

Postman: Introduction

Processamento Estruturado de Informação (LEI/LSIRC)

Estruturação e Organização de Informação (LSIG)

What is Postman

- Postman is a **collaboration** platform for API development;
- Postman can help if you are **developing** APIs as well;
- Postman allows you very quickly create a **request** with the required **HTTP method** and **parameters**, submit the request and easily **inspect** the results.

What is Postman

- REST APIs are **everywhere** nowadays but at the same time they are getting **more complex** to get started with:
 - different HTTP methods;
 - Headers;
 - Cookies;
 - File uploads
 - Authentication with api keys, tokens, OAuth, (...)

Why Use Postman?

- **Use of Collections** - Postman lets users **create collections** for their API calls. Each collection can create **subfolders** and **multiple requests**;
- **Collaboration** - **Collections** and **environments** can be imported or **exported** making it easy to share files.
- **Creating Environments** - Having multiple environments aids in less repetition of tests as one can use the same collection but for a different environment. This is where parameterization will take place which we will discuss in further lessons.

Why Use Postman?

- **Creation of Tests** - Test checkpoints such as **verifying** for successful HTTP response status can be added to each API calls which help ensure test coverage;
- **Automation Testing** - Through the use of the **Collection Runner** or Newman, tests can be run in **multiple iterations** saving time for repetitive tests;
- **Debugging** - Postman console helps to check what data has been retrieved making it easy to debug tests.

Send a request

- Most everybody knows Postman as an HTTP client – a handy way to **send a request** and **receive a response**;
- Example using the amiibo API for a **GET** request:
 - Set your HTTP request to GET.
 - In the request URL field, input link
 - Click Send
 - You will see 200 OK Message
 - There should be the results in the body which indicates that the request was successful.

Send a request

The screenshot displays a REST client interface with a red box highlighting the request configuration and the response body. The request is a GET to `https://www.amiiboapi.com/api/amiibo?character=zelda`. The response status is 200 OK, and the body is a JSON object containing details for the character Zelda.

Request Configuration:

- Method: GET
- URL: `https://www.amiiboapi.com/api/amiibo?character=zelda`

Query Params:

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> character	zelda	
Key	Value	Description

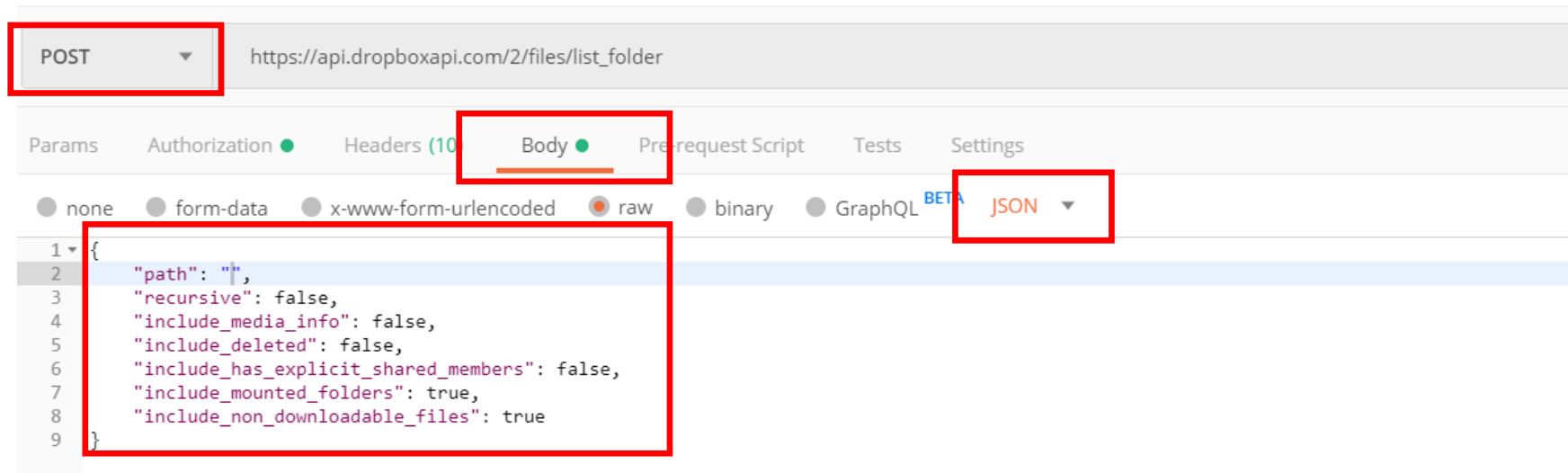
Response Status: Status: 200 OK, Time: 1180ms, Size: 641 B

Response Body (JSON):

```
1 {
2   "amiibo": [
3     {
4       "amiiboSeries": "Super Smash Bros.",
5       "character": "Zelda",
6       "gameSeries": "The Legend of Zelda",
7       "head": "01010000",
8       "image": "https://raw.githubusercontent.com/N3evin/AmiiboAPI/master/images/icon_01010000-000e0002.png",
9       "name": "Zelda",
10      "release": {
11        "au": "2014-12-12",
12        "eu": "2014-12-19",
13        "jp": "2014-12-06",
14        "na": "2014-12-14"
15      },
16      "tail": "000e0002",
17      "type": "Figure"
18    }
19  ]
20 }
```

with POST Requests

- Post requests are different from Get request as there is data manipulation with the user adding data to the endpoint;
- Using the same data from the previous tutorial in Get request, let's now add our own user.



Working with POST Requests

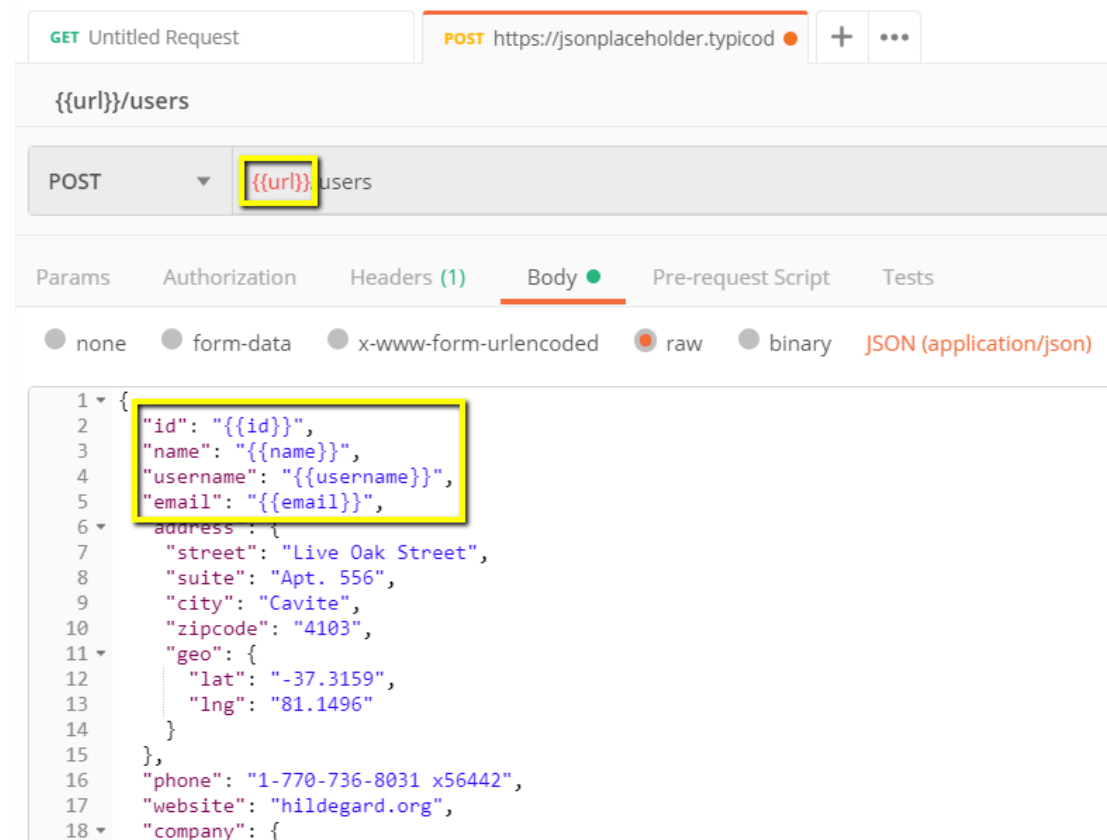
- There are different **content types** defined by W3C:
 - **x-www-form-urlencoded**: to send simple **text/ASCII** data by URL;
 - **form-data**: to send **non-ASCII text or large binary data**;
 - **Raw** to send **plain text** or **JSON** or any other kind of string. The **content-type** (application/json) header defines the format;
 - **Binary** can be used when you want to **attach** non-textual data to the request, e.g. a video/audio file, images, or any other binary data file.

Parameterize Requests

- **Data Parameterization** is one of the most useful features of Postman;
- Instead of creating the **same requests** with different data, you can use **variables** with parameters;
- These **data** can be **from** a **data file** or an **environment variable**;
- Parameterization helps to **avoid repetition**.

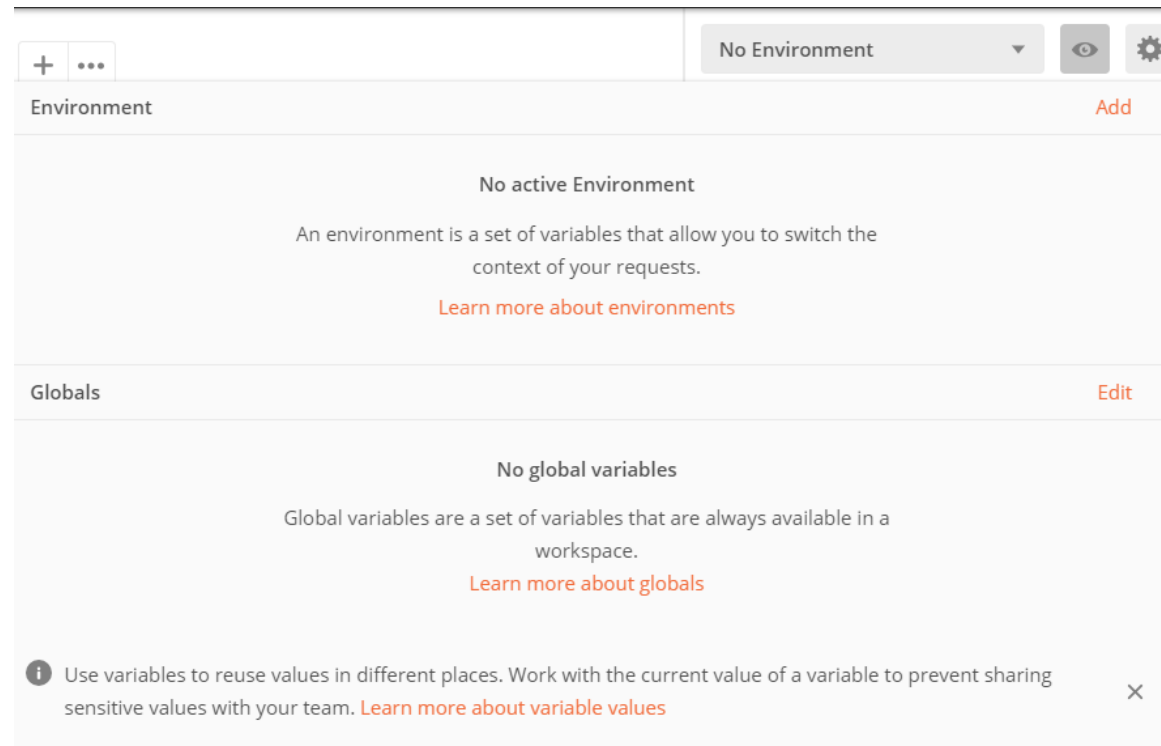
Parameterize Requests

- Parameters are created through the use of double curly brackets: `{{sample}}`.
- Example:



Parameterize Requests

- To use the parameter you need to set the **environment**:

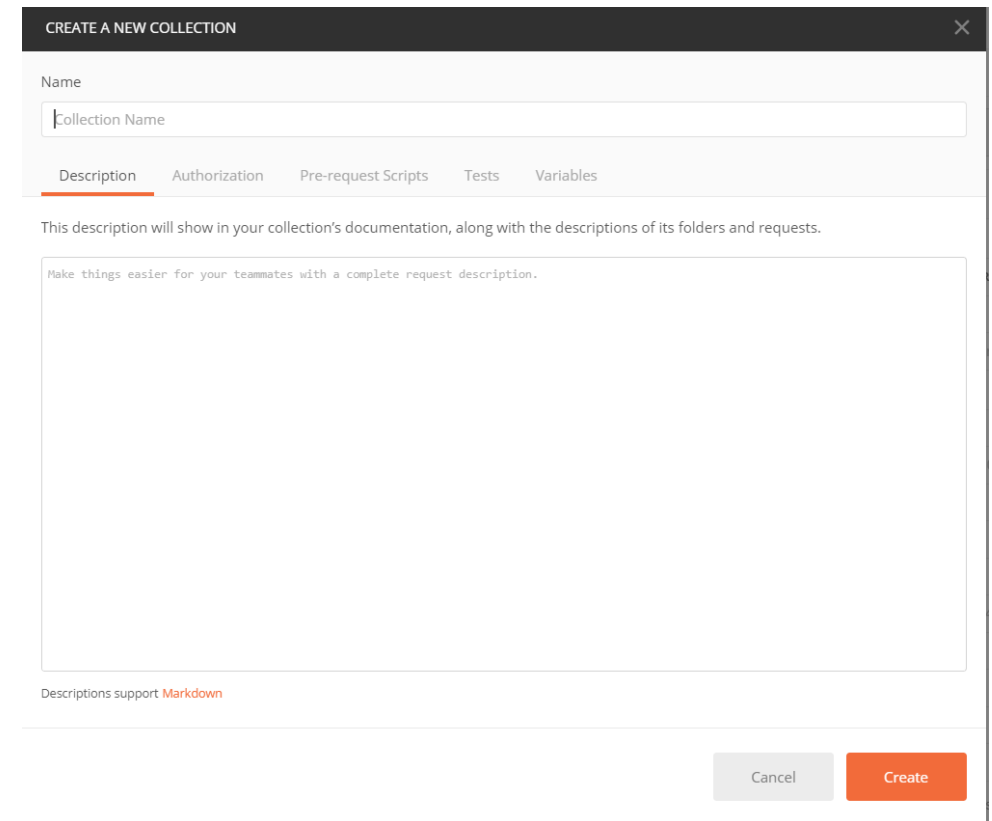


Postman Tests

- Postman **Tests** are JavaScript codes added to requests that verify results;
- We can use this section to create **environment variables**:
 - `pm.environment.set("XMLresponse", responseBody);`
 - This code sets an environment variable (XMLresponse) with the request's **response Body**;

How to Create Collections

- A Postman Collection lets you **group individual requests** together;
- You can organize these **requests** into folders;



The screenshot shows the 'CREATE A NEW COLLECTION' dialog box in Postman. It has a dark header bar with the title 'CREATE A NEW COLLECTION' and a close button. Below the header, there is a 'Name' field with a placeholder 'Collection Name'. Underneath the name field is a tabbed interface with five tabs: 'Description' (which is selected and underlined), 'Authorization', 'Pre-request Scripts', 'Tests', and 'Variables'. Below the tabs, there is a text area for the description. Above the text area, a note states: 'This description will show in your collection's documentation, along with the descriptions of its folders and requests.' Inside the text area, there is a placeholder text: 'Make things easier for your teammates with a complete request description.' At the bottom of the dialog, there is a small note: 'Descriptions support [Markdown](#)'. At the very bottom right, there are two buttons: 'Cancel' and 'Create'.

How to Create Collections

SAVE REQUEST

Requests in Postman are saved in collections (a group of requests).
[Learn more about creating collections](#)

Request name

Request description (Optional)

Make things easier for your teammates with a complete request description.

Descriptions support [Markdown](#)

Select a collection or folder to save to:

Search for a collection or folder

All Collections

+ Create Collection

dropbox

Cancel

Save

Comments (0)

Send Save

Cookies Code

DESCRIPTION ... Bulk Edit Presets

Description

2j-AHvMVEcAAAAAAwpaG0W2HLIT5g4IXLI5GkaZg-WDB_EITxIR6Kr-f1rRge

Runtime/7.17.1

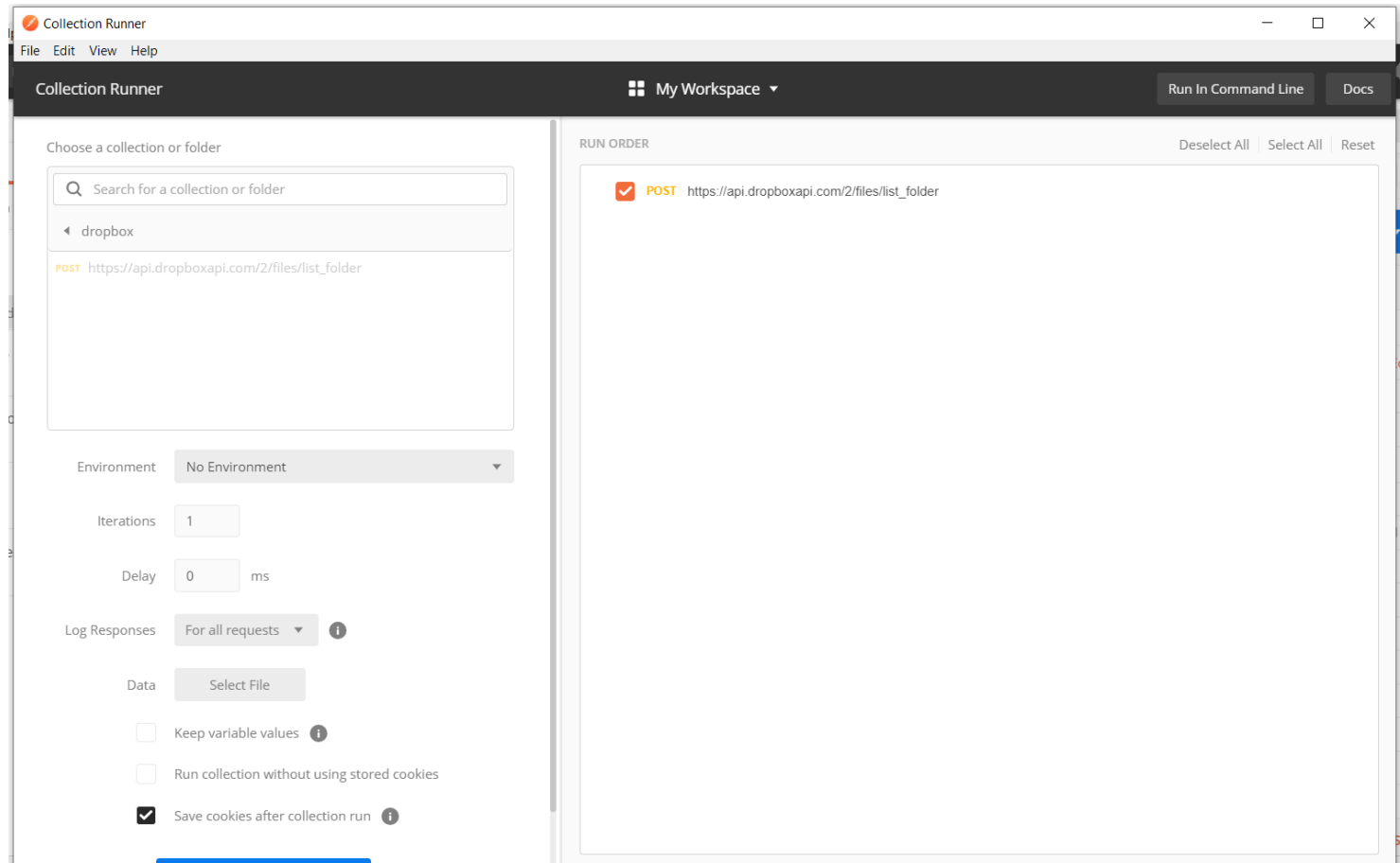
7-7e5f-4230-821f-15572c2ee4f3

boxapi.com

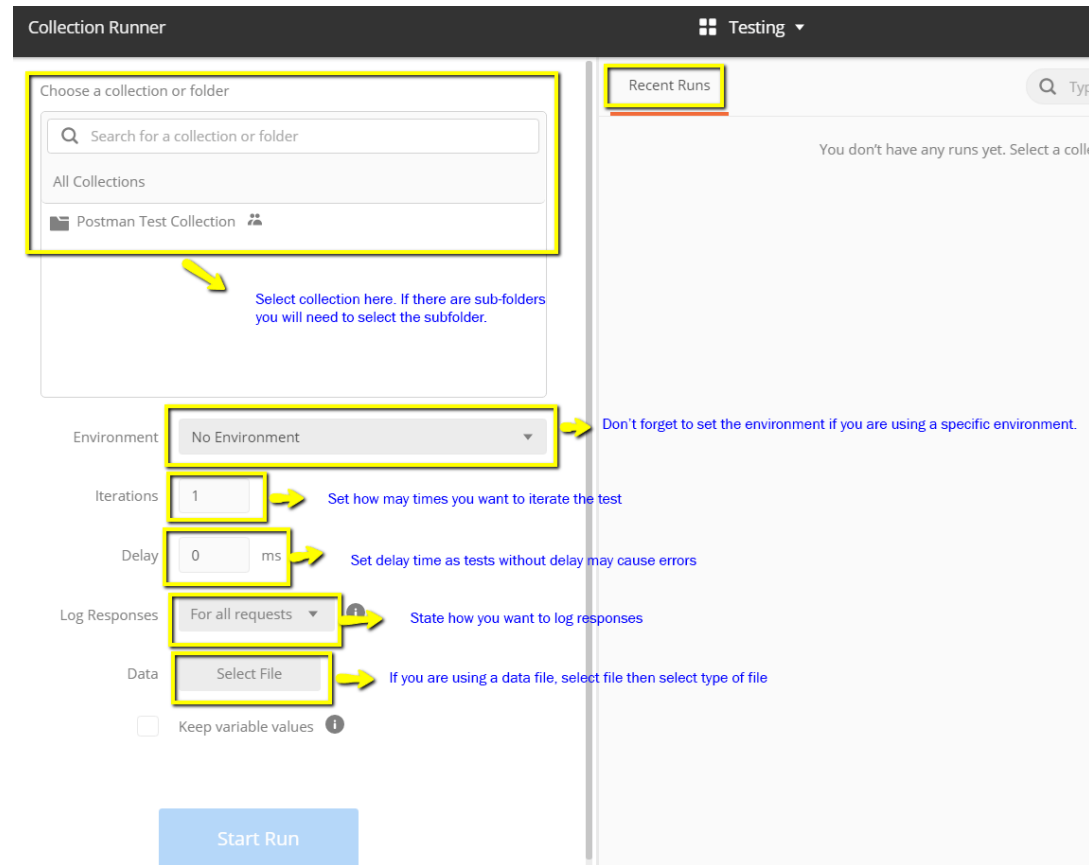
late

How to Run Collections

- The **Collection Runner** allows you to run sets of requests in a specified **sequence**;
- The **Collection Runner** will log your request test results, and your scripts **can pass data between requests** as well as altering the request workflow.



How to Run Collections using Collection



How to Run Collections using Collection

The screenshot displays the Postman Collection Runner interface. At the top, the 'Run Results' tab is active. The collection name 'Postman Test Collection' is shown, along with 'No Environment'. A summary box indicates 6 PASSED and 0 FAILED tests. A yellow box highlights this summary, with a red circle '1' and an arrow pointing to it. Below the summary, the test results for 'Iteration 1' are shown. A yellow box highlights the first test case, which includes a GET request and two assertions (Status code is 200 and Check if user with id1 is Leanne Graham). A red circle '2' and an arrow point to this box. Another yellow box highlights the second test case, which is a POST request with no tests. A red circle '3' and an arrow point to this box. The results for 'Iteration 2' are also visible, showing similar test cases and results.

Iteration	Test Case	Status	Message	Response Code	Response Time	Response Size
Iteration 1	GET {{url}}/users	200 OK	840 ms	5.645 KB		
	PASS	Status code is 200				
	PASS	Check if user with id1 is Leanne Graham				
	POST {{url}}/users	201 Created	931 ms	487 B		
Iteration 2	GET {{url}}/users	200 OK	79 ms	5.645 KB		
	PASS	Status code is 200				
	PASS	Check if user with id1 is Leanne Graham				
	POST {{url}}/users	201 Created	823 ms	487 B		

Bibliografia/referências

- Referências Web:

- https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Basico_sobre_HTTP
- <https://www.restapitutorial.com/>
- http://www.swennenhuis.nl/basexfordummies/BaseX_for_dummies.pdf
- https://docs.basex.org/wiki/Main_Page
- <https://docs.basex.org/wiki/RESTXQ>
- <https://www.postman.com/>

- Livro:

- Anders M. and Michel S., An introduction to XML and Web Technologies, Addison-Wesley, 2006;