

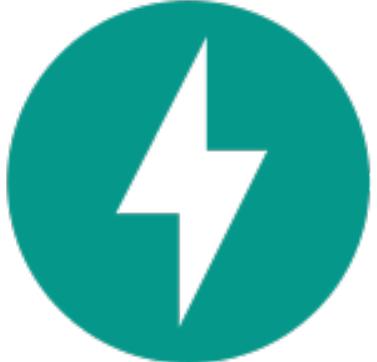
Geek University

Evolua seu lado geek!

www.geekuniversity.com.br

Conceitos Essenciais sobre APIs



 **FastAPI**

The image features the FastAPI logo. It consists of a teal circular icon on the left containing a white lightning bolt symbol, followed by the word "FastAPI" in a large, bold, teal sans-serif font.



Conceitos Essenciais sobre APIs

Muitas APIs encontradas na Internet utilizam um conceito chamado **REST**, que significa **Representational State Transfer**, ou Transferência Representacional de Estado.

Entender o que é **REST** irá ajudar a você a desenvolver melhores **APIs** para os sistemas que desenvolve.

{ REST }

Você já deve ter ouvido falar também em **RESTfull**, que é um padrão de criação de APIs amplamente utilizado, e muitas pessoas até mesmo confundem achando que **REST** é diferente de **RESTful**.

No fim das contas, estamos falando da criação de uma interface de comunicação utilizando puramente **HTTP**.



Conceitos Essenciais sobre APIs

Uma **API** nada mais é do que uma interface de comunicação de aplicações de forma programática.

Ou seja, criamos uma interface para que diferentes aplicações se comuniquem de forma simples e eficiente.

Criamos estas interfaces, chamadas **APIs** utilizando padrões de design chamado **RESTful**.

Estas **APIs** são chamadas **APIs REST**.





REST - Representational State Transfer

{ REST }

O protocolo **HTTP**, que é onde a Internet “roda” é por design, sem estado.

Isso significa que toda requisição feita a um servidor é única pois estas requisições não guardam dados (estados) entre uma requisição e outra.

É como se toda vez que você encontrasse um amigo tivesse que se apresentar para ela novamente. Pois nem você nem seu amigo guardam dados (estados) entre vocês.

O **REST** não muda isso, mas coloca toda a responsabilidade de “lembrar” os dados (estados) da requisição no cliente, que pode ser seu navegador/computador/aplicação.

Isso porque a cada requisição, o servidor que responde pela mesma pode ser diferente. Ele pode nunca ter tido contato com o cliente que o está contactando. Por outro lado, o cliente é o mesmo e o cliente sabe quais dados precisa seja para realizar autenticações ou mesmo para acessar diferentes **endpoints**.



Entendendo os Endpoints

Para falar de **endpoints** devemos falar um pouco de gramática.

Você não entendeu errado não. Eu disse: **Gramática**.

Na gramática da língua portuguesa, por exemplo temos **substantivos**, **verbos**, adjetivos, advérbios, pronomes...e por ai vai. São ao todo 10 classes gramaticais.

Está lembrado?

Mas o que isso tem a ver com **endpoints**?

Para criar bons **endpoints** você precisa saber o que é substantivo e verbo. Pois usamos estes conceitos para criá-los.



Entendendo os Endpoints - Substantivos

Em uma **API REST**, nós temos um elemento chamado 'resources' (recursos).

Um 'resource' pode ser por exemplo um **model** na nossa aplicação.

Então imagine que tenhamos uma aplicação que tenha um **model** 'categorias' outro 'produtos'.

Cada um destes seria um 'resource'

São estes recursos que nós queremos realizar operações **CRUD** (**C**reate, **R**etrieve, **U**pdate, **D**elete) através da nossa API.

Nós fazemos isso através de URI específicas na nossa aplicação, por exemplo:

<http://www.meusistema.com.br/api/v1/produtos> ou <http://www.meusistema.com.br/api/v1/categorias>

Estas **URIs** são os **endpoints**



Entendendo os Endpoints - Substantivos

Um **endpoint** pode representar uma coleção de registros ou um registro individual.

Exemplo:

Coleção: /api/v1/produtos

Individual: /api/v1/produtos/42

Veja que a **URI** é a mesma, pois se trata do mesmo recurso. O que diferencia a **URI** de acesso a uma coleção para um indivíduo é que no indivíduo estamos especificando um identificador.

Entendendo os Endpoints - Substantivos



Uma exemplificação ruim (errada) do design de **APIs REST**

Exemplo:

Coleção: /api/v1/produtos

Individual: /api/v1/42/produto

O design da criação correta de endpoints faz toda a diferença na boa utilização da API que você está criando.

Veja a **URI** do acesso ao recurso individual que estamos fazendo acesso ao recurso '42' com identificador 'produto'. Ou seja, é um acesso invertido, onde a leitura é confusa e não deveria ser utilizado. Inclusive está fora do especificado pelo padrão **RESTFul**. Além disso um recurso deve por recomendação ser um **substantivo** no plural.

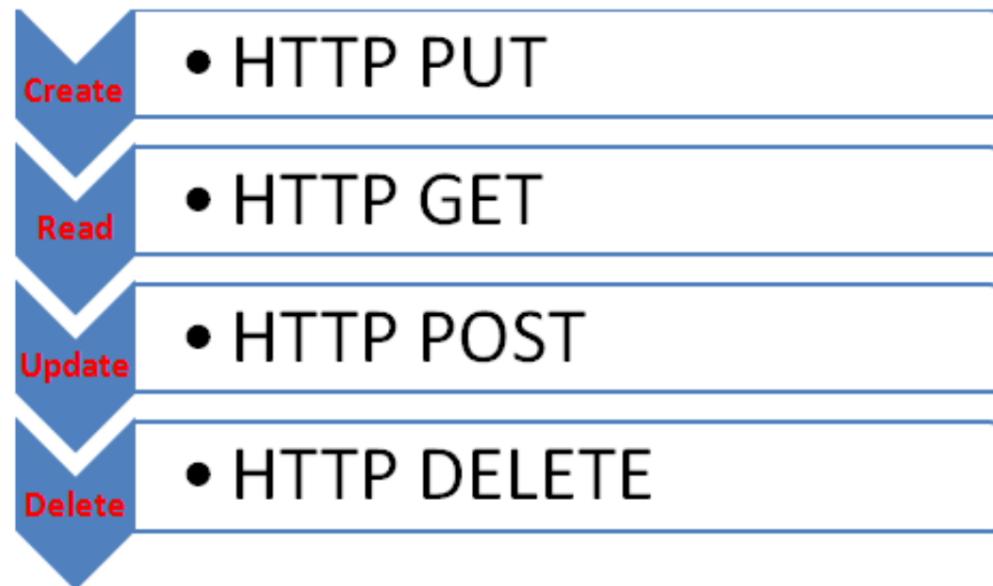
OBS: Você pode utilizar um **substantivo** no singular, desde que mantenha um padrão em toda a sua API.



Entendendo os Endpoints - Verbos

Verbos indicam uma ação. Ou seja, coisas que estamos fazendo ou que queremos fazer.

Em APIs **RESTful**, fazemos uso dos verbos **HTTP** para indicar a ação que queremos realizar com nosso recurso.



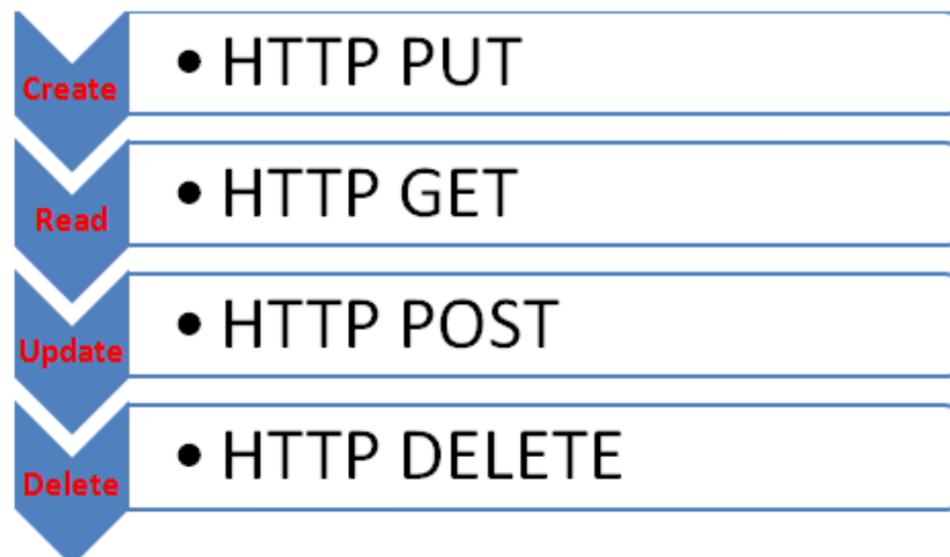


Entendendo os Endpoints - Verbos - GET

Usamos o método (verbo) HTTP GET tanto para acessar uma coleção de recursos quanto um recurso individual.

HTTP GET => /api/v1/produtos

HTTP GET => /api/v1/produtos/42



Entendendo os Endpoints - Verbos - POST

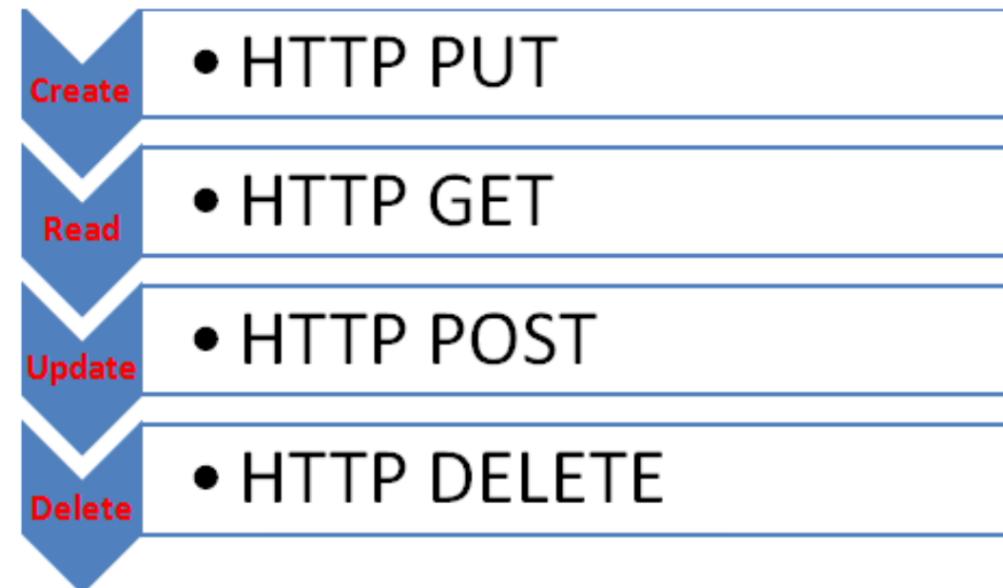


Usamos o método (verbo) HTTP POST para adicionar (criar) um novo recurso na coleção.

HTTP POST => /api/v1/produtos

~~HTTP POST => /api/v1/produtos/42~~

Não usamos POST para esta URI pois ela não é uma coleção





Entendendo os Endpoints - Verbos - PUT

Usamos o método (verbo) HTTP PUT para atualizar um recurso existente na coleção.

~~HTTP PUT => /api/v1/produtos~~

HTTP PUT => /api/v1/produtos/42

Não usamos PUT para esta URI pois ela não é um recurso individual.



Entendendo os Endpoints - Verbos - DELETE

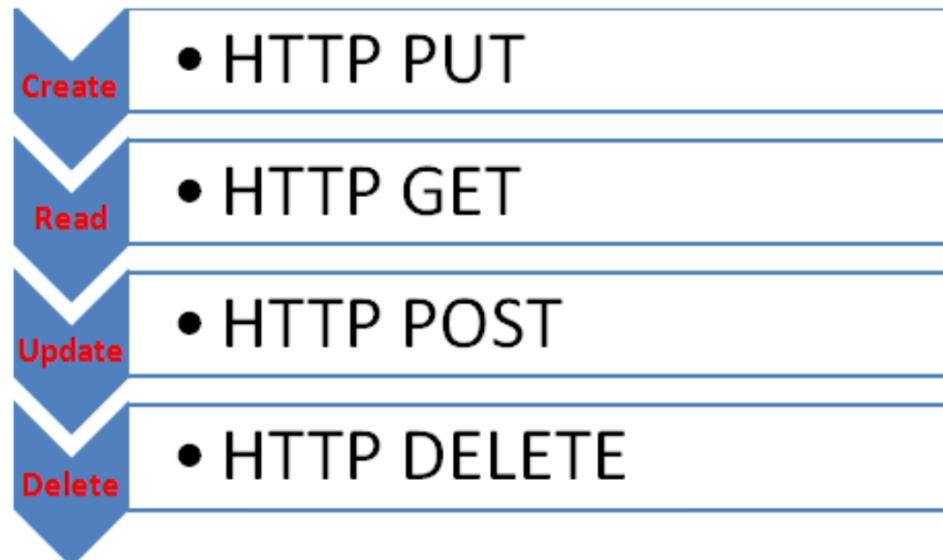


Usamos o método (verbo) HTTP DELETE para excluir um recurso existente na coleção.

~~HTTP DELETE => /api/v1/produtos~~

HTTP DELETE => /api/v1/produtos/42

Não usamos DELETE para esta URI pois ela não é um recurso individual.





Conceitos Essenciais sobre APIs

Com a correta utilização de substantivos e verbos nós podemos criar qualquer tipo de **API** utilizando os conceitos e recomendações de padrão **RESTful**.

É isso que vamos aprender neste curso, de forma simples, correta e eficiente.



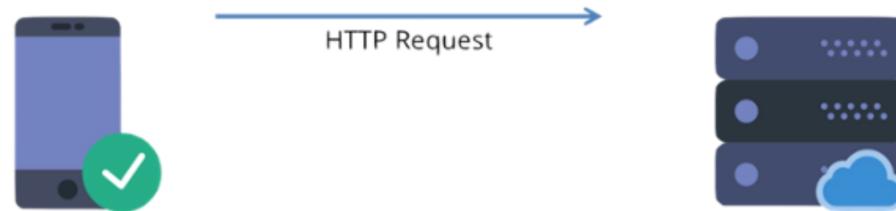
Entendendo as Requests

As requisições ([requests](#)) contém muito mais informações do que um simples verbo **HTTP** e uma **URI** para qual foi feita esta requisição.

Nós podemos mudar aspectos desta nossa requisição para que possamos então alterar o formato das respostas que serão enviadas pelo servidor **HTTP**.

Exemplo:

`/api/v1/produtos?order=desc&limit=10`



Tudo que está após o símbolo de interrogação (?) - question mark) são conjuntos de pares chave/valor que podem ser utilizadas pela **API** para alterar os dados de acordo com estes parâmetros. Esta forma de passar dados em uma requisição é chamada de “[query string](#)”.

Esta **URI** do exemplo, possui a chave ‘`order`’ com o valor ‘`desc`’ que fará a coleção de produtos ser apresentada de forma ordenada por ordem decrescente. Possui também a chave ‘`limit`’ com o valor ‘`10`’ que fará a coleção de produtos ser limitada a 10 produtos.



Entendendo as Requests

Não existia nada e nem uma regra que te impeça de passar estes dados extras diretamente via **query string**. Mas esta nem sempre é a forma recomendada.

Exemplo:

/api/v1/produtos?format=xml

No exemplo acima, estamos enviando via query string que queremos que os dados sejam apresentados no formato xml.

Ao invés de utilizar query string neste caso, poderíamos prover uma URI que aponta diretamente para o formato desejado, conforme:

/api/v1/produtos.xml

Basta que prestemos atenção no **cabeçalho da requisição HTTP**, que nos foi enviado:

Accept: application/xml



Cabeçalho da request - Accept

Especifica o formato do arquivo que o requester (solicitante) quer.

Accept: application/xml

Accept: application/json

Accept: application/pdf



Cabeçalho da request - Accept-Language

Especifica a língua do conteúdo, por exemplo Portuguese, English, Spanish, Russian, etc



Cabeçalho da request - Cache-Control

Especifica se o conteúdo pode ser consumido do cache e em quanto tempo o cache é atualizado.



Cabeçalho da request

Nem sempre precisamos prestar atenção no cabeçalho de um request, ou mesmo especificar estes detalhes ao realizar uma requisição.

Muitas vezes o servidor ou a aplicação já está definida para receber requisições com tipos padrão, por exemplo application/json

Diferentes métodos HTTP enviam dados de formas diferentes.



Versão da API

Você deve ter notado que nos exemplos de URI utilizados nesta seção apresentamos algo como:

/api/v1/produtos

Esta nomenclatura utilizando 'v1' indica a versão da API.

Uma API é uma aplicação, e como toda aplicação evolui com o tempo. E nós queremos manter a compatibilidade para todos os clientes que utilizam nossas APIs.

Então uma API recém lançada na versão 1 pode por exemplo ter algumas ações, e na versão 2 desta API pode ser que estas ações funcionem diferente, ou nem existam, ou ainda existam outras.

É importante notar que novos clientes irão utilizar a API na nova versão, mas pode haver clientes mais antigos que ainda utilizam a versão anterior da API. Enquanto houver clientes é importante manter o legado.

/api/v2/produtos

Entendendo as Responses

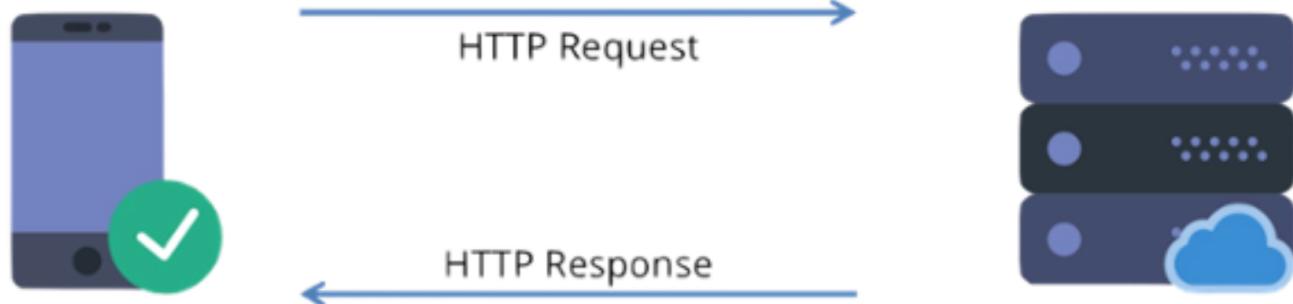


Então o usuário enviou uma requisição para a nossa **API REST**.

E agora?

Agora a gente avalia algumas coisas:

- Na requisição existe query string?
- Qual foi o verbo HTTP que realizou a ação?
- Quais são os dados do cabeçalho?
- Qual o formato requisitado?
- E claro, a gente prepara os dados da coleção ou indivíduo do recurso solicitado.



Feito isso, é hora de preparar a **response** (resposta) àquela requisição.



Entendendo as Responses

Em uma response, claro, enviamos (data) dados, mas tem mais coisas que são enviadas junto.

Assim como uma Request (requisição) HTTP, a Response (resposta) HTTP também possui cabeçalho.

Por exemplo:

Content-Type: text/javascript => Este content-type deve bater com o Accept especificado no cabeçalho da requisição.

Last-Modified: Tue, 15 Jan 2020 12:45:26 GMT => Ainda temos a data de criação ou última modificação do recurso.

Expires: Thu, 22 Jan 2020 16:00:00 GMT => Ou ainda até quando este dado pode ser considerado ‘atual’

Status: 200 OK => O código de status HTTP, na qual pode nos guiar para nossas próximas ações.

Entendendo as Responses - Status



HTTP Status Codes



Entendendo as Responses - Status



Códigos de status **HTTP** entre **200** a **299** indica que tudo está **OK**.

Por exemplo, ao consultar um recurso e o servidor consegue encontrá-lo, o código de status é **200 OK**

Ao criar um novo recurso, tudo correndo bem, o código de status deverá ser **202 OK**

E assim por diante.

Entendendo as Responses - Status



Códigos de status **HTTP** entre **300** a **399** indica que a requisição foi entendida, mas que o recurso está em algum outro local.

A URI pode ser sido redirecionada para outro endereço e etc.

Entendendo as Responses - Status



Códigos de status **HTTP** entre **400** a **499** indica que a requisição foi realizada com algum erro, do lado do cliente (solicitante).

Por exemplo, a **URI** foi informada errada, gerando **404 Not Found**

Ou ainda a **URI** pode ser requisitada apenas com o verbo **GET** e não **POST**, então o status é **405** ou **403**

Este range de códigos de status é o mais longo, com diferentes códigos para diferentes acontecimentos.

Vale a pena dar uma relida na documentação do protocolo **HTTP** para mais detalhes.

Entendendo as Responses - Status



Códigos de status **HTTP** entre **500 a 599** indica que a requisição foi realizada mas houve algum erro do lado do servidor.

Por exemplo, um dos mais comuns é o erro **500 Server Error**

Este erro é muito genérico, pois informa que houve erro no servidor, mas não especifica qual foi o erro.

Entendendo as Responses - Status



É função sua como desenvolvedor para informar códigos de status **HTTP** corretamente para os clientes das **APIs** que você desenvolver.

Quando melhor descritivos, melhores serão suas **APIs**.

Neste curso da **Geek University** vamos aprender as melhores práticas sempre.



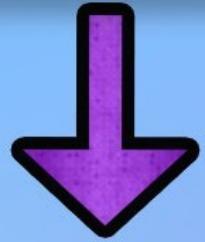
Formato de Dados em APIs

APIs não são algo novo e vem evoluindo constantemente ao longo dos anos.

XML e JSON



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<youtube>
  <canal>
    <nome>Canal TI</nome>
    <data_inscricao>23-02-2015</data_inscricao>
  </canal>
  <canal>
    <nome>Outro canal</nome>
    <data_inscricao>01-01-2017</data_inscricao>
  </canal>
</youtube>
```



```
{
  "youtube": {
    "canal": [
      {
        "nome": "Canal TI",
        "data_inscricao": "23-02-2015"
      },
      {
        "nome": "Outro canal",
        "data_inscricao": "01-01-2017"
      }
    ]
  }
}
```



Formato de Dados em APIs

APIs não são algo novo e vem evoluindo constantemente ao longo dos anos.

Já foram usados vários formatos e tipos de dados para comunicação entre sistemas através das APIs, mas atualmente o formato mais utilizado é o JSON.

{JSON}
JavaScript Object Notation



Formato de Dados em APIs

Felizmente Python foi criado com uma visão de futuro, e para nossa sorte os dicionários Python são bem parecidos com o formato JSON.



Dict



```
{'name': 'Apple'  
  'color': 'Red'  
  'quantity': 10 }
```



JSON

```
{"name": "Apple", "color":  
  "Red", "quantity": 10}
```

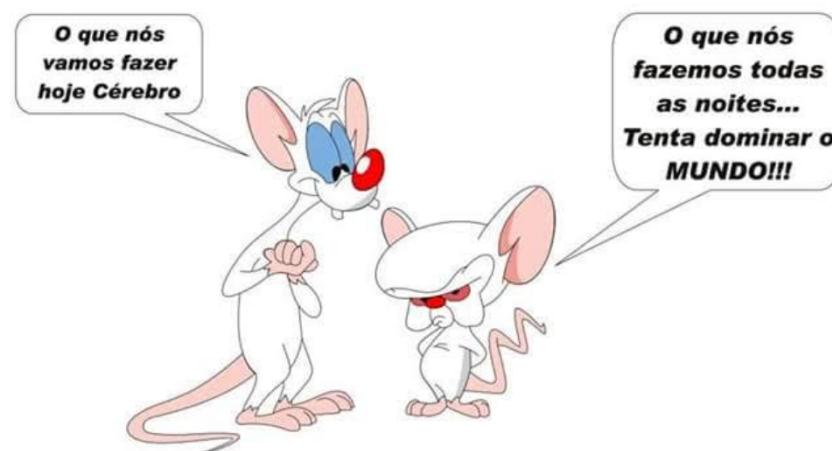


Entendendo sobre a segurança de APIs

Uma **API** que não consegue suprir a demanda é tão ruim quanto não ter nenhuma **API**.

Os usuários não irão ficar aguardando até que os servidores coloquem sua **API** de volta no ar. Se eles tiverem opções eles irão atrás destas opções.

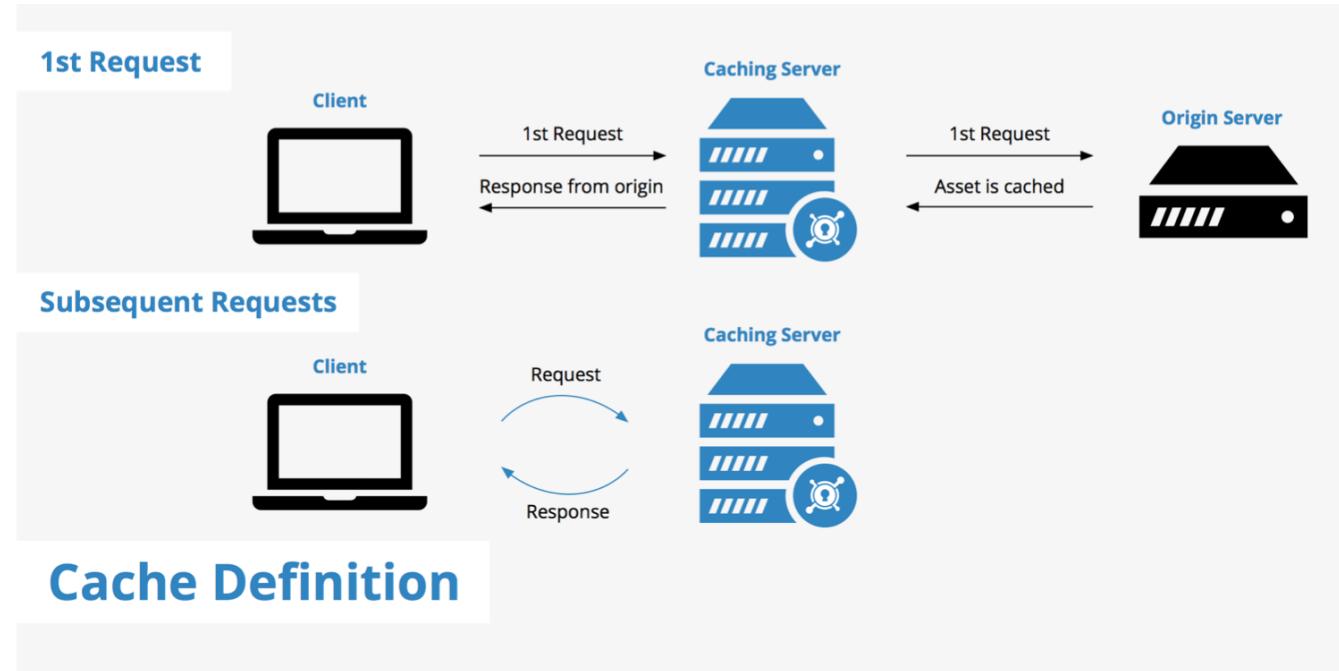
E você não quer. Você quer conquistar mais e mais clientes até dominar o mundo.





Entendendo sobre a segurança de APIs

Fazendo uso de cache, por exemplo um cliente pode fazer uma requisição e os dados vem direto do servidor que processou e enviou a resposta. Um segundo cliente faz uma nova requisição mas agora os dados podem ser pegos direto do cache. Mais rápido e eficiente.



Ferramentas como o **Redis** ou o Memcache podem ser utilizados para isso.



Entendendo sobre a segurança de APIs

Nem todo o cache do mundo irá te salvar se sua **API** for inundada de requisições.

Imagine que você tenha um servidor com capacidade de 1000 requisições por segundo.

Para a grande maioria das aplicações este número é mais que suficiente. Mas lembre-se, sua intenção é **dominar o mundo!**

Então numa noite de sexta-feira já quase hora do fim do expediente, o servidor recebe 1001 requisições no mesmo segundo e cai.

Isso não deveria acontecer.

O seu servidor não poderia estar recebendo mais requisições que o suportado.

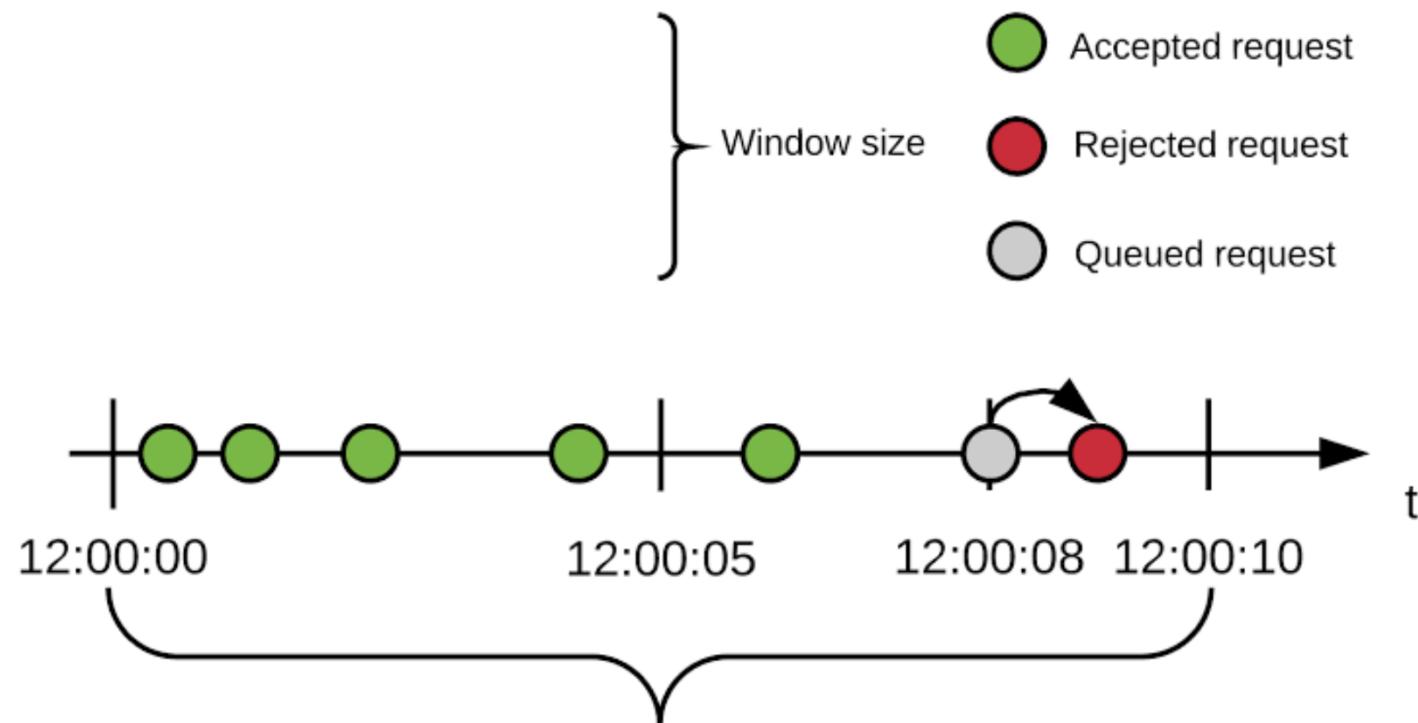
Você como desenvolvedor pode/deve adicionar limite de requisições.

Isso pode ser feito inclusive na venda de serviços, muito comum atualmente, na qual um cliente contrata 50, 100, 500 ou 1000 requisições por mês para acessar seus serviços.



Entendendo sobre a segurança de APIs

Neste exemplo temos uma janela de 5 requisições a cada 10 segundos. Ou seja, o cliente só pode fazer 5 requisições no espaço de tempo de 10 segundos. Neste exemplo ainda o cliente tentou fazer uma sexta requisição que foi rejeitada.





Entendendo sobre a segurança de APIs

Um último passo para entender tudo que envolve a segurança de **APIs** está a autenticação e autorização.

É muito difícil você conseguir limitar clientes a acessarem suas **APIs** se você não sabe quem são seus clientes ou quem está tentando fazer acesso à sua **API**.

Lembre-se, o **HTTP** é **Stateless** (não guarda estado)

Como os clientes irão conseguir uma conta de acesso cada a você de acordo com as regras do seu sistema.

Mas existem algumas formas de manusear a autenticação de clientes em **APIs REST**.



Entendendo sobre a segurança de APIs

A forma mais comum de autenticação de clientes em **APIs REST** é fazendo uso de **Tokens**

De forma geral, o token é uma chave criptográfica que identifica o cliente. Ou seja, quando o cliente cria uma conta na sua aplicação ele recebe uma chave (**Public Key**) e através desta chave (**token**) ele envia no cabeçalho ou no corpo da requisição para realizar a **autenticação**.

Como as requisições **HTTP** são **Stateless** este token de autenticação sempre é enviado.



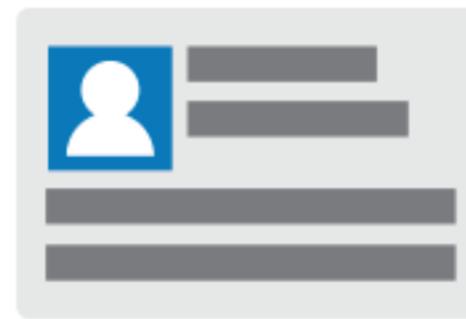
Entendendo sobre a segurança de APIs

Por fim deve ficar claro as diferenças entre Autenticação e Autorização.



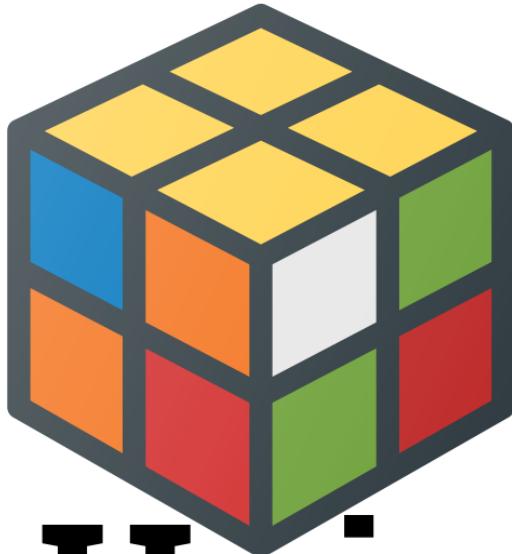
Authorization

What you can do



Authentication

Who you are



Geek University

Evolua seu lado geek!

www.geekuniversity.com.br