


Instituto Politécnico de Castelo Branco
Escola Superior de Tecnologia

APIs e Serviços REST (versão resumida)

Unidade Curricular
Aplicações Internet Distribuídas / Aplicações Distribuídas

Licenciatura em Engenharia Informática
Mestrado em Desenvolvimento de Software e Sistemas Interactivos
UTC Informática
Est-IPCB
Ano Letivo 2022/2023

Prof.º Doutor Alexandre Fonte
(adf@ipcb.pt)

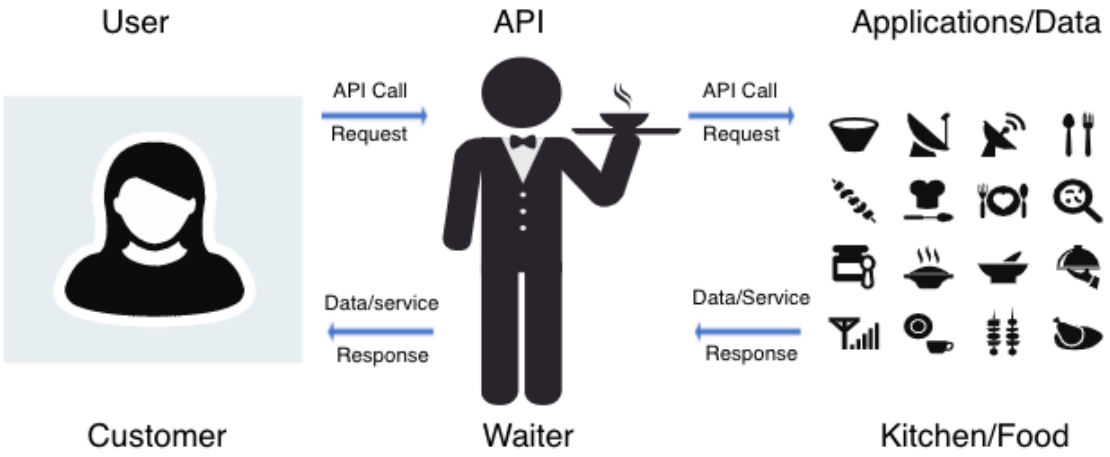
Versão: 14 outubro de 2022

Sumário

- APIs e Web Services RESTful
  - Conceito de API
  - Estilos Arquitecturais
  - Protocolo HTTP
  - Conceitos REST
    - Estrutura de um URI e Convenções REST
    - Criação de Operações CRUD
    - Internet MediaTypes
    - Controlo de Versões

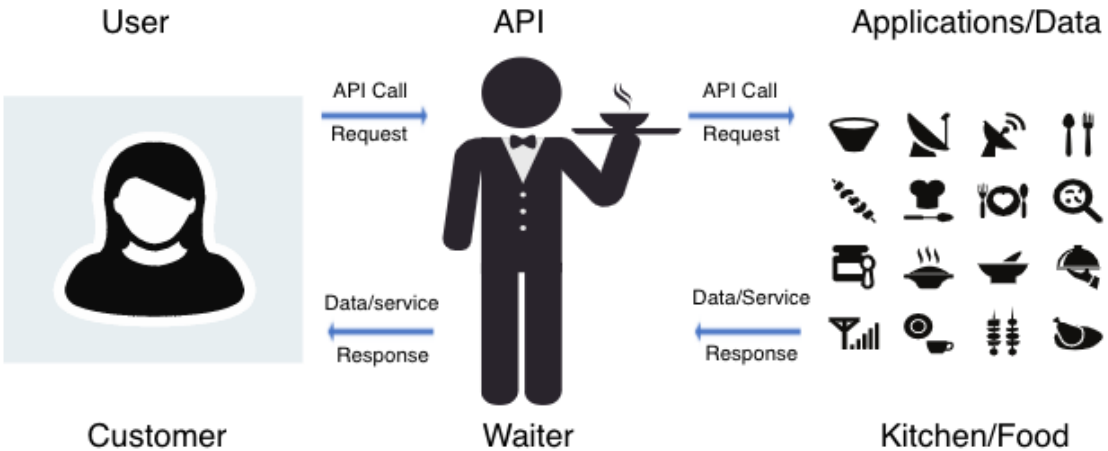
Conceito de API

- Uma API (Application Programming Interface) é um software ou componente que permite que outras aplicações/ componentes de software acedam aos seus **dados** ou **serviços**.
- É semelhante a uma tomada eléctrica.
- É um conjunto de regras que descrevem/determinam a forma como uma aplicação pode interagir com outra e as instruções para permitir que a interação ocorra.
- Uma API é semelhante a um empregado de mesa de um restaurante:



Como Funciona uma API

- Here's how APIs work:
  - A client initiates an API call to retrieve data from the application - also known as a request. Leveraging the API's URI (Uniform Resource Identifier), this request is then processed from the application to the web server and includes headers, a request verb and occasionally a request body.
  - After the API receives a valid request it makes a call to the external program or web server.
  - The web server sends the requested information as a response to the API.
  - Finally, API sends the response to the client application.



Estilos Arquitecturais de APIs

- Simple Object Access Protocol (SOAP)
  - Usa SOAP e um protocol de transporte Internet (e.g., HTTP ou outro)
- Representational State Transfer (REST)
  - Usa HTTP e modela tudo como um recurso que pode fornecer dados
  - N aplicações cliente
  - A lógica de negócio e dados estão desacoplados. Num pedido é devolvido "todos" os dados sob o "domínio".

curl http://localhost:1337/elusive/cat/1

{"id":1,"name":"Snow Leopard","conservationStatus":"Vulnerable"}

Estilos Arquitecturais de APIs (cont.)
Simple Object Access Protocol (SOAP)

- Simple Object Access Protocol (SOAP) é um protocolo de mensagens baseado em XML. Ele é usado para comunicação entre aplicações em diferentes plataformas ou criadas usando diferentes linguagens de programação.
- SOAP é:
  - **Independent:** executadas em diferentes sistemas operativos/plataformas aplicacionais
  - **Extensible:** Adicione recursos como confiabilidade e segurança
  - **Neutro:** Pode ser usado em qualquer protocolo, incluindo HTTP, SMTP, TCP, UDP ou JMS
- Uma mensagem SOAP é um documento XML que pode conter os quatro elementos a seguir:
  - **Envelope** – o elemento raiz do documento XML.
  - **Header** - contém informações específicas da aplicação, como autorização, atributos específicos do SOAP e assim por diante
  - **Body** - contém os dados a serem transportados para o destinatário
  - **Fault** - fornece informações de erro e / ou status

Estilos Arquitecturais de APIs (cont.)
Simple Object Access Protocol (SOAP)

- Exemplo de mensagem SOAP:

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header/>
    <soap:Body>
        <soap:Fault>
            <faultcode>soap:Server</faultcode>
            <faultstring>Query request too large.</faultstring>
        </soap:Fault>
    </soap:Body>
</soap:Envelope>
```

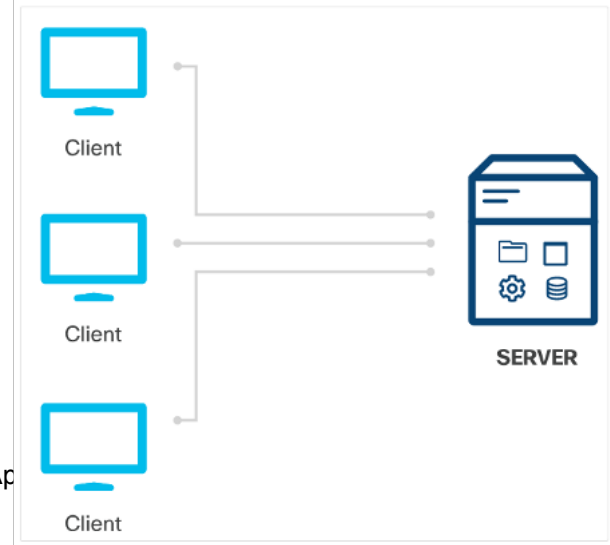
Estilos Arquitecturais de APIs (cont.)
REpresentational State Transfer (REST)

- REpresentational State Transfer (REST) é um estilo arquitectural da autoria de Roy Thomas Fielding (um dos autores da especificação HTTP)
- Uma API é considerada “RESTful” se respeitar os principios REST:
  - **P1: Uso de uma interface Uniforme, restrita**
    - Esta desacopla as implementações do cliente e do serviço, tornando-os independentes um do outro
    - Utilizando o protocolo de comunicações HTTP
    - Utilizando os quatro principais métodos HTTP para definir operações sobre os recursos

      GET, POST, PUT, PATCH e DELETE

Estilos Arquitecturais de APIs (cont.)
REpresentational State Transfer (REST)

- Uma API é considerada “RESTful” se respeitar os principios REST:
- **P2: Comunicações stateless.**
  - Os pedidos HTTP são independentes e podem ocorrer em qualquer ordem
  - Cada pedido deverá ser uma operação atómica
  - Nenhuns dados do cliente são armazenados entre pedidos.
  - Os únicos dados guardados são os dos recursos.



Estilos Arquitecturais de APIs (cont.)
REpresentational State Transfer (REST)

- Uma API é considerada “RESTful” se respeitar os principios REST:
  - **-P3: Tudo é Endereçavel via um URI**
    - •Um recurso tem um identificador único
    - •Exemplo: **https://adventure-works.com/orders/1**
  - **-P4: Orientação à representação de recursos**
    - •Toda a informação e funcionalidade no lado servidor é vista com um recurso
    - •Um recurso é qualquer tipo objecto de informação ou serviço que pode ser acedido pelo cliente
    - •***Um recurso é representado na forma*** **de docs XML/JSON**
    - •Exemplo: {"orderId":1,"orderValue":99.90,"productId":1,"quantity":1}

Revisão Protocolo HTTP (Opcional)

Componentes de um pedido HTTP

- Método
  - GET, POST, PUT, DELETE, etc.
- URI – Uniform Resource Identifier
  - http://www.ipcb.pt/index.html
- Cabeçalhos (headers) – permite ao cliente passar informação adicional
  - User-Agent, Content-Type, Accept-Language, Authorization, Cookie etc.
- Corpo (Body) – dados a enviar, se existirem
  - O formato dos dados (e.g., xml, json) deve ser do tipo do cabeçalho Content-Type

```
GET /download.html HTTP/1.1
Host: www.ethereal.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.6) Gecko/20040113
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.ethereal.com/development.html
```

Métodos de Requisição HTTP

- Por vezes referidos como 'HTTP verbs'
- GET
  - Obtém/consulta a informação usando um determinado URI
  - Não existem corpo, os parâmetros são passados no URI
- POST
  - Envia dados ao servidor, usado para submeter formulário HTLM
  - Os dados são apresentados de diferentes formatos, especificados no cabeçalho Content-Type
- PUT
  - Editar ou atualizar uma entidade de dados (criada antes)
- DELETE
  - Remove an existing data entity
- Existem muitos outros métodos, mas pouco usados ou com pouco interesse (ver https://developer.mozilla.org/pt-BR/docs/Web/HTTP)

Respostas HTTP

- **Status Code + Descrição**
  - 200 OK, 404 Not Found, 400 Bad Request, 401 Unauthorized, 500 Internal Server Error

- Response Headers – permite a um servidor passar informação
  - Server, Content-Type, Last-Modified, Set-Cookie, etc.

- Corpo – dados retornados pelo servidor
  - O formato dos dados deve coincidir com os do Content-Type

```
HTTP/1.1 200 OK
Date: Thu, 13 May 2004 10:17:12 GMT
Server: Apache
Last-Modified: Tue, 20 Apr 2004 13:17:00 GMT
ETag: "9a01a-4696-7e354b00"
Accept-Ranges: bytes
Content-Length: 18070
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=ISO-8859-1

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
```

Respostas HTTP (Cont.)

HTTP Status /Código de Estado HTTP

- O código de estado HTTP informa o cliente se o pedido foi aceite ou em caso de erro pode ajudar o cliente a determinar a razão do erro e pode, por vezes, fornecer sugestões para corrigir o problema.
- **Os códigos de estado HTTP consistem em três dígitos**, onde o primeiro dígito é a categoria de resposta e os outros dois dígitos são atribuídos por ordem numérica.
- Existem cinco categorias diferentes de códigos de estado HTTP:

1xx – Informational – para fins informativos, as respostas não contêm um corpo

2xx – Success – o servidor recebeu e aceitou o pedido

3xx – Redirection – o cliente precisa de realizar uma acção adicional para que o pedido seja completado

4xx -- Client Error – o pedido contém um erro, tal como erro de sintaxe ou entrada inválida

5xx -- Server Error – incapaz de satisfazer o pedido válido.

Respostas HTTP (Cont.)

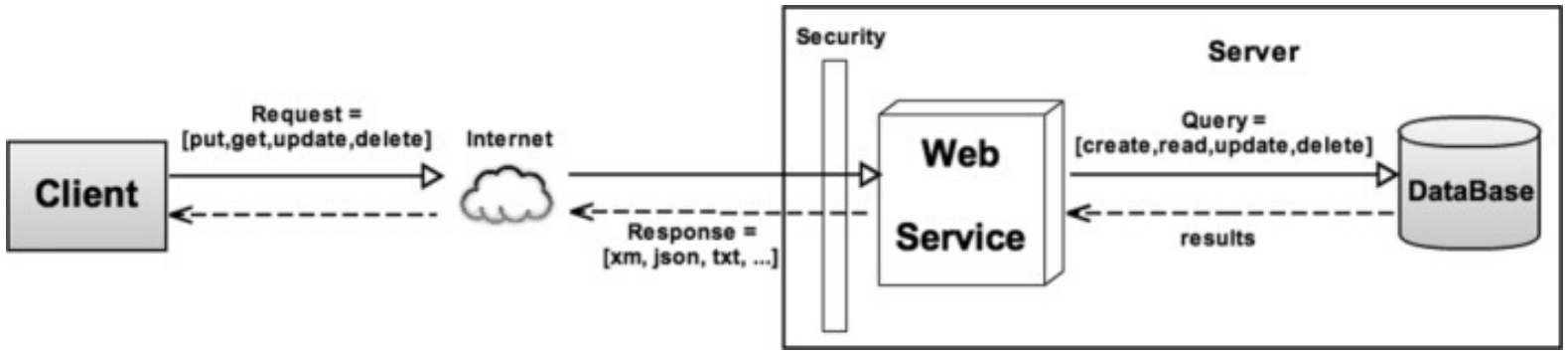
Os códigos de status HTTP mais comuns:

HTTP Status Code	Status Message	Description
200	Ok	Request was successfully and typically contains a payload (body)
201	Created	Request was fulfilled and the requested resource was created
202	Accepted	Request has been accepted for processing and is in process
400	Bad Request	Request will not be processed due to an error with the request
401	Unauthorized	Request does not have valid authentication credentials to perform the request
403	Forbidden	Request was understood but has been rejected by the server
404	Not Found	Request cannot be fulfilled because the resource path of the request was not found on the server
500	Internal Server Error	Request cannot be fulfilled due to a server error
503	Service Unavailable	Request cannot be fulfilled because currently the server cannot handle the request

API REST

API REST

- Uma API REST é uma API que opera/comunica sobre o protocolo HTTP.
  - Define um conjunto de funções que os programadores podem usar para executar pedidos e receber respostas via protocolo HTTP
  - Usa os mesmos conceitos que o HTTP:
    - Mesmo Modelo Pedidos/Respostas HTTP
    - Mesmos Verbos HTTPs
    - Mesmos Códigos de Estado HTTP
    - Mesmos Cabeçalho e Body HTTP



Princípios REST

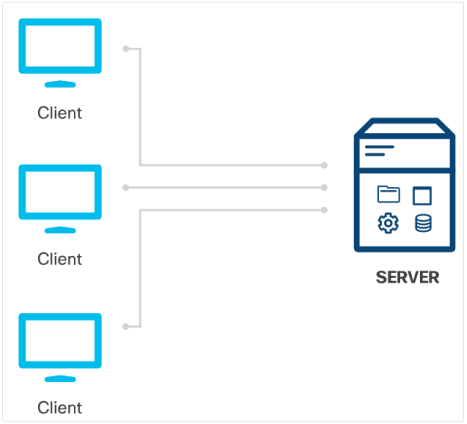
Uma API é considerada “RESTful” se respeitar os princípios REST:

- **P1: Uso de uma interface Uniforme, restrita**
- **P2: Comunicações stateless.**
- **P3: Tudo é Endereçavel via um URI**
- **P4: Orientação à representação de recursos**

Princípios REST

- **P1: Uso de uma interface Uniforme, restrita**
  - Esta desacopla as implementações do cliente e do serviço
  - Utilizando o protocolo de comunicações HTTP
  - Utilizando os quatro principais métodos HTTP para definir operações sobre os recursos
    - GET, POST, PUT, PATCH e DELETE

- **P2: Comunicações stateless.**
  - Os pedidos HTTP são independentes e podem ocorrer em qualquer ordem
  - Cada pedido deverá ser uma operação atómica
  - Os únicos dados guardados são os dos recursos
  - O estado da sessão eventualmente é armazenado no lado do cliente.



Princípios REST

- **P3: Tudo é Endereçável via um URI (Uniform Resource Indentifier)**
  - -Um recurso tem um identificador único
  - -Exemplo: `https://adventure-works.com/encomendas/1`
- **P4: Orientação à representação de recursos**
  - -Toda a informação e funcionalidade no lado servidor é vista como um recurso
  - -Um recurso é qualquer tipo objeto de informação ou serviço que pode ser acedido pelo cliente
  - -**Um recurso é representado na forma** de docs XML/JSON ou outro tipo MediaType
  - -Exemplo:

```
{"orderId":1,"orderValue":99.90,"productId":1,"quantity":1}
```

Tudo é endereçavel
Estrutura de um URI

- Antes da codificação deve-se planear sobre como é que o serviço/ API será exposto (i.e., qual é a estrutura do URI do recurso e das operações suportadas sobre esse URI)
- Estrutura de um URI
  - Protocolo + Hostname do servidor:porta + aplicação (opcional) + Path Coleção Recursos + ID
- Exemplo de um URI

http://www.apple.com/projetoapp/smartphones/iphones/xs

smartphones= colecção de recursos (context root)

iphones= item ou recurso da coleção

xs= ID ou chave primária (é um parametro)

Organização da API em torno dos recursos
Estrutura de um URI

- Convenções para os nomes nos URIs
  - Nomes no plural devem referenciar coleções de itens
  - Exemplo: **`/clientes`** é o caminho para a coleção de recursos
  - Enquanto: **`/clientes/5`** é o caminho para o cliente com ID = 5. Neste caso a definição do path será: **`/clientes/{id}`**
  - Exemplo a evitar:
    - **`https://adventure-works.com/clientes // BOM`**
    - **`https://adventure-works.com/criar-cliente // EVITAR`**

Organização da API em torno dos recursos
Estrutura de um URI

- Os URIs podem reflectir as associações entre recursos
  - Exemplo 1: **`/clientes/5/encomendas`** pode representar todas as encomendas para o cliente com o ID = 5.
  - Exemplo 2 (ordem contrária): **`/encomendas/99/cliente`** pode representar o cliente da encomenda 99
  - Exemplo 3 (mais complexo): **`/clientes/1/encomendas/99/produtos`** podem representar os produtos da encomenda 99 do cliente 1

Definição das Operações em termos de métodos HTTP

Operações CRUD

- Exemplos (Correctos vs Incorrectos)

Ação	Uso Correcto	Incorrecto
Consultar lista de clientes	`GET /clientes`	`POST|GET /clientes/listar`
Consultar detalhes do cliente 123	`GET /clientes/123`	`POST /detalhes-cliente {"id_cliente":"123"}`
Criar nova Encomenda	`POST /clientes/123/encomendas {"id_encomenda":99,…}`	`POST /cria-nova-encomenda {"id_cliente":"123", "id_encomenda":99,…}`
Atualiza Encomenda 99 do cliente 123	`PUT /clientes/123/encomendas/99 {"id_encomenda":99,…}`	`POST /encomendas/99/update {"id_cliente":"123", "id_encomenda":99,…}`
Cancelar Encomenda	`DELETE /clientes/123/encomendas/99`	`POST /delete-encomenda {"id_encomenda":99,…}`

Organização da API em torno dos recursos
Estrutura de um URI – cenário non resource

- Os URIs podem não refletir o acesso a um recurso
- Podemos criar um URI para invocar uma função e obter o retorno numa mensagem HTTP de resposta
- Por exemplo, uma API que implementa as operações de uma simples calculadora, como adição e subtração, e fornece dois URIs para expor estas operações como pseudo-recursos e usa uma query string para especificar os parâmetros requeridos:

  -Exemplo URI operação soma:

  •/soma?operando1=99&operando2=1 como retorno devemos obter uma mensagem HTTP com o valor 100.

  -Exemplo URI operação subtração

  •/subtraccao?operando1=99&operando2=1 como retorno devemos obter uma mensagem HTTP com o valor 98.

Definição das Operações em termos de métodos HTTP

Operações CRUD

- A API REST pode usar um ou mais dos quatro métodos HTTP seguintes:
  - **GET** - Por convenção, um pedido GET é usado para **obter** a representação de um recurso especificado no URI. O corpo da mensagem de resposta deve conter os detalhes do recurso requerido
  - **POST** - Por convenção, um pedido POST é usado para **criar** um novo recurso no URI especificado. O body da mensagem POST deve fornecer os detalhes do novo recurso.
  - **PUT** - Por convenção, um pedido PUT é usado para **alterar/atualizar** um recurso existente. O body da mensagem PUT especifica o recurso a alterar.
  - **DELETE** - Por convenção, um pedido DELETE é usado para **apagar** um recurso existente especificado no URI

Definição das Operações em termos de métodos HTTP
Operações CRUD

- A API REST pode usar um ou mais dos quatro métodos HTTP (resumo)

Exemplo:

Recurso	POST	GET	PUT	DELETE
`/clientes`	Cria um novo cliente	Retorna todos os clientes	N/A	Remove todos os clientes
`/clientes/1`	N/A	Retorna os detalhes do cliente 1	Atualiza os detalhes do cliente 1 caso exista (senão cria-o)	Remove o cliente 1
`/clientes/1/encomendas`	Cria uma nova encomenda para o cliente 1	Retorna todas as encomendas para o cliente 1	N/A	Remove todas a encomendas do cliente 1

Recursos com múltiplas representações
Internet Media Types (Originalmente MIME Types)

- Os clientes e servidores REST trocam representações de recursos.
- No HTTP os formatos são especificados pelo uso de diferentes Internet Media Types
- Um Internet Media Type identifica o tipo de representação no corpo de uma mensagem HTTP

- **Os 5 Tipos de média de Topo**
  - Text, Image, Audio, Video, **Application**

- **Subtipos Preferidos Dados não binários**
  - application/xml – Troca de dados usando o formato XML.
  - application/json - Troca de dados usando o formato JSON.
- **Outros**
  - Text/Plain – Formato por omissão; o conteúdo esta no formato texto.
  - Text/HTML – o conteúdo é uma página HTML.
  - image/jpeg

Recursos com múltiplas representações
Internet Media Types (Originalmente MIME Types)

- A API REST a criar deve estar sempre em conformidade com a especificação HTTP
- São os cabeçalhos HTTP que gerem a negociação dos tipos de média usando os cabeçalhos **content-type** (tipos no body) **e accept** (tipos aceites na resposta)
- **Content-type** num pedido ou resposta HTTP especifica o formato de representação no Body. Por exemplo:

```
POST https://adventure-works.com/encomendas HTTP/1.1
Content-Type: application/json; charset=utf-8
Content-Length: 57

{"Id":1,"Name":"Gizmo","Category":"Widgets","Price":1.99}
```

- Um pedido cliente pode incluir um cabeçalho **Accept** contendo uma lista de tipos Internet que o cliente aceitará na mensagem de resposta. Por exemplo:

```
GET https://adventure-works.com/encomendas/2 HTTP/1.1
Accept: application/json
```

Definição das Operações em termos de métodos HTTP
Códigos de Status Http

- **Resumo dos Códigos de Status HTTP (API REST)**
  - Método GET
    - Um método GET bem-sucedido a API retorna o código de status **HTTP 200 (OK).**
    - Se o recurso não puder ser encontrado, o método deve retornar **HTTP 404 (Não encontrado).**
  - Métodos POST
    - **Se o POST cria um novo recurso**, a API retorna o código **HTTP 201 (Criado). O corpo da resposta deve conter uma representação do novo recurso.**
    - Se o cliente coloca dados inválidos no pedido, o servidor retorna o código de status **HTTP 400 (requisição incorreta)**.

Definição das Operações em termos de métodos HTTP
Códigos de Status Http

- **Resumo dos Códigos de Status HTTP (Cont...)**
  - Métodos PUT
    - Se o PUT cria um novo recurso, a API retorna o código **HTTP 201 (Criado).**
    - Se o PUT atualiza um recurso existente, retorna **200 (OK)** + a representação da entidade **ou 204 (sem conteúdo).**
    - Se não for possível atualizar, devolve **HTTP 409 (Conflito)**
  - Métodos DELETE
    - Se a operação delete é bem sucedida, o servidor deve responder com um **HTTP status code 204**, sem mais informação no body.
    - Se o recurso a eliminar não existe, deve ser devolvido **HTTP 404 (Not Found).**

Definição das Operações em termos de métodos HTTP

Códigos de Status Http

- **Resumo dos Códigos de Status HTTP (Cont...) – TOP 10 Códigos**
  - Mais Info em: https://www.restapitutorial.com/httpstatuscodes.html ou em https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design

HTTP Status Codes

This page is created from HTTP status code information found at ietf.org and Wikipedia. Click on the **category heading** or the **status code** link to read more.

1xx Informational

100 Continue
101 Switching Protocols
102 Processing (WebDAV)

2xx Success

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

3xx Redirection

300 Multiple Choices
301 Moved Permanently
302 Found
303 See Other
★ 304 Not Modified
305 Use Proxy
306 (Unused)
307 Temporary Redirect
308 Permanent Redirect (experimental)

4xx 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 Request Entity Too Large
414 Request-URI Too Long
415 Unsupported Media Type
416 Requested Range Not Satisfiable
417 Expectation Failed
418 I'm a teapot (RFC 2324)
420 Enhance Your Calm (Twitter)
422 Unprocessable Entity (WebDAV)
423 Locked (WebDAV)
424 Failed Dependency (WebDAV)
425 Reserved for WebDAV
426 Upgrade Required
428 Precondition Required
429 Too Many Requests
431 Request Header Fields Too Large
444 No Response (Nginx)
449 Retry With (Microsoft)
450 Blocked by Windows Parental Controls (Microsoft)
451 Unavailable For Legal Reasons
499 Client Closed Request (Nginx)

5xx 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 (Experimental)
507 Insufficient Storage (WebDAV)
508 Loop Detected (WebDAV)
509 Bandwidth Limit Exceeded (Apache)
510 Not Extended
511 Network Authentication Required
598 Network read timeout error
599 Network connect timeout error

★ **"Top 10"** HTTP Status Code. More REST service-specific information is contained in the entry.

Definição das Operações em termos de métodos HTTP

Códigos de Status Http

- Exemplos de Códigos de Status HTTP Recomendados para o Exemplo dos Clientes

Recurso	POST	GET	PUT	DELETE
`/clientes`	**201 (Created),** Cria um novo cliente. **Nota:** Por convenção retorna o cliente submetido no body da resposta.	**200 (OK),** retorna a lista de todos os clientes	**404(Not Found).** **200 (OK)** se a implementação permitir atualizar em massa (bulk) a coleção de todos os clientes – usar com cautela.	**404(Not Found).** **200 (OK)** se queira remover todos os clientes – usar com cautela.
`/clientes/{id}`	**404 (Not Found).** não encontrado ou pedido inválido	**200 (OK),** Retorna os detalhes do cliente {id} **404 (Not Found)** se ID not found ou inválido.	**200(OK)** + retorno da entidade no body **ou** **204(No content)** se atualiza os detalhes do cliente {id} caso exista, senão cria-o e retorna **201(Created)** tal como o POST	**204 (No content),** Remove o cliente {id} ou **404 (Not Found)** se ID not found ou inválido.

Estratégias de Controlo de Versões de uma API REST

- É muito improvável que uma API REST permaneça estática.
- A atualização de uma API no lado servidor para lidar com novos requisitos é um processo relativamente simples, contudo pode ter efeitos devastadores nas aplicações cliente que consomem a API.
- Deve-se permitir que as aplicações cliente existentes continuem funcionando, e simultaneamente que novas aplicações cliente tirem proveito das novas funções e recursos.
- O controlo de versões permite que uma API indique os recursos e as funções que expõe.
- **(1) Controle da versão no URI ou (2) na String de Consulta**

GET https://adventure-works.com/v2/clientes/3 HTTP/1.1

HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

```
{"id":3,"name":"Contoso LLC","dateCreated":"2014-09-04T12:11:38.0376089Z","address":{"streetAddress":"1
Microsoft Way","city":"Redmond","state":"WA","zipCode":98053}}
```

Passamos a usar um segmento no URI expondo a versão da API

GET https://adventure-works.com/cliente/3?versao=2 HTTP/1.1

HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

```
{"id":3,"name":"Contoso LLC","dateCreated":"2014-09-04T12:11:38.0376089Z","address":{"streetAddress":"1
Microsoft Way","city":"Redmond","state":"WA","zipCode":98053}}
```

Passamos a usar um novo parâmetro version omitido na versão original

Estratégias de Controlo de Versões de uma API REST (2)

- **(3) Controle da versão usando o Cabeçalho HTTP**

```
GET https://adventure-works.com/clientes/3 HTTP/1.1
Custom-Header: api-version=2

HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{"id":3,"name":"Contoso LLC","dateCreated":"2014-09-04T12:11:38.0376089Z","address":{"streetAddress":"1
Microsoft Way","city":"Redmond","state":"WA","zipCode":98053}}
```

Passamos a usar o cabeçalho HTTP **Custom-Header**

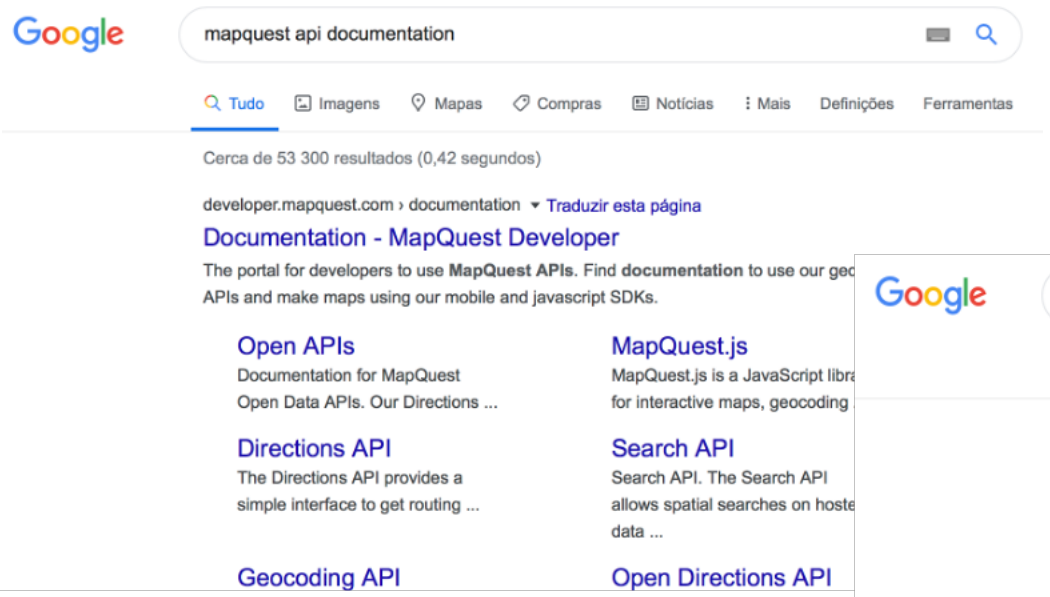
Documentação das APIs REST
Iniciativa Open API

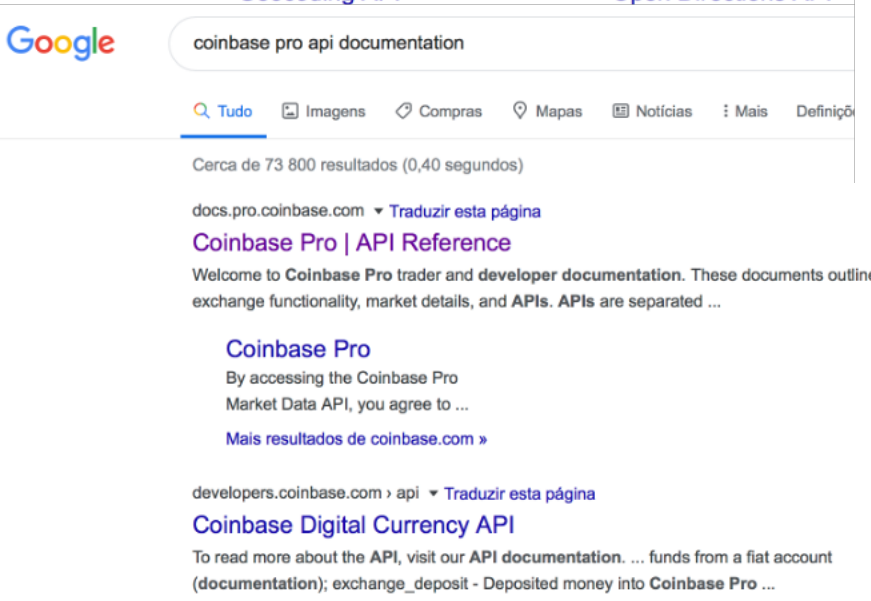
- É um consórcio criado para normalizar as descrições das APIs REST.
- A principal especificação é OpenAPI Specification (OAS), também chamada Swagger 2.0 specification
- Swagger (OpenAPI) é uma especificação independente da linguagem para descrever as APIs REST.
- Permite que computadores e humanos possam compreender a API sem acesso ao código fonte, com os seguintes objetivos:
  - Minimizar a quantidade de trabalho necessária para ligar a serviços dissociados.
  - Reduzir a quantidade de tempo necessária para documentar com precisão um serviço.

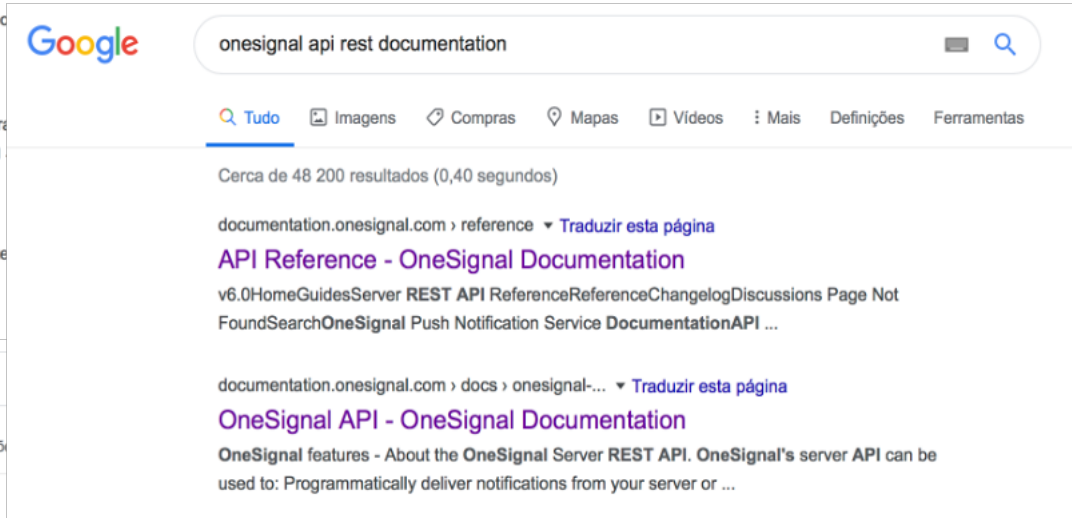
Documentação das APIs REST
Iniciativa Open API – documento OpenAPI specification (openapi.json)


```
{
  "openapi": "3.0.1",
  "info": {
    "title": "API V1",
    "version": "v1"
  },
  "paths": {
    "/api/Todo": {
      "get": {
        "tags": [
          "Todo"
        ],
        "operationId": "ApiTodoGet",
        "responses": {
          "200": {
            "description": "Success",
            "content": {
              "text/plain": {
                "schema": {
                  "type": "array",
                  "items": {
                    "$ref": "#/components/schemas/ToDoItem"
                  }
                }
              },
              "application/json": {
                "schema": {
                  "type": "array",
                  "items": {
                    "$ref": "#/components/schemas/ToDoItem"
                  }
                }
              },
              "text/json": {
                "schema": {
                  "type": "array",
                  "items": {
                    "$ref": "#/components/schemas/ToDoItem"
                  }
                }
              }
            }
          }
        }
      },
      "post": {
        ...
      }
    },
    "/api/Todo/{id}": {
      "get": {
        ...
      },
      "put": {
        ...
      },
      "delete": {
        ...
      }
    }
  },
  "components": {
    "schemas": {
      "ToDoItem": {
        "type": "object",
        "properties": {
          "id": {
            "type": "integer",
            "format": "int32"
          },
          "name": {
            "type": "string",
            "nullable": true
          },
          "isCompleted": {
            "type": "boolean"
          }
        },
        "additionalProperties": false
      }
    }
  }
}
```

Documentação de uma API (exemplos)

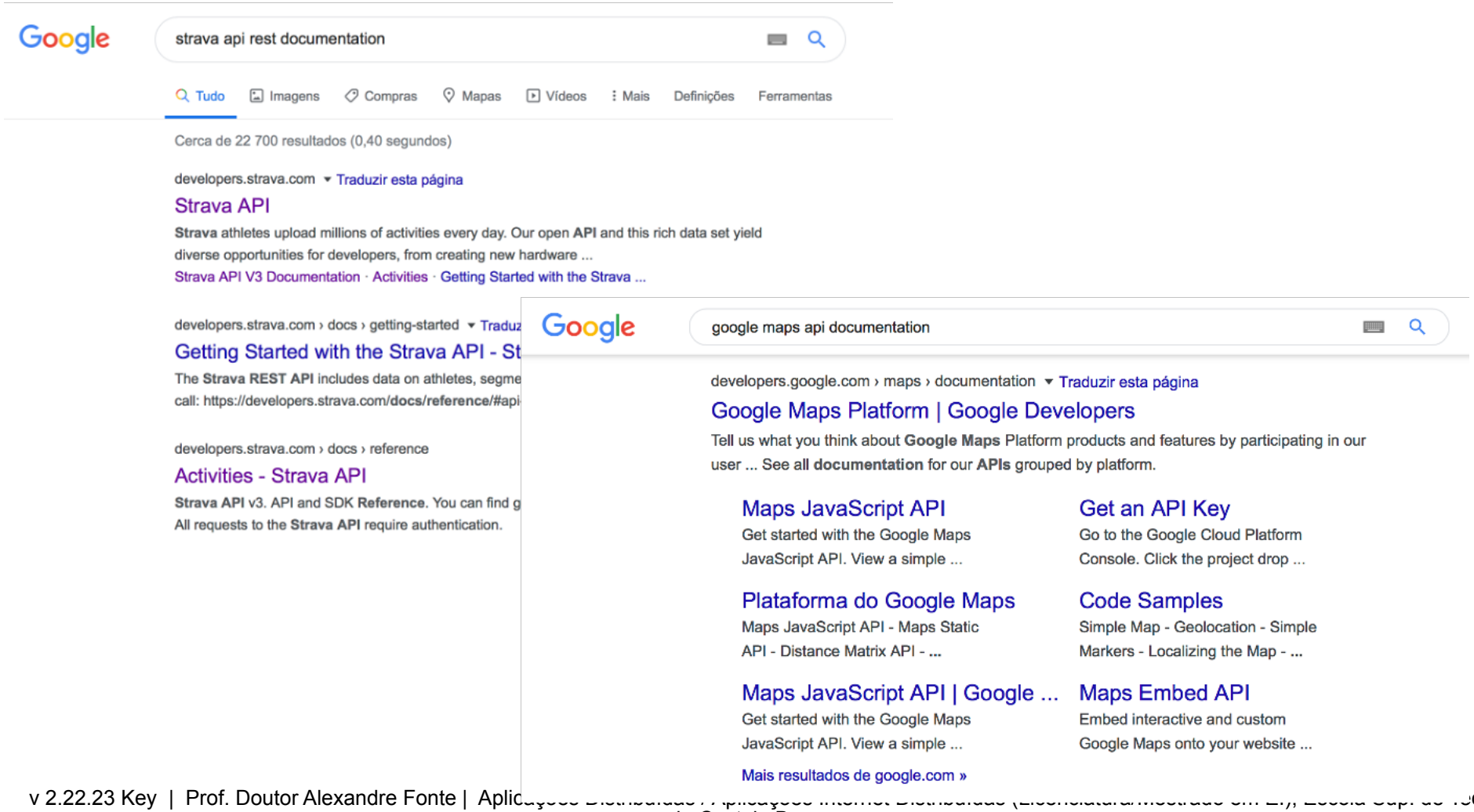






Documentação de uma API (exemplos)

- Normalmente procura-se na Internet:



Questões