**API Testing Introduction**

* We will perform API testing at the back end side, so, it is one of the back end testing.
* To perform API testing we use SOAP, RestAssured, Postman, etc.
* API Testing we can do manually as well as automation, to perform API testing manually we use POSTMAN.

To perform API testing through automation we used RestAssured (in restassured we will try to automate rest api’s, to automate api using restassured pre-requisites is we have to know java programming basic, basic of testing framework).

**Client** – Client is a computer from where we try to access the application.

**Server** – where exactly we are getting all those pages , Actual web application is installed, Through browser we try to access different application / pages,

* Between client and server we need connection throught internet we establish the connection between client and server.
* Some organization keep server at remote location so through the URL we able to access it.

**Example:**

* If I send some request ro server in the form of “URL” then server will give us the response in the form of data.

**Client Server Architecture:**

There are 3 types of Architecture

* 1-Tyre Architecture
* 2-Tyre Architecture
* 3-Tyre Architecture

1. **1-Tyre Architecture**

* We have single system, we do not have any internet we have a single system where the client will be running,
* When we run client and server in same machine its comes under 1-Tyre Architecture

1. **2-Tyre Architecture**

* We have multiple client computers and they are accessing the database & this database will be there in some other machines
* Different client application will use the same database.
* Example: Bank

1. **3-Tyre Architecture**

* This is divided into 3 parts
* Client tire- means we
* business logic – should be business login is running , it will take request from the client & same request sent to database & fetch the data again sent to business logic & business logic send response to the client.
* database tier – actual data is stored here

**What is API**

* Application Program interface is the way of communication between two application where those application will be developed in different platform & those application having different functionality & but we will combine those app and we can communicate each other by using API.
* API is a way of communication between two layers.
* API is present inside the middle layer (Business logic layer)
* A – which contains some functionality and which is assisting to the application
* P – Contains a programming logic
* I – is acting as a mediator are acting as an interface

**Types Of API**

1. Simple object access protocol (SOAP)
2. REST (Representational state transfer)

**Simple object access protocol (SOAP)**

* Different types of api avaialable in the market SOAP is old one.
* It support XML type of data.

**REST (Representational state transfer)**

* Latest use.
* It support XML, json, html, text different types of data
* Both SOAP & Rest are the web services.

**API & WebServices**

* API take request from us & send some response.
* When we keep this api on the internet is called we services.
* ALL webservices are API’s only.
* But all api’s are not web services because until we keep this in the internet we cannot say that as a web service
* Also without having internet also we create out api’s , for design and develop api internet is not required.
* When the api is available for public that time e will call them as a web service.
* So web service and api technically they are same but once we put this API in the internet that is called as web service.
* Web service use the HTTP protocol.
* A web service needs a network while an API doesn’t need a network.

**Rest API HTTP Methods**

Rest api having different types of methods but maily we are use only four methods.

1. GET
2. POST
3. PUT
4. DELETE

Those methods are API method or HTTP request.

What we do on the eb whatever application we are accessing, whatever transaction we’re doing all the transaction all the operations will comes under the at least one type of request.

* **GET:** It's like asking for information. You use it when you want to see something on a webpage, like checking the weather online.
* **POST**: It's like sending information. You use it when you fill out a form on a website, like submitting a comment or making a purchase.
* **PUT:** It's like updating something. You use it when you want to change information on a website, like editing your profile.
* **DELETE:** It's like removing something. You use it when you want to delete a message or a file from a website.

**HTTP VS HTTPS**

* HTTP : hypertext transfer protocol.
* HTTPS : hypertext transfer protocol more secure communication will happen
* These two different types of protocol used by web applications

Example : suppose when we’re sending the data through the client app to the server our data will be transferred as it in the original format, there is a chance of hacking the data so they are not more secure.

But if our web application is using https & the data is more secure so when we send some request or data which will be converted into encrypted format & throught out the network it will reaches to the server in encrypted format ,that will more secure.

**Terminologies**

1. URI – uniform resource identifier
2. URL – uniform resource locator
3. URN – uniform resource name

**Example :** [**https://google.com/articles/articlename**](https://google.com/articles/articlename)

URL/URI -- <https://google.com/articles/articlename>

URI -- google.com/articles/articlename

URN/Resource – articlename

Scheme/Protocol – https://

HOST – google.com

PATH/EndPoint -- [articles/articlename](https://google.com/articles/articlename)

**Feature & Resource & Payload**

**Feature** –

* Is the term used in manual testing to test some functionality

**Resource** –

* is the term used in API automation testing referring some functionality.
* Something is there is the & we able to access it is called Resource

**Payload** –

* Payload means body in the HTTP request **& response message.**
* The data which we are sending along with the request & also data which is coming along with response called payload.
* Payload is nothing but data.
* Request payload
* Response payload
* Some types of reauest we need to send payload but some types of request not need payload.

**Sample API Websites**

**Reqres.in**

* Here different types of API is available
* HOST -- <https://reqres.in/>
* EndPoint -- [/api/users?page=](https://reqres.in/api/users?page=2)2
* URL -- https://reqres.in[/api/users?page=](https://reqres.in/api/users?page=2) 2
* When we paste above URL then all the informationwe will fetch.
* Only GET request is possible throught browser for other request we have another application like postman.

**Postman**

* Postman is api testing tool.
* We can do manual testing of api’s using postman.
* Rest & Soap both are supported in postman.
* Postman tool is supported both desktop and web version, both having same kind of UI and options.
* If we have desktop version download and install it which will be there permanently
* If it is a web version you have to access postman through URL.
* What ever changes will do on desktop version that automatically reflect on web version.

**Steps:**

* Download & install the Postman.
* Create an account through your mail id.
* After installation we need to create workspace (Workspace is area where we maintain all the things /files / test cases, everything will be saved in workspace)
* We can create, add , delete. Edit the workspace.
* Collection- Collection contains number of folders & HTTP request. Collection is a folder where we can store all the test cases & different types of HTTP request.
* Inside one workspace we can create n not of collection, We can create, add , delete. Edit the collection.
* If you want to run all test cases together, click on the 3 dots present inside your collection and select run collection.
* Suppose you want to run all the collection simple run your collection.
* Without creating any collection we can not create any HTTP request, so save & execute get, post ,put, delete request we need collection.
* Suppose you want to export collection or save it as file then just click on 3 dot which is infront of collection and select export option then select recommended version & then save it.

**Whenever we send any request then we need to do validation on**

* **Status code** – if your request is successful we will get a 200 status code.
* How much **time it is taken** to get the response
* How much **data we capture**
* How much data **size of data** we have got
* **Body** of the response , content in the response we need to validate.
* Need to check **content** comes in which format XML or JSON.
* Need to validate **Cookies**
* Need to validate **Headers**
* Need to validate **Test result** – whenever we add some validation & validation is passed then you’ll get the test results passed, if the validation got failed then you will get the failure messages.

Cookies is nothing but temporary files which are created by the browser which contains the data similarly whenever you send some request the sere r will give some response & the cookie is also part of the response, so what kind of cookies they have created those things we have to validate.

**HTTP Status Code**

* HTTP status code is very important at the time of validation so based on the status code we will check whether our, request is done or not.
* There are different types of status code:
* All the status code are divided into three different levels/series
* 200 – Successful Request
* 400 – Unauthorized access of API’s
* 500 – Server Error

|  |  |  |
| --- | --- | --- |
| **200** | **400** | **500** |
| 200-Ok | 400 – Bad Request | 500 –Internal Server Error |
| 201- Created | 401 – Unauthorized | 501 – Not Implemented |
| 202 – Accepted | 403 – Forbidden | 502 – Bad Gateway |
| 203 – Non Authoritative info | 404 – Not Found | 503 – Service Unavailable |
| 204 – No content |  | 504 – Timeout |

**How To Create Our Own API**

To create our own api w ehave to install a few software & then we wil able to create our oun api’s.

**Install below S/W**

* NodeJS
* NPM- node package manager (This will automatically come along with the node.js because we need this component to to run or to create our own api)
* Json-server (by using this npm we need to install json-server)

**How to download NodeJS**

1. Go to google and type download node js select first link : <https://nodejs.org/en/download>
2. Click on windows version
3. After download click on that for installation
4. There is one check box comes no need to select that checkbox.
5. Click on install

**Set up Environment Variable:**

1. Go to Environment Variable
2. Click on path select Edit
3. Check two node js path automatically added there or not if node js path is not added add it.

**Need to execute some commands through command prompt**

1. Open cmd
2. node - -version (check version of node js)
3. npm - -version (check version of npm)

**Install Json server**

1. Open cmd
2. To install json server we have to execute one command in the cmd
3. npm install –g json-server

**Create Student data in Json Format:**

{

"Students": [

{

"id": 1,

"name": "John",

"location": "India",

"phone": "1234567890",

"courses": [

"java",

"Selenium"

]

},

{

"id": 2,

"name": "kim",

"location": "US",

"phone": "3456789876",

"courses": [

"python",

"appium"

]

},

{

"id": 3,

"name": "smith",

"location": "canada",

"phone": "9876543213",

"courses": [

"c#",

"RestAPI"

]

}

]

}

* Currently we have 3 records in student data.
* Extension of file should be .json
* What ever we are doing add, update, delete that will be happen in this particular file.
* We need to create json file keep this in any location.
* Once we have json file
* Then we can create our own api based on this Json files.
* Once you have this json file then we can create our own api based on this json files.
* Just go to your file location and then open the cmd
* At the search bar write cmd
* Write some command on cmd
* **Json-server filename.json**
* Then you get the path & resources, and our API will start running don’t close that cmd window just minimize it, when we close cmd then API will stop running.
* Copy the resources/url path from there & open the browser & paste it , then you will get list of all user which is in your json file.

<http://localhost:3000/Student>

* We can run same/dummy API in postman tool.
* By using this url we can perform get,post,put, delete operations.

1. **Get Single Student Data:**

* URL - <http://localhost:3000/Student/1>
* For get request no need to send request payload.
* 200 is status code

1. **Get All Student Data:**

* URL - <http://localhost:3000/Student>
* For get request no need to send request payload.
* 200 is status code.

1. **Create new Student Data:**

* URL - <http://localhost:3000/Student>
* For create new request need to send request payload.

{

"name": "smith",

"location": "canada",

"phone": "9876543213",

"courses": [

"c#",

"RestAPI"

]

}

* When we create new data no need to pass id bcz id will generate automatically by server.
* 201 is status code.

1. **Update Student Data:**

* URL - <http://localhost:3000/Student/4>
* For update student data we need to pass student id, which student we have to update specify in the request body.

{

"id": 3,

"name": "Sujata",

"location": "canada",

"phone": "9876543213",

"courses": [

"c#",

"RestAPI"

]

}

* 200 is status code.

1. **delete Student Data:**

* URL - <http://localhost:3000/Student/4>
* No need to pass payload , we nwwd to pass stu id, which student we want to delete.
* 204 is status code.

**How to Import Collection:**

* Go to poatman.
* Click on import.
* Click on upload files
* Select your file/collection.

**JSON – Java Script Object Notation**

* Whenever your communicating between client & server, the data will be communicated through Json format .
* Json is data format, we will send data only in this format , we cant send the data in the text, XLX, PDF, HTMl format because these are very high weighted data.
* Whenever we are transporting the data it should be low weighted data.
* Json is a syntax for storing & exchanging data between client & server.
* Read the data from json is easy to understand.
* It is derived from **JavaScript**.
* The filename extension is **.json**
* **Application/json(media type)** is representing the json file.

**JSON Data Types**

* Number
* String
* Boolean
* Null
* Object (contain the multiple records)
* Array

In JSON we have to represent data in the form of **key & value** pairs.

Key:value

**Data Types**

1. **Number**

{“age”:30}

1. **String**

{“name”:”Sujata”}

1. **Boolean**

{“safe”:true}

Json value true or false.

1. **Null**

{“middlename”:null}

1. **Object (contain the multiple records)**

{

“emp”: {“name”:”Sujata”, “age”:30,”city”:”Pune” }

}

1. **Array**

**{**

**“emp”:[“sujata”,”suresh”,”Shri"]**

**}**

**JSON Syntax**

* Data should be in name / value pairs.
* Data should be separated by commas.
* Curly braces should hold objects.
* Square brackets hold arrays.

**Example:**

{

"Students": [

{

"id": “01”,

"name": "John",

},

{

"id": “02”,

"name": "kim",

"location": "US",

},

]

}

**JSON Vs XML**

|  |  |
| --- | --- |
| **JSON** | **XML** |
| Json is simple to read & write | XML is less simple as compared to JSON |
| It also Supports Array | It doesn’t support array |
| JSON files are more human readable than XML | XML files are less human readable |
| IT supports only test ans number data types. | It supports may dtata types such as text, number, images, charts, graphs |

**JSON Object & JSON Array**

Json object holds key:value pair, each key is represented as a string in JSON & values can be of any type.

The key & values are separated by colon, Each key/value pair is separated by cumma.

The Curly brace { represents JSON object.

**Example**:

{

“employee”:{

“name”:”sujata”,

“Sal”:10000

}

}

**JSON Array**

* JSON array represents ordered of values.
* JSON array can store multiple values it can store string, number, Boolean, object in json array.
* In json array value must be separated by cumma.
* The square [ ] represents json array

**JSON path**

* Suppose I want to extract name of the 1st student, from our json file.
* If we want to extract name then we have to star from beginning.

Students[0].name

Students[2].id

* Some time it is difficult to find out json path , so there are two tools are available
* <https://jsonpathfinder.com/>
* <http://jsonpath.com/>

1. <https://jsonpathfinder.com/>

Click on one of the Above link / open one of the Above link

It will automatically generate json path.

* Copy & paste our json file,
* Other side automatically it will generate an json path.
* If we expand the student the it will automatically generated XML path.
* Suppose if we want to get json path of “phone” from forst student then other side at array 0 click on “phone” field , automatically path will be generated for that “phone” filed.
* In path ‘X’ is just representation of root node, no need to copy X when we copy the json path.

Students[0].phone

1. <http://jsonpath.com/>

* It used to validate what evre json path we identifying is correct extracting data from json or not.
* Paste the Jason path which we capied from jsonpath finder & paste it in JSONPath field then in evaluation result it will shows exact value for that json path.

**Response Validation**

Below are the response validation

* Status code
* Headers
* Cookies
* Response Time
* Response Body

Once we send request & once we get a response from the request & before sending the request and response we have to know what type of response we will get, status, cookies, header

Assertion is nothing but what which is a validation point, so to add this assertion in postman there is one library PM here so many functions are available, so by using this function we can add validation point.

**There are 2 types of function**

1. **Normal function**

**Syntax**

Pm.test(“Test Name”, function()

{

//assertion;

}

);

1. **Arrow Function**

**Syntax**

Pm.test(“Test Name”, () =>

{

//assertion;

}

);

**Testing Status Codes :**

* To test the status code in postman go to any request there are some tags like params, authorization, headers, body, pre-request script, tests, settings.
* Click on Tests inside that test we have to write all the validation functions.

**Validate the status code**

pm.**test**("Status code is 200", () **=>**

{

    pm.response.to.have.status(200);

});

**Check the status code text:**

pm.**test**("Response body has string 'created'", () **=>**

{

    pm.response.to.have.jsonBody('name', 'created');

});

* After that save the file & click on send
* Inside the test results we can see the result pass or fails.

**Testing Headers**

* When we send any request, we got some headers in the response.
* In the headers it contains a lot of info about many things, data or n/w info, server info, Content-

Type, Content-Length.

* We need to check every heads\r is having proper value.
* We need to check

Content-Type or Content length (response is in which format json, xml)

* There are some options which are dynamically changing which we can not predict those values like cache control, keep alive.

**There are two validation we have to do on headers**

* When we send request get the response first we need to check all this header is present or not (presence of header).
* Other is value of header

**Syntax to verify headers using content type & content type value**

**//Header presence**

pm.**test**("Content-Type header is present", () **=>** {

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

});

//Value of the heades

pm.**test**("Content-Type header is application/json", () **=>** {

    pm.expect(pm.response.headers.**get**('Content-Type')).to.eql('application/json; charset=utf-8');

});

**Testing Cookies**

Cookies value keep on changing & are dynamic bcz when we send request we may not expect the same values for the cookies but there are some cookies their values are same sometimes they will get the same value for the cookie.

Once we send request there is one tab under this response.

Each cookies having some value, domain, path, expires , Source some details will be there for that particular cookie.

We need to verify the presence of cookies & their values.

pm.**test**("Cookie 'AEC' is present", () **=>** {

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

});

pm.**test**("Cookies AEC has value 'en-gb'", () **=>** {

    pm.expect(pm.cookies.**get**('AEC')).to.eql('Ad49MVFnQK2APdwKG6dsHLrpHwa7zhtd6OH99ZedaQuh2cNC3A9nhN-Y3tY');

});

**Testing Response Time**

* 353 ms is response time.
* Depends on complexity of data this time keep on changing.

pm.**test**("response time is less than 200ms", () **=>** {

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

});

**Response Body**

* Response is keep changing is depends upon what kind of request you send.
* JSON path is important for response body.

**Below are the validation for response body:**

1. **Asserting a value type**

We need to validate type of value **“id”:1** here need to validate id is integer type , we have to validate type of field or data.

**Example**:

**/\***

**""id": 1,**

**"name": "John",**

**"location": "India",**

**"phone": "1234567890",**

**"courses": [**

**"java",**

**"Selenium"**

**\*/**

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

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

    pm.expect(jsonData).to.be.an(“object”);

});

pm.**test**("Response should have the correct name", ()=> {

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

});

pm.**test**("Response should have the correct location",  ()=> {

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

});

pm.**test**("Response should have the correct phone number", **function** () {

    pm.expect(jsonData.phone).to.be.a("1234567890");

});

pm.**test**("Response should have the correct courses", **function** () {

    pm.expect(jsonData.courses).to.be.an(“array”);

});

**Asserting Array Properties**

Suppose I have an array in body , I have to check this array contains these values exactly same values or not, I want to verify test array properties.

**/\***

**""id": 1,**

**"name": "John",**

**"location": "India",**

**"phone": "1234567890",**

**"courses": [**

**"java",**

**"Selenium"**

**\*/**

**//test array contents in body**

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

**pm.test("Test array properties", () => {**

**//single array value validation**

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

**// multiple array value validation**

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

**});**

**Validating JSON Field in Response**

**/\***

**{**

**""id": 1,**

**"name": "John",**

**"location": "India",**

**"phone": "1234567890",**

**"courses": [**

**"java",**

**"Selenium" ]**

**}**

**\*/**

pm.**test**("Value of location field is India", () **=>** {

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

    pm.expect(jsonData.id).to.eql(1);

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

    pm.expect(jsonData.phone).to.eql("1234567890");

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

    pm.expect(jsonData.courses[1]).to.eql("Selenium");

});

**Validating JSON Schema**

Schema is a structure.

Response:

**{**

**""id": 1,**

**"name": "John",**

**"location": "India",**

**"phone": "1234567890",**

**"courses": [**

**"java",**

**"Selenium" ]**

**}**

**JSON Schema**

* Schema is describing where or the type of data which we can store in the json file & whatever the type we specified in the json schema same type of data we can store in the json file.
* Here we have to check our response is following our schema or not.
* Goto google search for json to jsonschema convertor
* Paste your json code there automatically json schema will be displayed

{

  "$schema": "http://json-schema.org/draft-07/schema#",

  "title": "Generated schema for Root",

  "type": "object",

  "properties": {

    "id": {

      "type": "integer" // Change "number" to "integer"

    },

    "name": {

      "type": "string"

    },

    "location": {

      "type": "string"

    },

    "phone": {

      "type": "string"

    },

    "courses": {

      "type": "array",

      "items": {

        "type": "string"

      }

    }

  },

  "required": [

    "id",

    "name",

    "location",

    "phone",

    "courses"

  ]

};

**JSON Schema Validation**

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

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

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

});

Write code in post man like

// JSON Schema validation

var schema **=** {

  "$schema": "http://json-schema.org/draft-07/schema#",

  "title": "Generated schema for Root",

  "type": "object",

  "properties": {

    "id": {

      "type": "integer" // Change "number" to "integer"

    },

    "name": {

      "type": "string"

    },

    "location": {

      "type": "string"

    },

    "phone": {

      "type": "string"

    },

    "courses": {

      "type": "array",

      "items": {

        "type": "string"

      }

    }

  },

  "required": [

    "id",

    "name",

    "location",

    "phone",

    "courses"

  ]

};

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

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

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

});

**Postman Scripts**

We have two types of scripts:

1. Pre-request Script
2. Tests

Go to postman open any request beside the “Tests” there is one option “Pre-request Script”, there are we can write the scripts.

When we put any scripts in “Pre-request Script” it will be executed before starting the request , before & after the execution.

Suppose am sending some request after then I will get some response & before sending the request “Pre-request Script”

**Pre-request Script-🡪Request🡪Response🡪Tests**

Above is sequence for execution the request.

“Pre-request Script” & “Tests” available in three different levels.

* Collection level
* Folder Level
* Request level

**Sequence of request execution**

When we specify request at 3 level the execution sequence will be:

**Collection level-🡪Folder level🡪Request level🡪**

**Reuest🡪Response🡪**

**Collection level-🡪Folder level🡪Request level**

* Create one collection inside postman, create one folder inside that collection, add the request inside the folder.
* When we select our created folder 2 option’s will be showing there “Pre-request Script” & “Tests”.
* When we select our collection 2 option’s will be showing there “Pre-request Script” & “Tests”.
* So inside the collection folder & request add one line

**Add at Pre-requsites:**  
Pre-request script at collection level

Pre-request script at Folder level

Pre-request script at Request level

**Add at Test**

Test script at collection level

Test script at Folder level

Test script at Request level

**Then Run your collection.**

At the buttom of postman there is console window.

**Output**

Pre-request script at collection level

Pre-request script at Folder level

Pre-request script at Request level

▶GET https://www.google.com/200

9.81 s

Test script at collection level

Test script at Folder level

Test script at Request level

**What is use of “Pre-request Script” & “Tests”.**

The “**Pre-request Script**” will be executed before sending the request , some times there will some pre-requisites before sending the request.

When sending post request need to pass request body,

**Test** will execute after Sending request.

**Types of Variables in Postman**

**What is variable**

Variable is nothing but which can store some data.

**Why we need variable**

To avoid duplication, rework

Some times we have some repetition , so we have some data which is required for every type of request

Example-URL

**Where we use variable.**

Some times e refer the variable inside the body also, we can create variable at different levels collection,folder& request level.

We can create variables at following levels / **Scope of variable**

* Global
* Collection
* Environment
* Local
* Data

**Variable Types**

**Default Type**

**How to create Global variable**

Global variable we can access every where, throughout the workspace, collection, folder, request.

* Go to Postman
* There is one option next to “no Environment field “View” click on that.
* There is 2 section “environment” & “global”.
* Click on add button infront of global variable.

Provide the value

Variable- url\_global

Initial\_value : <https://reqres.in>

Current value : https://reqres.in & save it.

**{{url\_global}}/api/users?page=2**

Write URL like above inside the curly bracket write our variable name.

Collection variable

It is accessible within collection only,

If we create variable at collection level , then multiple request which are created within one collection there we can use it.

Select Collection

Click on 3 dots

Click on edit

There is variable part

Then provide the values:

Variable- url\_collect

Initial\_value : <https://reqres.in>

Current value : https://reqres.in & save it.

**Environment variable**

An environment is a set of variables that allow you to switch the context of your requests.

Suppose I want to execute all request in one specific environment QA, DEV, PROD.

Every environment there is different values and url is there.

Suppose I want to execute collection / my all request in QA environment then I use the variables.

We have to create our variable at environment level, & we have to use whichever environment we want to execute.

So in every environment QA, DEV, PROD there is variable we have to switch our environment occurding to our need.

* Go to Postman
* There is one option next to “no Environment field “View” click on that.
* First we have to create the Environment then create the Environment variable.
* Write the Environment name like QA, Dev, PROD & save it.
* Switch to QA environment
* Then provide the values:

Variable- url\_qa\_env

Initial\_value : <https://reqres.in>

Current value : https://reqres.in & save it

* When we send any request that time we need to select environment variable.

Like this add environment variable for PROD,DEv

**Local Variable**

We can create at request level or local variable we can create a request level.

* Go to postman, select any request
* Inside the Pre-request Script we can define the variable.
* Write some scripts before sending request local variable will be created & that variable will be used in the request at run time.
* pm.variables.**set**("url\_local","https://reqres.in");
* We can access this local var only for that particular request. For other request it is not possible because this is only for this particular request.
* If you want to go for another request we have to create local variable for that request.

**Creating variables using pre-request scripts:**

We can create local , global, environment variable by using pre-request scripts at he run time those variables will be created

After execution is completed we can also unset those variables or we can also remove those variable

//global variable

pm.globals.**set**("userid\_global","2");

Got to view we can seen after sending request global variable will be created.

How to use in URL : **{{url\_local}}/api/users?page={{userid\_global}}**

//Environment Variable

pm.environment.**set**("userid\_qa\_env","2");

//collection variable

pm.collectionVariables.**set**("userid\_collect","2");

//local variable

pm.variables.**set**("url\_local","https://reqres.in");

**How to Unset /Remove Variables**

* Go to The request
* Suppose after using global variable you want to delete it.
* When our request get executes variables are created inside the “View”
* So if you want to create global variable , use it & delete it after completion of my request.
* We have to add one more line inside the scripts.
* So we have to write the scripts inside the “TESTS” for remove variables.
* Instate of “Set” use “unset”.

//global variable

pm.globals.unset("userid\_global");

//Environment Variable

pm.environment.unset("userid\_qa\_env");

//collection variable

pm.collectionVariables.unset("userid\_collect");

Go to views we can see all the variables removed from there.

**How to get data inside the TESTS or Pre-request Scripts**

**Suppose I want to print the data before remove the variable so inside the “Tests” write scripts**

//Get Value Of Variables

console.**log**(pm.globals.**get**("userid\_global"));

console.**log**(pm.globals.**get**("userid\_qa\_env"));

console.**log**(pm.globals.**get**("userid\_collect"));

console.**log**(pm.globals.**get**("url\_local"));

//remove data from variable

//local variable

pm.variables.unset("url\_local");

//global variable

pm.globals.unset("userid\_global");

//Environment Variable

pm.environment.unset("userid\_qa\_env");

//collection variable

pm.collectionVariables.unset("userid\_collect");

**also inside the console output wil be displayed**

**For the variables we use set, get & unset methods.**

**Data Variable**

We have to create data variable in external files in the sense what it can be CSV file or can be text file or Excel Sheet somewhere & we will retrieve them & use in our request

This will be used in the data driven testing

Data which is giving from the external file and will use the same data as part of our request as per our HTTP request is called data driven testing

**Chaining Of API’s**

When we send any api request we will get some response, this particular response we will use as input for another type of request.

When we get response this response we have to store in different variables, once we store this in the variables & we can use those variables as part of the request.

Create the user http://localhost:3000/Students

Get the data http://localhost:3000/Students/{{id}}

Delete user http://localhost:3000/Students/{{id}}

//extract the data

var jsonData**=**JSON.**parse**(responseBody);

// capture id

pm.environment.**set**("id",jsonData.id);

use above script to get the id

From the create request extract the id & use are request to another method

**Gorest API**

URL -- <https://gorest.co.in/> website

**Endpoints:**

POST /public/v2/users Create a new user

GET /public/v2/users/2901 Get user details

PUT|PATCH /public/v2/users/2901 Update user details

DELETE /public/v2/users/2901 Delete user

Token : 5214c15243e4a0c2fd221e2a54a177fa98d925d36a1a1a79f083de9b73623681

We need token number to access this API

Local api –need to run json server

Global api- we can directly run

To access above url we needs to generate on token that we have to pass as part of authorization called Bearer Token.

Bearer Token is one of the type of authorization so we need to create the token then only we will able to send the request then only we will able to use this particular API.

* First sign up the above link.
* If you have git account click on that add uname & password
* Enter the security code which will received in your mail
* Then automatically token will be generated 5214c15243e4a0c2fd221e2a54a177fa98d925d36a1a1a79f083de9b73623681
* There is limit for that token 90 we can send 90 request in a day only,.

Steps:

1. First create all the request using above url & endpoints
2. We have to pass token for all request , some time it is difficult to pass token to all request or change the token
3. So we set token as collection level variable it is easy to access everywhere inside the collection.
4. Click on collection there is one tab authorization
5. Select type bearer token & paste your token & save it.
6. Also don’t add token inside any request.
7. If we add token inside request that will be applicable for that request only, but if we add token at collection level that will be applicable for that all request or for whole collection.

How to generate name & email randomly

* Before sending the post request we have to write the script.
* That script automatically generate name & email address
* Write this scripts under the pre-request script

// Generate a random string

var randomString **=** Math.**random**().**toString**(36).**substring**(2);

// Generate a random email address

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

// Generate a random name

var userName **=** "jim" **+** randomString;

// Add the above data to environment variables

pm.environment.**set**("email\_env", userEmail);

pm.environment.**set**("name\_env", userName);

ADD Body

{

    "name": "*{{name\_env}}*",

    "gender": "female",

    "email": "*{{email\_env}}*",

    "status": "active"

}

How to extract id from response

We have to add scripts inside the “Tests” tab

//extract entire body

var jsonData**=**JSON.**parse**(responseBody);

//get id & create environment variable

pm.environment.**set**("userid\_env",jsonData.id);

How to Use id in other request to get,update, delete user

Go to Get user

* Get request we need to add tests

GET- [https://gorest.co.in/public/v2/users/{{userid\_env}}](https://gorest.co.in/public/v2/users/%7b%7buserid_env%7d%7d)

// Validating JSON fields in the response

pm.**test**("Values of JSON field", () **=>**

{

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

    // Check if name, email, and id are correct

    pm.expect(jsonData.id).to.eql(pm.environment.**get**("userid\_env"));

    pm.expect(jsonData.email).to.eql(pm.environment.**get**("email\_env"));

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

}

);

**Updateuser**

PUT <https://gorest.co.in/public/v2/users/{{userid_env}}>

{

    "name": "*{{name\_env}}*",

    "email": "*{{email\_env}}*",

    "status": "active"

}

**Delete User**

Delete- [https://gorest.co.in/public/v2/users/{{userid\_env}}](https://gorest.co.in/public/v2/users/%7b%7buserid_env%7d%7d)

//unset variable

pm.environment.unset("userid\_env");

pm.environment.unset("email\_env");

pm.environment.unset("name\_env");

lca.lufthansa.com