DIGITALHOUSE

Configuración de proyecto Gateway

Certified Tech Developer
The Ultimate Degree

Antes de arrancar

Primero debemos crear un proyecto en nuestro IDE de desarrollo y agregar la siguiente dependencia dentro del **pom.xml**:

```
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>
```

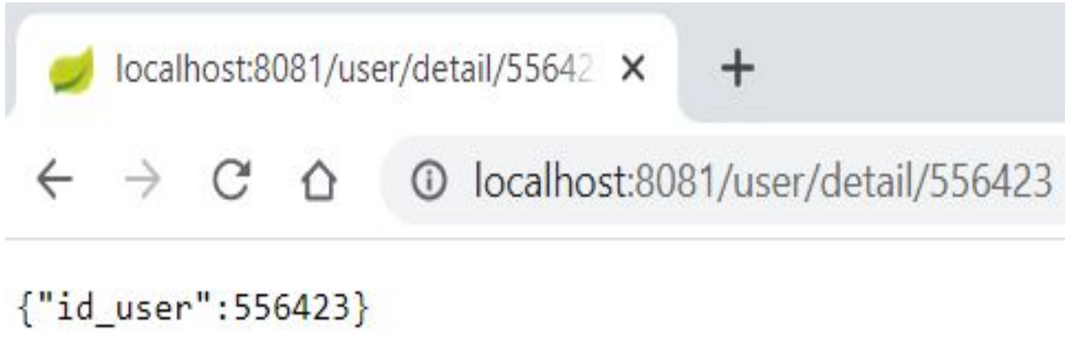
Configuración de reglas de navegabilidad en el Gateway (routes y predicates)

Luego, debemos especificar las reglas que van a regir en el Gateway analizando la información del request. Esto lo podemos realizar tanto programáticamente como por configuración de aplicación.

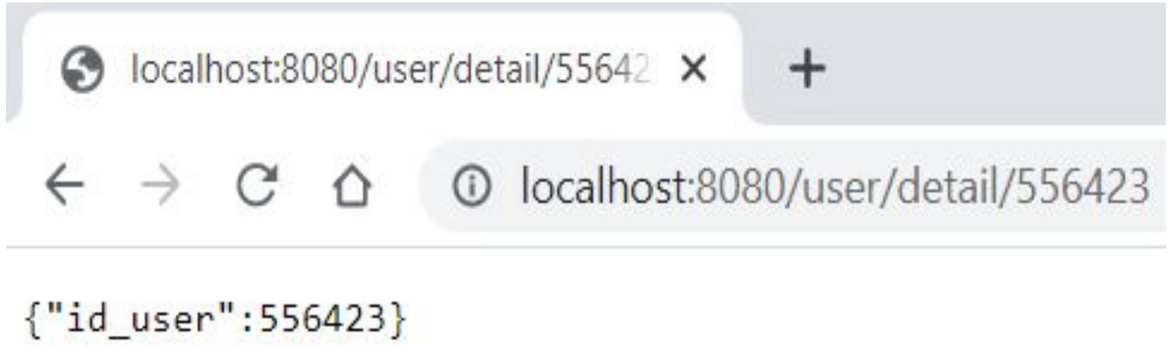
En esta materia lo haremos a través de reglas definidas en el archivo **application.yml** dado que el mismo puede ser externalizado por **Spring Cloud Config** y no depende de una recompilación para modificar una regla dada.

```yaml
server:
  port: 8080

server:
  cloud:
    gateway:
      routes:
        - id: productRoute #identificador de la ruta
          uri: http://localhost:8082 #URL donde se hará el redirect
según el predicado definido
          predicates: #Reglas de análisis del request
            - Path=/product/** #path de URL de request a considerar
        - id: userRoute #identificador de la ruta
          uri: http://localhost:8081
          predicates:
            - Path=/user/**
```

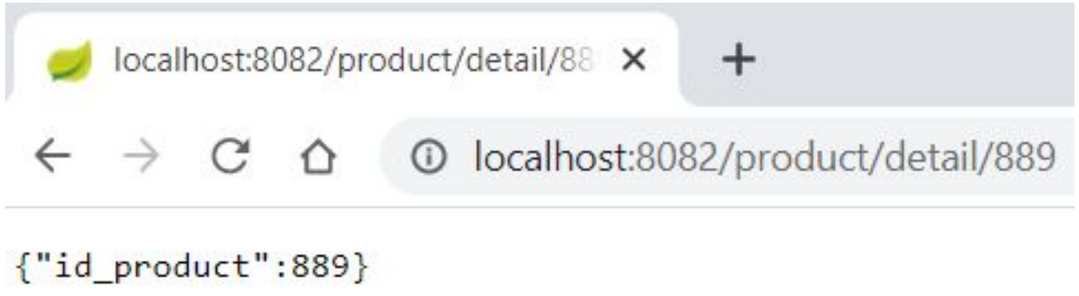
Inicializamos el microservicio de productos en el puerto **8082** y el de usuarios en el puerto **8081** para luego consumirlos de acuerdo a las reglas del **gateway** especificadas mediante HTTP desde un navegador.



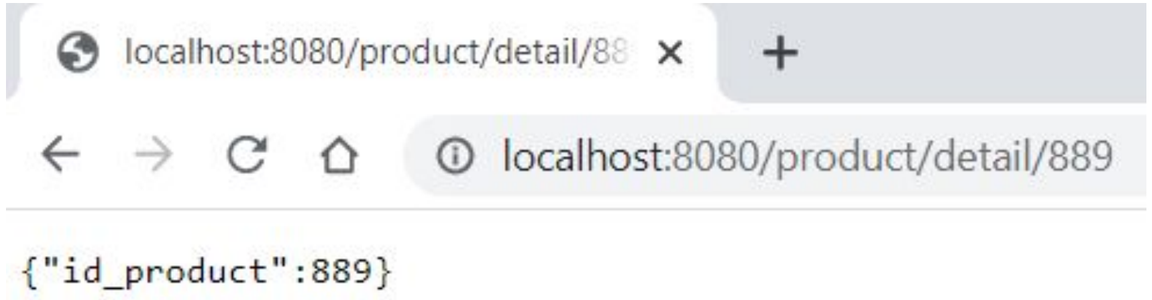
Consumimos el **microservicio de usuario** de forma directa.



Consumimos el **microservicio de usuario** de forma indirecta por Gateway.



Consumimos el **microservicio de producto** de forma directa.



Consumimos el **microservicio de producto** de forma indirecta por Gateway.

Configuración de filtros

Continuando con el ejemplo propuesto, nos resta crear los filtros previos y posteriores del servicio de usuarios y productos. En los filtros del servicio de usuario implementaremos filtros existentes en el framework para agregar información al header de request y response. Para esto editaremos nuevamente nuestro **application.yml**:

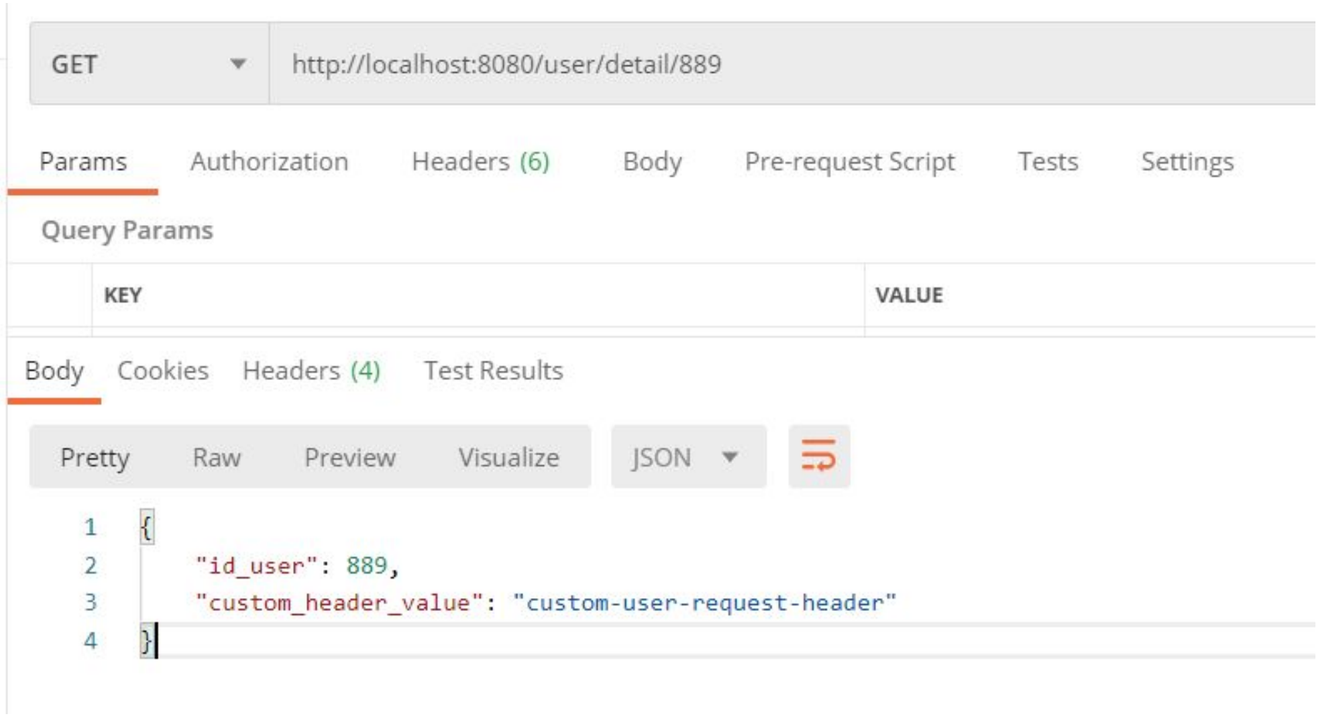
```yaml
server:
  port: 8080

server:
  cloud:
    gateway:
      routes:
        - id: productRoute #identificador de la ruta
          uri: http://localhost:8082 #URL donde se hará el redirect según el predicado definido
          predicates: #Reglas de análisis del request
            - Path=/product/** #path de url de request a considerar
        - id: userRoute #identificador de la ruta
          uri: http://localhost:8081
          predicates:
            - Path=/user/**
          filters:
            - AddRequestHeader=user-request-header, custom-user-request-header
            - AddResponseHeader=user-response-header, custom-user-response-header
```

En este ejemplo estamos agregando la propiedad **filters** informando datos de cabecera en request mediante **AddRequestHeader** y datos de cabecera en response mediante **AddResponseHeader**.

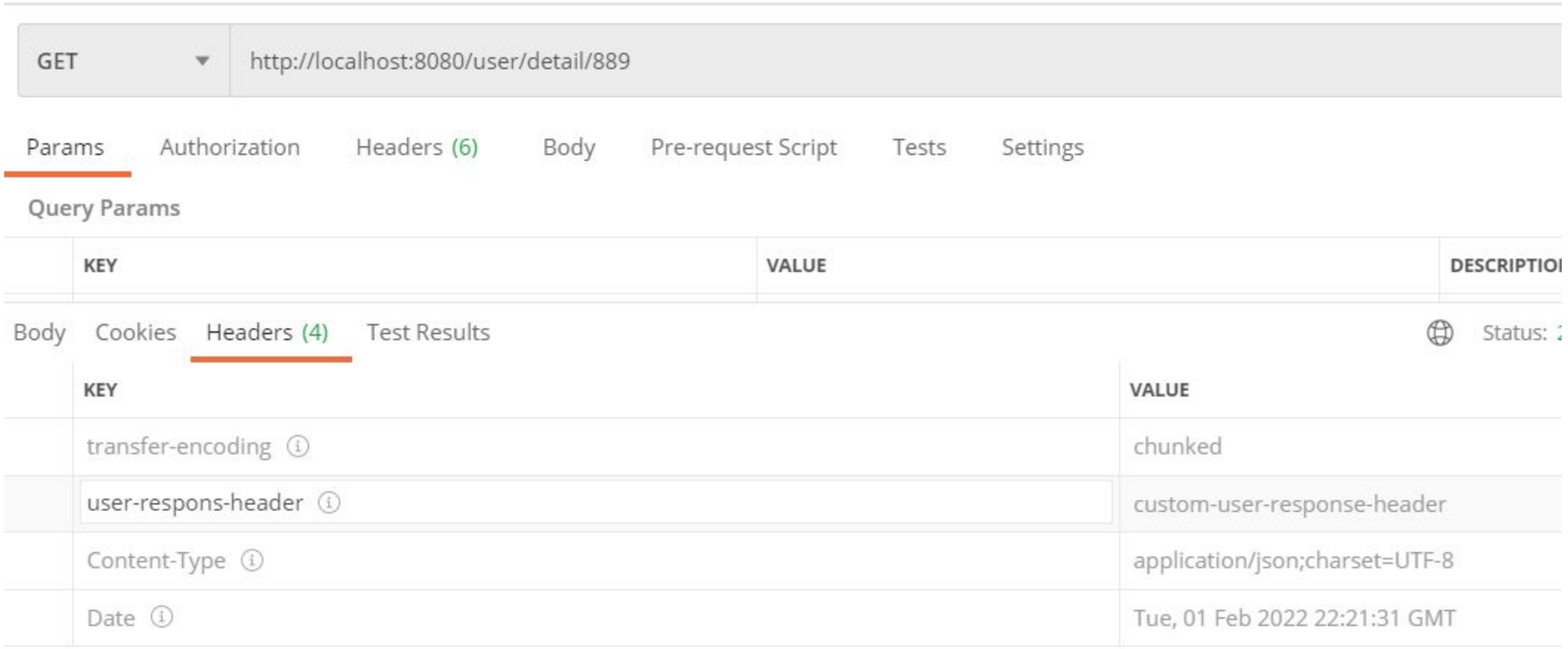
Para consumir el nuevo dato de cabecera, agregamos en nuestro servicio REST de usuarios el tratamiento de esta variable:

```
@RestController
class UserService {

    @RequestMapping(method = RequestMethod.GET, path = “/user/detail/{id}”)
    public Map<String, Object> detail(@PathVariable(“id”) Long idUser,
          @RequestHeader(“user-request-header”) String header) {
        Map<String, Object> response = new HashMap<>();
        response.put(“id_user”, idUser);
        response.put(“custom_header_value”, header);
    }
}
```

Ahora, probemos el consumo de este endpoint del microservicio de usuarios visualizando las cabeceras custom, tanto en el request como en el response, utilizando **Postman** como herramienta para realizar peticiones HTTP. El request header lo devolvemos como dato del servicio de usuario:



El response header lo podemos visualizar en los datos de cabecera de la respuesta HTTP:



Configuración de filtros personalizados

Tenemos casi todo nuestro caso de uso con Cloud Gateway implementado. Nos restan los filtros que planificamos para el servicio de productos. Como dijimos inicialmente, la principal ventaja del patrón Edge Server es la capacidad de trabajar con ***cross-cutting concerns***. Agreguemos entonces una capacidad genérica de loguear todas las requests que llegan al **Cloud Gateway**, independientemente del origen de la consulta.

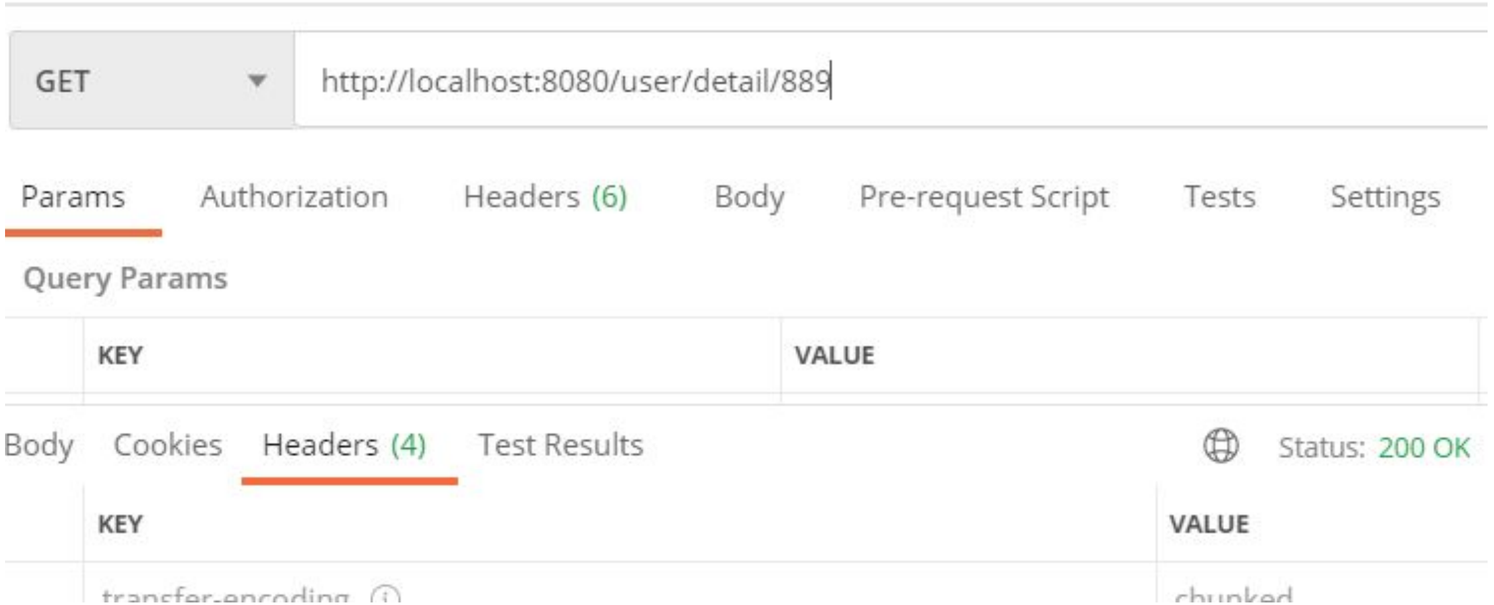
Para esto implementaremos un filtro global desarrollando una clase que implemente Log4j para loguear los datos del header en un filtro previo y loguear los datos de la hora de respuesta en un filtro posterior. Dicha clase deberá heredar de la factoría de filtros que provee Cloud Gateway (**AbstractGatewayFilterFactory**) donde debemos implementar el comportamiento del método **public GatewayFilter apply(Config config)**.

```java
@Component
public class LogFilter extends AbstractGatewayFilterFactory<LogFilter.Config> {

    private static Logger log = Logger.getLogger(LogFilter.class.getName());

    public LogFilter() {
        super(Config.class);
    }

    @Override
    public GatewayFilter apply(Config config){
        return (exchange, chain) -> {
            //Filtro previo a la invocación del servicio real asociado al gateway
        log.info("Path requested: " + exchange.getRequest().getPath());
    return chain.Filter(exchange).then(Mono.fromRunnable(() -> {
            //Filtro posterior a la invocación del servicio real asociado al gateway
            log.info("Time response: " + Calendar.getInstance().getTime());
    }));
        };
    }

    public static class Config {
        //Put the configuration properties
    }
}
```

Ahora debemos referenciar nuestro filtro personalizado en el archivo **application.yml** dentro de la propiedad **default-filters** de nuestro proyecto:

```yaml
server:
  port: 8080

server:
  cloud:
    gateway:
      default-filters: #Filtro por defecto de todas las requests realizadas al gateway
        - name: LogFilter
      routes:
        - id: productRoute #identificador de la ruta
          uri: http://localhost:8082 #URL donde se hará el redirect según el predicado definido
          predicates: #Reglas de análisis del request
            - Path=/product/** #path de URL de request a considerar
        - id: userRoute #identificador de la ruta
          uri: http://localhost:8081
          predicates:
            - Path=/user/**
          filters:
            - AddRequestHeader=user-request-header, custom-user-request-header
            - AddResponseHeader=user-response-header, custom-user-response-header
```

Si volvemos a llamar al microservicio de usuarios desde Postman, veremos por consola el login personalizado agregado.




```
GatewayApplication
2022-02-01 20:46:23.271  INFO 15956 --- [ctor-http-nio-3] com.example.service.LogFilter           : Path requested : /user/detail/889
2022-02-01 20:46:24.321  INFO 15956 --- [ctor-http-nio-3] com.example.service.LogFilter           : Time Response : Tue Feb 01 20:46:24 ART 2022
```

DIGITALHOUSE

¡Muchas gracias!