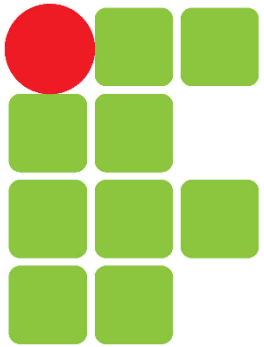


INSTITUTO FEDERAL
SÃO PAULO

Arquitetura REST para APIs Web

Desenvolvimento Web Backend (BRADWBK)

Prof. Luiz Gustavo Diniz de Oliveira Véras

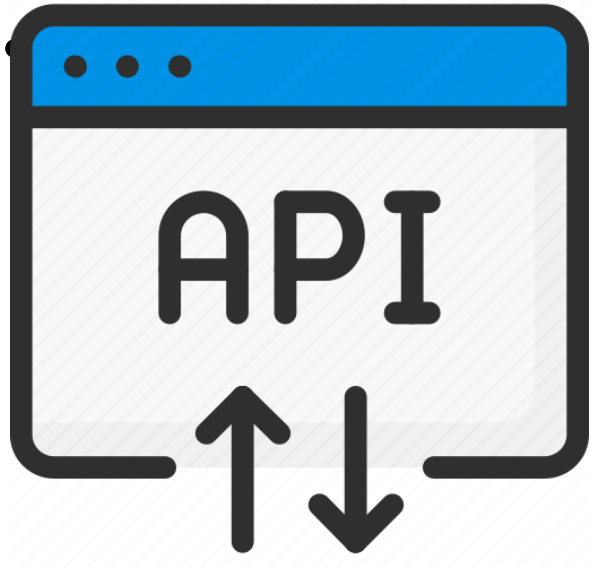
E-mail: gustavo_veras@ifsp.edu.br

Objetivos

- ✓API Web
- ✓REST
- ✓Recursos do Framework Spring para APIs REST
- ✓Criando WebServices com Spring MVC
- ✓Recursos HTTP no Spring MVC

API Web

- Uma API é a **interface** que um programa de software apresenta a outros programas, para humanos e, no caso de APIs da web, para o mundo via a Internet.
- APIs são os blocos de construção que permitem a **interoperabilidade** entre as principais plataformas de negócios na web.



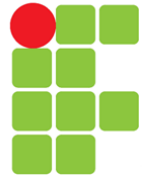
Veio da necessidade de **trocar informações** com provedores de dados que estão equipados para resolver problemas específicos, sem que outras empresas necessitem gastar tempo para resolver esse problema.

API Web

Exemplos de uso de APIs

- Criação de identidades e sua manutenção em contas de software em nuvem;
- Criação de endereços de e-mail para seu software de design colaborativo para o aplicativo da web que ajudam você a pedir entrega de pizza
- Como dados de previsão de clima são compartilhados de uma fonte confiável
- Processar seus cartões de crédito e habilitar empresas para coletar seu dinheiro perfeitamente sem se preocupar com detalhes da tecnologia financeira e suas leis e regulamentos correspondentes.

API Web



Para desenvolver uma boa API, é importante focar na sua utilidade para quem irá utilizá-la (**o cliente**). Não iremos focar nesses pontos no curso, mas podemos analisar o conselho de um profissional de TI da Uber.

> *"Uma boa API pode se resumir ao problema que você está tentando resolver e quão valioso é resolvê-lo. Você pode estar disposto a usar uma API confusa, inconsistente e mal documentada se isso significar que você está obtendo acesso a um conjunto de dados exclusivo ou funcionalidade complexa. Caso contrário, boas APIs tendem a oferecer **clareza** (de propósito, design, e contexto), **flexibilidade** (capacidade de ser adaptado a diferentes casos de uso), **poder** (completude da solução oferecida), **hackeabilidade** (capacidade de aprender rapidamente por meio de iteração e experimentação) e documentação."*
>
> —Chris Messina, developer experience lead at Uber

Modelagem API WEB

- Definimos inicialmente o paradigma de API necessário para o desenvolvimento da nossa aplicação. No curso seguiremos utilizando o REST (*Representational State Tranfer*).

APIs baseadas em Request-Response (HTTP)

- **REST**
- RPC (Remote Procedure Call)
- GraphQL

APIs orientadas à eventos

- WebHooks
- WebSockets
- HTTP Streaming

REST

REST (Representational State Transfer) refere-se a um grupo de restrições de design dentro da arquitetura de software que geram sistemas distribuídos eficientes, confiáveis e escaláveis. Um sistema é denominado RESTful quando adere a todas essas restrições.

REST

REST é uma maneira simples de organizar interações em sistemas independentes e a forma mais popular atualmente de desenvolver APIs Web.

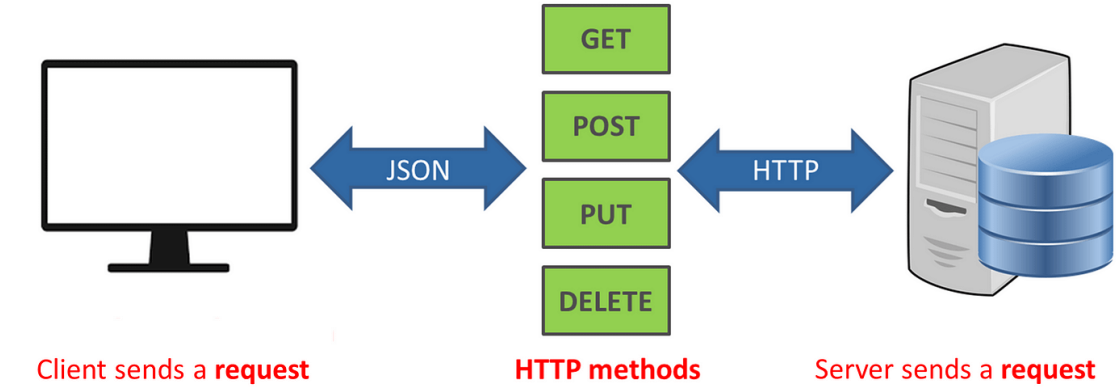
Roy Fielding o apresentou pela primeira vez em 2000 em sua famosa dissertação.

Como a implementação de REST mais conhecida para Web envolve o HTTP e o utiliza como base, foi importante primeiro entendermos como o HTTP de fato funciona antes de apresentá-lo.

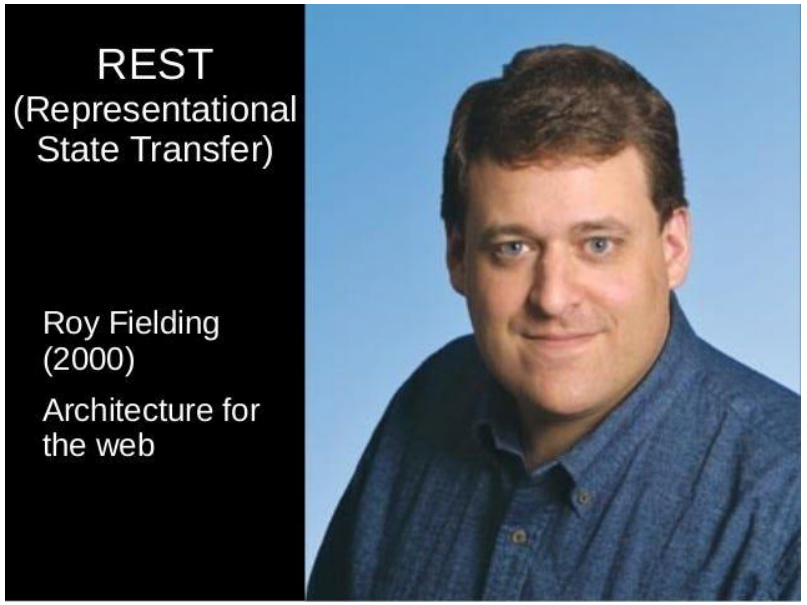
Roy Fielding

REST (*Representational State Transfer*)

API REST, também chamada de API RESTful, é uma interface de programação de aplicações (API ou API web) que está em conformidade com as restrições do estilo de arquitetura REST.



REST significa "*Representational State Transfer*", que em português significa transferência de estado representacional. Essa arquitetura foi criada pelo cientista da computação **Roy Fielding.**

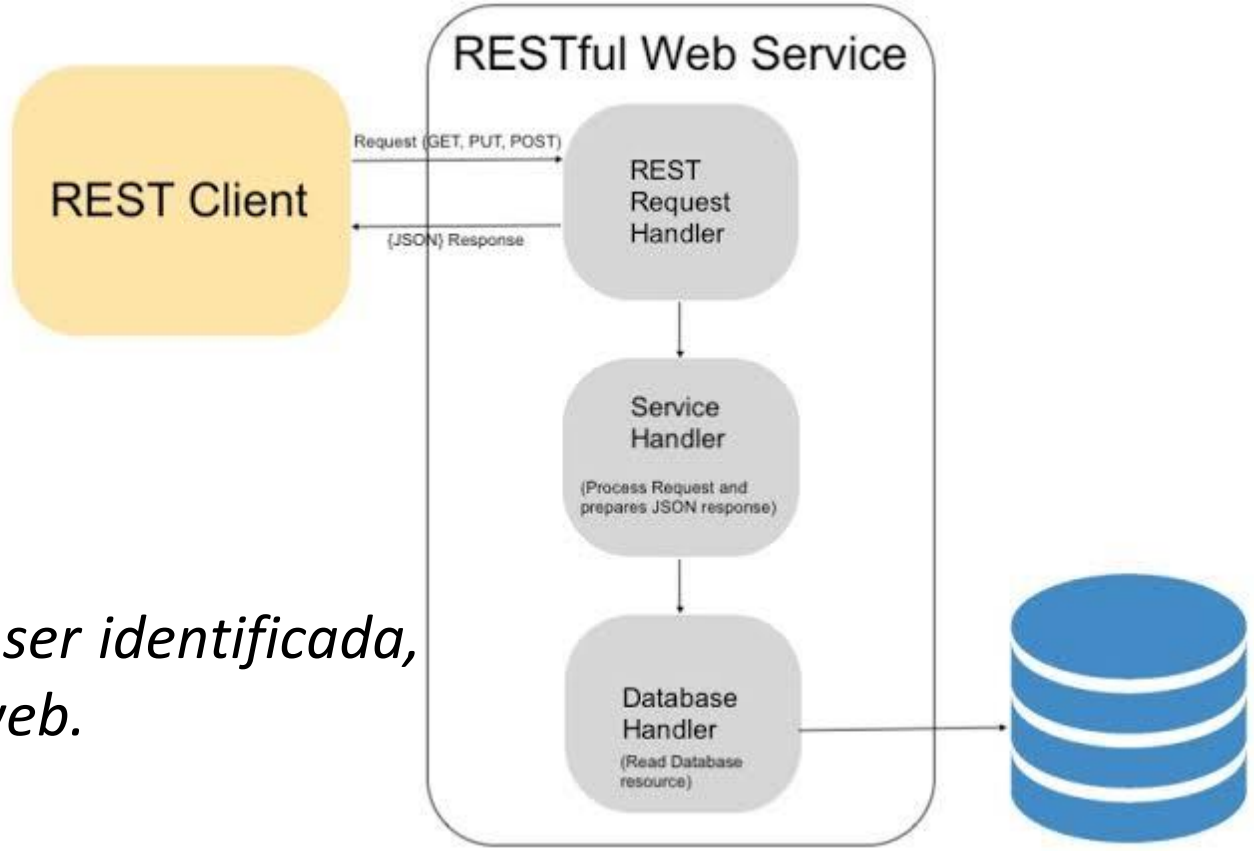


REST

RESTful Applications:

Aplicações que utilizam o padrão REST para comunicação. Em uma aplicação RESTful a comunicação é realizada preferencialmente via URLs (representa um *resource*).

```
http://myserver.com/catalog/item/1729
```



*Um **recurso** é uma entidade que pode ser identificada, nomeada, endereçada ou tratada na web.*

Arquitetura Restful

O que define uma aplicação Restful?

Princípios Restfull:

- **Client–server**
- **Stateless**
- **Cacheable**
- **Uniform interface**
- **Layered system**
- **Code on demand (optional)**

REST (*Representational State Transfer*)

Pra quê serve o REST?

fornecem muita flexibilidade. Os dados não são vinculados a recursos ou métodos, então o REST pode atender a vários tipos de chamadas, retornar diferentes formatos de dados e até mesmo mudar estruturalmente com a implementação correta de hipermídia.

Para nós, o que é importante?

- Aplicação Cliente/Servidor se comunicando via HTTP;
- Servidor não deve armazenar nenhuma informação do Cliente;
- Ter uma interface uniforme (Verbos HTTP + *Paths* bem definidos) entre os componentes para que as informações sejam transferidas em um formato padronizado (vamos utilizar o JSON).

Arquitetura Restful

Cliente-servidor - separando as questões da interface do usuário das questões de armazenamento de dados, melhoramos a portabilidade da interface do usuário em várias plataformas e melhoramos a escalabilidade simplificando os componentes do servidor (Ex: front-end em React e back-end em Spring).

Sem estado - cada solicitação do cliente para o servidor deve conter todas as informações necessárias para entender a solicitação e não pode tirar proveito de qualquer contexto armazenado no servidor. O estado da sessão é, portanto, mantido inteiramente no cliente (Ex: Na memória do servidor não há nenhum dado do cliente mantido, apenas no momento de atender a uma requisição).

Arquitetura Restful

Armazenável em cache - as restrições de cache exigem que os dados em uma resposta a uma solicitação sejam implicitamente ou explicitamente rotulados como armazenáveis ou não armazenáveis em cache (do cliente). Se uma resposta puder ser armazenada em cache, o cache do cliente terá o direito de reutilizar os dados de resposta para solicitações equivalentes posteriores. (Ex:)

Interface uniforme - ao aplicar o princípio geral da engenharia de software à interface do componente (webservice), a arquitetura geral do sistema é simplificada e a visibilidade das interações é aprimorada. Para obter uma interface uniforme, várias restrições arquitetônicas são necessárias para orientar o comportamento dos componentes. REST é definido por quatro restrições de interface: **identificação de recursos**; **manipulação de recursos por meio de representações**; **mensagens autodescritivas**; e **hipermídia como motor de estado do aplicativo**.

Arquitetura Restful

Sistema em camadas - O estilo de sistema em camadas permite que uma arquitetura seja composta de camadas hierárquicas, restringindo o comportamento do componente de forma que cada componente não possa "ver" além da camada imediata com a qual está interagindo (Ex: camada de aplicação não se comunica com a camada de dados diretamente).

Código sob demanda (opcional) - O REST permite que a funcionalidade do cliente seja estendida ao baixar e executar o código na forma de miniaplicativos ou scripts. Isso simplifica os clientes, reduzindo o número de recursos necessários para serem pré-implementados (Ex:).

Arquitetura Restful

Siste que
uma indo
o co ente
não ndo.

Códi dade
do c a de
mini o o
núm

Se a arquitetura de uma aplicação não segue esses princípios, então ela não é Restful. Veja mais em:

https://restfulapi.net/

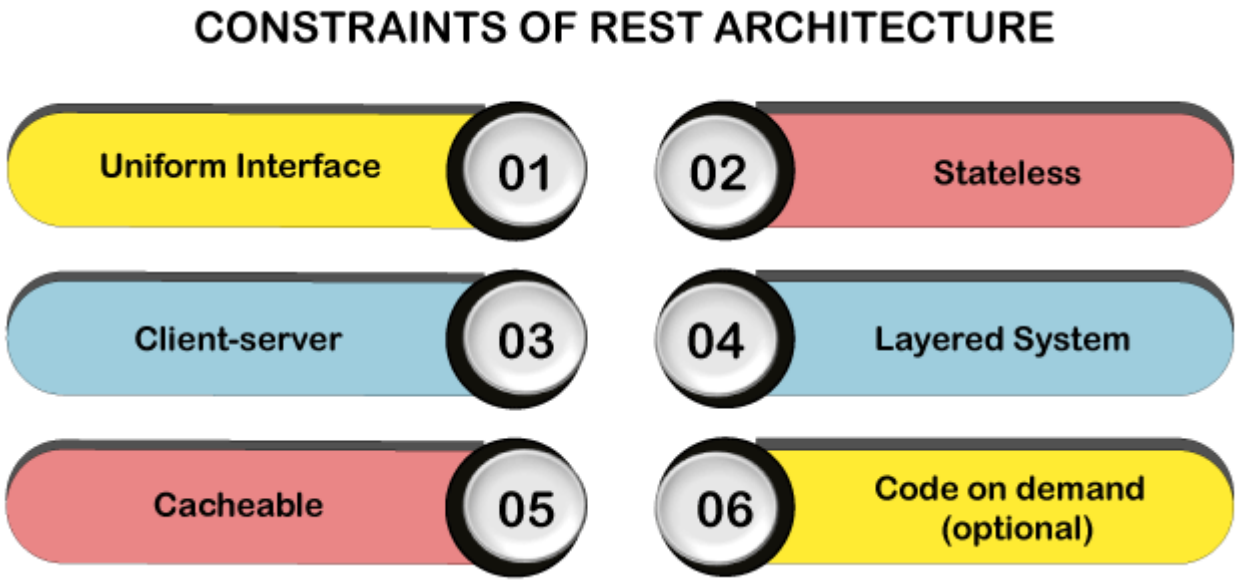
Veja uma implementação em:

https://www.youtube.com/watch?v=ghTrp1x_1As

REST (*Representational State Transfer*)

Pra quê serve o REST?

O REST é um conjunto de restrições para organização de APIs Web.



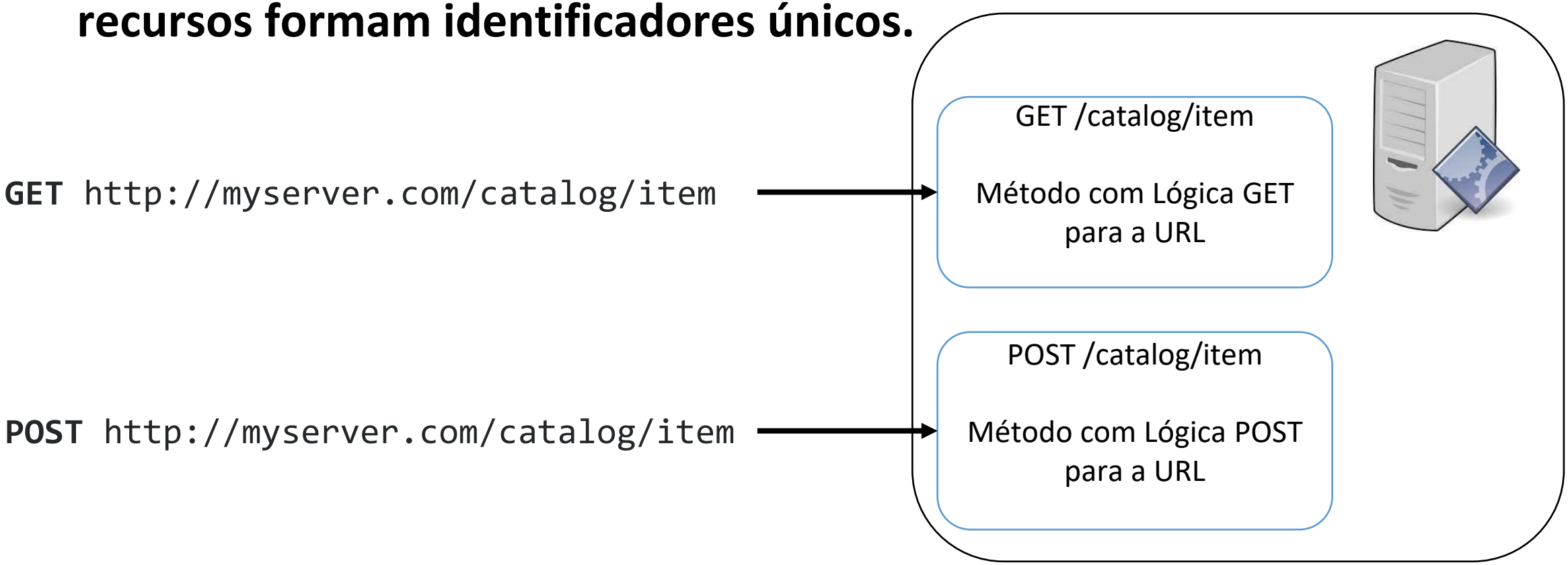
REST

Aqui estão algumas regras gerais que as APIs REST seguem:

- Métodos HTTP como GET, POST, UPDATE e DELETE informam o servidor sobre a ação a ser executada, usualmente relacionadas a operações CRUD.
- Método HTTP diferente invocado para uma mesma URL fornece funcionalidade diferente. (**`GET /user`** é diferente de **`POST /user`**)
- Os códigos de status de resposta HTTP padrão são retornados pelo servidor indicando sucesso ou falha.
- APIs REST podem retornar respostas JSON ou XML.

REST

Os verbos HTTP mais importantes para criar uma API RESTful são GET, POST, PUT e DELETE. **A combinação dos verbos com as URLs dos recursos formam identificadores únicos.**



REST

Aqui estão algumas regras gerais que as APIs REST seguem:

- Os recursos fazem parte de URLs, ex: ***`/users`***, ***`/perfil`***, ***`/registro`***.
- Para cada recurso, duas URLs são geralmente implementadas:
  - uma para a coleção, como **`/users`**,
  - e uma para um elemento específico, como **`/users/123`**.
- Substantivos são usados em vez de verbos para recursos.
  - Exemplo: em vez de **`/getUserInfo/U123`**, use **`/users/U123`**.

REST

Exemplo: Operações CRUD, Verbos HTTP e convenções REST

Operation	HTTP verb	URL: /users	URL: /users/U123
Create	POST	Create a new user	Not applicable
Read	GET	List all users	Retrieve user U123
Update	PUT or PATCH	Batch update users	Update user U123
Delete	DELETE	Delete all users	Delete user U123

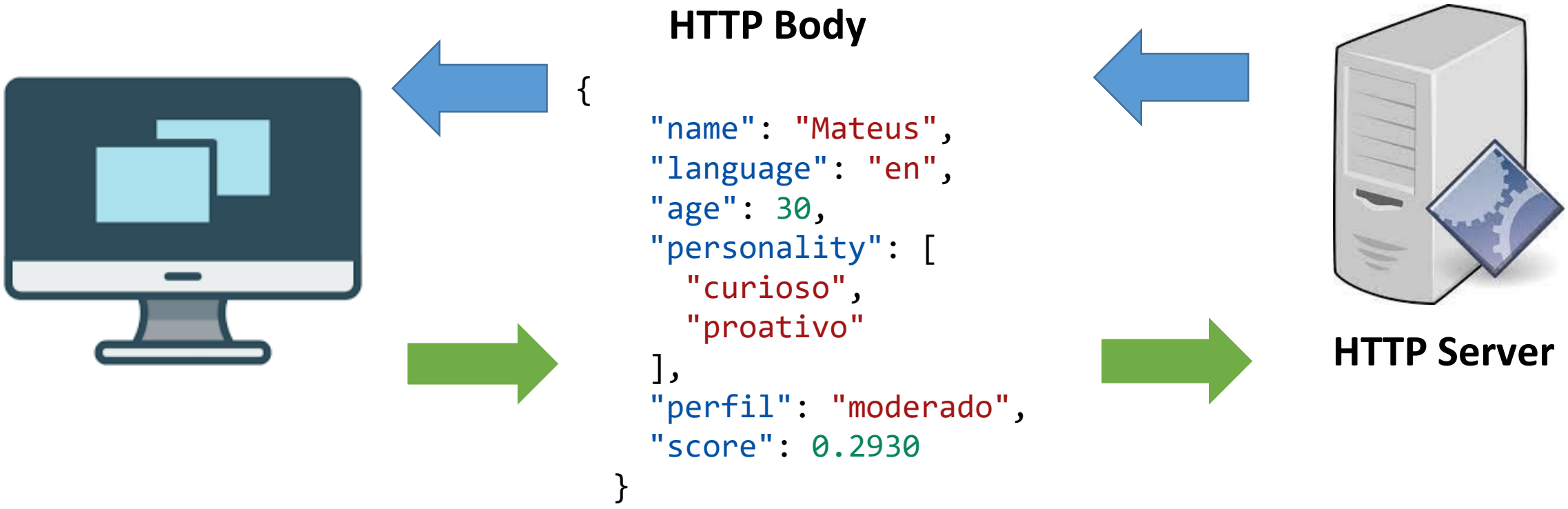
REST

Os verbos HTTP mais importantes para criar uma API RESTful são GET, POST, PUT, e DELETE.

Verbo	Objetivo	Uso	Cache
GET	Recupera um novo item de um recurso	Links	Sim
POST	Cria um novo item em um recurso	Forms	Não
PUT	Substitui um itemexistente em um recursos	Forms	Não
DELETE	Deleta um item em um recurso	Forms/links	Não

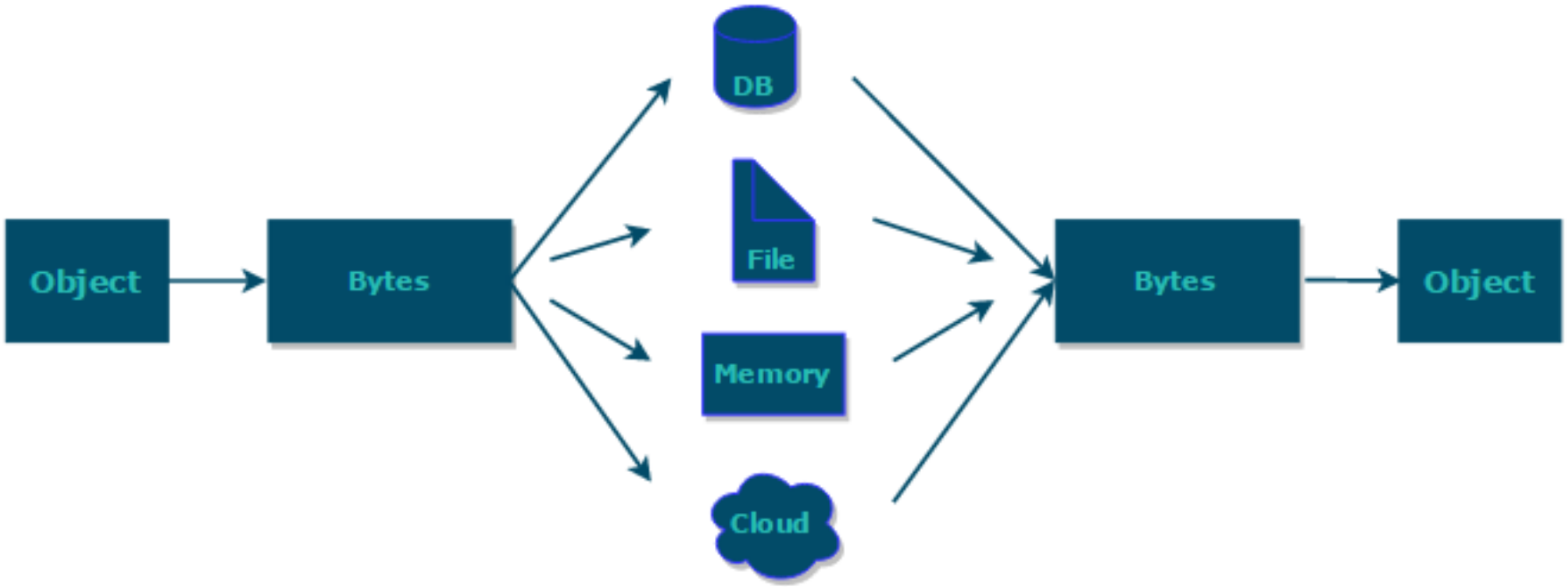
REST

REST usa várias representações para representar um recurso como texto, JSON, XML. JSON (Javascript Object Notation) é o mais popular.



Serialização de objetos

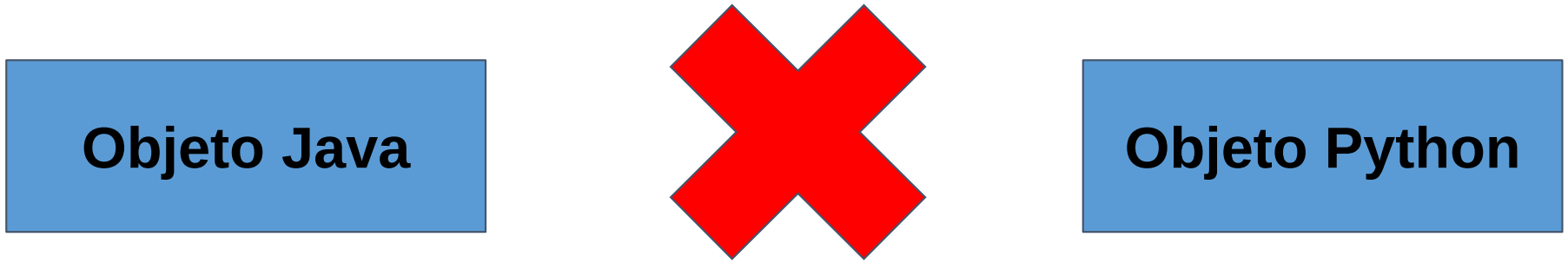
É um processo para converter uma estrutura de dados ou um objeto em um formato que possa ser armazenado ou transferido.



Serialização de objetos

A serialização possibilitará capturar o estado do objeto ou a estrutura de dados, e transformar em uma cadeia de bytes quando for necessário. Torna-se possível também, recuperar os bytes persistidos para fazer o processo inverso, e ter os dados de volta para a aplicação em execução. Isso é necessário especialmente quando precisamos transferir objetos de uma linguagem para outra através de um formato comum.

Ex:



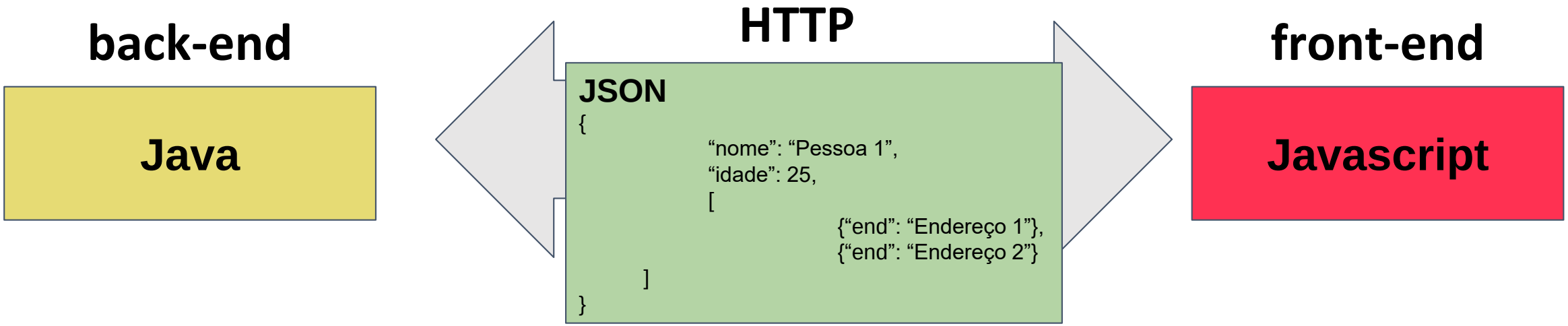
Não são compatíveis

Quais formatos de serialização?

- Stream de bytes, byte-stream-based encoding
- XML
- **JSON**
- YAML

Serialização de objetos

- No nosso curso, iremos realizar a serialização do Java para o Javascript utilizando o formato JSON.

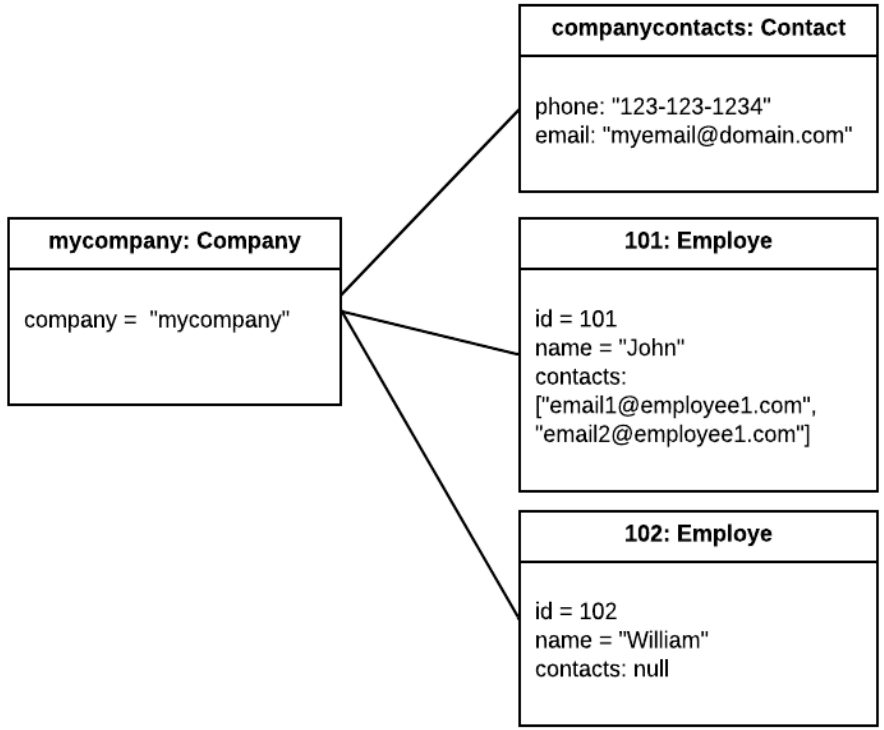


JSON

- Este acrônimo significa “JavaScript Object Notation” e, como o nome implica, o formato usa objeto JavaScript e sintaxe literal de array para converter dados estruturas que consistem em objetos e matrizes em strings.
- JSON suporta números primitivo, strings e também os valores true, false e null, bem como matrizes e objetos construídos a partir desses valores primitivos.
- JavaScript suporta serialização e desserialização JSON com as duas funções **JSON.stringify()** e **JSON.parse()**.
- No Java a serialização é realizada de forma automática.

JSON Exemplo: Representando dados de um empresa

```
JSON Object → {
                                          String Value
                "company": "mycompany",
                "companycontacts": {  ← Object Inside Object
                  "phone": "123-123-1234",
                  "email": "myemail@domain.com"
                },
                "employees": [  ← JSON Array
                  {
                    "id": 101,
                    "name": "John",
Array Inside Array →  "contacts": [
                      "email1@employee1.com",
                      "email2@employee1.com"
                    ]
                  },
                  {
                    "id": 102,  ← Number Value
                    "name": "William",
                    "contacts": null  ← Null Value
                  }
                ]
              }
```



Criando WebServices com Spring MVC

Spring Web MVC é o framework web original construído na API Servlet (é uma interface de programação de aplicativos (API) que permite a criação de aplicações web dinâmicas) e foi incluído no Spring Framework desde o início. O nome formal, "Spring Web MVC", vem do nome do seu módulo de origem (spring-webmvc), mas é mais comumente conhecido como "Spring MVC".

Ele promove a modularidade, a reutilização de código e a separação de preocupações, permitindo que os desenvolvedores construam aplicativos web robustos e escaláveis.

Alguns recursos ofertados pelo Spring
https://docs.spring.io/spring-framework/reference/web/webmvc.html

- Filters
- HTTP Message Conversion
- Annotated Controllers
- Functional Endpoints
- URI Links
- Asynchronous Requests
- CORS
- Error Responses
- Web Security
- HTTP Caching

Criando WebServices com Spring MVC

Os recursos do Spring MVC são acessados principalmente por meio de anotações (annotation)

Manipulação de Requisições e Respostas HTTP

•@RestController → Indica que a classe é um controlador REST.
•@RequestMapping → Define a URL base do controlador e o método HTTP.
•@GetMapping, @PostMapping, @PutMapping, @DeleteMapping, @PatchMapping → Mapeiam métodos HTTP específicos para handlers.

Obtenção de Dados da Requisição

•@RequestParam → Captura parâmetros da query string (?chave=valor).
•@PathVariable → Captura valores da URL (/recurso/{id}).
•@RequestBody → Obtém o corpo da requisição, útil para POST e PUT.
•@RequestHeader → Obtém valores dos cabeçalhos HTTP.

Criando WebServices com Spring MVC

Os recursos do Spring MVC são acessados principalmente por meio de anotações (annotation)

Manipulação de Respostas
•ResponseEntity<T> → Permite controle sobre status HTTP, cabeçalhos e corpo da resposta.
•@ResponseStatus → Define o status HTTP diretamente no método.

Configuração de CORS
•@CrossOrigin → Permite configurações de CORS diretamente nos controladores.

Filtros e Interceptadores
•HandlerInterceptor → Intercepta requisições antes e depois de chegarem ao controlador.
•Filters → Aplicáveis a nível de aplicação para manipular requisições e respostas.

Associando o HTTP com recursos do Spring

Requisição HTTP

Client

```
> GET /api/path/meudado?filter=valor HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.88.1
> Accept: */*
```

Server com Spring MVCWeb

Requisição HTTP

```java
@RestController
@RequestMapping(path = "api")
public class PathController {

    @GetMapping("path/{valor}")
    public ResponseEntity<Data> getMethodName(
            @PathVariable("valor") String valor,
            @RequestParam String filter,
            @RequestHeader("User-Agent") String userAgent) {
        System.out.println("O que recebeu: " + valor);
        System.out.println("User-Agent recebido: " + userAgent);
        Data data = new Data();
        data.setAttr(valor);
        return ResponseEntity.ok()
                             .header("Cache-Control", "max-age=3600")
                             .body(data);
    }
}
```

Resposta HTTP

Client

```
< HTTP/1.1 200
< Cache-Control: max-age=3600
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Wed, 26 Mar 2025 14:16:14 GMT
<
{"attr":"meudado"}
```

Resposta HTTP

Associando o HTTP com recursos do Spring

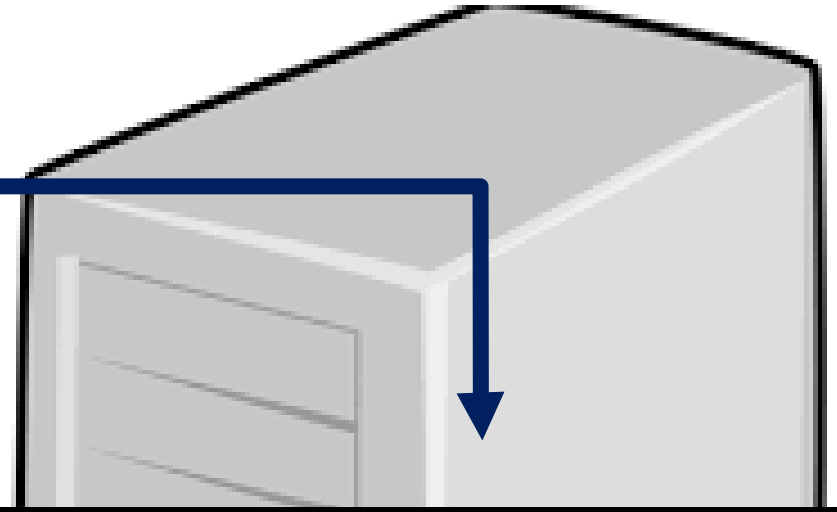
Server com Spring MVCWeb

Requisição HTTP

Client

```
> GET /api/path/meudado?filter=valor HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.88.1
> Accept: */*
```

Requisição HTTP



```java
@RestController
@RequestMapping(path = "api")
public class PathController {

    @GetMapping("path/{valor}")
    public ResponseEntity<Data> getMethodName(
            @PathVariable("valor") String valor,
            @RequestParam String filter,
            @RequestHeader("User-Agent") String userAgent) {
        System.out.println("O que recebeu: " + valor);
        System.out.println("User-Agent recebido: " + userAgent);
        Data data = new Data();
        data.setAttr(valor);
        return ResponseEntity.ok()
                             .header("Cache-Control", "max-age=3600")
                             .body(data);
    }
}
```

Resposta HTTP

Client

```
< HTTP/1.1 200
< Cache-Control: max-age=3600
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Wed, 26 Mar 2025 14:16:14 GMT
<
{"attr":"meudado"}
```

Resposta HTTP

Associando o HTTP com recursos do Spring

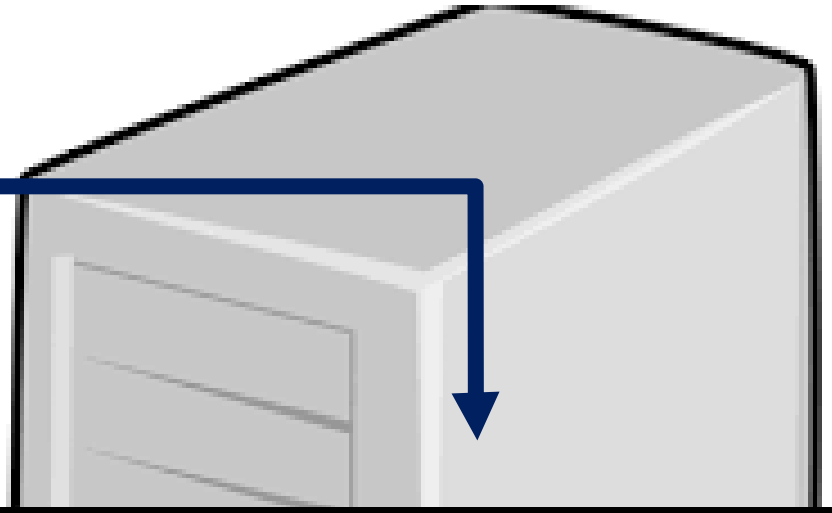
Server com Spring MVCWeb

Requisição HTTP

Client

> GET /api/path/meudado?filter=valor HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.88.1
> Accept: */*

Requisição HTTP



```java
@RestController
@RequestMapping(path = "api")
public class PathController {

    @GetMapping("path/{valor}")
    public ResponseEntity<Data> getMethodName(
            @PathVariable("valor") String valor,
            @RequestParam String filter,
            @RequestHeader("User-Agent") String userAgent) {
        System.out.println("O que recebeu: " + valor);
        System.out.println("User-Agent recebido: " + userAgent);
        Data data = new Data();
        data.setAttr(valor);
        return ResponseEntity.ok()
                .header("Cache-Control", "max-age=3600")
                .body(data);
    }
}
```

Resposta HTTP

< HTTP/1.1 200
< Cache-Control: max-age=3600
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Wed, 26 Mar 2025 14:16:14 GMT
<
{"attr":"meudado"}

Client

Resposta HTTP

Criando WebServices com Spring MVC

Vamos fazer um exemplo simples envolvendo as anotações abordadas.

1. Inicialmente crie um projeto Spring, como orientado na aula anterior.
2. Vamos criar uma classe chamada **`User`**, com login e senha, para realizar os testes necessários.
3. Vamos criar a classe **`UserController`** para servir de nosso WebService REST.

Criando WebServices com Spring MVC

Organização do projeto

```
src
└── main
    └── java
        └── com
            └── example
                ├── controller   // Camada de Controller (Web)
                ├── service      // Camada de Serviço (Negócio)
                ├── repository   // Camada de Repositório (Acesso a Dados)
                └── model        // Camada de Model (Entidades)
```

Para os testes vamos utilizar essa classe User

```
public class User {
    private String login;
    private String password;

    public String getLogin(){
        return login;
    }
    public void setLogin(String login){
        this.login = login;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

Exemplos básicos com o Spring MVC

```java
@RestController
public class UserController {
    @GetMapping("/exemplo")
    public String endPoint1(){
        return "Exemplo API Rest";
    }
    @GetMapping("/exemplo/{parametro}")
    public String endPoint2(@PathVariable("parametro") Long valor){
        return "Parâmetros recebido: " + valor;
    }
    @PostMapping("/exemplo/inserir")
    @ResponseStatus(HttpStatus.CREATED)
    public User endPoint3(@RequestBody User novoObjeto){
        return novoObjeto;
    }
    /* Continua */
```

```
    @PutMapping("/exemplo/atualizar")
    @ResponseStatus(HttpStatus.CREATED)
    public User endPoint4(@RequestBody User objetoParaAtualizar){
        return objetoParaAtualizar;
    }

    @DeleteMapping("/exemplo/remove/{id}")
    public String endPoint5(@PathVariable("id") Long id){
        if(id == 1)
            return "Remoção de informação com id " + id + "realizada";
        else{
            return "Objeto para id " + id + "não encontrado";
        }
    }
}
```

Atenção:

- **@ResponseStatus** serve para definir o status HTTP.
- Por padrão será 200, que corresponde a **@ResponseStatus(HttpStatus.OK)**

Criando WebServices com Spring MVC

Acessando nossa API (Interface Uniforme)

No navegador, testes as URLs com verbo GET.

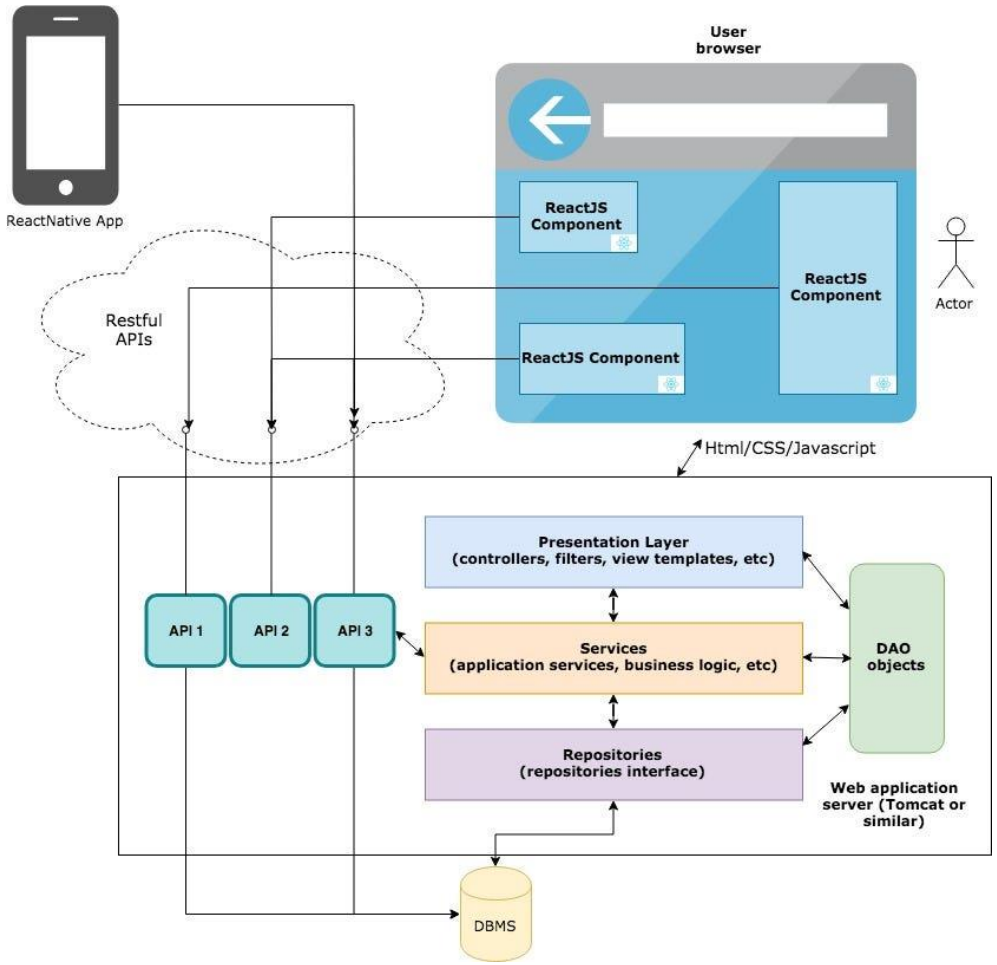
Verbo	A URL mapeia para...	... Este Endpoint
GET	localhost:8080/exemplo	@GetMapping("/exemplo")
GET	localhost:8080/exemplo/99	@GetMapping("/exemplo/{parametro}")
POST	localhost:8080/exemplo/inserir	*@PostMapping*("/exemplo/inserir")
PUT	localhost:8080/exemplo/atualizar	*@PutMapping*("/exemplo/atualizar")
DELETE	localhost:8080/exemplo/remove/1	@DeleteMapping("/exemplo/remove/{id}")

Para o POST, PUT (onde usamos **@RequestBody**) e DELETE**,** precisamos de ferramentas especificas.

Criando WebServices com Spring MVC

Algumas ferramentas para teste de API

- A arquitetura da figura ilustra como a aplicação desenvolvida será organizada.
- Entretanto, ainda não temos uma interface gráfica para “conectar” a uma API Web que criarmos.
- Podemos utiliza ferramentas como Clientes enquanto não temos as interfaces web disponíveis.



Fonte: https://medium.com/swlh/moving-from-java-spring-framework-to-a-reactjs-apis-architecture-dd85787cc616

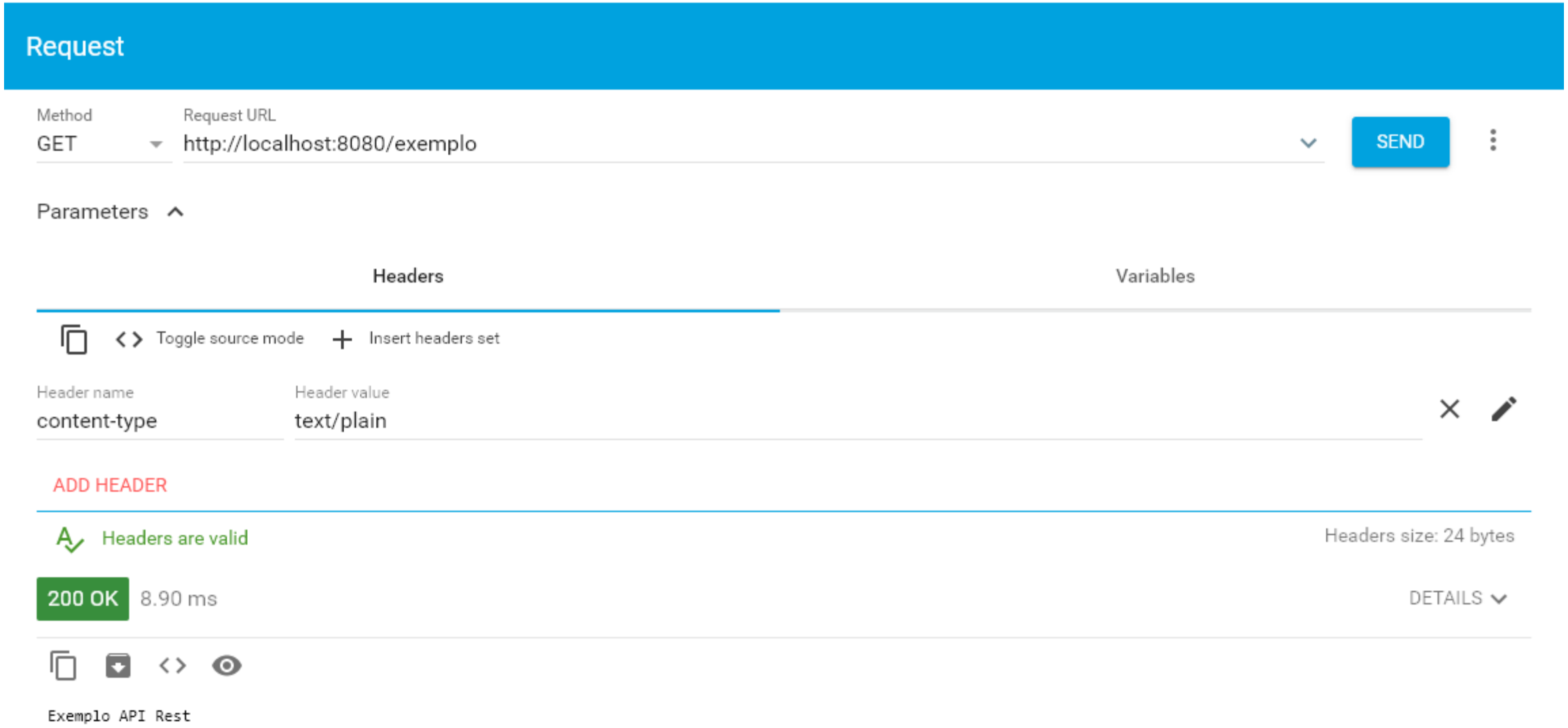
Criando WebServices com Spring MVC

Algumas ferramentas para teste de API

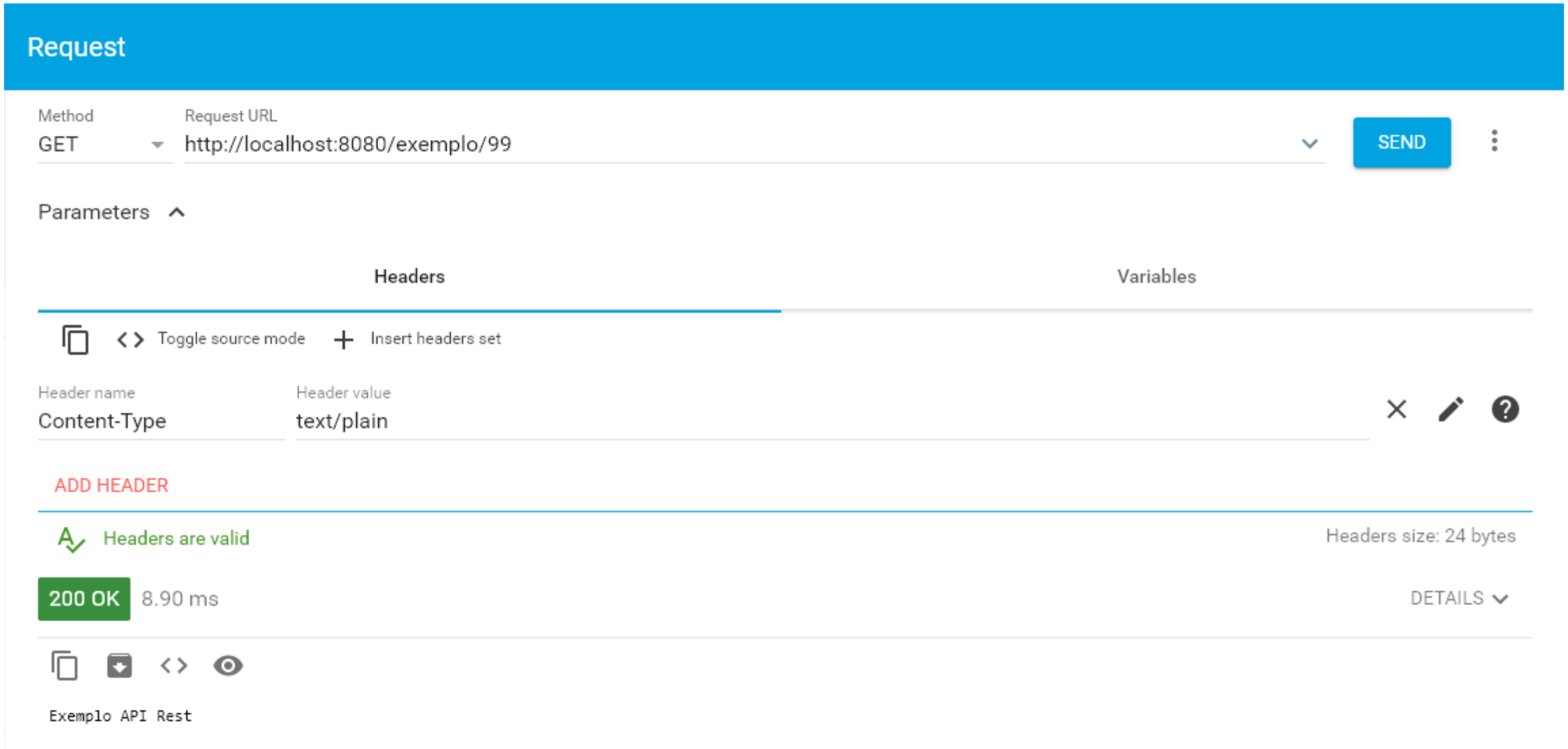
No Navegador, testes as URLs com verbo GET.

- **Postman**
- **Thunder Client (VSCode)**
- SoapUI
- Assertible
- **Advanced REST cliente (Chrome Plugin)**
- RESTer (Firefox Plugin)

Configuração GET



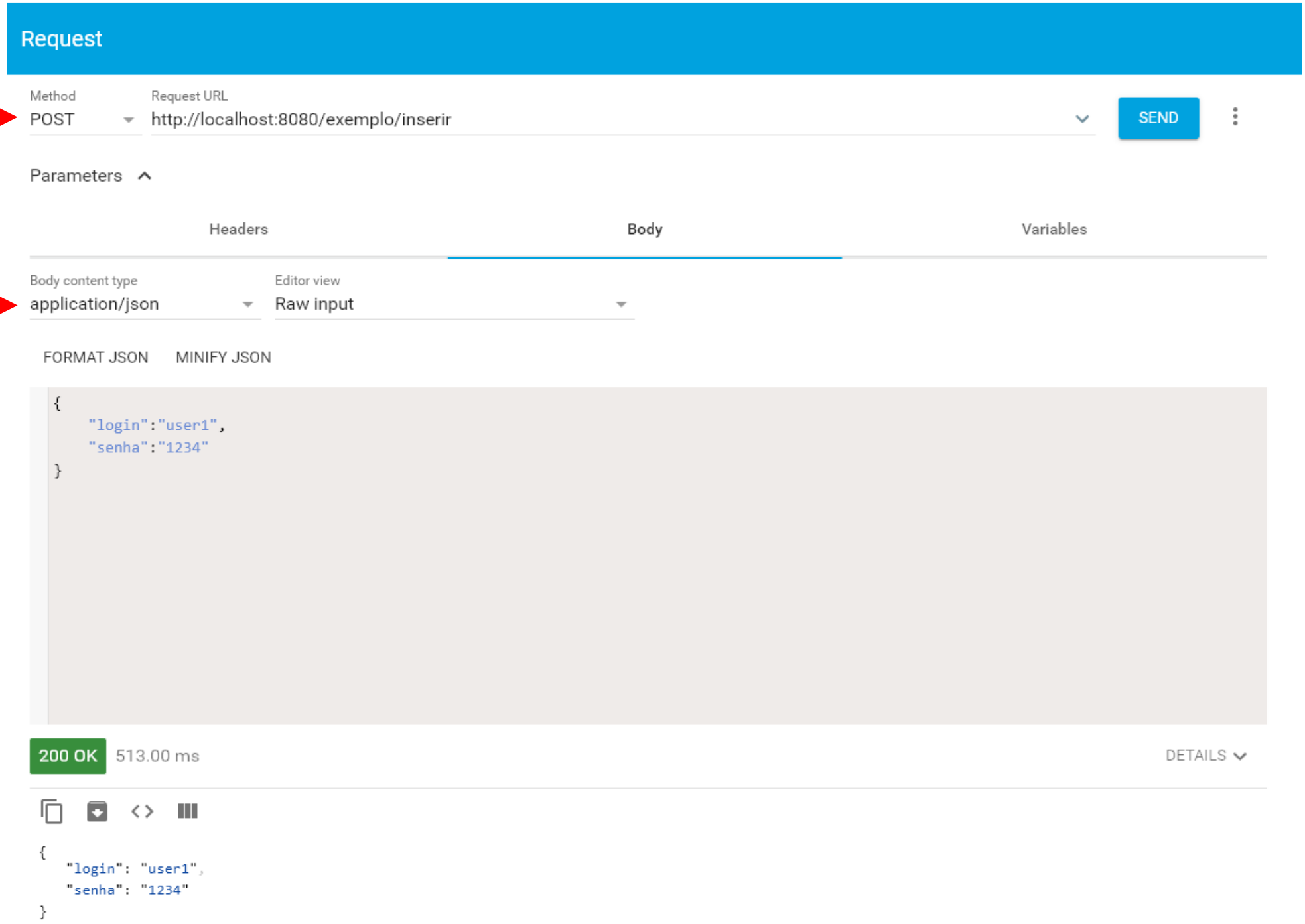
Configuração GET (Com parâmetros)



Configuração POST

Atenção com a mudança do método HTTP.

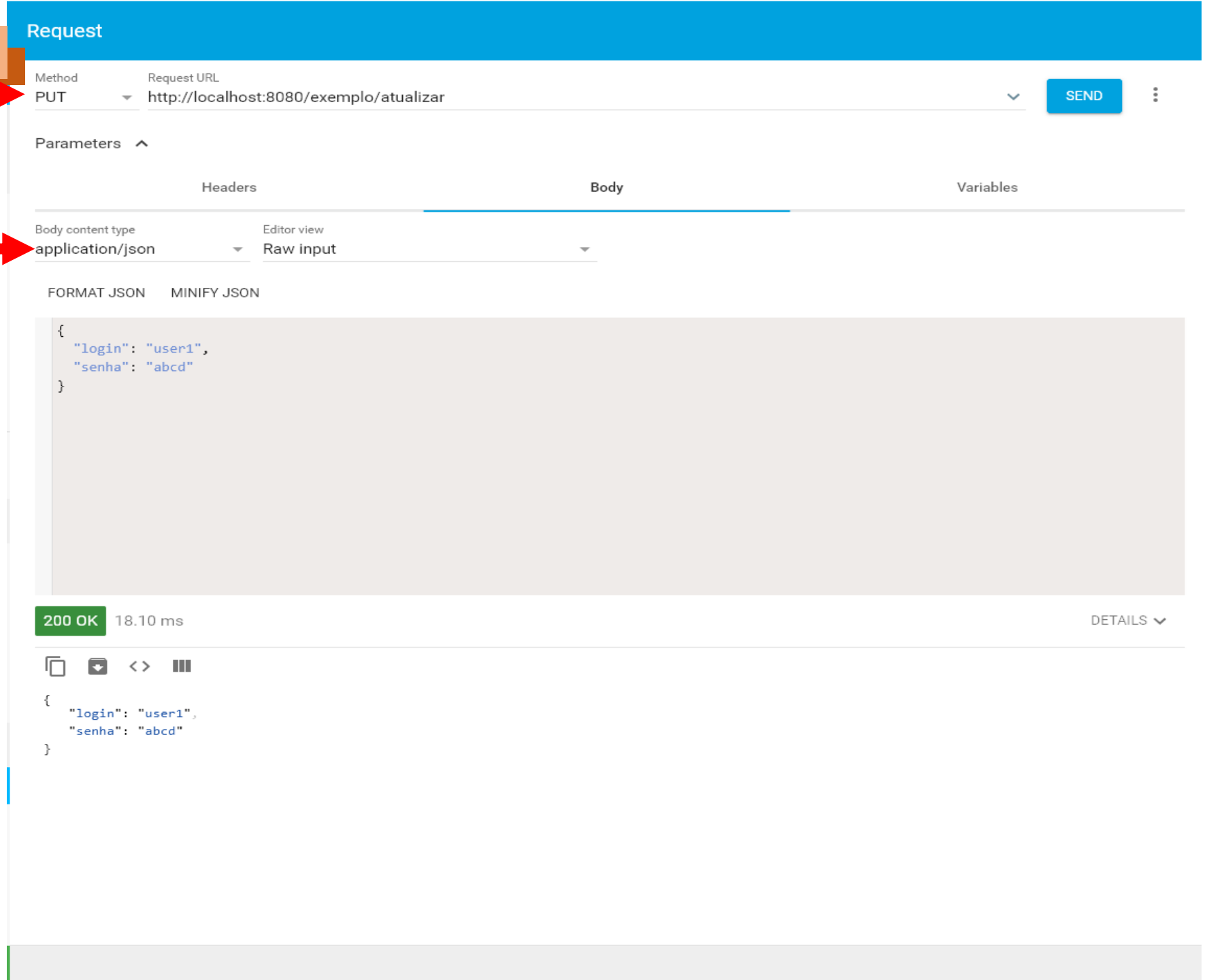
Atenção com o formato do dado a ser transmitido. No caso estamos usando JSON.



Configuração PUT

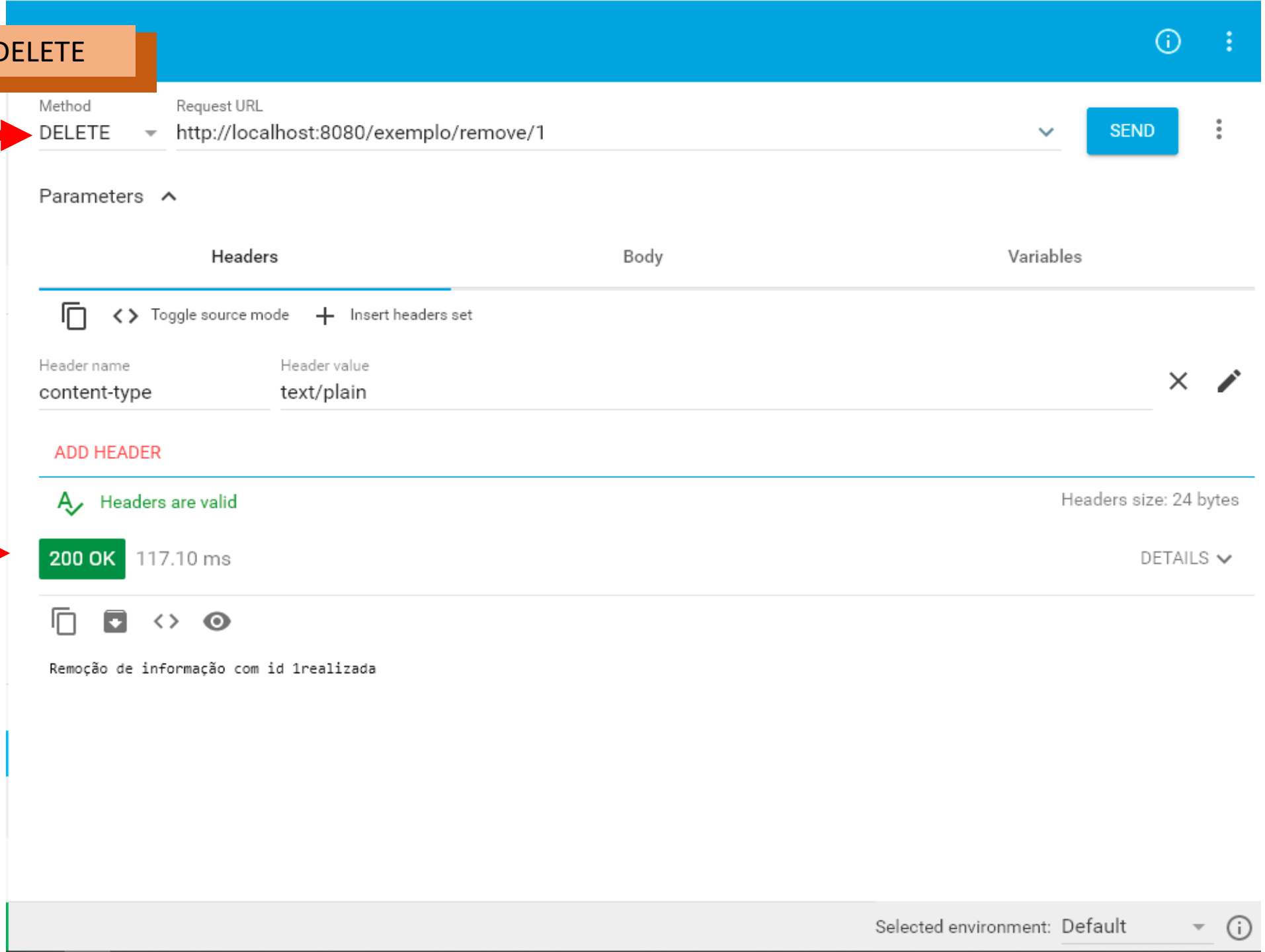
Atenção com a mudança do método HTTP.

Atenção com o formato do dado a ser transmitido. No caso estamos usando JSON.



Configuração DELETE

Atenção com a mudança do método HTTP.



Se tudo der certo, uma resposta 200 será retornada.

Criando WebServices com Spring MVC

Nos casos que enviamos um dado JSON, perceba como a estrutura do mesmo deve corresponder com a da classe no Java.

JSON

```
{
    "login": "user1",
    "senha": "1234",
}
```

Correspondentes

JAVA

```
class User{
    private String login;
    private String senha;
    public String getLogin() {
        return login;
    }
    public void setLogin(String login) {
        this.login = login;
    }
    public String getSenha() {
        return senha;
    }
    public void setSenha(String senha) {
        this.senha = senha;
    }
}
```

Fontes

SURWASE, Vijay. REST API modeling languages-a developer's perspective. **Int. J. Sci. Technol. Eng**, v. 2, n. 10, p. 634-637, 2016. url: http://www.ijste.org/articles/IJSTEV2I10199.pdf

https://restfulapi.net/