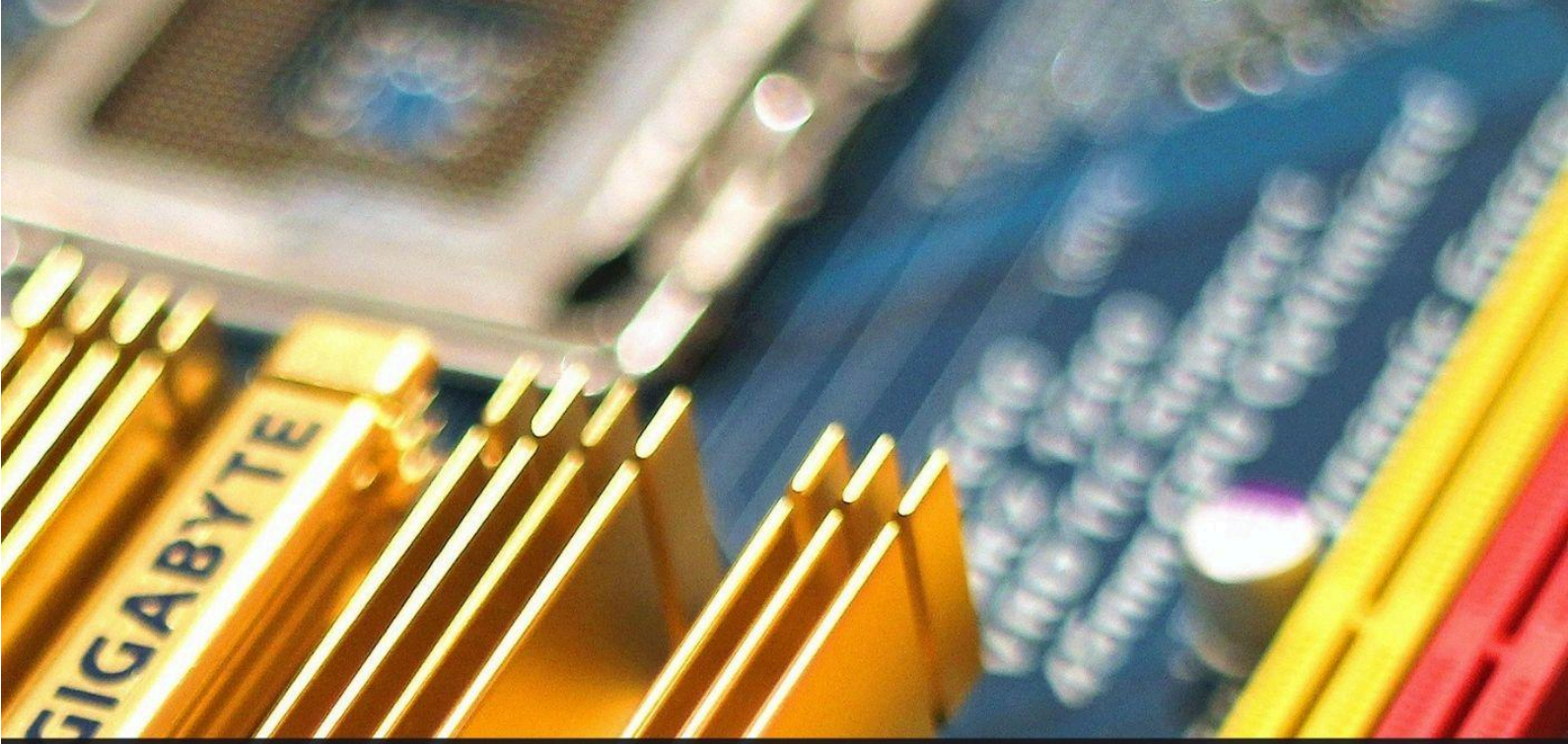
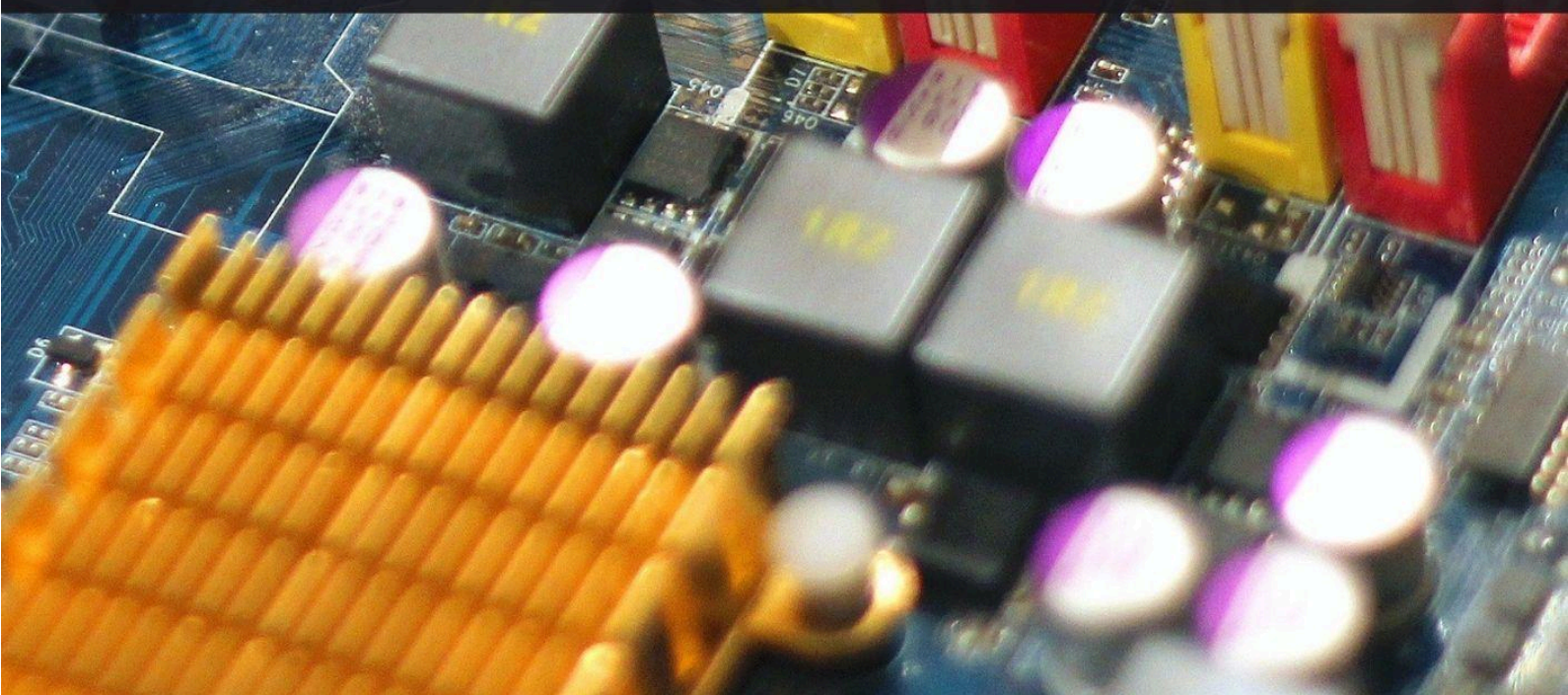


DESENVOLVIMENTO DE APIs E MICROSSERVIÇOS



4

Protocolo HTTP e redes

Lucas Mendes Marques Gonçalves

Resumo

O que acontece quando você acessa um site?
Como isso é considerado na hora de criar e acessar servidores?
Essa compreensão é o objetivo dessa aula

4.1. URLs

A primeira tecnologia de rede que precisamos estudar, se nos interessa acessar ou desenvolver APIs, é a URL.

URL, do inglês **Uniform Resource Locator**, é uma string que usamos para identificar um computador ou um recurso em uma rede. Normalmente, usamos para descrever um arquivo ou serviço na internet.

Por exemplo, quando acessamos http://www.pudim.com.br/pudim.jpg, estamos baixando um arquivo chamado pudim.jpg, e esse arquivo vem do servidor www.pudim.com.br.

Vamos entender algumas partes que podem ser usadas para compor uma URL.

Tome o exemplo:

http://teste.example.com.br:8080/cadastro/u?usr=joao&id=123#parte5

O teste.example.com.br é chamado **domínio**. Especifica com qual computador queremos conversar.

:8080 chama-se **porta**. O intuito é designar um processo dentro de um computador. Estamos acessando o servidor teste.example.com.br, e ele pode ter vários processos aguardando a nossa conexão. Queremos especificamente o processo registrado no número 8080.

http:// é o **protocolo**. Agora que sabemos com que máquina falar, e com que processo dentro da máquina, temos que convencionar como falar. Processos diferentes podem esperar protocolos diferentes, como **mysql://**, **ftp://** ou **ssh://.** Nessa disciplina

estaremos inteiramente focados no protocolo **http** (e seu primo muito próximo, o **https** - programadores normalmente podem ignorar a distinção, e o profissional de devops ou administradora de sistema lida com isso)

/cadastro/u é chamado **caminho**. Serve para informar ao servidor que recurso desejamos. Lembra uma estrutura de pastas, uma dentro da outra.

?usr=joao&id=123 é a **query string**. Serve para informar ao servidor que recurso desejamos. Lembra uma estrutura de dicionário, com chaves e valores (no caso, por exemplo, temos a chave usr associada ao valor joao).

#parte5 Esse pedaço da URL chama **fragmento**. Ele não é enviado ao servidor, mas é apenas usado para processamento local.

Atualmente, é considerada boa prática usar bastante **caminho**, e evitar a **query string** -- isso quando estamos construindo o servidor. Se o servidor não é nosso, usamos o que nos foi dado.

4.2. HTTP

O protocolo mais utilizado para acessar recursos na internet é o HTTP.

Quando um cliente faz um pedido HTTP para um servidor, ele manda:

1. A URL do pedido (como http://www.pudim.com.br/pudim.jpg);
2. Um verbo (tipicamente, *GET, POST, PUT, DELETE ou HEAD*) para especificar uma ação a fazer - GET para baixar a imagem, por exemplo;
3. Um corpo da requisição (um arquivo que queremos que o servidor processe);
4. Headers (detalhes da conexão - o mais famoso é o cookie).

De volta, ele espera receber:

5. Um corpo da resposta (um arquivo que o servidor nos devolve);
6. Um código de status (um código numérico, com 200, 201, 400, 404, 500) que diz se o pedido foi bem sucedido ou não;
7. Headers (detalhes da conexão).

Vamos discutir brevemente os possíveis códigos de status e verbos HTTP.

4.3.Verbos HTTP

Existem muitos verbos HTTP. Vamos nos focar nos seguintes:

- **GET** – Utilizado como parte da navegação para baixar recursos (downloads).
- **POST** – Utilizado para enviar recursos ao servidor.
- **PUT** – Utilizado para enviar recursos ao servidor.
- **DELETE** – Utilizado para excluir recursos do servidor.
- **HEAD** – Igual ao GET, mas sem o corpo da resposta. Se eu pedir uma imagem, eu recebo tudo menos a imagem - é só pra eu me preparar, saber o tamanho da imagem, se ela mudou desde o último acesso, qual o formato do arquivo (e outras coisas úteis que aparecem nos headers).

Podemos caracterizar os verbos com base na possibilidade de envio e recebimento de arquivos (corpo da requisição e corpo da resposta). Vemos isso na tabela 4.1.

Tabela 4.1. Verbos HTTP e corpo da mensagem

Verbo HTTP	Corpo na requisição	Corpo na resposta
GET	Não	Sim
POST	Sim	Sim
PUT	Sim	Sim
DELETE	Não	Sim
HEAD	Não	Não

Fonte: do autor, 2021.

No caso do GET, por exemplo, uma requisição não deve enviar arquivo, mas pode receber.

Outra questão importante é a caracterização dos efeitos colaterais. Quando uma API disponibiliza uma determinada URL acessível com o verbo GET, está fazendo promessas a respeito de quais efeitos colaterais podem ou não ocorrer.

O efeito principal de uma chamada HTTP é a devolução de uma resposta (por exemplo, uma imagem, uma página, um dicionário)

Qualquer outro efeito é chamado de EFEITO COLATERAL: Uma compra, salvar as informações de um formulário, salvar um slide do google, enviar um email.

Os verbos diferem a respeito desses efeitos. Alguns são **seguros**, não tem efeito colateral nunca. Alguns são **idempotentes**, podem ter efeito colateral só na primeira chamada, não podem ter nas seguintes.

Tabela 4.2. Verbos HTTP

Verbo HTTP	Corpo na requisição	Corpo na resposta	Idempotente	Seguro
GET	Não	Sim	Sim	Sim
POST	Sim	Sim	Não	Não
PUT	Sim	Sim	Sim	Não
DELETE	Não	Sim	Sim	Não

HEAD	Não	Não	Sim	Sim

Fonte: do autor, 2021.

Vamos destrinchar a tabela 4.2. O que isso quer dizer, na prática? Quando publicamos uma API, e a url http://www.site.com.br/pessoas/12 está acessível por GET, os acessos não devem causar nenhum tipo de efeito colateral. O servidor "esquece" o acesso logo que ele foi feito.

Quando publicamos uma API, e a url http://www.site.com.br/pessoas/12 está acessível por DELETE, os acessos podem causar efeito colateral (provavelmente a remoção deste usuário). Mas se for mandado o mesmo request 5 vezes, o efeito em si deve ocorrer apenas uma vez.

Note o uso da palavra DEVE. Cumpre ao programador (ou programadora) fazendo a API ter esse cuidado. Não é uma coisa automática.

4.4. Códigos de status

Na resposta a nossos pedidos HTTP, o servidor nos informa o que ocorreu através do código de status.

Um código 2xx (por exemplo: 200, 201, 202) representa um pedido bem sucedido. 200 diz apenas “pedido bem sucedido” e os demais servem para dar mais detalhes. 201 diz que o pedido foi bem sucedido, e resultou na criação de um novo recurso (um arquivo foi salvo).

Um código 3xx tem relação com redirecionamento (essa página agora deve ser vista usando outra URL) ou cache (se a página tem que ser baixada novamente ou a versão que você já tem é suficientemente atual).

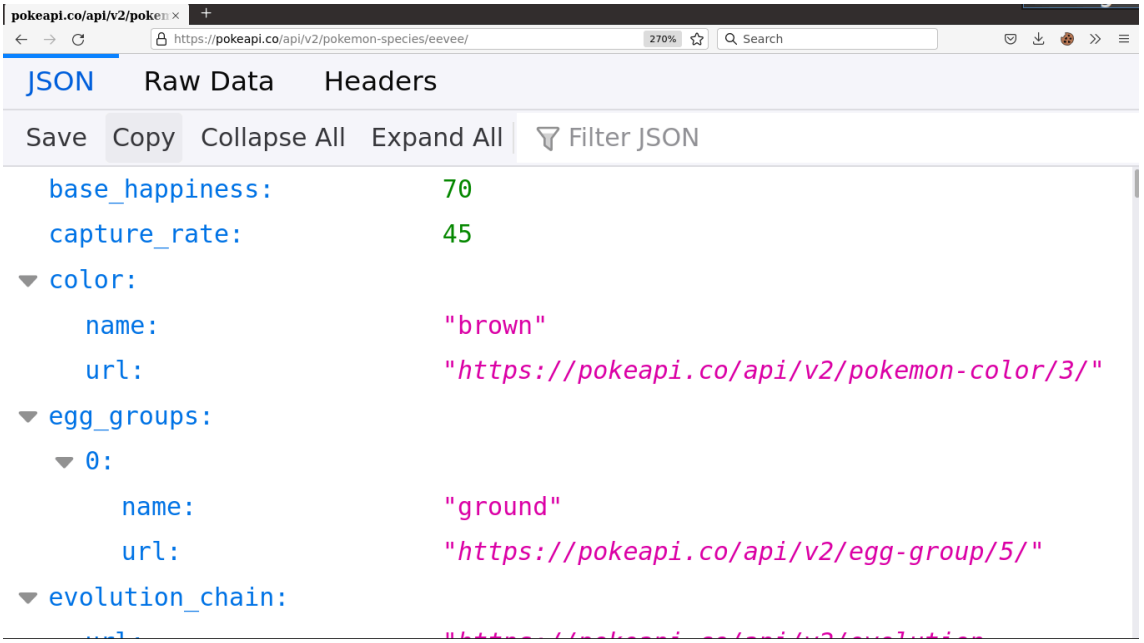
Um código 4xx representa erro do usuário. 400, por exemplo, é “usuário cometeu algum erro”, enquanto 404 representa “a URL digitada está incorreta” e 401 significa “usuário não tem permissão para fazer o pedido”.

Um código 5xx representa problemas no servidor. Por exemplo, o banco de dados parou de funcionar. Como cliente, não há o que fazer, só esperar.

4.5. Acesso via firefox

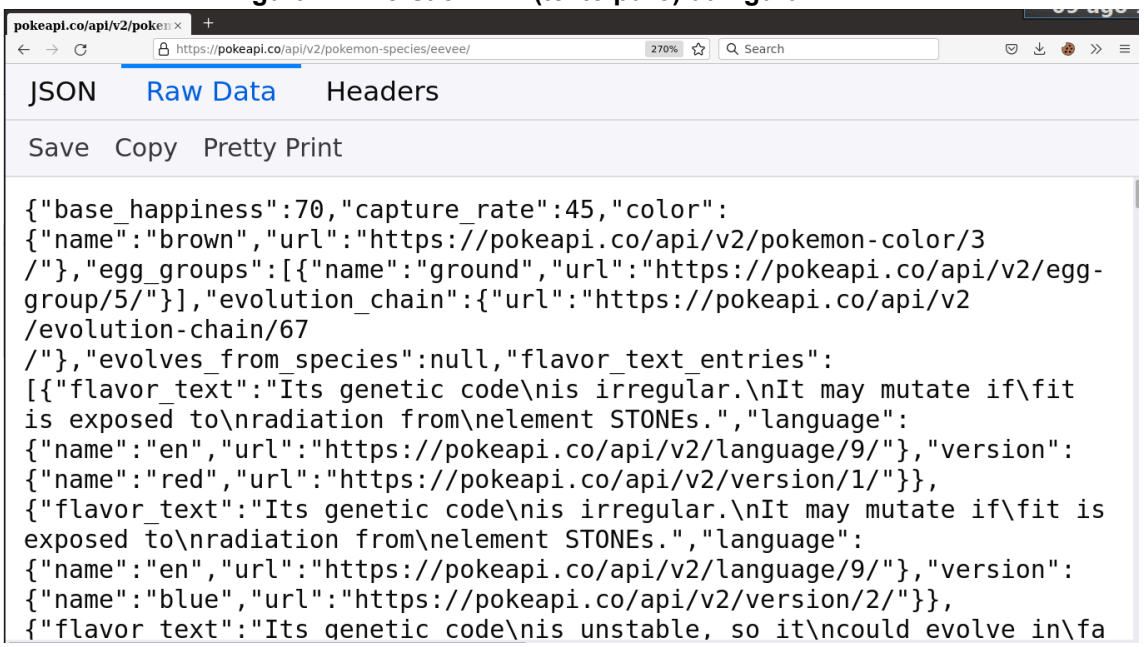
Quando a API disponibiliza URLs interessantes via GET, podemos fazer o acesso no firefox. Os dicionários fornecidos (assumindo o formato mais comum: *JSON*) serão interpretados automaticamente e podemos interagir com eles de forma mais fácil.

Figura 4.1. Acesso a servidor via firefox



Fonte: do autor, 2021.

Figura 4.2. Versão RAW (texto puro) da figura 4.1



Fonte: do autor, 2021

Como podemos ver na figura 4.2, a leitura de um arquivo json puro pode ser complicada. Por isso, é interessante usar o firefox (ou instalar uma extensão em outro navegador) para ler arquivos .json. Com o firefox, temos um ambiente bem interessante para entender os arquivos .json que baixámos, como podemos ver na figura 4.1.

4.6. Acesso via POSTMAN

Para URLs disponibilizadas com outros verbos (que não GET), vale a pena usar um

programa especializado. Uma opção é o POSTMAN (vide: https://www.postman.com/downloads/ e instruções na video aula).

4.7. Instruções para acesso como na videoaula

Na videoaula, utilizamos um servidor para experimentar as funcionalidades do POSTMAN e do protocolo HTTP. É altamente recomendável seguir esses passos em casa. Para isso, você deve abrir o cmd do windows e rodar os seguintes comandos:

1. pip install --user flask
2. pip install --user requests

Depois disso, poderá executar o servidor fornecido usando o comando:

python corredores.py (mas antes terá que abrir no cmd o diretório onde baixou o programa corredores.py).

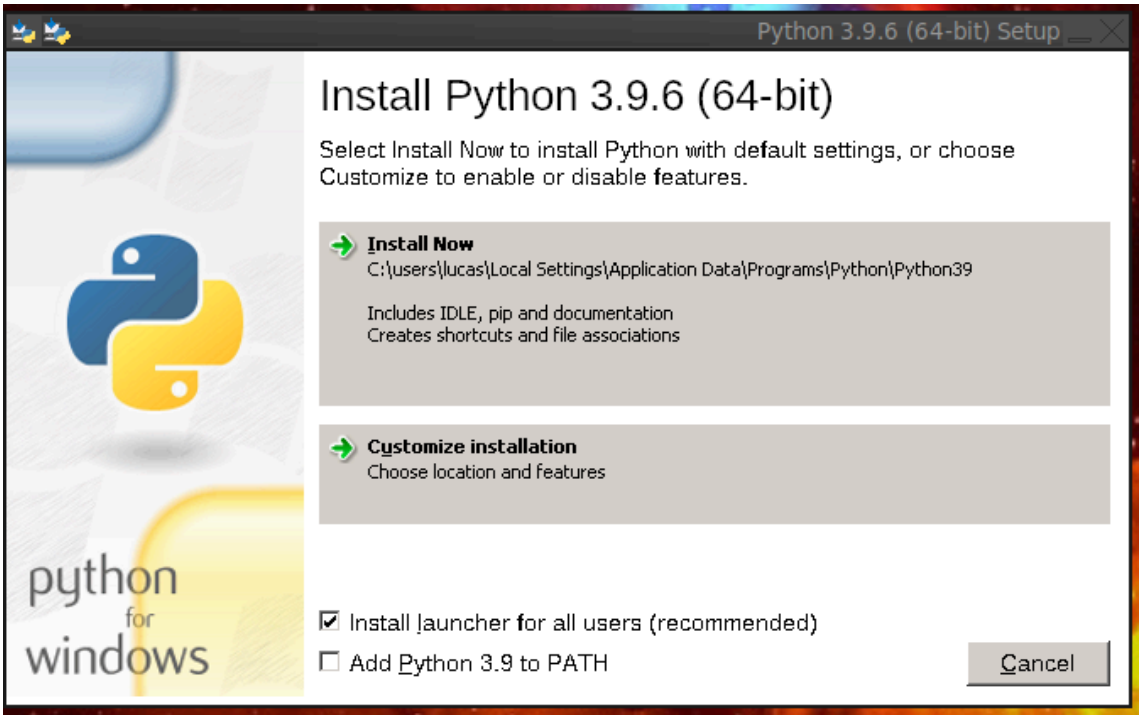
4.8. Solução de problemas

Se você tiver o erro: **O comando pip não é um programa válido - 'pip' não é reconhecido como um comando interno ou externo.** Use os passos abaixo para resolver o problema:

1) Se você está usando linux ou mac, rode o comando de instalação usando pip3 no lugar do pip;
2) Se você está usando windows, experimente o comando python no cmd. Se funcionar (ou seja, o python funciona e o pip não), sua situação não é usual. Peça ajuda no fórum ou ao professor;
3) Se ambos os comandos (pip e python) não funcionarem no cmd, reinstalar o python deve resolver.

Ao reinstalar, marcar a opção “adicionar o python no path” ou “adicionar o python nas variáveis de ambiente”. Isso faz com que os comandos “python” e “pip” passem a ser comandos válidos no cmd.

Figura 4.3. Instalador do python



Fonte: Python, 2021.

Na figura 4.3, vemos que a caixa **Add Python 3.9 to PATH**, está desmarcada. Ache essa opção na parte de baixo da imagem.

Essa é a opção que faz com que os comandos **python** e **pip** estejam disponíveis no **cmd**, e deve ser marcada.

Depois de desinstalar e reinstalar, feche o cmd e abra um novo, para ele carregar os novos comandos.

Referências

Múltiplos autores. **Códigos de status de respostas HTTP.** Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status>. Acesso em: 03 ago. 2021.

Múltiplos autores. **Métodos de requisição HTTP.** Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Methods>. Acesso em: 03 ago. 2021.

URL. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2021. Disponível em: <https://pt.wikipedia.org/w/index.php?title=URL&oldid=61899626>. Acesso em: 03 ago. 2021.