REST

Princípios e Boas Práticas

REST

Representational State Transfer é um modelo a ser utilizado para se projetar arquiteturas de software distribuído, baseadas em comunicação via rede.

REST 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.

Identificação dos Recursos

Toda aplicação gerencia algumas informações. Uma aplicação de um E-commerce, por exemplo, gerencia seus produtos, clientes, vendas, etc. Essas coisas que uma aplicação gerencia são chamadas de **Recursos** no modelo **REST**.

Um **Recurso** é uma abstração de um tipo de informação que a aplicação gerencia.

No **REST** todo **Recurso** deve possuir uma **identificação única**.

A identificação do recurso deve ser feita utilizando-se o conceito de **URI** (Uniform Resource Identifier), que é um dos padrões utilizados pela Web.

- ✔ http://servicorest.com.br/produtos
- ✔ http://servicorest.com.br/clientes
- ✔ http://servicorest.com.br/clientes/57
- ✔ http://servicorest.com.br/vendas

URIs são a interface de utilização dos serviços e funcionam como um contrato que será utilizado pelos clientes para acessá-los.

Boas Práticas

Utilize URIs legíveis

Utilize nomes legíveis por humanos, que sejam de fácil dedução e que estejam relacionados com o domínio da aplicação.

Utilize o mesmo padrão de URI na identificação dos recursos

Mantenha a consistência na definição das URIs. Crie um padrão de nomenclatura e o siga.

- ✗ http://servicorest.com.br/produto (Singular);
- ✗ http://servicorest.com.br/clientes (Plural);
- ✗ http://servicorest.com.br/processosAdministrativos (Camel Case);
- ✗ http://servicorest.com.br/processos_judidicais (Snake Case).

} EVITE

Evite adicionar na URI a operação a ser realizada no recurso

A manipulação dos recursos deve ser feita utilizando-se os métodos do protocolo HTTP, que será discutido mais adiante.

Evite definir URIs que contenham a operação a ser realizada em um recurso:

- ✗ http://servicorest.com.br/produtos/cadastrar
- ✗ http://servicorest.com.br/clientes/10/excluir
- ✗ http://servicorest.com.br/vendas/34/atualizar

Evite adicionar na URI o formato desejado da representação do recurso

Um serviço REST pode suportar múltiplos formatos para representar seus recursos (XML, JSON e HTML).

O formato desejado por um cliente ao consultar um serviço REST deve ser feita via Content Negotiation.

Evite definir URIs que contenham o formato desejado de um recurso, tais como:

- ✗ http://servicorest.com.br/produtos/xml
- ✗ http://servicorest.com.br/clientes/112?formato=json

Evite alterações nas URIs

A URI é a porta de entrada de um serviço. Quando alterada causa impacto nos clientes que estavam consumindo, pois você alterou a forma de acesso a ele.

Após definir uma URI e disponibilizar a manipulação de um recurso por ela, evite ao máximo sua alteração.

Nos casos mais críticos, no qual realmente uma URI precisará ser alterada, notifique os clientes desse serviço previamente.

Verifique também a possibilidade de se manter a URI antiga, fazendo um redirecionamento para a nova URI.

Métodos HTTP para manipulação dos recursos

Os recursos gerenciados por uma aplicação, e identificados por sua URI, podem ser manipulados de diversas maneiras. É possível criá-los, atualizá-los, excluí-los, dentre outras operações.

Numa requisição **HTTP** para um serviço, além da **URI** que identifica o recurso que ele pretende manipular, é necessário informar o tipo de manipulação a ser realizada.

O tipo de manipulação são os métodos do protocolo **HTTP**.

Principais métodos do protocolo **HTTP** e o cenário de utilização:

Método	**Cenário de utilização**
GET	Obter os dados de um recurso.
POST	Criar um 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.
OPTIONS	Obter quais manipulações podem ser realizadas em um determinado recurso.

Geralmente as aplicações apenas utilizam os métodos GET, POST, PUT e DELETE, mas se fizer sentido em sua aplicação utilizar algum dos outros métodos, não há nenhum problema nisso.

A seguir o padrão de utilização dos métodos HTTP em um serviço REST, que é utilizado pela maioria das aplicações e pode ser considerado uma boa prática.

Como exemplo será utilizado um recurso chamado Cliente.

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.
PATCH	/clientes/id	Atualizar os dados de um determinado cliente.
DELETE	/clientes/id	Excluir um determinado cliente.

Representações dos recursos

Os recursos ficam armazenados no servidor pela aplicação que os gerencia.

Quando são solicitados pelas aplicações clientes, por exemplo em uma requisição do tipo GET, eles não “abandonam” o servidor, como se tivessem sido transferidos para os clientes.

O servidor transfere para a aplicação cliente apenas uma representação do recurso.

Um recurso pode ser representado utilizando-se formatos específicos, tais como XML, JSON, HTML, CSV, dentre outros.

Comunicação Stateless

A Web é o principal sistema que utiliza o modelo REST.

Requisições feitas por um cliente a um serviço REST devem conter todas as informações necessárias para que o servidor as interprete e as execute corretamente.

Clientes não devem depender de dados previamente armazenados no servidor para processar uma requisição.

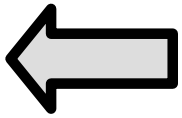
Qualquer informação de estado deve ser mantida pelo cliente e não pelo servidor.

Evite manter dados de autenticação/autorização em sessão

A principal dificuldade em criar um serviço REST totalmente Stateless ocorre quando precisamos lidar com os dados de autenticação/autorização dos clientes. A dificuldade ocorre porque é natural armazenar as informações em sessão, pois é uma solução comum ao se desenvolver uma aplicação Web tradicional.

A principal solução utilizada para resolver esse problema é a utilização de Tokens de acesso, que são gerados pelo serviço REST e devem ser armazenados pelos clientes, via cookies ou HTML 5 Web Storage, devendo também ser enviados pelos clientes a cada nova requisição ao serviço.

Já existem diversas tecnologias e padrões para se trabalhar com Tokens, dentre elas:

- OAUTH
- **JWT (JSON Web Token)** 
- Keycloack

Códigos de Retorno do HTTP

No protocolo HTTP, toda requisição feita por um cliente a um servidor deve resultar em uma resposta, sendo que nela existe um código HTTP, utilizado para informar o resultado da requisição, tenha ela sido processada com sucesso ou não.

Os códigos HTTP são agrupados em classes:

Classe	Semântica
2xx	Indica que a requisição foi processada com sucesso.
3xx	Indica ao cliente uma ação a ser tomada para que a requisição possa ser concluída.
4xx	Indica erro(s) na requisição causado(s) pelo cliente.
5xx	Indica que a requisição não foi concluída devido a erro(s) ocorrido(s) no servidor.

A boa prática consiste em conhecer os principais códigos HTTP e utilizá-los de maneira correta.

Códigos de Retorno do HTTP

Código	Descrição	Quando utilizar
200	OK	Em requisições GET, PUT e DELETE executadas com sucesso.
201	Created	Em requisições POST, quando um novo recurso é criado com sucesso.
206	Partial Content	Em requisições GET que devolvem apenas uma parte do conteúdo de um recurso.
302	Found	Em requisições feitas à URIs antigas, que foram alteradas.
400	Bad Request	Em requisições cujas informações enviadas pelo cliente sejam invalidas.
401	Unauthorized	Em requisições que exigem autenticação, mas seus dados não foram fornecidos.
403	Forbidden	Em requisições que o cliente não tem permissão de acesso ao recurso solicitado.
404	Not Found	Em requisições cuja URI de um determinado recurso seja inválida.
405	Method Not Allowed	Em requisições cujo método HTTP indicado pelo cliente não seja suportado.
406	Not Acceptable	Em requisições cujo formato da representação do recurso requisitado pelo cliente não seja suportado.
415	Unsupported Media Type	Em requisições cujo formato da representação do recurso enviado pelo cliente não seja suportado.
500	Internal Server Error	Em requisições onde um erro tenha ocorrido no servidor.
503	Service Unavailable	Em requisições feitas a um serviço que está fora do ar, para manutenção ou sobrecarga