REST API

Prof. Me. Hélio Esperidião

A origem do termo REST (Rest API)

- Os conceitos do REST foram estabelecidos na tese de doutorado de Roy Fielding nos anos 2000, onde o princípio fundamental é utilizar o protocolo HTTP para a comunicação de dados.
- Ver entrevista: https://www.youtube.com/watch?v=w5j2KwzzB-0
- Site: https://roy.gbiv.com/
- Currículo: https://roy.gbiv.com/vita.html

O que é REST

- REST (Representational State Transfer) é uma arquitetura de software utilizada para projetar sistemas de comunicação em rede, geralmente na construção de APIs (***Application Programming Interfaces***).
- É baseado em princípios como a utilização de URIs (*Uniform Resource Identifiers*) para identificar recursos.
- Utiliza os métodos HTTP (GET, POST, PUT, DELETE) para operar sobre esses recursos.
- A representação dos recursos é feita em formatos como JSON ou XML.
- REST é amplamente adotada na construção de serviços web devido à sua simplicidade e escalabilidade

STATELESS

- Em uma **REST API**, o termo **stateless** significa que cada requisição feita pelo cliente ao servidor deve conter todas as informações necessárias para que o servidor entenda e processe a requisição, sem depender de informações armazenadas em sessões anteriores.
- **Stateless** = sem estado.
- O servidor não armazena o "estado" do cliente entre requisições.
- Cada requisição é independente das outras.

Requisições e comunicações

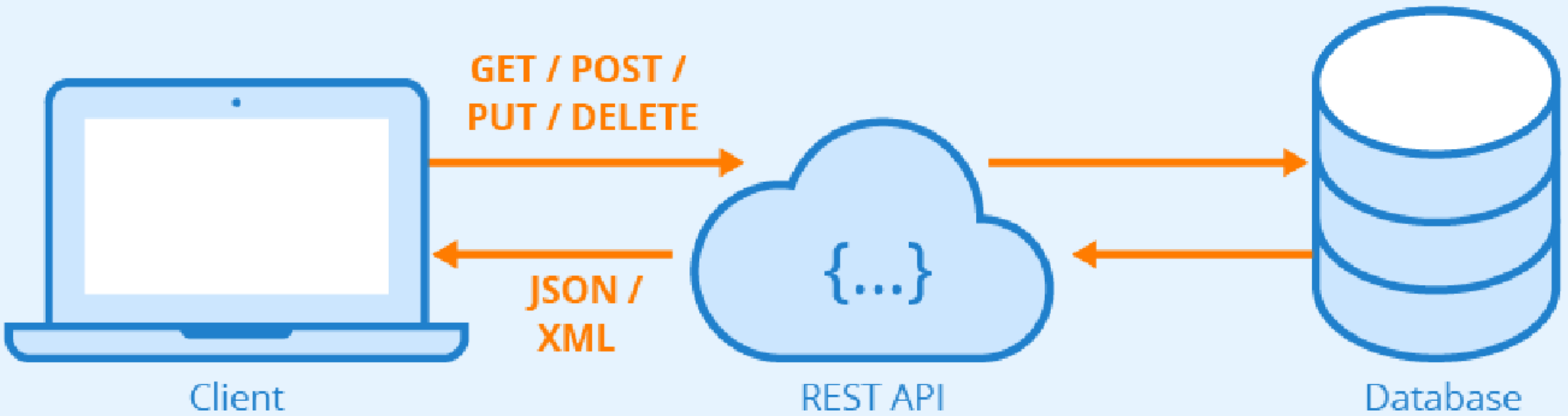
- O REST precisa que um cliente faça uma requisição para o servidor para enviar ou modificar dados. Um requisição consiste em:

- **Método HTTP**: Define que tipo de operação o servidor vai realizar;
- **Header:** deve conter o cabeçalho da requisição que passa informações sobre a requisição;
- **Rota**: para o servidor, como por exemplo https://helioesperidiao/alunos/;
- **Body** : Corpo da requisição sendo esta informação opcional.

Verbos/Métodos HTTP

- Em aplicação REST, os métodos mais utilizados são:
  - **GET**: é o método mais comum, geralmente é usado para solicitar que um servidor envie um recurso;
  - **POST:** foi projetado para enviar dados de **entrada** para o servidor. Na prática, é frequentemente usado para suportar formulários HTML;
  - **PUT:** Atualiza documentos em um servidor;
  - **DELETE** : Apaga dado ou coleção de dados do servidor
  - **PATCH**: Atualizar parcialmente 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.

A Deep Look into RESTful APIs



Códigos de Respostas

- **200 (OK):** requisição atendida com sucesso;
- **201 (CREATED)**: objeto ou recurso criado com sucesso;
- **204 (NO CONTENT):** objeto ou recurso excluído com sucesso;
- **400 (BAD REQUEST):** ocorreu algum erro na requisição
- **404 (NOT FOUND):** rota ou coleção não encontrada;
- **500 (INTERNAL SERVER ERROR):** ocorreu algum erro no servidor

Códigos de Respostas

- **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.
- **429 Too Many Requests**: No caso do serviço ter um limite de requisições que pode ser feita por um cliente, e ele já tiver sido atingido.
- **503 Service Unavailable**: Em requisições feitas a um serviço que esta fora do ar, para manutenção ou sobrecarga.

E qual a diferença entre REST e RESTful?

- **REST** (Representational State Transfer)
  - É um estilo de arquitetura proposto por **Roy Fielding** em sua tese de doutorado em 2000.
  - Ele define um conjunto de princípios que um sistema deve seguir para ser considerado REST:
  - Alguns desses princípios são:
    - Stateless (sem estado)
    - Uso de HTTP e seus métodos corretos (GET, POST, PUT, DELETE, etc)
    - Recursos identificados por rotas
    - Representações em formatos como JSON ou XML
    - **REST é a teoria**.

REST vs RESTfull

- **RESTful**
  - É um adjetivo usado para descrever serviços ou APIs que seguem os princípios REST.
  - Se uma API segue corretamente os princípios do REST, dizemos que ela é uma API RESTful.
  - **Analogia rápida:**
    - REST = Conceito / estilo / teoria
    - RESTful = Programa que está aplicando esse estilo corretamente

Padronização de nomes e métodos:

- Endpoints devem ser compostos unicamente por nomes (SUBSTANTIVOS NO PLURAL), não use verbos;
- Utilize kebab-case para palavras compostas, mas evite;
- Use letras minúsculas;
- Não termine seus endpoints com “/”;
- Use diferentes verbos HTTP para suas operações.
    - Por exemplo, POST é usado para criar um Recurso.

Nomes patronizados

Verbo	Rota
GET	http://helioesperidiao.com/produtos
GET	http://helioesperidiao.com/clientes
GET	http://helioesperidiao.com/clientes/57
GET	http://helioesperidiao.com/vendas

Verbos HTTP e suas rotas REST: Exemplo simples

Verbo	**Ação**	**Exemplo de Rota**	**Significado**
GET	Ler dados	/cargos	Lista todos os cargos
GET	Ler um item específico	/cargos/5	Mostra o cargo com ID 5
POST	Criar novo recurso	/cargos	Cria um novo cargo
PUT	Atualizar por completo o recurso	/cargos/5	Atualiza todos os dados do cargo com ID 5
PATCH	atualizar apenas parte do recurso	/cargos/5	Atualização **parcial** **(**Envia apenas os campos modificados**)**
DELETE	Remover recurso	/cargos/5	Remove o cargo com ID 5

Exemplo com hierarquia

Verbo	Rota	Descrição
GET	/marcas	Lista todas as marcas
GET	/marcas/chevrolet/modelos	Lista modelos da Chevrolet
GET	/marcas/chevrolet/modelos/onix	Detalhes do modelo Onix
GET	/marcas/chevrolet/modelos/onix/anos	Lista anos disponíveis do modelo Onix
GET	/marcas/chevrolet/modelos/onix/anos/2022	Detalhes do Onix 2022

Exemplo de Rota Hierárquica complexa

Verbo	**Rota**	**Descrição**
GET	/turmas/1b/alunos	Lista os alunos da turma 1B
POST	/turmas/1b/alunos	Cadastra novo aluno na turma 1B
GET	/turmas/1b/alunos/45	Detalhes do aluno 45 da turma 1B
DELETE	/turmas/1b/alunos/45	Remove o aluno 45 da turma 1B

ROTAS DE PRODUTOS

Método	Rota	Descrição
GET	/produtos	Lista todos os produtos
GET	/produtos/123	Detalha o produto com ID 123
GET	/produtos/123/avaliacoes	Lista avaliações do produto
GET	/produtos/busca?nome=mouse	Busca produtos pelo nome
GET	/produtos/mais-vendidos	Lista os produtos mais vendidos

ROTAS DE CATEGORIAS

Método	Rota	Descrição
GET	/categorias	Lista todas as categorias
GET	/categorias/eletronicos/produtos	Lista produtos da categoria "eletrônicos"
GET	/categorias/roupas/masculinas	Lista subcategoria "roupas masculinas"

ROTAS DE PEDIDOS

Método	Rota	Descrição
GET	/clientes/42/pedidos	Lista pedidos do cliente 42
GET	/pedidos/202308/itens	Lista itens do pedido
POST	/pedidos	Cria novo pedido
PUT	/pedidos/202308/status	Atualiza status do pedido

Formato Padrão para Filtros em REST

A padronização de filtros em APIs RESTful tem como objetivo tornar as consultas mais previsíveis, reutilizáveis e fáceis de entender.

Ela define como os parâmetros devem ser passados nas URLs para que clientes possam consultar dados com mais precisão.

Filtros

Filtro	URL	Explicação
Por nome	/produtos?nome=camiseta	Filtra produtos com nome "camiseta"
Por faixa de preço	/produtos?precoMin=50&precoMax=200	Produtos entre R$50 e R$200
Por categoria	/produtos?categoria=eletronicos	Apenas produtos da categoria informada
Paginado	/produtos?page=2&limit=10	Página 2 com 10 itens por página
Ordenado	/produtos?sort=preco&order=asc	Ordena pelo preço em ordem crescente
Múltiplos valores	/produtos?ids=10,15,18	Busca os produtos com esses IDs
Por data	/pedidos?dataInicio=2023-01-01&dataFim=2023-01-31	Pedidos feitos em janeiro de 2023

Boas práticas

Use nomes de parâmetros claros e consistentes (ex: precoMin, categoria, sort).

Mantenha os filtros opcionais. Sempre permita paginação (page, limit) para grandes resultados.

Use operadores padrão quando necessário: gte, lte, ne, like, etc.

Exemplo:
/produtos?preco[gte]=100&preco[lte]=500

Tabela com operadores comuns para filtros em APIs

Estruturas Comuns com **Bracket Notation**

Operador	Exemplo	Descrição	Exemplo de Rota
eq	preco[eq]=50	Igual a (ex: preço igual a 50)	/produtos?preco[eq]=50
gt	preco[gt]=50	Maior que (ex: preço > 50)	/produtos?preco[gt]=50
lt	preco[lt]=50	Menor que (ex: preço < 50)	/produtos?preco[lt]=50
gte	preco[gte]=50	Maior ou igual a (ex: preço >= 50)	/produtos?preco[gte]=50
lte	preco[lte]=50	Menor ou igual a (ex: preço <= 50)	/produtos?preco[lte]=50
in	categoria[in]=livros,eletronicos	Pertence a um conjunto de valores (ex: categoria é um dos valores especificados)	/produtos?categoria[in]=livros,eletronicos
nin	categoria[nin]=roupas	Não pertence a um conjunto de valores (ex: categoria não é "roupas")	/produtos?categoria[nin]=roupas
like	nome[like]=celular	Similar a (como um LIKE no SQL, para correspondência parcial de string)	/produtos?nome[like]=celular
ne	preco[ne]=50	Diferente de (ex: preço ≠ 50)	/produtos?preco[ne]=50
exists	nome[exists]=true	Verifica se o campo existe (ex: nome existe)	/produtos?nome[exists]=true