



# Introdução a Web Services REST com Spring Framework

*Parte 7 → Content Negotiation*

Prof. Me. Jorge Luís Gregório  
[www.jlgregorio.com.br](http://www.jlgregorio.com.br)



# Agenda

- O que é Content Negotiation?
- Formas de prover Content Negotiation
- **Prática 01**
  - Configurando as dependências no arquivo *pom.xml* para suportar o formato XML
    - Criando um arquivo de configuração para suportar Content Negotiation em XML via Query Param
  - Alterando o Controller da aplicação.
  - Testando via Postman

- **Prática 02**

- Provendo Content Negotiation via Request Header

- **Prática 03**

- Adicionando suporte à serialização em formato YAML.

A large, stylized green leaf graphic is positioned on the left side of the slide, partially overlapping the text box. It has a light green fill and a darker green outline, with a central vein and several smaller veins branching out.

# O que é Content Negotiation?

# O que os clientes esperam de sua API?

- Os clientes esperam diferentes formatos como resposta às requisições;
- Apesar do formato JSON ser o padrão do Spring e o mais usado, às vezes o cliente precisa de dados no formato XML, YAML, entre outros;
- Configurar sua API para que ela retorne diferentes formatos de dados é essencial para garantir a comunicação com o maior número de clientes;
- A especificação [RFC 2616](#) diz:
  - *HTTP has provisions for several mechanisms for “content negotiation” — the process of selecting the best representation for a given response when there are multiple representations available. (REST API Tutorial, 2021)*

# Agent vs Server

- ***Server-driven Negotiation:*** nessa abordagem é o servidor que determina o melhor formato de resposta, normalmente em JSON ou XML. Não é uma abordagem muito usada, pois é o cliente que saber qual é o melhor formato para ele utilizar;
- ***Agent-driven Negotiation:*** nessa abordagem, é enviado um parâmetro na requisição que determina o tipo de resposta que o cliente precisa, desde que esteja disponível no servidor.
- Há três maneiras de prover Content Negotiation →

# Content Negotiation via Extension

- Essa abordagem é a menos utilizada;
- Consiste em adicionar a extensão com o tipo de resposta desejado no fim da requisição.
- Exemplo:
  - <http://www.my-webservice.com/product/v1/all.json>
  - <http://www.my-webservice.com/product/v1/all.xml>

# Content Negotiation via Query Parameter

- Prover *Content Negotiation* via ***Query Parameter*** é muito simples;
- Basta informar o parâmetro (normalmente ***mediaType***) diretamente na URL de requisição com o formato desejado. Claro, o servidor deve suportar o formato informado, caso contrário a requisição retornará um erro.
- **Exemplo:**
  - <http://www.myapi/users/v1/?mediaType=xml>
  - <http://www.myapi/products/v1/?mediaType=json>

# Content Negotiation via Cabeçalhos HTTP

- Usando cabeçalhos HTTP (HTTP Request Headers): nessa abordagem, é informado o parâmetro **Content-Type** cujo valor é o formato desejado.  
Exemplos:
  - `Content-Type: application/xml`
  - `Content-Type: text/html`
- Da mesma maneira, para determinar qual é o tipo de formato esperado no cliente, utiliza-se um cabeçalho de resposta (HTTP Response Header) **Accept**. Exemplo:
  - `Accept: application/json`
- Quando não há Accept, o servidor retorna o formato padrão.



A large, stylized green leaf graphic is positioned on the left side of the slide, partially overlapping the green text box. It has a light green fill and a darker green outline.

# **Prática 01 – Content Negotiation via Query Parameter**

# Configurando o arquivo pom.xml

- O formato padrão de resposta do Spring é JSON;
- Nesse exemplo, vamos adicionar o suporte aos formatos XML e YAML;
- Abra o arquivo ***pom.xml*** e adicione as seguintes dependências:

```
<!-- Content Negotiation XML Support -->
<dependency>
  <groupId>com.fasterxml.jackson.dataformat</groupId>
  <artifactId>jackson-dataformat-xml</artifactId>
</dependency>

<!-- Content Negotiation YAML Support -->
<dependency>
  <groupId>com.fasterxml.jackson.dataformat</groupId>
  <artifactId>jackson-dataformat-yaml</artifactId>
</dependency>
```

# Criando um arquivo de configuração

- No pacote principal da aplicação, crie um novo pacote chamado ***config***;
- Neste pacote, crie uma classe Java chamada **WebConfig**;
- Essa classe deve implementar a interface *WebMvcConfigurer*, que possui o método *configureContentNegotiation*, que iremos sobrescrever.
- Ela também deve possuir duas anotações: **@Configuration** (é uma classe de configuração) e **@EnableMvc** (habilita o suporte ao Spring MVC e suas classes de configuração)
- O código dessa classe está no próximo slide.

# WebConfig.java

O código a seguir define o *Content Negotiation* para as requisições GET via **Query Parameter.**

As requisições POST ou PUT, o corpo da requisição deve estar em formato XML.

```
@Configuration
@EnableWebMvc
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void configureContentNegotiation(ContentNegotiationConfigurer configurer){
        configurer.favorParameter(true)
                    .parameterName("mediaType") //..query parameter name
                    .ignoreAcceptHeader(true)
                    .useRegisteredExtensionsOnly(false)
                    .defaultContentType(MediaType.APPLICATION_JSON) //..default format
                    .mediaType("xml", MediaType.APPLICATION_XML) //..XML format
                    .mediaType("json", MediaType.APPLICATION_JSON); //..JSON format
    }
}
```

# Configurando o Controller

- Agora é necessário ir até o Controller e definir o suporte aos diferentes tipos de formato.
- Nos métodos GET é necessário configurar a anotação **@GetMapping** com o parâmetro **produces**. Exemplo:

```
@GetMapping(value = "/{id}", produces = {"application/json", "application/xml"})
```

- Nos métodos POST e PUT, é necessário usar também o parâmetro **consumes**, indicando que ele pode consumir diferentes tipos de conteúdo. Exemplo:

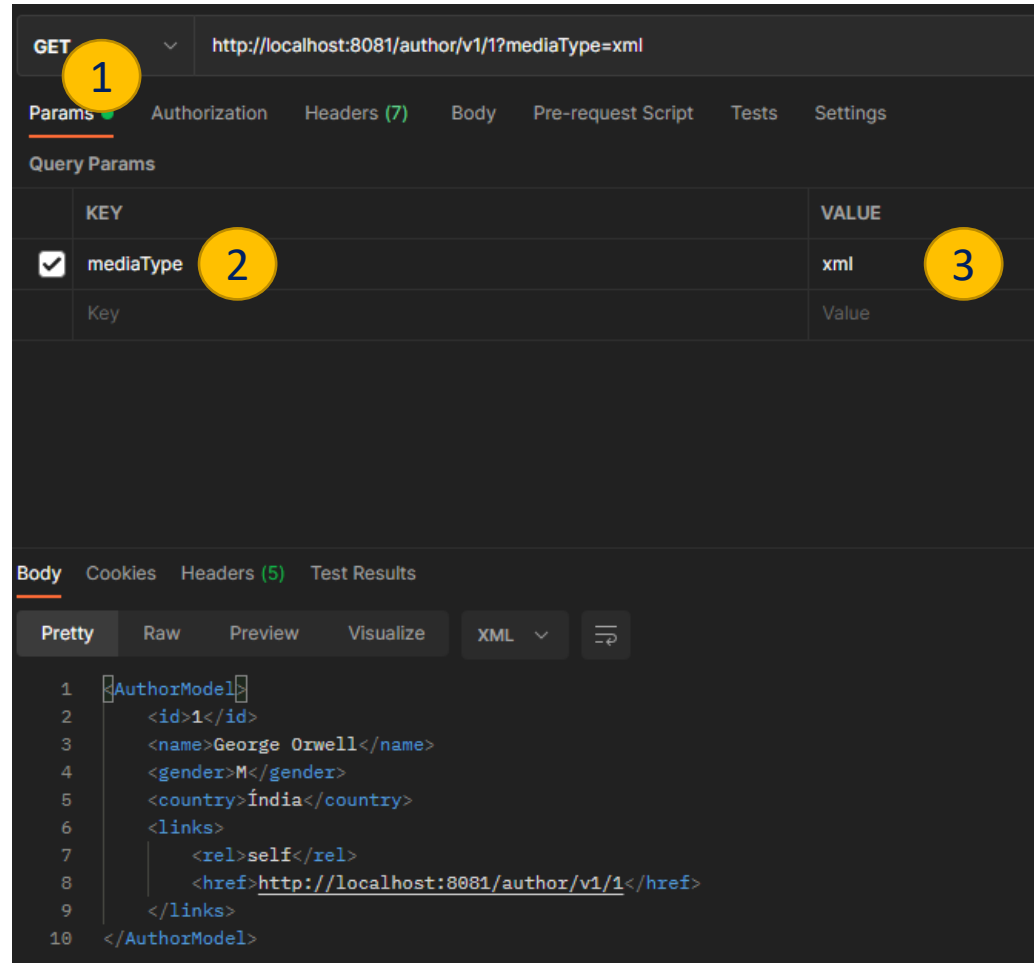
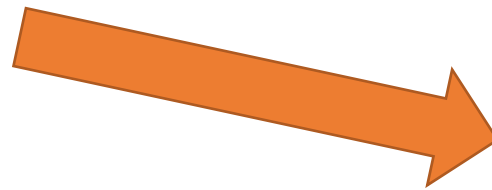
```
@PostMapping(produces = {"application/json", "application/xml"}, consumes = {"application/json", "application/xml"})
```

# Testando no Postman

- Verbo GET

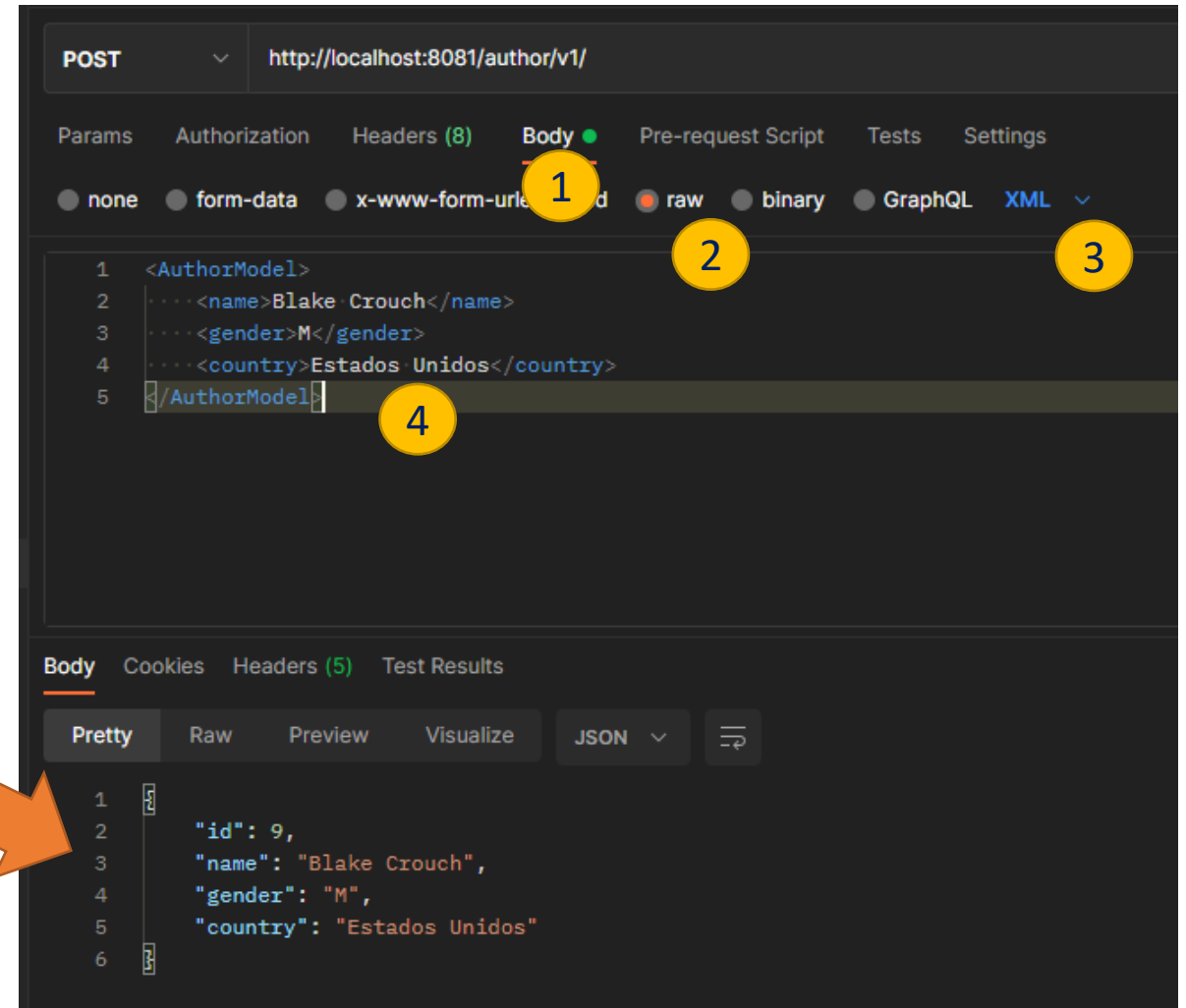
1. Abra a guia **Params**;
2. Adicione o parâmetro **mediaType**;
3. O valor desse parâmetro é **xml**

Faça a requisição:



# Testando no Postman

- Verbo POST
  1. Clique na guia **Body**;
  2. Selecione raw;
  3. Depois o tipo XML;
  4. Adicione o formato XML do novo recurso;
- Faça a requisição e verifique o retorno.



A decorative graphic on the left side of the slide, consisting of several overlapping, stylized green leaves or petals in various shades of green, creating a circular, organic shape.

## Prática 02 – Content Negotiation via Request Header



# Configurando a classe WebConfig.java

- Para suportar o Content Negotiation via *request header*, basta alterar a classe **WebConfig.java**.
- Vamos remover o método **parameterName** e, no método **ignoreAcceptHeader**, vamos mudar para true. Veja:

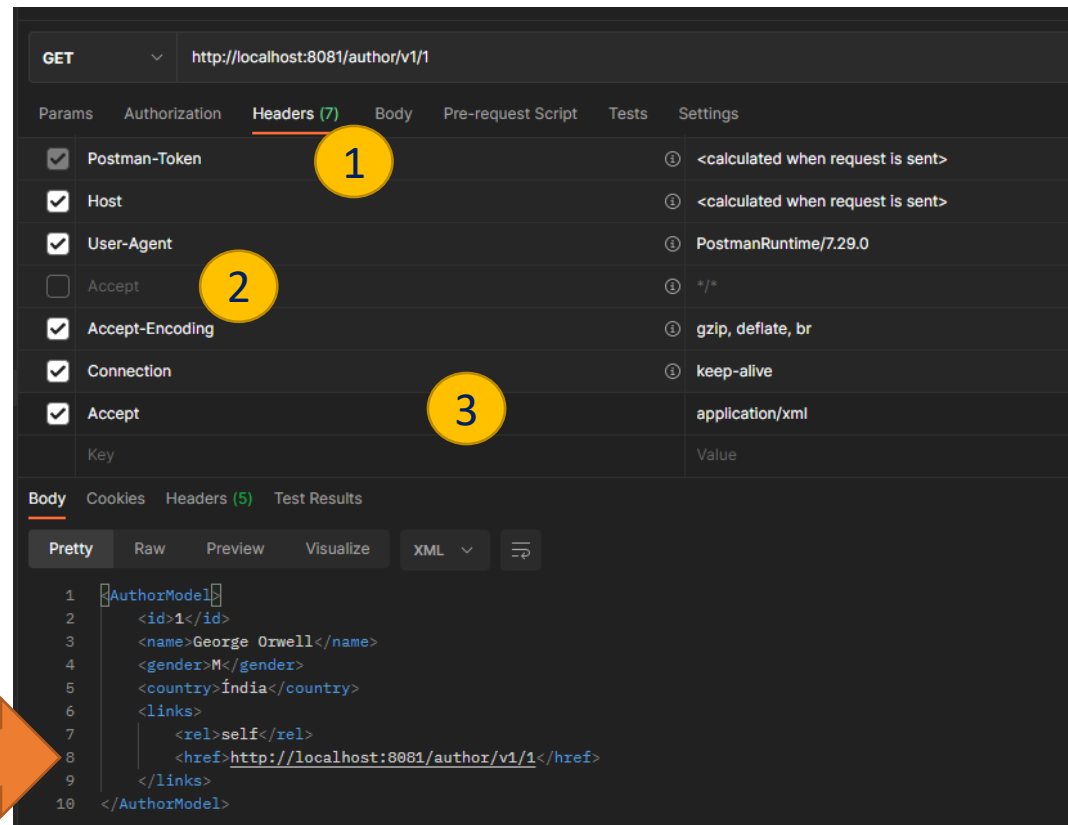
```
@Override
public void configureContentNegotiation(ContentNegotiationConfigurer configurer){
    configurer.favorParameter(true)
        .ignoreAcceptHeader(false) //.. remove parameterName method and don't ignore Accept Header
        .useRegisteredExtensionsOnly(false)
        .defaultContentType(MediaType.APPLICATION_JSON)
        .mediaType("xml", MediaType.APPLICATION_XML)
        .mediaType("json", MediaType.APPLICATION_JSON);
}
```


# Testando no Postman

- Verbo GET

1. Abra a guia **Headers**;
2. Desabilite o *header Accept* padrão;
3. Adicione um novo *header Accept* e coloque o valor *application/xml*

Faça a requisição:





## Prática 03 – Suportando serialização YAML

# O que é YAML?

- YAML (pronuncial-se “iâmél”) é um formato de arquivo legível por humanos usado como uma alternativa para os formatos JSON e XML;
- Suas principais características são a simplicidade e a legibilidade;
- O formato YAML é usado principalmente em:
  - Arquivos de configuração;
  - Compartilhamento de dados entre sistemas;
  - Mensagens entre processos;
- A versão atual é a 1.2.2 (outubro/2021)
- Para mais informações, visite o site oficial: <https://yaml.org/>

Exemplo de arquivo em formato YAML:

```
client:
  name: João
  age: 30
  gender: Masculino
  profession: Programador
  dependent:
    name: Maria
    gender: Feminino
```

O mesmo arquivo em formato JSON fica assim:

```
{
  "client": {
    "name": "João",
    "age": 30,
    "gender": "Masculino",
    "profissao": "Programador",
    "dependent": {
      "name": "Maria",
      "gender": "Feminino"
    }
  }
}
```

# Configurando o arquivo pom.xml

- Para suportar Content Negotiation em formato YAML é necessário adicionar a seguinte dependência:

```
<!-- Content Negotiation YAML Support -->  
<dependency>  
    <groupId>com.fasterxml.jackson.dataformat</groupId>  
    <artifactId>jackson-dataformat-yaml</artifactId>  
</dependency>
```

# Criando um conversor para YAML

- Crie um novo pacote chamado `serialization.converter`
- Dentro deste pacote, crie uma classe chamada ***YamlJackson2HttpMessageConverter***
- Essa classe deve estender a classe ***AbstractJackson2HttpMessageConverter***
- Veja o código completo:

```
package br.com.jlgregorio.mybooks.serialization.converter;

import com.fasterxml.jackson.dataformat.yaml.YAMLMapper;
import org.springframework.http.MediaType;
import org.springframework.http.converter.json.AbstractJackson2HttpMessageConverter;

public class YamlJackson2HttpMessageConverter extends
AbstractJackson2HttpMessageConverter {

    public YamlJackson2HttpMessageConverter(){
        super(new YAMLMapper(), MediaType.parseMediaType("application/x-yaml"));
    }

}
```

# Editando a classe WebConfig.java

- Agora precisamos fazer 3 coisas na classe WebConfig.java:
  1. Criar um tipo para a media type x-yaml;
  2. Criar um método que intercepte a requisição em YAML e faça a conversão;
  3. Alterar o método *configureContentNegotiation* para retornar YAML.
- O código completo está no slide a seguir:

```

package br.com.jlgregorio.mybooks.config;

import br.com.jlgregorio.mybooks.serialization.converter.YamlJackson2HttpMessageConverter;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.MediaType;
import org.springframework.http.converter.HttpMessageConverter;
import org.springframework.web.servlet.config.annotation.ContentNegotiationConfigurer;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

import java.util.List;

@Configuration
@EnableWebMvc
public class WebConfig implements WebMvcConfigurer {

    //..add a media type
    private static final MediaType MEDIA_TYPE_YAML = MediaType.valueOf("application/x-yaml");

    //..add a intercetor
    public void extendMessageConverters(List<HttpMessageConverter<?>> converters){
        converters.add(new YamlJackson2HttpMessageConverter());
    }

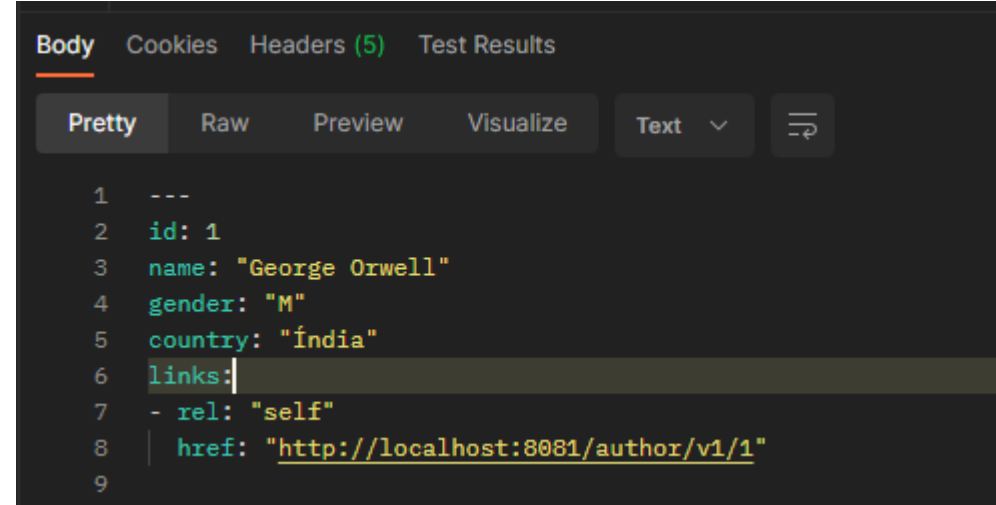
    @Override
    public void configureContentNegotiation(ContentNegotiationConfigurer configurer){
        configurer.favorParameter(true)
            .ignoreAcceptHeader(false) //.. remove parameterName method and don't ignore Accept Header
            .useRegisteredExtensionsOnly(false)
            .defaultContentType(MediaType.APPLICATION_JSON)
            .mediaType("xml", MediaType.APPLICATION_XML)
            .mediaType("json", MediaType.APPLICATION_JSON)
            .mediaType("yaml", MEDIA_TYPE_YAML); //..adding YAML support
    }
}

```



# Ajustando o Controller - GET

- Para que a API retorne YAML é necessário ajustar os métodos do controller.
- Para isso, basta adicionar aos métodos que estão anotados com **@GetMapping** o suporte ao YAML.
- Exemplo:
  - `@GetMapping(produces = {"application/json", "application/xml", "application/x-yaml"})`
- Faça o teste no Postman ajustando o *Query Parameter* ou o *Request Header* para que a resposta seja em formato YAML.



The screenshot shows the Postman interface with the 'Body' tab selected. The response is displayed in 'Pretty' format. The JSON object contains the following data:

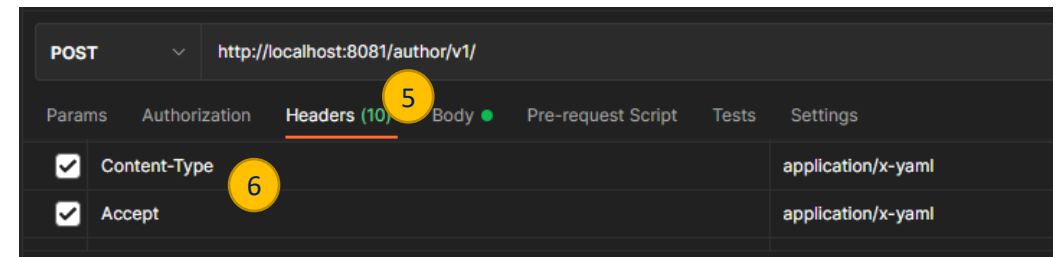
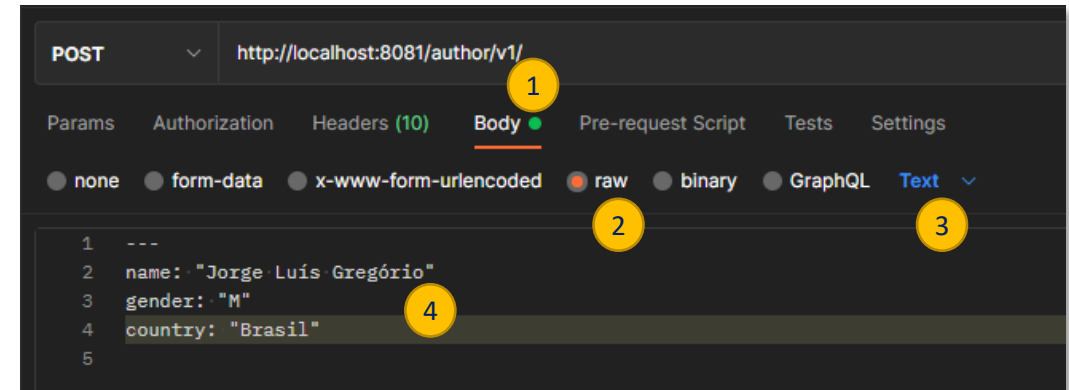
```
1  ---
2  id: 1
3  name: "George Orwell"
4  gender: "M"
5  country: "Índia"
6  links:
7    - rel: "self"
8      href: "http://localhost:8081/author/v1/1"
9
```

# Ajustando o Controller – POST e PUT

- Ajuste os métodos POST e PUT para consumir e produzir YAML. Veja o exemplo:

```
@PostMapping(produces = {"application/xml", "application/json",  
"application/x-yaml"}, consumes = {"application/xml", "application/json",  
"application/x-yaml"})
```

- Agora vamos Testar Via POSTMAN
- Para enviar dados em formato YAML é necessário:
  1. Clicar na aba Body;
  2. Definir o tipo raw;
  3. Deixar no formato text – o POSTMAN não suporta envio em YAML;
  4. Definir o conteúdo a ser enviado;
  5. Clicar na aba Headers
  6. Definir dois cabeçalhos:
    - Content-Type = application/x-yaml
    - Accept = application/x-yaml



# Conclusão

- Fazer com que sua API produza e consuma diferentes tipos de formatos é essencial para torná-la acessível a múltiplos clientes;
- Os principais formatos são JSON e XML, mas o formato YAML é uma excelente alternativa, considerando sua simplicidade e legibilidade;

# Sobre mim

## JORGE LUÍS GREGÓRIO

- Professor da Faculdade de Tecnologia “Prof. José Camargo” – Fatec Jales, e da Escola Técnica Estadual Dr. José Luiz Viana Coutinho – Etec Jales;
  - Articulista do Jornal de Jales – Coluna “Fatecnologia”;
  - Apresentador do Tech Trends, podcast oficial da Fatec Jales;
  - Bacharel em Sistemas de Informação; Especialista em Desenvolvimento de Software para Web e Mestre em Ciência da Computação.
  - Trabalha com tecnologia desde 1998, tendo atuado como analista de suporte; administrador de redes de computadores; desenvolvedor de software, *webdesigner* e professor.
- 
- Site oficial: [www.jlgregorio.com.br](http://www.jlgregorio.com.br)
  - Perfil do LinkedIn: [www.linkedin.com/in/jlgregorio81](http://www.linkedin.com/in/jlgregorio81)
  - Currículo Lattes: <http://lattes.cnpq.br/3776799279256689>

