



**Universidade Federal do Ceará**  
**Centro de Ciências/Departamento de Computação**  
**Código da Disciplina:** CK0236  
**Professor:** Ismayle de Sousa Santos

**Aula 15**

# Técnica de Programação II

## Testes de API

---



qpg4p5x



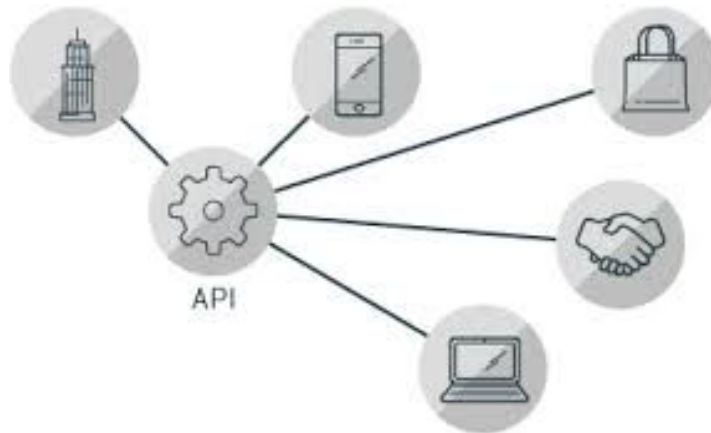
ismaylesantos@great.ufc.br



@IsmayleSantos

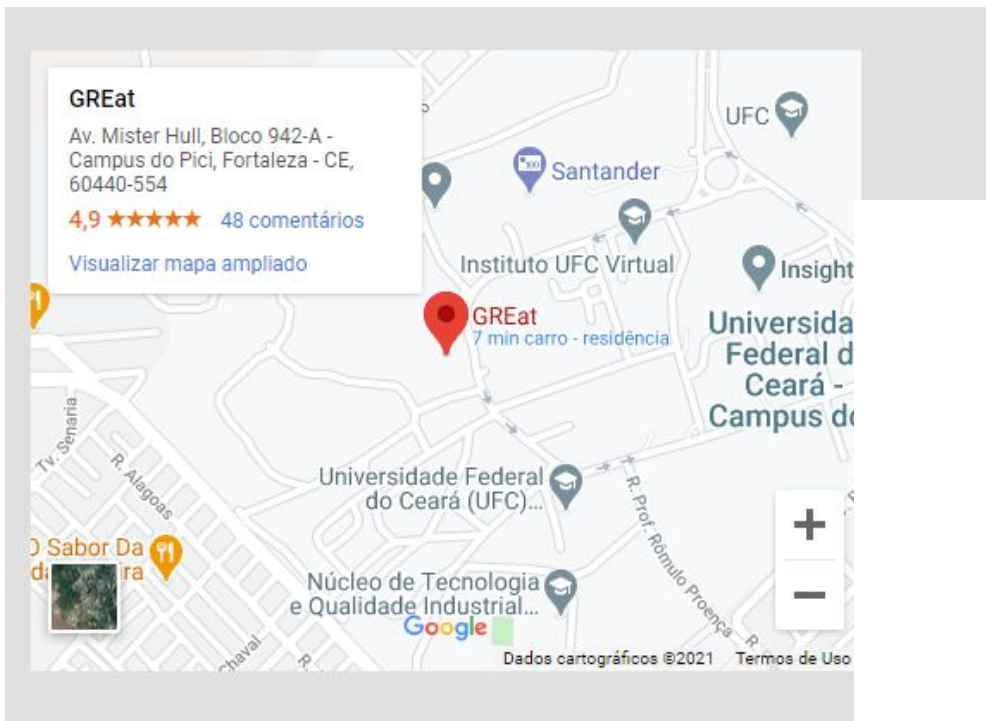
# O que é uma API?

- Interface de Programação de Aplicações
  - API (**A**pplication **P**rogramming **I**nterface)
  - É um conjunto de assinaturas que são disponibilizadas para os usuários de uma biblioteca/framework desenvolverem os aplicativos deles [Swebok 2014]
- Uma API permite a interoperabilidade entre sistemas



# O que é uma API?

- Exemplo
  - API do Google Maps
    - No site do GReat é utilizada a API do Google Maps



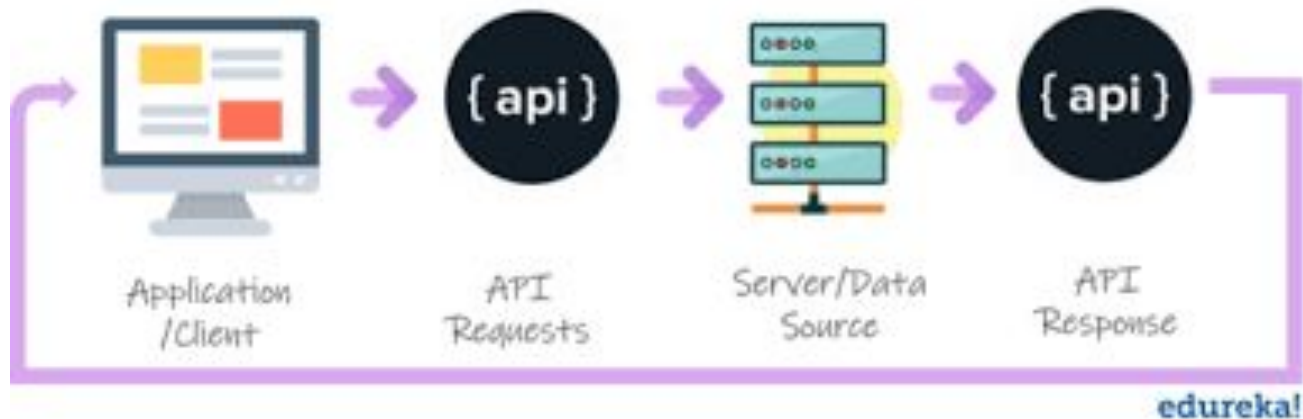
```
<style type="text/css">...</style>
<script src="chrome-extension://mooikfkahbdckldjjndioac
kbalphokd/assets/prompt.js"></script>
<script type="text/javascript" charset="UTF-8" src="http
s://maps.googleapis.com/maps-api-v3/api/js/43/6/intl/p
t_br/common.js" nonce></script>
<script type="text/javascript" charset="UTF-8" src="http
s://maps.googleapis.com/maps-api-v3/api/js/43/6/intl/p
t_br/util.js" nonce></script>
<script type="text/javascript" charset="UTF-8" src="http
s://maps.googleapis.com/maps-api-v3/api/js/43/6/intl/p
t_br/map.js" nonce></script>
<style>...</style>
<script type="text/javascript" charset="UTF-8" src="http
s://maps.googleapis.com/maps-api-v3/api/js/43/6/intl/p
```

# Testes de Interface

- **Teste de interface**
    - **Objetiva verificar se interfaces de componentes fornecem a troca correta de dados e informações de controle [Swebok]**
    - **Usualmente casos de testes são gerados a partir da especificação de interfaces**
    - **Simula o uso de APIs por aplicações do usuário final**
      - **Envolve a geração de parâmetros para as chamadas de API, configuração de parâmetros ambientais e a definição de dados internos que afetam a API**
-

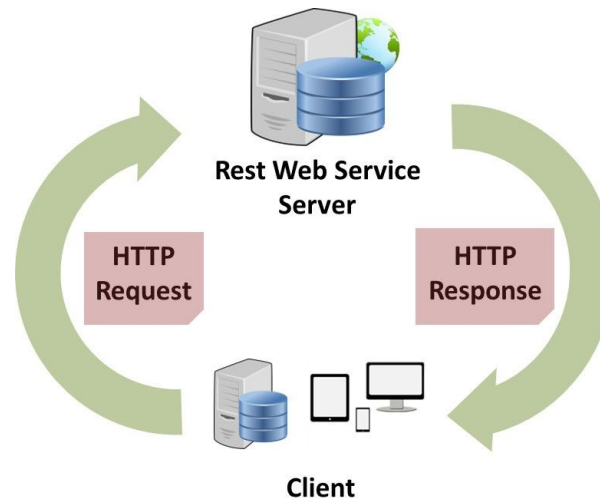
# Testes de Interface

- No teste de API, você vai simular uma requisição e verificar a resposta



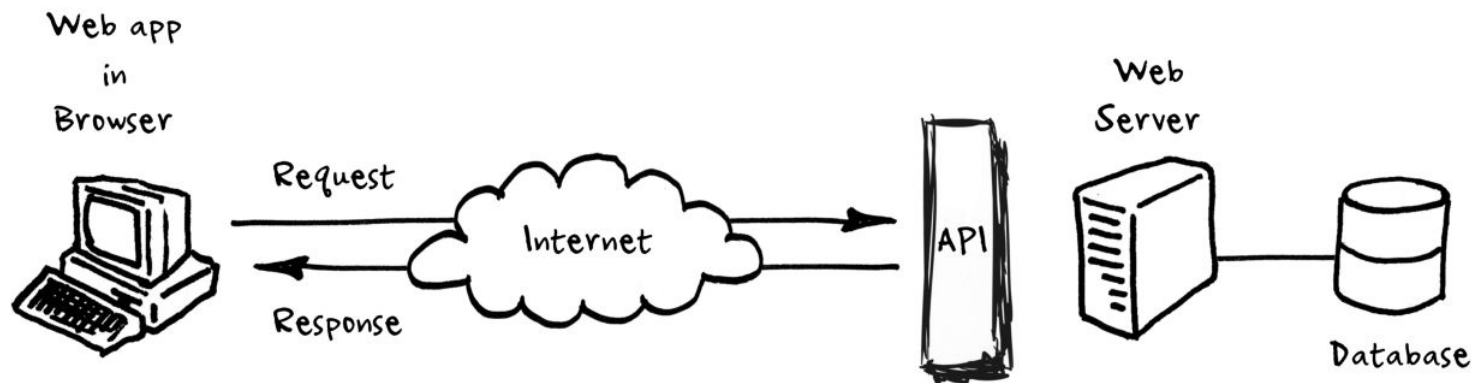
# Rest

- Padrão REST (Representational State Transfer)
  - Transferência de Estado Representacional
  - Representa uma série de princípios (arquiteturais) visando a padronização de rotas, requisições e comunicações sem estado
- Restful é o termo atribuído a uma API que implementa o padrão REST



# Requisição

- **Requisição HTTP**
  - Mensagens enviadas pelo cliente com alguns parâmetros
  - A resposta contém alguns dados e status



*HTTP é um protocolo cliente-servidor*

# Formato de uma requisição HTTP

- Exemplo de Requisição
  - Na resposta temos o cabeçalho e o corpo



Essa caixa é uma Requisição, lembre-se dela!



# Formato de uma requisição HTTP

- Métodos em REST
  - GET
    - Buscar
  - POST
    - Registrar
  - PUT
    - Alterar
  - DELETE
    - Remover

Task	Method	Path
Create a new customer	POST	/customers
Delete an existing customer	DELETE	/customers/{id}
Get a specific customer	GET	/customers/{id}
Search for customers	GET	/customers
Update an existing customer	PUT	/customers/{id}

# Formato de uma requisição HTTP

- URI (Uniform Resource Identifier)
  - Indica o endereço e recurso a ser utilizado
  - Exemplo de api fictícia
    - Endereço: <http://api.lojinha.com>
    - Recurso: login
    - URI: <http://api.lojinha.com/login>

Apenas um  
**Exemplo**

# Formato de uma requisição HTTP

- Cabeçalho
    - Suporte do padrão REST
      - XML
      - JSON
      - HTML
    - Como vamos enviar JSON
      - Content-Type
        - application/json
-

# JSON

- **JavaScript Object Notation**

- <https://www.json.org/>
- É uma forma de representação dos dados baseado na sintaxe de object JavaScript

```
"restaurant": {  
  "name": "Fish Witch",  
  "address": "214 NE Broadway",  
  "zipcode": "97232",  
  "phone": "503-000-0000",  
  "website": "http://fishwitch.com",  
  "email": "hellofishy@fishwitch.com"  
}
```

# Formato de uma requisição HTTP

- Corpo
  - Envia os dados via JSON
  - Supondo Atributos
    - Usuario
    - Senha

```
{  
  "usuarioLogin": klebinho  
  "senhaSenha": 123456  
}
```



# Formato de uma requisição HTTP



# Envio da Requisição

- Ao receber a requisição, a API REST da Lojinha
    - extrai os dados do corpo da requisição
    - pesquisa no banco por um usuário klebinho com senha 123456
    - devolve um Cabeçalho e um Corpo
  - No cabeçalho
    - Retorna o código de resposta (Status Code)
  - No corpo
    - Poderia vir o Token do login do Kleber

```
{  
  "token": b467200db234509zo98  
}
```
-

# Envio da Requisição

- E se agora quisermos pesquisar aquele usuário?
  - Basta fazer uma requisição GET

GET	http://api.lojinha.com/produto
token = 761b69db-ace4-49cd-84cb	

---



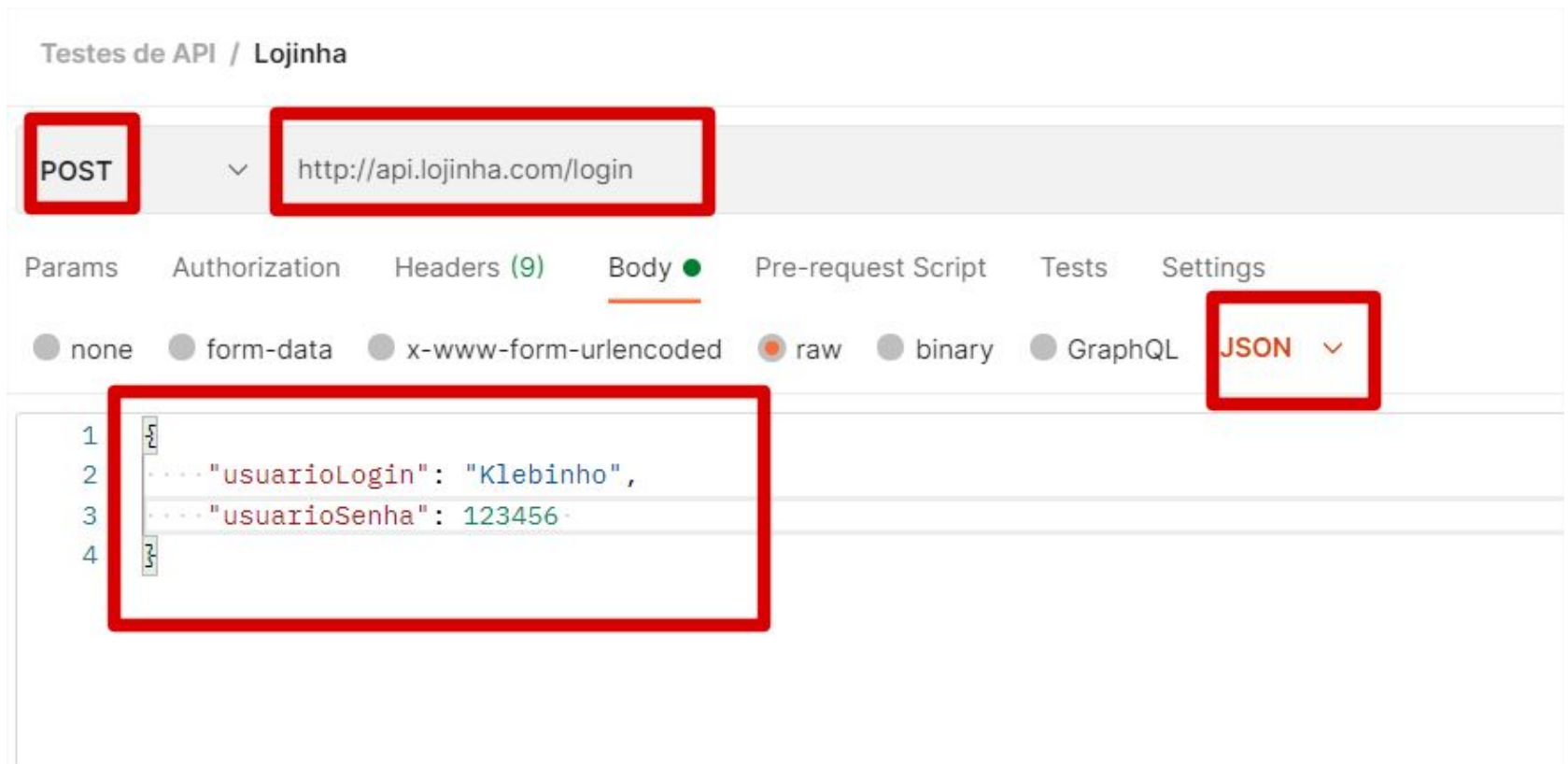
# Ferramentas de Teste de API

- Ferramentas
  - Postman
  - Advanced REST Client
  - Soap UI



# Postman

- Exemplo de Envio - [API fictícia](#)



# Postman

- Exemplo de Resposta - API fictícia



# Swagger

- Modelo de documentação de APIs
  - Linguagem de Descrição de Interfaces para APIs RESTful
- Feito pelo desenvolvedor
- Mantém a lista das URIs, métodos, corpos de requisição e respostas



# Exemplo - Swagger

- <https://petstore.swagger.io/>

## Swagger Petstore 1.0.5

[ Base URL: petstore.swagger.io/v2 ]  
<https://petstore.swagger.io/v2/swagger.json>

This is a sample server Petstore server. You can find out more about Swagger at <http://swagger.io> or on [#swagger](http://irc.freenode.net). For this sample, you can use the api key `special-key` to test the authorization filters.

[Terms of service](#)

[Contact the developer](#)

[Apache 2.0](#)

[Find out more about Swagger](#)

Schemes

HTTPS

Authorize



**pet** Everything about your Pets

Find out more: <http://swagger.io>

**POST** `/pet/{petId}/uploadImage` uploads an image



**POST** `/pet` Add a new pet to the store



**PUT** `/pet` Update an existing pet



**GET** `/pet/findByStatus` Finds Pets by status



# Encontrando Falhas

- Você pode criar diferentes requisições para expor o sistema a situações que podem resultar em falhas
  - Alguns pontos a serem testados
    - As regras de negócio foram implementadas corretamente?
    - Cada método faz o que deveria?
    - A estrutura de resposta segue o Swagger?
    - Os códigos de estados estão corretamente configurados?
    - O tempo de resposta está satisfatório?
-

# Encontrando Falhas

- Exemplos de Testes

```
{  
    usuarioLogin: Klebinho ;  
    usuárioSenha: 123456  
}
```

```
{  
    usuarioLogin: ,  
    usuárioSenha: 123456  
}
```



# Encontrando Falhas

- Exemplos de Testes

```
{  
  usuarioLogin: true,  
  usuárioSenha: 123456  
}
```

```
{  
  usuarioLogin: klebinho,  
  usuárioSenha: @!  
}
```





# Encontrando Falhas

- Exemplos de Testes - GET
    - Suponha que a API espere algo no formato
      - `http://api.lojinha.com/usuarios?UserId=2`
    - Exemplos de Testes
      - `http://api.lojinha.com/usuarios?UserId=x`
      - `http://api.lojinha.com/usuarios?UserId=`
      - `http://api.lojinha.com/usuarios?UserId=8985612`
        - *(id inexistente)*
      - `http://api.lojinha.com/usuarios?UserId=""`
-

# Códigos de Retorno

- Existem Vários códigos
  - <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status>
- Classes
  - Respostas de Informação (100–199)
  - Respostas de Sucesso (200–299)
  - Redirecionamentos (300–399)
  - Erros do cliente (400–499)
  - Erros do servidor (500–599)
- Exemplo

**201 Created**

A requisição foi bem sucedida e um novo recurso foi criado como resultado. Esta é uma típica resposta enviada após uma requisição POST.

---

# Códigos de Retorno

- Observação
    - Métodos GET retornam o código 200 mesmo que nenhum recurso tenha sido encontrado
      - O corpo da resposta é um array vazio
    - Métodos DELETE e PUT retornam o código 200 se forem processadas com sucesso
-

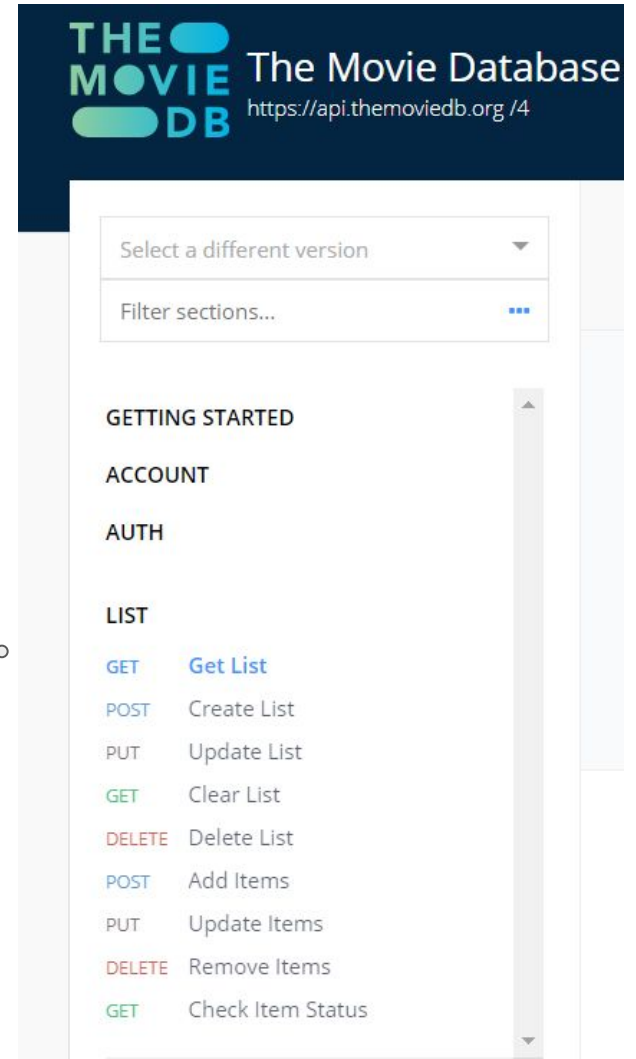
# Exemplo Real

- The Movie DB
    - É um banco de dados de filmes e TV
      - Atores, ano de lançamento etc
      - [www.themoviedb.org](http://www.themoviedb.org)
    - API pública
      - precisa de cadastro para utilizar
    - Documentação
      - <https://developers.themoviedb.org/3/getting-started/introduction>
-

# Exemplo Real

- The Movie DB
  - Documentação da API
    - <https://developers.themoviedb.org/3/getting-started/introduction>

Nos exemplos vamos  
usar métodos da lista  
AUTH e LIST



# Exemplo Real

- No postman
    - Workspace
      - Serve para organizar os projetos, agrupar as Coleções
        - *\* precisa criar conta no Postman*
    - Ambiente
      - Podemos criar variáveis para o ambiente
    - Coleções
      - Conjunto de Requisições
    - Pasta
-

# Exemplo Real

- Exemplo Ambiente

The screenshot shows the 'Movie DB' environment configuration page. On the left sidebar, 'Movie DB' is highlighted with a red box. The main area displays a table of variables for the 'Movie DB' environment. The table has columns for 'VARIABLE', 'INITIAL VALUE', and 'CURRENT VALUE'. A variable named 'baseUrl' is listed with a checked checkbox and its initial value is 'https://api.themoviedb.org/4', which is also highlighted with a red box. The 'CURRENT VALUE' column is empty.

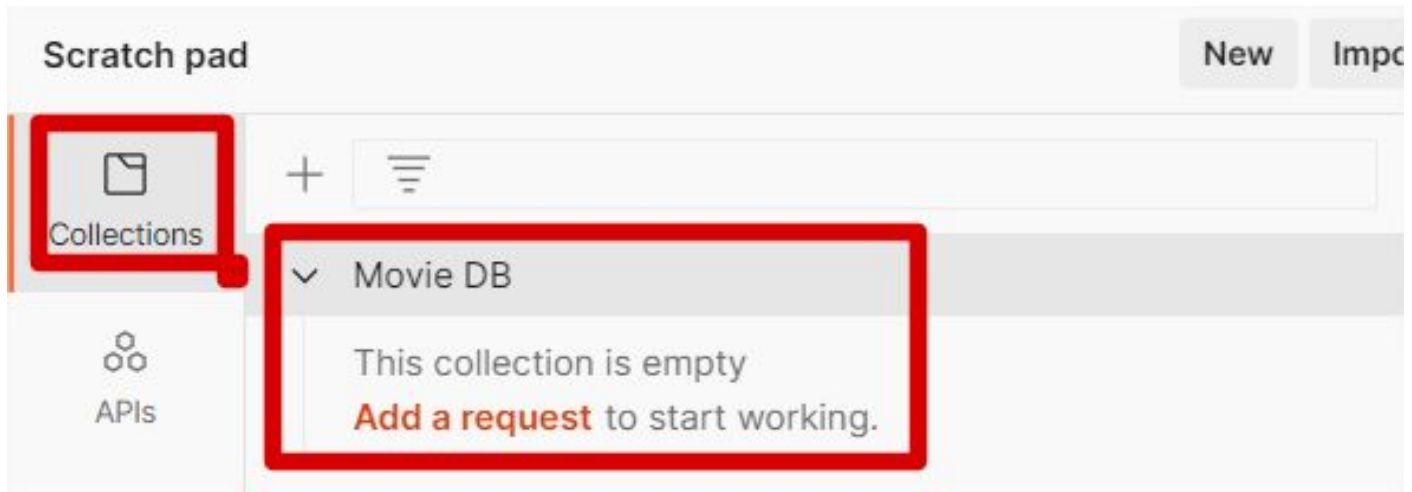
VARIABLE	INITIAL VALUE	CURRENT VALUE
<input checked="" type="checkbox"/> baseUrl	https://api.themoviedb.org/4	
Add a new variable		

The screenshot shows a dropdown menu for selecting an environment. The menu is open, showing two options: 'No Environment' and 'Movie DB'. The 'Movie DB' option is highlighted with a red box. The 'No Environment' option is also visible above it.

Depois tem que  
alterar o ambiente  
utilizado

# Exemplo Real

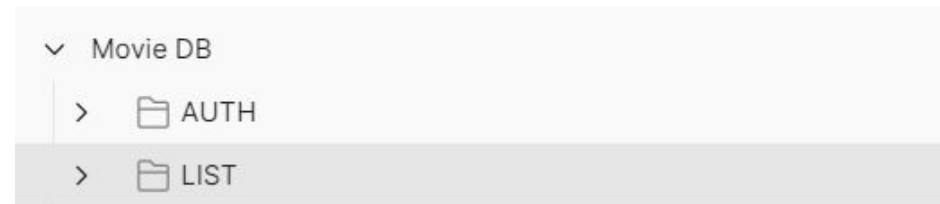
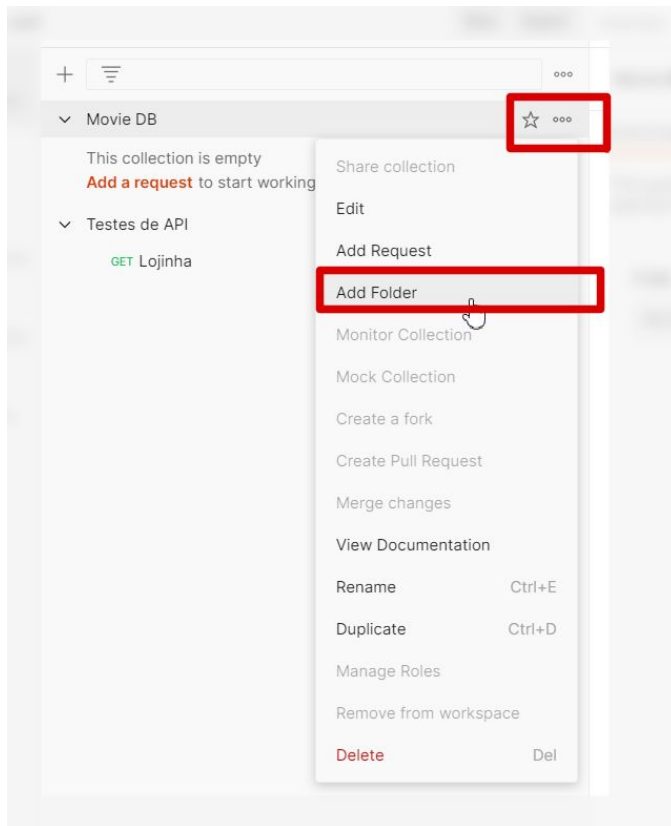
- Exemplo Coleção





# Exemplo Real

- Exemplo de Pastas



# Exemplo Real

- Como funciona a autorização no Movie DB
  - a. [Postman] Gerar Token de Requisição
    - Cria Request-Token utilizando Token de leitura
  - b. [Site] Autorizar Request-Token
    - [https://www.themoviedb.org/auth/access?request\\_token={request\\_token}](https://www.themoviedb.org/auth/access?request_token={request_token})
    - *Autoriza Request-Token*
  - c. [Postman] Gerar Token de Acesso
    - Cria Access\_Token (que pode ser usado nas requisições)

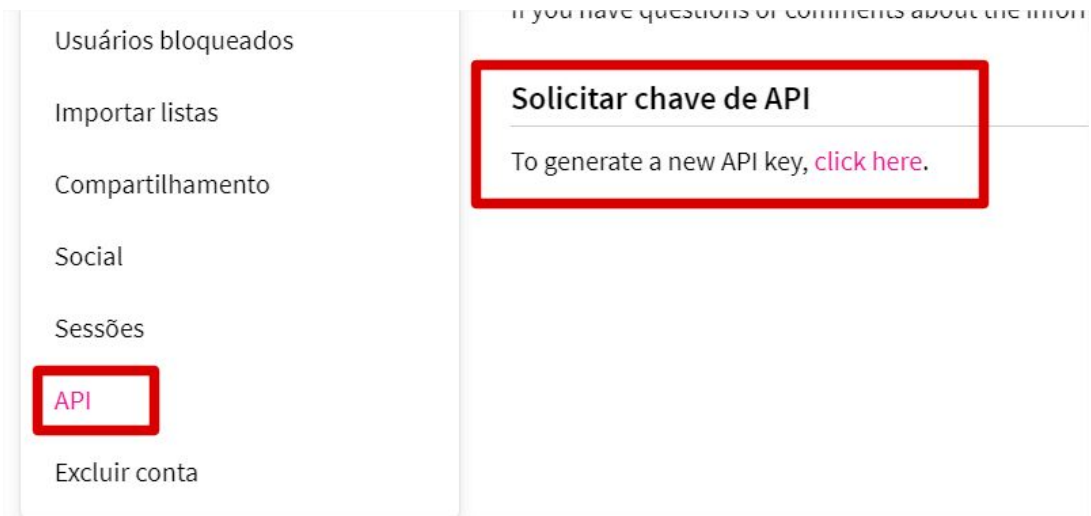
*Isso é uma particularidade desta API aberta*

<https://developers.themoviedb.org/4/auth/user-authorization-1>

---

# Exemplo Real

- Passo 1: Criando Request\_Token
  - a. Efetuar login no sistema
    - <https://www.themoviedb.org/>
  - b. No menu de perfil clicar em configurações
    - <https://www.themoviedb.org/settings/api>
  - c. No submenu API
    - Solicitar chave



# Exemplo Real

- **Passo 1: Criando Request\_Token**
  - a. Depois de preencher os dados solicitados copie o Token de Leitura da API

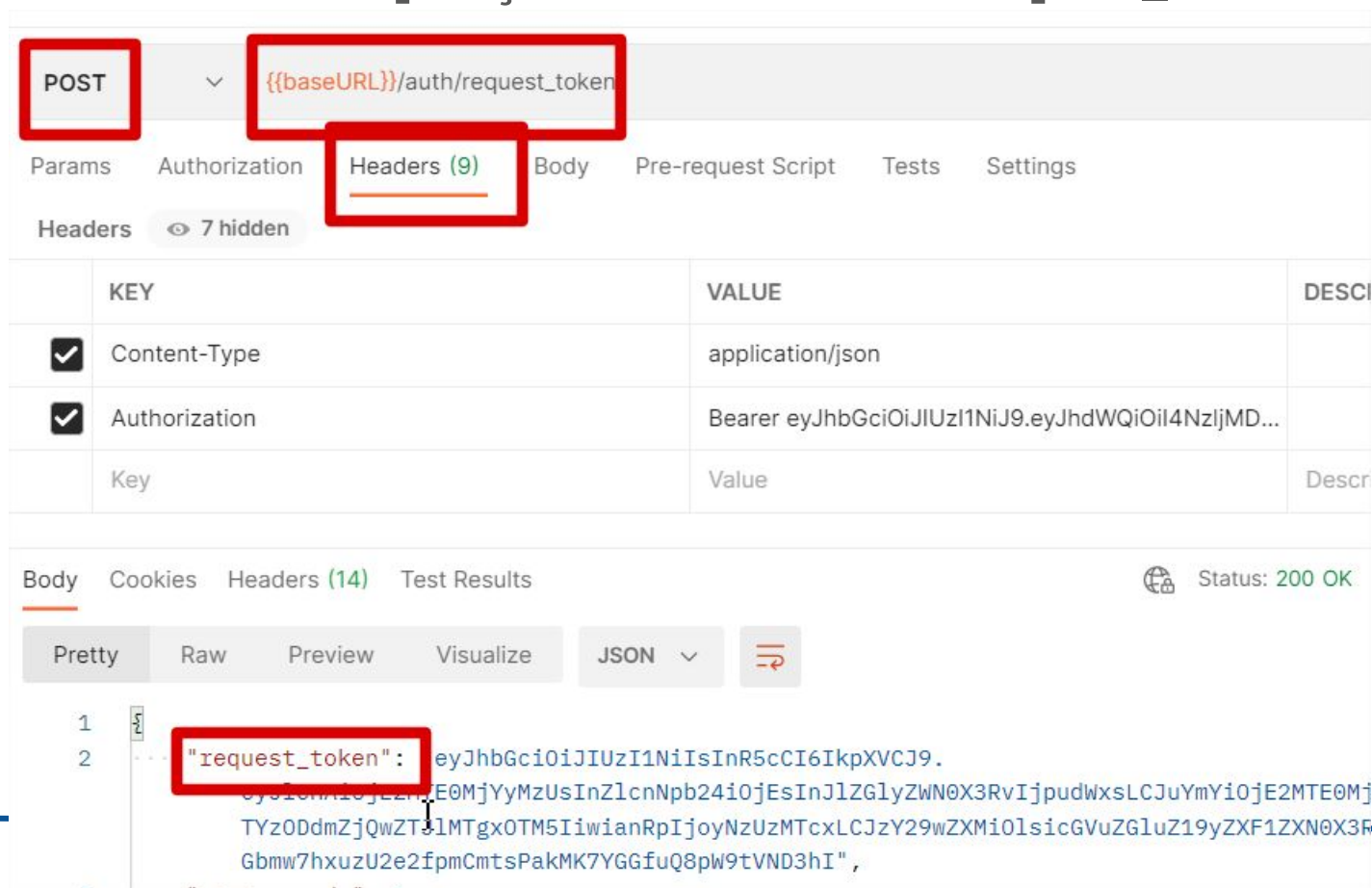
Token de Leitura da API (v4 auth)

```
eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiI4NzljMDdhZTdkYzhmNzE2Mzg3ZmY0MGUyZTE4MTkzOSIsInN1YiI6IjYwMGM1YjMxMGI1ZmQ2MDAzYzViINTM4YiIsInNjb3BlcyI6WyJhcGlfcmlhZCI6ImVhZCJ2ZXJzaW9uIjoxfQ.EE5fPwo5olSrTqGwS3QpSSWutqFAsiQCxixd8WRXpBA
```

- b. Vamos mandar uma requisição POST
    - <https://developers.themoviedb.org/4/auth/create-request-token>
-

## Exemplo Real

- **Passo 1: Criando Request\_Token**
  - a. Ao enviar a requisição vamos receber o request\_token



# Exemplo Real

- Passo 2: agora vamos autorizar o request\_token
  - [https://www.themoviedb.org/auth/access?request\\_token={request\\_token}](https://www.themoviedb.org/auth/access?request_token={request_token})

## Pedido de autenticação de terceiros

Aplicativo de Teste está pedindo sua permissão para ler e gravar dados em seu nome. Isso é necessário se você quiser fazer coisas como manter suas listas ou classificar filmes fora do TMDb.

Se deseja continuar, clique no botão aprovar abaixo. Se não, você pode simplesmente fechar esta janela.

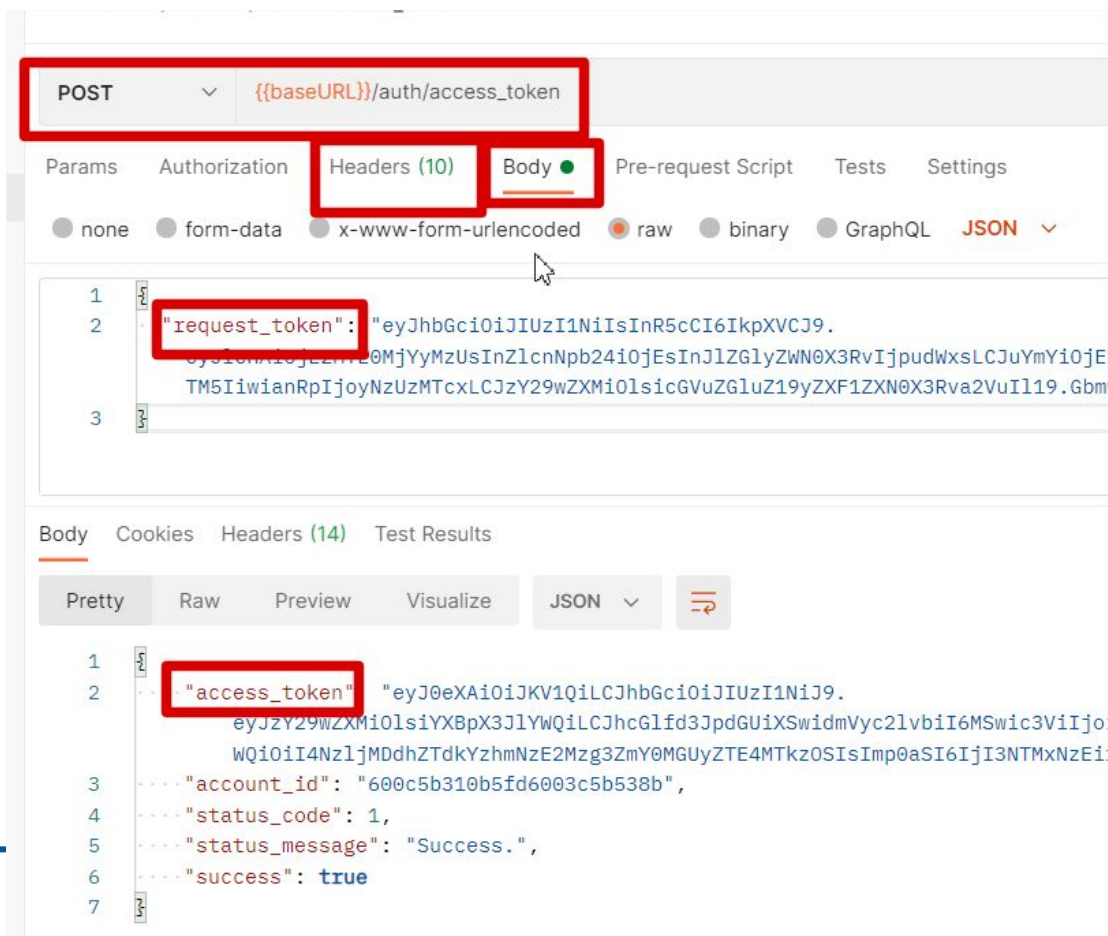
Aprovar



## Exemplo Real

- **Passo 3: Gerando Access Token**

- <https://developers.themoviedb.org/4/auth/create-access-token>



# Exemplo Real

- Fazendo outras requisições
  - Criar Lista
    - <https://developers.themoviedb.org/4/list/create-list>

Movie DB / Criar Lista

POST `{{baseURL}}/list`

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** ▼

```
1 {
2   "name": "My List",
3   "iso_639_1": "en"
4 }
```

Body Cookies Headers (13) Test Results

Pretty Raw Preview Visualize **JSON** ▼

```
1 {
2   "id": 7073238,
3   "status_code": 1,
4   "status_message": "The item/record was created successfully.",
5   "success": true
6 }
```

Idioma e Nome da Lista



# Exemplo Real

- Fazendo outras requisições
  - Get Lista
    - <https://developers.themoviedb.org/4/list/get-list>

Movie DB / LIST / Get List

GET `{{baseURL}}/list/7073238`

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Headers 6 hidden

	KEY	VALUE
<input checked="" type="checkbox"/>	Authorization	Bearer eyJ0eXAiOi
<input checked="" type="checkbox"/>	Content-Type	application/json

Body Cookies Headers (14) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "average_rating": 0,
3   "backdrop_path": null,
4   "comments": [],
5   "created_by": {
6     "gravatar_hash": "7d7ba31feae4692d2dfa8ed47016708a",
7     "name": "User",
8     "profile_path": null,
9     "username": "User",
10    "website": null
11  },
12  "first_release_date": null,
13  "first_release_year": null,
14  "genres": [],
15  "homepage": null,
16  "id": 7073238,
17  "imdb_id": null,
18  "in_production": true,
19  "is_adult": false,
20  "is_favorite": false,
21  "language": "en",
22  "last_release_date": null,
23  "last_release_year": null,
24  "logo_path": null,
25  "name": "The Movie Database",
26  "overview": null,
27  "poster_path": null,
28  "release_date": null,
29  "release_year": null,
30  "runtime": null,
31  "seasons": null,
32  "status": "Released",
33  "tagline": null,
34  "title": "The Movie Database",
35  "type": "TV Show",
36  "vote_average": 0,
37  "vote_count": 0
38 }
```

# Exemplo Real

- Fazendo outras requisições
  - Atualizar Lista
    - <https://developers.themoviedb.org/4/list/update-list>

The screenshot displays a REST client interface. The top section shows a PUT request to the URL `{{baseUrl}}/list/7073238`. The 'Body' tab is selected, and the request body is a JSON object: `{ "description": "This list is pretty awesome." }`. The bottom section shows the response body, which is a JSON object: `{ "status_code": 12, "status_message": "The item/record was updated successfully.", "success": true }`. Both the request and response JSON bodies are highlighted with red rectangles.

```
PUT {{baseUrl}}/list/7073238
```

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded **raw** binary GraphQL **JSON** ▼

```
1 {
2   "description": "This list is pretty awesome."
3 }
```

Body Cookies Headers (13) Test Results

Pretty Raw Preview Visualize **JSON** ▼

```
1 {
2   "status_code": 12,
3   "status_message": "The item/record was updated successfully.",
4   "success": true
5 }
```

# Outras formas de automatizar testes de API

**REST-assured**



# Trabalho Prático TP-7

- Criar Testes Automatizados de API
  - Pode ser no projeto final ou no movie DB
  - Se for no movie DB
    - Add Items
      - <https://developers.themoviedb.org/4/list/add-items>
    - Update Items
      - <https://developers.themoviedb.org/4/list/update-items>
    - Remove Items
      - <https://developers.themoviedb.org/4/list/remove-items>

https://www.themoviedb.org/tv/1402-the-walking-dead

media Type  
media id

---

# *Obrigado!*

## *Por hoje é só pessoal...*

# **Dúvidas?**



qpg4p5x



ismaylesantos@great.ufc.br



@IsmayleSantos

---