



kasperanalytics.com

9123820085

# Postman

✉ hr@kasperanalytics.com

in kasper-analytics

# TableofContents

<b>1. Postman – Introduction .....</b>	<b>1</b>
Need of Postman .....	1
Working with Postman.....	2
<b>2. Postman – Environment Setup .....</b>	<b>7</b>
Standalone Application.....	7
Chrome Extension.....	11
<b>3. Postman – Environment Variables .....</b>	<b>16</b>
Create Environment .....	16
Environment Variables Scope.....	18
<b>4. Postman – Authorization .....</b>	<b>20</b>
Types of Authorization .....	22
Authorization at Collections .....	27
<b>5. Postman – Workflows.....</b>	<b>30</b>
<b>6. Postman – GET Requests.....</b>	<b>35</b>
Create a GET Request .....	35
<b>7. Postman – POST Requests .....</b>	<b>41</b>
Create a POST Request .....	41
<b>8. Postman – PUT Requests.....</b>	<b>47</b>
Create a PUT Request .....	48
<b>9. Postman – DELETE Requests .....</b>	<b>51</b>
Create a DELETE Request.....	51

<b>10. Postman – Create Tests for CRUD.....</b>	<b>54</b>
Tests in Postman.....	54
<b>11. Postman – Create Collections .....</b>	<b>57</b>
Create a New Collection.....	57
<b>12. Postman – Parameterize Requests.....</b>	<b>60</b>
Create a Parameter Request .....	60
<b>13. Postman – Collection Runner .....</b>	<b>63</b>
Execute Tests with Collection Runner .....	63
<b>14. Postman – Assertion.....</b>	<b>66</b>
Writing Assertions .....	66
Assertion for Object Verification.....	68
Assertion Types.....	69
<b>15. Postman – Mock Server.....</b>	<b>71</b>
Benefits of Mock Server .....	71
Mock Server Creation .....	71
Example Request .....	77
<b>16. Postman – Cookies .....</b>	<b>83</b>
Cookies Management .....	83
Access Cookies via Program.....	85
<b>17. Postman – Sessions .....</b>	<b>87</b>
<b>18. Postman – Newman Overview .....</b>	<b>89</b>
Newman Installation.....	89
<b>19. Postman – Run Collections using Newman .....</b>	<b>91</b>

Run Collections .....	91
Common command-line arguments for Newman.....	93
<b>20. Postman – OAuth 2.0 Authorization .....</b>	<b>95</b>

## 1. Postman – Introduction

Postman is an Application Programming Interface (API) testing tool. API acts like an interface between a couple of applications and establishes a connection between them.

Thus, an API is a collection of agreements, functions, and tools that an application can provide to its users for successful communication with another application. We requireban API whenever we access an application like checking news over the phone, Facebook, and so on.

Postman was designed in the year 2012 by software developer and entrepreneur Abhinav Asthana to make API development and testing straightforward. It is a tool for testing the software of an API. It can be used to design, document, verify, create, and change APIs.

Postman has the feature of sending and observing the Hypertext Transfer Protocol (HTTP) requests and responses. It has a graphical user interface (GUI) and can be used in platforms like Linux, Windows and Mac. It can build multiple HTTP requests – POST, PUT, GET, PATCH and translate them to code.

### NeedofPostman

Postman has a huge user base and has become a very popular tool because of the reasons listed below:

- Postman comes without any licensing cost and is suitable for use for the teams with any capacity.
- Postman can be used very easily by just downloading it.
- Postman can be accessed very easily by logging into your own account after installation on the device.
- Postman allows easy maintenance of test suites with the help of collections.

Users can make a collection of API calls which can have varied requests and sub-folders.

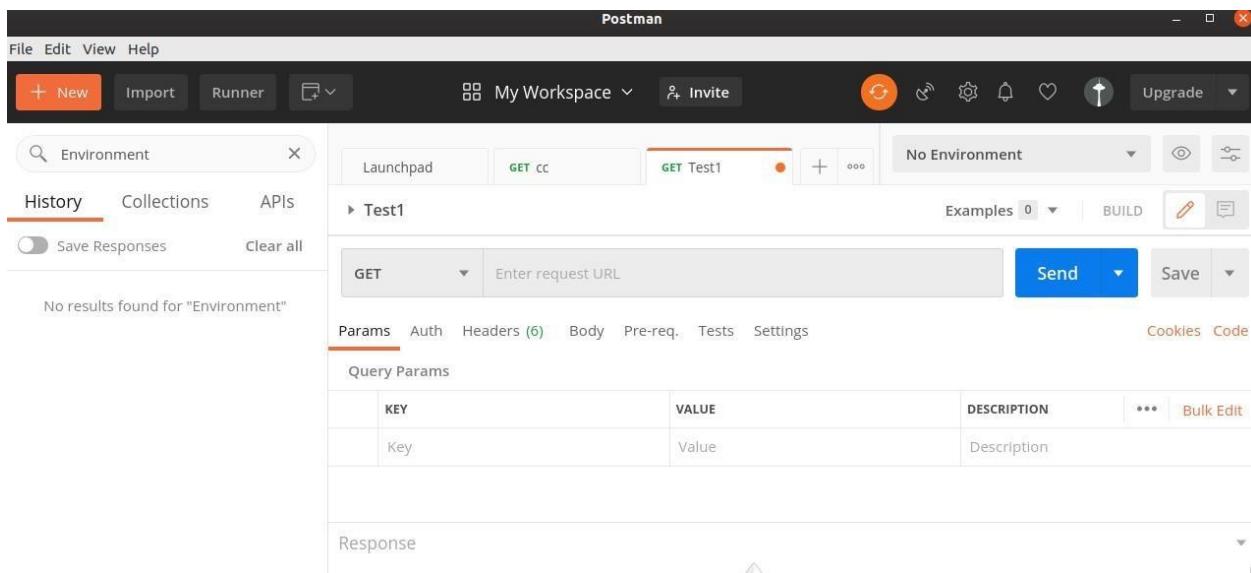
- Postman is capable of building multiple API calls like SOAP, REST, and HTTP.
- Postman can be used for test development by addition of checkpoints to HTTP response codes and other parameters.
- Postman can be integrated with the continuous integration and either continuous delivery or continuous deployment (CI/CD) pipeline.
- Postman can be integrated with Newman or Collection Runner which allows executing tests in much iteration. Thus we can avoid repeated tests.
- Postman has big community support.
- The Postman console allows debugging test steps.
- With Postman, we can create more than one environment. Thus, a single collection can be used with various configurations.
- Postman gives the option to import/export Environments and Collections, enabling easy sharing of files.

## **WorkingwithPostman**

To start working with Postman, we have the navigations as shown below. It primarily consists of four sections:

- Header
- Response
- Sidebar
- Builder

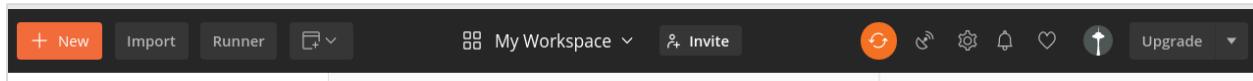
Given below is the screenshot of the navigations available in Postman:



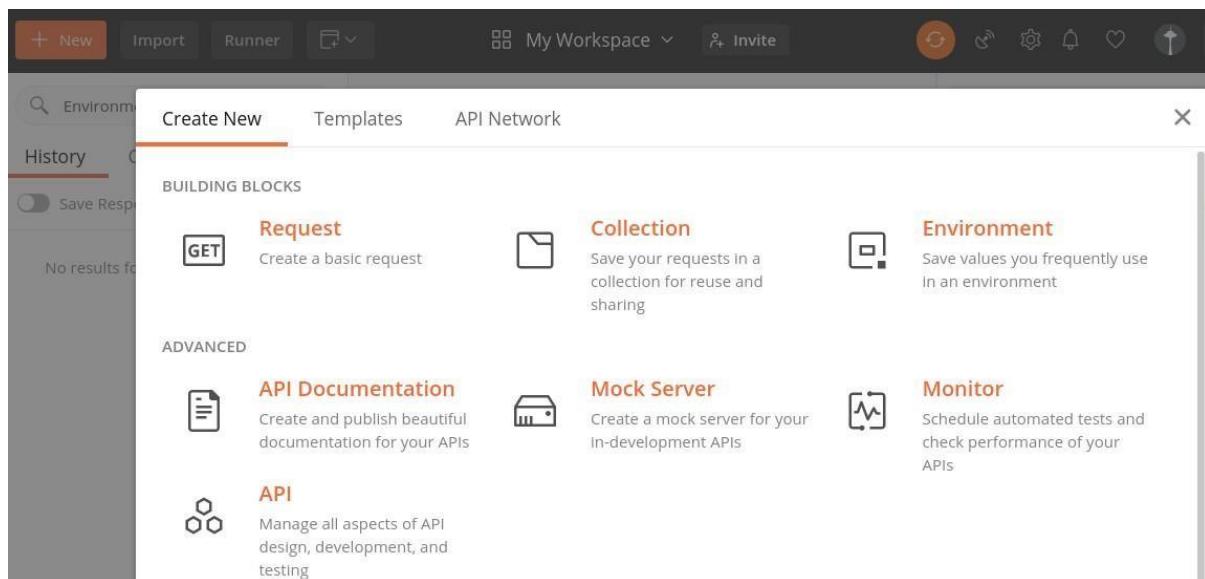
The screenshot shows the Postman application window. The top navigation bar includes File, Edit, View, Help, + New, Import, Runner, My Workspace (set to "My Workspace"), Invite, and Upgrade. Below the bar, there's a toolbar with icons for Environment, History, Collections, APIs, Save Responses, and Clear all. The main workspace shows "Test1" with a GET request labeled "GET Test1". The request editor has "GET" selected, an "Enter request URL" field, and a "Send" button. Below the request editor are tabs for Params, Auth, Headers (6), Body, Pre-req., Tests, and Settings. A "Query Params" table is shown with one row: Key (Value) and Description (Description). At the bottom is a "Response" section.

## Header

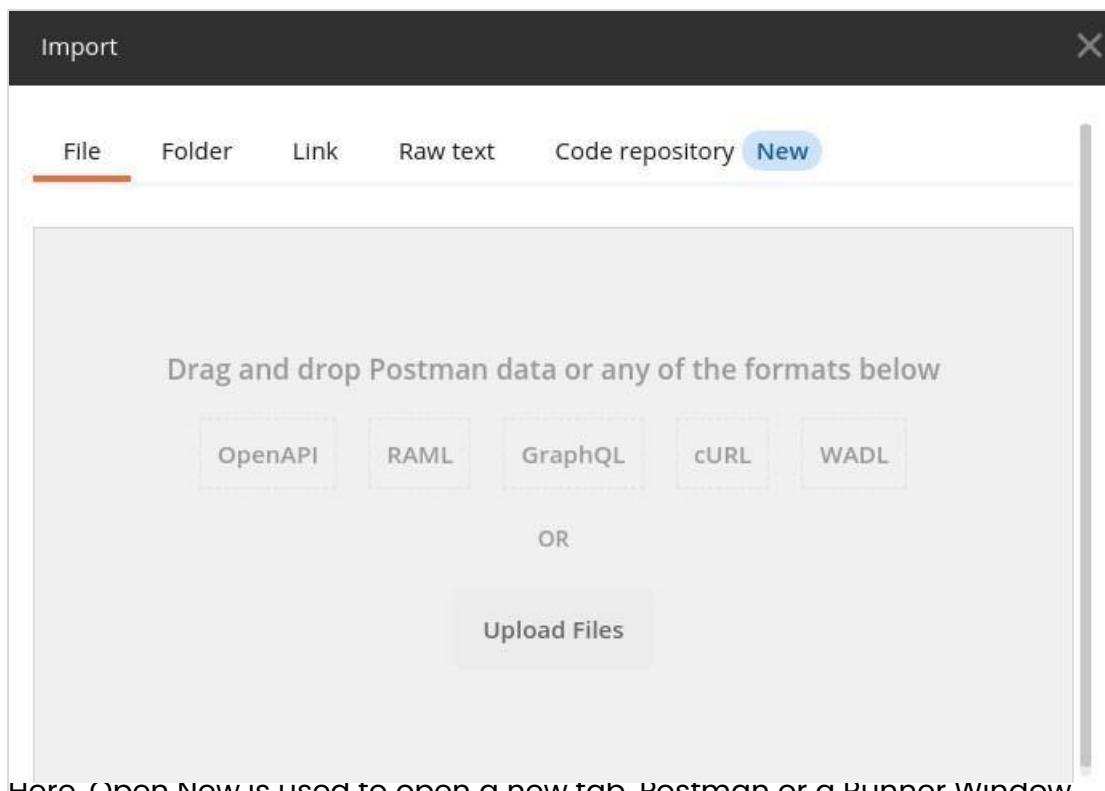
Postman consists of New, Import, Runner (used to execute tests with Collection Runner), Open New, Interceptor, Sync menus, and so on. It shows the workspace name – My Workspace along with the option for Invite for sharing it among teams.



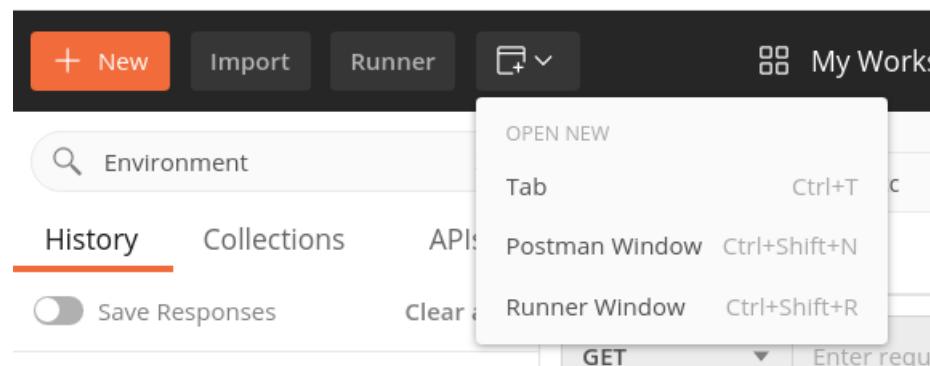
New menu is used to create a new Environment, Collection or request. The Import menu helps to import an Environment/Collection.



We can import from a File, Folder, Link, Raw text or from Code repository options which are also available under Import.

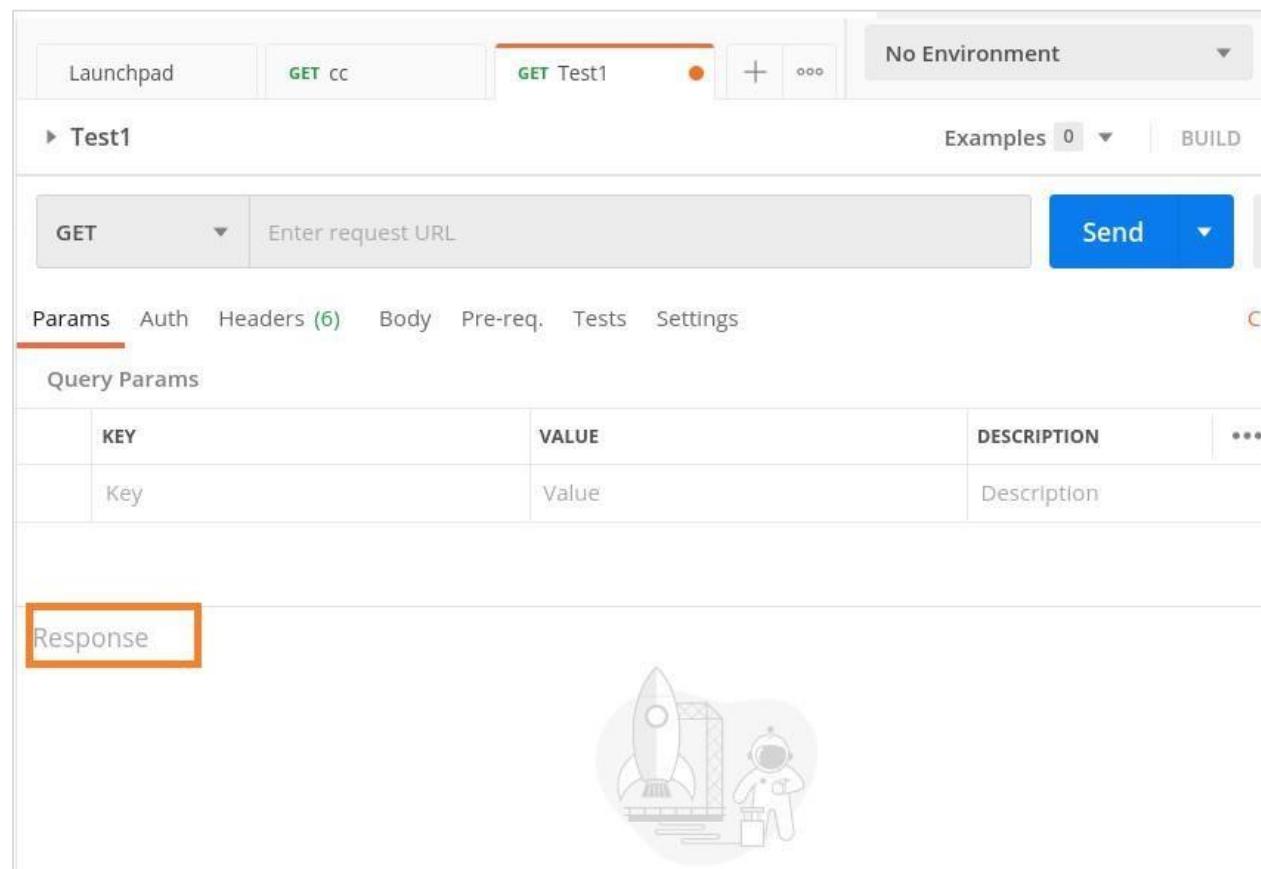


Here, Open New is used to open a new tab, Postman or a Runner Window.



## Response

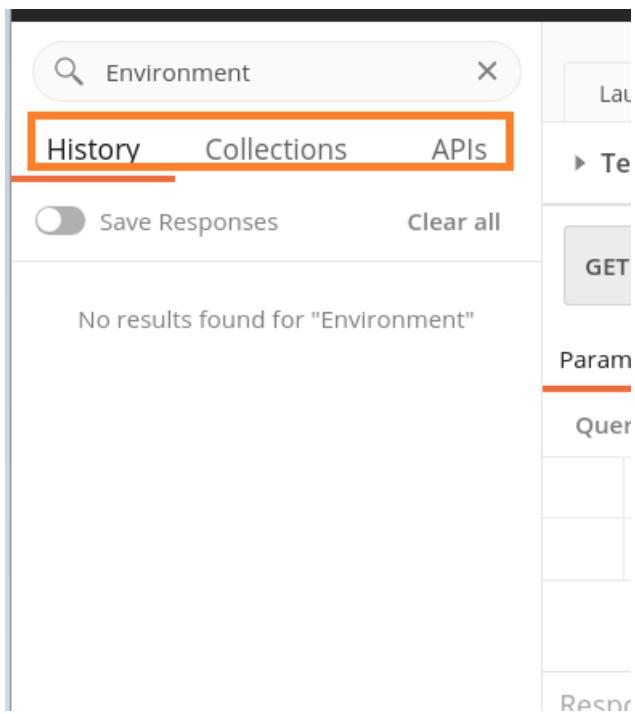
Response section shall have values populated only when a request is made. It generally contains the Response details.



KEY	VALUE	DESCRIPTION	...
Key	Value	Description	

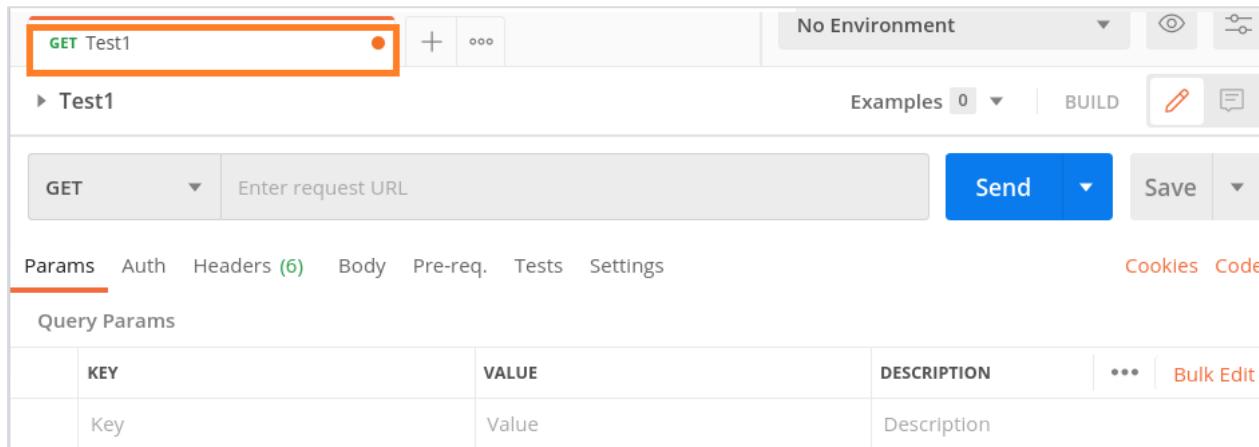
## Sidebar

Sidebar consists of Collections (used to maintain tests, containing folders, sub-folders, requests), History (records all API requests made in the past), and APIs.



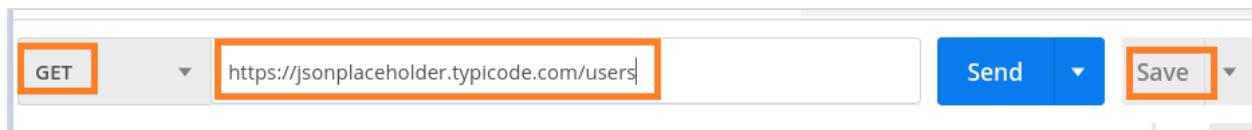
## Builder

Builder is the most important section of the Postman application. It has the request tab and displays the current request name. By default, Untitled Request is mentioned if no title is provided to a request.



KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

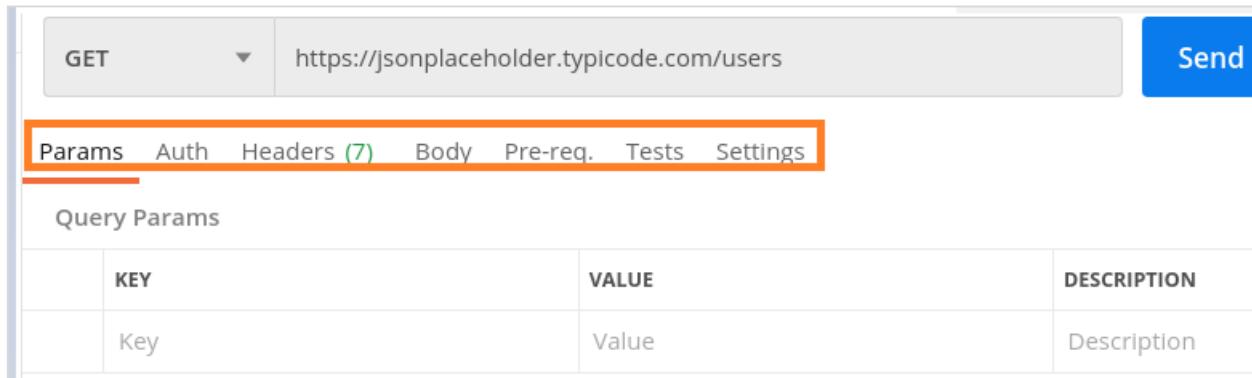
The Builder section also contains the request type (GET, POST, PUT, and so on) and URL. A request is executed with the Send button. If there are any modifications done to a request, we can save it with the Save button.



The Builder section has the tabs like Param, Authorization, Headers, Body, Pre-req., Tests and Settings. The parameters of a request in a key-value pair are mentioned within the Params tab. The Authorization for an API with username, password, tokens, and so on are within the Authorization tab.

The request headers, body are defined within the Headers and Body tab respectively. Sometimes, there are pre-condition scripts to be executed prior to a request. These are mentioned within the Pre-req. tab.

The Tests tab contains scripts that are run when a request is triggered. This helps to validate if the API is working properly and the obtained data and Response code is correct.



KEY	VALUE	DESCRIPTION
Key	Value	Description

## 2. Postman – Environment Setup

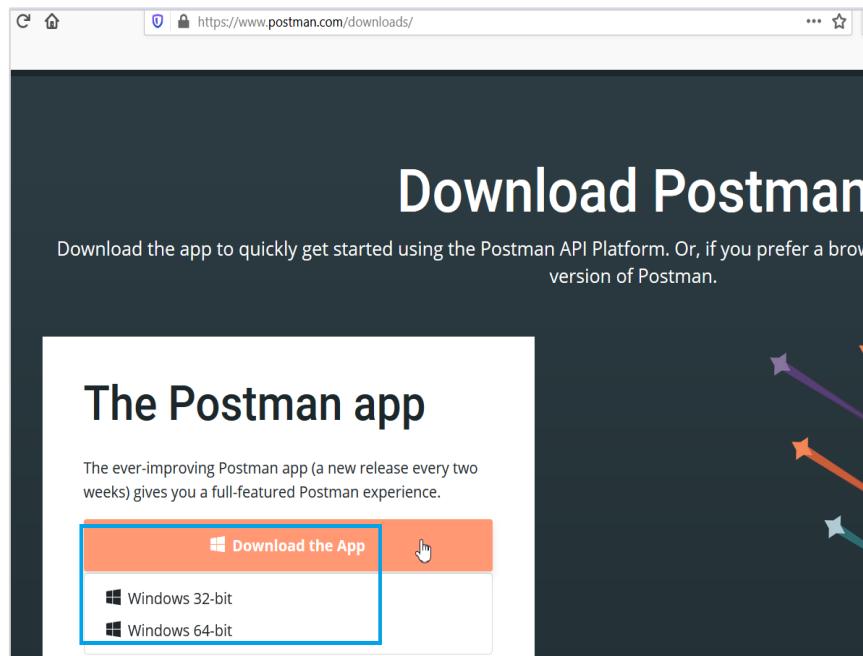
Postman can be installed in operating systems like Mac, Windows and Linux. It is basically an independent application which can be installed in the following ways:

- Postman can be installed from the Chrome Extension (will be available only in Chrome browser).
- It can be installed as a standalone application.

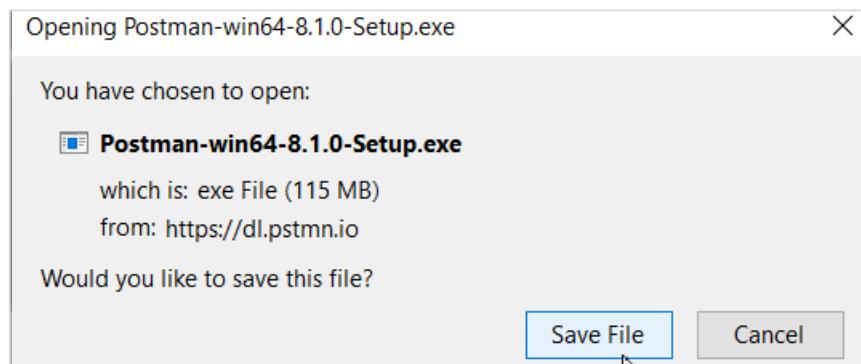
### Standalone Application

To download Postman as a standalone application in Windows, navigate to the following link <https://www.postman.com/downloads/>

Then, click on Download the App button. As per the configuration of the operating system, select either the Windows 32-bit or Windows 64-bit option.



The pop-up to save the executable file gets opened. Click on Save File.



As the download is completed successfully, the executable file gets generated.

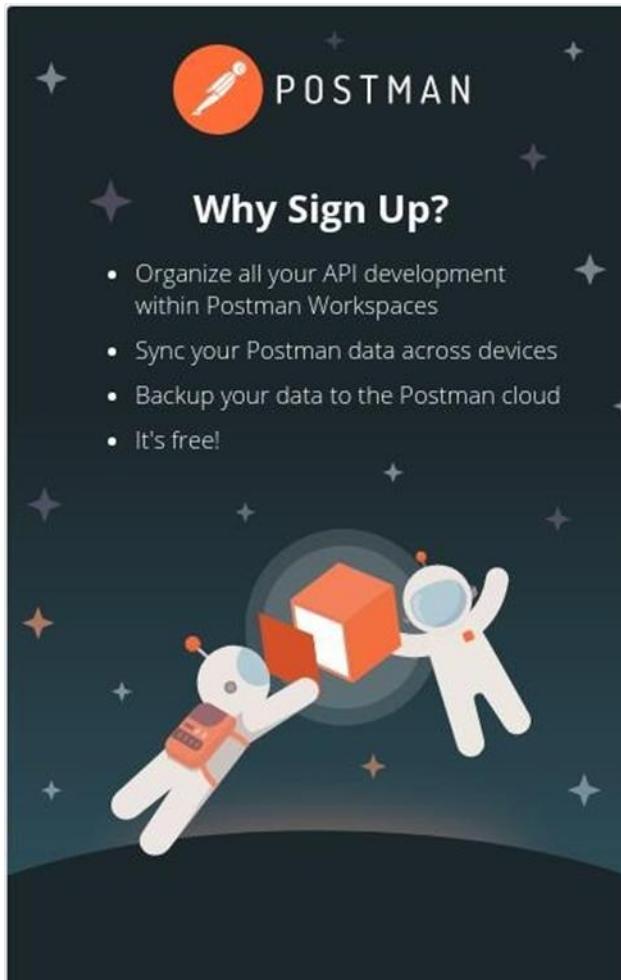
Name	Date modified	Type	Size
 Postman-win64-8.1.0-Setup.exe	4/6/2021 2:19 PM	Application	118,136 KB

Double-click on it for installation.



After installation, the Postman landing screen opens. Also, we have to sign up here. There are two options to create a Postman account, which are as follows:

- Click on the Create free account.
- Use the Google Account.



The image shows the Postman account creation interface. On the left, there's a promotional banner for 'Why Sign Up?' featuring two astronauts in space, one holding a large orange cube. The banner lists benefits: 'Organize all your API development within Postman Workspaces', 'Sync your Postman data across devices', 'Backup your data to the Postman cloud', and 'It's free!'. On the right, the 'Create Postman Account' form is displayed. It includes fields for 'Email' (with a placeholder 'john.doe@example.com'), 'Username' (placeholder 'john\_doe'), 'Password' (placeholder 'password123'), and a 'SHOW' link. There are also checkboxes for 'Sign up to get product updates, news, and other marketing communications.' (unchecked) and 'Keep me signed in' (checked). Below the form, a note says 'By creating an account, I agree to the [Terms](#) and [Privacy Policy](#)'. At the bottom are 'Create free account' and 'Sign up with Google' buttons.

Proceed with the steps of account creation and enter relevant details like name, role, and so on.

## Welcome to Postman!

Tell us a bit about yourself so we can help you get the most out of Postman.

What's your name?

 Change profile photo

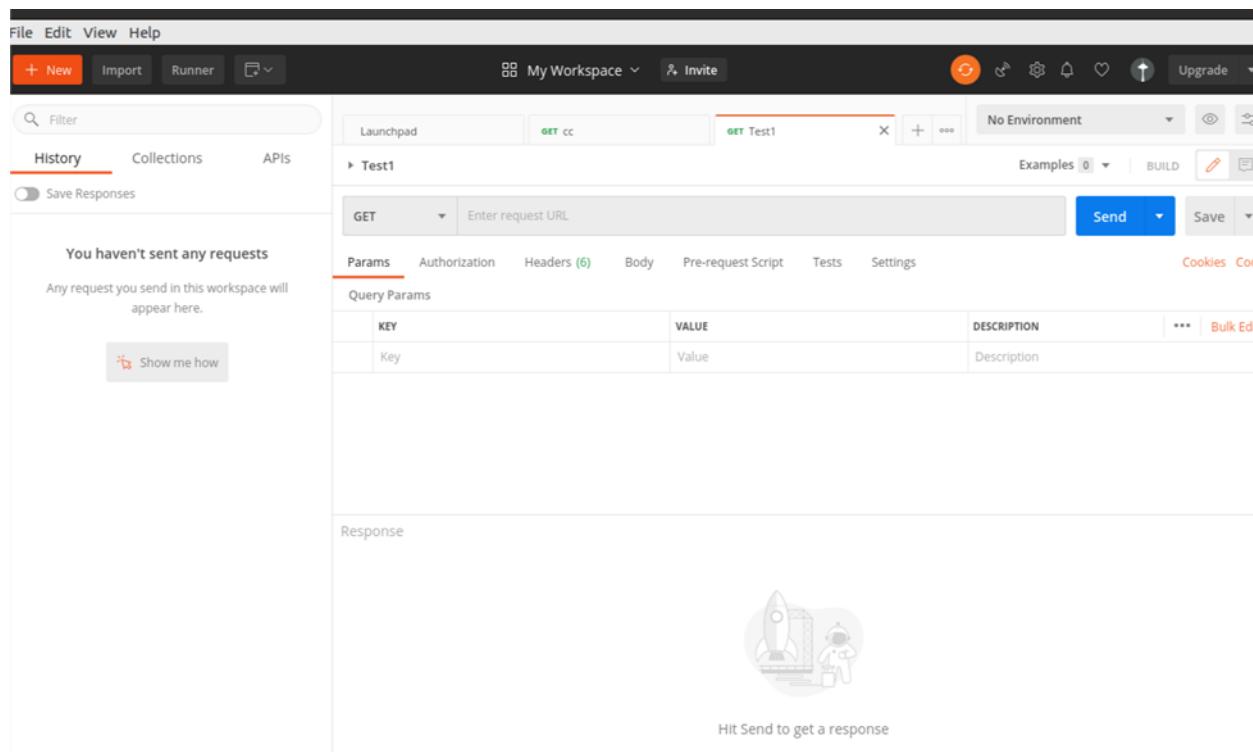
Which of these roles is closest to yours?

How do you plan to use Postman?

API documentation  Automated testing  
 Debugging and manual testing  Designing and mocking APIs  
 Monitoring  Publishing APIs

[Continue](#)

Finally, we shall land to the Start screen of Postman. The following screen will appear on your computer:

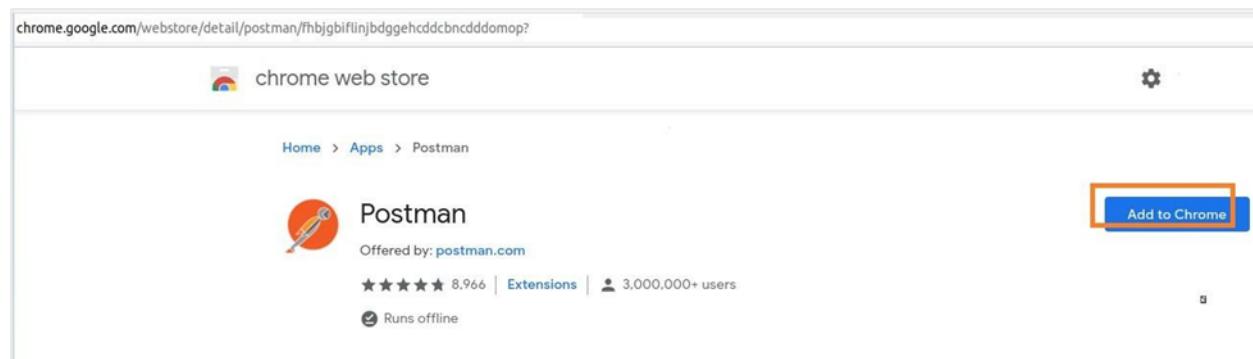


## Chrome Extension

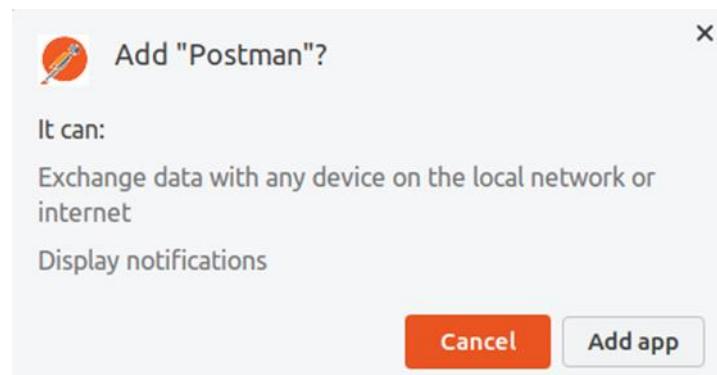
To download Postman as a Chrome browser extension, launch the below link in Chrome:

<https://chrome.google.com/webstore/detail/postman/fhbjgblinjbddggehcdcbnccccdomop>

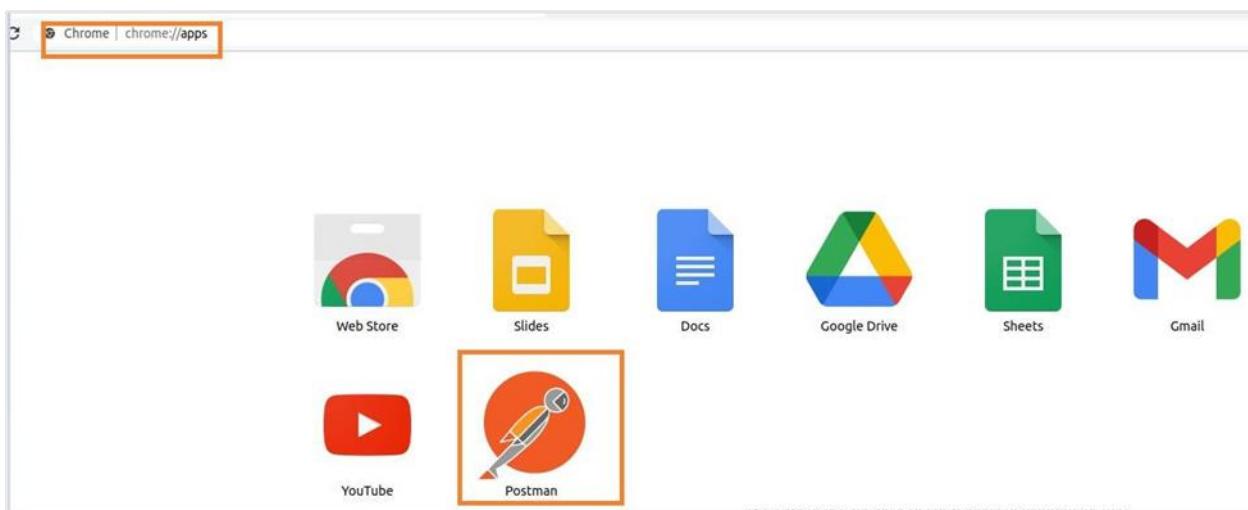
Then, click on Add to Chrome.



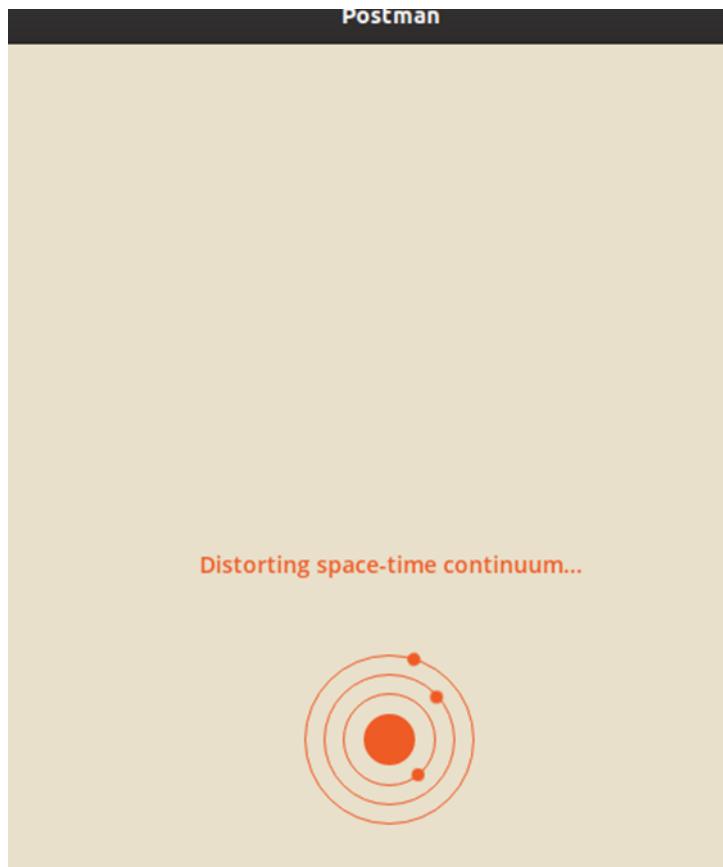
A pop-up gets displayed, click on the Add app button.



Chrome Apps page gets launched, along with the Postman icon. Next, we have to click on the Postman icon.

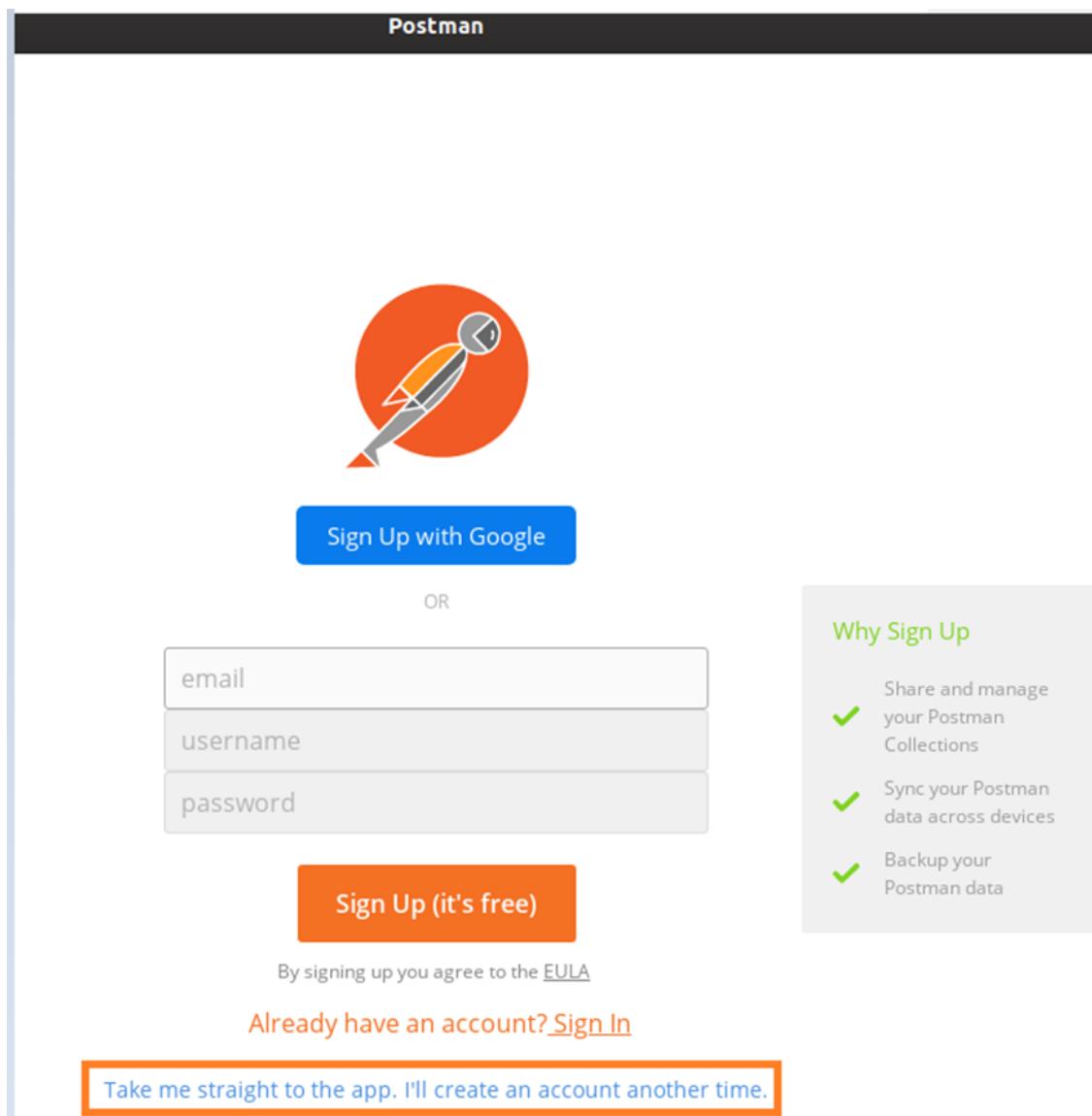


Installation of Postman kicks off.

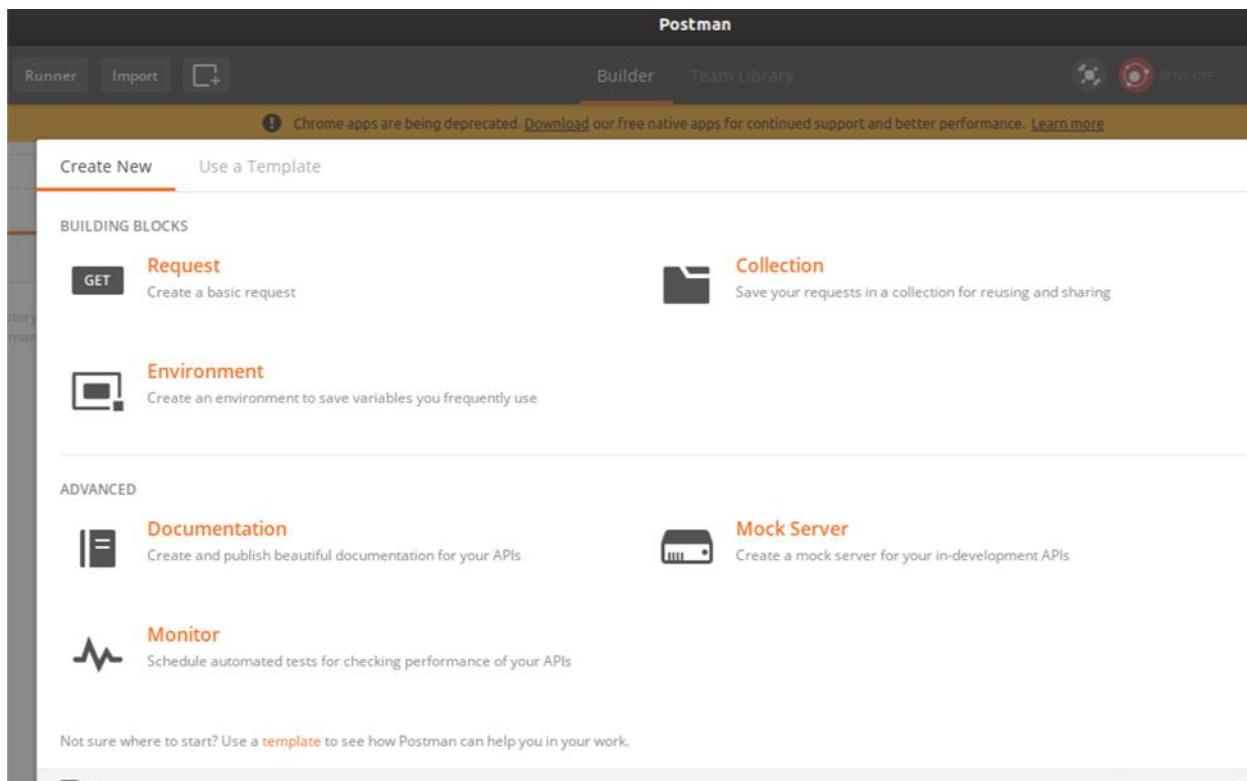


Once the installation is completed, the Postman registration page is opened. We can either proceed with the registration as explained previously (while installing Postman as a standalone application) or skip it by clicking on the link Take me straight to the app.

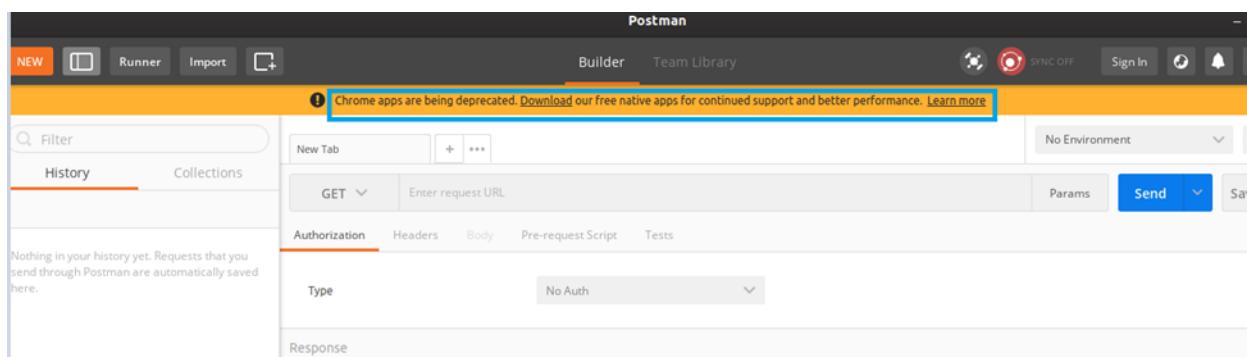
We can create an account later.



Registration is an important step as it enables access to user data from other machines. Next, the Postman welcome page opens up.



Once we close the pop-up and move to the following page, we get the message -  
 Chrome apps are being deprecated.



It is always recommended to install Postman as a standalone application rather than a Chrome extension.

### 3. Postman – Environment Variables

Variables give the option to hold and repeat parameters in the requests, collections, scripts and so on. If we need to modify a value, we need to do it in only one place. Thus, the variables help to minimise the chance of errors and increase efficiency.

In Postman, an environment consists of a key-value pair. It helps to identify each request separately. As we create environments, we can modify key-value pairs and that will produce varied responses from the same request.

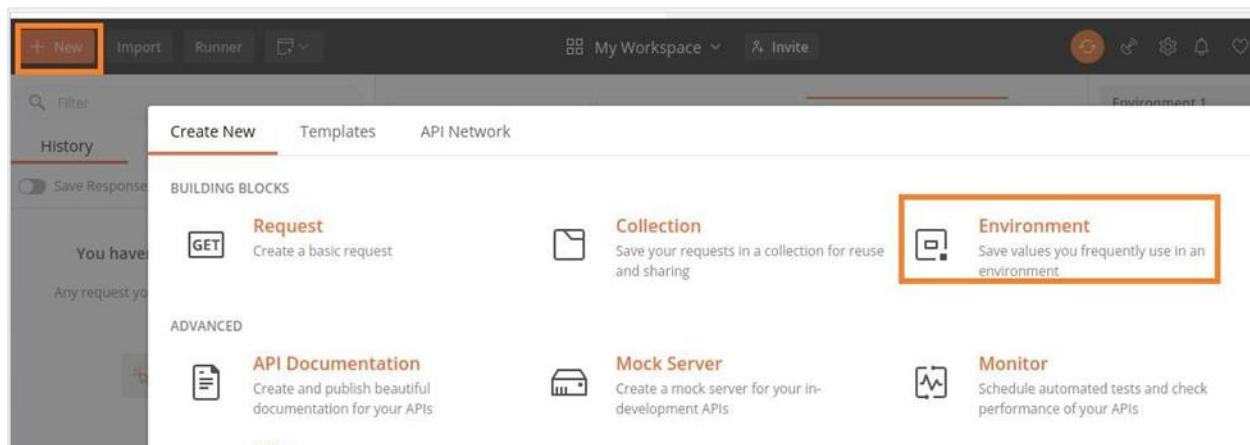
The key in the key-value pair in the environment is known as the Environment variable. There can be multiple environments and each of them can also have multiple variables. However, we can work with a single environment at one time.

In short, an environment allows the execution of requests and collections in a varied data set. We can create environments for production, testing and development. Each of these environments will have different parameters like URL, password, and so on.

#### Create Environment

Follow the steps given below to create an environment in Postman:

Step 1: Navigate to the New menu and then click on Environment.



✉️ [hr@kasperanalytics.com](mailto:hr@kasperanalytics.com)

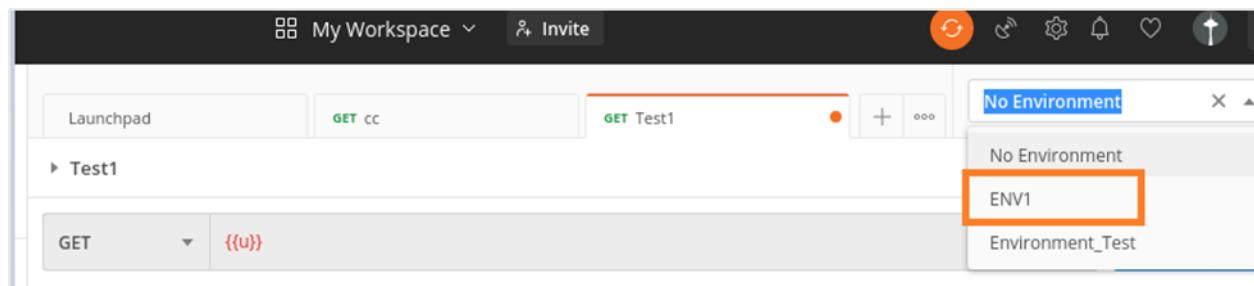
 [kasper-analytics](#)

**Step 2: MANAGE ENVIRONMENTS** pop-up gets opened. We have to enter the Environment name. Then, add a variable name and value.

Here, we have added the variable u and the value as <https://jsonplaceholder.typicode.com/users>. Close the pop-up.

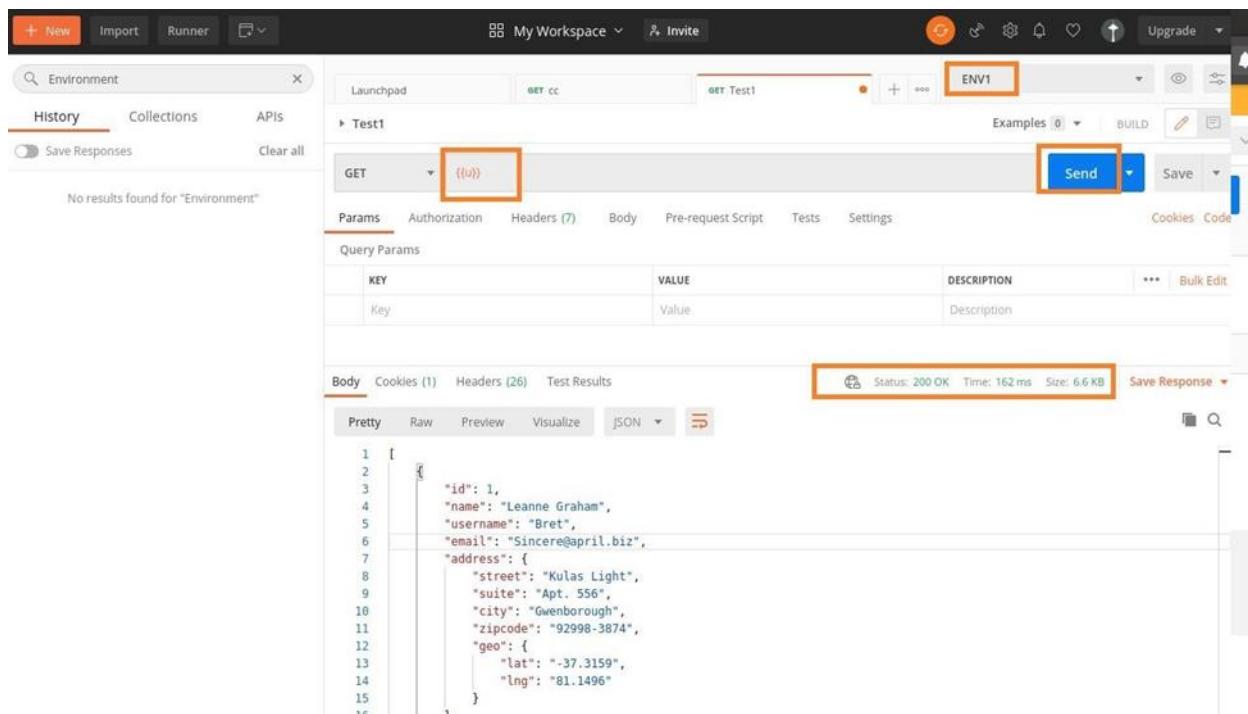


Step 3: The new Environment (ENV1) gets reflected as one of the items in the No Environment dropdown.



Step 4: Select the ENV1 environment and enter {{u}} in the address bar. To utilise an Environment variable in a request, we have to enclose it with double curly braces ({{<Environment variable name>}}).

Step 5: Then, click on Send. This variable can be used instead of the actual URL. We have received the Response code 200 OK (meaning the request is successful).



The screenshot shows the Postman interface. A GET request named 'Test1' is selected. The URL field contains 'GET {{u}}'. The 'ENV1' environment is selected. The 'Send' button is highlighted with a red box. The response status is 200 OK, and the JSON response body is displayed:

```

1  {
2   "id": 1,
3   "name": "Leanne Graham",
4   "username": "Bret",
5   "email": "Sincere@april.biz",
6   "address": {
7     "street": "Kulas Light",
8     "suite": "Apt. 556",
9     "city": "Gwenborough",
10    "zipcode": "92998-3874",
11    "geo": {
12      "lat": "-37.3159",
13      "lng": "81.1496"
14    }
15  }

```

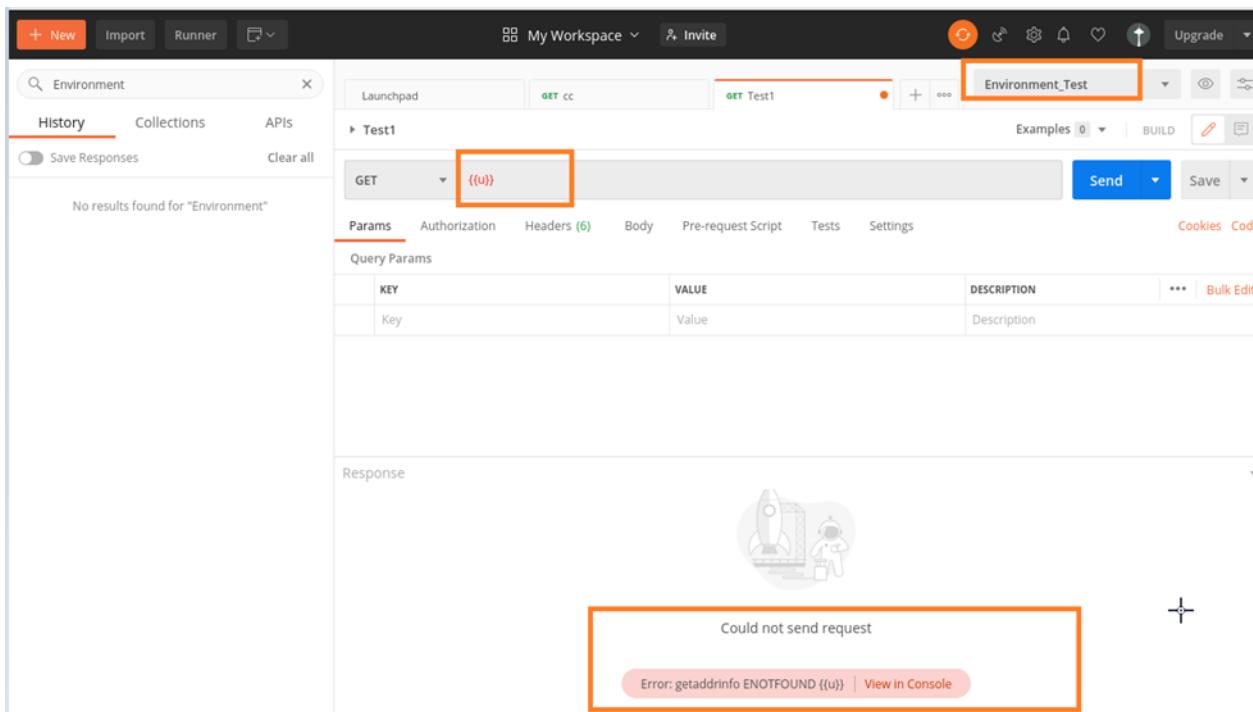
## Environment Variables Scope

The scope of an Environment variable is within the environment for which it is created. This means it has a local scope confined to that environment. If we select another environment, and try to access the same Environment variable, we shall get an error.

In this chapter, we have created an Environment variable u within the ENV1 environment and on sending a GET request, we got the desired response.

However, if we try to use the same Environment variable u from another Environment, say Environment Test, we will receive errors.

The following screen shows the error, which we may get if we use the same Environment variable u from another environment:



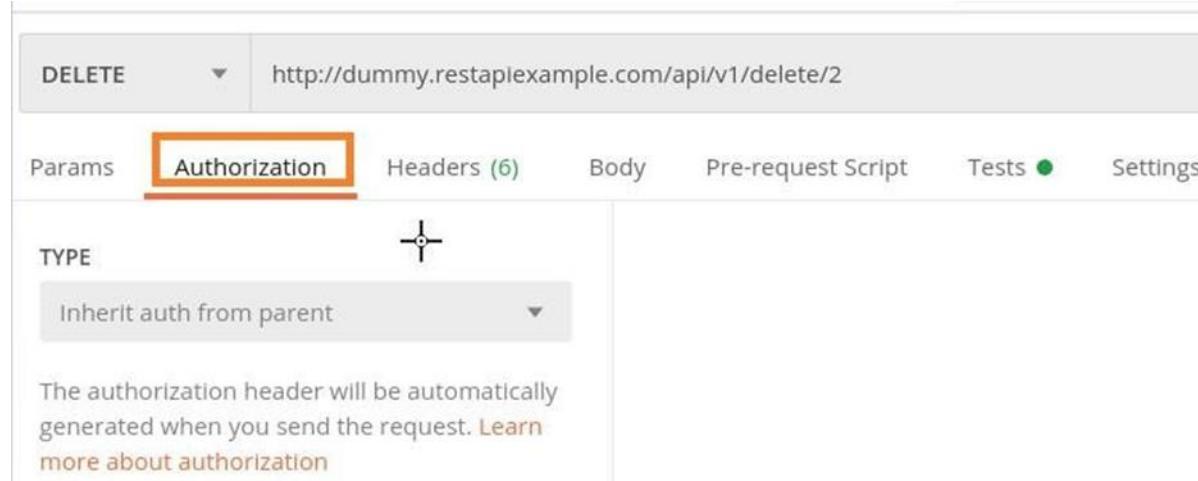
The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'New', 'Import', 'Runner', and other workspace-related options. The main area shows a 'Test1' collection with a single 'GET' request. The URL field for this request contains the placeholder `{{u}}`. To the right of the URL field, the 'Environment' dropdown is set to 'Environment\_Test'. In the 'Response' section, there's an error message: 'Could not send request' and 'Error: getaddrinfo ENOTFOUND {{u}}'. A 'View in Console' link is also present.

## 4. Postman – Authorization

In Postman, authorization is done to verify the eligibility of a user to access a resource in the server. There could be multiple APIs in a project, but their access can be restricted only for certain authorized users.

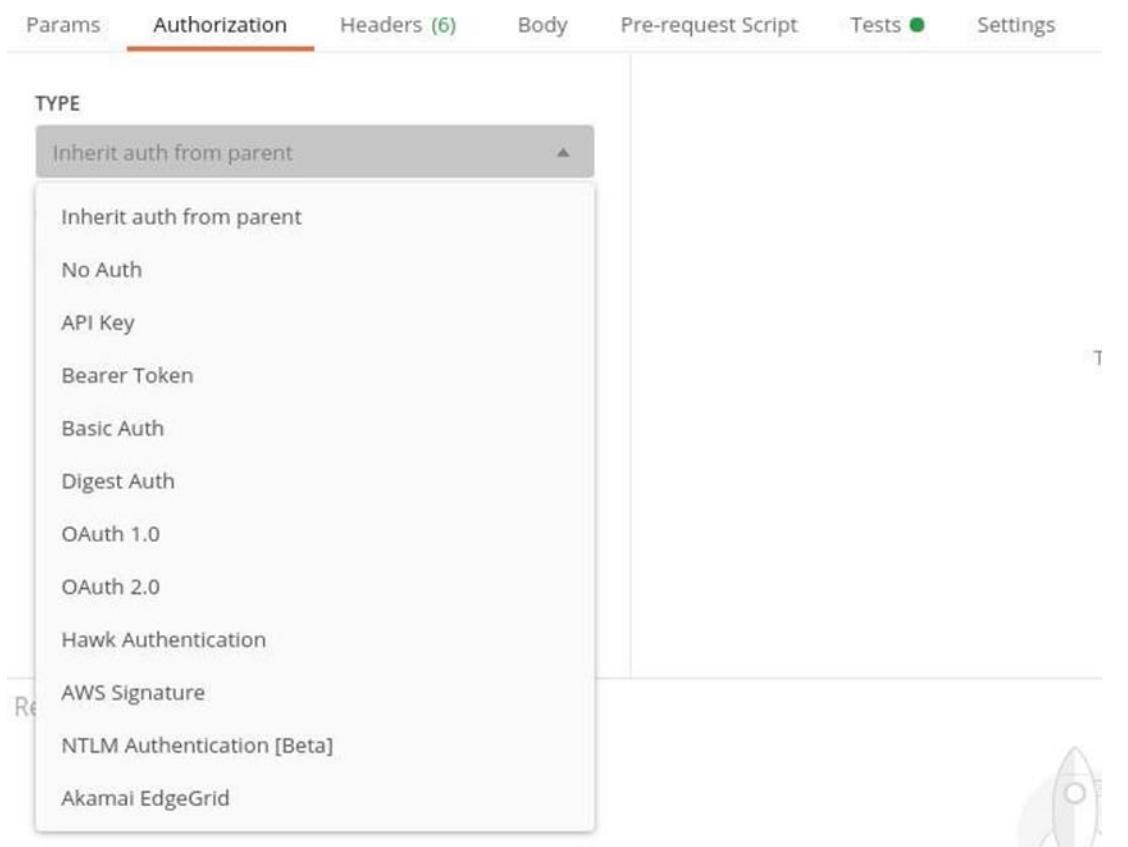
The process of authorization is applied for the APIs which are required to be secured. This authorization is done for identification and to verify, if the user is entitled to access a server resource.

This is done within the Authorization tab in Postman, as shown below:



The screenshot shows the Postman interface with the Authorization tab selected. The URL is set to `http://dummy.restapitutorial.com/api/v1/delete/2`. The Authorization tab is highlighted with an orange border. In the TYPE dropdown, 'Inherit auth from parent' is selected. A note below states: 'The authorization header will be automatically generated when you send the request. Learn more about authorization'.

In the TYPE dropdown, there are various types of Authorization options, which are as shown below:

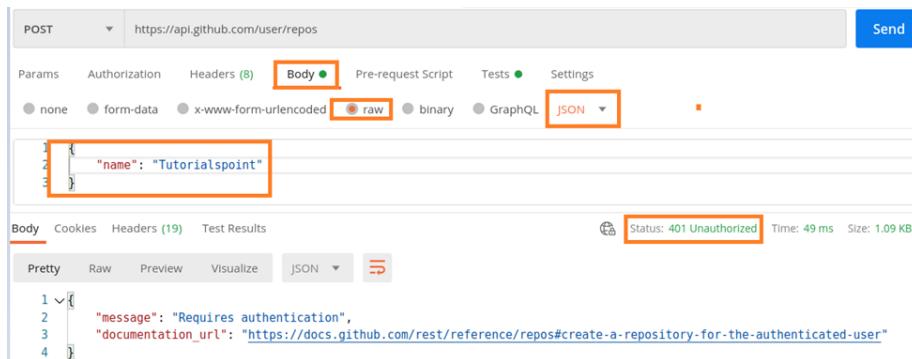


Let us now create a POST request with the APIs from GitHub Developer having an endpoint <https://www.api.github.com/user/repos>. In the Postman, click the Body tab and select the option raw and then choose the JSON format.

Add the below request body:

```
{
  "name" : "Tutorialspoint"
}
```

Then, click on Send.



The screenshot shows the Postman interface with a successful POST request to <https://api.github.com/user/repos>. The response status is 401 Unauthorized. The response body indicates that authentication is required.

 [hr@kasperanalytics.com](mailto:hr@kasperanalytics.com)

 [kasper-analytics](#)

The Response code obtained is 401 Unauthorized. This means, we need to pass authorization to use this resource. To authorize, select any option from the TYPE dropdown within the Authorization tab.

## Types of Authorization

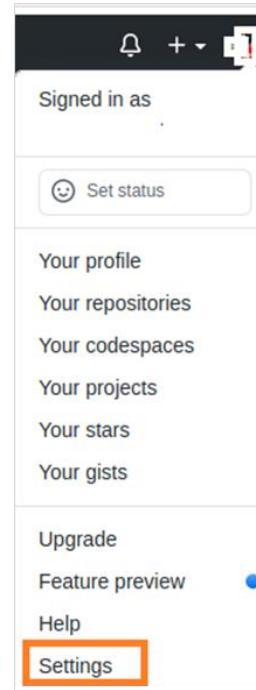
Let us discuss some of the important authorization types namely Bearer Token and Basic Authentication.

### Bearer Token

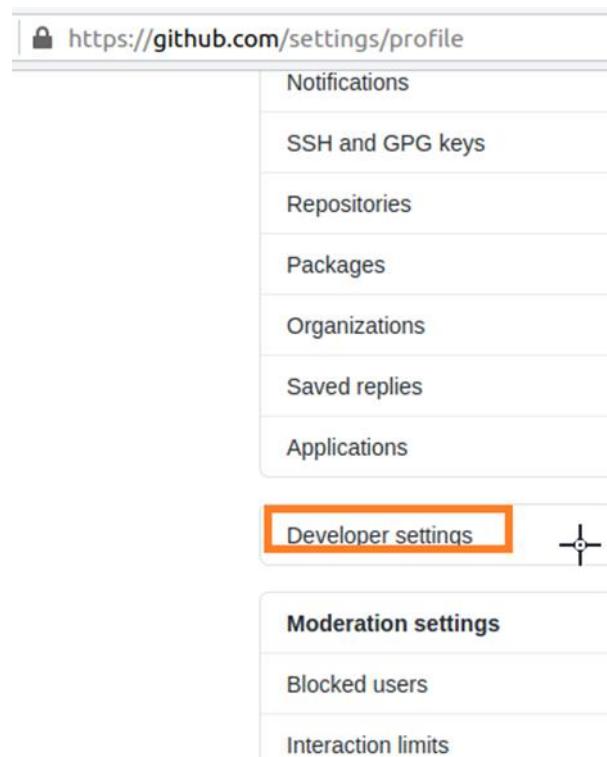
For Bearer Token Authorization, we have to choose the option Bearer Token from the TYPE dropdown. After this, the Token field gets displayed which needs to be provided in order to complete the Authorization.

Step 1: To get the Token for the GitHub API, first login to the GitHub account by clicking on the link given herewith: <https://github.com/login>.

Step 2: After logging in, click on the upper right corner of the screen and select the Settings option.



Now, select the option Developer settings.

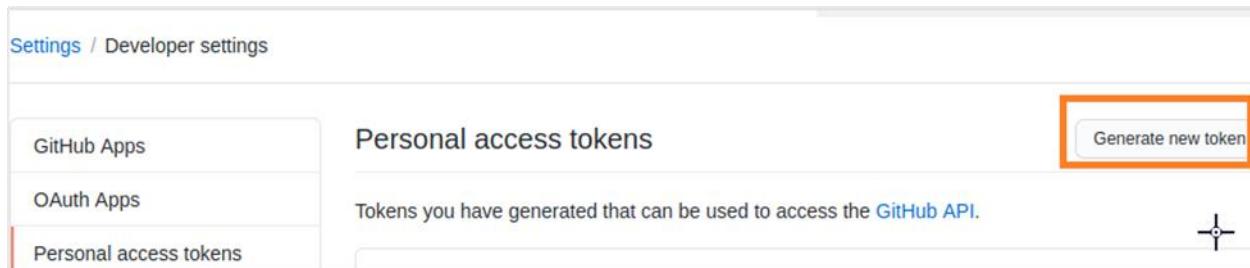


Next, click on Personal access tokens.



The screenshot shows the "Settings / Developer settings" page. On the left, there is a sidebar with three options: GitHub Apps, OAuth Apps, and Personal access tokens, with "Personal access tokens" highlighted by an orange border. The main content area is titled "GitHub Apps" and contains a sub-section titled "Personal access tokens". A button labeled "Generate new token" is located in the top right corner of this section. Below the "Personal access tokens" section, there is a note: "Want to build something that integrates with and extends GitHub? Register a new GitHub App to get started developing on the GitHub API. You can also read more about building GitHub Apps in our developer documentation." There is also a "New GitHub App" button.

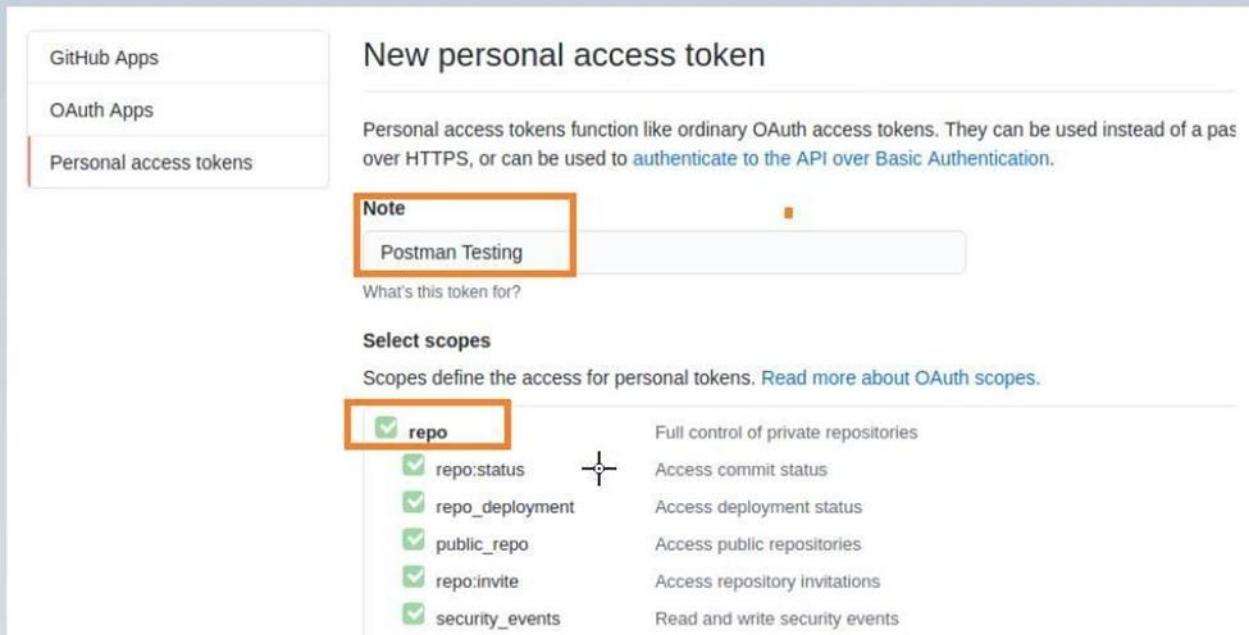
Now, click on the Generate new token button.



The screenshot shows the "Settings / Developer settings" page with the "Personal access tokens" section selected. The main content area is titled "Personal access tokens" and contains a sub-section titled "Tokens you have generated that can be used to access the GitHub API.". In the top right corner of this section, there is a button labeled "Generate new token" which is highlighted with an orange border. There is also a small icon next to the button.

Provide a Note and select option repo. Then, click on Generate Token at the bottom of the page.

Finally, a Token gets generated.



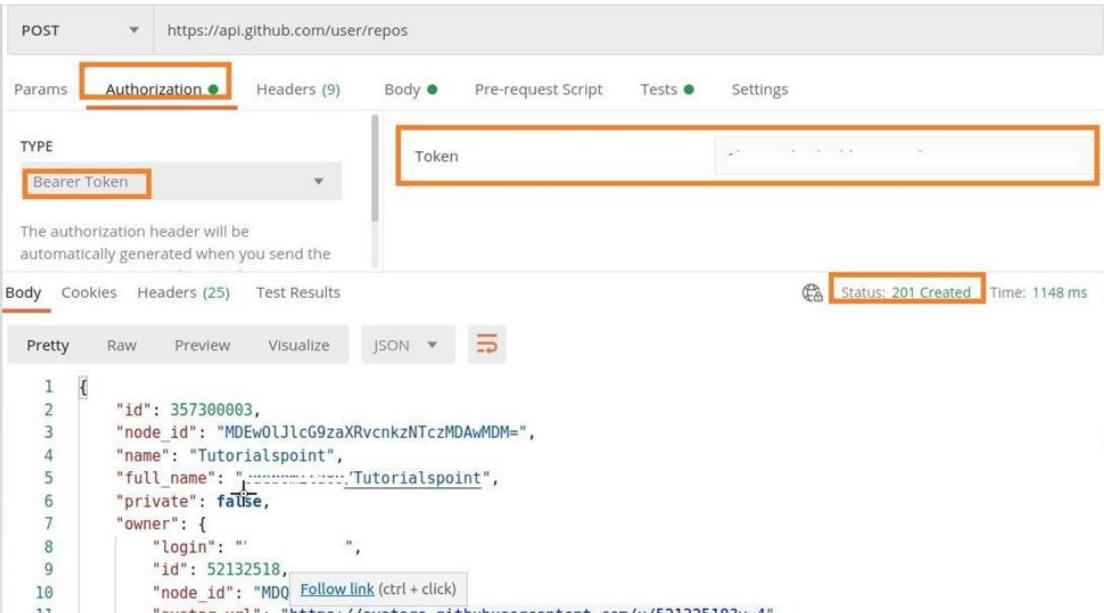
The screenshot shows the GitHub 'Personal access tokens' creation interface. A note 'Postman Testing' is added. Under 'Select scopes', the 'repo' scope is selected, highlighted with an orange border. Other available scopes include 'status', 'deployments', 'public\_repo', 'invitations', and 'security\_events'. A note states: 'Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password over HTTPS, or can be used to authenticate to the API over Basic Authentication.' A link to 'Read more about OAuth scopes' is provided.

Copy the Token and paste it within the Token field under the Authorization tab in Postman. Then, click on Send.

Please note: Here, the Token is unique to a particular GitHub account and should not be shared.

## Response

The Response code is 201 Created which means that the request is successful.



The screenshot shows a Postman request to 'https://api.github.com/user/repos'. The 'Authorization' tab is selected, showing a 'Bearer Token' type and a token input field. The response status is 'Status: 201 Created'. The JSON response body is displayed, showing details of a newly created repository:

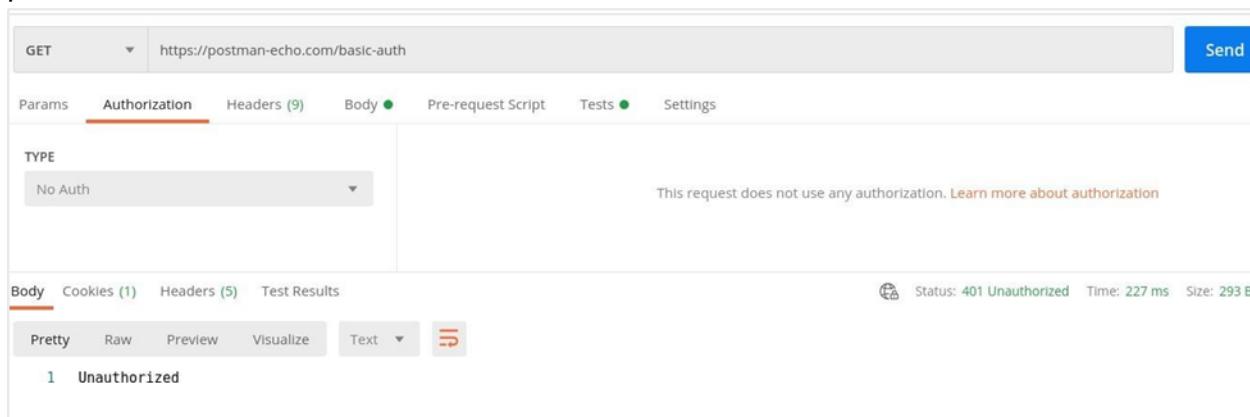
```

1 {
2   "id": 357300003,
3   "node_id": "MDEwOlJlcG9zaXRvcnkzNTczMDAwMDM",
4   "name": "Tutorialspoint",
5   "full_name": "wwwTutorialspoint/Tutorialspoint",
6   "private": false,
7   "owner": {
8     "login": "Tutorialspoint",
9     "id": 52132518,
10    "node_id": "MDQ6VXNlcjUyMTMzMDU4",
11    "avatar_url": "https://avatars.githubusercontent.com/u/52132518?v=4"
}

```

## Basic Authentication

For Basic Authentication Authorization, we have to choose the option Basic Auth from the TYPE dropdown, so that the Username and Password fields get displayed. First we shall send a GET request for an endpoint (<https://postman-echo.com/basic-auth>) with the option No Auth selected from the TYPE dropdown. Please note: The username for the above endpoint is postman and password is password.



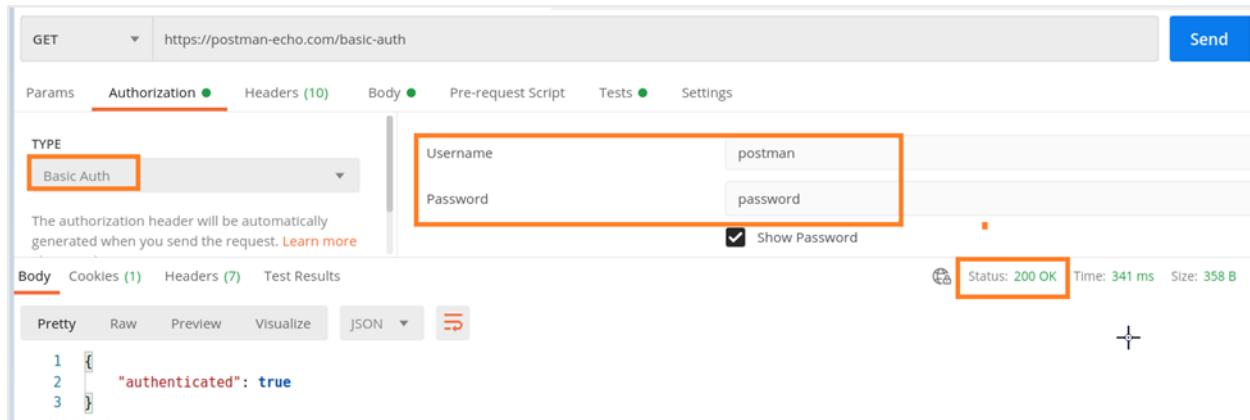
The screenshot shows a Postman request configuration. The method is set to GET, and the URL is https://postman-echo.com/basic-auth. In the 'Authorization' tab, the 'TYPE' dropdown is set to 'No Auth'. The response status is 401 Unauthorized, indicating that basic authentication was not provided.

The Response Code obtained is 401 Unauthorized. This means that Authorization did not pass for this API.

Now, let us select the option Basic Auth as the Authorization type, following which the Username and Password fields get displayed.

Enter the postman for the Username and password for the Password field. Then, click on

Send.



The screenshot shows a Postman request configuration. The method is set to GET, and the URL is https://postman-echo.com/basic-auth. In the 'Authorization' tab, the 'TYPE' dropdown is set to 'Basic Auth'. The 'Username' and 'Password' fields are filled with 'postman' and 'password' respectively. The response status is 200 OK, indicating successful authentication.

The Response code obtained is now 200 OK, which means that our request has been sent successfully.

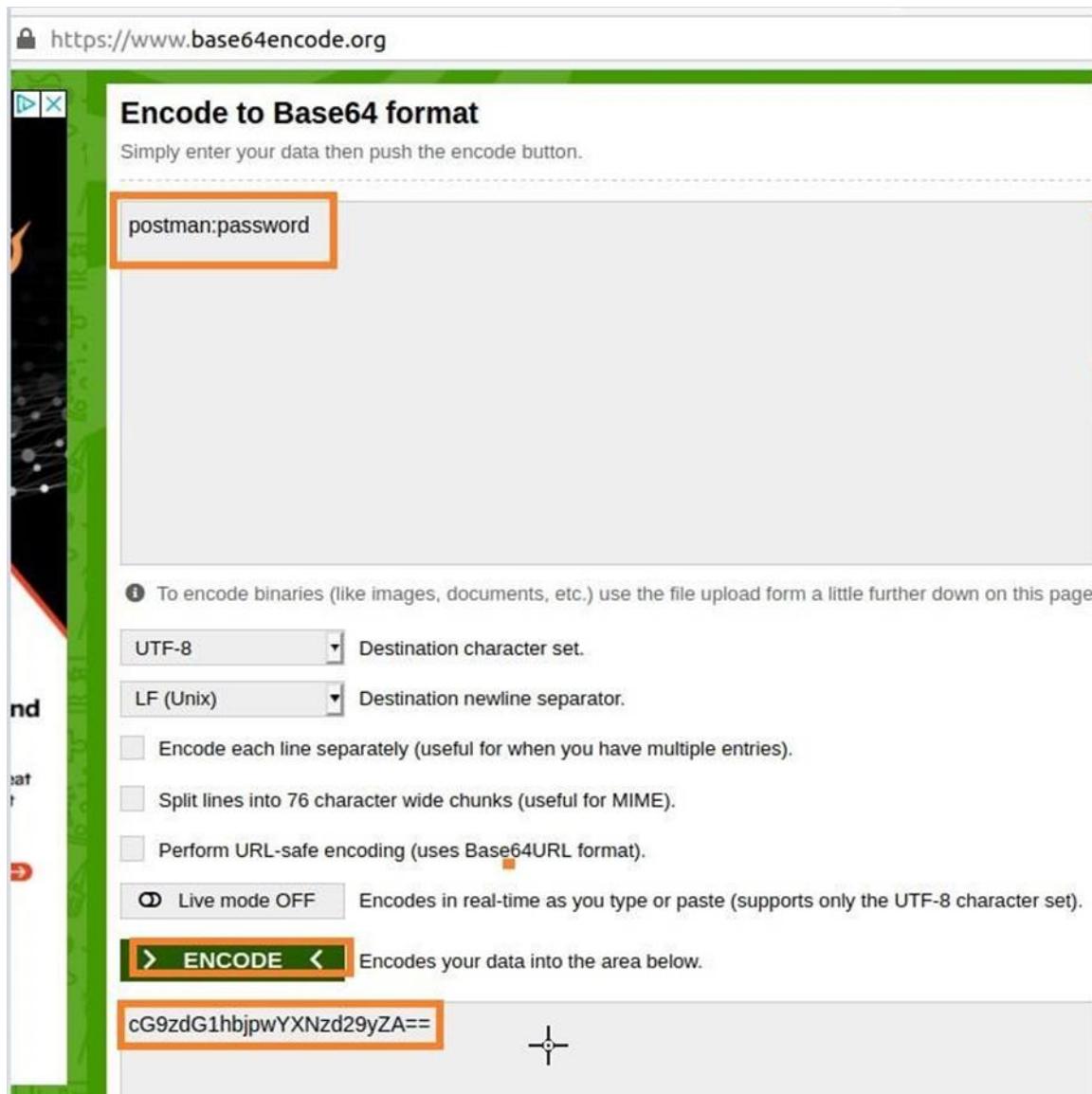
#### No Auth

We can also carry out Basic Authentication using the request Header. First, we have to choose the option as No Auth from the Authorization tab. Then in the Headers tab, we have to add a key: value pair.

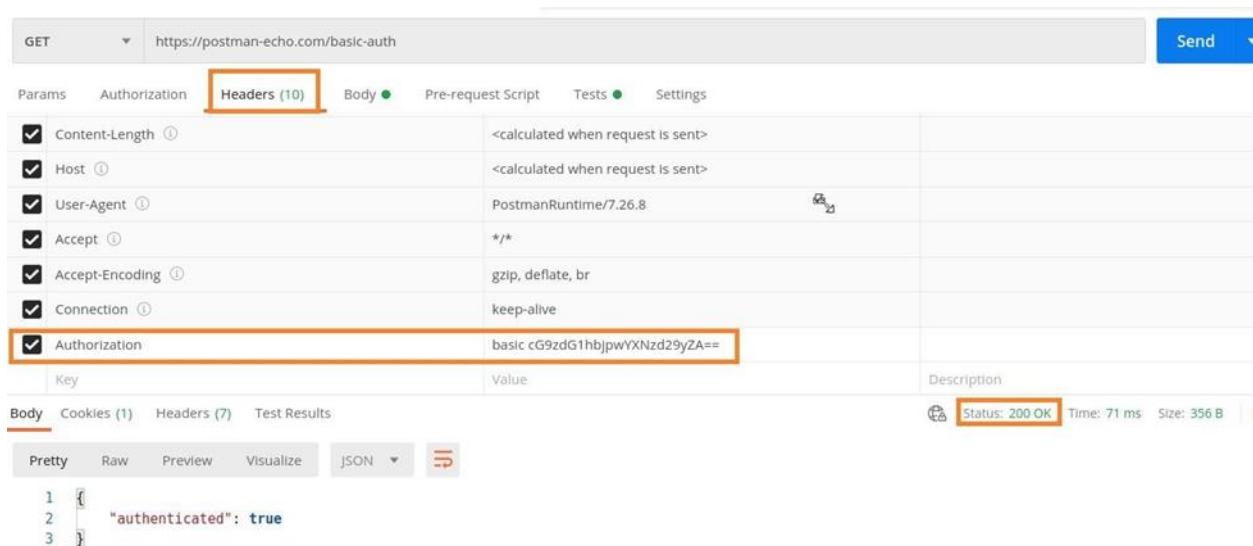
We shall have the key as Authorization and the value is the username and password of the user in the format as basic <encoded credential>.

The endpoint used in our example is: <https://postman-echo.com/basic-auth>. To encode the username and password, we shall take the help of the third party application having the URL: <https://www.base64encode.org>

Please note: The username for our endpoint here is postman and password is password. Enter postman: password in the edit box and click on Encode. The encoded value gets populated at the bottom.



We shall add the encoded Username and Password received as cG9zdG1hbjpwYXNzd29yZA== in the Header in the format - basic cG9zdG1hbjpwYXNzd29yZA==. Then, click on Send.



GET https://postman-echo.com/basic-auth

Headers (10)

Key	Value	Description
Content-Length	<calculated when request is sent>	
Host	<calculated when request is sent>	
User-Agent	PostmanRuntime/7.26.8	q
Accept	/*	
Accept-Encoding	gzip, deflate, br	
Connection	keep-alive	
Authorization	basic cG9zdG1hbjpwYXNzd29yZA==	

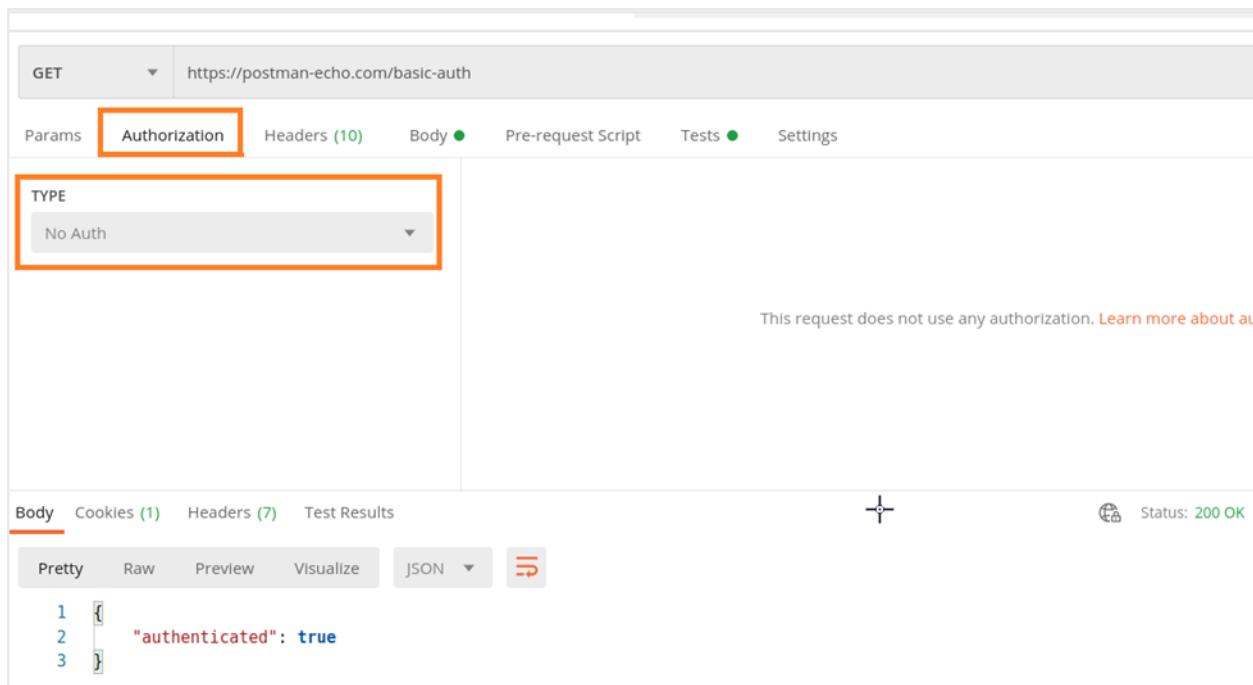
Status: 200 OK Time: 71 ms Size: 356 B

```

1 {
2   "authenticated": true
3 }

```

No Auth selected from the TYPE dropdown.



GET https://postman-echo.com/basic-auth

Authorization

TYPE

No Auth

This request does not use any authorization. [Learn more about authorization](#)

Status: 200 OK

```

1 {
2   "authenticated": true
3 }

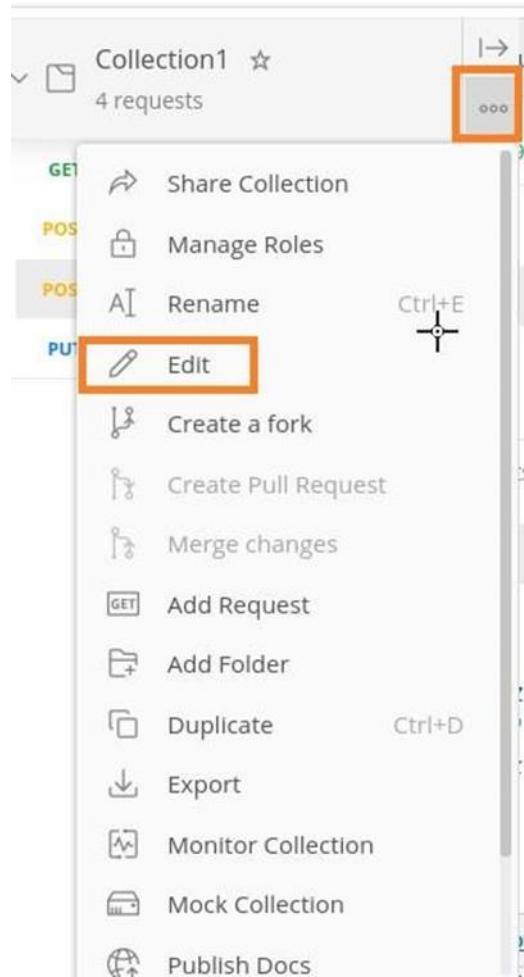
```

The Response code obtained is 200 OK, which means that our request has been sent successfully.

## Authorization at Collections

To add Authorization for a Collection, following the steps given below:

Step 1: Click on the three dots beside the Collection name in Postman and select the option Edit.



Step 2: The EDIT COLLECTION pop-up comes up. Move to the Authorization tab and then select any option from the TYPE dropdown. Click on Update.

**EDIT COLLECTION** X

Name  
Collection1

Description    **Authorization**    Pre-request Scripts    Tests    Variables

This authorization method will be used for every request in this collection. You can override this by specifying one in the request.

**TYPE**

No Auth

No Auth  
API Key  
Bearer Token  
Basic Auth  
Digest Auth  
OAuth 1.0  
OAuth 2.0  
Hawk Authentication  
AWS Signature  
NTLM Authentication [Beta]  
Akamai EdgeGrid

This collection does not use any authorization. [Learn more about authorization](#)

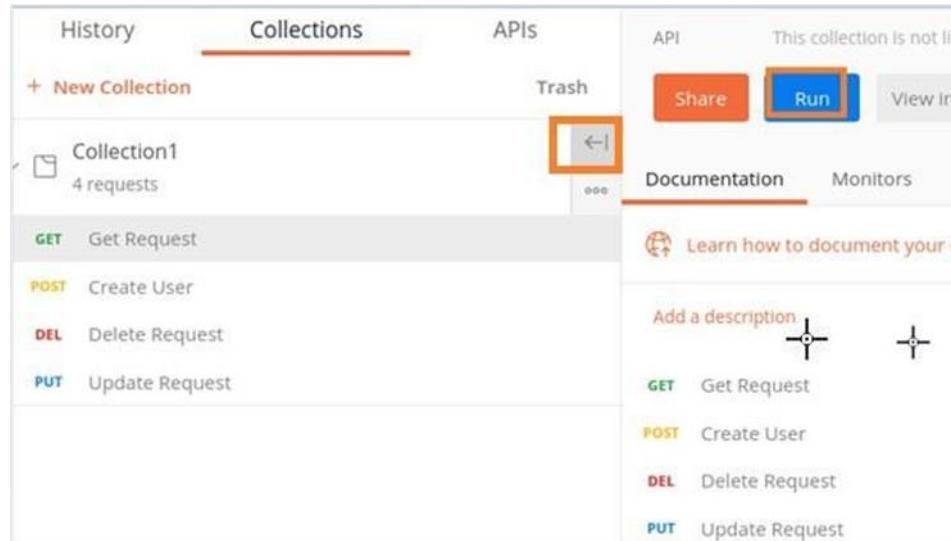
Cancel **Update**

## 5. Postman – Workflows

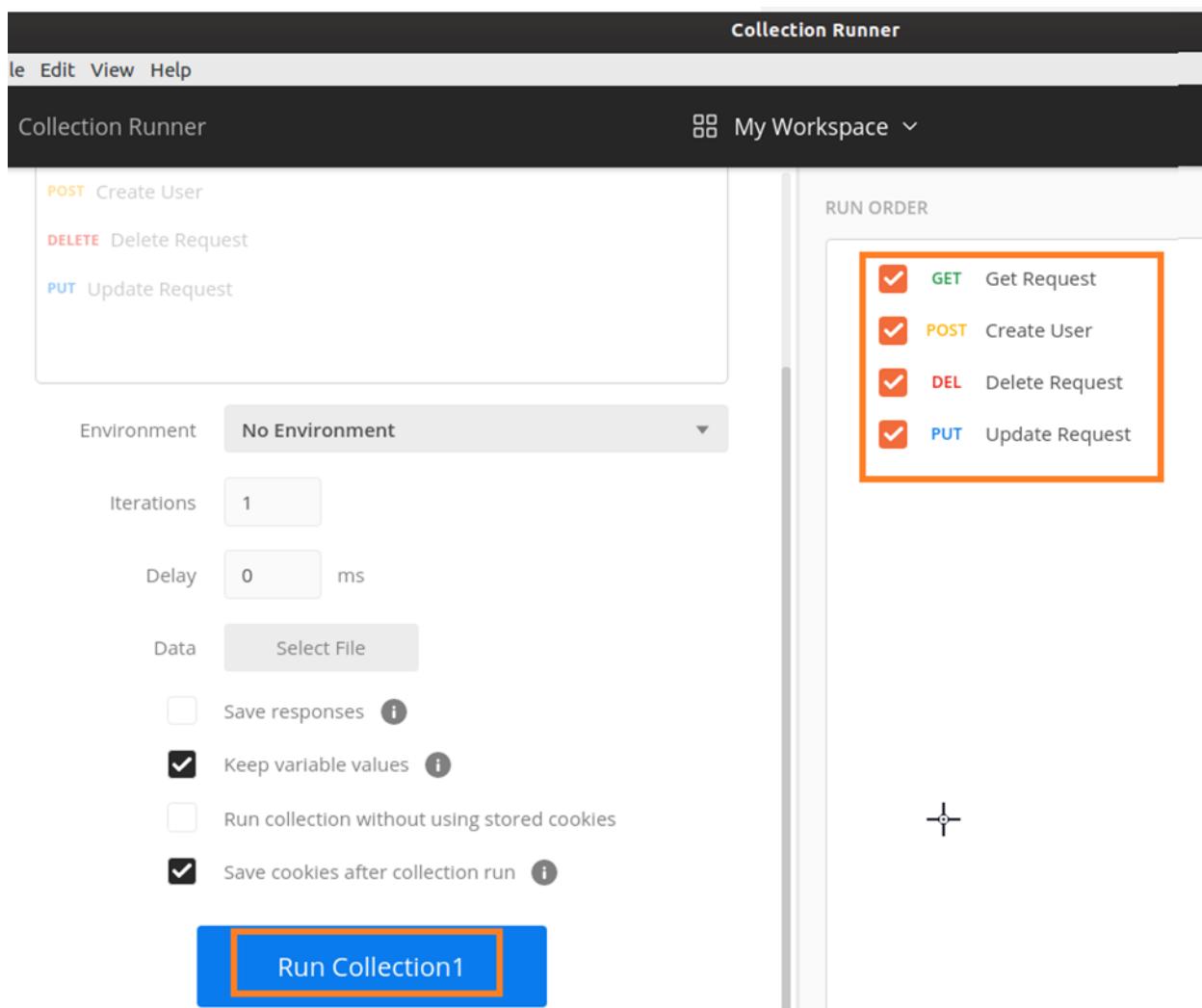
In a Postman Collection, the requests are executed in the order in which they appear. Every request is run first by the order of the folder followed by any request at the Collection root.

Let us create a Collection (Collection1) with four requests. The details on how to create a Collection is discussed in detail in the Chapter about Create Collections.

Step 1: Click on the arrow appearing to the right of the Collection name in the sidebar. Then, click on Run button to trigger execution of requests within the Collection.

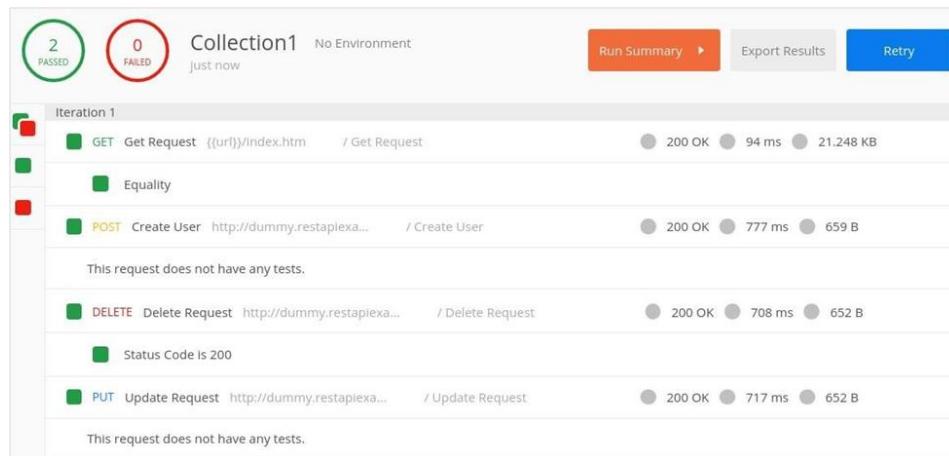


Step 2: The Collection Runner pop-up comes up. The RUN ORDER section shows the order in which the requests shall get executed from top to the bottom. (GET->POST->DEL->PUT). Click on the Run Collection1 button.



The screenshot shows the 'Collection Runner' interface. On the left, there's a sidebar with three requests: 'POST Create User', 'DELETE Delete Request', and 'PUT Update Request'. Below this is a dropdown for 'Environment' set to 'No Environment', and input fields for 'Iterations' (1) and 'Delay' (0 ms). On the right, a 'RUN ORDER' section lists the same three requests, each with a checkmark and a color-coded icon (green for GET, orange for POST, red for DELETE, blue for PUT). A large orange rectangle highlights this list. At the bottom is a prominent blue button labeled 'Run Collection1'.

Step 3: Execution Results show the GET request executed first, followed by POST, then DEL and finally PUT, as mentioned in the RUN ORDER section in the Step 2.



The screenshot shows the execution results for 'Collection1' with 'No Environment' selected. It indicates 2 passed and 0 failed tests. The results are grouped by iteration:

- Iteration 1:**
  - GET Request: Status 200 OK, 94 ms, 21.248 KB
  - Equality
  - POST Create User: Status 200 OK, 777 ms, 659 B
  - This request does not have any tests.
  - DELETE Delete Request: Status 200 OK, 708 ms, 652 B
  - Status Code is 200
  - PUT Update Request: Status 200 OK, 717 ms, 652 B
  - This request does not have any tests.

If we want to change the order of the request to be executed (for example, first the Get Request shall run, followed by Create User, then Update Request and finally the Delete Request). We have to take the help of the function `postman.setNextRequest()`.

This function has the feature to state which request shall execute next. The request name to be executed next is passed as a parameter to this function. As per the workflow, we have to add this function either in the Tests or Pre-request Script tab under the endpoint address bar in Postman.

The syntax for execution of a request in Postman is as follows:

```
postman.setNextRequest("name of request")
```

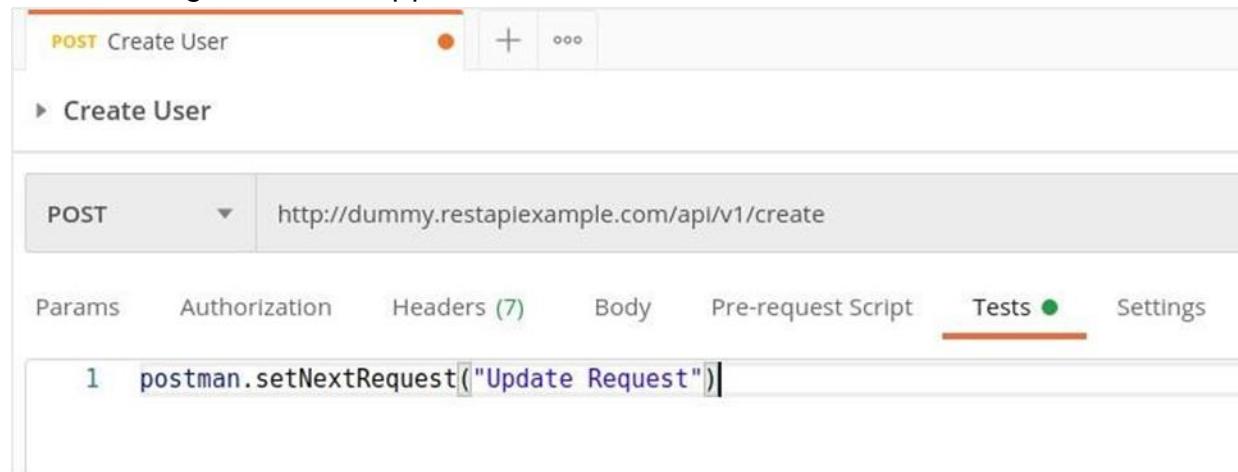
### Implementation of a Workflow

The implementation of a workflow in Postman is explained below in a step wise manner:

Step 1: Add the below script under the Tests tab, for the request – Create User.

```
postman.setNextRequest("Update Request")
```

The following screen will appear:



The screenshot shows the Postman interface for a 'Create User' request. The request details are as follows:

- Method: POST
- URL: http://dummy.restapiexample.com/api/v1/create
- Headers: (7)
- Body
- Pre-request Script
- Tests (selected)
- Settings

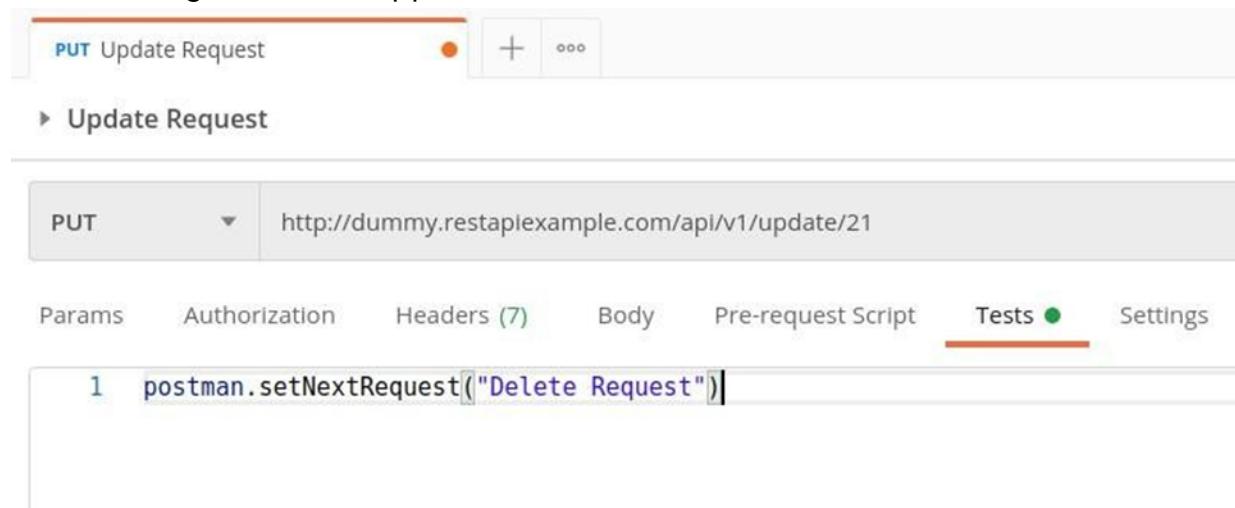
The 'Tests' tab contains the following script:

```
1 postman.setNextRequest("Update Request")
```

Step 2: Add the below script under the Tests tab, for the request – Update Request.

```
postman.setNextRequest("Delete Request")
```

The following screen will appear:

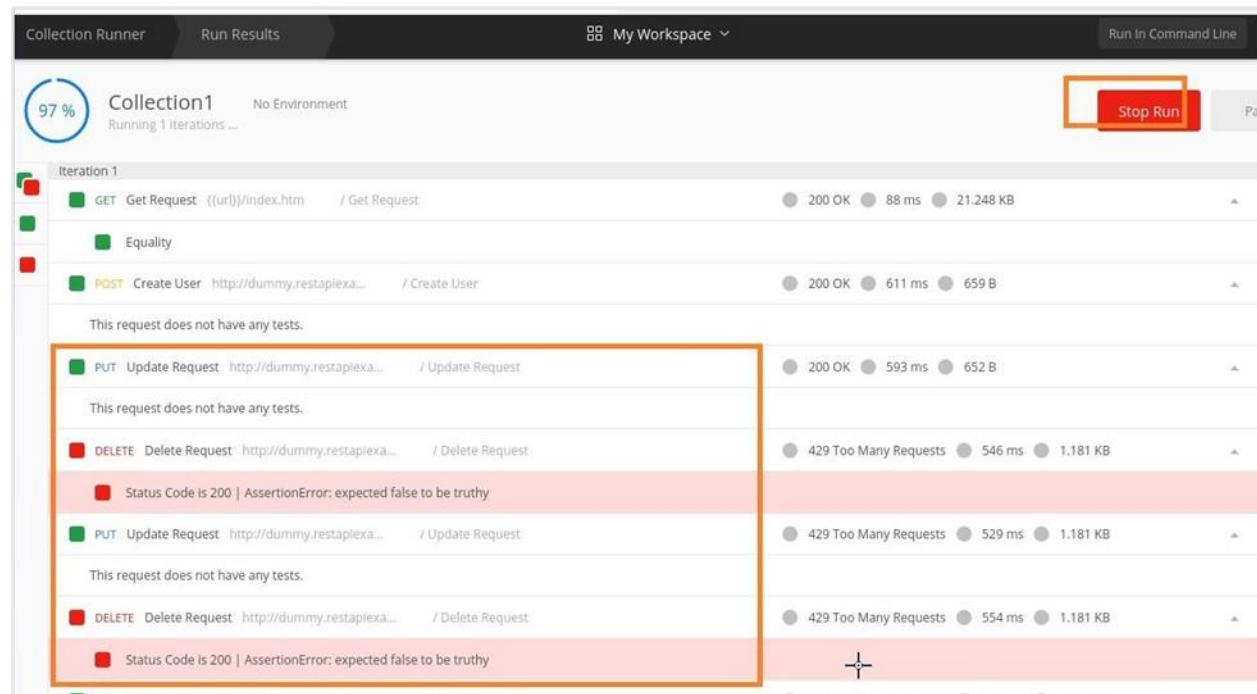


The screenshot shows the Postman interface with the following details:

- PUT Update Request** is selected in the top bar.
- The URL is set to `http://dummy.restapilexample.com/api/v1/update/21`.
- The **Tests** tab is active, containing the script: `1 postman.setNextRequest("Delete Request")`.
- Other tabs visible include Params, Authorization, Headers (7), Body, Pre-request Script, and Settings.

## Output of Workflow

Given below is the output of the workflow:



The Collection Runner output shows the following details:

- Collection1** is running 1 iteration.
- Iteration 1** contains the following requests:
  - GET Get Request**: Status 200 OK, 88 ms, 21.248 KB.
  - Equality**: Status 200 OK, 611 ms, 659 B.
  - POST Create User**: Status 200 OK, 593 ms, 652 B.
  - PUT Update Request**: Status 200 OK, 593 ms, 652 B.
  - DELETE Delete Request**: Status 429 Too Many Requests, 546 ms, 1.181 KB. An assertion error is shown: "Status Code Is 200 | AssertionError: expected false to be truthy".
  - PUT Update Request**: Status 429 Too Many Requests, 529 ms, 1.181 KB.
  - DELETE Delete Request**: Status 429 Too Many Requests, 554 ms, 1.181 KB. An assertion error is shown: "Status Code Is 200 | AssertionError: expected false to be truthy".
- A red box highlights the second **PUT Update Request** and the two **DELETE Delete Request** entries, indicating they are part of an infinite loop.
- A red button labeled **Stop Run** is visible in the top right corner.

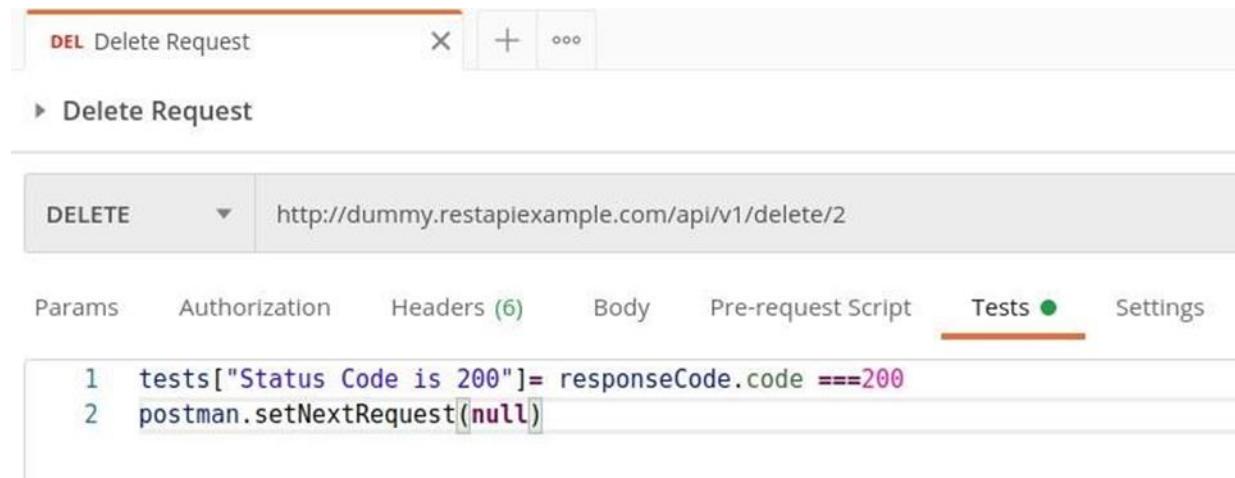
The output shows that Update Request and Delete Request are running in an infinite loop until we stop it by clicking the Stop Run button.

## Infinite Workflow Loop

If we want to stop the infinite Workflow loop via script, we have to add the below script for the request – Delete Request.

```
postman.setNextRequest(null)
```

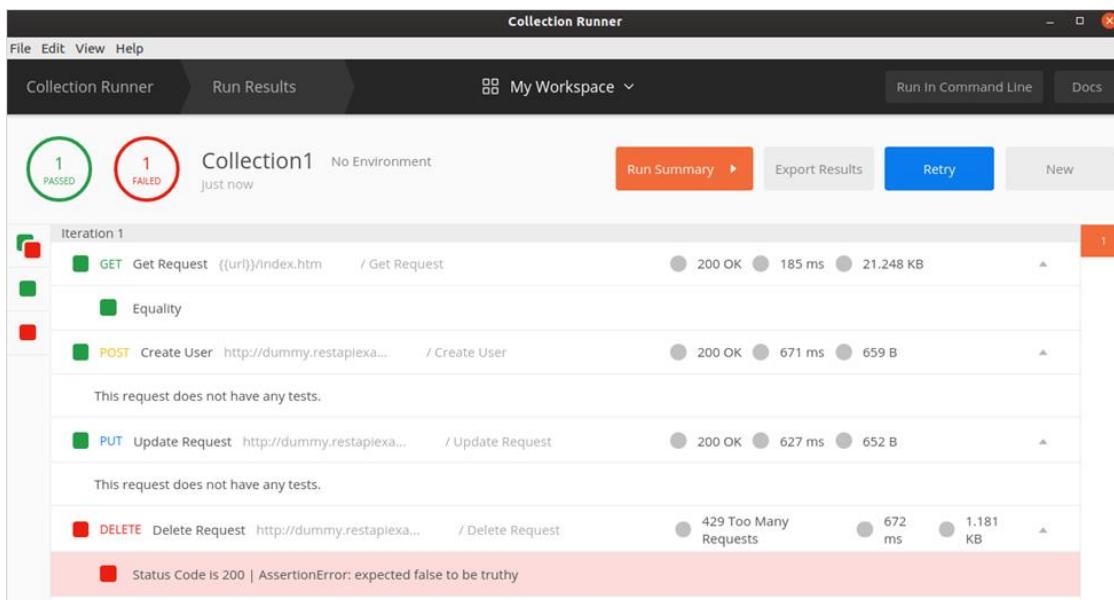
The following screen will appear:



The screenshot shows the Postman Collection Editor interface. A collection named "Delete Request" is open. In the "Tests" tab, the following script is written:

```
1 tests["Status Code is 200"] = responseCode.code === 200
2 postman.setNextRequest(null)
```

Again run the same Collection and output shall be as follows:



The screenshot shows the Collection Runner interface. The collection "Collection1" has 1 Passed and 1 Failed test. The failed test is for the "DELETE" request, which resulted in a 429 Too Many Requests error. The details of the failed test are shown in the table:

Method	URL	Test Status	Response Status	Time	Size
DELETE	http://dummy.restapiexample.com/api/v1/delete/2	Failed	429 Too Many Requests	672 ms	1.181 KB

The message at the bottom indicates: "Status Code Is 200 | Assertion Error: expected false to be truthy".

The output shows the order of execution as Get Request, Create User, Update Request and finally Delete Request.

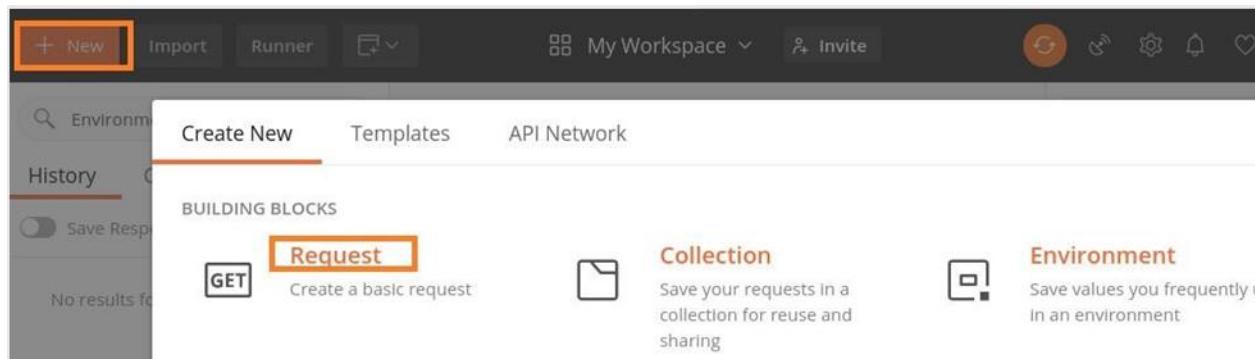
## 6. Postman – GET Requests

A GET request is used to obtain details from the server and does not have any impact on the server. The GET request does not update any server data while it is triggered. The server only sends its Response to the request.

### Create a GET Request

Follow the steps given below to create a GET request successfully in Postman:

Step 1: Click on the New menu from the Postman application. The Create New pop-up comes up. Then click on the Request link.



Step 2: SAVE REQUEST pop-up comes up. Enter the Request name then click on Save.

SAVE REQUEST

Requests in Postman are saved in collections (a group of requests).

[Learn more about creating collections](#)

Request name

Test1

Request description (Optional)

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

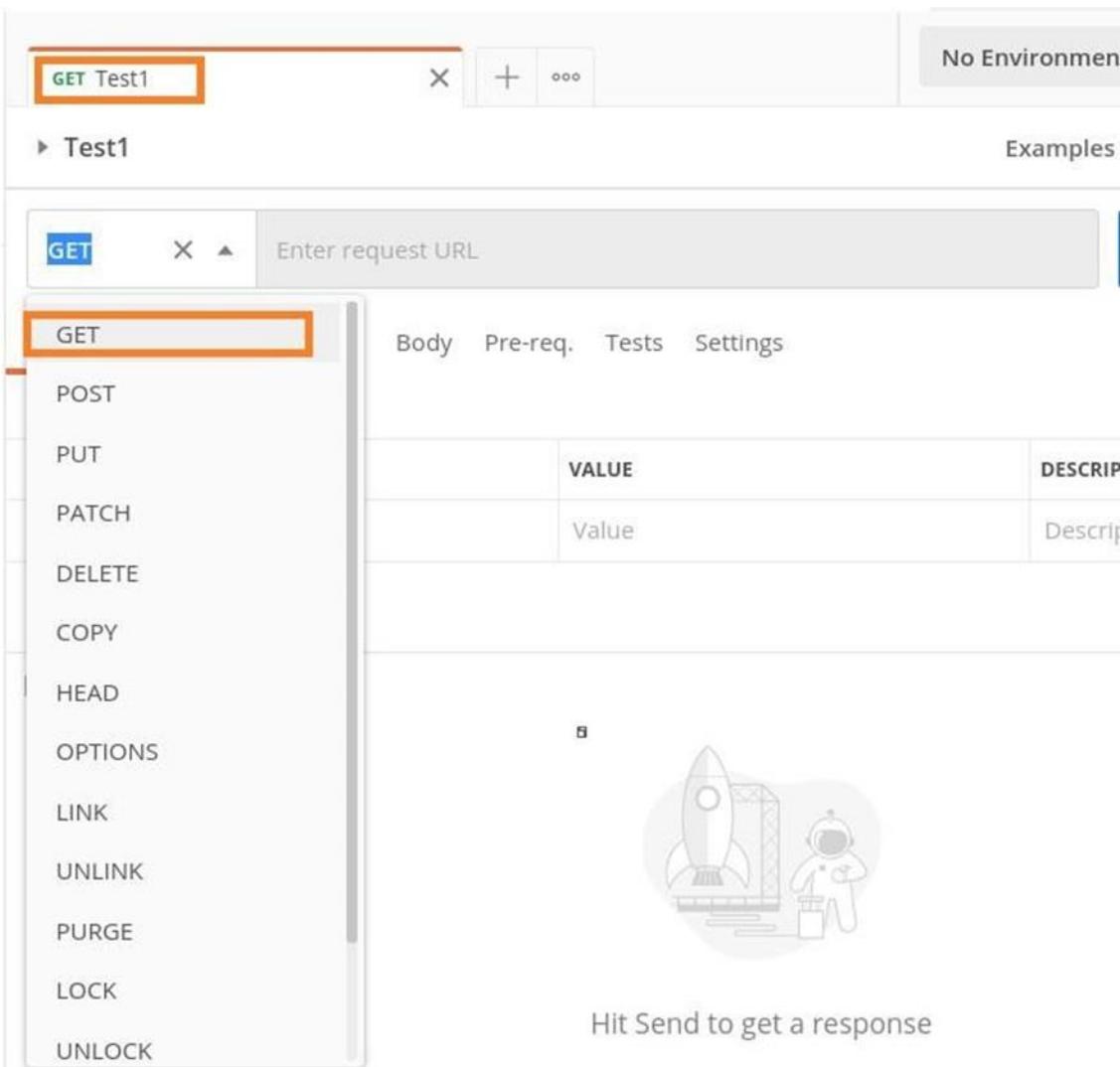
Descriptions support [Markdown](#)

Select a collection or folder to save to:

Cancel

Save to FirstTest

Step 3: The Request name (Test1) gets reflected on the Request tab. We shall then select the option GET from the HTTP request dropdown.

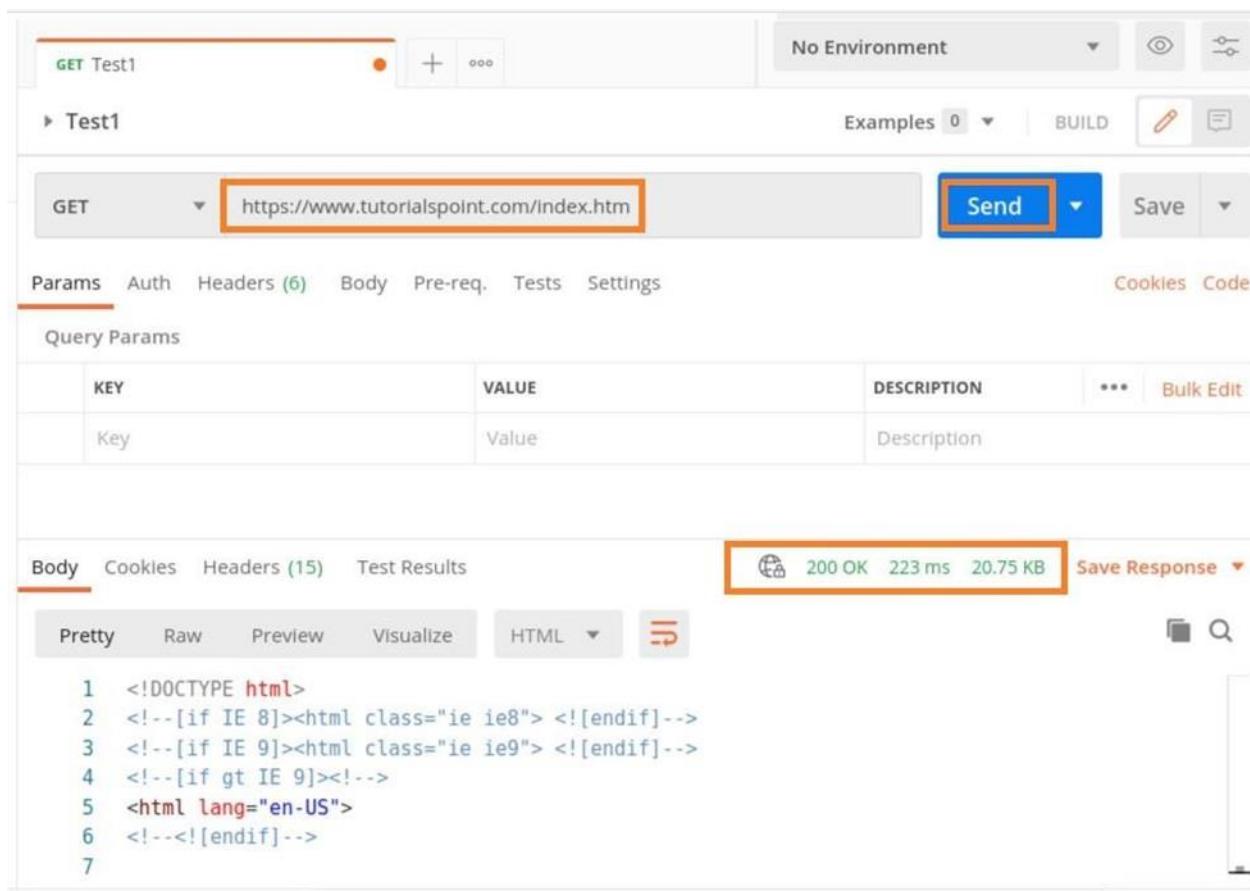


The screenshot shows the Postman application interface. At the top, there is a header with a search bar containing "GET Test1", a close button ("X"), a plus sign ("+"), and three dots ("..."). To the right of the search bar is a button labeled "No Environment". Below the header, the title "Test1" is followed by a "Examples" link. The main workspace has a "GET" method selected from a dropdown menu. An "Enter request URL" input field is present. On the left, a sidebar lists various HTTP methods: GET (selected), POST, PUT, PATCH, DELETE, COPY, HEAD, OPTIONS, LINK, UNLINK, PURGE, LOCK, and UNLOCK. To the right of the sidebar, tabs for "Body", "Pre-req.", "Tests", and "Settings" are visible. A table below these tabs contains one row with a single column labeled "Value" and a placeholder "Value". In the center-right area, there is a small icon of a rocket launching from a pad with an astronaut nearby, and the text "Hit Send to get a response".

Step 4: Enter an URL – <https://www.tutorialspoint.com/index.htm> in the address bar and click on Send.

### Response

Once a request has been sent, we can see the response code 200 OK populated in the Response. This signifies a successful request and a correct endpoint. Also, information on the time consumed to complete the request (223 ms) and payload size (20.75 KB) are populated.



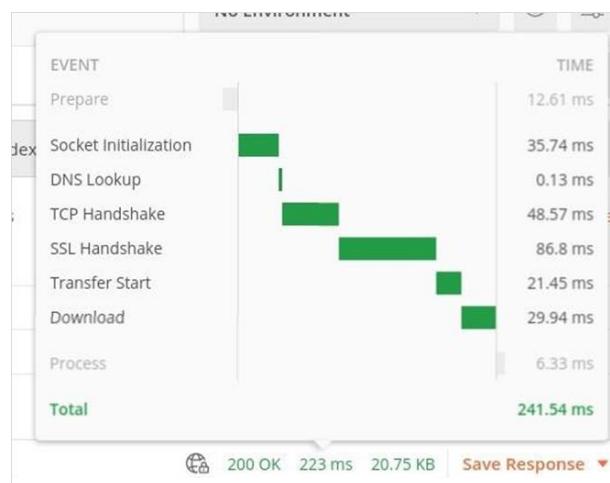
The screenshot shows the Postman interface with a test named "Test1". The request URL is set to <https://www.tutorialspoint.com/index.htm>. The response details show a 200 OK status with a time of 223 ms and a payload size of 20.75 KB. The response body is displayed in a pretty-printed HTML format.

```

1  <!DOCTYPE html>
2  <!--[if IE 8]><html class="ie ie8"> <![endif]-->
3  <!--[if IE 9]><html class="ie ie9"> <![endif]-->
4  <!--[if gt IE 9]><!-->
5  <html lang="en-US">
6  <!--<![endif]-->
7

```

On hovering over the response time, we can see the time taken by different events like DNS Lookup, SSL Handshake and so on.



On hovering over the payload size, the details on the size of response, headers, Body, and so on are displayed.

Response Size	20.75 KB
Body	20.27 KB
Headers	494 B
Request Size	218 B
Body	0 B
Headers	218 B

All size calculations are approximate

The Response Body contains the sub-tabs – Pretty, Raw and Preview. The Pretty format shows color formatting for keywords and indentation for easy reading. The Raw format displays the same data displayed in the Pretty tab but without any color or indentation.

The Preview tab shows the preview of the page.

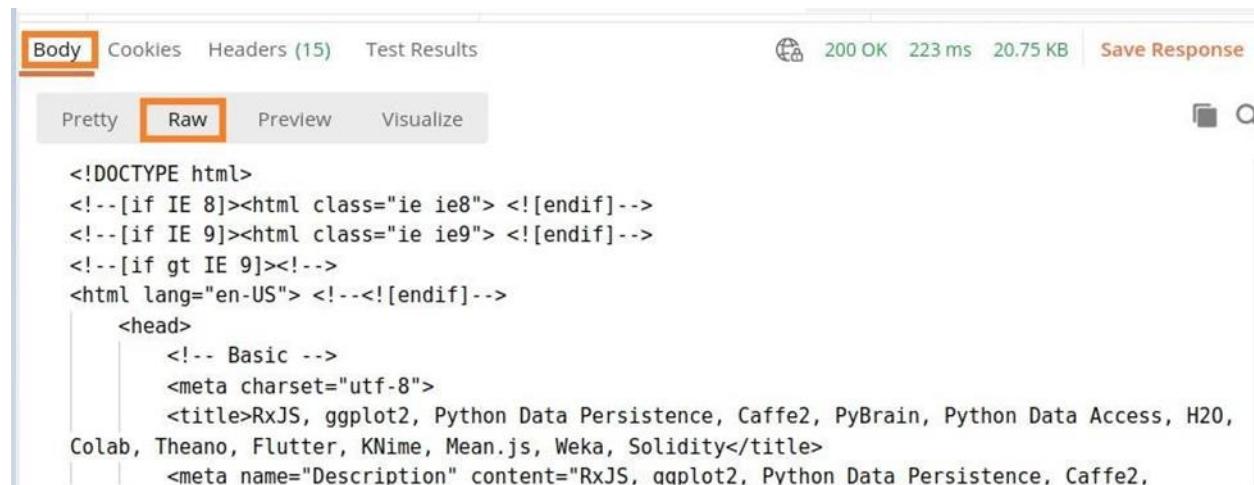
KEY	VALUE	DESCRIPTION	...
Body	Cookies Headers (15) Test Results	200 OK 223 ms 20.75 KB	<a href="#">Save Re</a>

Pretty Raw Preview Visualize HTML ▾

```
1  <!DOCTYPE html>
2  <!--[if IE 8]><html class="ie ie8"> <![endif]-->
3  <!--[if IE 9]><html class="ie ie9"> <![endif]-->
4  <!--[if gt IE 9]><!-->
5  <html lang="en-US">
6  <!--<![endif]-->
7
8  <head>
9    <!-- Basic -->
10   <meta charset="utf-8">
11   <title>RxJS, apollo2, Python Data Persistence, Caffe2, PyBrain, Python Data Access
12   <!--[if IE 9]><html class="ie ie9"> <![endif]-->
13   <!--[if gt IE 9]><!-->
14   <html lang="en-US">
```

## Raw tab

The following screen will appear:

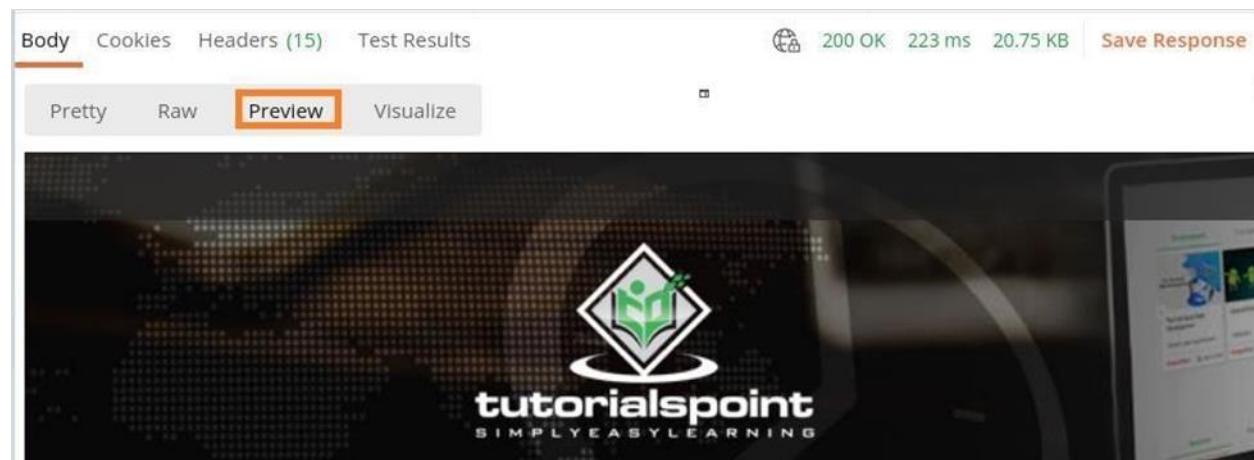


```

<!DOCTYPE html>
<!--[if IE 8]><html class="ie ie8"> <![endif]-->
<!--[if IE 9]><html class="ie ie9"> <![endif]-->
<!--[if gt IE 9]><!-->
<html lang="en-US"> <!--<![endif]-->
    <head>
        <!-- Basic -->
        <meta charset="utf-8">
        <title>RxJS, ggplot2, Python Data Persistence, Caffe2, PyBrain, Python Data Access, H2O, Colab, Theano, Flutter, KNime, Mean.js, Weka, Solidity</title>
        <meta name="Description" content="RxJS, ggplot2, Python Data Persistence, Caffe2,
    
```

## Preview tab

The following screen will appear:



The Response also contains the Cookies, Headers and Test Results.

KEY	VALUE	DESCRIPTION
Body	Cookies Headers (15) Test Results	200 OK 223 ms 20.75 KB
Strict-Transport-Security	max-age=63072000; includeSubdomains	
Vary	Accept-Encoding	
X-Cache	HIT	
X-Content-Type-Options	nosniff	
X-Frame-Options	SAMEORIGIN	
X-XSS-Protection	1; mode=block	
Content-Length	20753	

## 7. Postman – POST Requests

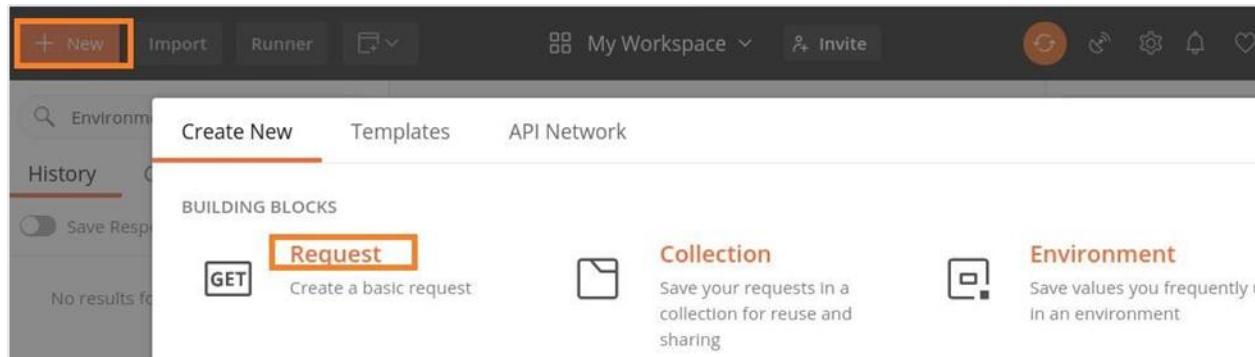
Postman POST request allows appending data to the endpoint. This is a method used to add information within the request body in the server. It is commonly used for passing delicate information.

Once we send some the request body via POST method, the API in turn yields certain information to us in Response. Thus, a POST request is always accompanied with a body in a proper format.

### Create a POST Request

Follow the steps given below to create a POST request successfully in Postman:

Step 1: Click on the New menu from the Postman application. The Create New pop-up comes up. Then, click on the Request link.



Step 2: SAVE REQUEST pop-up comes up. Enter the Request name then click on Save.

SAVE REQUEST

Requests in Postman are saved in collections (a group of requests).

[Learn more about creating collections](#)

Request name

Test1

Request description (Optional)

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

Descriptions support [Markdown](#)

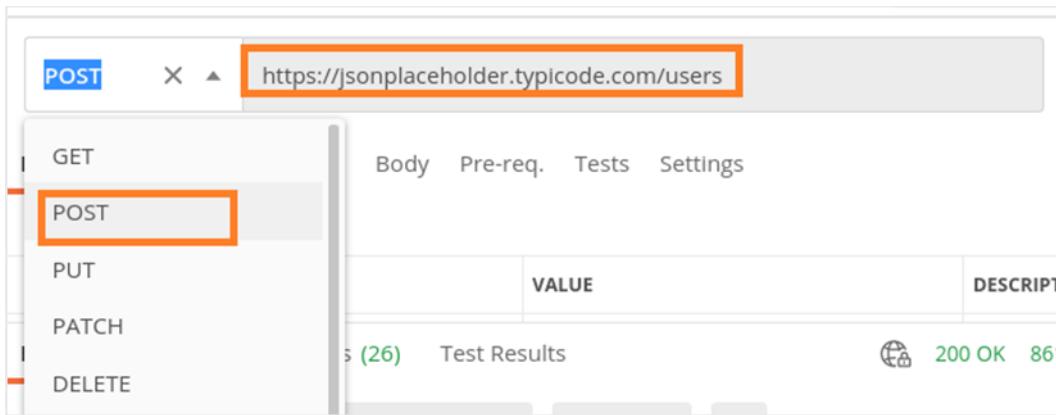
Select a collection or folder to save to:

Cancel

Save to FirstTest

Step 3: The Request name (Test1) gets reflected on the Request tab. Also, we shall select the option POST from the HTTP request dropdown.

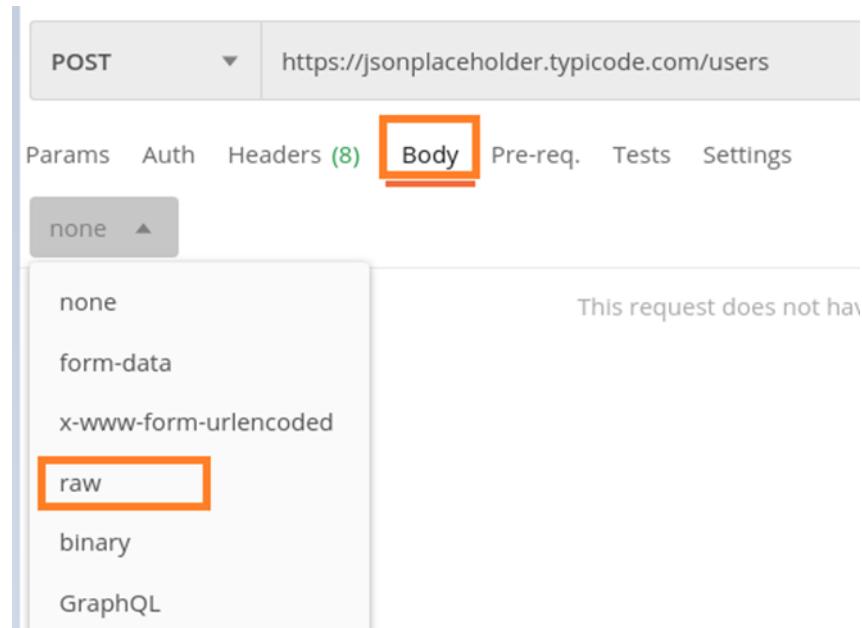
Then, enter an URL <https://jsonplaceholder.typicode.com/users> in the address bar



The screenshot shows the Postman interface with the following details:

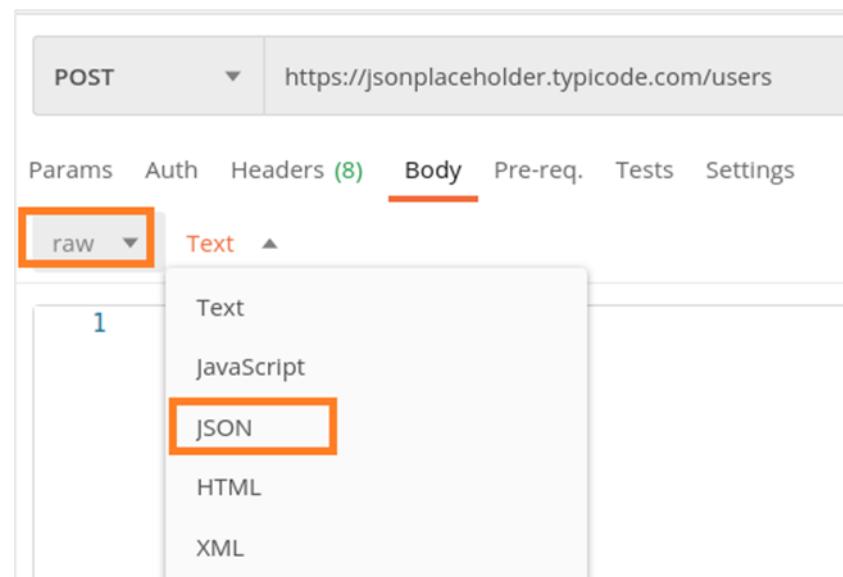
- Method:** POST (highlighted with an orange box)
- URL:** https://jsonplaceholder.typicode.com/users
- Body:** Pre-request, Tests, Settings tabs are visible.
- Test Results:** Shows 26 results, 200 OK status, and a size of 86.
- HTTP Methods:** A sidebar lists GET, POST (highlighted with an orange box), PUT, PATCH, and DELETE.

Step 4: Move to the Body tab below the address bar and select the option raw.



The screenshot shows the Postman interface with a POST request to `https://jsonplaceholder.typicode.com/users`. The 'Body' tab is selected, indicated by an orange box around it. A dropdown menu is open, showing options: none, form-data, x-www-form-urlencoded, raw (which is also highlighted with an orange box), binary, and GraphQL. To the right of the dropdown, a message says 'This request does not have a body.'

Step 5: Then, choose JSON from the Text dropdown.



The screenshot shows the same Postman interface as above, but now the 'Text' tab under the 'Body' tab is selected, indicated by an orange box around it. A dropdown menu is open, showing options: raw (which is highlighted with an orange box), Text, JavaScript, JSON (which is also highlighted with an orange box), HTML, and XML. The number '1' is visible in the background of the dropdown menu.

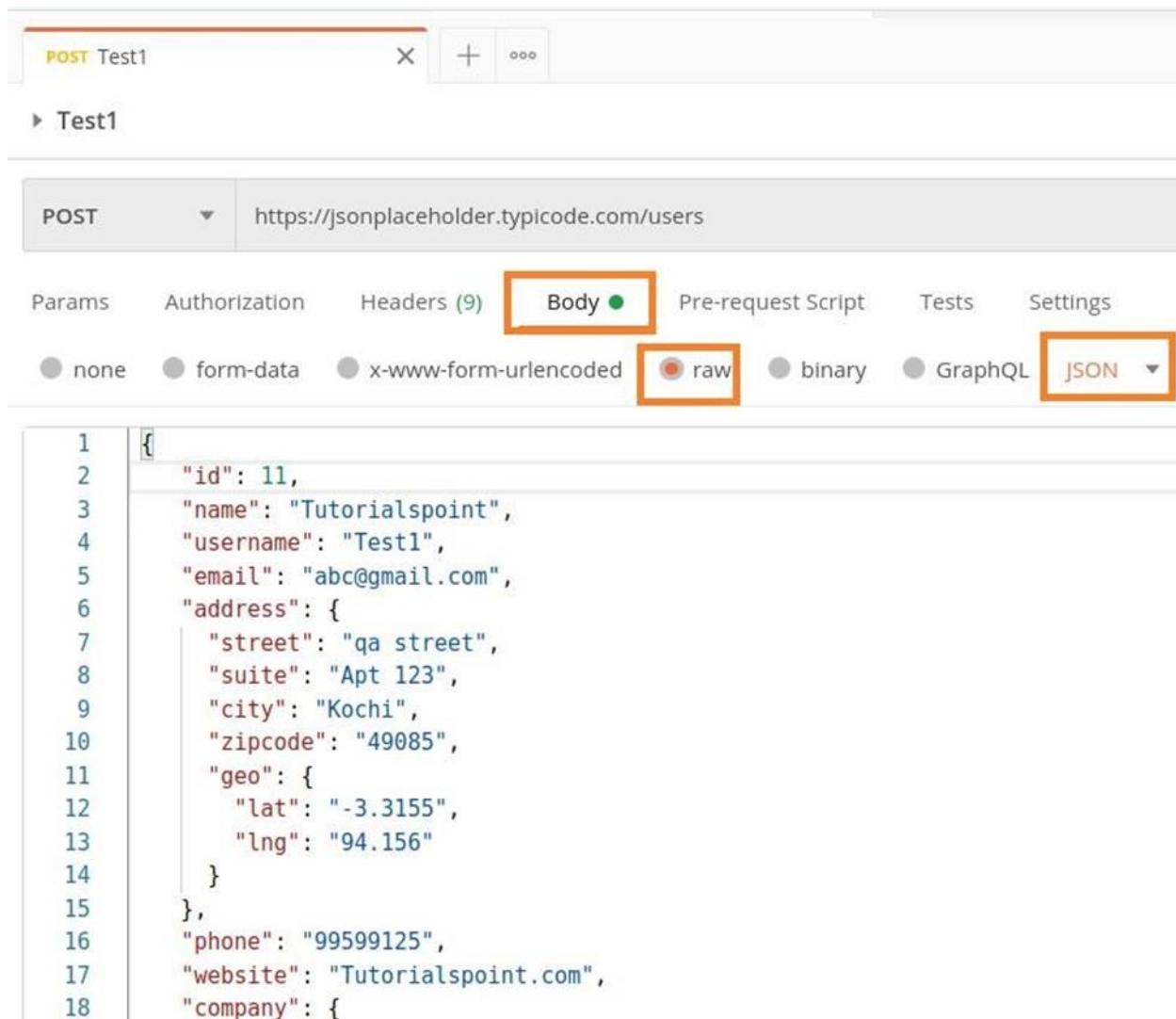
Step 6: Copy and paste the below information in the Postman Body tab.

```
{  
    "id": 11,  
    "name": "Tutorialspoint",  
    "username": "Test1",  
    "email": "abc@gmail.com",  
    "address": {  
        "street": "qa street",  
        "suite": "Apt 123",  
        "city": "Kochi",  
        "zipcode": "49085",  
        "geo": {  
            "lat": "-3.3155",  
            "lng": "94.156"  
        }  
    },  
    "phone": "99599125",  
    "website": "Tutorialspoint.com",  
    "company": {  
        "name": "Tutorialspoint",  
        "catchPhrase": "Simple Easy Learning",  
        "bs": "Postman Tutorial"  
    }  
}
```

The above data that is being sent via POST method is only applicable to the endpoint: <https://jsonplaceholder.typicode.com/users>

To pass the data in the correct JSON format, we can use the Jsonformatter available in the below link:

<https://jsonformatter.curiousconcept.com/>



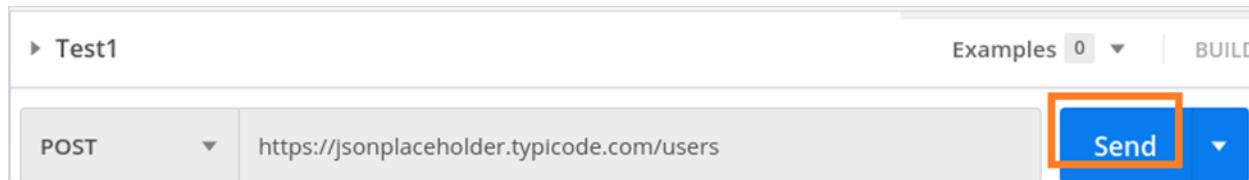
The screenshot shows the Postman application interface. At the top, there is a header bar with a 'POST' button, the title 'Test1', and a close button ('X'). Below the header, a navigation bar has 'Test1' selected. The main workspace shows a POST request to 'https://jsonplaceholder.typicode.com/users'. The 'Body' tab is active, highlighted by an orange box. Under the 'Body' tab, the 'raw' option is selected, also highlighted by an orange box. The JSON dropdown menu is open, with 'JSON' selected, also highlighted by an orange box. The request body contains a JSON object with 18 numbered lines:

```

1  {
2    "id": 11,
3    "name": "Tutorialspoint",
4    "username": "Test1",
5    "email": "abc@gmail.com",
6    "address": {
7      "street": "qa street",
8      "suite": "Apt 123",
9      "city": "Kochi",
10     "zipcode": "49085",
11     "geo": {
12       "lat": "-3.3155",
13       "lng": "94.156"
14     }
15   },
16   "phone": "99599125",
17   "website": "Tutorialspoint.com",
18   "company": {

```

Step 7: Click on the Send button.

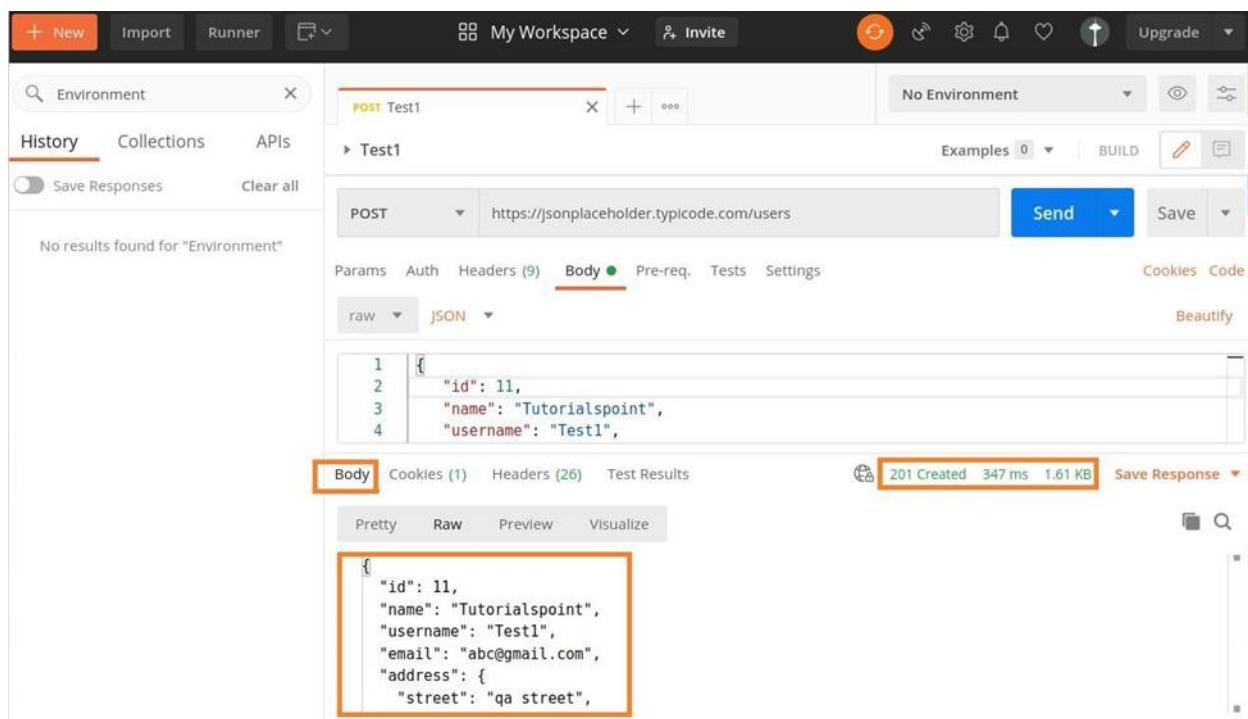


The screenshot shows the Postman interface again, focusing on the bottom right corner where the 'Send' button is located. This button is highlighted with an orange box. The rest of the interface shows the same setup as the previous screenshot, with the 'Test1' collection, 'POST' method, and target URL 'https://jsonplaceholder.typicode.com/users'.

### Response

Once a request has been sent, we can see the response code 201 Created populated in the Response. This signifies a successful request and the request we have sent has been accepted by the server.

Also, information on the time consumed to complete the request (347 ms) and payload size (1.61 KB) are populated.

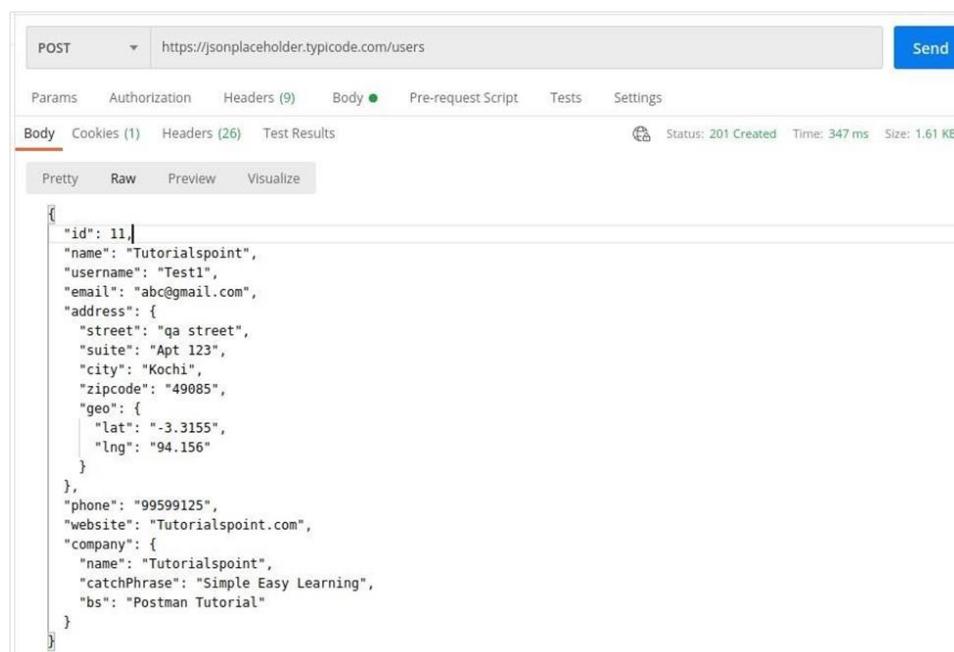


The screenshot shows the Postman interface with a successful POST request to `https://jsonplaceholder.typicode.com/users`. The request body is:

```
{
  "id": 11,
  "name": "Tutorialspoint",
  "username": "Test1",
  "email": "abc@gmail.com",
  "address": {
    "street": "qa street",
    ...
  },
  "phone": "99599125",
  "website": "Tutorialspoint.com",
  "company": {
    "name": "Tutorialspoint",
    "catchPhrase": "Simple Easy Learning",
    "bs": "Postman Tutorial"
  }
}
```

The response body is identical to the request body, indicating a successful creation of a new user. The status bar at the bottom shows `201 Created 347 ms 1.61 KB`.

We can see that the Response body is the same as the request body which we have sent to the server.



The screenshot shows the detailed response body from the previous request:

```
{
  "id": 11,
  "name": "Tutorialspoint",
  "username": "Test1",
  "email": "abc@gmail.com",
  "address": {
    "street": "qa street",
    "suite": "Apt 123",
    "city": "Kochi",
    "zipcode": "49085",
    "geo": {
      "lat": "-3.3155",
      "lng": "94.156"
    }
  },
  "phone": "99599125",
  "website": "Tutorialspoint.com",
  "company": {
    "name": "Tutorialspoint",
    "catchPhrase": "Simple Easy Learning",
    "bs": "Postman Tutorial"
  }
}
```

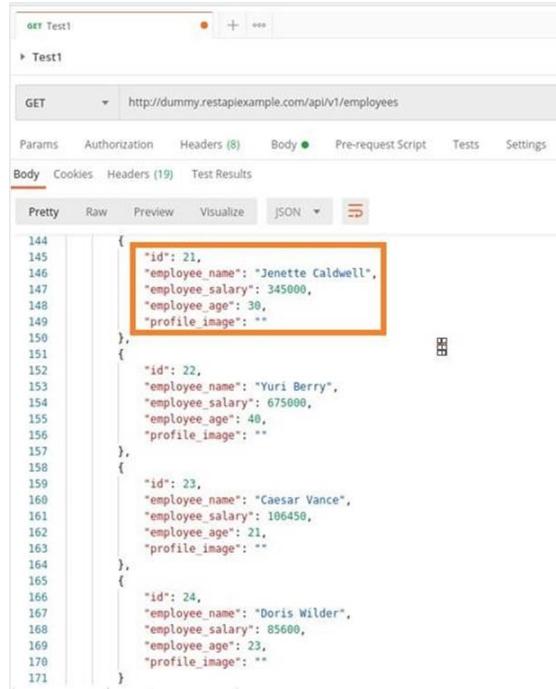
## 8. Postman – PUT Requests

A Postman PUT request is used to pass data to the server for creation or modification of a resource. The difference between POST and PUT is that POST request is not idempotent.

This means invoking the same PUT request numerous times will always yield the same output. But invoking the same POST request numerous times will create the similar resource more than one time.

Before creating a PUT request, we shall first send a GET request to the server on an endpoint: <http://dummy.restapiexample.com/api/v1/employees>. The details on how to create a GET request is explained in detail in the Chapter – Postman GET Requests.

On applying the GET method, the Response body obtained is as follows:



```

144 [
145   {
146     "id": 21,
147     "employee_name": "Jenette Caldwell",
148     "employee_salary": 345000,
149     "employee_age": 30,
150     "profile_image": ""
151   },
152   {
153     "id": 22,
154     "employee_name": "Yuri Berry",
155     "employee_salary": 675000,
156     "employee_age": 40,
157     "profile_image": ""
158   },
159   {
160     "id": 23,
161     "employee_name": "Caesar Vance",
162     "employee_salary": 106450,
163     "employee_age": 21,
164     "profile_image": ""
165   },
166   {
167     "id": 24,
168     "employee_name": "Doris Wilder",
169     "employee_salary": 85600,
170     "employee_age": 23,
171     "profile_image": ""
172 }

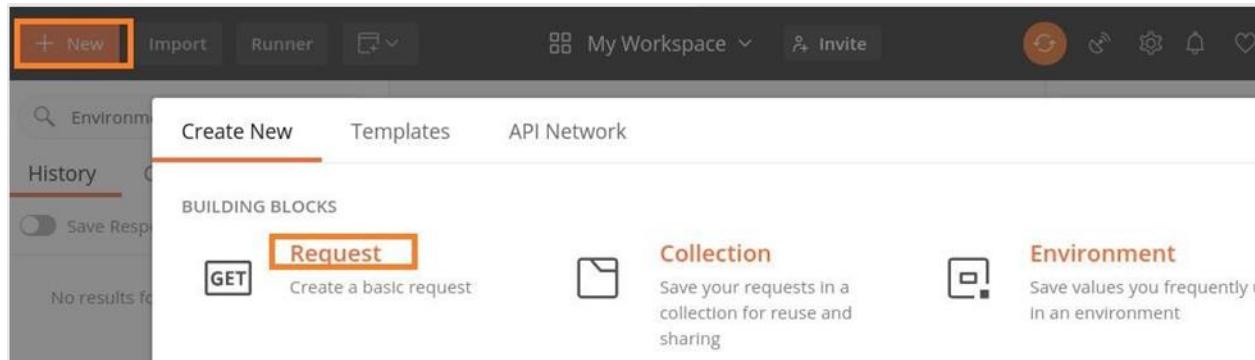
```

Now, let us update the employee\_salary and employee\_age for the id 21 with the help of the PUT request.

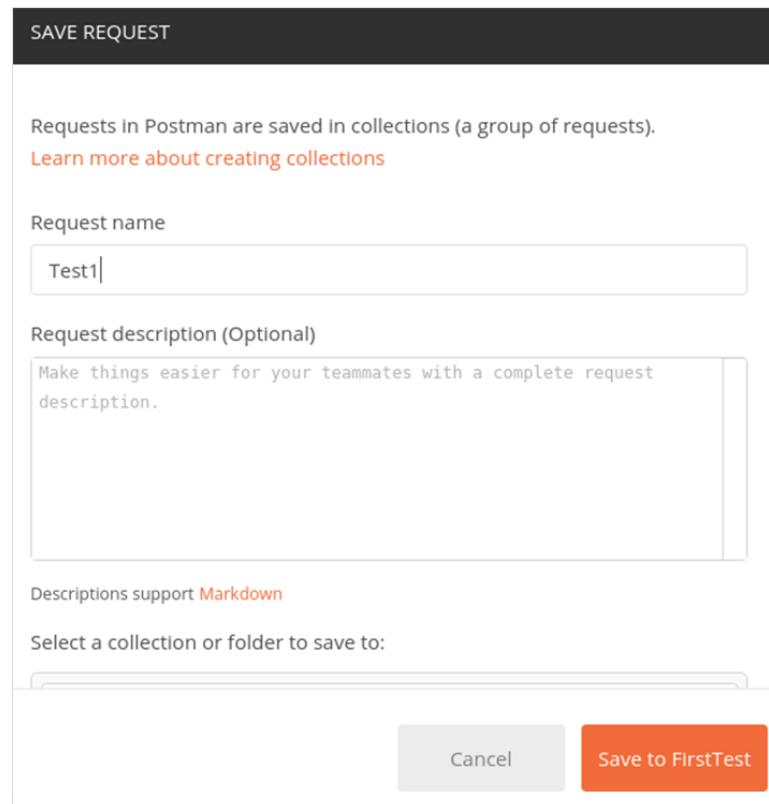
## Create a PUT Request

Follow the steps given below to create a PUT request in Postman successfully:

Step 1: Click on the New menu from the Postman application. The Create New pop-up comes up. Then, click on the Request link.



Step 2: SAVE REQUEST pop-up comes up. Enter the Request name then click on Save.

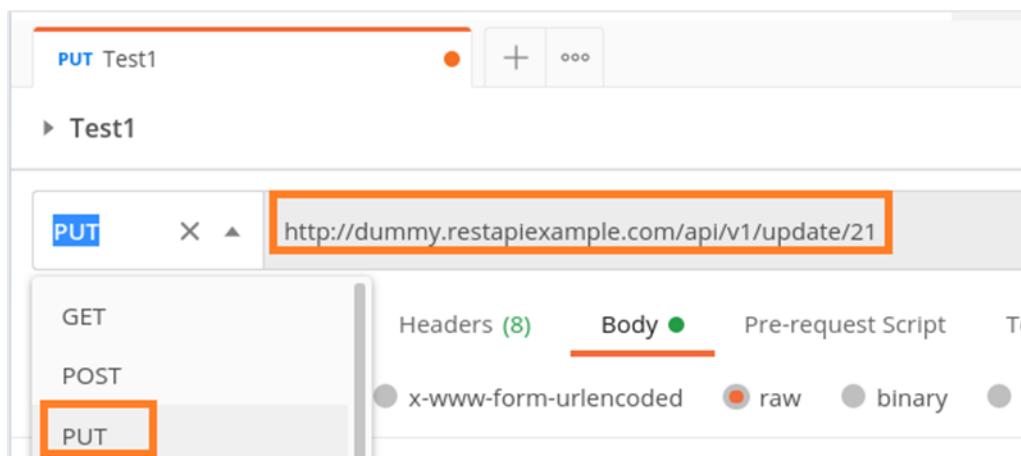


Step 3: The Request name (Test1) gets reflected on the Request tab. We shall select the option PUT from the HTTP request dropdown.

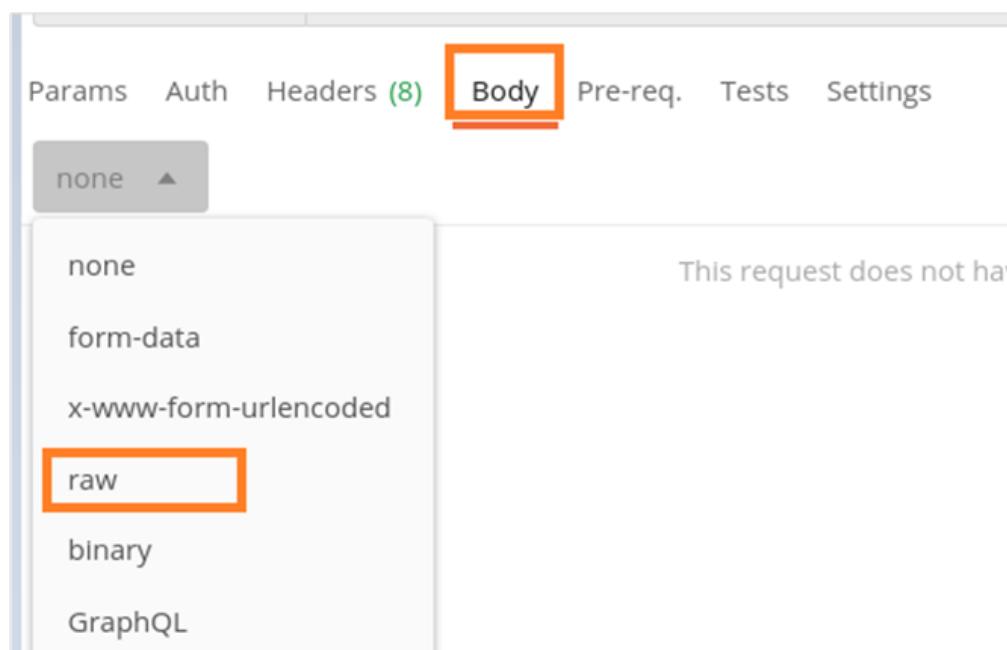
Then enter the URL - <http://dummy.restapiexample.com/api/v1/update/21> (endpoint for updating the record of id 21) in the address bar.

It must be noted that in a PUT request, we have to mention the id of the resource in the server which we want to update in the URL.

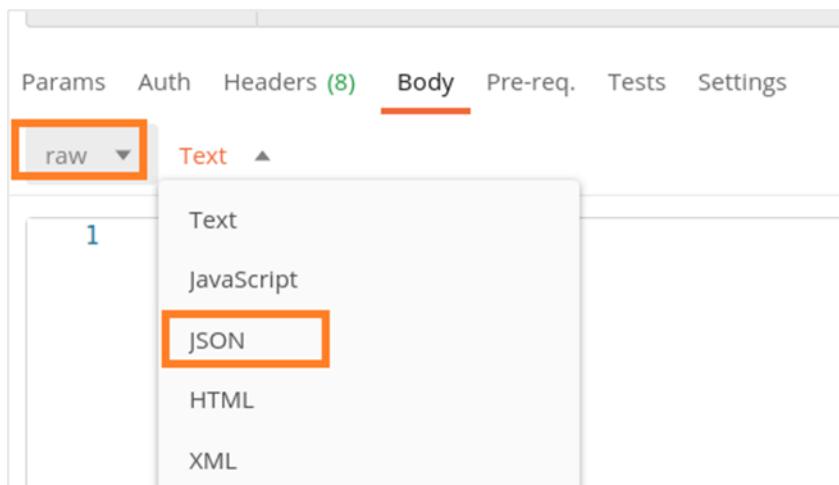
For example, in the above URL we have added the id 21.



Step 4: Move to the Body tab below the address bar and select the option raw.



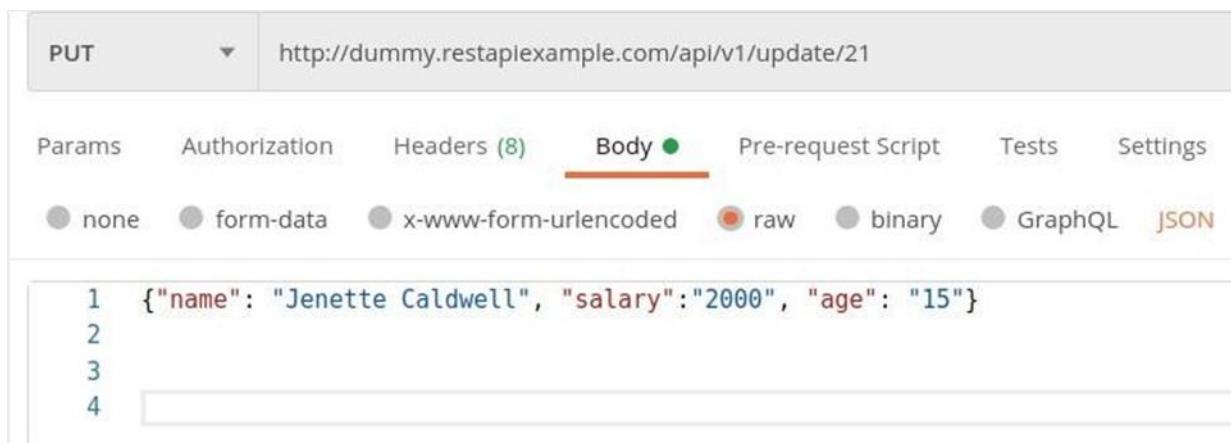
Step 5: Then, choose JSON from the Text dropdown.



Step 6: Copy and paste the below information in the Postman Body tab.

```
{ "name": "Jenette Caldwell", "salary": "2000", "age": "15"}
```

The overall parameters to be set for a PUT request are shown below:

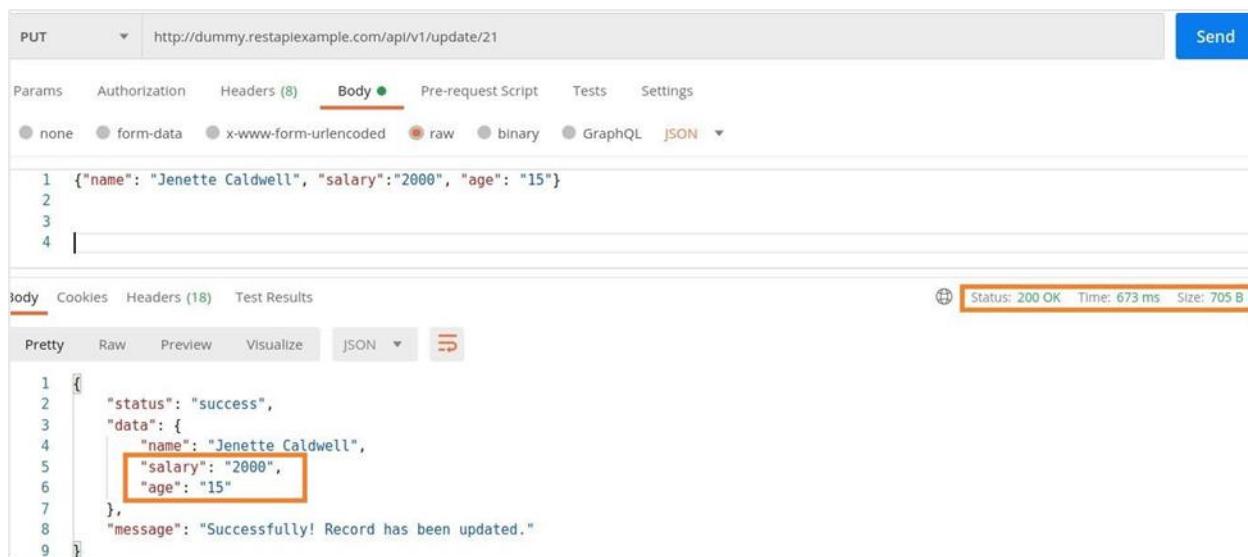


Step 7: Click on the Send button.

## Response

Once a request has been sent, we can see the response code 200 OK populated in the Response body. This signifies a successful request and the request we have sent has been accepted by the server.

Also, information on the time consumed to complete the request (673 ms) and payload size (705 B) are populated. The Response body shows the salary and age got updated to 2000 and 15 respectively for the employee having id 21.



The screenshot shows a POST request to `http://dummy.restapitexample.com/api/v1/update/21`. The Body tab is selected, showing the JSON payload:

```

1 {"name": "Jenette Caldwell", "salary": "2000", "age": "15"}
2
3
4

```

The Response tab displays the JSON response:

```

1 {
2   "status": "success",
3   "data": {
4     "name": "Jenette Caldwell",
5     "salary": "2000",
6     "age": "15"
7   },
8   "message": "Successfully! Record has been updated."
9 }

```

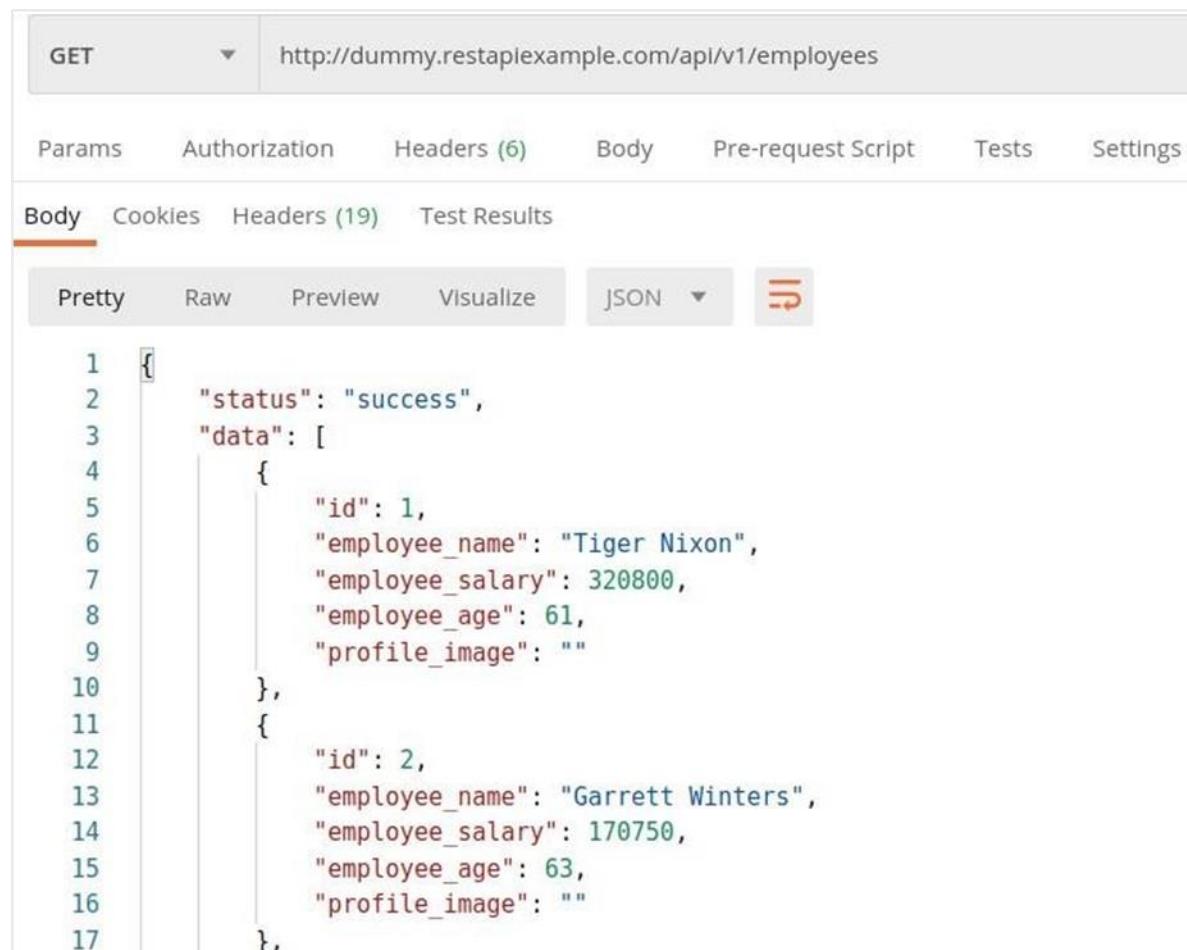
A red box highlights the updated salary value "2000". The status bar at the bottom right indicates `Status: 200 OK Time: 673 ms Size: 705 B`.

## 9. Postman — DELETE Requests

Postman DELETE request deletes a resource already present in the server. The DELETE method sends a request to the server for deleting the request mentioned in the endpoint. Thus, it is capable of updating data on the server.

Before creating a DELETE request, we shall first send a GET request to the server on the endpoint: <http://dummy.restapitexample.com/api/v1/employees>. The details on how to create a GET request is explained in detail in the Chapter on GET Requests.

On applying the GET method, the below Response Body is obtained:



The screenshot shows the Postman interface with a successful GET request to <http://dummy.restapitexample.com/api/v1/employees>. The response body is displayed in Pretty JSON format, showing a list of employee records. The JSON structure is as follows:

```

1  {
2      "status": "success",
3      "data": [
4          {
5              "id": 1,
6              "employee_name": "Tiger Nixon",
7              "employee_salary": 320800,
8              "employee_age": 61,
9              "profile_image": ""
10         },
11         {
12             "id": 2,
13             "employee_name": "Garrett Winters",
14             "employee_salary": 170750,
15             "employee_age": 63,
16             "profile_image": ""
17         }
]

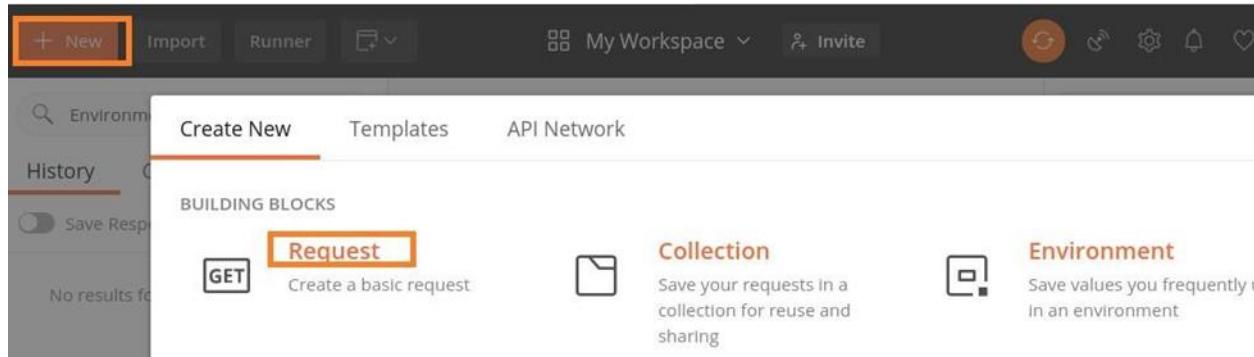
```

Let us delete the record of the id 2 from the server.

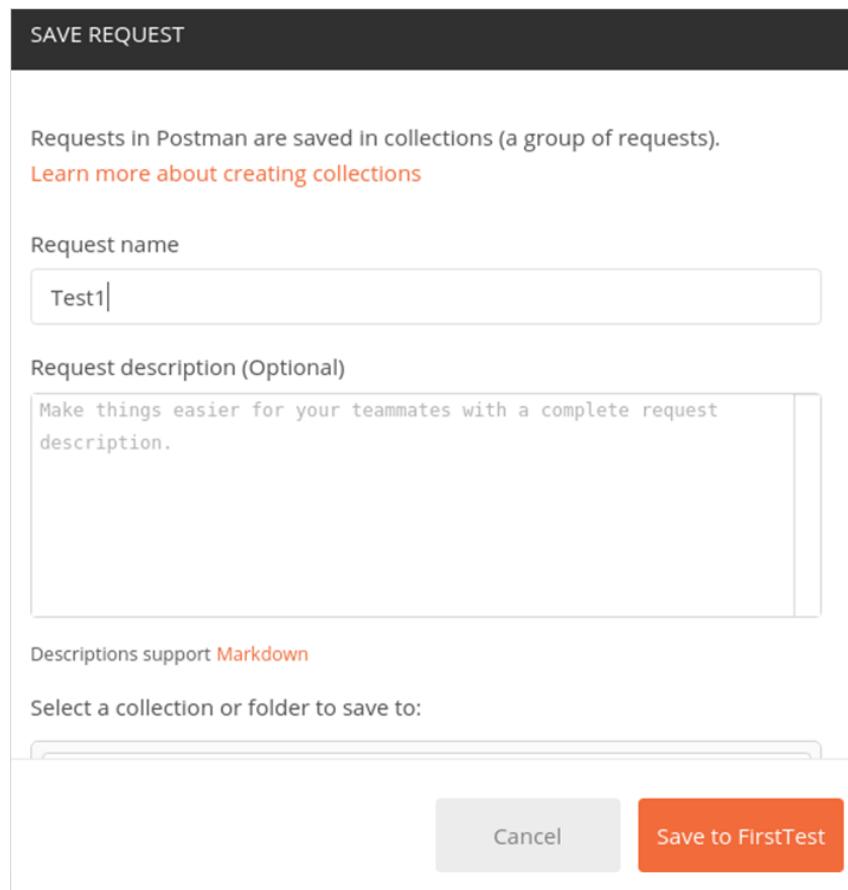
## Create a DELETE Request

Follow the steps given below to create a DELETE request in Postman successfully:

Step 1: Click on the New menu from the Postman application. The Create New pop-up comes up. Then, click on the Request link.



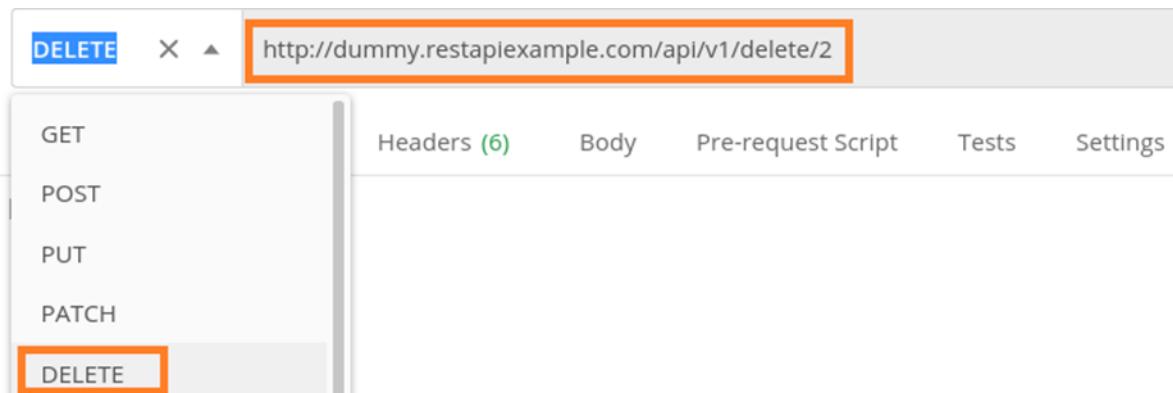
Step 2: SAVE REQUEST pop-up comes up. Enter the Request name then click on Save.



Step 3: The Request name (Test1) gets reflected on the Request tab. We shall select the option DELETE from the HTTP request dropdown.

Then enter the URL - <http://dummy.restapixexample.com/api/v1/delete/2> (endpoint for deleting the record of id 2) in the address bar.

Here, in the DELETE request, we have mentioned the id of the resource in the server which we want to delete in the URL.

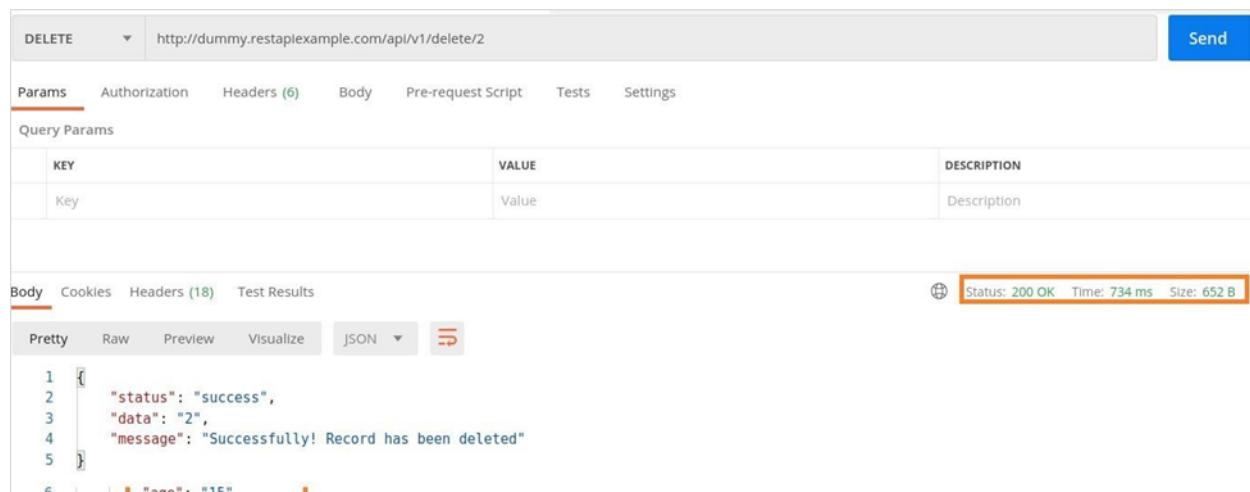


Step 4: Click on the Send button.

### Response

Once a request has been sent, we can see the Response code 200 OK populated in the Response. This signifies a successful request and the request we have sent has been accepted by the server.

Also, information on the time consumed to complete the request (734 ms) and payload size (652 B) are populated. The Response shows the status as success. The record id 2 gets deleted from the server.



The screenshot shows a Postman request for a DELETE operation on the endpoint <http://dummy.restapieexample.com/api/v1/delete/2>. The response status is 200 OK, time taken is 734 ms, and the size is 652 B. The response body is a JSON object:

```

1  {
2    "status": "success",
3    "data": "2",
4    "message": "Successfully! Record has been deleted"
5  }
6  |   "ana" + "15"

```

After deletion of the record with id 2, if we run the GET request on the endpoint: <http://dummy.restapieexample.com/api/v1/employee/2>, we shall receive 401 Unauthorized status code.

## 10. Postman – Create Tests for CRUD

CRUD stands for Create, Retrieve, Update and Delete operations on any website opened in a browser. Whenever we launch an application, the retrieve operation is performed.

On creating data, for example, adding a new user for a website, the create operation is performed. If we are modifying the information, for example, changing details of an existing customer in a website, the update operation is performed.

Finally, to eliminate any information, for example, deleting a user in a website, the delete operation is carried out.

To retrieve a resource from the server, the HTTP method – GET is used (discussed in details in the Chapter – Postman GET Requests). To create a resource in the server, the HTTP method – POST is used (discussed in details in the Chapter – Postman POST Requests).

To modify a resource in the server, the HTTP method – PUT is used (discussed in details in the Chapter – Postman PUT Requests). To delete a resource in the server, the HTTP method – DELETE is used (discussed in details in the Chapter – Postman DELETE Requests).

### Tests in Postman

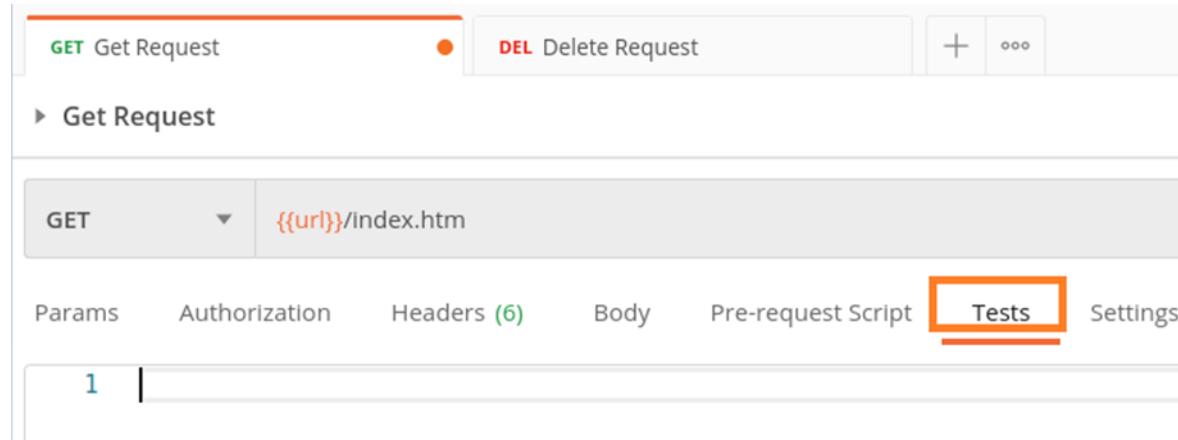
A Postman test is executed only if a request is successful. If a Response Body is not generated, it means our request is incorrect and we will not be able to execute any test to validate a Response.

In Postman, tests are developed in JavaScript and can be developed using the JavaScript and Functional methods. Both the techniques are based on the language JavaScript.

## JavaScript Method

Follow the steps given below to develop tests in Javascript:

Step 1: Tests developed in the JavaScript method are mentioned within the Tests tab under the address bar.



The screenshot shows the Postman interface with a 'Tests' tab highlighted by a red box. The top navigation bar includes 'GET Get Request' and 'DEL Delete Request'. Below the navigation are sections for 'GET' method, URL endpoint {{url}}/Index.htm, and various request parameters like 'Params', 'Authorization', 'Headers (6)', 'Body', 'Pre-request Script', and 'Tests'. A test script '1' is visible in the 'Tests' section.

Step 2: Add the below JavaScript verifications within the Tests tab:

```
tests["Status Code should be 200"] = responseCode.code === 200
tests["Response time lesser than 10ms"] = responseTime<10
```

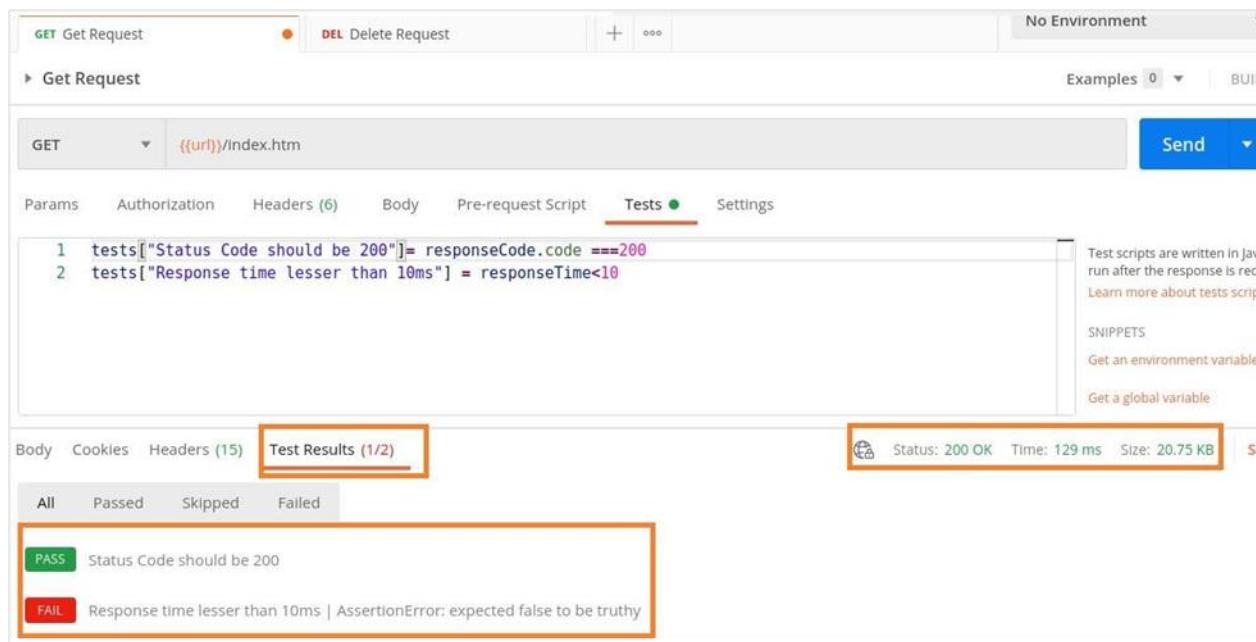
We can add one or more than one test for a particular request.

Here, tests is a variable of type array which can hold data types- integer, string, Boolean and so on. The Status Code should be 200 and Response time lesser than 10ms are the names of the tests. It is recommended to give meaningful names to test.

The responseCode.code is the response code obtained in the Response and the responseTime is the time taken to get the Response.

Step 3: Select the GET method and enter an endpoint then click on Send. Response

In the Response, click on the Test Results tab:



The screenshot shows the Postman interface with a GET request to `http://{{url}}/Index.htm`. The Tests tab contains the following code:

```

1 tests["Status Code should be 200"] = responseCode.code === 200
2 tests["Response time lesser than 10ms"] = responseTime < 10
  
```

The Test Results tab indicates 1/2 tests passed. The results show:

- PASS Status Code should be 200
- FAIL Response time lesser than 10ms | Assertion Error: expected false to be truthy

The response summary shows:

- Status: 200 OK
- Time: 129 ms
- Size: 20.75 KB

The Test Results tab shows the test which has passed in green and the test which has failed in red. The Test Results (1/2) means one out of the two tests has passed.

Response shows the status as 200 OK and Response time as 129ms (the second test checks if the Response time is less than 10ms).

Hence, the first test got passed and the second one failed along with the Assertion error.

## Functional Method

Follow the steps given below to develop a test in with functional method:

Step 1: Tests developed in the Functional method are mentioned within the Tests tab under the address bar.

Step 2: Add the below code within the Tests tab:

```

pm.test["Status Code is 401"], function(){
    pm.response.to.have.status(401)
}
  
```

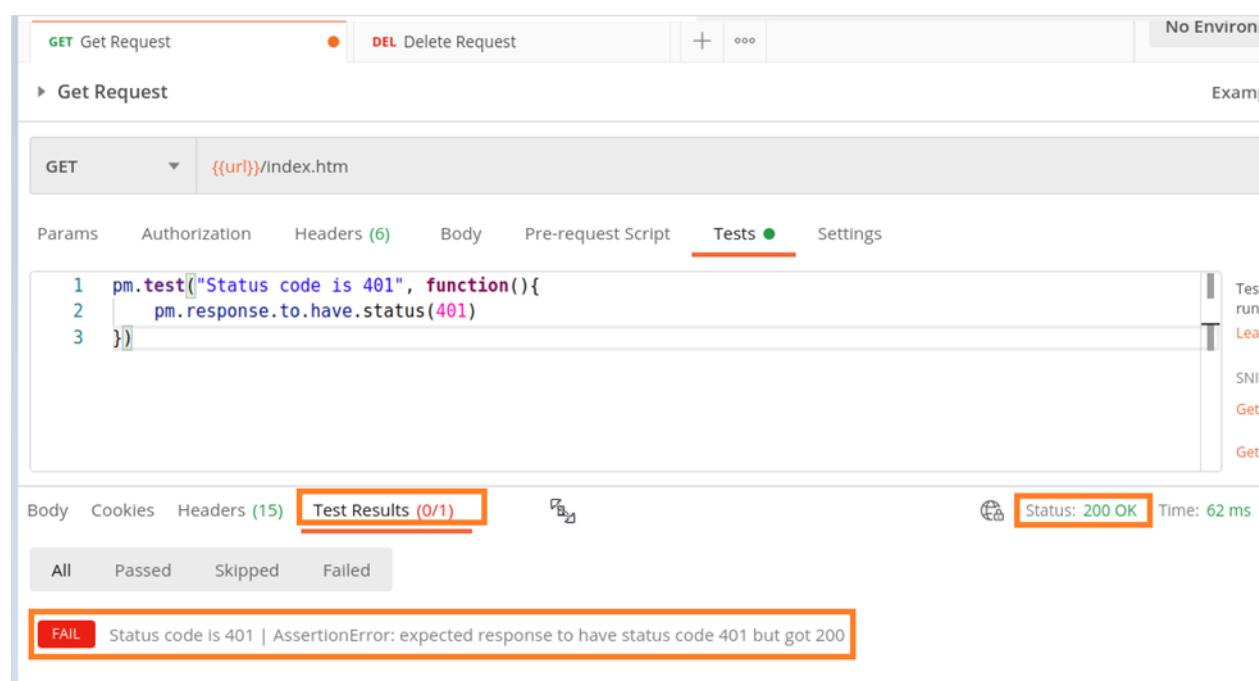
Here, pm.test is the function for the test being performed. Status Code is 401 and it is the name of the test which shall be visible in the Test Result after execution.

The pm.response is used for obtaining the response and adding assertions on it to verify the header, code, status, and so on.

Step 3: Select the GET method and enter an endpoint then click on Send.

### Response

In the Response, click on the Test Results tab:



The screenshot shows the Postman interface with a 'Get Request' collection selected. A single GET request is defined with the URL {{url}}/Index.htm. The 'Tests' tab is active, containing the following script:

```
1 pm.test("Status code is 401", function(){
2   pm.response.to.have.status(401)
3 })
```

The 'Test Results' tab is highlighted in red, showing 0/1 results. The status bar indicates 'Status: 200 OK' and 'Time: 62 ms'. A red box highlights the error message: 'FAIL Status code is 401 | Assertion Error: expected response to have status code 401 but got 200'.

The Test Results tab shows the test in red as the test has failed. The Test Results (0/1) means zero out of the one test has passed. Response shows the status as 200 OK (the test checks if the response code is 401).

Hence the test shows failed along with the Assertion error.

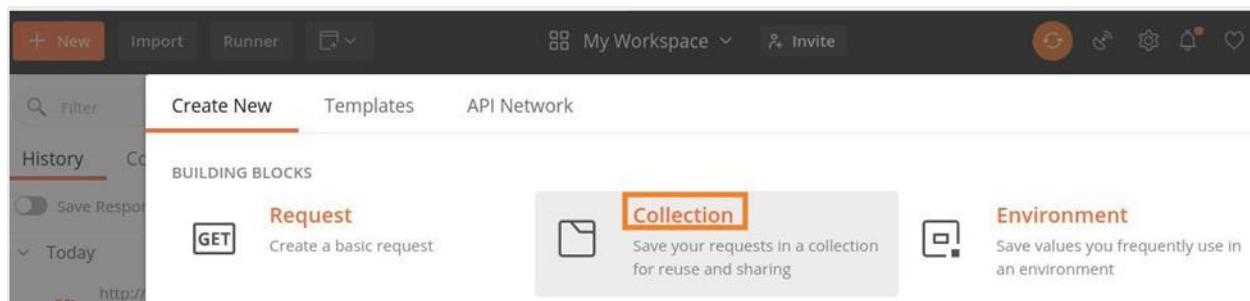
## 11. Postman – Create Collections

A group of requests that have been saved and organized into folders is known as the Collections. It is similar to a repository. Thus, Collections help to maintain the API tests and also split them easily with teams.

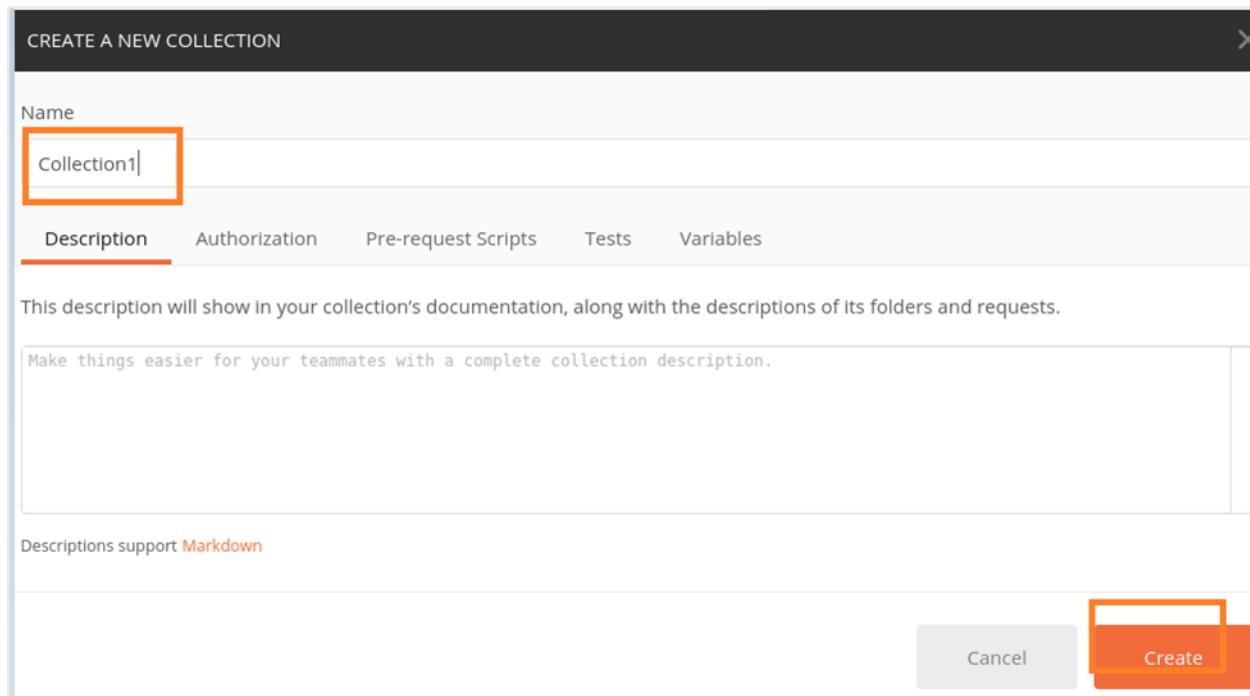
### Create a New Collection

Follow the steps given below to create a new collection in Postman:

Step 1: Click on the New icon from the Postman application. The Create New pop-up comes up. Then click on the Collection link.



Step 2: CREATE A NEW COLLECTION pop-up comes up. Enter a Collection Name and click on the Create button.



CREATE A NEW COLLECTION

Name  
Collection1

Description    Authorization    Pre-request Scripts    Tests    Variables

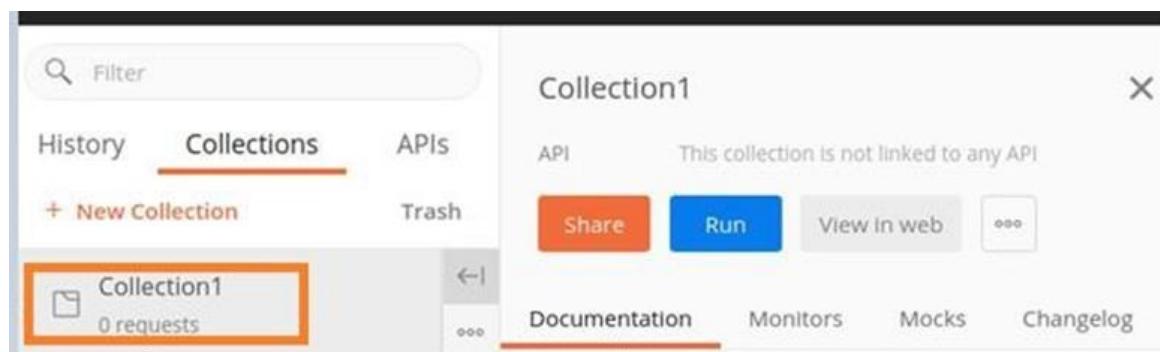
This description will show in your collection's documentation, along with the descriptions of its folders and requests.

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

Descriptions support [Markdown](#)

Cancel    Create

Step 3: The Collection name and the number of requests it contains are displayed in the sidebar under the Collections tab



Filter

History    **Collections**    APIs

+ New Collection    Trash

Collection1

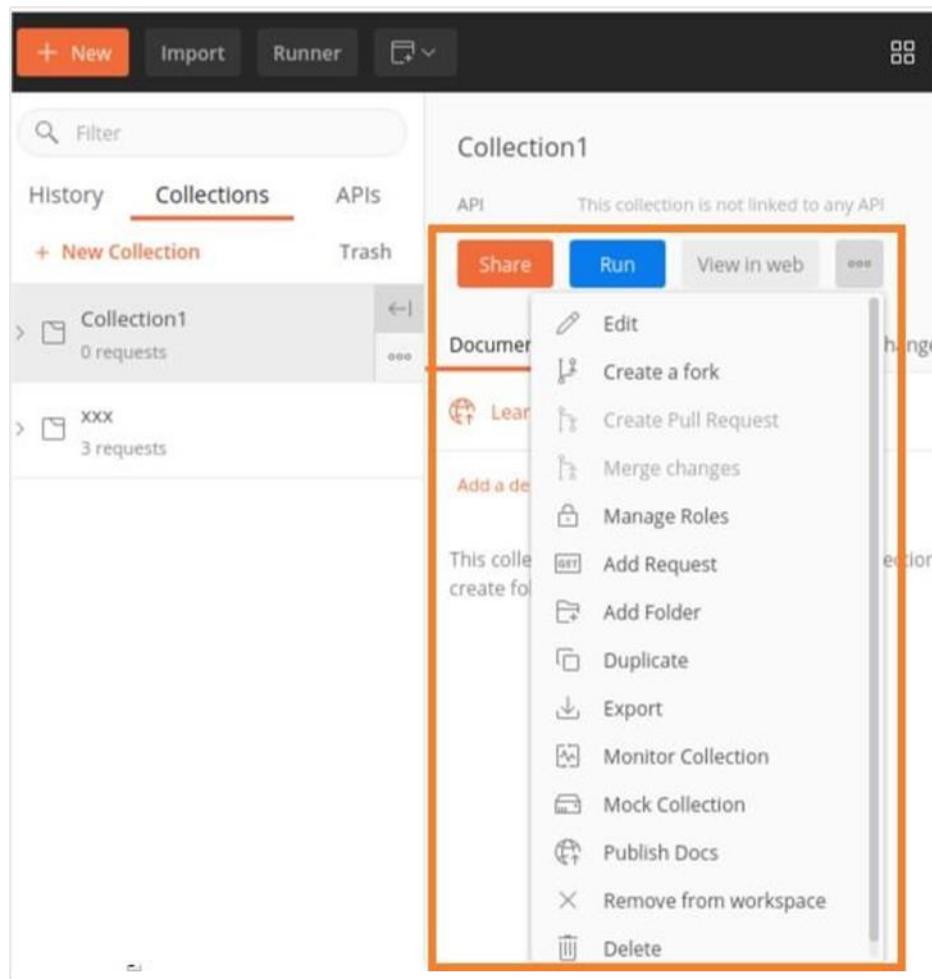
API    This collection is not linked to any API

Share    Run    View in web    ...

Collection1    0 requests

Documentation    Monitors    Mocks    Changelog

Step 4: To the right of the Collection name, we have the options like Share, Run and so on available. Click on the three dots to get more options to select.



Step 5: Click on Add Request. The SAVE REQUEST pop-up comes up. Enter Request Name and select the Collection we have created. Then, click on the Save to Collection1 button.

SAVE REQUEST

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:

[+ Create Folder](#)

Step 6: The Collection with its request gets displayed to the side bar under the Collections tab.

+ New Import Runner 

Filter

History Collections APIs

+ New Collection Trash

Collection1  
1 request

GET Get Request

✉ [hr@kasperanalytics.com](mailto:hr@kasperanalytics.com)

 [kasper-analytics](#)

## 12. Postman – Parameterize Requests

We can parameterize Postman requests to execute the same request with various sets of data. This is done with the help of variables along with parameters. A parameter is a part of the URL used to pass more information to the server.

The data can be used in the form of a data file or an Environment variable. Parameterization is an important feature of Postman and helps to eliminate redundant tests. Parameters are enclosed in double curly braces {{parameter}}.

### Example

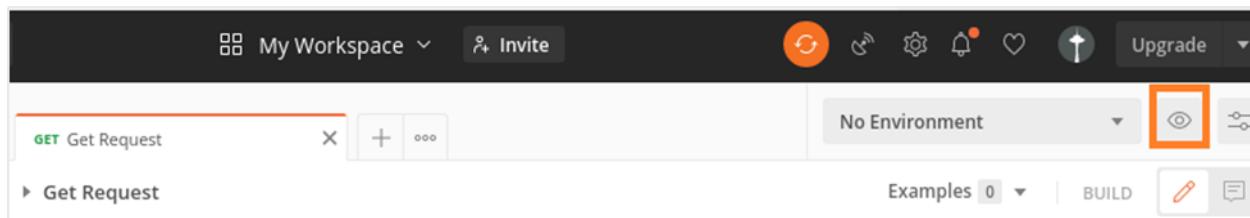
Let us take an example of an URL: <https://www.tutorialspoint.com/index.htm>. We shall create a variable as url then use it for parameterization of request. We can refer to it in the format {{url}} in Postman.

A parameter is in the form of a key-value pair. So to point to the URL: <https://www.tutorialspoint.com/index.htm>, we can mention it as {{url}}/index.htm. So here, the url is the key and the value set is <https://www.tutorialspoint.com>.

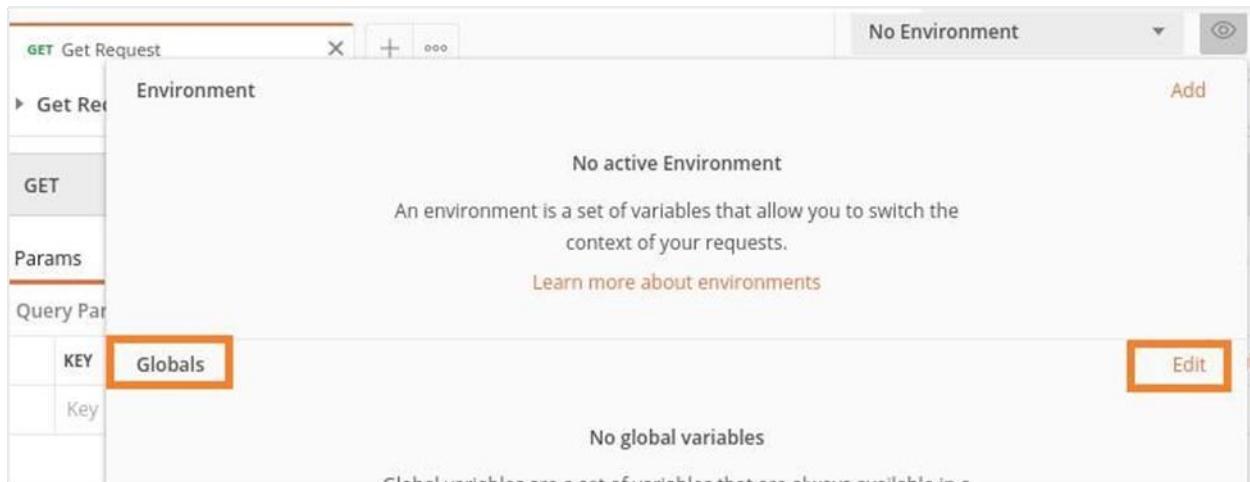
### Create a Parameter Request

Follow the steps given below to create a parameter request in Postman:

Step 1: Click on the eye icon to the right of the Environment dropdown in the top right corner in the Postman application.



Step 2: Click on the Edit link in the Globals section.



GET Get Request

No Environment

Environment

GET

No active Environment

An environment is a set of variables that allow you to switch the context of your requests.

Learn more about environments

Params

Query Par

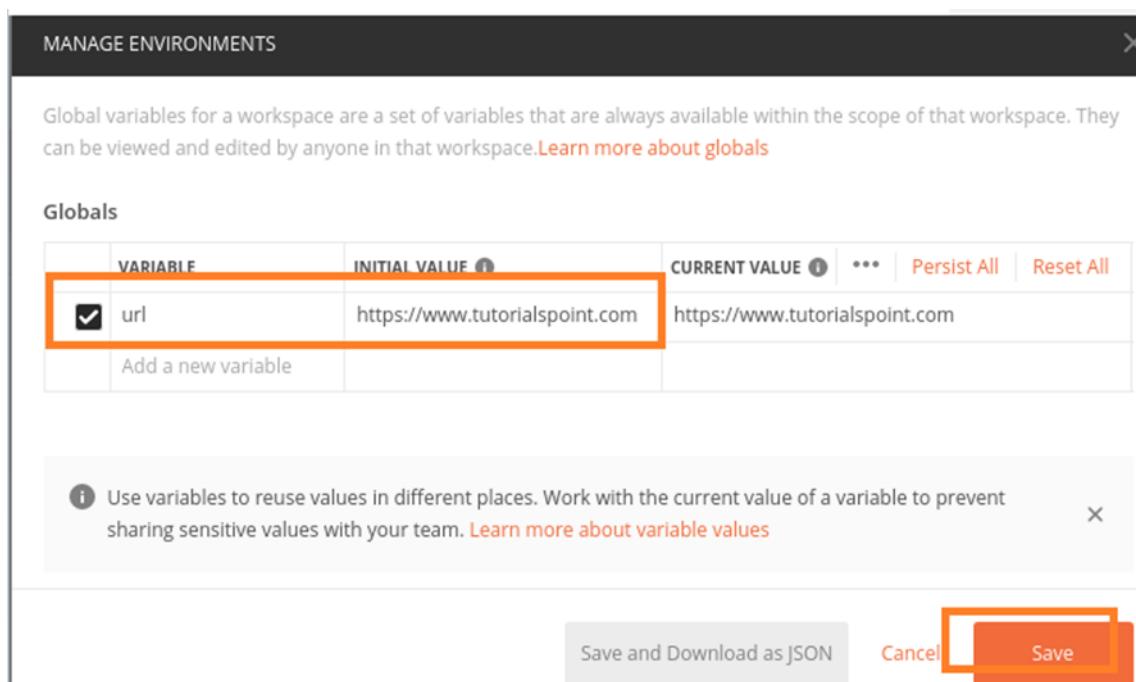
KEY	Globals
Key	

Edit

No global variables

Global variables are a set of variables that are always available in a

Step 3: MANAGE ENVIRONMENTS pop-up comes up. Enter URL for the VARIABLE field and <https://www.tutorialspoint.com> for INITIAL VALUE. Then, click on Save.



MANAGE ENVIRONMENTS

Global variables for a workspace are a set of variables that are always available within the scope of that workspace. They can be viewed and edited by anyone in that workspace. [Learn more about globals](#)

Globals

VARIABLE	INITIAL VALUE	CURRENT VALUE	...	Persist All	Reset All
<input checked="" type="checkbox"/> url	https://www.tutorialspoint.com	https://www.tutorialspoint.com			
Add a new variable					

Use variables to reuse values in different places. Work with the current value of a variable to prevent sharing sensitive values with your team. [Learn more about variable values](#)

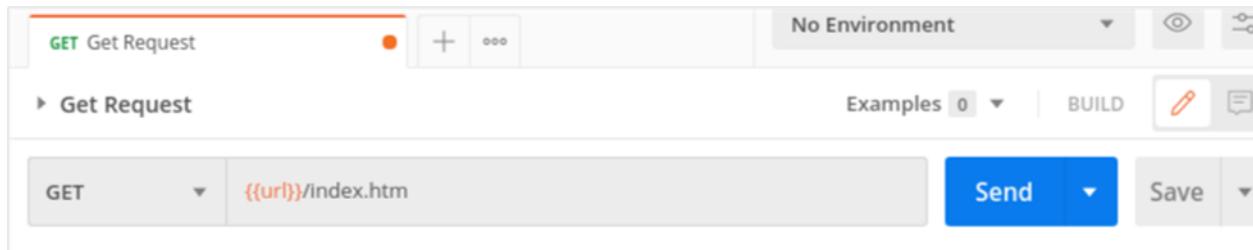
Save and Download as JSON

Cancel

Save

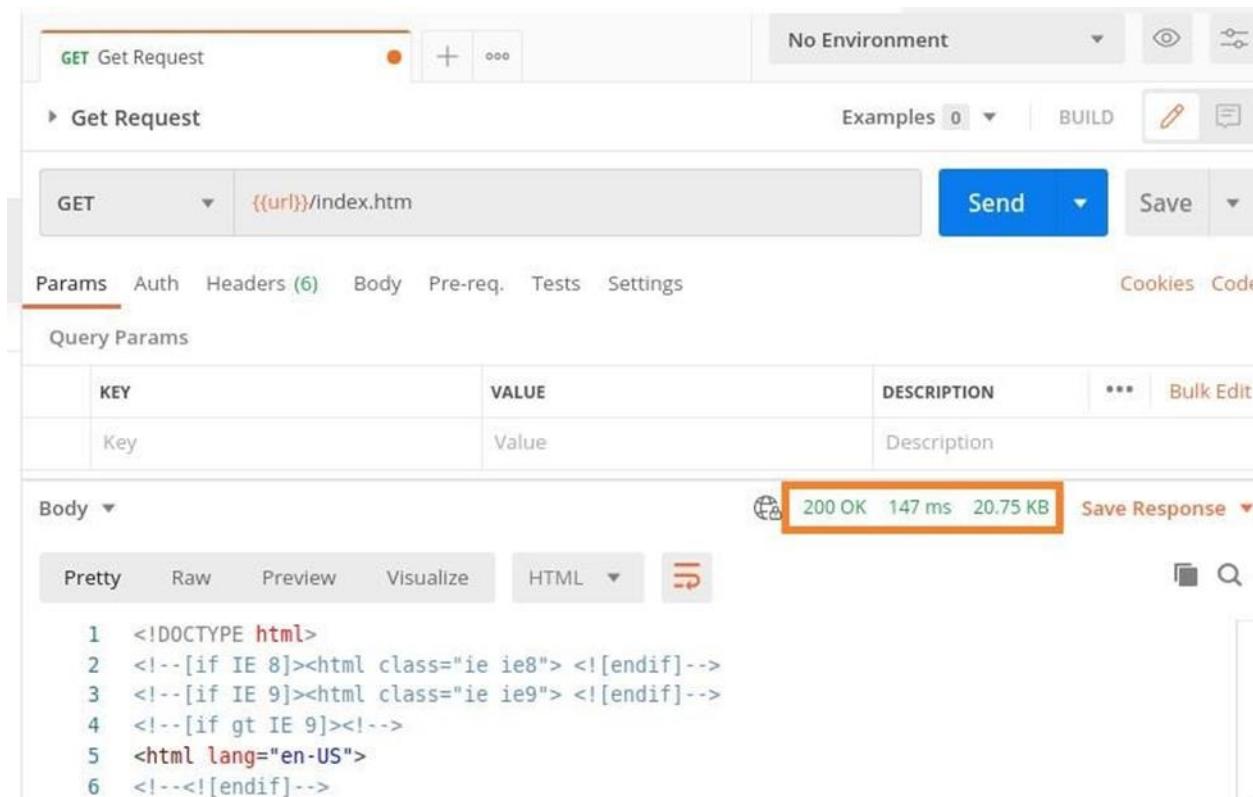
Step 4: Click on close to move to the next screen.

Step 5: In the Http Request tab, enter {{url}}/index.htm in the address bar. Select the GET method and click on Send.



## Response

Once a request has been sent, we can see the response code 200 OK populated in the Response. This signifies a successful request and a correct endpoint.



The screenshot shows the Postman interface after sending the request. At the top, the status is '200 OK' with a duration of '147 ms' and a size of '20.75 KB'. Below this, the 'Body' section is expanded, showing the response content in 'Pretty' format:

```

1 <!DOCTYPE html>
2 <!--[if IE 8]><html class="ie ie8"> <![endif]-->
3 <!--[if IE 9]><html class="ie ie9"> <![endif]-->
4 <!--[if gt IE 9]><!-->
5 <html lang="en-US">
6 <!--<![endif]-->

```

## 13. Postman – Collection Runner

Postman Collection Runner is used to execute a Collection having multiple requests together. All the requests within a Collection will be executed simultaneously. The Collection Runner does not produce any Response Body.

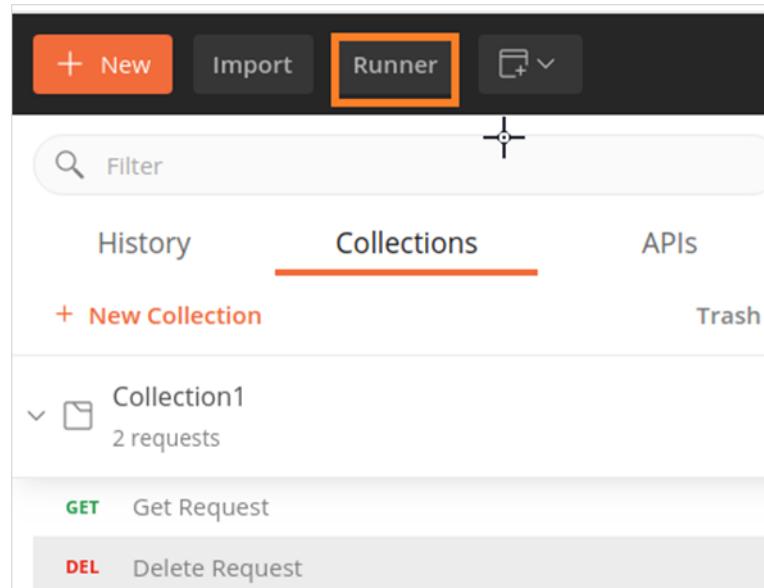
The Collection Runner console displays the test results for individual requests. It is mandatory to have more than one request within the Collection to work with Collection Runner.

The details on how to create a Collection is discussed in detail in the Chapter on Create Collections.

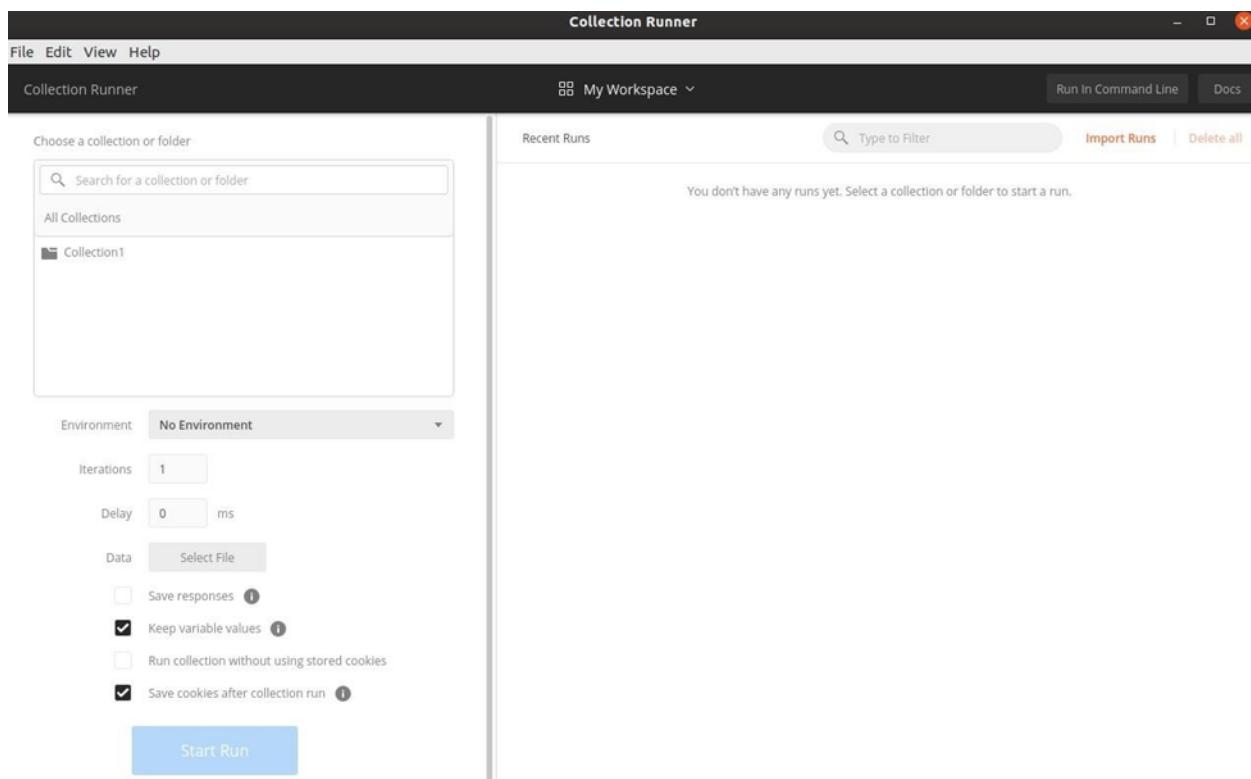
### Execute Tests with Collection Runner

Follow the steps given below to execute the tests with Collection Runner in Postman:

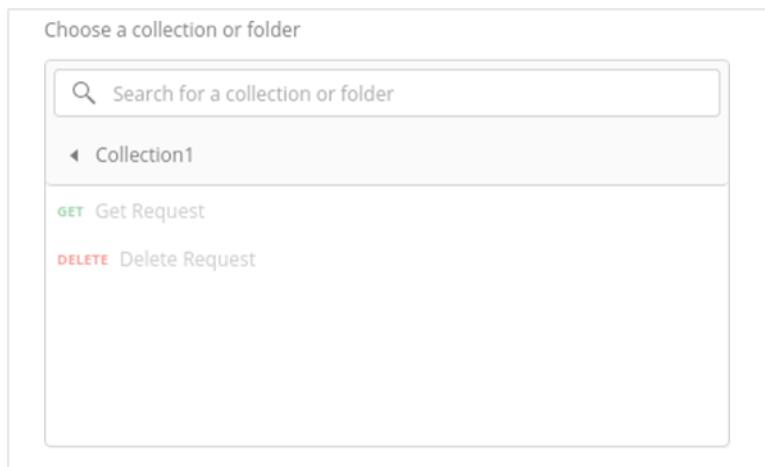
Step 1: Click on the Runner menu present at the top of the Postman application.



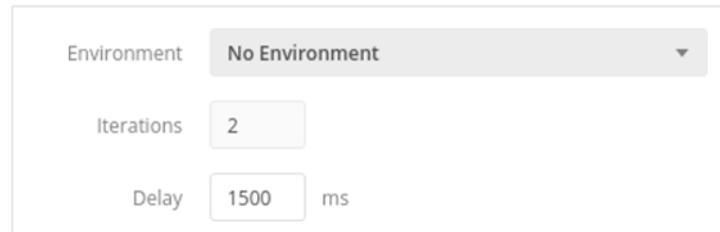
Step 2: The Collection Runner screen shall appear.



Step 3: Select the Collection name from Choose a collection or folder.



Step 4: Select an environment from the Environment dropdown to run the requests in a particular environment. Then, specify the number of times we need to iterate the request. We can also set a delay time in milliseconds for the requests.

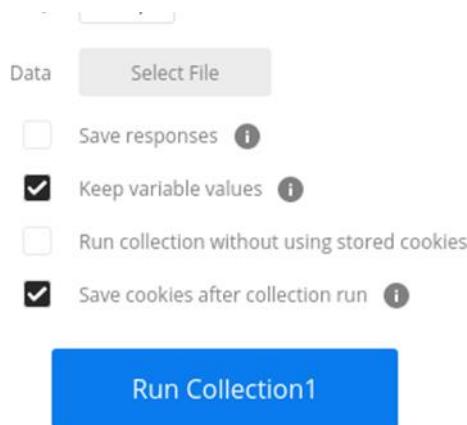


Environment: No Environment

Iterations: 2

Delay: 1500 ms

Step 5: If we have data in a file, then we have to choose the file type from Data. Then, click on the Run Collection1 button.



Data: Select File

Save responses i

Keep variable values i

Run collection without using stored cookies

Save cookies after collection run i

**Run Collection1**

Step 6: The Run Results page shall come up. Depending on the delay time provided, the tests should get executed.

The test results (Pass/Fail) should be displayed for each iteration. The pass status is represented in green and failed ones are represented in red. If there is no test implemented for a particular request, then it shall display the message as - This request does not have any tests.

This is the environment in which the tests are executed and the Collection names are visible at the top of the Collection Runner. For each request, the status code, time taken, payload size, and test verification are also displayed.

Collection Runner

File Edit View Help

Collection Runner Run Results My Workspace Run In Command Line Docs

Collection1 No Environment just now

4 PASSED 0 FAILED

Run Summary Export Results Retry New

Iteration 1

- GET Get Request {{url}}/index.htm / Get Request 200 OK 117 ms 21.248 KB
- Status Code is 200
- DELETE Delete Request http://dummy.restapiexa... / Delete Request 200 OK 640 ms 652 B
- Status Code is 200

Iteration 2

- GET Get Request {{url}}/index.htm / Get Request 200 OK 19 ms 21.248 KB
- Status Code is 200
- DELETE Delete Request http://dummy.restapiexa... / Delete Request 200 OK 706 ms 652 B
- Status Code is 200

## 14. Postman – Assertion

Assertions are used to verify if the actual and expected values have matched after the execution of a test. If they are not matching, the test shall fail and we shall get the reason for failure from the output of the test.

An assertion returns a Boolean value of either true or false. In Postman, we can take the help of JavaScript Chai Assertion Library to add assertions in our tests. It is available in the Postman application automatically.

The Chai – Assertions are easily comprehensible as they are defined in a human readable format. The Assertions in Postman are written within the Tests tab under the address bar.

The documentation for Chai is available in the following link:

<https://www.chaijs.com/>



```
1 pm.test["Status code is 401", function(){
2     pm.response.to.have.status(401)
3 })
```

### Writing Assertions

Let us write an assertion to check if a particular text – Postman is within an array of strings.

```
pm.test["Text is present"], function(){
    pm.expect(['Java', 'Postman']).to.include('Postman')
})
```

### Output

The output is as follows:

Params    Authorization    Headers (6)    Body    Pre-request Script    **Tests** ●    Settings

```

1 pm.test("Text is present", function(){
2     pm.expect(['Java', 'Postman']).to.include('Postman')
3 }
4

```

Body   Cookies   Headers (15)   **Test Results (1/1)**

All   Passed   Skipped   Failed

PASS Text is present

Let us write an Assertion to check if an array is empty.

```

pm.test["Array contains element"], function(){
    pm.expect(['Java', 'Postman']).to.be.an('array').that.is.not.empty
}

```

### Output

The output is given below:

Params    Authorization    Headers (6)    Body    Pre-request Script    **Tests** ●    Settings

```

1 pm.test("Array contains element", function(){
2     pm.expect(['Java', 'Postman']).to.be.an('array').that.is.not.empty
3 }
4

```

Body   Cookies   Headers (15)   **Test Results (1/1)**

All   Passed   Skipped   Failed

PASS Array contains element

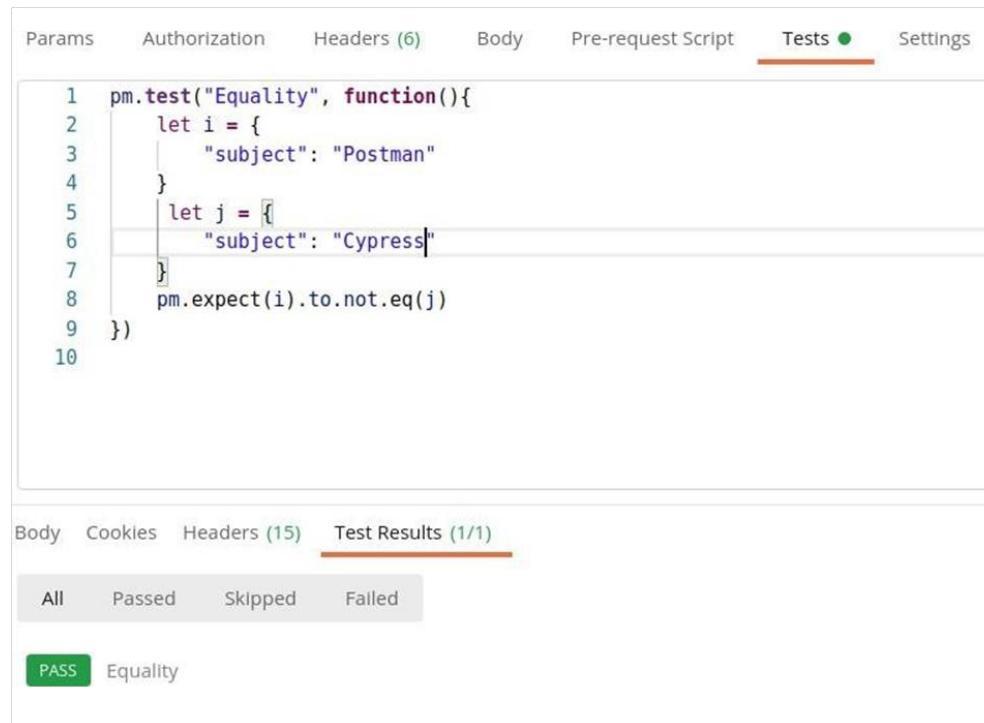
### Assertion for Object Verification

Let us write an Assertion for object verification with `eql`. It is used to compare the properties of the object i and j in the below example.

```
pm.test("Equality", function(){
let i = {
"subject" : "Postman"
};
let j= {
"subject" : "Cypress"
};
pm.expect(i).to.not.eql(j);
```

### Output

The output is mentioned below:



The screenshot shows the Postman interface with the 'Tests' tab selected. The test code is displayed in the 'Tests' section:

```
1 pm.test("Equality", function(){
2   let i = {
3     "subject": "Postman"
4   }
5   let j = [
6     {
7       "subject": "Cypress"
8     }
9   ]
10  pm.expect(i).to.not.eql(j)
```

The 'Test Results' section shows 1/1 result:

- All: PASS
- Passed: Equality
- Skipped: 0
- Failed: 0

The property defined for object i is Postman while the property defined for j is Cypress. Hence, `not.eql` Assertion got passed.

## Assertion Types

In Postman, we can apply assertions on different parts of Response. These are explained below:

### Status Code

```
The assertion for status code is as follows:  
pm.test["Status Code is 401"],  
function(){  
    pm.response.to.have.status(401)  
}
```

The above assertion passes if the Response status code obtained is 401.

```
pm.test["Status is Forbidden"], function(){  
    pm.response.to.have.property('status', 'Forbidden')  
}
```

The above assertion is applied on the Response property – status having the value Forbidden.

### Time taken by Response

The assertion for time taken by response is as follows:

```
pm.test("Response time above 500 milliseconds", function () {  
    pm.expect(pm.response.responseTime).to.be.above(500)  
})
```

The above assertion passes if the Response time is above 500ms.

### Type of Response Format

The assertion for type of response format is as follows:

```
pm.test("Response type is JSON", function(){
    pm.response.to.be.json;
})
```

The above assertion passes if the Response is of JSON type.

### Header of Response

The assertion for header of response is as follows:

```
pm.test("Header Content-Encoding is available", function () {
    pm.response.to.have.header("Content-Encoding")
})
```

The above assertion passes if the Response has a header Content-Encoding.

### Text of Response

```
pm.test("Response Text", function () {
    pm.expect(pm.response.text()).to.include("Tutorialspoint")
})
```

The above assertion passes if the Response text contains the text Tutorialspoint.

## 15. Postman – Mock Server

A mock server is not a real server and it is created to simulate and function as a real server to verify APIs and their responses. These are commonly used if certain responses need to be verified but are not available on the web servers due to security concerns on the actual server.

### Purpose of Mock Server

A Mock Server is created for the reasons listed below:

- A Mock Server is created if the APIs to be used in Production are still in development.
- A Mock Server is used if we want to avoid sending requests on real time data.

### Benefits of Mock Server

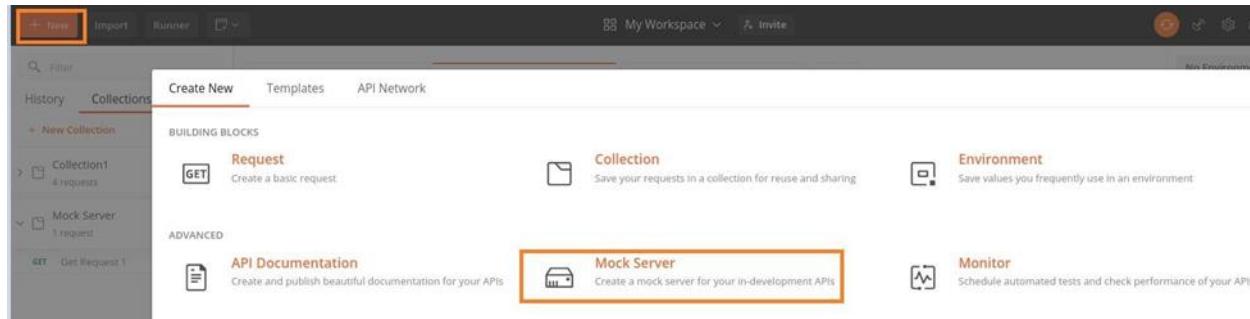
The benefits of Mock Server are listed below:

- Simulation of real API features with examples.
- Mock server can be appended to a Collection.
- Verify APIs with mocking data.
- To identify errors and defects early.
- To identify dependencies in API before it is released for actual usage.
- It is used by engineers to build a prototype for a concept and showcase it to higher management.
- While developing the front end of an application, the developer should have some idea on the response features that shall be obtained from the real server on sending a request. A Mock Server can be really helpful at this time.

## Mock Server Creation

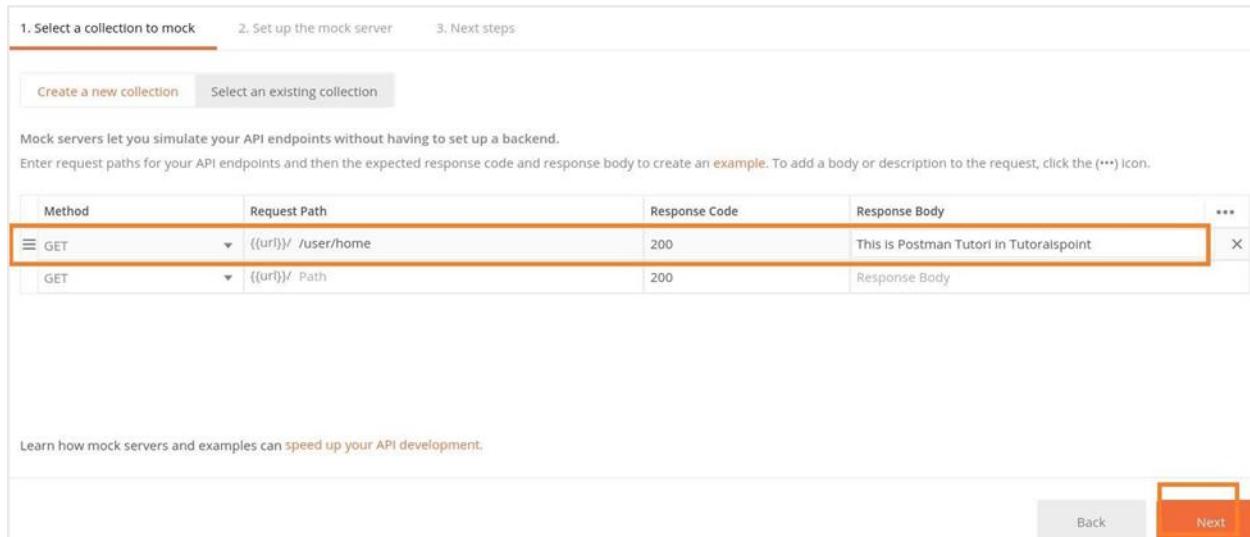
Follow the steps given below for creation of mock server in Postman:

Step 1: Click on the New icon from the Postman application. Then, click on Mock Server.



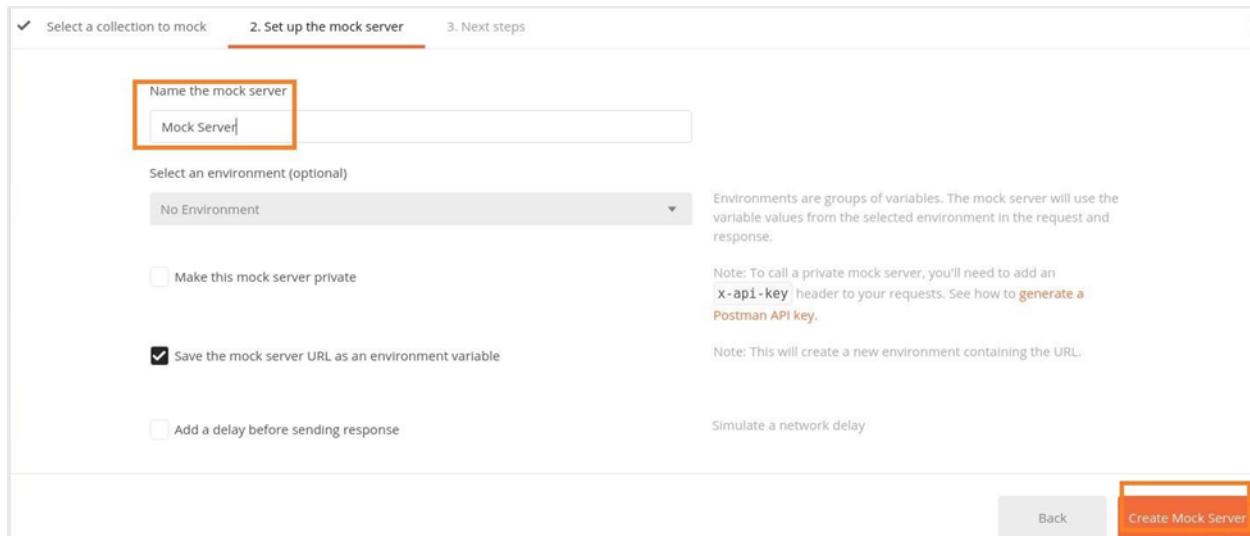
Step 2: Select GET from the Method dropdown, enter a Request Path as /user/home,

Response Code as 200, and a Response Body. Then, click on Next.



Method	Request Path	Response Code	Response Body
GET	{{url}}/ /user/home	200	This is Postman Tutorial in TutorialsPoint
GET	{{url}}/ /Path	200	Response Body

Step 3: Enter a Mock Server name and click on the Create Mock Server button.



✓ Select a collection to mock    2. Set up the mock server    3. Next steps

Name the mock server  
Mock Server

Select an environment (optional)  
No Environment

Environments are groups of variables. The mock server will use the variable values from the selected environment in the request and response.

Make this mock server private

Note: To call a private mock server, you'll need to add an `x-api-key` header to your requests. See how to [generate a Postman API key](#).

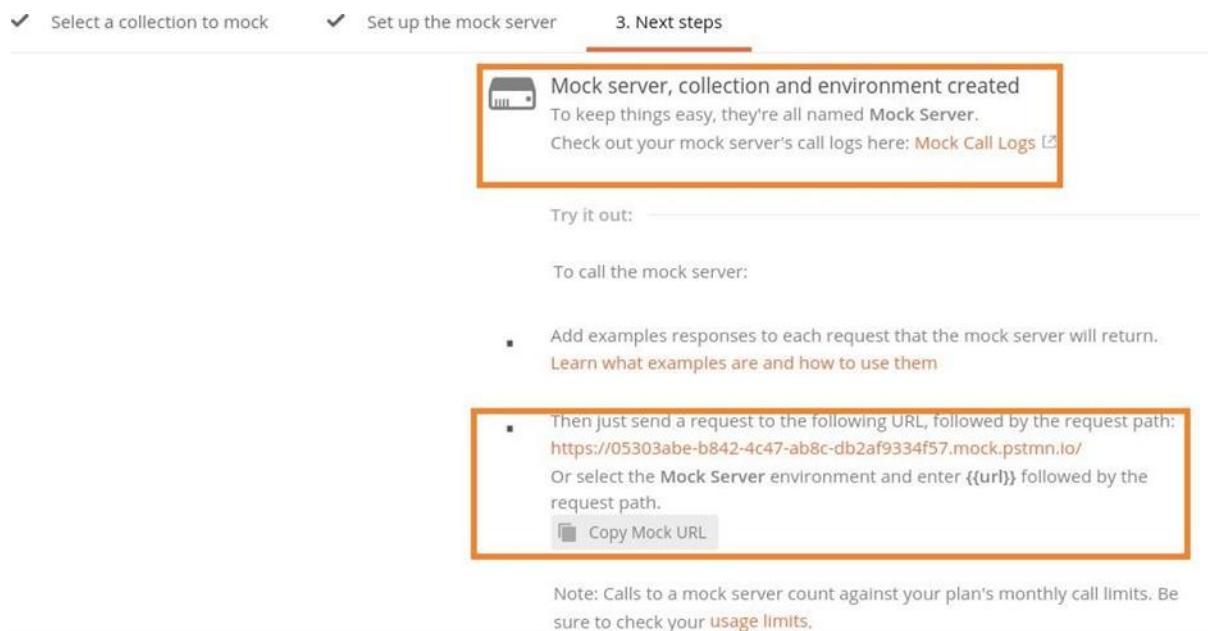
Save the mock server URL as an environment variable

Note: This will create a new environment containing the URL.

Add a delay before sending response    Simulate a network delay

Back    **Create Mock Server**

Step 4: The Mock Server gets created along with the Mock URL. The Copy Mock URL button is used to copy the Mock link. Click on the Close button to proceed.



✓ Select a collection to mock    ✓ Set up the mock server    3. Next steps

 Mock server, collection and environment created  
To keep things easy, they're all named Mock Server.  
Check out your mock server's call logs here: [Mock Call Logs \[?\]](#)

Try it out:

To call the mock server:

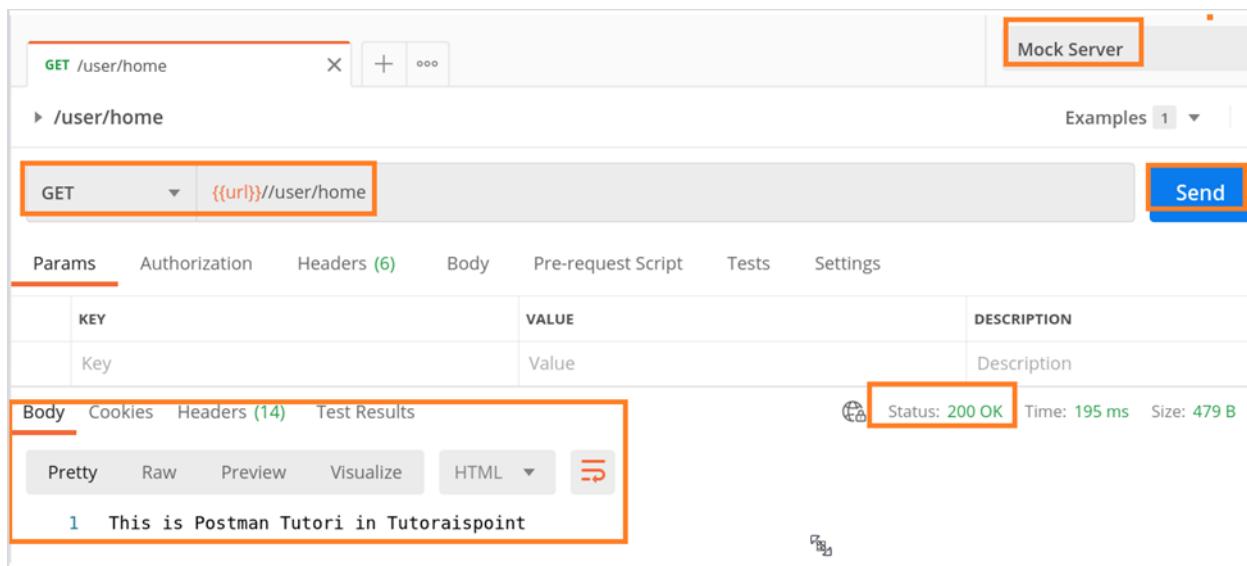
- Add examples responses to each request that the mock server will return.  
[Learn what examples are and how to use them](#)

Then just send a request to the following URL, followed by the request path:  
<https://05303abe-b842-4c47-ab8c-db2af9334f57.mock.pstmn.io/>  
Or select the Mock Server environment and enter `{{url}}` followed by the request path.  
**Copy Mock URL**

Note: Calls to a mock server count against your plan's monthly call limits. Be sure to check your usage limits.

Step 5: Select Mock Server as the Environment from the No Environment dropdown and click on Send. The Response code obtained is 200 OK which means that the request is successful.

Also the Response Body shows the message – This is Postman Tutorial for Tutorialspoint which is the same as we passed as a Response Body in the Step 2.



The screenshot shows the Postman interface with the following details:

- Request URL: GET /user/home
- Method: GET
- Body: "1 This is Postman Tutori in Tutoraispoint"
- Status: 200 OK
- Time: 195 ms
- Size: 479 B

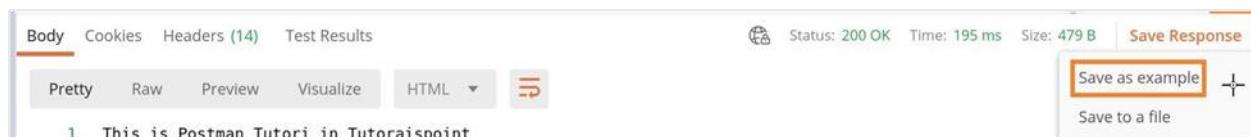
Step 6: The value of URL can be obtained by clicking on the eye icon at the right upper corner of the screen.



VARIABLE	INITIAL VALUE	CURRENT VALUE
url	https://05303abe-b842-4c47-ab8c-db2af9334f57.mock.pstmn.io	https://05303abe-b842-4c47-ab8c-db2af9334f57.mock.pstmn.io

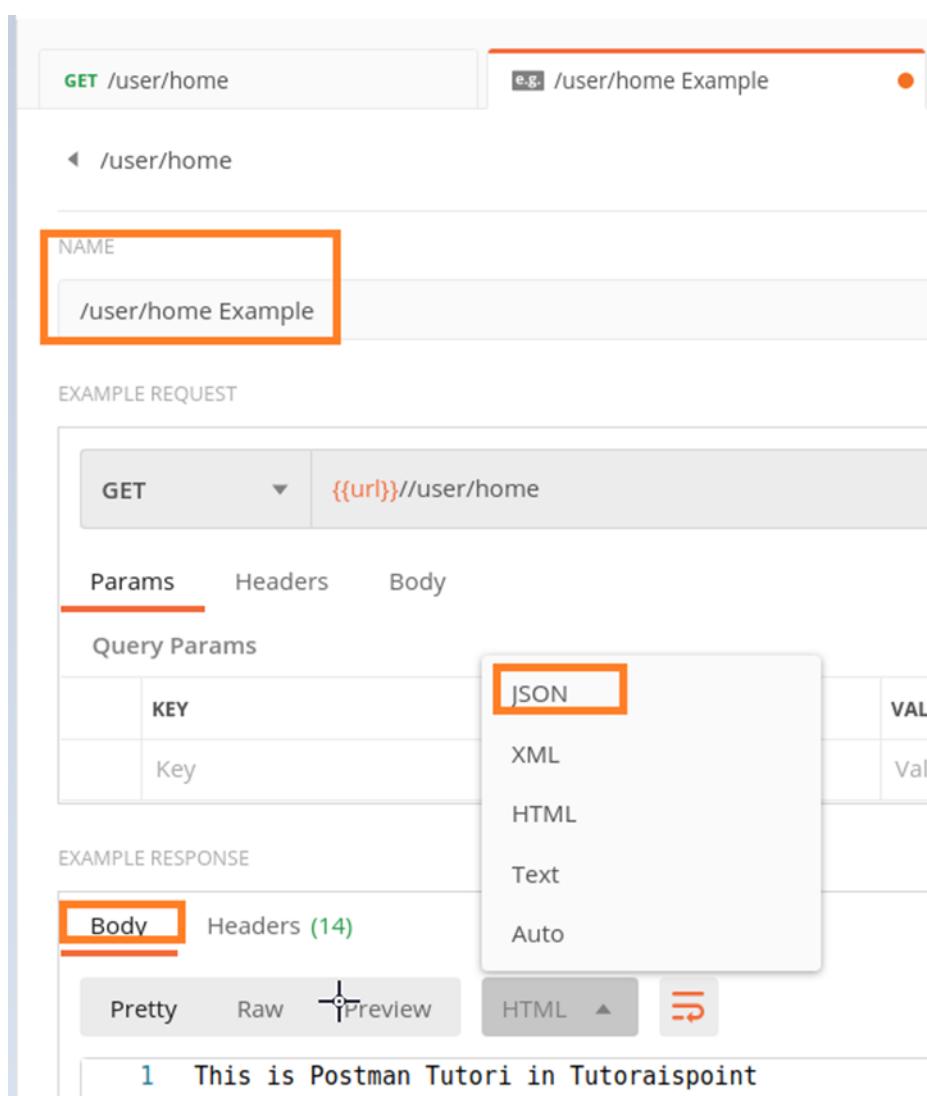
So the complete request Mock URL should be: https://05303abe-b842-4c47-ab8c-db2af9334f57.mock.pstmn.io/user/home (represented by {{url}}/user/home in the address bar in Step 5). We have appended /user/home at the end of the url value since it is the Request Path we have set for the Mock Server in Step2.

Step 7: We have seen that the Response Body is in text format. We can get the response in JSON format as well. To achieve this select the option Save as example from the Save Response dropdown.



The screenshot shows the Postman interface. In the top right corner, there is a 'Save Response' dropdown menu. The 'Save as example' option is highlighted with a red box. Other options in the dropdown include 'Save to a file' and a '+' sign for creating a new example.

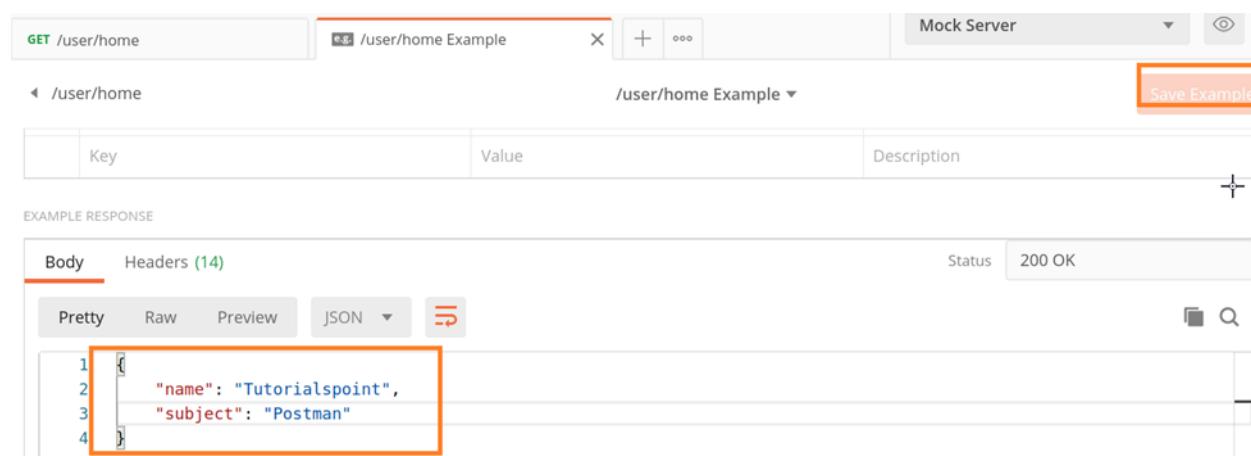
Step 8: Provide an Example name and select JSON from the Response Body section.



The screenshot shows the Postman interface for a GET request to '/user/home'. The 'NAME' field under 'EXAMPLE REQUEST' is highlighted with a red box and contains the value '/user/home Example'. In the 'Body' tab of the 'EXAMPLE RESPONSE' section, the 'JSON' option is highlighted with a red box. Below it, other options like 'XML', 'HTML', 'Text', and 'Auto' are listed. The response body preview shows the text '1 This is Postman Tutori in Tutoraispoint'.

Step 9: Add the below Response Body in JSON format. Then click on Save Example.

```
{
  "name": "Tutorialspoint",
  "subject": "Postman"
}
```



The screenshot shows the Postman interface with the following details:

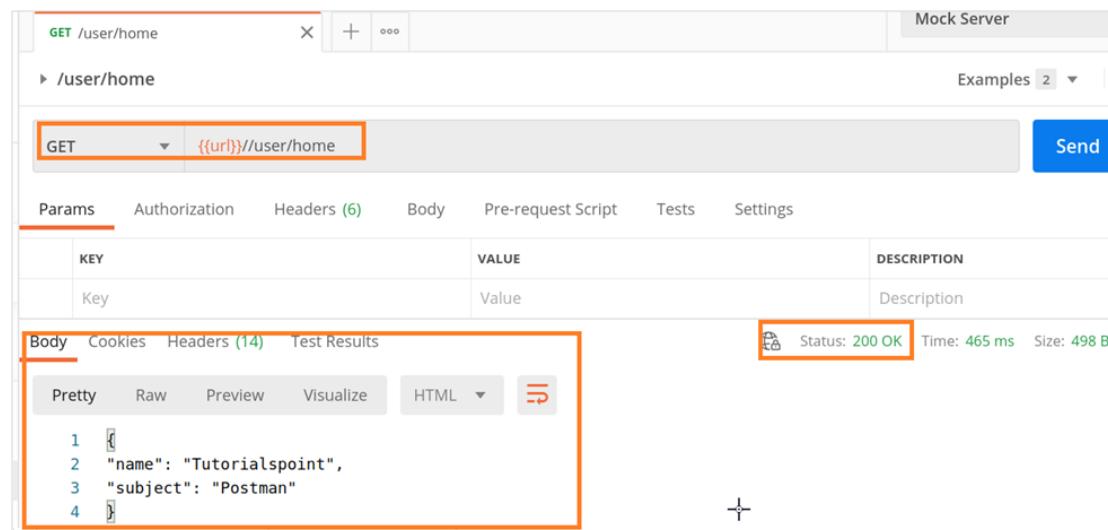
- Request URL: GET /user/home
- Example Name: /user/home Example
- Mock Server dropdown
- Save Example button (highlighted)
- Table headers: Key, Value, Description
- Body tab selected
- Status: 200 OK
- Pretty, Raw, Preview, JSON buttons
- Body content: 

```
1 {
  2   "name": "Tutorialspoint",
  3   "subject": "Postman"
  4 }
```

 (The entire JSON block is highlighted with an orange box)

Step 10: Finally, send the GET request on the same endpoint, and we shall receive the same Response Body as we have passed in the Example request.

The below image shows Response is in HTML format:



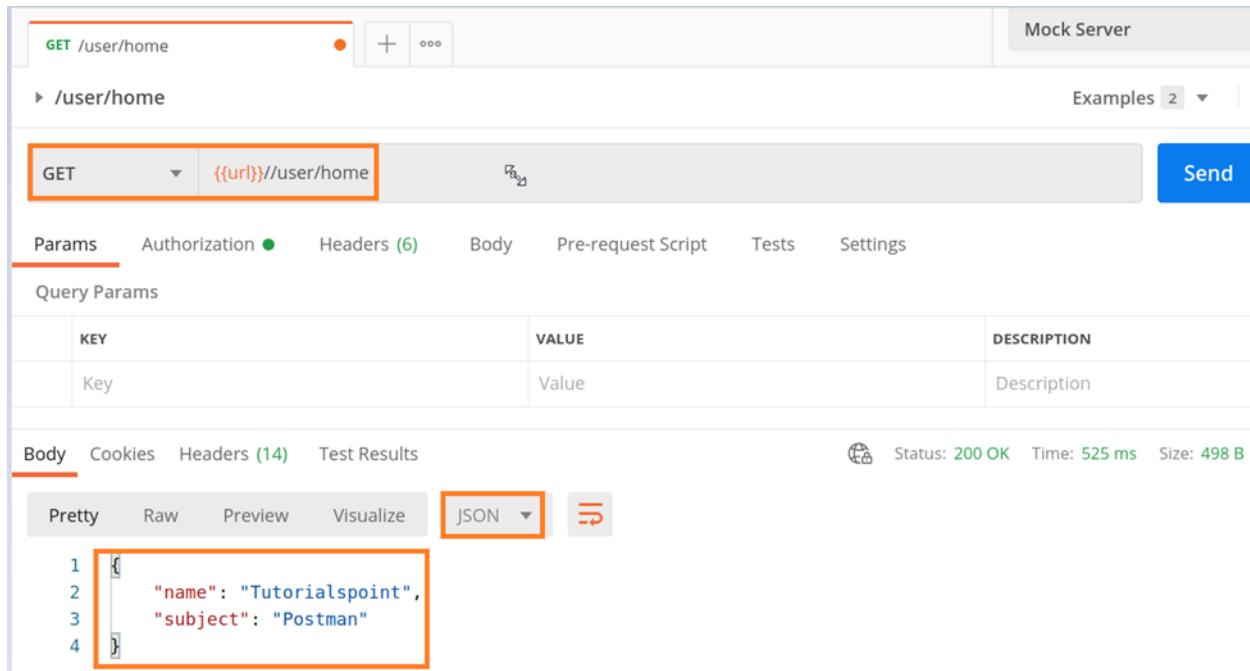
The screenshot shows the Postman interface with the following details:

- Request URL: GET /user/home
- Endpoint: {{url}}/user/home
- Send button
- Params tab selected
- Body tab selected (highlighted)
- Status: 200 OK
- Body content: 

```
1 {
  2   "name": "Tutorialspoint",
  3   "subject": "Postman"
  4 }
```

 (The entire JSON block is highlighted with an orange box)

The below image shows Response is in JSON format:



The screenshot shows the Postman interface with a collection named 'Mock Server'. A GET request to '/user/home' is selected. The response body is displayed in JSON format, showing a single object with 'name' and 'subject' fields.

```

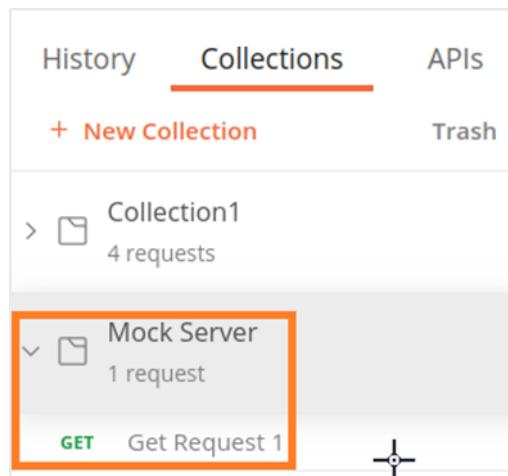
1 [ { "name": "Tutorialspoint", "subject": "Postman" } ]
  
```

### Example Request

Follow the steps given below for Mock Server Creation by example request:

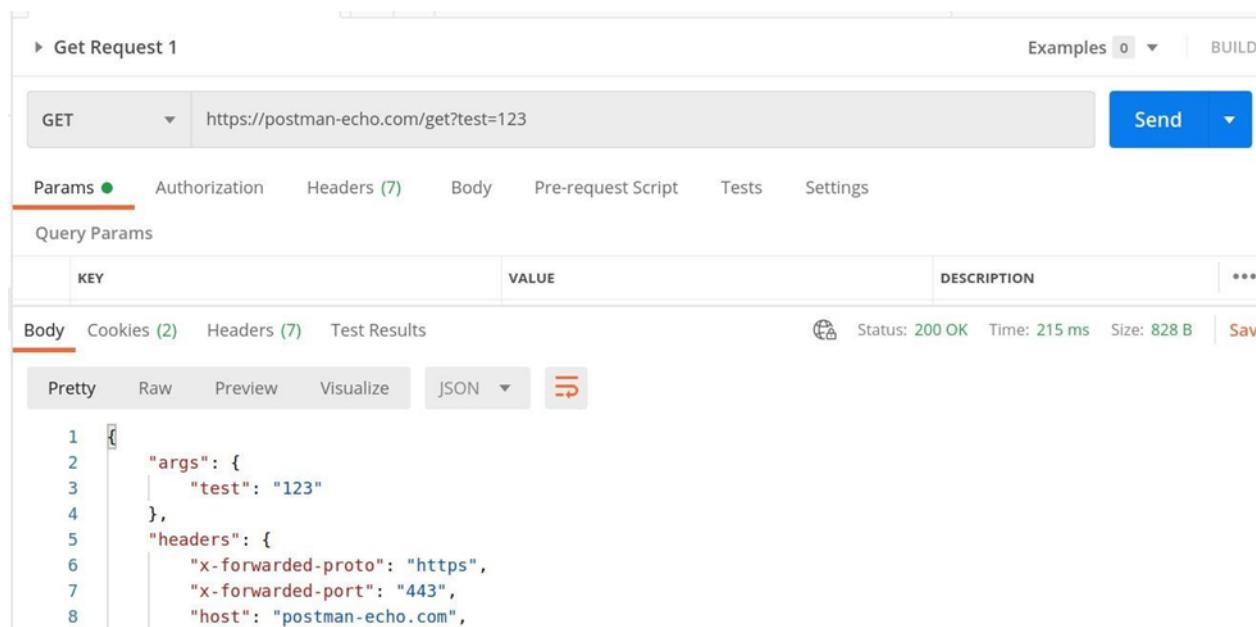
Step 1: Create a Collection and add a request to it.

The details on how to create a Collection is discussed in detail in the Chapter – Postman Create Collections.



The screenshot shows the Postman collections view. A new collection named 'Mock Server' is created and contains one request named 'Get Request 1'.

Step 2: Add the endpoint <https://postman-echo.com/get?test=123> and send a GET request.



The screenshot shows the Postman interface with a GET request to <https://postman-echo.com/get?test=123>. The response status is 200 OK, and the response body is:

```

1  {
2   "args": {
3     "test": "123"
4   },
5   "headers": {
6     "x-forwarded-proto": "https",
7     "x-forwarded-port": "443",
8     "host": "postman-echo.com",

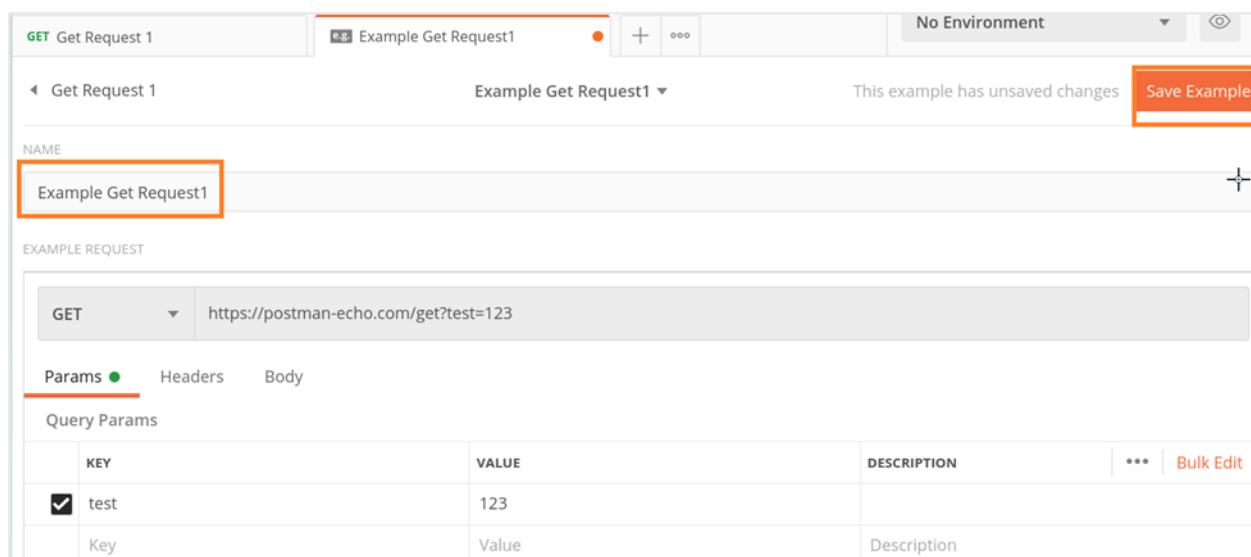
```

Step 3: From Response Body, select the option Save as Example from the Save Response dropdown.



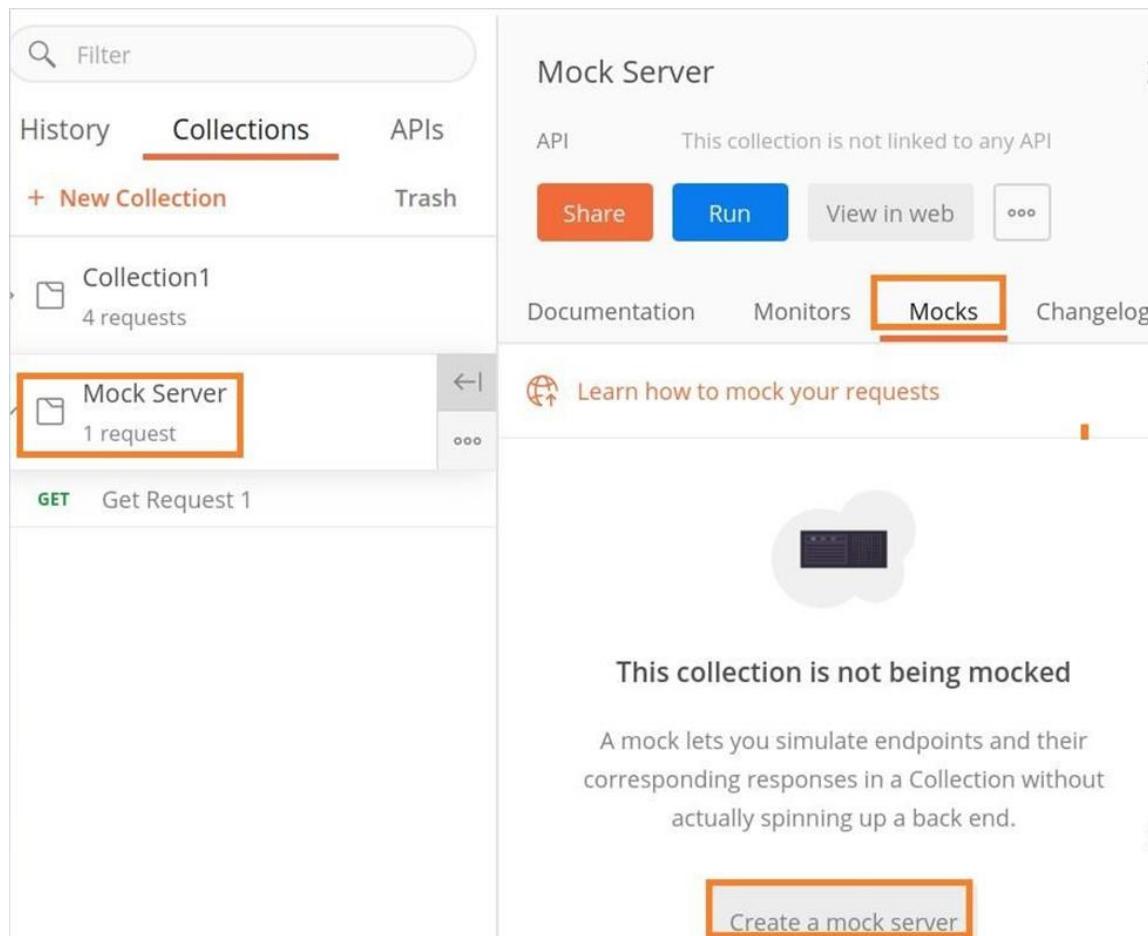
The screenshot shows the Postman interface with the 'Save Response' dropdown open, and 'Save as example' is highlighted.

Step 4: Give an Example name and click on the Save Example button.



The screenshot shows the Postman Collection Editor interface. At the top, there are tabs for 'GET Get Request 1' and 'Example Get Request1'. A red box highlights the collection name 'Example Get Request1'. To the right, there are buttons for 'Save Example' and 'No Environment'. Below the tabs, there's a 'NAME' field containing 'Example Get Request1' with a red box around it. Under 'EXAMPLE REQUEST', there's a dropdown set to 'GET' and a URL 'https://postman-echo.com/get?test=123'. Below this, under 'Params', there's a table with one row: 'test' (checked) with value '123'. Other tabs for 'Headers' and 'Body' are visible.

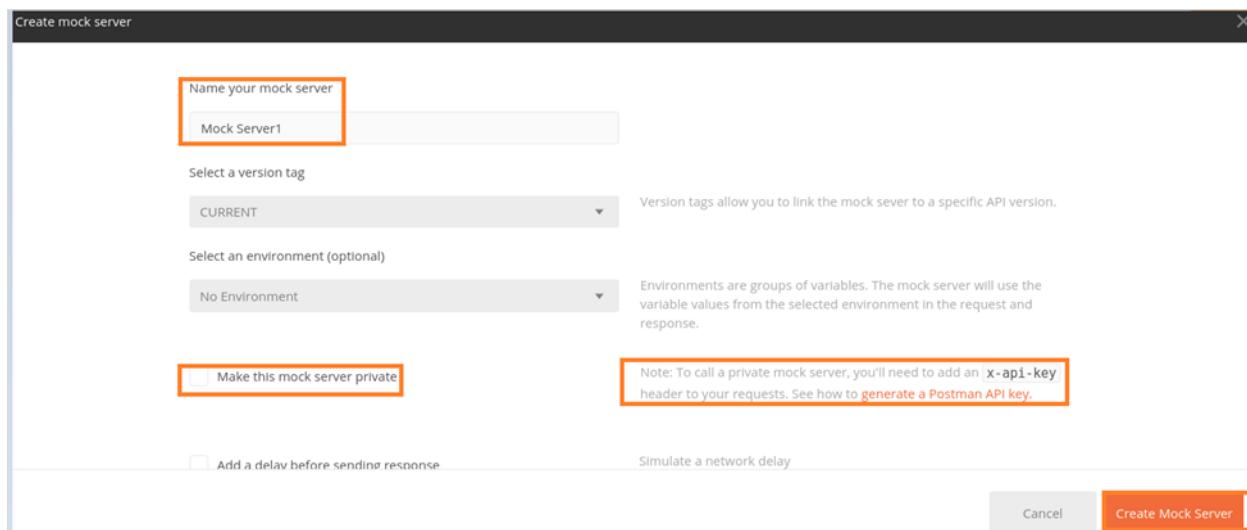
Step 5: Click on the Collection name Mock Server (that we have created) and click on the Mock tab. Then, click on Create a mock server.



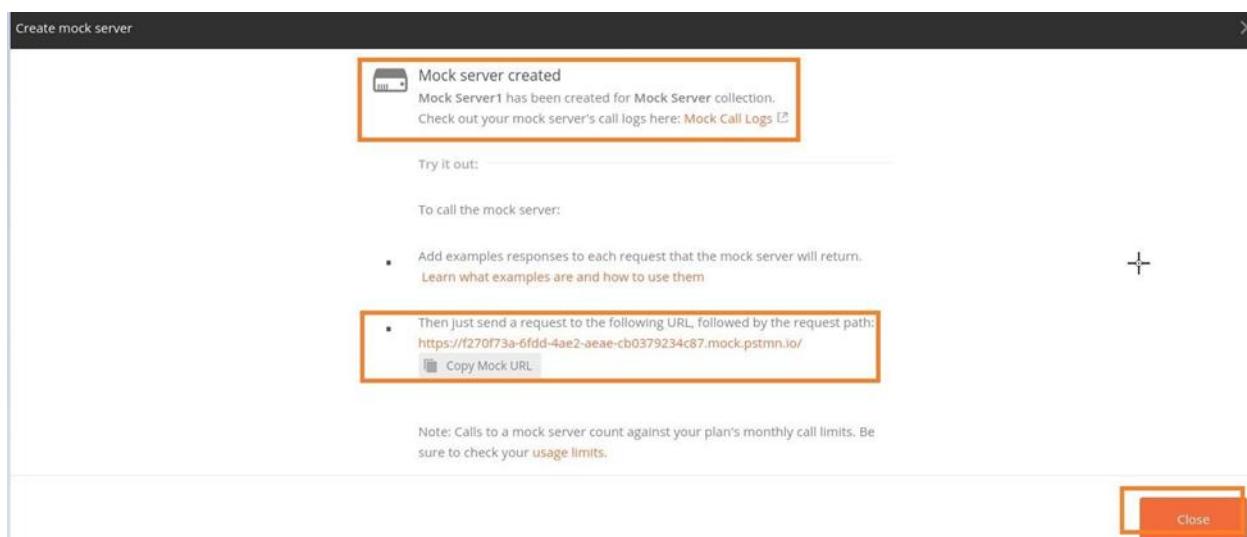
The screenshot shows the Postman collections overview. On the left, there's a sidebar with 'History', 'Collections' (selected), 'APIs', and 'Trash'. A red box highlights the 'Mock Server' collection. Below it is 'Collection1' with '4 requests'. At the bottom of the sidebar, there's a 'GET Get Request 1' entry. On the right, the 'Mock Server' collection details are shown: 'Mock Server' (Collection name), 'This collection is not linked to any API', 'Share', 'Run', 'View in web', 'Mocks' (tab selected with a red box), 'Documentation', 'Monitors', 'Changelog', and a link 'Learn how to mock your requests'. Below this, a message says 'This collection is not being mocked' with a small icon. At the bottom, there's a button 'Create a mock server' with a red box around it.

Step 6: The Create mock server pop-up comes up. Provide a name to the Mock Server and then click on the Create Mock Server button.

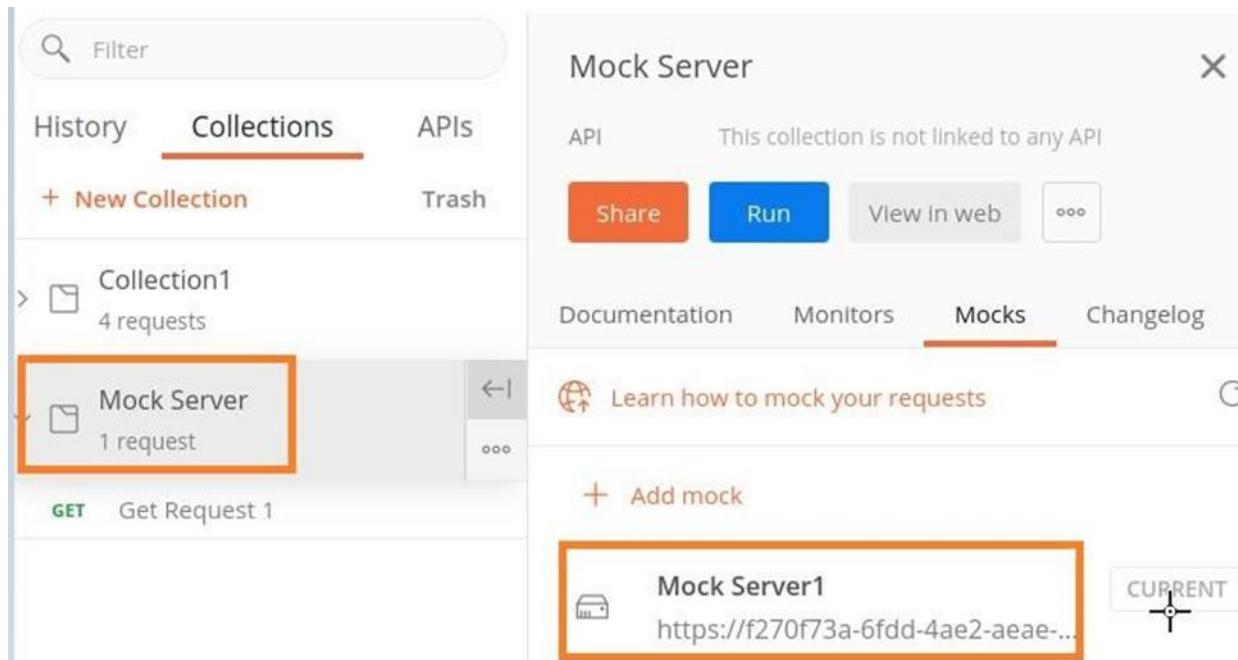
Please note: We can make a Mock Server private or public. To make a Mock Server private, we have to check the checkbox Make this mock server private. Then, we need to utilise the Postman API key.



Step 7: The message – Mock server created shall come up. Also, we shall get the Mock URL. We can copy it with the Copy Mock URL button. Then, click on Close.



Step 8: The Mock Server which we have created gets reflected under the Mock tab in the Collections sidebar. Click on the same.

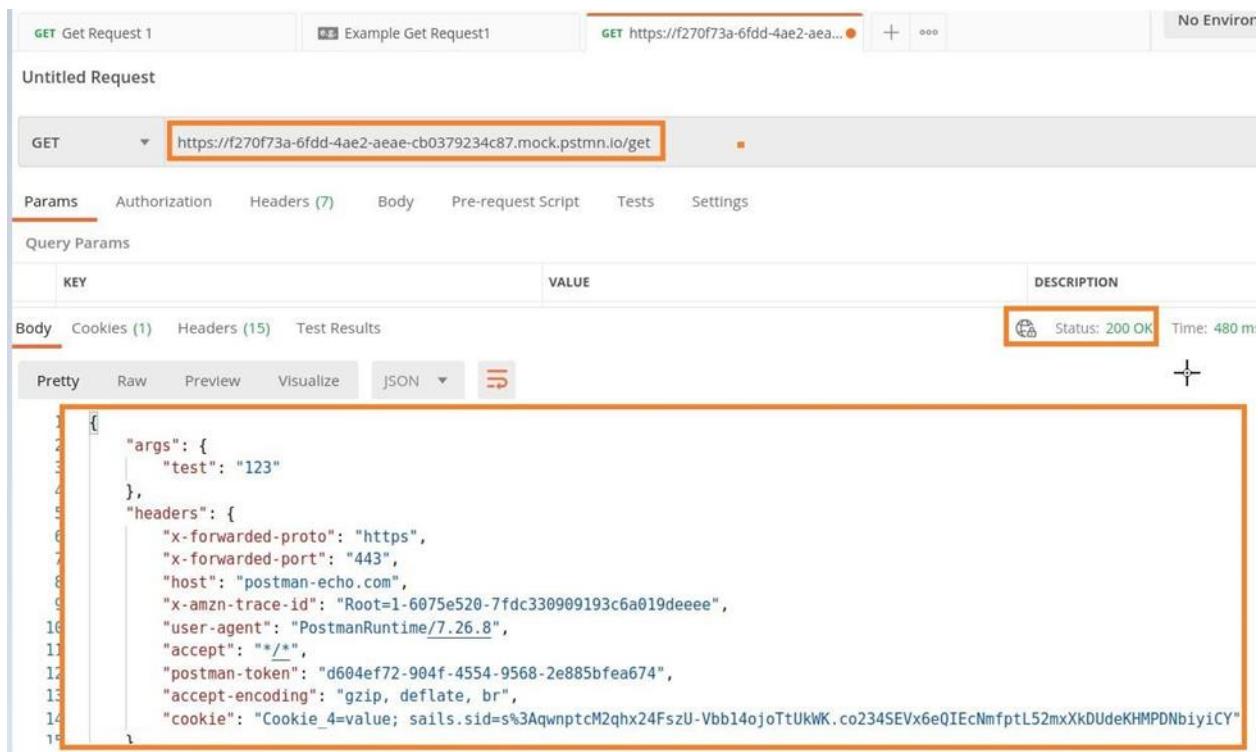


The screenshot shows the Postman interface. On the left, there's a sidebar with tabs: History, Collections (which is selected and highlighted in orange), APIs, and Trash. Below these tabs is a '+ New Collection' button. The main area displays a list of collections: 'Collection1' (with 4 requests) and 'Mock Server' (with 1 request). The 'Mock Server' collection is highlighted with an orange border. Below the collections, there's a 'GET' request labeled 'Get Request 1'. To the right of the collections list is a detailed view for the 'Mock Server' collection. This view includes a header 'Mock Server' with an 'X' icon to close it. Below the header, there's an 'API' section stating 'This collection is not linked to any API' and buttons for 'Share', 'Run', 'View in web', and 'More'. Underneath this are tabs for 'Documentation', 'Monitors', 'Mocks' (which is selected and highlighted in orange), and 'Changelog'. A 'Learn how to mock your requests' link is also present. At the bottom of the detailed view, there's a '+ Add mock' button and a list of existing mocks, with 'Mock Server1' and its URL 'https://f270f73a-6fdd-4ae2-aeae-cb0379234c87.mock.pstmn.io' listed. This URL is also highlighted with an orange border.

Step 9: We shall add a new request and paste the URL we have copied in Step 7. To send a GET request, we shall append the value - /get at the end of the pasted URL.

For example, here, the Mock URL generated is: <https://f270f73a-6fdd-4ae2-aeae-cb0379234c87.mock.pstmn.io>.

Now to send a GET request, the endpoint should be: <https://f270f73a-6fdd-4ae2-aeae-cb0379234c87.mock.pstmn.io/get>.

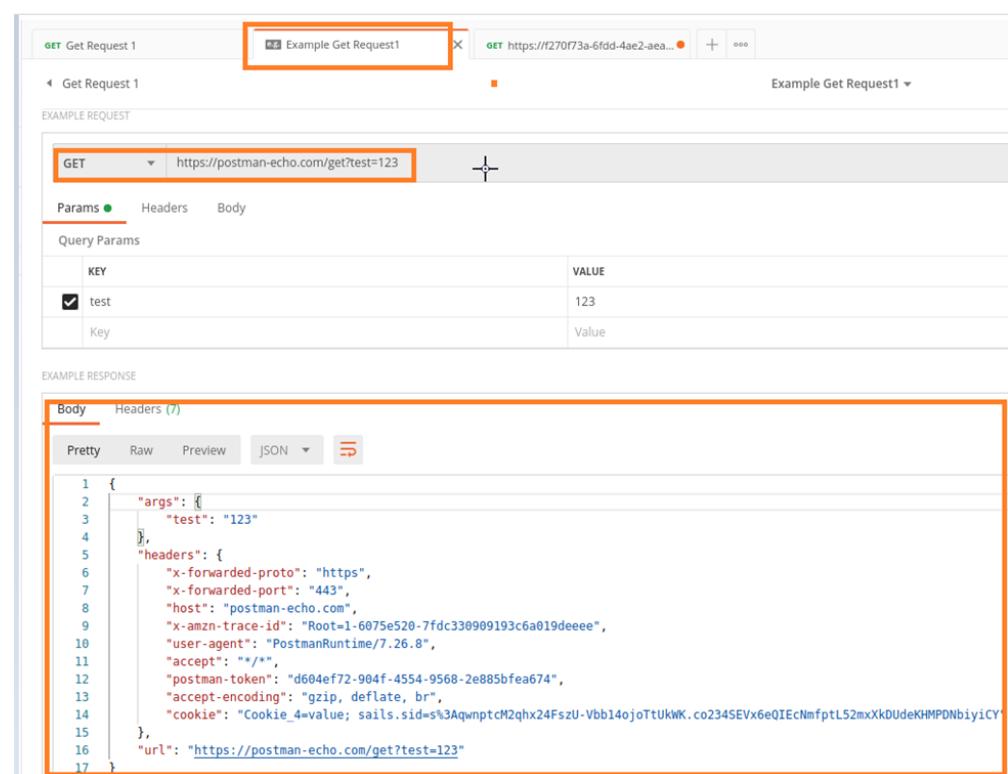


The Response Body received by mocking the server is the same as the Example request. Response obtained in the Example request is as follows:

```

1  [
2    "args": {
3      "test": "123"
4    },
5    "headers": {
6      "x-forwarded-proto": "https",
7      "x-forwarded-port": "443",
8      "host": "postman-echo.com",
9      "x-amzn-trace-id": "Root=1-6075e520-7fdc330909193c6a019deeee",
10     "user-agent": "PostmanRuntime/7.26.8",
11     "accept": "*/*",
12     "postman-token": "d604ef72-904f-4554-9568-2e885bfea674",
13     "accept-encoding": "gzip, deflate, br",
14     "cookie": "Cookie_4=value; sails.sid=s%3AqwnptcM2qhx24FsU-Vbb14ojoTtUkWK.co234SEVx6eQIEcNmfpL52mxXkDUdeKHMPDNbiyiCY"
15   }
16 ]

```



The Response Body received by mocking the server is the same as the Example request. Response obtained in the Example request is as follows:

```

1  [
2    "args": {
3      "test": "123"
4    },
5    "headers": {
6      "x-forwarded-proto": "https",
7      "x-forwarded-port": "443",
8      "host": "postman-echo.com",
9      "x-amzn-trace-id": "Root=1-6075e520-7fdc330909193c6a019deeee",
10     "user-agent": "PostmanRuntime/7.26.8",
11     "accept": "*/*",
12     "postman-token": "d604ef72-904f-4554-9568-2e885bfea674",
13     "accept-encoding": "gzip, deflate, br",
14     "cookie": "Cookie_4=value; sails.sid=s%3AqwnptcM2qhx24FsU-Vbb14ojoTtUkWK.co234SEVx6eQIEcNmfpL52mxXkDUdeKHMPDNbiyiCY"
15   },
16   "url": "https://postman-echo.com/get?test=123"
17 ]

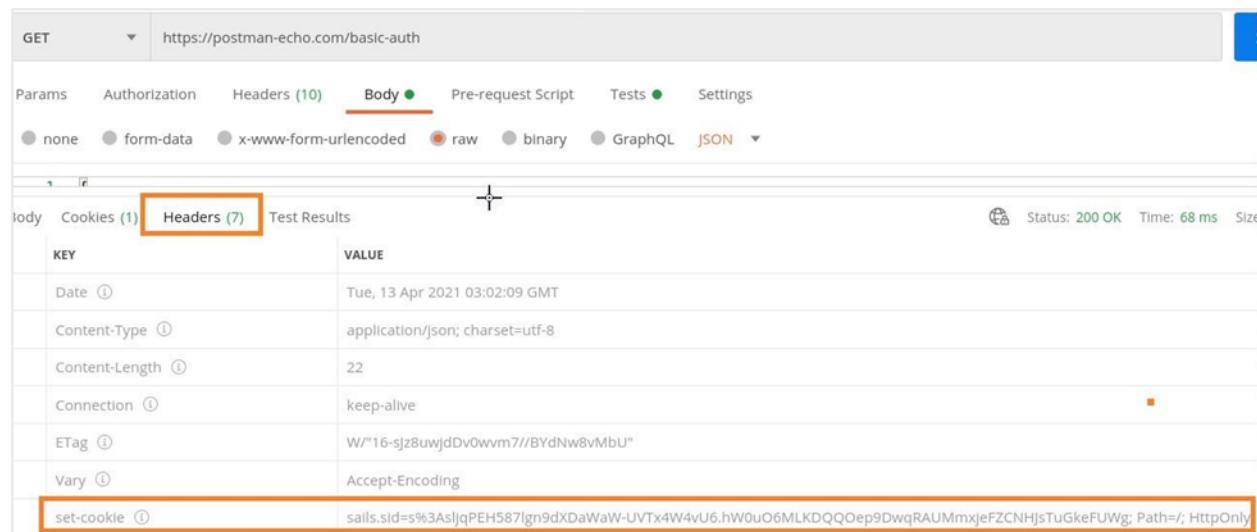
```

## 16. Postman – Cookies

The cookies are information sent by the server and stored in the browser. As soon as a request is sent, the cookies are returned by the server. In Postman, the cookies are mentioned under the Headers and Cookies tab in the Response.

Let us apply a GET request on an endpoint and find the cookies.

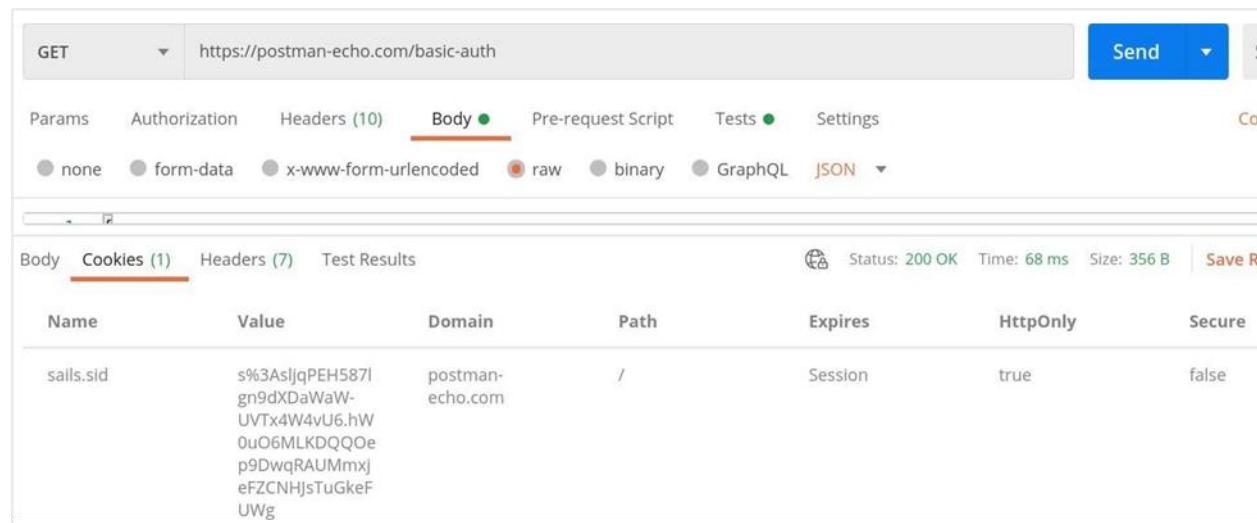
In the Headers tab, the cookie sent by the server is set with the key – set-cookie.



A screenshot of the Postman interface showing a successful GET request to <https://postman-echo.com/basic-auth>. The 'Headers' tab is selected, displaying various response headers. The 'set-cookie' header is highlighted with an orange border, containing the value: `sails.sid=s%3AsljqPEH587lgn9dXDaWaW-UVTx4W4vU6.hW0uO6MLKDQQeop9DwqRAUMmxjeFZCNHjsTuGkeFUWg; Path=/; HttpOnly`.

KEY	VALUE
Date	Tue, 13 Apr 2021 03:02:09 GMT
Content-Type	application/json; charset=utf-8
Content-Length	22
Connection	keep-alive
ETag	W/"16-sjz8uwjdDv0wvm7//BYdNw8vMbU"
Vary	Accept-Encoding
set-cookie	sails.sid=s%3AsljqPEH587lgn9dXDaWaW-UVTx4W4vU6.hW0uO6MLKDQQeop9DwqRAUMmxjeFZCNHjsTuGkeFUWg; Path=/; HttpOnly

In the Cookies tab, the same cookie details will also get displayed.

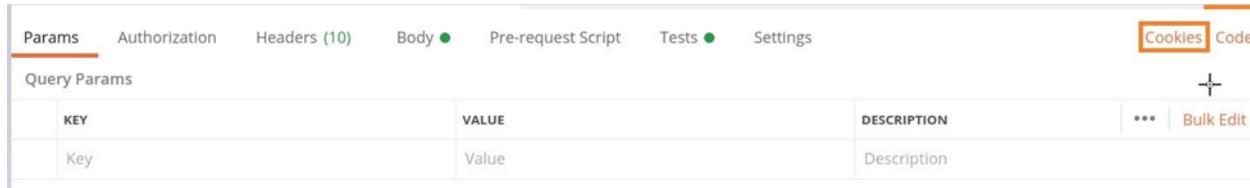


A screenshot of the Postman interface showing the same GET request to <https://postman-echo.com/basic-auth>. The 'Cookies' tab is selected, displaying the cookie details. The cookie 'sails.sid' is listed with its value, domain, path, expiration, and other properties.

Name	Value	Domain	Path	Expires	HttpOnly	Secure
sails.sid	s%3AsljqPEH587lgn9dXDaWaW-UVTx4W4vU6.hW0uO6MLKDQQeop9DwqRAUMmxjeFZCNHjsTuGkeFUWg	postman-echo.com	/	Session	true	false

## Cookies Management

In Postman, we can manage cookies by addition, deletion, and modification of cookies. Under the Params tab, we have the Cookies link to perform operations on cookies.



The screenshot shows the Postman interface with the 'Params' tab selected. At the top right of the tab, there is a 'Cookies' button which is highlighted with a red box. Below the tabs, there is a table titled 'Query Params' with columns for KEY, VALUE, and DESCRIPTION. A '+' icon is located at the bottom right of the table area.

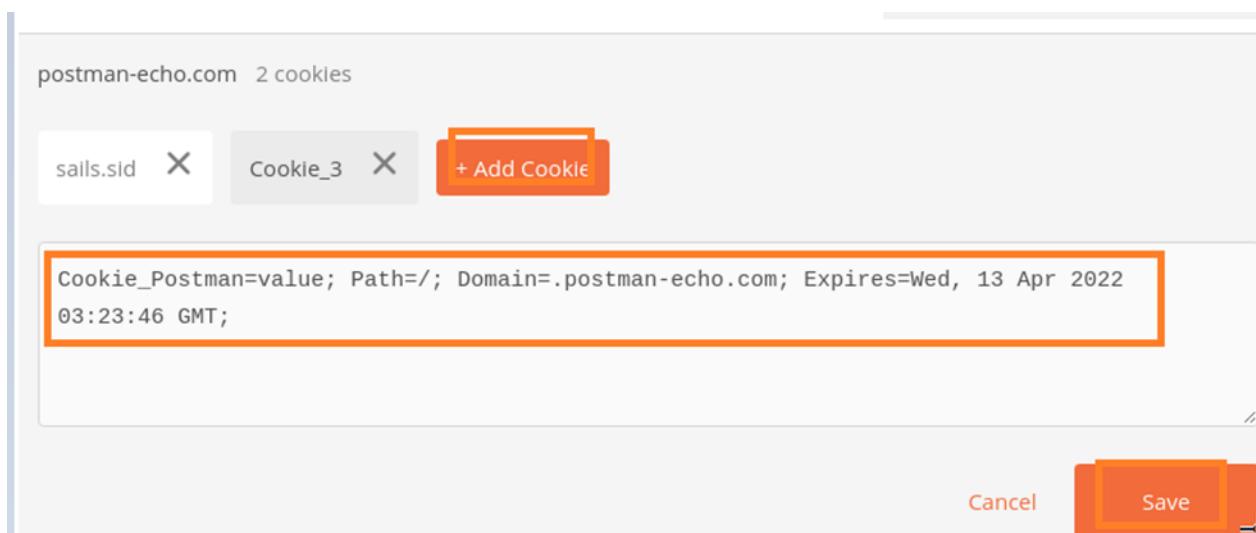
Click on the Cookies link. MANAGE COOKIES pop-up shall open where all the available cookies are present with the option to add and delete a cookie.



## Cookies Addition

Follow the steps given below for adding cookie in Postman:

Step 1: Click on the Add Cookie button. A text box shall open up with pre-existing values inside it. We can modify its values and then click on Save.



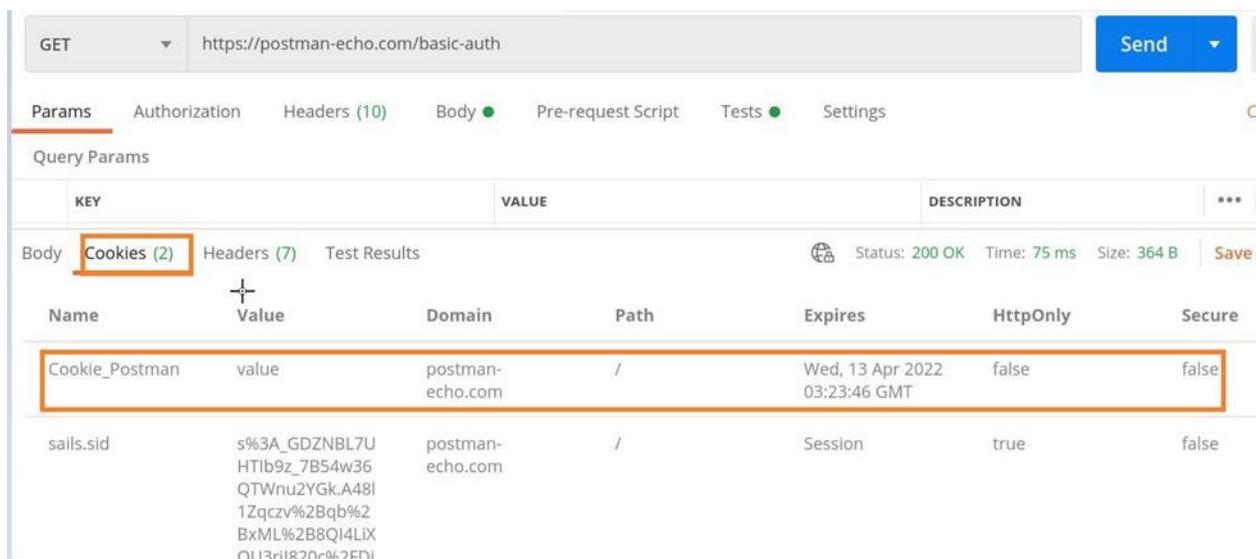
The screenshot shows the Postman interface with the 'Cookies' tab selected. It displays two existing cookies: 'sails.sid' and 'Cookie\_3'. A new cookie, 'Cookie\_Postman', has been added and is highlighted with an orange border. The cookie details are as follows:

```
Cookie_Postman=value; Path=/; Domain=.postman-echo.com; Expires=Wed, 13 Apr 2022 03:23:46 GMT;
```

At the bottom right, there are 'Cancel' and 'Save' buttons, with 'Save' also being highlighted with an orange border.

Step 2: Send the request again to the server.

The Response code obtained is 200 OK. Also, the Cookies tab in the Response now shows the newly added cookie – Cookie\_Postman.



The screenshot shows the Postman interface with the 'Response' tab selected. A GET request is made to 'https://postman-echo.com/basic-auth'. The 'Cookies' section of the response is highlighted with an orange border. It contains the following data:

KEY	VALUE	DESCRIPTION	...			
Cookie_Postman	value	postman-echo.com	/	Wed, 13 Apr 2022 03:23:46 GMT	false	false

Other sections like 'Body', 'Headers', and 'Test Results' are also visible in the response table.

### Access Cookies via Program

Cookies can be handled programmatically without using the GUI in Postman. To work with cookies, we have to first generate a Cookie jar. It is an object which has all the cookies and methods to access them.

## Cookie Jar Creation

The syntax for Cookie Jar Creation is as follows:

```
const c = pm.cookies.jar();
```

## Cookie Creation

We can create a cookie with the .set() function. It accepts URL, name of cookie, value of cookie as parameters.

The syntax for cookie creation is as follows:

```
const c = pm.cookies.jar();
c.set(URL, name of cookie, value of cookie, callback(error, cookie));
```

## Get Cookie

We can get a cookie with the .get() function. It accepts the URL, name of cookie as parameters. It yields the cookie value.

The syntax for getting a cookie is as follows:

```
const c = pm.cookies.jar();
c.set(URL, name of cookie, value of cookie, callback(error, cookie));
c.get(URL, name of cookie, callback(error, cookie));
```

## Get All Cookies

We can get all cookies of a specific URL within a Cookie jar with the .getAll() function. It accepts a URL as a parameter. It yields all the cookie values for that URL.

The syntax for getting all cookies is as follows:

```
const c = pm.cookies.jar();
c.set(URL, name of first cookie, value of first cookie, callback(error,
cookie));
c.set(URL, name of second cookie, value of second cookie, callback(error,
cookie));
c.getAll(URL, callback(error, cookie));
```

## Delete Cookie

We can delete a cookie with the .unset() function. It accepts the URL, name of cookie to be deleted as parameters.

The syntax for deleting cookie is as follows:

```
const c = pm.cookies.jar();
c.set(URL, name of cookie, value of cookie, callback(error, cookie));
c.unset(URL, name of cookie, callback(error, cookie));
```

## Delete All Cookies

We can delete all cookies of a specific URL with the .clear() function. It accepts a URL as a parameter. It removes all the cookie values for that URL.

The syntax for deleting all cookies is as follows:

```
const c = pm.cookies.jar();
c.set(URL, name of first cookie, value of first cookie, callback(error,
cookie));
c.set(URL, name of second cookie, value of second cookie, callback(error,
cookie));
c.clear(URL, callback(error, cookie));
```

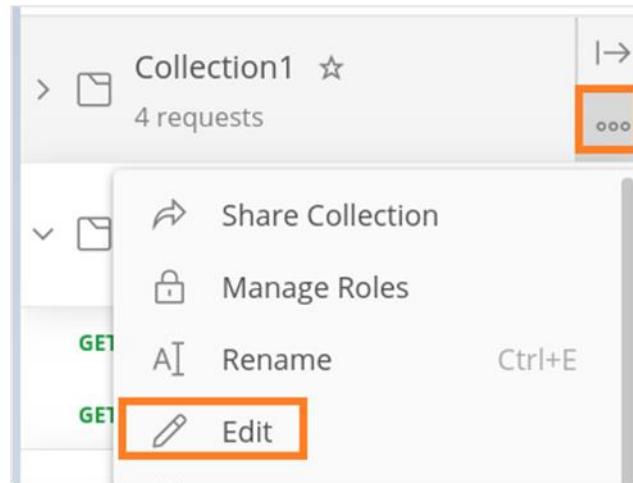
## 17. Postman – Sessions

A session is a temporary fold that stores values of variables. They are used for the present instance and have a local scope. In Postman, we can modify the session variable value to share workspace among teams.

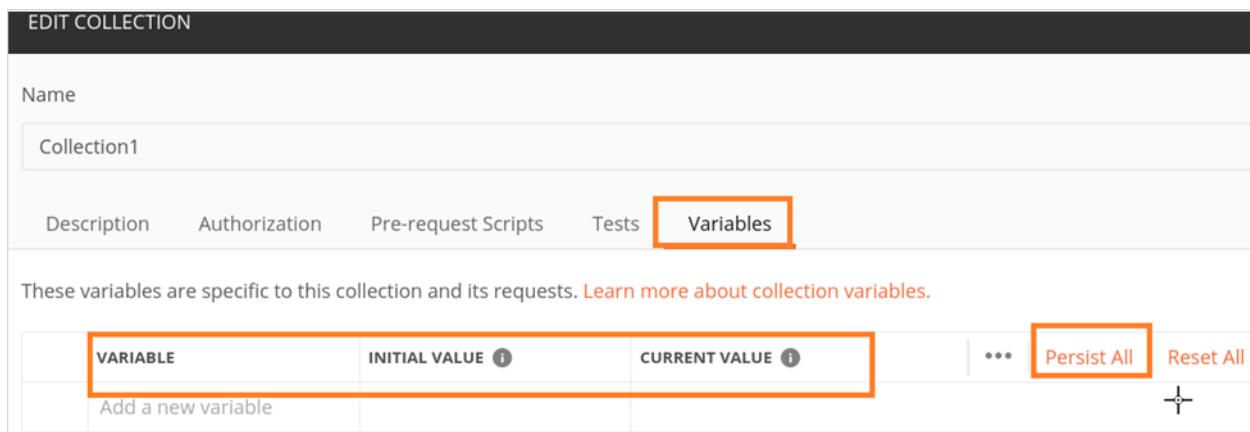
Postman gives the feature of local session share. Even if a Collection can be shared among teams, the sessions are never shared. Different tokens have to be generated while a task is to be carried out in a team structure.

A session has a local scope for a user within his Workspace and any modifications he makes shall not be reflected in the server. In Postman, a session can store Environment variables, global variables and so on.

We can assign current values to Collection variables and to the global and Environment variables. To assign a current value to the Collection, click on the three dots appearing beside the Collection name, then click on Edit.



In the EDIT COLLECTION pop-up, move to the Variables tab.



The screenshot shows the 'Edit Collection' dialog in Postman. At the top, there's a dark header bar with the title 'EDIT COLLECTION'. Below it, a 'Name' field contains 'Collection1'. Underneath the name, there are tabs for 'Description', 'Authorization', 'Pre-request Scripts', 'Tests', and 'Variables'. The 'Variables' tab is highlighted with an orange border. A note below the tabs states: 'These variables are specific to this collection and its requests. [Learn more about collection variables.](#)' Below this note is a table with three columns: 'VARIABLE', 'INITIAL VALUE ⓘ', and 'CURRENT VALUE ⓘ'. There is one row in the table with the text 'Add a new variable' in it. To the right of the table are three buttons: '...', 'Persist All' (which is also highlighted with an orange border), and 'Reset All'. There is also a small '+' icon.

The CURRENT VALUE is local to the user and never in sync with the server of Postman. We can also replace or modify the INITIAL VALUE with CURRENT VALUE. Also, it must be remembered that the INITIAL VALUE gets impacted only if we apply the option Persist on a variable. After that, it gets in sync with the server of Postman.

## 18. Postman – Newman Overview

Newman is a potential command-line runner used in Postman. We can execute and verify a Postman Collection from the command-line as well. Newman has features which are consistent with Postman.

We can run the requests within a Collection from Newman in the same way as in the Collection Runner. Newman can occupy both the NPM registry and GitHub. We can also perform Continuous Integration or Deployment with Newman.

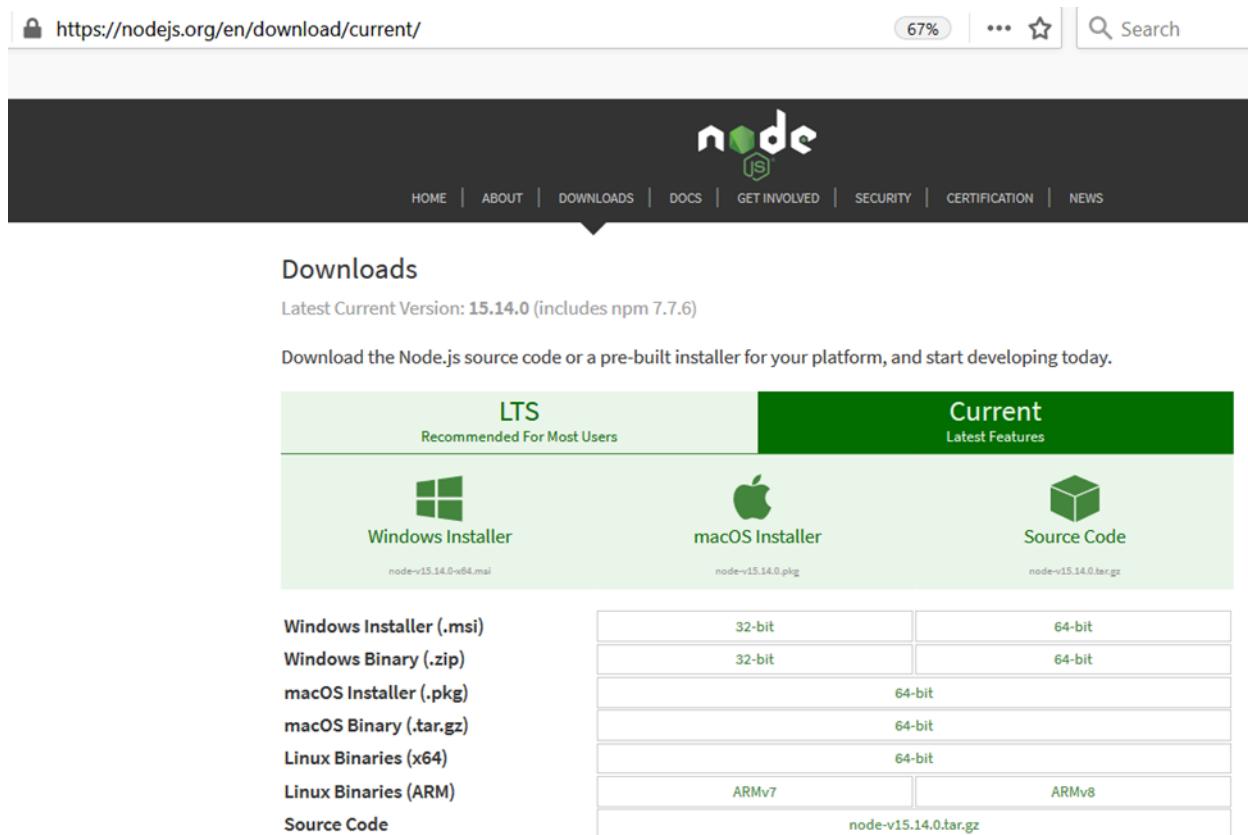
A status code of 0 is thrown by Newman if all the execution is done without errors. The Continuous Integration tools read the status code and accordingly fail/pass a build.

We can add the flag --bail to the Newman to pause on an error encountered in a test with a status code of 1. This can be interpreted by the CI tools. Newman is based on node.js and uses npm as a package manager.

### Newman Installation

The installation of Newman requires Node.js and npm. Follow the steps given below to install Newman:

Step 1: Navigate to the link: <https://nodejs.org/en/download/current/> for downloading the Node.js.



The screenshot shows the Node.js download page at <https://nodejs.org/en/download/current/>. The top navigation bar includes links for HOME, ABOUT, DOWNLOADS, DOCS, GET INVOLVED, SECURITY, CERTIFICATION, and NEWS. A progress bar indicates 67% completion. The main content area features a "Downloads" section with a heading "Latest Current Version: 15.14.0 (includes npm 7.7.6)". Below this, a call-to-action says "Download the Node.js source code or a pre-built installer for your platform, and start developing today." Two tabs are shown: "LTS Recommended For Most Users" (selected) and "Current Latest Features". Under the LTS tab, there are links for Windows Installer (node-v15.14.0-x64.msi), macOS Installer (node-v15.14.0.pkg), and Source Code (node-v15.14.0.tar.gz). A table below lists download links for various platforms:

	32-bit	64-bit
Windows Installer (.msi)		
Windows Binary (.zip)	32-bit	64-bit
macOS Installer (.pkg)		64-bit
macOS Binary (.tar.gz)		64-bit
Linux Binaries (x64)		64-bit
Linux Binaries (ARM)	ARMv7	ARMv8
Source Code	node-v15.14.0.tar.gz	

Step 2: Once the download is completed, execute the below command to verify that the Node.js is installed properly.

The command for verifying the installation in Windows is as follows:

```
node --v
```

The command for verifying the installation in Linux is as follows:

```
node --version
```

The below image shows the version v10.15.2 of the Node.js is installed in the system.

```
osboxes@osboxes:~/Desktop$ node --version
v10.15.2
osboxes@osboxes:~/Desktop$
```

Step 3: The npm is allocated with Node.js so once we download the Node.js then npm gets downloaded by default. To verify if npm is available in our system, run the below command:

The command for verifying the installation in Windows is as follows:

```
npm --v
```

The command for verifying the installation in Linux is as follows:

```
npm --version
```

The below image shows the version 5.8.0 of the npm installed in the system:

```
osboxes@osboxes:~/Desktop$ node --version
v10.15.2
osboxes@osboxes:~/Desktop$ npm --version
5.8.0
osboxes@osboxes:~/Desktop$
```

Step 4: For installation of Newman, run the below mentioned command:

```
npm install -g newman.
```

Step 5: To verify the version of newman, run the below commands: The command for verifying the installation in Windows is as follows:

```
newman --v
```

The command for verifying the installation in Linux is as follows:

```
newman --version
```

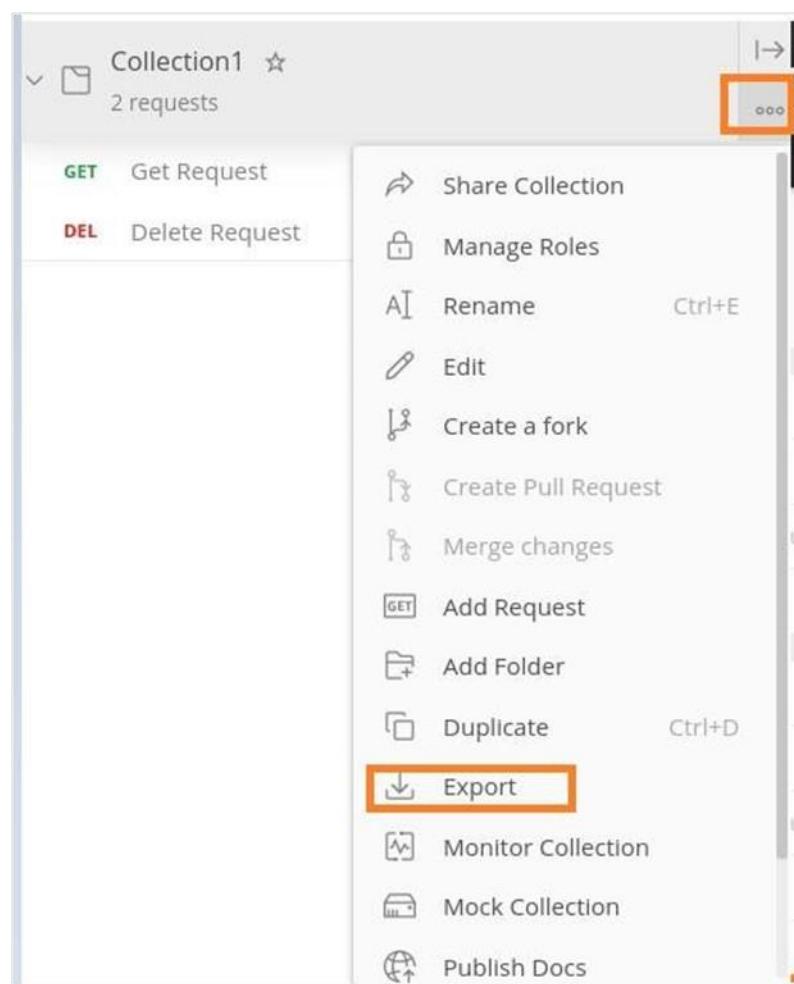
## 19. Postman – Run Collections using Newman

To run Collections using Newman, we have to first launch the Postman application and click on the three dots available beside the Collection name. The details on how to create a Collection are discussed in detail in the Chapter – Postman Create Collections.

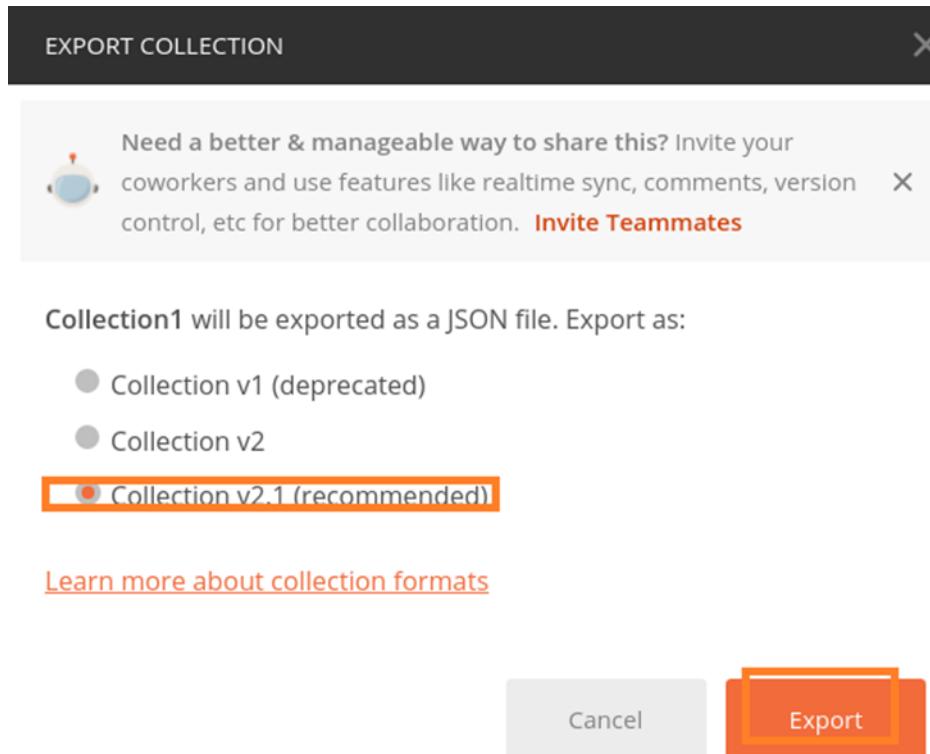
### Run Collections

Follow the steps given below to run collections using Newman:

Step 1: Click on Export.

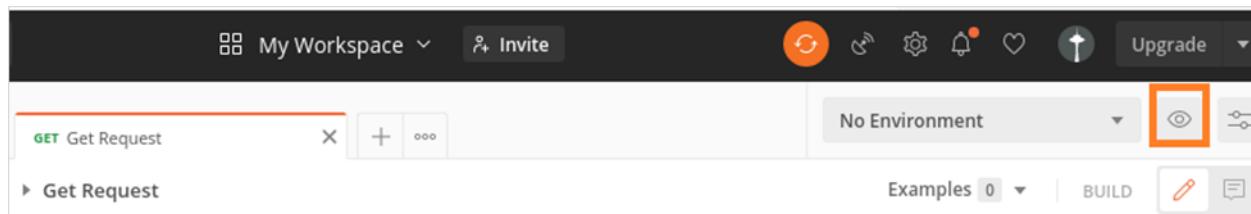


Step 2: Select the option Collection v2.1(recommended) from the EXPORT COLLECTION pop-up. Click on Export.

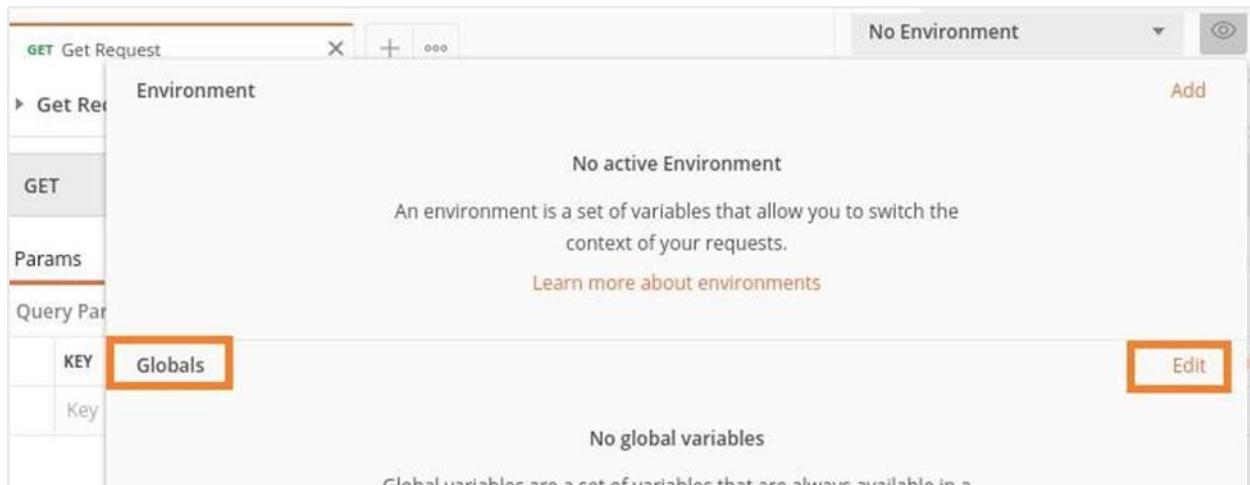


Step 3: Choose a location and then click on Save.

Step 4: Next, we shall export the Environment. Click on the eye icon to the right of the No Environment dropdown.



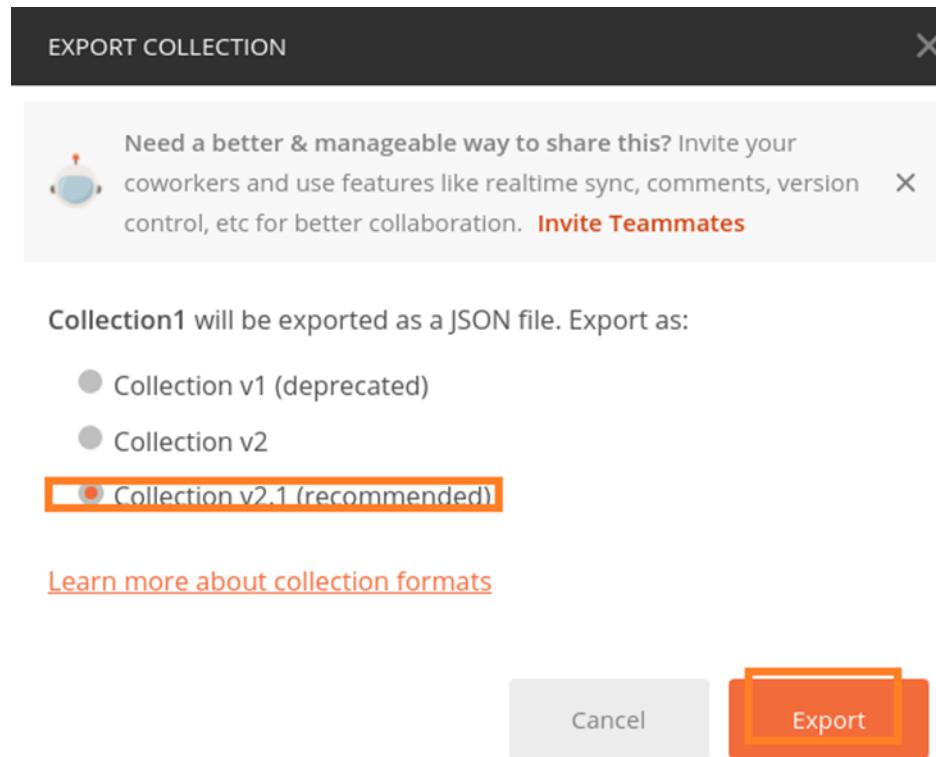
Step 5: Click on the Edit link in the Globals section.



The screenshot shows the Postman interface with a 'GET Get Request' collection selected. In the 'Environment' tab, there is a table for 'Globals'. The 'Key' column has two rows: 'KEY' and 'Key'. The 'Value' column for 'Key' is empty. An 'Edit' button is located at the bottom right of the table, which is also highlighted with an orange box.

Step 6: MANAGE ENVIRONMENTS pop-up comes up. Enter URL for the VARIABLE field and <https://www.tutorialspoint.com> for INITIAL VALUE. Then, click on Download as JSON.

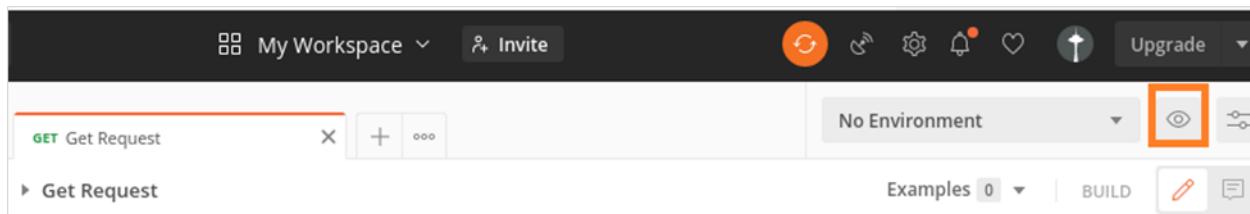
Finally, choose a preferred location and click on Save



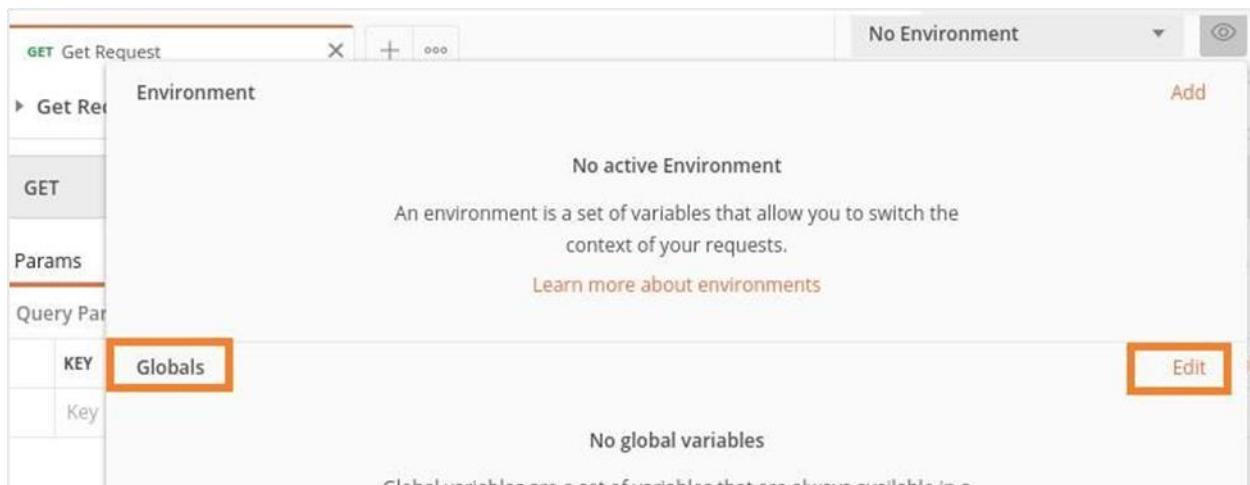
The dialog box has a title bar 'EXPORT COLLECTION' with a close button. Below it is a promotional message: 'Need a better & manageable way to share this? Invite your coworkers and use features like realtime sync, comments, version control, etc for better collaboration. [Invite Teammates](#)' with a red 'X' icon. The main content area says 'Collection1 will be exported as a JSON file. Export as:' followed by three radio button options: 'Collection v1 (deprecated)', 'Collection v2', and 'Collection v2.1 (recommended)', with the last one being selected and highlighted with an orange box. At the bottom are 'Cancel' and 'Export' buttons, with 'Export' also highlighted with an orange box.

Step 3: Choose a location and then click on Save.

Step 4: Next, we shall export the Environment. Click on the eye icon to the right of the No Environment dropdown.



Step 5: Click on the Edit link in the Globals section.



Step 6: MANAGE ENVIRONMENTS pop-up comes up. Enter URL for the VARIABLE field and <https://www.tutorialspoint.com> for INITIAL VALUE. Then, click on Download as JSON.

Finally, choose a preferred location and click on Save

MANAGE ENVIRONMENTS X

Global variables for a workspace are a set of variables that are always available within the scope of that workspace. They can be viewed and edited by anyone in that workspace. [Learn more about globals](#)

**Globals**

VARIABLE	INITIAL VALUE <span style="color: #808080;">(1)</span>	CURRENT VALUE <span style="color: #808080;">(1)</span>	...	Persist All	Reset All
<input checked="" type="checkbox"/> url	https://www.tutorialsp...	https://www.tutorialspoint.com			
<a href="#">Add a new variable</a>					

(1) Use variables to reuse values in different places. Work with the current value of a variable to prevent sharing sensitive values with your team. [Learn more about variable values](#) X

[Download as JSON](#) Cancel Save

Finally, choose a preferred location and click on Save.

Step 7: Export the Environment to the same location where the Collection resides.

Step 8: From the command-line move to the directory path where the Collection and the Environment is stored. Then, execute the command given below:

```
newman run <"file name">.
```

The file name should always be in inverted quotes; else it shall be taken as a directory name.

Common command-line arguments for Newman

The common command-line arguments for Newman are given below:

- To execute a Collection in an Environment, the command is as follows:

```
newman run <name of Collection> -e <name of Environment>
```

- To execute a Collection for a number of iterations, the command is as follows:

```
newman run <name of Collection> -n <iteration count>
```

- To execute a Collection with data file, the command is as follows:

```
newman run <name of Collection> --data <name of file> -n <iteration count>  
<name of Environment>
```

- Configure delay time in between requests, the command is as follows

```
newman run <name of Collection> -d <time of delay>
```

## 20. Postman – OAuth 2.0 Authorization

The OAuth 2.0 is an authorization technique available in Postman. Here, we first obtain a token for accessing the API and then utilise the token to authenticate a request. A token is used to ensure that a user is authorised to access a resource in the server.

If we make an attempt to access a secured URL without the token, a Response code **401 Unauthorized** shall be obtained. To start with, the application passes an authorization request for the end user to access a resource.

As the application allows the user access, it asks for an access token from the server by providing user information. In turn, the server yields an access token. The client can then access the secured data via the access token.