O QUE É REST E QUAIS OS MÉTODOS/VERBOS HTTP

Dupla: Jonas Lopes e Mayke Bezerra

Professor: Júlio Martins

Falando de serviços WEB, ao desenvolver uma aplicação/serviço web, que disponibilizará dados para algum cliente(apps, webapps ou outro sistema), precisamos prestar bastante atenção no modelo de interface que utilizaremos.

Isso é importante para que os desenvolvedores que utilizarem seu serviço web tenham fácil uma melhor compreensão do que cada ***endpoint*** faz, de acordo com os verbos HTTP utilizados para a requisição. A maneira mais comum para modelar uma interface como a que vemos nos dias de hoje é a arquitetura ***REST***.

Para entender o que é REST é necessário antes, entender algumas definições importantes.

Para entender o REST e como ele funciona é necessário primeiramente entender o que é uma API. Após isso é preciso possível de onde se originou o REST.

O que é API?

Interface de Programação de Aplicação, ou do inglês **Application Programming Interface**, é um conjunto de padrões e rotinas estabelecidos por um software para a que seja possível a utilização de suas funcionalidades por aplicativos que não pretendem envolver-se em detalhes da implementação do software, mas apenas usar seus serviços. (fonte Wikipedia)

No caso desse texto estamos nos referindo especificamente sobre as APIs de serviço web, ou web services APIs, que são interfaces de programação para troca de dados via web, utilizando protocolo HTTP.

Isso é feito através de endpoints. Endpoints no entanto são os endereços web que executam as ações e são acessados diretamente pelos clientes.

O que é REST?

Representational state transfer (REST) or RESTful web services são uma forma de permitir a interoperabilidade entre sistemas de computadores na Internet. Os serviços da Web compatíveis com REST permitem que os sistemas solicitantes acessem e manipulem representações textuais de recursos da Web usando um conjunto uniforme e predefinido de operações sem estado (**stateless**).

REST é um dos modelos de arquitetura que foi descrito por Roy Fielding, um dos principais criadores do protocolo HTTP, em sua tese de doutorado e que foi adotado como o modelo a ser utilizado na evolução da arquitetura do protocolo HTTP.

REST por fim pode ser considerado como um conjunto de princípios, que quando aplicados de maneira correta em uma aplicação, a beneficia com a arquitetura e padrões da própria Web.

Quando a arquitetura REST é aplicada à APIs de serviços web, chamamos de RESTful APIs.

Verbos HTTP

O protocolo HTTP define alguns métodos as vezes chamados de verbos para indicar a ação desejada que será realizada em um recurso identificado. O que este recurso representa podem ser dados pré-existentes ou dados gerados dinamicamente, isso depende da implementação do servidor.

Em resumo, os verbos HTTP são os métodos de requisição que utilizamos para acessar os endpoints de uma RESTful API.

Uma das primeiras etapas ao se iniciar um projeto de API é planejar os endpoints que existirão para o acesso aos dados e para as ações específicas.

Um exemplo de endpoints pode ser visto na figura a seguir:

Endpoint	Método	Ação
/users	GET	Retorna a lista de usuários
/users	POST	Insere um novo usuário
/users/{id}	GET	Retorna o usuário com id = {id}
/users/{id}	PUT	Substitui os dados do usuário com id = {id}
/users/{id}	PATCH	Altera items dos dados do usuário com id = {id}
/users/{id}	DELETE	Remove o usuário com id = {id}

Vendo esse exemplo fica bem explícito o que cada verbo HTTP significa. É algo intuitivo ao nome.

Definindo melhor cada um desses verbos temos:

POST

O verbo POST é utilizado principalmente para criar novos recursos/dados. Na criação bem-sucedida é retornado o status HTTP 201.

Ele não considerado um método seguro, pois altera o estado do recurso no servidor. Ele também não é idempotente, o que significa que se ele for executado duas vezes de forma idêntica serão criados dois itens diferentes com o mesmo conjunto de dados.

GET

O método HTTP GET é usado para ler ou recuperar(get) uma representação de um recurso. Caso a requisição seja bem-sucedida, é retornada uma representação em JSON e um código de resposta HTTP de 200 (OK). Em caso de erro, ele geralmente retorna um 404 (NOT FOUND) ou 400 (BAD REQUEST).

De acordo com o design da especificação HTTP, requisições GET são amplamente usadas apenas para ler dados, mas jamais para alterá-los. Portanto, quando usados dessa forma, são considerados seguros.

Além disso, GET é idempotente, o que significa dizer que fazer várias solicitações idênticas acaba tendo o mesmo resultado de uma única solicitação.

PUT

PUT é mais utilizado para atualizar alguns recursos. Com o corpo da requisição contendo a representação recém-atualizada do recurso original.

Na atualização bem-sucedida, retorna 200 (ou 204 se não retornar qualquer conteúdo no corpo). Retornar os dados do recurso no body é opcional e fazer isso pode causar maior consumo de banda.

PUT não é uma operação segura, pois modifica estado no servidor, mas é idempotente. Em outras palavras, se você atualizar um recurso usando PUT e, em seguida, fazer essa mesma chamada novamente, o recurso ainda estará lá e ainda terá o mesmo estado.

PATCH

PATCH é usado para modificar um recurso parcialmente. A requisição só precisa conter as alterações específicas para o recurso, não o recurso completo.

Se parece com o verbo PUT, mas o corpo contém um conjunto de instruções descrevendo como um recurso no servidor deve ser modificado para produzir uma nova versão.

PATCH não é nem seguro, nem idempotente e, portanto, candidato a ser evitado em detrimento dos outros verbos.

DELETE

DELETE é fácil de entender. Faz jus ao nome. Ele é usado para excluir um recurso identificado por um URI.

Na exclusão bem-sucedida, devolve o status HTTP 200 (OK) ou o status HTTP 204 (NO CONTENT) sem corpo de resposta.

Há uma advertência sobre idempotência no DELETE. Chamar DELETE em um recurso uma segunda vez geralmente retornará um 404 (NOT FOUND) já que ele já foi removido, faz sentido, portanto, não é mais acessível. Isso, por algumas opiniões, faz operações DELETE não mais idempotente, no entanto, o estado final do recurso é o mesmo. Retornar um 404 é aceitável e comunica com precisão o status da chamada. Operações DELETE são idempotentes.