

# Guia de Referência Rápida

# REST APIs

Visão básica dos principais conceitos de REST APIs

## Modelagem

## Versionamento

Bounded Context

Domain Driven Design

Serviços

Entidades

verbos

substantivos

id

HTTP

request

response

headers

URL

path parameters

query parameters

2xx

3xx

4xx

5xx

status codes

verbos

GET

POST

PUT

PATCH

DELETE

HATEOAS

Segurança

OpenId Connect

OAuth 2.0

# Por que REST API?

Guia de Referência Rápida

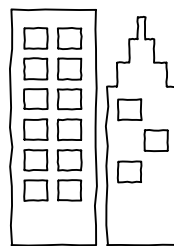
## REST API

Assim como o inglês é tido como uma língua mais "padrão", o REST é tido como uma forma de integração mais "padrão" do que as outras. Não significa que seja a melhor para todos os cenários, mas é a mais popular, logo, amplamente suportada e conhecida pelo mercado, linguagens e frameworks.



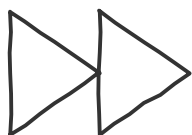
### Multi dispositivos e multicanal

O desenvolvedor escreve o código uma única vez e ele pode ser usado por vários dispositivos diferentes.



### Expõe seu negócio para parceiros

Por se tratar de tecnologia padrão de mercado, permite conexão com parceiros para criação de novos modelos de negócio e inovação.



### Agilidade

Negócios mudam mais devagar do que seus clientes e canais de venda. APIs permitem tirar lógica de negócio dos canais de venda tornando-os mais ágeis para absorver as mudanças que o mercado impõe.



### Facilmente encontrável

Analistas em novos projetos conseguem encontrar as funcionalidades já disponíveis via APIs e sair usando.



### Popularidade

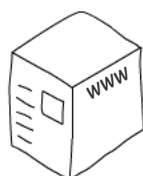
O site [programmableweb.com](http://programmableweb.com), em 2019, já listava mais de 22000 APIs públicas



### Abstrai complexidade

Basta conectar um equipamento à tomada e ele funciona sem que se conheça detalhes da geração e transmissão da rede elétrica até a tomada.

Da mesma forma, APIs abstraem a complexidade dos sistemas e expõe suas funcionalidades em uma interface simples e conhecida.



### É web

Por ser baseada no HTTP, permite reaproveitar o poder da infraestrutura web já presente nas empresas.

# O que é?

É uma API que expõe estados de recursos e é implementada segundo um estilo arquitetural baseado no protocolo HTTP.

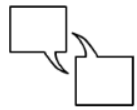
Guia de Referência Rápida

## REST API

### Application Programming Interface (API)



É uma interface para troca de dados entre sistemas.



Tem um contrato muito bem definido entre o provedor e o consumidor.

### Javascript Object Notation (JSON)

É o formato mais comum para trocar informações via API.

```
{
  "nome": "José",
  "idade": 34,
  "altura": 1.53,
  "vip": true,
  "endereco": "null"
}
```

### Hyper Text Transfer Protocol (HTTP)

É o protocolo da web que é composto de:

- método (verbs) que indicam a ação a ser feita;
- endereço (URL) de um recurso na rede;
- parâmetros de entrada (query strings)
- mensagem de request;
- mensagem de response;
- headers com informações de controle;
- body com mensagem de response.

As REST APIs são construídas utilizando-se dos mesmos elementos do HTTP: expõem as informações em URLs, filtra com query strings, manipula-as através dos verbos etc.

### Quer ver na prática?

Identifique as partes do protocolo em uma página no Chrome. Aperte Ctrl+Shift+I

The screenshot illustrates the components of an HTTP request and response in a web browser. The address bar shows the URL, which includes the domain, path, and query parameters. The Network tab in the developer tools shows a list of requests, with the first one selected. The 'Headers' pane shows the request details, including the status code (200), headers, and the body of the response. Arrows point to various parts of the interface: 'URLs' points to the address bar, 'verbs' points to the HTTP method (GET), 'status codes' points to the status bar, 'parameters' points to the query string, 'headers' points to the Headers pane, and 'body' points to the Body pane.

### Histórico



# Os 5 Princípios

O estilo arquitetural REST é definido por basicamente 5 princípios.

Guia de Referência Rápida

## REST API

### Cliente / Servidor



#### Cliente

Preocupa-se com User Experience, Interface e múltiplos dispositivos.



#### Servidor

Preocupa-se com performance, autenticação, autorização, escala, cache, armazenamento e segurança.

Servidor é desacoplado do cliente.

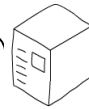
Não compartilham nenhum código no mesmo processo.

Podem evoluir em paralelo

### Interface Uniforme



Contrato



#### Recursos

HTTP/1.1 GET <http://api.viagens.com/v1/destinos/14/opinioes>

#### Representação dos recursos

{json} <XML/>

#### Mensagens auto-descritivas: metadados

HTTP Status Code, content-type, accept, host

#### Hypermedia

Dados + Ações (Hypermedia As The Engine Of Application State)

### Statelessness



Não se armazena estado, sessão, nenhum dado sobre uma requisição que já aconteceu.



Cada requisição vinda do cliente é independente da anterior.



O cliente envia na requisição toda a informação que o servidor necessita para processar a requisição.

### Caching



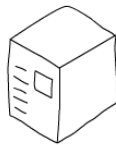
Cache local



Cache local



Cache compartilhado



Cache compartilhado

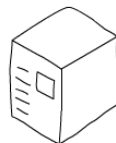
Compensa a perda de performance do Statelessness, diminui a quantidade de requisições e reduz tráfego na rede.

É controlado via headers cache-control, expires, last-modified, Etag etc.

### Sistema de Camadas



API Gateway



Cada camada só tem acesso na próxima.

Camadas podem ser adicionadas, removidas ou modificadas conforme necessidade.

# Recursos

São representações de coisas que existem no mundo real.

Guia de Referência Rápida

## REST API

### Estado de um recurso

É o conjunto de atributos cujos valores representam o estado do recurso.

A proposta do REST é permitir as transferências, ou seja, mudanças de estado dos recursos.

### Exposição dos recursos

Recursos são expostos e representados via URL. As URLs devem ser definidas seguindo algumas boas práticas.

### Alteração de estado



Ex: Um veículo qualquer, se diferencia de outro por conta do conjunto de suas características e normalmente uma delas é o identificador, pois mesmo que as outras características mudem, o identificador nunca muda.

Alteração de estado acontece quando uma ou mais características mudam.

```
{
  "numeroDoChassi": "9BG114LW04C0001",
  "ano": 2014,
  "marca": "GM",
  "modelo": "Onix",
  "proprietario": "José da Silva"
}
```

Maria Fernanda

Hum ... mudou o estado!



## Exemplos

`http://api.walmartlabs.com/v1/items/12417832`

base path      recursos

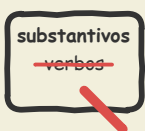
`https://api.instagram.com/v1/users/{user-id}/relationship`

base path      recursos

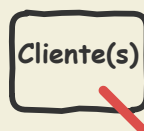
`https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.DBforMySQL/servers/{serverName}/databases/{databaseName}`

base path      recursos

## Boas práticas



Use substantivos, não verbos  
GET /clientes não /listarClientes



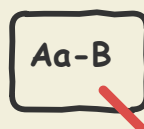
Use plurais  
➢ Todos os clientes: /clientes  
➢ Cliente específico: /clientes/007



Versione seus contratos na URL  
GET /v1/pagamentos



Organize seus recursos em estruturas hierárquicas  
Ex: ordens contém produtos  
GET /ordens/18953/produtos/1



Mantenha o case consistente  
Escolha camelCase, PascalCase, kebab-case ou snake\_case para os atributos e parâmetros.  
Prefira kebab-case nas URLs (alguns servidores ignoram o case)



Tenha um domínio separado do site  
https://api.twitter.com  
https://api.walmartlabs.com



# Verbos

Vêm do HTTP e indicam ao servidor qual ação (CRUD) deve ser aplicada em um determinado recurso representado por uma URL.

Guia de Referência Rápida

## REST API



Podem ser **seguros**, quando são provacam mudança de estado no servidor.



Podem ser **idempotentes**, quando "n" requisições iguais provocam o mesmo resultado no estado do recurso.

### POST

Inseguro  
Não idempotente  
Não usa cache

Solicita a criação de uma nova instância de um recurso.

Request:  
POST http://api.com/usuarios  
{“nome”: “Alguém”,  
“e-mail”: “meuemail@email.com”}

Response:  
HTTP 201 - Created  
Location: /usuarios/abc123

### GET

Seguro  
Idempotente  
Pode usar cache

Solicita a representação de uma ou mais instâncias do recurso.

Request:  
GET /usuarios/abc123

Response:  
HTTP 200 - Ok  
{“id”: “abc123”, “nome”: “Alguém”,  
“e-mail”: “meuemail@email.com”}

#### Pouco usados

**OPTIONS** para listar os verbos possíveis para um recurso

**HEAD** para retornar apenas os cabeçalhos

**TRACE** para responder de volta o que o servidor recebeu

Inseguro  
Idempotente  
Não usa cache

### PUT

Modifica completamente a instância de um recurso, fazendo substituição.

Request:  
PUT /usuarios/abc123  
{“nome”: “Alguém mudou de nome”,  
“e-mail”: “meu-NOVO-email@email.com”}

Response:  
HTTP 200 - ok

Caso a instância não exista, ele cria.

Inseguro  
Idempotente  
Não usa cache

### PATCH

Modifica parcialmente um recurso.

Request:  
PATCH /usuarios/abc123  
{“nome”: “Alguém mudou só o nome”}

Response:  
HTTP 200 - ok

Inseguro  
Idempotente  
Não usa cache

### DELETE

Apaga uma ou mais instâncias do recurso.

Request:  
DELETE http://api.com/usuarios/abc123

Response:  
HTTP 204 - No Content

### Cenários "não CRUD"

Em alguns casos sem característica de CRUD, os recursos representarão uma ação e usa-se POST ou GET conforme característica de segurança e idempotência. Ex:

POST /email/123/enviar  
GET /somar?a=18&b=77

# Códigos de retorno

Vêm do HTTP e indicam o sucesso ou insucesso no processamento da requisição enviada ao servidor.

Guia de Referência Rápida

## REST API

Existem muitos códigos de retorno (status codes) no HTTP. Para o REST, alguns são amplamente adotados, outros menos. Seguem alguns dos principais:

### 1xx - Informação

Tem pouco ou nenhum uso com REST.

### 2xx - Sucesso

200 Ok	Requisição processada com sucesso
201 Created	Uma nova instância foi criada
202 Accepted	O recurso será atualizado/criado de forma assíncrona
204 No Content	A requisição foi processada com sucesso e não há body na resposta
206 Partial Content	O servidor encontrou uma grande quantidade de registros na resposta e devolveu de forma paginada

### 3xx - Redirecionamento

301 Moved Permanently	O recurso foi definitivamente movido para uma outra URL definida no header Location
303 See Other	O recurso foi definitivamente movido para uma outra URL definida no header Location
307 Temporary redirect	Redirecionamento de uma página para outro endereço, porém que é com caráter temporário. Provavelmente por conta de alguma manutenção no sistema

### 4xx - Erro do cliente

400 Bad Request	Servidor não consegue entender a requisição, pois existe algum parâmetro inválido ou falta dele
401 Unauthorized	Credenciais inválidas
403 Forbidden	Credenciais válidas, mas sem acesso no recurso
404 Not Found	O servidor não encontrou o recurso solicitado pelo cliente através da URL
410 Gone	O recurso (URL) não existe mais e esta condição é permanente
422 Unprocessable Entity	A requisição está correta ao nível de parâmetros, mas existem erros ao nível de negócio que impedem o processamento da requisição

### 5xx - Erro do servidor

500 Internal Server Error	Erro mais genérico para informar que o servidor encontrou um cenário inesperado de erro que não soube conseguir processar a requisição
503 Service Unavailable	O servidor não está respondendo porque está fora do ar, em manutenção ou sobrecarregado. É um problema temporário
504 Gateway Timeout	O servidor, enquanto atuando como gateway ou proxy, não conseguiu responder em tempo

### Mais informações sobre o retorno

Para os casos de sucesso, o retorno da requisição é o próprio recurso solicitado.

Para os casos de erro, é importante devolver um conteúdo, preferencialmente padronizado, com mais detalhes sobre o problema.

```
{
  "code": "10023",
  "message": "Alguns campos estão preenchidos incorretamente.",
  "details": "https://developer.empresa.com/apis/erros/10023",
  "fields": [
    {
      "name": "dataPedido",
      "message": "O formato de data é inválido. Utilize data no padrão yyyy-MM-DD.",
      "value": "01-05-2019",
      "details": "https://developer.empresa.com/apis/erros/err-gen-086"
    },
    {
      "name": "valorPagamento",
      "message": "Valor não é number.",
      "value": "R$ 27.568,90",
      "details": "https://developer.empresa.com/apis/erros/err-gen-073"
    }
  ]
}
```

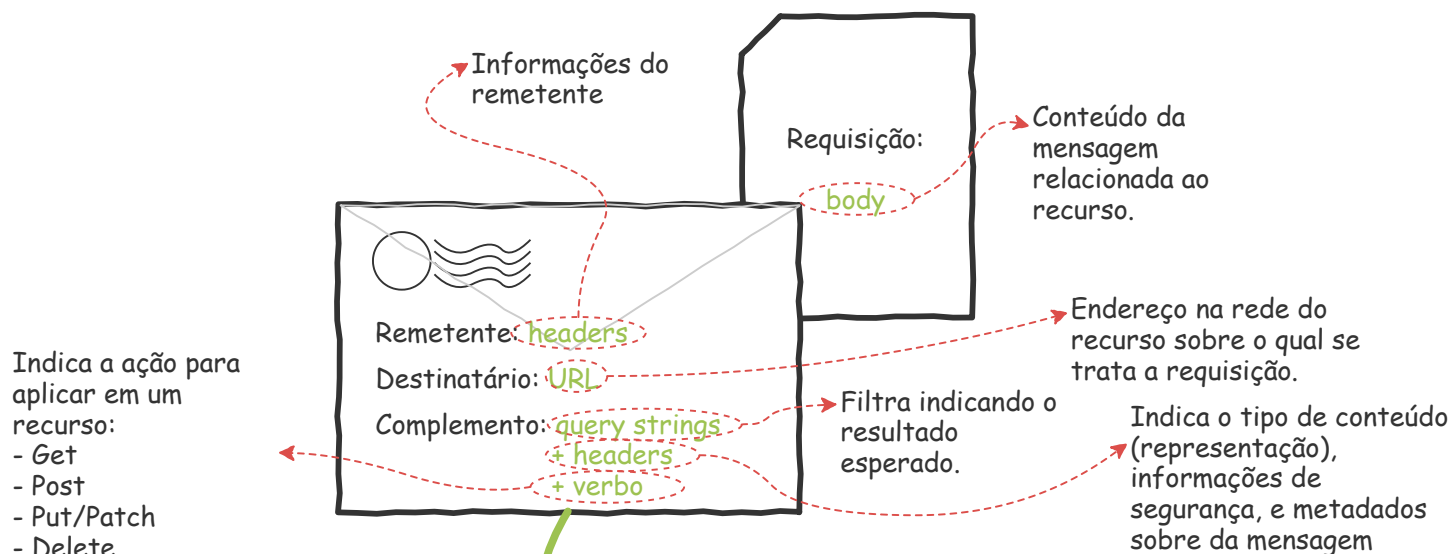
# Fluxo da requisição

Uma requisição e resposta em REST segue basicamente o funcionamento do protocolo HTTP.

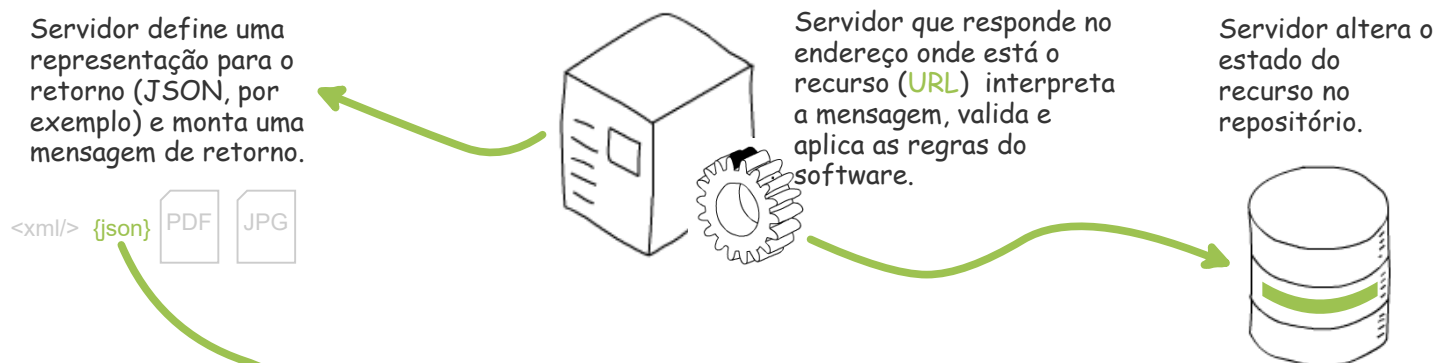
Guia de Referência Rápida

## REST API

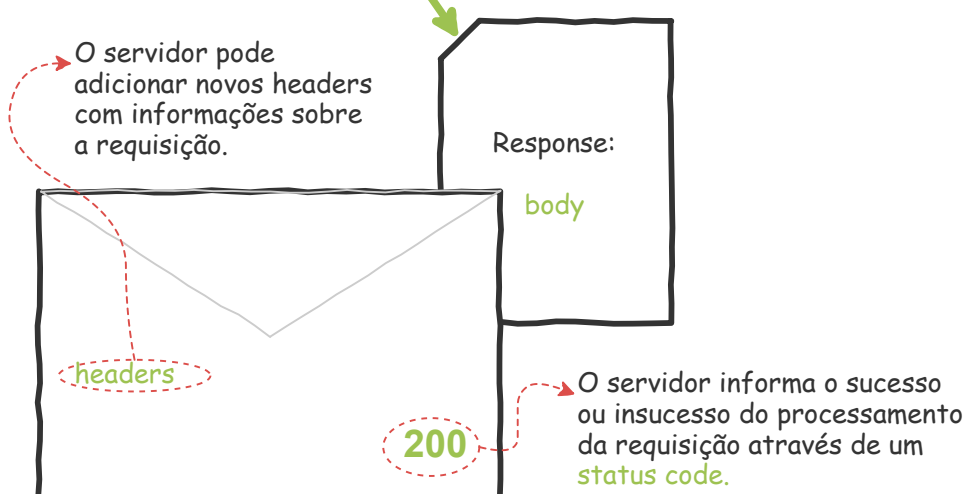
### Request



### Processamento



### Response





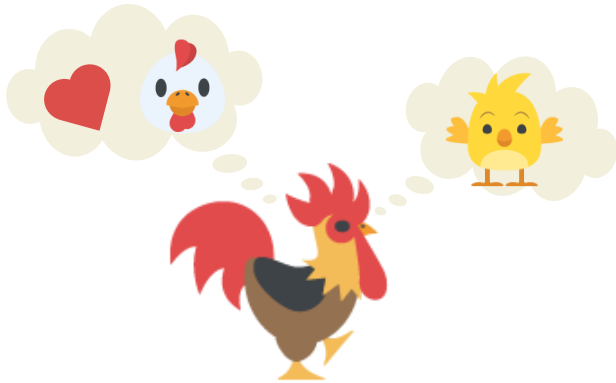
# Requisições assíncronas

Algumas mudanças de estado dos recursos podem não ser processadas imediatamente. Para isso, um fluxo para acompanhar o processamento se faz necessário.

Guia de Referência Rápida

## REST API

### 1 Requisição para um recurso que demora para ser processado



Request:  
POST /pintinhos  
{  
 "nome": "Amarelinho",  
 "papai": "Sr. Crista",  
 "mamae": "Dona Carijó"  
}

### 2 Resposta indicando o encaminhamento para um novo recurso



Response:  
HTTP 202 Accepted  
Location: /ovinhos/59637

### 6 Consulta ao novo recurso para verificar o andamento

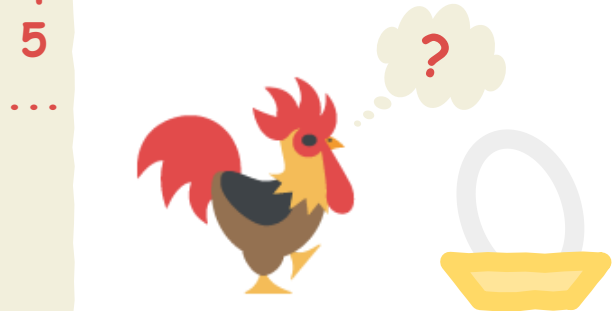


Request:  
GET /ovinhos/59637

Response:  
HTTP 303 See Other  
{  
 "id": 59637,  
 "situacao": "nasceu",  
 "temperatura": null,  
 "TTC": null,  
 "mensagens": []  
}  
Location: /ovinhos/59637  
Content-Location: /pintinhos/2368

! O fluxo descrito se chama **pooling**, alternativamente, ao solicitar a criação do "Amarelinho", o "Sr. Crista" poderia informar seu endereço e quando o "Amarelinho" nascesse, ele receberia a informação o "Amarelinho" diretamente no seu endereço. Este processo é o que chamamos de Call Back.

### 3 4 5 Consultas ao novo recurso para verificar o andamento



Request:  
GET /ovinhos/59637

Response:  
HTTP 200 ok  
{  
 "id": 59637,  
 "situacao": "chocando",  
 "temperatura": 30,  
 "TTC": 2019-07-16T19:20:30.00-03:00,  
 "mensagens": [{  
 "codigo": "103",  
 "mensagem": "Segundo o veterinário, tudo ok.",  
 "tipo": "info"  
 }]  
}

### 7 Consulta ao novo recurso criado



Request:  
GET /pintinhos/2368

Response:  
HTTP 200 ok  
{  
 "id": 2368,  
 "nome": "Amarelinho",  
 "papai": "Sr. Crista",  
 "mamae": "Dona Carijó",  
 "idade": 1  
}

# Query strings

São conjunto de 'chave=valor' presentes na URL que iniciam com o caracter '?' e são separados por '&'.

Guia de Referência Rápida

## REST API

Alguns comportamentos se repetem em praticamente todas as APIs, por isso, alguns padrões para as query strings podem ser convencionados e adotados por todos os sistemas da empresa.

Isso reduz a curva de aprendizado do cliente e permite que alguns códigos possam ser reusados.



Para pesquisas em qualquer atributo, é possível usar a "forma Google" de pesquisar.

`GET /aeroportos/GRU/restaurantes?q=vegano`

`GET /vestuarios?q=algodao`



Em um GET, é possível filtrar as instâncias a serem retornadas usando seus atributos.

`GET /usuarios?signo=sagitario&signo=escorpio&estadoCivil=solteiro`

`GET /aeroportos/GRU/restaurantes?tipo=japones&faixaPreco=$$`



Alguns recursos que retornam uma quantidade maior de instâncias podem ter seus resultados paginados.

`GET /clientes?page=2&limit=50`



Quando se pesquisa intervalos, pode-se utilizar um 'from' 'to'.

`GET /clientes?fromIdade=21&toIdade=32`

`GET /cartoes/3/faturas?fromDataVencimento=2020-10`



Para limitar a quantidade de registros, pode-se utilizar o 'top'.

`GET /ofertas-noturnas?top=3`



Ordenação pode ser feita com uma estrutura de sort.

`sort=[{atributo}:{asc|desc}]`

`GET /ofertas-noturnas?sort=porcentagemDesconto,data:desc`



Você pode devolver respostas parciais dos recursos que têm muitos atributos.

`GET /clientes/123?fields=nome,idade`  
200 OK

`{ "nome": "José", "idade": 34 }`



Para aninhar mais de uma requisição, use um 'expand'.

`GET /cartoes/123`  
+ `GET /cartoes/123/faturas/18`  
= `GET /cartoes/123?expand=faturas&faturas.id=18`  
200 OK  
`{ "id": 123,`  
`"numero": "5436 **** * 3658",`  
`"faturas": [`  
`{`  
`"id": 18,`  
`"vencimento": "2020-06-18",`  
`"valor": 2365.48`  
`}`  
`]`  
`}`

E se um determinado conjunto de atributos é sempre requisitado, pode-se definir "views".

`GET /clientes/123?view=contatos`  
200 OK

`{ "nome": "José",`  
`"email": "@",`  
`"endereço": "Rua A",`  
`"telefone": "912345678" }`

Pode usar todos juntos?

**Sim!**

# HATEOAS

Hypermedia As The Engine Of Application State é capacidade pela qual um cliente pode interagir com a API através de links informados pelo servidor.

Guia de Referência Rápida

**REST API**

## Como funciona

Em cada resposta da API, ela deve prover links que permitam que o cliente descubra todas as possibilidades de alteração do estado de um recurso à partir do estado atual.

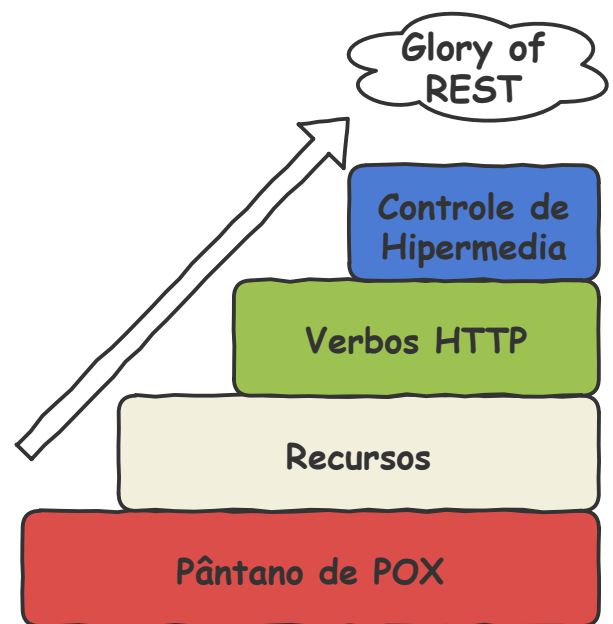
Request:

```
POST /consultas
{
  "idMedico": "antonio59",
  "inicio": "14:00",
  "fim": "14:30",
  "idPaciente": "jose85"
}
```

Response:

```
HTTP 201 Created
{
  "data": {
    "id": "123",
    "idMedico": "antonio59",
    "inicio": "14:00",
    "fim": "14:30",
    "idPaciente": "jose85"
  },
  "links": [
    {
      "rel": "self",
      "href": "/consultas/123",
      "title": "Esta consulta",
      "method": "GET"
    },
    {
      "rel": "cancelar",
      "href": "/consultas/123",
      "title": "Cancelar consulta",
      "method": "DELETE"
    },
    {
      "rel": "alterar",
      "href": "/consultas/123",
      "title": "Alterar horário desta consulta",
      "method": "PATCH"
    },
    {
      "rel": "agendamentos-medico",
      "href": "/medicos/antonio59/consultas",
      "title": "Listar consultas para o médico",
      "method": "GET"
    },
    {
      "rel": "agendamentos-paciente",
      "href": "/consultas?paciente=jose85",
      "title": "Listar consultas do paciente",
      "method": "GET"
    }
  ]
}
```

## Richardson Maturity Model



Leonard Richardson, autor de um dos primeiros grandes livros de REST criou um modelo de maturidade que representa os passos para a adoção das práticas REST.

Ele percorre desde o pântano de Plain Old XML (POX), quando não haviam padrões para troca de mensagens, passa por separar bem os recursos em URLs, usar corretamente os verbos do HTTP e códigos de resposta e por fim, o uso de HATEOAS.

Muitos dos sistemas não adotaram o uso e descoberta das APIs via HATEOAS.





# Modelagem

É o processo de entender a demanda de negócio e transformá-la em recursos expostos via URL respeitando os padrões arquiteturais e regras do REST.

Guia de Referência Rápida

## REST API

### KISS (Keep It Simple, Stupid)

-  Modele suas APIs pensando nos seus clientes (desenvolvedores), não nos seus dados.
-  Evite entregar mais de uma forma de fazer a mesma coisa.
-  Foque nos casos principais primeiro, lide com as exceções depois.
-  Mantenha padrões.

### Granularidade

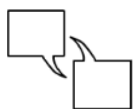
Busque no máximo 2 níveis no aninhamento de atributos.

```
{  
  "id": 123,  
  "nome": "José",  
  "dataNascimento": "1985-01-01",  
  "endereco": {  
    "logradouro": "Rua Feliz"  
    "numero": "10"  
    "complemento": "casa 2"  
    "CEP": "10000100"  
  }  
}
```

### Domain Driven Design

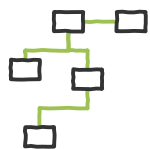
O DDD fornece um conjunto de recursos, padrões e práticas que ajudam a documentar e modelar software. Entender o negócio sobre o qual o software trata é essencial para se criar boas APIs.

#### Linguagem ubíqua



É o uso da linguagem falada que seja de conhecimento comum entre todos os envolvidos com um conceito de negócio: o jargão. Um conceito de negócio deve ter um nome único e seu significado deve ser claro para todos.

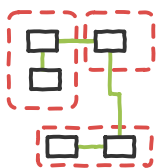
#### Domain Model



É uma representação de conceitos do mundo real (do negócio) que são pertinentes ao um ou mais domínios a serem trabalhados em um software. Estes conceitos incluem as entidades, serviços e as relações entre eles.

entidade = substantivos + ID  
serviços = verbos

#### Bounded Context



São fronteiras que separam conceitos que têm alguma relação entre si. O significado desta fronteira deve ser claro para todo o time. Um bounded context pode inspirar divisões para softwares diferentes, contratos de APIs, componentes etc.

### Passo-a-passo

#### Listar os requisitos

Ter a lista de funções do negócio que serão expostas via API.

#### Identificar as entidades

Entender com os especialistas de negócio como ele funciona e identificar os substantivos que representam as entidades.

#### Identificar os serviços

Identificar funções do negócio que não são entidades através dos verbos que aparecem na conversa: processar, faturar, reservar etc.

#### Fazer o Domain Model

Desenhar as entidades, serviços, relacionamentos e bounded contexts e alinhar o desenho com os especialistas do negócio para garantir que o entendimento de tudo foi correto.

#### Converter o Domain Model em REST

- Converter as entidades aninhando-as em URLs.
- Converter os serviços CRUD nos verbos HTTP.
- Definir os atributos de entrada e saída.
- Definir os códigos HTTP a serem retornados.

# Contrato

As URLs e os parâmetros de entrada e saída da API podem ser documentados em linguagens próprias pra isso. Esta documentação é chamada de contrato.

Guia de Referência Rápida

## REST API

### Contract First

Trabalhar em produzir um bom contrato de API antes de começar o desenvolvimento do código:

- faz com que muitas dúvidas apareçam e sejam sanadas com antecedência;
- traz um entendimento mais completo do negócio;
- evita impedimentos e retrabalho durante o desenvolvimento;
- permite que o cliente possa adiantar o desenvolvimento antes da API estar pronta.

O contrato, que contém cada item presente nas REST APIs, como URLs, path parameters, query parameters, headers etc., assim como descrições e exemplos, pode ser utilizado para:

- geração de código fonte para implementação em diversas linguagens;
- geração de documentação web publicável, navegável e que é capaz de fazer chamadas às APIs;
- configuração de API Gateways;
- configuração de testes automatizados;
- etc.

### Open API Specification

Antes conhecido como Swagger Specification, o OAS é um formato para documentar REST APIs.

A especificação permite que o documento seja escrito em JSON ou YAML.

### Mais informações

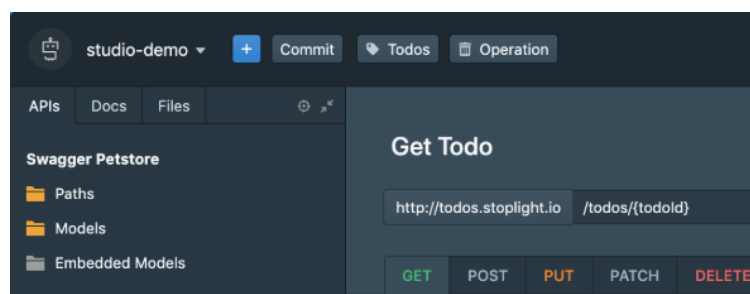
<https://swagger.io/docs/specification/about/>

<https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md>

### Exemplo

```
openapi: 3.0.0
info:
  title: Sample API
  description: Sample API documentation.
  version: 0.1.9
servers:
  - url: http://api.example.com/v1
    description: Main (production) server
  - url: http://staging-api.example.com
    description: Internal staging server for testing
paths:
  /users:
    get:
      summary: Returns a list of users.
      responses:
        '200': # status code
          description: A JSON array of user names
          content:
            application/json:
              schema:
                type: array
                items:
                  type: string
  /users/{userId}:
    get:
      summary: Returns a user by ID.
      parameters:
        - name: userId
          in: path
          required: true
          description: User Identification.
          schema:
            type: integer
            format: int64
            minimum: 1
      responses:
        '200':
          description: OK
```

### Ferramentas



- stoplight.io/studio
- editor.swagger.io
- github.com/42Crunch/vscode-openapi



# Versionamento

Mais de uma versão de API pode existir ao mesmo tempo, visando manter compatibilidade com os clientes antigos.

Guia de Referência Rápida

## REST API

"A única coisa que não muda é que tudo muda."

Heráclito, Filósofo

### Quebra de contrato

Acontece quando uma alteração na estrutura da API gera erro no cliente caso ele se mantenha chamando a API da mesma forma.

- ☒ Remoção de atributos de resposta.
- ☒ Adição de atributos, headers, query parameters etc. obrigatórios na requisição.
- ☒ Remoção de recursos (URLs).
- ☒ Alteração de atributos, headers, query parameters, URLs etc. obrigatórios ou não.

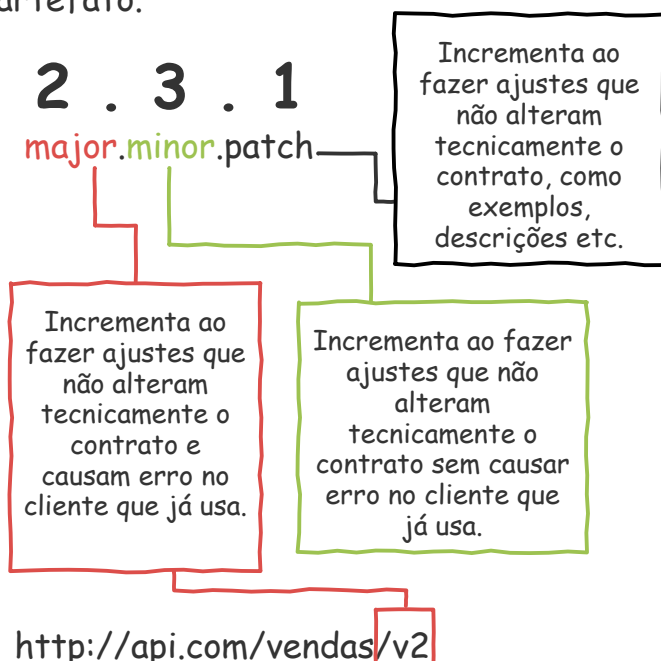
### Alteração de contrato

Acontece quando uma alteração na estrutura da API não gera erro no cliente caso ele se mantenha chamando a API da mesma forma.

- ☒ Adição de atributos de resposta.
- ☒ Adição de atributos, headers, query parameters etc. não obrigatórios na requisição.
- ☒ Adição de recursos (URLs).
- ☒ Quando se tem um documento da especificação da API, altera alguma descrição ou exemplo.

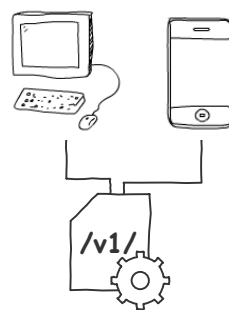
### Semantic Versioning

Ao alterar o documento do contrato da API, é interessante adotar o versionamento semântico para controlar o artefato.



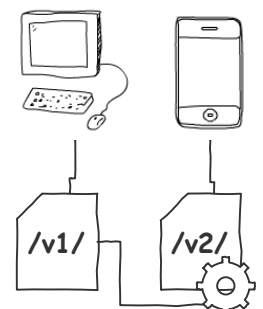
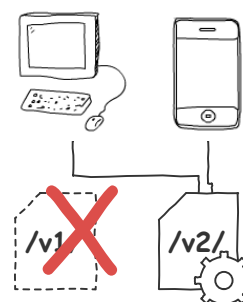
### Estratégia de migração

1 - Todos os clientes na v1



2 - A v1 faz proxy para a implementação da v2. Os clientes são solicitados para migrar para a v2.

3 - Todos os clientes migraram para a v2. A v1 é desligada.



# OpenID Connect e OAuth 2.0

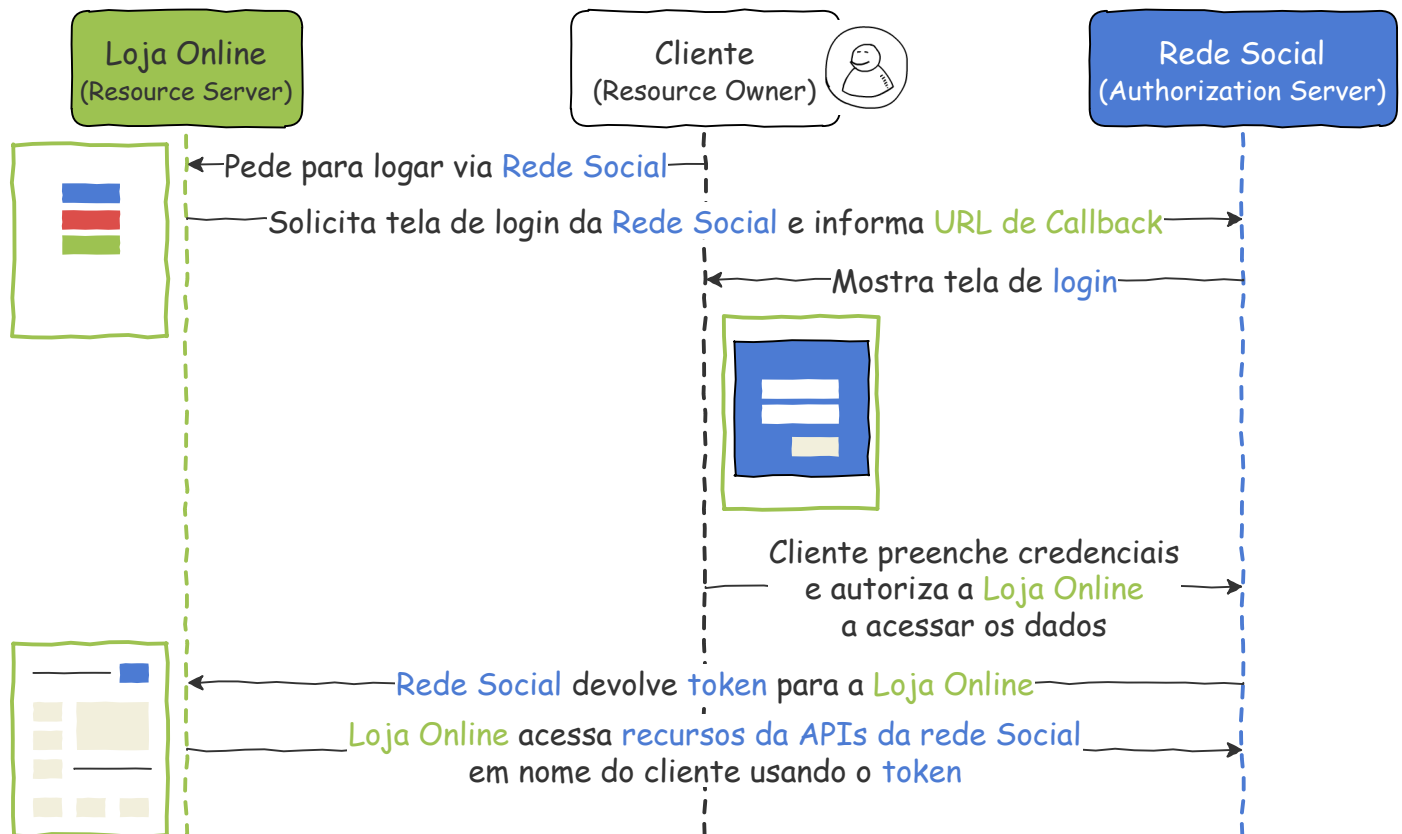
OpenID Connect é uma camada de identificação do usuário que roda sobre o OAuth 2.0. O OAuth 2.0 é um padrão para autorizar acessos sem ficar tráfegando senhas.

Guia de Referência Rápida

REST API

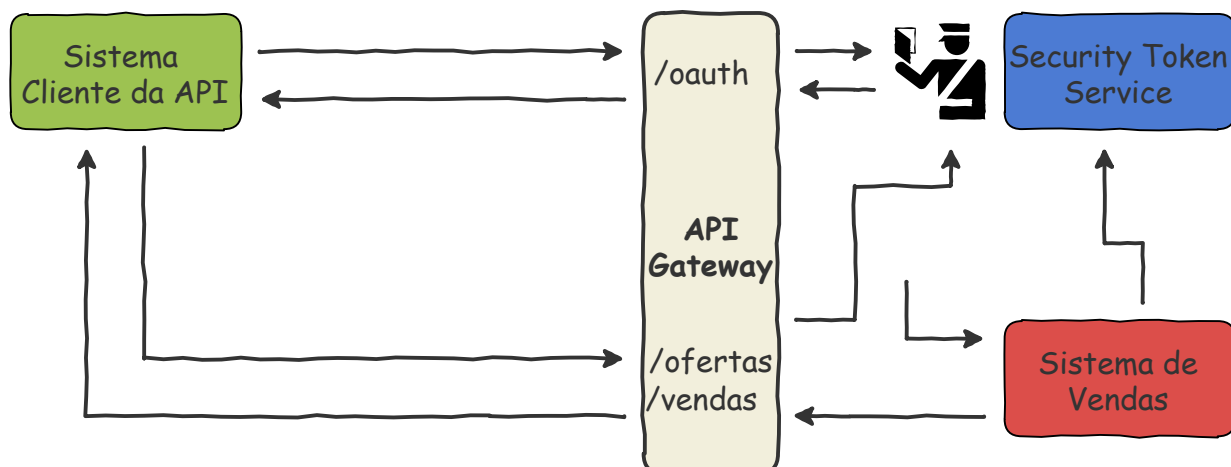
## OpenId Connect

Permite que o sistema A acesse o sistema B em nome do cliente sem que o sistema A tenha acesso às credenciais do cliente (login e senha). A requisição de acesso gera um token e este token pode ser revogado por vontade do cliente diretamente no sistema B.



## OAuth 2.0

Exemplo de fluxo Resource Owner do OAuth 2.0. O **Sistema de Vendas** não conhece as credenciais do **Sistema Cliente da API** e confia no **Security Token Service** para validar o token.



Em caso de token inválido para uma determinada REST API, os sistemas retornam um HTTP 401 Unauthorized ou 403 Forbidden.