

Criando Nossa Própria API

DigitalHouse>

O que vamos ver hoje

- Criar API juntos aplicando conceitos do material online
- Aprofundamento de conceitos de API e REST API (status Code e métodos HTTP)

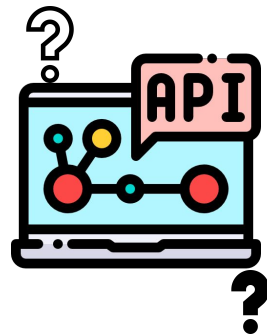


TODA **API** DEVE SEGUIR
O PADRÃO **REST**?

Antes de entender padrões é preciso entender no que serão aplicados...

As APIs são:

- A. rotas de um sistema
- B. comunicação com DB
- C. **a** e **b**
- D. nenhuma das anteriores



API

- **API** é um acrônimo para **Application Programming Interface** ou, em português, algo como Interface para Programação de Aplicações.
- Nada mais é do que uma coleção de funcionalidades que permite que programadores possam acessar dados, informações ou sistemas completos sem precisar necessariamente entender como são feitas as implementações naquele ecossistema.
- Formato de comunicação universal (**JSON** / **XML**)



O padrão REST

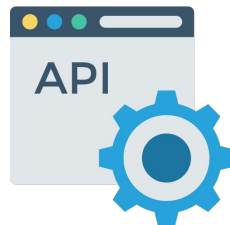
- **Métodos HTTP** Semânticos:

- **GET**

- sem passagem de parâmetro: retorna lista
 - com passagem de parâmetro: retorna um registro

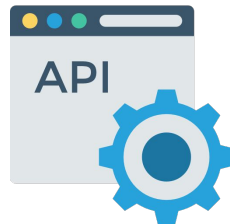
- **POST**

- com passagem de parâmetro: cria um registro



O padrão REST

- **PUT**
 - com passagem de parâmetro: atualiza um registro
- **PATCH**
 - com passagem de parâmetro: atualiza dados específicos de um registro
- **DELETE**
 - com passagem de parâmetro: exclui um registro



O padrão REST

- Deve-se preocupar com o status resultante de cada requisição
- Isso porque o consumidor da API precisa saber se houve sucesso e qual o tipo de erro caso ocorra um



Lista de Status Code e seus significados:

<https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status>

E agora, sabendo de tudo isso,
qual é a sua resposta à primeira
pergunta?

Deve seguir o padrão **REST**?

Bora praticar!



Atividade I - Uma API para um sistema existente

- Criar endpoints no padrão REST que permitam:
 - listagem
 - edição
 - adição



Mais detalhes do enunciado no doc **Atividade I**

Para ir além

→ **Documentação no site oficial MDN:**

https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Client-side_web_APIs/Introduction



REST

“ **REST** é um tipo de arquitetura de serviços que **fornece padrões** entre sistemas de computador para estabelecer **como** eles se **comunicarão** entre si. ”



Índice

1. Conceitos-chave
2. Formatos de envio de dados

1 | CONCEITOS-CHAVE

ARQUITETURA CLIENTE - SERVIDOR

Um dos padrões desta arquitetura indica que a aplicação do cliente e a aplicação do servidor podem ser desenvolvidas, ampliadas e alteradas, sem interferir umas nas outras.

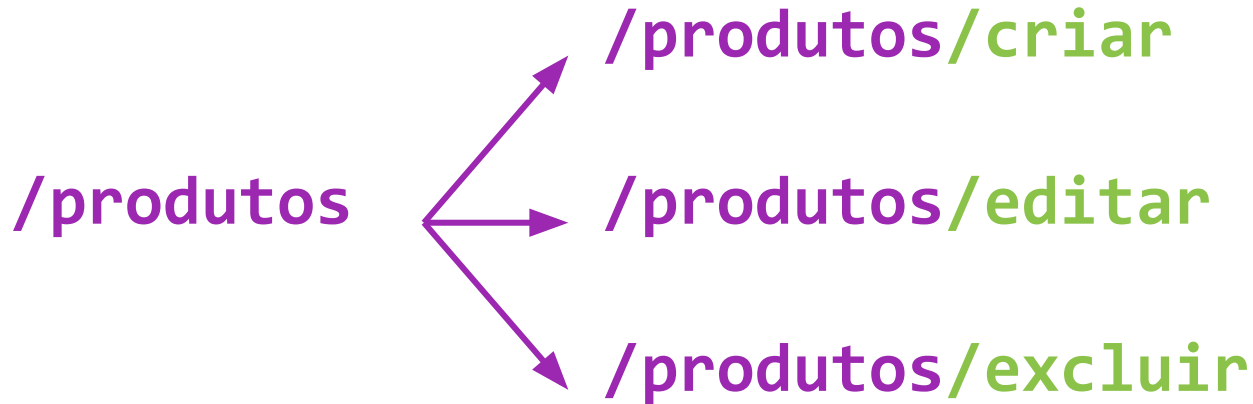
Quando estamos criando uma API sob essa arquitetura, normalmente chamamos as solicitações do cliente de **ENDPOINTS** e a resposta do nosso servidor de **RECURSOS**.



Como boa prática, recomendamos trabalhar primeiro com o servidor e depois com a interface do cliente.

RECURSOS UNIFORMES

Um endpoint está ligado ao recurso que solicitamos. Um recurso no sistema deveria ter somente um identificador lógico, e este prover acesso à toda a informação relacionada.



STATELESS

O paradigma REST propõe que todas as interações entre cliente e servidor devem ser tratadas como novas e de forma absolutamente independente.

Portanto, se queremos uma aplicação que possa distinguir os usuários que estão logados dos que não estão, devemos mandar toda a informação de autenticação necessária em cada pedido.



Isso permite desenvolver aplicações mais **confiáveis**, **performáticas** e **escaláveis**.

Cacheable

No paradigma REST, o *cache* de dados é uma ferramenta muito importante, que é implementada do lado do cliente para melhorar a performance e reduzir a demanda ao servidor.

O *cache* deve ser aplicado aos recursos tanto quanto possível.

2 | Formatos de envio de datos

JSON

É o formato mais comum para o envio de dados.

Quando queremos enviar dados no formato JSON, devemos adicionar no header o seguinte:

```
{  
  "Content-Type": "application/json"  
}
```



Como boa prática, é recomendado trabalhar com este formato.

RAW

É utilizado para mandar dados com texto sem nenhum formato em específico.

TEXT

É utilizado para enviar dados que não estejam em formato JSON, como arquivos html e css.

URL-encoded

É o envio de dados por meio de codificação em forma de URL, algo muito similar a uma *query string*.

Um dado enviado por esse método seria visto da seguinte maneira:

```
{ }
```

```
email%3Dcosme%40fulano.fox%26password%3Dverysecret
```


Consumo de APIs Back-end

DigitalHouse>

O que vamos ver hoje

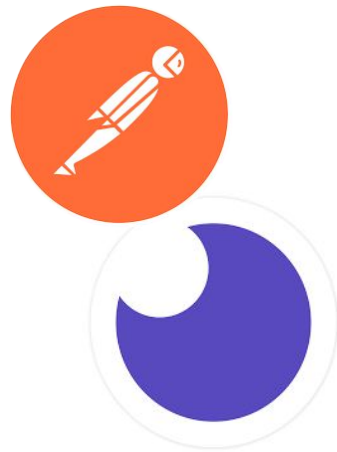
- Consumir a API criada na última aula
- Consumir API de terceiros
- Consumo no backend e ferramentas
API Client



POR QUE FAZER USO DE UMA
FERRAMENTA **API CLIENT**?

API Client

- Permite testes diretos para consumir endpoints (sem precisar de ter um projeto backend só para isso)
- Uso gratuito (para uso individual e simples)
- Contém e permite preparar documentação da API (casos de resposta de requisições com erro ou sucesso)



As duas ferramentas mais utilizados são Postman e Insomnia

Vamos ver então como funciona na prática!



Atividade I - Como o Postman lida com nossa API?

Consulte os seguintes endpoints criados na última aula:

- listagem de itens
- edição
- exibição de único item



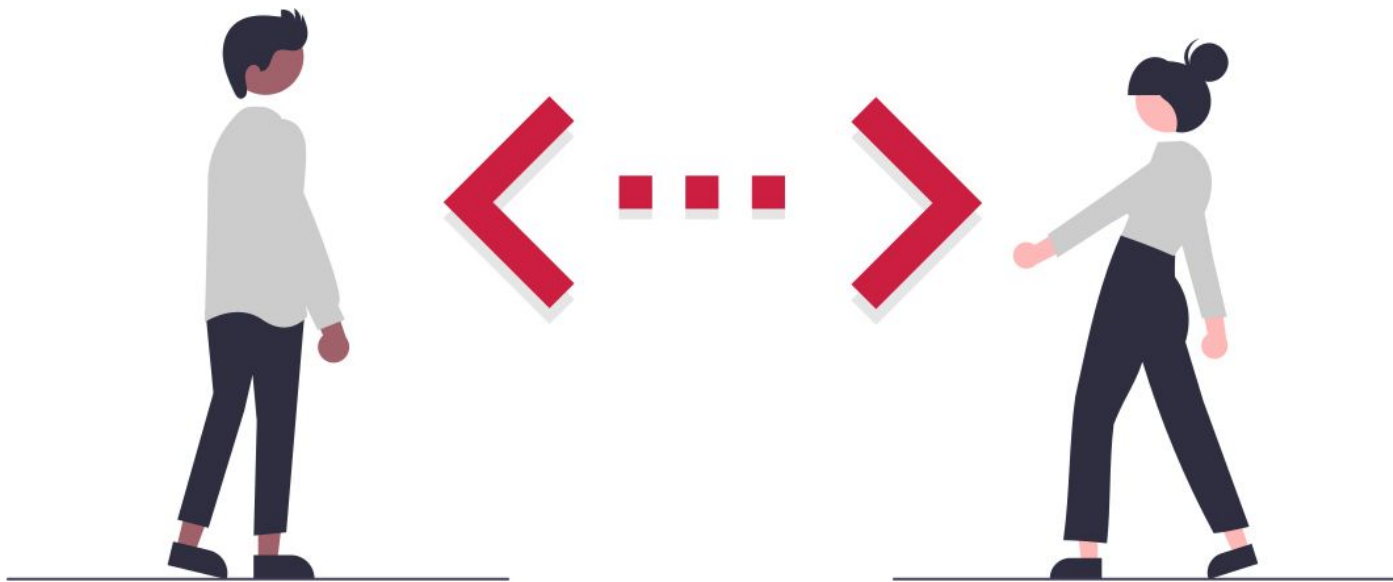
Para mais detalhes da prática consulte o doc **Atividade I**

Está lembrado dos passos para consumo de API no Backend?

- Instalando AXIOS `npm install axios`
- Criar arquivo de configuração do endpoint - **default.js**
- Montar o request requerindo para o arquivo a **biblioteca axios** e o **arquivo de configuração**
 - Montar **url** e especificar **método http**
- Tratar response obtida com **.then()** e **.catch()**



Hora de transformar essas informações em código!



Atividade II - Do postman para o código

Chegou o momento de passarmos tudo o que testamos no Postman para o código!

- Configure a biblioteca axios no projeto
- Crie os requests para cada endpoint a ser consumido
- Desenvolva como o sistema lidará com casos de sucesso ou falha no request



Para mais detalhes da prática consulte o doc **Atividade II**

Atividade III - E para consumir uma API pública?

Já se perguntou como os sites de compra e venda preenchem as informações básicas de endereço para cálculo de frete/cadastro apenas com o número do CEP?

Vamos descobrir como isso acontece nesta prática



Para mais detalhes da prática consulte o doc **Atividade III**

Para ir além

→ **Documentação no site oficial Axios:**

<https://axios-http.com/docs/intro>



POSTMAN

“ É uma **ferramenta** que simplifica muitas tarefas na hora de **testar** o funcionamento de uma **API** sem ter uma interface. ”



Índice

1. Instalação
2. Interface
3. Teste

1 | INSTALAÇÃO

INSTALAÇÃO

A primeira coisa que faremos será acessar o site oficial do Postman <https://www.postman.com/downloads/> e baixar o instalador que corresponde ao sistema operacional que usamos. Em seguida, iremos executamos a instalação.

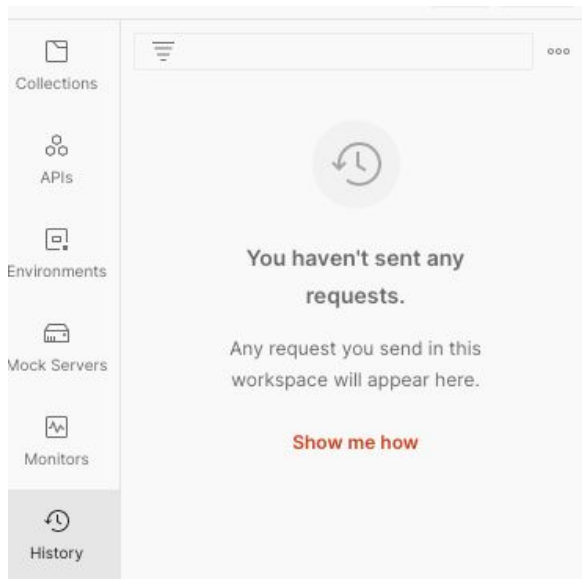


Postman também pode ser usado de forma online.

2 | INTERFACE

INTERFACE

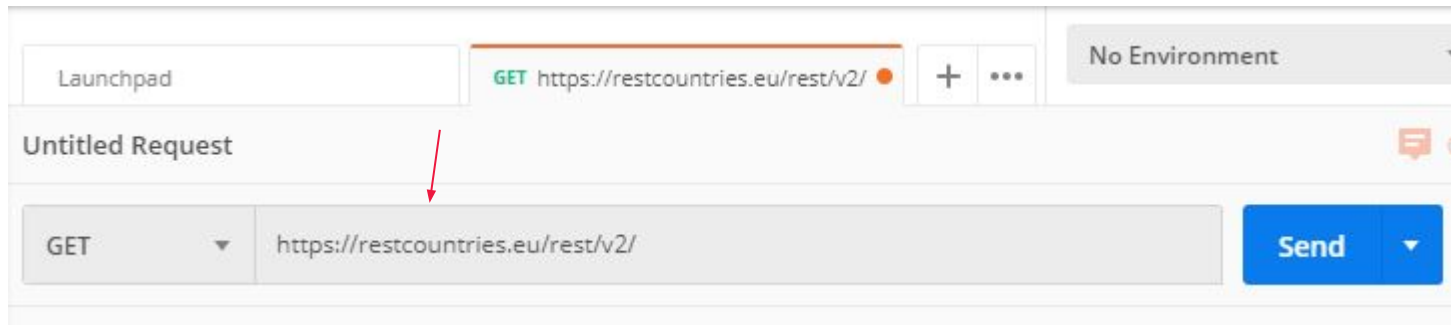
No painel da esquerda, encontraremos o histórico de todas as solicitações que fizemos. Obviamente, podemos apagar algumas ou todas elas.



INTERFACE

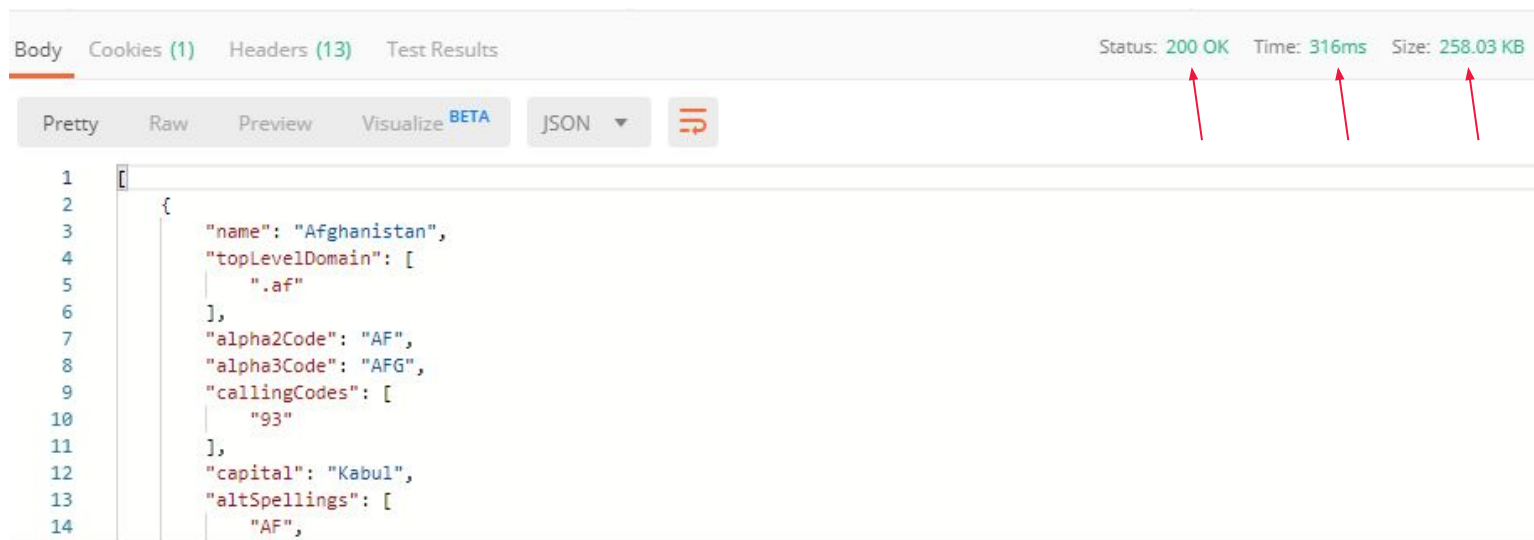
No painel da direita é onde mais iremos operar. Faremos nossa primeira requisição por **GET** a um **endpoint** (anteriormente visto).

Para isso, clicamos no símbolo **+**. Depois, inserimos nossa **URL** para testar. Neste caso, escolhemos **https://restcountries.eu/rest/v2/** para listar todos os países. Finalmente, clicamos em **SEND** para ver os resultados.



INTERFACE

Como podemos observar, há um indicador de **status** à direita, outro indicador de quanto **tempo** levou para realizar o request, o **tamanho** do json e abaixo encontra-se o **resultado**.



The screenshot displays the 'Body' tab of a web browser's developer tools. The response status is '200 OK', the time taken is '316ms', and the size is '258.03 KB'. The response body is a JSON object representing Afghanistan, with fields for name, top-level domain, alpha codes, calling codes, capital, and alt spellings. Red arrows point from the text in the paragraph above to the status, time, and size indicators in the interface.

Body Cookies (1) Headers (13) Test Results Status: 200 OK Time: 316ms Size: 258.03 KB

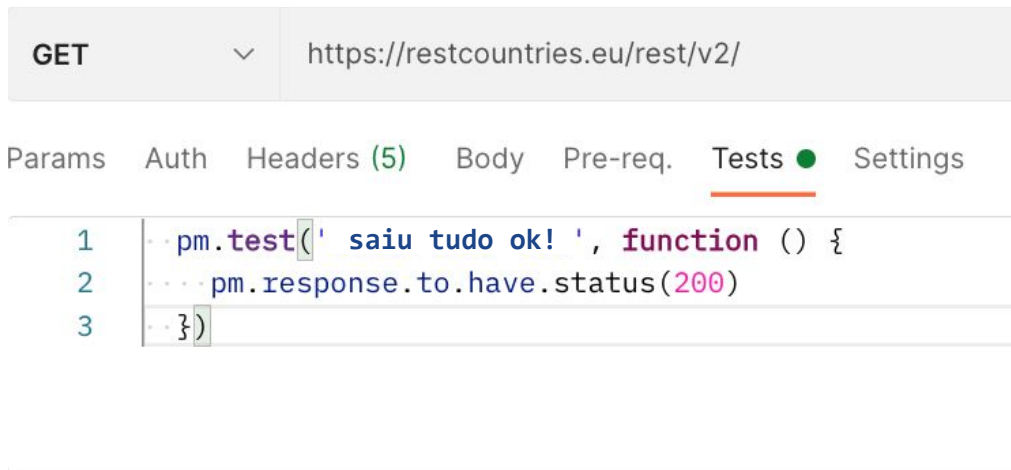
Pretty Raw Preview Visualize BETA JSON

```
1 [
2   {
3     "name": "Afghanistan",
4     "topLevelDomain": [
5       ".af"
6     ],
7     "alpha2Code": "AF",
8     "alpha3Code": "AFG",
9     "callingCodes": [
10      "93"
11    ],
12    "capital": "Kabul",
13    "altSpellings": [
14      "AF",
```

3 | TESTE

TESTE

Vamos para a aba **Tests**. No espaço em branco, podemos escrever usando a sintaxe de Javascript que já conhecemos. Para isso, usaremos as funções "**pm**" do PostMan, que nos permitem executar os testes.



AXIOS

“ O objetivo do **Axios** é fazer comunicação **HTTP** através de código, configurando **requisições** como **objetos Javascript**. ”



Índice

1. Instalação
2. Requisições
3. Resposta

1 | INSTALAÇÃO

ADICIONAR A BIBLIOTECA

Para incorporar a biblioteca ao **back-end**, simplesmente temos que **executar** o comando em nosso **console**:

```
>_ npm install axios
```

Para incorporar a biblioteca ao **front-end**, podemos incluir uma tag de **script** com o link correspondente, ou usar um **CDN**, desta forma:

```
HTML https://cdnjs.cloudflare.com/ajax/libs/axios/0.21.1/axios.min.js
```

3 | CONFIGURAÇÕES

CONFIGURAÇÕES

- Primeiro, criaremos uma pasta dentro de **/src** chamada **requests**, para que possamos ter todas as requisições para cada recurso dentro do mesmo diretório.
- Em seguida, criaremos o arquivo **default.js** na pasta de **requests**, este arquivo gera um objeto que representará todos os parâmetros padrão que utilizaremos em cada requisição.

DEFAULT.JS

- **Timeout:** representa o **tempo máximo** de espera de resposta para uma requisição. Se o servidor demorar mais, o **Axios** cancelará a requisição.
- **BaseURL:** seu valor é a url que será usada em todas as requisições que fizermos.

```
{}  
  
const default = {  
  baseURL: 'www.spotify.com/api/',  
  timeout: 4000  
};  
  
module.exports = default;
```

4 | REQUISICÃO

MONTAR AS REQUISIÇÕES

- Primeiramente, identificamos qual recurso queremos acessar, e assim, poderemos criar um arquivo que conterá nossa requisição e todos aqueles que interagem com este recurso.

Dentro deste arquivo, **teremos** duas partes:

- A primeira conterá a **url que identifica** o recurso e também a importação de **Axios** e padrões.
- A segunda será um **objeto literal** que representará **os serviços** que serão funções, que podemos acessar pelo **nome da chave**.

3 | RESPOSTA

CONSUMIR UMA REQUEST

- Sempre que precisamos usar o serviço, primeiro importamos o módulo e o armazenamos em uma variável para invocar os métodos que ele contém.
- Em seguida, chamamos o método de que precisamos para retornar os dados para nós.

{}

```
const artistRequest = require('./src/requests/artistsRequest');
artistRequest.getArtista(59).then(response => {
  console.log(response.data);
}).catch(error => {
  console.log(error.response);
});
```

OUTRAS OPÇÕES

Vale lembrar que o **Axios** não é a única opção para se comunicar com **APIs**, outra boa opção é a **Fetch API**, que já vem como padrão no JavaScript para **front-end**.

Caso queira utilizá-lo no **back-end**, basta baixar o **Node Fetch** via NPM da mesma forma que é feita com o axios.



