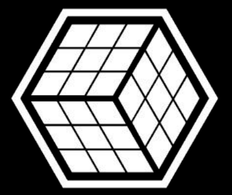
Sua API é RESTful?

Boas práticas de desenvolvimento e aprendizados a partir do livro REST API Design Rulebook



THE DEVELOPER'S CONFERENCE

Kamila Santos

Backend Developer na Ame Digital, Microsoft MVP, co-autora do livro Jornada Java e co-organizadora das comunidades WoMakersCode, DevsJavaGirl e Perifacode.

https://www.linkedin.com/in/kamila-santos-oliveira/

@kamila_code

https://dev.to/kamilahsantos

Afinal o que é REST?

REST -> Representational State Transfer , que **representa um conjunto de princípios de arquitetura para a WEB**

REST API

Interface de programação de aplicativos REST que segue os padrões REST . Na qual um cliente pode acessar seus recursos via **endpoints**

Um pouco de história....

Nos anos 90 para facilitar o compartilhamento , **Tim Berners-Lee** criou a "**World Wide Web**" e criou alguns termos:

URI: Uniforme Resource Indentifier

Atribui a cada recurso web um **endereço exclusivo**.

HTTP: Hypertext Transfer Protocol

linguagem baseada em mensagens que permite a **comunicação entre computadores via internet**.

HTML: Hypertext Mark Up Language (HTML)

representa documentos com **informações e links** para outros documentos relacionados

Por volta de 1993, **Roy Fielding**, co-fundador do **Projeto Apache Http Server** , começou a se preocupar com algumas questões relacionadas a **escalabilidade da WEB**

E concluiu que ela está relacionada com um conjunto de **6 fatores**:

1 - Client Server

A separação das responsabilidades é o ponto focal dessa categoria.

O client e o server podem ser desenvolvidos e implementados de forma independente da stack só precisam seguir uma **interface uniforme**.

2 - Uniform Interface

As interações entre componentes dependem de uma interface uniforme , **se um componente quebra esse contrato a chance de falhas é grande**.

Fielding definiu **4 regras** para essa interface uniforme:

1 - Identificação dos recursos

Cada conceito existente na Web e conhecido como um recurso que tem um identificador único , como por ex: **http://thedevconf.com.br/**, que é a URL raiz de um site específico

2 - Manipulação de recursos

através de representações

Um mesmo recurso pode ser representado de formas diferentes por seus clientes:

Por ex, o mesmo recurso pode ser visualizado pelo front End de uma forma e pelo backend de outra. A representação é um modo de interagir com o recurso mas não o recurso em si.

3 - Mensagens autodescritivas

O estado desejado por um recurso pode ser representado dentro da mensagem de solicitação de um cliente.

O estado atual de um recurso pode ser representado dentro da mensagem de resposta que volta do Backend.

4 - Hypermedia como o "motor"de estado de aplicação

Representação de um recurso incluindo links para recursos relacionados, links esses que são tópicos que tecem a Web em conjunto, permitindo **"navegar"** entre as informações.

Sistema em camadas

Proxys e gateways que geralmente interceptam mensagens por motivos de segurança, load balancer, etc

Cache

A aplicação cacheia as informações por x determinado tempo , esses dados podem ajudar a diminuir **o tempo de latência aumentando a disponibilidade**, dentre outros fatores...

Stateless

O Server não precisa armazenar estados de aplicação de seus clientes. Desse modo, **cada client deve incluir todas as informações necessárias a cada nova interação com o cliente**.

Codigo sobre demanda (opcional)

Permite que web servers **transfiram temporariamente programas executáveis**, como plug-ins e scripts para seus clientes.

Acho que já perceberam de quem são essas regras né?

Em **2000**, essa arquitetura WEB mais tarde virou o que chamamos hoje de **REST**

O modelo de maturidade de Richardson

Seguir **todos os princípios REST é bastante complicado**, muitas vezes não é possível seguir todos eles

Pensando nisso, **Leonard Richardson criou o modelo de maturidade de Richardson**, que divide os elementos REST em três categorias, visando alcançar a **“glória do REST”**

As categorias são:

Recursos (resources)

Verbos http (http verbs)

Controles de hypermedia (hypermedia controls).

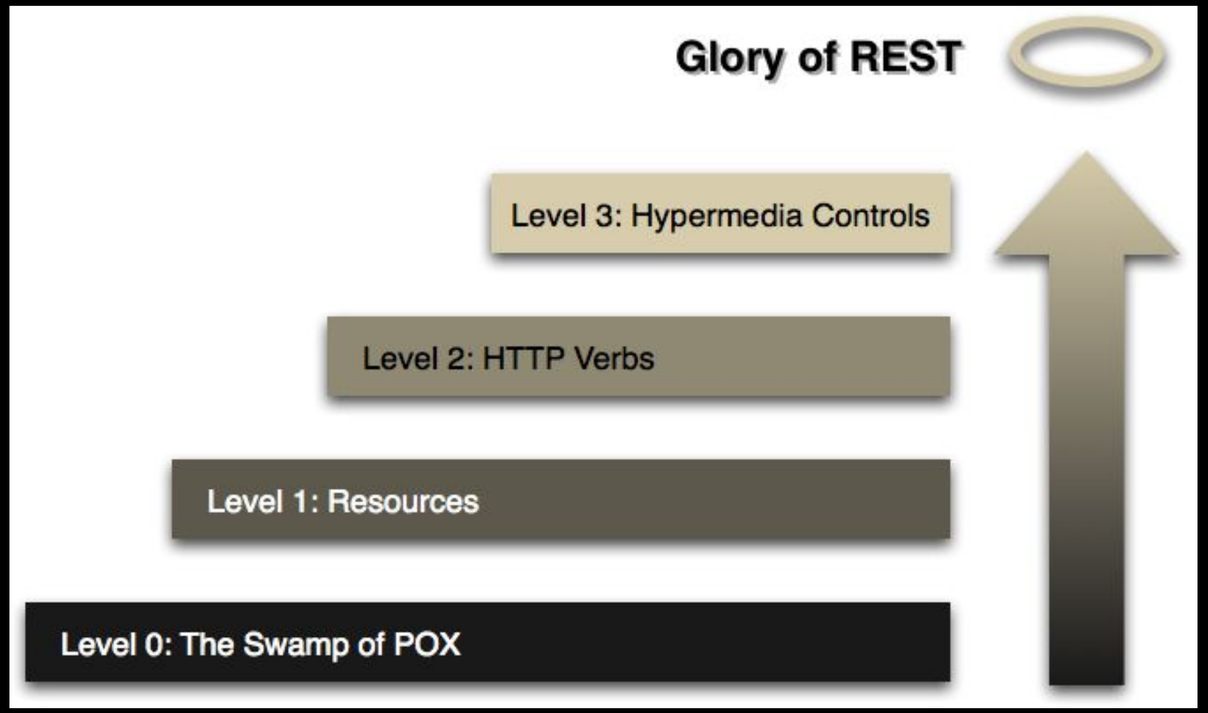


imagem retirada de:

https://martinfowler.com/articles/richardsonMaturityModel.html

Nível 0 - POX

Nesse nível **quase não há padrão**, **nomes de recursos não são bem definidos** e são usados somente para chamar mecanismos baseados em **Remote Procedure Invocation**..

RPI

Aplica o princípio do encapsulamento para integração de aplicações. Caso um aplicativo precise de informações de outro , poderá solicitar diretamente a ele,

RPI

Se um aplicativo necessita alterar os dados de outro, ele o fará realizando uma chamada para o outro aplicativo

RPI

Cada aplicação pode manter a integridade dos dados que possui. Além disso, cada aplicação pode alterar seus dados internos sem que todos os outros aplicativos sejam afetados.

Nível 0 - POX

A troca de mensagens nesse nível 0 pode ser serializada em formatos como XML, texto e JSON

Nível 0 - POX

Nesse nível **geralmente só são utilizados os métodos GET E POST e as URIs não seguem os padrões REST**.

Nível 0 - POX

GET /getSpeaker/6

GET /saveSpeaker

POST changeSpeaker/6

Nível 1 - Resources

Neste nível, usamos recursos como uma maneira de **modelar APIs**. Precisamos saber identificar visualmente o que cada recurso representa **utilizando um substantivo, preferencialmente no plural**.

Nível 1 - Resources

GET /speakers/6

POST /speakers

PUT /speakers/6

DELETE /speakers/6

NÍVEL 2 – VERBOS HTTP

Nesse nível os **verbos http são utilizados para seu real propósito e devemos ter os status corretos para cada ação**, por ex:
para um POST executado com sucesso esperamos um status 201.

NÍVEL 3 – HATEOAS

Foco principal a representação **hypermedia**, possibilitando que um documento descreva seu estado atual, e o seu relacionamento com outros elementos ou futuros estados .**Hypermedia pode ser definido como a capacidade de um recurso se relacionar com os demais em uma coleção**.

NÍVEL 3 – HATEOAS

```
GET /speaker/6
{
    "name": "KAMILA SANTOS",
    "links": [ {
        "rel": "self",
        "href": "http://localhost:8080/speaker/6"
    }, {
        "rel" : "delete",
        "href": "http://localhost:8080/speaker/6"
    ]
}
```

Significado de cada nível

O **Nível 1** lida com a complexidade utilizando a abordagem dividir e conquistar, dividindo um grande terminal de serviço em vários recursos.

Significado de cada nível

O **nível 2** traz um conjunto padrão de verbos para que consigamos lidar com ações semelhantes da mesma maneira, removendo variações desnecessárias.

Significado de cada nível

O **Nível 3** trata da descoberta, fornecendo uma forma de tornar um protocolo mais autodocumentado.

Agora vamos começar as regras:

Sobre o formato das URIs :

elas seguem a **RFC 3986** que define a sintaxe genérica das URIs

Por exemplo:

[schema]://[autoridade]/path[?query params][& outros filtros]

Regra: Separador "/" deve ser utilizado para indicar uma relação hierárquica

Por ex:

https://api.thedevconf/speakers/kamila

Regra: Não incluir "/" como último caractere

Incluir uma "/" fixa no fim da URI não adiciona nenhum valor semântico e pode causar confusão.

Por ex:

https://api.thedevconf/speakers/

https://api.thedevconf/speakers

são a mesma coisa

Regra: "-" podem ser utilizados para melhorar a legibilidade das URIs

Para melhorar a legibilidade de URIs onde iria um espaço pode utilizar um "_"

Por ex:

https://api.thedevconf/speakers/kamila-santos

Regra: Não utilize "_" nas URIs pois passam a impressão de que é algo clicável.

Regra: Utilize letras minúsculas em URIs

Regra: Extensões de arquivos não devem ser incluídas em URIs

Por ex:

https://api.school.restapi.org/students/avatar/kamila.jpg

https://api.school.restapi.org/students/avatar/kamila

As informações referentes ao Media Type devem ser informadas no Header

Regra: Utilize nomes de subdomínios consistentes

O domínio e os primeiros nomes de subdomínios devem identificar o proprietário do serviço. O nome completo de um domínio de uma API deve utilizar o nome API em parte dele

Regras: o portal de Devs da API também deve ter um subdomínio consistente.

Muitas APIs possuem um "developer portal" que deve possuir um nome semelhante a:

https://developer.apixyz.com

Arquétipos de recursos

Auxiliam na comunicação e funcionamento das estruturas existentes numa API REST

Documento

É a representação única de um conceito, geralmente um registro no BD.

Ex:

http://api.events.com/programming/backend

Collection

É um diretório de recursos gerenciado por servidor , escolhem o que conter em cada diretório e quais serão as URIs para cada novo recurso.

Ex:

http://api.events.com/speakers/kamila/presentations

Store

É um repositório de recursos gerenciado pelo client, ele pode inserir e retirar recursos dele mas não há a criação de novas URIs.

Controller

Modela o que é referente ao processo de funcionamento da API , com parâmetros de busca, retornos, requests, responses...

Os nomes dos controllers geralmente aparecem como última parte em um caminho URI, não permitindo outros recursos "filhos" a partir dele

Ex:

POST reminders/12345/resend

Design do Path das URIs

Regra: Nomes no singular devem ser utilizados para representar nomes de documentos

A URI que tem por objetivo representar um único documento deve ser nomeada no singular.

Ex: http://api.events.com/speakers/kamila

Regra: Nomes no plural devem ser utilizados para nomes de collection

Regra: Nomes no plural devem ser utilizados para nomes de stores

Regra: Um verbo ou uma frase verbal deve ser utilizada para nomes de controllers

Regra: Variáveis do path podem ser substituídas por valores baseados em identidade

Algumas dessas variáveis não são fixas, mas sim preenchidas conforme a informação solicitada/retornada.

Por ex: Ex:

http://api.events.com/speakers/[speakerID]/presentations/[presentationId]

Regra: Operações CRUD não devem ser informadas como parte da URI

Design de queries em URIs

Regras: O componente de query da URI pode ser utilizado para filtrar collections ou stores

Por ex:

GET/ /speakers?stack=java

Request Methods

Regra: GET deve ser utilizado somente para buscar a representação do estado atual de um recurso

Regra: HEAD utilizada para retornar response headers

o HEAD busca a mesma coisa que o GET
mas só retorna os headers o response bidy vem vazio

Regra: PUT deve ser utilizado para atualizar recursos mutáveis

Regra: POST deve ser utilizado para criar novos recursos em collections

Por ex:

POST /reminders/1234/resend

Regra: DELETE deve ser utilizado para remover um recurso

Após um recurso ser excluído , se for solicitado novamente o recurso via GET ou HEAD deve ser retornado 404

REGRA: OPTIONS deve ser utilizado para recuperar as operações permitidas para aquele recurso

REGRA: 200 ok deve ser utilizado para indicar casos gerais de sucesso e diferente do 204 , deve retornar um response body

REGRA: 201 created deve ser utilizado para indicar que algum recurso foi criado com sucesso e também deve retornar um response body com a resposta dessa criação

REGRA: 202 accepted deve ser utilizado nos casos de ações assíncronas que começaram a ser processadas

REGRA: 204 no content deve ser utilizado quando a response deve ser vazia

REGRA: 301 moved permanently deve ser utilizado para recursos que foram alocados em novos endereços

Em casos que o recurso foi definido em uma nova URI a API deve especificar no header qual a nova localização do recurso

REGRA: 303 see other, deve ser utilizado nos casos em que um controller terminou seu trabalho mas ao invés de retornar um response body , responde com outra URI.

URI esta que pode conter uma mensagem de status temporária ou para outros recursos.

REGRA: 304 not modified deve ser utilizado quando se deseja mostrar ao client que ele já está com a versão mais recente , ao contrário do 204 esse retorna um response body

REGRA: 401 unauthorized deve ser utilizado quando o cliente não tem autorização para acessar determinado recurso

REGRA: 403 forbidden deve ser utilizado para proibir o acesso a determinada parte da aplicação

REGRA: 406 not acceptable deve ser utilizado quando o tipo de mídia solicitado não é suportado

HTTP Headers

Regra: Content-type deve ser utilizado

Regra: É possível utilizar o content-length , nas responses ele indica o número de bytes utilizado e se as pessoas requisitar um HEAD também pode utilizar esse campo para saber o tamanho desse body sem acessar ele

Regra: Location pode ser utilizada para especificar qual a URI daquele novo recurso criado

Regra: Informações referentes a cache como Cache-control, Expires e Date também podem ser incluídas no Header

Segurança

Mantenha simples a proteção das APIs

Cada vez que você torna mais complexa a solução e a chance de deixar algum vazio na segurança é maior.

Sempre use HTTPS, pois permite enviar várias solicitações em uma única conexão

Use hash de senha

Nunca exponha informaçõe sensíveis nas URLs

JWT (JSON Web Token)

É um padrão aberto **(RFC 7519)** que define um modo compactado e independente para transmitir informações de forma segura .

Informações essas que podem ser verificadas pois são assinadas digitalmente. Podendo ser essa assinatura via segredo ou par de chaves pública/privada.

Qual a estrutura de um token JWT:

- **Header**
- **Payload**
- **Signature**

Header

Geralmente composto de duas partes: o tipo do token e o algoritmo utilizado

por ex:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Payload

Contém os claims, que são informações sobre um usuário e alguns dados adicionais

Registered claims

são recomendados , porém não obrigatórios, e podem conter informações como: : iss (issuer), exp (expiration time), sub (subject), aud (audience), dente outras

Public claims

São informações que podem ser definidas com o que for necessário naquele token, mas para evitar problemas devem ser definidos no IANA JSON Web Token Registry ou como um URI

Private claims

Também são personalizadas para passar as informações necessárias entre as partes

```
{
  "sub": "1234567890",
  "name": "Kamila",
  "admin": true
}
```

codificada em base 64

Signature

Para fazer a signature, devemos pegar o header codificado, o payload codficado, um secret, um algortimo e então, seria algo semelhante a:

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

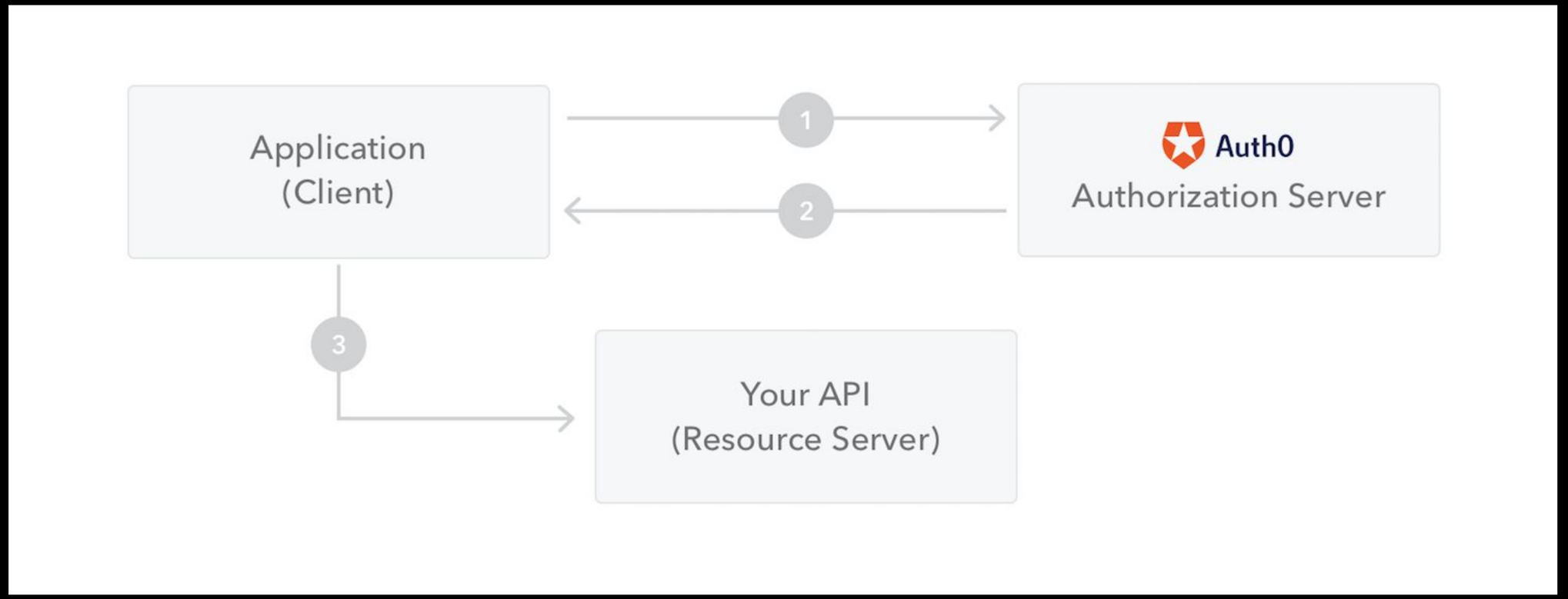


Imagem retirada de :

https://jwt.io/introduction

Oauth2

Permite que um usuário acesse um site de terceiros ou alguma aplicação acesso aos recursos desse usuário, sem revelar suas credenciais/identitdade

O oauth introduz uma camada de autorização e divide a função do cliente daquela do proprietário do recurso.

O client solicita acesso a recursos controlados por um proprietário e hospedados em um servidor de recursos e recebe um conjunto de credencias diferente do proprietário do recurso

Esses tokens são gerados no formato JWT e possui escopos específicos e é válido por um determinado período de tempo.

Referências:

https://www.oreilly.com/library/view/rest-api-design/9781449317904/

https://restfulapi.net/

Referências:

https://mundoapi.com.br/destaques/alcancando-a-excelencia-do-rest-com-um-modelo-de-maturidade-eficiente/

Referências:

https://www.enterpriseintegrationpatterns.com/patterns/messaging/EncapsulatedSynchronousIntegration.html

Referências:

https://martinfowler.com/articles/richardsonMaturityModel.html

Referências:

https://jwt.io/

https://auth0.com/docs/protocols/protocol-oauth2

Obrigada :)

https://www.linkedin.com/in/kamila-santos-oliveira/

@kamila_code

https://dev.to/kamilahsantos