

Consumindo e criando  
**testes** para sua **API** like a  
boss com **Postman**



# Quem sou eu?



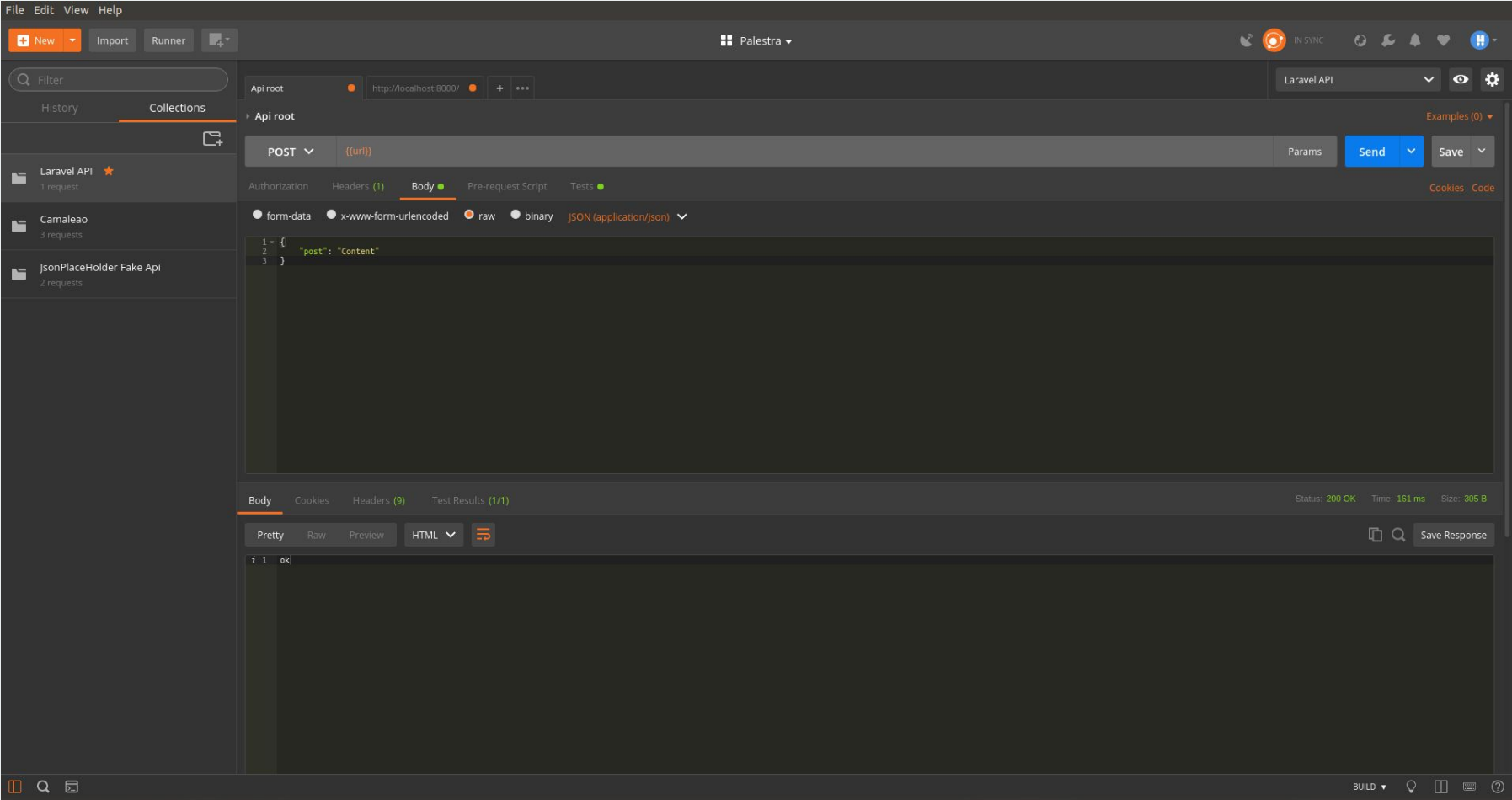
- Cientista da computação e programador PHP fullstack há 7 anos.
- Especializado em e-commerce
- Desenvolvedor na Mundipagg

# Agenda

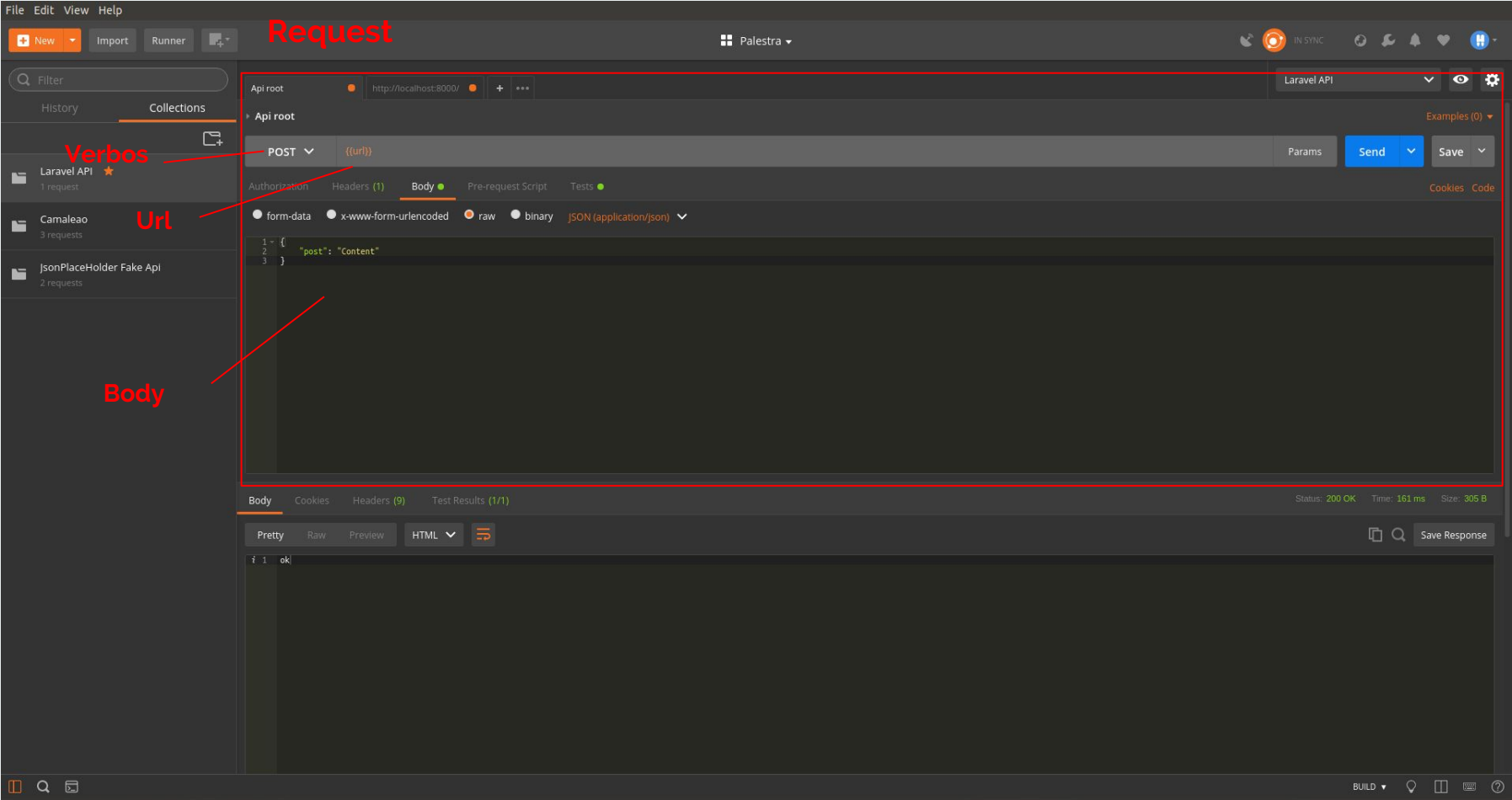


- Interface e requests básicos
- Code snippets
- API do Postman
- Environments
- Testes vs Pre-requests
- Testes
- Pre-requests
- O que dá pra fazer?
- Escrevendo testes
- Runner
- Postman monitor
- Gerando docs a partir de collections
- Newman

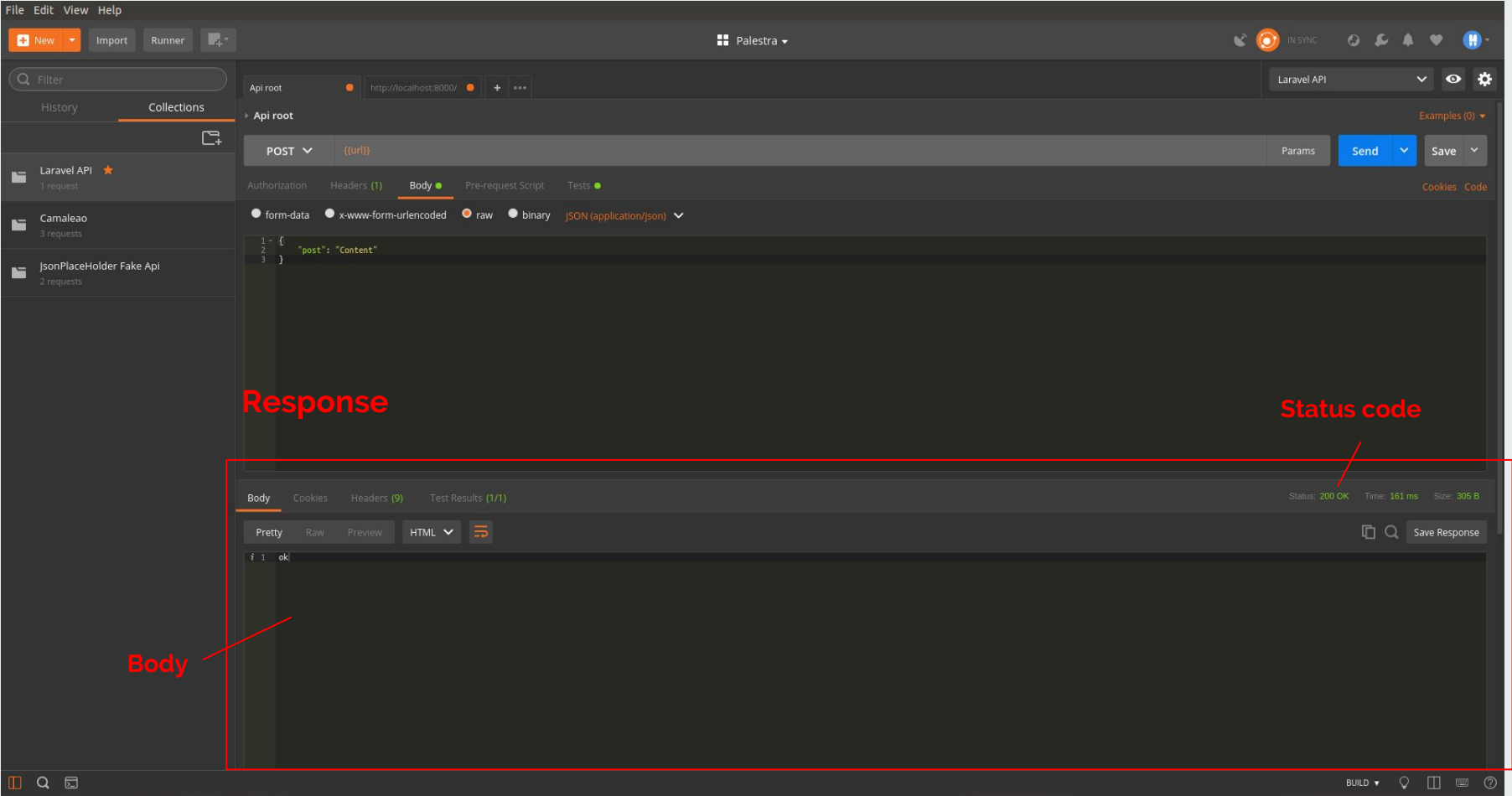
# Interface



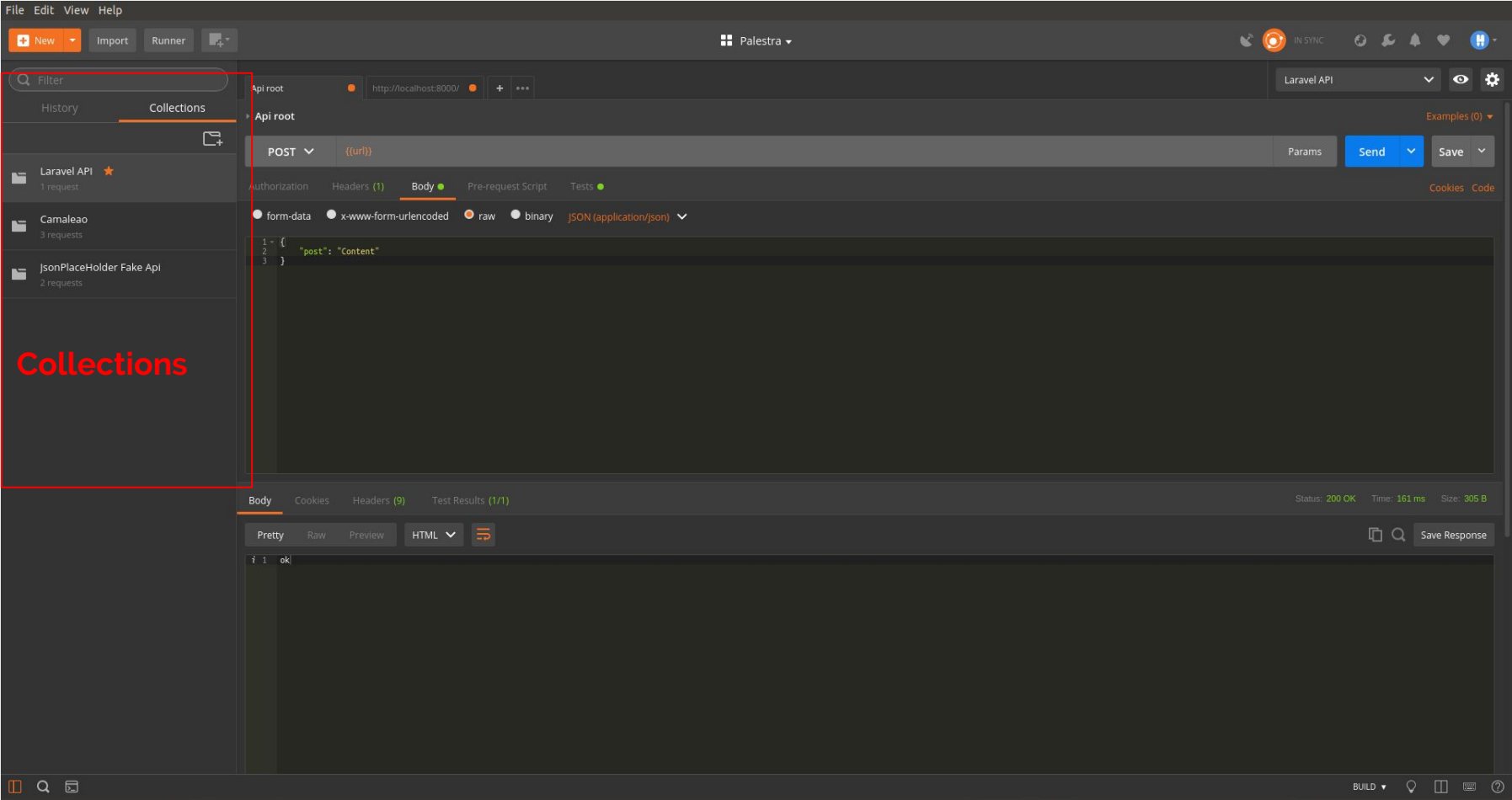
# Interface



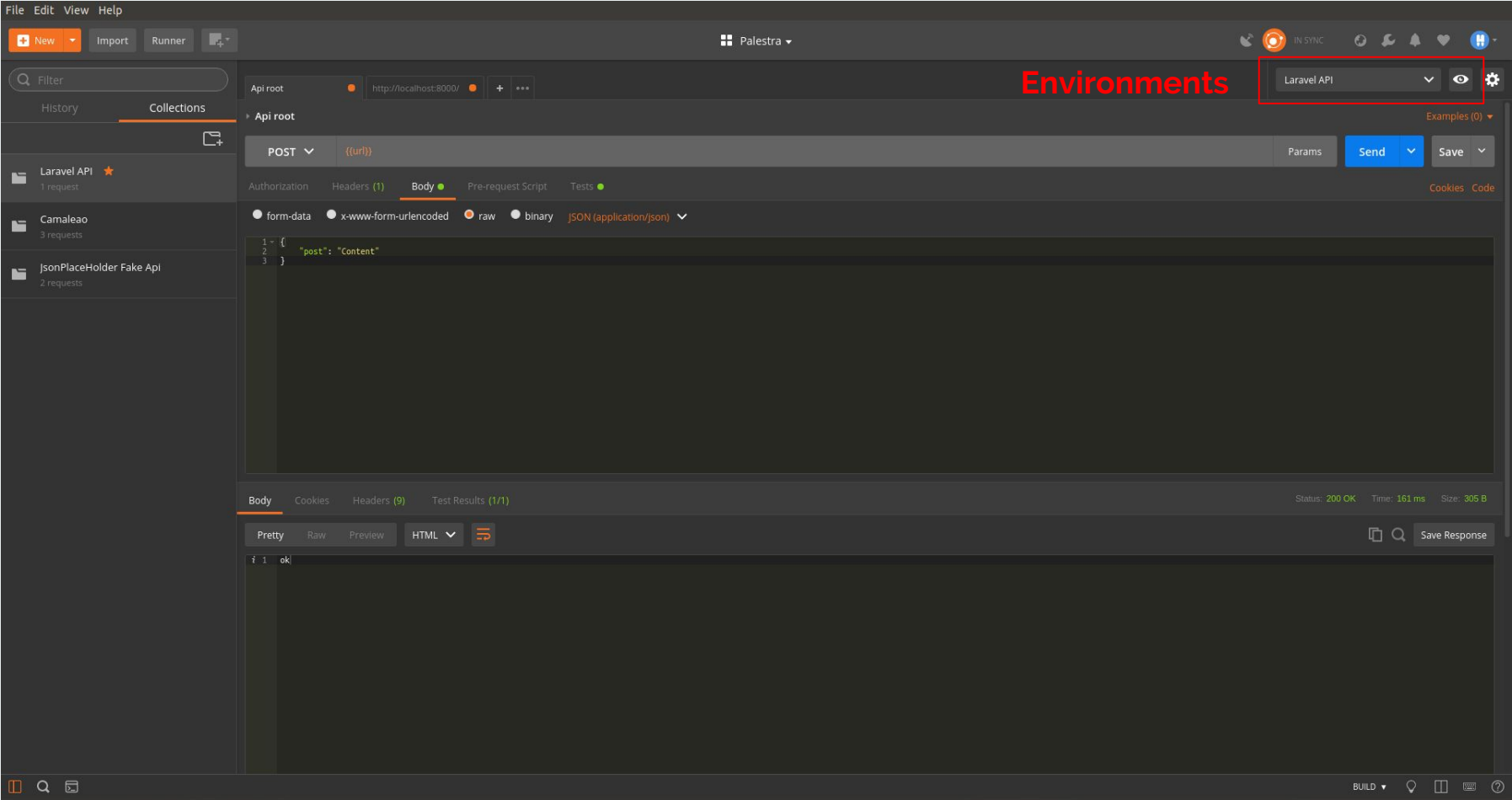
# Interface



# Interface

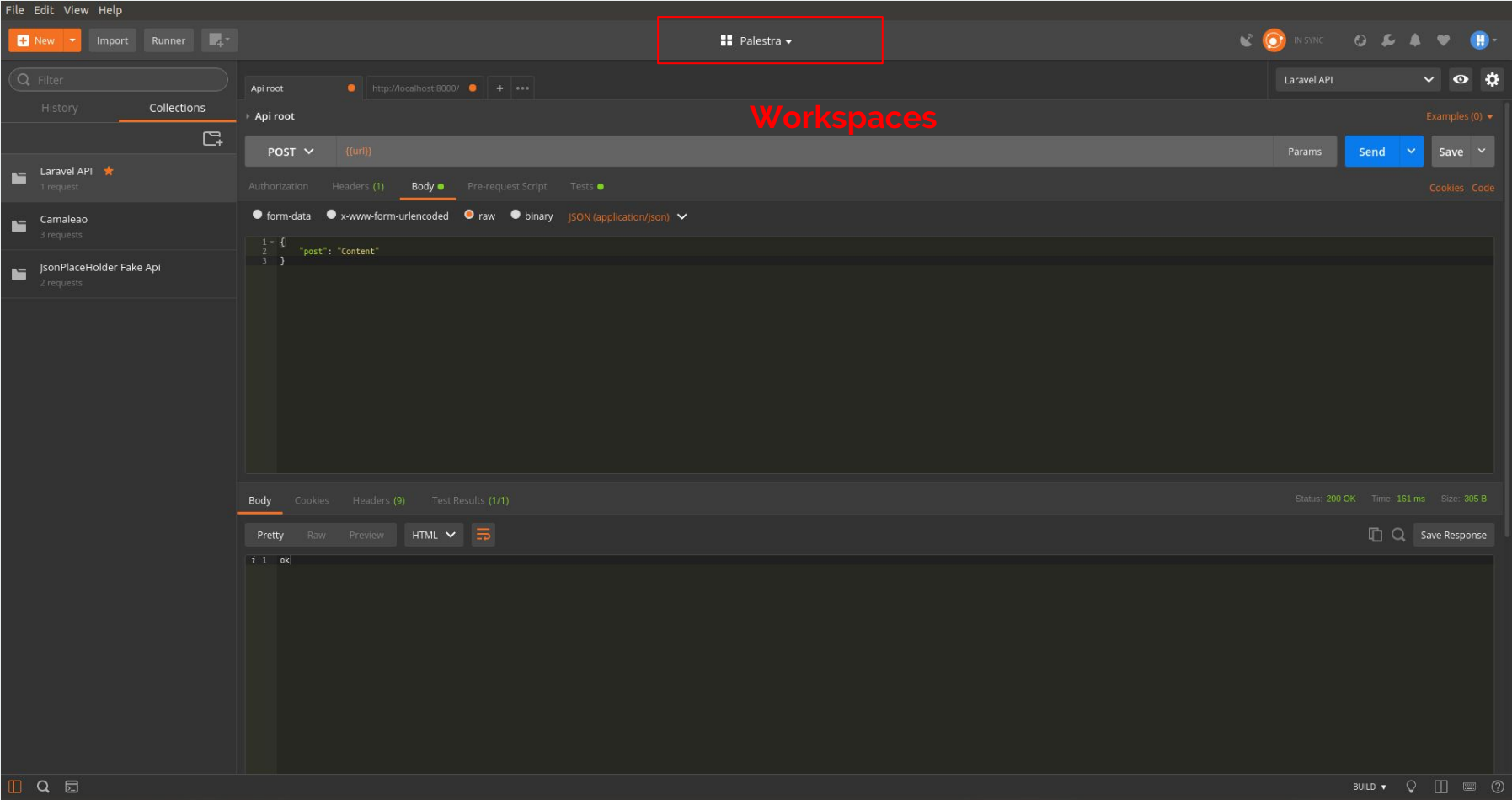


# Interface

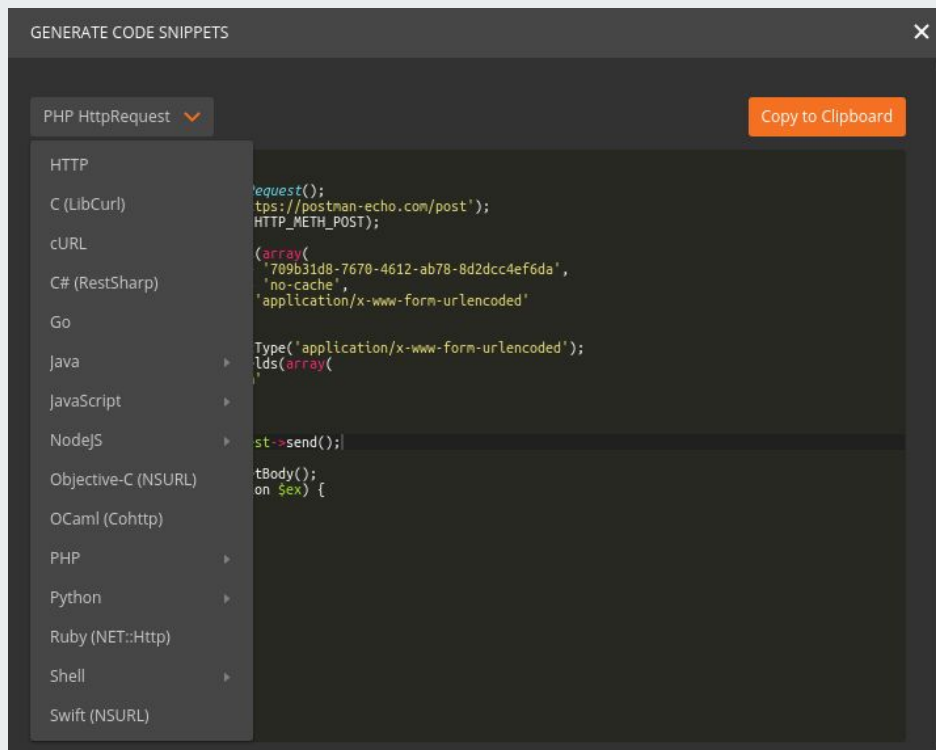




# Interface




# Code snippets



Da aba de requisições para sua linguagem favorita.


# Api do Postman


“A API do Postman permite que você acesse programaticamente os dados armazenados na conta do Postman com facilidade.”




### Postman API

An API to access your Postman data

 Backup

 Notifications

 0 Existing API keys

DetailsExisting API Keys

The Postman API has several endpoints to help you integrate Postman even more deeply with your development toolchain. You can add new [collections](#), update existing collections, update [environments](#), add and run [monitors](#) directly through the API. This API allows you to programmatically access data stored in your Postman account with ease. Get started with the API by clicking the Run in Postman button at the top of the [Postman API documentation](#) page and use the Postman app to send requests.

#### Available Integrations

##### Get Postman API Key

Use the Postman API key to authorize your requests to the Postman API

+ Get API Key

# Environments

Permite definir e obter valores em variáveis que podem ser utilizada em todos os seus requests dentro de uma collection.

Por ex:

**Edit Environment**

Laravel API

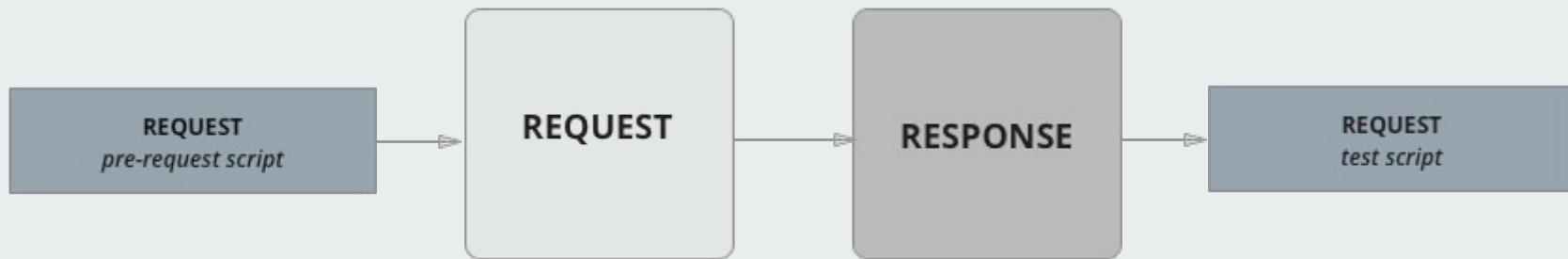
	Key	Value	Bulk Edit
<input checked="" type="checkbox"/>	url	http://localhost:8000/api/	

GET ▾

`{{url}}/users`

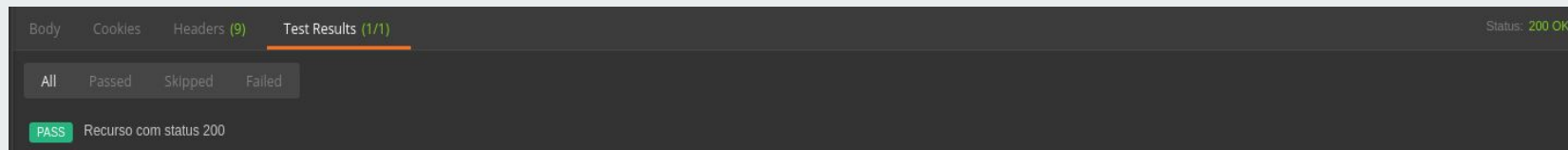
# Testes vs Pre-requests

A grande diferença está no momento em que são executados. Antes ou depois da request



# Testes

Scripts escritos em javascript que são usados para validar a resposta de um request



# Pre-requests



- São escritos da mesma maneira que os testes, exceto pela ausência do `pm.response`
- Muito usados para preencher variáveis com timestamp e valores randômicos

# O que dá pra fazer?



- Verificar status code (2xx, 4xx, 5xx, etc)
- Deu 200, mas o teste falhou
- Verificar tempo de resposta
- Pegar/setar valor de uma variável de ambiente
- Verificar conteúdo retornado no body da response
- Verificar json schema com biblioteca Tiny Validator for JSON
- Validar header da response
- Enviar uma request via teste



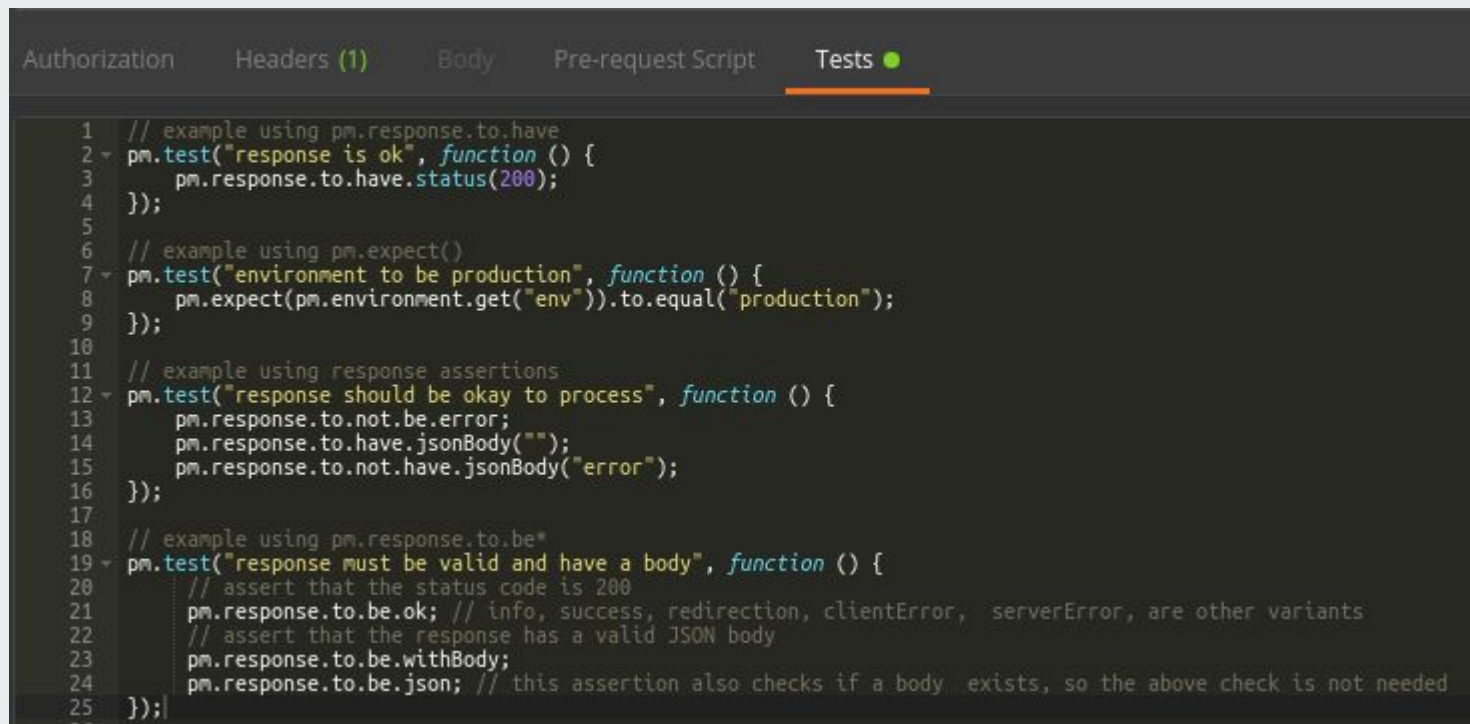
# Escrevendo testes



## Tipos

- `pm.test`
- `pm.environment`
- `pm.globals`
- `pm.variables`
- `pm.expect`
- `pm.response`
- `pm.sendRequest`

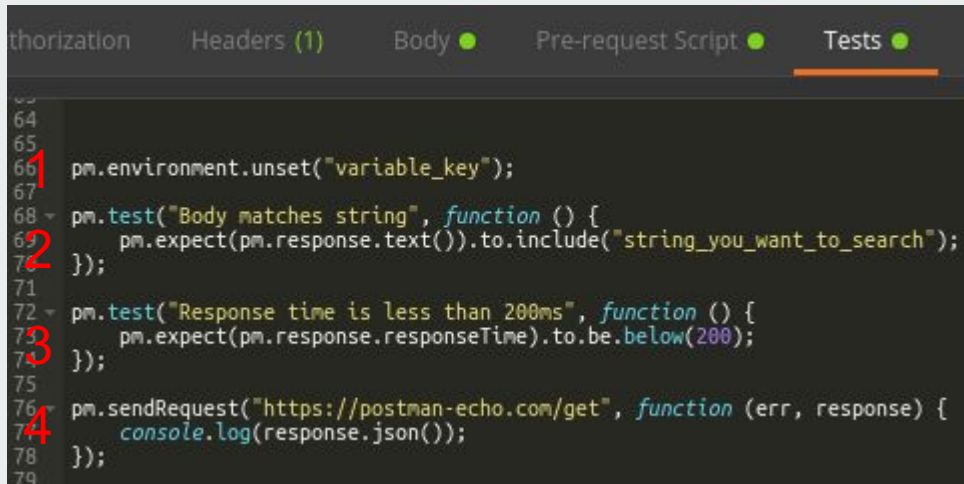
# Escrevendo testes



The screenshot shows a REST client interface with a dark theme. At the top, there are tabs for 'Authorization', 'Headers (1)', 'Body', 'Pre-request Script', and 'Tests'. The 'Tests' tab is selected and highlighted with an orange underline. Below the tabs, there is a list of test cases, each starting with a line number and a comment. The test cases are as follows:

```
1 // example using pm.response.to.have
2 pm.test("response is ok", function () {
3     pm.response.to.have.status(200);
4 });
5
6 // example using pm.expect()
7 pm.test("environment to be production", function () {
8     pm.expect(pm.environment.get("env")).to.equal("production");
9 });
10
11 // example using response assertions
12 pm.test("response should be okay to process", function () {
13     pm.response.to.not.be.error;
14     pm.response.to.have.jsonBody("");
15     pm.response.to.not.have.jsonBody("error");
16 });
17
18 // example using pm.response.to.be*
19 pm.test("response must be valid and have a body", function () {
20     // assert that the status code is 200
21     pm.response.to.be.ok; // info, success, redirection, clientError, serverError, are other variants
22     // assert that the response has a valid JSON body
23     pm.response.to.be.withBody;
24     pm.response.to.be.json; // this assertion also checks if a body exists, so the above check is not needed
25 });
```

# Escrevendo testes



The screenshot shows the Postman 'Tests' tab with the following code. Red numbers 1 through 4 are placed to the left of specific lines of code to indicate the steps described in the adjacent list:

```
63  
64  
65  
66 1 pm.environment.unset("variable_key");  
67  
68 2 pm.test("Body matches string", function () {  
69     pm.expect(pm.response.text()).to.include("string_you_want_to_search");  
70 });  
71  
72 3 pm.test("Response time is less than 200ms", function () {  
73     pm.expect(pm.response.responseTime).to.be.below(200);  
74 });  
75  
76 4 pm.sendRequest("https://postman-echo.com/get", function (err, response) {  
77     console.log(response.json());  
78 });  
79
```

1 - Limpa variável de ambiente

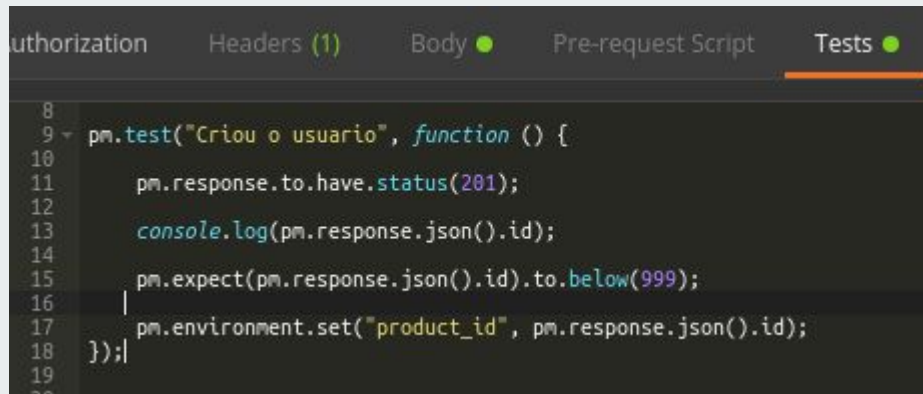
2 - Procura string no body response

3 - Tempo de resposta menor que 200ms

4 - Envia uma nova requisição

# Escrevendo testes

## Exemplo



The screenshot shows a REST client interface with four tabs: 'Authorization', 'Headers (1)', 'Body', and 'Tests'. The 'Tests' tab is selected and highlighted with an orange underline. Below the tabs, a code editor displays a JavaScript test script. The script is as follows:

```
8
9 pm.test("Criou o usuario", function () {
10
11     pm.response.to.have.status(201);
12
13     console.log(pm.response.json().id);
14
15     pm.expect(pm.response.json().id).to.below(999);
16
17     pm.environment.set("product_id", pm.response.json().id);
18 });
19
20
```

# Escrevendo testes

## Exemplo



```
8  
9 pm.test("Criou o usuario", function () {  
10  
11     2 pm.response.to.have.status(201);  
12  
13     3 console.log(pm.response.json().id);  
14  
15     4 pm.expect(pm.response.json().id).to.below(999);  
16  
17     5 pm.environment.set("product_id", pm.response.json().id);  
18 });  
19  
20
```

The screenshot shows a code editor with a dark theme. The 'Tests' tab is selected and highlighted with an orange underline. The code is a PM2 test script. Five red numbers are placed above specific lines of code: '1' above the opening curly brace of the test function, '2' above the status assertion, '3' above the console log, '4' above the expect assertion, and '5' above the environment set call.

1 - Mensagem em caso de sucesso

2 - Espera status 201 (created)

3 - Loga o id no console

4 - Espera um id menor que 999

5 - Seta variável de ambiente com o id do produto

# Collection runner

Collection Runner

Palestra

Run In Command Line

Docs

Choose a collection or folder

Search for a collection or folder

< Laravel API

GETs

Environment

No Environment

Iterations

1

Delay

0

ms

Log Responses

For all requests

Data

Select File

☐

Persist Variables

Run Laravel API

Recent Runs

Type to Filter

Import Test Run

<div></div>	JsonPlaceholder Fake Api	Palestra	PASSED	6 Apr, 2018
<div></div>	JsonPlaceholder Fake Api	No Environment	PASSED	6 Apr, 2018
<div></div>	JsonPlaceholder Fake Api	Palestra	PASSED	6 Apr, 2018
<div></div>	JsonPlaceholder Fake Api	Palestra	PASSED	6 Apr, 2018
<div></div>	JsonPlaceholder Fake Api	Palestra	PASSED	6 Apr, 2018

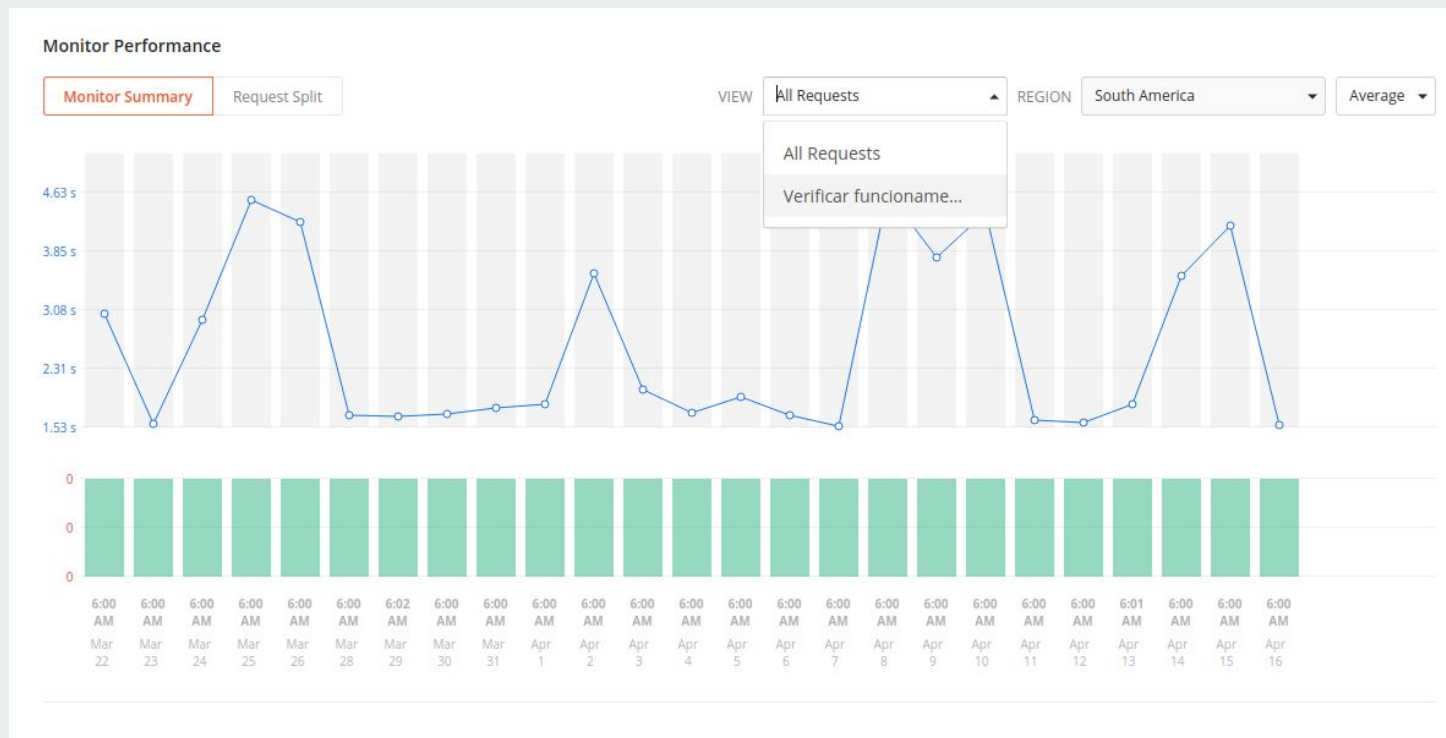
# Collection runner

The screenshot displays the Collection Runner interface for a collection named "Laravel API". At the top, there are two circular progress indicators: a green one showing "7 PASSED" and a red one showing "0 FAILED". To the right of these indicators, the text "Laravel API" and "just now" are visible. Further right, there are four buttons: "Run Summary" (orange), "Export Results" (grey), "Retry" (blue), and "New" (grey).

The main area of the interface is a table listing the test results for "Iteration 1". Each row represents a test case, with columns for the test name, the URL, the status, the response time, and the response size. The tests are as follows:

Test Name	URL	Status	Response Time	Response Size
POST Create product	http://localhost:8000/api/... / Product / Post / Create product	201 Created	67 ms	223 B
PASS	Criou o produto			
PASS	Id existe e tem que ser menor que 999			
POST Create product (Bad request)	http://localhost:8000/api/... ost / Create product (Bad request)	400 Bad Request	8 ms	57 B
PASS	Não conseguiu criar produto com descrição maior igual a 500 caracteres			
GET Get products	http://localhost:8000/api/... API / Product / Get / Get products	200 OK	9 ms	2.06 KB
PASS	Pegou lista de produtos			
GET Get product	http://localhost:8000/api/... I API / Product / Get / Get product	200 OK	10 ms	226 B
PASS	Retornou produto pelo id			
PASS	Tempo de resposta menor que 50ms			
GET Api root	http://localhost:8000/api/... ravel API / Product / Get / Api root	200 OK	7 ms	1.858 KB
PASS	Api root OK			
PUT Update product	http://localhost:8000/api/... I / Product / Put / Update product	200 OK	51 ms	226 B
This request does not have any tests.				

# Postman monitor





# Gerando docs a partir de collections

The image shows the Postman application interface. On the left, the sidebar displays the 'Laravel API' collection with sub-items 'Introduction', 'Posts', and 'Gets'. The main panel shows the 'Laravel API' collection details, including a 'Posts' section with a 'POST Create product' endpoint. The URL is 'http://localhost:8000/api/products/'. The 'HEADERS' section shows 'Content-Type' as 'application/json'. The 'BODY' section contains a JSON object: 

```
{  "name": "Iphone",  "description": "iPhone is a revolutionary new mobile phone that allows you to make a call by simply tappi"}
```

. On the right, a 'Sample Request' tab is open, showing a PHP script for creating a product using cURL: 

```
<?php

$curl = curl_init();

curl_setopt_array($curl, array(
  CURLOPT_URL => "http://localhost:8000/api/products/",
  CURLOPT_RETURNTRANSFER => true,
  CURLOPT_ENCODING => "",
  CURLOPT_MAXREDIRS => 10,
  CURLOPT_TIMEOUT => 30,
```

# Newman

```
→ Create product (Bad request)
POST http://localhost:8000/api/products/ [400 Bad Request, 369B, 6ms]
✓ Não conseguiu criar produto com descrição maior igual a 500 caracteres

[] Product / Get
→ Get products
GET http://localhost:8000/api/products/ [200 OK, 1.85KB, 8ms]
✓ Pegou lista de produtos

→ Get product
GET http://localhost:8000/api/products/43 [200 OK, 521B, 8ms]
✓ Retornou produto pelo id
✓ Tempo de resposta menor que 50ms

→ Api root
GET http://localhost:8000/api/ [200 OK, 2.11KB, 5ms]
✓ Api root OK

[] Product / Put
→ Update product
PUT http://localhost:8000/api/products/43 [200 OK, 521B, 37ms]
```

	executed	failed
iterations	1	0
requests	6	0
test-scripts	12	0
prerequisite-scripts	11	0
assertions	7	0
total run duration: 422ms		
total data received: 4.09KB (approx)		
average response time: 23ms		

Rodando os testes de suas collections com o Newman

```
newman run MyCollection.json -e Environment.json
```



# Contato



[github.com/michelp1](https://github.com/michelp1)



[@Michel\\_PL](https://twitter.com/Michel_PL)



# Útil

- Postman (<https://www.getpostman.com>)
- Tiny Validator for JSON (<https://github.com/geraintluff/tv4>)
- Docs da api do Postman (<https://docs.api.getpostman.com>)
- Repositório da palestra (<https://github.com/michelp/palestras>)