Postman Interview questions

**Postman tutorials:** <https://www.tutorialspoint.com/postman/postman_quick_guide.html>

**What is the difference between request and response headers?**

Request headers provide information sent by the client to the server, while response headers include information sent by the server back to the client.

**What are common HTTP request headers?**

Common HTTP request headers include Authorization, Content-Type, User-Agent, Accept, Cache-Control, and Cookie.

**What are common HTTP response headers?**

Common HTTP response headers include Content-Type, Content-Length, Set-Cookie, Cache-Control, Server, and Location.

**What is the Content-Type header used for?**

The Content-Type header specifies the type of the resource, such as text/html, application/json, or image/png, informing the client how to handle the content.

**6) What are the important authorization methods supported by Postman?**

Postman offers the following API request authorization options 1) API Key, 2) Oauth 1.0 and 2.0, 3) Basic auth, 4) Digest auth, 5) Bearer Token, 6) AWS Signature, 7) Hawk Authentication, 8) NTLM Authentication.

**7) How do you log variable values in Postman?**

I can log the variable values in the Postman console by using the following command:

console.log(pm.variables.get("first\_name"));

9) What is the History tab in Postman?

All the request you send in Postman appears under the History tab of the sidebar. It is very much similar to browser history, which you can clear whenever you want.

10) What is Basic Auth in Postman?

In Postman, Basic Auth is an authorization method provided for HTTP user agents like web browsers to enter username and password. After entering the username and password that you can associate with the request.

11) What is a binary form in POST methods?

Post binary form is designed to send the information in a format which is impossible to enter manually. These options are used to send larger files like CSV files, etc.

12) What is the main difference between Authorization and authentication?

Here are a few differences between authorization and Authentication:

Authorization is the act of allowing or permitting someone, whereas authentication is validating that someone is genuine who will be authorized.

Authentication always comes first, while authorization comes after authentication.

Authorization is open to anyone with permission, whereas authentication requires you to have a password.

13) What is the Payload in Postman?

The Payload of an API Module is the body of your request and response message. When making an API request, it contains the data you send to the server. You can send and receive Payload in various formats, for example, JSON or XML.

14) What is a collection in Postman?

A collection in Postman enables you to group similar requests. It also allows you to systematically arrange the requests into folders.

15) What is a Pre-Request script?

Pre-request scripts help you to execute JavaScript before a request runs. It is used for setting variable values, parameters, headers, and body data before a request runs.

: what is **Bulk Edit** ?  
**Bulk Edit** is used to enter a set of key-value pairs for parameters.

16) How the Postman variables are accessed?

The Postman variables are always accessed by using the variable name:

{{variable name}}

17) What is the environment in Postman?

The environment in Postman is a set of key-value pairs. We can build multiple environments and switch among them with a click of a button.

18) Why does Postman never accepts any other encoding apart from Base64?

Because, base64 can transform the data into the text form and send it as HTML form.

Moreover, we must rely upon the same 64 characters in any encoding language.

Example: characters (A–Z, a–z), the numerals (0–9), and the "+" and "/" symbols, with the "=" symbol as a special suffix code.

19) Which kind of encoding does Postman accepts for authorization credentials?

Postman only accepts Base64 encoding, which is provided inbuilt in Postman. Otherwise, it would benefit when you use 3rd party websites that help you to convert the credentials into base64.

20) What are the different scopes of an environment variable in Postman?

Scope of a variable in Postman is defined as the boundaries it can access.

Here are important scopes of an environment variable in Postman:

Local Scope: This type of variable can be accessed only in the environment in which it was created.

Global Scope: This type of variable can be accessed globally in any environment or no environment.

21) Is it possible to import local variables in Postman Monitors?

Postman monitors allow you to import local variables but not global variables.

22) What is the Postman execution order for a collection?

For all the requests in a collection, the scripts will execute in the following given order:

Step 1) A pre-request script associated with a collection will run before every request.

Step 2) A pre-request script associated with a folder will run before every request in a specific folder.

Step 3) A test script associated with a collection will run after every request.

Step 4) A test script associated with a folder will run after the request in the specific folder.

**: pe-request script/Test script will run – Collection > folder > request level.**

23) Can you have two global scope variables with the same name in Postman?

No, the global scope never has duplicate/same names, while variables having local scope can have the same name in various environments.

24) How do you access postman variables?

You can log the variable values in the console by using the command:

console. Log (pm. Variables.get(‘name’);

25) What is the use of the collection in Postman?

In Postman, a collection is used to group similar requests. It helps you to arrange requests systematically into folders.

26) How can you use POSTMAN to generate random numbers in a specific range?

In your Pre-request script define your variable using random() method:

pm.globals.set('randomNum', Math.floor(Math.random() \* 5));

Then in your URL call your variable in the URL like so:

Output:

{{randomNum}}

27) How do you remove local variables?

Local variables are automatically removed once the tests have been executed.

28) What is ‘Postman Collection runners?

Postman Collection Runner is used for running a Collection also used for Data-driven testing.

**29) What do you mean by postman monitors?**

**A monitor is used for running a collection periodically to monitor its performance and response.**

Postman Monitors provide a way to automatically run test scripts and perform other tests at regular intervals.

The postman monitor feature is used for running collections for a specified time decided by user. However, the user must be logged in to their account to use this feature.

30) Why saving your work in the Postman cloud is not advisable?

You should not save your work in Postman as your business details do not remain confidential. Moreover, saving your on-Postman cloud may cause a security breach as it requires sign-in. Therefore, saving your work in the Postman cloud is not advisable.+

31) What are the standard rules of an API test design?

Here are the key principles of an API test design:

Setup: Create objects, start services, and initialize data.

Execution: Apply API or the scenario, including logging

Verification: It is used for evaluating the result of the execution

Reporting: Indicates Pass, failed, or blocked status

Clean up: Pre-test state

32) What is the Team workspace in Postman?

This is where the collaboration happens. this workspace is used to share and collaborate on APIs, collections, monitors, environments, and mocks.

33) What is the 301-status code?

301 status code tells us the request is redirecting from one website page to another. It tells the search engines that the old page has become outdated, and the engine has to index the new page or URL.

34) What is status code 201?

Status code 201 is created only when a resource is successfully created using a PUT or POST request. It returns a link to a newly created one with the help of the location header.

**35) What is the procedure to remove local variables?**

**Use unset to remove any variable.** The local variables can be automatically removed when you have executed and completed the tests.

37) How are Query Parameters different from Path Variables?

In Postman, Path Variables are used to identify the path for a specific resources, and Query Parameters are used to sort or filter the resources.

**38) What are the main drawbacks of Postman?**

Here are some cons/ disadvantages of using Postman:

**Postman cannot process more than 1000 API requests.**

It is not easy to manage the collections and requests in a massive size project.

Postman is not an ideal API tool for workspace management in the form of code as there can be lots of code duplication while handing the dynamic API requests.

39) What are some of the JS libraries available in Postman?

Some JS libraries available in Postman are

1) Lodash: Lodash is a JavaScript library that provides the functions and methods for working with arrays, objects and strings.

2) Moment: moment is used to add some formatting to the time object to suit a specific endpoint when POSTing data

3) GUID: GUID is short for Global Unique Identifier. It is hexadecimal digits that are used to solve the purpose of uniqueness.

41) How can you view the logs of requests and responses in Postman?

You can use the Postman Console window to view request logs and response logs.

42) **What is the importance of setNextRequest in Postman?**

setNextRequest helps you to control the excution workflow. It is needed to change the order of the requests being executed.

We also can move the requests up and down manually. Also move to any folder.

43) What test code allows you to check whether the response status is 200 or not?

Following is a test code to check whether the response status is 200 or not:

tests[“Status Code is 200”] = responseCode.code == 200;

44) What do you understand by ScratchPad?

Scratch Pad is a place that helps us work without connecting Postman servers.

45) How can you iterate a request 100 times in Postman?

You can iterate a request 100 times in Postman by using Collection Runner.

46) What would happen if {{$randomInt}} dynamic variable is added?

You need to add a random integer between 0 and 1000.

47) How do you access a variable value from a CSV/JSON file inside pre-request and test scripts?

data.variable\_name

data['variable\_name']

49) Which programming language is used for Postman tests?

JavaScript is used for Postman tests.

50) Which tool can be used to run Postman Collections in Jenkins?

Newman can be used to run Postman Collection in Jenkins.

**51) how to generate bearer token ?**

Bearer token is usually generated by a Server in response to a login request.

the client must send the token in the authorization header with a request to protect its resources.

: Select POST request method > enter domain URL > Auth > Basic Auth > user name + password

> Body > raw > JSon > {

"USERNAME" : "admin",

"PASSWORD" : "demo12"

} > send.

: then it will generate Bearer Token in response with expiration date.

: Developers gives us URL, Status, and Body for API test.

: note- developer provide us user name and password.

: Developer provide us the necessary values that we can use for Auth 2.0.

**52. Question: Can you explain the process of creating a new request on Postman ?**

**Answer**: To create a new request on Postman, you first need to have the API request details, which are typically based on a user story provided by the business. The developer then writes the code and provides an endpoint, which becomes the API request. In Postman, you can use Swagger to get the details of the API. By copying the curl command from Swagger, you can import the request into Postman and save it under the appropriate folder in your collection.

53. How many ways we can run a collection ?

Ans: we can run a collection

1. Locally

2. using collection runner

3. postman monitor to run periodically.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

software testing material

1. What is Postman?

Postman is a collaboration platform for API development. It is a popular API client and it enables you to design, build, share, test, and document APIs.

Using the Postman tool, we can send HTTP/s requests to a service, a well as get their responses. By doing this we can make sure that the service is up and running.

Being originally a Chrome browser plugin, Postman now extends its solution with the native version for both Mac and Windows.

2) What is an HTTP request?

An HTTP request is a program that the client makes to a name host located on a server. It works as a communication interface or a request-response protocol between a client and server. The primary use of the HTTP request is to access a resource on the server. To initiate the HTTP request, the client uses components of a URL (Uniform Resource Locator) that also includes the information needed to access the resource.

2. Why Postman?

Postman has become a tool of choice for over 8 million users.

Free: It is free to download and use for teams of any size.

Easy: Just download it and send your first request in minutes.

APIs Support: You can make any kind of API call (REST, SOAP, or plain HTTP) and easily inspect even the largest responses.

Extensible: You can customize it for your needs with the Postman API.

Integration: You can easily integrate test suites into your preferred CI/CD service with Newman (command line collection runner)

Community & Support: It has a huge community forum

3. What is an API?

API is an acronym and it stands for Application Programming Interface. API is a set of routines, protocols, and tools for building Software Applications. APIs specify how one software program should interact with other software program.

In simple words, API stands for Application Programming Interface. API acts as an interface between two software applications and allows the two software applications to communicate with each other. API is a collection of software functions which can be executed by another software program.

4. Name some tools used for API Testing?

Some of the tools used to do API Testing are as follows

Postman

Katalon Studio

SoapUI

Tricentis Tosca

Apigee

Jmeter

5. What are the core components of an HTTP request?

An HTTP request includes five key elements:

HTTP methods – Set of request methods to perform desired action for a given resource (GET, PUT, POST, DELETE)

Uniform Resource Identifier (URI) – Describes the resource

HTTP Version, (example- HTTP v1.1)

Request Headers, (example- Content-type : application/json, text/html, image/png, Content-Length : 511)

Payload – It is basically a Request Body which includes message content.

6. State The Core Components of an HTTP Response?

Every HTTP response contains four key elements.

Status/Response Code – These are response codes issued by a server to a client’s request. For example, 404 means Page Not Found, and 200 means Response is OK.

HTTP Version – describes HTTP version, for example-HTTP v1.1.

Response Header – Includes information for the HTTP response message. For example, Content-type, Content-length, date, status and server type.

Response Body – It contains the data that was requested by a client to server.

7. What API information is exposed in Web Developer tools?

Request headers, Response body, Response cookies

8. What can we use to get API information from web developer tools into Postman?

Copy as URL can get API information from web developer tools into Postman.

9. In which type of encoding does postman accept authorization credentials?

Postman accepts Base64 encoding only. This is provided inbuilt in postman or else you can also refer 3rd party websites to convert the credentials in base64.

10. Why does Postman accept Base64 encoding only?

We use base64 particularly because it transmits the data into the textual form and sends it in easier form such as HTML form data. Also, we can rely on the same 64 characters in any encoding language that we use.

11. What is meant by the term environment in postman?

An environment in postman is a set of key value pairs. we can create multiple environments in postman which can be switched quickly with a press of a button. There are 2 types of environment, global and local.

12. Can global scope variables have duplicate names in postman?

Since global variables are global i.e. without any environment, global variables cannot have duplicate names. Local variables can have the same name but in different local scopes.

13. Which one will be preferred in postman- a global variable or a local variable?

In postman, if 2 variables have the same name( one being local, other global) then the higher priority is of the local variable. it will overwrite the global variable.

14. What is a Postman Collection?

A Postman Collection lets us group individual requests together. Simply it allows us to organize the requests into folders.

15. What do you mean by postman monitors?

The postman monitor is used for running collections periodically.

Collections are run for specified time defined by the user. Postman Monitor requires the user to be logged in. Monitor reports are shared by users over email on a daily/weekly basis.

16. What do you understand by the term Postman Collection runners?

A postman collection runner is used to perform Data-driven testing. The group of API requests are run in a collection for the multiple iterations with different sets of data.

17. Can local variables be imported in Postman Monitors?

Yes. Postman monitors allow to import local variables but it does not allow to import global variables.

18. What is the purpose of Postman cloud if we are working in a company? Why?

A Postman cloud is a common repository of companies to access Postman collections. Any team member can access data/collections from anywhere. In Postman cloud, work can be saved instantly after logging in.

19. Why is it not preferred to save work in Postman cloud?

It is not preferred to save your work in Postman cloud as company’s work is not allowed to be leaked and remain confidential. Security breaches can be experienced if Postman cloud is used as Postman cloud requires sign in. **Therefore Postman Cloud is discouraged for saving work and team workspace is highly encouraged.**

20. What is the purpose of status code 304?

It means NOT MODIFIED. It is used to reduce network bandwidth usage in case of conditional GET requests. Response body should be empty. Headers should have date, location etc.

21. Define status code 201?

It means created, when a resource is successfully created using POST or PUT request. It returns a link to a newly created resource using the location header.

22. When do we use global variables, collection variables, and local variables?

Global variables are general purpose variables, ideal for quick results, and prototyping. They are used while passing data to other requests.

Collection variables can be mostly used for storing some constants that do not change during the execution of the collection. They are used for constants that do not change during the execution and also for URLs / authentication credentials if only one environment exists.

Local variables are only available within the request that has set them or when using Newman/Collection runner during the entire execution. They are used whenever you would like to override all other variable scopes. Local variables are used for dynamic variables.

23. How do you remove local variables?

Local variables are automatically removed once the tests have been executed. **Use unset..**

24. How can we stop executing requests or stop the collection run?

postman.setNextRequest(null);

25. What is the difference between form data and x-www-form-urlencoded ?

The difference between the form data and x-www-form-urlencoded is that the url will be encoded when sent through x-www-form-urlencoded.

26. Where are query parameters used in a GET request?

Query parameters are used in the URL in a GET request.

27. How can we access a Postman variable?

We can access a Postman variable by entering the variable name as {{var}}

28. What is the HTTP response code for a POST request with incorrect parameters?

400 Bad Request is an ideal response code for request with incorrect parameters.

29. How can you iterate a request 100 times in Postman?

By using Collection Runner

30. How can we organize requests in Postman?

We can organize requests in Postman with the Collections.

31. Which programming language is used for Postman tests?

JavaScript

32. What will execute first in a Collection Run?

Pre-request scripts at the Collection level are executed first in a Collection run.

33. What are some of the JS libraries available in Postman?

Lodash, Moment, GUID

34. Which tool can be used to run Postman Collections in Jenkins?

Newman can be used.

35. How can we log requests and responses in Postman?

We can view requests logs and response logs through the Postman Console window.

36. What is GUID?

GUID stands for Global Unique Identifier. It is basically hexadecimal digits separated by hyphens. GUID solves the purpose of uniqueness.

In Postman, we use this to generate and send a random value to APIs.

{

"id": "{{$guid}}",

}

**:::::::::::::::::::::::::::::::::::::::::: Mukesh Postman ::::::::::::::::::::::::::::::::::::**

(How To Download and Install Postman In Windows- Postman Tutorial For Beginner)

(Must Watch-Features and Quick Tour of Postman- Postman Tutorial For Beginner)

(How To Send Get Request In Postman- Postman Tutorial For Beginner)

(How To Send Post Request In Postman- Postman Tutorial For Beginner)

(How To Send DELETE Request In Postman- Postman Tutorial For Beginner )

(What is Collection In Postman and What is Collection Runner -Postman Tutorial For Beginner)

(Import and Export collection in Postman- Postman Tutorial For Beginner)

(How To Write API Test Cases In Postman Using JavaScript and Chai BDD- Postman Tutorial For Beginner)

(What Is Environment in Postman- Postman Tutorial For Beginner)

(What is Global Variable in Postman- Postman Tutorial For Beginner)

(How To Build Workflow in Postman- How to run multiple request in order-Postman Tutorials)

(How To Debug Scripts in Postman- Logs in Postman- Postman Tutorials in Beginner)

(Pre Request In Postman- How to use Pre Request In Collection-- Postman Tutorial For Beginner)

(How To Execute CURL Commands In postman- Postman Tutorial For Beginner)

(How To Run Postman From Command Line Using Newman- Postman Tutorial For Beginner)

(How To Generate HTML Report In Postman Using Newman- Postman Tutorial For Beginner)

(Advance HTML Reporting In Postman using HTMLExtra- Postman Tutorial For Beginner)

(How To Run Postman Test From Jenkins- Postman Integration with Jenkins)

(How To Fix Jenkins Console Output Character Issue)

(How To Integrate Postman with Github and Execute from Jenkins)

(Should I Use Postman Or RestAssured For API Testing )

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Postman practice \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* How to Create an WorkSpace ?**

**Workspaces – Create WorkSpace – blank workspace – name – save.**

**1. Collection : is a group of similar type of requests, works as a folder.**

: click on new or (+) > Collection > do name > enter.

**: Documentation -** click on Collection / folder > over view > add descriptions

Example:

**Date:** 6/4/2024

**Function Name:** Buyer Address

**SE Name:** John

**Tester Name:** Jawad

**Reviewer Name:** Shiblu Ahmed

**Test Objective:** Buyer should be able to add the address inputting valid information

**Expected Result:** Buyer should be able to add the address successfully

**Actual Result:** Buyer was able to add the address successfully

**Status:** passed

**Test Data:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Mobile No** | **State** | **City** | **Zip Code** | **Address** |
| Jawad Karim | 576467657 | MI | Hamtramck | 56765 | sanjoe lane |

**2. Request:**   
: **GET** – is used to get the resource data.

click on new or (+) > select request type(GET) > enter URI > save > name > select/create a collection.

: **POST** – is used to create new resource.

click on new or (+) > select request type(POST) > enter URI > body > raw > json > enter body.

**: PUT** – is used to update the entire resource. It will re-initialize the entire resource.

click on new or (+) > select request type(PUT) > enter URI > body > raw > json > enter entire body.

PUT also can update parial body then we have to specify ID of the part to the end point.

**: PATCH** - is used to update the resource partially. we have to specify ID of the part to the end point.

click on new or (+) > select request type(PATCH) > enter URI > body > raw > json > enter partial body.

**: POST is super keyword work for PUT and PATCH as well.**

: **DELETE** - click on new or (+) > select request type(DELETE) > enter URI > send (without body).

: **DELETE** - click on new or (+) > select request type(DELETE) > enter URI > send (with body).

: we can drag and drop requests from one collection to another collection.

:

**3. how to run collections:**

Click on 3dots(…) of collection > Run collection > Select/deselect requests > Run selected collection.

**4. how to add folders in collection:**

Click on 3dots of collection > add folder > do name.

Now we can add requests in folders.

**5. how to import and export collection ?**

Export: click on three dots(...) > export > export > select file > save.

Import: click on import option > import file/folder/from Link- paste the URI or Raw Text.

**6. how to share a collection ?**

Click on three dots(…) > share > enter email/ group name > OR copy link > share the link.

**7. what is Test in Postman ?**

: test is used to put the validations in your request, make sure API is working as expected.

: a request can have many validations

: tests are written in javascript in postman.

: tests can be added at request, folder, and collection level.

**: how to set the test…**

Select request, folder, or collection > Tests > select a test type from right-side > send request > then click **Test** **results** at bottom to see the result.

Test type can be- status code, response time, response body, response headers etc.

**8. what is Enviroment ?**

Environment is a set of key-value pairs.

Why we need: if we have many requests using same values then its better to use an Environment variable if any changes happen then we will change on the Environment value only and it will effect all the requests. Example = baseURI.

Create Environment: we can create Environment/Local level and global level.

Click on eye button > add (Envirorment or global level) > do Environment name(smoke, regression etc), vairable name(baseURI etc) > initial value(<http://reqres>... (for team)), current value(<http://reqres>... (for run)). We also can copy environment.

How to use Environment:

Select an Environment > add the variables with values >

To use => examples:

In URI - {{baseURI}}/page2.

Int payload – “{{firstName}}”

Print in console – pm.veriables.get(“variableName”);

**9. how to build/maintain/chain workflow ?**

Click a request > Tests > write in the text field

[ **postman.setNextRequest(“requestName”);** ]

Note: it will run specified request and keep goes next requests, any request skipped before specified one will not run unless we set **setNextRequest** for it..

: to stop **setNextRequest** after execution of this [ **postman.setNextRequest(“null”);** ]

**10. how to debug and log in postman ?**

**: Debug** is to identify what went wrong in the API call. When we run a re quest all the responses will be logged in postman console, under **All Logs we can define log, info, warning, error.**.

: we can filter the logs in console window by clicking on **All Logs** drop-down. We also can write the debuging logs manually in Pre-request Scrips or Test Scripts. Like console.log/info/warn/error(); etc.

: To debug the collection/request, Postman provides the Postman Console feature.

Step 1: Show Postman Console option is in the View tab on the top. Also, we have an icon of Postman Console at the bottom. Click on the icon. It will open the Postman Console window.

Step 2: Now, whatever request we will run, it will be logged in the console window.

: Send an API request. Check logs in the console window.

: All the request headers, body, response headers, response body, status codes, response size, and all the script outputs will be logged in the console window.

: If there is any variable used in the request, their actual value will also be shown in the Console window.

Step 3: Expand the response headers, network, request headers, body to view the data.

Step 4: We can also select the raw format and pretty format from the option in right.

Step 5: We can also access errors/info/warnings/logs separately from the Console window. There is an option of Clear too to clear the logs displayed.

Step 6: Also, we can use console.log(), console.info(), console.warn(), console.error() statements in Pre-request scripts or Tests scripts to log on to the Postman console.

**Log** is to log the information and result in the console.

console.log();

console.info();

console.error();

console.warn();

// Example of logging messages in Pre-request Script

console.**log**("this is log messages”);

console.**info**("name is set as info : "**+**name);

console.**warn**("this is warn message”);  //we check the result using if else in

console.**error**("this is error message”); Test then log messages as warn or error

**\*\* Trouble shooting \*\***

: Some time we might make mistake then check the console logs and see what happened.

: some time everything is correct but still getting error then visit help **Github issue tracker**.

: create variables in Pre-request Script-

var name **=** "jawad"; //declare the variable

console.**log**("name is set as : "**+** name); //print it in the console as one String.

console.**log**("name is set as : "**,** name); //print it in the console as separate String.

**// check the response data in Test Scripts.. are we getting correct data or not !**

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

console.**log**(jsonData);

var pageNo **=** jsonData.page;

console.**log**(pageNo);

var number **=** Math.**random**(); // returns less than 1 (0.(16 digits))

console.**log**("number: "**+**number);

**//add the varible to Environment level and global level.**

var name **=** "jawad";

pm.globals.**set**("userName", name); // can see the variable in Environment after run.

pm.environment.**set**("userName", name); // specific Enviroment level.

**// use variable in payload (post request body).**

Click on body then add the code below...

{

"id": *{{id}}*,

"name" : *“{{userName}}”*;

  "job" : "automation";

}

: *{{Reqres\_url}}***/api/users?page=2** // use variable in request url

**11. Pre-request Script and Tests Script in postman**

: order of execution-

Pre-request Script: collection > folder > request level Script will run before request run.

Tests Script: collection > folder > request level Script will run after request run.

**12. what is variable in postman ?**

: variable is a key-value paires. When need to use same value again and again then we need to use variable.

: variable can be created at ....

**global level**(accessible from any collecttion)- eye/view button > global add > name > value > save.

**collection level**(accessible from collection)- select collection (…) > Edit > varible > name > value > save.

**Environment level**- eye/view button > Environment add > name > value > save.

**Local/request level-** Pre-request Script > pm.variables.set(“key”, “value”); > save.

**: creating variable using Pre-request Script…**

// local/request variables

pm.variables.set(“key”, “value”); > save.

var name **=** "jawad1";

console.**info**("name is: "**+**name);

pm.variables.**set**("name", "jawad2");

console.**log**("Name: ", pm.variables.**get**("name"));

//collection variables

pm.collectionVariables.set(“key”, “value”);

//global variables

pm.globals.set(“key”, “value”); > save.

//Environment variables

pm.environment.set(“key”, “value”); > select an Environment > save.

**data level**- comes from external files.

**13. how to remove variable after execution ?**

//Local variable remove

pm.variables.unset(“key”);

//global variables remove

pm.globals.unset(“key”) > save.

//Environment variables remove

pm.environment.unset(“key”) > select an Environment > save.

//collection variables remove

pm.collectionVariables.unset(“key”);

**14. how to print the variable values in console ?**

//print Local variable

console.log(pm.variables.get(“key”));

// print collection variables

console.log(pm.collectionVariables. get (“key”));

// print global variables

console.log(pm.globals. get (“key”));

// print Environment variables

console.log(pm.environment. get (“key”));

**15. how to validate the response in postman?**

**Response validations:**

**Status code**

**Headers**

**Coockies**

**Response time**

**Validate response body data.**

**Note :**

first we need to know proper expected values before sending a request then we can compare the result.

We need to run the request one time before setting validation.

For validation we need to use chai assertion library.

Using javaScript we can write function for validation in two diferent ways for **Tests**.

**Normal function: function(){ }**

**Arrow function: () =>{ }**

**GET Getting started with tests**

**// verify status code, message, headers, response time, coockies, body**

const jsonData **=** pm.response.json(); **//create object of entire response**

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

pm.response.to.have.status(201);

pm.response.to.have.status(“Created”);

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

pm.expect(pm.response.headers.**get**('Content-Type')).to.include('application/json');

pm.expect(pm.response.status).to.equal("OK");

pm.expect(pm.response.code).to.equal(200);

pm.expect(pm.response.code).to.be.oneOf([200, 201]);

pm.expect(pm.response.responseTime).to.be.below(200);

pm.expect(pm.cookies.**has**('JSESSIONID')).to.be.true;

pm.expect(pm.cookies.**get**('isLoggedIn')).to.eql('1');

pm.expect(jsonData).to.be.an("object"); **// verify data type**

  pm.expect(jsonData.name).to.be.a("string"); **// verify data type**

pm.expect(jsonData.data.length).to.eql(6); **// verify array size**

pm.expect(jsonData.data[0].id).to.eql(7);

pm.expect(jsonData.data[0].address).to.be.empty;

pm.expect(jsonData.data[0].address).to.be.null;

pm.expect(jsonData.data[0].address).to.be.undefined;

pm.expect(jsonData.data[0].department).to.include("products"); **// 1 of array values**

});

------------------------------------------------

**: verify response Body**

{

    "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://reqres.in/img/faces/7-image.jpg"

        },

}]

**//test body data**

**Asserting the data type of the value.**

/\* response has this structure:

{

  "name": "Jane",

  "age": 29,

  "hobbies": [

    "skating",

    "painting"

  ],

  "email": null

}

\*/

const jsonData **=** pm.response.json(); //create object of entire response

pm.**test**("verify data type of the response", () **=>** {

  pm.expect(jsonData).to.be.an("object");

  pm.expect(jsonData.name).to.be.a("string");

  pm.expect(jsonData.age).to.be.a("number");

  pm.expect(jsonData.hobbies).to.be.an("array");

  pm.expect(jsonData.website).to.be.undefined;

  pm.expect(jsonData.email).to.be.null;

});

**: Asserting value of array fields / properties.**

**For that we need to find the Json path using Json pathFinder.**

data : [{

  "id": 7,

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

  "first\_name": "Michael",

  "last\_name": "Lawson",

  "avatar": <https://reqres.in/img/faces/7-image.jpg>

"address": [],

  "department": [ "products", "services" ],

}]

const jsonData **=** pm.response.json();  //create object of entire response

pm.**test**("Test array data of the response", () **=>** {

pm.expect(jsonData.data.length).to.eql(6); **//array size validation**

pm.expect(jsonData.data[0].id).to.eql(7);

pm.expect(jsonData.data[0].email).to.eql("michael.lawson@reqres.in");

pm.expect(jsonData.data[0].first\_name).to.eql("Michael");

pm.expect(jsonData.data[0].last\_name).to.eql("Lawson");

pm.expect(jsonData.data[0].avatar).to.eql("https://reqres.in/img/faces/7-image.jpg");

pm.expect(jsonData.data[0].address).to.be.empty;

pm.expect(jsonData.data[0].department.to.include("products");

**// validate array size**

var totalData **=** jsonData.data.length;

console.**log**("total data: "**+** totalData);

});

**: Validating Json schema validation.**

**Json schema is the description of layer / type of data...**

**Json body-**

"data": [

        {

            "id": 7,

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

            "first\_name": "Michael",

            "last\_name": "Lawson",

            "avatar": "https://reqres.in/img/faces/7-image.jpg"

        }]

**Schema of Json body- we need to get the schema using ‘Json to JsonSchema’ tool online.**

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

const schemaData **=** {"$schema": "[http://json-schema.org/draft-04/schema#](http://json-schema.org/draft-04/schema)",

"type": "object",

"properties": {

"data": {

"type": "array",

"items": [

{

"type": "object",

"properties": {

"id": {

"type": "integer"

},

"email": {

"type": "string"

},

"first\_name": {

"type": "string"

},

"last\_name": {

"type": "string"

},

"avatar": {

"type": "string"

}

}

**Validation- of Scema…**

pm.**test**('Schema is valid', **function**() {

//check schema is valid or not. tv means tiny validator.

pm.expect(tv4.validate(jsonData, schemaData)).to.be.true; // or

pm.response.to.have.jsonSchema(schemaData);

});

**: We can validate GET, POST, PUT, PATCH, DELETE method same way..**

**: GET method will be validated with body as well…**

**16. what request chaining ?**

**We can take the response/value from previous request and use that in the next request is called request chaining.**

**// sample API : https://gorest.co.in/**

**End-point….**

**POST public/v2/users Create a new user**

**GET public/v2/users/1905 Get user details**

**PUT|PATCH public/v2/users/1905 Update user details**

**DELETE public/v2/users/1905 Delete user**

**Generate a Token: GoRest home page > click ‘Get your access token’ …..**

**Token : 148ec7a18af4d282483b01ca31e5b7591ad19e51fca15e2877dac16d22b12abb**

**Body for POST request:**

{

"name":"Tenali Ramakrishna",

"gender":"male",

"email":"tenali.ramakrishna@15ce.com",

"status":"active"

}

: for sending request we need to pass Authorization…

Go to the cllection/folder/request > Click on Authorization > inherit authorization > Bearer Token > paste the Token.

Every time we need to pass new email, so we need to generate new email and name for every request.

So, before sending the POST request I will write some logic which will automatically generate name and email or any value.

Go to Pre-request Script > add… **POST** request

// create randon number data

var random\_number **=** Math.**floor**(Math.**random**()**\***100);

console.**log**("random\_number: "**+** random\_number);

//create random string data

: .**toString**(36) means 0-9 numbers + 26 letters = 36 & it will return 13 digits value.

: .**substring**(2) return 13 - (2 digits from left) = 11 digits from right of 13 digits.

var random **=** Math.**random**().**toString**(36).**substring**(2); //create random string

// create random userEmail

var userEmail **=** "jim"**+**random**+**"@gmail.com";

// create random userName

var userName **=** "jim"**+**random;

pm.environment.**set**("uEmail",userEmail); //create Environment variable for email

pm.environment.**set**("uName",userName);  //create Environment variable for name

Now we can use this variables in the POST request body…

{

"name":"*{{uName}}*",

"gender":"male",

"email":"*{{uEmail}}*",

"status":"inactive"

}

Now create an Environment variable for ‘userID’ using the response at **‘Tests’** script option **POST** request **validation**.

var jsonData **=** JSON.**parse**(responseBody); //store response body into a variable

pm.environment.**set**("userId", jsonData.id);  // create an Environment variable

---------------------------------------------

**GET** **method:**

URI - [https://gorest.co.in/public/v2/users/{{userId}}](https://gorest.co.in/public/v2/users/%7b%7buserId%7d%7d)

**//validating response data at Tests script option**

pm.**test**("values of json fields", **function**(){

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

    pm.expect(jsonData.id).to.eql(pm.environment.**get**("userId"));

    pm.expect(jsonData.name).to.eql(pm.environment.**get**("uName"));

    pm.expect(jsonData.email).to.eql(pm.environment.**get**("uEmail"))

})

---------------------------------------------------------------------------------

**PUT** **method: URI -** https://gorest.co.in/public/v2/users/{{userId}}

Pre-request Script…

var random **=** Math.**random**().**toString**(36).**substring**(2); //create random string 2 digits

var userEmail1 **=** "jim"**+**random**+**"@gmail.com";  // create random userEmail

var userName1 **=** "jim"**+**random;   // create random userName

pm.environment.**set**("uEmail1",userEmail1); //create Environment variable for email

pm.environment.**set**("uName1",userName1);  //create Environment variable for name

**body…**

{

"name":"*{{uName1}}*",

"email":"*{{uEmail1}}*"

}

**Tests Script for validation….**

var jsonData **=** JSON.**parse**(responseBody); //store response body into a variable

pm.environment.**set**("userId", jsonData.id);  // create an Environment variable

**Delete method : URI -** https://gorest.co.in/public/v2/users/{{userId}}

**// we also can delete Environment variables after completion (it is best practise)**

pm.environment.unset("uEmai");

pm.environment.unset("uName");

pm.environment.unset("uEmail1");

pm.environment.unset("uName1");

**17. how to do Data Driven Test in Postman ?**

: create csv file - write data in excel > do column name for variable > save as name.csv.

: example.csv file

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *endPoint* | *userTitle* | *fName* | *lName* | *userEmail* |
| api/users | developer | Robin | joese | ashd@gmail.com |
| api/users | manager | iron | kally | asdf@gmail.com |
| api/users/qa | qa | sejan | tom | ashs@gmail.com |

: use {{column/variable name}} in the request body to pass data

: example URI- https://reqres.in /{{*endPoint* }}

: example POST request body to use CSV data

{

    "email": "*{{userEmail}}*",

    "first\_name": "*{{fName}}*",

    "last\_name": "*{{lName}}*",

    "title": "*{{userTitle}}*"

}

: set Tests to verify response data with CSV data— to make sure correct data was used from CSV file.

**: now execute test-**

click on three dots of collection or folder > run collection or folder > Data file[select file] > run collection/folder.

**: to run data driven test in cmd –**

**newman run “file.json” –e “envinronmenFile.json” –d “file.csv/json” –r html > enter**

example

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

pm.expect(jsonData.email)

: retreive CSV data example-

**data[‘**userTitle**’]** //here data mean CSV file

**: test and print CSV data in console- example**

pm.**test**("verify response data with CSV data", **function** (){

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

    pm.expect(jsonData.email).to.eql(data['userEmail']);

})

pm.**test**("verify response tile with CSV data", **function** (){

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

    pm.expect(jsonData.title).to.eql(data['userTitle']);

})

// print the value in console from CSV file

console.**log**("email from CSV file : "**+** data['userEmail']);

console.**log**("tite from CSV file : "**+** data['userTitle']);

: click (…) on collection > Run Collection / Runner > data(select file) > run collection/folder > select file > run.

: request will run as many times as many rows (datas) in each column.

: then POST request will create an unique ID for the new resource.

**Data driven test using Json file…**

**: simple approach for POST request body….**

**: copy the sample Json data from reqres and paste it in the body then update the datas..**

**: I don’t have to keep the entire data, I can remove some data as I want to.**

{

    "data": [

        {

            "email": "jawadkarim@yahoo.com",

            "first\_name": "jawad",

            "last\_name": "karim",

            "avatar": "https://reqres.in/img/faces/1-image.jpg"

        }

    ],

    "support": {

        "url": "https://reqres.in/#support-heading-forTest"

    }

}

Response body is generated.........

{

    "data": [

        {

            "email": "jawadkarim@yahoo.com",

            "first\_name": "jawad",

            "last\_name": "karim",

            "avatar": "https://reqres.in/img/faces/1-image.jpg"

        }

    ],

    "support": {

        "url": "https://reqres.in/#support-heading-forTest"

    },

    "id": "682",

    "createdAt": "2023-12-12T16:57:23.909Z"

}

Id is automatiacally generated for us...

**Using Json file…**

**: validate Json file before using – open any Json formatter to validate**

**: add Json data in text/notepad inside an Array [ ] and validate in a Json formatter.**

**OR**

**: write Json data in text/notepad like below > validate > download > save/rename( name.json) –**

**[**

**{**

**"email": "jawadkarim@yahoo.com",**

**"first\_name": "jawad",**

**"last\_name": "karim",**

**"avatar": "https://reqres.in/img/faces/1-image.jpg"**

**},**

**{**

**"email": "jawadkarim@yahoo.com",**

**"first\_name": "newaj",**

**"last\_name": "sharif",**

**"avatar": "https://reqres.in/img/faces/1-image.jpg"**

**}**

**]**

: example POST request body to use JSON data

{

"email": "*{{email}}*",

"first\_name": "*{{first\_name}}*",

"last\_name": "*{{last\_name}}*",

"avatar": "*{{avatar}}*"

}

**Test Scripts to use JSON data**

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

pm.**test**("verify email", **function** () {

     pm.expect(jsonData.email).to.eql(data['email']);

});

pm.**test**("verify first name", **function** () {

     pm.expect(jsonData.first\_name).to.eql(data['first\_name']);

});

pm.**test**("verify last name", **function** () {

     pm.expect(jsonData.last\_name).to.eql(data['last\_name']);

});

**If we want to use/test csv/json data in scripts then….**

tests["verify name"] **=** responseBody.**has**(data.name); //data.name coming from csv/json file. We can not use variable like {{name}} inside the scripts.

tests["verify email"] **=** responseBody.**has**(data["abc@gmail.com"]);

**: use data in request like – {{userName}}**

**: click (…) of collection > run collection/Runner > select file > run.**

**: we can convert csv to Json file using any online tool. Just google csv to Json**

**18. How to debug in postman ?**

**It is used to print the status on console…**

**console.log()**

**console.info()**

**console.warn()**

**console.error()**

**Debugging scripts can be written under either the Pre-request Script tab or the Tests tab, with helpful messages logged in the Postman Console**

**The Postman Console logs the following information:**

**: The primary request that was sent, including all underlying request headers, variable values, and redirects**

**: The proxy configuration and certificates used for the request**

**: Network information such as IP addresses, ciphers, and protocols used**

**: Log statements and asynchronous requests from test or pre-request scripts**

**: The raw response sent by the server before it's processed by Postman**

**19. How to run postman using Jenkins**

**: go to Jenkins > create a new Job > Freestyle Project > ok**

**: install node.js and newman**

**: build > add build step > Execute windows batch command > collection URI.**

**20. How many ways we can run the collections ?**

**: run collection and generat report on command line...**

: **node.js + npm is required to install newman.**

**First I need to download nvm for windows then install node.js and npm**

Then Install **newman** Type **(npm install -g newman) in commandline.**

To Install **newman-html-reporter** Type **(npm install -g newman-reporter-html) in commandline.**

For **htmlextra-reporter** Type **(npm install -g newman-reporter-htmlextra) in commandline.**

**----------------------------- install nvm , node js, npm, newman ------------------------------------------**

**: install nvm then install node and npm using uvm then install newman.**

**: google(download nvm for windows) then go to github page > scroll > click download (latest nvm version)**

**: cmd > nvm -v (check nvm version)**

**: cmd > nvm install lts (lts=long tertm support)**

**: cmd > nvm use 22.11.0 (nvm use version number) then check node and npm version number**

**: cmd > node -v) and (cdm > nmp -v)**

**: cmd > npm install -g newman (then check version (cmd > newman -v) if u can see version then its good. )**

**if not then ...**

**: cmd > npm install -g @vue/cli (then chck vue verson(cmd > vue -v) u can see vue version' then try to install newman again)**

**: npm install -g newman**

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

**Check version-**

**newman –v.**

**npm –v. npm (note package manager)**

**: export collection and environment- to run the collection on command line we need to export our collecion first.**

**In postman collection click on (…) > Export > Export > save. (collection will be saved as name.json file in computer)**

**Now run the cllection on cmd…**

**Approach 1-**

**open cmd > type- newman run > go to saved file > drag the file to cmd > enter.**

**Or open cmd > type- newman run “ file path or link “ -e “environment file path” > -d “data file path” > -r html, htmlextra, cli or json > enter. (or drag and drop files)**

**\*(use –e for environment and –g for global variables).**

**\*for csv data file save excel file as .csv file**

**\*for json data file convert csv to json file using online tool. Then save.**

**\*\* How to run collection and generate html report with newman ?**

**Install newman html reporter**

**Run in cmd - npm install newman-reporter-html**

**then add –r html at the end of file path then enter**

**Example command: newman run https://collectionfile.com –r html**

**Report will be stored in c-drive/users/Jawad/newman folder**

**Approach 2- run the collection remotely using url. (remote environments). can share the url**

**Create an url of your colleciton: Postman collection > click on (…) > share > via API > Copy Link.**

**Now open cmd >**

**type- newman run “collection url”**

**> -e “environmen url” or -g “globals variable url” > enter.**

**Approach 3- run postman collection using Jenkins (in real job we run collection GIT \_url only).**

**Open Jenkins > new Item > do name > freestyle project > build > Execute Windows batch command -**

**type newman run collection\_git\_URL > apply.**

**Now go to project > build now.**

**\*\*\*\* run postman collection from local using Jenkins and generate html reports\*\*\*\***

**Open Jenkins > new Item > do name > freestyle project > ok**

**: install node js**

**Dashboard > manage jenkins > tools > add node js > install automatically or provide path.**

**: install HTMLpublisher**

**Dashboard > manage jenkins > plugins > available plugins > type HTMLpublisher > install**

**: now go to job/project > source code(none) > build > Execute Windows batch command >**

**type…**

**newman run "postman\_collection.json file path" -e "environment.json file path" -d "testData.json file path" --reporter-htmlextra-export folder-path to store report\report folder name/ ReportName.html**

**example attaching html report:**

**newman run "C:\Users\jawad\OneDrive\Documents\Pet API test Data\Pet API.postman\_collection.json" -e "C:\Users\jawad\OneDrive\Documents\Pet API test Data\Pet Api.postman\_environment.json" -d "C:\Users\jawad\OneDrive\Documents\Pet API test Data\PetAPItestData.json" -r htmlextra --reporter-htmlextra-export C:\Users\jawad\newman\results/Newman\_HTML\_Report.html**

**run postman collection from Github using Jenkins**

**Open Jenkins > new Item > do name > freestyle project > ok**

**Source code management (Git) > enter Git url > build > Execute Windows batch command >**

**type …**

**newman run “ collectionFileName.json” Or**

**newman run “collectionFileName.json” –e ”environmentFile.json” –d “dataFileName.json” Or**

**newman run “collectionFileName.json” –e ”environmentFile.json” –d “dataFileName.json” -r Or**

**newman run “collectionFileName.json” –e ”environmentFile.json” –d “dataFileName.json” –r --disable-unicode**

**(verified)**

**newman run "Pet\_API.postman\_collection.json" -e "Pet\_Api.postman\_environment.json" -d "PetAPItestData.json" -r htmlextra - -reporter-htmlextra-export "C:\Users\jawad\newman\results/Newman\_HTML\_Report5.html"**

**> apply > save. Now go to project > build now.**

**Github token: ghp\_VNvqbNeJXox0Wx1eAHsAaCYFMA0Wea1YOh03**

**Approach 4- run collection using jenkins from GitHub**

**: install Git app then I will find Git UI and Gitbash, on mouce right click.**

**: Create a folder and store the Exported Collection into this folder then right click on that folder and click Gitbash here.**

**First push your collecion to Github…**

**: create an empty remote repository –**

**open GitHub > new button > do name.. copy the URL**

Open **exported collection file** then Open **Gitbash –** andtype below commands

**1) create a local repository-**

**$ git init (it will create an empty local git repository)**

**2) connect local repo with remote repo**

**$ git remote add origin https://github.com/collectionRepo.git**

**3) add file to index/stage**

**$ git add local file name ( eCommerce.postmancollection.json )**

**4) save/commit to local repo with message**

**$ git commit –m “commit message”**

**5) push repo to github**

**$ git push –u origin master ( -u is optional. If I use it will ask for user name & password )**

**$ git push origin master ( -u is optional. If I don’t use it will not ask for user name & password )**

**$ git add README.md**

**$ git branch –M main**

**6) to check the status**

**$ git status**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Response validations \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**We need to set the validations in Test scripts to verify the data and types of data.**

**Response:**

**{**

**“id” : 2,**

**“name” : “jawad”,**

**“location” : “usa”,**

**“phone” : 23455677,**

**“courses” : [**

**“java”,**

**“selenium”**

**]**

**}**

**1. validate status**

pm.**test**("validate status code/ message", **function**(){

    pm.response.to.have.status(200);

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

    pm.expect(pm.response.code).to.be.oneOf([200, 201]);

}

)

**2. validate headers**

pm.**test**("validate header is present", **function**(){

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

    pm.expect(pm.response.headers.**get**("Content-Type")).to.eql

("application/json; charset=utf-8");

} )

**3. validate Coockies**

pm.**test**("validate 'word' is present in Coockie", **function**(){

    pm.expect(pm.cookies.**has**("language")).to.be.true;

    pm.expect(pm.cookies.**get**("language ")).to.eql("en-gb");

} )

**4. validate Response time**

pm.**test**("validate response time is less than 300 ms", **function**(){

    pm.expect(pm.response.responseTime).to.be.below(300);

} )

**5. validate type of data**

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

pm.**test**("validate data type of the response", **function**(){

    pm.expect(jsonData).to.be.an("object");

    pm.expect(jsonData.name).to.be.a("String");

    pm.expect(jsonData.id).to.be.a("number");

    pm.expect(jsonData.data).to.be.an("array");

} )

**6. validate values of fields in response**

pm.**test**("validate values of the fields", **function**(){

    var jsonData **=** pm.response.jsonData;

    pm.expect(jsonData.id).to.eql(2);

    pm.expect(jsonData.name).to.eql("jawad");

pm.expect(jsonData.data.length).to.eql(4);

    pm.expect(jsonData.data[0].courses).to.eql("java");

} )

**7. validate array properties**

pm.**test**("validate array properties", **function**(){

    pm.expect(jsonData.courses).to.include("java");

    pm.expect(jsonData.courses).to.have.members(["java", "selenium"]);

    } )

**8. validate Schema data types are correct or not**

**: to validate Json Schema we need to Generate JSon Schema.**

**: google > type Json to Json Schema**

**> copy response body then paste to converter**

**> copy Json Schema then create a variable in postman script as**

**var schema = (paste the schema here); // then test it.**

**var schema = {**

**"$schema": "http://json-schema.org/draft-04/schema#",**

**"type": "object",**

**"properties": {**

**"data": {**

**"type": "object",**

**"properties": {**

**"id": {**

**"type": "integer"**

**},**

**"email": {**

**"type": "string"**

**},**

**"first\_name": {**

**"type": "string"**

**},**

**"last\_name": {**

**"type": "string"**

**},**

**"avatar": {**

**"type": "string"**

**}**

**},**

**"required": [**

**"id",**

**"email",**

**"first\_name",**

**"last\_name",**

**"avatar"**

**]**

**},**

**"support": {**

**"type": "object",**

**"properties": {**

**"url": {**

**"type": "string"**

**},**

**"text": {**

**"type": "string"**

**}**

**},**

**"required": [**

**"url",**

**"text"**

**]**

**}**

**},**

**"required": [**

**"data",**

**"support"**

**]**

**}**

**:Schema validation**

pm.**test**('Schema is valid', **function** () {

   pm.response.to.have.jsonSchema(schema);

  pm.expect(tv4.validate(jsonData, schemaData)).to.be.true;

});

: tv4 means tiny validator version 4.

**\*\*\*\*\*\*\*\*\*\*\*\*\* \*\* varibles in prerequest scripts and Test scripts \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**we can set / get / unset environment / global variables from the snippets at right side window.**

pm.environment.**set**("variable\_key", "variable\_value");

pm.globals.**set**("variable\_key", "variable\_value");

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

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

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

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

======================== Postman ts4u + Manish ======================

Ts4u portal: <https://portal.schoolshub.ai/chat/663a51e7d85851001f6c4b82>

Manual test plan Mock interview questions:

link:- <https://docs.google.com/spreadsheets/d/11_sXHsag95BxCZ3sB_XOIBLMzAfA8Y0ChsnDaMSqHh4/edit?usp=sharing>

**Postman questions ts4u:** https://docs.google.com/spreadsheets/d/11\_sXHsag95BxCZ3sB\_XOIBLMzAfA8Y0ChsnDaMSqHh4/edit#gid=0

**Agile ALM:** https://agilealm.cloud/organizations

ashrafulislamashik960@gmail.com

Pass: ashik@2430

schema validation:

. copy the body and paste to schema vaildator

. copy the schema then create variable in Tests

var validSchema = "copy schema"

: creatiate variable using Pre-request Script…

pm.variables.set(“key”, “value”); > save.

var name = "jawad1";

console.info("name is: "+name);

pm.variables.set("name", "jawad2");

console.log("Name: ", pm.variables.get("name"));

pm.globals.set(“key”, “value”);

pm.globals.unset(“key”);

pm.environment.unset(“key”);

pm.collectionVariables.unset(“key”);

console.log(pm.variables.get(“key”));

// create randon number data

var random\_number = Math.floor(Math.random()\*100);

console.log("random\_number: "+ random\_number);

//create random string data

: .toString(36) means 0-9 numbers + 26 letters = 36 & it will return 13 digits value.

: .substring(2) return 13-(2 digits from left) = 11 digits from right of 13 digits.

var random = Math.random().toString(36).substring(2); //create random string

// create random userEmail

var userEmail = "jim"+random+"@gmail.com";

// create random userName

var userName = "jim"+random;

=====================================================================

Postman + Github

1. Team lead/admin will create a repository in Github exml: postman api\_testing.

and will share with the team.

2. Create a folder Postman\_Collections

3. Export the postman collection and environment into that folder also store csv or json data file into folder.

4. right click on that folder - click Gitbash here

If you use cmd then type **cd folder directory** (to go to expected folder)

cd = change directory | dir = directory.

5. git init (create empty local git repo in the same folder)

6. Create a README.md file inside the same folder.(optional)

--- add exported collection files ot index ----

**git add** fileName or (**git add .** to add all file)

git add README.md (file will be added to git stage/index)(optional)

git add file\_name/path (file will be added to git stage/index)

--- commit the staged files ---

9. **git commit -m "first commit"**

--- specify the branch of github repo (optional) ---

**git branch** (see all the branches & current one will be high-lighted)

**git checkout branch-name** (switch to the branch)

**git checkout -b new-branch-name** (create & checkout new-branch)

10. (rename the branch)

**git branch -m newName**

11. --- add/specify the Url of github repo --- make connection

**git remote add origin https://github.com/Jawad-Karim/ECommerce-API-test.git**

12. --- push to Github ----

**git push –u origin master** (collection repo will be available in github)

lets say 5 or 10 members in your team...

Now import the repo into postman to work/update test cases.

Create a folder 'member1' ---

click github repo file > copy code

click postman import > paste code > save as copy/replace > start working on it.

then Export the postman collection into that folder and push it to Github again.

postman -import > select file > import/save > start working on it

**================================= Git lab ===================================**

Git Fetch fetch gets all the change history of a tracked branch/repo.

git fetch origin

git status

git log origin/master to view the logs.

git diff origin/master to see the difference betweeen local master & origin master

git merge origin/master we can merge our current branch (master) with origin/master:

pull is a combination of fetch and merge. It is used to pull all changes from a remote repository into the branch you are working on.

git pull Pulling a Branch from GitLab

git commit -a -m "Updated Resized image"

Now push our changes to our remote origin:

git push origin

git branch to see the branches

git branch -a to see all local and remote branches:

git checkout html-skeleton

Switched to a new branch 'html-skeleton'

The GitHub flow works like this:

Create a new Branch

Make changes and add Commits

Open a Pull Request

Review

Deploy

Merge

**How generate a Personal Access Token in Gitlab.**

Profile Icon > Edit Profile or Preferences > Access Token > Add new token > do name, exp date > select all scope that will be for user.

**How generate a Project Access Token in Gitlab.**

On the left sidebar, select Search or go to and find your project.

Select Settings > Access tokens > Add new token > Enter a name > Enter an expiry date

> Select a role > scopes > Select Create project access token.

**How generate a Group Access Token in Gitlab.**

On the left sidebar, select Search or go to and find your group.

Select Settings > Access tokens > Add new token > Enter a name > expiry date > select a role

> scopes > Create group access token.

**NB. To use gitlab repo in Jenkins we need to install gitlab pluggins in Jenkins.**

**Dashboard > Manage jenkins > Plugins > Available plugins > > >**

**Gitlab token:** **PetStoreAPI -** glpat-ahKqtyPKWnsKrQsRbBCC

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Git basic commands \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

git init (will initialize emply repo(.git) with a branch(master) in the specified folder.

git add file name or [git add . (. means all files)] (will add files to stage area)

git commit -m "commit message" (will save the staged files into local repo)

git status (display the state of add + commit commands)

git branch branch-name (create new branch)

git branch -m new-branch-name (rename branch) 1st checkout to a branch then use this command.

git branch (List all of the branches in your repository (local + remote))

git branch -a (List all remote branches.)

git branch -d branch-name (delete the branch)

git checkout branch-name (switch to branch-name)

git checkout –b branch-name (create & switch to branch-name)

git fetch branch-name (download content from remote to local repo)

git pull remote\_repo\_url (fetch & merege new commits from remote to local repo)

git clone remote\_repo\_url (create a copy of the repo into same/different directory/place)

git remote add origin github-repo-url (make connection initiated repo to remote repo)

git push -u origin master (upload the initiated origin content to remote repo master branch)

git merge branch-name (will merge different branches into a single branch(main/master)

PetStoreAPITokenGithub : ghp\_53p0yTFOr083jNbMMUfbcEwVPkqXu90QX9iF

\*\*\*\*\*\*\*\*\*\*\*\*\* commands from github repo \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

or create a new repository on the command line

echo "# Test-Repo" >> README.md

git init

git add README.md

git commit -m "first commit"

git branch -M main

git remote add origin https://github.com/Jawad-Karim/Test-Repo.git

git push -u origin main

push an existing repository from the command line

git remote add origin https://github.com/Jawad-Karim/Test-Repo.git

git branch -M main

git push -u origin main