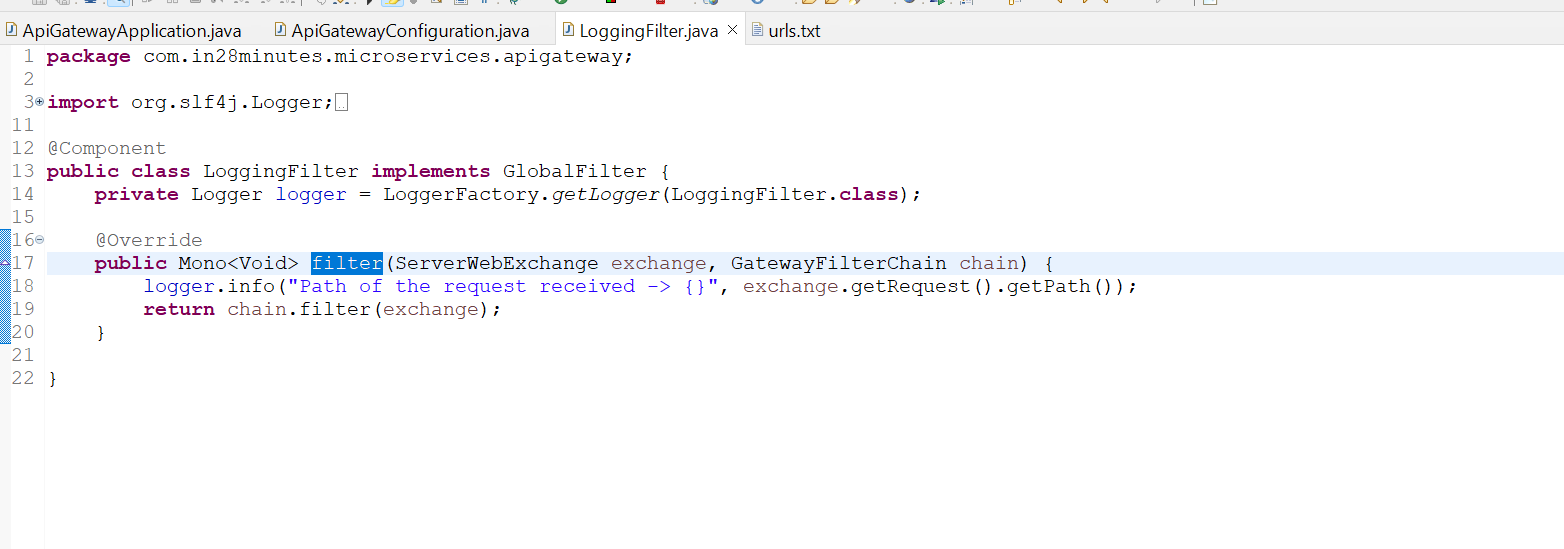
<http://localhost:8765/> it will fetch response from http://httpbin.org:80

and response looks like : -

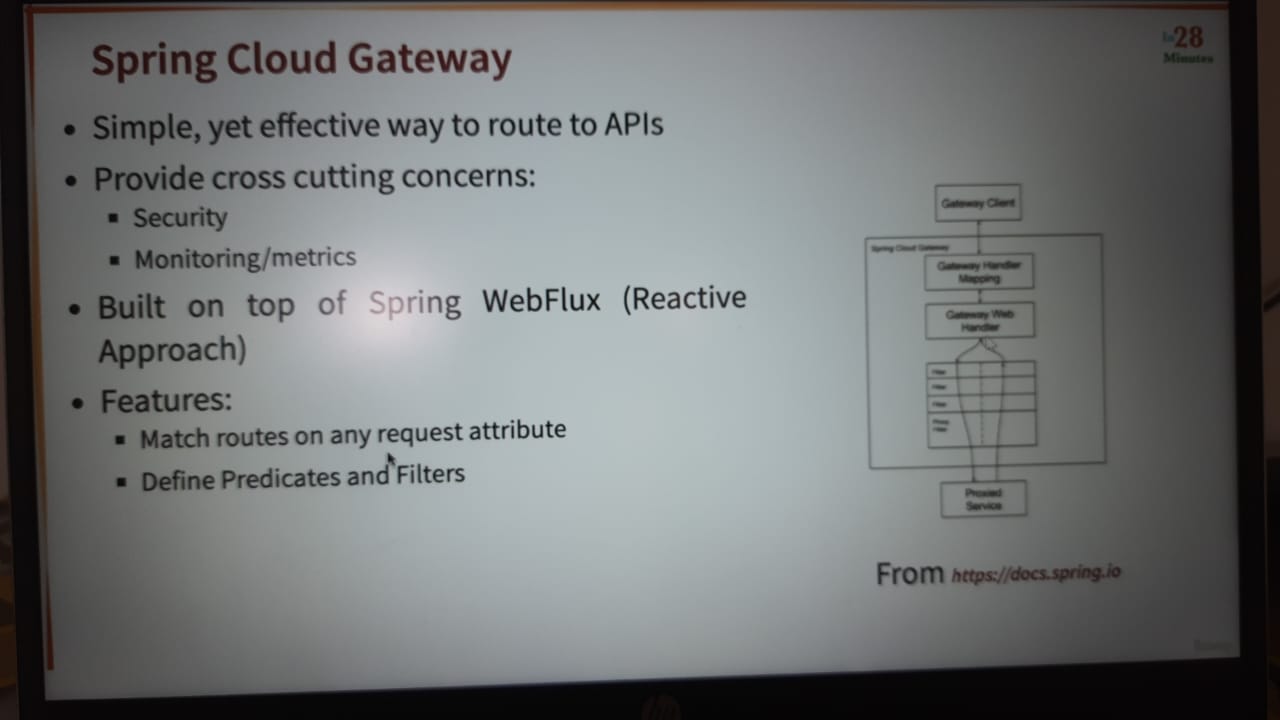




Now, will implement spring cloud gateway logging filter



Conclusion of api gateway:-



---------