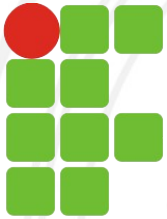


INSTITUTO FEDERAL
PIAUÍ
Campus Parnaíba

API REST

Prof. Msc Denival A. dos Santos

Introdução

- Interface de Programação de Aplicação, cujo acrônimo **API** é um conjunto de rotinas e padrões estabelecidos por um software para a utilização das suas funcionalidades por aplicativos que não pretendem envolver-se em detalhes da implementação do software, mas apenas usar seus serviços.
- **Web services** APIs, que são interfaces de programação para troca de dados via web, utilizando protocolo HTTP. Isso é feito através de **endpoints** que são os endereços web que executam as ações e são acessados diretamente pelos clientes.
- Representational state transfer (REST) or **RESTful** web services são uma forma de proporcionar 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).

Introdução

- Quando a arquitetura REST é aplicada à APIs de serviços web, chamamos de RESTful APIs.
- O protocolo **HTTP** define métodos (às vezes referidos como verbos) para indicar a ação desejada a ser realizada no recurso identificado. O que este recurso representa, se são dados pré-existentes ou dados gerados dinamicamente, depende da implementação do servidor.
- Os verbos HTTP são os métodos de requisição que utilizamos para acessar os endpoints de uma RESTful API.
- Uma das primeiras coisas que deve ser feita ao iniciar um projeto de API é planejar os endpoints que existirão para o acesso aos dados e para as ações específicas.

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}

Introdução

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

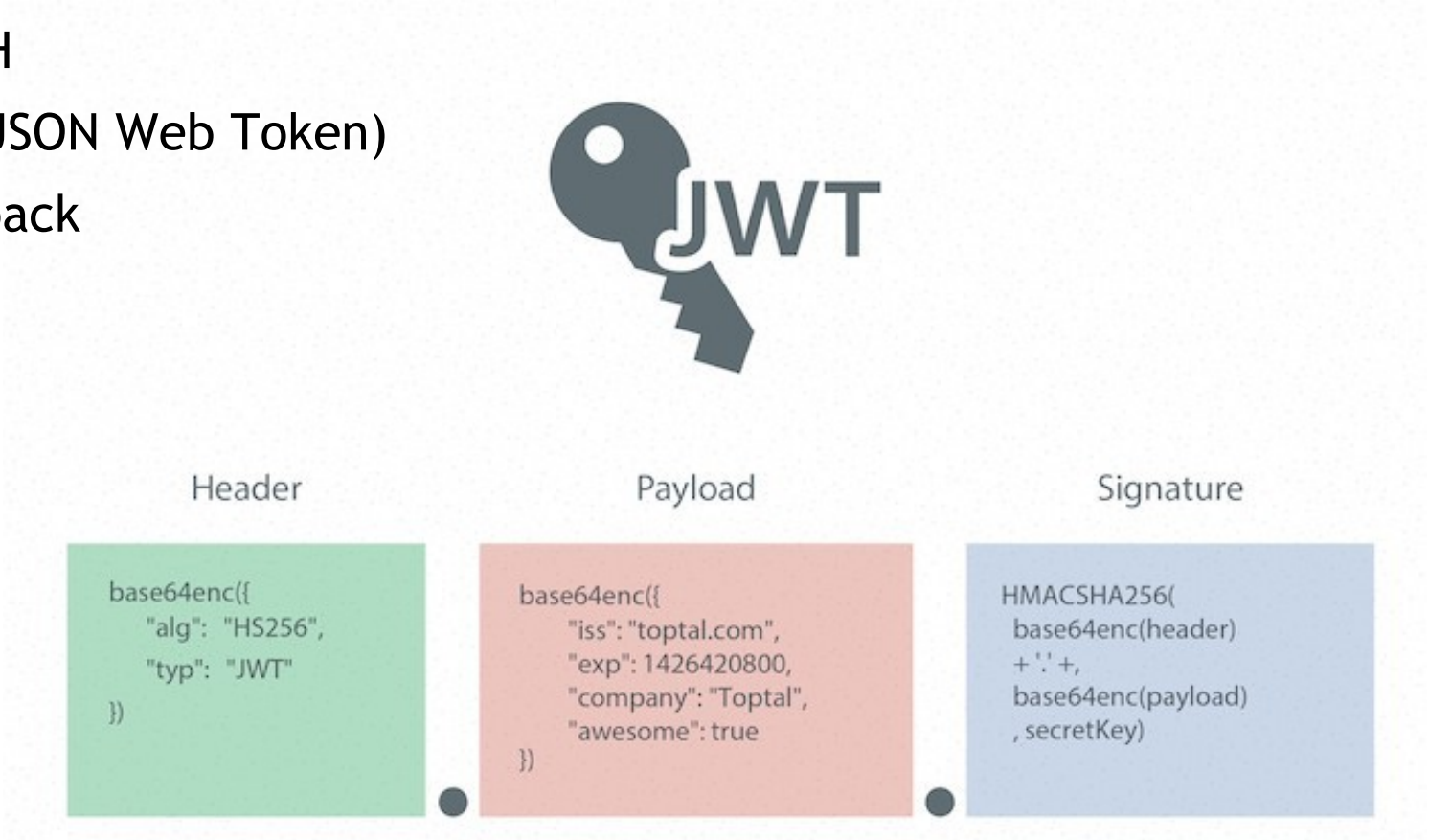
  - http://servicorest.com.br/produtos;
  - http://servicorest.com.br/clientes;
  - http://servicorest.com.br/clientes/57;
  - http://servicorest.com.br/vendas.

Boas práticas

- Utilizar URI legíveis
- Utilize o mesmo padrão de URI na identificação do recurso.
  - Evite:
    - http://servidor.com.br/produto (Singular)
    - http://servidor.com.br/clientes (Plural)
- Evite adicionar na URI a operação a ser realizada no recurso
  - Exemplo:
    - http://servidor.com.br/clientes/10/excluir
- Evite alterações nas URI´s
- Evite adicionar na URI o formato desejado de representação do recurso
  - http://servidor.com.br/clientes/112?formato=json

Boas práticas

- Evite manter dados de autenticação/autorização em sessão, prefira tecnologias e padrões para se trabalhar com Tokens, dentre elas:
  - OAUTH
  - JWT (JSON Web Token)
  - Keycloack



Códigos de resposta (status codes) HTTP

- Nas descrições dos verbos HTTP foram citados diversas vezes os status code do protocolo HTTP.
- Esse é outro item importante para a arquitetura de uma API REST, porque, da mesma maneira que acontece como os verbos HTTP, elas formam um padrão facilmente reconhecido por quem for consumir o web service.
- Os principais códigos utilizados para as respostas de um endpoint são o 200 (OK), o 201 (CREATED), o 204 (NO CONTENT), o 404 (NOT FOUND) e o 400 (BAD REQUEST).
- Os principais códigos utilizados para as respostas de um endpoint são o 200 (OK), o 201 (CREATED), o 204 (NO CONTENT), o 404 (NOT FOUND) e o 400 (BAD REQUEST).
- Os códigos de sucesso tem o padrão 20x, os de redirecionamento 30x, os de erro do cliente 40x e os de erro de servidor 50x.

Representação de recursos

- Um recurso pode ser representado de diversas maneiras, utilizando-se formatos específicos, tais como XML, JSON, HTML, CSV, dentre outros.
- É considerada uma boa prática o suporte a múltiplas representações em um serviço REST.
- Os três principais formatos suportados pela maioria dos serviços REST são:
  - HTML
  - XML
  - JSON

```xml
<cliente>
  <nome>Rodrigo</nome>
  <email>rodrigo@email.com.br</email>
  <sexo>Masculino</sexo>
  <endereco>
    <cidade>Brasilia</cidade>
    <uf>DF</uf>
  </endereco>
</cliente>
```

Métodos do protocolo HTTP

- **POST**
  - O verbo POST é mais frequentemente utilizado para criar novos recursos. Na criação bem-sucedida, retornar o status HTTP 201.
  - Ele não é um método seguro, pois altera o estado do recurso no servidor.
- **GET**
  - O método HTTP GET é usado para ler ou recuperar uma representação de um recurso. Em caso de sucesso, retorna 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 (juntamente com HEAD) são usadas apenas para ler dados e jamais alterá-los. Portanto, quando usados dessa forma, são considerados seguros.

Métodos do protocolo HTTP

- **PUT**
  - O PUT é mais utilizado para substituir (ou atualizar) recursos, executando a requisição para uma URI de recurso conhecido, 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).
  - PUT não é uma operação segura, pois modifica estado no servidor
- **DELETE**
  - DELETE é bastante fácil de entender. 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.

Métodos do protocolo HTTP - Exemplo

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

Roteamento básico

- O Roteamento refere-se à determinação de como um aplicativo responde a uma solicitação do cliente por um endpoint específico, que é uma URI (ou caminho) e um método de solicitação HTTP específico (GET, PUT, etc).
- Cada rota pode ter uma ou mais funções de manipulação, que são executadas quando a rota é correspondida.
- A definição de rotas aceita a seguinte estrutura:

app.METHOD(PATH, HANDLER)

  - app é uma instância do express.
  - METHOD é um método de solicitação HTTP.
  - PATH é um caminho no servidor.
  - HANDLER é a função executada quando a rota é correspondida.

Exemplos de Rotas

Responder a uma solicitação POST na rota raiz (/) com a página inicial do aplicativo:

```
app.post('/', function (req, res) {
  res.send('Got a POST request');
});
```

Responder a uma solicitação PUT para a rota /user:

```
app.put('/user', function (req, res) {
  res.send('Got a PUT request at /user');
});
```

Responder a uma solicitação DELETE para a rota /user:

```
app.delete('/user', function (req, res) {
  res.send('Got a DELETE request at /user');
});
```