1. Assisted Practice: Working with GET Requests

Open the Postman tool

Create a collection 🡪 right click on it -🡪 create a request 🡪 Give a name and save it

Lets use the API : <https://openweathermap.org/>

Login id : [apitestpostmannov@gmail.com](mailto:apitestpostmannov@gmail.com)

Password : Apitest@123

Login with above credentials

Now go to APIkeys

Generate your API Keys 🡪

Go to Create Key and enter postman and click on generate 🡪 to generate key

For demo..take the default key only

Copy the key—default

### Now click on API tab on the top 🡪 click on Current Weather Data or any Api doc

### Now look at the doc , we have different API calls that can use in your postman request

### Copy the first url under By city name

### Which is :

### api.openweathermap.org/data/2.5/weather?q={city name}&appid=[{API key}](https://home.openweathermap.org/api_keys)

### here:

### api.openweathermap.org/data/2.5/weather 🡪 request

### and we here it has 2 parameters : city name and app id

### question mark here mean query parameters

### Lets paste this now in postman

### Below in postman you can see 2 parameters are there

### Cityname – give value as London

### App id – give value as your appid form the webpage

### Now you will see the full get request

### PATH PARAMETER

### Are represented with /

### So on the url remove ? and query variables and add / after weather and give :

### api.openweathermap.org/data/2.5/weather/

### api.openweathermap.org/data/2.5/weather/:id

### this will open path variable tab below

### path variable means the request will find that resource on the endpoint for you

### Lets discuss further on rest request

### Params section. Displays paramters and path variables

### Authorization: used to authorize tyour request. We will discuss latter as of now we are adding appid in the request itself

### Headers: these are metadata that are sent to your server along with your request for it to process the request

### You can create your own header also

### Body : whenever you want to send a request body with your request you can use body option

### Setting

### Test

### Pre-req script

### Now click on send button to send the above request as:

### api.openweathermap.org/data/2.5/weather?q=london&appid=8d986ab6418d17bd3557ce75dfd08a54

### You will get a response from server with status code as 200 and status as OK

### You will get response body

### Explain the response body section

### Explain cookies

### Explain Header in response

### Explain test results

### Explain save response

### <https://learning.postman.com/docs/sending-requests/responses/>

### Variables demo 1:

### Postman Variables:

### simple url

### value of var is hardcoded

### https://postman-echo.com/get?var=Hello

### lets add variable here

### environment variable

### my\_variable ==> initial value == hello

### request

### https://postman-echo.com/get?var={{my\_variable}}

### send request.. successfull

### chnage variable value to Hello everyone!

### my\_variable ==> initial value == hello Everyone!

### request

### https://postman-echo.com/get?var={{my\_variable}}

### send request.. successfull with hello everyone!

### Global Variables in postman

### <https://learning.postman.com/docs/sending-requests/variables/>

### Create Variables:

### Lets use the api request

### api.openweathermap.org/data/2.5/weather?q=london&appid=8d986ab6418d17bd3557ce75dfd08a54

### Lets see what could be the values that can replaced with a variable:

### It could the base URL as all the request will have the same variable

### It could be your app id as all the requests will have same id

### Click on the top right side eye symbol to add environment varibales

### We have 2 types: environment & global

### Lets look at Global variables:

### Global varaibles are those that are available all through your apis, your collections and request

### Will be available to your entire workspace

### Click on Add to add the variable

### Take base URl and make it as the variable

### Variable name: BaseUrl

### Current value as : api.openweathermap.org

### Valriable value are 2 types:

### Initial & Current value

### Initial value means: value which can be seen and shared by all in the team working in that workspace

### Available across your collection

### Current Value : its more secure and is used only available onlyfor your id

### Even if you share your API with others even they cannot see your variable.

### If you set initial value it will set even for current value same

### Only current value then its like secret key

### Save the variable.

### Click on the eye to see your variable details

### Now lets use in the request

### Go to the get request

### Go to the request URL

### And change the url that you have set as global variable to valriable value enclosed in curly braces

### api.openweathermap.org/data/2.5/weather?q=london&appid=8d986ab6418d17bd3557ce75dfd08a54

### Change to

### {{BaseUrl}}/data/2.5/weather?q=london&appid=8d986ab6418d17bd3557ce75dfd08a54

### Create another global variable for appIDCli

### Click on edit button to add more vaiables

### Added variables for City 🡪 current value = London

### Added variable appId 🡪 currentvalue = 8d986ab6418d17bd3557ce75dfd08a54

### Update the variables in the request now

### This is how the request will look like.

### {{BaseUrl}}/data/2.5/weather?q={{City}}&appid={{appId}}

### Now click on send button,

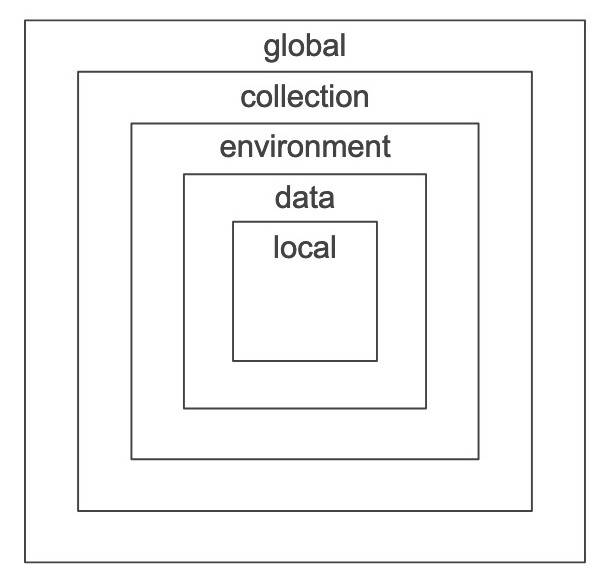
### Lets send the request this time with variable.

### You should get the same response

## Variable scopes

Postman supports the following variable scopes:

* Global
* Collection
* Environment
* Data
* Local



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 and a local variable both named username, the local value will be used when the request runs.

### Choosing variable scope

Variable scopes are suited to different tasks in Postman:

* **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/).

Since global variables can create confusion, you should only use them sparingly. For example, use a global variable to test something or when your project is at an early prototyping stage.

* **Collection variables** are available throughout the requests in a collection and are independent of environments, and don't change based on the selected environment.

Collection variables are suitable if you are only using a single environment, for example for auth or URL details.

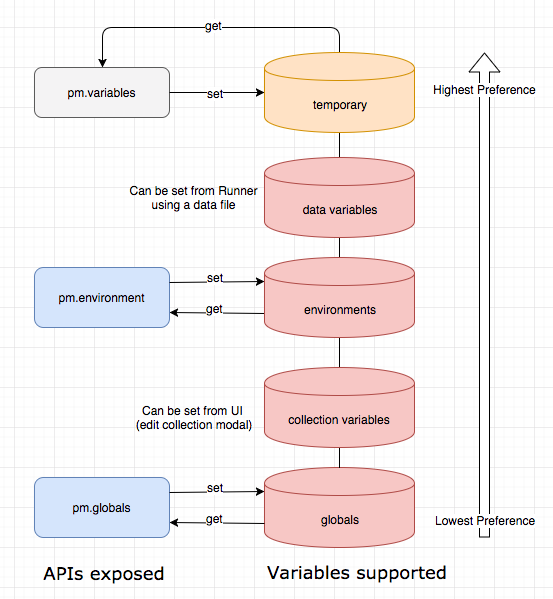
* **Environment variables** enable you to tailor your processing to different environments, for example local development versus testing or production. Only one environment can be active at a time.

If you have only one environment, using collection variables can be more efficient. However environments enable you to specify [role-based access levels](https://learning.postman.com/docs/sending-requests/managing-environments/#working-with-environments-as-a-team).

* **Local variables** are temporary, and only accessible 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.

* **Data variables** come from external CSV and JSON files to define data sets you can use when running collections with Newman or the Collection Runner.



### MANAGING ENVIRONMENTS in postman & adding environment varibales

### Clcik on eye to select ADD for environment

### Suppose you want to test in 3 different environments like dev, qa & prod

### You can use different environment variables for each environment

### Add the same global variables as your environment variables

### And delete the gloabal variables now

### Select the environment on the top and send the request now.

### Request will be successful

### On the top there is one more option left side to create environment.

### You can create more environments and add variables in it

### Add all the 3 values in it and send the variables again the request

### It will be successful

### Collections:

### It is nothing but a set of request of grouped together

### Collection of requests for your java app

### Collection of request for qa env

### Collection of requests for test cycle 1

### Using collection you can make your documentation

### We can also create collection variables also.

### Create a collection:

### On the left side top🡪 collection🡪 + symbol

### Click on it

### Give name ad description and save it

### Now add requests to the collection

### Add more requests to the collection

### Now add folder to collection b clicking on 3 dots on collection

### Move the requests in the folder

### We can create many folder and add requests to it

### You Can share your collection

### You can monitor the collection

### You can publish your collection

### Add Collection Variable

### Create variable as

### CityName : currentvalue 🡪 New Delhi

### Now update the request and environment

### {{baseUrl}}/data/2.5/weather?q={{CityName}}&appid={{appId}}

### Request will be successful

### Local variable:

### Same weather api request

### Collection variable : appid

### Global variable : baseUrl

### Environment variable : City

### Now go to pre-req script

### Add the snippet to set environment variable pm.environment.set("City", "mumbai");

### Where value of city variable is set to mubai now

### Local variable takes precedence here..

### Execute request from any environment… value of city will be Mumbai only

### 

### Authorization:

### Create a collection

### Name : github requests

### Create a new request

### Name: get all repos

### Save it

### Request url to get all repository for a user

### <https://api.github.com/user/repos>

### In this we have no path variable

### We have to add github authorization

### Under the request 🡪 authorization 🡪 select bearer authorization

### All read about each authorization method from documentation

### We have to give git hub token

### Create github token

### Settings 🡪 developer settings 🡪 personal access token

### Generate a new token🡪 select only repo

### Take this token and add under the request 🡪bearer token

### Now send the request. It will be successful

### Create a new request to get any one repo

### Again get a specific url

### <https://api.github.com/repos/:owner/:repos>

### here owner and repos are path parameter

### owner = sonal0409

### repos = any repo name.. 16SepAppium

### Under Authorization

### Don’t give any token and just run it,, it will give error

### Now under bearer auth add the token.

### Click on send it will be successful

### It is not good practice to just directly print the token number in your request.

### As you publist your request everyone can see your token

### we can hide in the form of environment variables

### So create a environment variable

### Create environment 🡪 githubdemo

### Variable name token and current value as token value

### Save it

### Go to the requests and chenage the token name to {{token}}

### Now click on send request it will be successful

### Now if you have 100 equests you will have to give token under each request

### Which we don’t want to do.

### So lets add authorization to the collection and inherit it to the requests

### Go to collection🡪 authorization🡪type 🡪 bearer token 🡪 add token as {{token}}

### No need to give the exact token as it is already in the environment variable for the environment githubdemo

### POST request:

### Post request is used to send a request for a resource to be created

### We will use github api for post request

### Lets create a repository on github using postman POST request

### Go to the collection

### Create a new request – it is POST request this time

### Api request is : <https://api.github.com/user/repos>

### First let us update a environment variable as the url of git hub 🡺 <https://api.github.com>

### Go to environments—select environment name as github

### Already one variable token will be there

### Add another variable url and its value as <https://api.github.com>

### Now save it

### Come to the request which is POST

### Give address as : {{variablename}}/user/repos 🡺 {{url}}/user/repos

### Authentication is

### Bearer token at collection level.

### Authentication for request is inherit auth from parent

### Give a body to the request 🡪

### Click on raw—select json

### Request body should be

{

  "name": "APITestDemo",

  "description": "This is your first repository",

  "homepage": "https://github.com",

  "private": **false**,

  "has\_issues": **true**,

  "has\_projects": **true**,

  "has\_wiki": **true**

}

### Here name is mandatory

### Private = false means it is public repository

### Click on send button

### The response will be of status code which is 201 which is created

### Get request response is 200.

### API Tests: write your own code

### We can test the request is pass or fail now

### Under every request there is Tests option

### You will write the code in java script

### Select the request as : Get a specific repository

### This request based on your username and repo name will get the results

### Let us validate if the:

### Status code is correct

### Status message is correct

### Response time is correct

### Header is present or not.

### Go to tests section on right side we have snippet

Select the snippet as : Status code is 200

This will validate If the response of the request is 200 or not

This is how the function will be. You can write it too..

pm.**test**("Status code is 200", **function** () {

    pm.response.to.have.status(200);

});

Again send the request.. you will see in response test(1/1) and status as pass

Postman uses BDD to write the assertion.In above code second line is assertion

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

2.Vaidate if the response consist of code name as OK

From the snippet select the snippet as

Status code name has String

or you can write the code as

pm.test("Status code name has OK", function () {

    pm.response.to.have.status("OK");

});

Click on send button. See that the status of the test is pass under response test Result section

1. Validate the response time

From the snippet select the snippet as

Response time is less than 200ms

or you can write the code as

pm.**test**("Response time is less than 200ms", **function** () {

    pm.expect(pm.response.responseTime).to.be.below(600);

});

Click on send button. See that the status of the test is pass under response test Result section

1. Validate if the response comes with a header

pm.**test**("Content-Type is present", **function** () {

    pm.response.to.have.header("Content-Type");

});

Here we are just checking if the header is coming or not.

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

1. Validate the body of the response : this will be functional validation test

So take the new request now : Get all repositories

Use the snippet JSON value check

pm.**test**("Your test name", **function** () {

    var jsonData **=** pm.response.json();

    pm.expect(jsonData.value).to.eql(100);

});

Here jsonData is a variable that holds your json data when your response is going to be json

Lets validate the name of the repo

Make changes in the code line 3 as:

JsonData.value = > json.name

To.eql(100) = > (“anyreponamefromresponse”)

pm.**test**("Your test name", **function** () {

    var jsonData **=** pm.response.json();

    pm.expect(jsonData.name).to.eql("16SepAppium");

});

Change description of function also

pm.**test**("Your repo name", **function** () {

    var jsonData **=** pm.response.json();

    pm.expect(jsonData.name).to.eql("16SepAppium");

});

Run the send button.

Your test will fail.. the reason is there are multiple results and postman doent know which name

The response is an array of elements

Every element is at an index

When you say jsonData.name 🡺 postman doesn’t know which element to valdate

So use array index

So send request with array index … jsonData[0].name

pm.**test**("Your repo name", **function** () {

    var jsonData **=** pm.response.json();

    pm.expect(jsonData[0].name).to.eql("16SepAppium");

});

Now lets validate the login id is correct or not in response body

If you observe the response body.. login filed is available under the owner field so your function will same as above

Copy it

But jsonData wil first go to owner and then to login

pm.**test**("check login name", **function** () {

    var jsonData **=** pm.response.json();

    pm.expect(jsonData[0].owner.login).to.eql("Sonal0409");

});

Give the correct login id in the above method

1. Now validate if a text contains in the body or not.

We will use here snippet as

pm.**test**("Body matches string", **function** () {

    pm.expect(pm.response.text()).to.include("This is your first repository");

});

Here this will check if the response body contains this string or not.

It can be anywhere in the body

# **Generate Random Data With Dynamic Variables In Postman**

If you see the post request it has repo name which is static is request body

For the first time you run the request, post request will be successful.

Next time you run the request , it will fail

As repo name already exisit

So we have to write pre-req script so that each the request is run it generates a random name or a patterned name to the repo

1. In the request body change name as variable

{

  "name": "*{{repoName}}*",

  "description": "This is your first repository",

  "homepage": "https://github.com",

  "private": **false**,

  "has\_issues": **true**,

  "has\_projects": **true**,

  "has\_wiki": **true**

}

Now go to pre-request script

And create the required valable and an environment variable

var repositoryName**=** "SampleApi" **+** pm.variables.replaceIn('{{$randomInt}}');

pm.environment.**set**("repoName",repositoryName);

# **Reuse Common Tests for API Requests using Collection in Postman**

Some of tests likes checking status code as 200

Status string as OK

Checking response time range for eavery test

These tests could be common for many of your requests on the Collection

You would have to repeat the tests again and again in every request

Better is you create the test at collection level

So as to reuse the tests for many requests

Lets add tests for all the requests

Get all repo

pm.**test**("Your repo name", **function** () {

    var jsonData **=** pm.response.json();

    pm.expect(jsonData[0].name).to.eql("16SepAppium");

});

pm.**test**("check login", **function** () {

    var jsonData **=** pm.response.json();

    pm.expect(jsonData[0].owner.login).to.eql("Sonal0409");

});

pm.**test**("Body matches string", **function** () {

    pm.expect(pm.response.text()).to.include("This is your first repository");

});

Get a specific repo

pm.**test**("Status code is 200", **function** () {

    pm.response.to.have.status(200);

});

pm.**test**("Status code name has OK", **function** () {

    pm.response.to.have.status("OK");

});

pm.**test**("Response time is less than 200ms", **function** () {

    pm.expect(pm.response.responseTime).to.be.below(600);

});

pm.**test**("Content-Type is present", **function** () {

    pm.response.to.have.header("Content-Type");

});

Create a repo

Add the following snippet

Status Code: Successful POST request

pm.**test**("Successful POST request", **function** () {

    pm.expect(pm.response.code).to.be.oneOf([201, 202]);

});

pm.**test**("Status code name has string", **function** () {

    pm.response.to.have.status("Created");

});

Delete a Repo

Test:

pm.**test**("Status code is 204", **function** () {

    pm.response.to.have.status(204);

});

pm.**test**("Status code name has string", **function** () {

    pm.response.to.have.status("No Content");

});

Now lets see what all test can made common for the request.

How to add them to collection

Right click on collection and edit it.

Go to test section and first add the common tests across all requests

pm.**test**("Status code is 200", **function** () {

    pm.response.to.have.status(200);

});

pm.**test**("Status code name has OK", **function** () {

    pm.response.to.have.status("OK");

});

pm.**test**("Content-Type is present", **function** () {

    pm.response.to.have.header("Content-Type");

});

pm.**test**("Response time is less than 1000ms", **function** () {

    pm.expect(pm.response.responseTime).to.be.below(1000);

});

Now among these 4 .. first 3 tests can be common for get requests

And last one can be common for create a delete request

So you can write like this if and else condition

**if**(pm.info.requestName**==**"Create a Repository"**||** pm.info.requestName**==**"Delete a repo")

{

pm.**test**("Response time is less than 1000ms", **function** () {

    pm.expect(pm.response.responseTime).to.be.below(1000);

});

}

**else**{

pm.**test**("Status code is 200", **function** () {

    pm.response.to.have.status(200);

});

pm.**test**("Status code name has OK", **function** () {

    pm.response.to.have.status("OK");

});

pm.**test**("Content-Type is present", **function** () {

    pm.response.to.have.header("Content-Type");

});

}

Here in if condition :

pm.info.requestName**==**"Create a Repository"

request name is name of request you have given in the collection.

Save the changes.

You can also remove the tests from each request if they are already added under collection tests tab.

**HOW to RUN a Collection with Collection RUNNER**

**RUN API Request multiple times with different data sets using external JSON file**

We will use weather API for this test

Where you want to pass multilple data to a variable in the request

Use the following API

api.openweathermap.org/data/2.5/weather?q=london&appid=8d986ab6418d17bd3557ce75dfd08a54

Add the variable in the API request first

Like this:

api.openweathermap.org/data/2.5/weather?q={{city}}&appid=8d986ab6418d17bd3557ce75dfd08a54

right now the city variable has no value

Now add Tests to the request also

1. To validate the status code

pm.**test**("Status code is 200", **function** () {

    pm.response.to.have.status(200);

});

1. To validate status message

pm.**test**("Status code name has string", **function** () {

    pm.response.to.have.status("OK");

});

1. To validate city name in response

pm.**test**("Check City Name", **function** () {

    var jsonData **=** pm.response.json();

    pm.expect(jsonData.name).to.eql(pm.iterationData.**get**("city"));

});

Note that

Environment will be : No environment

City value will not be set here.

Now open a notepad and lets create data as a json file add below data and save the file as test.json:

Test.json

[

{

"city": "London"

},

{

"city": "New York"

},

{

"city": "Mumbai"

}

]

Save this in a folder

Now start the ruuner

Add the weather api collection to runner

Under data filed

Select the file : you will observe the number of iterations have increased

The number of iteration will be equal to number of values in json file

Data file type is going to Json

You can click on preview also

Now run the api

All the tests will run with different data from json file

Set up new man on windows postman

You will need npm and nodejs to set it up

Check if it is already available

npm -version

node -v

Command to install new man through command prompt

# npm install -g newman

# newman -v

Now go to post

Take the collection github, click on 3 dots

Export the collection to folder in documents folder

Create a new folder as GITHUb and save the collection

Now go to environment

Click on edit

Click on 3 dots

You can export the environments variables

In the same folder

Now go to cmd line

Go to the directory of folder

Cd

And execute command

newman run GitHub.postman\_collection.json -e GitHub.postman\_environment.json

you collection will run

now if you want to execute multiple times

newman run GitHub.postman\_collection.json -e GitHub.postman\_environment.json -n 2

-n : number of time run the collection

2 times

# **Generate HTML Reports from command-line with Newman**

<https://www.npmjs.com/package/newman-reporter-htmlextra>

Install html extra

npm install -g newman-reporter-htmlextra

Already installed

Go to directory where you have collection and ebvironment variables

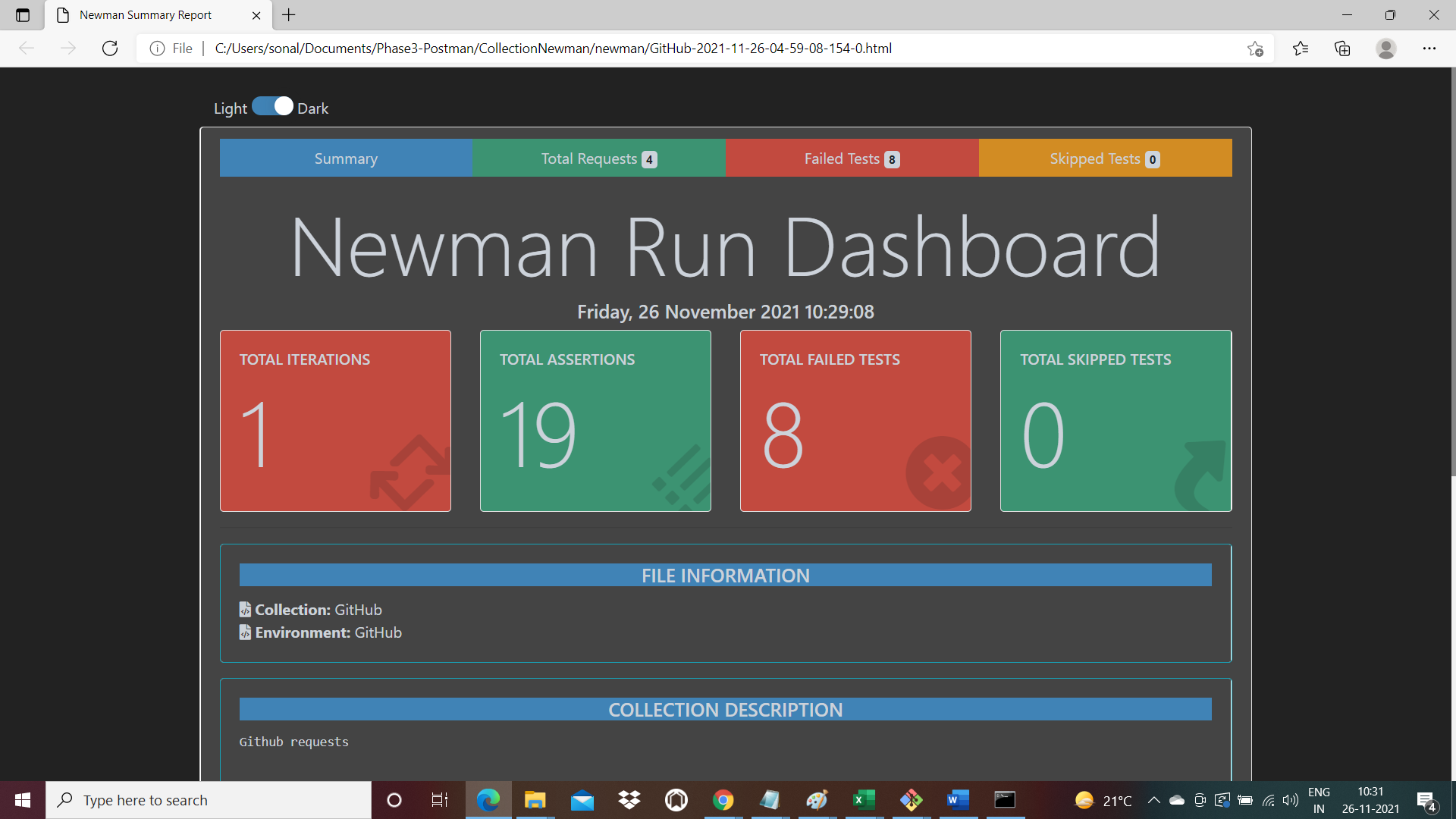
And execute below command

cd C:\Users\sonal\Documents\Phase3-Postman\CollectionNewman

newman run GitHub.postman\_colle

ction.json -e GitHub.postman\_environment.json -r htmlextra

C:\Users\sonal\Documents\Phase3-Postman\CollectionNewman>newman run GitHub.postman\_collection.json -e GitHub.postman\_environment.json -r htmlextra



# **Chaining API Requests in Postman**

It is a process in which we can connect all the requests

Like, output of one request becomes input to the next request.

Lets create a Post request for github.

This request will create a repo in github with random name

Request 1:

CREATE a REPO with system generated name

Request type : POST

URL : <https://api.github.com/user/repos>

Here url can also be an environment variable like this:

url = <https://api.github.com>

and request can be given as

{{url}}/user/repos

Authorization : Inherit form parent

Body 🡪 raw 🡪json

{

  "name": "*{{repoName}}*",

  "description": "This is your first repository",

  "homepage": "https://github.com",

  "private": **false**,

  "has\_issues": **true**,

  "has\_projects": **true**,

  "has\_wiki": **true**

}

Above json body we are passing repo name as variable

Whose value is captured from pre-req script

Pre req script to generate repo name

var repositoryName**=** "SampleApi" **+** pm.variables.replaceIn('{{$randomInt}}');

pm.environment.**set**("repoName",repositoryName);

Under Tests:

Write a test that will take owner value from the json response body.

We have defined a variable json which will parse response body

Capture value of owner🡪login value and set it as environment variable =🡺 json.owner.login

Name of env variable : owner

var json **=** JSON.**parse**(responseBody)

pm.environment.**set**("owner",json.owner.login);

Save the request.

Run it to check

* Random name repo is created
* In Enviromnets a new env variable owner is available with value as sonal0409.

Create a second request:

Request to query a specific repository

Request type: GET

Request URL :

<https://api.github.com/repos/:owner/:repos>

It needs 2 parameters, owner and reponame

Parameters:

Don’t hard code the value

Fetch from environment variables created in request 1 : owner and repoName

So parameter:

Owner = {{owner}}

Repos = {{repoName}}

Authorization : inherit from parent

Tests:

pm.**test**("Status code is 200", **function** () {

    pm.response.to.have.status(200);

});

pm.**test**("Status code name has OK", **function** () {

    pm.response.to.have.status("OK");

});

pm.**test**("Response time is less than 200ms", **function** () {

    pm.expect(pm.response.responseTime).to.be.below(600);

});

Run the request

It will return the repo name and details

Request 3:

Request that will delete the created repo

Request type : Delete

URL : {{url}}/repos/:owner/:repo

url : <https://api.github.com/repos/:owner/:repos>

It needs 2 parameters, owner and reponame

Parameters:

Don’t hard code the value

Fetch from environment variables created in request 1 : owner and repoName

So parameter:

Owner = {{owner}}

Repos = {{repoName}}

Authorization : inherit from parent

Tests:

pm.**test**("Status code is 204", **function** () {

    pm.response.to.have.status(204);

});

pm.**test**("Status code name has string", **function** () {

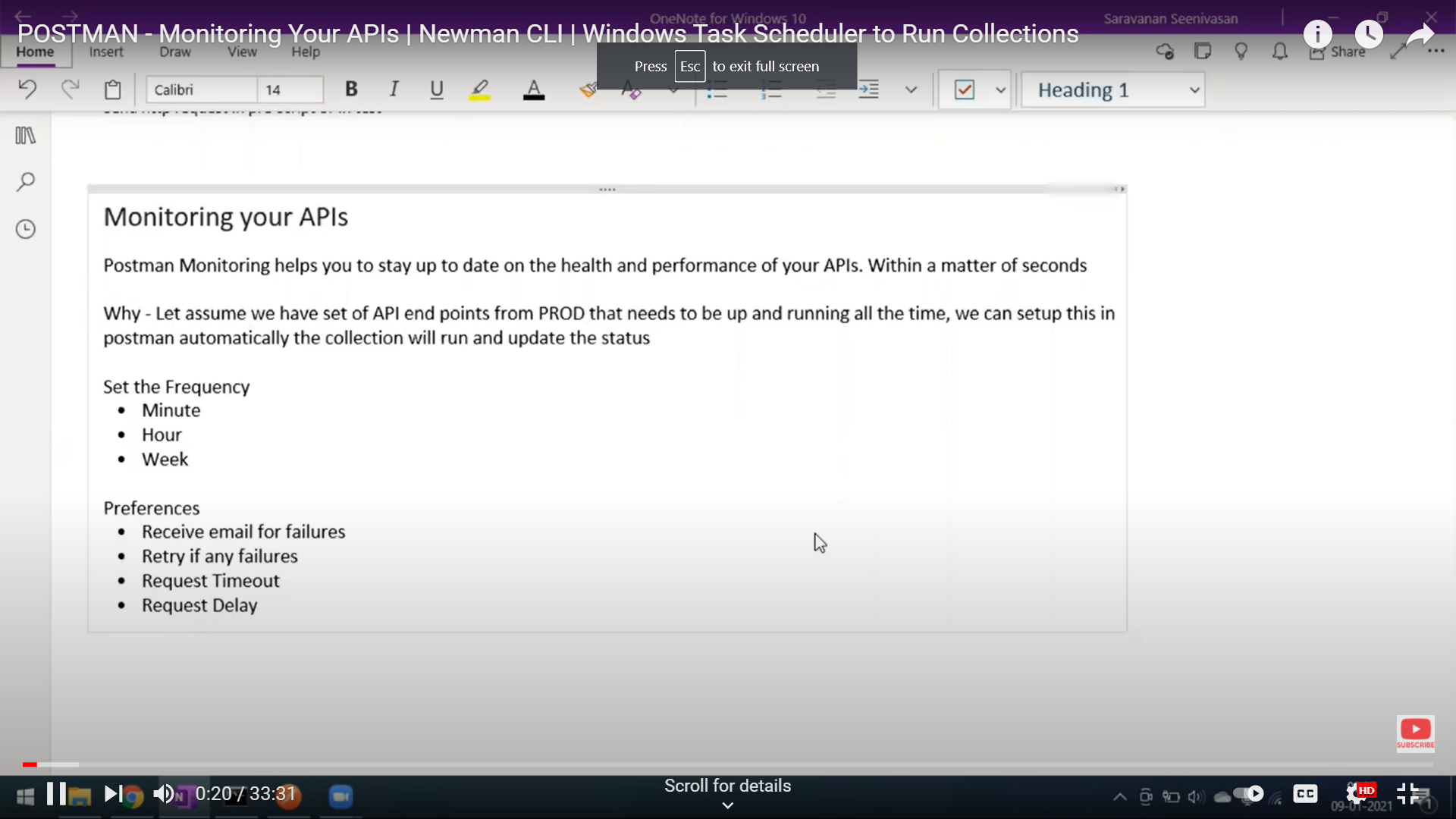
    pm.response.to.have.status("No Content");

});

Execute the request. It will delete the repo

Hence chained all 3 requests

# **Monitor API Collections in Postman**



# **OAUTH 2.0 Authorization using Gmail API**