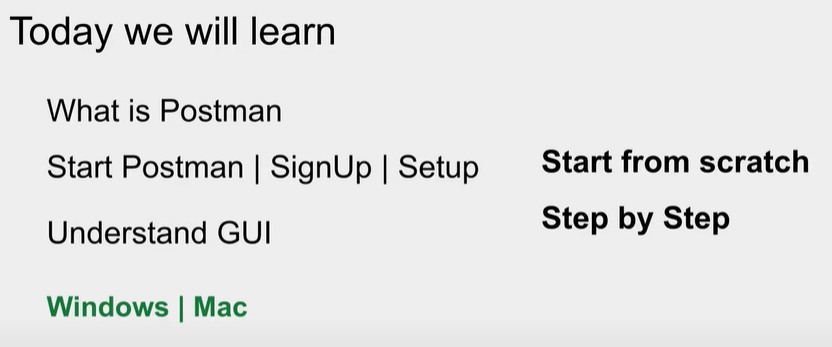
**Postman Desktop APP Version 9.27.0**

**Postman version 9.28.1**





-Postman is tool used for API Development,API Testing and API Management.

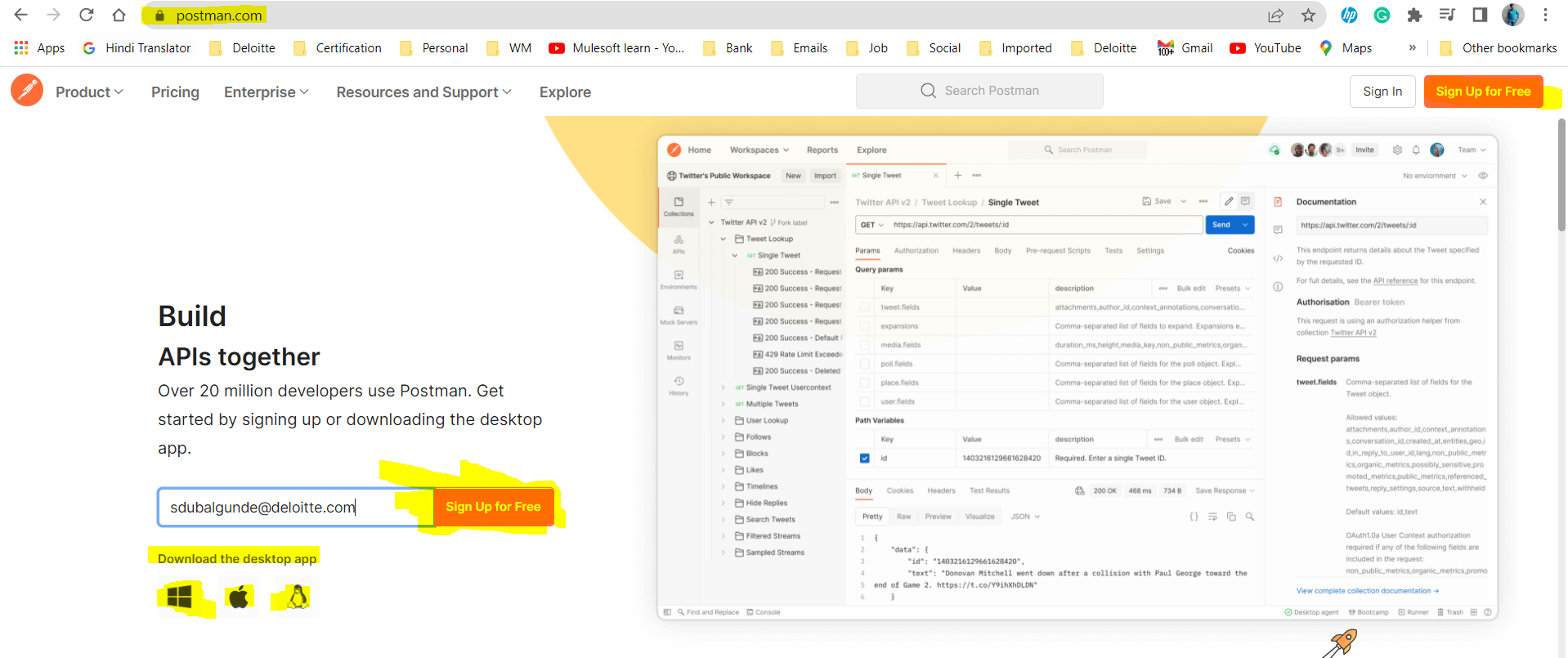
-Postman can be access via two methods:

1. Via Web (Thru Browser you can use Postman)
2. Via APP (thru downloading desktop version of the Postman App)

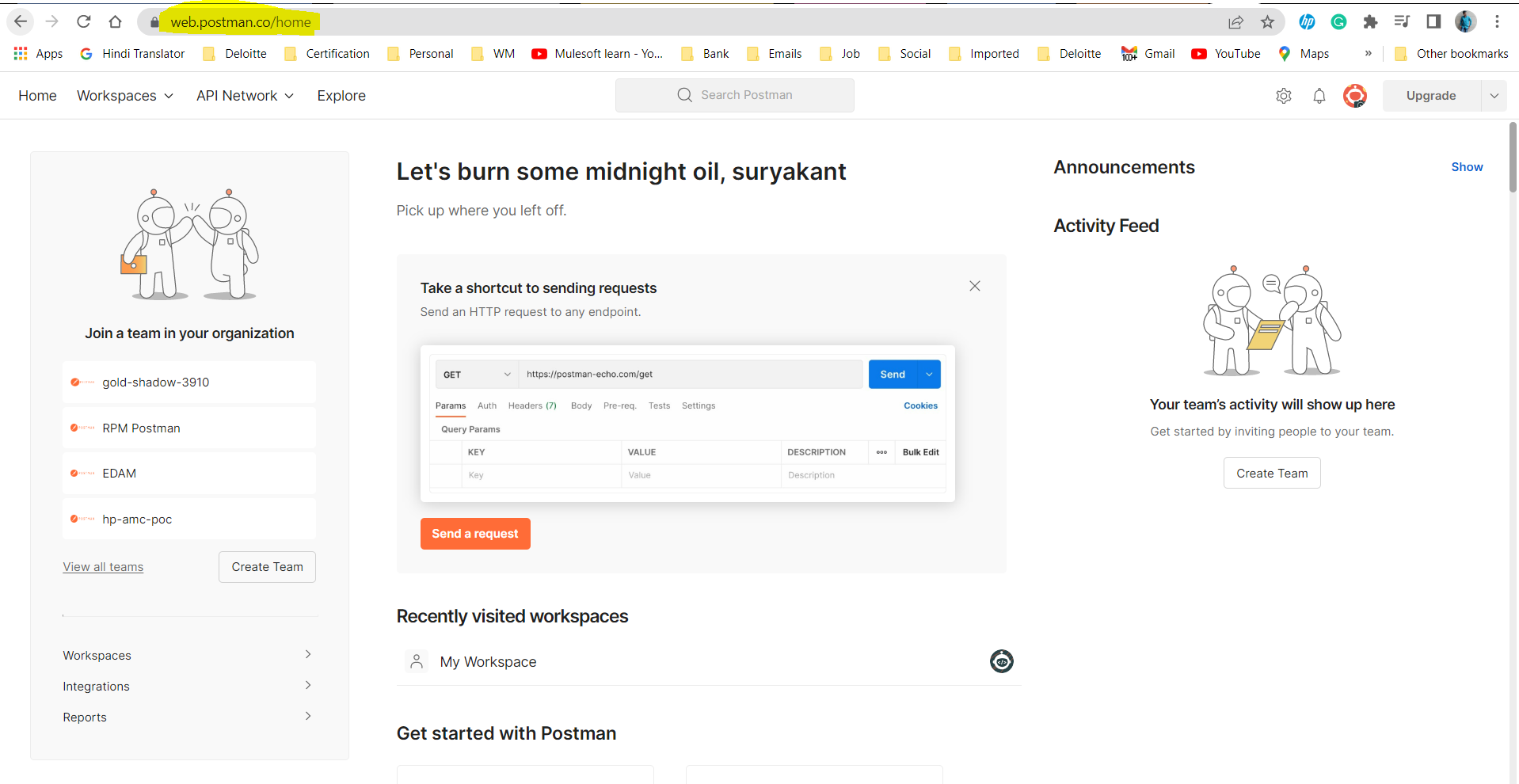
-To work with postman you need to create Postman account.

-Postman account can be created as well as postman Desktop App can be downloaded from [Postman Site To Create Account & Download The Postman App](https://www.postman.com/downloads/) .

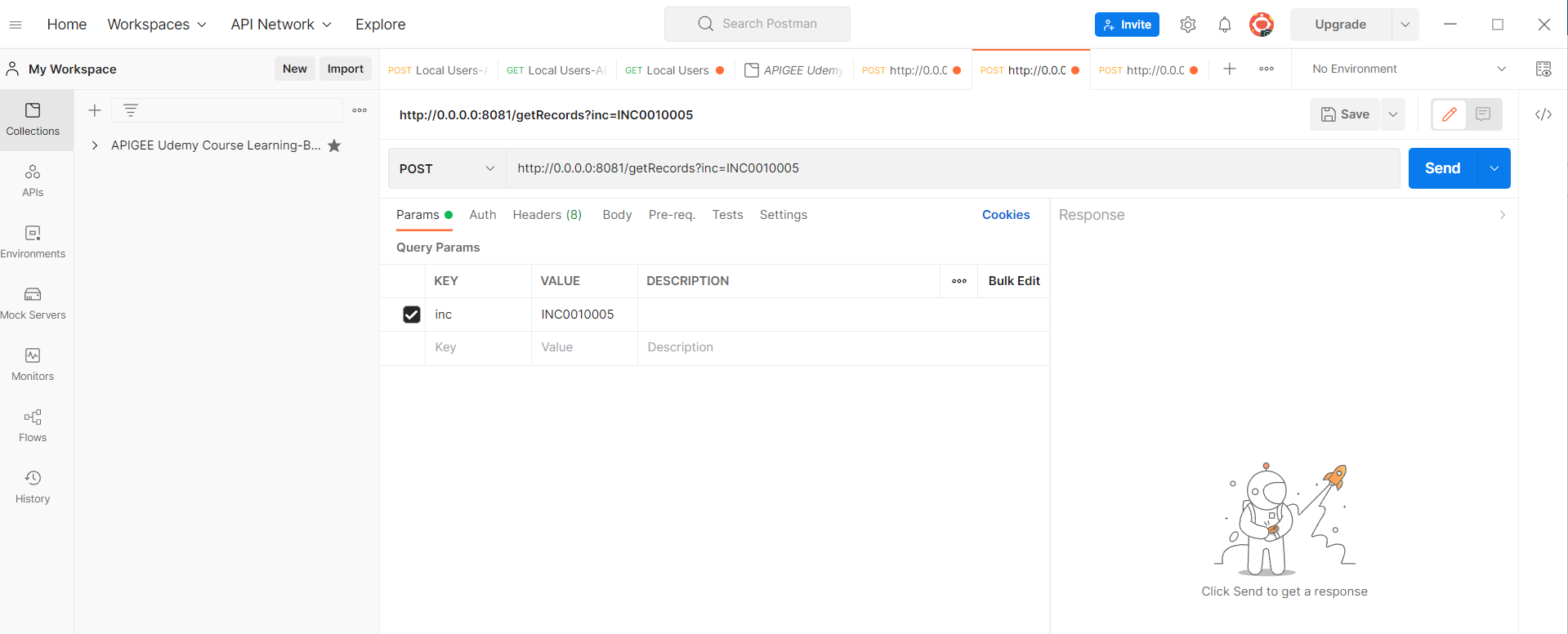
Once you click on the above link,it will go to postman.com site there @top right cornor you can see sing up for free or sign in button.To create your account newly select sign up for free then new page will open where you need to provide basic details like username,email id(where you will get verify account mail) n password then click on create account.



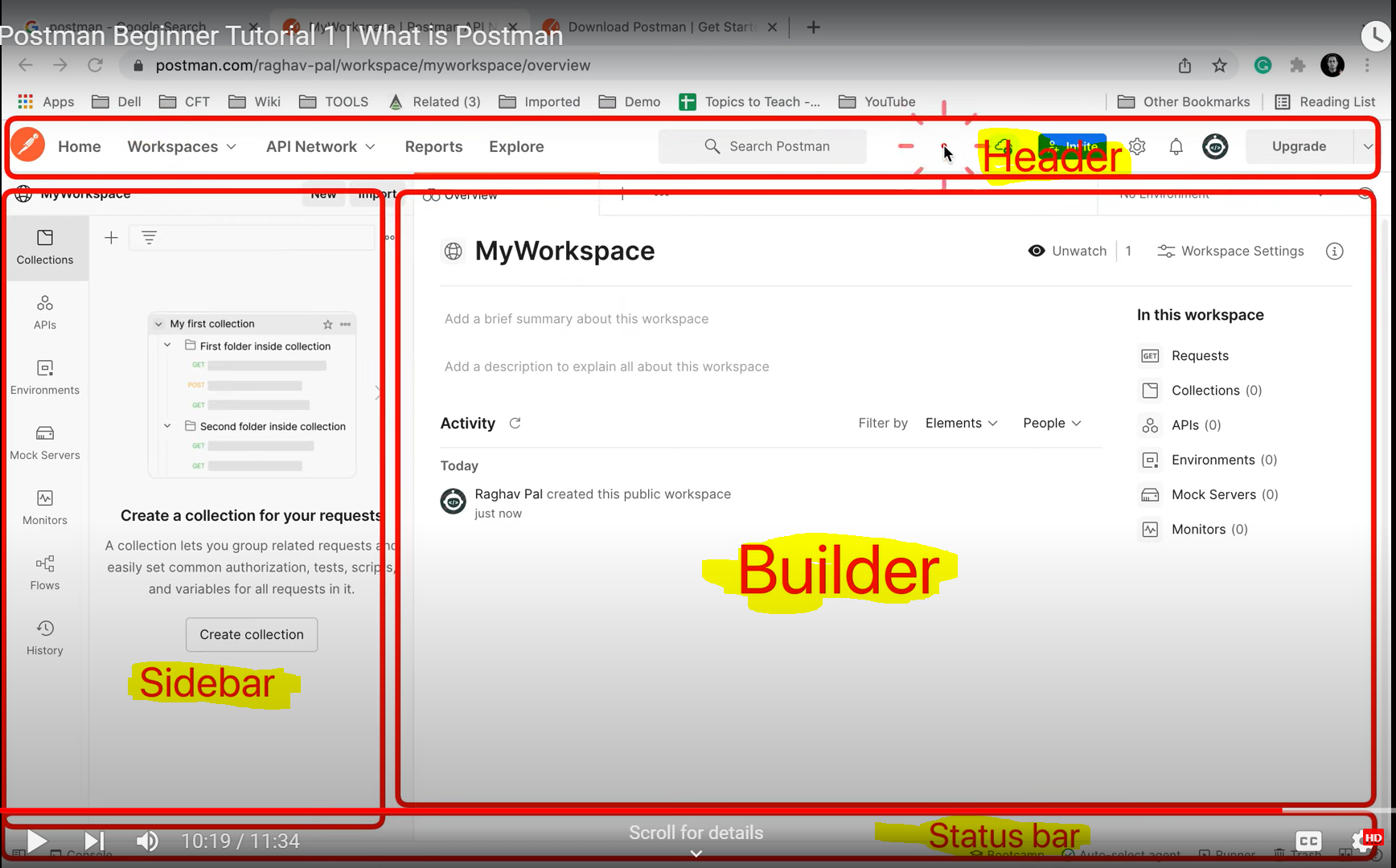
Web Version Of Postman:-



App Version of Postman:-



Various Section of Postman:-



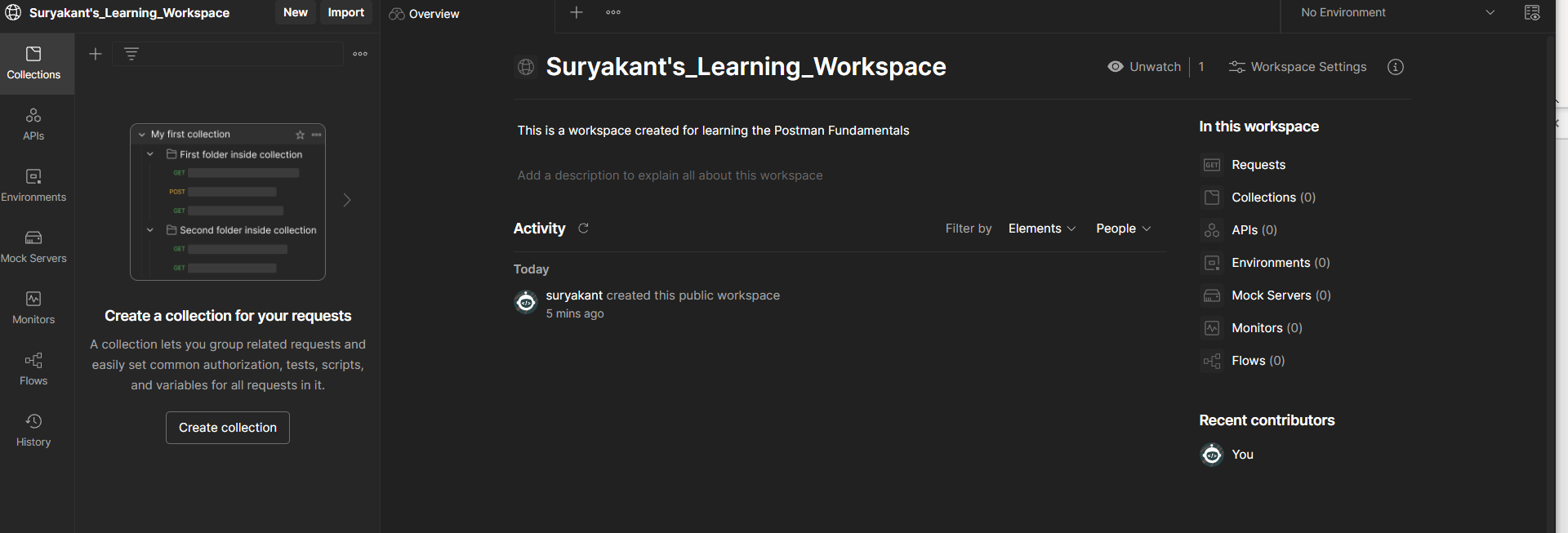
-How to setup your working directory in Postman:-

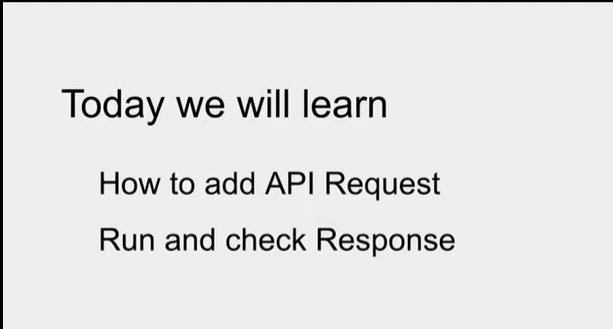
Click on Gear Icon->Go to settings->General->Working Directory->Choose your Path.

-How to create a workspace?

A workspace is a placeholder which can hold requests,collections,APIs,Mock Servers etc.

Click on New in sidebar or go to workspace in header->Create Workspace->Enter name,description and set the visibility level(personal(only u can access),public (Everyone can access),teams(all team members can access),Private(only invited team members can access)).



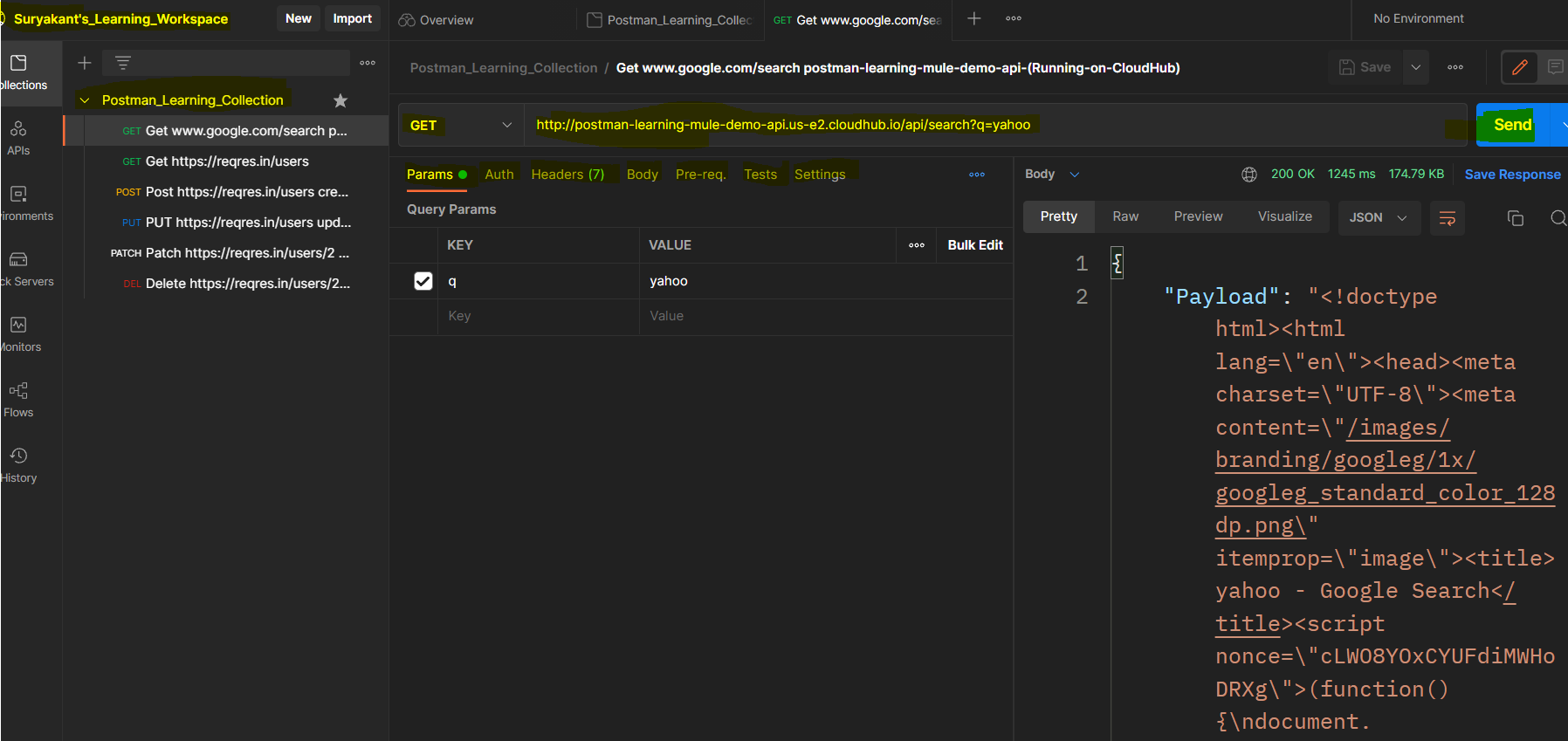


-After creating Workspace,We need to have Collection created for storing or saving the requests in organized manner.

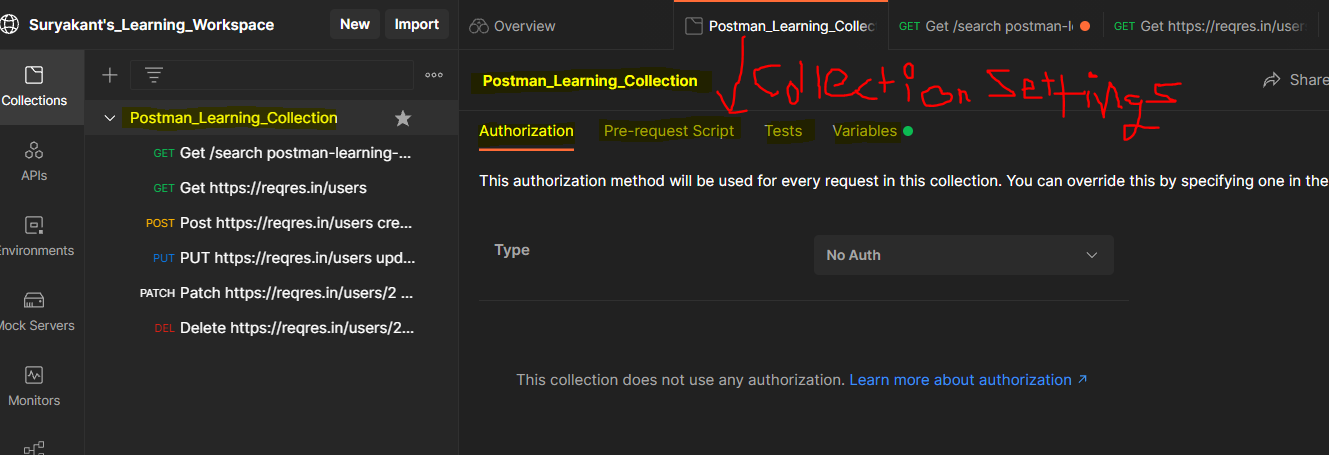
A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.

How to create a collection?

Click on New in sidebar or click on create collection (+)button in sidebar



We can have many requests under 1 collection which can use collection-level defined authentication policy,Pre-request script (script will execute before every request in this collection) ,Tests(will execute after every request in this collection),variables(variables specific to this collection and its request).



-Authorization:

**Authorization is a policy used to authenticate the request whether request is coming from proper src or not.**

We can specify Authorization @collection-level as well as @Request-level

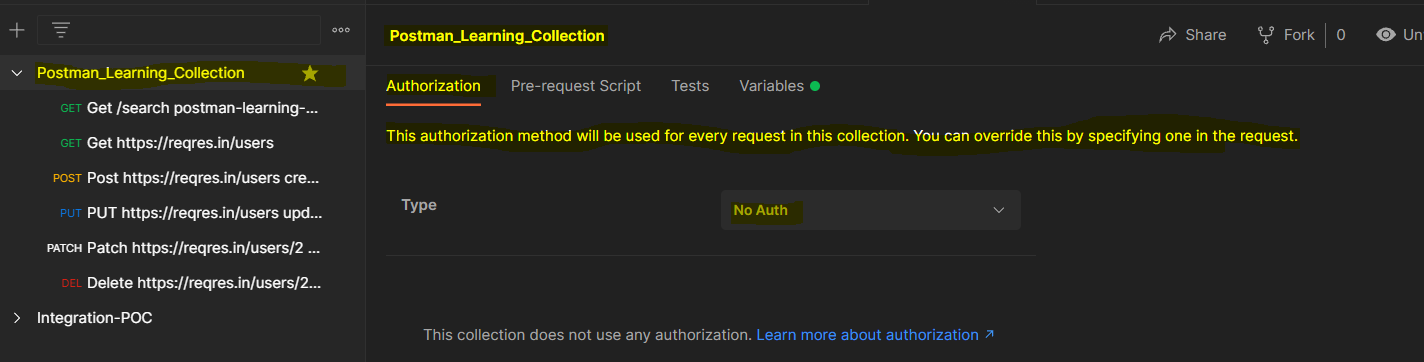
We can specify @collection-level Authorization by clicking onto the collection name.

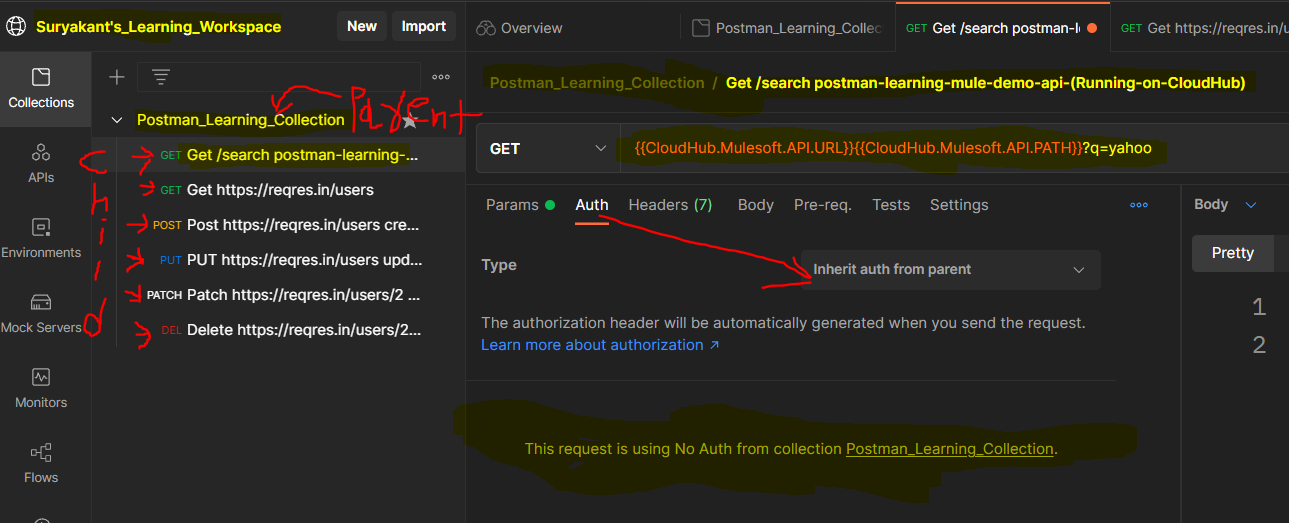
We can specify @Request-level Authorization by clicking onto the request name.

**Authorization setup @collection-level can be used by every request specified in the collection that means Authorization setup setted @collection-level is automatically applied on all the requests that are inside the collection but if you want you can change the authorization setup @request-level to allow particular request in a collection to use basic auth or another request to use no auth or another request to use oauth 2.0 or another request to use the same auth policy which is defined @collection-level by specifying inherit auth from parent etc.**

Let say @ collection levl we setup Authorization as Basic Auth

If we want a particular request in a collection to use different Authorization policy(let say OAuth 2.0) we can do that by choosing OAuth 2.0 @request-level authorization setup or if you want to use same Auhorization setup setted @Collection-level then in a particular request you can select iinherit auth from parent option from dropdown or if you want a particular request to don’t use any auth policy then you can choose no auth from dropdown.





-pre-request-script:

**pre-request-script are scripts in JS(Javascript) that runs before the request is sent to the server.**

We can specify pre-request-script @collection-level as well as @Request-level

We can specify @collection-level pre-request-script by clicking onto the collection name.

We can specify @Request-level pre-request-script by clicking onto the request name.

**pre-request-script setup @collection-level can be used by every request specified in the collection that means** pre-request-script **setup setted @collection-level is automatically applied on all the requests that are inside the collection that also means first @collection-level pre-request-script will execute n then it will execute the @Request-level pre-request-script if any defined in pre-request-script tab @Request-level. you can define pre-request-script @ both level n the execution order for pre-request-script is like**

**pre-request-script @collection-level(if defined)**

**pre-request-script @Request-level(if defined)**

**-------------------------------------------------------------------**

**if pre-request-script @collection-level is defined n pre-request-script @Request-level is not defined then**

**pre-request-script @collection-level(defined) is executed**

**pre-request-script @Request-level( not defined) will be ignored**

**if pre-request-script @collection-level is not defined n pre-request-script @Request-level is defined then**

**pre-request-script @collection-level( not defined) will be ignored**

**pre-request-script @Request-level( defined) is executed**

**if pre-request-script @collection-level is not defined n pre-request-script @Request-level is also not defined then**

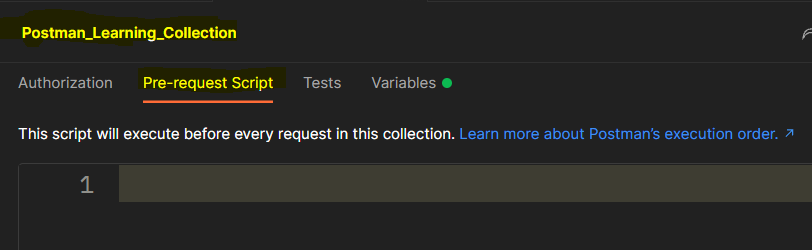
**pre-request-script @collection-level( not defined) will be ignored**

**pre-request-script @Request-level( not defined) will be ignored**

**if pre-request-script @collection-level is defined n pre-request-script @Request-level is defined then**

**pre-request-script @collection-level(defined) is executed**

**pre-request-script @Request-level( defined) is executed**



-Tests:

**Tests are scripts in JS(Javascript) that runs after the response is sent back to the postman.**

We can specify Tests @collection-level as well as @Request-level

We can specify @collection-level Tests by clicking onto the collection name.

We can specify @Request-level Tests by clicking onto the request name.

Tests **setup @collection-level can be used by every request specified in the collection that means** Tests **setup setted @collection-level is automatically applied on all the requests that are inside the collection that also means first @collection-level** Tests **will execute n then it will execute the @Request-level** Tests **if any defined in** Tests **tab @Request-level. you can define** Tests **@ both level n the execution order for** Tests **is like**

Tests **@collection-level(if defined)**

Tests **@Request-level(if defined)**

**-------------------------------------------------------------------**

**if** Tests **@collection-level is defined n** Tests **@Request-level is not defined then**

Tests **@collection-level(defined) is executed**

Tests **@Request-level( not defined) will be ignored**

**if** Tests **@collection-level is not defined n** Tests **@Request-level is defined then**

Tests **@collection-level( not defined) will be ignored**

Tests **@Request-level( defined) is executed**

**if** Tests **@collection-level is not defined n** Tests **@Request-level is also not defined then**

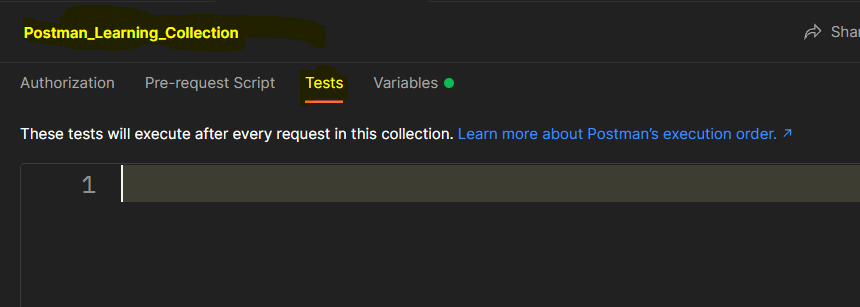
Tests **@collection-level( not defined) will be ignored**

Tests **@Request-level( not defined) will be ignored**

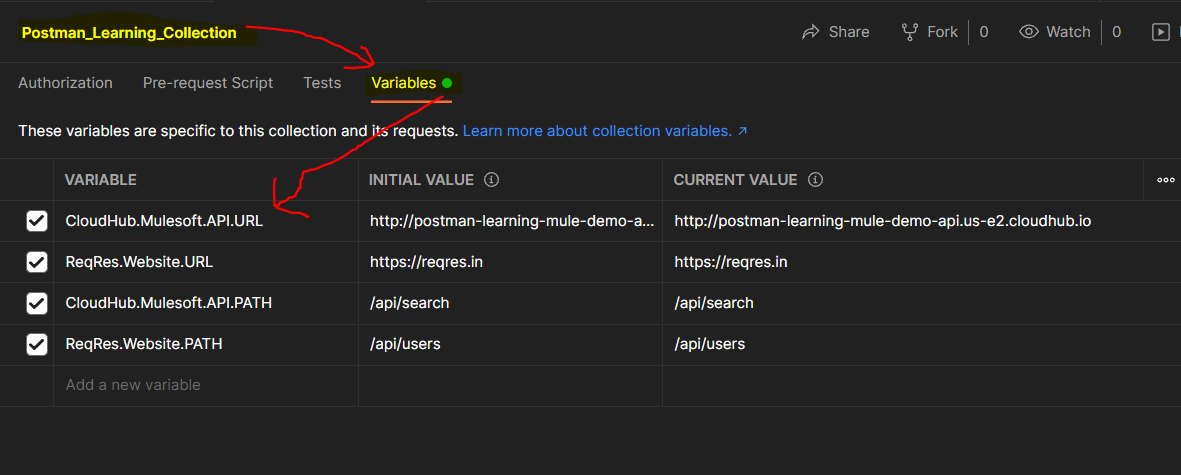
**if** Tests **@collection-level is defined n** Tests **@Request-level is defined then**

Tests **@collection-level(defined) is executed**

Tests **@Request-level( defined) is executed**

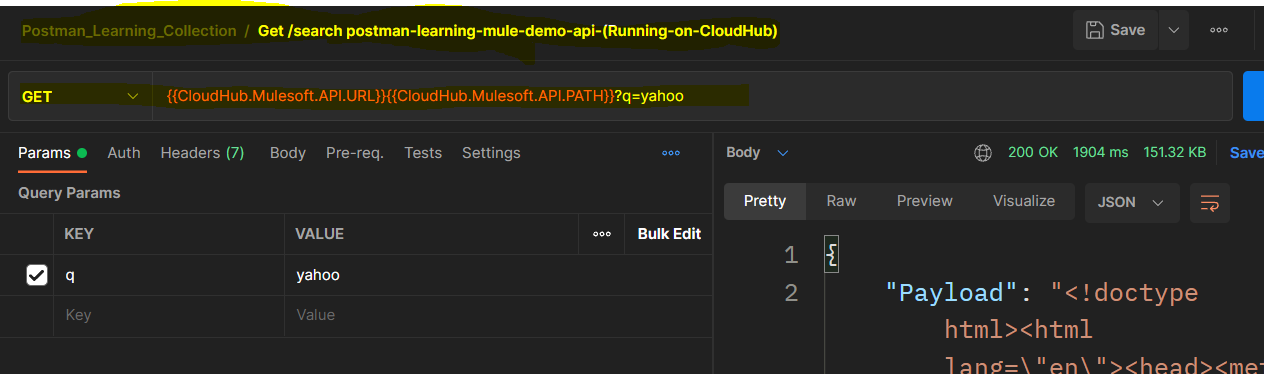


-Variables:



The above image shows all the variables declared @collection-level which are nothing but collection variable that can be accessed only at that collection-level in which collection they have been declared.that means if we have one collection called “demo” n we created some collection variables @Demo collection so that variables can be accessed inside only in @Demo Collection.They are not be accessible outside of @Demo collection that is inside different collection or inside a complete workspace.

Use of Collection variables defined at Collection level settings in one of the request:



Variables types:

**Global variables** enable you to access data between collections, requests, test scripts, and environments. Global variables are available throughout a [workspace](https://learning.postman.com/docs/collaborating-in-postman/using-workspaces/creating-workspaces/).

**Collection variables** are available throughout the requests in a collection and are independent of environments. Collection variables don't change based on the selected environment. Collection variables are suitable if you're using a single environment, for example for auth or URL details.

**Environment variables** enable you to scope your work to different environments, for example local development versus testing or production. One environment can be active at a time. If you have a single environment, using collection variables can be more efficient, but environments enable you to specify [role-based access levels](https://learning.postman.com/docs/sending-requests/managing-environments/#working-with-environments-as-a-team).

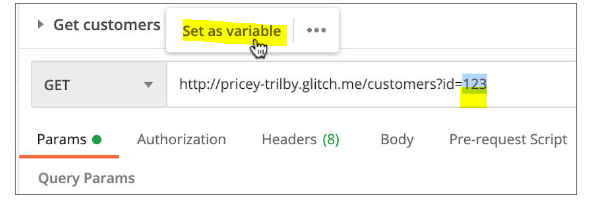
**Data variables** come from external CSV and JSON files to define data sets you can use when running collections with [Newman](https://learning.postman.com/docs/running-collections/using-newman-cli/) or the [Collection Runner](https://learning.postman.com/docs/running-collections/intro-to-collection-runs/). Data variables have current values, which don't persist beyond request or collection runs.

**Local variables** are temporary variables that are accessed in your request scripts. Local variable values are scoped to a single request or collection run, and are no longer available when the run is complete. Local variables are suitable if you need a value to override all other variable scopes but don't want the value to persist once execution has ended.

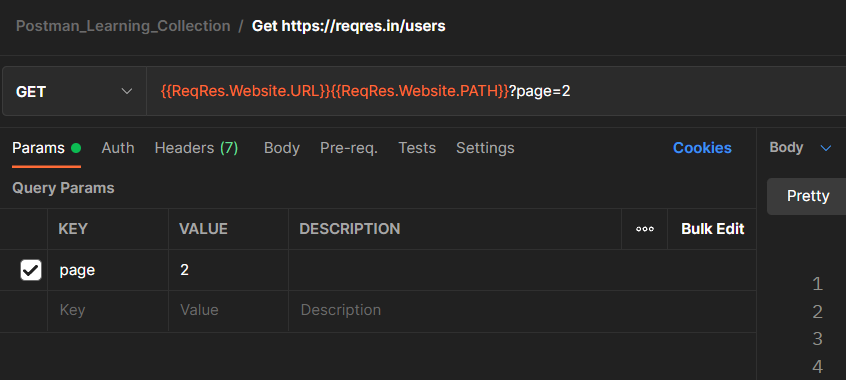
If a variable with the same name is declared in two different scopes, the value stored in the variable with narrowest scope will be used. For example, if there is a global variable named username and a local variable named username, the local value will be used when the request runs.

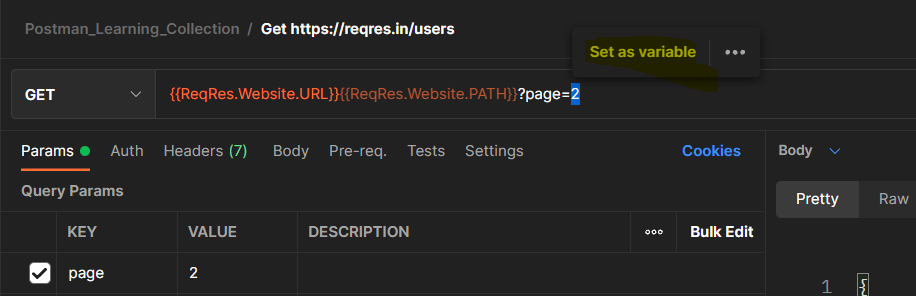
We can define the variable by two ways:

1) By selecting the data you need, for example in the API address, parameters, headers, or body. N then click on **Set as variable**.

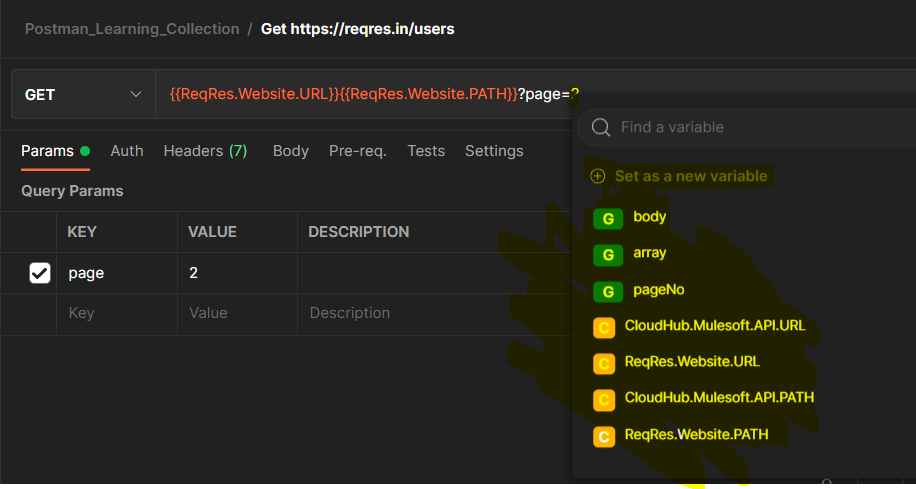


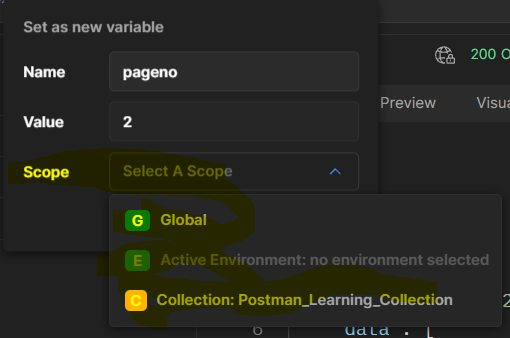
Let say we have below api url in request builder panel



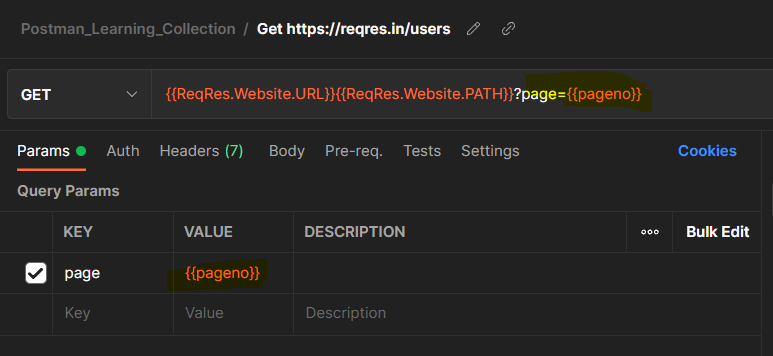
We want value 2 to be used as variable value then we can select the value 2 n then one prompt set as variable will be there ,click on that 

After clicking on set as variable we will get another prompt which has already defined variable n there scopes like g=global,c=collection as well as we have option to set as a new variable.we can create a new variable by selecting option to set as a new variable->new prompt will ask for variable name n variable scope n it will have variable value=2,which is unmodifiable. 





After giving particular name n scope ->click on set variable.After creating a variable you will see that 2 is replaced by same variable name which we have provided while setting the variable



And depending upon selected scope selected which we have provided while setting the variable in the same scope that variable is created.



2) We can create variables by

Global or environment variable:

1. Select **Environments** in the sidebar.
2. Select **Globals/click on + button in sidebar to create a new environment**



You can also view global variables by selecting the environment quick look icon Quick Look icon at the top right.

The environment quick look shows the selected environment along with global variables in your workspace. You can edit the current value for an existing variable inline by selecting the value. To add a variable, select **Edit** next to the global section.

To add a new global variable:

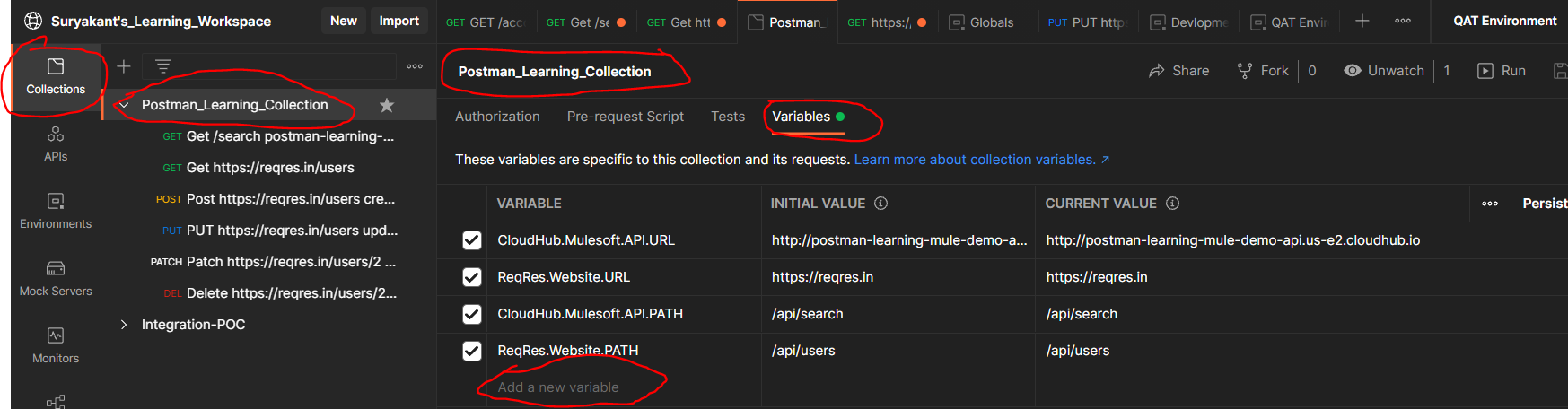
1. Select **Add a new variable**, and enter a name for the variable.
2. Select a **Type** for the new variable.
3. Add an **Initial Value**, and if you choose, a **Current Value**.
4. Select Save icon **Save** to confirm your changes.

To edit an existing global variable:

1. Change the desired variable value.
2. Select Save icon **Save** to confirm your changes.

To create collection variables-

go to specific collection->go to variable tab->add new variable





## Using variables

You can use double curly braces to reference variables throughout Postman. For example, to reference a variable named "username" in your request authorization settings, you would use the following syntax with double curly braces around the name:

{{username}}

When you run a request, Postman will resolve the variable and replace it with its current value.

For example, you could have a request URL referencing a variable as follows:

https://postman-echo.com/get?customer\_id={{cust\_id}}

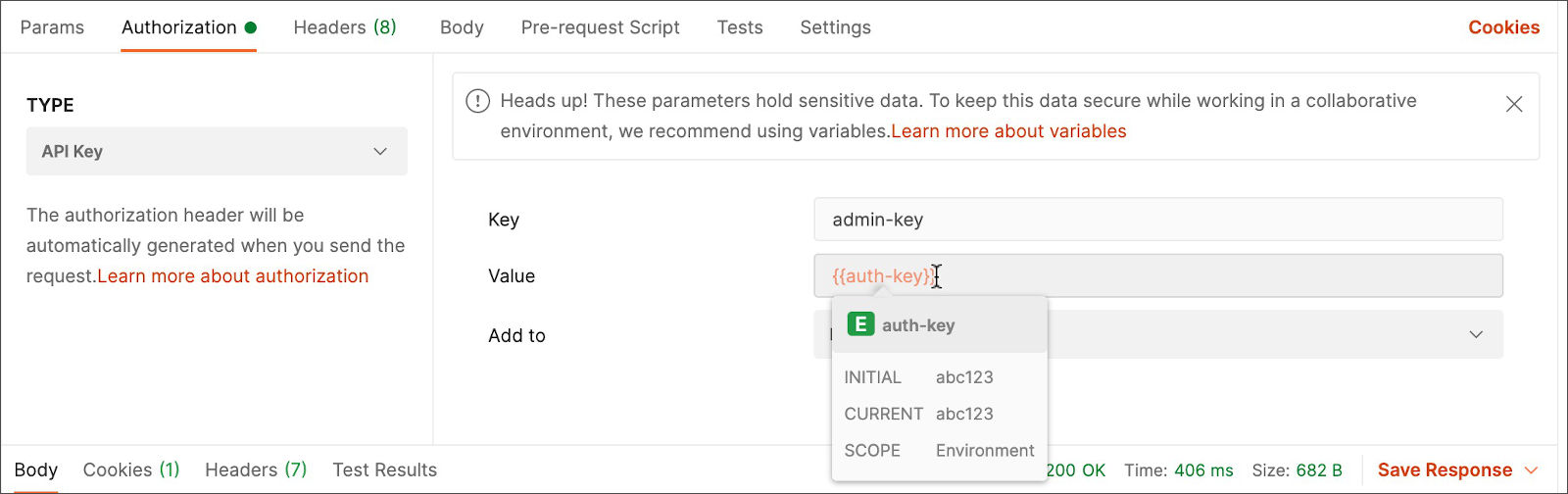
Postman will send whatever value you currently have stored for the cust\_id variable when the request runs. If cust\_id is currently 3, the request will be sent to the following URL including the query parameter:

https://postman-echo.com/get?customer\_id=3

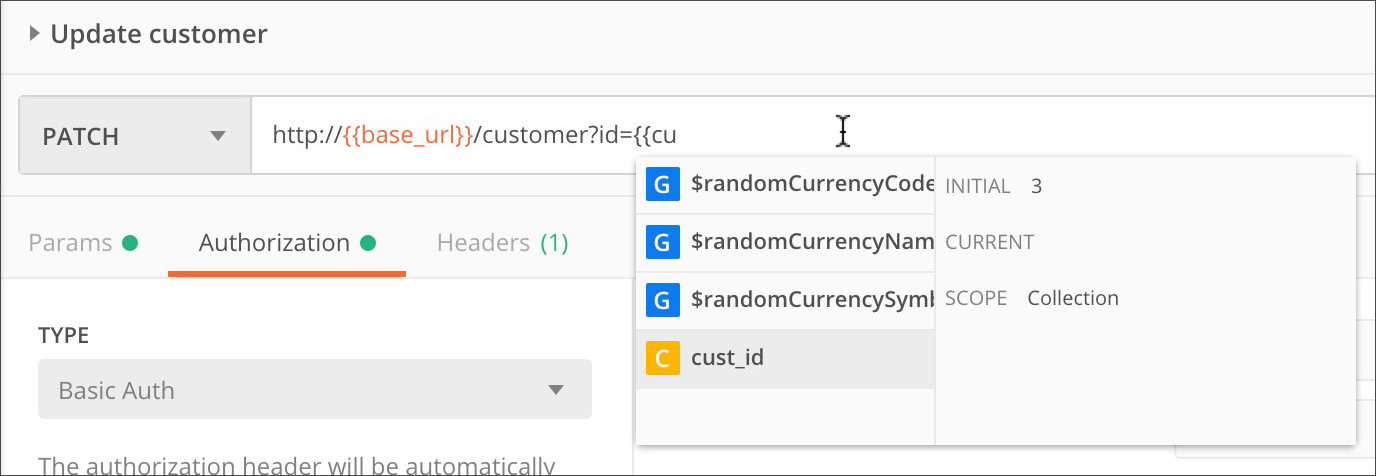
If you are want to access a variable from within a request body, wrap its reference in double-quotes:

{ "customer\_id" : "{{cust\_id}}" }

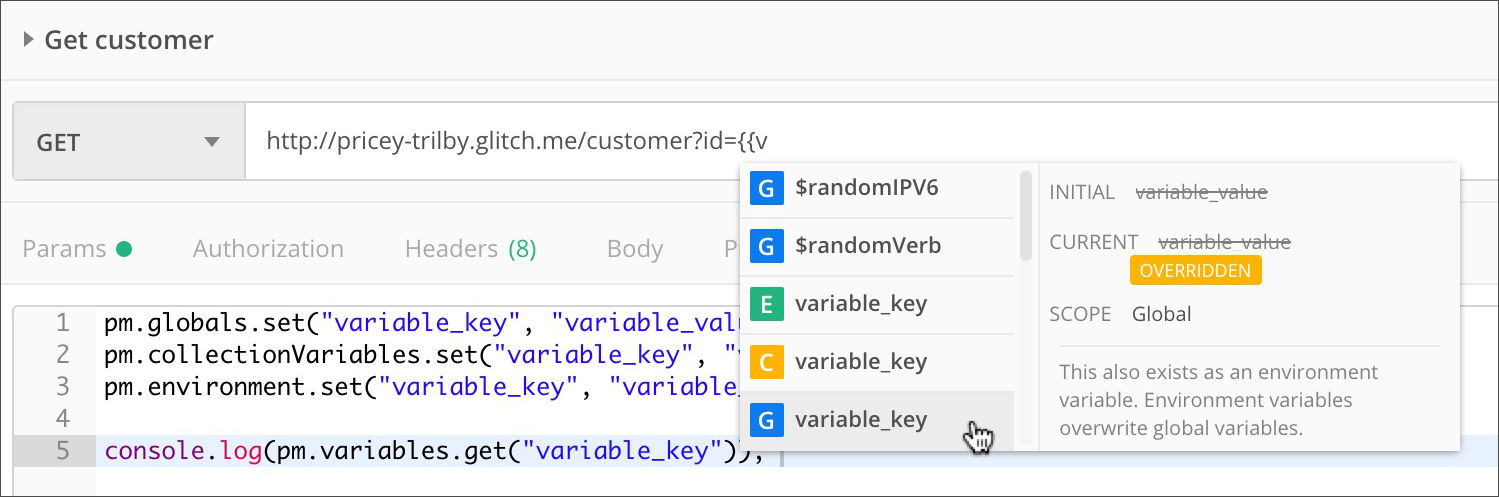
You can use variables in request URLs, parameters, headers, authorization, body, and header presets.



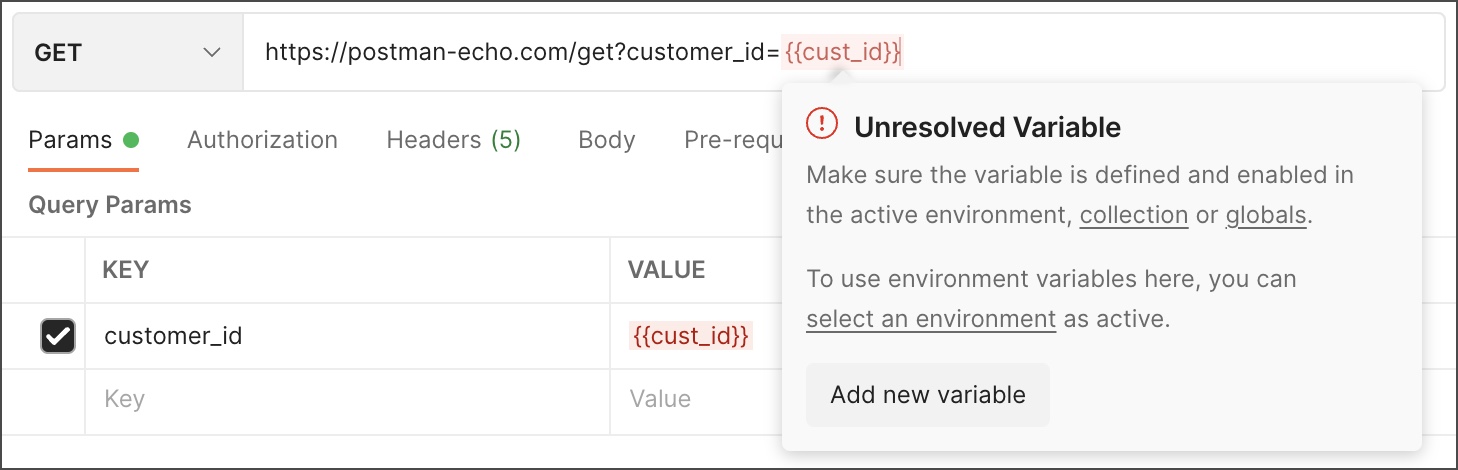
When you hover over a variable, Postman shows an overview of its current status. As you add variables to your requests, Postman prompts you with any that are currently defined.



The prompt indicates the current value, scope (highlighted by color), and overridden status where relevant.



If a variable is unresolved, Postman highlights it in red. For information on how to fix an unresolved variable, see [Fixing unresolved variables](https://learning.postman.com/docs/sending-requests/variables/#fixing-unresolved-variables).



### Using dynamic variables

Postman provides dynamic variables you can use in your requests.

Examples of dynamic variables include:

* {{$guid}} : A v4-style GUID
* {{$timestamp}}: The current Unix timestamp in seconds
* {{$randomInt}}: A random integer between 0 and 1000

See the [Dynamic Variables](https://learning.postman.com/docs/writing-scripts/script-references/variables-list/) section for a full list.

### Using variables in scripts

You can retrieve the current value of a variable in your scripts using the object representing the scope level and the .get method:

//access a variable at any scope including local

pm.variables.get("variable\_key");

//access a global variable

pm.globals.get("variable\_key");

//access a collection variable

pm.collectionVariables.get("variable\_key");

//access an environment variable

pm.environment.get("variable\_key");

Using pm.variables.get() to access variables in your scripts gives you the option to change variable scope without affecting your script functionality. This method will return whatever variable currently has highest precedence (or narrowest scope).

To use [dynamic variables](https://learning.postman.com/docs/sending-requests/variables/#using-dynamic-variables) in pre-request or test scripts, use pm.variables.replaceIn(), for example pm.variables.replaceIn('{{$randomFirstName}}').

See the [Sandbox Reference](https://learning.postman.com/docs/writing-scripts/script-references/postman-sandbox-api-reference/) for more details about scripting with variables.

#### Logging variables

You can log variable values to the [Postman Console](https://learning.postman.com/docs/sending-requests/troubleshooting-api-requests/) while your requests run.

Use the following syntax in your script to log the value of a variable:

console.log(pm.variables.get("variable\_key"));

To view the results, select Console icon **Console** in the footer. You can also access the console from the **View** menu > **Show Postman Console**.

## Logging Variable Sharing and persisting data

When you edit global, collection, and environment variables in Postman, there is a **Current Value** that you can choose to **Persist** or **Reset** for individual variables. You can also select **Persist All** or **Reset All** to apply this setting to all variables. These enable you to control what happens within your local instance of Postman, independently of how the data is synced with anyone sharing your workspace, requests, collections, and environments.

When you create or edit a variable, you can enter both an initial and a current value. When you create a new variable in Postman, if you leave the current value empty, it will autofill with the initial value. If you specify a current value, it will be local to your instance. The **Persist** option lets you push your current value to the shared data, updating the initial value to match the current value.

If you don't have Editor access to an environment, you can't edit the initial value of an environment variable. You can edit the current value, and your edit won't be visible to anyone sharing your [workspace](https://learning.postman.com/docs/collaborating-in-postman/using-workspaces/creating-workspaces/).

Using **Persist** makes your current value [sync](https://learning.postman.com/docs/getting-started/syncing/) with Postman's servers and be reflected for anyone sharing your collection or environment. To reset your current local values to reflect the initial shared values, use **Reset**.

To persist individual values:

1. Hover over a variable's current value.
2. Select the more actions icon More actions icon next to the value.
3. Select **Persist**.

Your local session in Postman can use values that are transient and visible to you, but aren't synced or shared with your team. This lets you develop and test using private credentials or experimental values, without risk of exposing these details or affecting others on your team.

For example, your team could have a shared API key and individual API keys. You could do experimental development work locally using your personal key, but use the shared key for team collaboration. Similarly, you could have a variable that represents exploratory work you're doing locally but aren't ready to share with the team. You can later choose to persist the local data so that others on your team can also access it.

You can edit a current value inline using the environment quick look icon Environment quick look icon in the top right of Postman.

For more information on working with variables as a team, see [Managing environments](https://learning.postman.com/docs/sending-requests/managing-environments/#creating-environments).

Local and data variables have current values, which don't persist beyond request or collection runs.

## Fixing unresolved variables

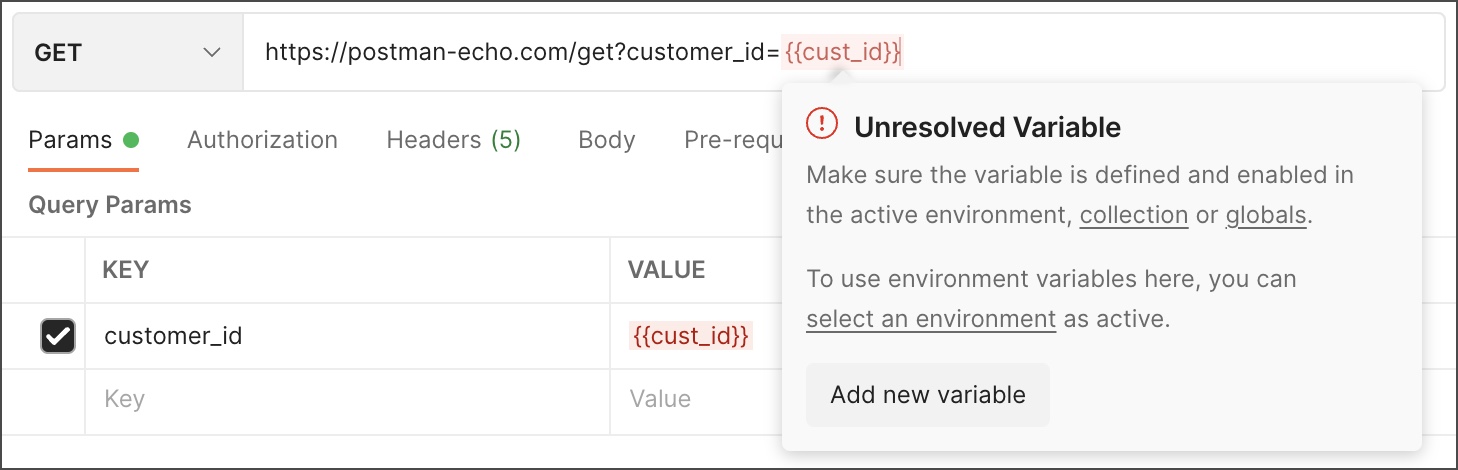
An unresolved variable is a variable that's not defined in an [active scope](https://learning.postman.com/docs/sending-requests/variables/#variable-scopes) (environment, collection, or globals) that's available for the request it’s used in.

For example, for the request https://postman-echo.com/get?customer\_id={{cust\_id}}, Postman expects to be able to find a definition for {{cust\_id}} in the environment the request uses, in the collection the request is saved in, or at the global level. If Postman doesn't find a definition for {{cust\_id}} in one of those scopes, then it flags the variable as unresolved. If you send a request that includes an unresolved variable, the request might fail.

A variable can be unresolved for a few reasons:

* The variable isn't present in an [active scope](https://learning.postman.com/docs/sending-requests/variables/#variable-scopes) for the request
* The variable was created but the changes weren't saved
* The environment in which the variable is present isn't active
* The variable is turned off in an active environment

When you are working on an API request, Postman highlights unresolved variables in the **URL builder**, the **Params** tab, the **Authorization** tab, and the **Headers** tab. Postman highlights unresolved variable text in red. For more details about the error and how to resolve it, hover over the unresolved variable.

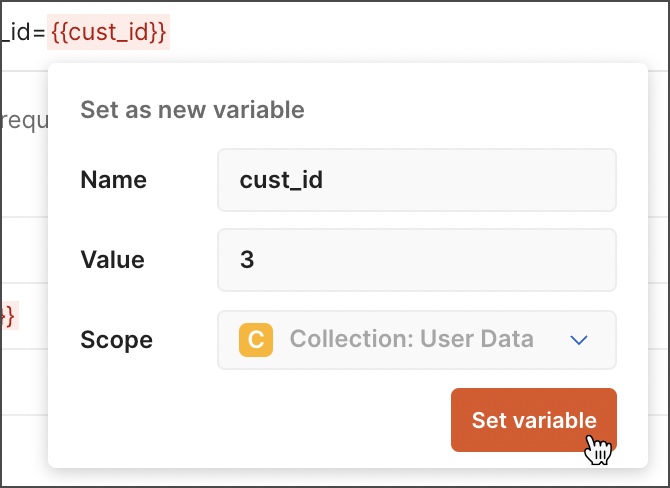


To check if the variable is available and in scope for the request:

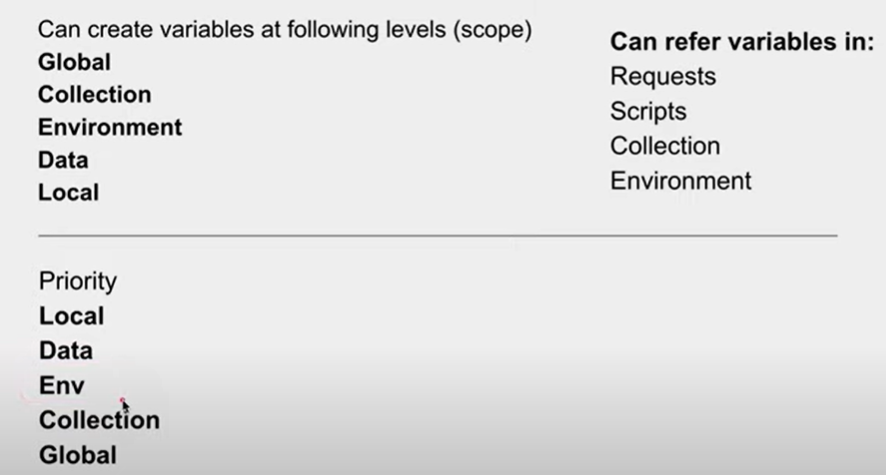
1. Select one of the **collection** or **globals** links. To turn on an environment, use the \*\*select an **environment** link.
2. Turn on or make the necessary changes to the value of the variable.
3. Select Save icon **Save** to confirm your changes.

If the variable is unresolved because it doesn't exist:

1. Select **Add new variable**.
2. Enter a **Name**, set a **Value** for the variable, and select the appropriate **Scope** (global, collection, or environment) from the dropdown.
3. Select **Set variable**.



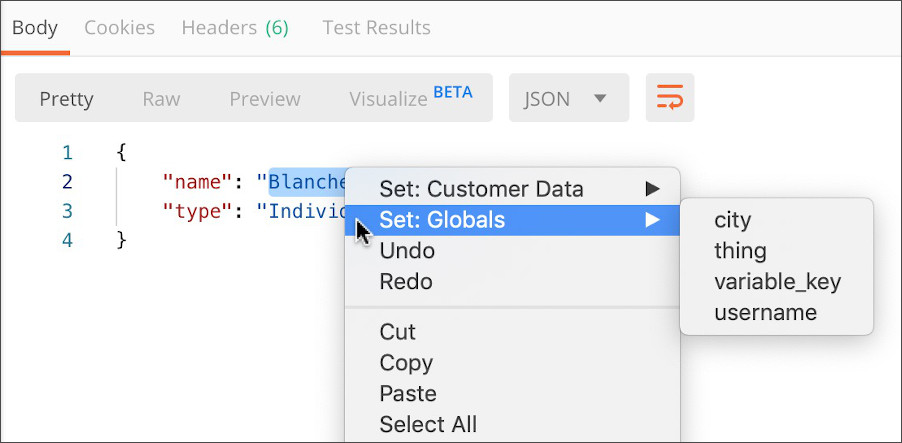
Variables that are [defined programmatically in a script](https://learning.postman.com/docs/sending-requests/variables/#defining-variables-in-scripts) are resolved differently depending on the variable scope. This means that unresolved variables will also be handled differently. Local variables that are set programmatically using pm.variables.set may appear to be unresolved since they're not stored and are only used at runtime, but if they're set and used correctly the request will still run successfully. Environment, global, and collection variables that are set programmatically are saved for later use, so they will ;resolve if they're set and used correctly. Depending on how an unresolved variable is used in a script, you may receive a 400 Bad Request error response from the API, or Postman may be unable to send the request at all. Open the [console](https://learning.postman.com/docs/sending-requests/troubleshooting-api-requests/#debugging-in-the-console) to help identify unresolved variables in your scripts.



### Setting response body values as variables

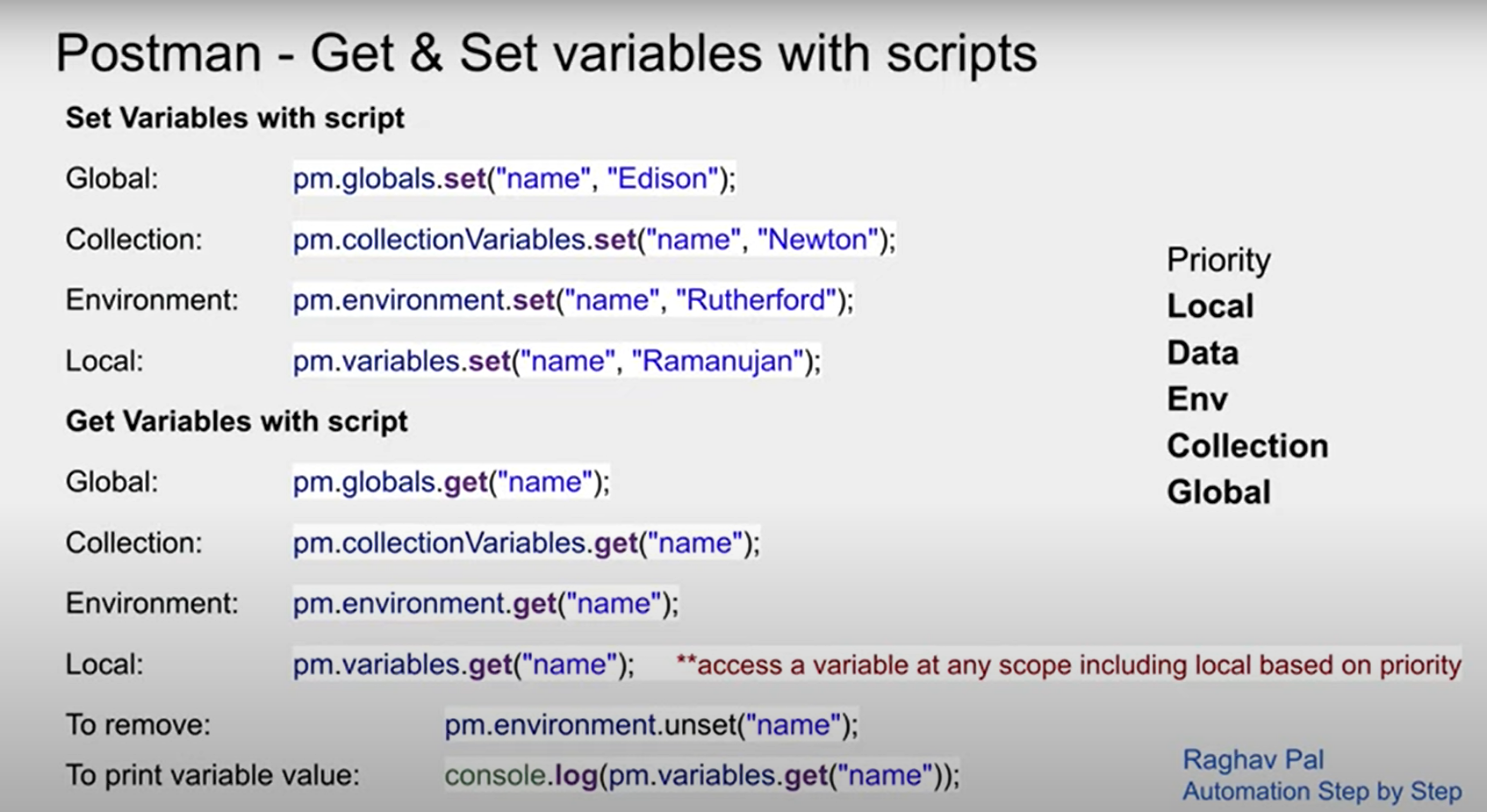
You can set the values for existing variables to values from a request's response body:

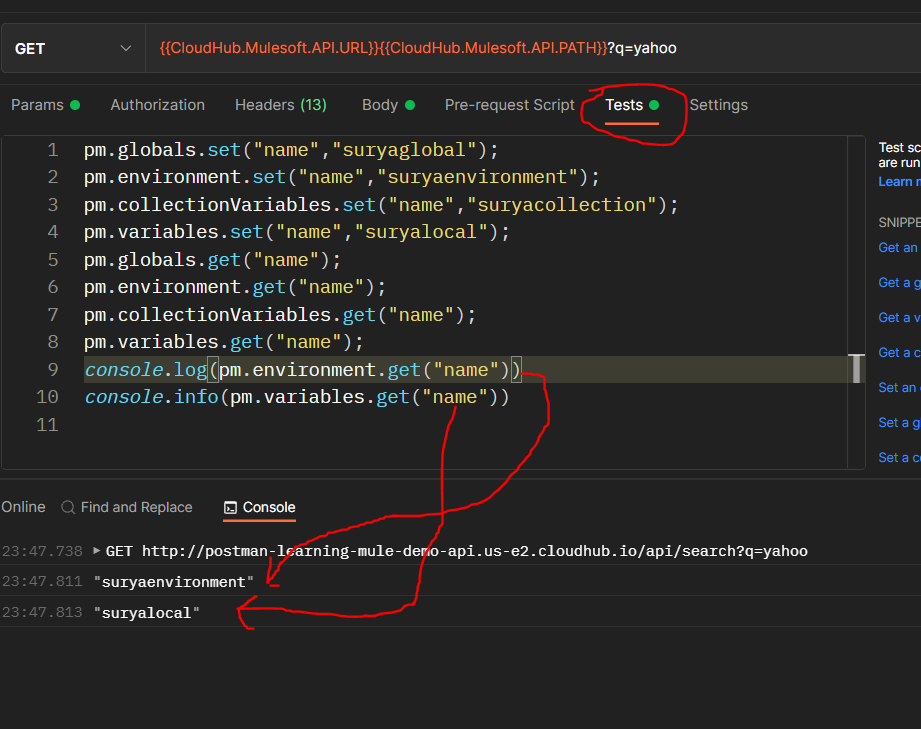
1. Select the text, then right-click or Control-click.
2. Select the relevant scope (environment or global), then select the name of the variable.



**To use variable in any request place like authorization tab,param tab,we just need to add {{Variable\_name}}**

**We have Environment variables (allow you to switch the context of your requests) and Global Variable (Accessible inside a complete workspace) and collection variables(Accessible within collection)**





**It is important to remember that priority is most important thing in this variables.**

**Priority is as below:**

**Local**

**Data**

**Environment**

**Collection**

**Global**

**Let say we declared same name variable @global,@environment,@local,@collection**

pm.globals.set("name","suryaglobal");

pm.environment.set("name","suryaenvironment");

pm.collectionVariables.set("name","suryacollection");

pm.variables.set("name","suryalocal");

**n then we are fetching the name variable value @ global,@environment,@local,@collection**

pm.globals.get("name");

pm.environment.get("name");

pm.collectionVariables.get("name");

pm.variables.get("name");

**after that we are logging the name variable values @local n @environment at console**

*console*.log(pm.environment.get("name"))

*console*.info(pm.variables.get("name"))

**then we are unsetting the local n environment name variable values**

pm.variables.unset("name")

pm.environment.unset("name")

**after unsetting the local n environment name variable value if we want to log**

*console*.info(pm.variables.get("name"))

**,Do you think it will give an error if after unsetting the name variable pm.variables(@local-level) if we are trying to log the unsetted variable value in log ?**

**Answer is no. this is becoz postman variables follows priority basis, if we have same variable name declared @multiple scopes .in this above case we are unsetting both name variable value @local as well as @environment level n variable priority is**

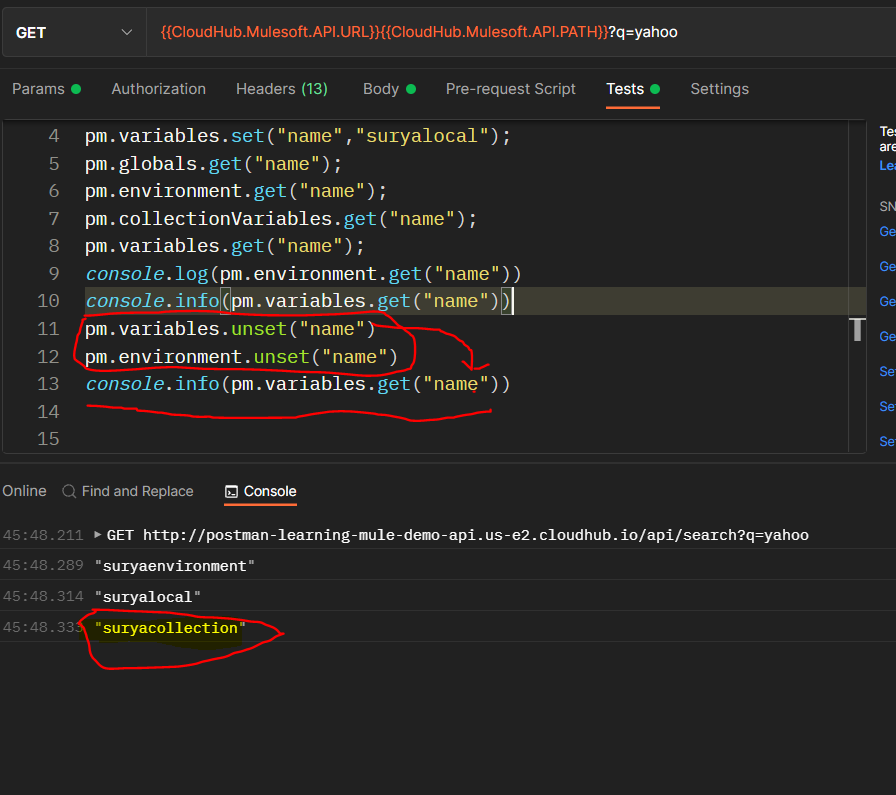
**Local**

**Data**

**Environment**

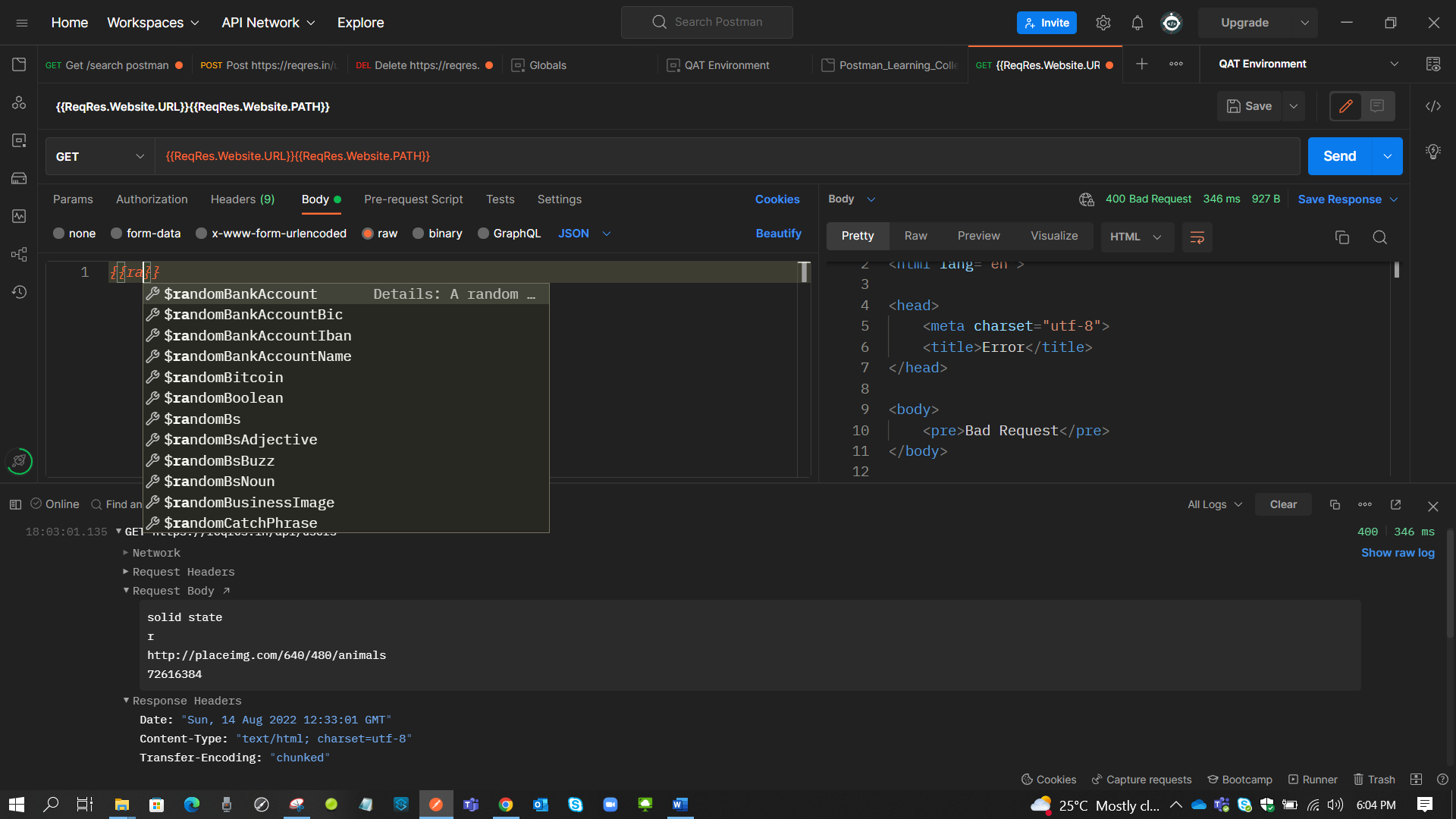
**Collection**

**Global**

**So since we have same name variable created at different scopes so what it will follow priority basis.so with that since we have unsetted name variable @local as well as @environment level ,it will take the name variable value declared @collection level(since we don’t have declared @data level name variable)** 

**Note:Postman variables are case-sensitive so we have to use same name as the variable name is declared. Like to use UserName variable we have to use that as UserName then only it will work.**

**To insert dynamic random values in your request then you can write {{ra it will prompt so many suggestion like {{$randomINT}} etc.**

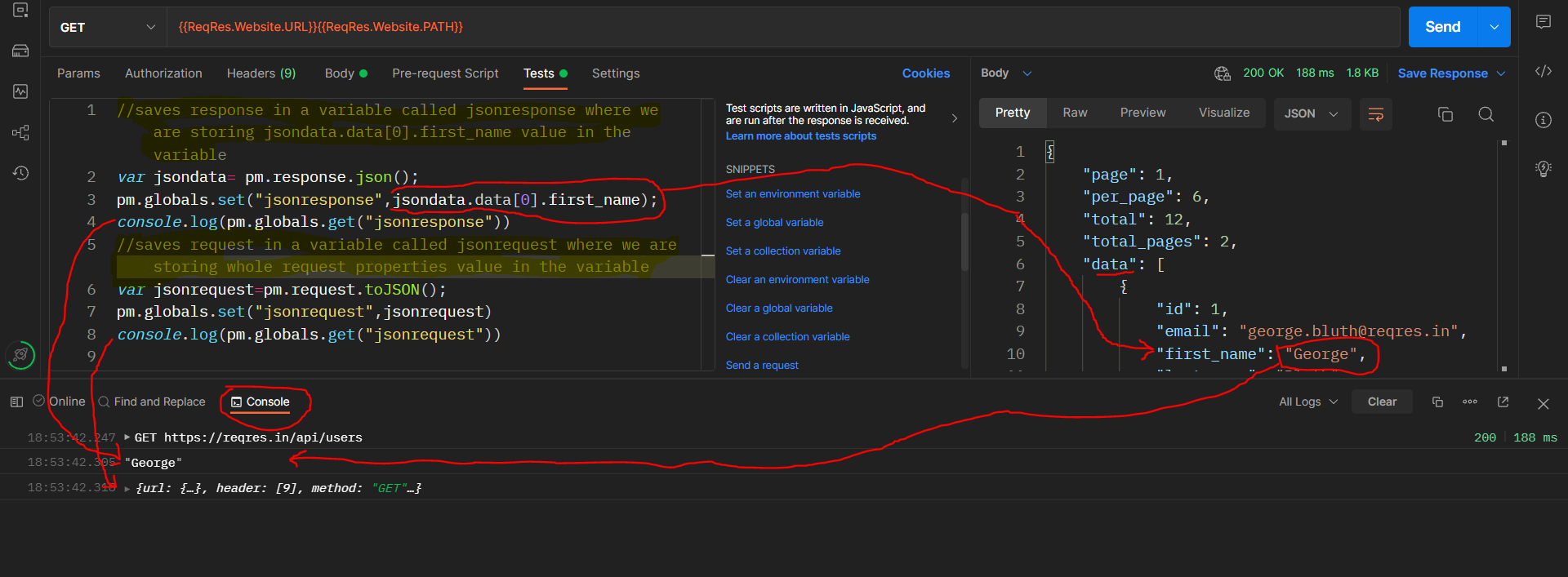


How to store request/response in a variable(@local/@environment/@collection/@global)

**Let say we have sent an request n the response which we got from a request we want to save in an variable how to do that? Or let say we want to save a request that we are sending in a variable how to do that?**

**For this purpose I am using global variables becoz scope of global variables is broadest. Since we are talking about the response to save we have to use TESTS tab becoz It will work only after the postman gets the response from client.So will use tests tab here.**

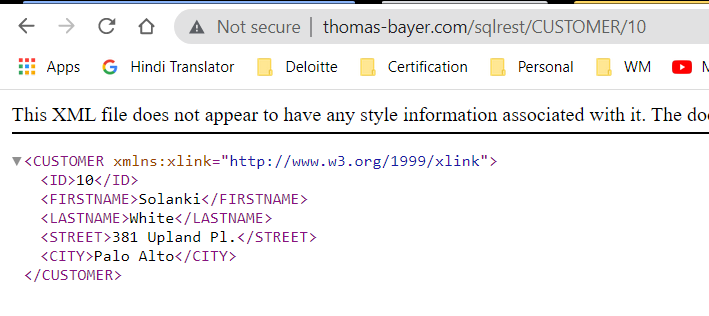
**+ Since we are talking about the request to save we can use pre-request Script tab/Tests tab .So will use tests tab here.**



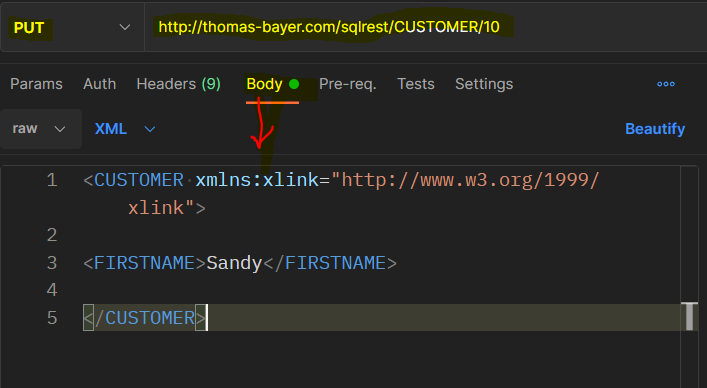
**Get 200-Ok (Fetches the records,safer n idomptent(multiple calls does not make any changes in server state) request body not required**

**Post 201-created(Creates a resource,not safe and non-idomptent) request body required**

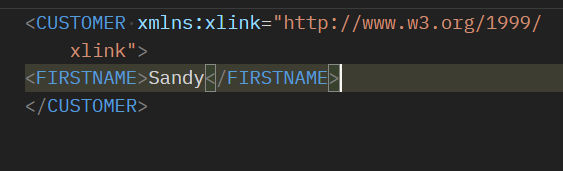
**Put 200-ok(Put will Updates a resource with a new data we provide ,if resource not exists already then creates new one. request body required**



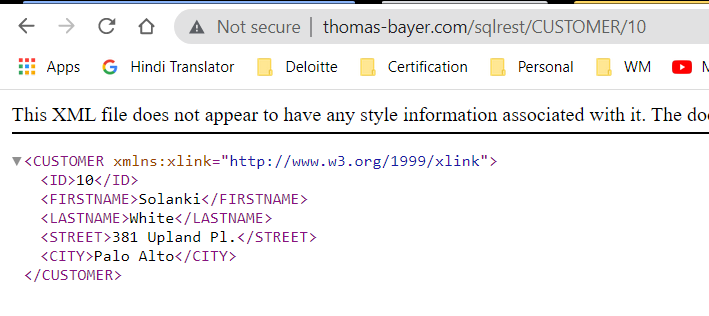
**Let say we do put method on** [**http://thomas-bayer.com/sqlrest/CUSTOMER/10**](http://thomas-bayer.com/sqlrest/CUSTOMER/10) **then with some random data in body like**



**Then GET** [**http://thomas-bayer.com/sqlrest/CUSTOMER/10**](http://thomas-bayer.com/sqlrest/CUSTOMER/10) **will have information like**

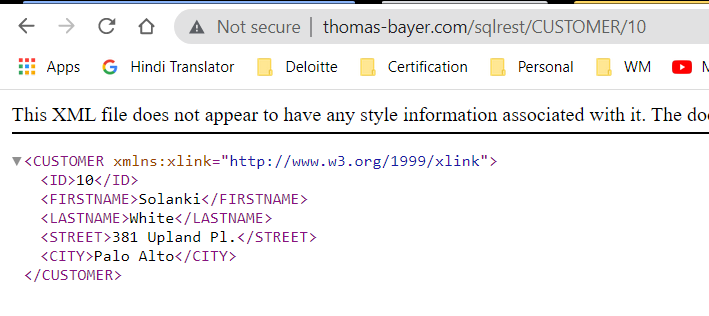


**and not the below one**



**That means PUT first checks for the resource availability(CUSTOMER/10) if resource exists (CUSTOMER/10) it updates the whole info by given data in body if resource not exists(CUSTOMER/10) then creates a new resource with that resource(CUSTOMER/10).**

**Patch 200-ok(updates a resource partially that means lets say we want to update only some values withing that resource then will go for Patch)request body required.**

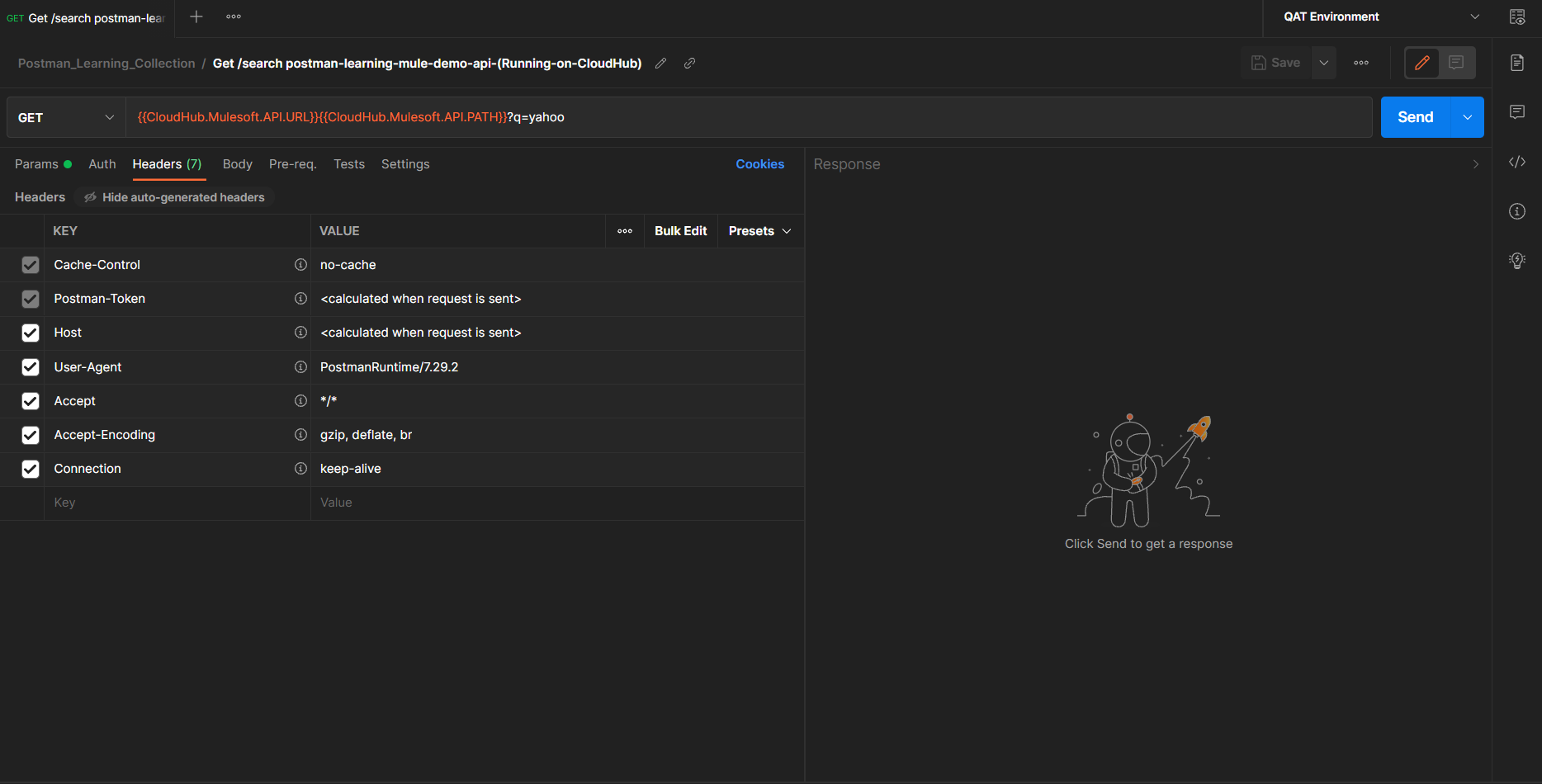


**Let say we do PATCH method on** [**http://thomas-bayer.com/sqlrest/CUSTOMER/10**](http://thomas-bayer.com/sqlrest/CUSTOMER/10) **to update only <FIRSTNAME>Shepard</FIRSTNAME> value then we can go for PATCH method where we have to provide <FIRSTNAME>Shepard</FIRSTNAME> value in request body**

**Delete 204-no content(Deletes a resource that means lets say we want to delete a resource then will go for Delete)request body not required.**

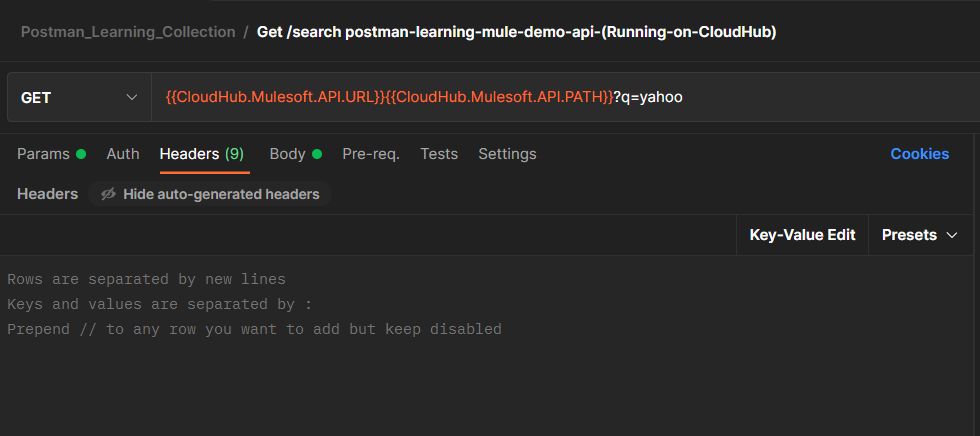
**To get the idea of different Http statuses then go through this link** [**https://developer.mozilla.org/en-US/docs/Web/HTTP/Status**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status)

**Request-Builder panel:**

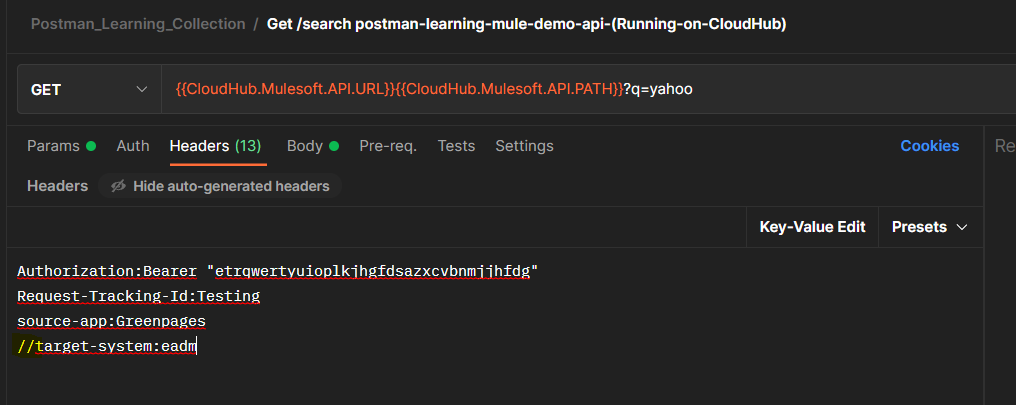


Request-Builder panel is a place where we can actually work

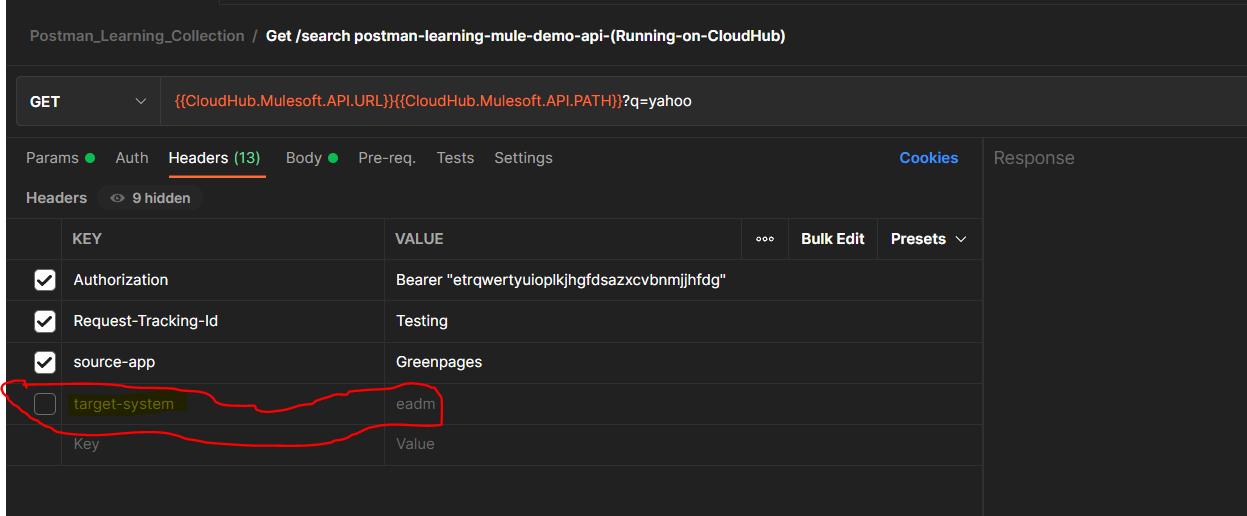
Bulk-Edit option:

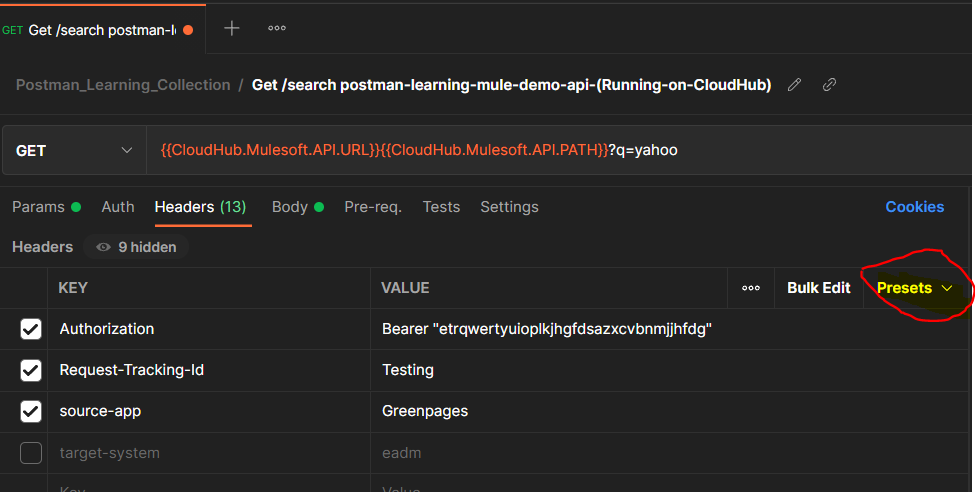
This option enables us to edit/add the properties at once.After clicking on Bulk Edit option we can see an editor where we can add or edit the properties where each property is separated by new line n key n value are separated by : and if we want to disable the property when we are creating/modifying it we can do that by adding // in front of that particular property key.

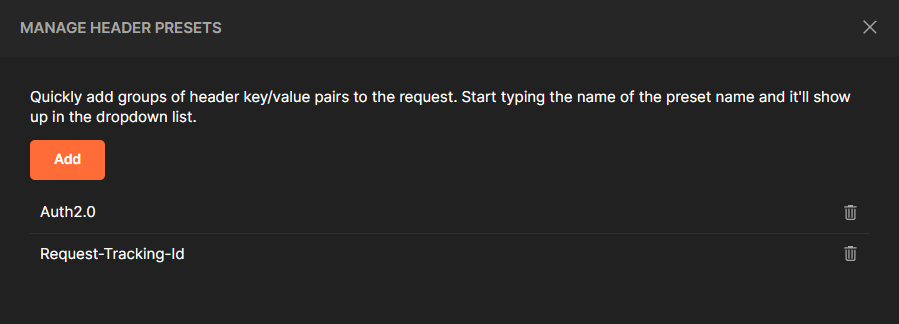
Like below



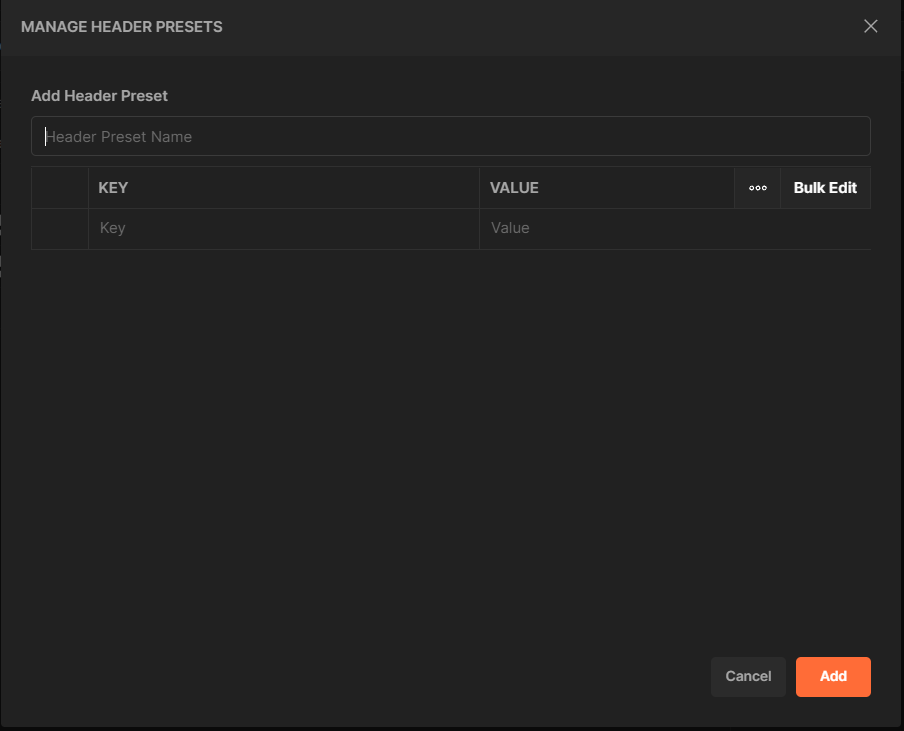
It will result in below after selecting key-value edit button:

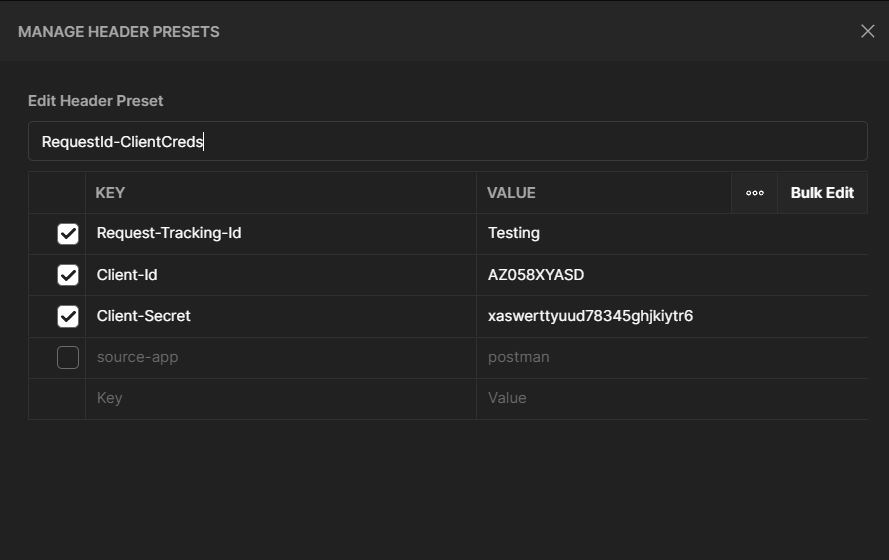


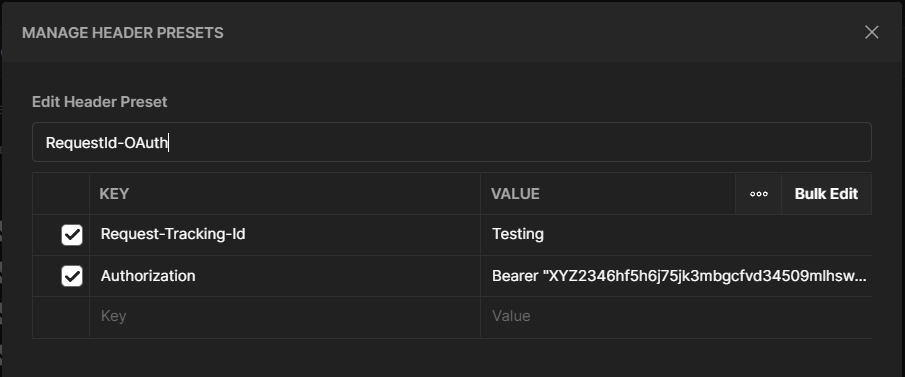
-Presets option:-presets option enables us to declare the headers in advance and use them inside any requests in headers tab 



To add new presets we can go to headers tab inside that click on presets->manage presets->add->headers presets name n then enter key n value .







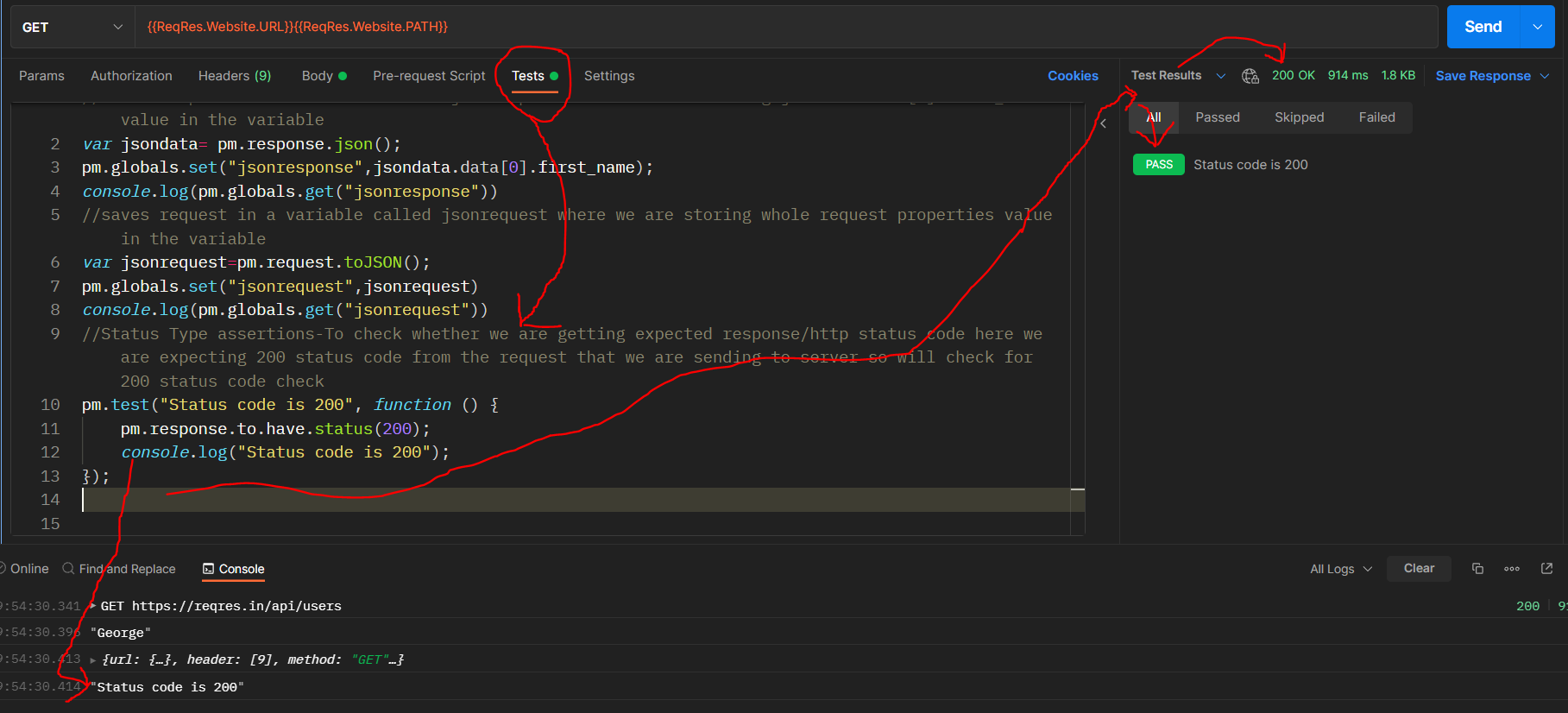
How to use Presets:

Once we add the presets,we can use that presets by going to headers tab n typing the header preset name when u start typing the name u will get suggestions select one of the wanted suggestion n click on that wanted suggestion so you will find that all headers that are defined in that suggested header preset name will get populated automatically.then you can use that headers according to your requirement.

**Assertions: (To test whether we are getting what we were expecting)**

To check or test that we are getting the same status code that we are expecting we use status type assertions by using Tests Tab in which we can add the test to check whether we are getting expected status code from the response.in the below example we are expecting http status code to return is 200 so we are adding status type assertion for 200 code.

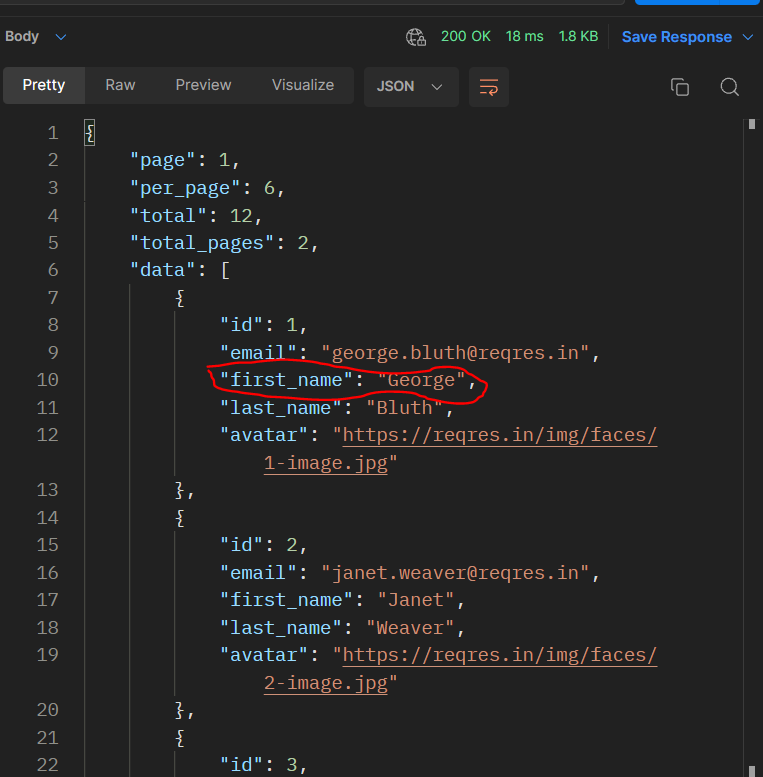
We can check the result of this test in response panel ->Test Results Tab (will show if the test is passed or failed) the test will be failed if we are expecting 200 but getting 404 or any other code.

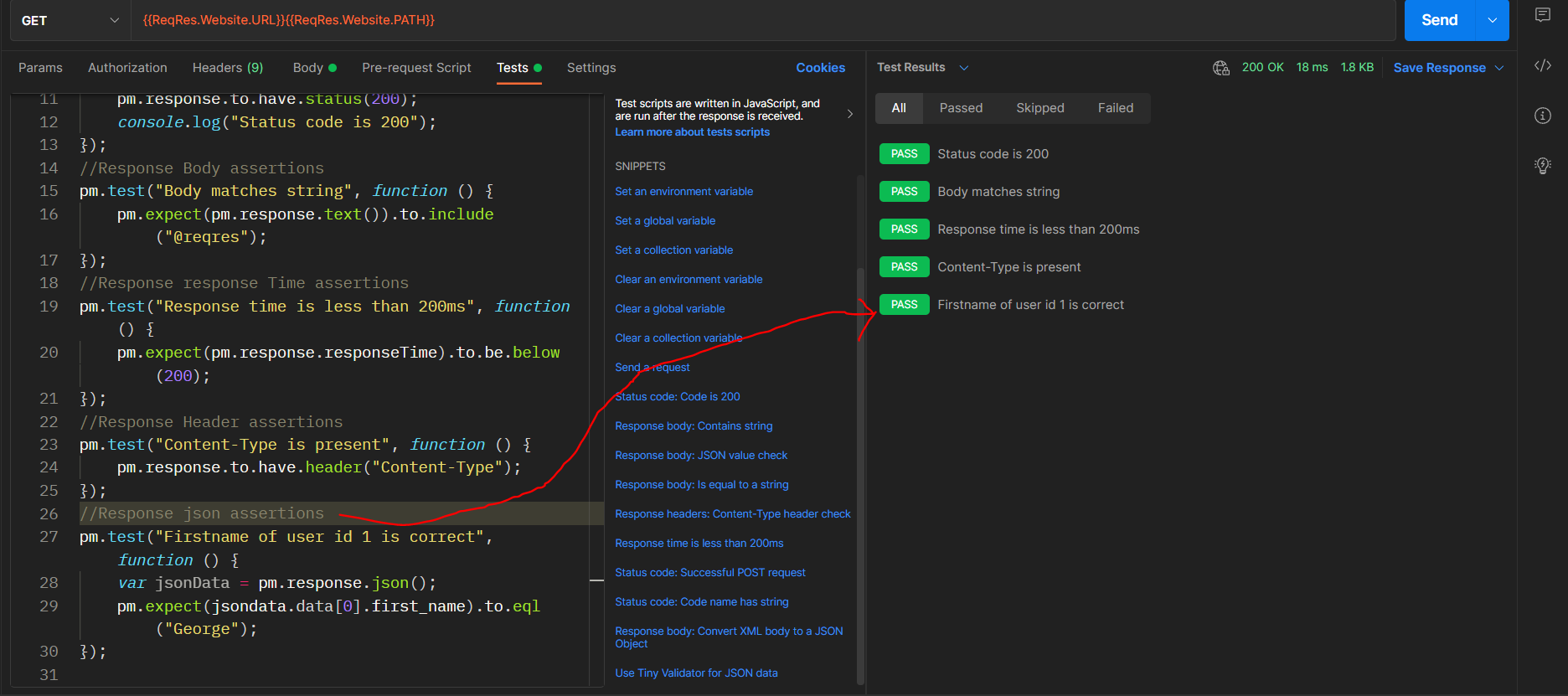


Same way we can verify the Response body assertions as well as Response Header/Response Response Time assertions by using Tests tab by going into the tests tab->selecting a code snippet like below

F

Json Assertion tests for particular response field value



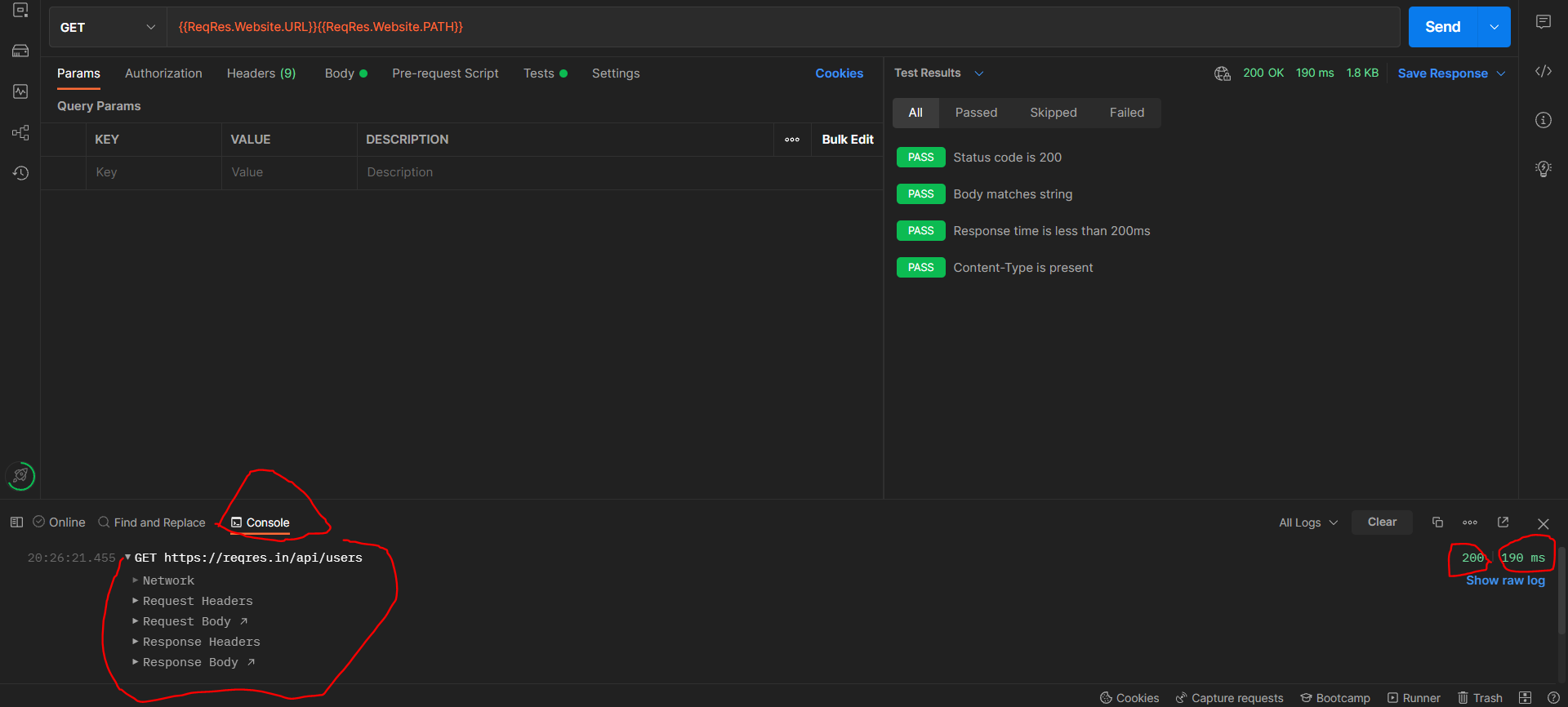


A JSON assertion validates that a specific field value was returned; a body assertion just validates that a word or text string was returned in the body.

**Debugging the Requests thru Console:**

We can debug the request using console in the postman where we will find details like what was requested url,response code,response time,Network details,Request headers,Request body,response headers,response body.

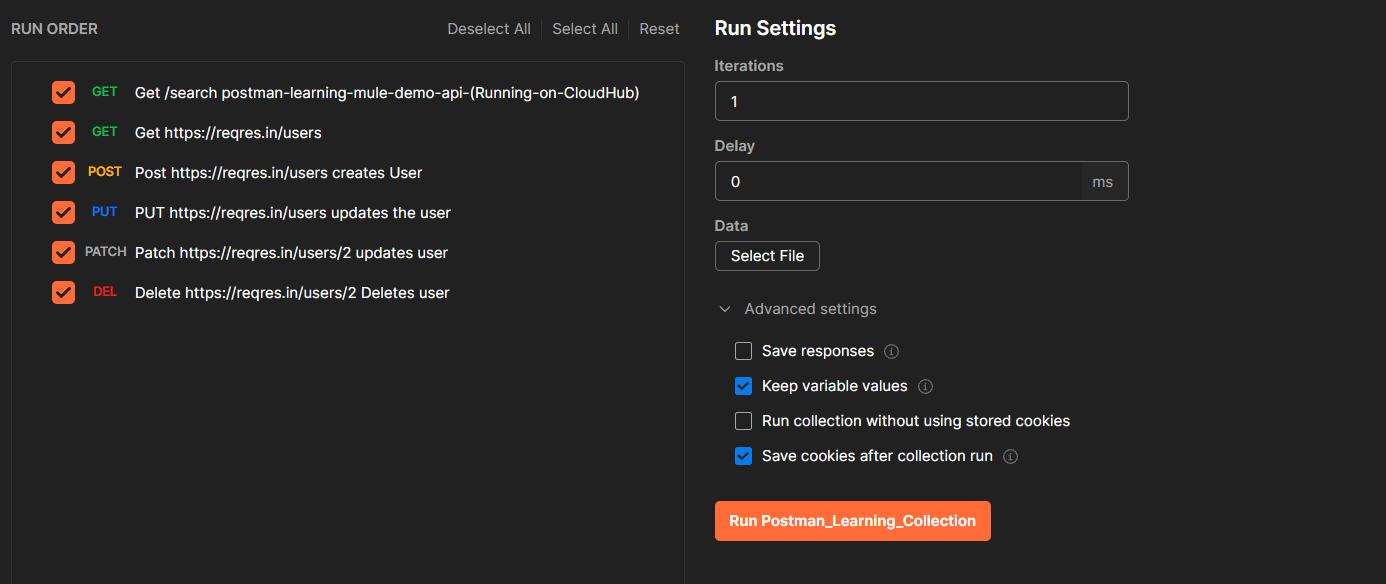
Console can be find @status bar(bottom left of the postman window)



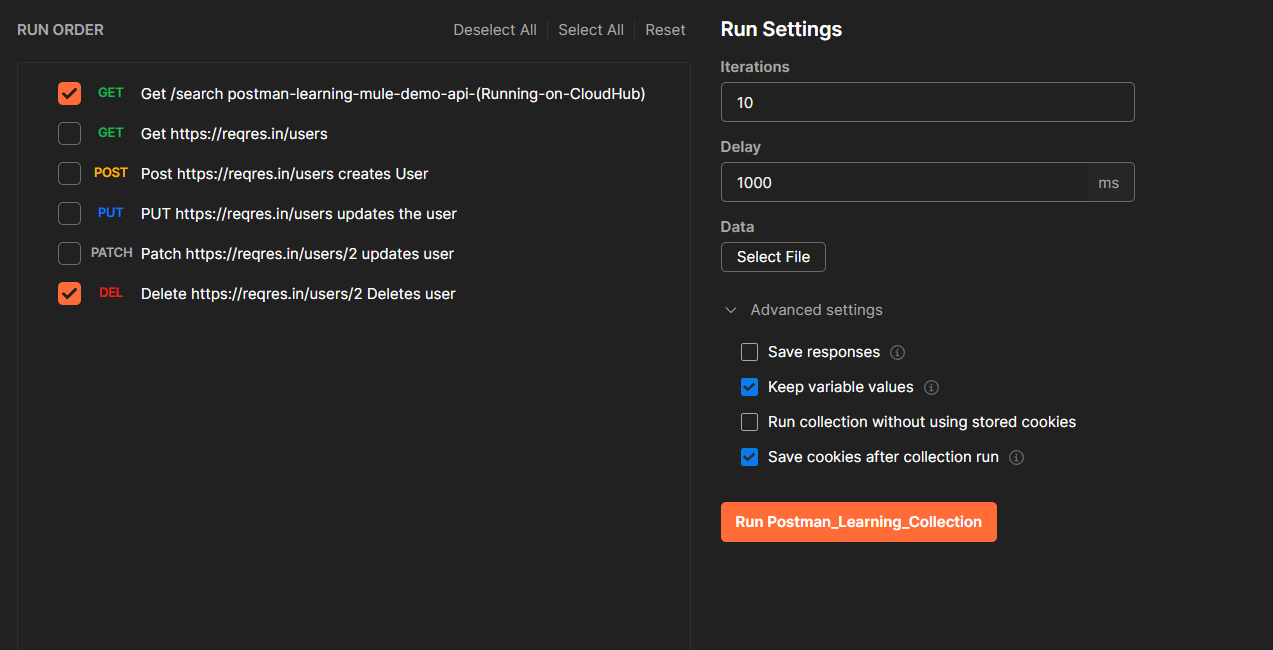
Collection Runner:-

Allows us to run the all requests inside on one click (based on no of iteration,based on delay in each request run)or if we want we can select which request we want to run n rest we can unselect to avoid.we can run the requests in any no of times but we need to mention the iterations no that will run the collection that many times. We can introduce delay in each request run by mentioning Delay time (which is in milliseconds).

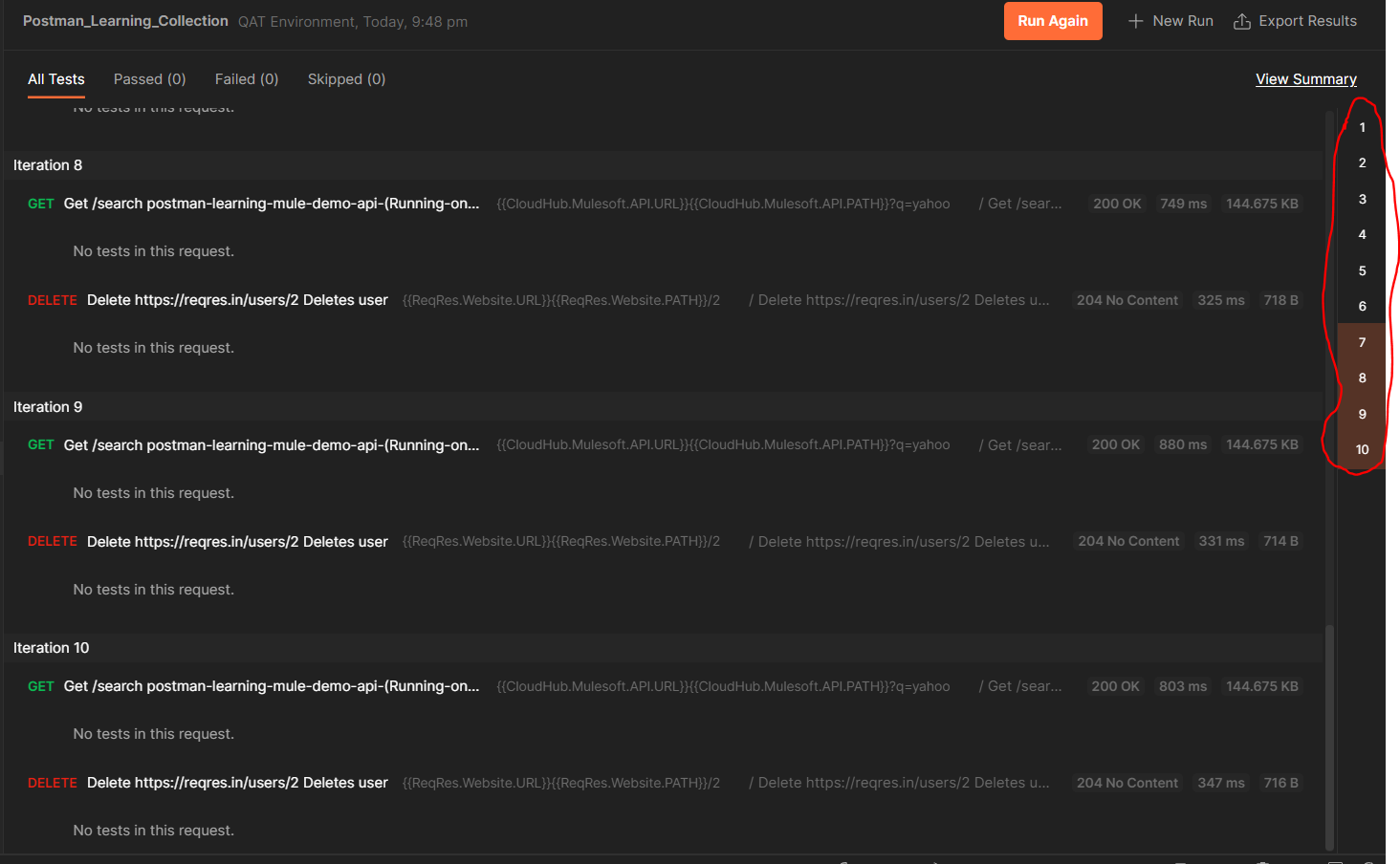
To start collection runner,go to ur collection->right click n then select Run collection



Let say we want only 2 requests to run @10 times with 1sec(1000ms) delay



After clicking on Run Postman\_learning\_Collection results will be



Note that e are getting response code,response time,url,method,response size but here we are getting like No tests in this request this is becoz we havent specified any assertions tests in @Requests-level Tests Tab or @ collection-level Tests Tab.

Exporting Collection n Environment variables:

We can export Collection as well as Environment variables just by going to … n then selecting Export option.

It will export the files in JSON format.