Aula Node

# Arquitetura REST

## O que é ?

***Rest*** significa *Representation State Transfer*, do Inglês, Transferência de Estado Representacional. Trata-se de uma abstração da arquitetura da web. Resumidamente, o REST consiste em princípios/regras que, quando seguidas, permitem a criação de um projeto com interfaces bem definidas. Desta forma, permitindo, por exemplo, que aplicações se comuniquem.

## Definição

A arquitetura REST é baseada em “resource”, em tradução livre, recursos, isso significa que se trata de ***coisas*** ou ***recursos***. Ele é identificado por URI’s (como por exemplo, URL), é possível que múltiplas URI’s “apontem” para o mesmo recurso/API. Além disso, o recurso não é retornado diretamente, mas sim sua **representação**. É possível que haja diferentes representações do mesmo recurso, utilizando formatos diferentes, essas diferenças podem ser baseadas pelo recurso, itens na requisição, etc.

## Tipos de representação

O termo ***representação*** provém de como os dados / recursos são manipulados em uma aplicação REST, eles representam parte, ou o todo, de um estado de recurso (por exemplo, um usuário ou a lista de usuários) e essa representação é transmitida entre o cliente e o servidor. As representação são tipicamente em ***JSON*** ou ***XML***, mas nada impede de ser de um tipo diferente, como ***CSV*** para uma tabela (Common Separated Value, Lista separada baseado em Virgula), ***HTML***, etc.

É importante ressaltar que, apesar de implícito, a representação **não** é o recurso em sí, mas sim sua **representação** (eu sei, meio confuso, imagina como uma cópia “estática” do recurso, qualquer ação feito nela não afetará o recurso original)

## Os 6 princípios

Notas: Não fique muito preso a esses princípios, estou colocando-os aqui apenas para informa-los que eles existem.

* **Uniform Interface**: Em tradução livre, *Interface Uniforme*, define a interface entre cliente e servidores, ele simplifica e separa a arquitetura, o que permite com que cada parte se desenvolva independente. Ela é baseada em 4 regras:
  + **Resource-Based:** Recursos individuais são identificados pela requisição usando URIs, como identificadores de recursos. Os recursos em si são separados das representações enviadas ao cliente. Por exemplo: O servidor não retorna seu banco de dados, mas uma representação em HTML, XML ou JSON que representa parte do banco de dados.
  + **Manipulation of Resources Through Representations:** Em tradução livre, Manipulação dos recursos através de suas representação, é quando o cliente possui informações suficientes, através da representação de um recurso, incluindo metadados, para modificar ou deletar o recurso original, **caso** tenha permissões suficientes para isso.
  + **Self-descriptive Messages:** Cada mensagem / resposta possui informações suficientes que descreve como processar a mensagem. Por exemplo: O tipo de arquivo para se “decodificar”, como html, jpeg, mp4...
  + **Hypermedia as the Engine of Application State (HATEOAS):** Em tradução livre, hipermídia como o estado do mecanismo do aplicativo, é a definição de que o cliente envia o conteúdo / estado através do body, query-string, headers ou da URI da **requisição** (request). Já o servidor, envia o conteúdo / estado através do body, *código* de reposta ou headers da **resposta** (response).
* **Stateless:** Em tradução livre, Sem Estado, apenas diz que o *estado / state* necessário para utilizar a requisição já esta contido na própria requisição, como parte do body, URI, query-string ou headers. Depois que o servidor realizar a devida lógica com a requisição, a mesma é perdida, junto com seu estado. Isso significa que o servidor não precisa se preocupar com o cliente, o cliente já envia tudo que é necessário para ele.
* **Cacheable:** Quando você realiza o chache de alguma coisa, é como você pré-carregasse um arquivo, deixando-o pronto para o envio. No mundo da internet, as respostas podem ou não ser cacheadas. Um cache bem desenvolvido aumenta a escalabilidade e a performance de seu projeto.
* **Client-Server:** O princípio de “Uniform Interface” separa cliente de servidores. Essa separação significa que, por exemplo, os cliente não se preocupam com o banco de dados, o que permanece interno para cada servidor e aumente a *portabilidade* do código do cliente seja melhorado. Os servidores não estão preocupados com a interface do cliente ou o seu estado, o que permite eles serem mais simples e escaláveis. E ambos, servidores e clientes, também podem ser completamente substituídos e desenvolvidos de forma independente, contanto que a interface não seja alterada.
* **Layered System:** Um cliente comum não pode saber se ele está conectado diretamente ao servidor final, ou um intermediário à ele. Servidores intermediários tem capacidade de aumentar a escalabilidade do seu projeto através do controle de diversas camadas do seu projeto.
* **Code on Demand (Opcional):** Em português, Código à Demanda, define que os servidores são capazes de temporariamente estender ou customizar a funcionalidade de um cliente, transferindo lógica para ele que podem ser executado, como o JavaScript.

# Status de Resposta

Dito tudo isso, só precisamos agora aprender sobre os códigos de resposta. Eles não são difíceis e também não há a necessidade de gravar todos, falarei sobre os principais aqui mas deixarei um link com todos.

De maneira geral, os códigos de resposta podem ser organizados entre:

1. Respostas de informação (100-199)
2. Respostas de sucesso (200-299)
3. Redirecionamentos (300-399)
4. Erros do cliente (400-499)
5. Erros do servidor (500-599)

O status mais conhecido é o 404 – O servidor não pode encontrar o recurso solicitado. Ocorre quando o cliente solicita algo que não existe no servidor -, sendo digno de páginas próprias para ele.

Os mais utilizados são: 200, 201, 204, 404 e 405

* 200 (OK): Esta requisição foi bem sucedida. O significado do sucesso varia de acordo com o método HTTP
* 201 (Created): A requisição foi bem sucedida e um novo recurso foi criado como resultado. Esta é uma tipica resposta enviada após uma requisição POST.
* 204 (No content): Não há conteúdo para enviar para esta solicitação, mas os cabeçalhos podem ser úteis. O user-agent pode atualizar seus cabeçalhos em cache para este recurso com os novos.
* 404 (Not Found): O servidor não pode encontrar o recurso solicitado.
* 405 (Method Not Allowed): O método de solicitação é conhecido pelo servidor, mas foi desativado e não pode ser usado. Os dois métodos obrigatórios, GET e HEAD, nunca devem ser desabilitados e não devem retornar este código de erro.

Também existem os status de redirecionamento. Ocorre quando uma URL de um site mudou ou a própria página te redireciona para outra página (Por exemplo: Login realizado, caso autorizado irá para o painel administrativo, se não for, mensagem de erro). Eles podem ser 301 ou 308 para redirecionamento permanentes, 302, 303, 307 para temporários e os redirecionamento especiais, 300 ou 304. Não há necessidade de gravar todos, apenas tenha ciência que eles existem.

# RESTful API ?

Bem, já entendemos o que é uma API, agora, o que é uma ***RESTful API*** ? Esse termo é bem popular no ramo de desenvolvimento web hoje em dia, e apesar de muitas pessoas desenvolverem *RESTful API* no dia a dia, elas tem dificuldade de definir o que elas são.

De uma maneira bem simples, uma ***API*** é considerada ***RESTful*** quando você obedecer tudo o que falamos anteriormente, principalmente às regras da arquitetura REST, essa é principal, simples, não ?

Nota: **Teoricamente**, o último princípio, Code on Demand, é o único dos 6 que caso violado a aplicação não é considerada mais *RESTful*

Leia mais em:

* <https://becode.com.br/o-que-e-api-rest-e-restful/> (Português)

Vídeos auxiliares:

* <https://www.youtube.com/watch?v=S7MduKwvVGk> (Português)
* [h](https://www.restapitutorial.com/lessons/whatisrest.html)t[tps://www.restapitutorial.com/lessons/whatisrest.html](https://www.restapitutorial.com/lessons/whatisrest.html) (Inglês)

Fontes:

<https://becode.com.br/o-que-e-api-rest-e-restful/>

https://www.restapitutorial.com/lessons/whatisrest.html

https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Redirecionamento

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