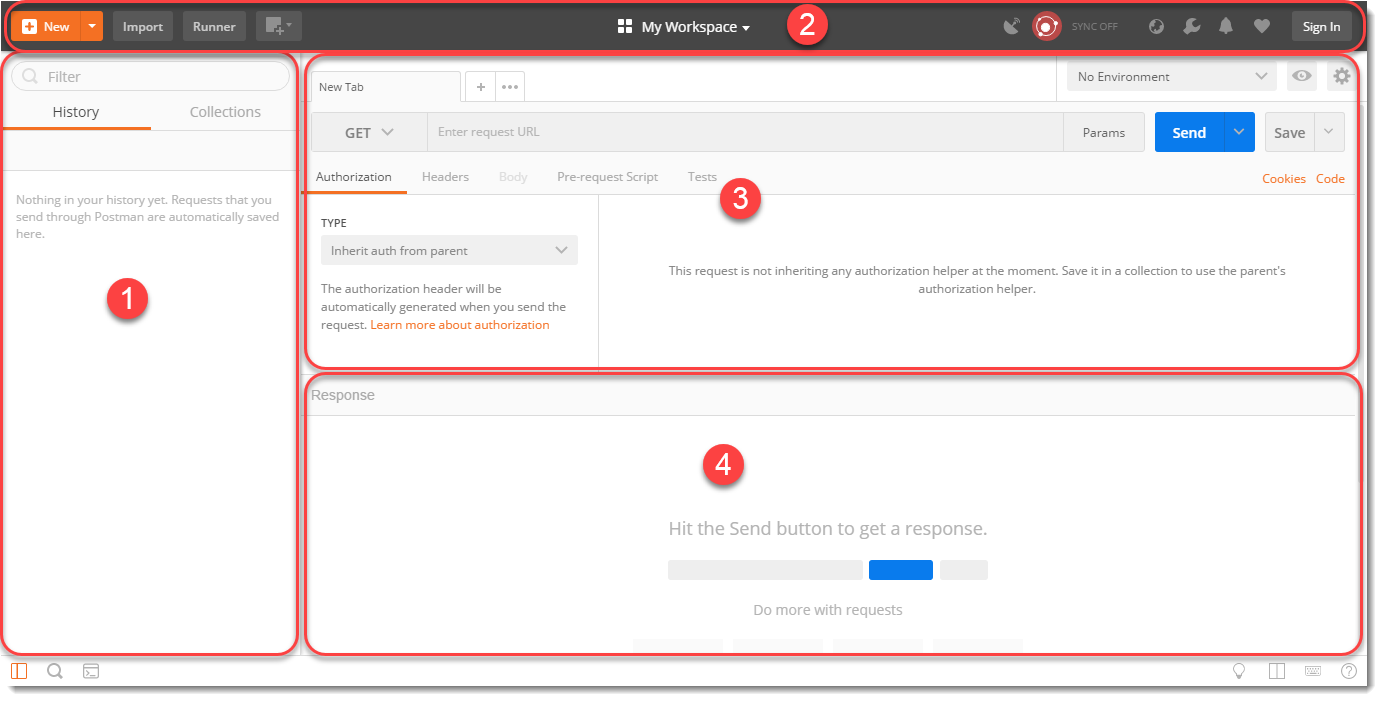
**POSTMAN NOTES**

Postman navigation can be divided into four UI structures as shown below.



1. ***Sidebar section***

* ***History***
* ***Collections***

1. ***Header section***

* ***New***
* ***Import***
* ***Interceptor***
* ***Sync***

1. ***Builder section:****These items will help users to create a new Request. We will learn about these items in detail in the coming chapters*

* ***Tabs***
* ***HTTP Method type***
* ***URL bar***
* ***Header's list***

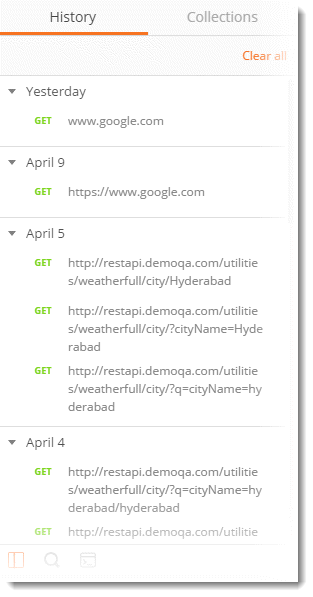
1. ***Response section:****It is filled only when invoking a REST request. This section will be populated with the details of the received Response. We will learn more about it in the coming chapters. Now let us see individual sections in detail.*

**PostMan - Left Sidebar Section**

The sidebar is a very important part of the Postman. The sidebar has two main parts or tabs which are ***History*** and ***Collections***.

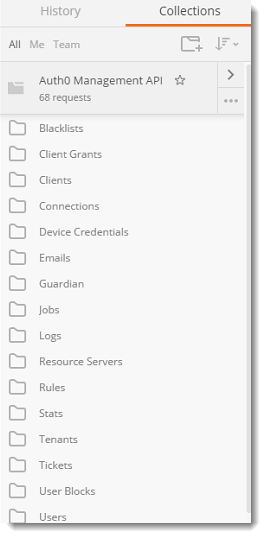
***History Tab***

Postman records a history of your API request just like any other web browser automatically. As soon as you invoke a REST request, it is saved in the history and can be seen below the ***History Tab.*** It comes in handy when you have to search for some particular request that you entered in the past without entering again.



***Collections***

The concept of grouping requests is called ***Collections*** and each **Collection** is displayed under the **Collection** Tab. As shown in the image below. A collection in Postman can be imagined similar to a folder in your system. You create a folder, for example, movies, and keep movies in it so that you know where all your movies are. Similarly in Postman, we save the similar kind of requests under some collection name (*that we define*) and when we open any collection we get all the Requests under that heading, As shown in the below image



**Postman - Header Section**

The below image shows just the Header of the Postman application.

header

The header has the following items

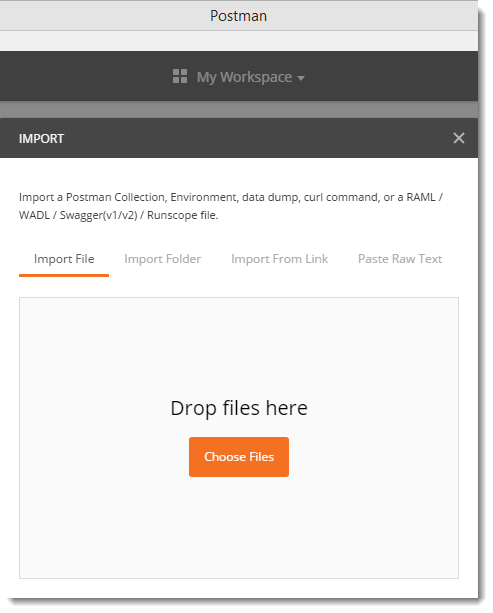
***New***

Choosing this option will let you choose what ***"new"*** you want to start. For example, a collection would open the panel where you can enter a new collection to start and its corresponding requests. Selecting "***request***" in ***New*** will open the request panel where you can enter and save the requests into the collection of your choice. A new option lets you create the following:

* ***Request***
* ***Collection***
* ***Environment***
* ***Documentation***
* ***Mock Server***
* ***Monitor***

***Import***

Import option lets you import files of different formats. Importing means choosing the files located in your system or through a link and running it through Postman. As can be seen from the image it allows you to import a ***Postman Collection***, ***Environment, Curl command***, etc. Importing a collection is the most common among all.



***Interceptor***

Recall we learned that if you are installing the application from chrome then a separate interceptor is required for the proxy server. This interceptor is inbuilt in the native app. You can set a proxy server here to capture all the API requests that you send through your browser. A proxy server can be used to capture all the requests that you send through your browser or from your phone or any other system.

interceptor

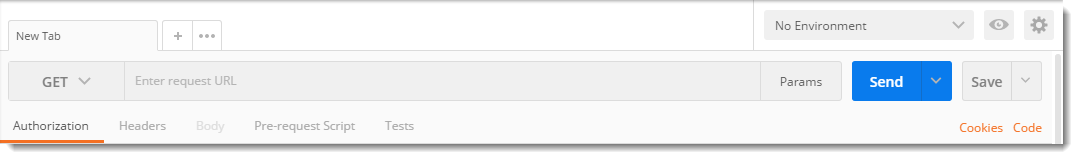
***Sync***

Sync option is for synchronizing the API requests that you have sent on any machine to the Postman cloud. When you are working in Postman and making changes or sending requests, if you ***Sync*** is on, it will automatically be saved in your ***Postman's cloud storage***. This way you can have them saved and whenever you sign in on a different machine to use ***Postman***, they will automatically appear. This feature requires you to sign in  (*If you did not during the installation part*).

Sync

**Postman - Builder Section**

A builder part of the ***Postman*** is basically what a CPU is to a computer. It is the main part that controls all the functionalities and methods to be incorporated inside the API.



A builder part has the following main parts:

* ***Request Type:***
* ***Endpoint Address Bar:***
* ***Params***: This option let the user define different ***Query Parameters*** for the request.

***Request Type***

This is the request type method for the API. It indicates the type of [***HTTP Request***](https://toolsqa.com/rest-assured/rest-api-test-using-rest-assured/) that has been sent. There are different kinds of requests which we will discuss as we proceed further, but just to know, there are four main types of requests namely ***GET***, ***POST***. ***PUT*** and ***DELETE***.

***Endpoint Address Bar***

This is the box, besides the request type option, to enter the ***EndPoint*** (*API*). It acts just like a browser with a similar interface for the New tab. We enter our required endpoint into the bar which is our main URL.

***Params***

Params are the parameter option that allows us to write the parameters of the URL. The parameters are embedded into a URL and are very important to get the desired result. They also help us in getting efficient usage of the memory and bandwidth. This will be discussed in a complete chapter later on.

***Authorization***

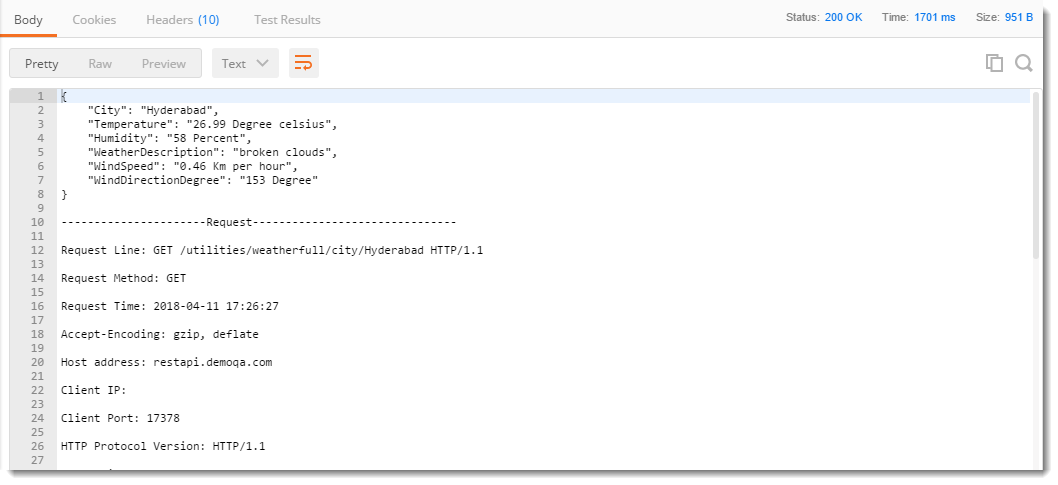
The authorization process verifies whether you have permission to access the data you want from the server. Not all data is available for everyone inside a company, so there lies the solution as Authorization. With the authorization, the server first checks whether the data you are asking for can be shown to you. If it can be, you get the desired response.

***Header***

A header in the HTTP request or response is the additional information that is needed to be conveyed between the client-server. HTTP headers are mainly intended for the communication between the server and client in both directions.

***Postman - Response Section***

A response box is a box that shows the response from the server that we receive after requesting through API. A response box has many options in it, which won't be feasible to explain here in this chapter. In the coming chapters you will learn about the response, although if you want you can visit the chapter [***here***](https://toolsqa.com/client-server/http-response/).

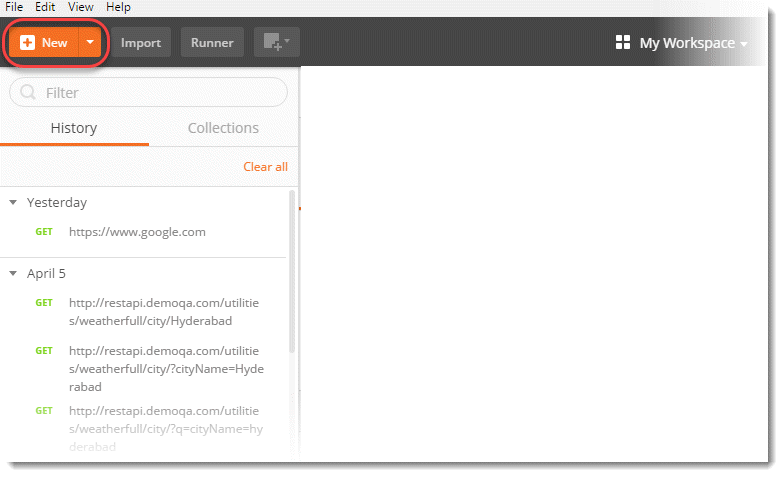


Now, since we have installed Postman and have become familiar with the interface, it's time to start our first steps on Postman for which Postman is actually used for. To start with this tutorial, we will start with the ***Header*** part of the Postman and follow the steps.

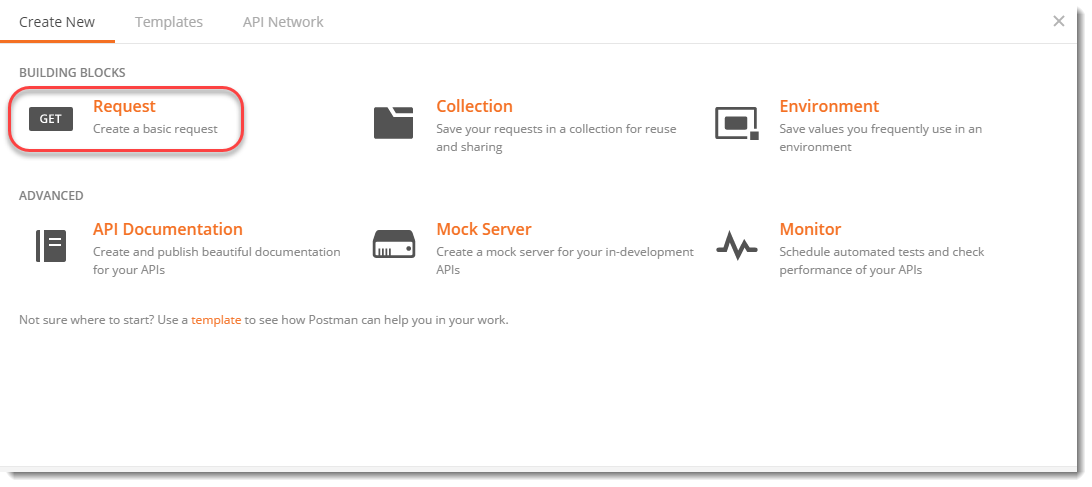
You may also go through the recording of the Postman Tutorial where our experts have explained the concepts in depth.

***Create New Request in Postman***

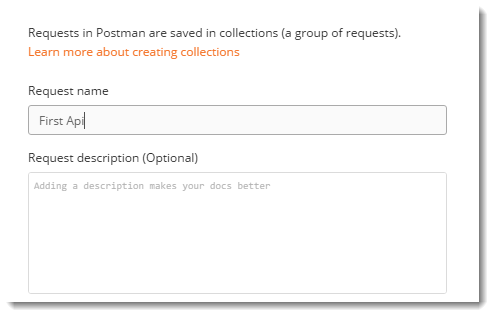
1. Click on the ***NEW*** option in the header part.



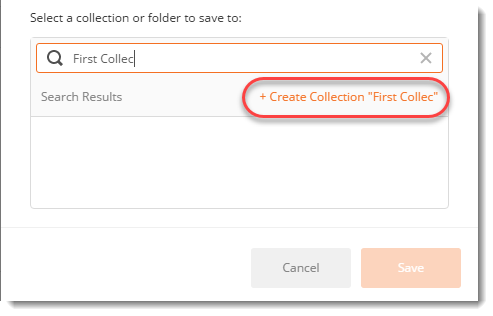
1. Click on ***Request.***



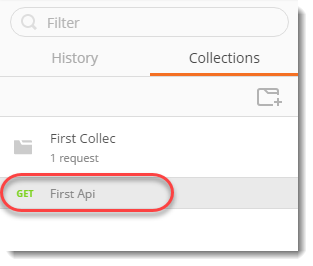
1. Enter a meaningful ***Request Name***, like ***First Api*** we are using. You can also use the description about the API to remember later about what that API did for other teammates and yourself, but it's optional and we won't be using that in this tutorial.



1. Enter a meaningful Collection name in the bottom panel, like ***First Collec*** we are using, and select ***+Create Collection*** as shown. Press ***Save***.



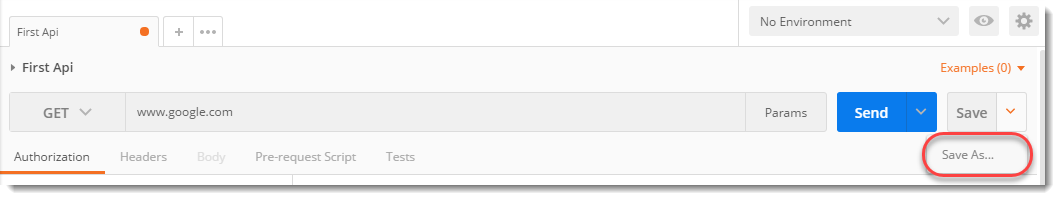
1. Select *Collections* tab in the sidebar, then you will notice all the collections folders, select *First Collec* and then select *First Api* under the First Collec tab.



1. Enter ***www.google.com*** in the ***Address Bar*** and press ***Send***.



1. Press ***Save*** if you wish to overwrite "***First Api***" or press the dropdown as shown and Save as a new request.



The ***Save As*** option opens the same panel which opened through *New Request* at the start of this tutorial. It gives the option to enter the name and associate the request to some collection.

This way you have created a Request and saved it under the desired collection. In the next tutorial, we will send our first ***GET*** request.

Since we have now walked through Postman and seen [***How to Create and Save a new Request in postman***](https://toolsqa.com/postman/create-new-request-in-postman/), it's time to get our hands on the first ***GET Request in Postman***. When we request from a client machine (*User*) to a server machine, we follow an architecture and HTTP Protocol. I suggest you go through the below tutorials to establish a nice understanding of ***HTTP Protocol, Request & Response***. These can be viewed here:

* [***Client Server Architecture and HTTP Protocol***](https://toolsqa.com/client-server/client-server-architecture-and-http-protocol/)
* [***HTTP Request***](https://toolsqa.com/client-server/http-request/)
* [***HTTP Response***](https://toolsqa.com/client-server/http-response/)

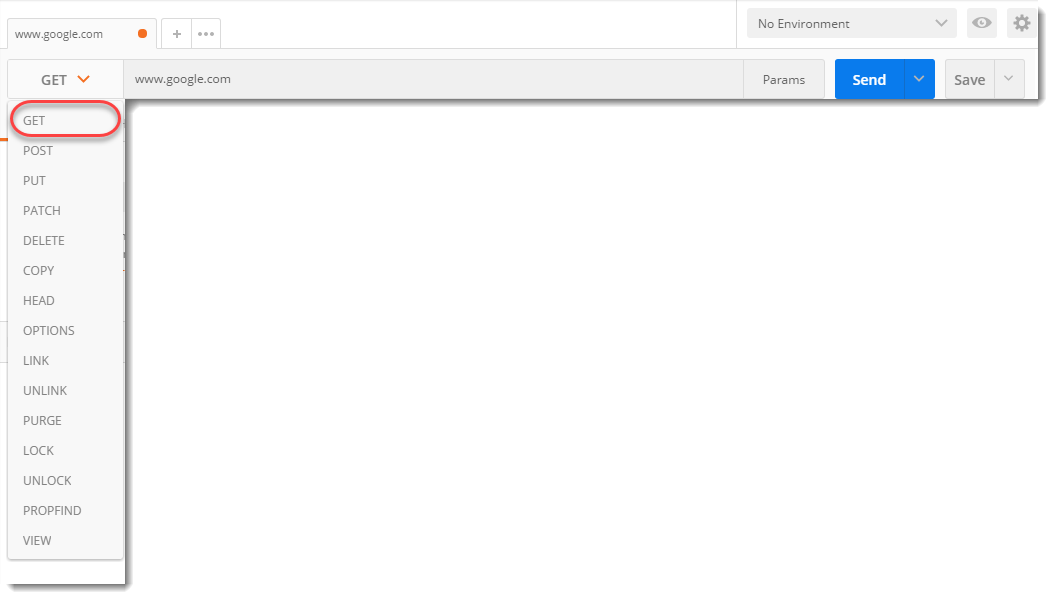
Assuming you are now familiar with the HTTP protocols and architecture, we will now talk about one specific type of request which is a ***GET*** request. A *GET* request is used to get the information from the server and does not have any side effects on the server. *Side-effects mean there is no updation/deletion/addition of data on the server when you are making this type of request, you just request from the server and the server responds to the request.*

A *GET* request has all its information inside the ***URL***, and since URL is visible all the time, it is advisable not to use this type of request while you send some sensitive information such as passwords. ***For example****, when you press search after writing anything in the search box of google.com, you actually go for a GET request because there is no sensitive information and you are just requesting the page with search results, you notice the same search string in URL.*

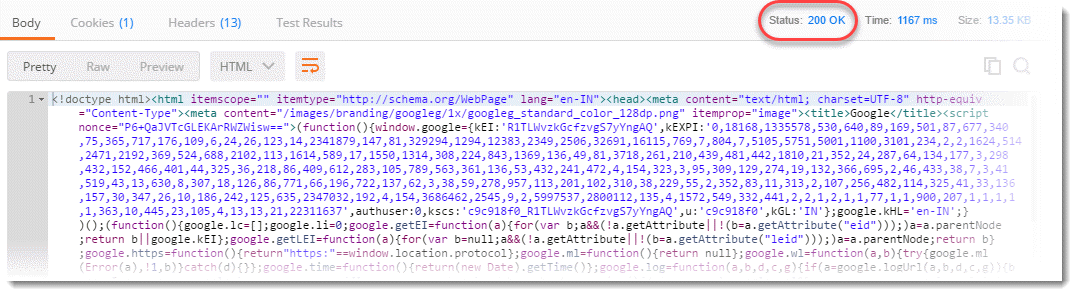
You may also go through the recording of the Postman Tutorial where our experts have explained the concepts in depth.

In this image, as you can see, there is a drop-down button which has different types of request types according to the API need. As of now do not worry about all of these different [***HTTP Requests***](https://toolsqa.com/client-server/http-request/), as we will cover each of these in this Postman Tutorial series. But for now, just focus on the GET Request.

1. Select ***GET*** from the list of request types.

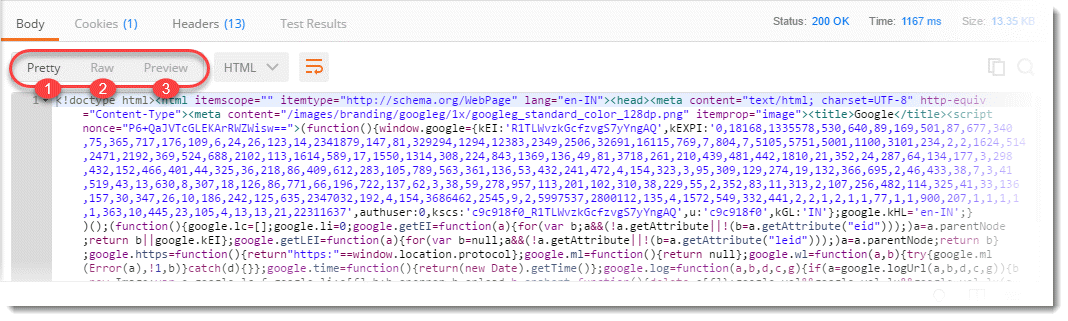


1. Enter ***www.google.com*** in the address bar as written in the above image and *Press****Send.*** Now, look at the ***Status Code.***



Different status codes have different meanings and it does not matter whether it is a GEt request or any other type of request. In this scenario, we have status code ***200 OK*** which means that EndPoint is correct and it has returned the desired results. We will show some more status codes later.

The colorful text inside the box below is the Response from the server. If you observe closely inside the response box you will see the page code has been sent to us. The above tab says ***Body***. Body means you have selected to view the response body which is been shown inside the box. In Body, you will see three options.



* ***Pretty:****In this code will be shown in a colorful manner with different keywords colored differently and will be indented for some of the formats for good reading.*
* ***Raw:****Same as pretty part with no colors and in single lines.*
* ***Preview:****This shows the preview of the page that has been sent. Don't worry about the google doodle if it has not been loaded properly. Try any other website by yourself.*

A ***response*** is a more detailed topic than it needs to be explained in this chapter. We will be explaining the response completely in the next chapter.

In the last tutorial we learnt about [***Get Request in Postman***](https://toolsqa.com/postman/get-request-in-postman/). In this tutorial we will understand how to deal with ***Response in Postman***. It will be beneficial if you understand the underlying details of an [***HTTP Request***](https://toolsqa.com/client-server/http-request/) and an [***HTTP Response***](https://toolsqa.com/client-server/http-response/).

### *****What is Response?*****

A ***Response*** is a message that is received by the server in return to a ***Request*** that we send. When we request something, the server acts upon the ***Request*** and sends back a packet of the requested information. A response depends on the request mainly. Every request has a different kind of response and it is very important that we extract useful information from all of the responses. Postman has a beautiful interface for response and is very user-friendly. We can see a lot of information in the Postman for any response without doing much effort, or any if I might say.

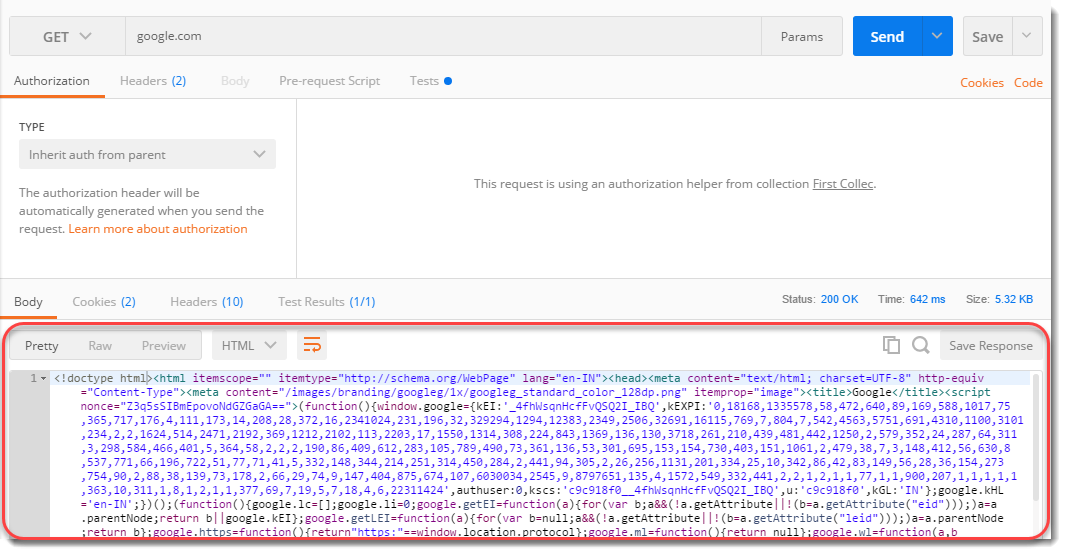
You may also go through the recording of the Postman Tutorial where our experts have explained the concepts in depth.

# Understanding Response in Postman

Talking about ***Response in Postman,*** the Response user interface contains lots of different things. We will deal with them in detail in this tutorial. The user interface has the following information blocks

* ***Response Status and Information***
* ***Response Body***
* ***Response Cookies***
* ***Response Header***

Let's start by getting a response for ***www.google.com*** which looks like this:



## Response Status and Information

### *****Status Code :*****

A ***status code*** tells you the status of the request. There can be a lot of mistakes in the request and without looking at the status code, we might not always get what went wrong to our request. Sometimes, there can be a typing mistake in the URL or there can be a problem at the server-side, status code help us know about what went wrong (if something went wrong). There are different status codes and each of them has a different meaning.

You can learn about the complete list of status code [***here***](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes).

Status_Code_200_2

Status code ***200 OK*** means that the request was correct and the desired response has been sent to the client. Now, change the URL to ***http://restapi.demoqa.com/utilities/weatherfull/city/hyderabd .*** Press Send and see the status code now.

400_Bad_Request

It says ***400 BAD REQUEST***. It is so because we have changed the name of the city from ***Hyderabad to Hyderabad***. This means the request was not correct, hence the bad request response. Similarly, you can see other status codes also for different requests.

### *****Time*****

***Time*** is the duration which the response took after we sent the request and received the response. This is very important sometimes because many projects have Service Level Agreements(SLA) for the time it should take a web service to return a response, this time can be a used to determine the SLA of the web service endpoint.

Time

***NOTE***: The time given here is not the actual time that the request will take. It is just approximate but almost what it would be because there are a lot of things that Postman do after getting a response such as formatting and dividing Headers and cookies separately. As the additional work by Postman can be roughly considered as a constant time (WebServiceTime + Constant processing time by Postman). Therefore, it is an approximate of the time and is proportional to what the actual time will be. So you can consider this as actual time as well.

### *****Size*****

***Size*** is just the response size when it will be saved inside the memory. This response size is the size of complete response and headers and cookies and everything that has been sent along with the response.

Size_Reponse

***NOTE***: The response size that is shown in the Postman is approximate response size and not the exact size.

## Response Body

A ***body*** depicts the body of the response, which is the main response content, that has been sent from the server. In this case as you can see it is a web page code being sent to us as a response. Now, there lie three ways ahead of us to look at this response:



### *****Pretty*****

***Pretty*** is a prettier version of the content being sent. The content is prettier as it is more readable. It has coloured keywords and different colours have different meanings. This makes a code more readable and look nicer. This formatting is done by Postman itself after getting the code.



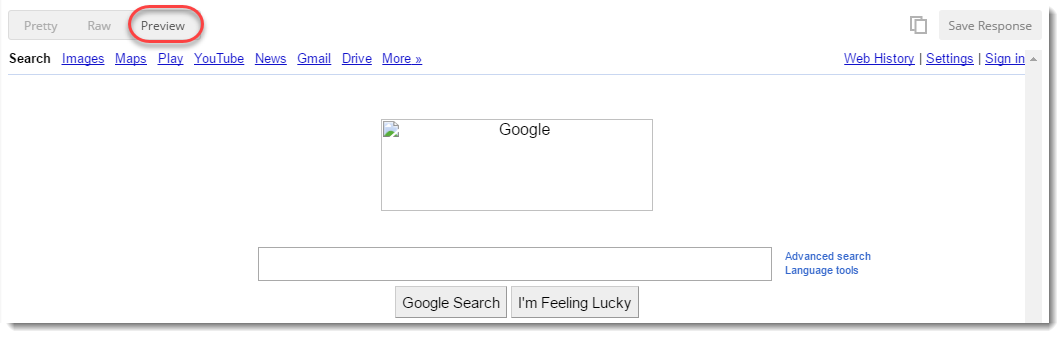
### *****Raw*****

Once you click on Preview you will get just the plain view of the content, as received from the server. It is just a raw version of the code without any colorful keywords. By looking at this code you might get why the other code is called "***Pretty***".



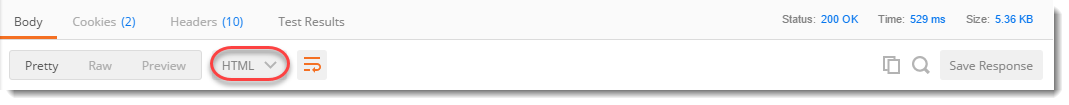
### *****Preview*****

Preview of the code will show you the preview of the page, had the page been run inside a browser. Click on preview and you will see the exact page as you would have seen inside a browser. So this would let you know the response preview without visiting the browser.



### *****Format Type*****

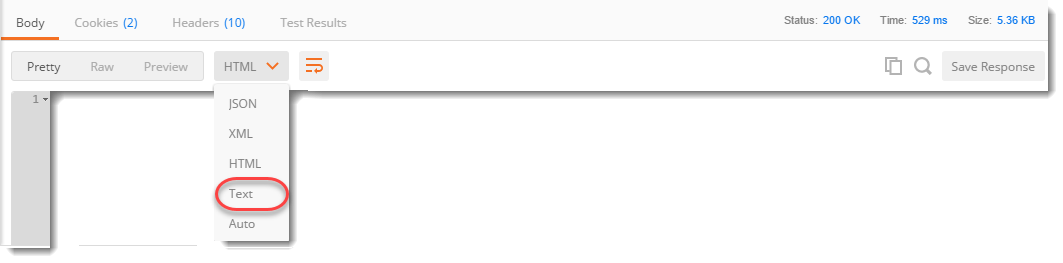
As discussed above, a request has a defined response to it as defined by the ***Content-Type*** header. That response can be in any format. For example, in this case we have the response as a HTML code file.



Postman is smart enough to detect the response type and show you in the desired format, but sometimes Postman can also make a mistake. For example, use ***http://restapi.demoqa.com/utilities/weatherfull/city/hyderabad*** to get a response.

You will see that we have received a status code 200 and still there is no response. This is because Postman has failed to recognize the format of the response and is expecting a HTML file as seen in the dropdown.

Select ***Text*** in dropdown and you will be able to see the response now.

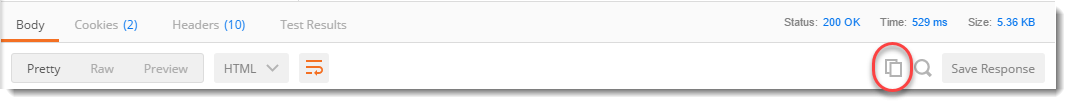


Sometimes, the server sends the response in two or more different formats. The type of response will be visible to its corresponding format type.

***Note:*** Content-Type header defines the format of the response. For e.g. the Content-Type header may say that the response is JSON, however, the content being sent is XML or a malformed JSON. In that case Postman will not be able to do much. Take it as an exercise to understand why Postman is not able to understand the format of response returned by ***http://restapi.demoqa.com/utilities/weatherfull/city/hyderabad***

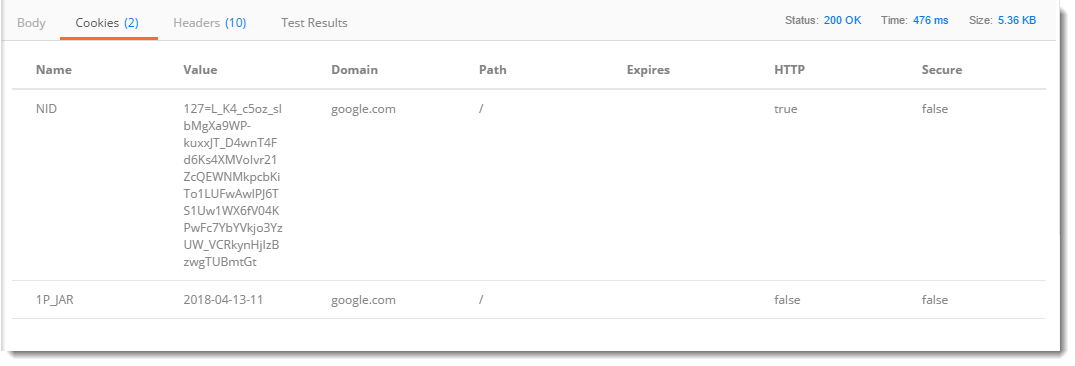
### *****Copy Response*****

The icon with two rectangles that you see in the corner is used for copying the complete response to the clipboard which is very handy to send the response to your teammates or using afterwards.



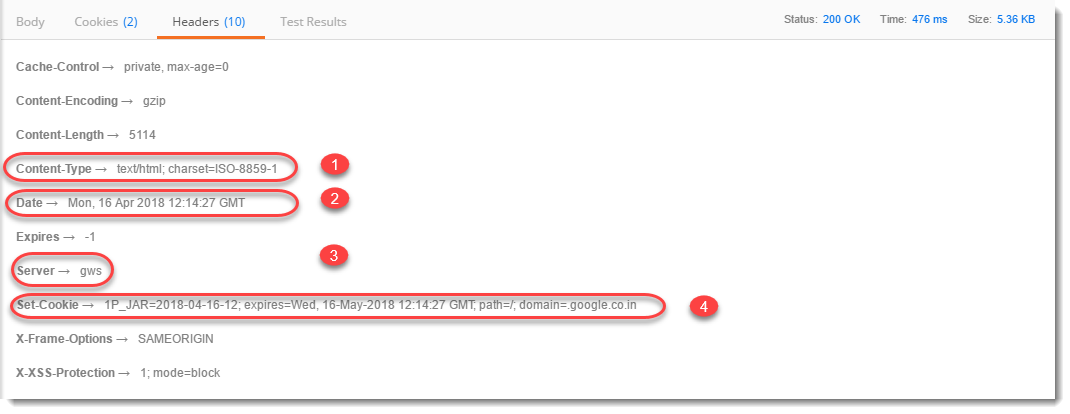
## Cookie

Cookies are the small files which are related to the server files (website pages). Once you visit a website for the first time, a cookie is downloaded on the client's machine. This cookie contains the information which can be used by the same website when you visit again. This helps the website to get you the specific response and specific information based on your last visit. In postman we can clearly see the cookies that have been sent from the server as a response. This makes it easy for the client to see what cookies are being saved inside his browser. We cannot manipulate this cookies since they are sent from server, Postman is used just to separate it from the response and have a clear view.



## Header

***Headers*** in an HTTP request or response is the additional information that is transferred to the user or the server. In postman, the headers can be seen in the ***Headers*** tab.



Once you click on header you can see different information such as below. Although, every entry in the Headers tab is a header item we will just take a look at the most important ones.

* ***Content-Type :*** This is the content type of the response. In the above example when we used www.google.com the content type is given as ***text/HTML*** because the response is being sent in the HTML which is one of the options.
* ***Date :*** This option shows the date, day and time of the response along with the time zone.
* ***Server :*** This option tells the name of the server which has responded to the request. In the above example, the server name is shown as ***gws*** which corresponds to ***Google Web Server.***
* ***Cookie expire time :*** As the name suggests, this option tells the expire time of the cookie that has been sent along with the response.

***What are Parameters in Request?***

Request Parameters are part of the URL which is used to send additional data to the Server. Let us analyze a simple URL:

***https://www.bing.com/search?q=ToolsQA***

In this URL Request parameter is represented by the "***q=ToolsQA***" part of the URL. Request parameter starts with a question mark (*?*). Request parameters follow ***"Key=Value"*** data format. In our example ***"q"*** is the Key and ***"ToolsQA"*** is the value. The server reads the Request parameter from the URL and sends a Response based on the Request Parameter. In our example, Bing will return search results for ToolsQA. If you change the Value from ***ToolsQA*** to ***DemoQA***, you will get results for ***DemoQA*** instead of ToolsQA. This means that the Server reads the Request parameter and responds based on that.

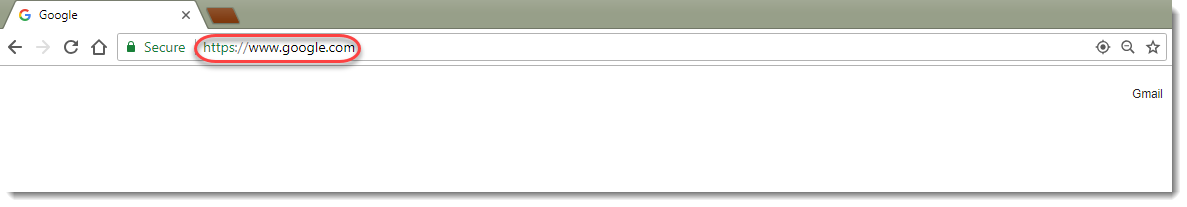
*In simpler words, let us say I designed a page that can show you the student list of a particular class. Now, the list will depend on the class you select, which will be passed in the URL as a parameter while the page I designed will be the same for every class. I don't have to design many pages as many as there are classes. This way we improve the efficiency and usage at both levels.*

***Parameters*** can be passed in ***GET Request***, if you are not sure how to do a GET Request using Postman, please take a look at the previous article [***How to make a GET Request***](https://toolsqa.com/postman/get-request-in-postman/). Since now you know how to make a GET request, we will move ahead with sending parameters in a GET request.

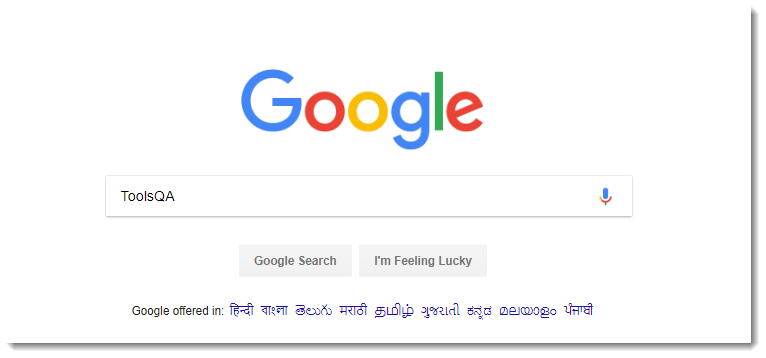
You may also go through the recording of the Postman Tutorial where our experts have explained the concepts in depth.

Before talking about Parameters and understanding them clearly, we will send the URL to our browser.

1. Go to your browser and write ***www.google.com*** in your address bar

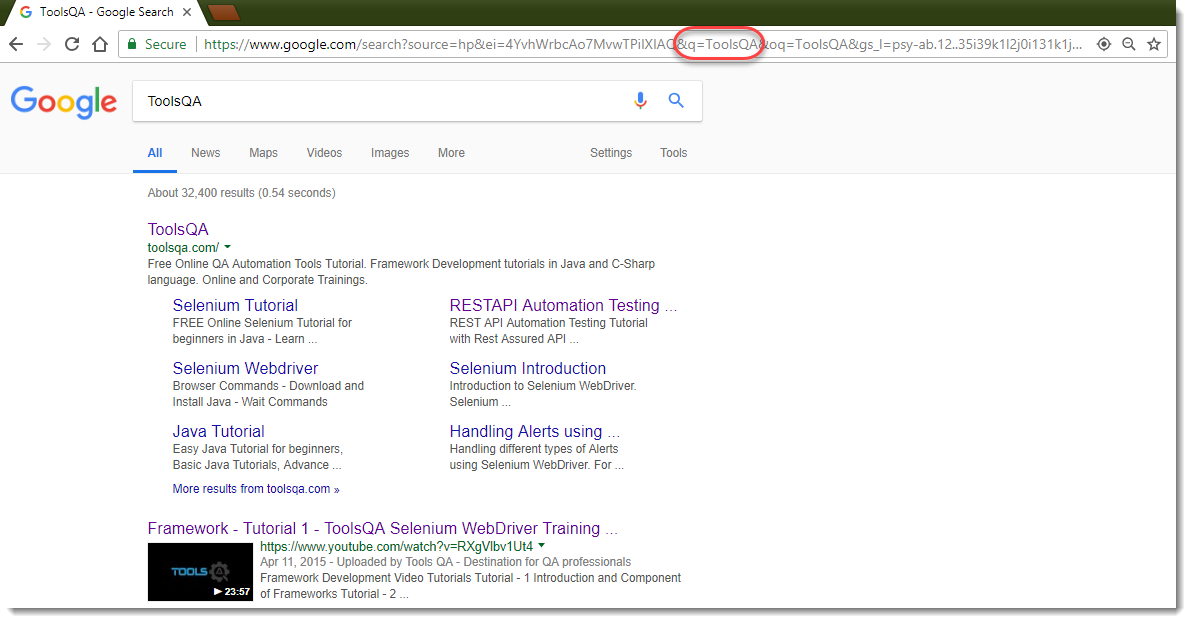


1. You will see the response page from Google. Type ***ToolsQA*** in the search bar and press ***Google Search***.



Now you need to understand here that the page which shows the results will remain the same just the results will differ according to the search. You just now searched for ToolsQA, this serves as a parameter in the URL to tell the server that we need the results of the ToolsQA specifically. The server responds according to the search parameter.

A URL consists of many parameters such as ***source id*** and ***encoding format*** etc. See the URL below and you will see ***&q=ToolsQA*** which is added in the URL to tell the server.



***Note****: here****"q"****is the****key****represents****query****and****ToolsQA****is the****value****of the key or****search term****for the query.*

Now, we will try to achieve the same results through Postman.

**Request Parameters in Postman**

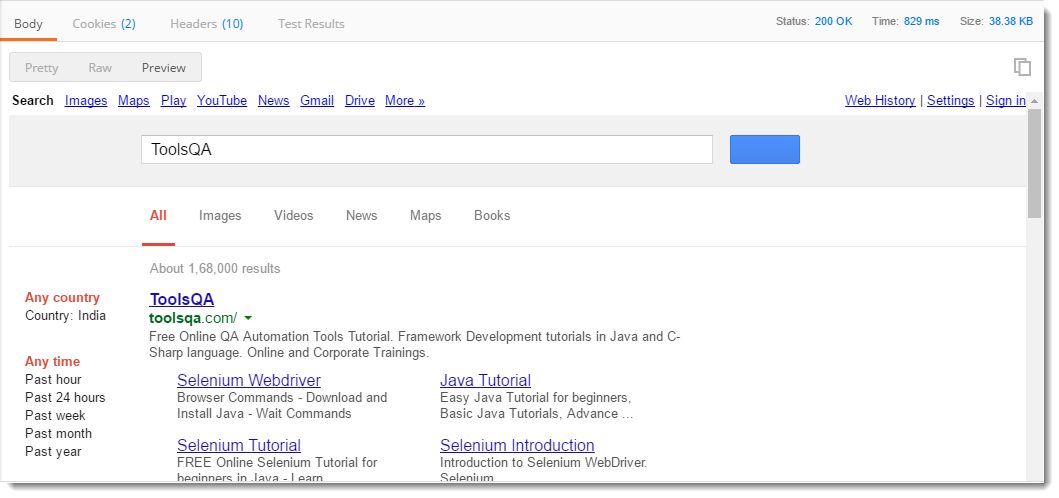
1.Just prepare a [***GET Request in Postman***](https://toolsqa.com/postman/get-request-in-postman/) with the URL *www.google.com/search* and then click on ***Params.***



1. Write the following things written under ***Key-Value pair*** as shown. Again ***q*** stands for ***query*** here and ToolsQA is the search term. Now press ***Send***.



1. Look at the preview, you would see that instead of the google home page we have received a response for a specific search query which is ***ToolsQA***. Instead of ToolsQA you could write anything and receive its response. This indicates that we have passed some information (*Parameters*)  about the result we wish to see.

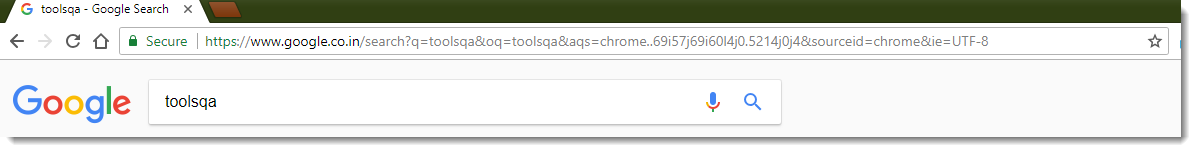


***NOTE****: As discussed above, you can see that different search queries give different results but the page design remains the same, just the content differs.*

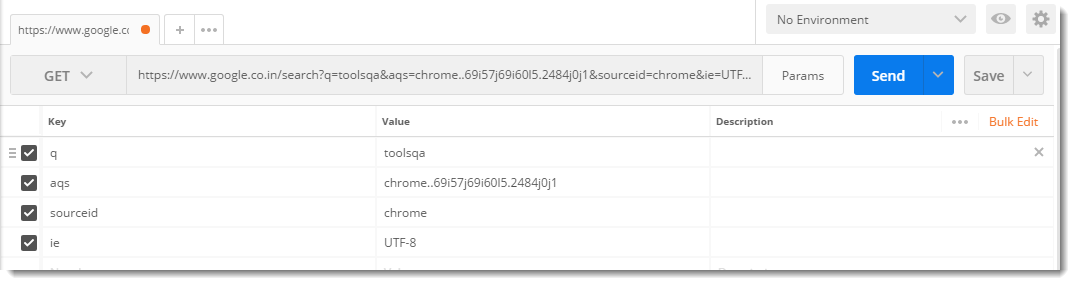
***Multiple Parameters***

You can also use ***multiple parameters*** inside a single query. As we discussed above while sending the search query as *ToolsQA,* there are many parameters that are sent in the URL. *ToolsQA* was for the results that were to be shown, another parameter such as encoding format is also used to tell the server in which format can the result be encoded and sent to the client. In the above example, the default encoding format used is UTF-8.

See the above image and focus on just the URL sent to the server



In the above URL wherever you see ***&*** it must be succeeded by a parameter such as ***&ie=UTF-8*** means ***ie*** is a *key parameter* with a value *UTF-8*. You can write each of the parameters that you see above in the postman and send a request with multiple parameters.



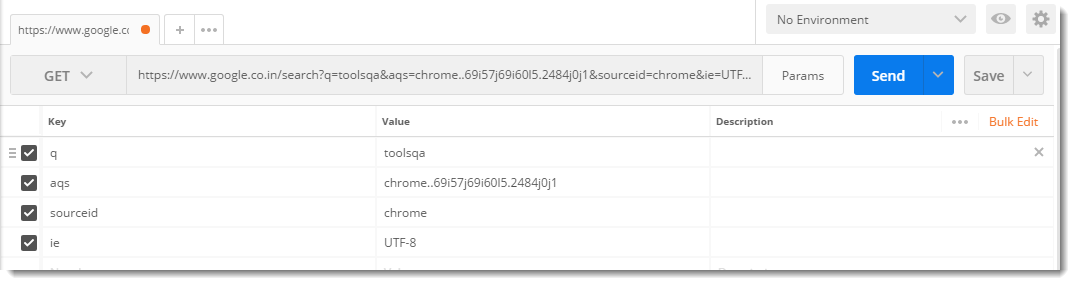
These parameters are not for our users to study in detail. Even if you change the parameters, the changes reflected will not be seen on the page and you will still get the same response as before because all these parameters are for internal activities in the server such as logging the submission.

***Separating parameters from URL***

If you are wondering how to separate a given complete URL with its parameters to use in Postman then Postman has it sorted out for you. You do not need to worry about the parameters in a URL. You can simply paste the URL and Postman will fill the parameters itself.

*For example, copy and paste this URL inside your postman like shown below****https://www.google.co.in/search?q=toolsqa&oq=toolsqa&aqs=chrome..69i57j69i60l5.2885j0j4&sourceid=chrome&ie=UTF-8***

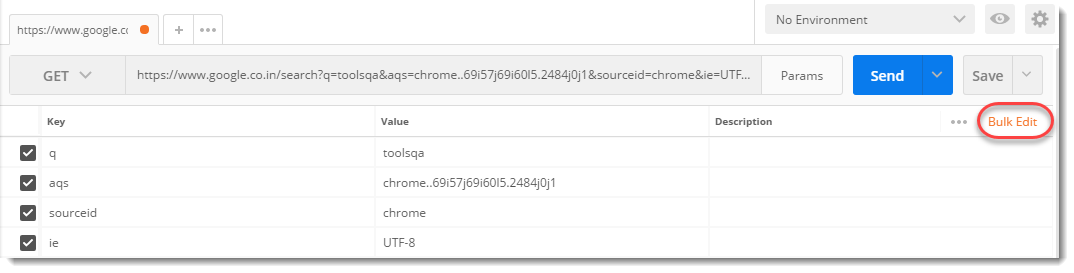
Now click on ***Params*** and you can see that everything is sorted out itself and the parameters are as in the above image (*more or less*).



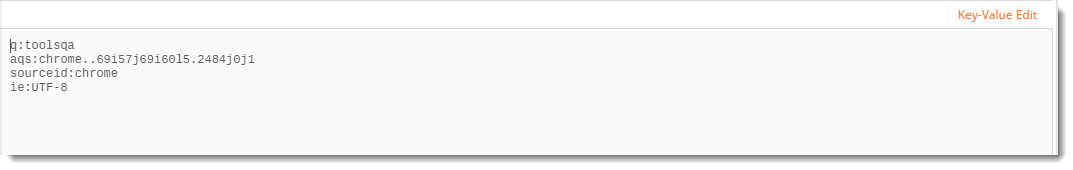
***Copy parameters to another Postman Request***

Another interesting feature about *Params* is that Postman removes the headache of remembering and entering the same parameters again and again to every query, instead it lets you enter once and forget about entering the same parameters again. *For example, let's say you have to run the same query that we just run but with a few fewer parameters.* For achieving the same,

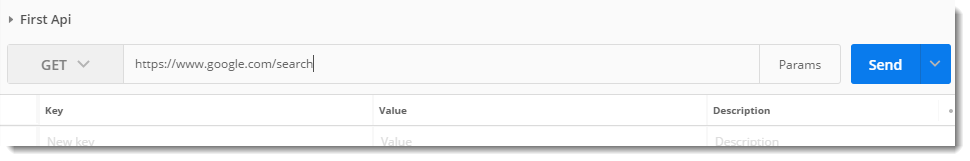
1. Click on ***Bulk Edit,*** you will see the list of all parameters



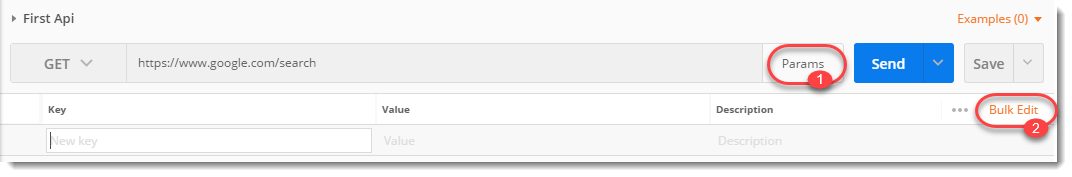
1. Copy everything



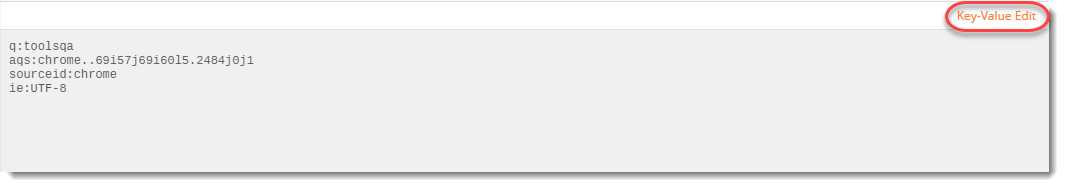
1. Open a new tab and write your URL which is ***www.google.com/search*** in this case



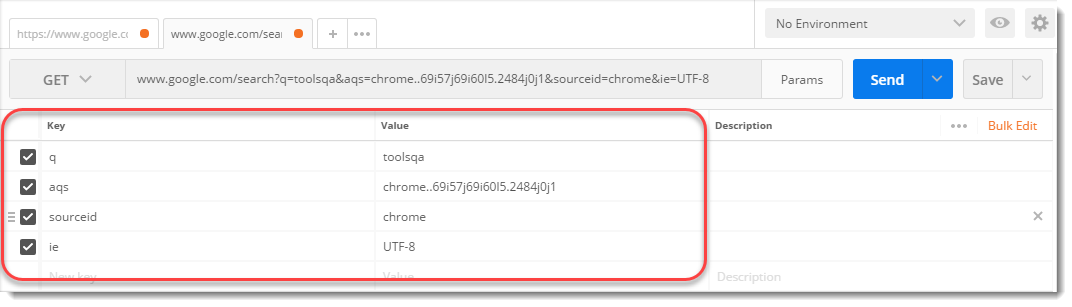
1. Click on ***Params***, then ***Bulk Edit***



1. ***Paste*** everything you copied in the editor and click on ***Key-Value*** edit



Here you will see every parameter has been adjusted automatically to the new request.



This makes Postman really very efficient while using the parameter option and leaves us out of the context of its complexity. A parameter is a very important part of a URL and readers are recommended to observe the different parameters in a URL for better learning and understanding, whereas this was all about parameters usage inside Postman. Next, we will see about the response in Postman.

### *****What is a POST Request?*****

A ***POST*** is an ***HTTP Verb*** similar to a **GET** request, this specifies that a client is posting data on the given **Endpoint**. A ***POST*** request is a method that is used when we need to send some additional information inside the body of the request to the server. When we send a POST request we generally intend to have some modification at the server such as ***updation, deletion, or addition***. ***One of the classic example of a POST request is the Login page. When you first Sign Up for anything, let's say Facebook, you send your personal information such as a password to the server. The server creates a new account with the same details and that account is added permanently on the Facebook server.*** You just created a new resource on to the server. ***POST*** requests are very popular and are mostly used whenever you are sending some sensitive information such as submitting a form or sending sensitive information to the server.

In this tutorial, we will explore different features of ***POST*** Requests and how we can create them in Postman. Before we will try to use an example to get a clear idea about a ***POST*** Request.

### *****POST Request in Postman*****

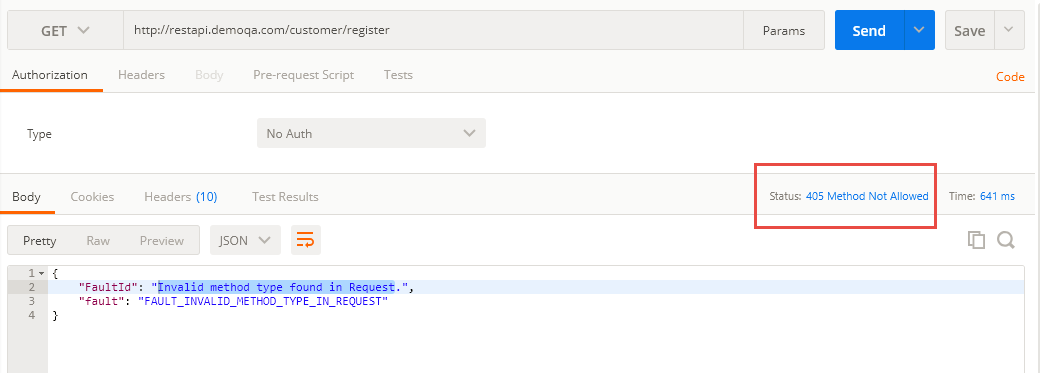
Every ***REST endpoint*** has its own ***HTTP verb*** associated with it. If an endpoint specifies that it should be called using the POST HTTP verb, then clients are bound to call the Endpoint with ***POST HTTP verb*** only. Let's first check what happens when we request the GET method instead of the ***POST*** method for a ***POST Endpoint***. Also to check what happens when we do ***POST*** Request without ***Body***.

##### ***GET Request on POST Endpoint***

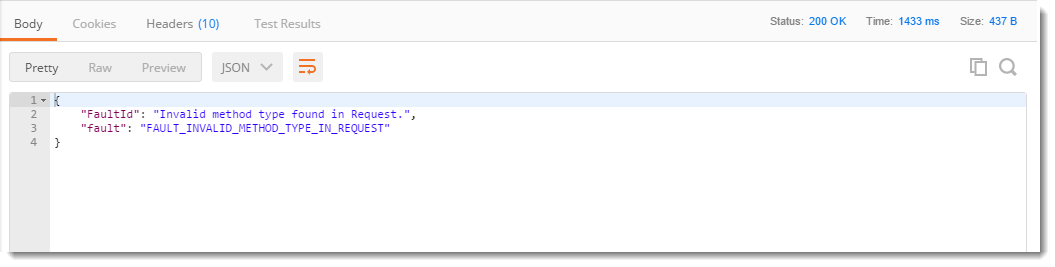
1. Use the API ***http://restapi.demoqa.com/customer/register*** (This API is used for registering a new customer) in the Postman endpoint bar and press ***Send***. Make sure that **GET** is selected in the Method type dropdown.



1. See the HTTP status code, it will be ***405 Method not allowed.*** Which means that we are hitting the endpoint with incorrect method type. The below image shows the details.



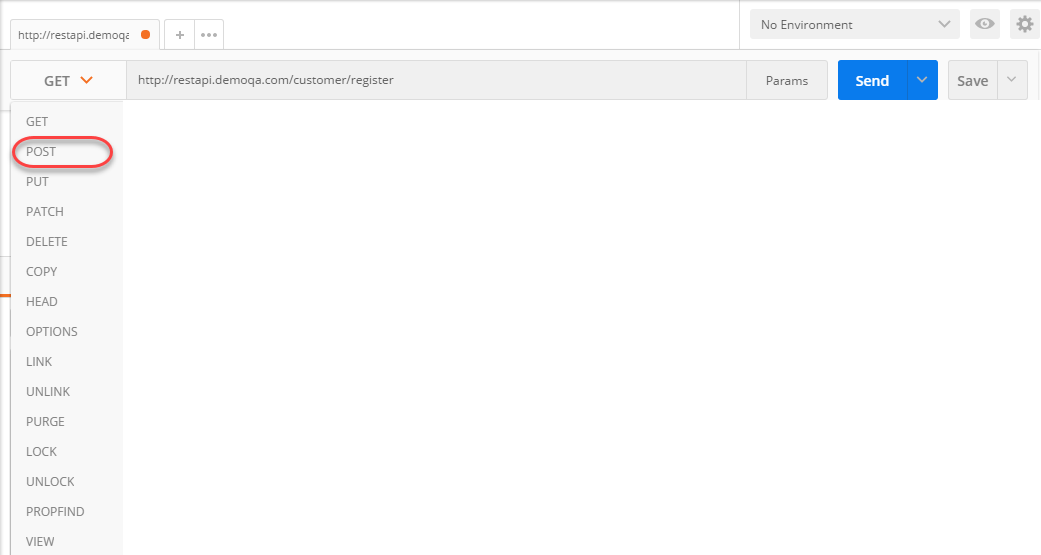
1. See the response below under the ***Body*** tab and focus on ***fault error***.



It means that the method type we used is not valid and another method type is expected. So we will try to change that and see if we get the correct response.

##### ***POST Request without Body***

1. Change the method type to ***POST*** and press ***SEND***



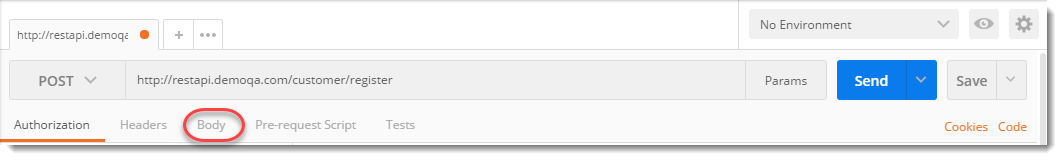
1. Now, look at the ***Response Body*** and ***Response Status code***.



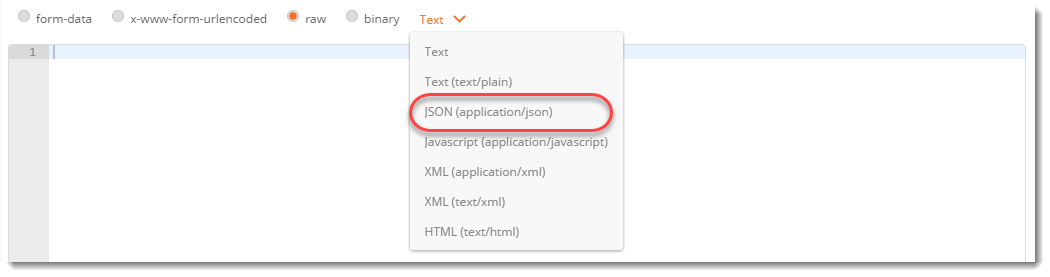
***Fault Invalid Post Request*** means that the ***post data that we entered is not valid***. Recall that we add the information inside the body of the request, so we need to enter something into the request body and see if that format matches the format expected. Also, you can see the status code which says ***400 BAD Request***. It means that the request parameters are not matching the server parameters to get a response.

##### ***Post Request in Postman***

1. Now let us add a ***Request Body*** to our POST request. Every Endpoint will be documented with what kind of Method type and the format of the body that it expects. Let us see what body this request expects and how to add it. For that click on the ***Body*** tab.



1. Click on raw and select format type as ***JSON***, since we have to send the incorrect format that the server expects.



1. This endpoint expects a ***Json*** body which contains the details of the new user. Below is a sample ***Json*** body. Copy and Paste the following in the body tab of Postman.

{

\* “FirstName”: “value”\*

\* “LastName : “value”,\*

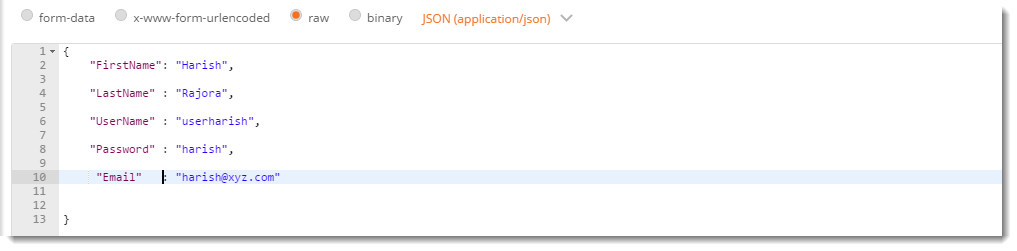
“UserName : “value”,

“Password”: “value”,

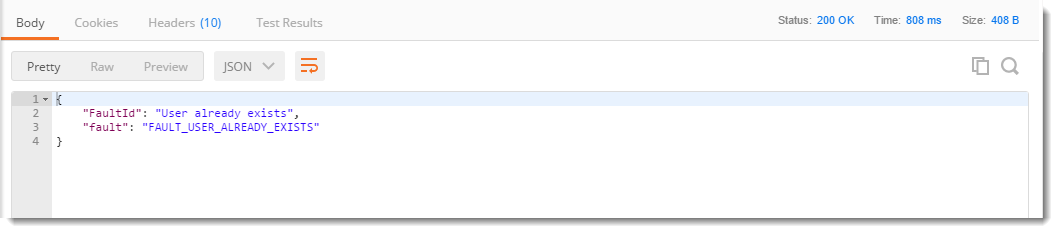
“Email”: “Value”

}

Change the attribute ***value*** to any value you want (take reference from the below image).

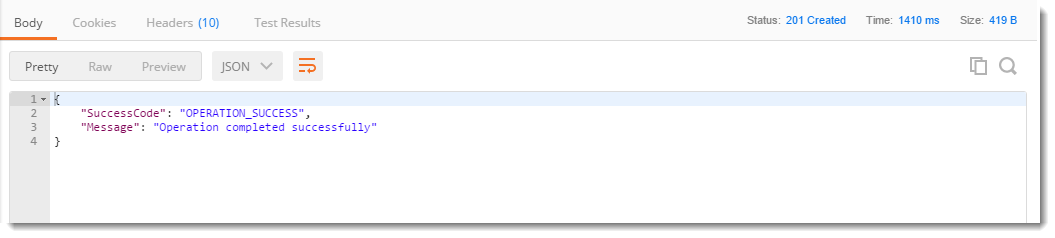


1. Press ***Send*** and see the Response Body and Response Status.



The error ***Fault User Already Exits*** means that in the database, a similar entry has already been created by you or anyone else earlier. Whereas if you see that the Response Status is ***200 OK,*** which means that server accepted the request and sent back a successful response. We can also infer from this that the response body was correct and the server was able to interpret the response body. Now in this API Request, Email and Username should be unique. So you can change those values (anyone will also work).

If the values are unique, you will get this response



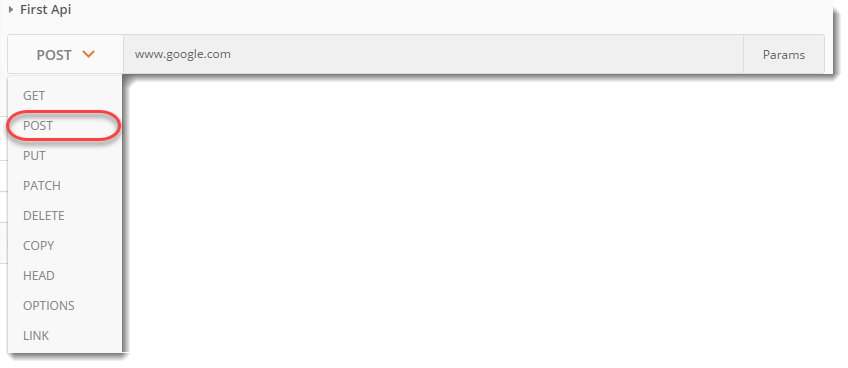
***Operation completed successfully*** means that your entry has been ***created successfully in the database.***

So, by this example, it is clear that whenever we need to send a POST request, it should be accompanied by the Body. The body should be in the correct format and with the correct keys to get a correct response from the server. Now, we will learn in detail about every feature of Post request in Postman.

## Different ways to send the data in a POST Request in Postman

As we discussed earlier, sending a POST request means sending a request with the data wrapped inside the body of the request. There can be different types of data and similarly, there are different ways of sending data. As you will follow these steps, you will learn in detail about it.

1. Select the method request type as ***POST*** in the builder as shown.



As soon as you select the POST request type in Postman you will see that the option Body is enabled which has different options to send the data inside the body. These options are:

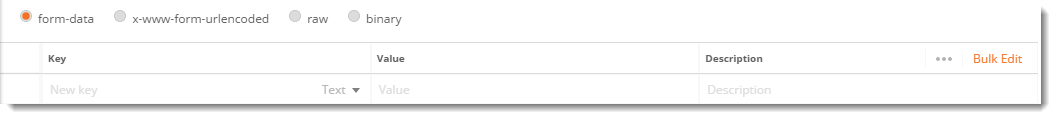
* ***Form-data***
* ***X-www-form-urlencoded***
* ***Raw***
* ***Binary***



### *****Form Data*****

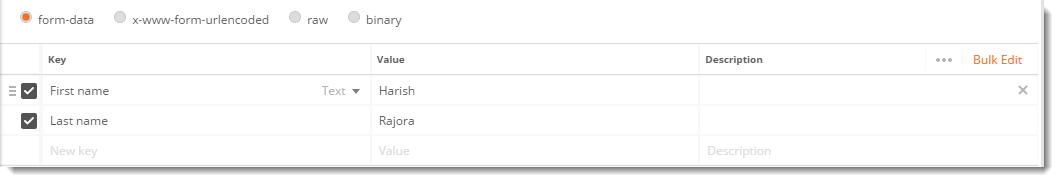
Form data as the name suggests is used to send the data that you are wrapping inside the form like the details you enter when you fill out a form. These details are sent by writing them as ***KEY-VALUE*** pairs where the key is the "***name***" of the entry you are sending and value is its ***value***. The following steps will make it clear.

1.Select ***form-data***



1. Add the following KEY-VALUE pair

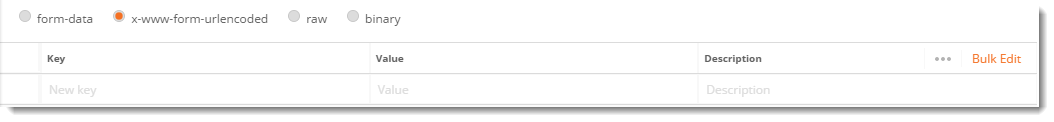
* ***First name***: Harish
* ***Last name***:  Rajora



Here, the first name in the field of some form (text field here) that is needed to be entered and ***Harish*** is its value i.e. the value the user has entered. Similarly goes for Last name id.

### *****x-www-form-urlencoded*****

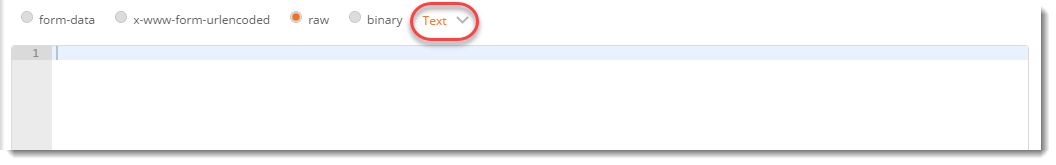
Form data and ***x-www-form-urlencoded*** are very similar. They both are used for almost the same purposes. But the difference between the form data and ***x-www-form-urlencoded*** is that the URL will be encoded when sent through ***x-www-form-urlencoded***. Encoded means the data which is sent will be encoded to different characters so that it is unrecognizable even if it is under attack.



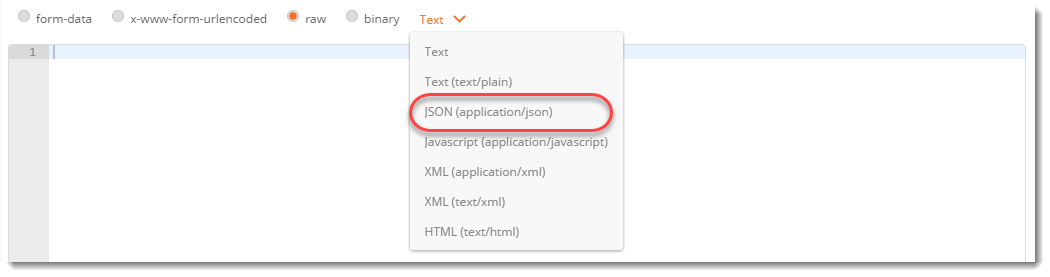
### *****Raw*****

Raw is the most used part or option while sending the body in the POST method. It is important from the point of view of Postman. Raw means the body message is shown as a stream of bits representing the request body. These bits will be interpreted as a string server.

1. Click on the dropdown besides ***binary*** and there can be seen all the options in which you can send the request



1. Click on ***JSON(application/json)***



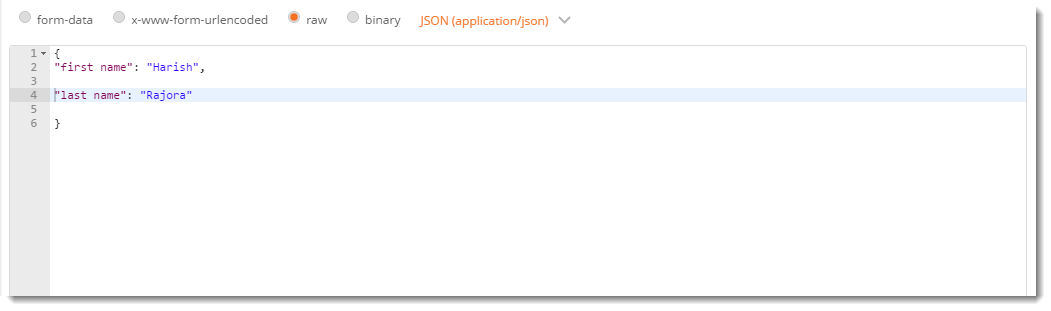
1. In the editor below copy and paste this

{

"first name": "Harish",

"last name": "Rajora"

}



This is the same data that was sent previously with form-data but instead it is now sent with the JSON format.

### *****Binary*****

Binary is designed to send the information in a format that cannot be entered manually. Since everything in a computer is converted to binary, we use these options which cannot be written manually such as an image, a file, etc. To use this option

1. Click on ***binary***, a CHOOSE FILES option will be available

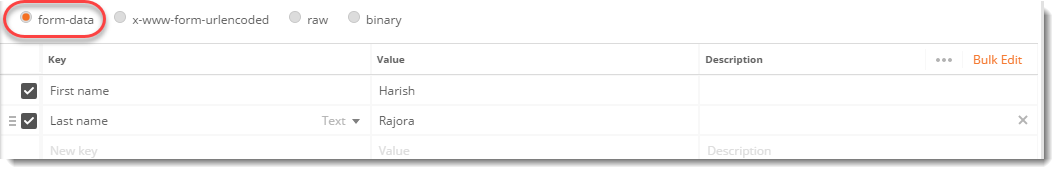


1. Choose any file, such as an image file.

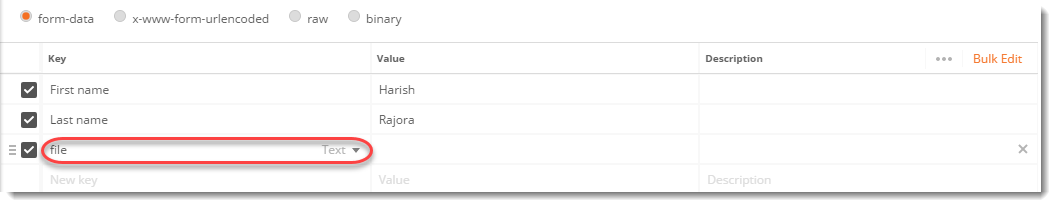


***Note***: If you wish to send to the server some data along with the file then it can also be done in the form-data.

Click on the ***form-data***



Enter ***file*** as a key



and you will see a hidden drop-down that says Text as default. You can choose any file format and then select the file from the system.

Always remember what your server is expecting. You cannot send another format than what your server expects, otherwise, there will be no response or incorrect response which can obviously be seen by the status code of the response. So now, we have learned about the POST method and how to use it in Postman. We will move on to the next tutorial now which is ***Collections***.

***What is Authorization?***

The meaning of authorization can be seen as a question which is, ***are we eligible to access a***[***secured resource***](https://toolsqa.com/postman/postman-navigation/)***on the Server?*** If the answer is yes, then in technical terms we can say that we are ***Authorized*** to access the resource. If the answer is No, we can say that we are not ***Authorized*** to access the resource. *For example, let us say you have added your and your sister's fingerprint to your phone. You and your sister can open the same mobile phone, which means only you and your sister are authorized to open the phone and see the data*. Similarly, while there could be many ***APIs*** in a company or a project. It is not necessary that everyone will have access to all the APIs. Only authorized people can access the secured APIs.

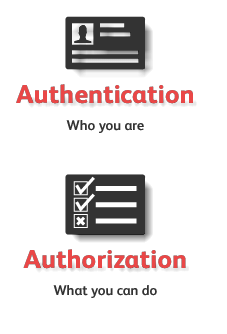
***Authorization Vs Authentication***

***Authorization*** and ***Authentication*** are two closely related terms. These two terms can also be confusing at first. In this section, we will clear the confusion about these two terms.

***Authentication*** is a process of presenting your credentials to the system and the system validating your credentials. These credentials tell the system about who you are. This enables the system to ensure and confirm a user’s identity. *Here system can be anything, it can be a computer, phone, bank or any physical office premises.*

Whereas ***Authorization*** is a process of allowing or denying someone from accessing something, once ***Authentication*** is done. *So in layman terms****Authentication****tells who you are while****Authorization****tells what you can do*.

*When a person accesses the server with the****key/password****, the server checks whether the person is available in the directory and is also associated with the same****key/password****. If it is, you are good to go (Authentication). If you have access to the resource, then you will be granted access to the resource (Authorized)*.



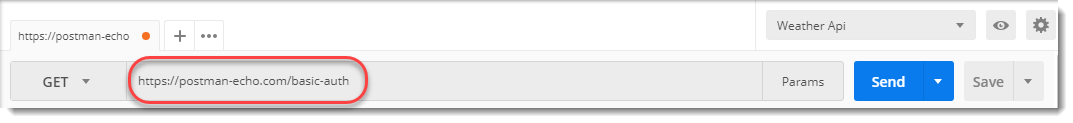
We will see the following short example to tell you how does a server rejects an unauthorized person.

**Authorization using Postman**

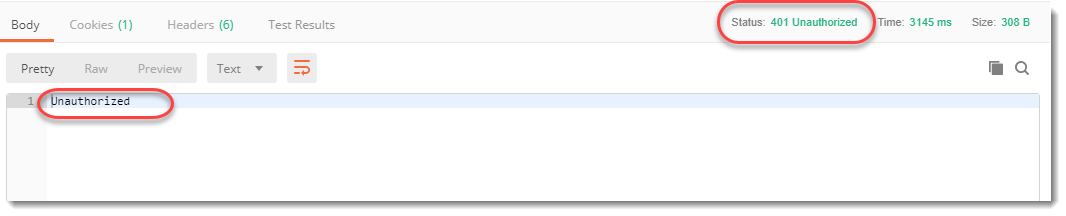
***Checking Authorization***

For this chapter, we will be using the endpoint ***https://postman-echo.com/basic-auth***

1. Create a *GET* request and enter the endpoint as ***https://postman-echo.com/basic-auth***



Press ***send*** and look at the response



***Note****: The status code is****401****which corresponds to unauthorized access and the response message says****Unauthorized****.*

The status code and response from the server indicate that we are not authorized to access the *API* we are trying to access(*See*[***Responses tutorial***](https://toolsqa.com/postman/response-in-postman/)*to learn more*\*\*). Later in the tutorial, we will try to access the same *API* using the credentials as we discussed in the last section.

***Need for Authorization***

In the last section, we discussed that a resource owner does not allow access to the resources to everyone in the company. This is because it can lead to possible security breaches. *If I allow an intern to access my database APIs then inadvertently he can change the data and that data can be lost forever which can come as a cost to the company*. There are numerous reasons possible for the same. Maybe a person changes the data for money or a person can leak the data to another company. Authorization plays a very important role in deciding the accesses and tightening the security. Let us see the different types of Authentication available to us.

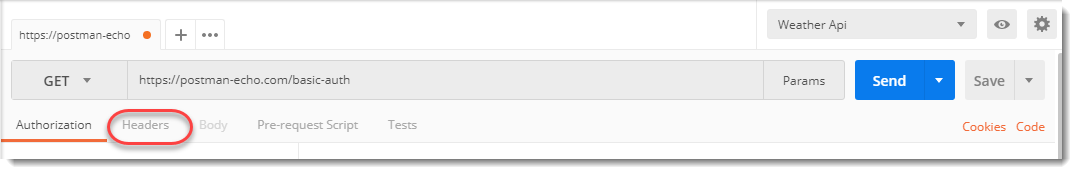
***Basic Access Authentication / HTTP Basic Authentication***

A ***Basic Access Authentication*** is the most simple and basic type of authorization available. It requires just a ***username*** and ***password*** for checking the authorization of any person (*That is why we say basic access****authentication***). The username and password are sent as header values in the ***Authorization header***. While using basic authentication we add the word Basic before entering the username and password. ***These username and password values should be encoded with Base64 otherwise the server won't be able to recognize it***. We will follow these steps to check whether we can access the same API we used above or not

**Checking authorization using credentials**

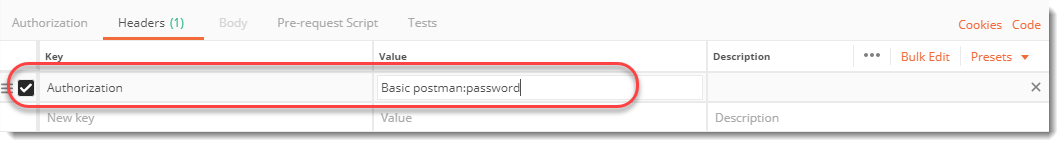
1.Enter the endpoint ***https://postman-echo.com/basic-auth in*** *GET request*.

1. Go to ***Headers***



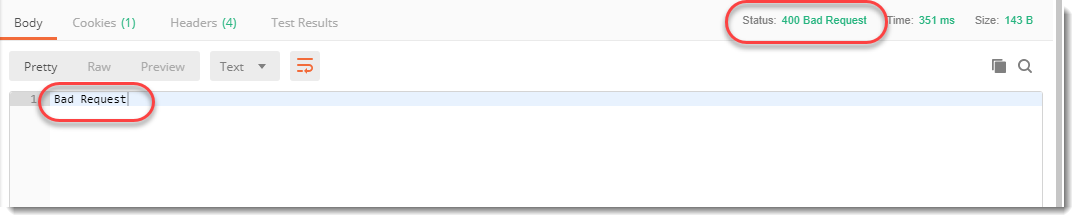
1. Enter the following key-value pairs in Header

*Authorization:    Basic postman: password*



***Note****: We are using the username as****postman****and password as****password***

1. Press ***Send*** and see the response box and status code.



It still says ***400***, ***Bad Request***.(*This part we have already covered in the*[***Responses Chapter***](https://toolsqa.com/postman/response-in-postman/)*under Status codes and their meaning*). Can you guess why?\*\*\* If you remember what we learned in the last section, a basic access authentication requires a username and password to be encoded in Base64 but here we just sent the username and password in plain text. As a result, the server returned a 400, Bad Request status code. Before we move forward it will be beneficial to understand what ***Base64*** encoding is.

***What is Base64 encoding?***

Encoding is used in authentication because we don't want our data to be transmitted directly over the network. There are numerous reasons for that. Network scanners can read your Request and retrieve the Username and Password sent without encoding. Also, bits and bytes transmitted directly can be considered as inbuilt command bits by the modem or other equipment in the network chain. *For example, if there is an inbuilt command of****0101101010****which means reset to the modem then while transmitting we have may want to get a data sequence of 001101010****0101101010****11020. Here the modem might interpret it as a reset command and will reset itself*. In order to avoid such problems, it is beneficial to encode the data.

We use **base64** particularly because it transmits the data into a textual form and sends it in an easier form such as HTML form data. We use **Base64** particularly because we can rely on the same 64 characters in any encoding language that we use. Although we can use higher base encoding methods also but they are very hard to convert and transmit which wastes time unnecessarily.

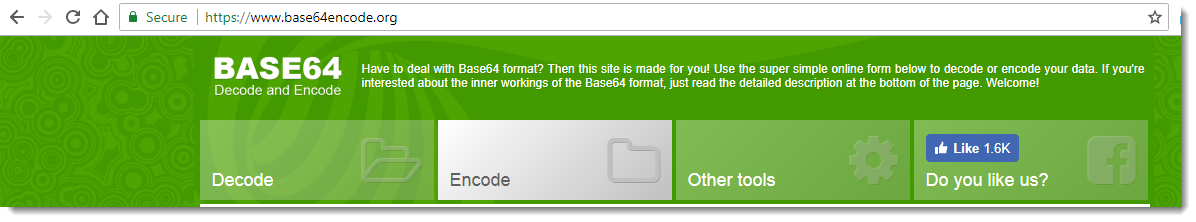
Coming back to the original problem of sending a **Base64** encoded string in **Authorization** header. We have two ways in front of us for creating a **Base64** encoded string:

* ***Through third party website***
* ***Through Postman***

We will see both of the options one by one. For now, follow the steps for accessing the API by decoding from a third-party website.

**Authenticating by encoding through a third party website**

1. Go to ***https://www.base64encode.org/***



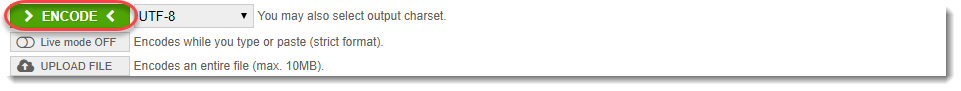
***Note****: There are thousands of websites available for the same purpose. You can use anyone just make sure you encode to the same value as us. Also, we are using****Microsoft Edge****as the browser, though it should not make any difference*.

1. Paste in the box the following values

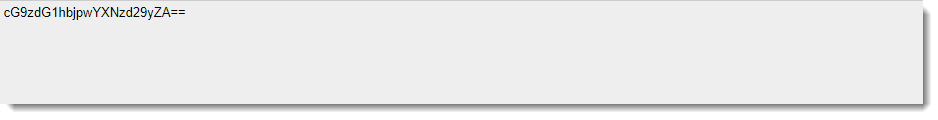
*postman: password*



3.Press ***Encode***.

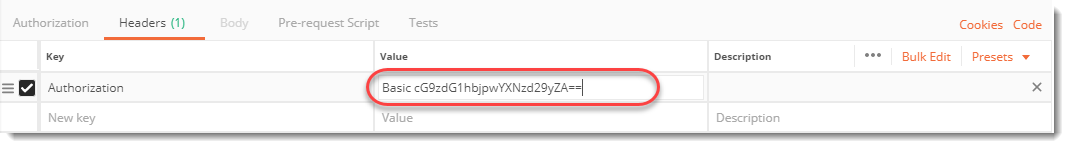


1. Copy the encoded text.

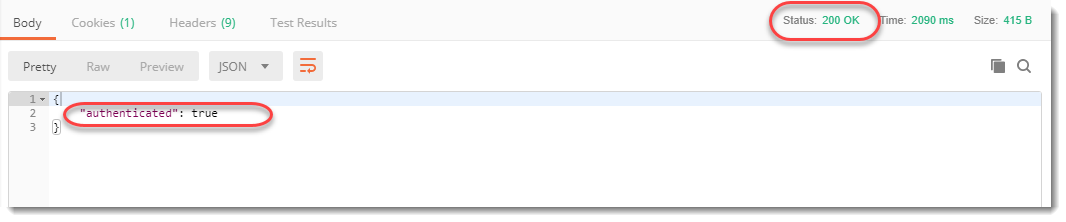


***Note****: Do not use space in between any two texts or symbols.****postman: password****will encode to a different value while****postman: password****will encode to a different one. Needless to say, both will be considered wrong. Use****postman:password****only*.

1. Go to the postman app and instead of ***postman: password***, paste the encoded value



1. Press ***send*** and see the value of the response box and the status code.

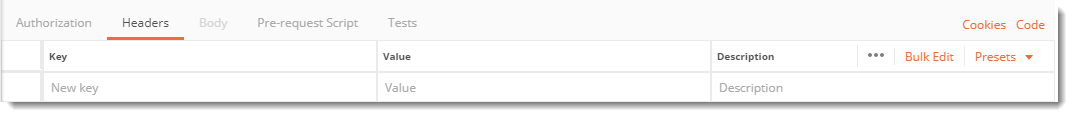


***200 OK, authenticated*** means we have provided correct credentials and now we are authorized to access the data.

**Authenticating by encoding through Postman**

Instead of going to a third-party website, we will try to encode using Postman.

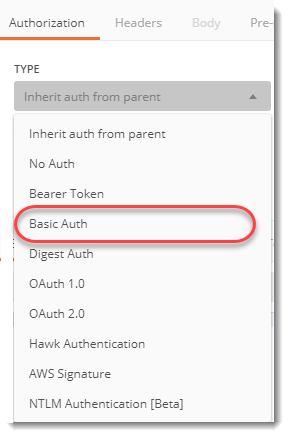
1. Erase the key-value pair that we entered earlier so that it now has no values.



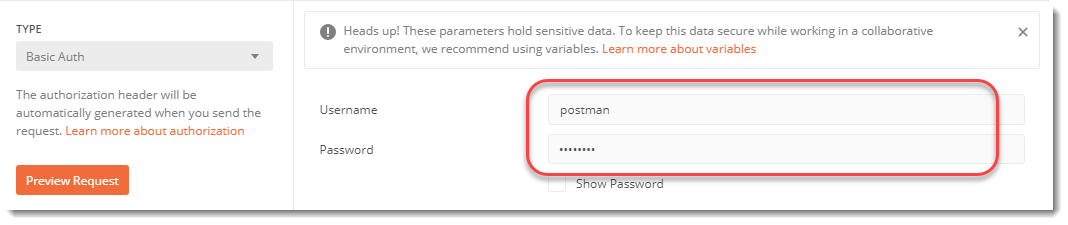
1. Go to the ***authorization*** tab



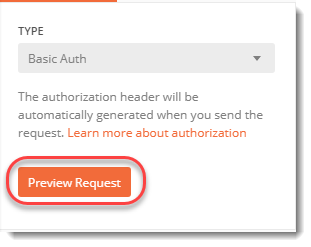
3.Select ***Basic Auth*** in the ***Type*** dropdown



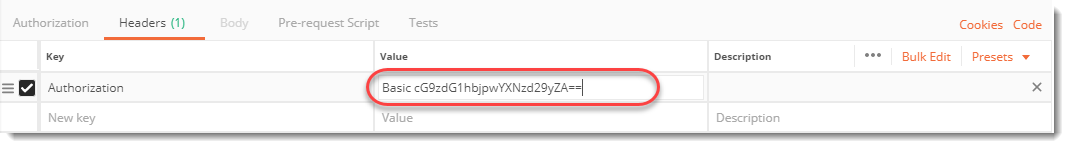
4.Enter username as ***postman*** and ***password*** as password



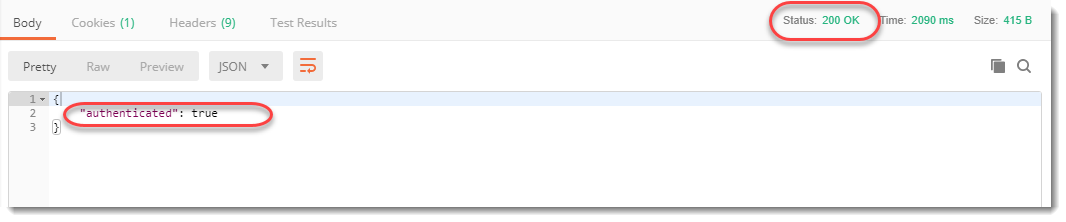
5.Press ***Preview Request***



1. Go to ***Header*** and see that Postman has converted the username and password for you.

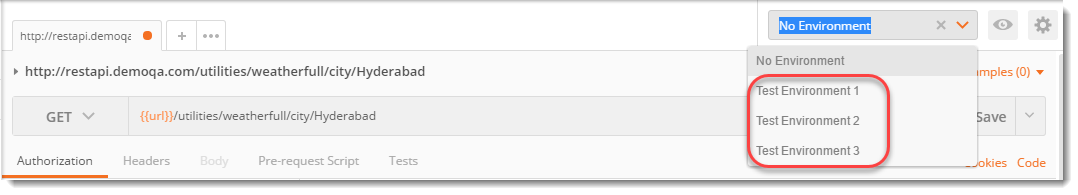


7.Press ***send*** and voila! we are authenticated.



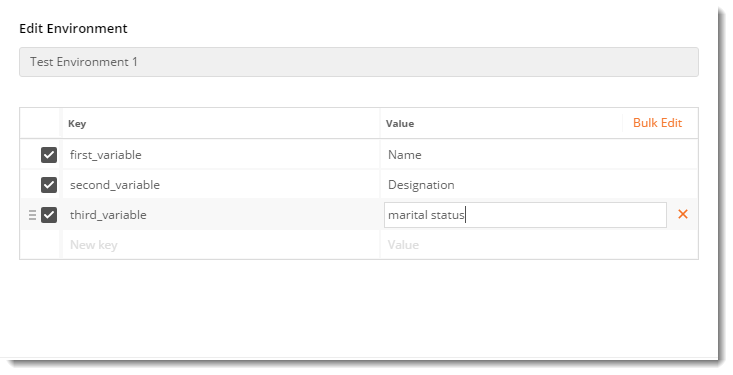
***What is an Environment in Postman?***

An environment in Postman is a ***set of key-value pairs***. An environment helps us to differentiate between the requests. When we create an environment inside Postman, we can change the value of the key value pairs and the changes are reflected in our requests. An environment just provides boundaries to variables. When we create different environment we can make track of all the variables and how to use them inside our requests. There can be many variables inside one environment. At once, we can work only in one environment although we can create any number of environments in Postman. The below screenshot shows three environments that we created.



***What is an Environment Variable in Postman?***

A variable in the Postman is same as in any programming language. A variable is an entity whose value can be changed. ***The key part in the key-value set in environment is called variable***. This variable can have any value and in place of the ***key*** we can use the variable name in every request. This will be clear with an example shown below and steps shown thereafter.



The above image shows three variables inside environment ***Test Environment 1***

**Environment Variables in Postman**

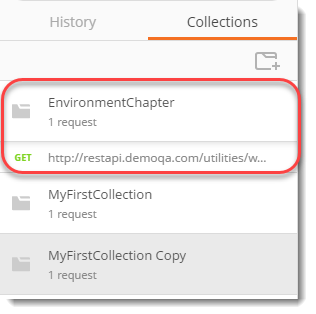
Now, we will use Postman to create an Environment and Environment Variable and it is very easy to do that but it has three steps involved in the process:

1. ***Create an Environment***
2. ***Create Environment Variables***
3. ***Use an Environment Variable in the Request***

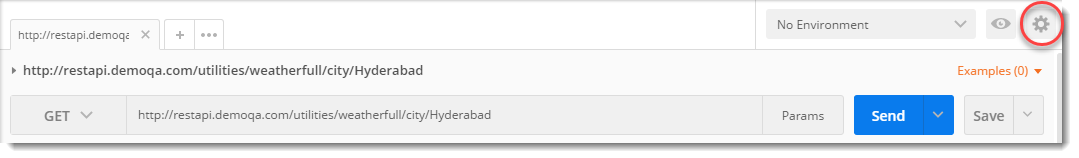
***Step 1: How to Create an Environment in Postman***

1.[***Create a new Collection***](https://toolsqa.com/postman/collections-in-postman/) and name it as ***EnvironmentChapter.***

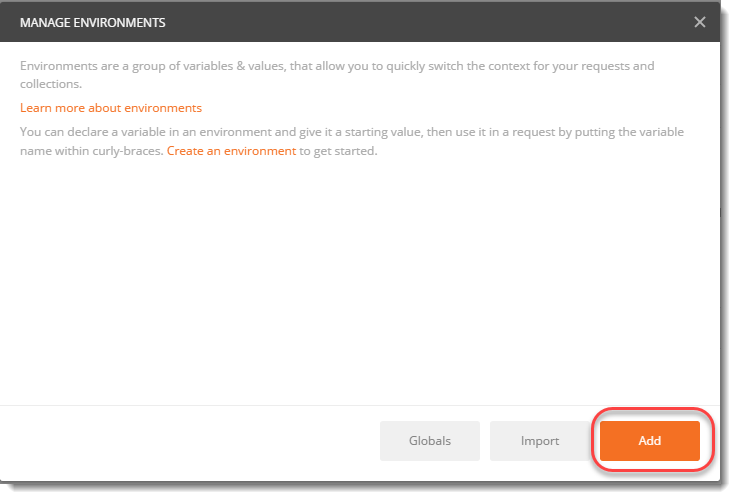
1. Add ***Weather Api Request*** in the collection used in the [***Get Request***](https://toolsqa.com/postman/get-request-in-postman/) chapter.



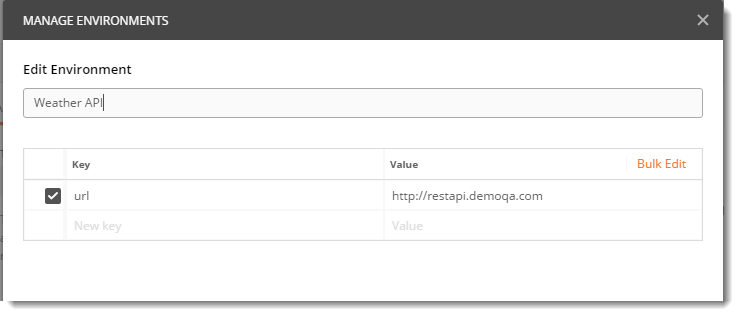
1. Click on the ***gear*** icon which says Manage Environment.



1. Click on ***Add.***



1. Name the environment as ***Weather API***

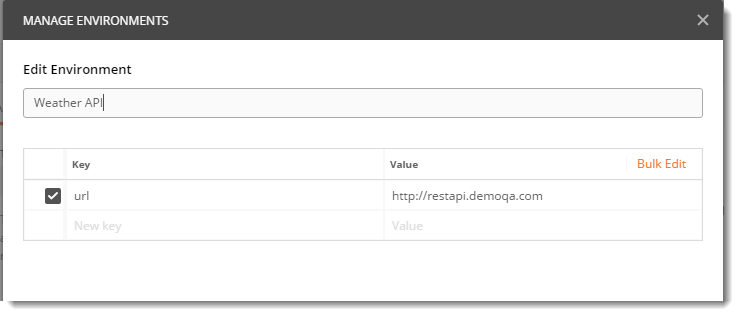


***Step 2: How to Create Environment Variables in Postman***

1.Now in the same window, enter the following key-value pair. Where ***Key is the name of the variable and Value is the text string.***

***Key****: URL*

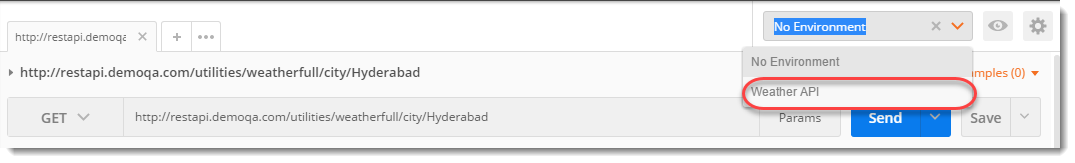
***Value****: http://restapi.demoqa.com*



Click on ***Add*** and close the panel.

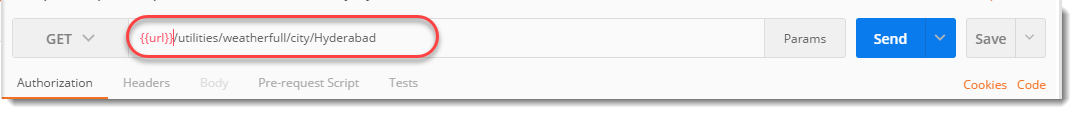
***Step 3: How to Use Environment Variables in Postman***

1. Select the dropdown which says No Environment and select ***Weather API*** environment in that.



Now we can access all the variables of this environment.

1. In the address bar change  ***http://restapi.demoqa.com*** to ***{{url}}***



1. Click on ***Send***.

Now, we have created an environment and used a variable called ***URL*** here. This variable can be now used instead of the actual URL. You can see the response which is same as before we were using the full URL.

So, if by any chance the URL changes, we can just go to environment and change the URL value and it will be reflected in every request.

***NOTE****: Remember to save the request by clicking****Save****button. In the future chapters, we will use this modified request only.*

**Scope of Variables in Postman**

A scope of anything is the ***boundary in which that thing can be accessed and perform***. *For example, if you are an engineer and do not have a passport, your scope is limited to India since you cannot go outside. While having a passport changes your scope to the world.* Similarly, variables in Postman have two scopes

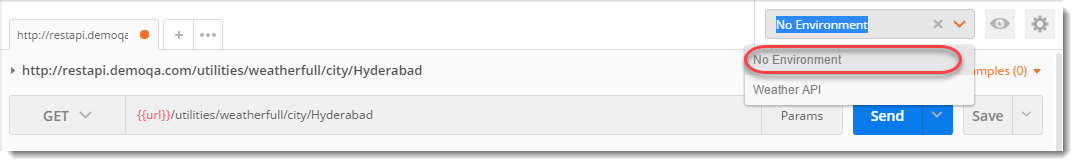
* ***Local Scope***
* ***Global Scope***

***Local Scope***

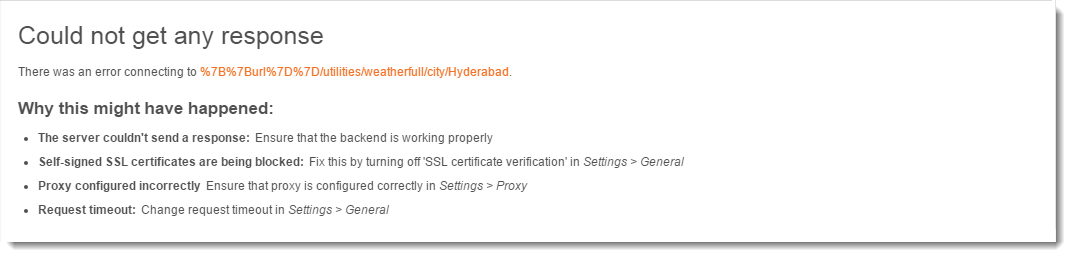
***Local Scope Variables*** can only work inside the environment in which it was created. ***Changing the environment will stop the access to that variable and we will encounter an error***.

The variable URL that we just created above is the local variable because it has a scope only till the environment Weather API. In the following steps we will explore the limitations of local variable by accessing local variable in other environment, where it is not present.

1.Go to the dropdown where we selected ***Weather API*** and select any other value (*if you have*) or ***No Environment***.



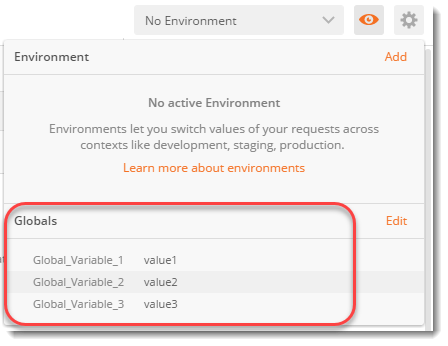
1. Click ***Send***.



*This error occured because Postman does not know about****URL****variable because we have changed the environment. Therefore, URL is a local variable having scope only till the****Weather API****environment.*

***Global Scope***

***Global Scope Variables*** can work outside the environment also. They are global and it does not matter which environment is selected. In the following image you can see three global variables by clicking the ***Eye*** icon.



**Global Variables in Postman**

Now, we will use Postman to create ***Global Variables.***

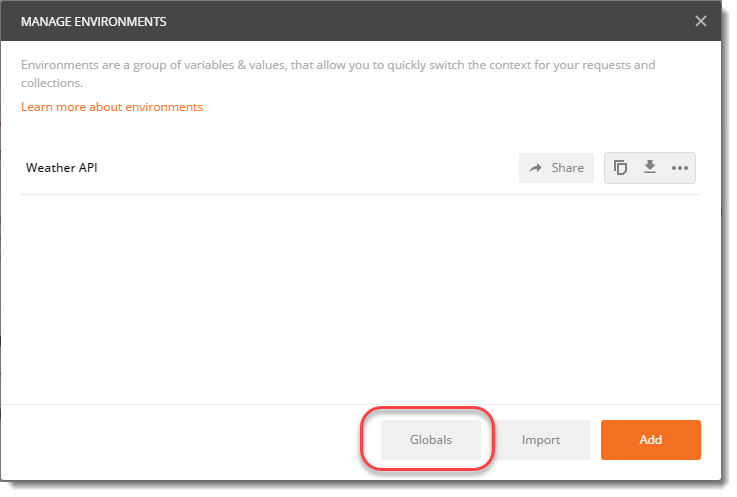
***Create an Environment:****Just because global variables are not associated with any particular environment, there is no need to create an environment for global variables.*

1. ***Create Global Variables***
2. ***Use Global Variable in the Request***

***Step 1: How to create a Global Variable in Postman***

1.Go to the same ***gear icon*** to open the environment panel which we did at the time of creating *Local Variable.*

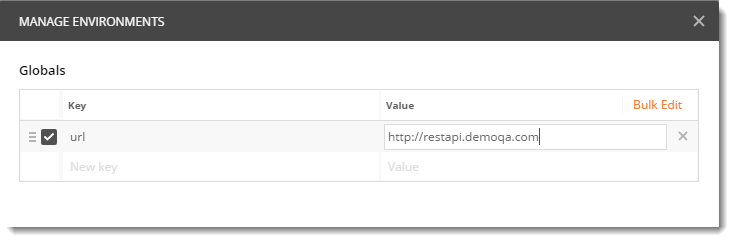
1. Select ***Globals*** to add a global variable.



1. Add the following key-value pair

***Key****: URL*

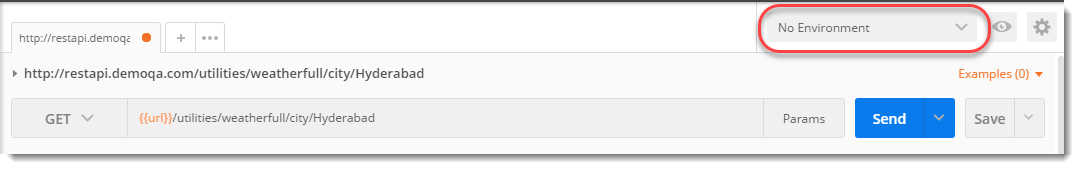
***Value****: http://restapi.demoqa.com*



1. Save and close the panel.

***Step 2: How to use Global Variable in Postman Request***

1.The request which we created above, just select the ***No environment*** from the environment dropdown.



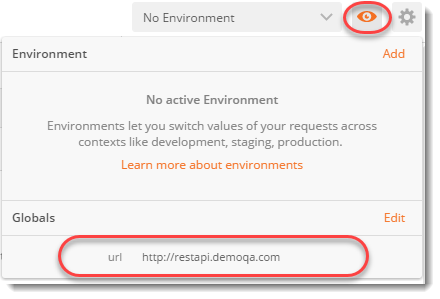
1. Press ***Send*** and now see the result.



It works now because we have created a global variable which can be used with every environment.

***NOTE****: Global scope cannot have duplicate/same names while variables having local scope can have the same name in different environments.*

For convenience Postman also has a feature which lets you see all the current variables and environment. Just click on ***Eye*** icon and it list down all the Environments and Global variables.



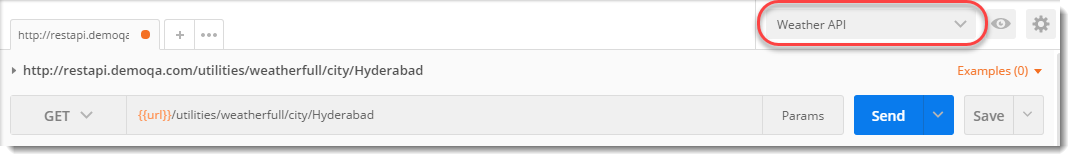
and you can see the global variable under ***Globals*** written. We have not selected any environment therefore there is no information about the environment. You can try it out yourself.

**Precedence in Variables**

As we discussed, two global variables cannot have same name while two local variables can have same name provided they are in different environments. But what if one local variable and one global variable has same name? *For example, you name a local variable ABC and a global variable ABC. Now when you select that respective environment both the variables will be activated. So, which will show its value?* This confusion is solved by precedence.

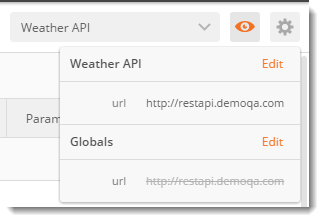
Precedence generally means priority. While two or more things strike together, the one with higher priority (*precedence*) is preferred. ***In Postman for same name of environment specific variable and global variable, environment specific variable or local variable has higher precedence***. It will overwrite the global one.

1.Now in the dropdown panel select ***Weather API*** instead of ***No Environment***



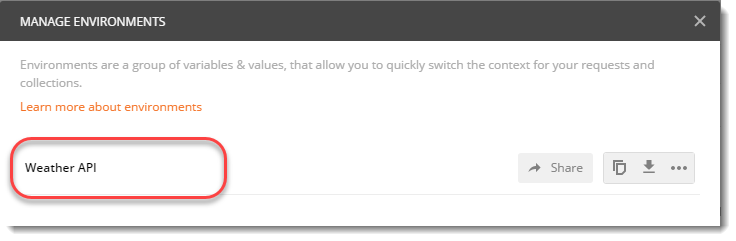
Now we have two variables of same name accessible. One in Weather API environment and one which is global.

1. Click on ***Eye*** icon to have a look

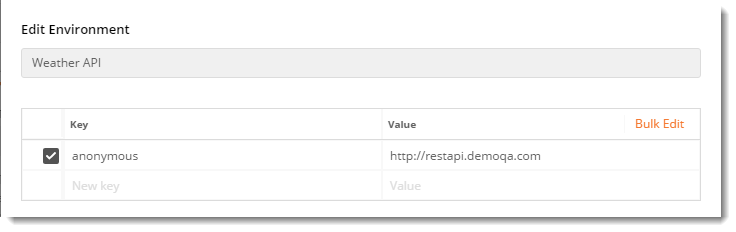


Here we have a problem, both the variables have same values. But if you look in the image above, ***global URL has been sliced off with a line***. This has happened because both the variables have the same name and the precedence will be given to the local variable so global variable will not be used.

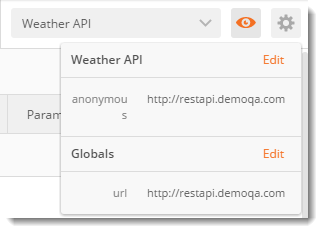
3.Go to the ***Manage Environment*** (*gear icon*) and click on ***Weather API*** environment



1. Change the ***url*** value to anything you like. Here we have changed it to ***anonymous***.



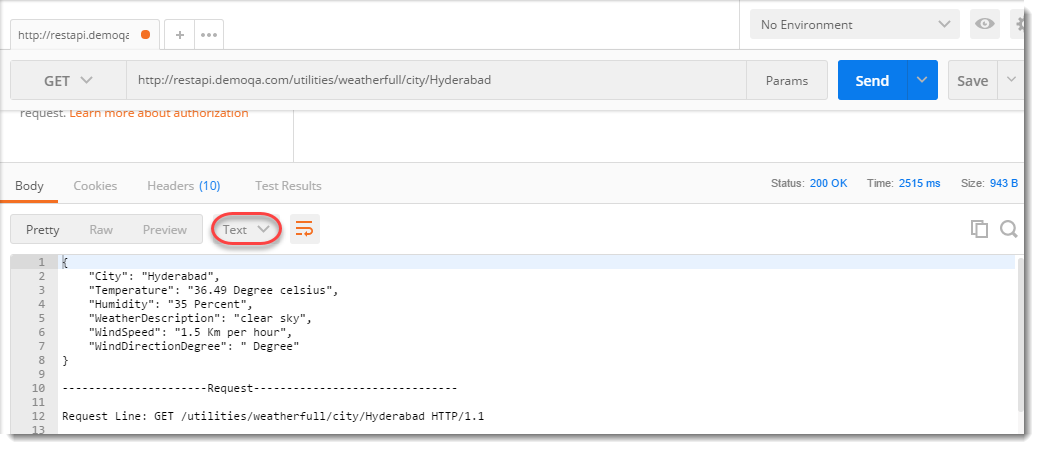
1. Close the panel and look at the current environments again by selecting the ***Eye*** icon



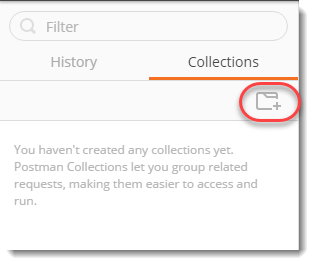
Both the variables are now accessible and can be used. If you press send now, you will get the correct response from global variable which you would get from local variable if they had same name. This is how precedence works.

### *****How to create and save collections in Postman?*****

1.First of all, hit the endpoint ***http://restapi.demoqa.com/utilities/weatherfull/city/Hyderabad*** and in the response box change the format to ***Text*** to see the result.



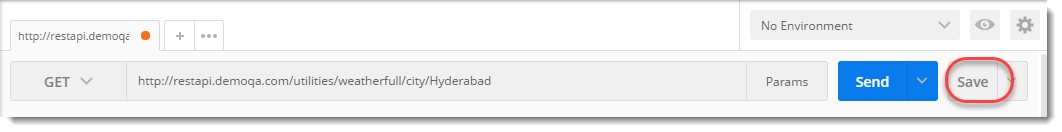
1. Now go to ***Collections*** Tab and press the icon that says, ***New Collection***.



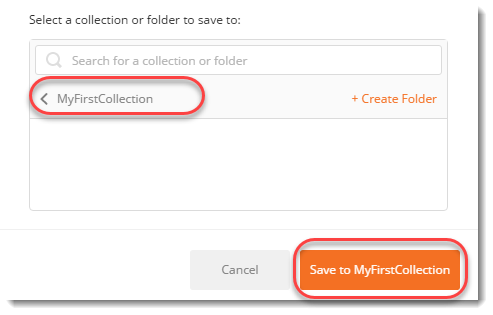
1. Write a name of your choice for your first collection and it's description. In the following image you can see I chose the name ***MyFirstCollection*** and a simple description.



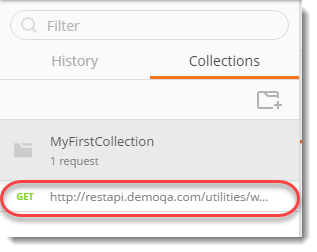
1. Press ***Create*** to create your first collection.
2. Now, you have your first collection created but it's empty as of now. Press ***Save*** button in the side of address bar.



1. Choose your collection name in the panel as shown in the image. Press ***Save***.



Now look at the collections tab, you will have the request saved below your collection name.

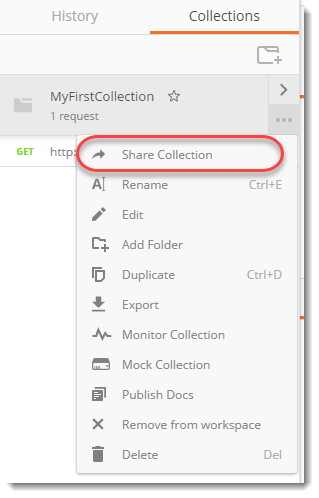


You can click on the ***Meatballs*** menu in the side of the collection name and you will see some options. We will talk about the following options in this tutorial.

* ***Share Collection***
* ***Export Collection***
* ***Add Folder to a Collection***
* ***Duplicate the Collection***
* ***Remove the collection from workspace***
* ***Delete the collection***
* ***Leave a comment on the collection***

### *****Share Collection*****

Share collection option is used to share the collection to other people such as your team members.



Sharing a collection is very important when you are working in a company or in a team. There are times when developers create a bunch of ***Postman Requests*** while developing APIs to test it. You can request developers to share their collections and you get benefits of it. Or even you create a bunch of ***Postman Requests***, save it to a ***Collections*** and share it with your team. It is not worth sharing every request one by one. Rather, we share our ***Collection*** as a whole to the team members or anyone to whom we want to share through different methods listed below.

For example, if you are a tester and find out some bug, you can save your steps to reproduce in a ***Collection***, and attach the same to the bug for the developer as a proof and to reproduce the issue. For using this option ***you must be signed into postman***. After signing into Postman you get two options to share the collection.

* ***Through workspace***
* ***Sharing through link***

#### **Workspace**

A workspace is a collaborative environment for a group of users to develop and test APIs. In simpler terms, workspace is the project in which you work. A setting done in a workspace remains in the same workspace. It is like a shell in which you work. There has to be a workspace to work in Postman, you cannot work isolated. There are two types of workspaces inside Postman

1. ***Team Workspace*** : To use the option of Team workspace your company or you must have purchased the Postman Pro version or else this feature won't work. In Team workspace option, your team can test API simultaneously on one workspace and anyone can edit and update (with permissions). This comes handy when teams are located in different locations or different buildings also. Once edited, everyone's API is updated automatically. There can be any number of Team workspaces.
2. ***Personal Workspace*** : A personal workspace is similar to the team workspace but the difference being this workspace is completely personal to the user who created it. Postman will have no contribution in updating it at any machine, if present.Collections present in the personal workspace is shared through link which is detailed below.

Individuals can organize their work in personal workspaces and teams can collaborate in team workspaces. When you start Postman, you are already inside personal workspace as shown.

My_Workspace_Header

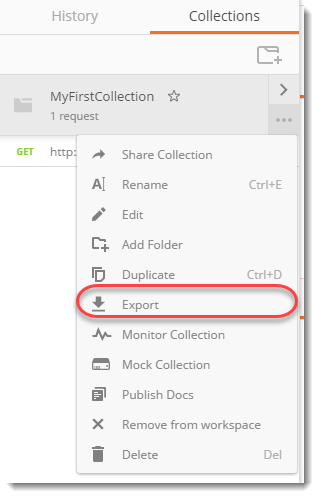
You can create unlimited personal workspace which will be personal to your machine and can test any number of API.

#### **Share Collection through a link**

Second option is by creating a link and sending it to the people of your team. This link will be of ***Postman Cloud***. So, your collection will be first uploaded to ***Postman cloud*** and then anyone can access it through that link. ***But, this feature is not recommended while working in a company sometimes because the APIs are personal and companies would not want them to be in other clouds which is not safe***.

### *****Export Collections in Postman*****

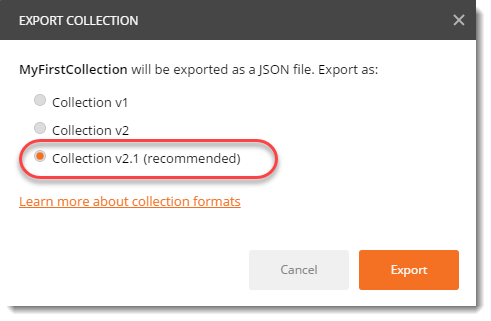
Export feature is used to export the collection as a whole by saving it to your computer and which can be later share with the team over the Email in a zip file. Or it can also be shared over a network sharing.



Follow these steps to learn about exporting the collection.

1.Click on ***export*** as shown in the above image.

2.You will see two or three options (depending on your version of Postman). For this tutorial we are using Postman ver 6.0.10. We will choose the option ***Collection v2.1*** and press ***Export***.

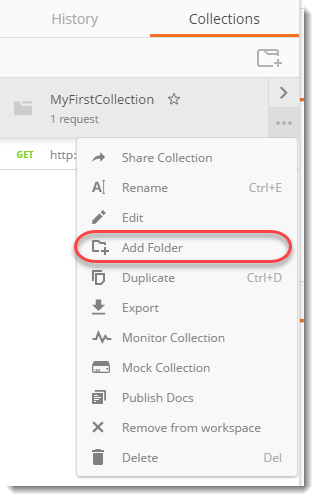


This will open the box to save the collection in JSON format on your computer wherever you want. Then this JSON file can be shared with your teammates by any means just like a file.

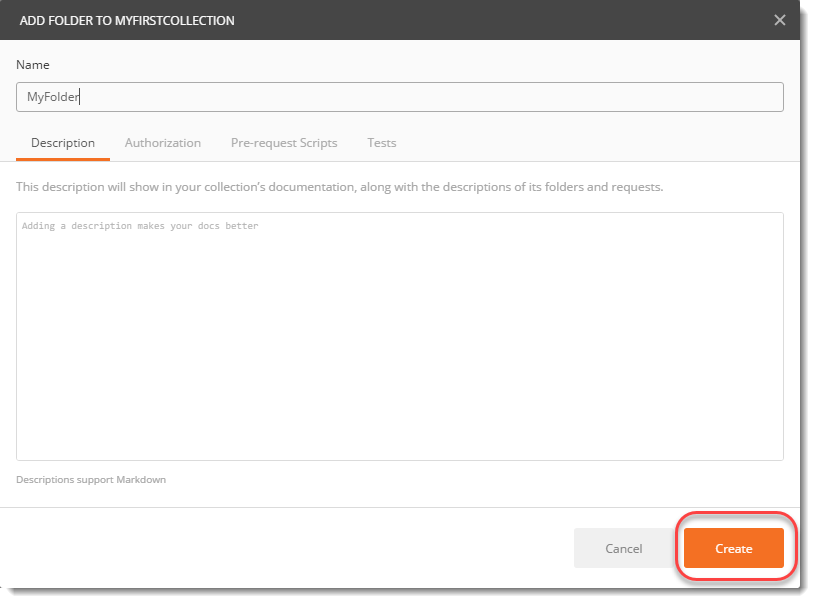
### *****Add Folder*****

Collections also let you make folders inside your collection and then save the requests inside your folders. This can further help you to sub-categorize the requests. For example, in the previous chapters we took the example of movies folder to explain collection where you can store all your movies. Making a folder is like making another folder inside Movies folder, such as "***English Movies***" which will contain all the English language movies, but they are still movies. Similarly, here we can make folders inside the collection and store our requests.

1.Choose the option ***Add Folder.***

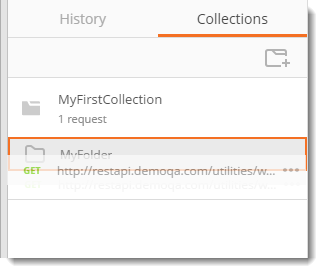


1. Name your folder and press ***Create***.



Now you have your folder created but as of now it is empty.

3.Drag the request and drop it on the folder name.

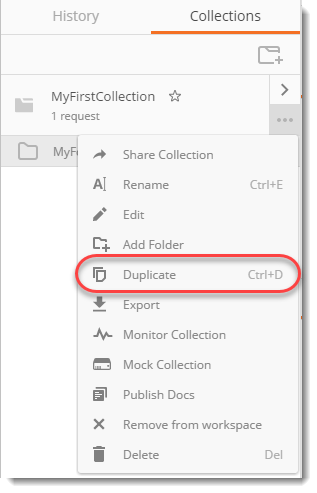


This will move your request to your folder and you are all set.

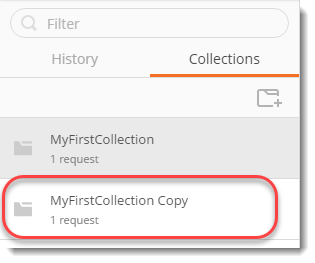
### *****Duplicate*****

The option ***Duplicate*** as the name suggests, duplicates the collection into a new collection. It means when you click on Duplicate, you make another copy of the same collection in the workspace.

1.Click on ***Duplicate***.

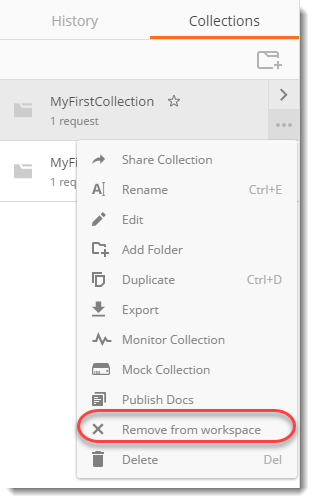


2.You can see the copy in the sidebar underneath the original collection.



### *****Remove From Workspace*****

Remove from workspace option deletes the collection from the workspace that you are working on.



If it is a personal workspace then you can delete it by choosing the option whereas if it is a team workspace then you need to have permission for deleting any collection or request from the workspace.

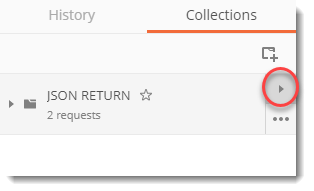
### *****Delete*****

Delete and remove from workspace option might seem similar but they differ in just one aspect. ***Remove from workspace deletes the collection from the current workspace but delete option will delete the collection or request from all the workspaces wherever it is present***. Needless to say, you need permissions for this.

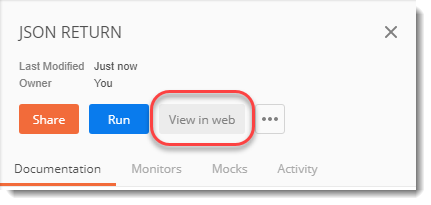
### *****Leaving a comment on the collections in Postman*****

It is quite obvious that when we work in teams, our workspace faces a lot of edit. This can be anything from editing a simple request to editing one test in that request. Now, since you need to apprise everyone what you did, Postman brings you the comment feature. Through this feature you can leave comments in the collection so that everyone can know what you just uploaded in a chronological order.

For this press the ***arrow button*** beside the collection name



Select ***View in Web*** from the options



This will open the collection in your browser. This can be seen by everyone working in the same workspace. Select ***comments*** on this page.



This will show you the popup to enter the comments.

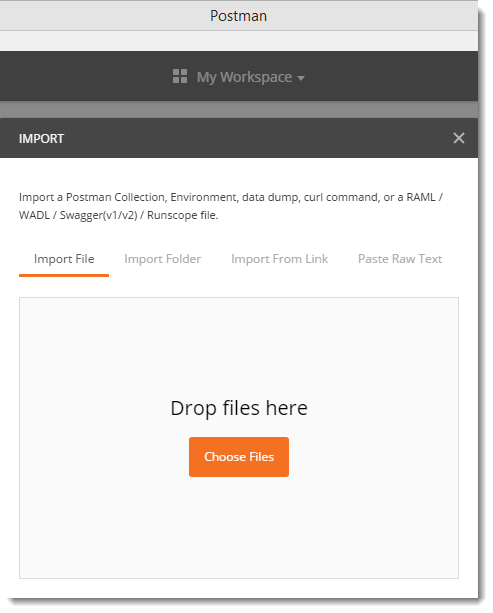
We will learn in the next section how to import collection in Postman.

### How to Import Collections in Postman

If we can export our collection then needless to say it must be imported in some other Postman. So, like export we have another option import but that does not reside inside the meatball menu. The import collection is a part of header as shown below.

Import_Header

Importing a collection is easy in Postman. When you click on import you will see the import panel in front of you.



Here, as you can see there are different methods that can be used to import a collection. These methods are

* ***Importing through drag and drop method***
* ***Importing the folder***
* ***Importing through link***
* ***Pasting the raw text***

#### **Importing through drag and drop**

Importing through drag and drop is pretty simple. It works the same as the upload feature in many websites such as Google Drive. If you have a collection file in you system just click on that file and drag it to this panel and release the mouse (or drop the file). This way the file/collection will be automatically uploaded in your Postman.

#### **Importing the folder**

Importing the folder is same as the previous option just the difference being, in the previous one we were uploading single collection but in this one we can upload multiple collections all at once. Just make a folder inside your system and paste or export many collections in it. When you import the same folder in your Postman, you will see all the Collections being uploaded. If you already have any of the collection from that folder, you will be asked to replace it or form a copy of it.

#### **Importing through link**

As we discussed in the previous tutorials, when we work in teams, we often use the Postman cloud or Team workspace which provide us the link to the collection so that everyone can use it without any problem. So importing through a link is the same feature. We can provide link of the same collection in the box and the collection will be imported.

#### **Pasting the raw text**

In the above image, the first line in the panel tells us what files can be imported in Postman. There are many files apart from a collection like curl or RAML etc. These files can be imported through raw text coding like in curl file. Although this feature is out of the scope of this course, therefore we won't be discussing it in detail.

**TEST in Postman**

A ***TEST*** in Postman is similar to the definition of the test in general. In Postman, we test our request for whatever we need to know about the request. *For example, if I need to know whether my request gave a status code of 201 or not.* This can be managed in Postman. Also, there is no bound on running the test in Postman on one request. ***One request can be passed through multiple tests and all of them can be seen simultaneously***.

There is one thing to remember though. A test does not run always. ***A test in Postman runs only when the request was successful***. If there has been no response to the fact that your request was not correct, we cannot run tests through it. Also, you need to know that ***tests are written in Javascript in Postman***. Although you don't need to be an expert but you should be slightly familiar with Javascript so that it helps. In this tutorial, we will try to explain everything so that you don't feel the need to know the Javascript and can write tests yourselves.

***TEST in postman can be written in two ways namely***

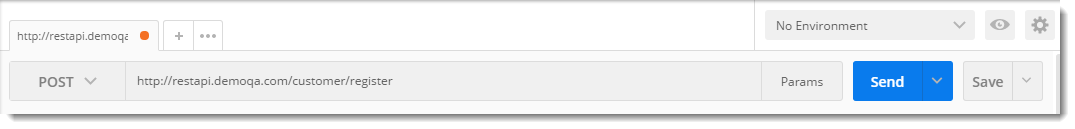
* ***Javascript method***
* ***Functional method***

Both the methods are used in Postman to write test today and both the methods use javascript as the base language. JavaScript method is the older method whereas the functional method is the new method. Although the Postman does not indicate that the support will end for the older method, it recommends to use the functional one and as it says on the official website of Postman, "***Functional method is the more powerful method among the two***". But since you need to know both, we will introduce you to the functional method.

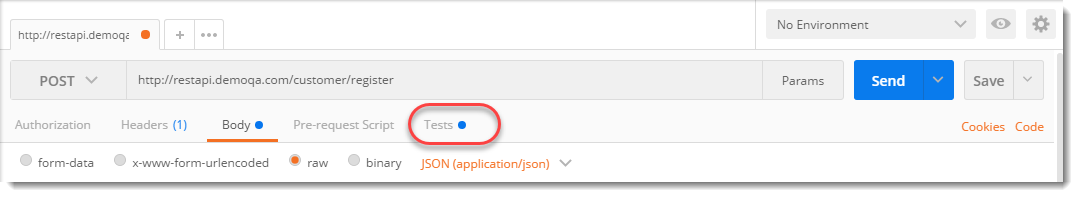
So now, we are good to perform our first test.

***How to set a Test in Postman with JavaScript Method?***

1. Use the API for creating the entry in customer register that we used in the [***POST Request***](https://toolsqa.com/postman/post-request-in-postman/) tutorial (*with body parameters*), don't worry about the entry being created or not.

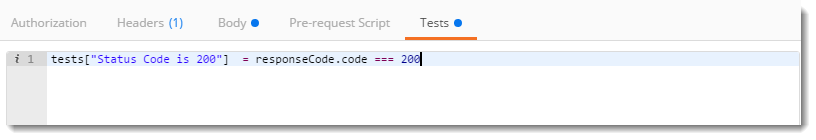


1. Open the ***Tests*** tab.



1. Write the following Javascript code as written in below

*tests["Status Code is 200"] = responseCode.code === 200*

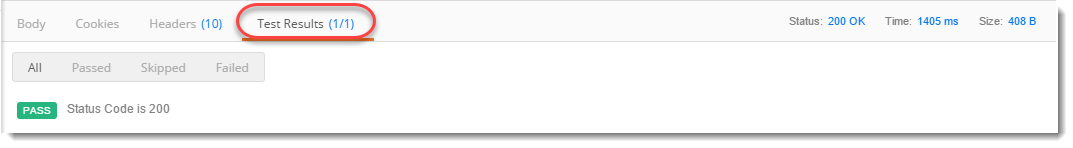


Now we will go through the above line to know the meaning.

* ***tests:****The first word we see is****"tests"****, which is a variable of type array. This array can contain any data types like string and int or even Boolean values.*
* ***Status Code is 20:****Status code is 200 is just a name or a simple string. We define this name to know what was the test performed by us therefore this name should be meaningful. If I write tests["****Passed****"], then I would not be able to know what kind of test has passed and this will also become more and more complicated if we run more than one tests, say 20 and one test fails. You could also write 'Status Code OK'.*
* ***responseCode.code: response code****refers to the response status code that we have received in the response box. We can run multiple test in Postman on response code such as knowing whether status code has a string or not. The second is an object which is called to know the status code (not the complete information but just the code) of the test. Once we call the responseCode.code, first all the information of the status code is saved temporarily and then object is called to check the status code. If the status code is equal to 200, tests save the value True in it.*

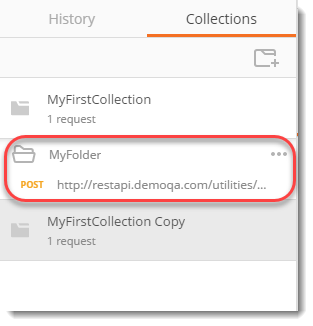
In the response box under the tests tab, those tests that have ***TRUE*** value shows ***PASS*** with the array name written or else ***FAIL*** is shown.

1. Click on ***send*** and look at the ***test results*** tab in the response box.



The result says that our test passed. This means that we were checking if we get 200 as status code and in this request, we got it.

1. Save the request in ***MyFirstCollection*** inside ***Myfolder***



By this, you have executed your first Request with Test successfully.

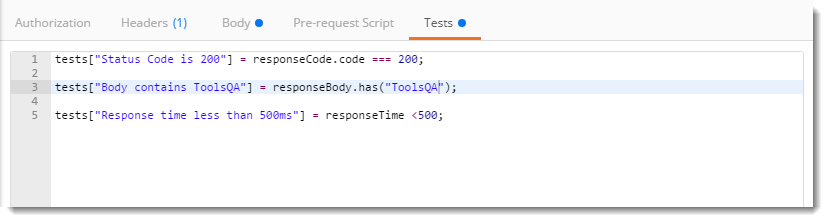
***How to set Multiple Tests for a Request in Postman with JavaScript Method?***

As we learnt earlier, we can use multiple tests on a single request and all those tests are displayed in the response box simultaneously. We will perform multiple tests on the same request that we used above. Write the following code inside the text editor.

*tests["Status Code is 200"] = responseCode.code === 200;*

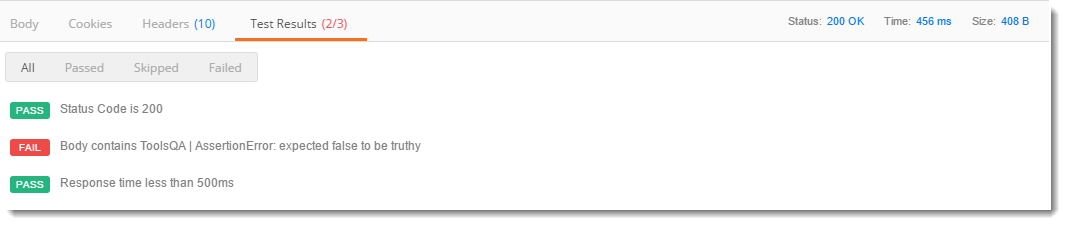
*tests["Body contains Fault"] = responseBody.has("ToolsQA");*

*tests["Response time less than 500ms"] = responseTime <1500;*



***NOTE****: The second test checks if there is a string****ToolsQA****in the body of the response and the third test checks if the response time is less than 500 ms or not.*

Now look at the response box in Postman, we have three tests written, out of which one failed which is the second one.  Because our response body does not contain ***ToolsQA***. This way we can perform many test in Postman simultaneously on one request. ***Remember to save the request***.

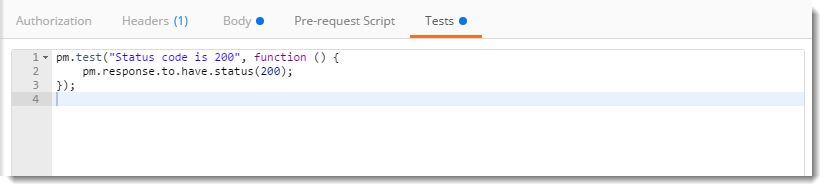


Our first test passed because we have a status code of 200 and our third test passed because our response time is 456 ms which is less than 500. ***Your response time may vary***.

***How to set Test in Postman using Functional Method?***

A simple functional method of ***testing whether the status code is 200*** is written below

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



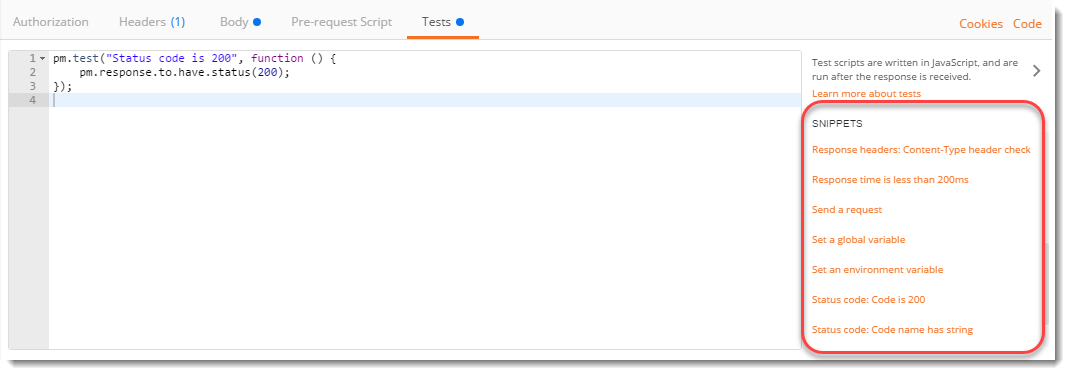
In the above image, the work is the same as we did in *JavaScript Test.* We are checking whether the status is 200 or not. We will look at the code above written test.

* ***pm.test****: This is the function for writing test specifications,****pm****here refers to Postman api and****test****refers to the specification of the function which is testing purposes*
* ***Status code is 200 :****This line is just a string which is the test name. When your test is performed this String will be written in front of the result. It is same as was in JS to know what the test was about*
* ***function(){}:****The next parameter is the function which is passed to perform the test*
* ***pm.response :****This is used to capture the response received and perform assertions on it such as status code, headers etc. This is same as responseCode in JS format.*

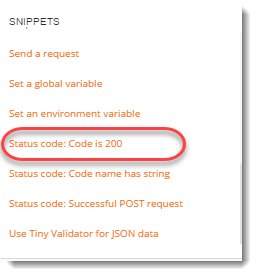
***Snippets in Postman to add Quick Test***

Since there are many times that a test is used on different requests and there are many requests present in one collection there arise a need to write some predefined test code which is used again and again or most frequently. In Postman, this section is called snippets. ***Snippets are the predefined test code in Postman that are written beforehand to use without writing the whole code***. Snippets save a lot of time and prevent errors that can happen while writing the code manually.

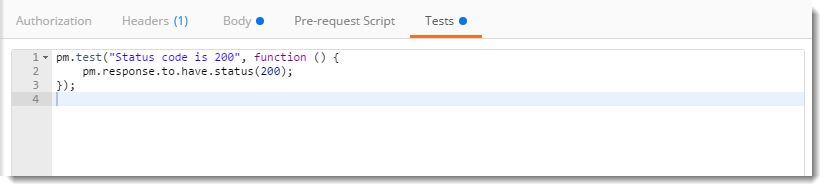
Snippets are located just beside the test editor.



Click on ***Status Code: code is 200***



Now, look at the editor



This code is exactly the same as that we have written in a functional method to test the status code.

***Note****: Since Postman prefers functional method, these snippets are available in functional methods only.*

***You can explore different snippets to better understand test codes of different assertions.***

***Collection Runner in Postman***

A ***collection runner in Postman*** as previously introduced is used for running a whole collection together. A collection runner runs all the requests in the collection or folder (*whatever you choose*) at once. Collection runner in Postman does not show any response, it is used to check the test cases whether they passed or not. A collection runner console shows all the tests at one location and their result. To run collection runner first make sure you have at least two requests in the folder ***MyFolder*** inside ***MyFirstCollection*** as shown.

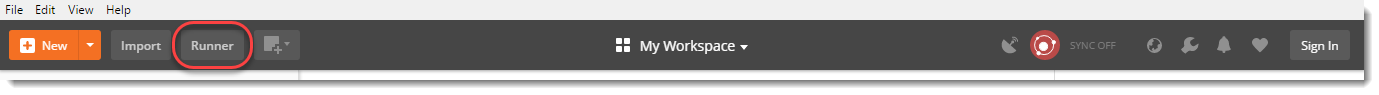
These two requests are the [***weather api***](https://bookstore.demoqa.com/swagger/#/BookStore/BookStoreV1BooksGet) (we used it in [***GET Request***](https://toolsqa.com/postman/get-request-in-postman/) chapter) and [***customer register api***](https://bookstore.demoqa.com/swagger/#/BookStore/BookStoreV1BooksGet) (we used it in [***POST Request***](https://toolsqa.com/postman/post-request-in-postman/) chapter). Remember the customer API is a Post request so it contains body parameters also.

***NOTE****: In weather API we have used the snippet named****Response Time is less than 200ms****which is self-descriptive.*

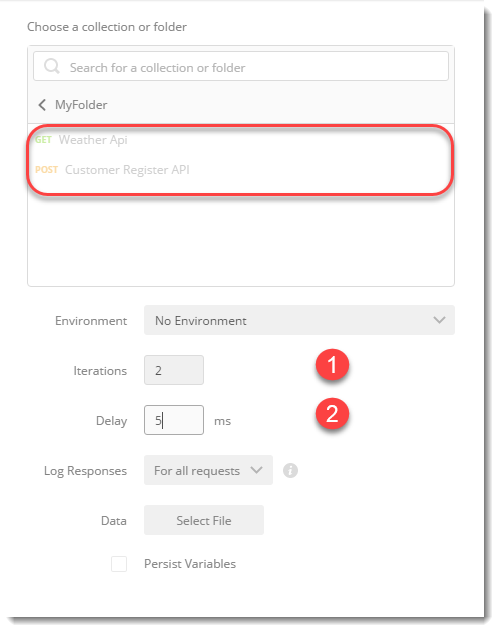
***How to run Collection Runner in Postman?***

Now we will look at How to run multiple requests together in Postman using Collection Runner.

1. Click on ***Runner***



1. Click on ***MyFirstCollection*** and then ***MyFolder***

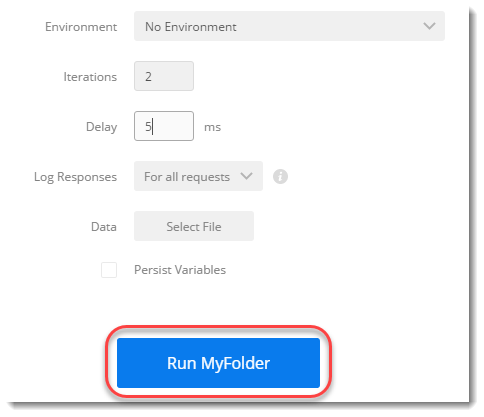


***Note****: I hope you have saved the requests in your collection which is shown above.*

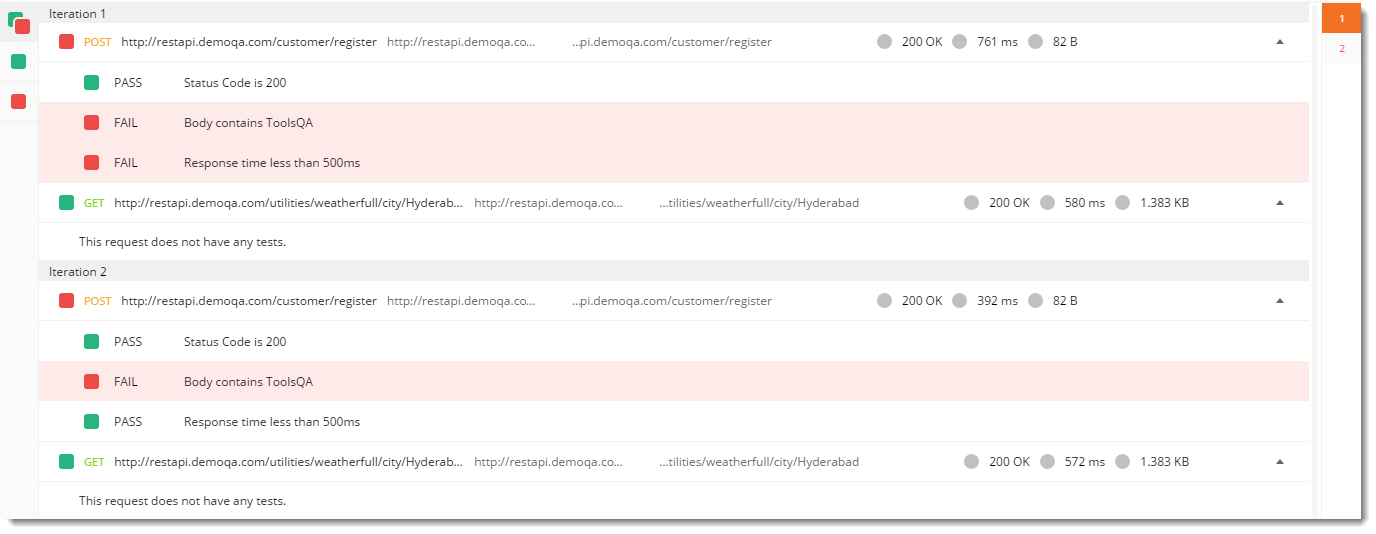
In the console you see two options:

* ***Iterations****: An iteration number is the number of times the same requests will run. For example, iteration set as 3 will run all the requests 3 times. Set it as 2.*
* Delay: A delay time is a time to wait between any two iterations. A delay time of 10ms will mean that Postman will wait 10ms after running one iteration before the second iteration. Set it as 5ms.

1. Click on ***Run MyFolder***



1. As you can see, all the tests with their results are available.

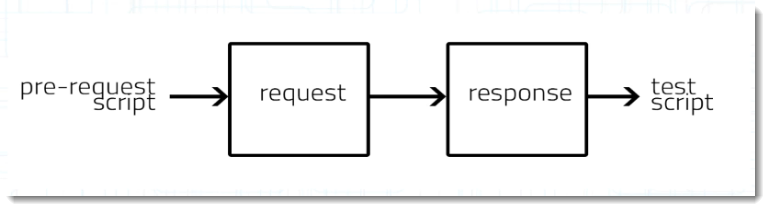


* *There are two iterations of each request. In the first iteration, I got the response time as 761ms which is greater than 500 ms but in the second iteration I got the response time as 392ms which is less, this caused one failure in that respective test.****Your test result may vary.***
* *Since we did not perform any test on the weather API request, the console says****The request does not have any test.***

Go on to perform multiple tests on weather API and then try to run them in collection runner.

***What are Scripts In Postman?***

Scripts are a piece of code that you can write and let Postman execute it at specific points in your test Lifecycle. Postman lets you write ***pre-requests scripts***, which will run before Request and ***tests scripts***, which will run after ***Response***. Scripts are used in Postman to enable dynamic behaviour to request and collections. It allows you to ***write tests***, ***change parameters*** and even ***pass data*** between the requests. A script can be added to the ***request, collection, folder*** or an i***ndependent request***. Scripts in Postman are written in ***Postman Sandbox.***

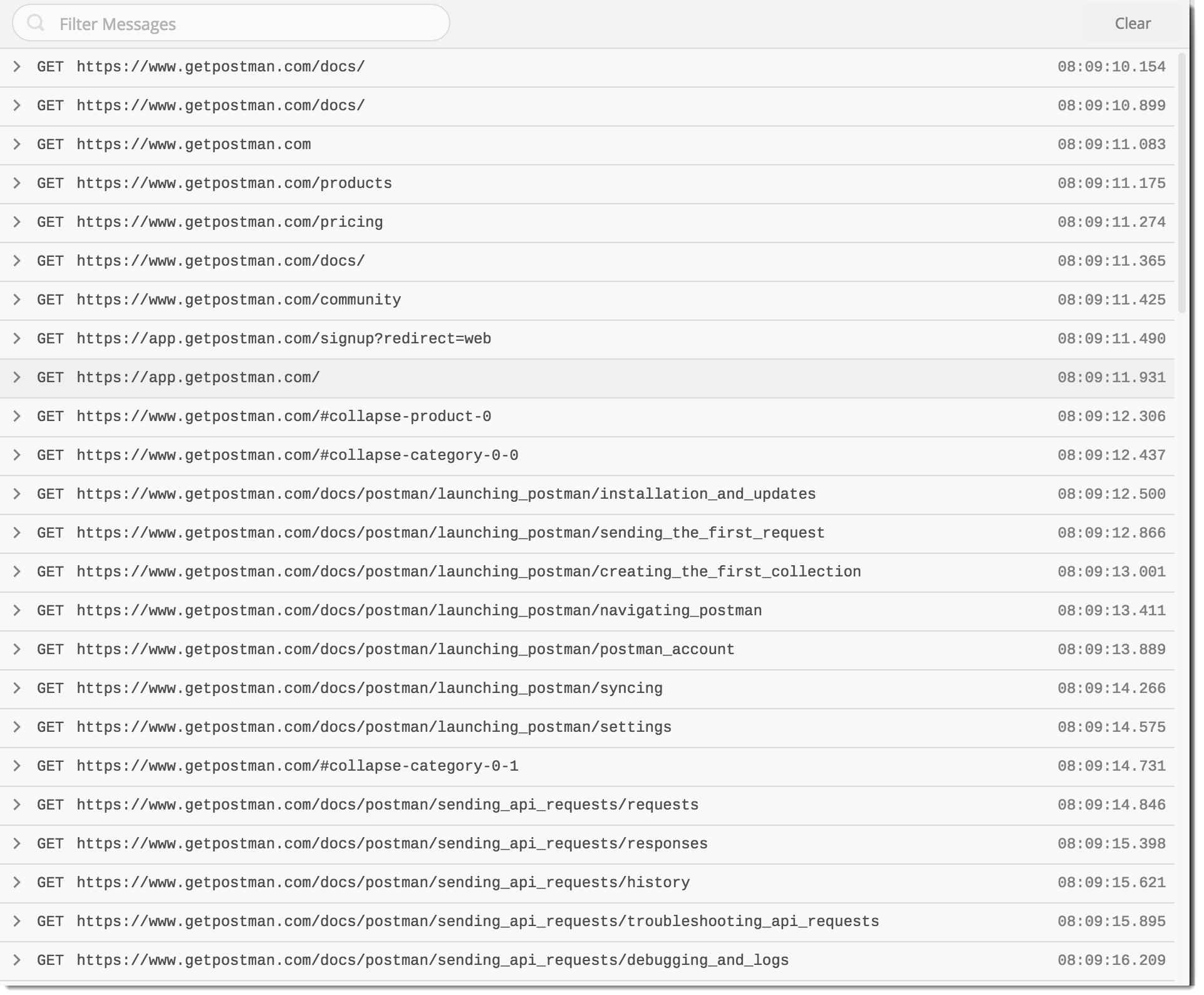


***What is Postman Sandbox?***

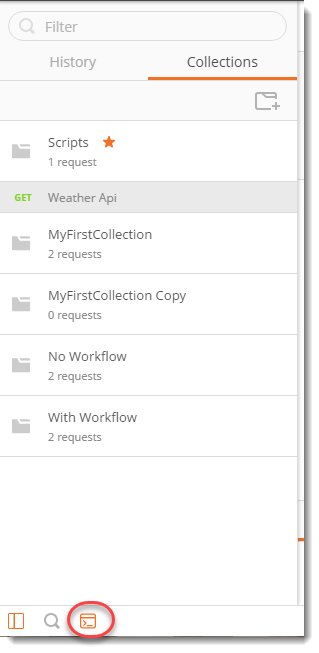
***Postman Sandbox is a powerful execution environment written in Javascript***, so any script you write to be run in Postman must be in Javascript like tests that we run in the [***tests tutorial***](https://toolsqa.com/postman/test-and-collection-runner-in-postman/). These scripts are then executed in this environment and we see the result thereafter. I hope you must have used a compiler at some point in your life. You need to code in the same language to which the compiler is designed *like Turbo C, you can write and run a C code in Turbo C compiler but not a python code*. Same is the case with the sandbox, that is why you need to write in JavaScript.

**What is Postman Console?**

As stated in the official Postman blog, " ***Postman Console is analogous to a browser’s version of the Developer Console, except that it’s tuned for API development***". There are certain times that we might not be able to see where the problem is in the execution of pre-request script in Postman. Postman console notes down everything that happens in the request and hence we can look at the console and see the error. The below image can be referred to have a look at a typical Postman console used for many requests.



Although Postman console can be opened by the shortcut commands that are described below, Postman also has a dedicated icon just for opening Postman console. This icon is located in the ***Sidebar*** ([***Postman Navigation***](https://toolsqa.com/postman/postman-navigation/) )

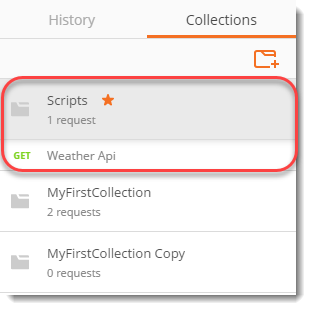


It behaves analogously to a browsers development console where everything is visible, all the requests that you have sent in that website or the code of the page too. If we need to catch an error or see how far our execution was right we use ***console.log*** feature. By this we can print on the console-specific log statement, this can help us track the execution and find issues in our code. This simple example will help you understand the concept.

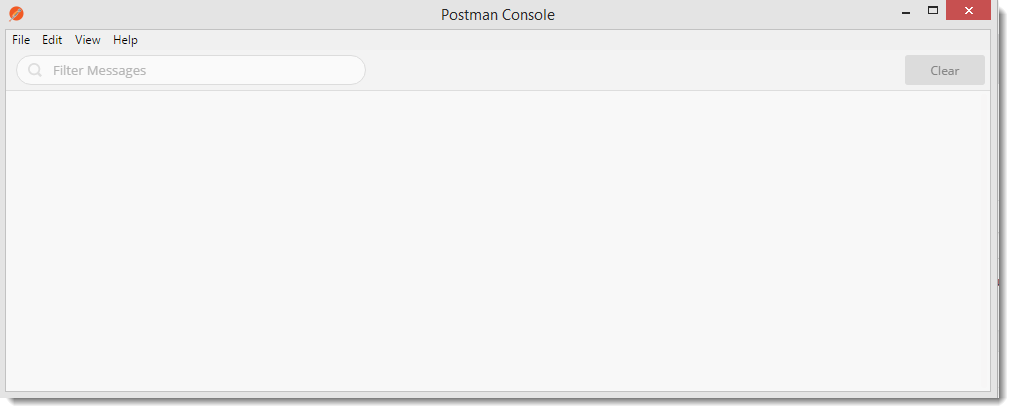
***How to see pre-request script logs in Postman console***

1.Create a new collection called ***Scripts*** (*See*[***Collection***](https://toolsqa.com/postman/collections-in-postman/)*Chapter*)

1. Write the [***weather api***](https://bookstore.demoqa.com/swagger/#/BookStore/BookStoreV1BooksGet) request in it.

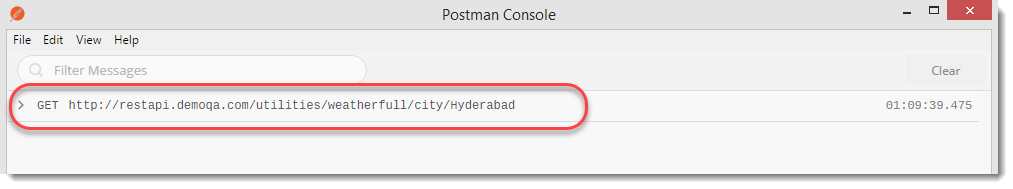


3.Open ***Postman Console*** by pressing ***Ctrl+Alt+C on Windows*** (*Cmd + Alt+ C on mac*).



***Note****: Always remember to open the console first before sending the request, or else your requests won't be logged in the console*.

4..Press ***Send*** and see what is visible on the Postman console.



As can be seen, the request is logged into the console. Logging into the console is done by Postman automatically but you can also do it on your own if you want to check your code. As discussed above ***console.log*** feature is used for this purpose. When we do ***console.log***(*string*), the string is printed as it is on the console. ***We can also pass variable instead of string***. This helps a lot. *Let say we have a function which does not give correct output to us. If we write console.log(variable\_name) in the console, we can easily see if the variable we are dealing with is having the same value as we intend or not.* In the next section we will be using ***console.log*** which will clear any doubts.

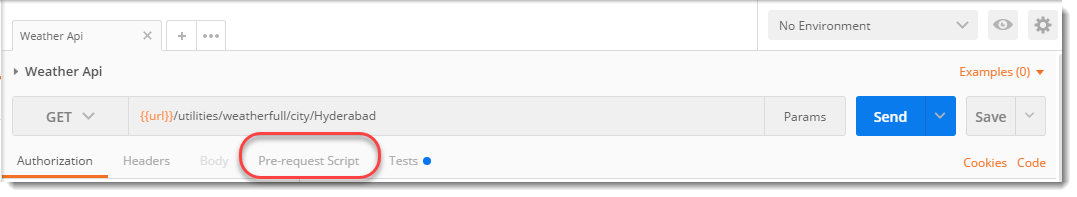
**What are Pre Requests scripts in Postman?**

As stated above, ***a pre-request script in Postman is a script that runs before the execution of request***. It runs in Postman sandbox and comes very handy when we have to do something dynamically while the execution is in the process. These can be setting the variables or clearing them as we will see later in the tutorial. A pre-request script in Postman can be run on a folder, a request or a collection but if we have specified scripts in all the three, there exists an order in which the script is executed

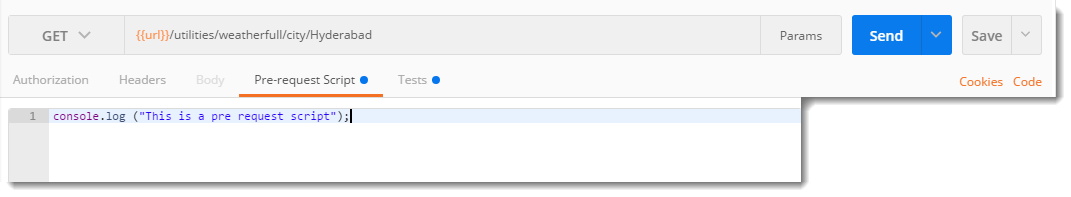
* ***A pre-request script associated with a collection will run prior to every request in the collection.***
* ***A pre-request script associated with a folder will run prior to every request in the folder***.

To demonstrate using Postman that pre-requests scripts run before the execution and tests scripts run after it, we will look at a very simple example here.

1. Go to the ***Pre-Requests*** Tab in the weather api in the same collection that we created above.

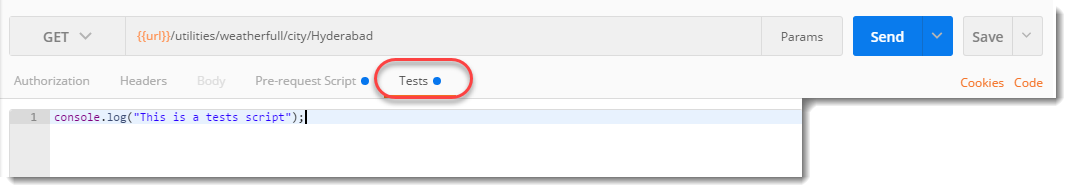


2.Write *console.log ("This is a pre request script");*

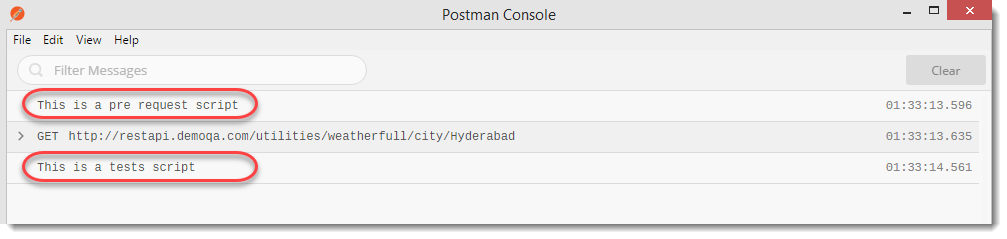


3.Go to the ***Tests*** tab and write

*console.log("This is a tests script");*



4.Press ***Send*** and open the ***Postman Console*** and have a look.



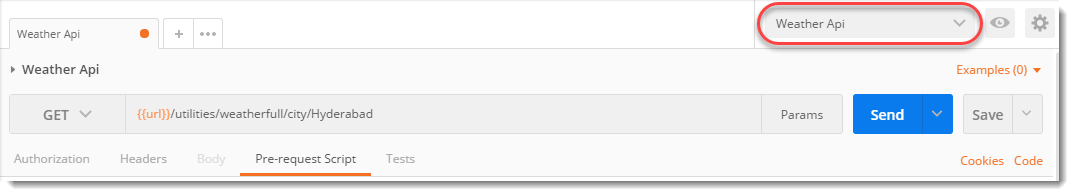
*The pre-request script has run before the execution of the request while the test script has run after the request*.

***Creating Variables using Pre-Request Script in Postman***

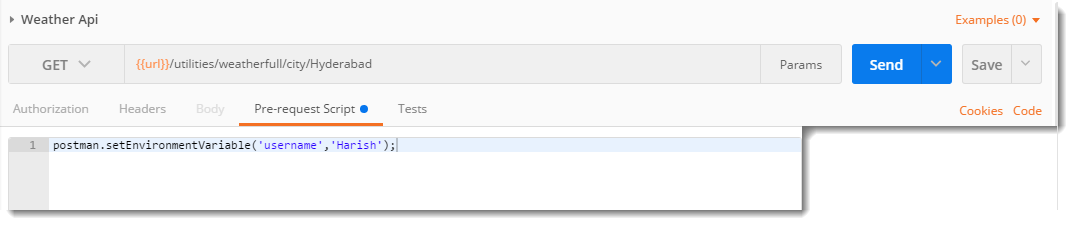
We use pre-request scripts in Postman for all the things we need to do before the execution of the request such as setting variables, clearing the variables or getting the values etc. In this tutorial, we will try to set the environment variable in the environment ***Weather Api*** that we created and used while learning about [***environments***](https://toolsqa.com/postman/environment-variables-in-postman/).

1.Go to the ***Pre-Request Script*** Tab inside the ***weather api*** request.

2.Confirm that you have ***Weather Api*** environment selected (*Learn from*[***Environment and Variables***](https://toolsqa.com/postman/environment-variables-in-postman/)*chapter*).

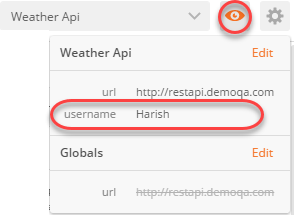


3.Write the following code inside the editor *postman.setEnvironmentVariable('username','Harish')*;



*This will create a variable inside the environment with the name "****username****" and value "****Harish****"*.

4.Press ***Send*** and look at the current variable by clicking the ***eye*** icon (*Learn about in*[***Environment and Variable***](https://toolsqa.com/postman/environment-variables-in-postman/)*chapter*)

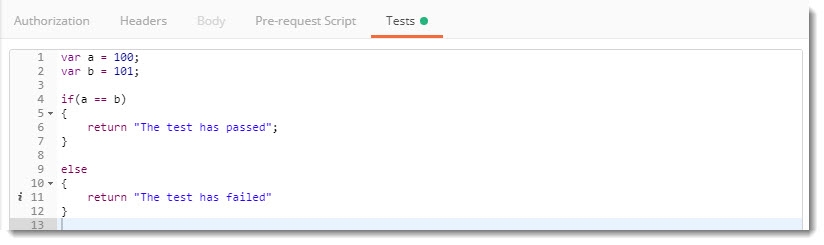


Look at the variables, we have the variable ***username*** present in the environment that we created through the script.

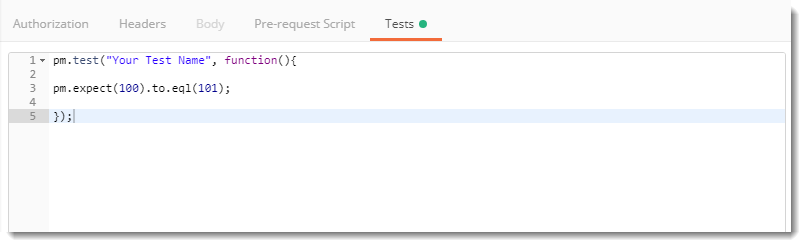
***Why to use Assertions?***

The sole purpose of a test is to identify that for a situation given parameters of the system are as expected. To force that the parameters of the system are correct we assert the expected values with the actual values during a test run. Assertions are used to assert that expected and actual values during a test run match. If they don't match, the test fails with the output pointing directly to the failure.

An assertion improves your test writing skills to a greater level. Postman provides JavaScript support to write tests which works under Postman Sandbox. As we learnt in the tutorial [***Set up Postman Tests***](https://toolsqa.com/postman/test-and-collection-runner-in-postman/), it is hard to write assertions or Functional methods in JavaS. In this tutorial we will learn how to write assertions using an external JavaScript library called ***Chai - Assertion Library.*** The assertions that we will be writing with this assertion library takes lot less effort compared to what we write directly in Javascript. The following image shows the difference with a very basic example.



***The above image contains a code where we are checking if a is equal to b or not***. The same can be written with chai assertion library in the following manner.



Isn't it so concise and easily readable? Think about the complex problems that we can easily write through this library.

An assertion is very useful in finding defects in the code as you can write an assertion just like a test, although they both are different. A test performs all the steps to reach to a particular state of the application and an assertion can validate the state of the application at that point. An assertion is very useful in finding defects in the application code. If you add assertions in the test, the test will fail once the assertions fails. But defining a more complex test in a mere easy way such as finding an element in an array will take just 2 lines of codes in assertion while it will take at least 5-10 lines in ***JavaScript*** tests. Reading a code also becomes very easy when we write assertions rather than writing the same thing in tests.

While writing assertions in Postman, there are two main steps involved:

* ***Parse the response body:****It is important to know what kind of response you are getting to perform a test on it. The most popular response is****JSON,****simply because it is very easy to read by humans and is machine readable also. It might happen that most of you might not even have to deal with any other response but that does not kill the fact that the response can be any format. There are many other formats of an HTTP response:*
  + ***XML***
  + ***HTML***
  + ***Text***
* ***Write test code:****Since we have already discussed about writing the tests in the*[***test and collection runner***](https://toolsqa.com/postman/test-and-collection-runner-in-postman/)*tutorial we will not be covering it here. But, there we studied about the test writing in Javascript method or functional method. While we have to write only in Javascript because of the Postman Sandbox, there exist one library which makes it easier for us to write a test which would have taken more lines of code if written in Javascript. This library is Chai Assertion Library which we will talk about now.*

But the assertion part is not confined to Chai Assertion Library. Chai Assertion is just a part of many assertions that Postman provide and also the only one being external to Postman. All the other Assertions works under Postman Sandbox which is of course Postman's. Taking the difficulty level in mind, chai assertion library is fairly easier than the other assertions and therefore in this tutorial we will be learning about the same. This tutorial will help you get familiar with the concept of Assertions so that in the next tutorial we can execute some difficult assertions.

**Chai Assertion Library**

***Chai assertion Library is included by Postman by default in its application***, so when you are writing chai assertions you don't have to worry about any other installation processes. The most amazing fact about assertions in Postman is that they write human readable tests. Tests written in assertions are so human readable that you might find it as a english sentence. All this makes your tests more easy to read and more friendly for humans. Although we are not needed to write very complex chai assertions as that are not required but we will cover the most common and frequently required assertions in Postman which will make your way complete while using this software.



Although , if you want to learn more about Chai Assertion Library you can visit this [***link***](https://www.chaijs.com/). In the next section we will learn about some assertions.

***How to write Assertions in Postman using Chai Assertion Library?***

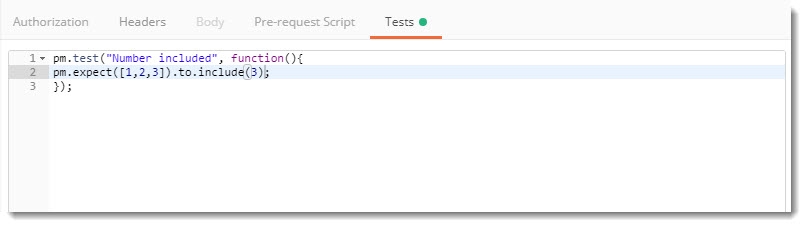
If you have visited the above link, you would have found out that there are numerous assertions available in Chai library. We will be using some of them in the later section but in the this section of Assertion, we will make you understand the concept and assertions.

***Assertion: Number is in array or not***

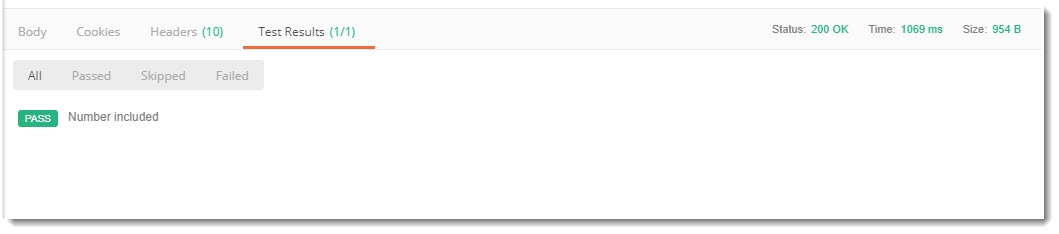
1.Open the [***weather api***](https://bookstore.demoqa.com/swagger/#/BookStore/BookStoreV1BooksGet) in Postman

2.Write the following code in the tests tab

*pm.test("Number included", function(){* *pm.expect([1,2,3]).to.include(3);* *});*



Press ***enter*** and you will see the obvious response.

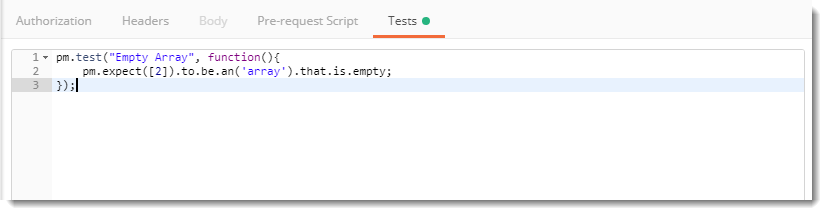


***Yes we see the number is included because 1,2,3 has 3. It is obvious.***

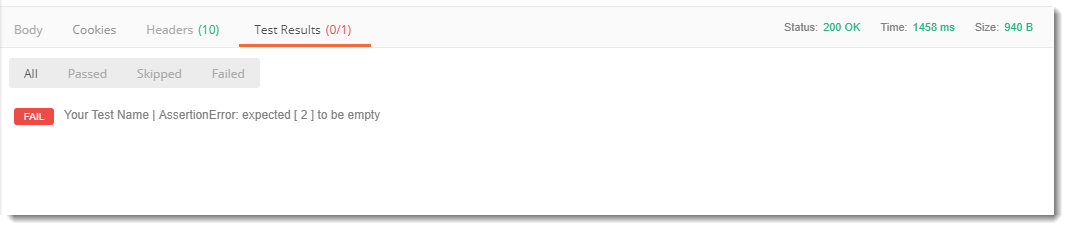
***Assert: An array to be empty***

1.Write the following code in the tests tab of weather api (*or any api of your choice*)

*pm.test("Empty Array", function(){* *pm.expect([2]).to.be.an('array').that.is.empty;* *});*



Guess the response before pressing ***enter***



Okay. So you must have got pretty familiar now with the Chai Assertion Library. We will now show you one more assertion to conclude this tutorial.

***Assertion: Verify objects***

*pm.test("Test Name", function(){*

*let a= {*

*"name" : "Harish"*

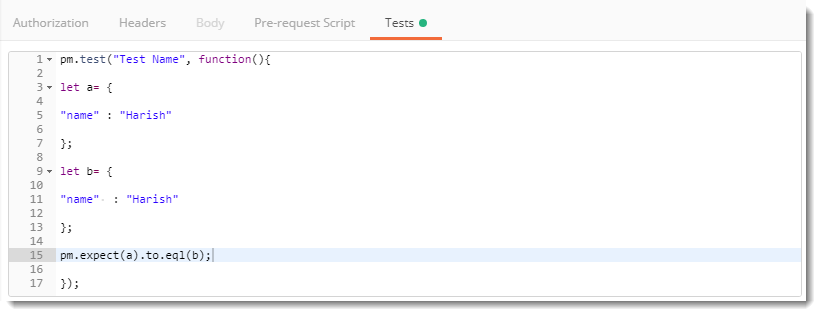
*};*

*let b= {*

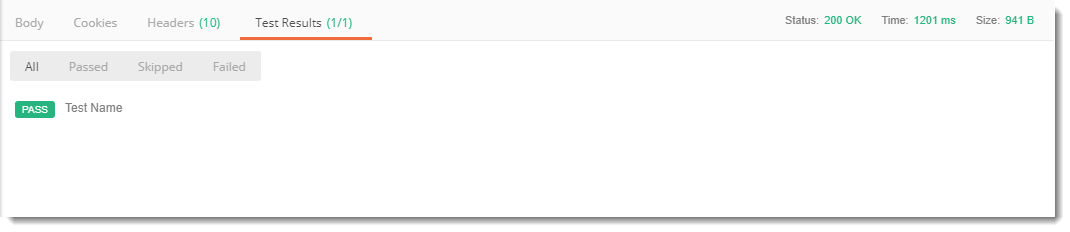
*"name"  : "Harish"*

*};* \* \*pm.expect(a).to.eql(b);

*});*

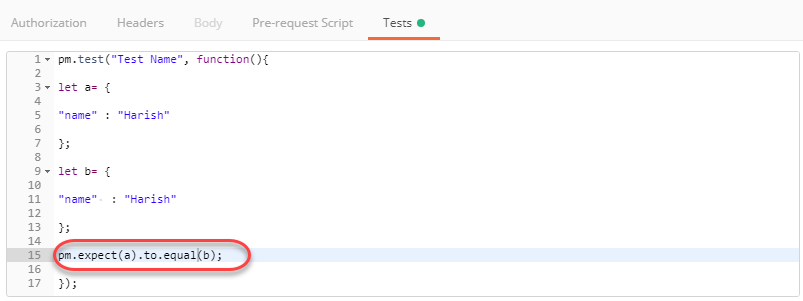


Press ***send*** and see the results.



It passes because the names are equal. But you might be wondering about equal and eql that we used above. Before clearing the air, let see the response for the same ***but with equal***.

Write the same code as above and r***eplacing eql with equal***.



Did you get the same response as in eql?



Although we are having the same code, ***eql and equal produces different responses***. When we use ***equal*** we compare the objects created, which are different here i.e. a and b. While using ***eql*** we compare the properties of the objects, in this case name. As the two names are same the comparison passes. ***equal uses the === operator which is called Strict equality. While eql is Deep equality which compares the individual properties of the object.***

**Different types of Asserts in Postman**

For instance we can think of sending a request and checking all the above stated things on the same. In the end of this tutorial, you can also add all the assertions in one single request to practice and improve your skills. So, we will start now.

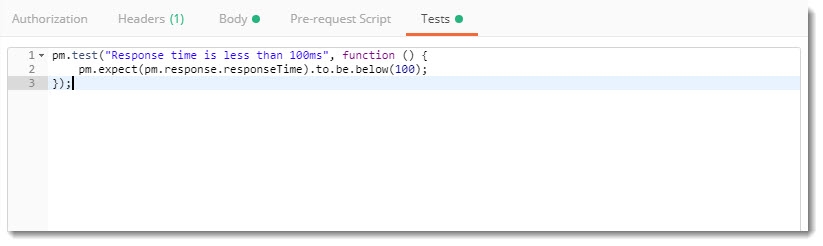
***Prerequisite:***

* ***POST method API EndPoint****: We are using*[***Customer Register***](https://toolsqa.com/postman/post-request-in-postman/)*API*

***Assert on Response Time***

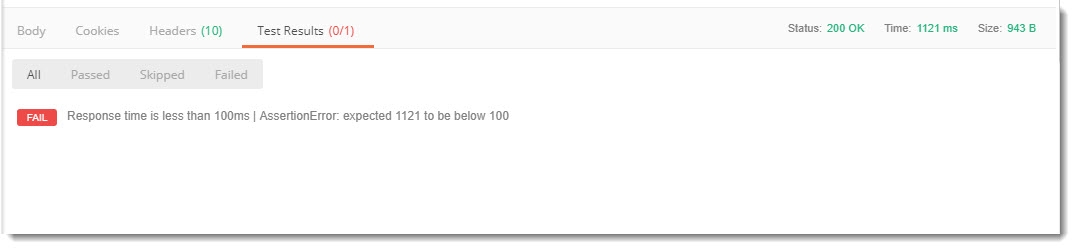
This assert helps us to verify the ***Response Time of the Request***. Below we are verifying that if the Response Time is less than 100ms. Go to the ***Tests*** tab and write the following code:

*pm.test("Response time is less than 100ms", function () {      pm.expect(pm.response.responseTime).to.be.below(100); });*



***NOTE****: This assertion can also be modified to check the time to be above a certain value (to.be.above(value)) and equal to a certain value (to.be.equal(value)).*

Press ***Send*** and see the response.

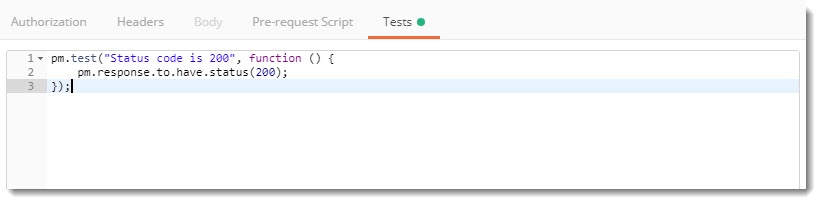


***Note****: In the above case, Assert got failed, as the response time was 1121ms. Also, the same is visible clearly in the response box as****AssertionError****:****expected 1121 to be below 100****which is false obviously.*

***Assert on Response Status Code***

This assertion is based on checking the ***Response Status Code***. In the below test,  we are verifying that if the Response Status Code is 200. Test will *PASS* in case of Status Code 200, else it will *FAIL* in case of any Status Code other than 200. Write the following code in the ***Tests*** tab:

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



You can place any status code inside the value box to check the value of the status. The same can also be expressed in Chai Assertion Library as

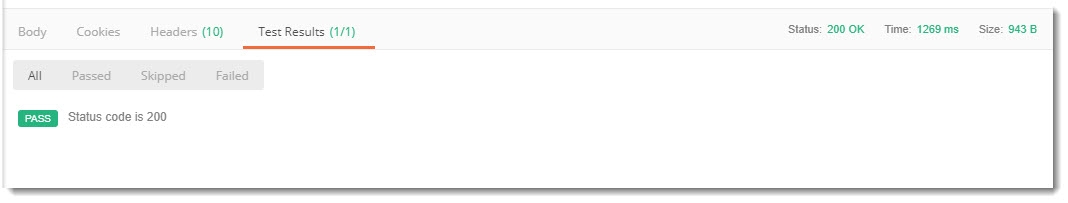
***For checking status being OK***

*pm.test("Status is OK", function () {      pm.response.to.be.ok; });*

***For checking status being BAD REQUEST***

*pm.test("Status is Bad Request", function () {      pm.response.to.be.badRequest; });*

Press ***send*** and see the response which is true in my case.



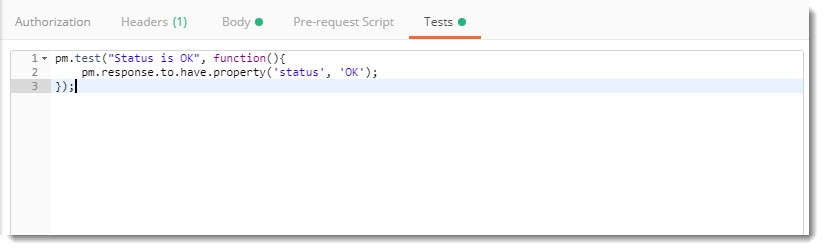
We got the response status code to be 200 and hence our assertion has passed.

***Assert on Response Status Code Meaning***

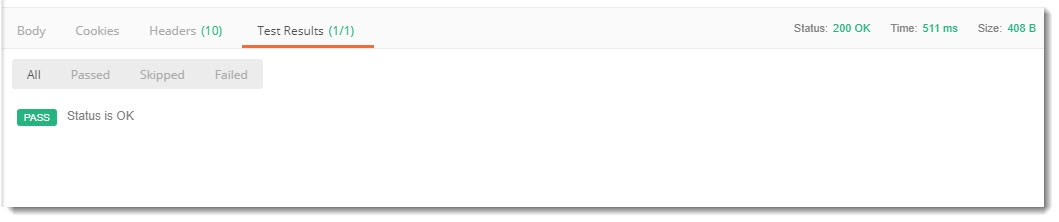
This assertion is based on checking a ***specific property***. In this assertion we will check a specific property and its value. In the example given below we are checking the property ***status*** and its value being ***OK***.

Write the following code inside ***Tests*** tab.

*pm.test("Status is OK", function(){      pm.response.to.have.property('status', 'OK'); });*



Press ***Send*** and see the result which will be true in my case.

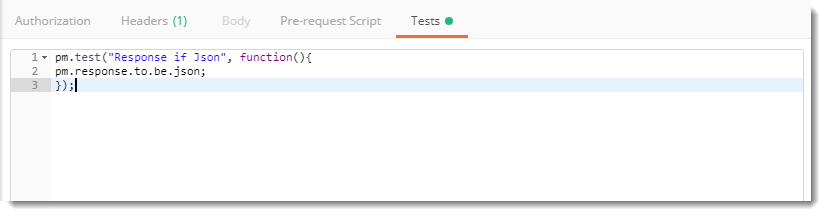


This one was quite understandable, I guess.

***Assert on Response Type***

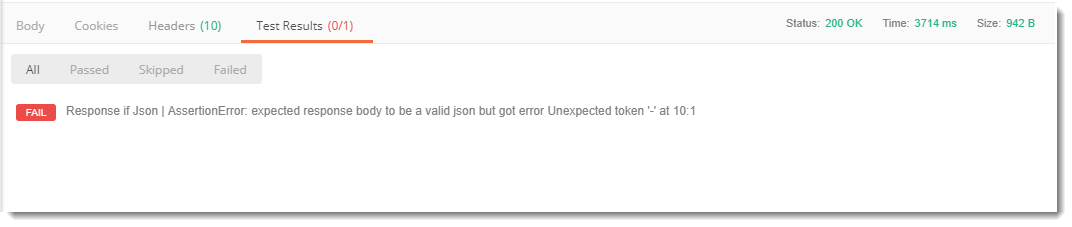
This assertion is based on verifying the ***Response Type***. In the below test,  we are verifying that if the Response Type is JSON. Write the following code in the ***Tests*** tab:

*pm.test("Response if Json", function(){      pm.response.to.be.json; });*



***Note****: I hope you remember that in*[***Get Request***](https://toolsqa.com/postman/get-request-in-postman/)*when we sent the request using weather api, we received the response in the text format rather than JSON format. We are using the same API here*.

Press ***Send*** and see the result.



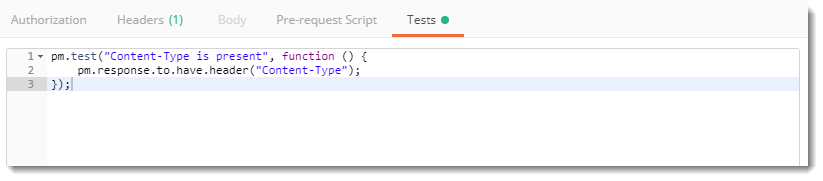
***The assertion has failed because of the response type***. We expected response type to be *JSON*, but the response that we get in weather api is in the *TEXT* format.

***Assert on Response Header***

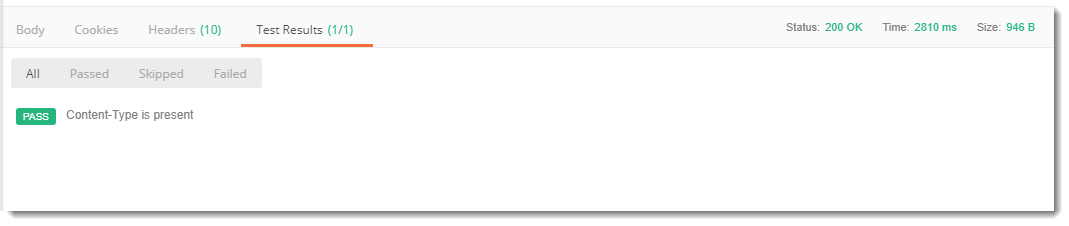
***This assertion is based on checking whether the header has content-type or not.***

Write the following in your tests tab

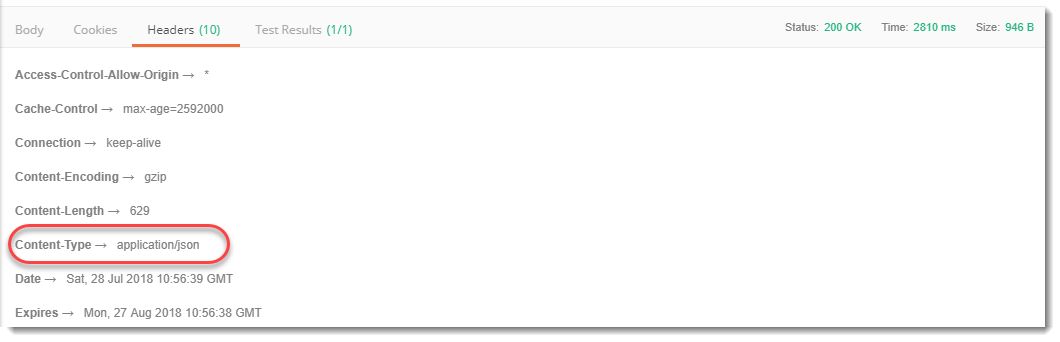
*pm.test("Content-Type is present", function () {      pm.response.to.have.header("Content-Type"); });*



This assertion checks if the content-type header is present in the response or not. Press ***Send*** and see if it is or not.



***Yes, the test passed***. But, how can we check if it was really present or not. As you can see besides ***Test Results, Headers*** is written. Go to ***Headers*** and ***Content-Type*** must be present there.



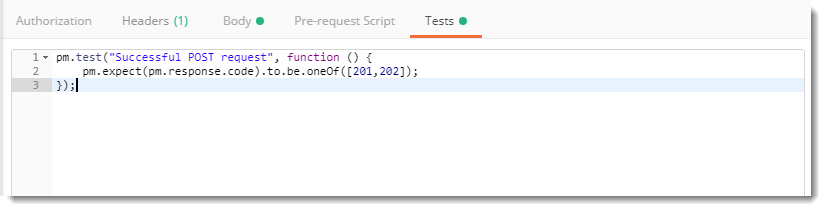
So now we have seen the assertions that are commonly used. ***Now, we will try to use both Chai Assertion along with these assertions to create some meaningful tests***.

***Assert for Multiple Status Code***

For this we will be using the ***customer register api*** since it uses POST method type to send the request or you can also use Weather API but ultimately the test will fail. You can download both the APIs from [***here***](https://toolsqa.com/wp-content/uploads/2018/07/Newman-Collection.postman_collection.zip).

Go to ***tests*** tab and write the following code

*pm.test("Successful POST request", function () {      pm.expect(pm.response.code).to.be.oneOf([201,202]); });*



***Note****: 201 is created and 202 is Accepted*.

Press ***send*** and see the response which will be pass if the status code is 201 or 202 or else will fail.

***Assert on Response Text***

Check if response contains a string

Write the following code in the ***tests*** tab of any API which is correct and gives response.

*pm.test("Body matches string", function () {      pm.expect(pm.response.text()).to.include("string\_you\_want\_to\_search"); });*

***Replace the query "string\_you\_want\_to\_search" with the string you want to search. If your response will contain the string your assertion will pass or else fail***.

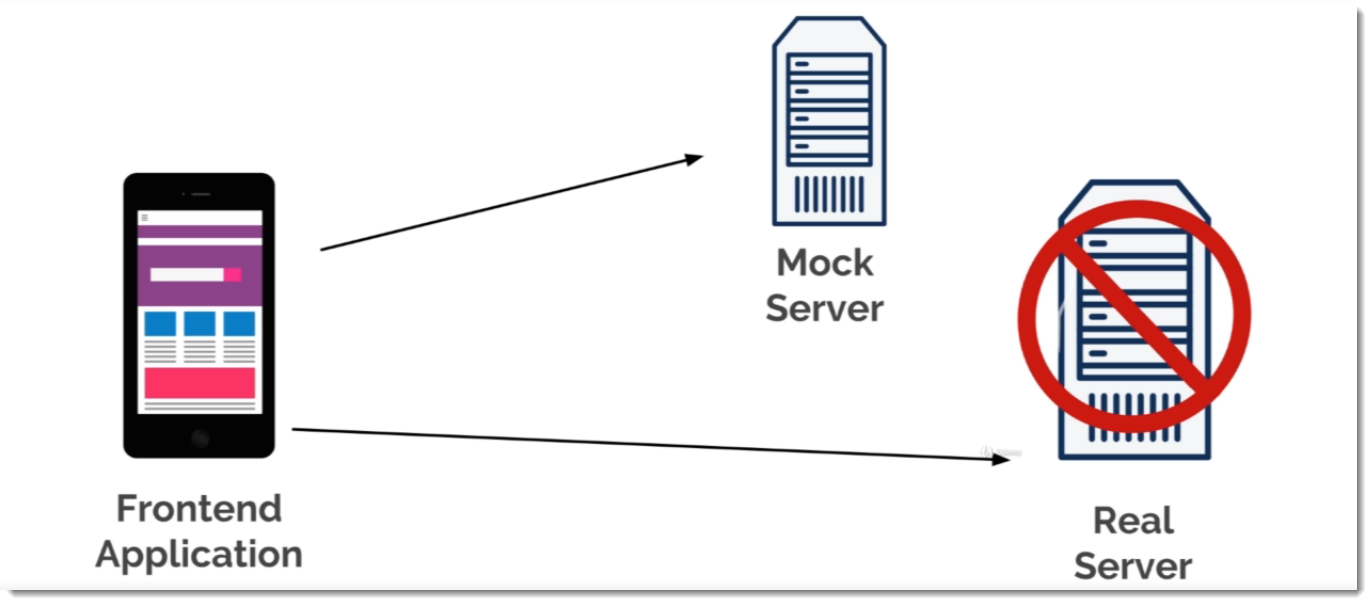
***What is Mock Server?***

***A mock server is a server that is not a real server. It is just a fake server that is simulated to work as a real server so that we can test our APIs and check the response or errors***. This server is set up in such a way that we get a particular response for a particular request that we desire to see. A mock server behaves like a real server and uses fake APIs, of course, for testing and development purpose. There are a number of reasons for which we require a mock server. Along with the case given above, it is also required in today's testing world. Such requirement is in Agile methodology which is recent and better than waterfall methodology. In this method, testing and development go side by side. For this, a tester needs to have the same requirement as the developer to work simultaneously. For which you need a mock server. In addition to this, few reasons are listed in the next section.

***Why we need a mock server?***

We need a mock server for a number of reasons. A mock server is required

* *To test your own API while developing and before making that API live on the server of your organisation.*
* *To get feedbacks and bugs quicker.*
* *For checking the dependencies in your API before it is available to everyone.*
* *For QA engineers to use this for testing/isolating external dependencies and integration testing.*
* *By front end developers to use it before actual endpoints are available. This is done while designing the UI so that you don't need to wait for the time till actual endpoints are developed. It saves a lot of time.*
* *For engineers to develop a prototype of their idea to present it to the investors for funding.*



The above image explains the same point mentioned above. A front-end developer needs to develop the UI for which he must know the responses he will get. For the same he cannot wait until the APIs are upon the server, so he uses the mock server in order to achieve the same and save time.

I guess now you must have known that mock server is a pretty important feature for a tester. It is very helpful in both the development and test phases of software. Continuing for the same we will now proceed to create our first mock server.

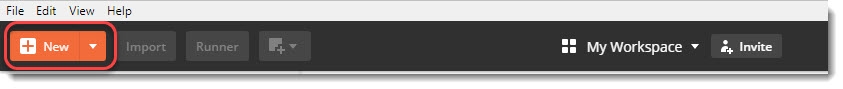
**How to create a mock server in Postman?**

In this section, we will create our first mock server in Postman but before that, you must know a few things about the mock server

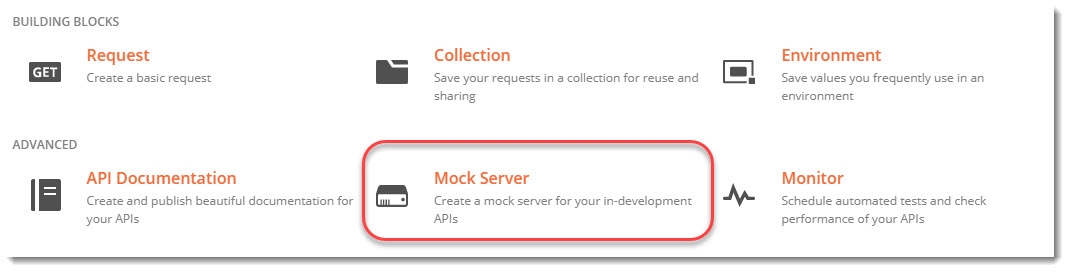
* *The mock server is already integrated inside the postman app and is not required externally.*
* *The mock server also has CORS (Cross-Origin Resource Sharing) enabled. It means that you won't get any cross-origin errors while using the mock server.*
* *The mock server is free to use i.e. it is available in free tier of Postman.*

Okay so now we will follow these steps to create our first mock server.

1.Click on the ***New*** button on the top left corner in the Header section



1. Select ***Mock Server*** in the Panel



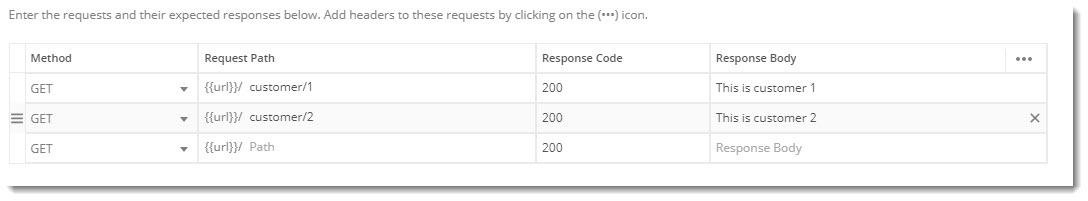
1. A new panel will be opened up which will enable us to create requests.



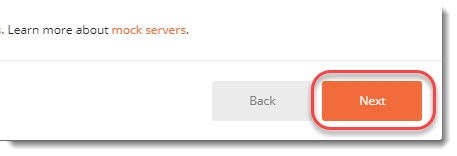
There are different columns in this mock server panel which stands for:

* *The first column Method is for the request type methods like GET, Post etc.*
* *The second Request Path will create the url for your API*
* *Response code will define the code you wish to get in response (*[***Read more about Response Code***](https://toolsqa.com/postman/response-in-postman/)*)*
* *Response Body will have the response body that you want to show (*[***Read more about the Response Body***](https://toolsqa.com/postman/response-in-postman/)*)*

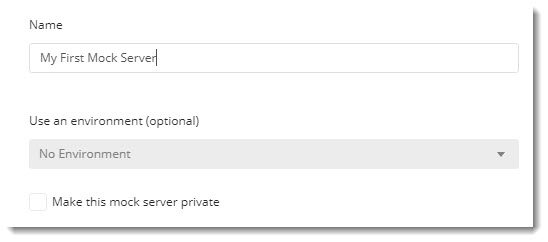
1. Fill up the columns as shown in the image.



1. Click on ***Next***

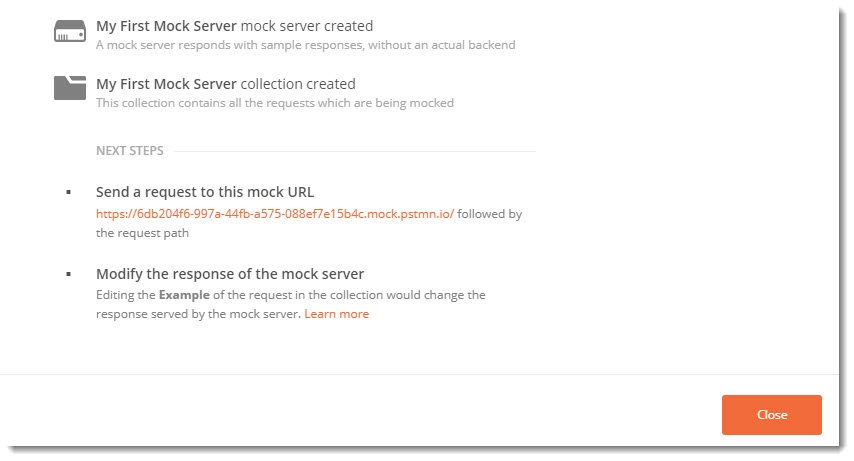


1. Name your Mock Server as per your choice



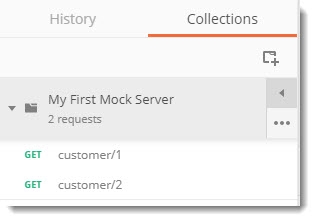
***Note****: You can make the server private also if you don't want to make your information accessible by everyone but it would require Postman API key in order to access the server. For the beginning, we will keep our server public to reduce complexity.*

1. Now the next screen will show you the URL through which you can access the server. This is the confirmation screen that your mock server has been created successfully.

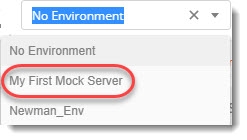


8.Click on ***Close*** and close the panel.

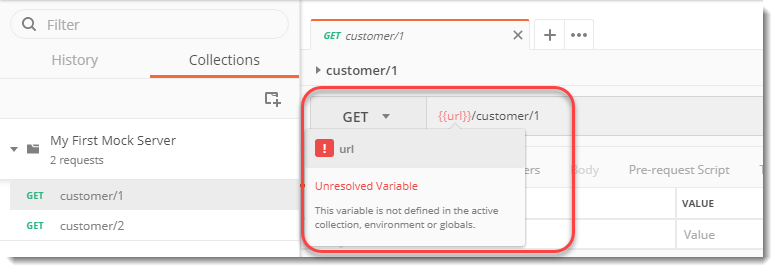
1. As soon as you close the panel, you will see that a new collection with the same name has been created with your APIs that you entered.



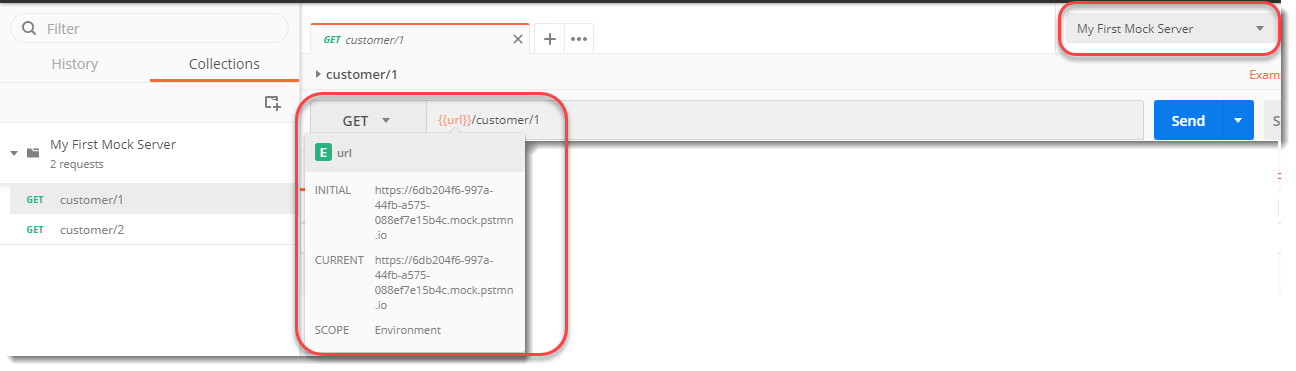
1. You can also notice that a new environment has been created as well ([***Refer Environments in Postman tutorial***](https://toolsqa.com/postman/environment-variables-in-postman/))



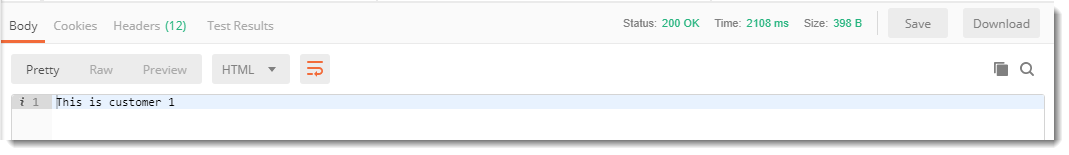
1. Select the first request in the collection and hover your mouse over the *{{url}}* written in the request



Can you guess why this is an unresolved variable? It's great if you can. Yes, the environment has not been selected. This is why the environment has been automatically created. Change the environment to the one created and hover again.



Now the URL value can be displayed. Press the ***Send*** button and see the response



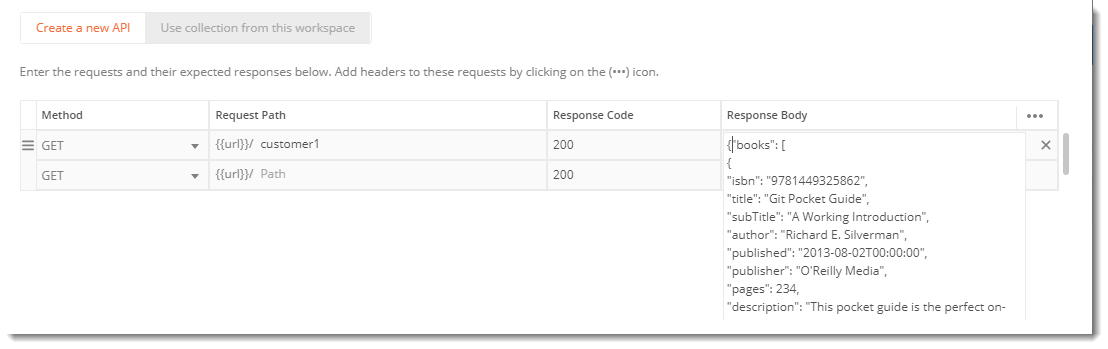
We have got the same response as we created while setting up the mock server in the beginning. Check the response code also.

**How to get the response in a different format in Mock Server?**

It is very easy to get the response of Mock Server in other formats also. Since we received text response in the above section, we will now see how to get the response in the most common format i.e. ***JSON***.

Follow steps 1 to 3 in the above section.

In the server creation panel, instead of writing plain text, write the response body in *JSON* format.



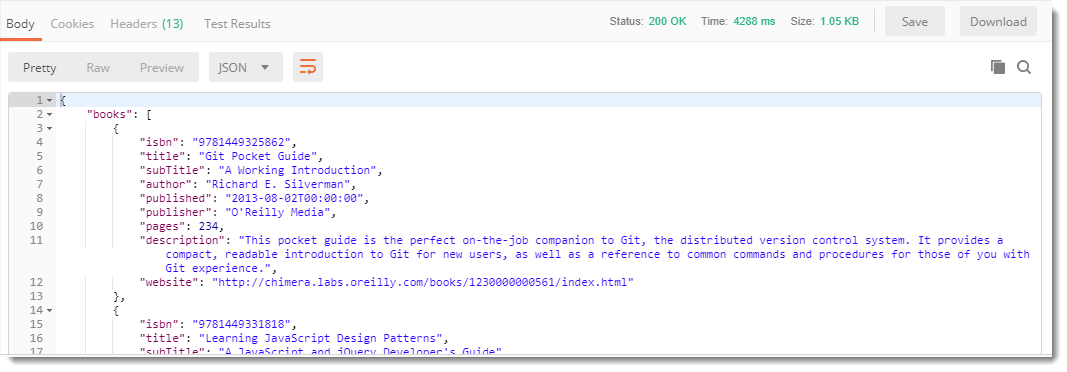
The above code is the data about a book store with different values of different books. The code is written below.

*{ "books": [ { "isbn": "9781449325862", "title": "Git Pocket Guide", "subTitle": "A Working Introduction", "author": "Richard E. Silverman", "published": "2013-08-02T00:00:00", "publisher": "O'Reilly Media", "pages": 234, "description": "This pocket guide is the perfect on-the-job companion to Git, the distributed version control system. It provides a compact, readable introduction to Git for new users, as well as a reference to common commands and procedures for those of you with Git experience.", "website":* *"https://chimera.labs.oreilly.com/books/1230000000561/index.html" }, { "isbn": "9781449331818", "title": "Learning JavaScript Design Patterns", "subTitle": "A JavaScript and jQuery Developer's Guide", "author": "Addy Osmani", "published": "2012-07-01T00:00:00", "publisher": "O'Reilly Media", "pages": 254, "description": "With Learning JavaScript Design Patterns, you'll learn how to write beautiful, structured, and maintainable JavaScript by applying classical and modern design patterns to the language. If you want to keep your code efficient, more manageable, and up-to-date with the latest best practices, this book is for* *you.","website":"https://www.addyosmani.com/resources/essentialjsdesignpatterns/book/" }*

*]* *}*

Create the server with the name of your choice (*I used****JSON RETURN****as the name*) and then press send after selecting the correct API and Environment.

You will receive the response in *JSON* format now. You might get HTML directly but change the format to *JSON* from the dropdown to beautify the response as shown.



## What are HTTP Cookies?

HTTP Cookies, also known popularly as browser cookies or internet cookies, is a small piece of information that saves onto the client's side, i.e., the web browser, and the server sends it. What piece of information is this? It depends entirely on the developer designing the website. A developer can save login information as an internet cookie, user browser history as an internet cookie, or anything else which may be of their use later on.

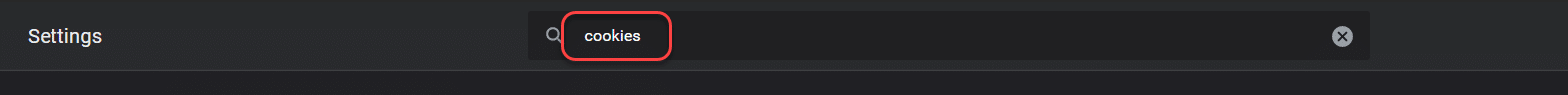
In other words, you can think of HTTP cookies as a memory for a particular website or maybe its identity. As soon as the user hits enter on a web address, if there is a browser cookie saved for that website, the server will recall the user and will serve them accordingly. There is a perfect statement that I read somewhere a long time ago; an ***HTTP cookie remembers stateful information for the stateless HTTP protocol***.

To make you familiar with how the HTTP cookies look like, let's explore our browser:

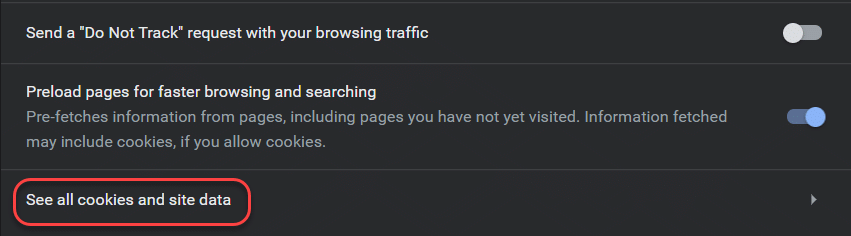
1. Type in ***chrome://settings***  in your Chrome Browser or Visit the " ***Settings***" section in any browser you are using.

It will open up the settings panel.

1. Type in the search panel, " ***Cookies***".



1. Open ***See all cookies and site data***.



And you can see all the websites that have saved an HTTP cookie or browser cookie on your system, and as I count them on my browser, they are literally in hundreds.

Select any one of the websites, and you will be able to see the HTTP cookie that a particular site has saved.

Okay, so, it is quite clear that even though we did not know about any such thing, we were helped by it for improving our browser experience. But we still do not understand why do we need HTTP cookies in the first place? Is it that important? What if I tell you that literally, an HTTP cookie is a reason for billions of dollars of trade? Let's see how.

### *****Why do we use HTTP Cookies?*****

If I want to familiarize myself with the HTTP cookies to a layman, the best term would be to describe these internet cookies as the shadow of you that exists only on the internet. They follow you EVERYWHERE!!  They have followed you here too! Honestly, the browser cookies are something that you cannot ignore, which makes them of utmost importance in the life of a developer and testers. In broader terms, there are three sections on which we use the internet cookies:

* ***Session Management***: As soon as the user logs into the website, a session creates for them with a session-id recognizing that session. The HTTP cookies can very well manage this. Through HTTP cookies, we can save your game scores or remember you as a previous user and login automatically. It can expand to anything that the server would like to remember; we can do that with our browser cookies.
* ***Tracking***: Tracking with HTTP cookies helps the business know your interests and provide better service to you. For example, if I explored a pen drive on Amazon, it implies that I am interested in it. Therefore, when I visit another website, it makes sense that if an advertisement serves to me, making it of my interest increases the chances of clicking it. It is just a small example of tracking. Tracking through the browser cookies can be used to show you the recommended products and much more.
* ***Personalization***: Personalisation through HTTP cookies helps the user personalize the website or any other component on the website according to themselves. For example, a popular search engine DuckDuckGo helps the user set a color for the page. When the user selects the color for the first time, the DuckDuckGo server sends a browser cookie wrapped with the username/system id so that anytime that particular user searches, the color page is the same.

So in a way, HTTP cookies are a two-way road. It provides businesses with a method to earn billions of dollars and provides the user with a better and comfortable experience. Think of a time when you would have to login again and again as soon as your session runs out on Amazon (You would know the pain while doing web scraping). What if you were watching an advertisement related to a hat when all I was interested in was a pen drive? HTTP cookies are beneficial for us, and as a developer and tester, we must know how to set these cookies.

### *****How to Set HTTP Cookies in a Browser?*****

In this section, we will explore the different attributes and methods used by the server to set the cookie on the ***user-agent side***, i.e., the browser. It is important to remember before we proceed to set-up HTTP cookie that there are ***two types*** of browser cookies:

* ***Session Cookies***: These types of browser cookies delete once the session ends.
* ***Permanent Cookies***: These types of browser cookies remain on the system and communicate with the server every time the website opens.

To set a cookie, we use the "***Set-Cookie***" header with a long list of attributes according to our needs.

***Syntax:***

Set-Cookie: *<cookie-name>* = *<cookie-value>*

With [***Postman***](https://www.toolsqa.com/postman/postman-tutorial/), we will able to see the complete response from the server along with the cookies; for this tutorial, we will just stick to the syntaxes.

#### **HTTP Cookies Attributes**

As mentioned in the previous section, internet cookies do have attributes that provide some more meaning to the cookie. Otherwise, the cookie is just a name and a value. These attributes will help us set-cookie on the user's browser. Let's understand all these attributes in more detail:

##### ***Expires***

The "***expires***" attribute of HTTP cookies provides the lifetime value of the cookie. Once the value reaches, the cookie deletes automatically. Providing a expires value is important in the browser cookies so that it gets refreshed periodically as the information keeps on changing according to the user behavior. If this attribute is not specified in the header, the HTTP cookie automatically becomes the session cookie and gets deleted once the session is over. We can set it syntactically as follows:

Set-Cookie: *<cookie-name>* = *<cookie-value>*; Expires = *<date>*

##### ***Max-Age***

Similar to the "***expires***" attribute, the ***max-age*** attribute specifies the time until the HTTP cookie expires. ***If both*** "***expires***" ***and*** " ***max-age***" ***attributes are specified, the "max-age" attribute has the precedence over it***. Also, a value of 0 or negative will expire the cookie immediately, so a non-zero positive value is expected in this attribute. We can set syntactically as follows:

Set-Cookie: *<cookie-name>*=*<cookie-value>*; Max-Age = *<number>*

##### ***Secure***

Specifying the ***secure*** attribute means encoding the cookie and saving confidential information on the client's system. We can request a secure HTTP cookie only via the \*HTTPS \* scheme. We can set it syntactically as follows:

Set-Cookie: *<cookie-name>*=*<cookie-value>*; Secure

##### ***Path***

The ***path*** value specifies the path that should be within the requested [***URL***](https://en.wikipedia.org/wiki/URL), or else the browser does not send the cookie to the server. A path URL may look like /Back-End/Postman on ***[ToolsQA](https://www.toolsqa.com/)***, so the browser cookie will be sent only when this path includes. It does not matter what is ahead of this path as long as the specified path exists. We can set it syntactically as follows:

Set-Cookie: *<cookie-name>*=*<cookie-value>*; Path=*<path-value>*

##### ***Domain***

The domain value specifies the host to which the HTTP cookie needs to send. For example, toolsqa.com is a domain name. All the subdomains come under major domain that specifies, and all the subdomains include in the cookie. We can set it syntactically as follows:

Set-Cookie: *<cookie-name>*=*<cookie-value>*; Domain=*<domain-value>*

##### ***HTTP-Only Cookie***

If the cookie is set for the HTTP- only attribute, then the client-side would not be able to access the cookie. Having an HTTP-Only attribute explores the possibilities of any flaw in the client-side and is more secure since the Javascript is not able to access the cookie from the client-side. We can set it syntactically as follows:

Set-Cookie: *<cookie-name>*=*<cookie-value>*; HttpOnly

All these attributes are optional, and it's up to the developer what they want for their website.

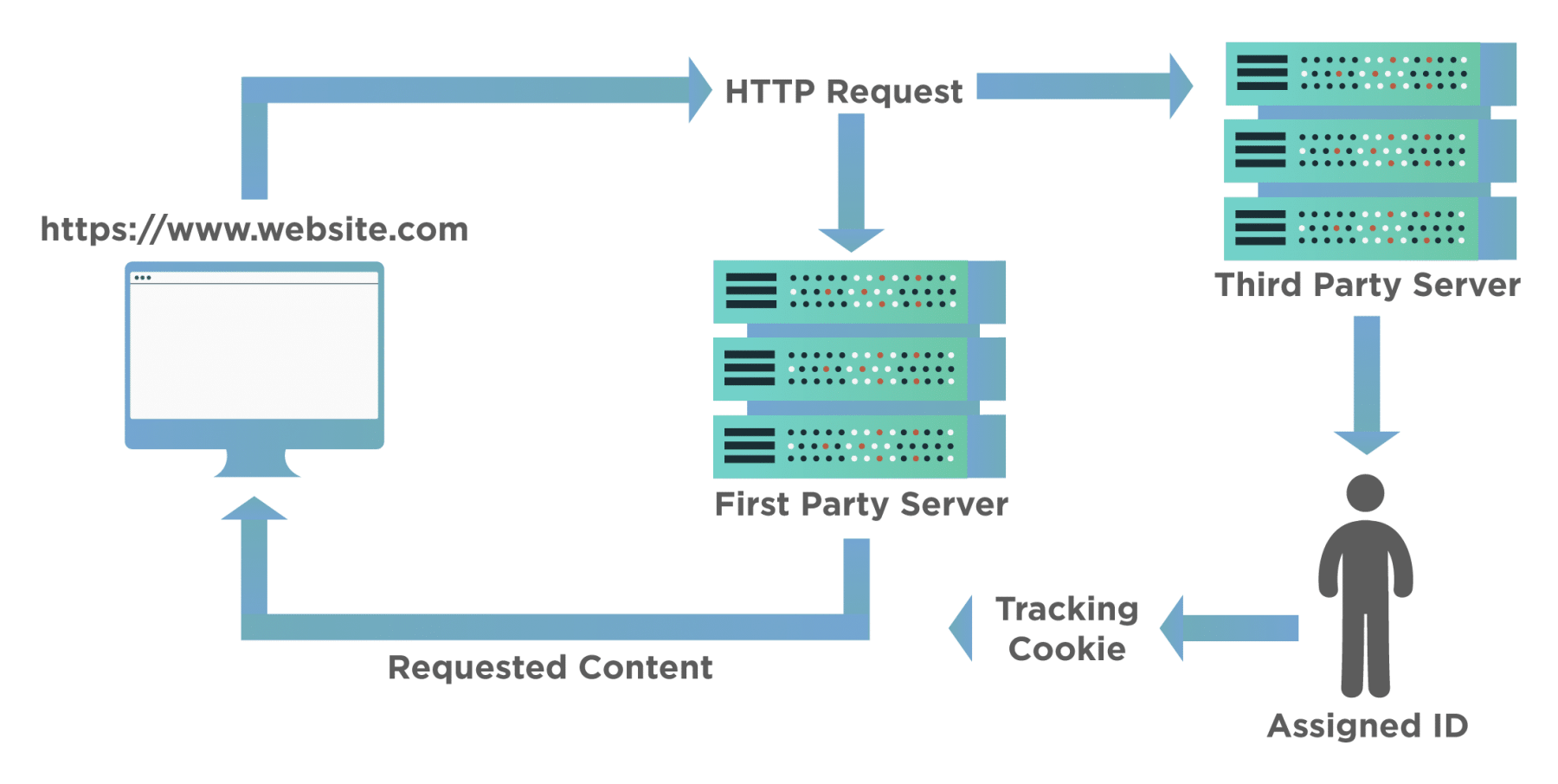
### *****What are Third-Party Cookies?*****

It has been a long time that we, as a user, are distracted by a popup whenever we visit a website over the internet. The popup says, "***This website uses third-party cookies***". Majority of the time, the popup allows us with only one option, "***Accept***". That's rude, isn't it? This makes us wonder, though, "***What are third-party cookies?***" and why websites use them?

***Third-party HTTP*** cookies are placed into the client's browser by other websites apart from the one the user is visiting (hence the name "***third-party***" cookie). For example, a third-party cookie may be set by the Ads agency for placing ads on another website that are using Ads from that particular agency. Third-party cookies are mainly used for ads purposes and tracking the user. Although a developer can use it for any purpose, they want. A user can think of third-party cookies as a partnership between the developer and the third-party to serve the user better. So the next time you see a popup, "***This website uses cookies***", almost all the time, it is to place the ads according to the user's interests. The practice is not new, but strict cyber laws have enforced for the browser developers to inform the user of these things.

#### **How to Set Third-Party Cookies?**

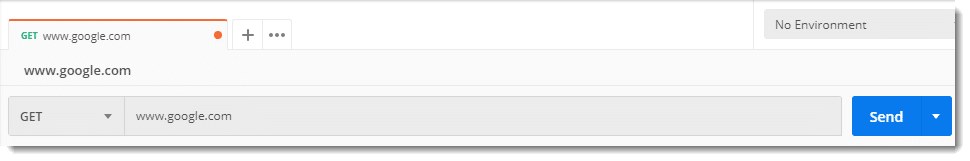
To set the third-party cookie, the developer should be willing for it. Therefore, the developer places a link into their website which, when loaded, hits the third-party server. The server then recognizes the user. If the user is new, a third-party HTTP cookie is placed onto his browser. If the user is not new, the request sent to the server retrieves the user information. For example, his interests, browsing history, etc. from the HTTP cookie and places an appropriate ad on the website.



It is a straight-forward process. Moreover, if you visit your cookie section on the browser, you can see all the "***Ad cookies***" in it. Once the user clears the cookies from their browser, you will notice how the ads change when you visit the same website again.

**Cookies in Postman**

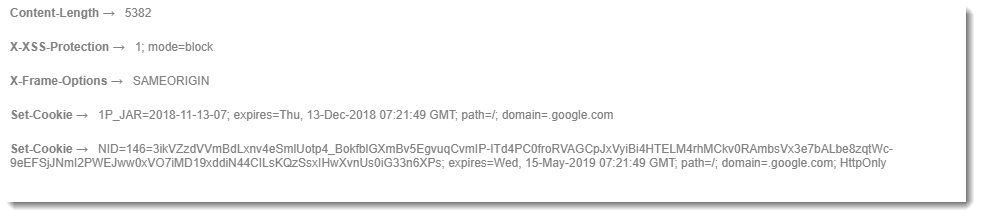
Since *cookies* are first returned from the server, lets see what cookies are being returned, when we access the Google server. Moving on to the Postman app, hit the following API ***www.google.com*** in Postman.



Now go the ***Headers*** tab in the response section.

Header_Response_2

Here you will find ***Set\_Cookie*** which is the cookie being sent by the server of google.



Header contains too many values and cookies is very important part of a header. Therefore, Postman also gives us a separate option of Cookies.

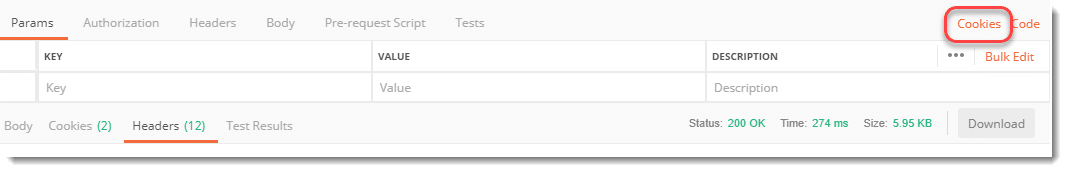
Cookie_Option_Response

***Note****: This will show the same cookies as we saw in the Header section. Cookies displayed in this section are the cookies related to Google. Site specific cookies.*

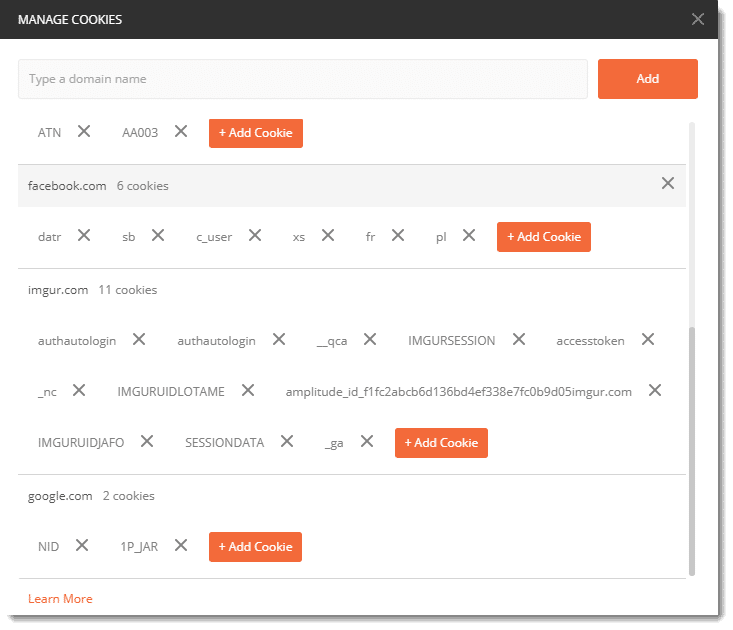
***Manage Cookies in Postman***

This is how we can see the cookies that we receive from the server to which we have hit the response. Postman also provides a ***Cookie Manager*** separately where you can ***Add, Delete or Modify the Cookies***.

Click "***Cookies***" on the top right.



This will open the cookie manager panel where you can see all the cookies are located.

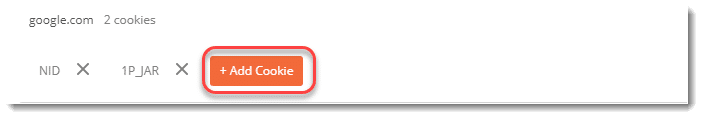


***Note****: Cookies displayed in this section are browser specific cookies, means cookies saved from your previous made requests, irrespective of websites.*

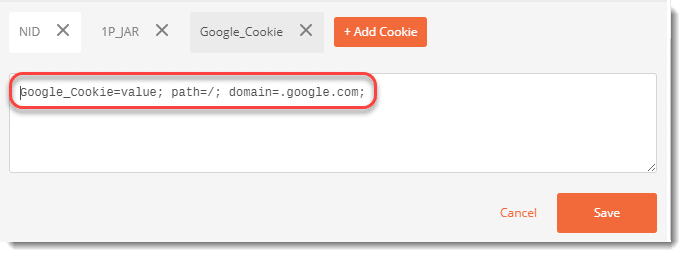
This cookie manager works same as a browser's. It will save all the cookies irrespective of the work you are currently doing. As you can see in my cookie manager it has cookies from ***imgur.com*** website which I used in the [***OAuth 2.0 tutorial***](https://OAuth 2.0) and since then I have used Postman many times. I have also cleared/deleted all the collections related to the ***Imgur*** but still the cookies are maintained by Postman just like a browser.

***Add Cookies in Postman***

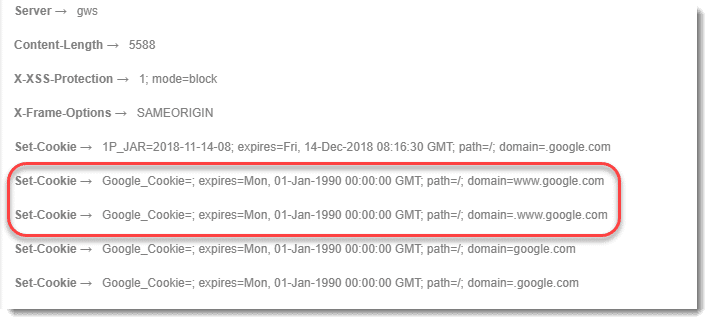
To add the cookie. Go to the ***google.com*** domain in the manager and click ***Add Cookie.***



A new text box will open up where it will have some values already written. Change those values as given in the image below.



Now you have added a new cookie to the domain ***google.com***. This cookie will be now sent along with the request to the server. Press ***Save*** and close the panel. Hit the endpoint again and see the header section now.



You can see the cookie that we added can be seen here. This is shown multiple times because Google server does not recognize this cookie and hence expiry date is also set to 1990. Notice the expiry date of other cookies.

In the same fashion, cookies can also be modified by opening the already saved cookie in the cookie manager. Please try it yourself as a practice.

**Executing Tests on Cookies**

In Postman the cookies can also be checked i.e. whether the expected cookie or the expected value is returned or not. This helps us a lot if we are receiving too many cookies. For this you need a few prerequisites.

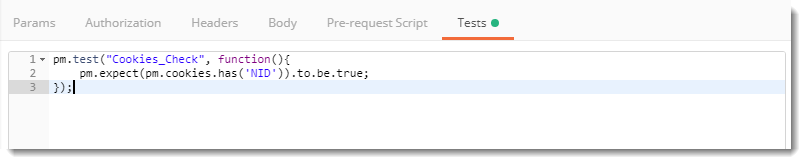
***Pre-Requisites***

* ***Knowledge of Tests- Refer*** [***How to set Tests in Postman***](https://toolsqa.com/postman/test-and-collection-runner-in-postman/)
* ***Knowledge of Assertions- Refer*** [***How to write Assertions in Postman***](https://toolsqa.com/postman/test-and-collection-runner-in-postman/)
* ***Knowledge of Chai Assertion Library- Refer*** [***Chai Assertion Library in Postman***](https://toolsqa.com/postman/test-and-collection-runner-in-postman/)

***Assertion:  Check if Cookie Exists***

Here we will check if are getting the cookie that we expect or not. In the test tab, write the following test

*pm.test("Cookies\_Check", function(){ pm.expect(pm.cookies.has('NID')).to.be.true; });*



***NOTE****: We already know that google.com has NID cookie saved. So we are just checking the same through tests. This will not be the case with other servers. So please check it beforehand for other domains.*

The test result will pass signalling that the cookie with the name NID exists in the request.

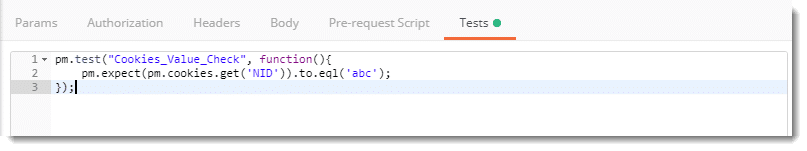


***Assertion: Check for a Value of Cookie***

We can also check for a specific value in a cookie. By this test we confirm that the cookie contains same value that we want to see.

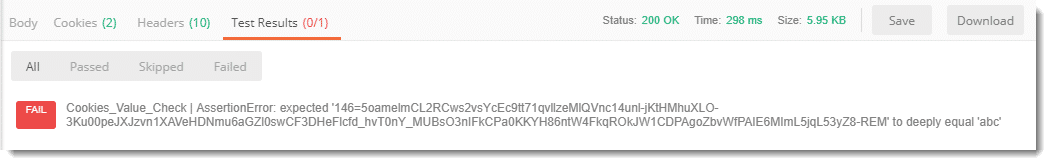
Write the following code in your tests tab

*pm.test("Cookies\_Value\_Check", function(){ pm.expect(pm.cookies.get('NID')).to.eql('abc'); });*



*This code will check if the cookie NID has the value****abc****or not.*

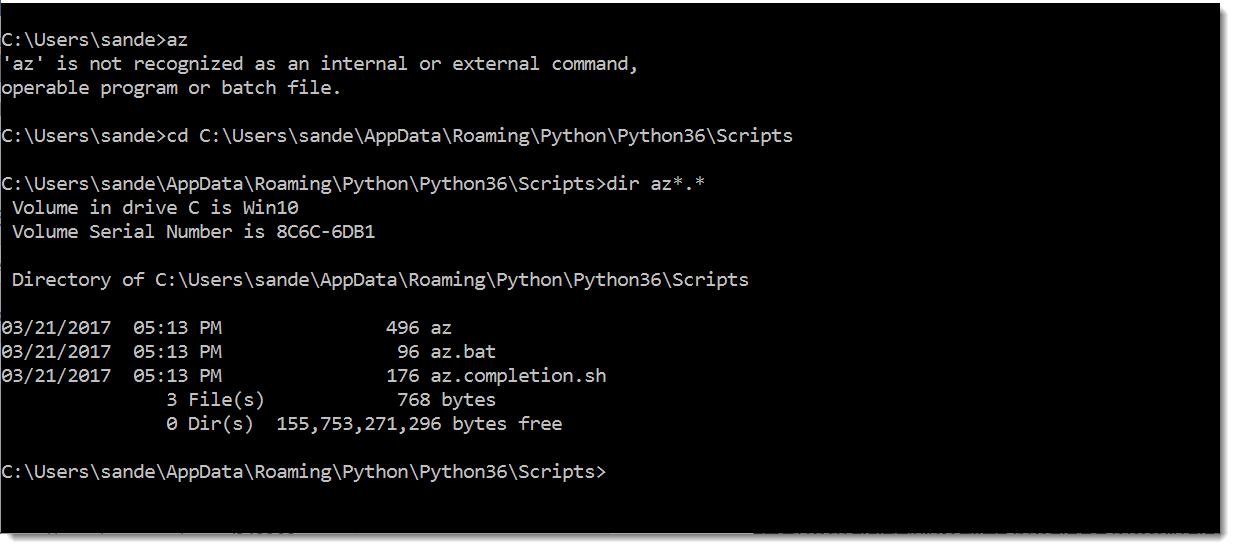
Since this is not the value of NID, we will get a failure status. Also, Postman will tell us the expected value i.e. the correct value of the NID cookie.



### *****What is Command Line Interface?*****

As the name suggests ***a command-line interface is a means of interacting with a computer program (or software) by typing line by line commands in your shell (command prompt or terminal).*** It is fairly common among the people related to computer science but it is also very well known among people who use a laptop or PC for their work.

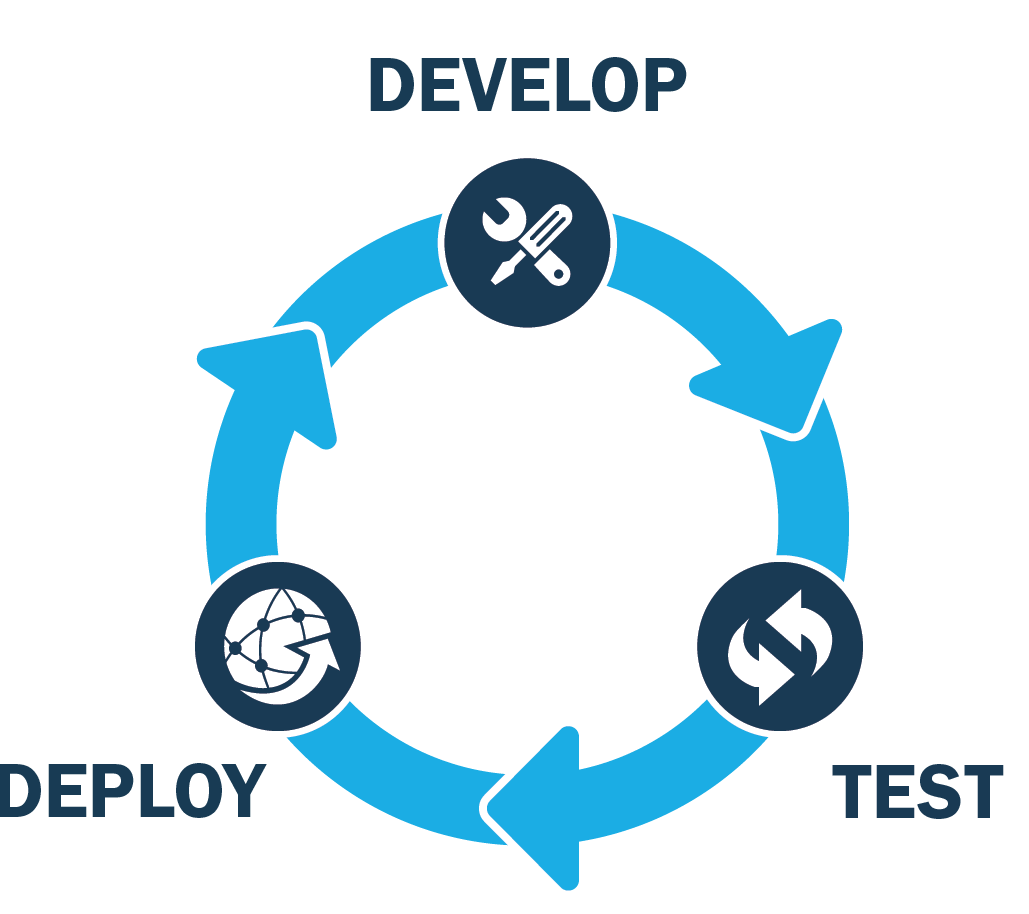
Command-line Interface or CLI was the main (or primary) source of interaction with a computer when the computers were just born. It was common in 1970s and 1980s. A command line interface works through a shell which converts your textual commands into operating system commands or functions to work on it. Since now powerful GUIs are available, it is not so common as it used to be. According to Wikipedia ***Programs with Command Line interface are easier to automate via scripting***. A shell of windows (command prompt) used for operating a program is shown below.



If you are really interested into knowing different softwares which uses CLI, history of CLI and everything else you can visit [***here***](https://en.wikipedia.org/wiki/Command-line_interface).

### *****What is Continuous Integration?*****

Many a times a developer works in a team/group and the team can be distributed in a room, in a same office or over the world. You might not have seen each other but you all work together on the same project developing different modules or modifying the same code base. Now let suppose, you add a new piece of code or edit a old piece of code. This change now needs to reflect to everyone so that they can be aware of the change you did in the program. ***A continuous integration means continuously integrating your changes and program to reflect in the online repository (or cloud)***.



This image shows three stages which is a graphical representation of what I stated above. Deploying is integrating to the main build and testing is the API testing on the new code while development is simply developing and making changes to the code. In a project, Back-end apis are very important for the system, as the whole frontend works on the same API Services. A lot of logic is written for the functionality of API Services.  Developers continuously alter the code and this can affect the API working and test results. Therefore there is a continuous need of executing the tests so that we are assured about the perfect working of APIs. This is done by unit tests which verifies the code issues and Postman tests. Unit tests are written by developers while the Postman tests are written by test engineers. Postman tests verify the actual integration as a whole. As soon as a change is reflected in the code, it is integrated with the main build to perform the API tests on it by which we can be assured that the patch/code is working successfully.

If we talk about the continuous integration in technical terms, a continuous integration is a practice in software engineering where every change is reflected in the larger code base such as a repository so that if there is any defect or modification, it can be identified as soon as possible. Continuous integration is very useful in automated testing, since you continuously alter the code, and the latest code is continuously verified by automated tests.

## What is Newman in Postman?

***Newman is a CLI (Command-line interface) tool which allows you to run a Postman collection directly from the command line***.  According to the official website of Postman, they describe Newman as ***a command-line Collection Runner for Postman***. This makes Postman with Newman a special mix. Newman allows you to run collections in the same way they are executed inside a Postman collection runner. As we discussed in the above section, continuous integration helps us to combine the different codes and execute tests continuously. This amalgam of tests and code is a success due to continuous integration, which in Postman is done by Newman.

* ***Node.js***
* ***NPM (Node package manager)***

But before starting the installation we will see what is NPM in the below section.

***What is NPM?***

Node Package Manager or ***NPM is a package manager for Javascript programming language and is the default package manager for Node.js.*** It is like a repository of projects and has knowledge of what requirements each project has.  According to the official website of npm, It is the world's largest software registry, with approximately 3 billion downloads per week. The registry contains over 600,000 packages (*building blocks of code*).

NPM makes it easy for the JS (*Javascript*) developers to share the code and problems on a repository. This code can then be reused by you in your next project or by anyone who wants the same feature that you have already developed. This makes it super easy for the developers to code better and in a less time.

Although too much knowledge about NPM is not necessary for us but there is much more to NPM than packages and registry. If you are interested in the same, you can visit their website [***here***](https://www.npmjs.com/). We will try to install Newman now and as we discussed above, ***Newman requires node.js and NPM***. So first we will try to install both of these things by following the steps.

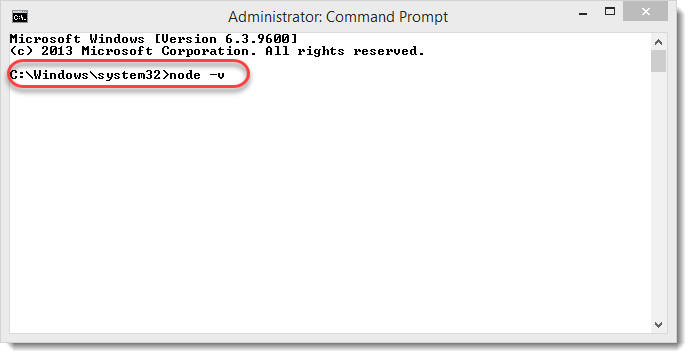
**How to install Node.js**

As mentioned earlier, we already have a tutorial for you to install the node.js on our website. But before visiting the page, you must be sure that you don't have node.js previously installed.

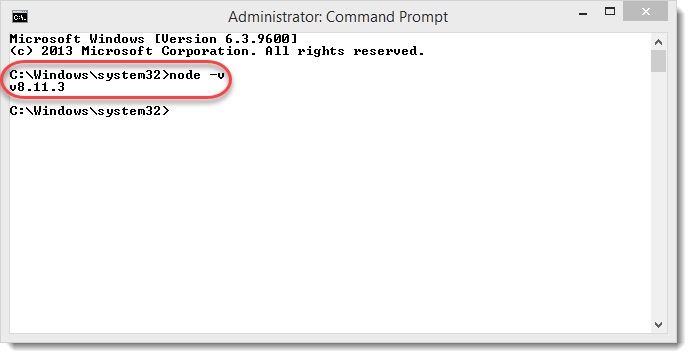
1. Open your ***Command Prompt*** (***Terminal*** for Mac)
2. Type the following

*node -v (****for windows****)*

*node --version (****for mac****)*



1. If you see a version number then you have node.js previously installed and do not need anything else to do.



1. If you see any error or anything else than the version number, then you must install node.js.

Also, if you have followed the steps that were given for the installation of node js on our website, you must have also got NPM installed in your system.

**How to install Node Package Manager**

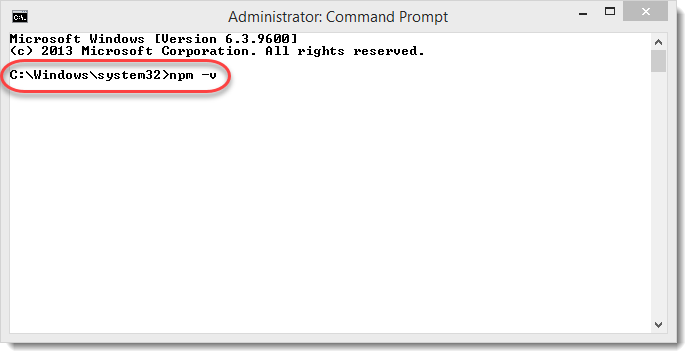
1. Open your ***command prompt*** (*terminal* for mac)

2.Type the following in your command prompt

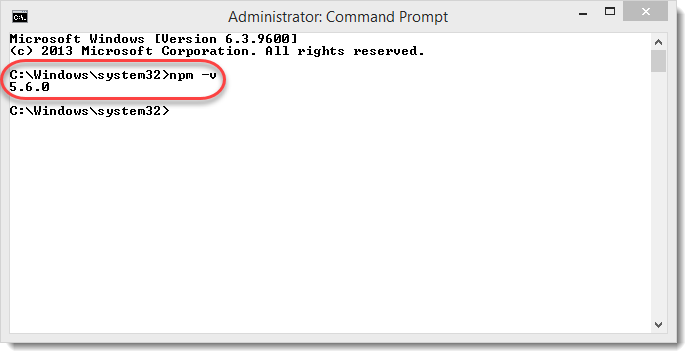
*npm -v in****windows***

*npm --version in****linux/mac***

You can also go through this [***link***](https://www.npmjs.com/get-npm) for download and learning about npm



3.If you see a version number as you press enter, then you already have npm installed and you can proceed further for installing Newman.



If you do not see a version then you might need to install it again from the tutorial on our website and check again.

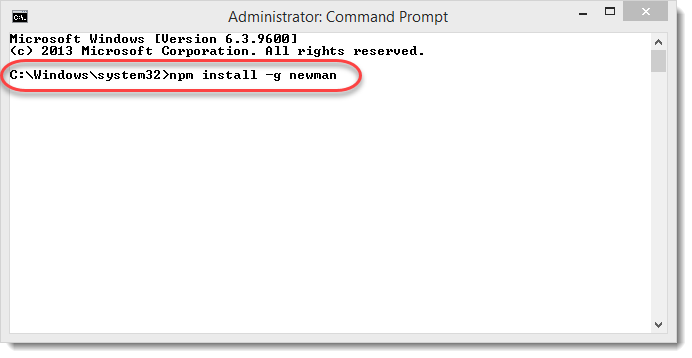
Since now we have both the prerequisites installed, we will now proceed to install Newman on our system.

**How to Install Newman using NPM?**

For installing Newman in your system, follow these steps.

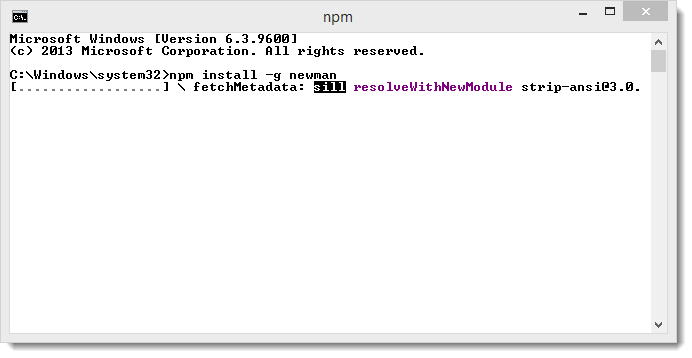
1.Open the ***command prompt*** (\*Terminal for mac)

1. Type *npm install -g newman*



***NOTE****: The command is same for Mac.*

1. This will install a new dependency through NPM. You will see the following screen after pressing enter (*if npm is successfully fetched and installed*).



1. It will take a few minutes to install Newman. Once installed you will be indicated with the following line.

*newman @3.9.4*

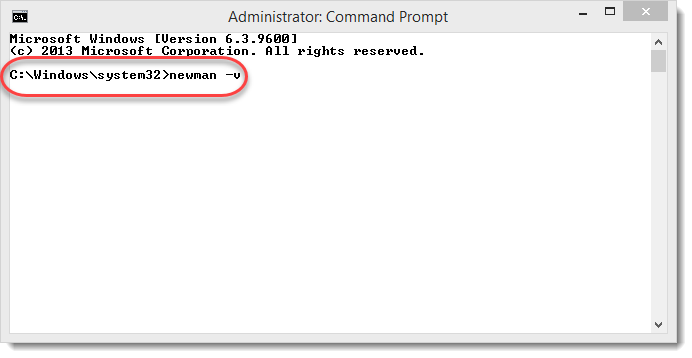
*added 196 packages in 187.889s (Time may vary).*

For confirmation, you can also check the version of Newman.

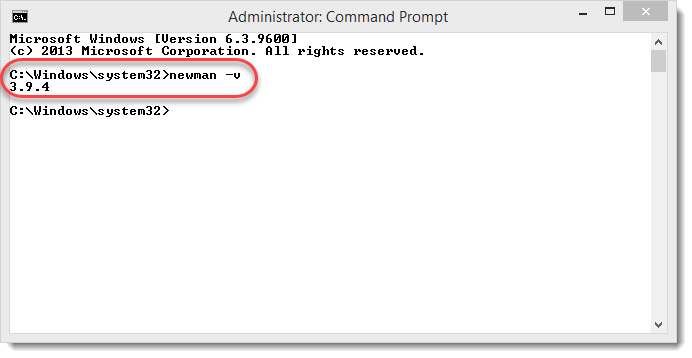
1. Type the following in your ***command prompt*** (*Terminal if Mac*)

*newman -v (****Windows****)*

**newman --version (Mac)**



1. If you see the version number after pressing enter, you have successfully installed Newma or else, it has failed and you must try again.



To start with running a collection with Newman, first you need to have a collection in your Postman. We will be using the same collection that we used in [***Collection Runner***](https://toolsqa.com/postman/test-and-collection-runner-in-postman/) tutorial which contains the following API requests. You can download it and import it in your postman through the following [***link***](https://toolsqa.com/wp-content/uploads/2018/07/Newman-Collection.postman_collection.zip). To use it, make sure you first need to Unzip the folder and upload the .txt file in postman. You can also refer the tutorial to follow the steps to [***import collection in postman***](https://toolsqa.com/postman/collections-in-postman/)

Now, we have added all the requests in our collection ***Newman Collection*** and everything is running fine in Postman. It's time for us to try to execute everything in our collection from Newman.

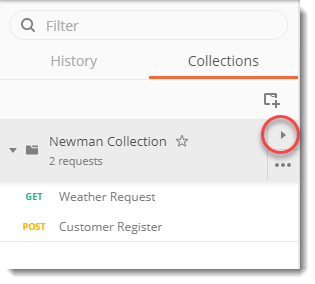
To run a collection through Newman, we have two ways to proceed.

* ***Through the Share Link***
* ***Through the Json File***

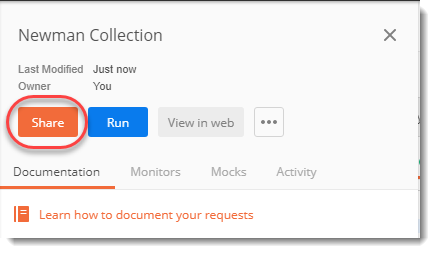
We will run the collection with both the methods.

**Running the collection using Newman through share link**

1.Click on the arrow besides the collection name.



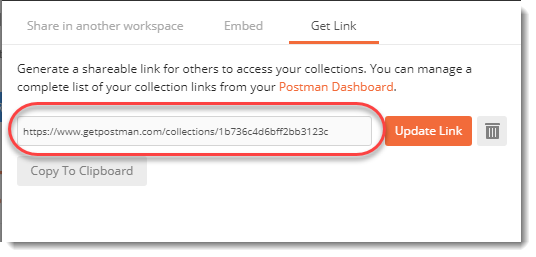
1. Click on ***Share***.



1. Click on ***Get Link***



1. Copy this link

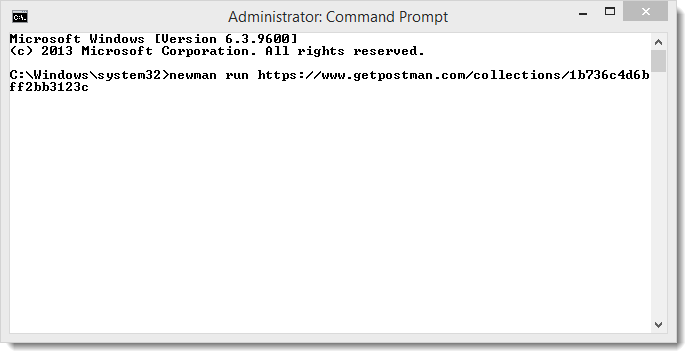


5.Open your shell (*command prompt for windows and terminal for mac*)

***NOTE****: We will be using the word shell which is technical word for terminal from now onwards*.

1. Type the following:

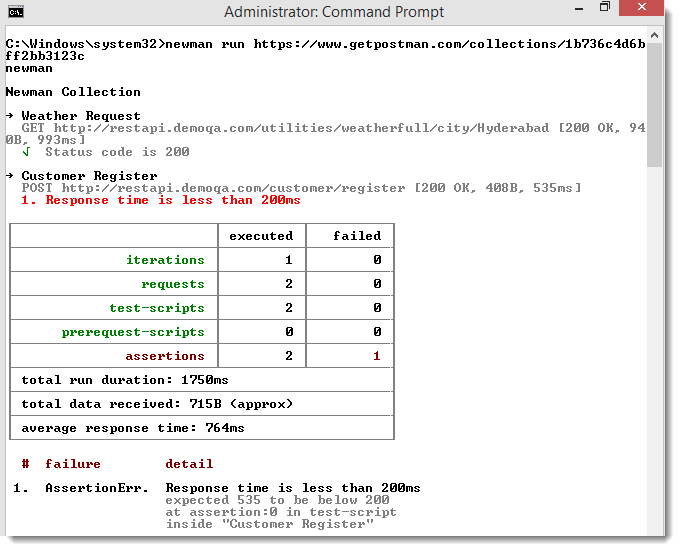
*newman run <link>*



***NOTE****: Please input your own link which will vary from the above that we used*.

1. Press ***enter***.

Your collection has successfully executed if you see the following screen



Once the collection has executed you will see the tests details as we saw in collection runner in Postman. We had one test for each of our requests, hence we see the results accordingly. It can also be seen from the image above, the details are similar to collection runner. ***We can see response status (time, size and status code) of each requests along with the test scripts that we executed***.

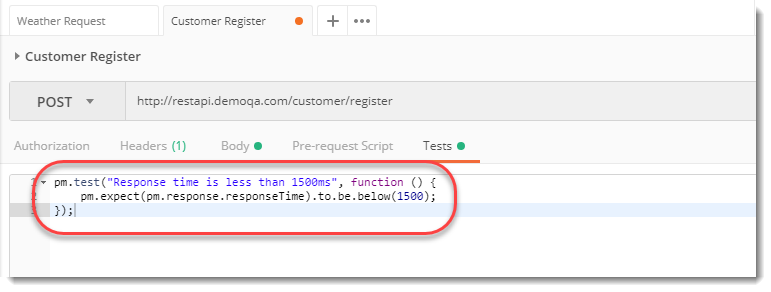
Since in the weather API request we asserted the status code to be 200, which turned out correct it was not the case in customer register API. In the customer register API we asserted that the response time to be less than 200ms which turned out to be 535ms and hence, false. It can also be seen by the red line under customer register api.

Now, let us try to change the same assertion in the customer API and see the results.

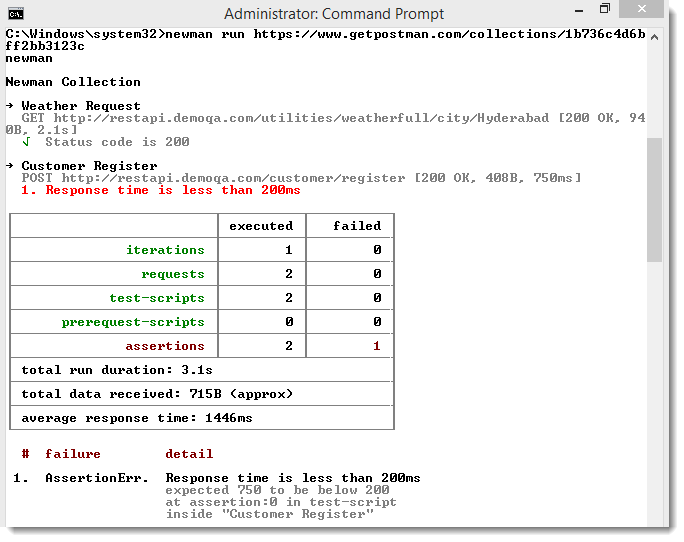
***How to update the link***

1.Go to ***customer register API***.

2.***Change the time to be 1500ms*** in the tests.



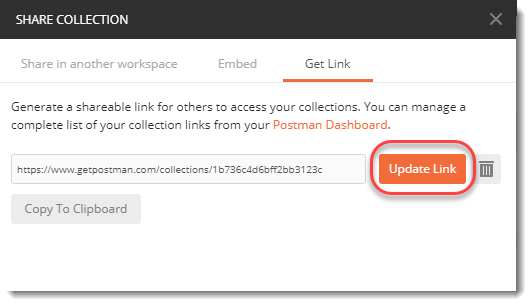
3.Save the request and run the same command again and press enter.



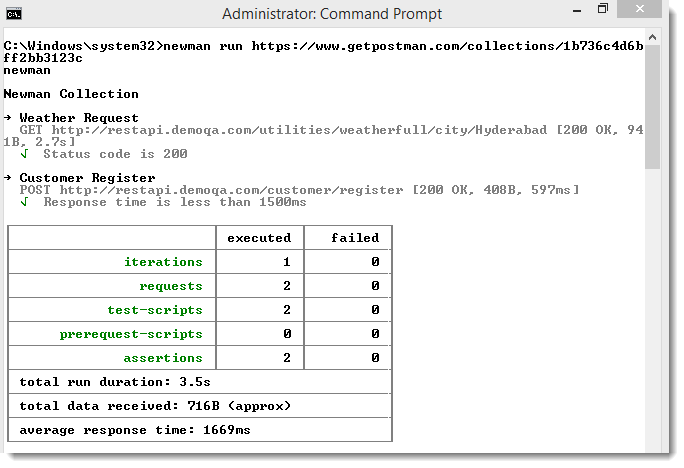
***Wait! It is not the same result that we expected***. This time the response time is 750 ms which is less than 1500 ms but still we get the same error with the same assertion line i.e. ***Response time is less than 200 ms***.

This has happened because of the link not been updated. If you want to run it again then you need to follow the same steps as we did above to get the link.

This time, ***click on update link***.



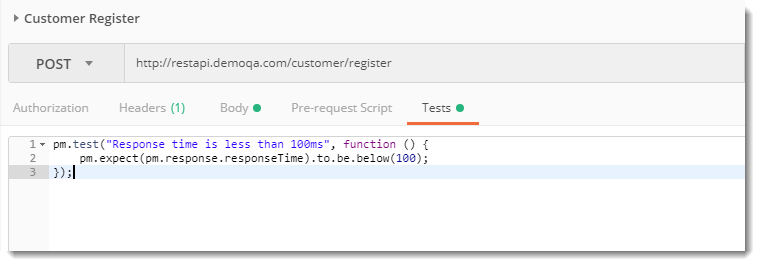
Copy the new link and run the same command in command prompt (*windows*) again.



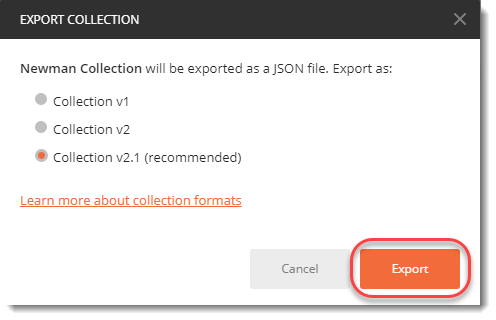
Voila! We have now got the updated and expected results out of our tests. So ***always remember to update the link once you have made the changes***. But there is a problem in this. This cannot work while working in the team. The link just acts as a snapshot of the Postman. Once you update something in the Postman it does not gets updated automatically until you update the link. While working in teams, changes are always happening and running through links makes your API more prone to the errors. So there is another way of testing the APIs through Newman.

**Running the collection using Newman through JSON file**

1.Now, to get an error we will change the response time in the customer register API to be 100ms. ***Save this API***.

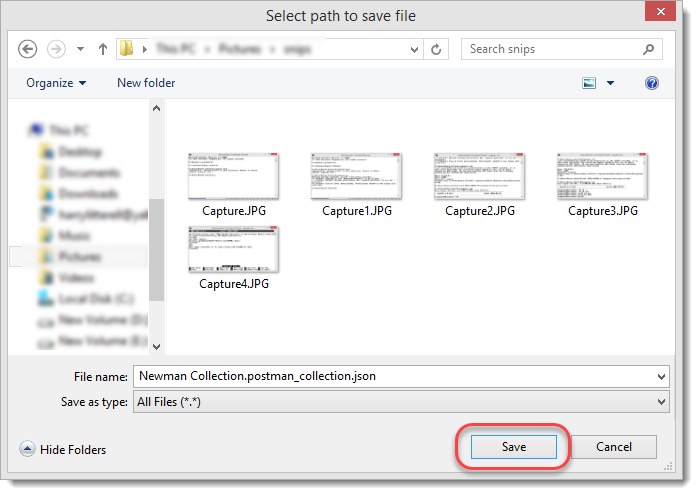


2.Click on the export link alongside the collection name (*learn in*[***Collections in Postman***](https://toolsqa.com/postman/collections-in-postman/)) and click on ***export*** in the following panel.



***Note****: Always use collection v2.1 which is recommended as discussed in the*[***Collections in Postman***](https://toolsqa.com/postman/collections-in-postman/)*chapter*.

1. Save the json file in your system and ***remember the directory***.

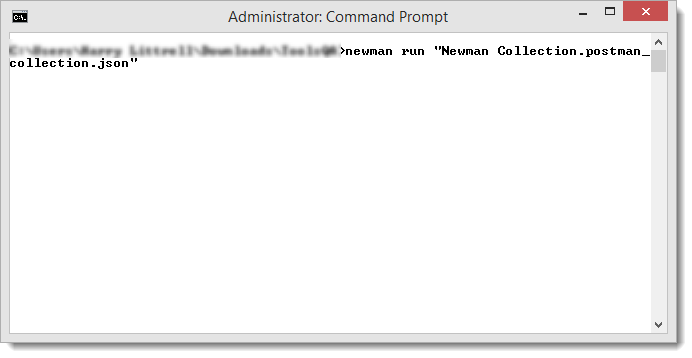


1. Once you save the json file, visit the shell of your system and ***change the current directory to the directory in which you saved this json file***.

*For example: If you saved the json file in C:\harish then change the directory to C:\harish*

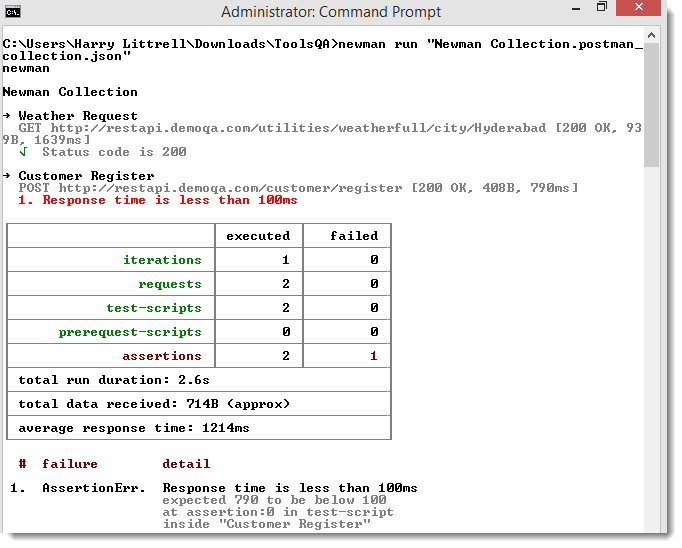
1. After changing the directory, run the following command

*newman run <name of the file>*



***Note****: Remember to place the file name in inverted commas otherwise the shell ill consider it as a directory name*.

1. Press ***enter*** and you will see the expected results of your collection Newman collection



By this, we have successfully executed the collection through Newman. Error is self understandable as discussed in above sections. It was quite fun working with Newman and running our first collection from the shell rather than from the Postman itself.

**Newman Optional Parameters & Configurations**

Till now we only learnt that we have a collection and we want to run it through Newman. In this tutorial as I mentioned we will be setting some other collection features to apply them to our collection and then run. So when we have to apply ***[options]*** (*as the Newman website says*) there is a particular syntax we follow, and this will help you remember everything we learn later in this tutorial in an easy way.

The newman syntax is as follows

*newman run <collection-file-source> [option]*

There are many ***options*** that can be applied to a collection through Newman or Postman app for example setting up an environment variable or specify the  . While we have learnt it in Postman application, there is a need to use them using Newman because we cannot set these options through the app and run it through Newman. So the options has been divided into four parts

* ***Utility****: This include help and getting the version*
* ***Basic Setup****: This contains setting up different options in your collection such as environment.*
* ***Request Options****: These are the options which directly affect the requests such as delay request (specifying delay between requests)*
* ***Misc****: These are other small options that does not fall in any other category discussed above. These include like disabling the color of the interface etc.*

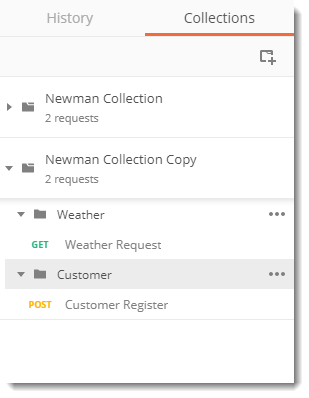
You can learn about each and every option on the [***Postman Newman documentation.***](https://www.getpostman.com/docs/v6/postman/collection_runs/command_line_integration_with_newman)

***Running a folder inside a collection using Newman***

In the [***collections***](https://toolsqa.com/postman/collections-in-postman/) tutorial we learnt about the folders inside a collection. ***A folder can be created inside a collection to combine similar APIs for better understanding***. *For example, If you have a folder named movies (similar to collections in Postman) then you can have two different folders inside it named Hindi movies and English movies (similar to folders inside a collection).* But in a real and big project we have huge number of APIs inside a collection, so we segregate those into different folders. There can be a need when someone like to just test one set of APIs which are in single folder. In that case there is no point to execute all the collection as a whole, as it will execute all the folder which comes under the collection. Just like Postman, Newman also gives us the ability to run just a folder from collections.

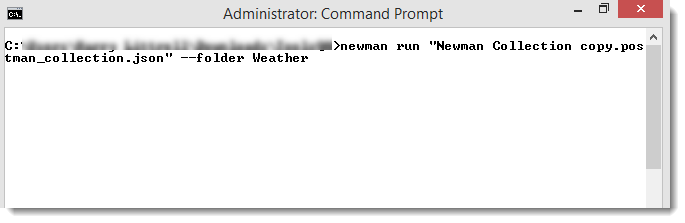
1.Using the same collection that we used in the previous tutorial of  ([***Running Collections with Newman***](https://toolsqa.com/postman/running-collection-using-newman/)), make two folders and move each request to a folder of their name (***it would be better if you make copy of Newman Collection since we need to get to original setting after this section***)

*For example: Move the customer Register API to Customer Register folder and Weather API to a weather API folder as shown.*



1. ***Export your Collection*** as JSON, as discussed in [***Running Collections with Newman***](https://toolsqa.com/postman/running-collection-using-newman/).
2. Go to the shell of your system and type the following command:

*newman run <collection\_name> --folder <folder name>*



1. Press ***Enter*** and see that the folder you wanted has been executed successfully.

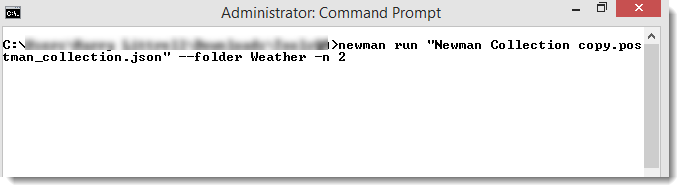


***Setting test iterations using Newman***

In the collection runner tutorial we learnt to set iterations on our collection which was actually the number of times our collection will run repeatedly. ***Iteration value set to 5 will execute all the APIs five times***. We will try to achieve the same here by following the steps:

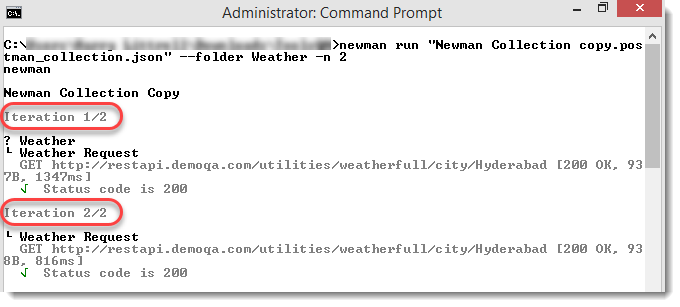
1. We will use the same exported json collection file that we did above.
2. Type the following command.

*newman run  <collection\_name> -n <number of iterations>*



***Note****: We are setting the iteration value to 2 here and running it on weather folder only.*

1. Press ***Enter*** and you will be able to see all your tests and APIs being executed two times.



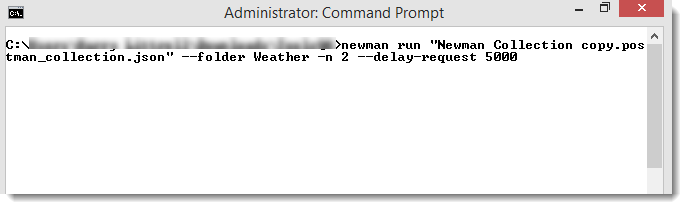
***Setting delay using Newman***

Delay is also discussed in the Collection Runner tutorial, ***that delays are the time intervals between execution of each iteration***. So a delay of 2 seconds will run the folder again after every 2 seconds. We will try to achieve same through Newman by following the steps.

1.We will be using the same exported JSON file that we are using till now.

1. Go to your shell and type the following command.

*newman run <collection\_name> -n 2 --delay-request 5000*



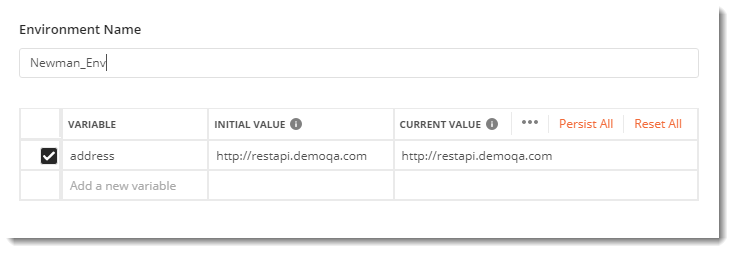
***Note****: We are using 5s delay and running on only one folder. You can run it on both or other folder.*

1. Press ***enter*** and you can see that the second execution of collection runner was after a delay of 5 seconds.

***Setting environment variables using Newman***

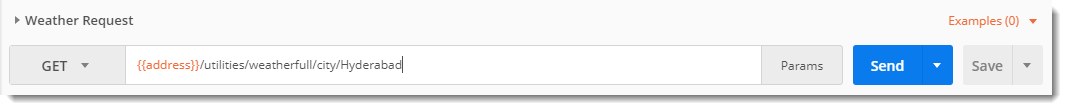
Just as we used the environment variables in Postman, we can also set the environment variables in Newman.

First of all you need to create one environment called ***Newman\_Env*** in Postman, which has only one environment variable ***address*** with the ***value of the address of url*** as shown.



***Note****: Refer the tutorial of*[***Environment Variables in Postman***](https://toolsqa.com/postman/environment-variables-in-postman/)*for help.*

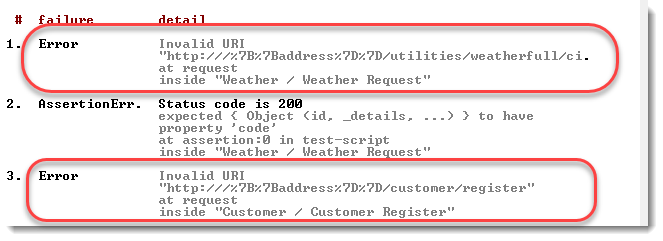
Now change the domain of url in the request to the variable ***address*** and see if both the tests passes and variables work or not in your postman application. Both test will work, as the environment variable is set in the Postman and tests are being executed with in the Postman.



Both test will work, as the environment variable is set in the Postman and tests are being executed with in the Postman.

Now we will try to run the same in Newman. For this you need to export the collection again since this copy is different from what you already have in your system.

Go to shell and again run the same command in Newman. You will encounter error now.



***Note****: Please refer to this tutorial for learning about running a collection in Newman.*

***Newman does not know what is address and hence throws an error INVALID URI***. This has happened because address is an environment variable stored in an environment of which Newman has no idea about. So it won't run until we specify this environment specifically in Newman. So here we go.

Remember what we learnt in the above section about specifying the options to run along with the collection. We will go with the same syntax here. ***For specifying environment --environment option is used***, so the complete syntax becomes

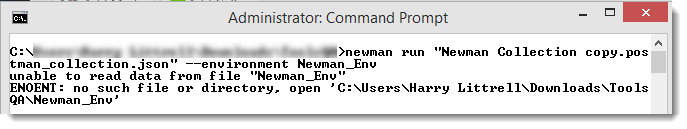
*newman run <collection> --environment <file>*

Let's start by specifying the environment, Write the following code in Newman:

*newman run <collection> --environment Newman\_Env*



Press ***enter*** to see the result.



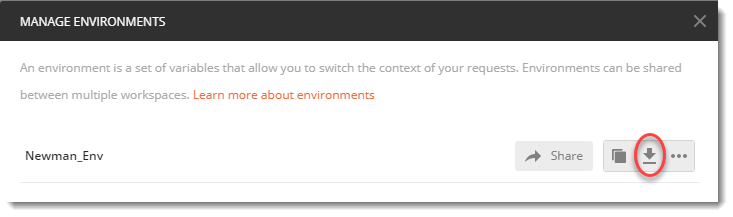
We are still getting the error. It is so because Newman does not work that way. For telling anything to Newman such as setting the environment, we can only do so by specifying the ***file*** which has environment and not environment directly. ***For this we need to export the environment***.

***How to export the environment***

Exporting the environment is very simple and straightforward as it was for exporting the collection. Follow the given steps to export an environment.

1.Go to ***settings****(****gear icon****)*

1. Alongside the environment name, you will see a ***download icon****(****downward arrow****)*



1. Press the icon and download the environment.

Now we have our environment downloaded, we can continue to set the environment using Newman. ***Remember to save the environment file in the same location as you have your collection***. Since we change the directory before running the collection, our system does not allow newman to access the file out of that directory.

Write the following code in Newman.

*newman run <collection> --environment <file>*

