BRUNNA CROCHES

PARTE 2

API | REST | SPA

Australian Pelican | Artist Image :Broinowski, Gracius Joseph | Dates:1837-1913

Guia iniciante: API

SUMMARY
REST | API | SPA

ABOUT ME

Brunna Croches

Developer Full Stack

Brunna Croches é Dev FullStack, advogada e empreendedora.

Apaixonada por tech, vem adquirido vasto conhecimento na área.

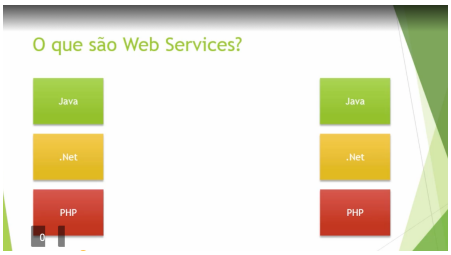
Desenvolveu projetos ricos em diversidade, buscando captar as próximas tendências e necessidades do mercado.

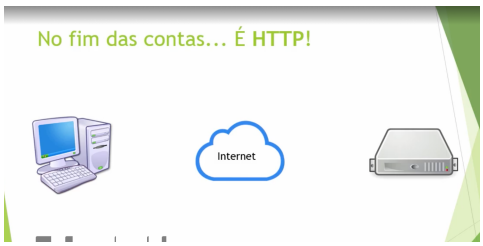
Neste e-book você aprenderá ou recapitulará de forma simplificada e otimizados conceitos de programação feito por ela.

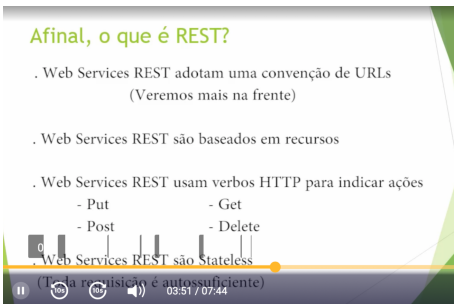
let's share

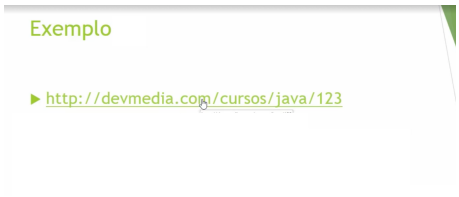
3.0 Guia REST

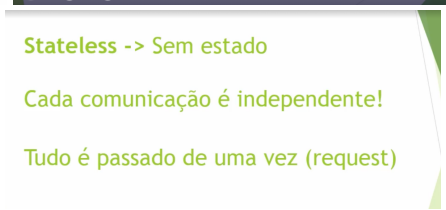
Introdução

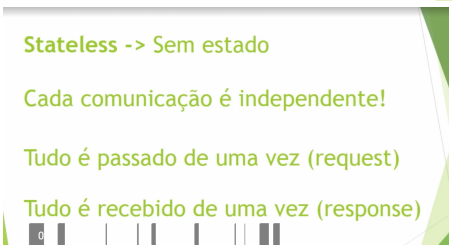




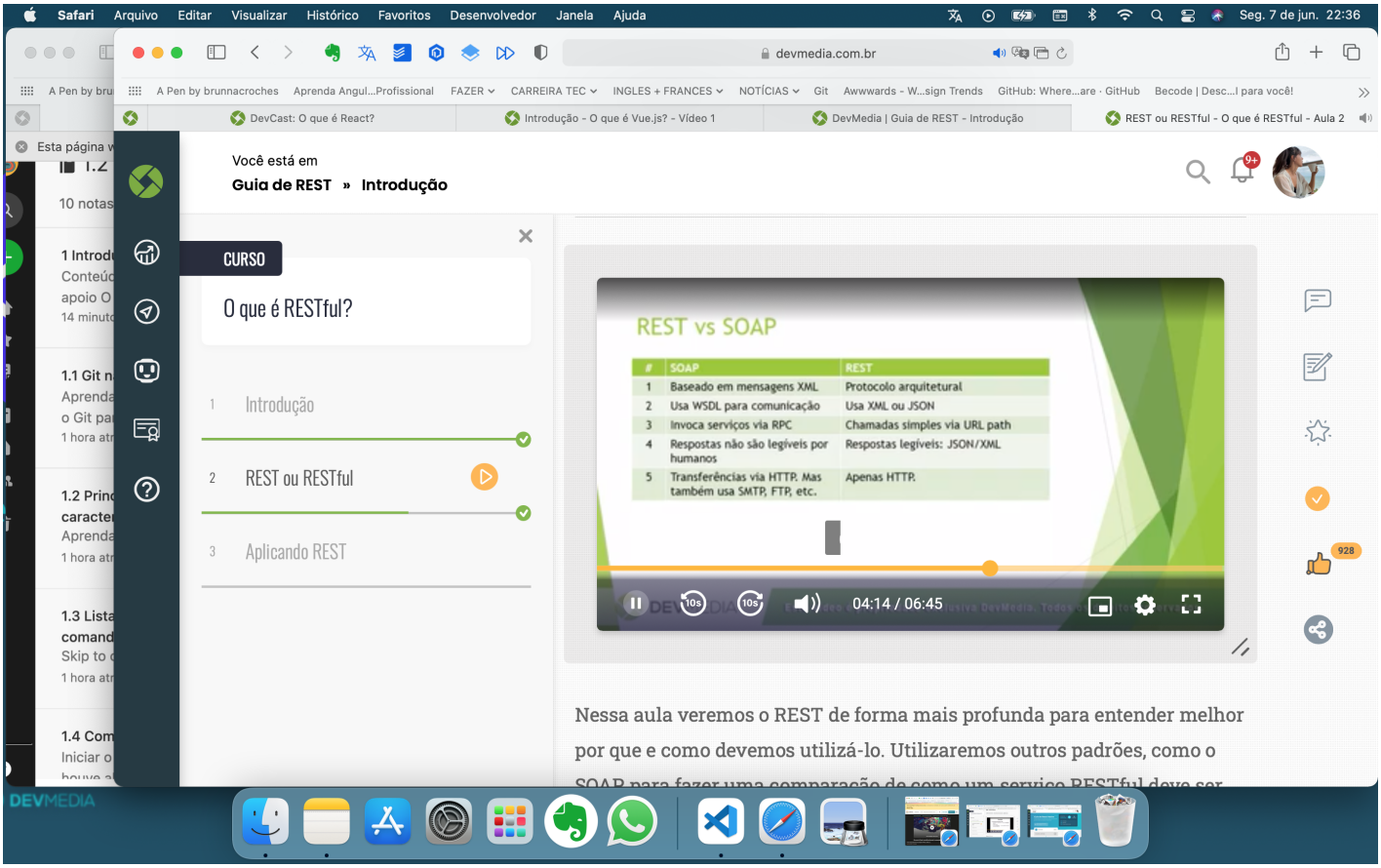


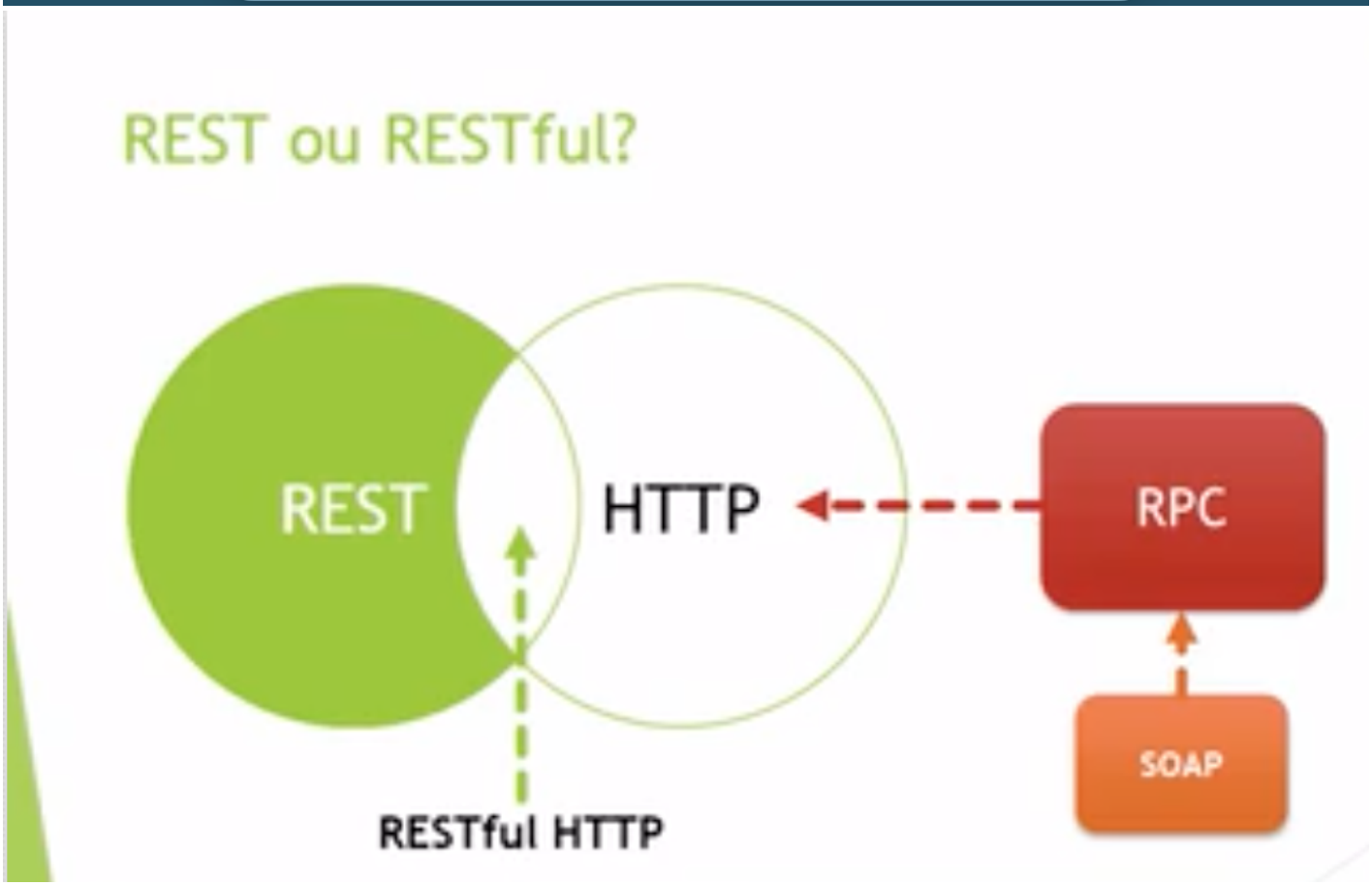






2) REST OU RESTful?

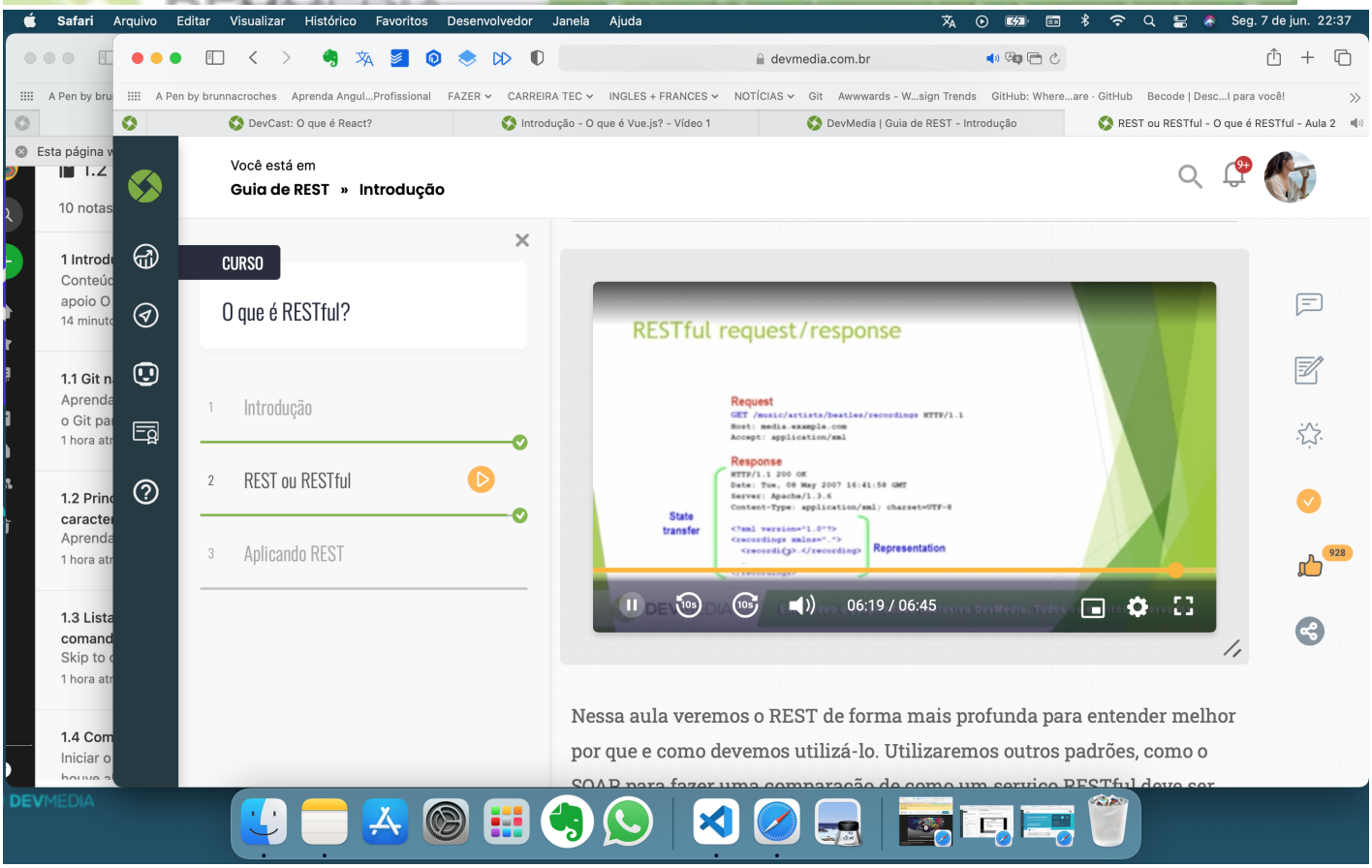

Safari Arquivo Editar Visualizar Histórico Favoritos Desenvolvedor Janela Ajuda
Seg. 7 de jun. 22:36
devmedia.com.br
DevCast: O que é React?
Introdução - O que é Vue.js? - Vídeo 1
DevMedia | Guia de REST - Introdução
REST ou RESTful - O que é RESTful - Aula 2
Você está em
Guia de REST » Introdução
CURSO
O que é RESTful?
1 Introdução
2 REST ou RESTful
3 Aplicando REST
REST vs SOAP
SOAP REST
1 Baseado em mensagens XML Protocolo arquitetural
2 Usa WSDL para comunicação Usa XML ou JSON
3 Invoca serviços via RPC Chamadas simples via URL path
4 Respostas não são legíveis por humanos Respostas legíveis: JSON/XML
5 Transferências via HTTP. Mas também usa SMTP, FTP, etc. Apenas HTTP.
04:14 / 06:45
928
Nessa aula veremos o REST de forma mais profunda para entender melhor por que e como devemos utilizá-lo. Utilizaremos outros padrões, como o
DEVMEDIA

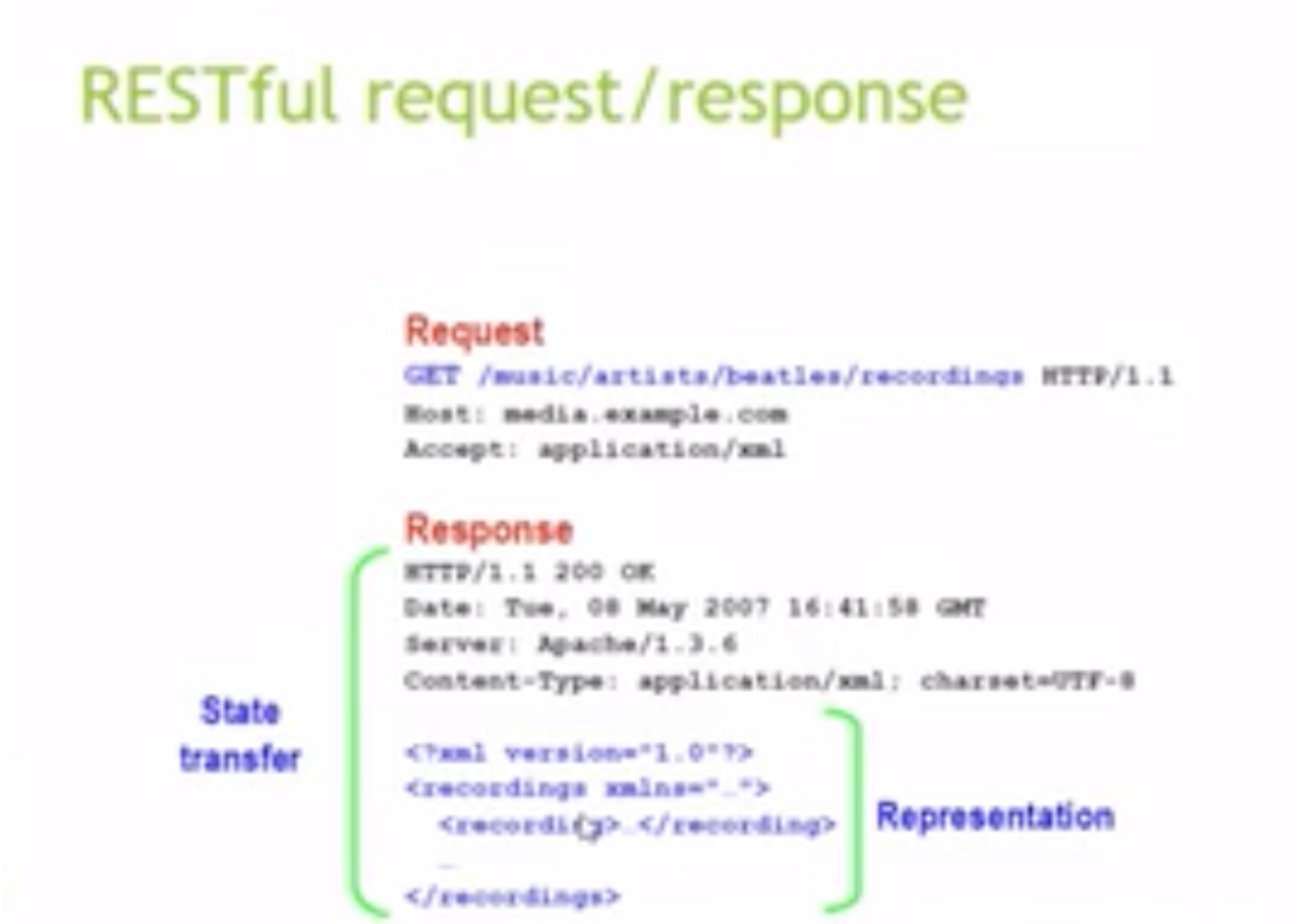

REST ou RESTful?
REST
HTTP
RPC
SOAP
RESTful HTTP

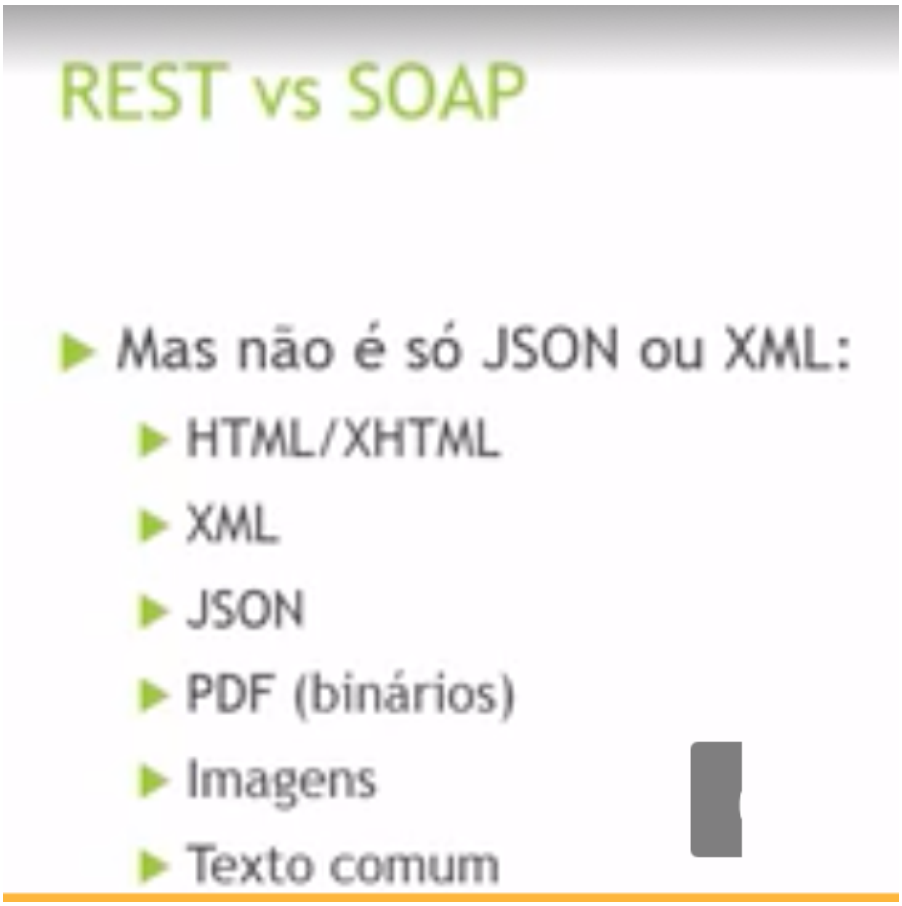
REST vs SOAP

#	SOAP	REST
1	Baseado em mensagens XML	Protocolo arquitetural
2	Usa WSDL para comunicação	Usa XML ou JSON
3	Invoca serviços via RPC	Chamadas simples via URL path
4	Respostas não são legíveis por humanos	Respostas legíveis: JSON/XML
5	Transferências via HTTP. Mas também usa SMTP, FTP, etc.	Apenas HTTP.
6	JavaScript pode chamar SOAP (difícil de implementar)	Consumo simples e fácil com JavaScript
7	Menos performático que o REST	Código mais simples, menos consumo de banda, intuitivo.


SOAP request/response
POST /Service1.asmx HTTP/1.1
Host: localhost
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length
<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
<soap12:Body>
<HelloWorld xmlns="http://tempuri.org/" />
</soap12:Body>
</soap12:Envelope>
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length
<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
<soap12:Body>
<HelloWorldResponse xmlns="http://tempuri.org/">
<HelloWorldResult>string</HelloWorldResult>
</HelloWorldResponse>
</soap12:Body>
</soap12:Envelope>


Safari Arquivo Editar Visualizar Histórico Favoritos Desenvolvedor Janela Ajuda
Seg. 7 de jun. 22:37
devmedia.com.br
DevCast: O que é React?
Introdução - O que é Vue.js? - Vídeo 1
DevMedia | Guia de REST - Introdução
REST ou RESTful - O que é RESTful - Aula 2
Você está em
Guia de REST » Introdução
CURSO
O que é RESTful?
1 Introdução
2 REST ou RESTful
3 Aplicando REST
RESTful request/response
Request
Response
State transfer
Representation
06:19 / 06:45
Nessa aula veremos o REST de forma mais profunda para entender melhor por que e como devemos utilizá-lo. Utilizaremos outros padrões, como o

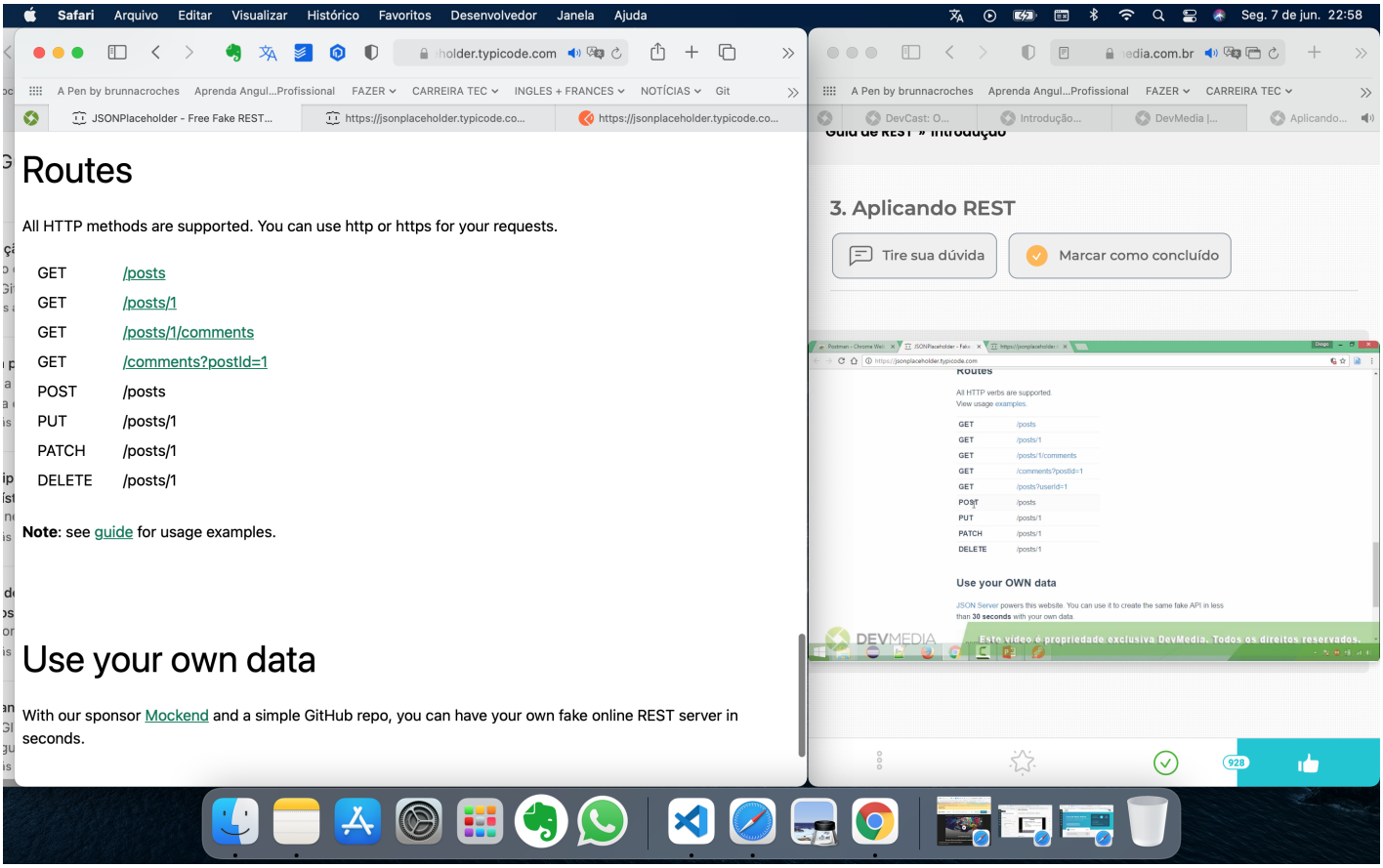


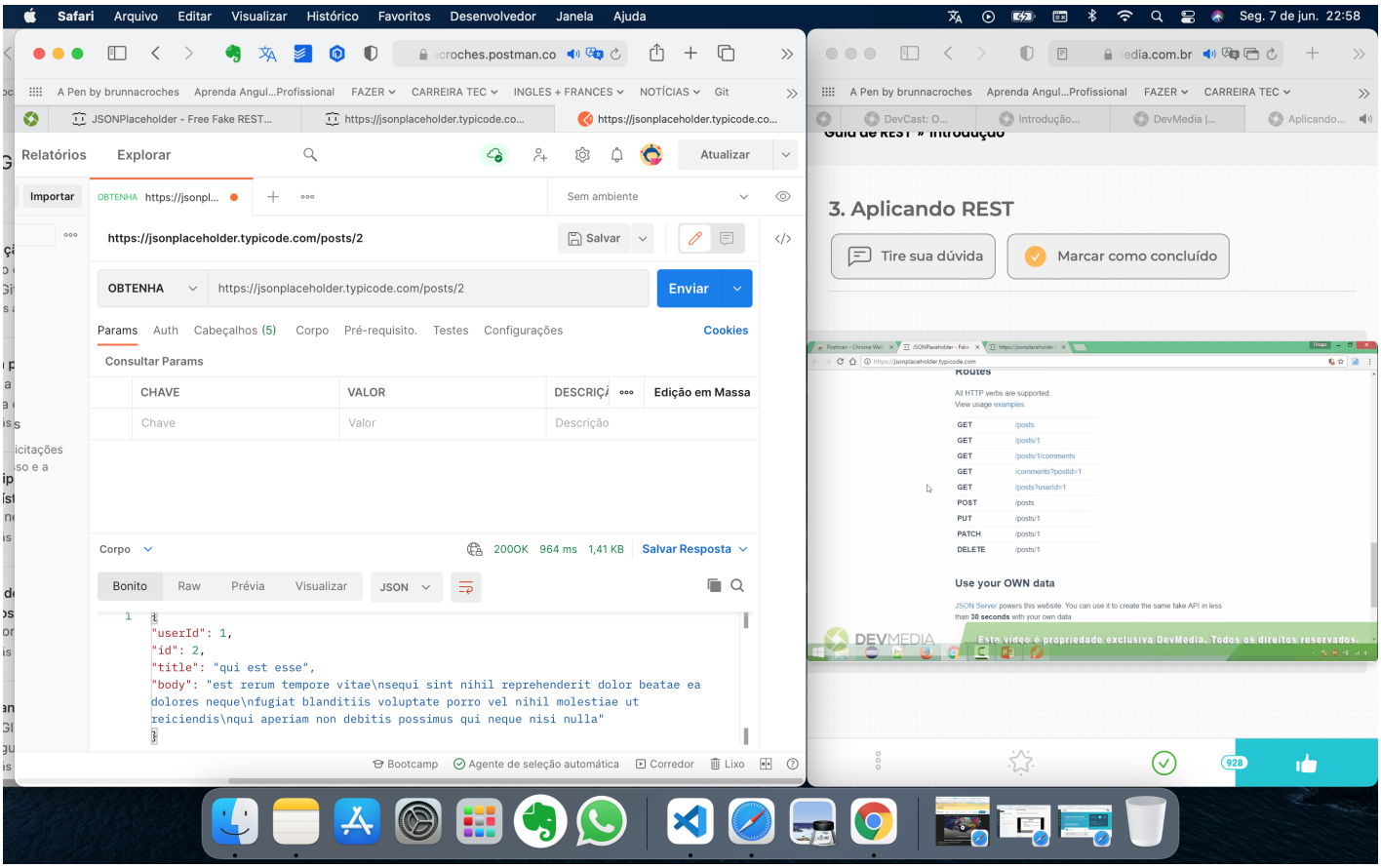


3. Aplicando REST

https://brunnacroches.postman.co/workspace/My-Workspace~efedf7a8-1a40-49f6-846e-2186f0482695/request/create?requestId=26ada06e-b678-4ca9-8be4-c11b23f21cc0

https://jsonplaceholder.typicode.com/guide/





4- Um bate papo sobre REST & RESTful

https://www.devmedia.com.br/um-bate-papo-sobre-rest-restful/39489

Fique por dentro do REST

Em algum momento na história das aplicações web, percebeu-se que o cliente final da aplicação poderia não ser apenas o navegador, mas outros dispositivos e até mesmo outras aplicações web. Foi então que passamos a pensar nas aplicações como serviços, disponibilizando para seus clientes dados e apenas dados.

REST é muitas vezes apresentado como um padrão arquitetural, mas na verdade trata-se de um conjunto de restrições que determinam como deve ser realizada a Transferência de Estado Representacional (Representational State Transfer — REST), de uma determinada entidade em um dado momento. Dentre as mais marcantes, tais restrições são o acesso a um recurso a partir de uma URL única, o envio e recebimento dos dados em documentos estruturados, sendo o formato mais comumente utilizado o json, e a utilização dos verbos HTTP para determinar como o web service deverá processar a requisição.

Outra característica do padrão REST é que a aplicação fica limitada a uma arquitetura cliente/servidor na qual um protocolo de comunicação que não mantenha o estado das transações entre uma solicitação e outra deve ser utilizada. Neste sentido, o protocolo mais utilizado é o HTTP, bem como, para representar o estado de cada entidade, utilizamos documentos em formato json.

Uma vez que é identificada a necessidade da aplicação fornecer dados para diferentes dispositivos, ou em diferentes formatos de documento como HTML, json, XML, entre outras, podemos considerar tornar essa aplicação um web service. E ao desenvolver este web service com um Framework REST, permitiremos que esse serviço seja facilmente consumido por aplicações cliente moderno, muitas vezes já planejado para lidar com este tipo de arquitetura cliente/servidor, baseada na troca de arquivos em json, como Angular, Ionic, React, etc.

3) Serviços RESTful: verbos HTTP

Quando começamos a desenvolver – ou consumir - nossos primeiros serviços RESTful, a primeira coisa que precisamos entender é o papel dos verbos HTTP dentro do contexto REST.
Este artigo pode lhe servir como referência de consulta para iniciar seus trabalhos com os verbos HTTP em serviços REST.

Saiba mais: O que é RESTful?

Fundamentos

A ideia geral é a seguinte: seu serviço vai prover uma url base e os verbos HTTP vão indicar qual ação está sendo requisitada pelo consumidor do serviço.
Por exemplo, considerando a URL www.dominio.com/rest/notas/, se enviarmos para ela uma requisição HTTP utilizando o verbo GET, provavelmente obteremos como resultado uma listagem de registros (notas, nesse caso). Por outro lado, se utilizarmos o verbo POST, provalmente estaremos tentando adicionar um novo registro, cujos dados serão enviados no corpo da requisição.
Da mesma forma, a URL www.dominio.com/rest/notas/1, por exemplo, poderia ser usada para diferentes finalidades, dependendo do verbo enviado na requisição. No caso do GET, essa URL provavelmente deveria nos retornar o registro de ID 1 (nesse caso, a nota de ID = 1). Já o verbo DELETE indicaria que desejamos remover esse registro.
Repare que a URL se mantém – o verbo indica o que estamos fazendo de fato. Por exemplo, não precisamos disponibilizar no serviço uma URL como /notas/listar ou /notas/remover/1.

Exemplo prático

Os exemplos a seguir foram feitos utilizando a extensão Postman, do Google Chrome, para comunicar com a API, criada no curso Criando serviços RESTful em .NET. Para saber mais sobre esse serviço assista também ao curso O que é RESTful. Contudo, os conceitos apresentados aqui podem ser aplicados a outras tecnologias, uma vez que REST é um padrão e independe de linguagem.

Verbo GET:

Sem passagem de ID: vai retornar todas as notas (ou as notas mais recentes, isso cabe a regra de negócio da aplicação).

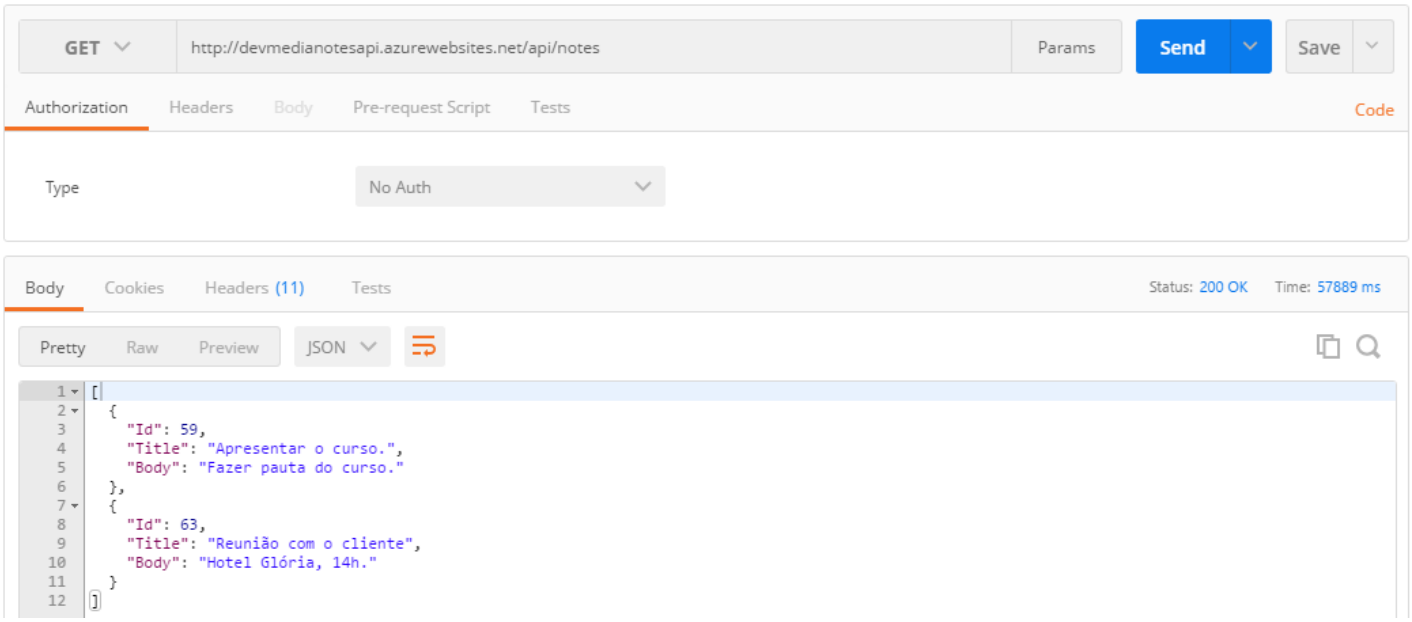


Figura 1. Requisição GET sem parâmetros

Com passagem de ID: vai retornar a nota com ID especificado.

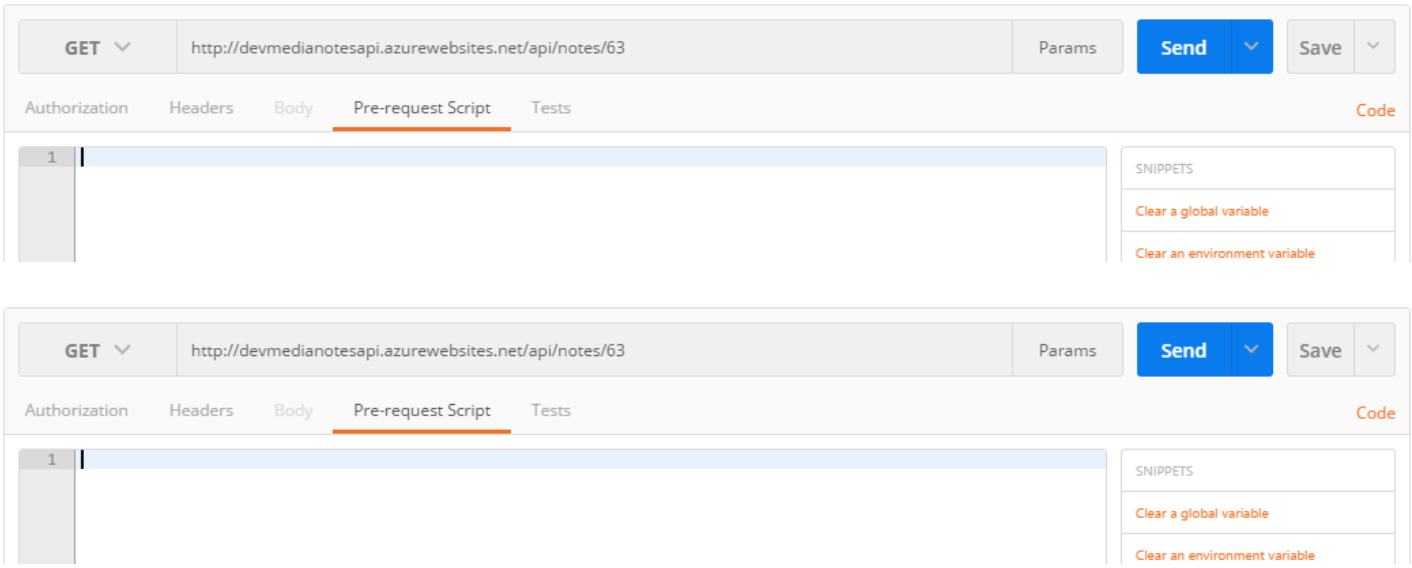


Figura 2. Requisição GET com parâmetros

Verbo POST:

Normalmente usado sem passagem de parâmetro – usado para criar uma nova nota.

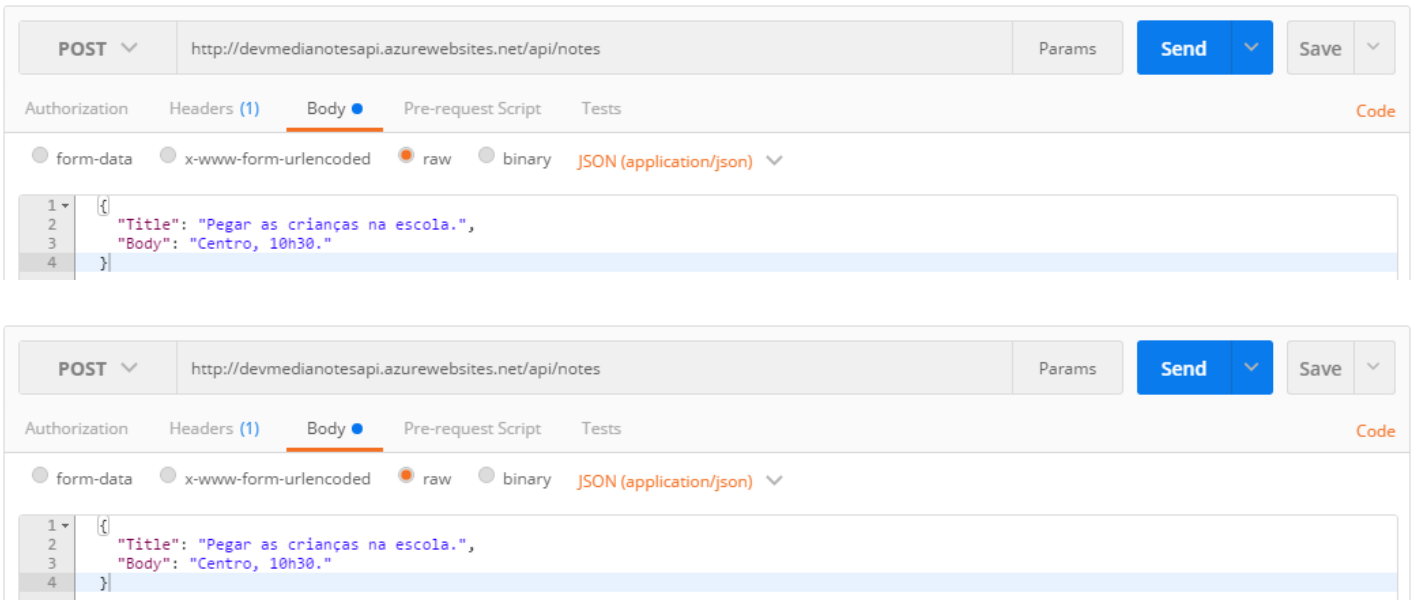


Figura 3. Requisição POST

Verbo DELETE:

Usado para remover o recurso (por exemplo uma nota): utilize com passagem de ID.

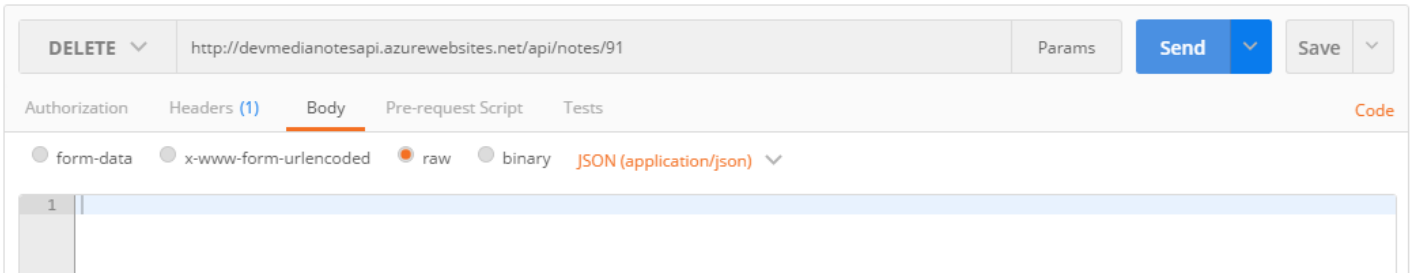


Figura 4. Requisição DELETE

Verbo PUT:

Normalmente usado com parâmetro; Use para editar o recurso – neste exemplo, uma nota.

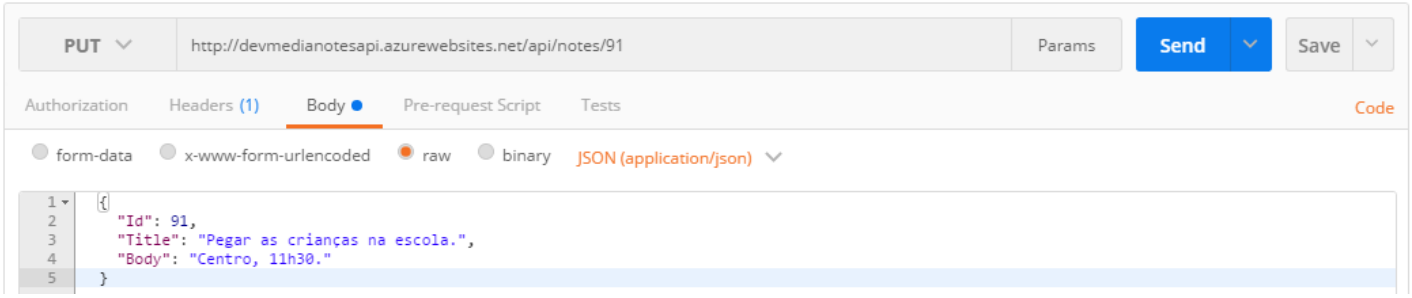


Figura 5. Requisição PUT

Nota:

A literatura indica que o verbo PUT deve passar todos os dados do recurso preenchidos, independente de quais dados você de fato editou. Por exemplo, digamos que sua classe nota possui os atributos titulo e descrição – e você editou apenas o título. A documentação indica que você deve passar ambos os atributos preenchidos para o serviço (mesmo só tendo editado o título).

Para resolver essa questão de forma elegante a comunidade adotou, por convenção, o uso de um quinto verbo HTTP: PATCH.

Verbo PATCH:

Usado para editar o recurso sem a necessidade de enviar todos os atributos – o consumidor envia apenas aquilo que de fato foi alterado (mais o ID como parâmetro, para que o serviço saiba o que vai ser alterado).

Padrões de resposta do serviço

A documentação indica que o serviço pode retornar o resultado em diversos formatos – JSON, XML, texto plano, etc. Contudo, atualmente o formato mais adotado tem sido o JSON, por seu formato leve, legível e sua fácil interpretação por diversas tecnologias.

Além disso, o protocolo HTTP dispõe de diveros códigos (ou status) que devem ser incluídos na resposta, indicando o resultado do processamento.

Os códigos iniciados em "2" indicam que a operação foi bem sucedida. Nessa categoria temos, por exemplo, código 200 (OK), iniciando que o método foi executado com sucesso; 201 (Created) quando um novo recurso foi criado no servidor; e 204 (No Content) quando a requisição foi bem sucedida, mas o servidor não precisa retornar nenhum conteúdo para o cliente

Já os códigos iniciados em "4" indicam algum erro que provavelmente partiu do cliente. Por exemplo, o código 400 (Bad Request) indica que a requisição não pôde ser compreendida pelo servidor, enquanto o 404 (Not Found) indica que o recurso não foi localizado.

Há, ainda, os códigos que indicam erro do lado do servidor. Nesse caso, eles iniciam com "5", como o 500 (Internal Server Error), que indica que ocorreu um erro internamente no servidor que o impediu de processar e responder adequdamente a requisição.

Pra conhecer todos os status do HTTP, você pode consultar a especificação do protocolo.

Diferenças entre PUT e POST

Encontramos na literatura indicações de que apenas três verbos são suficientes para um CRUD completo: GET, DELETE e PUT – sendo o PUT utilizado para criar ou editar um recurso.

Interpretando a documentação, temos o seguinte: PUT é usado para criar ou editar um recurso, enquanto POST pode ser utilizado para qualquer coisa, cabendo ao back-end a definição dessa semântica.

O mundo não-acadêmico, no entanto, adotou por convenção o uso do POST para incluir e do PUT para alterar – e em situações de programação mais elegantes, também o uso de PATCH para editar parcialmente.

3.1 REST e Restful

O que é API REST e RESTful Conheça as definições e diferenças?
O primeiro caso: a **API REST** (representational state transfer) é como um guia de boas práticas. ... Com o alicerce da **API REST**, estrutura-se a **RESTful** que significa a manipulação dos princípios da **REST** para o desenvolvimento de soluções ou serviços.

3.2 Diferenca dentra URI E URL

URI: Uniform Resource Identifier	
URI identifica	
exemplo de uma URI seria: cursos.alura.com.br/forum ela identifica a página do fórum da alura.	
URL: Uniform Resource Locator	
URL localiza um recurso, entretanto, localizadores também são identificadores, ou seja, as URLs são um subconjunto do conjunto de URIs.	
agora https://cursos.alura.com.br/forum é uma URL pois além de identificar a página do fórum, ela diz também como acessar ele, por meio do protocolo https.	

3.3 PUT OU GET?

HTTP

O HTTP assim como o português é um protocolo de comunicação.
Em um time, significa menos gasto de energia para resolver um problema. Torna o desenvolvimento ágil e mais eficiente.

O HTTP é um protocolo de comunicação. A definição de comunicação é:
"Comunicação consiste na emissão, transmissão e recepção de uma mensagem.
Através de métodos convencionais ou não. Por meio da fala ou da escrita, por sinais, signos ou símbolos"

O HTTP foi projetado para ser simples e legível por humanos. As mensagens podem ser lidas e entendidas.
Utiliza uma estrutura bem definida, como **verbos** GET, POST ou **substantivos** como **OPTIONS** ou **HEAD**.

URL

A URL indica qual o **recurso** que deseja obter do servidor.
O **HEADER** e o **BODY** são informações adicionais para o servidor processar a solicitação.

URI
(UNIFORM RESOURSE IDENTIFIER)

URI **identifica**

exemplo de uma
URI seria:
cursos.alura.com.br/for
um ela **identifica** a página do fórum da alura.

URL
(UNIFORM RESOURSE E LOCATOR)

URL **localiza** um recurso, entretanto, localizadores também são identificadores, ou seja, as URLs são um **subconjunto** do **conjunto** de **URIs**.

agora:
https://cursos.alura.com.br/forum
é uma URL pois **além de identificar a página** do fórum, **ela**

diz também como acessar ele, por **meio do protocolo https.**

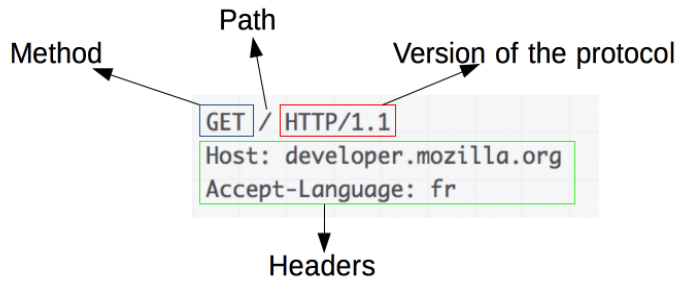
HTTP

Uma API e o Browser se **comunicam através de mensagens HTTP**.

Existem **dois tipos de mensagens HTTP, Requests e Responses**, cada uma com seu próprio formato.

REQUEST

Uma request possui o seguinte formato:



URI
(UNIFORM RESOURSE IDENTIFIER)

Na imagem é possivel observar os seguintes elementos:

- **Method** - É o método do HTTP. Abaixo há uma tabela com os métodos mais utilizados.
- **Path** - É o endereço do **resource** que deseja obter.
- **Version of the protocol** - Versão do protocolo HTTP. Atualmente o mais comum é o HTTP/1.1. Mas já há a versão 2 do HTTP. Inclusive o GRPC já suporta.
- **Headers** - Os headers entregam informações adicionais para o server.
- **Body** - É o conteúdo, geralmente é utilizado em conjunto com os verbos POST, PUT e PATCH.

Métodos mais utilizados:

HTTP Method **Descrição**

OPTIONS → **Busca** os metodos **http** válidos e outras opções

GET → **Busca** um **resource**

HEAD → **Busca apenas** o **header** de um resource

PUT → **Atualiza** um **resource**

POST → **Cria** um **resource**

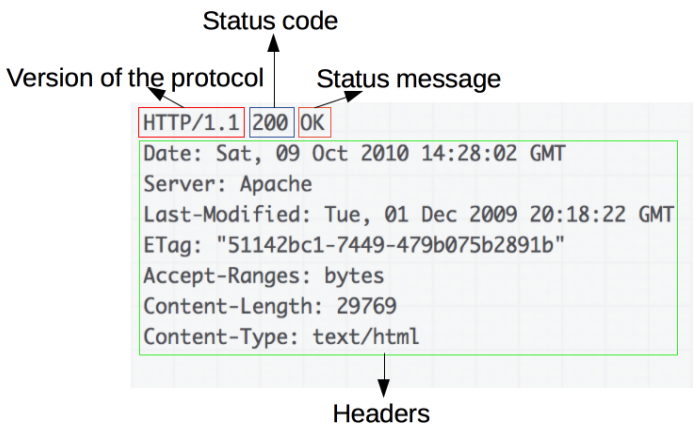
DELETE → **Remove** um **resource**

PATCH → **Atualiza parcialmente** um **resource**

RESPONSE

O HTTP é connectionless, isso significa que para toda chamada haverá um **RESPONSE**.
Mesmo que o client que chamou não esteja mais esperando a resposta. Por exemplo, quando o usuário fecha o browser.

A response, possui o seguinte aspecto:



No response é possivel observar:

- A versão do protocolo HTTP.
- **Status Code** - Código de status. O código de status é separado por categorias.
  - 2xx - **Status** de **sucesso**
  - 3xx - Categoria de **redirecionamento**
  - 4xx - **Erro** no **Cliente**
  - 5xx - **Erro** no **server**

- **Status Message** - A frente do Status Code virá uma breve **descrição do que significa o Status Code**.
- **Headers** - Haverá **informações para o Client**, por exemplo o formato da resposta, se o conteúdo pode ser cacheado.
- **Body** - **Opcionalmente**, dependendo do tipo de request, haverá no **body o conteúdo da resposta**.

O que é REST

REST é um estilo de arquitetura.
Ele fornece padrões para a comunicação entre sistemas.
REST não é um padrão exclusivo para HTTP.
Embora as bases do REST e do HTTP sejam as mesmas.

Na arquitetura REST, os clientes enviam solicitações para **recuperar** ou **modificar** recursos e os servidores enviam respostas para essas solicitações.

REST é o acrônimo (siglas são formadas pelas letras iniciais de outras palavras): de **RE**presentational **S**tate **T**ransfer. Esse padrão é descrito em seis restrições.

- Client-server
- Stateless
- Cacheable
- Code on Demand (Opcional)
- Uniform Interface

PRINCIPIOS DO REST

Client-Server - Separar as responsabilidade do **frontend** do **backend**. Esse é um conceito bem comum. Separar o front do back há ganhos significativos em testes, escalabilidade com reflexos até na organização dos times dentro da empresa.

Stateless - O servidor não mantém estado.
Cada solicitação do client deve conter informações necessárias para o server entender a solicitação. O estado da sessão é mantido inteiramente no client.

Cacheable - A resposta de uma solicitação deve implicitamente ou explicitamente informar se o dado pode ser mantido em cache ou não.
O cache deve ser mantido e gerenciado pelo Client.

Uniform interface - Este principio é definido por quatro restrições:

- **Identificar** os recursos (URI)
- **Manipular** recursos através de representações (Verbos HTTP).
- Mensagens **auto-descritivas**, cada requisição deve conter informações suficientes para o server processar a informação.
- **HATEOAS** - **H**ypermedia **A**s **T**he **E**ngine **O**f **A**pplication **S**tate. Esta restrição diz que a **response deve conter links de conteúdos relacionados ao resource**.

PRINCIPIOS DO REST

Por exemplo:

- GET http://api.jpproject.net/users/1

Response:

```
{
       "userName": "bruno",
       "email": "bhdebrito@gmail.com",
       "phoneNumber": "11989500000",
       "name": "bruno",
       "links": [
             {
         "href": "10/claims",
         "rel": "claims",
         "type" : "GET"
      }
   ]
}
```

Neste exemplo repare que o response contém links para o **resource** claims.
Dessa forma o controle de fluxo da aplicação é controlado através do server e não escrito na pedra pelo **Frontend**.

Nsse próximo exemplo de HATEOAS, onde é possivel saber as opções de Status de um resource.

```
{
 "userName": "bruno",
 “status”:”Active”,
 ...
 “links”: [
  {“rel”:“block”, “href”:”/users/10/block”},
  {“rel”:“confirm-email”, “href”:”/users/10/confirm-email””}
 ]
}
```

Layered system - O sistema deve ter uma arquitetura em camadas.
A camada acima não pode ver além da camada imediatamente abaixo dela.

Code on demand (opcional) - o REST permite que a funcionalidades do client seja estendida baixando e executando applets ou scripts.

RESOURCES

O REST é Resource Based. Esse termo irei tratar com o nome em inglês pois é um item chave dentro do conceito.

Um Resource é a chave da abstração no REST.
Um resource é qualquer coisa importante o suficiente para ser referenciado com um nome.
O REST usa um URI (Uniform Resource Identifier) para **identificar** o resource. Resource é qualquer coisa que possa ter uma URI.

REST não limita a escolha da nomenclatura, qualquer coisa que seja indentificável e tenha uma URI está previsto pelo padrão.

Escrevi um artigo dizendo que REST não é CRUD. Abordando em mais detalhes esse tópico.

Entender **Resources** é a chave para a criação de API's melhores.

CONTACT

Brunna Croches

Developer Full Stack

brunnacroches.dev

linkedin.com/brunnacroches

github.com/brunnacroches

@brunnacroches.dev

discord.com/brunnacroches

brunnacroches@gmail.com

let's share