**Entendendo endpoints e boas práticas no uso de métodos HTTP**

O modelo de arquitetura REST incorpora vários princípios para garantir o padrão na construção de aplicações WEB. Sem dúvida um dos princípios mais importantes é a utilização dos verbos http, alinhada com as boas práticas de programação. Mas por que isso é tão importante?

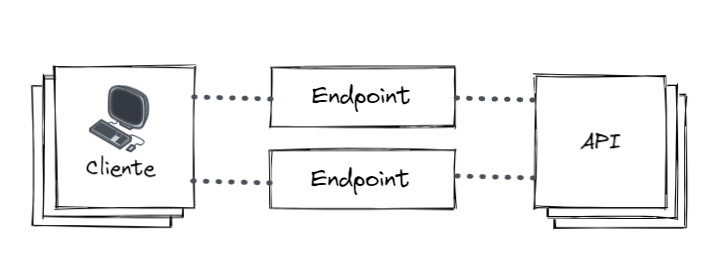
Imagine o cenário em que você precisa de uma requisição para listar **o nome** de todos os usuários de sua plataforma, mas acidentalmente você utilizou um método errado e acabou deletando o banco de dados de sua empresa? Ou mesmo você precisa listar o nome de usuários mas lista todas as compras do e-commerce? Desesperador, não é?!

Para evitar esse tipo de intercorrência, um conjunto de boas práticas foi idealizado para garantir a consistência e maior legibilidade do código. Nesse sentido, os **endpoints** e os métodos http apresentam importante papel nesse processo. Vamos entender um pouco mais?

**Endpoints**

Em primeiro lugar, vamos relembrar o que é um *endpoint*. Para aplicações web, o *endpoint* nada mais é do que uma referência que o cliente usa para acessar os serviços de sua aplicação. Essa referência é uma URL que faz a comunicação entre cliente e API; e a URL que é a referência dos serviços, é o **endpoint**

Para os protocolos de comunicação entre cliente e servidor, o **endpoint** relacionado aos terminais que fazem a conexão entre API e cliente ou servidor externo. Isso significa que o seu serviço pode ser acessado por essa referência, que é uma URL. Como podemos representar na imagem abaixo:



Nesse sentido, o **endpoint** é uma forma de identificar recursos na sua API, por exemplo:

* <http://alura.nossalivraria.com.br/clientes>; → para manipular a lista de clientes;
* <http://alura.nossalivraria.com.br/produtos>; → para manipular a lista de produtos.

Perceba que o exemplo acima não contém referências aos verbos http, ao recurso acessado ou mesmo sobre o formato de apresentação do arquivo (json, bson, xml). Outra boa prática é não fazer alterações nos nomes dos recursos, pois isso pode confundir durante o acesso.

Ok, tudo certo até então. No entanto, sabemos que precisamos dos **verbos http** para realizarmos as requisições e como fazemos isso?

**O uso de métodos HTTP**

Neste ponto já sabemos que há várias opções para gerenciar os dados durante a manipulação de recursos de uma API. Essas **opções** estão relacionadas aos verbos HTTP que, identificados pelo endpoint, podem criar, excluir, atualizar ou mesmo consultar os dados. Dessa forma, além de identificar os recursos desejados, é necessário enviar a informação sobre o tipo de manipulação que objetiva realizar a partir do recurso.

No artigo **”REST: Princípios e boas práticas”,** o instrutor considera uma lista dos principais métodos protocolo HTTP e o cenário de utilização de cada um deles, vamos conferir?

| **Método** | **Descrição** |
| --- | --- |
| GET | Obter os dados de um recurso. |
| POST | Criar um novo recurso. |
| PUT | Substituir os dados de um determinado recurso. |
| PATCH | Atualizar parcialmente um determinado recurso. |
| DELETE | Excluir um determinado recurso. |
| HEAD | Similar ao GET, mas utilizado apenas para se obter os cabeçalhos de resposta, sem os dados em si. |
| OPTIONS | Obter quais manipulações podem ser realizadas em um determinado recurso. |

Em aplicações REST é comum utilizar apenas os métodos GET, POST, PUT e DELETE. No entanto, se houver a necessidade de trabalhar com outros métodos você poderá utilizá-los. No artigo, o instrutor ainda listou alguns padrões dos métodos HTTP em uma aplicação REST, e o exemplo é um recurso chamado Cliente. Vejamos:

| **Método** | **URI** | **Utilização** |
| --- | --- | --- |
| GET | /clientes | Recuperar os dados de todos os clientes. |
| GET | /clientes/id | Recuperar os dados de um determinado cliente. |
| POST | /clientes | Criar um novo cliente. |
| PUT | /clientes/id | Atualizar os dados de um determinado cliente. |
| DELETE | /clientes/id | Excluir um determinado cliente. |

Como boa prática, em relação aos métodos do protocolo HTTP, evite utilizar apenas o método POST nas requisições que alteram o estado no servidor, tais como: cadastro, alteração e exclusão, e principalmente, evite utilizar o método GET nesses tipos de operações, pois é comum os navegadores fazerem cache de requisições GET, mas disparando antes mesmo do usuário clicar em botões e links em uma página HTML.

* Para conferir o artigo na íntegra, confira: [**REST: Princípios e boas práticas**](https://blog.caelum.com.br/rest-principios-e-boas-praticas)
* Para conhecer sobre HTTP: [**Curso de HTTP: Entendendo a web por baixo dos panos**](https://cursos.alura.com.br/course/http-fundamentos)
* Testes com Postman: [**Introdução ao Postman**](https://cursos.alura.com.br/extra/alura-mais/introducao-ao-postman-c182)