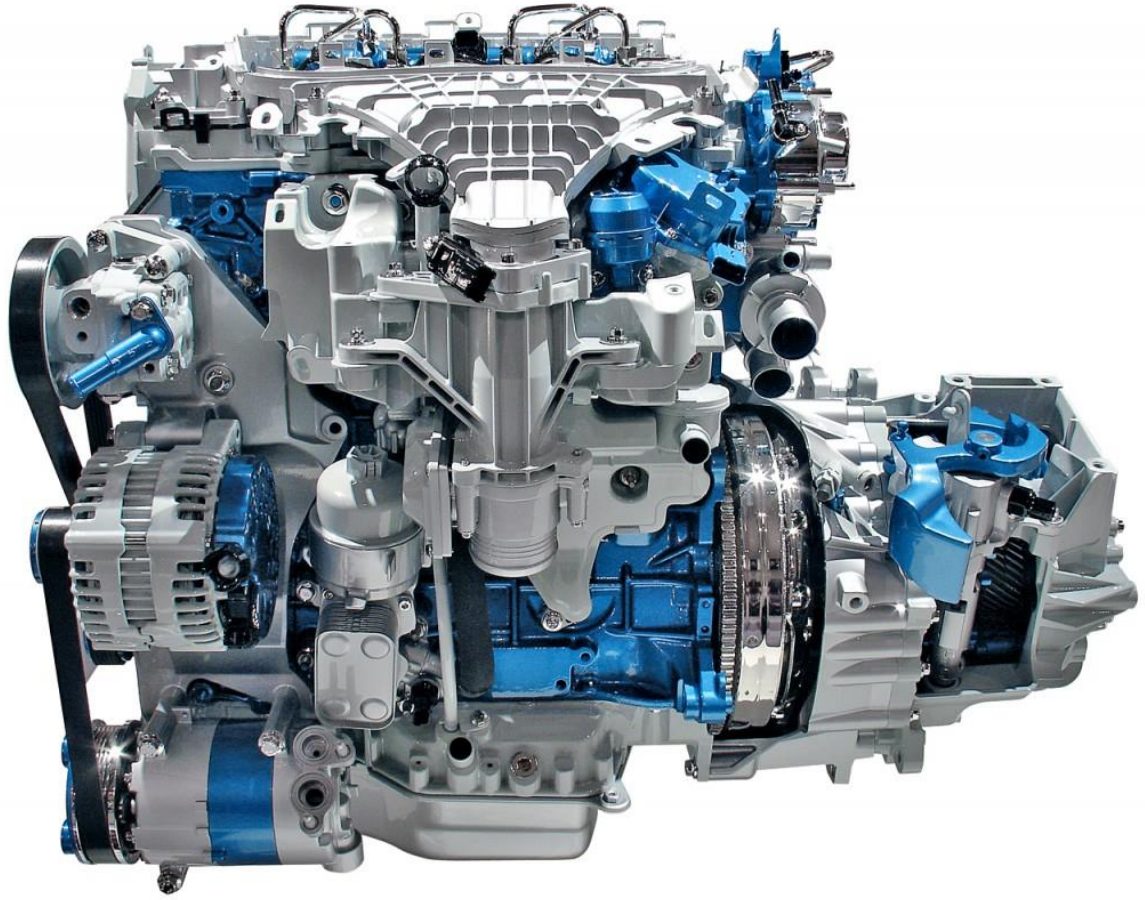
RESTFul API's com ASP.NET Core 2.0 do Zero A Nuvem

O que são Webservices



A definição do W3C

- Para o **W3C** os webservices são aplicações cliente servidor que se comunicam pela **World Wide Web's (WWW)** através do protocolo **HTTP (HyperText Transfer Protocol)** possibilitando a interoperabilidade entre softwares e aplicações executando em uma grande variedade de plataformas e frameworks. Caracterizam-se por sua grande interoperabilidade e extensibilidade podendo ser combinados de forma baixamente acoplada para executarem operações complexas. Programas proveem simples serviços que podem interagir uns com os outros gerando soluções sofisticadas.

No início tudo era muito obscuro

O mundo antes do REST!

Muitos 'padrões' diferentes :

RMI, SOAP, Corba, DCE, DCOM etc.

De muitas empresas diferentes :

Sun, Microsoft, IBM, OASIS, OMG etc.

Causando muitos problemas:

- ✓ Má interoperabilidade.
- ✓ Reinventar a roda.
- ✓ Atrelado a um fornecedor.

REST x SOAP!

#	SOAP	REST
1	Protocolo de troca de mensagens em XML	Um estilo arquitetural
2	Usa WSDL na comunicação entre cliente e servidor	Usa XML, JSON etc. para enviar e receber dados
3	Invoca serviços através de chamadas de método RPC	Simplesmente chama serviços via URL path
4	Não retorna um resultado facilmente legível para humanos	Resultado legível por humanos já que é simplesmente JSON ou XML por exemplo
5	Comunicação feita por HTTP mas pode usar outros protocolos como SMTP, FTP etc	Comunicação feita unicamente por HTTP
6	JavaScript pode invocar um serviço SOAP mas essa implementação é bastante complexa de se fazer	Fácil de invocar via JavaScript
7	Comparado com REST sua performance não é das melhores	Comparado com SOAP a performance é melhor consome menos recursos de processamento, código mais enxuto etc

SOAP

Client

+

=

Server

Dados + SOAP Standarts (Envelope)

Os dados trafegados entre cliente e servidor se tornam consideravelmente maiores. Assim como a personagem Vovozona de Martin Lawrence.

REST

Client

Dados

REST é como enviar os dados da forma que estão

Server


REST API

Definindo REST!

"Representational State Transfer (REST) é um estilo de arquitetura de software para sistemas distribuídos de hipermídia, como a World Wide Web"

IBM

REST é baseado em um conjunto de restriçoes

1. Cliente-servidor

Clientes e servidores separados.

2. Stateless server

O servidor não deve guardar o estado do cliente. Cada request de um cliente contém todas as informações necessárias para atendê-la.

3. Cacheável

O cliente deve ser informado sobre as propriedades de cache de um recurso para que possa decidir quando deve ou não utilizar cache.

4. Interface uniforme

Existe uma interface uniforme entre cliente e servidor.

- Identificação de recursos (URI).
- Manipulação de recursos a partir de suas representações.
- Mensagens auto descritivas.
- HATEOAS

5. Sistema em camadas

Deve suportar conceitos como balanceamento de carga, proxies e firewalls.

6. Código sob Demanda (opcional)

O cliente pode solicitar o código do servidor e executá-lo.

Formatos Suportados em Web Services REST

- XML
- JSON
- CSV
- Texto
- Imagens
- HTML
- PDF
- binário
- etc

Vantagens dos Web Services RESTful

- REST é um padrão arquitetural basicamente leve por natureza. Então quando você tiver limitações de banda prefira web services REST;
- Desenvolvimento fácil e rápido;
- Aplicativos Mobile tem ganhado cada vez mais espaço e precisam interagir rapidamente com os servidores e o padrão REST é mais rápido no processamento de dados das requests e responses.

Quem usa REST!

13 billion API calls / day *(May 2011)*
Approx. 75% of all traffic via API

5 billion API calls / day *(April 2010)*

5 billion API calls / day *(October 2009)*

1.4 billion API calls / day *(May 2012)*

1.1 billion API calls / day *(April 2011)*

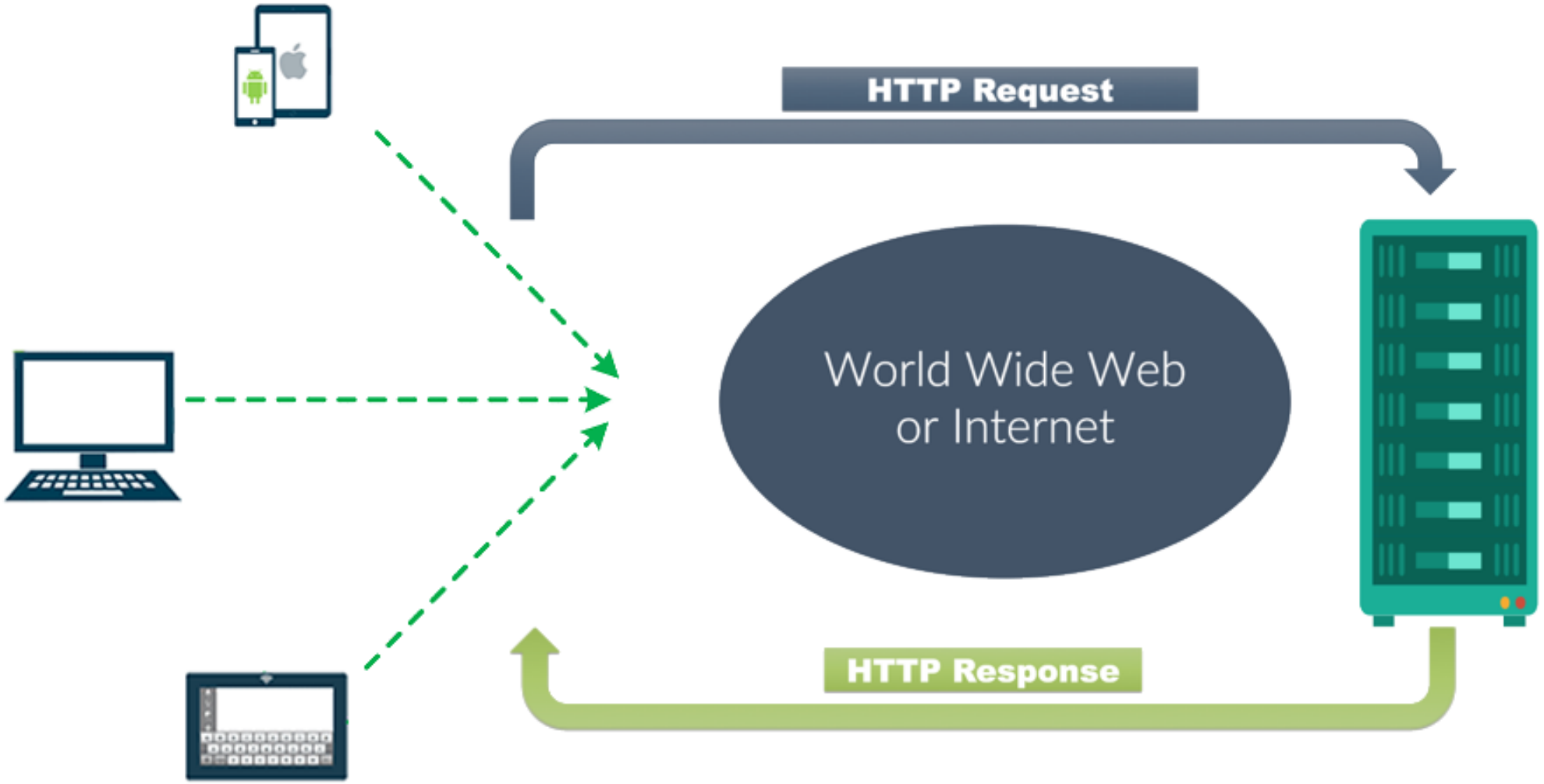
1 billion API calls / day *(Mar 2012)*

7,2 bilion API calls
(Mar/2015)

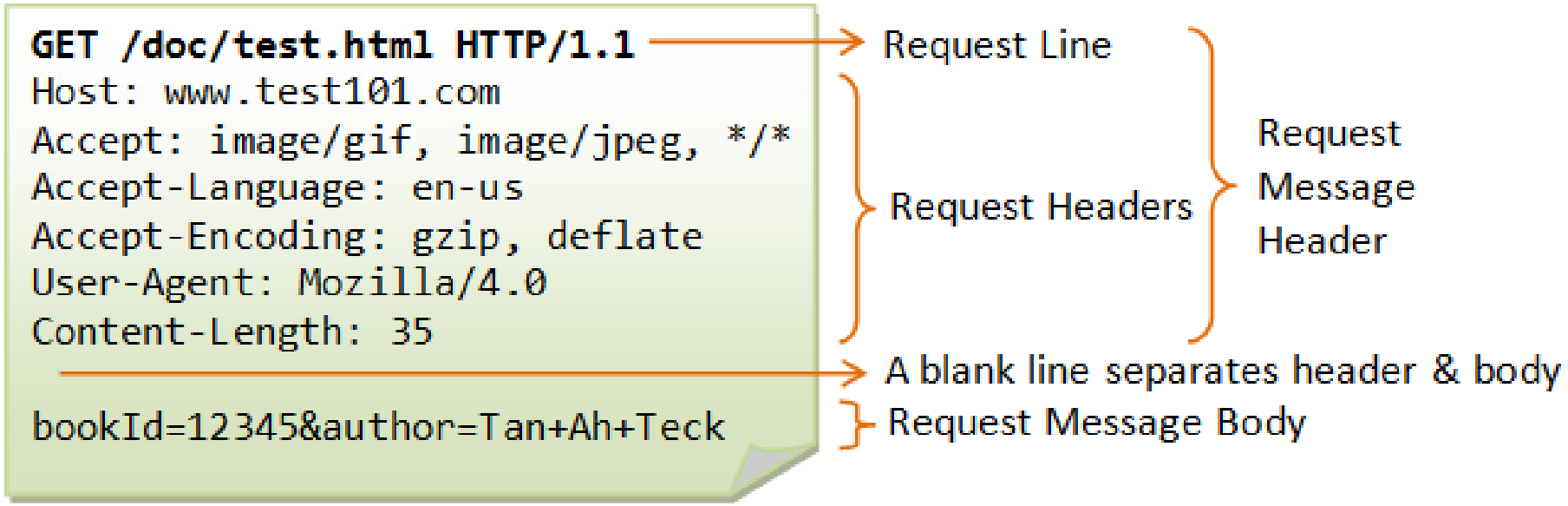
Fonte: ProgrammableWeb Founder

Request e Response

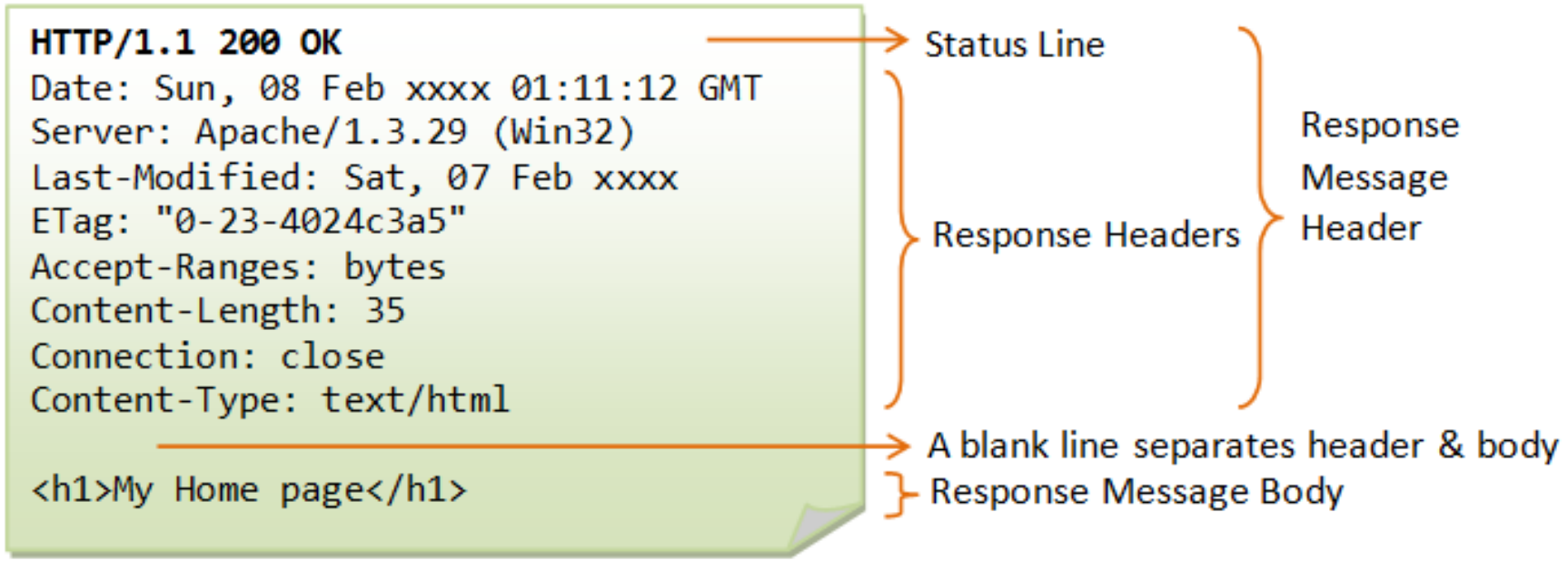
Request e Response



Request


GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35
bookId=12345&author=Tan+Ah+Teck
Request Line
Request Headers
Request Message Header
A blank line separates header & body
Request Message Body

Response


HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html
<h1>My Home page</h1>
Status Line
Response Headers
Response Message Header
A blank line separates header & body
Response Message Body

Tipos de Parâmetros - Path Params

GET {{host}}:{{port}}/api/persons/v1/find-with-paged-search/asc/10/1

GET {{host}}:{{port}}/api/persons/v1/find-with-paged-search/asc/10/1

Tipos de Parâmetros - Query Params

GET {{host}}:{{port}}/api/persons/v1/find-by-name?firstName=Leandro&lastName=Costa

GET {{host}}:{{port}}/api/persons/v1/find-by-name?firstName=Leandro&lastName=Costa

Tipos de Parâmetros - Header Params

GET {{host}}:{{port}}/api/file/v1

Authorization | Headers (3) | Body | Pre-request Script | Tests

Key	Value
Accept	text/plain, application/json, text/json, application/xml, text/xml
Content-Type	application/json
Authorization	Bearer {{bearer_token}}

Tipos de Parâmetros - Body Params

POST {{host}}:{{port}}/api/login/v1

Authorization Headers (2) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON (application/json)

```
{
    "login": "{{user}}",
    "accessKey": "{{password}}"
}
```

HTTP Status Codes

200 Resultado OK

400 Erro no Client

500 Erro no Server

HTTP Status Codes

1×× Informational

100 Continue
101 Switching Protocols
102 Processing

2×× Success

200 OK
201 Created
202 Accepted
203 Non-authoritative Information
204 No Content
205 Reset Content
206 Partial Content
207 Multi-Status
208 Already Reported
226 IM Used

3×× Redirection

300 Multiple Choices
301 Moved Permanently
302 Found
303 See Other
304 Not Modified
305 Use Proxy
307 Temporary Redirect
308 Permanent Redirect

4×× Client Error

400 Bad Request
401 Unauthorized
402 Payment Required
403 Forbidden
404 Not Found
405 Method Not Allowed
406 Not Acceptable
407 Proxy Authentication Required
408 Request Timeout
409 Conflict
410 Gone
411 Length Required
412 Precondition Failed
413 Payload Too Large
414 Request-URI Too Long
415 Unsupported Media Type
416 Requested Range Not Satisfiable
417 Expectation Failed
418 I'm a teapot
421 Misdirected Request
422 Unprocessable Entity
423 Locked
424 Failed Dependency
426 Upgrade Required
428 Precondition Required
429 Too Many Requests
431 Request Header Fields Too Large
444 Connection Closed Without Response
451 Unavailable For Legal Reasons
499 Client Closed Request

5×× Server Error

500 Internal Server Error
501 Not Implemented
502 Bad Gateway
503 Service Unavailable
504 Gateway Timeout
505 HTTP Version Not Supported
506 Variant Also Negotiates
507 Insufficient Storage
508 Loop Detected
510 Not Extended
511 Network Authentication Required
599 Network Connect Timeout Error

https://httpstatuses.com/

HTTP Status Codes em Serviços REST

- **200 OK** – Request de criação ou deleção executada com sucesso.

  201 Created – Criação de uma fila, tópico, fila temporária, tópico temporária, session, producer, consumer, listener, queue browser ou mensagem realizada com sucesso.

  204 No Content – deleção de uma fila, tópico, sessão, producer ou listener bem sucedida mas sem retorno de conteúdo.

HTTP Status Codes em Serviços REST

- **400 Bad Request** – O path informado está em um formato incorreto, um parâmetro ou valor do corpo da requisição não está formatado corretamente ou um parâmetro obrigatório não foi informado,
  ou está formatado corretamente mas pode estar eventualmente inválido (por exemplo, o ID informado não existe – NullPointerException, o conteúdo retornado é muito grande ou o ID informado já está em uso).

- **401 Unauthorized** – O cliente não tem autorização para executar requisições na operação em questão.

- **403 Forbidden** – O cliente não tem permissão para executar requisições na operação em questão.

HTTP Status Codes em Serviços REST

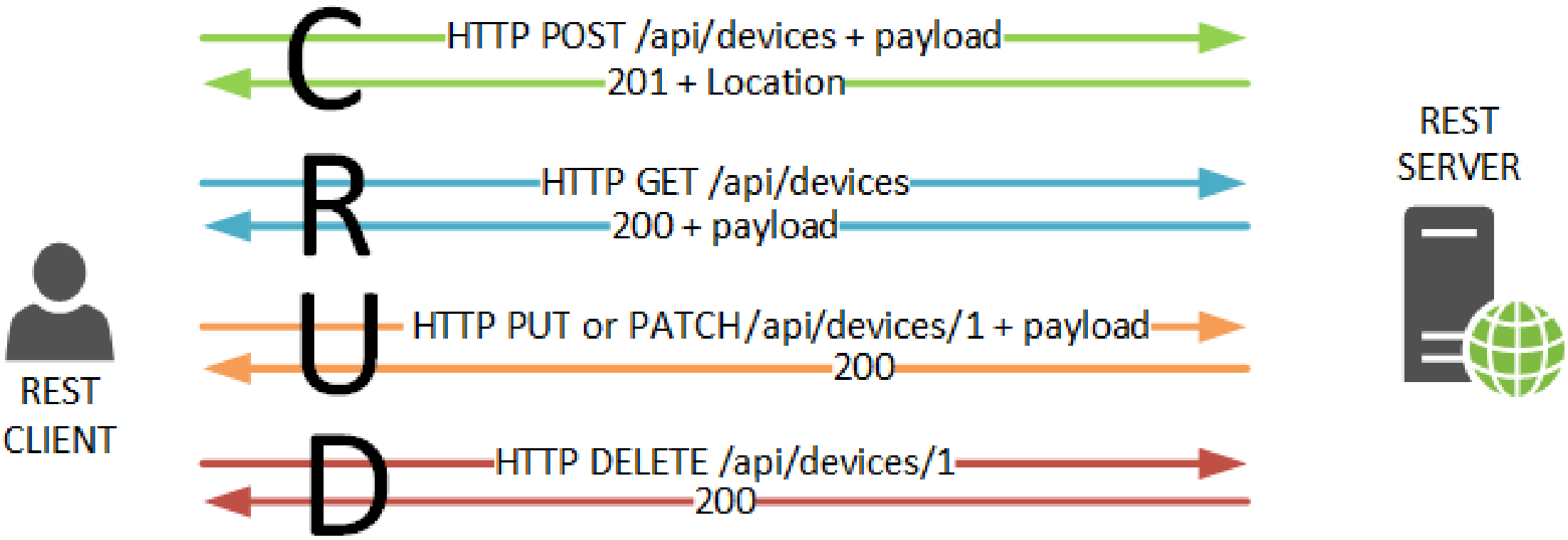
- **404 Not Found** – o objeto requisitado pelo path não existe (NullPointerException ou NullReferenceException).

  405 Method Not Allowed – O usuário não tem permissão de acesso ao path.

  409 Conflict – Um objeto já foi criado com as mesmas informações.

  500 Internal Server Error – Ocorreu uma falha no servidor, podendo ser desde uma falha no SQL por exemplo.

Os verbo HTTP



GET – READ/para selecionar/recuperar um recurso

- O verbo HTTP GET é usado para ler ou recuperar uma representação de um recurso. Em um “cenário feliz”, uma requisição GET retorna uma representação em **XML** ou **JSON** e um **HTTP** status code 200 (OK). Em um cenário de erro o retorno mais comum é **404 (NOT FOUND)** ou **400 (BAD REQUEST)**.

Parâmetros suportados

- Via URL (PATH ou QUERY PARAMS)
- Via HEADER

POST – CREATE/para inserir recurso

- O verbo **HTTP POST** é mais freqüentemente usado para criar novos recursos — inserir um novo item na base. Em uma aplicação **REST** perfeita quando uma operação é executada com sucesso, retorna-se o status code 200 ou 201.

Parâmetros suportados

- Via URL (PATH ou QUERY PARAMS)
- Via HEADER
- Via Body

PUT - UPDATE/para modificar um recurso

- O verbo **PUT** é comumente usado para atualizar informações, colocando um recurso conhecido no (body) corpo da requisição contendo novas informações que representam o recurso original.

- Um update bem sucedido, retorna um **status code** 200 (ou 204 quando não retorna nenhum conteúdo no body.

Parâmetros suportados

- Via URL (PATH ou QUERY PARAMS)
- Via HEADER
- Via Body

DELETE - DELETE/para remover um recurso

- O verbo DELETE é fácil de entender, ele é usado para deletar um recurso identificado por uma URI.
- Em uma deleção bem sucedida retorna-se um **status code 200 (OK)** juntamente com um response body, possivelmente uma representação do item deletado (o que acaba por demandar muita banda), ou uma response customizada.
- Ou retornar o **status code 204 (NO CONTENT)** sem response body ou um **status code 204** sem corpo, ou JSEND-style response com um **status code** 200 são as responses mais recomendadas.

Parâmetros suportados

- Via URL (PATH ou QUERY PARAMS)
- Via HEADER
- Via Body

Outros verbos menos conhecidos

- **PATCH** – O verbo **PATCH** pode ser usado para realizar updates parciais de um recurso. Por exemplo, quando você precisar alterar apenas um campo em um recurso, executar um POST com todo o objeto é pesado e acarreta em um maior consumo de banda.

- Use-o com moderação pois colisões entre múltiplas PATCH requests são mais perigosas que colisões entre PUT requests por que exige que o cliente tenha informações básicas do recurso ou irão corrompê-lo.

Outros verbos menos conhecidos

- **HEAD** – O verbo **HEAD** possui uma funcionalidade similar ao verbo **GET**, exceto pelo fato do servidor retornar uma response line e headers, mas sem um entity-body.

Outros verbos menos conhecidos

- **TRACE** – O verbo **TRACE** é usado para recuperar o conteúdo de uma requisição **HTTP** de volta podendo ser usado com o propósito de debug durante o processo de desenvolvimento.

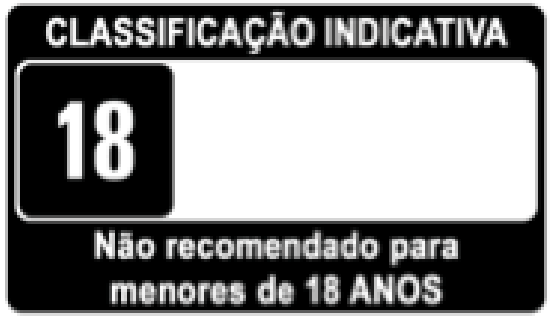
Outros verbos menos conhecidos

- **OPTIONS** – O verbo **OPTIONS** é usado pelo cliente para encontrar operações **HTTP** e outras opções suportadas pelo servidor. O cliente pode especificar uma **URL** para o verbo **OPTIONS** ou um asterisco (*) para se referir a todo o servidor.

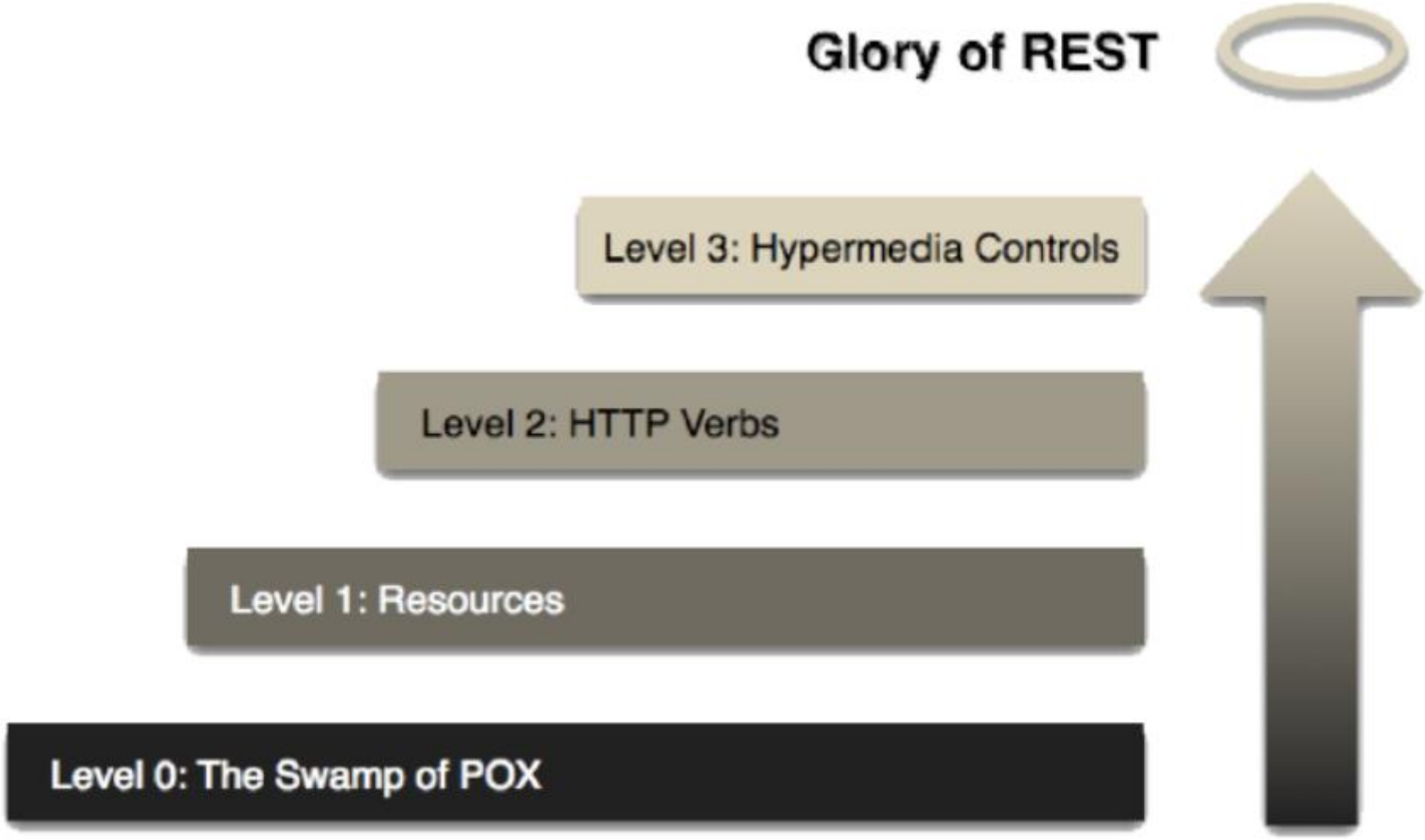
Outros verbos menos conhecidos

- **CONNECT** – O verbo **CONNECT** é usado pelo cliente para estabelecer uma conexão de rede com um servidor via **HTTP**.

Do REST ao RESTFul



Níveis de maturidade de Richardson



Então, são nível 0, 1 e 2 RESTful?

“O que precisa ser feito para tornar o estilo de arquitetura REST claro sobre a noção de que o hipertexto é uma restrição? Em outras palavras, se o mecanismo do estado do aplicativo (e, portanto, a API) não estiver sendo orientado ao hipertexto, então não poderá ser RESTful e não poderá ser uma API RESTFul por completo. Há algo de errado em algum lugar que precisa ser consertado?

Roy T. Fielding


HATEOAS
Hypermedia as the Engine of Application State

```
[
    {
        "id": 1,
        "firstName": "Leandro",
        "lastName": "Costa",
        "address": "Uberlândia - Minas Gerais - Brasil",
        "gender": "Male",
        "links": [
            {
                "rel": "self",
                "href": "http://localhost:50904/api/persons/v1/1",
                "type": "application/json",
                "action": "GET"
            },
            {
                "rel": "self",
                "href": "http://localhost:50904/api/persons/v1/1",
                "type": "application/x-www-form-urlencoded",
                "action": "POST"
            },
            {
                "rel": "self",
                "href": "http://localhost:50904/api/persons/v1/1",
                "type": "application/x-www-form-urlencoded",
                "action": "PUT"
            },
            {
                "rel": "self",
                "href": "http://localhost:50904/api/persons/v1/1",
                "type": "int",
                "action": "DELETE"
            }
        ]
    }
]
```

OPEN API

INITIATIVE

Documentando RESTFull API's

swagger

swagger

Select a spec: My API V1

RESTful API With ASP.NET Core 2.0 v1

/swagger/v1/swagger.json

Books

GET /api/Books/v{version}

PUT /api/Books/v{version}

POST /api/Books/v{version}

GET /api/Books/v{version}/{id}

DELETE /api/Books/v{version}/{id}

File

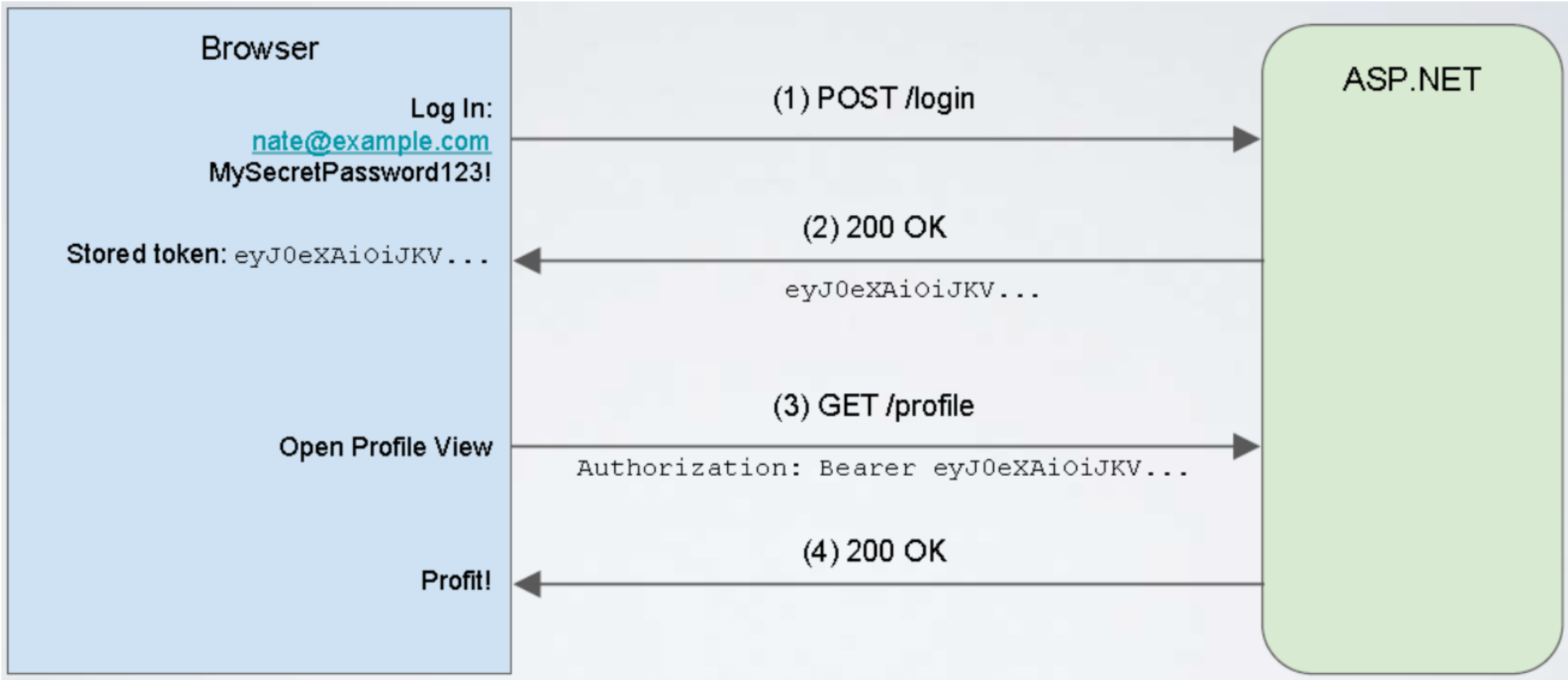
GET /api/File/v{version}

Login

POST /api/Login/v{version}

Autorização e Autenticação

Como funciona um token de autenticação



Anatomia de um JSON Web Token (JWT)

https://github.com/nbarbettini/SimpleTokenProvider

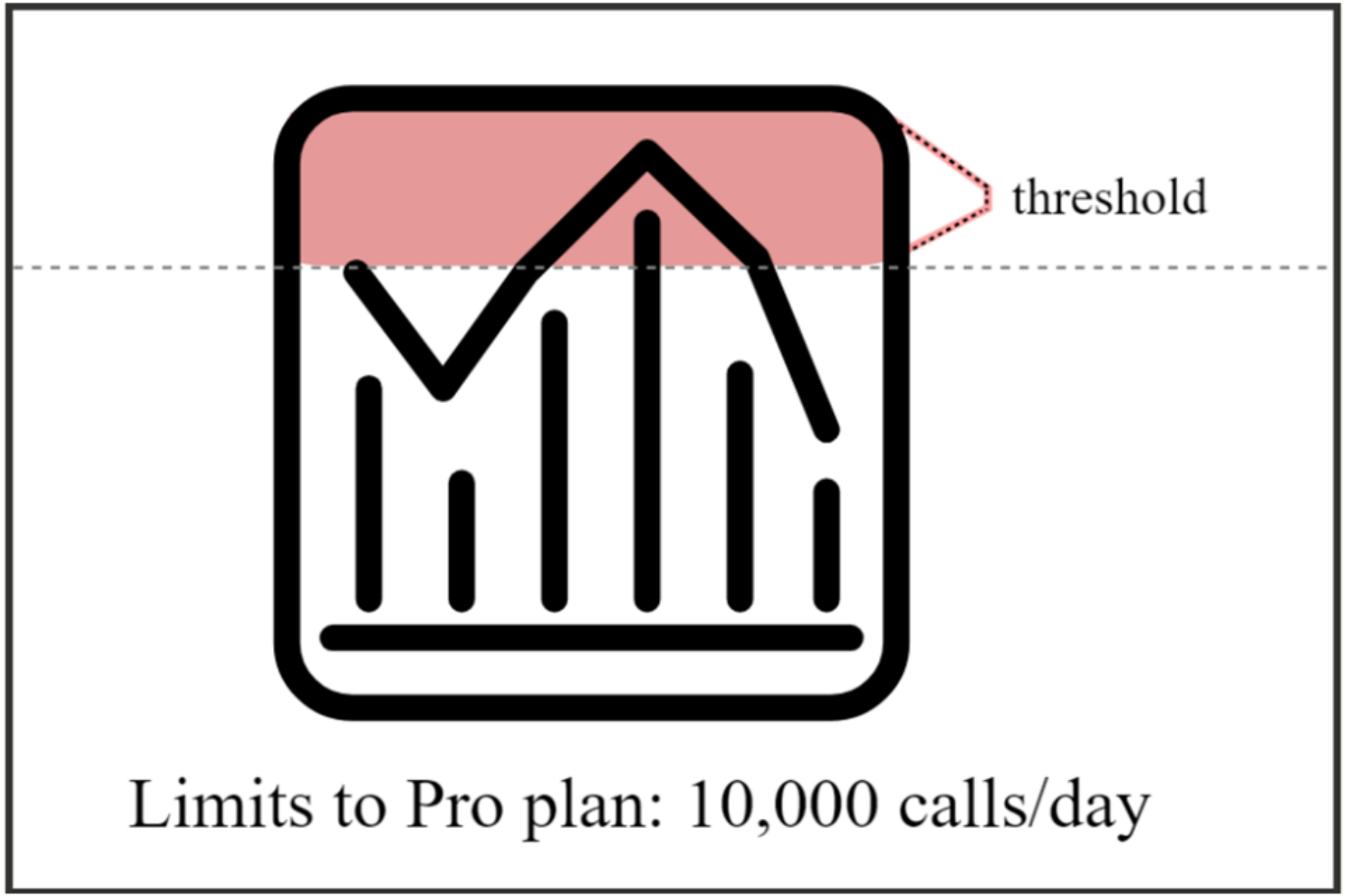
```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJPbmxpb
mUgSldUIEJ1aWxkZXIiLCJpYXQiOjE0NjU1ODAwNzEsImV4cCI6MTQ
5NzExNjA3NywiYXVkIjoid3d3LmV4YW1wbGUuY29tIiwic3ViIjoib
mF0ZUBleGFtcGxlLmNvbSIsImlzQXdlc29tZSI6InRydWUiLCJwcm9
2aWRlcyI6WyJzdGF0ZWxlc3MiLCJhdXRoZW50aWNhdGlvbiJdfQ.VX
rLbyQeJfDmwTAg-JnRsyD23RYMQJshTx79z2STu0U
```

Red = Header
Blue = Payload ("claims")
Green = Cryptographic signature (JWS)

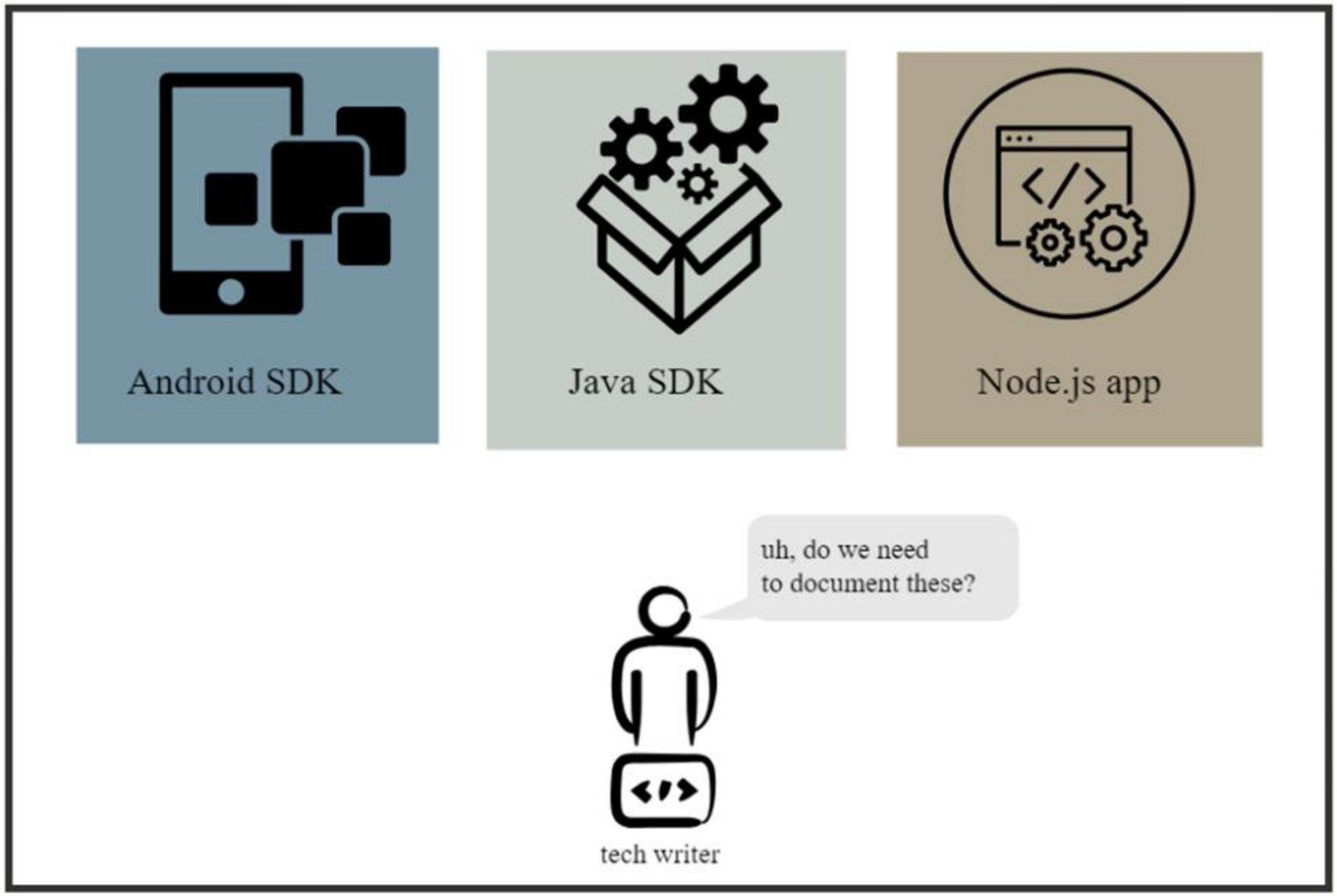
Versionamento de API's

Indo além do RESTFul

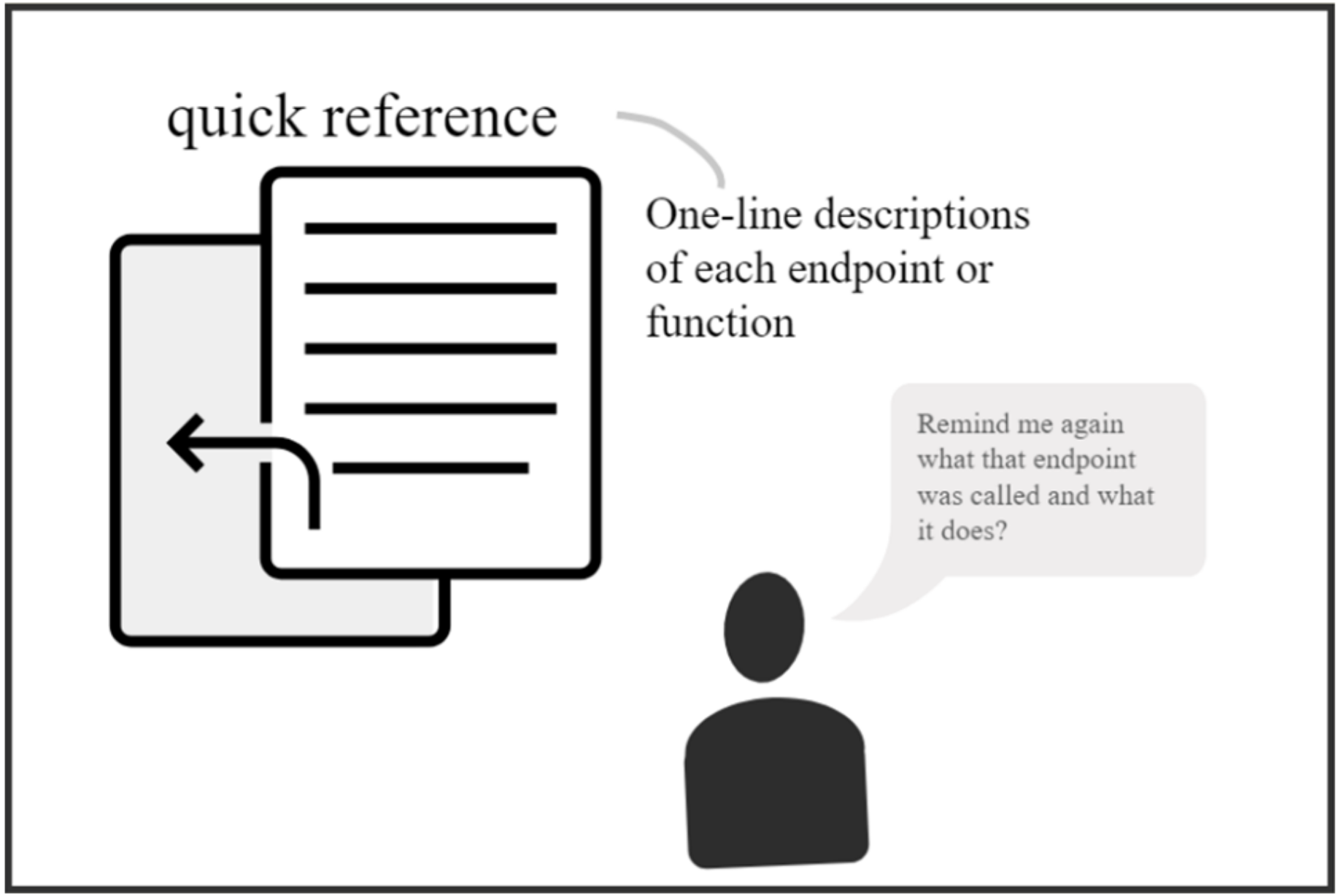
Limite de requisições e acessos



SDK’s e sample apps



Referência Rápida




BEST PRACTICE
BEST PRACTICE
BEST PRACTICE

Boas Práticas

- Paginação
- Filtros
- Definir recursos lógicos
- Tolerância a falhas
- Cache
- Conectividade

Boas Práticas

- Timeouts
- Documentação
- Utilizar SSL
- Versionamento
- Teste e validação
- Self-Service
- Divulgação

Boas Práticas

- Exportações
- I18n / Globalization
- Notificações
- Limite de campos
- Monitore sua API
- Selecione a Tecnologia Adequada
- etc.


That's all Folks!

REFERÊNCIAS

[Posts Sobre REST Semeru](http://www.semeru.com.br/blog/tag/rest/page/2/)

[Introduction to API Versioning Best Practices](https://nordicapis.com/introduction-to-api-versioning-best-practices/)

[RESTEasy JAX-RS](http://docs.jboss.org/resteasy/docs/3.0.7.Final/userguide/html/)

[HTTP - Methods](http://www.tutorialspoint.com/http/http_methods.htm)

[What is HATEOAS and why is it important for my REST API?](http://restcookbook.com/Basics/hateoas/)

[Using HTTP Methods for RESTful Services](http://www.restapitutorial.com/lessons/httpmethods.html)

[Introduction to RESTful Web Services—A JAX-RS Specification](https://dzone.com/articles/introduction-to-restful-we
service-a-jax-rs-speci)

[Learn REST: A Tutorial](http://rest.elkstein.org/)

REFERÊNCIAS

[Tools to Make HATEOAS Compliance Easier](https://nordicapis.com/tools-to-make-hateoas-compliance-easier/)

[10+ API Monitoring Tools](https://nordicapis.com/10-api-monitoring-tools/)

[HTTP Headers](https://docs.trafficserver.apache.org/en/5.3.x/sdk/http-headers.en.html)

[HTTP (HyperText Transfer Protocol)](https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html)

[Communication Networks/HTTP Protocol](https://en.wikibooks.org/wiki/Communication_Networks/HTTP_Protocol)

[HTTP Status Codes](https://httpstatuses.com/)

REFERÊNCIAS

[Features que sua API REST precisa ter desde o início](https://tableless.com.br/features-que-sua-api-rest-precisa-ter-desde-o-inicio/)

[HAL - Hypertext Application Language](http://stateless.co/hal_specification.html)
[ASP.NET API Versioning](https://github.com/Microsoft/aspnet-api-versioning/wiki/Versioning-via-the-URL-Path#aspnetcore)

[Beautiful REST API design with ASP.NET Core and Ion](https://github.com/nbarbettini/BeautifulRestApi)
[Introduction to REST APIs](http://idratherbewriting.com/learnapidoc/docapis_introtoapis.html)

[REST API in ASP.NET CORE](https://www.fortech.ro/rest-api-asp-net-core/)

[Zalando RESTful API and Event Scheme Guidelines](https://opensource.zalando.com/restful-api-guidelines/)

REFERÊNCIAS

[Modelo de maturidade de Richardson – os passos para a glória do REST](http://www.boaglio.com/index.php/2016/11/03/modelo-de-maturidade-de-richardson-os-passos-para-a-gloria-do-rest/)

[Alcançando a glória REST com o Modelo de Maturidade de Richardson](https://arrayoutofindex.wordpress.com/2017/06/17/alcancando-a-gloria-rest-com-o-modelo-de-maturidade-de-richardson/)

[REST - Modelo de Maturidade de Richardson](http://fernandoanselmo.blogspot.com.br/2013/09/rest-modelo-de-maturidade-de-richardson.html)