

# Instituto Federal de Educação, Ciência e Tecnologia de Guarulhos

▶ Guarulhos, 2023

Desenvolvimento de APIs em REST

# Objetivos

- REST API
- HATEOAS
- Entendendo Recursos
- Métodos HTTP
- Status Code

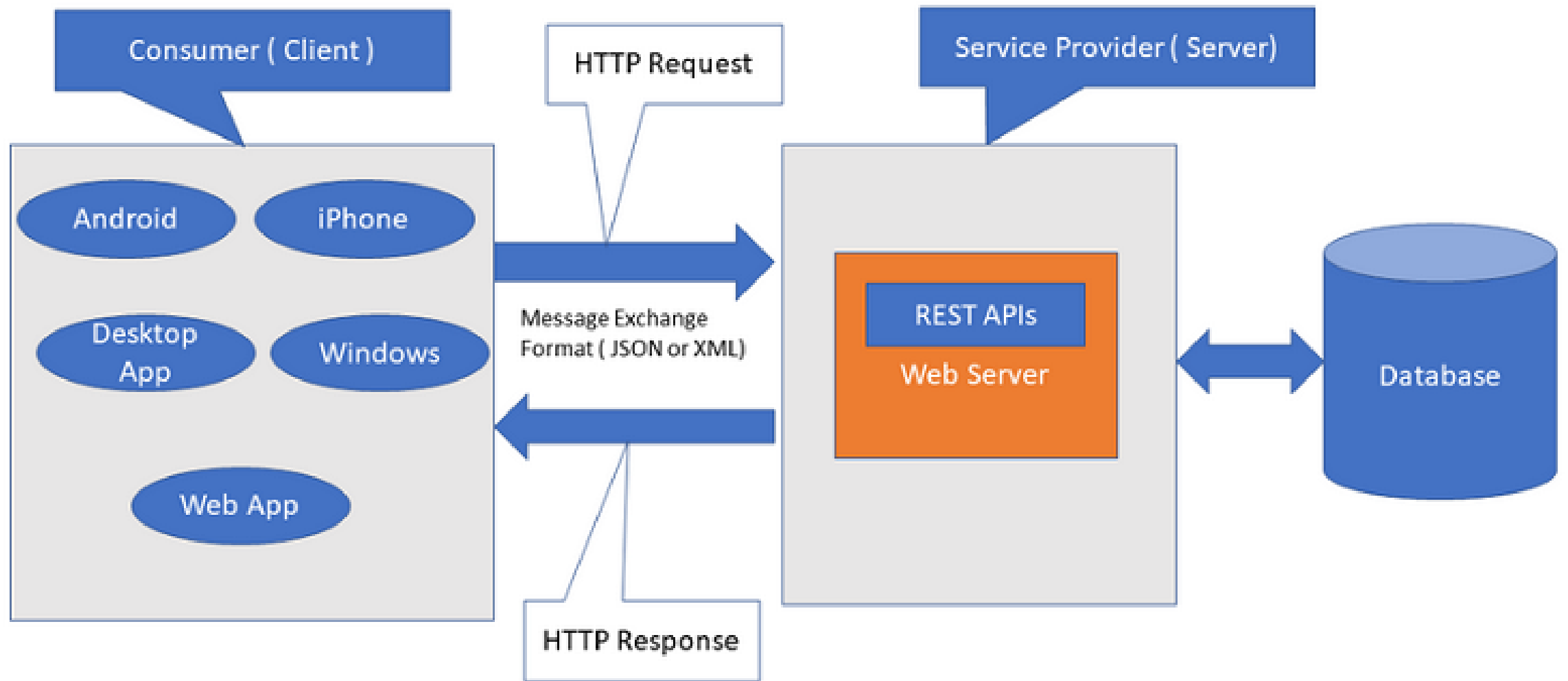


# O que é REST?\*

The REST significa transferência de estado representacional (**REpresentational State Transfer**)

- **State** significa dados
- **REpresentational** significa formatos (como XML, JSON, YAML, HTML, etc)
- **Transfer** significa transportar dados entre o consumidor e o provedor usando o protocolo HTTP

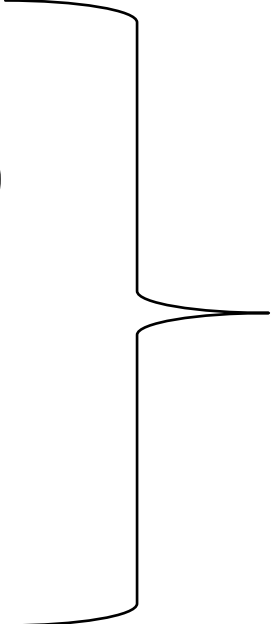
# Arquitetura REST



## ➤ O que é REST?

Roy Fielding cunhou o termo REST em sua dissertação de doutorado e propôs as seguintes seis restrições ou princípios como base:

- Client-server
- Stateless (sem estado)
- Sistema em camadas
- Cache
- Interface uniforme
- Código sob demanda



Aplicações que seguem essas restrições (constraints) são consideradas RESTful

# HATEOAS - Hypermedia as the engine of Application State

HTTP/1.1 200 OK

```
{
  "account": {
    "account_number": 12345,
    "balance": {
      "currency": "usd",
      "value": 100.00
    },
    "links": {
      "deposits": "/accounts/12345/deposits",
      "withdrawals": "/accounts/12345/withdrawals",
      "transfers": "/accounts/12345/transfers",
      "close-requests": "/accounts/12345/close-requests"
    }
  }
}
```

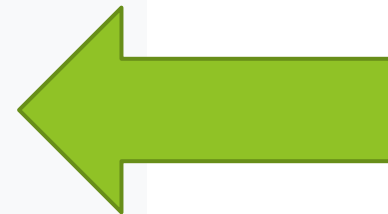
Hypermedia as the Engine of Application State (HATEOAS) é uma restrição da arquitetura de aplicações REST que a distingue de outras arquiteturas.



# HATEOAS - Hypermedia as the engine of Application State

HTTP/1.1 200 OK

```
{
  "account": {
    "account_number": 12345,
    "balance": {
      "currency": "usd",
      "value": -25.00
    },
    "links": {
      "deposits": "/accounts/12345/deposits"
    }
  }
}
```



# O Que é HATEOAS?

<https://www.treinaweb.com.br/blog/o-que-e-hateoas>

<https://www.youtube.com/watch?v=M5NWpt5d59E>





*“The key abstraction of information in REST is a resource”*

—Roy Fielding



# Entendendo recursos

***Table 1-1. URI and Resource Description***

URI	Resource description
<code>http://blog.example.com/posts</code>	Represents a collection of blog post resources.
<code>http://blog.example.com/posts/1</code>	Represents a blog post resource with identifier “1”; such resources are called singleton resources.
<code>http://blog.example.com/posts/1/comments</code>	Represents a collection of comments associated with the blog entry identified by “1”; collections such as these that reside under a resource are referred to as subcollections.
<code>http://blog.example.com/posts/1/comments/245</code>	Represents the comment resource identified by “245.”

# Representação

Os componentes REST interagem com um **recurso** transferindo suas **representações** para um lado e para o outro. Eles nunca interagem diretamente com o recurso.

# Representação

O mesmo recurso pode ter várias representações. Essas representações podem variar de formatos HTML, XML e JSON baseados em texto a formatos binários, como PDFs, JPEGs e MP4s.

# Representação

É possível para o cliente solicitar uma determinada representação, e este processo é denominado como **negociação de conteúdo**. Aqui estão as duas estratégias possíveis de negociação de conteúdo:

1) Pósfixar a URI com a representação desejada:

<http://www.example.com/products/143.json>

<http://www.example.com/products/143.xml>

2) Usando o Accept do cabeçalho da mensagem

# Métodos HTTP

Alguns dos métodos comumente usados são GET, POST, PUT e DELETE. Antes de nos aprofundarmos nos métodos HTTP, vamos rever suas duas características importantes - **segurança e idempotência**.

# Segurança

Um método HTTP é considerado seguro se não causar nenhuma alteração no estado do servidor. Considere métodos como GET ou HEAD, que são usados para recuperar informações/recursos do servidor. Essas solicitações geralmente são implementadas como somente leitura sem causar nenhuma alteração no estado do servidor e, portanto, considerado seguro.



# Idempotência

Uma operação é considerada idempotente se ela produzir o mesmo estado de servidor se nós o aplicamos uma vez ou várias vezes. Métodos HTTP como GET, HEAD (que são também seguros), PUT e DELETE são considerados idempotentes, garantindo que os clientes possam repetir uma solicitação e esperar o mesmo efeito de fazer a solicitação uma vez. A segunda solicitação e as subsequentes deixam o estado do recurso exatamente no mesmo estado que a primeira solicitação.

# GET

O método GET é usado para recuperar a representação de um recurso.

`http://blog.example.com/posts/1`

`http://blog.example.com/posts`

Como as solicitações GET não modificam o estado do servidor, elas são considerados **seguras e idempotentes**.

**GET**      **/posts/1 HTTP/1.1**

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

Accept-Encoding: gzip, deflate

Accept-Language: en-US,en;q=0.5

Connection: keep-alive

Host: blog.example.com

---

**Content-Type: text/html; charset=UTF-8**

Date: Sat, 10 Jan 2015 20:16:58 GMT

Server: Apache

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"

"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

  <head>

    <title>First Post</title>

  </head>

  <body>

    <h3>Hello World!!</h3>

  </body>

</html>

---

# HEAD

Assim como GET, o método HEAD também é seguro e idempotente e as respostas podem ser armazenado em cache no cliente

**HEAD /posts/1 HTTP/1.1**

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

Accept-Encoding: gzip, deflate

Accept-Language: en-US,en;q=0.5

Connection: keep-alive

Host: blog.example.com

---

**Connection: Keep-Alive**

Content-Type: text/html; charset=UTF-8

Date: Sat, 10 Jan 2015 20:16:58 GMT

Server: Apache

---

# DELETE

Como o método DELETE **modifica** o estado do sistema, ele não é considerado seguro. No entanto, o método DELETE é **considerado idempotente**; solicitações DELETE subsequentes ainda deixariam o recurso e o sistema no mesmo estado.

# PUT

PUT não é uma operação segura, pois altera o estado do sistema. No entanto, é considerado **idempotente**, pois colocar/atualizar o mesmo recurso uma vez ou mais de uma vez produziria o mesmo resultado.

# POST

O método POST é usado para criar recursos.

**POST /posts HTTP/1.1**

Accept: \*/\*

Content-Type: application/json

Content-Length: 63

Host: blog.example.com

BODY

{"title": "Second Post","body": "Another Blog Post."}

---

**Content-Type: application/json**

Location: posts/12345

Server: Apache

---

# POST

O método POST não é considerado seguro, pois altera o estado do sistema. Também, múltiplas Invocações POST resultam na geração de vários recursos, tornando-o não idempotente.



# PATCH

É usado para executar atualizações parciais em recursos

```
PATCH /posts/1 HTTP/1.1
```

```
Accept: */*
```

```
Content-Type: application/json
```

```
Content-Length: 59
```

```
Host: blog.example.com
```

```
BODY
```

```
{"replace": "title", "value": "New Awesome title"}
```

# Resumindo

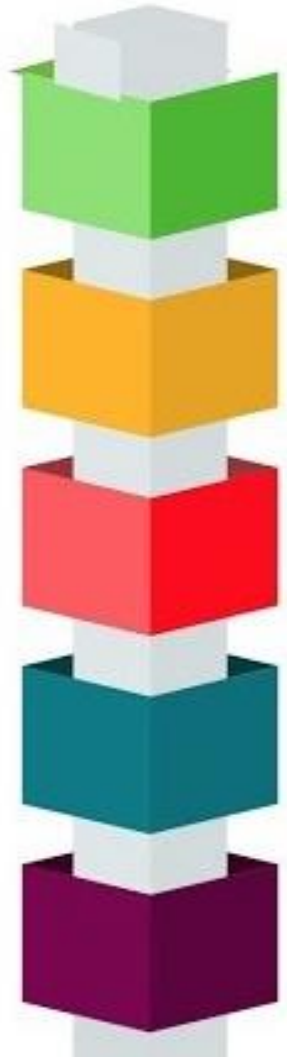
Create -> POST

Update -> PUT

Read -> GET

Delete -> DELETE

# HTTP Status Code



1XX  
INFORMATIONAL

2XX  
SUCCESS

3XX  
REDIRECTION

4XX  
CLIENT ERROR

5XX  
SERVER ERROR

# HTTP STATUS CODES

## 2xx Success

**200** Success / OK

## 3xx Redirection

**301** Permanent Redirect

**302** Temporary Redirect

**304** Not Modified

## 4xx Client Error

**401** Unauthorized Error

**403** Forbidden

**404** Not Found

**405** Method Not Allowed

## 5xx Server Error

**501** Not Implemented

**502** Bad Gateway

**503** Service Unavailable

**504** Gateway Timeout

# Referências

- ▶ Livro: Web Services Restful - Aprenda a criar web services RESTful em Java na nuvem do Google. Capítulos 4 e 5. Autor: Ricardo R. Lecheta.
- ▶ <https://start.spring.io/>
- ▶ <https://spring.io/quickstart>
- ▶ <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
- ▶ <https://docs.spring.io/spring-framework/docs/2.5.x/reference/aop.html>
- ▶ <https://docs.spring.io/spring-framework/docs/4.1.5.RELEASE/spring-framework-reference/html/overview.html>
- ▶ <https://www.geeksforgeeks.org/aspect-oriented-programming-and-aop-in-spring-framework/>
- ▶ <https://www.youtube.com/watch?v=vGuqKIRWosk>
- ▶ <https://www.youtube.com/watch?v=M5NWpt5d59E>
- ▶ <https://www.youtube.com/watch?v=eel1OVldfUw>
- ▶ <https://en.wikipedia.org/wiki/HATEOAS>
- ▶ Livro Spring REST - Building Java Microservices and Cloud Applications - Second Edition - Balaji Varanasi and Maxim Bartkov

# Referências

- <https://spring.io/quickstart>
- <https://www.javaguides.net/p/rest-api-tutorial.html>
- [https://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)
- Livro Spring Boot: up and Running - Building Cloud Native Java and Kotlin Applications - Capítulo 3
- <https://www.treinaweb.com.br/blog/entendendo-injecao-de-dependencia>
- <https://www.treinaweb.com.br/blog/o-que-e-hateoas>