DIGITALHOUSE

Configuração de projeto Gateway

Certified Tech Developer
The Ultimate Degree

Antes de começar

Primeiro, precisamos criar um projeto em nossa IDE de desenvolvimento e adicionar a seguinte dependência dentro da **pom.xml**:

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

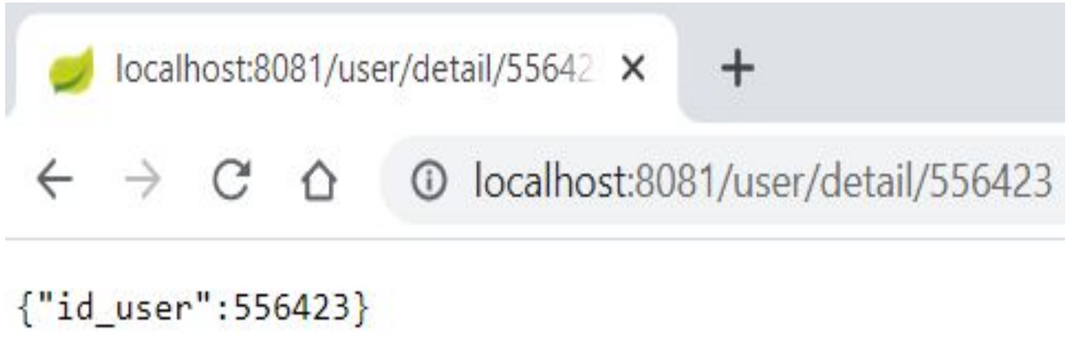
Configuração das regras de navegabilidade no Gateway (routes e predicates)

Em seguida, devemos especificar as regras que vão reger o Gateway analisando as informações da request, podemos fazê-lo tanto programaticamente quanto pela configuração do aplicativo.

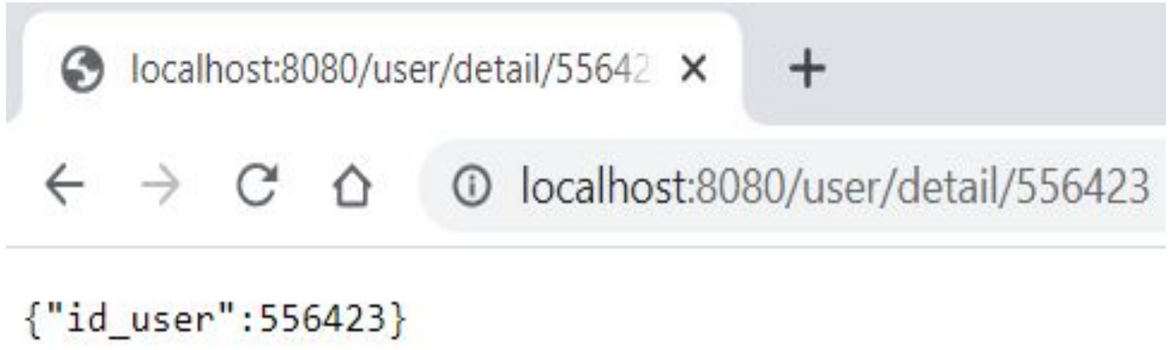
Neste caso, faremos através de regras definidas no arquivo **application.yml**, já que pode ser externalizado pelo **Spring Cloud Config** e não depende de uma recompilação para modificar uma determinada regra.

```yaml
server:
  port: 8080

server:
  cloud:
    gateway:
      routes:
        - id: productRoute #identificador da rota
          uri: http://localhost:8082 #URL onde se fará o redirect
conforme o predicado definido
          predicates: #Regras de análises da request
            - Path=/product/** #path de URL da request a considerar
        - id: userRoute #identificador da rota
          uri: http://localhost:8081
          predicates:
            - Path=/user/**
```

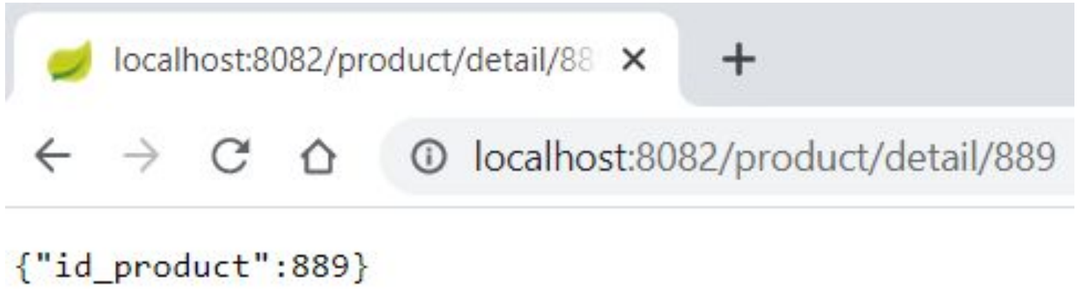
Inicializamos o microsserviço do produto na porta **8082** e o microsserviço do usuário na porta **8081** e depois consumimos as regras de **gateway** especificadas via HTTP a partir de um navegador.



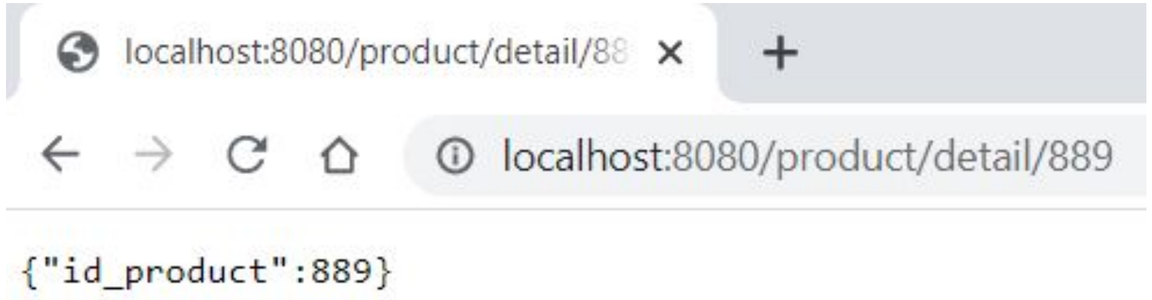
Consumimos o microsserviço do usuário diretamente.



Consumimos o **microsserviço do usuário** indiretamente através do Gateway.



Consumimos o **microsserviço do usuário** diretamente.



Consumimos o **microsserviço do usuário** indiretamente através do Gateway.

Configurações de filtro

Continuando com o exemplo proposto, precisamos criar os pré e pós-filtros para o serviço ao usuário e produto. Nos filtros de serviço ao usuário, implementaremos os filtros existentes na framework para adicionar informações ao header de request e response. Para isso, editaremos novamente nosso **application.yml:**

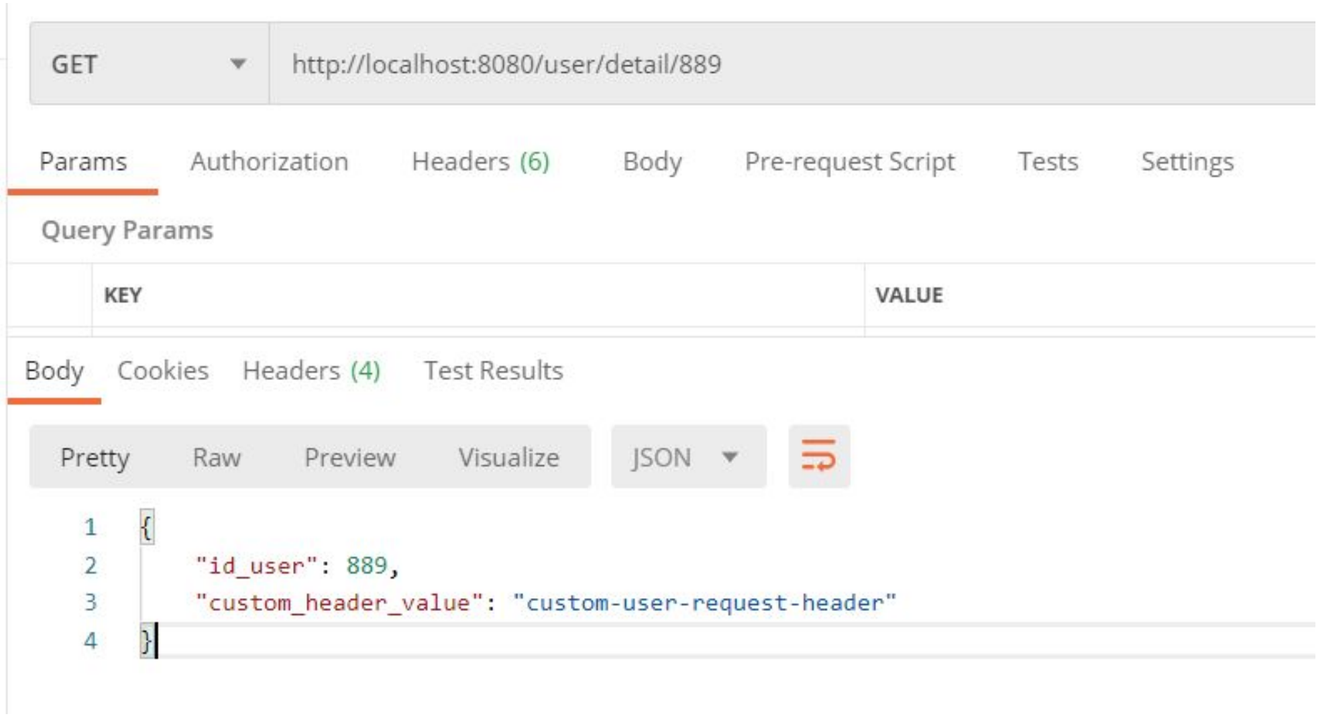
Neste exemplo, estamos adicionando a propriedade **filters** através do relatório de dados de cabeçalho em request via **AddRequestHeade**r e dados de cabeçalho em response via **AddResponseHeader**.

```yaml
server:
  port: 8080

server:
  cloud:
    gateway:
      routes:
        - id: productRoute #identificador da rota
          uri: http://localhost:8082 #URL onde se fará o redirect conforme o predicado definido
          predicates: #Regras de análises da request
            - Path=/product/** #path de url da request a considerar
        - id: userRoute #identificador da rota
          uri: http://localhost:8081
          predicates:
            - Path=/user/**
          filters:
            - AddRequestHeader=user-request-header, custom-user-request-header
            - AddResponseHeader=user-response-header, custom-user-response-header
```

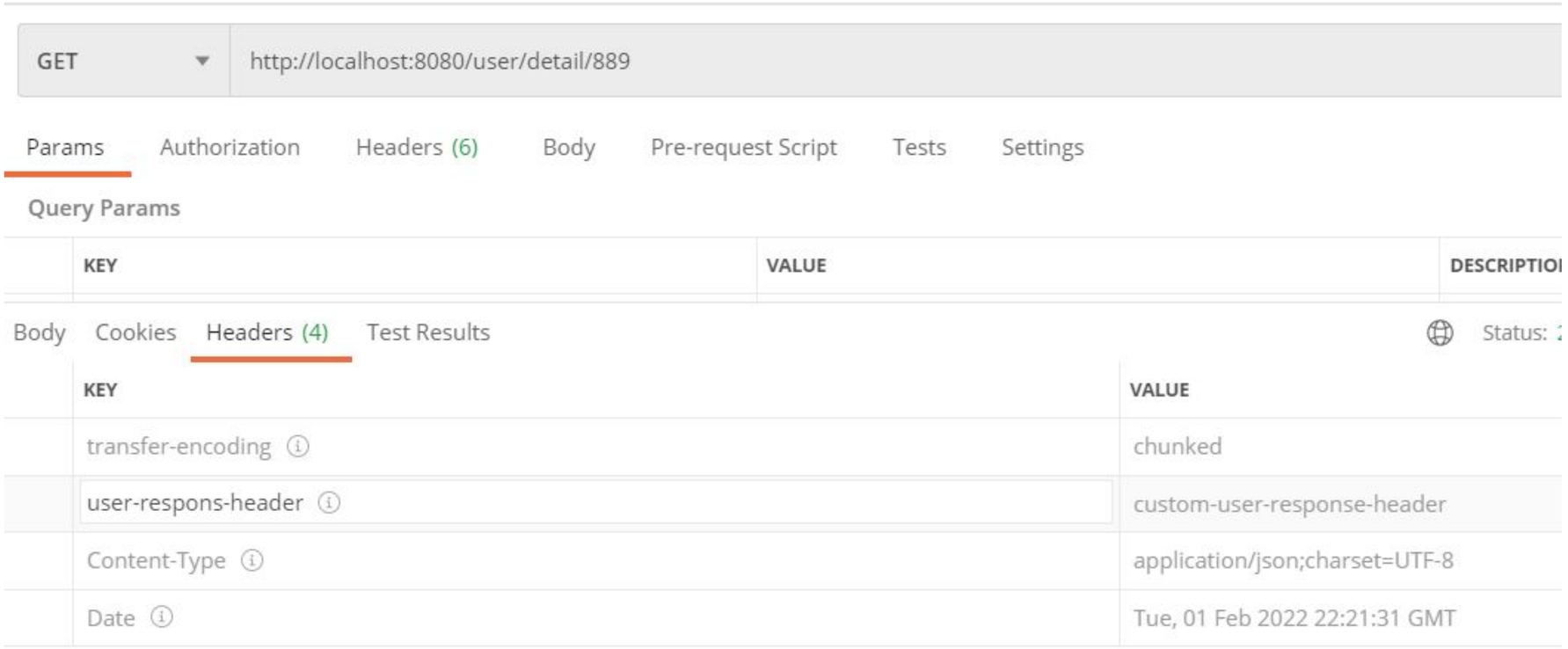
A fim de consumir os novos dados do cabeçalho, adicionamos em nosso serviço de usuário REST o tratamento desta variável:

```
@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);
    }
}
```

Agora, vamos testar o consumo deste endpoint de microsserviço do usuário visualizando os cabeçalhos custom, tanto na request quanto na response, usando o **Postman** como uma ferramenta para fazer pedidos HTTP. O request header é devolvido como dados de serviço do usuário:



O response header pode ser visto nos dados do cabeçalho da resposta HTTP:



Configurações personalizadas de filtros

Temos quase todo o nosso caso de uso com o Cloud Gateway implementado. Subtraímos os filtros que planejamos para o serviço do produto. Como dissemos inicialmente, a principal vantagem deste padrão de Edge Server é a capacidade de trabalhar com **cross-cutting concerns**. Vamos adicionar então uma capacidade genérica de registrar todas as requests que chegam ao **Cloud Gateway,** independentemente da origem da consulta.

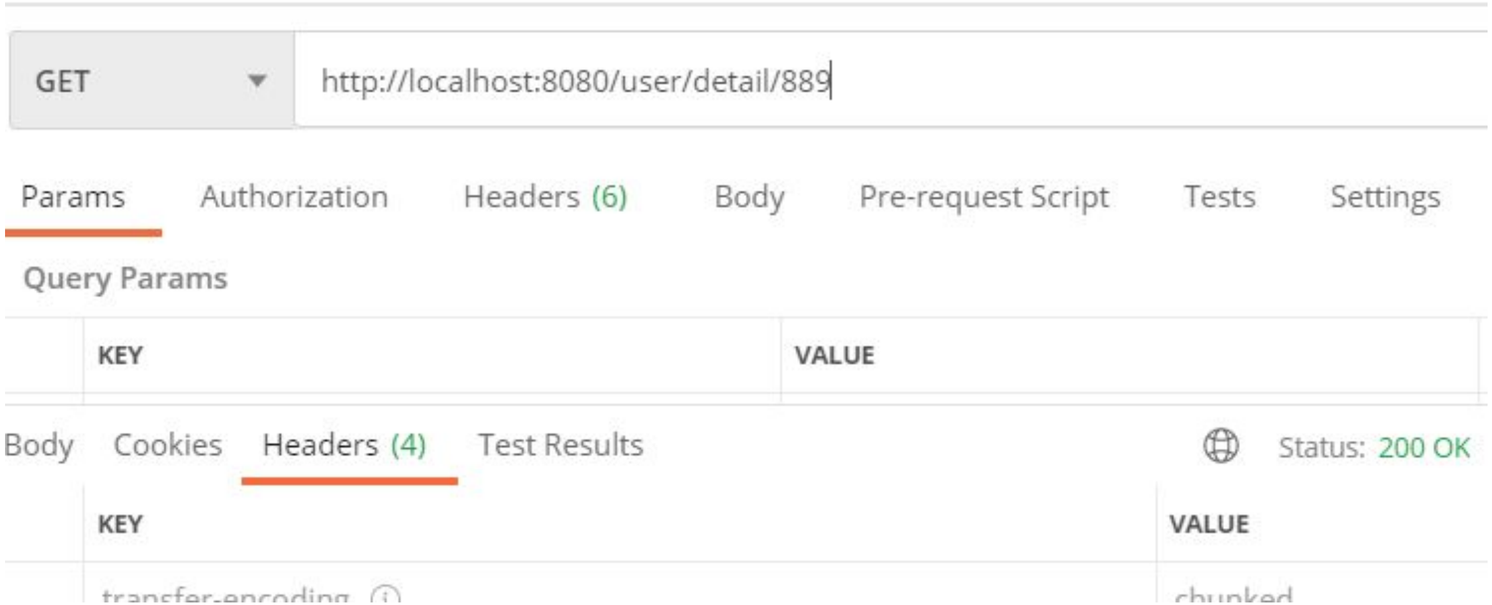
Para isso, implementaremos um filtro global desenvolvendo uma classe que implemente Log4j para registrar dados de header em um pré-filtro e dados de tempo de resposta de registro em um pós-filtro. Esta classe deve herdar da fábrica de filtros fornecida pela Cloud Gateway **(AbstractGatewayFilterFactory)**, onde devemos implementar o comportamento do 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 prévio à invocação do serviço real associado ao gateway
        log.info("Path requested: " + exchange.getRequest().getPath());
    return chain.Filter(exchange).then(Mono.fromRunnable(() -> {
            //Filtro posterior à invocação do serviço real associado ao gateway
            log.info("Time response: " + Calendar.getInstance().getTime());
    }));
        };
    }

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

Agora, devemos fazer referência a nosso filtro personalizado no arquivo **application.yml** dentro da propriedade **default-filters** de nosso projeto:

```yaml
server:
  port: 8080

server:
  cloud:
    gateway:
      default-filters: #Filtro por default de todas as requests realizadas ao gateway
        - name: LogFilter
      routes:
        - id: productRoute #identificador da rota
          uri: http://localhost:8082 #URL onde se fará o redirect conforme o predicado definido
          predicates: #Regras de análises da request
            - Path=/product/** #path de URL da request a considerar
        - id: userRoute #identificador da rota
          uri: http://localhost:8081
          predicates:
            - Path=/user/**
          filters:
            - AddRequestHeader=user-request-header, custom-user-request-header
            - AddResponseHeader=user-response-header, custom-user-response-header
```

Se chamarmos novamente o microsserviço de usuário do Postman, veremos o log in personalizado adicionado via console.




```
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

Muito obrigado!