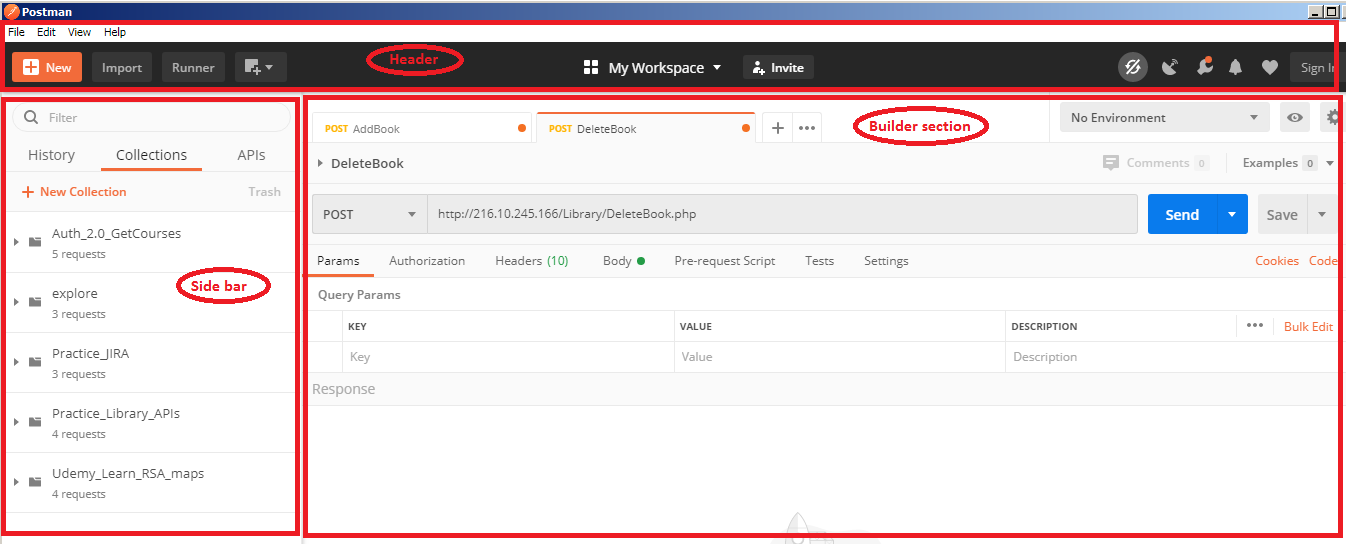
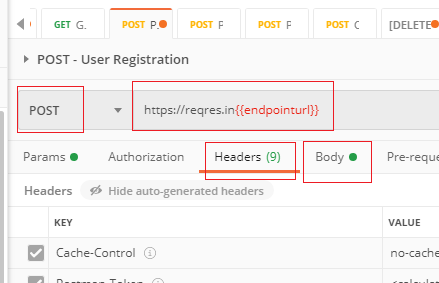
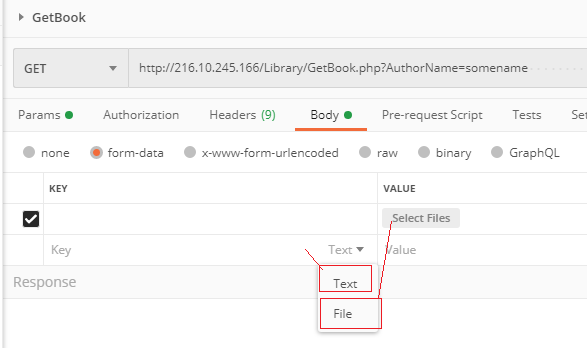
What is **Postman**?

* It’s an API Client
* It uses to develop, test, share, document APIs
* Postman UI:
* 
* HTTP Request mainly contains 4 Parts:
  + HTTP Method
  + Request URL
  + Header
  + Body
  + 
* In Body section, we could see 6 options
  + None: By default, Postman will select **None**—leave it selected if you do not need to send a body with your request.
  + Form-data: Website forms often send data to APIs as multipart/form-data. You can replicate this in Postman using the form-data Body tab. Form data allows you to send key-value pairs, and specify the content type. You can also attach files with form data.



* + X-www-form-urlencoded: It’s similar to form-data. URL-encoded data uses the same encoding as URL parameters. If your API requires url-encoded data, select x-www-form-urlencoded in the Body tab of your request. Enter your key-value pairs to send with the request and Postman will encode them before sending.
  + Raw: You can use raw body data to send anything you can enter as text. Use the raw tab, and the type drop-down list to indicate the format of your data (Text, JavaScript, JSON, HTML, or XML) and Postman will enable syntax-highlighting as well as appending the relevant headers to your request. You can set a content type header manually if you need to override the one Postman sends automatically. You can use variables in your body data and Postman will populate their current values when sending your request.
  + Binary: You can use binary data to send information you can't enter manually in the Postman editor with your request body, such as image, audio, and video files (you can also send text files).
  + GraphQL: You can send GraphQL queries with your Postman requests by selecting the GraphQL tab in the request Body. Enter your code in the Query area and any variables in the GraphQL Variables section.

* HTTP Response mainly contains 3 Parts:
  + Status Code
    - 2xx: Response is OK /successful
    - 4xx: Server did not understand your request; may be some issue in our request
    - 5xx: Some error at Server side; that’s not our fault
  + Body
  + Header
* HTTP Response can be saved for future reference. If we open the saved response, it will show corresponding Request details along with Response Details

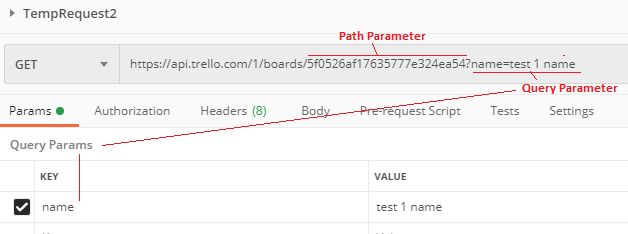
Path Parameters Vs Query Parameters

* Path and Query Parameters are part of URI
* Path parameter is basically used to identify a specific resource/resources
* E.g. Let's consider an example where you want identify the employee on the basis of employeeID, and in that case, you will be using the URI param.

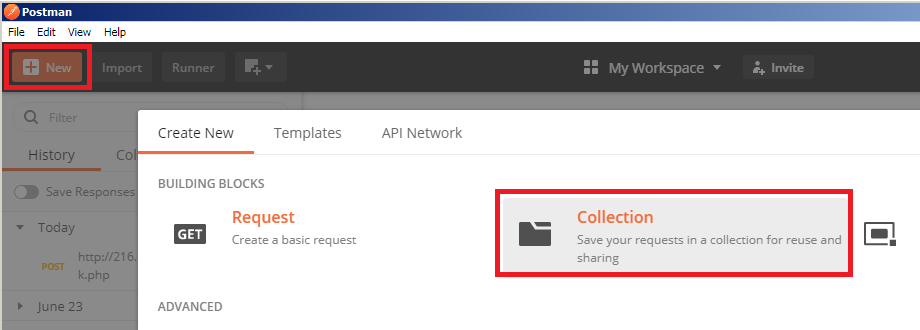
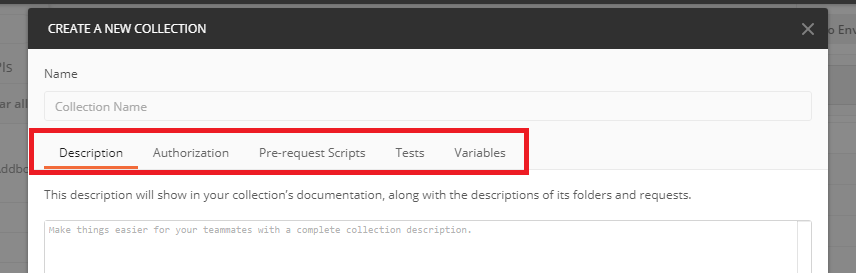
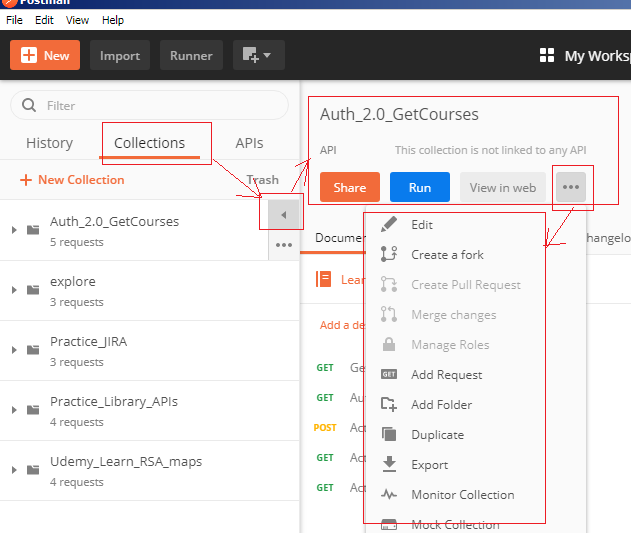
GET /employee/{employeeID}

* Query parameter is used to sort/filter those resources
* E.g. Take another example where you want to filter the employee on the basis of designation, and in that case, you will be using Query Parameter.

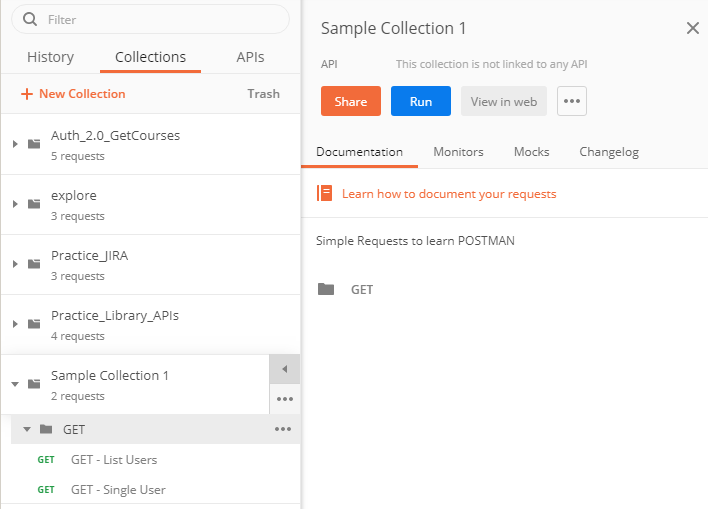
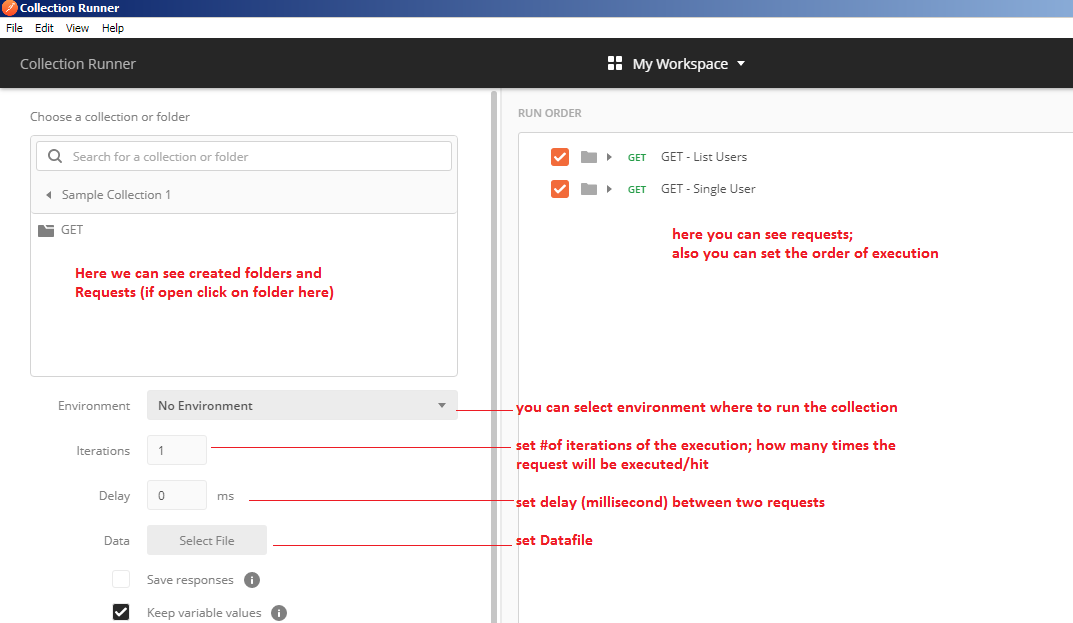
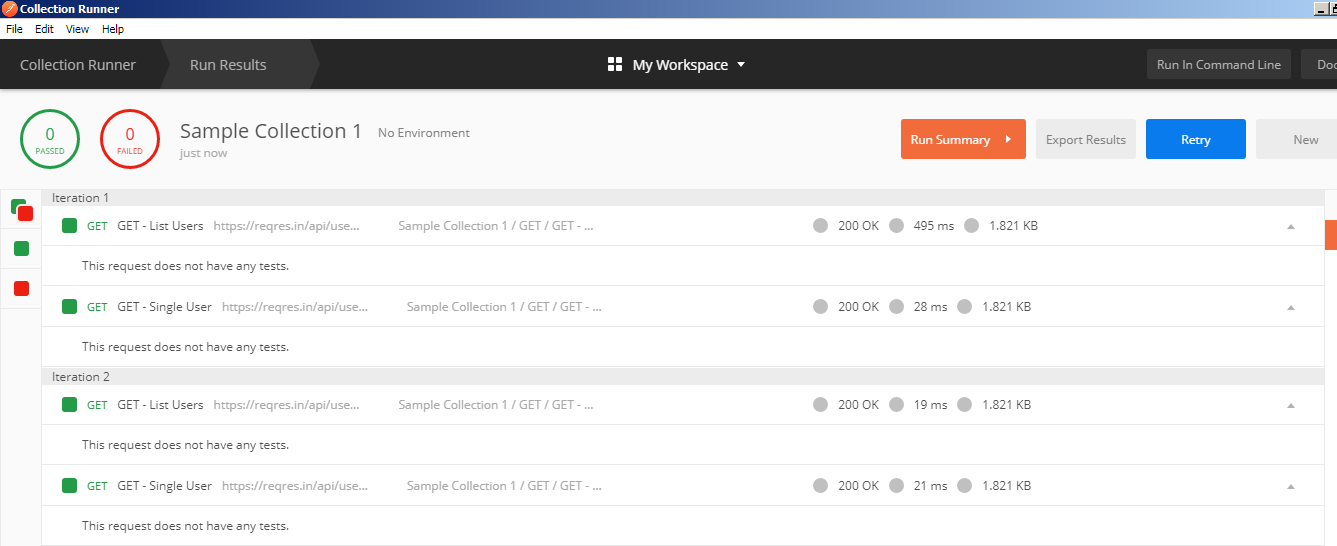
GET /employee?designation=SSE

* Into URI, Path parameter comes before “?”
* Query Parameter always comes after “?”
* 
* Into Path parameter, we can use KEY instead of value; as below:
* 
* Note that, Here path parameter key is preceded with **“:”** i.e. “:id”
* We may have multiple path parameters and multiple query parameters
* Query Parameters can be enabled or disabled i.e. can be removed or added using checkbox next to it
* Path parameters don’t have check box option to add/remove it from URI. Because, it’s intended path will be changed if you do so; query parameters are just like filtering criteria under same path.

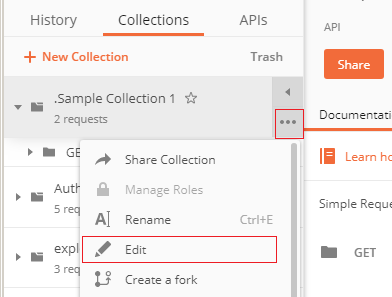
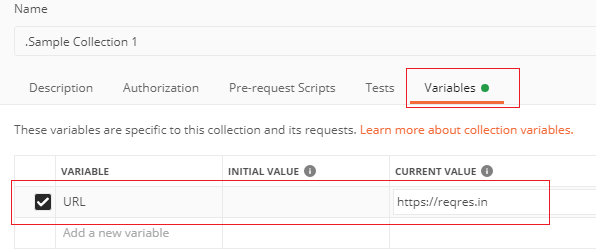
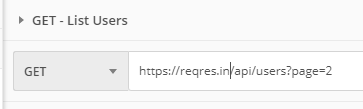
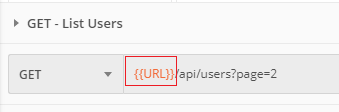
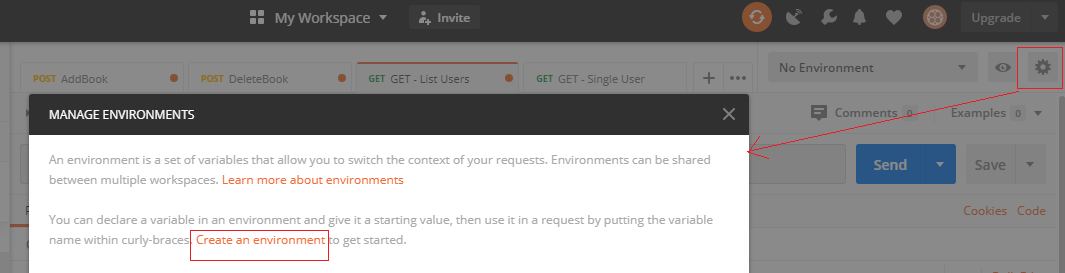
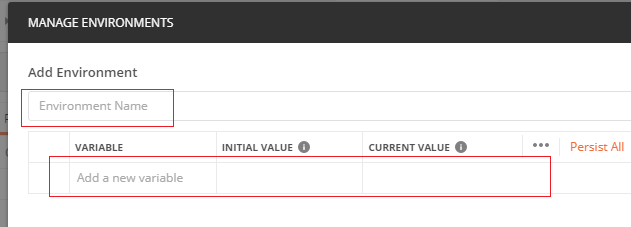
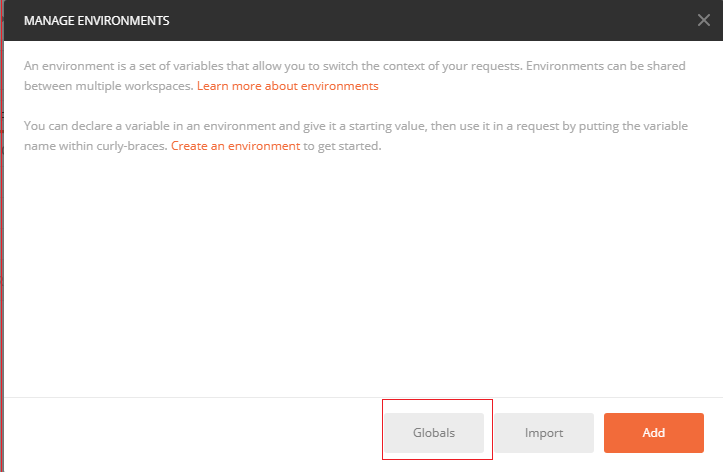
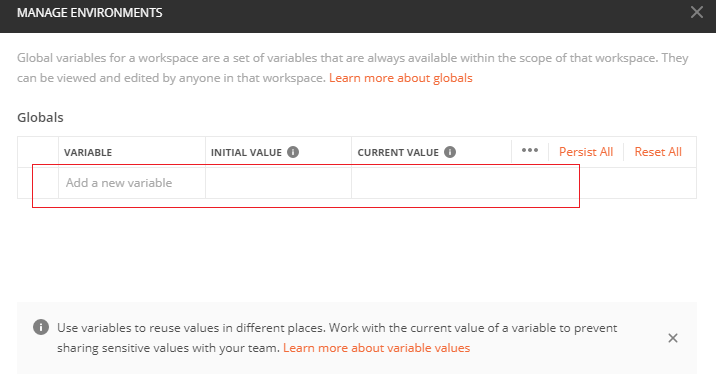
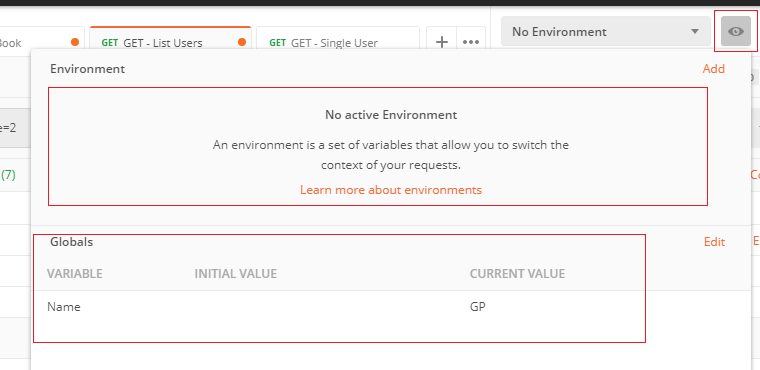
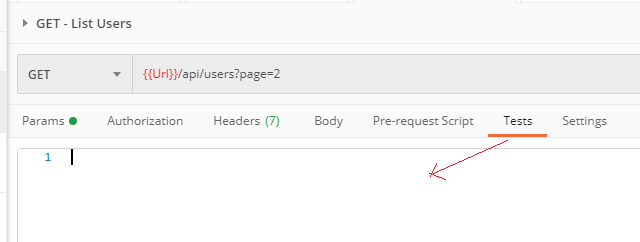
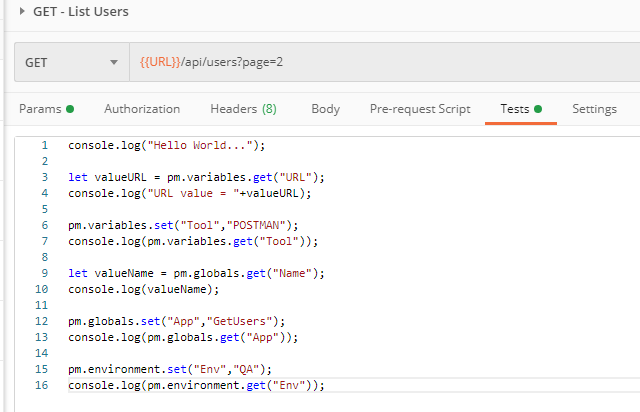
**Collection:**

* Collection is a group of API Requests; that can be stored and saved in logical arrangement   
  
* You can create a collection using menu … New > Collection
* On Create new Collection screen you can do below
  + Give Collection Name and Description (This description will show in your collection’s documentation, along with the descriptions of its folders and requests.)
  + Provide Authorization Type and corresponding details (This authorization method will be used for every request in this collection. You can override this by specifying one in the request.)
  + Pre-requisite scripts (This script will execute before every request in this collection.)
  + Give Tests (These tests will execute after every request in this collection.)
  + Provide Variables (These variables are specific to this collection and its requests)
  + 
* Authorization type, pre-requisites, tests, variables will be applicable to every request saved into the Collection.
* Also you can do followings with the collections:
  + Share
    - Export collection as a file / import from a file
    - Sharing a link
  + Run (run the entire collection)
  + View in Web
  + Edit
  + Add Folder
  + Add Requests
  + Create Duplicate
  + Export
  + …..etc
* 

How to **run a collection**:

* You can create a folder into Collection if you wanted to do
* Here I have created a folder “Get” under ‘Sample Collection 1’; This folder created to keep all GET method requests
* When you click the Collection arrow, you can see ‘Run’ button
* 
* When you click on ‘Run’, **‘Collection Runner’** window will get open
* 
* Once you run the collection, you will see below in **‘Run Result’** section
* 
* You can see Number Iterations, Requests, it’s result
* You can see Summary by using ‘Run Summary’ button
* You can Export the result in JSON format
* You can Retry the execution

**Variables:**

* It’s a same like other programming languages.
* Variables are nothing but an element that can store different values
* Why: To reuse same value at multiple places, to avoid repetition, to avoid rework when value changes
* Variables can be created from multiple places in Postman
  + **At “Collection” level**
  + **At “Environment” level**
  + **At “Global” level**
* **Let’s see at Collection level:**
* Click Collection “…” and Edit. You can create a variable there as below
* 
* 
* In the example, we used two GET requests, which has Same URL ‘https://reqres.in’.
* So, here we have created a variable for it and variable name is given as ‘URL’
* Now using the variable in endpoint url as below:
* 
* 
* Variable name is given in {{ }}
* Note that, Variables are case-sensitive
* Even though you have set the variable at Collection level, you can execute individual request.
* **Now let’s see at Environment Level:**
* Click on Settings > Create Environment > Add Variables
* 
* 
* **Now let’s see at Global Level:**
* Click on Settings > ‘Global’
* 
* 
* When you click ‘Quick Look’ icon, you can see created Global and Environment variables
* 
* We can see the logs in **Postman Console**.
* You can launch it from menu View > Show Postman Console
* In above examples, we have variable in endpoint uri. However in console complete valid URI will be shown (not with variable; variable being replaced with it’s value).
* Now let’s see how to **Set and Get Variables thru Scripts**
* When you at Request level, you can add the script under ‘Test’ section
* 
* 
* Below script is being used to PRINT in postman console

console.log("Hello World...");

* Below script is used to get the variable value; “URL” is a variable created at collection level.

pm.variables.get("URL");

* Here, we are assigning a variable value to another variable

let valueURL = pm.variables.get("URL");

* Below script is used to set a variable value to ‘Tool’ variable

pm.variables.set("Tool","POSTMAN");

* Similarly we can get and se the values at global and environment level:

let valueName = pm.**globals**.get("Name");

console.log(valueName);

pm.**globals**.set("App","GetUsers");

console.log(pm.**globals**.get("App"));

pm.**environment**.set("Env","QA");

console.log(pm.**environment**.get("Env"));

* We can get the variables using this script also

pm.**variables**.get("Env");

* When this statement gets executed, it will check all the levels for the variable (i.e. at collection level, environment level, global level, etc.) and it will get from the closest level (which has higher preference; which is discussed below)
* Note that we can also do such scripting at Collection level too. Goto Collections > Edit >
* And do scripting in ‘**TEST’** or **‘Pre-request scripts’**
* ‘Test’ can be created at REQUEST, FOLDER and COLLECTION levels
* Scripts in ‘Test’ will be executed once Request is executed and response is received
* Scripts in ‘Pre-request scripts’ will be executed before sending a request. It’s based on whether it’s written in Collection level or individual request level; it will get executed accordingly.
* Pre-request script generally used to create/set dynamic values or variable values required for a request
* Let’s see variable scope based on different levels  
  
* Variable preference is Environment, Collection and then Global i.e. if same variable is defined with different values at different levels, then at Environment level value will get higher preference than Global level
* We cannot change Collection variable values using script; So do store dynamic type variables (which changes values during execution) at Collection level ; constant type of variables can be stored at collection level.
* We can change Global and Environment variable values using script
* Data variables gets used when working with multiple data sets for a request. i.e. If the request has to execute with multiple data-sets; it exists only during the execution of iteration; It can only be set from CSV or JSON file
* We can unset or delete a variable using this script

pm.**globals**.unset("variable\_key");

pm.**environment**.unset("variable\_key");

* We can unset or delete all the variables using

pm.**globals**.clear();

pm.**environment**.clear();

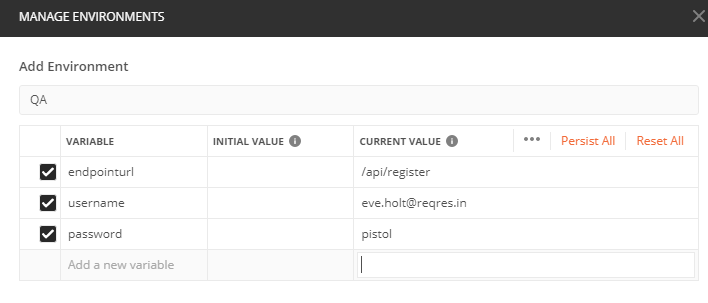
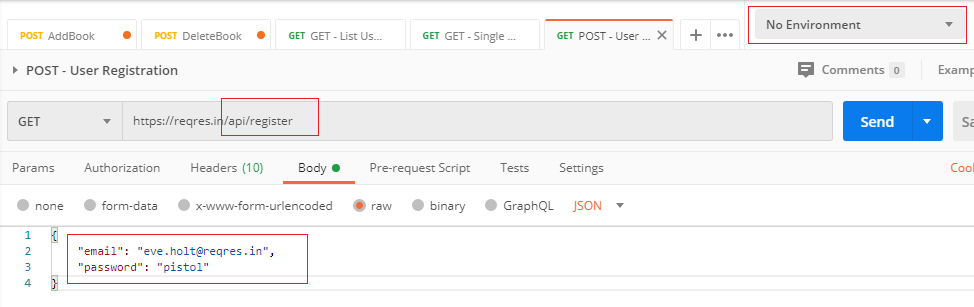
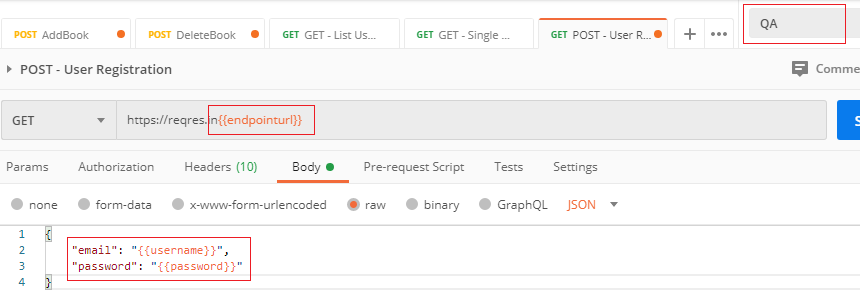
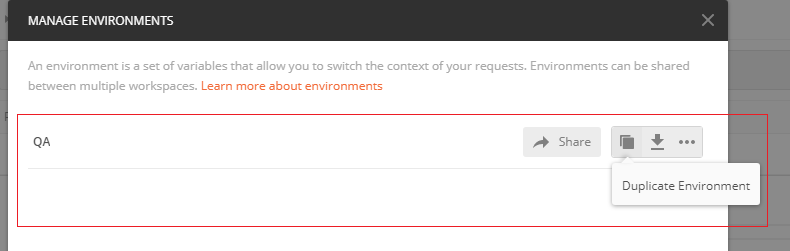
* When you create a variable, you can see ‘Current value’ and ‘Initial Value’ next to it.
* Values in **‘Current value’** will be visible to you and will be sent with request. And values in **‘Initial value’** will be visible to other team members when you share the collection/request. So your confidential data like IDs/Tokens etc. will be with you only.
* Verify first time ‘Current value’ and ‘Initial value’ holds same info; you have to change it.
* When you set variable value, using script set(), same value will be set to ‘Current Value’ and ‘Initial value’
* If you wanted to set the value only under ‘Current value’ and not under ‘Initial value’ then you have to Disable this settings …. **Menu File > Settings > General > Automatically persists variable values**
* You can copy all the variables at a time and paste it at different level; you can do it by selecting all the variables by Ctrl+Click, then Copy with Ctrl+C and paste it with Ctrl+V
* How to add Variables in request URL???

**Environment**:

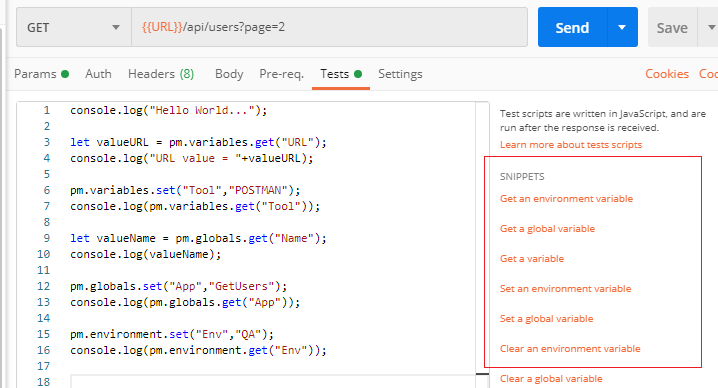
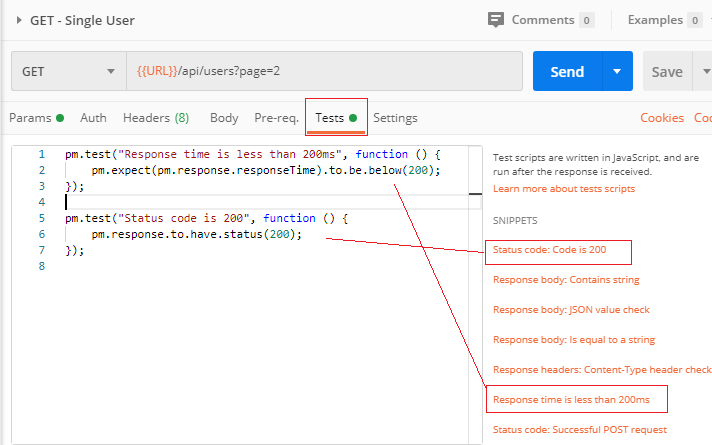
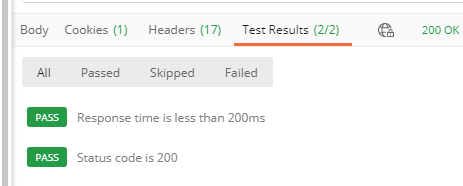
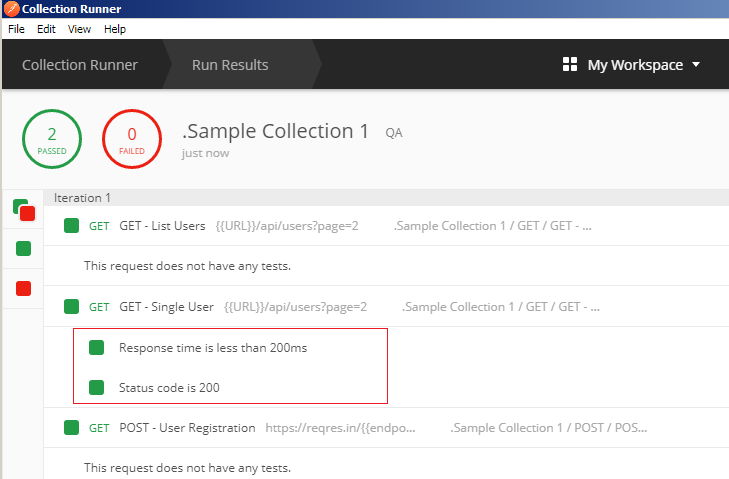
* Environment is a set of Key-Value pairs
* We can use these key-value pairs and refer them in Request, Test and pre-request scripts
* In real time, we may have to run the requests in multiple environments; means we have to use different urls, username, passwords, values etc.
* There are some values which changes as per different environment as mentioned above; such key-value pairs can be group together and set into an environment
* Let’s see how to do that and how to access it
* We can create environment from different ways

i.e. Click on Settings icon > Manage Environment > Create environment.

Also, click “New” button (top left) and click on ‘Environment’

* Below, ‘QA’ environment is created with different variables with their values
* 
* Lets see how to use these variables; before that see below snapshot without variables
* (Note: in below screenshots, HTTP method are wrongly selected as ‘GET’; assume those are ‘POST’)
* 
* Below is with environment variables.
* For that Select ‘Environment’ first in top right corner and then replace variables wherever applicable
* 
* You can easily create duplicate environment and just change the variable values accordingly
* 
* Advantage is, when you run same test on different environment, you just have to select Environment name from top right dropdown.
* In Collection Runner also, you have an option to select Environment.

**Test Script Creation:**

* In above section, we saw some info about script (where to create, difference in ‘Test’ and ‘pre-request script’ sections etc)
* One more thing that Script Snippet (templates) was also available and they can be added to your script
* 
* You can click the snippet you want, the script template will be added into the scripting area
* ‘Test’ can be created at REQUEST, FOLDER and COLLECTION levels
* Test: Postman Tests are javascript code that is executed after receiving the response.
* Pre-request script: these script are javascript code that is executed before sending the request.
* Now let’s create a small test at Request level
* 
* Here, we created two tests from the Snippet.
* One, to check if response time is less than 200ms
* And another, to check response status code is 200
* “Response time is less than 200ms” and “Status code is 200” are the Test Names. We can change the test Names
* Test Names reflects into the test result/report
* Once you send the request and response is received, you can see Test Result in “Test Results” section.
* 
* Similarly, when you send the requests from Collection Runner, you can see the Collection Result along with Test Result.
* 
* You can debug or see the logs of the test in Postman Console window
* In console window, you can see Request and Response Logs details including sent/received header, parameter, body etc.
* Also, you can see request/response in ‘**Pretty’** (easy to read for user) and ‘**Raw’** (actual format which is sent and received) format.
* Note that, the information in console window persists until the window is closed; once that console window is closed, logs will be cleared.
* You can use below statements in test or pre-request script to print info in console

**Console.log**(“This is a simple log message”);

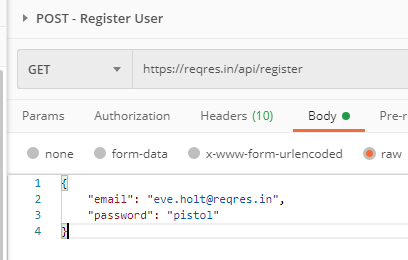
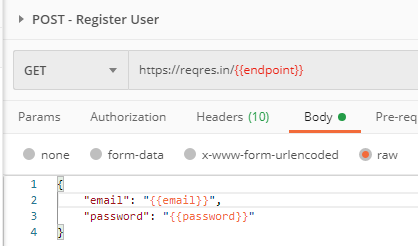
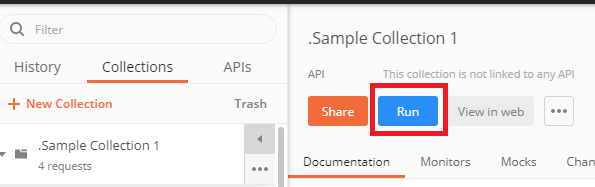
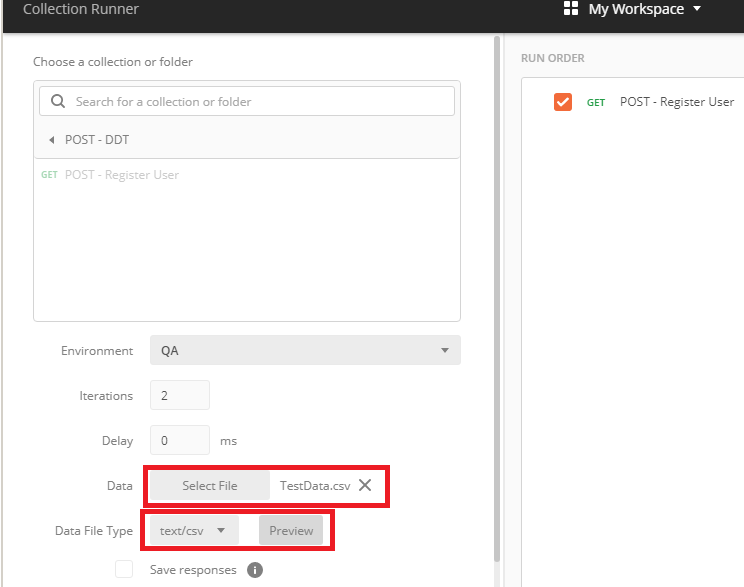
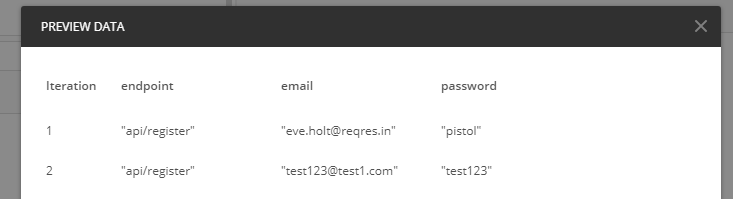
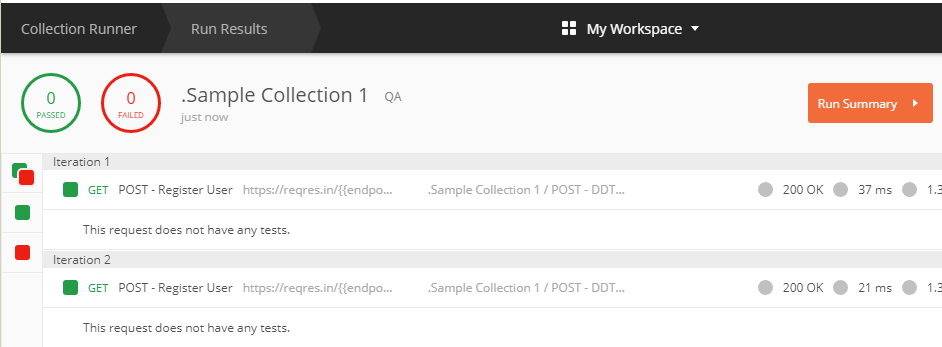
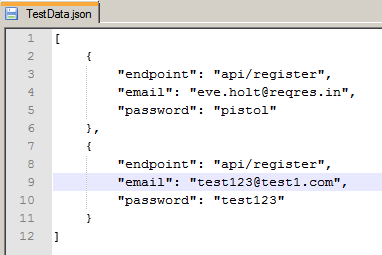
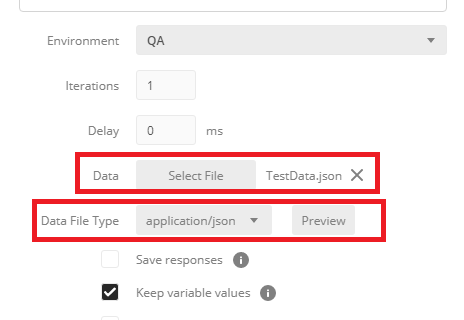
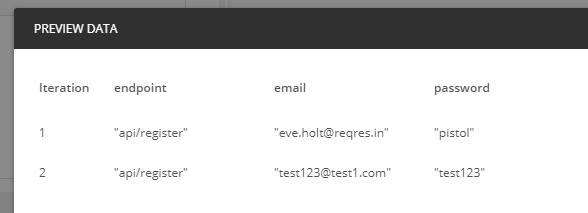
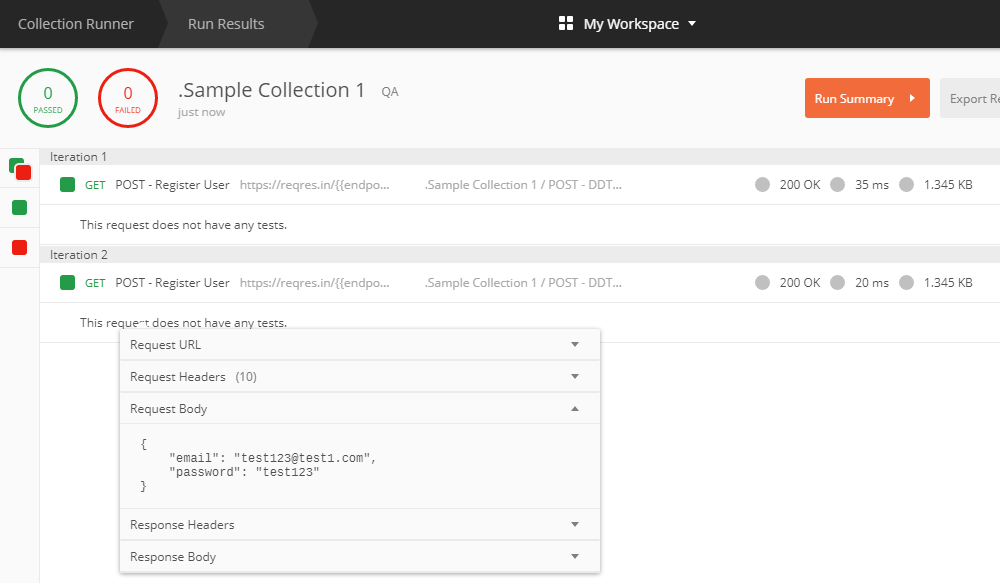
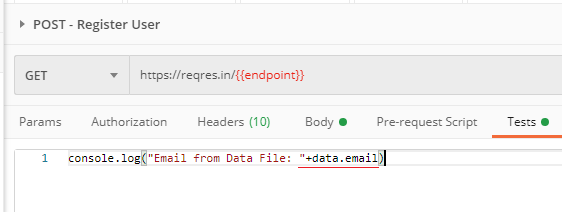
**Console.info**(“This is a info message”);

**Console.warn**(“This is a warning log message”);

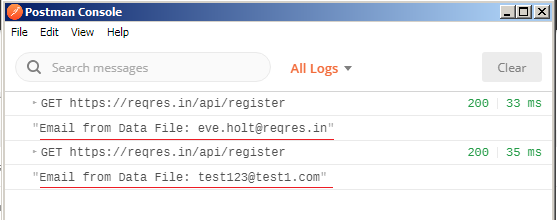
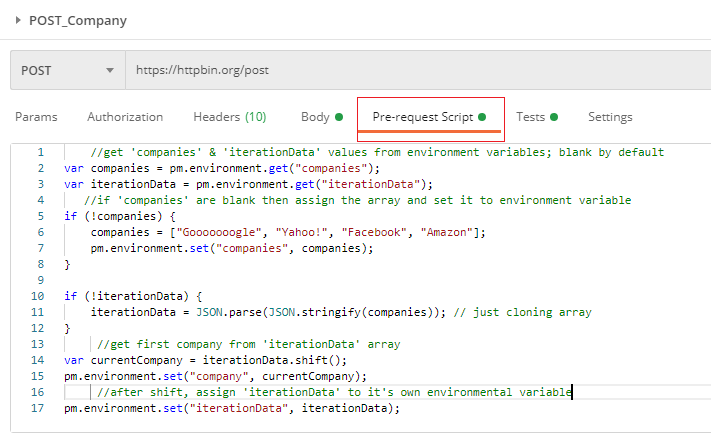
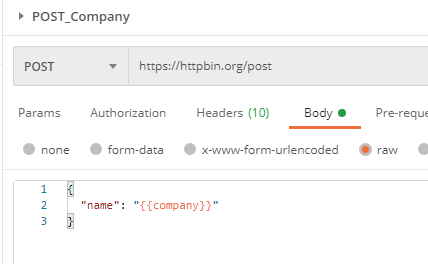
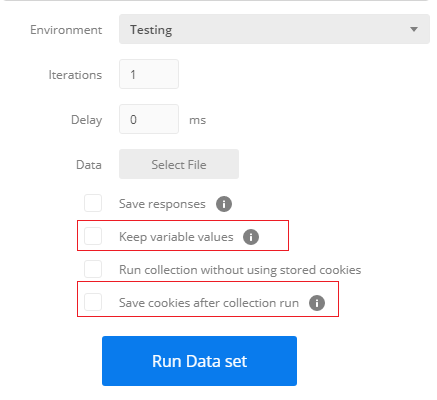
**Console.error**(“This is an error log message”);

* Similarly, you can see more logs from Developer Tools > Element / Console/Source etc
* For that Go to menu View > Developer > Show Dev Tools

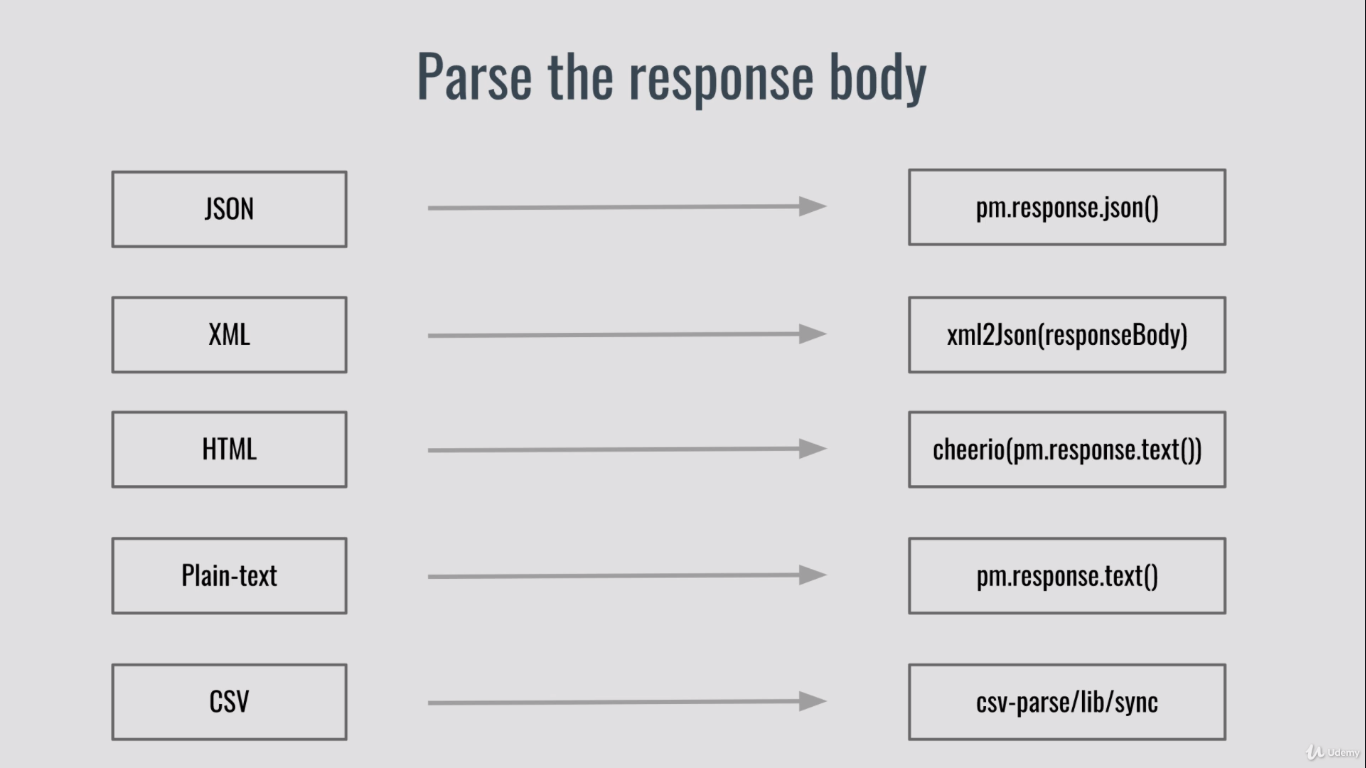
**Data Driven Testing:**

* We will see how to get the data from .csv, .json file and then how to use them in data driven testing.
* Above we saw that how to get/read the data from variables at different levels like Request, Environment, Global and use the data in Request and Test/scripts.
* Now, we will get the data from external files like .csv and .json.
* We will use below request with csv/json data file
* (Note: in below screenshots, HTTP method are wrongly selected as ‘GET’; assume those are ‘POST’)
* 
* We will get Resource URL, Email and Password from data files.
* Below .csv file is created with these info
* 
* Now add these variables into the request as below
* 
* Now , when you run the request thru Collection Runner, there is an option to select a Data File
* Select the Data File there
* Preview it; just to make sure if data file is looking as expected
* And run the collection runner
* Based on the selected data file, request will be executed;
* In this case, two iterations will be executed since data file contains two records of test data
* 
* 
* 
* 
* Note that, here you can or cannot define these variables into Environment or global level.
* Even though you don’t define, it will pick up from data file based on header
* Similarly can be executed for .json file as Data file. Below is .json file created
* 
* 
* 
* 
* Now, let’s see how to get data from external files into TEST or Pre-Request Scripts
* Like we get variables values in {{ }} in above example, we cannot use it same way in TEST or Pre-Request Scripts
* We have to use **“data.”** object to access the variable as below:
* 
* Another way to get data value from data file,

**pm.iterationData.get('email');**

* After running the request thru collection runner, below is the output you can see in Postman console
* 
* NOTE: Data variables can be used wherever Environment variables can be used (except in TEST and Pre-Request script)
* Let’s see Another way or method of Data Driven
* If there is an API Post request which creates a company and in it’s body we provide company name as an input
* Now assume you have to test different company names like having special chars into it, long name etc.
* There are different ways to do that like
  + Create duplicate requests and change the test data in request body => ( not suitable solution for multiple test data)
  + Use external data file => (like we saw in above section; but we can use this when run it with newman or collection runner)
  + Use pre-requisites scripts, variables and workflows => (we are going to see this here)
* 
* Defined ‘iterationData’, ‘companies’, ‘company’ as environmental variables.
* The **shift**() method removes the element at the 0th index and shifts the values at consecutive indexes down, then returns the removed value. If the length property is 0, undefined is returned.
* 
* 
* When you run this script using collection runner with below settings, it got executed 4 times and post all the companies from an Array one by one.
* 
* And when you run the request individually, very first time ‘Gooooooogle’ will be posted. And when run the request again, on second time ‘Yahoo!’ will be posted, 3rd time execution ‘Facebook’ will be posted and so on.

**Assertion:**

* Generally it’s two-step process i.e. Parse the Response Body and write the test code to validate something in response body
* We may get response in different formats like JSON, XML, Plain-Text etc.
* Below is the list of response body formats and methods to parse them
* 
* **Chai Assertion Library ??**
* Below are few examples of Assertions; let’s understand them
* Comparing 100 with 100; Assertion will be passed here

**pm.test(“Your Test Name”, function() {**

**pm.expect(100).to.eql(100);**

**});**

* Comparing 100 with 101; Assertion will be failed here. Also, you can give message that will be print if assertion got failed

**pm.test(“Your Test Name”, function() {**

**pm.expect(“John”).to.eql(“Jane”,’Expected is Jane’);**

**});**

* Comparing two objects; here assertion will failed since object ‘b’ has more properties than ‘a’

**pm.test(“Your Test Name”, function() {**

**let a = { “name”: “John”};**

**let b = { “name”: “John”, “age”: 20};**

**pm.expect(a).to.eql(b);**

**});**

* This is invert assertion; looking at above defined object ‘a’ and ‘b’, below assertion will be passed

**pm.expect(a).to.not.eql (b);**

* Let’s see what is difference between below equals assertions

**pm.test(“Your Test Name”, function() {**

**let a = { “name”: “John”};**

**let b = { “name”: “John”};**

**pm.expect(a).to.eql(b);**

**pm.expect(a).to.eqlual(b);**

**pm.expect(a).to.eqlual(a);**

**});**

.equal(): check the if the object is the same with which you are comparing

In above code, 2nd assertion will be failed since object ‘a’ is not object ‘b’ even though it has same parameters.

3rd assertion will be passed

1st assertion will be passed; because .eql() compares ‘a’ with ‘b’.

* Assertion can be made for Boolean values

**pm.expect(true).to.be.true;** //passed

**pm.expect(false).to.be.false;** //passed

**pm.expect(true).to.be.false;** //failed

**pm.expect(null).to.be.null;** //passed

**pm.expect(undefined).to.be.undefined;** //passed

* Note that if the variable is just declared without any value; then it’s value is ‘undefined’; not null.
* We can check if array is empty

**pm.expect([]).to.be.empty;** //passed

**pm.expect([].length).to.eql(0);** //passed

**pm.expect([1,2,3]).to.include(2);** //passed

**pm.expect([1,2,3]).to.include(8);** //failed

**pm.expect(2)to.be.oneOf([1,2,3]);** //passed;similar for String

**pm.expect(‘John Ray’)to.match(/^John/);** //passed;checking if ‘John Ray’ string starts with ‘John’

* **Assertion of Arrays:**
* Below is sample JSON Response Body. Here you wanted to verify if email id is “[lindsay.ferguson@reqres.in](mailto:lindsay.ferguson@reqres.in)” whose first name is “Lindsay”

{

    "page": 2,

    "per\_page": 6,

    "total": 12,

    "total\_pages": 2,

    "data": [

        {

            "id": 7,

            "email": "michael.lawson@reqres.in",

            "first\_name": "Michael",

            "last\_name": "Lawson",

            "avatar": "https://s3.amazonaws.com/uifaces/faces/twitter/follettkyle/128.jpg"

        },

        {

            "id": 8,

            "email": "lindsay.ferguson@reqres.in",

            "first\_name": "Lindsay",

            "last\_name": "Ferguson",

            "avatar": "https://s3.amazonaws.com/uifaces/faces/twitter/araa3185/128.jpg"

        },

        {

            "id": 9,

            "email": "tobias.funke@reqres.in",

            "first\_name": "Tobias",

            "last\_name": "Funke",

            "avatar": "https://s3.amazonaws.com/uifaces/faces/twitter/vivekprvr/128.jpg"

        },

    ],

    "ad": {

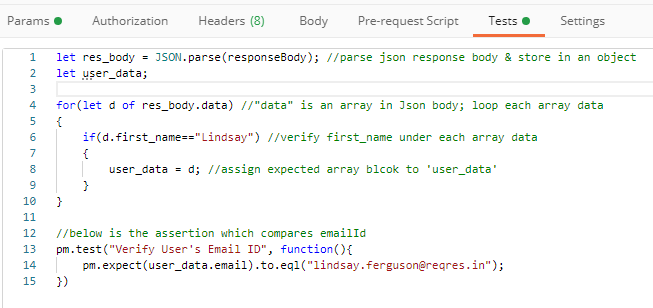
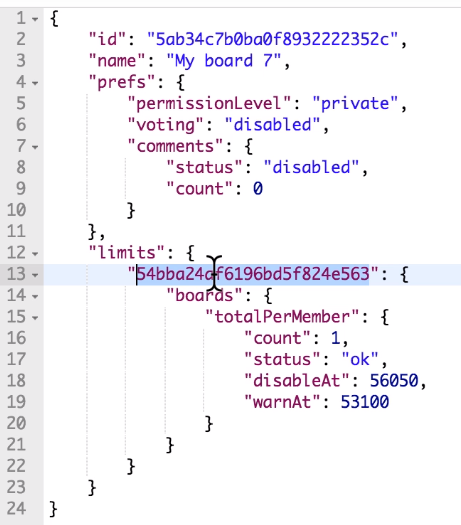
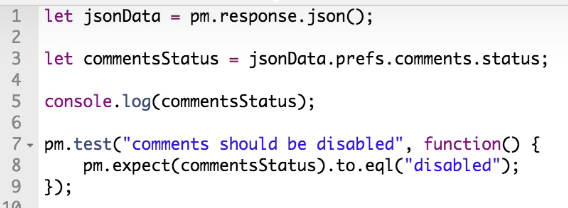
        "company": "StatusCode Weekly",

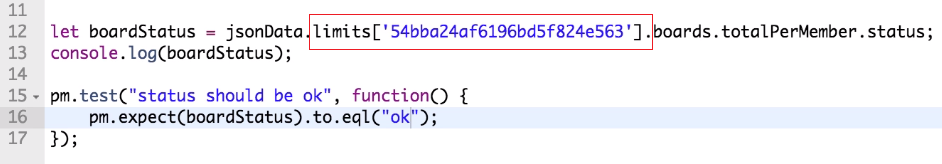
        "url": "http://statuscode.org/",

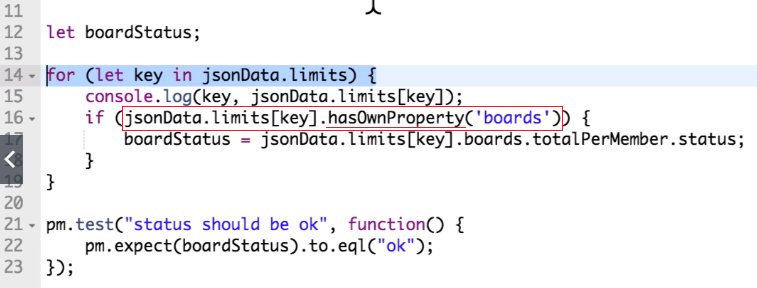
        "text": "A weekly newsletter focusing on software development, infrastructure, the server, performance, and the stack end of things."

    }

}

* Here, “data” is an array where you will get list of users and their details. Below code shows how to read each array object, find the one in which first\_name is ‘Lindsay’ and then get corresponding email id and check it with expected email id.
* 
* **Assertion on Nested Object:**
* Below is the sample Response body.
* 
* In below code, it’s verified if ‘prefs’ > ‘comments’ > ‘status’ is ‘disabled’
* 
* Below code shows, if ‘limits’ > ‘54bba…..’ > ‘boards’ > ‘totalPerMember’ > ‘status’ is ‘ok’
* Here, ‘54bba….’ Parameter cannot be accessed directly with dot (.) operator so; it’s handled as shown below:



* However, seems that the property ‘54bba…’ is dynamic. It may fail in another round.
* To handle this, we have to loop thru all the properties under ‘limits’ and check whose property is ‘boards’ (assume that ‘boards’ property is under only expected ‘54bba…’ like item)
* 
* We can make assertions on Headers and Cookies in Response
* **Headers:**
* This is how you retrieve a header from the response:

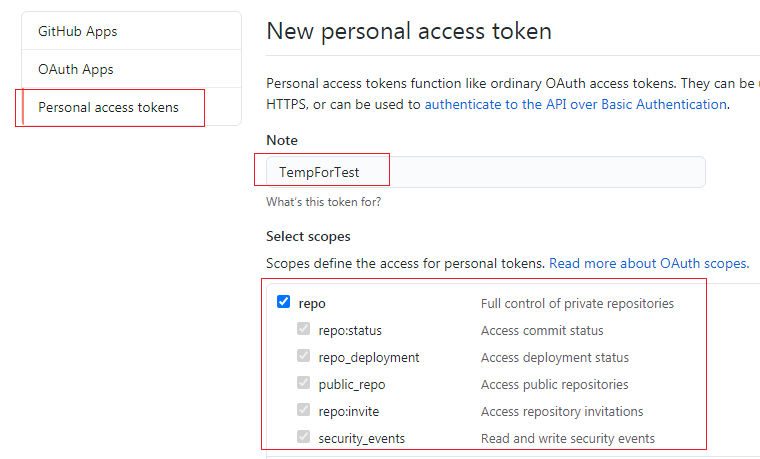
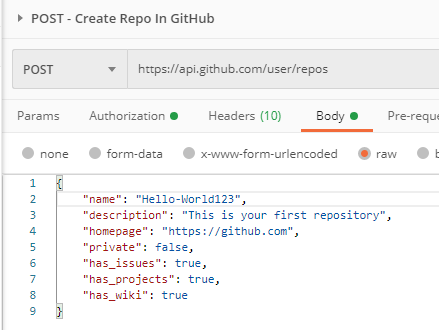
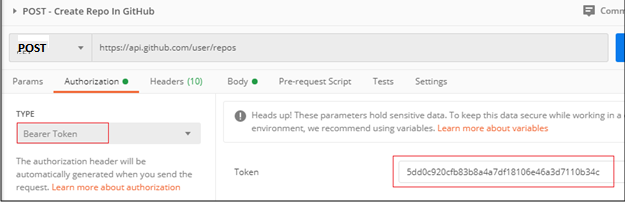
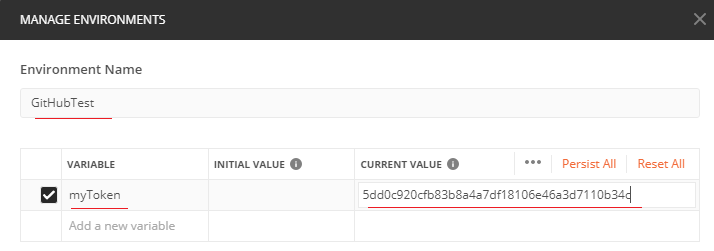
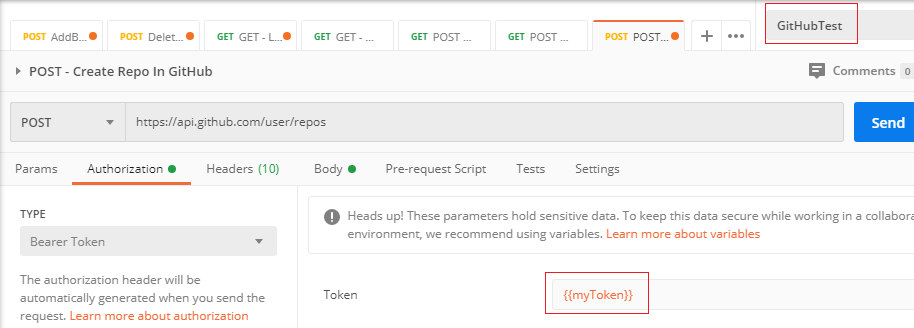
pm.response.headers.get('X-Cache')

* To check if header exists:

pm.response.to.have.header('X-Cache');

* Compare Header value:
* pm.expect(pm.response.headers.get('X-Cache')).to.eql('HIT');
* **Cookies:**
* In a similar fashion you can test cookies as well.
* Cookie exists:
* pm.expect(pm.cookies.has('sessionId')).to.be.true;
* Cookie has value:
* pm.expect(pm.cookies.get('sessionId')).to.eql(’a3se3s8sg7sg3');

**Authorization**

* Let’s see example of how to provide authorization token if request required authorization to get it successfully complete
* In this example, we will create a repository in GitHub; for that you need to login i.e. in the request, we have to provide valid token
* To generate the token in GitHub, Login to Gitub > Goto Settings > Developer Settings > Personal Access Token > Generate New Token > Give the Token Name > Select the Scope
* 
* Generate the Token and copy it
* 
* We will use that token into Postman request
* Get the Endpoint url for GitHub (search for GitHub Rest API in google > Goto the <https://developer.github.com/v3/> , you will get endpoint url <https://api.github.com> in there)
* Find out the ‘Create new Repository for authenticated user’ rest api on the same site and get the Resource URL (user/repos), also you will get required parameters there (‘name’ is required parameter for this api), get the example body and create a request in Postman; as below
* 
* NOW, add the token into ‘Authorization’ section
* You can select the Type as ‘Bearer Token’ or ‘Oauth2.0’
* 
* Send the request.
* “Hello-World123” Repository will be created in your GitHub Account.
* In above screenshot, you can see there is a note stating that ‘These parameters hold sensitive data. To keep this data secure while working in a environment, we recommend using variables’.
* Here token is a sensitive data.
* We can keep it in Global or Environment variable and use that variable in ‘Authorization’ section
* Creating variable in Environment level
* 
* Note that, values in **‘Current value’** will be visible to you and will be sent with request. And values in **‘Initial value’** will be visible to other team members when you share the collection/request. So your confidential data like IDs/Tokens etc. will be with you only.
* Verify first time ‘Current value’ and ‘Initial value’ holds same info; you have to change it.
* Select the Environment at top right corner and add the variable {{myToken}} in ‘Authorization’ section
* 
* Similarly, we can provide the ‘Authorization’ at ‘Collection level’ as well as ‘Folder level’.
* Collection > Edit Collection > Authorization
* Here that provided authorization will be application to all the requests into that collection.
* If you observe, you can see the section for ‘Authorization’ in Postman but not for ‘Authentication’.
* Because, using endpoint url and resource url we try to access the direct resource of an application or system; so it’s appropriate to say here as ‘Authorization’ rather than ‘Authentication’.
* **Authentication**: Proving your identity/validating user. E.g. For emails, you have to login with credentials then you will have complete access to your stuff or your account
* **Authorization**: giving/getting limited access. E.g. you may give access/permission of your one folder to a person; meaning that person is authorized to access that one folder
* Another example is, you logged successfully into a website or your office computer with your credentials; meaning you are Authenticated to use or perform action into your computer; however some sections like using USB or Sys file are not accessible to you; meaning you are not Authorized to access certain areas.
* **Two-Factor Authentication (2FA):** It’s additional layer of security on top of current authentication mechanism.
* Mostly it takes the form of OTP (One-Time Password) or TOTP (Time based OTP – this otp of changes every 30-40 seconds like I used as RSA/Octa token to login office n/w) as an additional authentication
* In case of 2FA, sometime you receive SMS on your phone for OTP; you have to enter OTP into the request i.e. there is manual intervention
* Process becomes like Send Request to Authentication URL > Receive OTP >Send another request with received OTP > get response
* Here Receive OTP and enter OTP into another request is a manual intervention
* To avoid manual intervention, we can use different tools like ‘twilio’
* (About Twilio: when you register with Twilio on their website, you can get a phone number; you can use that phone number to receive OTP and then using Twilio’s APIs you can get received SMS as a response, and get the OTP from it and use it in another request)
* E.g. If you set 2FA with GitHub account, use phone number which got from Twilio. When you send a request to GitHub’s Auth server, it will send the SMS to the number. Create another request which get the received SMS from Twilio (using it’s corresponding API); get the OTP from the response and use that OTP into final request.
* There are few challenges with this approach:
  + It may take some delay to receive OTP SMS into Twilio account; so when you send request to Twilio to get the latest SMS, you will not get it or you will get older sms
  + To overcome this, you may have to re-try this part 2-3 times to get some luck
  + If the execution of collection takes longer time, then OTP/authentication may be expired since the OTP is valid for few seconds

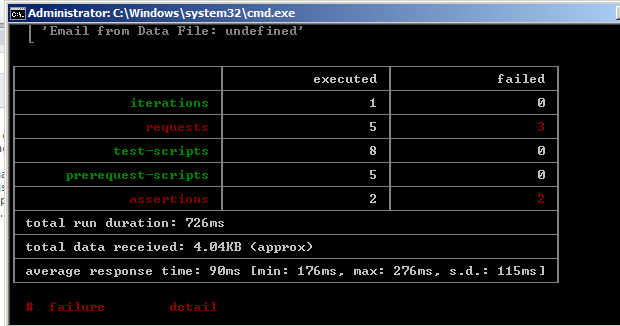
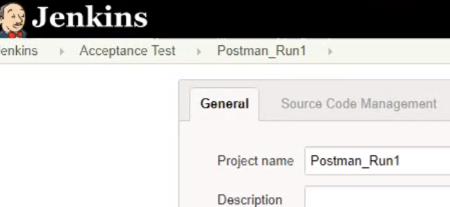
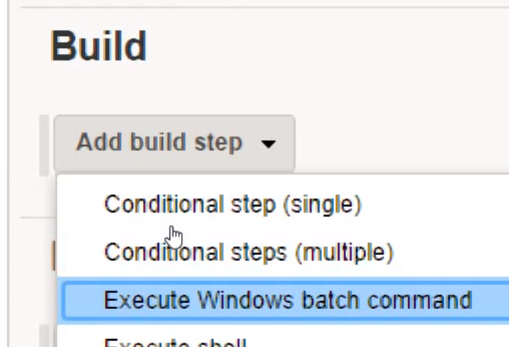
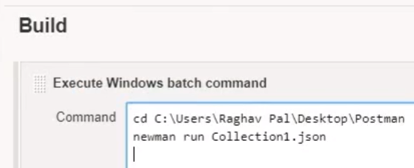
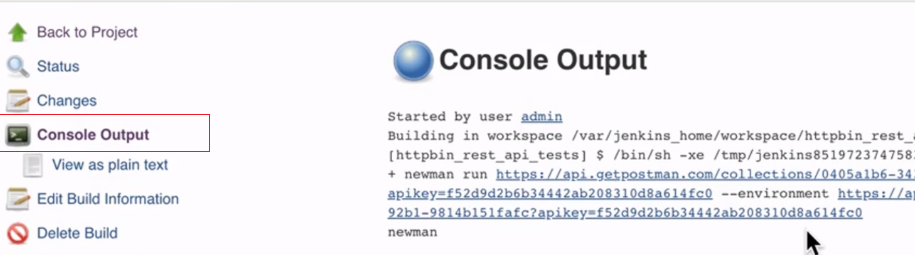
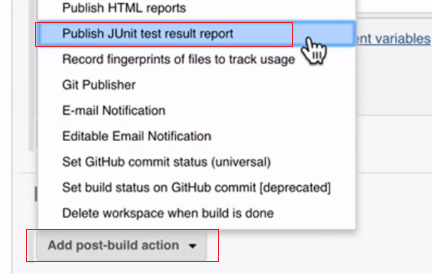
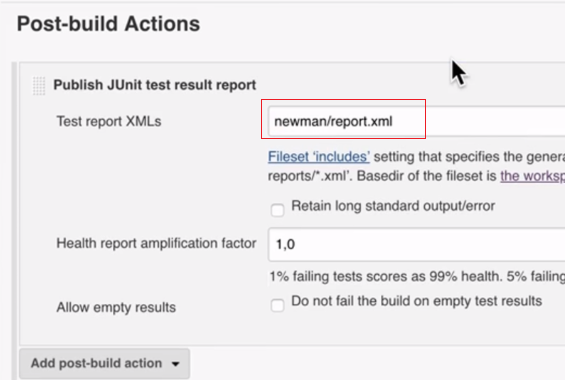
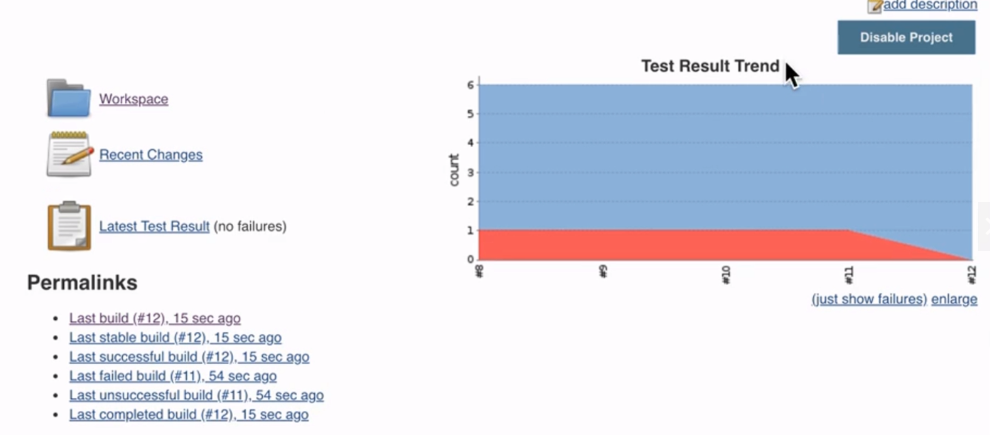
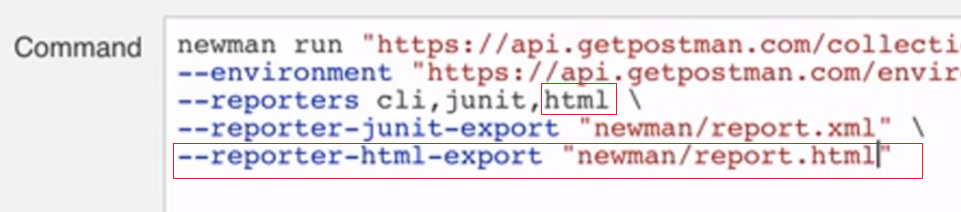
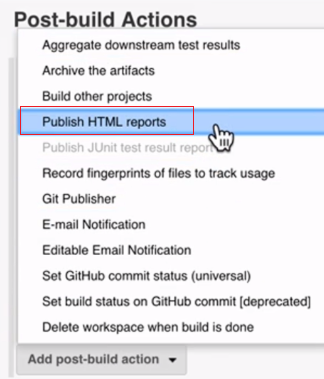
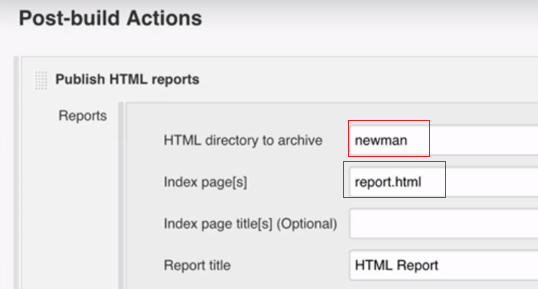
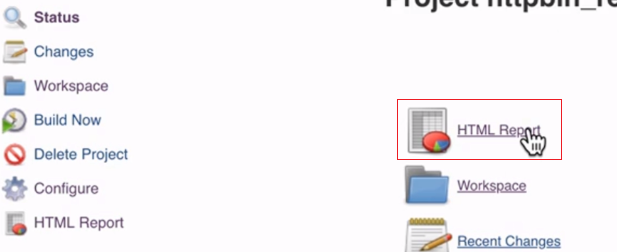
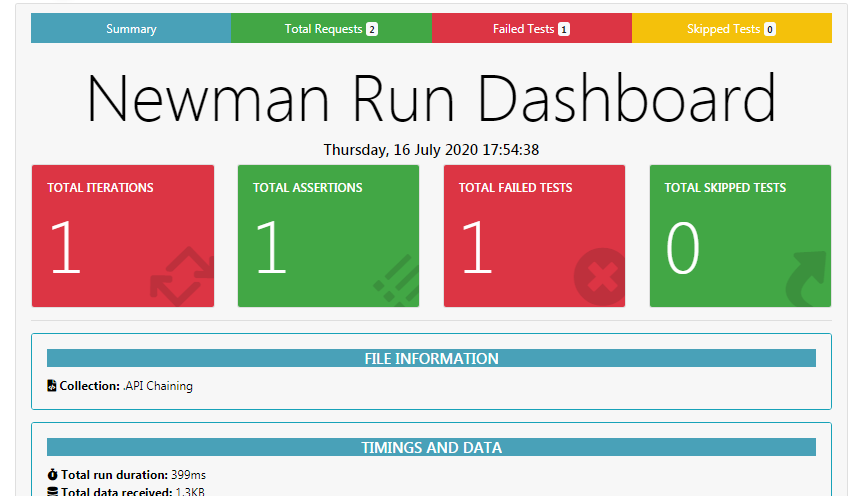
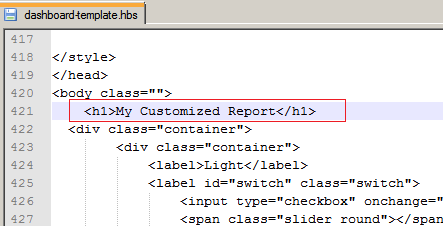
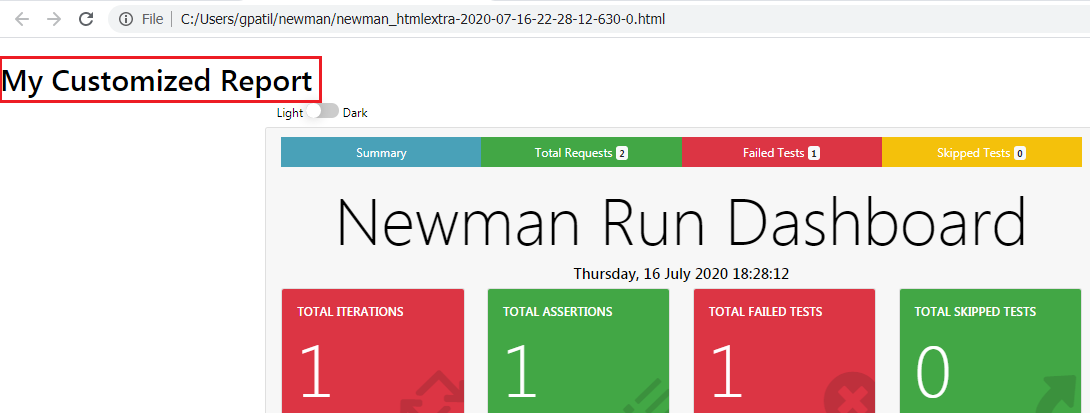
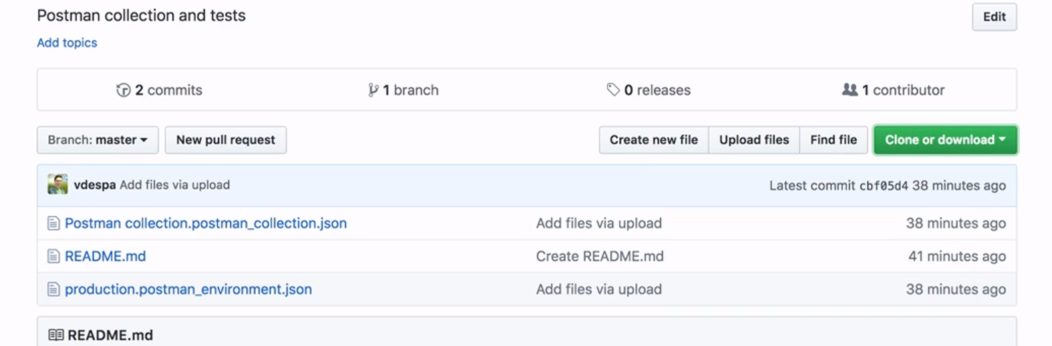
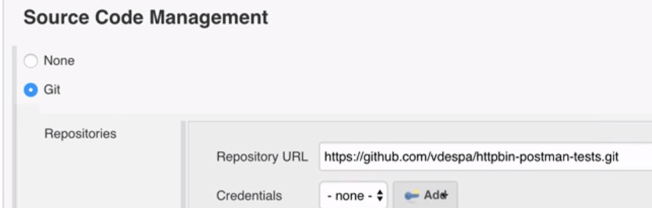
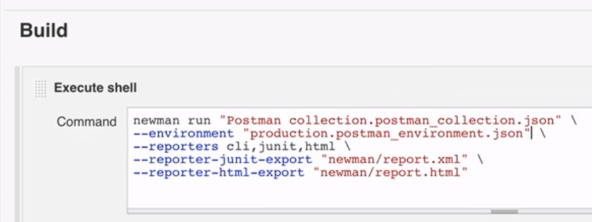
**Newman**

* Newman is a command line collection runner for postman
* Newman is Node.js program;
* NodeJs is a JavaScript runtime environment that can execute JavaScript without opening a browser.
* Newman runs inside the NodeJs runtime environment, so this is why NodeJs needs to be installed (locally on your computer or on the server running Jenkins or any other CI server).
* So first, Check in command prompt if Node.js is installed or not
* Hit C:/> **node -v**
* If you get version, means it’s installed on the machine; if not, then we have to install it.
* We can also check if node.js is installed with
* C:/> **npm –v**
* ‘npm’ is node package manager which is used to install packages over node.
* **NPM is the Node Package Manager**.
* This is like a repository of projects and has knowledge of what requirements each project has. For example, you want to install Newman, but it depends on other libraries which themselves depend on other libraries. To figure out all the dependencies each project needs, you just need to have NPM installed and NPM will do this work for you.
* To download Node.js, goto <https://nodejs.org/en/download/>
* Goto Long Term Solution (LTS) > Windows Installer > download and install it
* Once it’s installed, you can check the version of ‘node’ and ‘npm’
* Now, Install Newman
* In command prompt, run below command
* C:/> **npm install –g newman**
* Once it’s installed, you can see it’s installed files at C:\Users\gpatil\AppData\Roaming\npm\node\_modules
* If you wanted to do fresh install, then delete the files from this location and run the above command again.
* (***Troubleshooting Node.js / npm / Newman Problems (for Windows)***
* *A. Node.js cannot be called from the command prompt*

*Once you have successfully installed Node and restarted your computer, when you open a command prompt terminal and type in node -v, you get an error and not a version like v6.9.5*

* *Possible solution:*
* *In Windows, you need to set node.js folder path into system variables.*
* *1) open Control Panel -> System and Security -> System -> Advanced System Settings -> Environment Variables*
* *2) in "System variables" find variable PATH and add node.js folder path as value. Usually it is C:\Program Files\nodejs;. If variable doesn't exists, create it.*
* *3) Restart your computer.*
* *B. Newman cannot be called from the command prompt*

*Once you have installed newman and restarted your computer, when you open a command prompt terminal and type in newman --version, you get an error and not a version like v4.1.0*

* *Possible solution:*
* *In Windows, you need to set the folder where node.js is storing global dependencies into system variables.*
* *1) open Control Panel -> System and Security -> System -> Advanced System Settings -> Environment Variables*
* *2) in "System variables" find variable PATH and add the folder path as value. Usually it is something like: C:\Users\YOURUSERNAME\AppData\Roaming\npm*
* *Inside the folder you should see a file called newman.*
* *3) Restart your computer.*
* )
* Now, we will run the Postman collection from command prompt.
* For that, we have to export the collection from Postman
* Goto Collection > … > Export
* Save it as .JSON on local drive.
* Now goto command prompt; goto folder where this .json file kept
* And run below command
* C:/>…folder path where .json file kept… > **newman run “Exported filename.json”**
* Collection will be run thru command prompt and result will be shown in command prompt
* 
* Note that if you run below command, you will get all newman commands that you can execute i.e. it’s help command
* C:/> **newman run -h**
* You can run Collection Runner command from Jenkins
* Create FreeStyle project in Jenkins > Give Project Name
* 
* 
* 
* Give the command similarly we gave in above example
* Save and click on ‘Build now’ in Jenkins.
* You can see the output in Console screen
* 
* You can give different newman commands (which we saw earlier)
* **Jenkins Reports (XML)**
* You can see the Reports in more presentational format by doing some configuration in Jenkins
* (You can get more details at <https://github.com/postmanlabs/newman#configuring-reporters>)
* Append below commands inside commands section; following newman command:
* 
* ‘cli’, ‘junit’ are the reporter names; that should be separated with comma without space
* Into the next line, path is mentioned where to save the report; here we are saving into Jenkins Workspace folder itself.
* After that, we have to define Post-Build action to display the report in Jenkins UI
* Goto Post-build action > Publish JUnit test result report
* 
* Give the report path and save
* 
* You can see the result as below; also you can see ‘latest Test Result’
* 
* **Jenkins Reports (HTML)**
* You can generate HTML report too by adding followings into command section
* 
* Select ‘Publish HTML Report’ in POST-Build section
* 
* Give path and report name where we already saved it
* 
* You can see HTML report link and you can go there explore the report
* 
* It will show you basic report. More detailed Report will be seen below section
* **Jenkins Reports (HTML Extra)**
* We can generate better HTML report which show more information and we may customize the report
* For that, we have to install HTML Extra npm package on the local machine as well as on the CI server
* Install it using below command in command line
* C:/> **npm install –g newman-reporter-htmlextra**
* Now, when you run the Collections (for an example, collection Sharelink is https://www.getpostman.com/collections/8305796e7f88c12ee017), you have to run below command
* C:/users/gpatil> **newman run https://www.getpostman.com/collections/8305796e7f88c12ee017 --reporters=cli,htmlextra**
* when you run this command, Collection will be run and HTML report will be saved (at the location where you have run this command. For example, here you will find the report at C:/users/gpatil/newman folder
* HTML report will look like this with much more details:
* 
* Note that, you can find more info In NPM documentation (<https://www.npmjs.com/package/newman-reporter-htmlextra>) on this report.
* **Customizing HTML Report**
* You can create your own template and provide it’s reference into the above newman command to generated the report in required format
* To generate your own template, your required HTML/CSS skills
* Also, you can take existing template and make changes into it i.e. Make a copy of Handlebar templates
* Here, we will grab existing templates from npm documentation page. Goto <https://www.npmjs.com/package/newman-reporter-htmlextra> > Click Github Repository Link (right side section) > inside Github repository, click on ‘lib’ folder (<https://github.com/DannyDainton/newman-reporter-htmlextra/tree/master/lib>)
* You will get there report templates (.hbs)
* Download anyone of it on your local machine and save it as .hbs format
* Edit it with NotePad++ and customize it
* I have added this line into the template. Save it
* 
* Into the newman command provide the template path as below
* C:/users/gpatil> **newman run https://www.getpostman.com/collections/8305796e7f88c12ee017 --reporters=cli,htmlextra --reporter-htmlextra-template C:\Users\gpatil\newman\dashboard-template.hbs**
* After running the collection using this command, HTML report will be generated at the location where you have run this command.
* See the change into the report
* 
* Note that, you can add many handlebars and customize the report (More info is at <https://handlebarsjs.com/> and <https://www.npmjs.com/package/newman-reporter-htmlextra>)
* Now let’s see how to run Collections in Jenkins from GitHub
* Let’s consider that you have created a Repository in GitHub and you have saved there Exported Collection (.JSON) and exported Environment (.JSON)
* 
* Goto Jenkin > Create a Project > Configure > Source Code Management > Select GIT > Enter Repository location/url (you will get it from GitHub)
* 
* Into Build section, write newman commands with the Collection file and Environment file (.json) which are present into Github repository.
* 
* Save the configuration and run the build

[

***Troubleshooting Jenkins Problems (for Windows)***

*If you are running Jenkins locally, it may happen that you encounter some problems. The most common ones are listed below.*

#### *Node and Newman are installed locally but do not work in Jenkins*

*If you are getting a version output for node and newman while opening a terminal but they do not work in Jenkins, it might be related to the fact that Jenkins runs as a different user.*

***Possible solution #1***

*In Windows, you need to set node.js folder path into system variables.*

*1) open Control Panel -> System and Security -> System -> Advanced System Settings -> Environment Variables*

*2) in "System variables" find variable PATH and add node.js folder path as value. Usually it is C:\Program Files\nodejs;. If the variable doesn't exist, create it.*

*For newman you also need to add the folder where the newman bin was installed (usually it is something like: C:\Users\YOURUSERNAME\AppData\Roaming\npm*

*3) Restart your computer.*

***Possible solution #2***

*Please also look into the responses on Stackoverflow as well:*

*https://stackoverflow.com/q/53142260/766177*

***Possible solution #3***

*Contributed by William nkosi to solve the error with "node: command not found"*

[*https://www.techcoil.com/blog/how-i-make-my-jenkins-slave-windows-service-recognize-the-global-modules-installed-by-npm/*](https://www.techcoil.com/blog/how-i-make-my-jenkins-slave-windows-service-recognize-the-global-modules-installed-by-npm/)

#### *Encoding Problems ("â”œâ”€â”€ ")*

*If you see a bunch of "â”œâ”€â”€ " chars in the Jenkins console output, try one of the following solutions:*

***Possible solution #1***

*Try the changes as shown here:*[*https://www.linkedin.com/pulse/how-resolve-utf-8-encoding-issue-jenkins-ajuram-salim/*](https://www.linkedin.com/pulse/how-resolve-utf-8-encoding-issue-jenkins-ajuram-salim/)*. After this restart Jenkins and it should work better (contributed by Phil. Many thanks).*

***Possible solution #2***

*Have a look at this medium post:*

[*https://medium.com/pacroy/how-to-fix-jenkins-console-log-encoding-issue-on-windows-a1f4b26e0db4*](https://medium.com/pacroy/how-to-fix-jenkins-console-log-encoding-issue-on-windows-a1f4b26e0db4)*(contributed by Javier Gironella San Juan. Many thanks).*

***Possible solution #3***

*Try adding ONE of the following newman options to the newman run command:*

*--disable-unicode*

*--color off*

#### *Error: java.io.IOException: Cannot run program "sh"*

*This happens if you have specified your Windows command as "Execute shell" rather than "Execute Windows batch command".*

*]*

**Collection Workflow:**

* Let’s say you have 4 requests in a Collection [Request-1, Request-2, Request-3, Request-4];
* When you run a collection, these requests will be executed serially as you created or arranged order into the collection.
* But if you wanted to control that flow and wanted to execute in order of Request-1, Request-3, Request-2; how to do that?
* For that, we have to use below statement into the ‘Test’ section of a request.

**postman.setNextRequest(“NextRequestName”);**

* i.e. into Test section of Request-1, you have to add

**postman.setNextRequest(“Request-3”);**

* into Test section of Request-3, you have to add

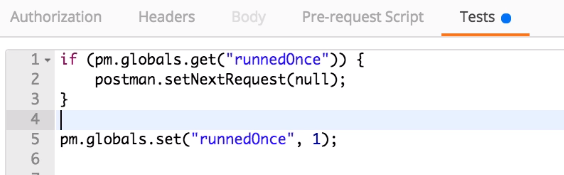
**postman.setNextRequest(“Request-2”);**

* In this case, Execution flow will be like this

Request-1 > Request-3 > **Request-2 > Request-3 > Request-2 > Request-3 > … stuck in loop**

* Here, after request-2, it will be executed in sequence i.e. it will pick up request-3 and because of statement we added in 3, it will pickup request-2 and it will goes and goes (infinite loop) since we did not add any such condition in Request-2.
* And If you wanted to execute in order of Request-1, Request-3, Request-2 and then Stop; don’t execute Request-4; how to do that?
* into Test section of Request-2, you have to add

**postman.setNextRequest(null);**

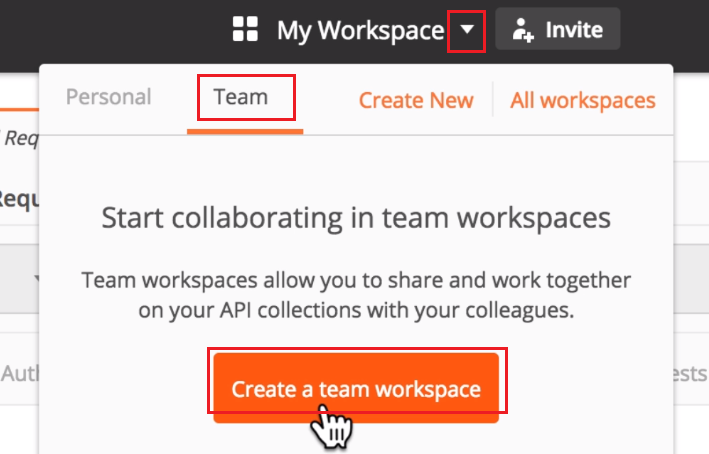
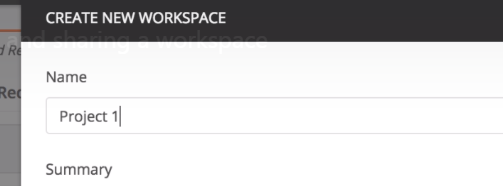
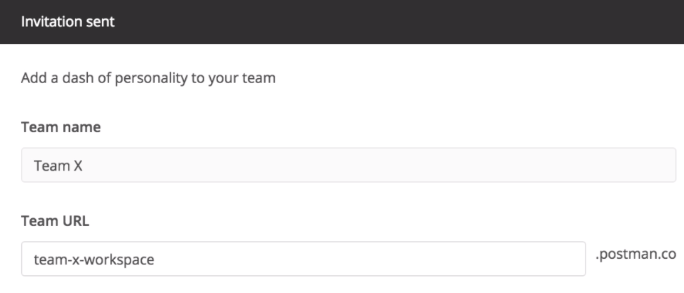
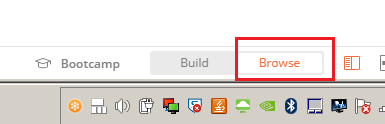
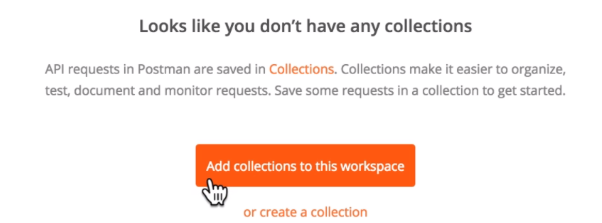
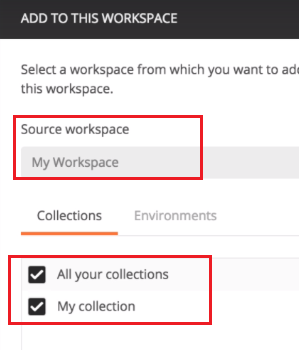
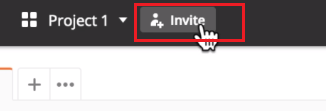
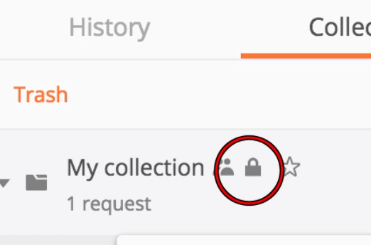
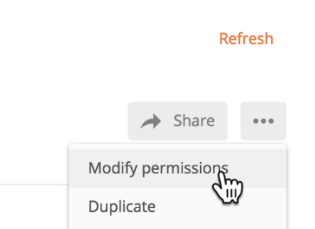
* Another way to avoid the above looping problem, you can define a Global variable
* Set the global variable value when the request gets executed first time and when it reached second time, check the variable value
* 
* Here, “runnedOnce” is a global variable with 0 value set initially
* When the requests gets executed first time, it will check it’s value and set the value to 1
* On next time run on the request (in same execution), it will check if the value is 1 (true), then it will execute **postman.setNextRequest(null);** and flow will be stopped here.
* Note that, here **postman.setNextRequest()** statement gets execute at the end; no matter where you are writing this line i.e. all statements (in Test section) will get executed in the request and then this setNextRequest() statement will get executed.

i.e. If you write this code into Request-2 then execution flow will be

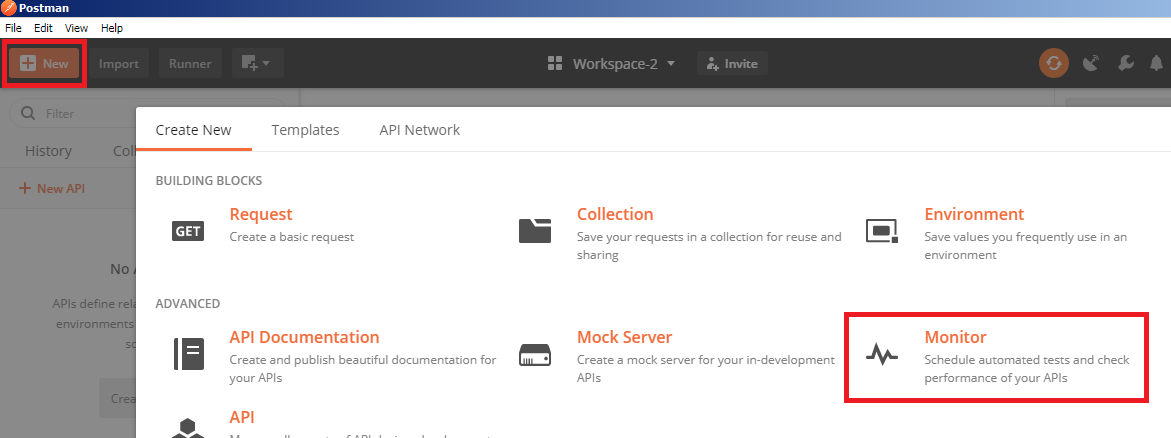
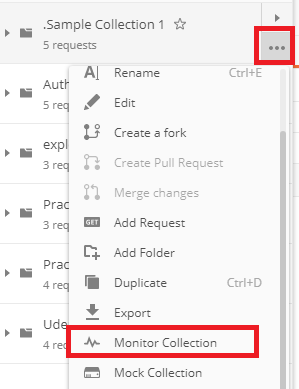
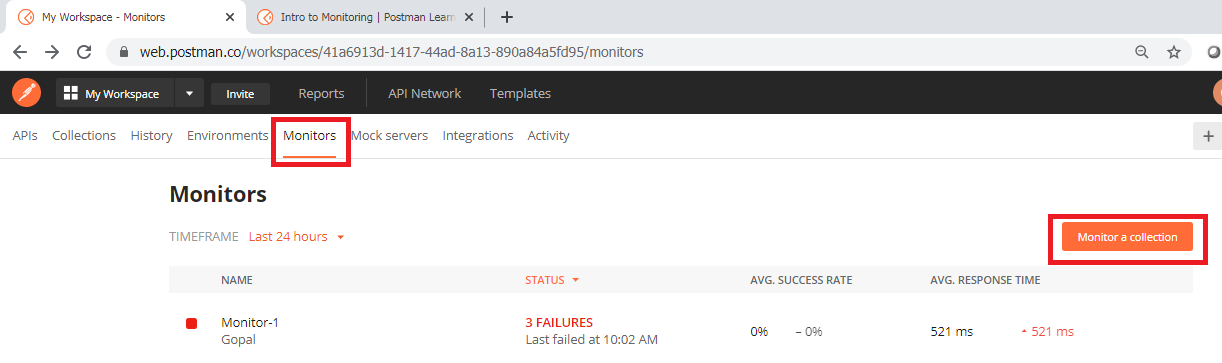
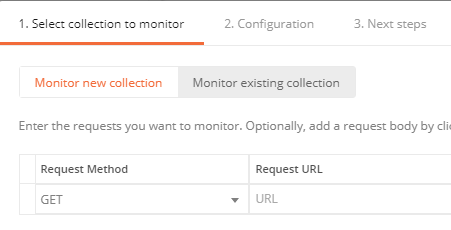
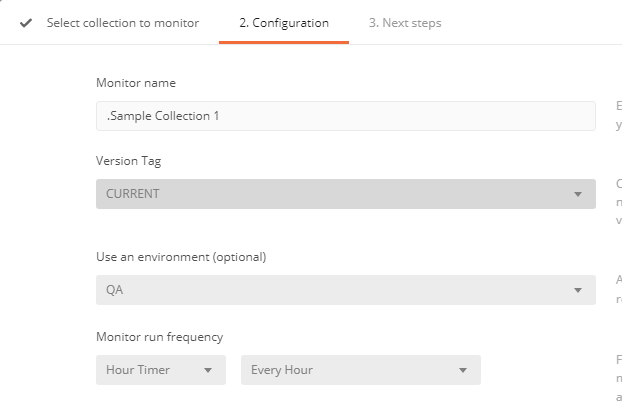
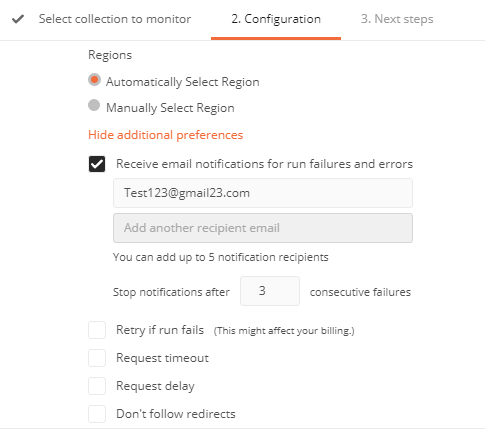
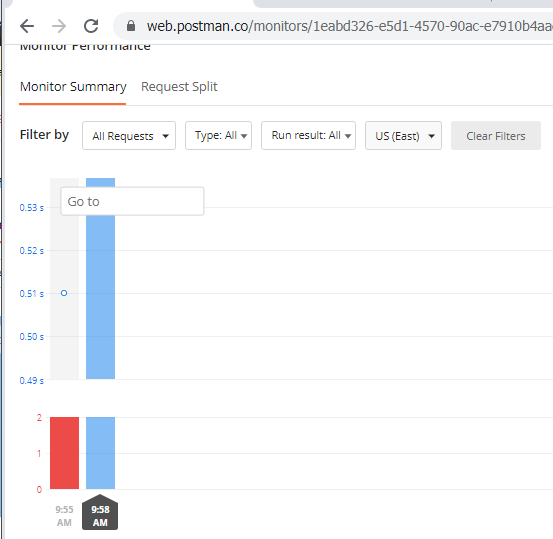
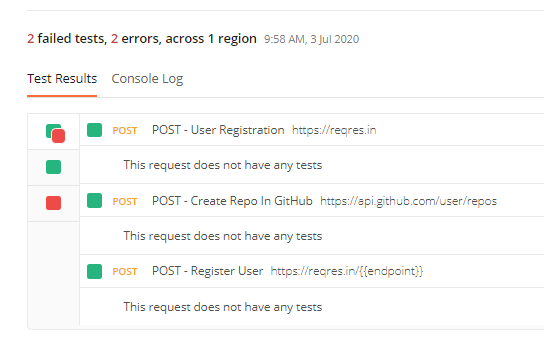
Request-1 > Request-3 > **Request-2 > Request-3 > Request-2 .**

* Request-3 and Request-2 will be executed twice
* NOTE: eventhough you write this **postman.setNextRequest(null)** statement in Pre-requsite section, still that request gets executed
* There is an advanced way to handle multiple requests in different order. For more info, check this <https://www.youtube.com/watch?v=FWYKOR0Zj28>
* console.log(**pm.info.requestName**); 🡺 gets current RequestName & print in console
* console.log(**pm.info.requestId**); 🡺 gets current Request ID & print in console

**Workspaces**

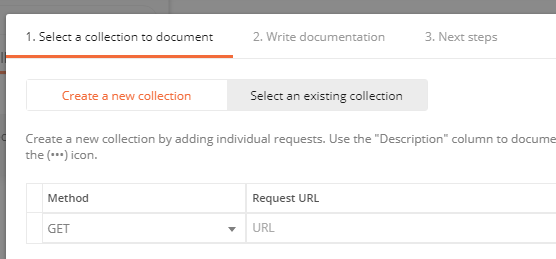
* Workspace is an area where we can group, organize and manage API Collections in Postman
* Workspaces are available in Postman 6.0 versions and above
* When you wanted to share your work with other team members you can use Workspace; so that they can view and/or edit your requests.
* Let’s consider User-1 and User-2; user-1 wanted to share his requests with user-2
* User-1:
* Goto Workspace > Team > Create new Team Workspace
* 
* Create Workspace Name
* 
* Enter Team Name
* 
* Now, you could be in Team workspace in Postman UI
* You can create new Collection/request or can import existing Collection into the Workspace
* You also can pull the request from your personal workspace into the Team workspace
* To pull the request from personal workspace; you can click on Browse button (bottom right of the postman)
* 
* Then click on ‘Add Collection to this workspace’ button
* 
* Select your personal workspace, select the collection and Add it into Team workspace
* 
* Click on ‘Build’ button to switch back to Collection/Request view. (Build button is near to Browse button should in above screenshot)
* Now, to add other users to this team workspace, Click on ‘Invite’ button
* 
* Add User’s email Id in next screen and invite them
* They will receive invitation emails
* After accepting invitation, they can see this workspace and collections in their postman
* By default, other users may have ‘View’ only permissions to that collection
* User-2 can see lock sign next to collection name
* 
* User-1 can give the permission by navigating again Browse button (bottom right of Postman)
* For the collection, click on … button > Modify permission (Can View / Can Edit)
* 
* User-1 can set Team’s permission as well as individual permission there
* **NOTE that:**
  + Users can be part of one team only; if user-1 wanted to be part of another team, he has to leave previous team
  + Team members can see all Workspaces and All collections
  + Permissions are assigned to Collections

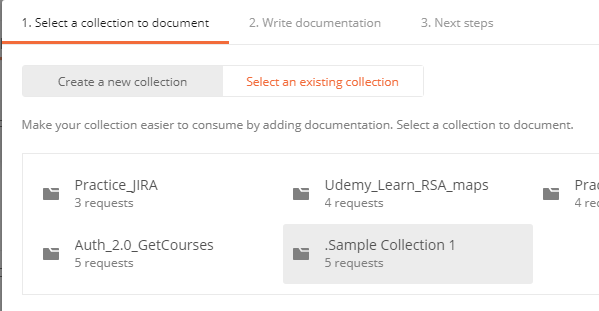
**Monitors**

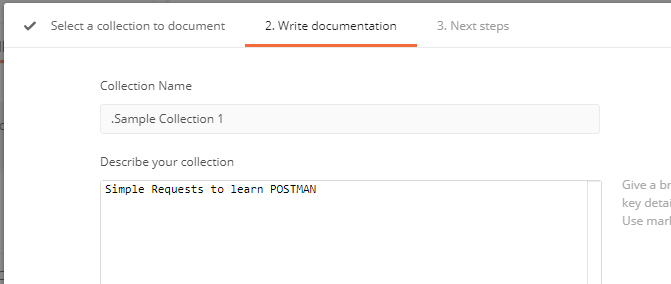
* Monitors help to run collections periodically to check the performance and response of APIs
* There are different ways to create Monitors
  + Thru Create New > Monitors
  + 
  + Thru Collections > … > Monitor Collection
  + 
  + Thru Postman Web > Monitors > Monitor a Collection
  + 
* When you create monitor from ‘Create New’ option, it will ask you to create Monitor new Collections or for existing collections
* 
* While creating Monitor it asks for Monitor Name, Environment, Monitor Run Frequency (minute, hour, week), Region, some additional options
* 
* 
* In additional option, you can give max 5 email to receive email notifications in cases of failures and errors
* Once monitors are configured, that monitors will be displayed in Postman Web
* And you can see there run monitor summary, test result, console logs and other details
* You can **Edit, Pause, Delete Monitors** from web site too; you can do same from Postman tool too.
* 
* 

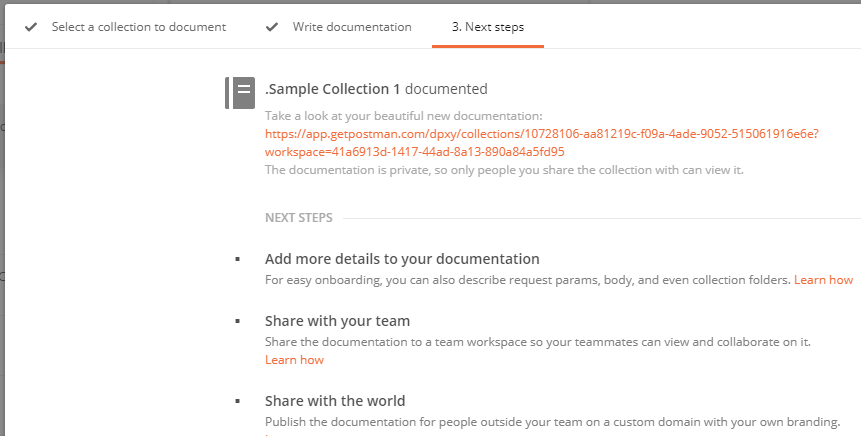
**API Documentations**

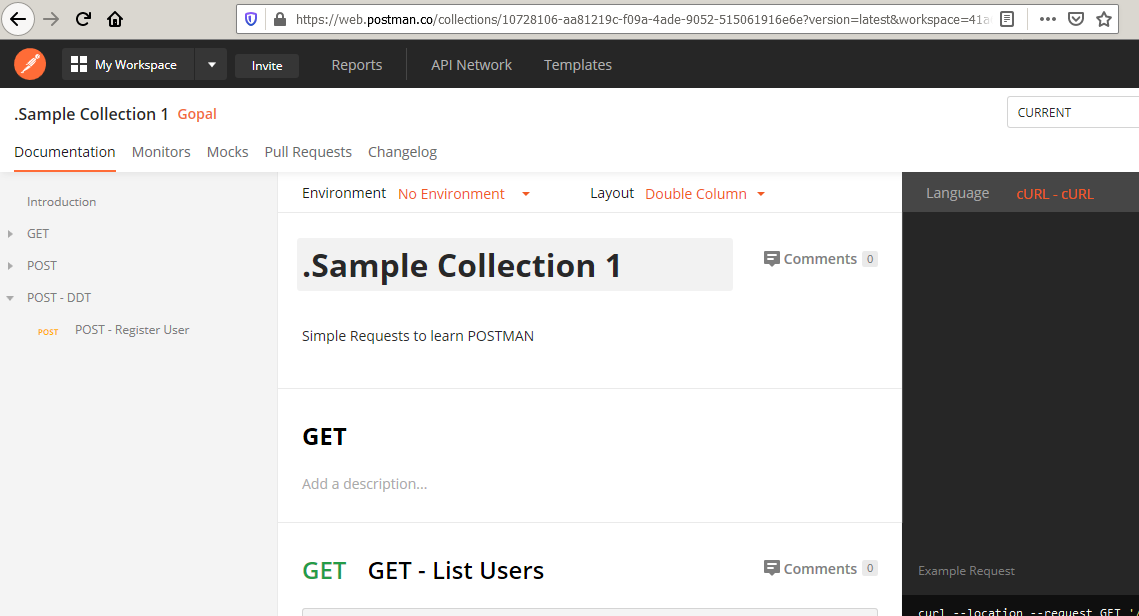
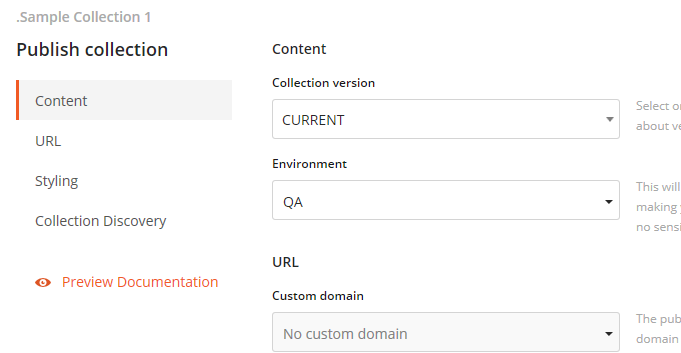
* It’s a Postman feature that let you share API documentations in a beautifully formatted web pages
* When you have created requests, collections in Postman and if you have to format and share it then we can use this feature
* You can create it from Create New > API Documentation
* It will ask if you wanted to create with new Collection or for existing colletions
* 



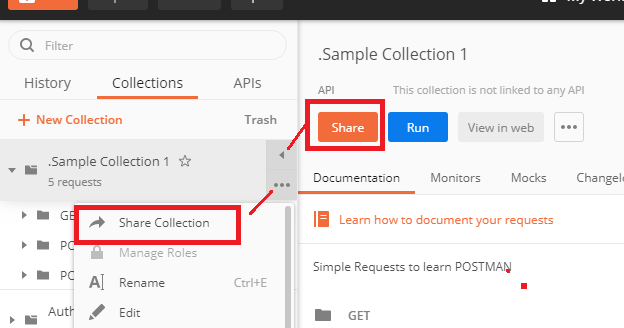
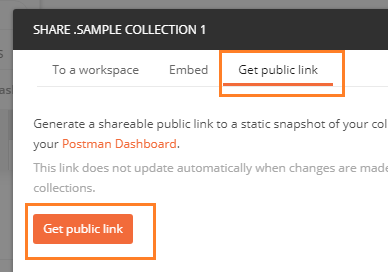
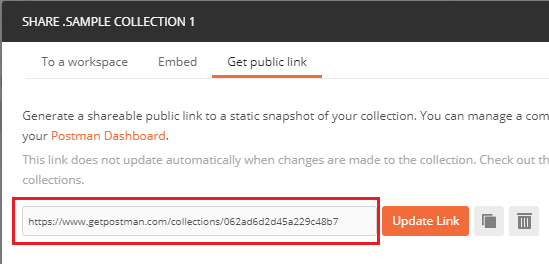




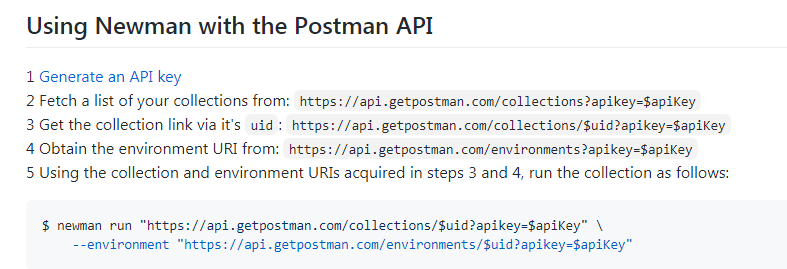
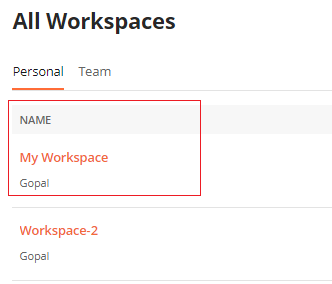
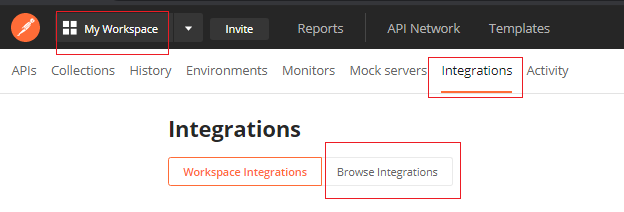
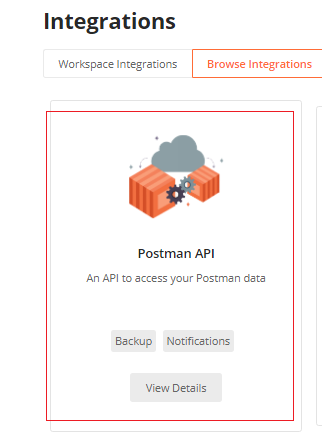
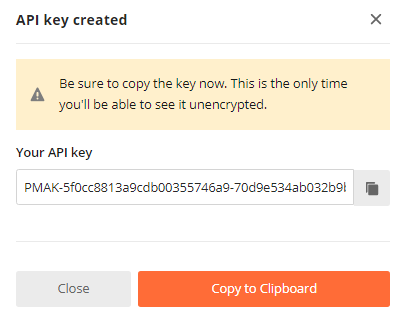
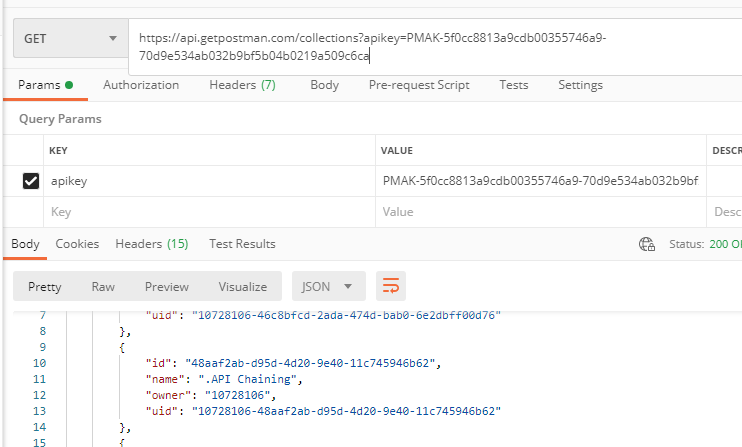
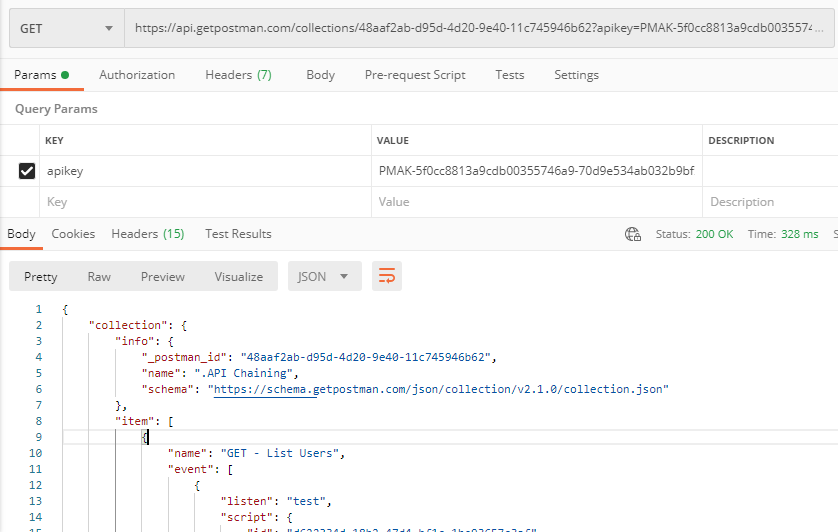


* When you click on the documentation link which you can see in above screenshot, it will open in new browser window.
* Well-structured Documentation will be displayed in html format
* Here, your ‘.Sample Collection-1’ documentation is created
* 
* It’s private documentation
* You can share the link someone so that they can see the Collection details
* Also, you can publish it by clicking on ‘Publish’ button on the web page where documentation is open
* 
* 
* While publishing the collection documents, it will ask to select Environment
* When Environment gets selected, corresponding Variable values will be replaced the used variables in Requests/scripts/test and documents gets published with the corresponding values.
* When you click on Publish button, collection will be published
* Now you can share the Web URL to public to see your documentations.
* You can also publish the document from Collections > … > Publish Docs
* It will again go to web page for that particular collection. You can publish there.
* If the documents are published for same collection, then it will show ‘Update’ option instead of ‘Create/publish’ option
* You can also publish the document from Postman Web > Collections > select Collection > Publish.
* So, there are three ways to create and publish the documents
  + Create New > API Documentation
  + Collections > … > Publish Docs
  + Postman Website > Collections > publish
* You can **edit, Unpublished** the documents from Postman Website > published Collection
* Note that, before publish the document, if there are any confidential data like password or security key, You must remove them (as a best practice)

**Run Collection Remotely**

* To run the collection remotely, we must need collection URL
* There are two ways to get the collection URL
  + Postman > Collections > Share
  + Postman > Collections > … > Share Collection
* 
* 
* When you click on ‘Get Link’, you will get and copy an URL
* Note that, you have to get new link/update link if you change the code
* 
* Now you can goto Command prompt and hit below command with the copied url.
* C:/> **newman run** [**https://www.getpostman.com/collections/062ad6d2d45a229c48b7**](https://www.getpostman.com/collections/062ad6d2d45a229c48b7)
* You can hit same command form any Remote machine; your collection will be executed at remote machine.
* Note that, Node, npm and newman must be installed on that machine (as we installed above).

**Run collection thru Postman API:**

* Postman API: Its way to interact with your collection using an API
* For that you have to do (<https://github.com/postmanlabs/newman#using-newman-with-the-postman-api>)
  + Generate an API Key for your account
  + Fetch Collections list from your account
  + Obtain the environment URI
  + Using the collection and the environment URI, run the collection in command line
* 
* Let’s see how to do that
* Login to your Web Postman account
* Goto the Workspace where your Collection is in
* 
* Navigate to Integrations tab> Browse Integration
* 
* Click on ‘View Details’ for “Postman API” option
* 
* Click on ‘Generate API’ key
* 
* Give the API Key Name and generate it
* 
* Create new Request (GET) in postman and enter https://api.getpostman.com/collections?apikey=$apikey
* And replace the copied API key at ‘$apikey’ in the URL.
* Hit the request, you will get all the collections created in your postman account.
* 
* From the Response, you will get the ID of the collection (which you wanted to run)
* Create another request Or edit the existing request itself https://api.getpostman.com/collections/$uid?apikey=$apiKey
* And Replace ‘$uid’ with the Collection ID.
* Hit the request, you will get all requests details into that collection
* 
* Copy the URL of the request
* Goto command line and hit below command with the copied URL in “” (Note, If it’s not working, try without “”)
* C:/> **newman run “**https://api.getpostman.com/collections/48aaf2ab-d95d-4d20-9e40-11c745946b62?apikey=PMAK-5f0cc8813a9cdb00355746a9-70d9e534ab032b9bf5b04b0219a509c6ca**”**
* By this way, you don’t have to save and download or upload Collections file (exported .JSON) and then run the collection; Or Generate/update collection key manually on every change.
* Using this method, You can directly run the collection as soon as you modified in postman. This is useful when you have to try something and check quickly.

**Run collection thru Postman API for specific Environment:**

* This is the extension of above topic.
* In command line, you can give Environment details in two ways
  + Download/Export Environment from Postman (just like Collection Export in .json file) and hit the command

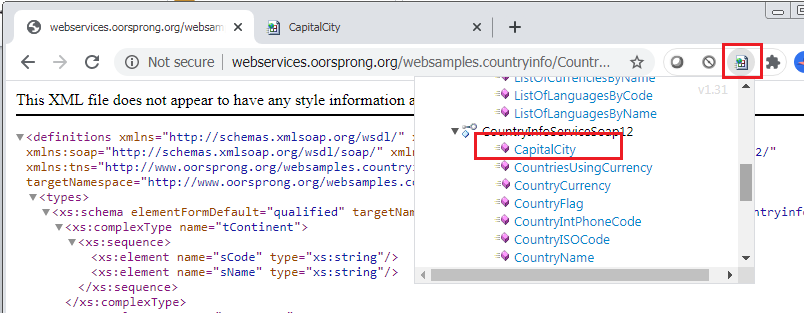
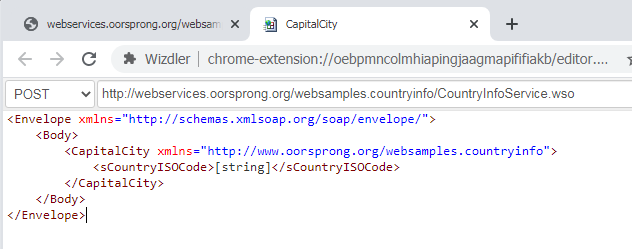
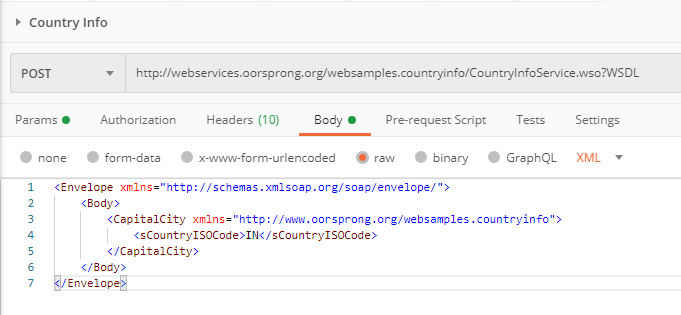
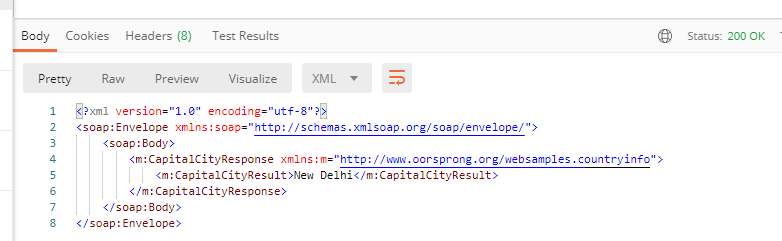
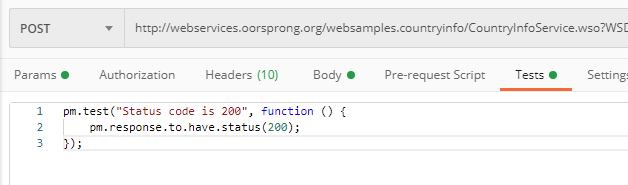
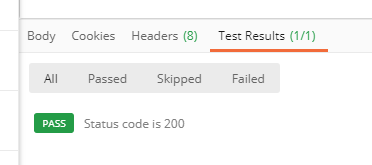
C:/> **newman run “**https://api.getpostman.com/collections/48aaf2ab-d95d-4d20-9e40-11c745946b62?apikey=PMAK-5f0cc8813a9cdb00355746a9-70d9e534ab032b9bf5b04b0219a509c6ca**” - -environment “C:\myFolder\Env\_QA.json”**

* + Using Postman API for Environment (just like we use above Postman API for collection)
* Note Other newman command line options will get here (<https://github.com/postmanlabs/newman#command-line-options>)

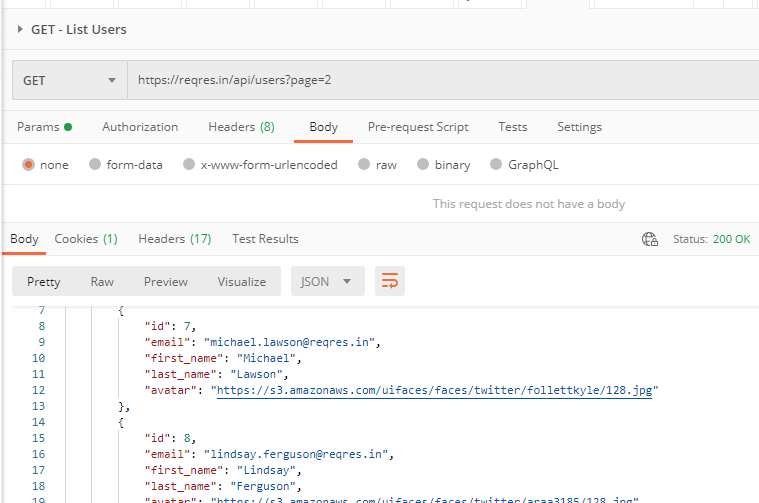
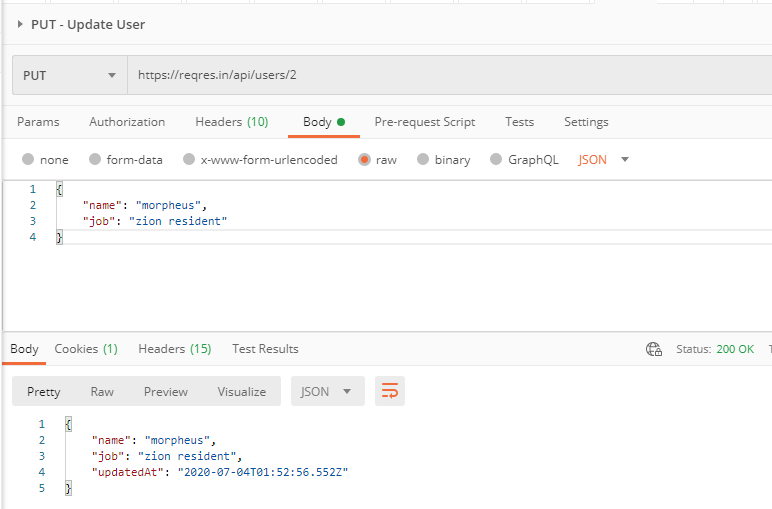
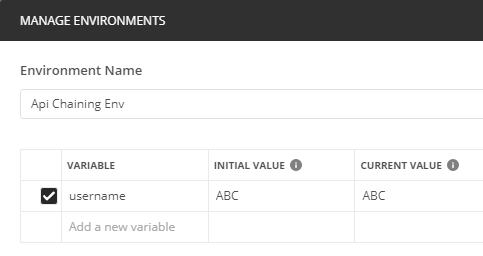
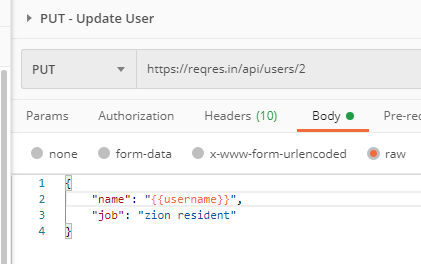
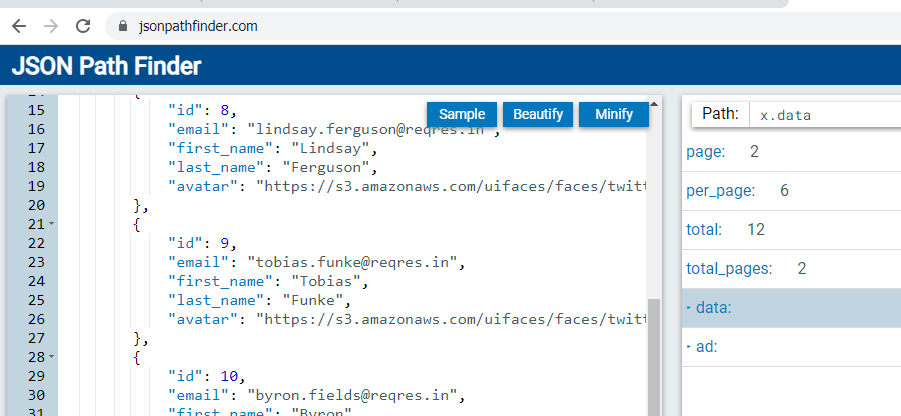
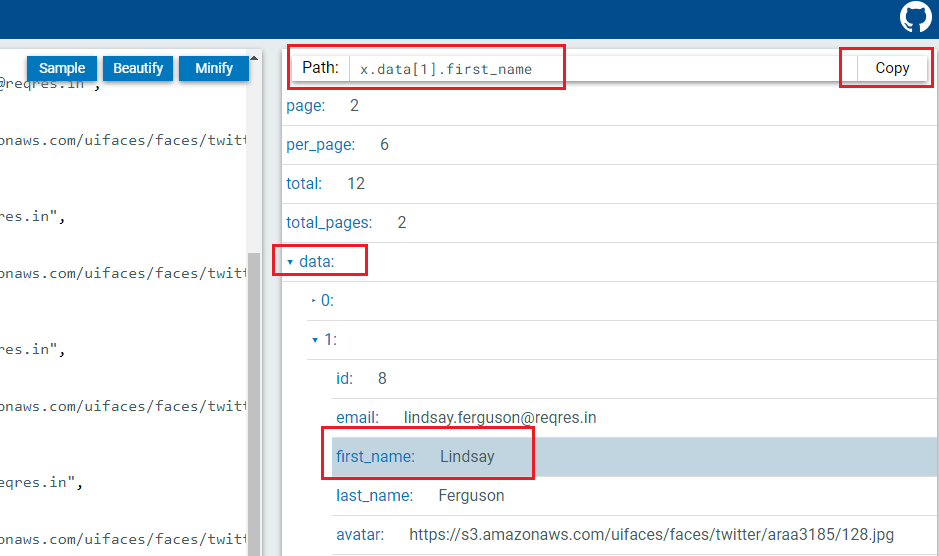
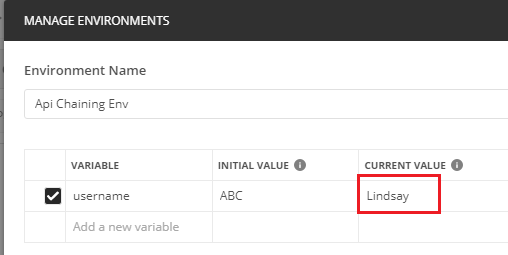
**SOAP Request in Postman**

* WSDL is Web Services Description Language which is used in SOAP
* SOAP Request file is xml based and defined in WSDL
* First , get the SOAP Request URL or wsdl URL
* For example, I am taking here <http://webservices.oorsprong.org/websamples.countryinfo/CountryInfoService.wso?WSDL>

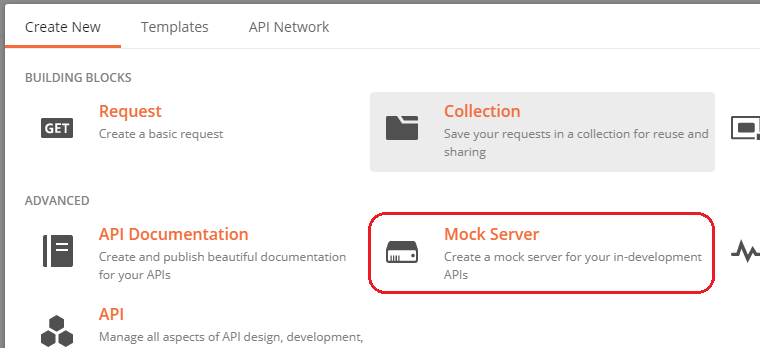
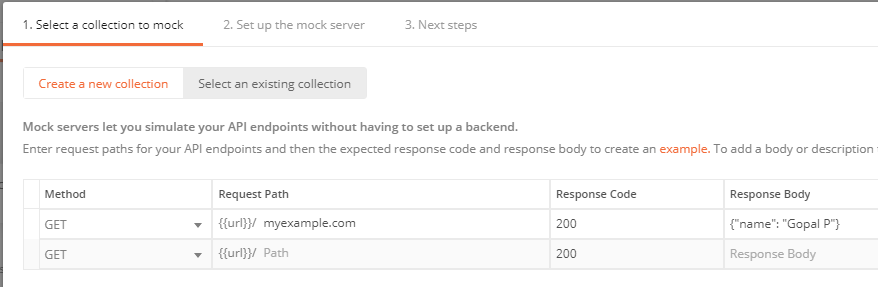
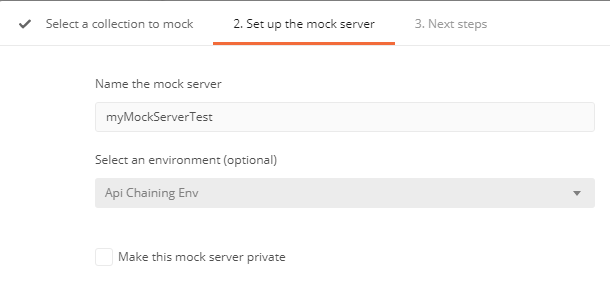
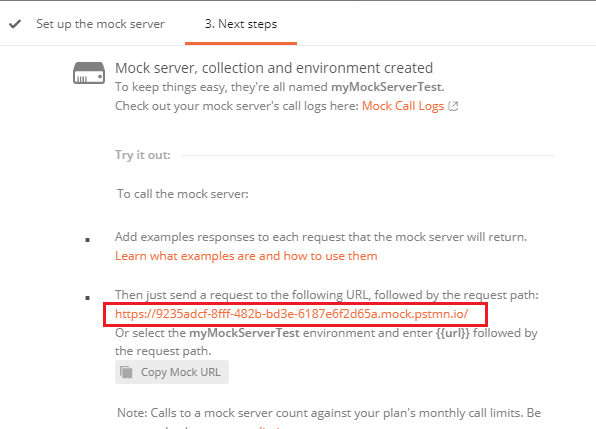
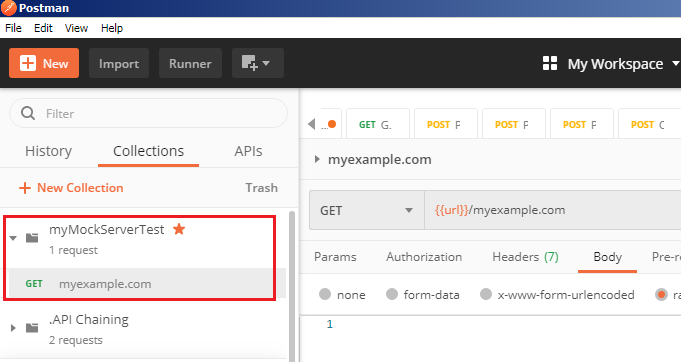
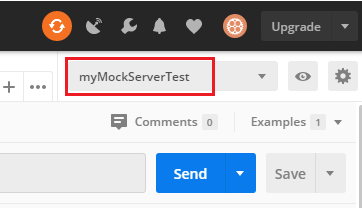
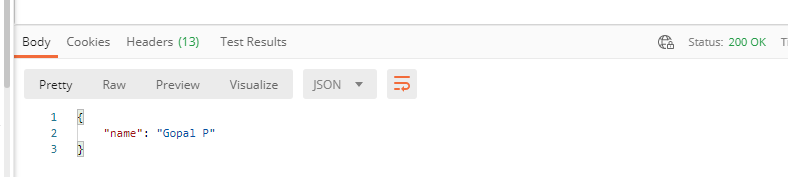
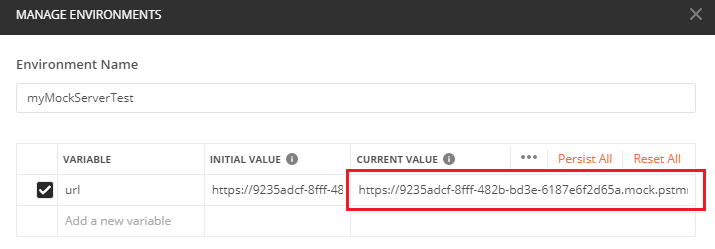
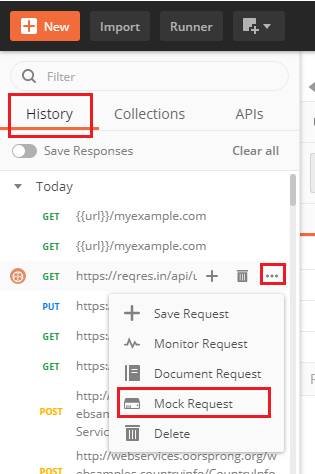
which returns country information

* Enter this URL in POSTMAN and select the method as POST (SOAP used POST method only)
* Goto ‘Body’ section, select “Raw” and Type as “XML” (SOAP used XML format only)
* Provide request data in ‘body’ section
* If you have WSDL url then we need a client or app which generate a SOAP request
* Also we can use plugins; here I used ‘Wizdler’ chrome plugin
* Open the above wsdl url in chrome
* Click on ‘wizdler’ plugin icon. You can see different requests in WSDL
* I clicked on ‘CapitalCity’; you will get the Request Template
* this request returns capital of a country
* 
* 
* Copy this request template and paste it into Postman body section
* Provide the Country Code; here I provided ‘IN’ for India
* 
* Into ‘Header’ section, Add **‘Content-Type’ as ‘text/xml’**
* Send the request. You will get response as below
* 
* Like API requests, here also we can add Test script to validate something; here validating if status code is 200.
* 
* You can see corresponding Test Result
* 
* We can run the Collection/request from command prompt too; same as we saw above
* i.e. Export the collection and save it in .json format and run the Collection Runner (newman) command
* and also, Click on Collection Share button; get the share link and run it from command prompt with the link.

**API Chaining**

* API Chaining is nothing but get the data from a response of one API and refer it in another API.
* For API chaining example, we have to follow below steps
  + Add a Request-1 in Postman
  + Add a Request-2 in Postman
  + User Environment variable in Request-2 to parameterize the value
  + Add a script to fetch a value from response of Request-1
  + Into same script, Set the fetched value to Environment variable (used in request-2)
  + Run and validate the collection (or both requests)
* Here, fetched value from response-1 will be used in request-2
* In below example, we are using ‘List Users’ and ‘Update’ sample API requests from <https://reqres.in/>
* Request-1 (List Users):
* 
* Request-2 (Update user):
* 
* Create Environment variable in request-2 for “name” parameter:
* 
* Select the corresponding Environment for Request-2 and set the variable name instead of name value:
* 
* Now add the TEST script in Request-1, which will fetch the user name value from it’s response and set that value to Environment variable
* To fetch a value from exact location of a response, we need to find json path of that parameter; how?
* Goto <https://jsonpathfinder.com/>
* Into left section, enter the response of request-1, you will get structured data into right side:
* 
* Expand the data, and select the required data
* 
* Here we selected ‘first\_name: Lindsay’
* You will get it’s path at top side; here it is “x.data[1].first\_name”
* Copy it and use it in the Request-1 Script as below:
* 
* In the above script, we parse the responseBody and saved into a ‘res\_body\_data’ variable
* Fetched the ‘data[1].first\_name’ value from the responsebody variable and saved into ‘first\_name\_value’ variable
* And print the value in console.
* And set the value to a ‘username’ environment variable.
* Line#5 in above code can be get from the Test Snippet (Set an Environment variable)
* When you run this request; you can see that Environment variable value is changed according to above script:
* 
* And when you run Request-2, this ‘Lindsay’ value will be used in it.

**Creating Mock API Server**

* A Mock API server imitates a real API server by providing realistic response to a request
* In another words, a mock API server simulates the server response (like real api response)
* When we need this?
  + When you need some APIs to complete your scenario and they are not yet developed
  + Also, when you need 3rd party API and you don’t have access to it
  + To test your own API
* Some examples when to use Mock Servers:
  + When you are developing an front end app (UI) which talks with API server behind the scene, but the API server/APIs are not ready yet; in that case you build up mock API server to complete and check UI app
  + For testing, lets say an front end app is communicating to an internal API server and that server communicate further to external Payment gateway server; But for testing purpose and to check different responses/errors you can play with external payment gateway; in this case, you create your own mock server and can test multiple scenarios how’s your app behave with different errors/responses from mocked APIs.
* We can create it from Postman Create New > Mock Server
* 
* It will ask you to create either with New Collection or with existing collection
* Let’s create with new Collection
* 
* Here, I am creating with method ‘GET’, given request path as “myexample.com” and given response body as {“name”: “Gopal P”}
* Request path and response body are the dummy one
* Next, Give Mock Server name, select environment if you want
* 
* You will get Mock server URL on next screen
* 
* You can see the Collection will be created along with the Environment
* Set the Appropriate Environment which is created here
* And Just send the request, you can the response which you have set it
* 
* 
* 
* If you go to the created Environment, you can see the {{url}} value is created and set (it’s api server)
* 
* Similarly, we can create Mock server from API in History
* 
* You have to follow same steps; that give mocker server name and create and run.

**Troubleshooting Postman requests:**

* You can see more technical logs in Postman Console. Open it before sending the request.
* There are few simple checking on the errors:
  + Make sure if there is no type in your URL/parameters etc.
  + Check with client if the service (API) you are calling is available, up and running
  + Type the domain or IP and port of the API you are calling in your browser; if does not work then it might be some network related problem at your side
  + Make sure you are using right protocol (HTTP / HTTPS)
  + Sometime calling API might be using Self signed certificate; in that case try to disable SSL certificate setting (File > Settings > General > SSL Certificate Verification) in Postman and try it. If it works then Add Client Certificate (File > Settings > Certificate > Add Client Certificate) and turn ON above mentioned SSL Certificate Verification

**API Testing in Postman:**

In above sections, we saw high level information around testing and different features. Here we will see little more details

* You know we write the Test script in ‘Test’ section
* Test script gets executed once you received a response of your request
* You can see test result in ‘Test Result’ section
* You can get some common test scripts from ‘Snippet’
* Now understand basic statement of test (‘Status Code: Code is 200’)

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

**pm.response.to.have.status(200);**

**});**

* Every test starts with **pm.test()** method. This is the method/function for writing test specifications
* This methods works as non-blocking way in case of error; meaning if you put something inside this method and it causes an error then it will not block other tests.
* Into pm.test(), we can add multiple assertions
* First parameter in the method is **“Status code is 200”**; it’s a name of test. You can give it anything that you want.
* Second parameter is **function(){ }** ; it’s so call callback function which is called when the underlying execution (in this case assertions) has finished
* And inside the callback function, there is an assertion i.e. **pm.response.to.have.status(200);**
* **Pm.response** is the response assertion API and can make assertions on the response object (status code, headers, body)
* Let’s see another snippet (‘Response body: JSON value check’)

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

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

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

**});**

* This will get the values from Response json and compare with expected
* We can modify it as you want (see below) and add more assertions into it

**pm.test("Verify Book NAME value in response", function () {**

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

**pm.expect(jsonData.name).to.eql(“Testing Fundamental”);**

**pm.expect(jsonData.price).to.eql(200.99);**

**pm.expect(jsonData.authors.author1).to.eql(“John k”);**

**});**

* ‘name’ and ‘price’ are the parameters/properties in Response body.
* If there is nested json, we have to use property names as ‘jsonData.authors.author1’;

e.g. response body is

{

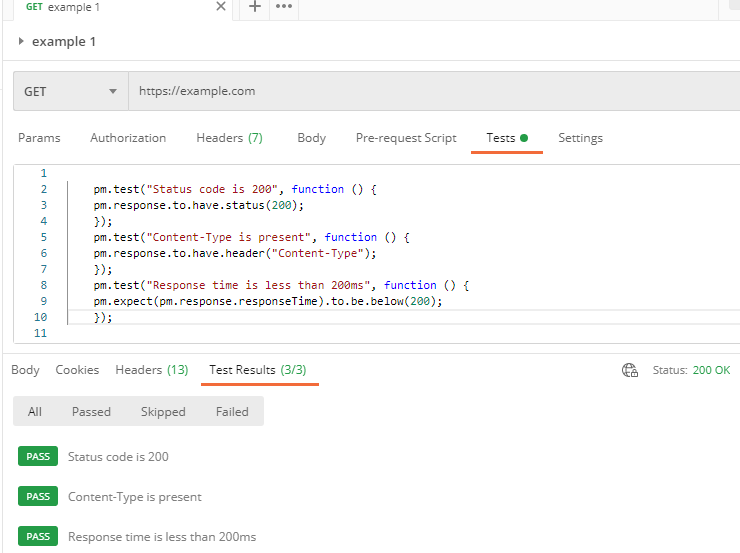
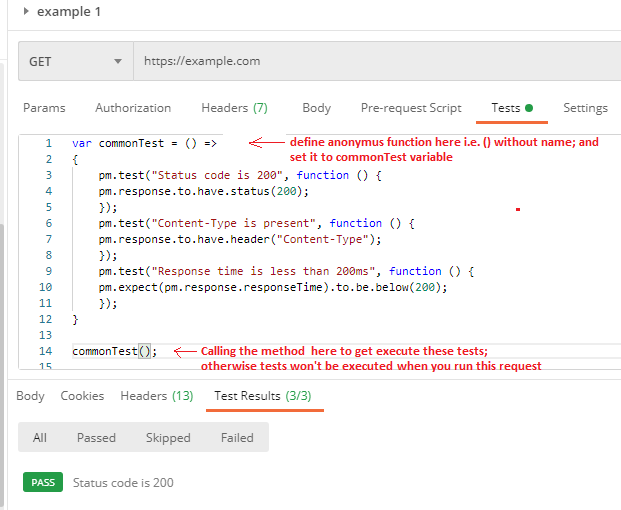
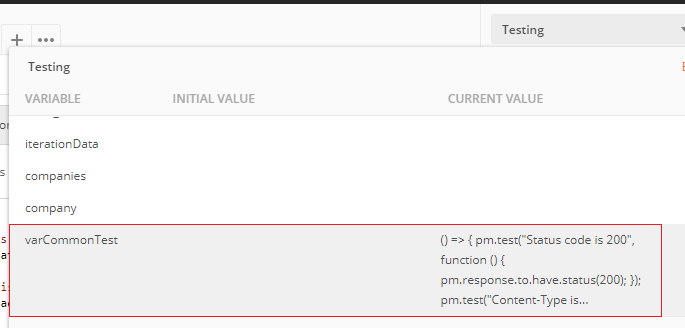
“name”: “Testing Fundamental”,

“Price”:200.99,

“authors”: {“author1”:”John k”, “author2”:”Pavan T” }

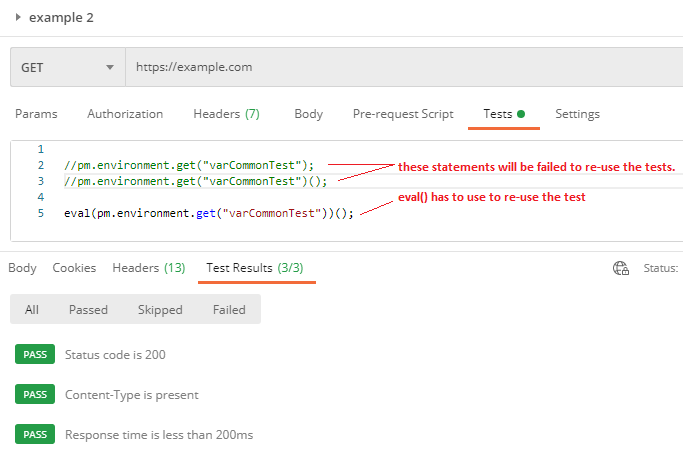
}

**Re-using Tests in Postman:**

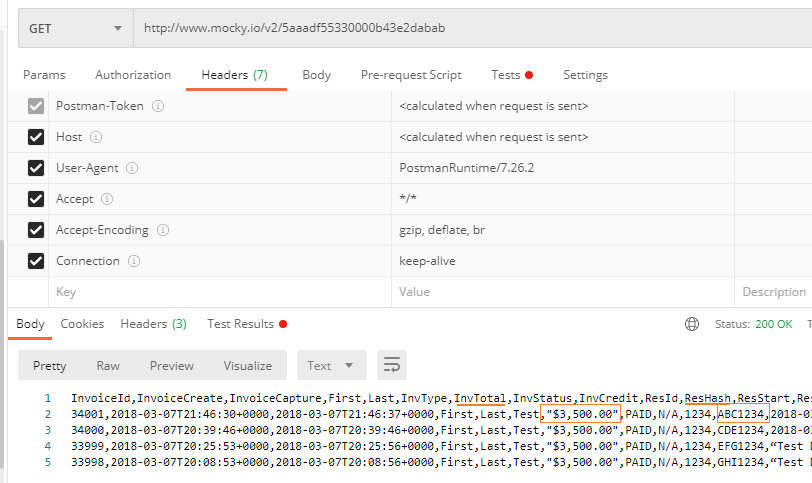
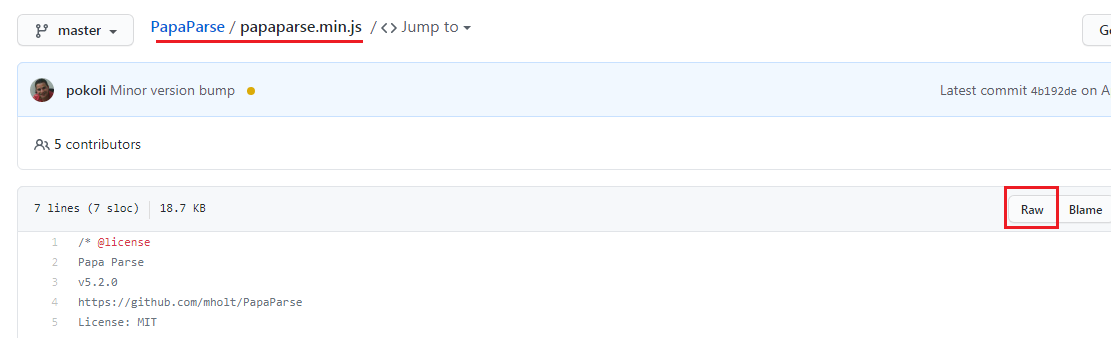
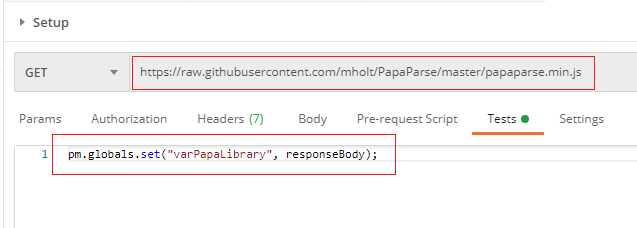
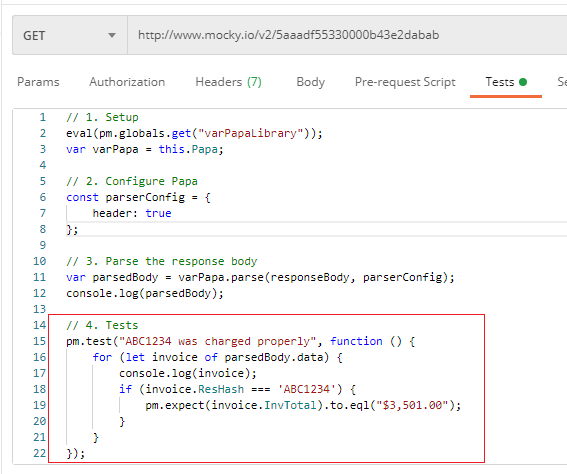
* To understand this, we will create a simple request with some Test codes first.
* Below we have created a ‘example 1’ request which included three different tests
* When you run this it will receive a response and generate Test Result
* 
* Now, we wanted to use these three tests into another request (without writing them again)
* For that, we have to write these test inside a function in ‘example 1’ request as below
* 
* Next step is that, set an Environmental variable and assign this method value to it
* 
* Here, we set ‘varCommonTest’ variable as environment variable with ‘commonTest()’ assigned to it.
* When you run this request once, Environment variable will be set with the statements inside the code
* 
* Now, we have to call these tests in ‘example 2’ request.
* For an example, I just created another request with same URL under same Collection; and getting the environment variable which we created and set using ‘example 1’ request.
* NOTE that; here in Re-using test just getting environment variable in traditional way will NOT work i.e. following statements will not work:

pm.environment.get("varCommonTest");

pm.environment.get("varCommonTest")();

* We have to use **eval()** method for it as below.
* 

**Parse CSV response:**

* Let’s say you received a long response as below and you have to verify that InvoiceTotal is $3,500 for ResHash ‘ABC1234’.
* 
* For that you have to parse the response
* Postman does not have feature to parse the information; we have to use 3rd party javascript library, **‘Papa Parse’** (<https://www.papaparse.com/>).
* We have to integrate this library into Postman
* Goto their webpage, click on ‘GitHub’ link on the home page; you will be navigated to GitHub location
* Into GitHub location, you can find **‘papaparse.min.js’** file
* Click on the file there > Click on the ‘Raw’
* 
* It will be opened into browser; Copy the URL from the browser
* I got it “<https://raw.githubusercontent.com/mholt/PapaParse/master/papaparse.min.js>”
* Create another request in Postman to setup the library
* And provide the above URL (for papaparse.min.js) into the Request
* Set the environment variable which will store it’s response body
* 
* Run this request. Response body will be saved into ‘varPapaLibrary’ variable
* Now, write below code into the original request in which you have to parse the response and perform a test
* To setup and configure the papa library, write below code:
* 
* “Header: true” is set up here since we have to parse Headers from the Response body of original request (where we have to perform test)
* We are parsing the response body here and saved it into ‘parseBody’ variable in JSON format.
* Now write test code as below:
* 
* In this code, test will be failed since expected value is given $3,501 and actual is $3,500.
* Into Test, we are iterating all invoices one by one from the response body and if ‘ResHash’ is matched then performing validation
* At last, you can add below statement to clean up/clear the global variable (if you want)

**pm.globals.unset("varPapaLibrary");**

How to validate JSON schema:

* <https://www.youtube.com/watch?v=haDQBmQii2g>
* <https://www.youtube.com/watch?v=P_So0vpNJCQ>

**Passing variables dynamically to the request at runtime (Newman)**

* **Problem:** Suppose, the URL to the REST service under the test is different every time you get a new environment for testing (cloud environment, dynamically allocated resources).

How can **machine\_name** and **port\_number** can be passed dynamically to the Postman tests when using Newman?

* **Possible solution:** You can set global variables using Newman from the CLI.

**C:/> newman run my-collection.json - -global-var "machineName=mymachine1234" - -global-var "machinePort=8080"**

In your request builder, just use them as normal variables **https://{‌{machineName}}:{‌{machinePort}}**.

**How I can connect with Databases using Postman**

* **Problem:**  If you wanted to validate the data in Response body against the Database (target DB)
* **Possible solution:**  for that you have to establish a connection with DB using Postman and hit SELECT query, get the data from DB and then validate.

However, Postman only make HTTP connection; so **Postman cannot be connected to DB**.

**There is another way to do that.**

You have to use 3rd party Middleware component, which will communicate with Postman using HTTP and then with Database using some database protocols;

It’s not easy as it sound. (that’s the reason, more info not given here)