# O Que É o REST?

Paulino Ng

2019-09-04

## Aprenda sobre como projetar serviços da Web usando o paradigma REST

**REST**, ou *REpresentational State Transfer*, é um estilo de arquitetura para fornecer padrões entre sistemas de computador na Web, facilitando a comunicação entre os sistemas. Os sistemas compatíveis com **REST**, geralmente chamados de sistemas *RESTful*, são caracterizados pela forma como são *sem estado* e isolam as preocupações do cliente e do servidor. Explicaremos o que esses termos significam e por que eles são características benéficas para serviços na Web.

## Separação do Cliente e do Servidor

No estilo arquitetural **REST**, a implementação do cliente e a implementação do servidor podem ser feitas independentemente, sem que cada um tenha conhecimento do outro. Isso significa que o código no lado do cliente pode ser alterado a qualquer momento sem afetar a operação do servidor e o código no lado do servidor pode ser alterado sem afetar a operação do cliente.

Desde que cada lado saiba qual formato de mensagens enviar para o outro, eles podem ser mantidos modulares e separados. Separando as preocupações da interface do usuário das preocupações com armazenamento de dados, melhoramos a flexibilidade da interface entre as plataformas e aprimoramos a escalabilidade, simplificando os componentes do servidor. Além disso, a separação permite que cada componente evolua independentemente.

Ao usar uma interface **REST**, diferentes clientes chegam nos mesmos pontos extremos **REST**, executam as mesmas ações e recebem as mesmas respostas.

## Ausência de Estado

Sistemas que seguem o paradigma **REST** são sem estado, o que significa que o servidor não precisa saber nada sobre o estado do cliente e vice-versa. Dessa forma, tanto o servidor quanto o cliente podem entender qualquer mensagem recebida, mesmo sem ver as mensagens anteriores. Esta restrição de ausência de estado é imposta através do uso de recursos, em vez de comandos. Recursos são os substantivos da Web - eles descrevem qualquer objeto, documento ou coisa que você pode precisar armazenar ou enviar para outros serviços.

Como os sistemas **REST** interagem através de operações padrões nos recursos, eles não dependem da implementação de interfaces.

Essas restrições ajudam os aplicativos **RESTful** a obterem confiabilidade, desempenho rápido e escalabilidade, como componentes que podem ser gerenciados, atualizados e reutilizados sem afetar o sistema como um todo, mesmo durante a operação do sistema.

Agora, vamos explorar como a comunicação entre o cliente e o servidor realmente acontece quando estamos implementando uma interface **RESTful**.

## Comunicação entre o Cliente e o Servidor

Na arquitetura **REST**, os clientes enviam solicitações para obter ou modificar recursos e os servidores enviam respostas a essas solicitações. Vamos dar uma olhada nas formas padrões de fazer solicitações e enviar respostas.

### Envio de Requisições

O **REST** exige que um cliente faça uma solicitação ao servidor para recuperar ou modificar dados no servidor. Uma solicitação geralmente consiste em:

- um verbo HTTP, que define que tipo de operação realizar
- um cabeçalho, que permite ao cliente passar informações sobre a solicitação
- um caminho para um recurso
- um corpo de mensagem opcional contendo dados

### Verbos do HTTP

Existem 4 verbos básicos de HTTP que usamos em pedidos para interagir com recursos em um sistema **REST**:

- **GET** - recuperar um recurso específico (por id) ou uma coleção de recursos
- **POST** - criar um novo recurso
- **PUT** - atualizar um recurso específico (por id)
- **DELETE** - remover um recurso específico (por id)

### Parâmetros no Header e no Accept

No cabeçalho da solicitação, o cliente envia o tipo de conteúdo que pode receber do servidor. Isso é chamado de campo de `Accept` e garante que o servidor não envie dados que não possam ser compreendidos ou processados pelo cliente. As opções para `content-type` são *MIME Types* (*Multipurpose Internet Mail Extensions*), sobre as quais você pode se aprofundar em MDN Web Docs. Os *MIME Types*, usados para especificar os tipos de conteúdo no campo `Accept`, consistem em um tipo e um subtipo. Eles são separados por uma barra inclinada (/).

Por exemplo, um arquivo de texto contendo HTML seria especificado com o tipo `text/html`. Se esse arquivo de texto contivesse *CSS*, ele seria especificado como `text/css`. Um arquivo de texto genérico seria denotado como `text/plain`. Este valor padrão, `text/plain`, não é um faz-tudo, substituto para qualquer coisa. Se um cliente está esperando `text/css` e recebe `text/plain`, ele não será capaz de reconhecer o conteúdo e renderizar adequadamente uma página HTML.

Outros tipos e subtipos comumente usados são:

- **imagem** — `image/png`, `image/jpeg`, `image/gif`
- **áudio** — `audio/wav`, `image/mpeg`
- **vídeo** — `video/mp4`, `video/ogg`
- **aplicação** — `application/json`, `application/pdf`, `application/xml`, `application/octet-stream`

Por exemplo, um cliente acessando um recurso com id 23 em um recurso `articles` em um servidor pode enviar um pedido **GET** assim:

```
GET /articles/23
Accept: text/html, application/xhtml
```

O campo de cabeçalho `Accept` neste caso está dizendo que o cliente aceitará o conteúdo em `text/html` ou `application/xhtml`.

### PATHs

As solicitações devem conter um caminho (*path*) para um recurso no qual a operação deve ser executada. Nas APIs **RESTful**, os caminhos devem ser projetados para ajudar o cliente a saber o que está acontecendo.

Convencionalmente, a primeira parte do caminho deve ser a forma plural do recurso. Isso mantém os caminhos aninhados simples de ler e fáceis de entender.

Um caminho como `fashionboutique.com/customers/223/orders/12` é claro no que ele indica, mesmo que você nunca tenha visto esse caminho específico antes, porque é hierárquico e descritivo. Podemos ver que estamos acessando o pedido com id 12 para o cliente com id 223. Os caminhos devem conter as informações necessárias para localizar um recurso com o grau de especificidade necessário. Ao se referir a uma lista ou coleção de recursos, não é necessário adicionar um ID a uma solicitação `POST`. Por exemplo, para um caminho `fashionboutique.com/customers` não precisa de um identificador extra, pois o servidor gerará um id para o novo objeto.

Se estamos tentando acessar um único recurso, precisaríamos acrescentar um ID ao caminho. Por exemplo: `GET fashionboutique.com/customers/:id` - recupera o item no recurso `customers` com o id especificado. `DELETE fashionboutique.com/customers/:id` - exclui o item no recurso `customers` com o id especificado.

### Envio de Respostas

### Tipos de Conteúdo

Nos casos em que o servidor está enviando dados para o cliente, o servidor deve incluir um `Content-Type` no cabeçalho da resposta. Esse campo de cabeçalho `Content-Type` alerta o cliente para o tipo de dados que está enviando no corpo da resposta. Esses tipos de conteúdo são tipos de MIME, assim como estão no campo `Accept` do cabeçalho da solicitação. O `Content-Type` que o servidor envia de volta na resposta deve ser de uma das opções que o cliente especificou no campo `Accept` da solicitação.

Por exemplo, quando um cliente está acessando um recurso com o id 23 em um recurso `articles` com essa requisição `GET`:

```
GET /articles/23 HTTP/1.1
Accept: text/html, application/xhtml
```

O servidor pode enviar de volta o conteúdo com o cabeçalho de resposta :

```
HTTP/1.1 200 (OK)
Content-Type: text/html
```

Isso significa que o conteúdo solicitado está retornando no corpo da resposta com um `Content-Type` de `text/html`, que o cliente disse que aceitaria.

### Códigos de Resposta

As respostas do servidor contém códigos de *status* para alertar o cliente com informações sobre o sucesso da operação. Como desenvolvedor, você não precisa conhecer todos os códigos de status (há muitos deles), mas você deve conhecer os mais comuns e como eles são usados:

| Código de Status | Significado |
|---|---|
| `200 (OK)` | Esta é a resposta padrão para requisições de HTTP bem-sucedidas. |
| `201 (CREATED)` | Esta é a resposta padrão para uma requisição de HTTP que resultou em um item sendo criado com sucesso. |
| `204 (NO CONTENT)` | Esta é a resposta padrão para requisições de HTTP bem-sucedidas, em que nada está sendo retornado no corpo da resposta. |
| `400 (BAD REQUEST)` | A requisição não pode ser processada devido à sintaxe de requisição incorreta, tamanho excessivo ou outro erro do cliente. |
| `403 (FORBIDDEN)` | O cliente não tem permissão para acessar este recurso. |
| `404 (NOT FOUND)` | O recurso não pôde ser encontrado neste momento. É possível que tenha sido apagado ou não exista. |
| `500 (INTERNAL SERVER ERROR)` | A resposta genérica para uma falha inesperada se não houver mais informações específicas disponíveis. |

Para cada verbo HTTP, há códigos de *status* esperados que um servidor deve retornar após o sucesso:

- **GET** - retornar `200 (OK)`
- **POST** - retornar `201 (CREATED)`
- **PUT** - retornar `200 (OK)`
- **DELETE** - retornar `204 (NO CONTENT)` Se a operação falhar, retorne o código de status mais específico possível correspondente ao problema encontrado.

### Exemplos de Requisições e Respostas

Digamos que temos um aplicativo que permite visualizar, criar, editar e excluir clientes e pedidos de uma pequena loja de roupas hospedada no `fashionboutique.com`. Poderíamos criar uma API HTTP que permitisse a um cliente executar estas funções:

Se quiséssemos visualizar todos os clientes (*customers*), a requisição ficaria assim:

```
GET http://fashionboutique.com/customers
Accept: application/json
```

Um cabeçalho de resposta possível seria parecido com:

```
Status Code: 200 (OK)
Content-Type: application/json
```

seguido dos dados requisitados dos clientes no formato `application/json`.

Crie um novo cliente postando os dados:

```
POST http://fashionboutique.com/customers
Body:
{
  "customer": {
    "name" = "Walter Carvalho"
    "email" = "walter.carvalho@loucademia.org"
  }
}
```

O servidor, em seguida, gera um ID para esse objeto e retorna-o de volta para o cliente, com um cabeçalho do tipo:

```
201 (CREATED)
Content-Type: application/json
```

Para visualizar um único cliente usamos o `GET` especificando o id do cliente:

```
GET http://fashionboutique.com/customers/123
Accept: application/json
```

Um cabeçalho de resposta possível seria semelhante a:

```
Status Code: 200 (OK)
Content-Type: application/json
```

seguido pelos dados para o recurso do `customer` com id 23 no formato `application/json`.