

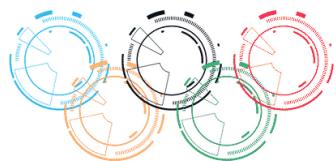
# Atividades do Módulo 3 - QA NA PRÁTICA



Esse [repositório](#) é dedicado às atividades realizadas durante o Módulo 3 - QA NA PRÁTICA do curso de Quality Assurance oferecido pelo [Instituto JogaJunto](#).

💡 Descrição da 14ª Atividade: \*

💡 Passo a passo para o desafio. Reunam-se em seus SQUADS. Metade dos SQUADS da turma serão responsáveis pela estrutura XML e a outra metade pela estrutura JSON. Os SQUADS responsáveis pelo JSON devem, a partir da estrutura do formulário e das informações disponibilizadas no case, criar um arquivo na estrutura JSON. Os SQUADS responsáveis pelo XML devem, a partir da estrutura do formulário e das informações disponibilizadas no case, criar um arquivo na estrutura XML.



## OLIMPÍADA DE INOVAÇÃO E DESENVOLVIMENTO DE APLICATIVOS

### INFORMAÇÕES DA EQUIPE

Nome do Líder (Responsável pela Inscrição)	Função
País:	Categoria: Dev App   Inovação
	Quantidade de integrantes
Nome Integrante 1:	Função
Nome Integrante 2:	Função
Nome Integrante 3:	Função
Nome Integrante 4:	Função

### INFORMAÇÕES DE CONTATO

E-Mail principal	
E-Mail secundário	
Telefone principal	Telefone recado

- A Squad a qual eu faço parte ficou responsável por criar um arquivo Json, porém, com propósito de agregar conhecimento, fiz a atividade com os dois parâmetros solicitados (Json e XML). Os arquivos correspondentes estão na pasta "Atividades" deste repositório. Segue abaixo o resultado:

JSON

```
1 {
2     "OLIMPÍADA DE INOVAÇÃO E DESENVOLVIMENTO DE APLICATIVOS": [
3         {
4             "INFORMAÇÕES DA EQUIPE": [
5                 "Nome do Líder (Responsável pela Inscrição)": "Carla Almeida",
6                 "Função do Líder": "Desenvolvedora Fullstack",
7                 "País": "Brasil",
8                 "Categoria Dev App | Inovação": "DEV APP",
9                 "Quantidade de integrantes": 5,
10                "Integrantes": [
11                    {
12                        "Nome": "Marcio Souza",
13                        "Função": "Desenvolvedor FrontEnd"
14                    },
15                    {
16                        "Nome": "Camila Santana",
17                        "Função": "Desenvolvedora BackEnd"
18                    },
19                    {
20                        "Nome": "Felipe Oliveira",
21                        "Função": "UX/UI Designer"
22                    },
23                    {
24                        "Nome": "Caique Silva",
25                        "Função": "Especialista em QA"
26                    }
27                ],
28            },
29            "INFORMAÇÕES DE CONTATO": [
30                "E-Mail principal": "carla.almeida@example.com",
31                "E-Mail secundário": "carla@example.com",
32                "Telefone principal": "+55 123 456 7890",
33                "Telefone recado": "+55 987 654 3210"
34            ]
35        }
36    }
37 }
```

---

XML

```
1 <olimpiada>
2   <informacoes_da_equipe>
3     <nome_do_lider>Carla Almeida</nome_do_lider>
4     <funcao_do_lider>Desenvolvedora Fullstack</funcao_do_lider>
5     <pais>Brasil</pais>
6     <categoria>DEV APP</categoria>
7     <quantidade_de_integrantes>5</quantidade_de_integrantes>
8     <integrantes>
9       <integrante>
10      <nome>Marcio Souza</nome>
11      <funcao>Desenvolvedor FrontEnd</funcao>
12    </integrante>
13    <integrante>
14      <nome>Camila Santana</nome>
15      <funcao>Desenvolvedora BackEnd</funcao>
16    </integrante>
17    <integrante>
18      <nome>Felipe Oliveira</nome>
19      <funcao>UX/UI Designer</funcao>
20    </integrante>
21    <integrante>
22      <nome>Caique Silva</nome>
23      <funcao>Especialista em QA</funcao>
24    </integrante>
25  </integrantes>
26 </informacoes_da_equipe>
27 <informacoes_de_contato>
28   <email_principal>carla.almeida@example.com</email_principal>
29   <email_secundario>carla@example.com</email_secundario>
30   <telefone_principal>+55 123 456 7890</telefone_principal>
31   <telefone_recado>+55 987 654 3210</telefone_recado>
32 </informacoes_de_contato>
33 </olimpiada>
34
```

## Integrantes da Squad:

| Beatriz Souza | Bruno Soares | Leanderson Lima | Rebeca Borges | Sara Cruz |

# Atividades do Módulo 3 - QA NA PRÁTICA



Esse [repositório](#) é dedicado às atividades realizadas durante o Módulo 3 - QA NA PRÁTICA do curso de Quality Assurance oferecido pelo [Instituto JogaJunto](#).

## 💡 Descrição da 14ª Atividade: \*

💡 Em SQUAD, faça o seguinte teste: A BIBLIOTECA DA CIDADE está criando um sistema para cadastro de filmes, e querem testar uma API que será usada pelas instituições parceiras para cadastro de livros. Seu SQUAD foi escolhido para realizar esse teste.

- 1 - Pegue todos os endpoints existentes na documentação do sistema.
- 2 - Cadastre 4 livros com as seguintes informações em um json: Título, Autor, Gênero e Edição.
- 3 - Faça requisições GET em todos os livros e veja se os cadastrados por você estão disponíveis.
- 4 - Adicione o teste a seguir: pm.test("Status code in 200", function(){pm.response.to.have.status(200)});
  - Uma das coisas que achei interessante sobre essa atividade é que tanto colegas da minha própria equipe quanto membros de outras equipes me procuraram para pedir ajuda na sua realização. Fiquei muito feliz por poder contribuir e auxiliar diversas pessoas nesse processo. Com a intenção de compartilhar essa ajuda com o maior número possível de indivíduos, decidi disponibilizar a resolução da atividade abaixo no formato de um "tutorial".

**Certifique-se de ter o Postman instalado em seu computador. Se você ainda não tem o Postman, você pode baixá-lo e instalá-lo a partir do [site oficial](#).**

Preparando o Postman:

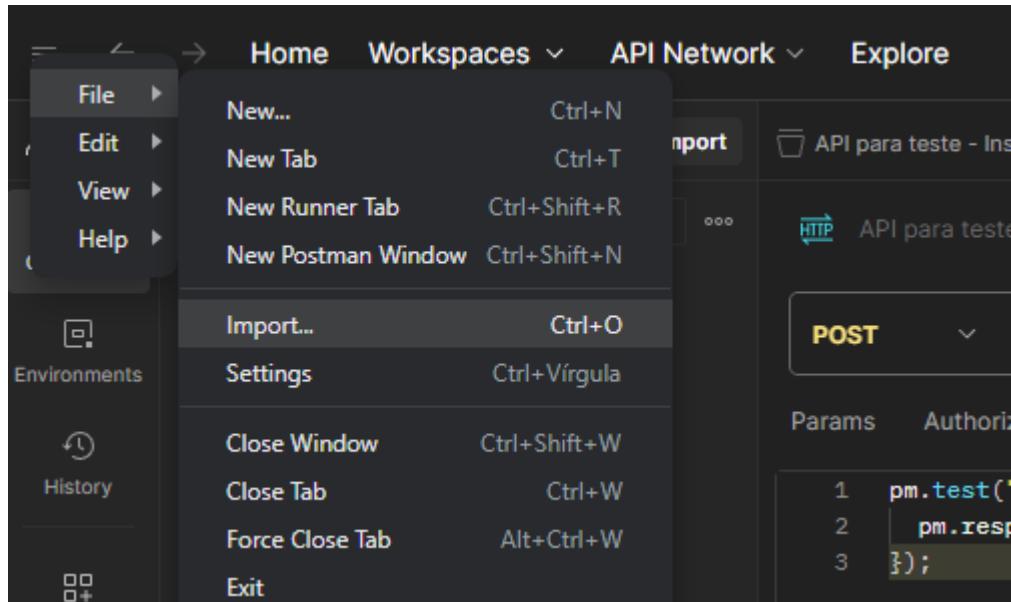
Anteriormente, busquei explicar de maneira didática. Nesta seção bônus, irei demonstrar a execução da atividade, além de compartilhar alguns atalhos que simplificaram o processo:

### 1. Criação das requisições (GET, POST, PUT, PATCH, DELETE)

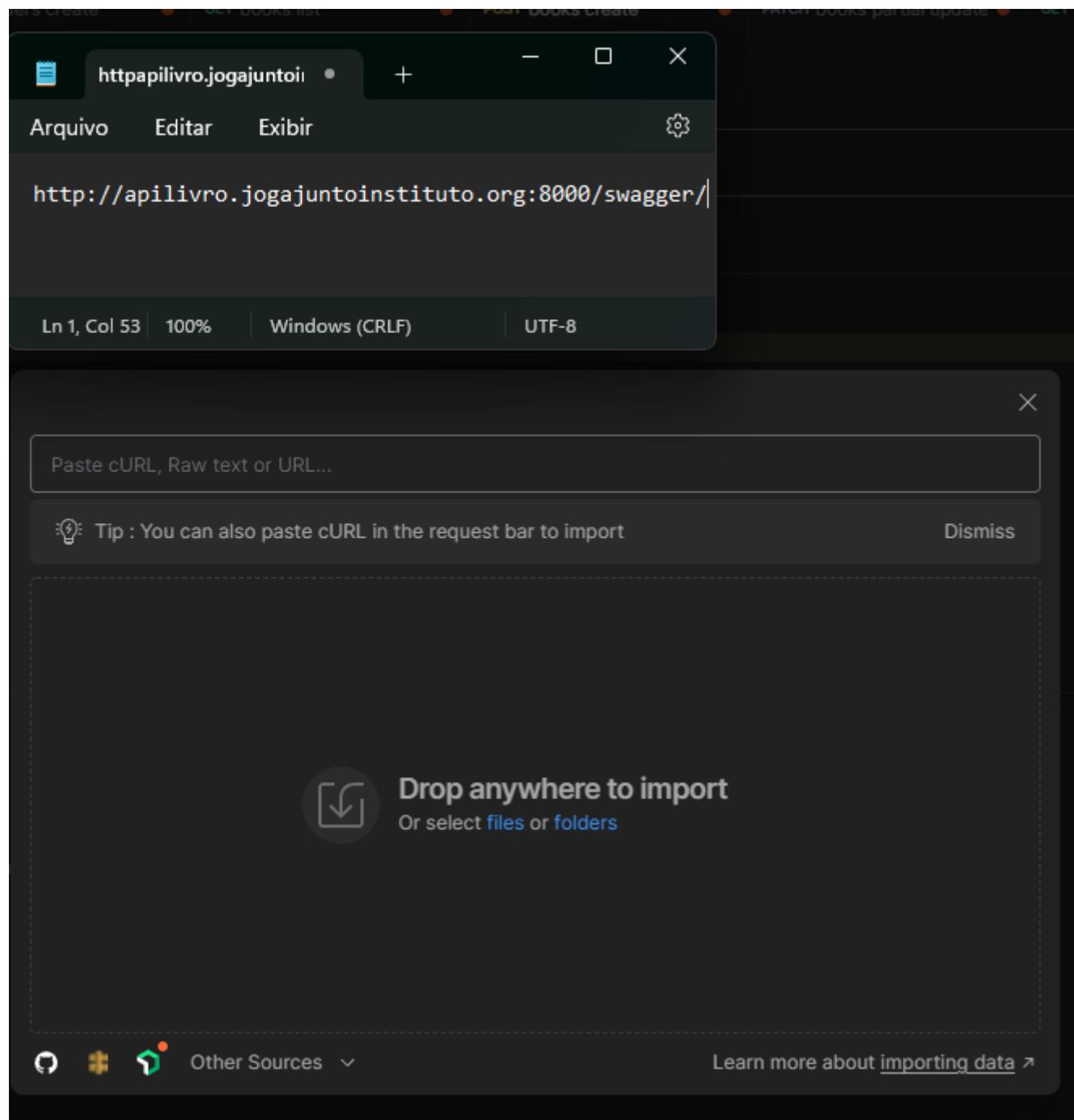
Ao invés de criar as requisições manualmente, o POSTMAN permite que você importe todas as requisições que já estão criadas na API através do link do Swagger.

---

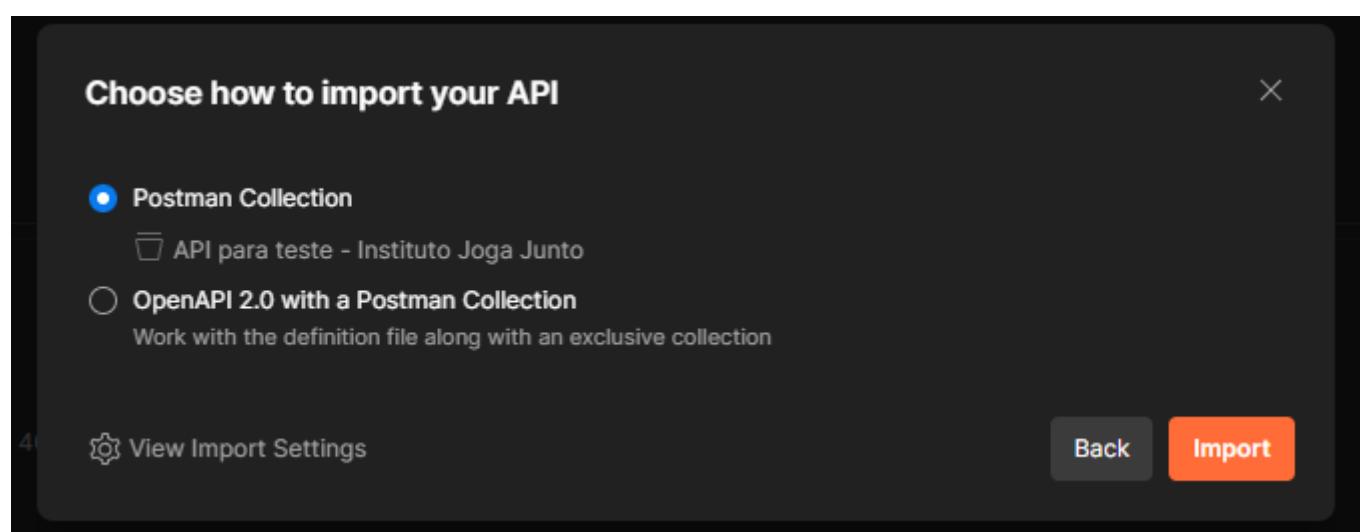
Clique no 'Menu' que fica no canto superior esquerdo representado pelo símbolo ≡. Em seguida File e depois Import...



Copie e cole a [URL da API](#) no campo designado.



Escolha a primeira opção.



Após o carregamento você terá toda a estrutura da API pronta para uso.

The screenshot shows a dark-themed Postman API collection. The root node is 'API para teste - Instituto Joga Junto ...'. It contains three main categories: 'authors', 'books', and 'genders'. Each category has a sub-node '{id}' and several methods listed under it. The methods are color-coded by HTTP verb: GET (green), PUT (blue), PATCH (purple), DEL (orange-red), and POST (yellow). The 'authors' category includes 'authors read', 'authors update', 'authors partial update', 'authors delete', 'authors list', and 'authors create'. The 'books' category includes 'books read', 'books update', 'books partial update', 'books delete', 'books list', and 'books create'. The 'genders' category includes 'genders read', 'genders update', 'genders partial update', 'genders delete', 'genders list', and 'genders create'.

A URL da API é automaticamente transformada e uma variável chamada "baseUrl".

The screenshot shows the 'Variables' tab in the Postman interface. It lists a single variable named 'baseUrl' with a value of 'http://apilivro.jogajuntoinstituto.org:8000'. The 'Initial value' and 'Current value' fields both show this same URL. Other tabs visible include 'Overview', 'Authorization', 'Pre-request Script', 'Tests', and 'Runs'.

Para evitar possíveis erros de autenticação, recomendo deixar a opção "No Auth" para executar os passos seguintes.

The screenshot shows the Postman interface with the following details:

- Left Sidebar:** My Workspace, Collections, Environments, History.
- Top Bar:** New, Import, GET books list, POST books create, POST authors create.
- Collection:** API para teste - Instituto Joga Junto (expanded), containing subfolders: authors, books, genders.
- Current View:** API para teste - Instituto Joga Junto (selected), Overview, Authorization (selected), Pre-request Script, Tests, Variables, Runs.
- Authorization Tab:** Type dropdown set to "No Auth". A dropdown menu lists various authorization types: No Auth (selected), API Key, Bearer Token, JWT Bearer, Basic Auth, Digest Auth, OAuth 1.0, OAuth 2.0, Hawk Authentication, AWS Signature, NTLM Authentication [Beta], Akamai EdgeGrid.

## Executando a atividade:

1. A primeira etapa da atividade consiste em obter todos os endpoints existentes na documentação do sistema.

Um endpoint em uma API é um ponto de acesso específico que permite que os clientes façam solicitações e interajam com o sistema ou serviço. Geralmente, um endpoint é identificado por uma URL única e corresponde a uma ação ou recurso específico que a API oferece.

Cada endpoint é associado a um método HTTP, como GET, POST, PUT, PATCH ou DELETE, que determina a ação que será executada sobre os dados. Os endpoints definem como os recursos da API podem ser acessados e manipulados, permitindo que os clientes realizem operações como obter informações, enviar dados, atualizar, criar ou excluir recursos.

Nesse contexto os endpoints são representados por cada seção que descreve as diferentes operações que podem ser realizadas na API para teste - Instituto Joga Junto v1, como listar autores, criar livros, obter detalhes de gêneros, entre outros. Cada endpoint possui uma URL única e um método HTTP associado que permite aos usuários interagir com as funcionalidades oferecidas pela API.

Para facilitar a compreensão vou manter a variável "baseUrl", lembrando que seu conteúdo é <http://apilivro.jogajuntoinstituto.org:8000>.

## Autores (Authors) ↗

- **Listagem de Autores** 

- Método: GET
- Endpoint: {{baseUrl}}/authors/

- **Criação de Autor** 

- Método: POST
- Endpoint: {{baseUrl}}/authors/

- **Detalhes de Autor** 

- Método: GET
- Endpoint: {{baseUrl}}/authors/:id/

- **Atualização de Autor** 

- Método: PUT
- Endpoint: {{baseUrl}}/authors/:id/

- **Atualização Parcial de Autor** 

- Método: PATCH
- Endpoint: {{baseUrl}}/authors/:id/

- **Exclusão de Autor** 

- Método: DELETE
- Endpoint: {{baseUrl}}/authors/:id/

## Livros (Books)

- **Listagem de Livros** 

- Método: GET
- Endpoint: {{baseUrl}}/books

- **Criação de Livro** 

- Método: POST
- Endpoint: {{baseUrl}}/books/

- **Detalhes de Livro** 

- Método: GET
- Endpoint: {{baseUrl}}/books

- **Atualização de Livro** 

- Método: PUT
- Endpoint: {{baseUrl}}/books/:id/

- **Atualização Parcial de Livro** 

- Método: PATCH
- Endpoint: {{baseUrl}}/books/:id/

- **Exclusão de Livro ✗**

- Método: DELETE
- Endpoint: {{baseUrl}}/books/:id/

## Gêneros (Genders)

- **Listagem de Gêneros 🔍**

- Método: GET
- Endpoint: {{baseUrl}}/genders/

- **Criação de Gênero +**

- Método: POST
- Endpoint: {{baseUrl}}/genders/

- **Detalhes de Gênero 🔍**

- Método: GET
- Endpoint: {{baseUrl}}/genders/:id/

- **Atualização de Gênero 🖊**

- Método: PUT
- Endpoint: {{baseUrl}}/genders/:id/

- **Atualização Parcial de Gênero 🖊**

- Método: PATCH
- Endpoint: {{baseUrl}}/genders/:id/

- **Exclusão de Gênero ✗**

- Método: DELETE
- Endpoint: {{baseUrl}}/genders/:id/

---

2. A próxima etapa da atividade consiste em cadastrar 4 livros, deve-se entender primeiro os requisitos necessários para que um livro seja criado. Essa informação podemos encontrar no próprio Swagger:

**OBS: sempre que abrir uma seção no Swagger cliente em 'Try it Out' no canto superior da seção ou 'Cancel' seguido de 'Try it Out' para conseguir visualizar as informações conforme as imagens abaixo.**

**books**

**GET** /books/ **books\_list**

**POST** /books/ **books\_create** **Cancel**

Parameters

Name	Description
<b>data</b> * required	object (body)
	<pre>{   "title": "string",   "description": "string",   "author": 0,   "gender": 0 }</pre>

Perceba que para dar um comando POST, ou seja, para cadastrar um livro é necessário ter o ID de um "author" e de um "gender". O que significa que devemos cadastrá-los primeiro.

- Requisitos para cadastrar um "author":

**authors**

**GET** /authors/ **authors\_list**

**POST** /authors/ **authors\_create** **Cancel**

Parameters

Name	Description
<b>data</b> * required	object (body)
	<pre>{   "name": "string" }</pre>

- Requisitos para cadastrar um "gender":

**genders**

**GET** /genders/ **genders\_list**

**POST** /genders/ **genders\_create** **Cancel**

Parameters

Name	Description
<b>data</b> * required	object (body)
	<pre>{   "name": "string",   "squad": "Joga Junto" }</pre>

Após entendido o que é necessário para cada, vamos visualizar isso no POSTMAN:

- Criando um "author":

The screenshot shows the Postman interface with a collection named "API para teste - Instituto Joga Junto". A POST request is being made to `((baseUrl))/authors/`. The Body tab contains the following JSON:

```

1
2   "name": "Leanderson_squad02"
3

```

The Tests tab contains a script:

```

1 pm.test("Status code is 201", function () {
2   pm.response.to.have.status(201);
3 });

```

The Test Results section shows 1/1 test passed with a status code of 201.

- Criando um "gender":

The screenshot shows the Postman interface with a collection named "API para teste - Instituto Joga Junto". A POST request is being made to `((baseUrl))/genders/`. The Body tab contains the following JSON:

```

1
2   "name": "Ação da Squad 02",
3   "squad": "02"
4

```

The Tests tab contains a script:

```

1 pm.test("Status code is 201", function () {
2   pm.response.to.have.status(201);
3 });

```

The Test Results section shows 1/1 test passed with a status code of 201.

Após termos o author e o gender criados podemos por fim criar os livros:

obs: O IDs de "author" e "gender" vão aparecer no campo "Response" assim que forem criados.

The screenshot shows the Postman interface with a collection named "API para teste - Instituto Joga Junto". A POST request is selected for the "books" endpoint. The body is set to "raw" and contains the following JSON:

```

1
2   "title": "A Aventuras da Squad 2 . 4",
3   "description": "Esse quarto livro foi criado nessa API com propósito de concluir a atividade de
4     número 15",
5   "author": 31,
6   "gender": 22

```

The screenshot shows the Postman interface with the same setup as the previous one. The response body is displayed in "Pretty" format:

```

1
2   "id": 60,
3   "title": "A Aventuras da Squad 2 . 5",
4   "description": "Esse quinto livro foi criado nessa API com propósito de dar um print mostrando que
5     deu certo",
6   "author": 31,
7   "gender": 22

```

The screenshot shows the Postman interface with the "Tests" tab selected. A test script is present:

```

1 pm.test("Status code is 201", function () {
2   pm.response.to.have.status(201);
3 });

```

The "Test Results" tab shows 1/1 result: PASS Status code is 201.

obs: Aqui você tem a opção de criar 1 por 1, ou os 4 livros de uma vez.

---

3. Dando um GET em "books" podemos confirmar que os livros já foram criados(essa já é a terceira etapa da atividade):

The screenshot shows a Postman collection named "API para teste - Instituto Joga Junto". The "books" endpoint is selected. A GET request is made to `http://{{baseUrl}}/books/`. The response status is 200 OK, and the response body is a JSON array containing four book objects:

```

[{"id": 50, "title": "A Aventuras da Squad 2", "description": "Esse livro foi criado nessa API com propósito de concluir a atividade de número 15", "author": 31, "gender": 22}, {"id": 51, "title": "A Aventuras da Squad 2 . 2", "description": "Esse segundo livro foi criado nessa API com propósito de concluir a atividade de número 15", "author": 31, "gender": 22}, {"id": 52, "title": "A Aventuras da Squad 2 . 3", "description": "Esse terceiro livro foi criado nessa API com propósito de concluir a atividade de número 15", "author": 31, "gender": 22}, {"id": 53, "title": "A Aventuras da Squad 2 . 4", "description": "Esse quarto livro foi criado nessa API com propósito de concluir a atividade de número 15", "author": 31, "gender": 22}

```

The screenshot shows the same Postman collection and request as the previous one. The "Tests" tab contains the following script:

```
pm.test("Status code is 200", function () {
  pm.response.to.have.status(200);
});
```

The test result is PASS: Status code is 200.

4. A ultima etapa dessa atividade pede para adicionar o teste a seguir: -> `pm.test("Status code in 200", function(){pm.response.to.have.status(200);});` <-

Nos prints anteriores percebe-se que essa parte já foi feita, portanto vou usar esse espaço para dar as referências de onde você pode consultar informações relacionadas.

O primeiro é a documentação do Postman tratando sobre scripts de teste:

<https://learning.postman.com/docs/writing-scripts/script-references/test-examples/>

Para o caso de um GET:

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```

Para o caso de um POST:

```
pm.test("Status code is 201", function () {
    pm.response.to.have.status(201);
});
```

Como eu sei onde o código de resposta é 200 ou 201? Os códigos de respostas HHTP seguem sempre o mesmo padrão, você pode encontrar vários exemplos aqui: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

Abaixo são os que foram mais usados nessa atividade:

# 200 OK

The HTTP `200 OK` success status response code indicates that the request has succeeded. A 200 response is cacheable by default.

The meaning of a success depends on the HTTP request method:

- `GET` : The resource has been fetched and is transmitted in the message body.
- `HEAD` : The representation headers are included in the response without any message body
- `POST` : The resource describing the result of the action is transmitted in the message body
- `TRACE` : The message body contains the request message as received by the server.

The successful result of a `PUT` or a `DELETE` is often not a `200 OK` but a `204 No Content` (or a `201 Created` when the resource is uploaded for the first time).

## Status

HTTP



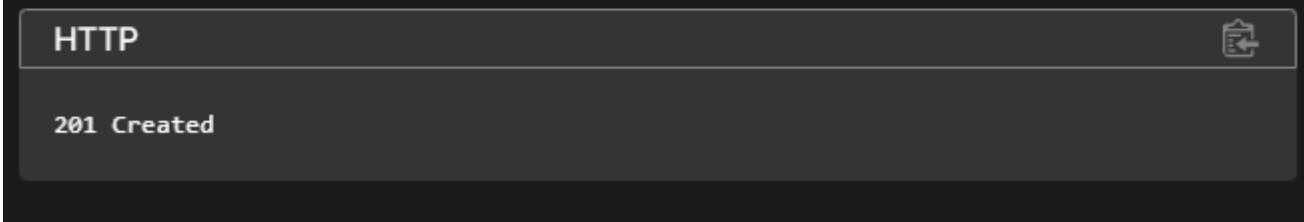
`200 OK`

# 201 Created

The HTTP `201 Created` success status response code indicates that the request has succeeded and has led to the creation of a resource. The new resource, or a description and link to the new resource, is effectively created before the response is sent back and the newly created items are returned in the body of the message, located at either the URL of the request, or at the URL in the value of the `Location` header.

The common use case of this status code is as the result of a [POST](#) request.

# Status



Eu espero que esse repositório te ajude não só a entregar a atividade, mas também a entender como as coisas foram feitas e como elas funcionam.

## Integrantes da Squad:

| Beatriz Souza | [Bruno Soares](#) | [Leanderson Lima](#) | [Rebeca Borges](#) | Sara Cruz |