# **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.

# **API**

API stands for Application Programming Interface. API acts as an interface between two software applications and allows them to communicate with each other. API is a collection of software functions that acts as a bridge between 2 applications.

Types of API Testing: Functional, Integration testing(with UI), Security, Load, Performance, Regression

# **Difference between Web Service and APIs**

|  |  |
| --- | --- |
| **WebService** | **API** |
| Web services are a type of API, which must be accessed through a network connection or an API wrapped in HTTP is Web Service | API acts as an interface between two software applications and allows them to communicate with each other |
| WebService needs a network for its operation | API doesn’t need a network for its operation |
| All Web services are APIs | APIs are not web services. |
| Web service uses three styles: REST, SOAP, and XML-RPC for communication. | API can be used for any style of communication. |

# **Main types of errors in API testing**

* Missing or duplicate functionality
* Fails to handle error conditions gracefully
* Security
* Not implemented errors
* Inconsistent error handling
* Performance
* Multi-threading issues
* Improper errors

SOAP: SOAP-stands for Simple Object Access Control, and it is an [XML](https://career.guru99.com/xml-interview-questions/) based protocol for accessing the Web Service and exchanging information between computers.

REST: REST stand for REpresentational State Transfer, REST is an architectural style that defines a set of principles to be followed while designing a WS. These principles are:

Uniform Interface – you can access any resource or data by a URI (Unique Resource Identifier). Using HTTP methods like GTE, PUT, POST, DELETE, we can access or modify resource.

Stateless – server should not store state of the session, each request from client to server must contain all data necessary to handle the request.

Cacheable – Caching happens at client side.

**Soap vs REST**

* SOAP is a protocol for accessing WS and REST is architecture style
* SOAP permits XML only whereas REST supports different message formats such as Plain text, HTML, XML, JSON.
* SOAP is more secure and structured format. Since REST supports various messaging formats like JSON, CSV and XML, it does not have specific standards.
* REST services are faster and easy to handle than SOAP.
* SOAP cannot be cached whereas REST can be cached
* SOAP is tied with SMTP and HTTP protocols whereas REST relies on HTTP only.
* SOAP uses named operations and interfaces to achieve its business logic whereas REST uses Uniform Resource Identifier (URI) and it has the methods like GET, PUT, POST and DELETE to expose their resources.

# **JSON**

JSON stands for JavaScript Object Notation. JSON is a lightweight format for transporting and storing data (in the form of key-value pairs). JSON is not a programming language.

# **API Request Elements**

When sending Rest Based request, following are required: Request Body (Payload), HTTP Method, HTTP Version, URI, Request Headers

**HTTP Methods:**

* Get: Doesn’t have a request Body
* Post: As you are creating/modifying data, request body is required
* Put: As you are creating/modifying data, request body is required
* Delete: Doesn’t have a request Body
* Sending/retrieving data, creating data, updating data, deleting data
* CRUD operations – Create, Retrieve, Update, Delete

**EndPoint URI**: URIs can be used to locate and access resources on the internet.

* URI: Uniform Resource Identifier (URI)
* EndPoint URL: Base URL/resource/(query/path)parameters
* Complete URL= Base URL + Version + API Path

**Headers:**

Headers: Meta data that is associated with the API request, additional details API needs to process our request.

**Request/Response Headers: Content-Type**, **Cache-Control, Content length, User Agent, Authorization(only in request)**

Authentication token: fixed for each request. For added security

# **Parameters**

* Path Parameters: Used to point to a specific resource within a collection.

<https://amazon.com/orders/112>

Here amazon.com is base URI, Order represents a resource (API that has a collection of all orders) and 112 is a parameter which is pointing to a specific order (this points to order 112 in all orders). Rerouting to further sub-resource

* Query Parameters: used to sort or filter the resources

<https://amazon.com/orders?sort_by_2/20/2020>

Has a question mark before writing parameter name. In above case, ’sort by’ is a parameter name and date is parameter value. Here we do not re-route to sub resource. Eg from Google search:

<https://www.google.com/search?q=Santorini&rlz=1C1CHBD_enIN781IN781&oq=Santorini&aqs=chrome..69i57j0i271l2.17629j0j1&sourceid=chrome&ie=UTF-8>

&- parameter separator

# **API Response 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.

# **Environment**

Environments is a set of variables in Postman allow us to run requests and collections against different data sets. We could have different environments for Dev, QA & Production. Each of these environments will have different configurations such as URL, token’s id and password, API keys etc.

An environment in postman is a set of key value pairs. 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. They define the scope of the variable.

# **Postman Collection**

A Postman Collection is a Test Suite that lets us group our requests together. Simply it allows us to organize the requests into folders.

# **Postman Monitor**

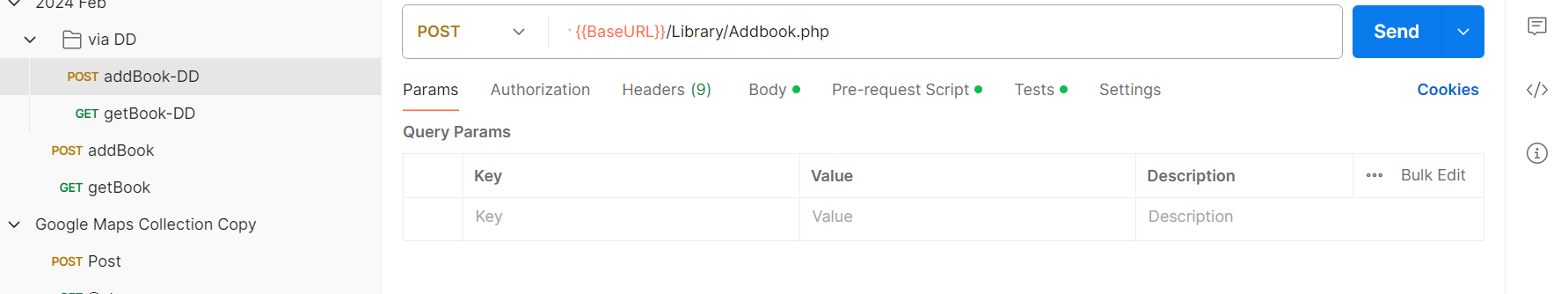
The postman Monitor is an automated way of running collections. Collections are triggered automatically as per user defined intervals. Monitors can help you schedule a collection of test runs to monitor the performance and response of your APIs even if you are not available or not handling them.

# **HTTP Status Codes**

* 200 (OK): Defines that the request was correct.
* 201 (Created): The value wrapped with the request has been created in the database. It is needless to say that the request was correct.
* 204(No Content): This status code means that the request was correct and received but there is no response to send to the client by the server.
* 400 (Bad Request): A bad request means that the syntax of the request was incorrect. It can happen if you have sent wrong parameters along with the request URL or in the body of the request.
* 401 (unauthorized request): We can incur such a status code when you are not authorized to access the server or you have entered wrong credentials.
* 404 (Not Found: A response code 404 means that the server was connected but it could not find what was requested. You can normally see this status code when you request a web page which is not available.
* 405 Method Not Allowed – HTTP method is not supported for the resource.
* 409 Conflicts – Use of PUT request for creation of same resource twice etc.
* 500 – Internal Server error – error at server side

# **Bulk Edit**

Bulk Edit feature of Postman is used for the convenience of adding parameters to a new request from the previous request. Viewable only at request level



Since a request can have many parameters and it is very difficult to copy and paste one by one, bulk edit feature helps us copy all the keys and their respective values at once and paste them.

# **What is the difference between authorization and authentication?**

Authenticationis a process of presenting your credentials to the system and the system validating your credentials. These credentials tell the system about who you are. Authorization is a process of allowing or denying someone from accessing something, once authenticationis done

# **Global variables, collection variables, and local variables**

**Collection variables** are used for constants that do not change during the execution and also for URLs / authentication credentials if only one environment exists. (Project level)

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

**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 automatically removed once the tests have been executed.

# **GUID**

Postman uses the [faker library](https://www.npmjs.com/package/@faker-js/faker) to generate sample data, including random names, addresses, email addresses, and much more. You can use these variables like any other variable in Postman.

GUID stands for Global Unique Identifier.

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

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

**const** randInt= pm.variables.replaceIn('{{$randomInt}}')

**$randomBoolean, $randomAlphaNumeric, $randomAbbreviation, $randomPassword.**

You can similarly use random names, random addresses, mobile numbers

# **What is Pre-Request Script in Postman?**

A pre request script is a script that runs before the execution of a request.

You can use it for setting variables. Or lets say you have a collection of requests that need to run sequentially. The second request depends on a value returned by the first request. In the first request’s Tests script, you set a value from the response to a variable. Then in the second request’s Pre-request Script, you process that value.

You can add pre-request scripts to entire collections or folders. It’s useful for defining common pre-processing steps that apply to multiple requests.

# **OAuth2 Authorization in Postman**

With OAuth 2.0, we first retrieve an access token for the API, then use that token to authenticate the requests. An Access Token is basically used to assure that the user is approved to access the data.

When we hit the secured endpoint without an access token, the Authentication error will occur and 401 Unauthorized status will be returned.

* The application sends an authorization request for the user to access the data.
* Once the application has granted access to the user, the application requests an access token from the server by giving the user details.
* The server then returns an access token.
* The client uses the access token to request the protected data.

OAuth/Bearer token: used by GitHub, JIRA

Basic Auth: uses username and password

APIKey: is part of query parameter. Google Maps uses API Key

Auth url – authorization server endpoint – which 3rd paty you are relying for authentication

Client id – used to identify who has requested access

Client secret – unique passcode associated with your Client id

Redirect url – after authentication from Google or FB, which site to redirect to (your application)

Grant type – authorization code, client credentials,

Scope- what info google will give you – name, mobile, email etc.

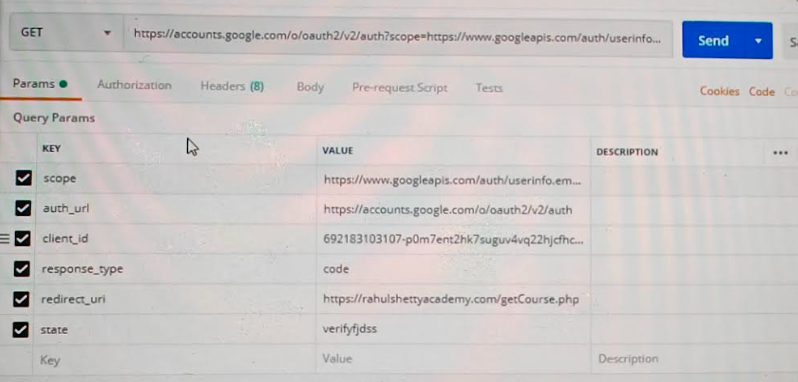
We can get these details in OAuth 2.0 Contract details

2 steps:

Request 1: GetAuthorization Code request –

* for this you need Authorization server URL
* query params – scope, auth url., client id, response type, redirect url

output of this is a code that you need to use in Step 2



Request 2: GetAccess token request –

* End Point or Access token url – get from developer
* Query Params – code, client id, client secret, redirect url

This will get you access token which you can use in your requests.

# **How To Run Collections using Newman**

Newman allows running automated tests using CLI which can also be integrated with Jenkins for CI/CD integration.

To run our collection using Newman do the following:

1. Install node.js. To check after installation, search in command prompt: node -v or npm -v

Node.js is an open source server environment. Node.js allows you to run JavaScript on the server.

2. Open the command line and enter: npm install -g newman

To check after installation, search in command prompt: newman -v

3. Go to the Postman workspace. In the collections box, click on the three dots and select Export. Save it somewhere on local machine

4. We will also need to export our environment. Click on the eye icon beside the environment dropdown in Global, select Download as JSON. Keep it in same folder as collections.

5. In cmd prompt, change the directory to where you have saved the collection and environment:

cd *C:\Users\Asus\Desktop\Postman Tutorial*

6. newman run PostmanTestCollection.postman\_collection.json -e Testing.postman\_globals.json

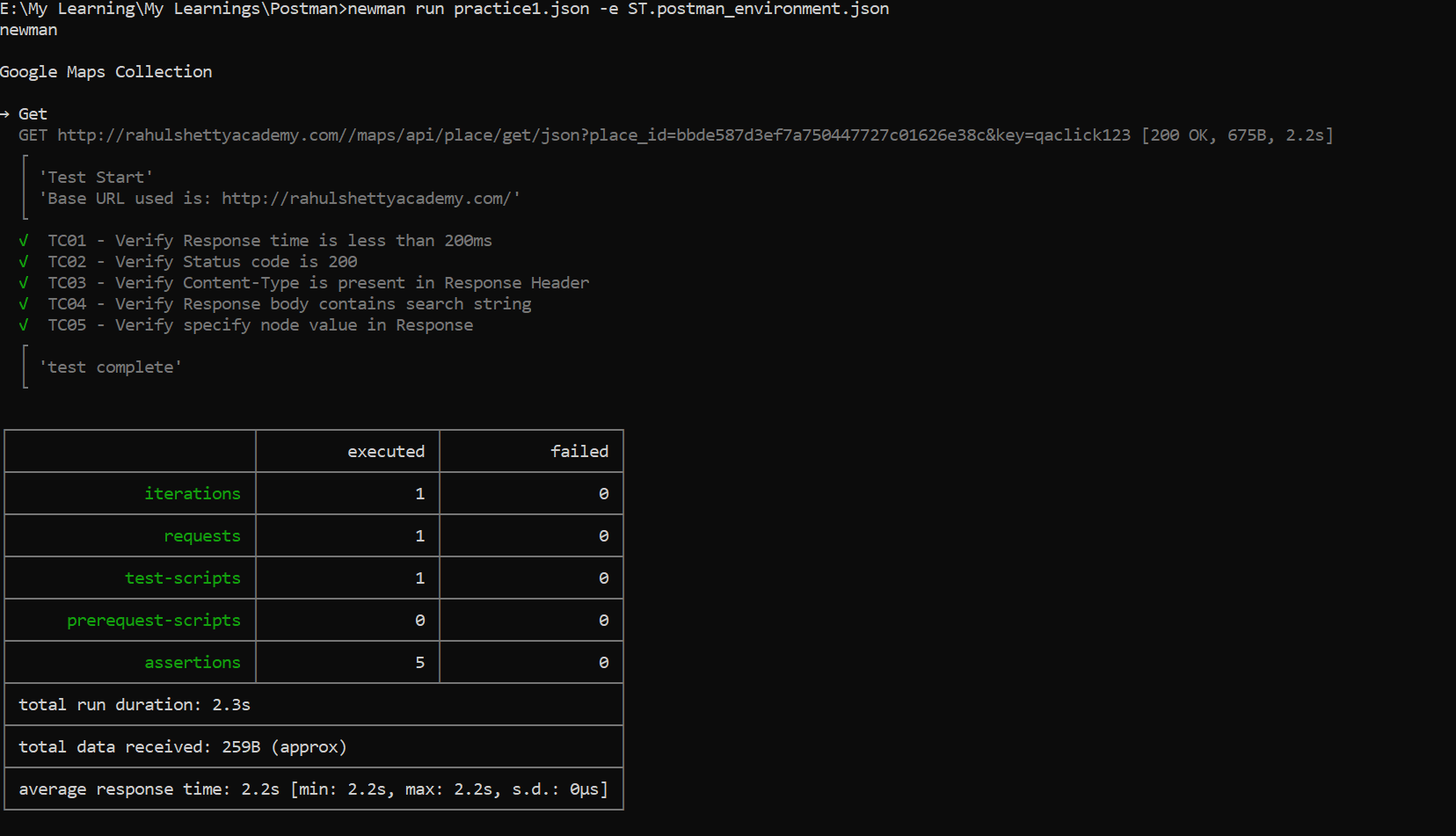
newman run {{path to collection json}} -e {{path to environment json if any}}

* Run collection using Newman with delay: newman run collection.json -- delay 10000

Method 2:

You can also run it via URL. To get the URL of your project, go to your collections and click on share collection to get your URL. You can now use this URL to run your collections via cmd. Please note that this does not export your environment but exports the collection variables.





# **Run a folder in a collection**

*newman run <collection\_name> –folder <folder name>*

# **Generate Basic HTML Reports, While Using Newman**

* Install npm dependency, run command: **npm install -g newman-reporter-html**
* To get reports with command: **newman run <exported collection path> –-r html**
* This will create a ‘newman’ folder containing your html report.

# **Generate Advanced HTML Reports, While Using Newman**

We will use **htmlextra reporter** for Newman and this is an **npm dependency** that we need to install separately.

* To install the **htmlextra reporter**, run command: **npm install -g newman-reporter-htmlextra**
* We get **htmlextra reports** with command: **newman run <url> –reporters=cli,htmlextra**

The report will be saved inside the folder where Newman was executed, inside a folder called Newman.

# **What is the order of preference scope for each Postman variable?**

Local Variables -> Data Variables -> Environment Variables -> Collection Variables -> Global Variables

LECG

# **How can you reuse your authentication token for different requests?**

We can create a Collection and add all the requests to that collection. We can add the authorization token in the collection and then we can select “Inherit auth from parent” option as authorization for every request.

# **What are the different authorization options available in Postman?**

API Requests can be authorized using the following options:

– API Key

– Bearer Token

– Basic auth

– Digest auth

– Oauth 1.0//– Oauth 2.0

– Hawk Authentication

– AWS Signature

– NTLM Authentication

# **Difference between API testing and Unit Testing?**

* Unit testing is conducted by the Development Team, API by Test team
* Unit testing is a form of White box testing, API is black box
* In unit testing, the scope of testing is limited, so only basic functionalities are considered for testing. In API testing, the scope of testing is wide, so all the issues that are functional are considered for testing

# **Postman Tests**

Postman Tests allow you to ensure that your API is working as expected.

Pre-defined scripts in postman



# **Get variables:**

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

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

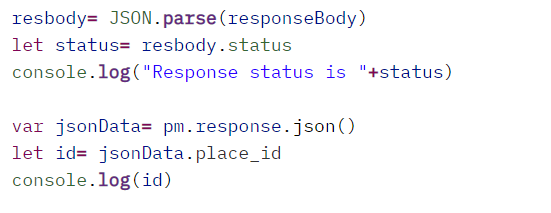
pm.variables.**get**("variable\_key");

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

You can use pm.<level>.**set**("variable\_key", "variable\_value") to set variable values.

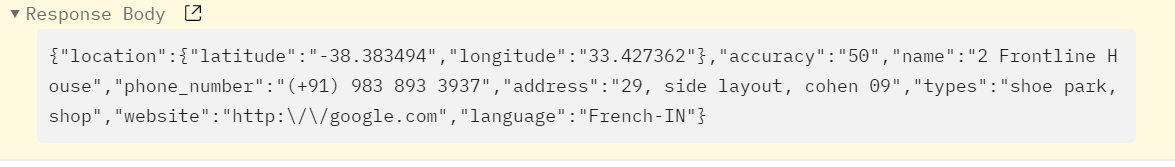
use ‘unset’ to clear variable values- pm.globals.unset("variable\_key");

# **2 ways to parse JSON body**

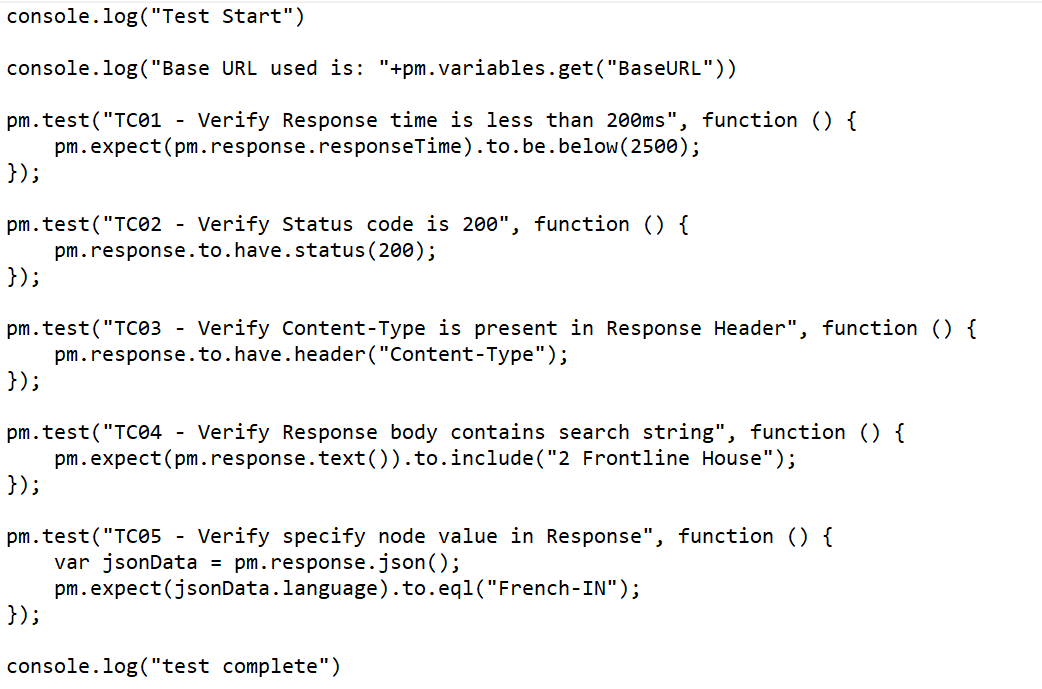


Real Time examples:

Response:



Tests:



# **How to check whether the field is returning null in the Postman automation?**

**pm.expect(jsonData.<yourfield>).to.eq(undefined);**

# **How to check if responseBody does not contain string in postman tests?**

**pm.expect(pm.response.text()).to.not.include("string\_you\_want\_to\_search");**

Check length of string

**pm.expect(jsonData.NAMTTL).to.have.length(1)**

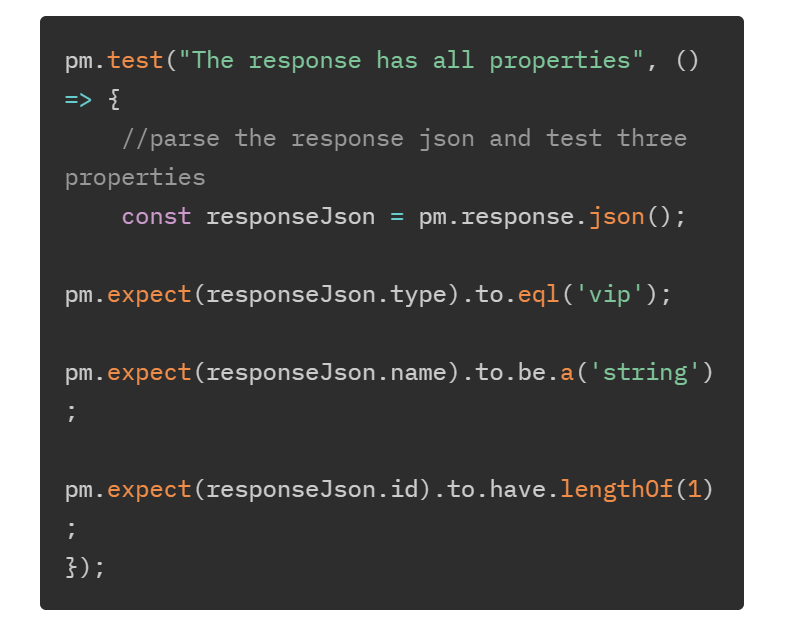
pm.test('TC09 Check value properties', **function** () {

    pm.expect(responseJSON.ID).to.be.a('string');

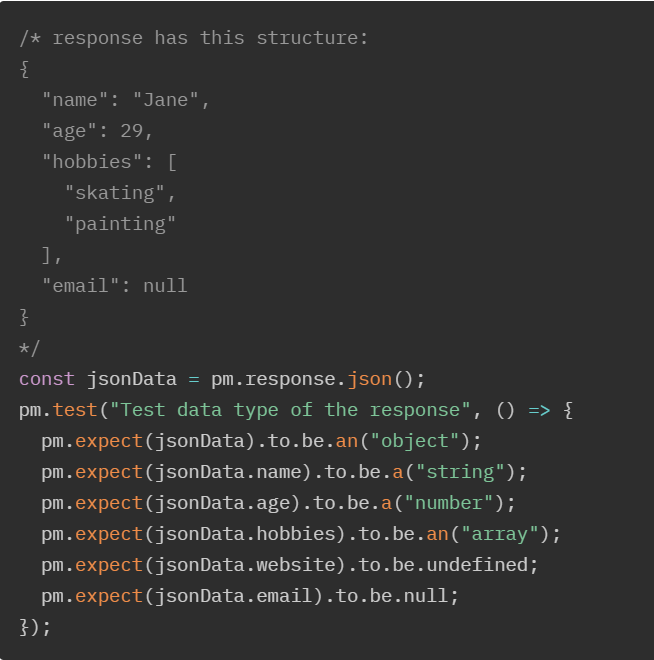
    pm.expect(responseJSON.ID).to.have.length(8);

})

# **Using Multiple Assertions**

\*length and not lengthOf

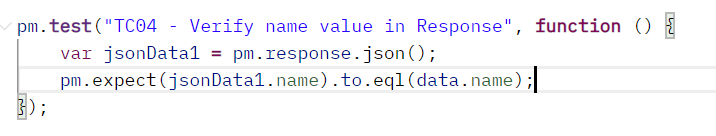
# **Asserting a value type**



# **Data Driven Testing**

To add assertion in responses while data drive testing, use **data.attributename**

Here attribute name is your column name in text/csv file.



Entire project can be found here.



Good example using data driven: use datadriver.xls to data drive

<https://www.getpostman.com/collections/693232e23004253c0b9b>

# **Notes: Test Scripts in Postman**

**pm.test():**

The pm.test() function is used to write test specifications. It accepts 2 parameters, the name of the test (as a string) and a function to return a boolean value. It can be used only in the Tests tab after the primary Postman request has been sent.

pm.test("response should be okay to process", function () {

pm.response.to.not.be.error;

pm.response.to.have.jsonBody("");

pm.response.to.not.have.jsonBody("error");

});

To check complete Response body

*var* response =pm.response.json();

* In above step, we are storing the response in json format in a variable ‘response’.

pm.test("Name match", *function* () {

    pm.expect(response.name).to.eql("Name1");

});

* Pm.expect is used for assertions.
* response.<body attributes> is used to check specific attribute
* to.eql, not.eq, to.include, to.not.include, to.contains, to.be.oneOf, to.be.below can be used as per requirements by passing input search key.
* If there are multiple search results, you can use indexing to reach specific result:

Pm.expect(response[1].body).to.eql(“abc”);

* pm.expect(response.products[0].category).to.eql('Book');
* status code : pm.expect(pm.response.code).to.be.oneOf([201,202]);
* response time: pm.expect(pm.response.responseTime).to.be.below(9);
* Header exists: pm.response.to.have.header(Content-Type);
* Something exists: pm.expect(pm.cookies.has('sessionId')).to.be.true;
* Cookie/something has value: pm.expect(pm.cookies.get('sessionId')).to.eql(’nnn');
* Partial text match in body: pm.expect(pm.response.text()).to.include('Order placed.');
* XML to JSON: const response = xml2Json(responseBody);
* Pm.iterationData.get(“value”): get the value from the data file
* To add a header in pre-reqs:

Pm.request.headers.add(

{

“key”: “Content-Type”,

“value”: “application/json”

});

* To delete header: pm.request.headers.remove(“header name”);
* To set sequence: pm.setNextRequest(request name);
* Response functions:
* Pm.response.status
* Pm.response.code
* Pm.response.headers
* Pm.response.responseTime
* Pm.response.text: for response text
* Pm.response.json: to parse through response
* If we have some special characters in property name, you cannot use “.notation” like *response.name-first.* In such case, wrap the entire property in double quotes and square brackets like*: response.[“name-first”]*
* Extracting Data From Responses and Chaining Requests:

Get the response body: var response= pm.response.json();

Set parameter value as global variable: pm.globals.set(“<var name>”, response.<attri>);

In second request, refer the variable using {{var name}}

* If we want to compare already saved variables (eg. Username) with values from another API response, we can use a method:

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

var jsonData = pm.response.json();

pm.expect(jsonData.value).to.eql(pm.globals.get("username"));

});

* If(condition) {

Do this;

}

* Pm.setNextRequest = for sequencing
* Pm.setNextRequest = null to stop collection run
* Compare 2 json responses:

In first request:

var responseExp= pm.response.json //saving response in variable

pm.globals.set(“expected response”, responseExp) //saving response as global variable

In second request

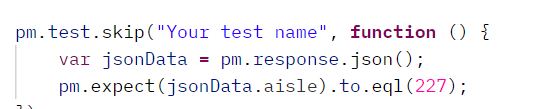
var responseAct= pm.response.json;

var responseExp= pm.globals.get(responseExp);

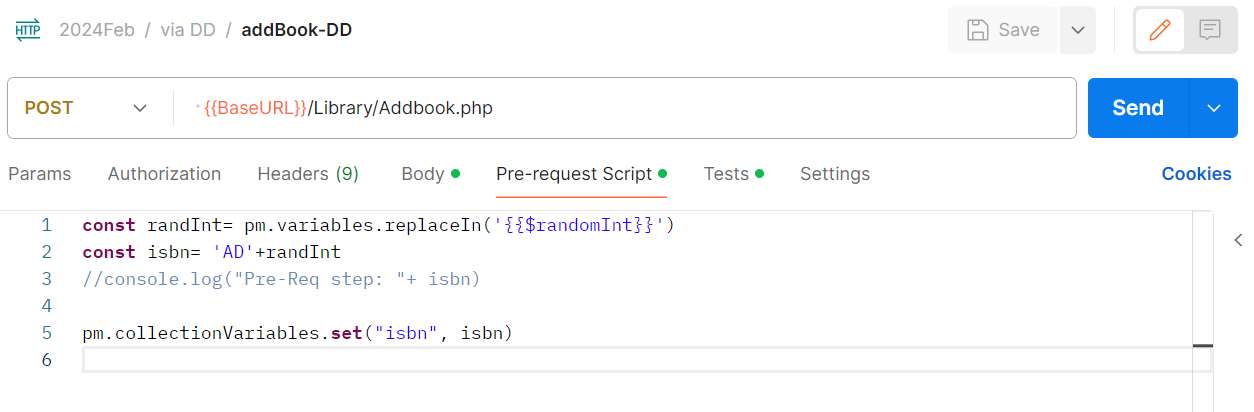
pm.expect(responseExp).to.eql(responseAct);

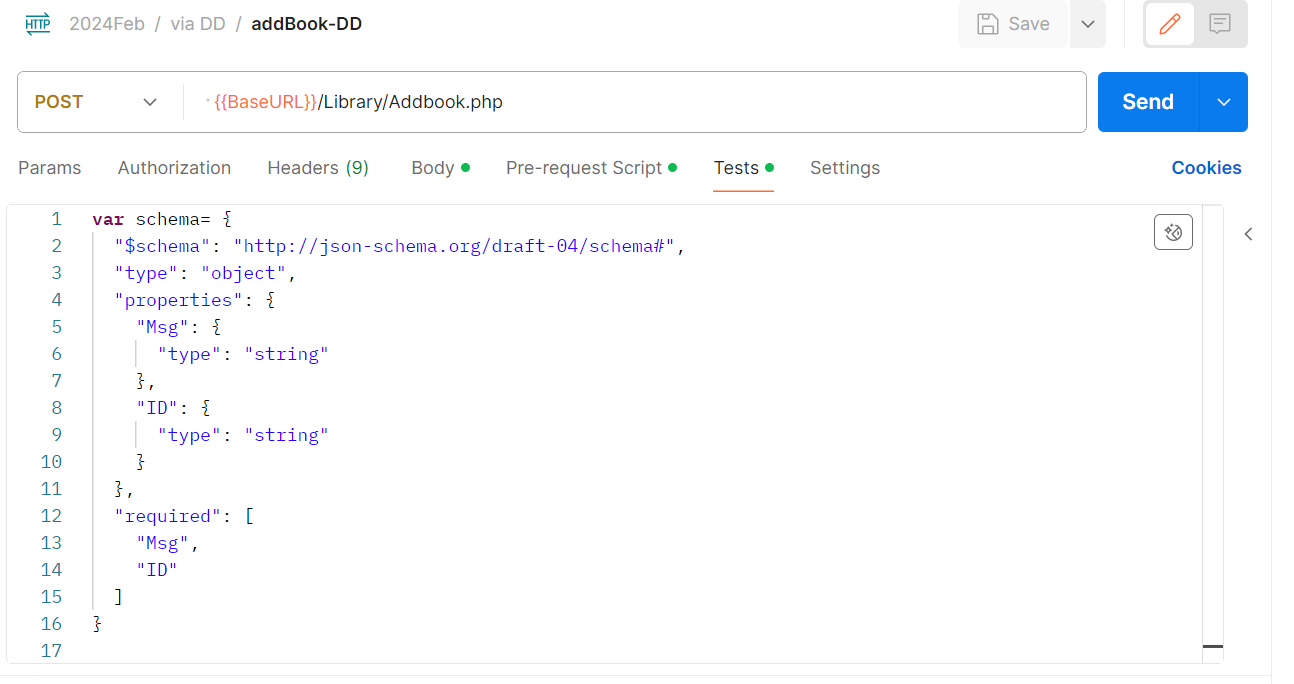
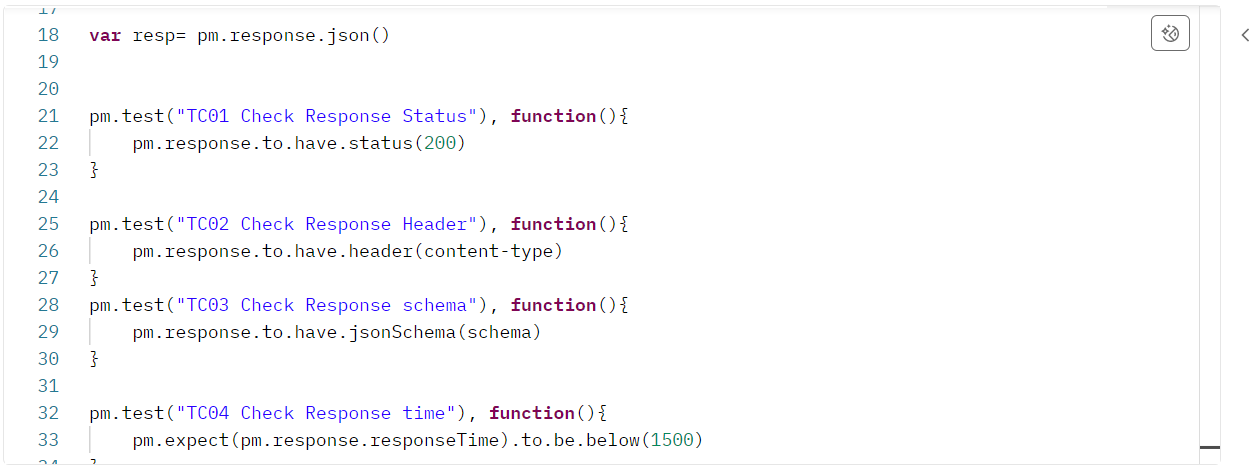
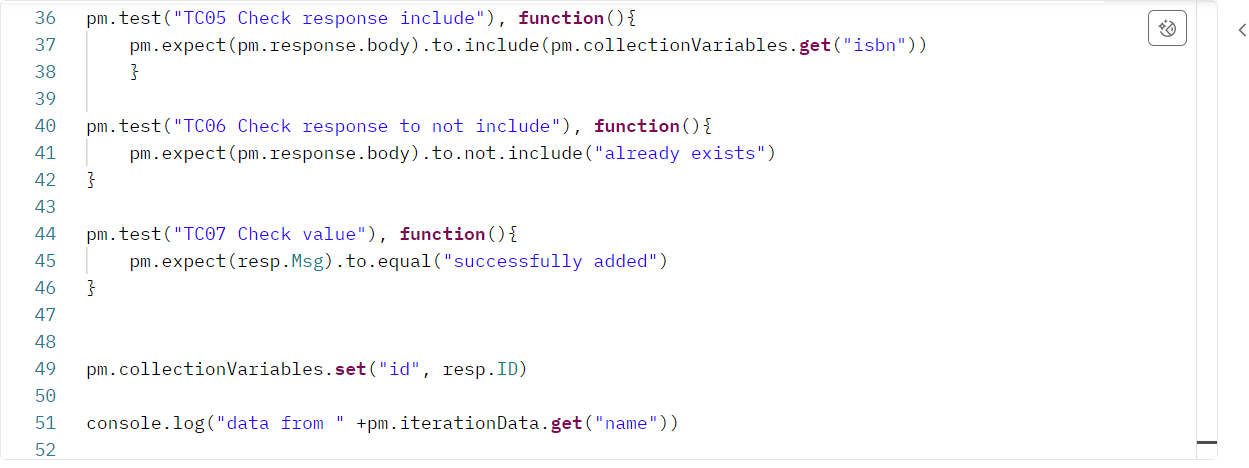
This exactly matches the 2 responses. But this fails when there are some attributes who’s values change on each request like ID or customer number. To ignore this, first you can compare the 2 JSONs to identify such attributes and then delete these attributes while the code is compared. You can use delete responseAct.<att name>

Skip a test



Recent project:

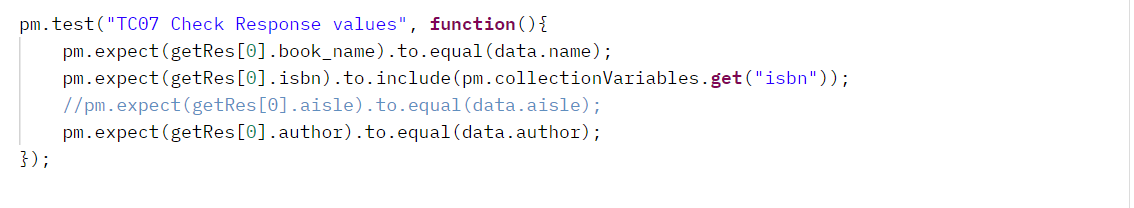


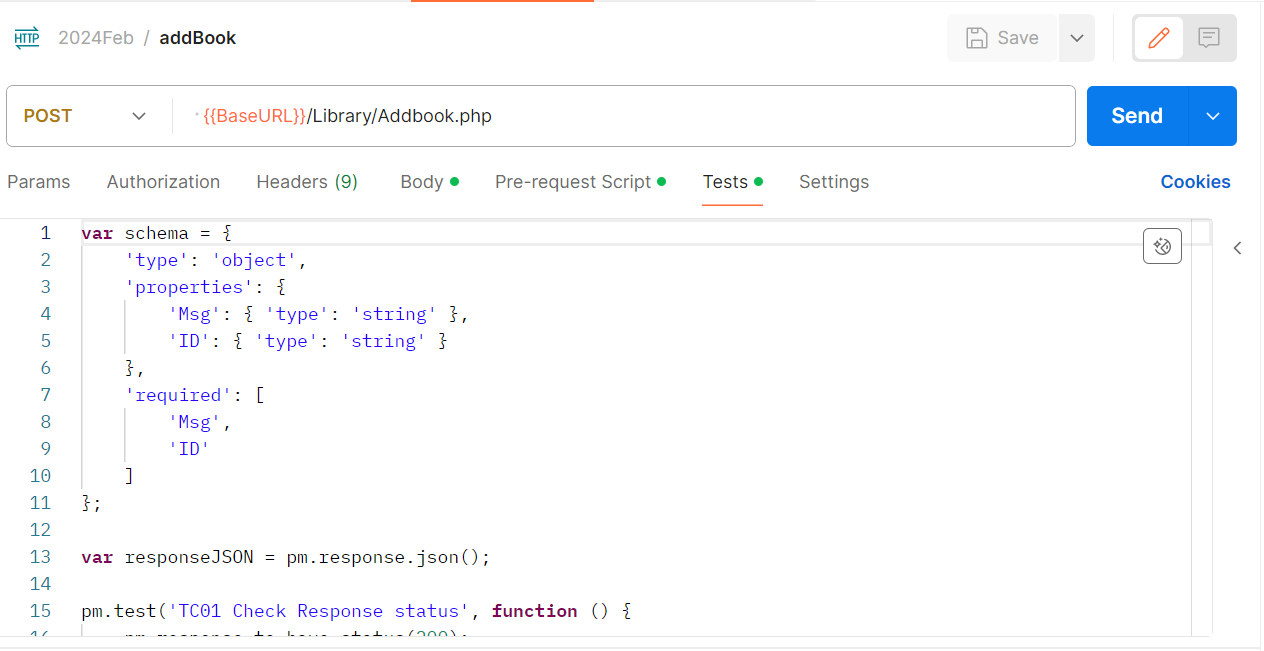
Get

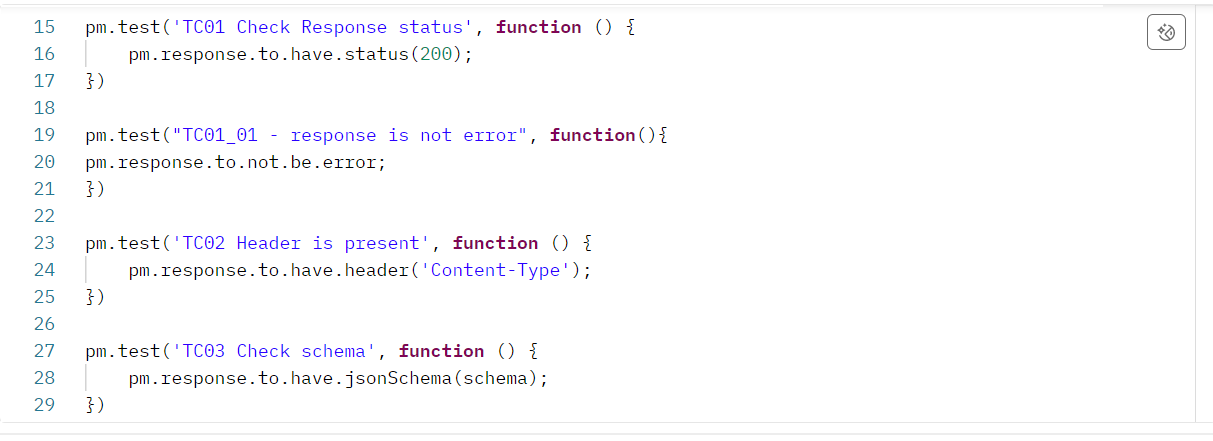


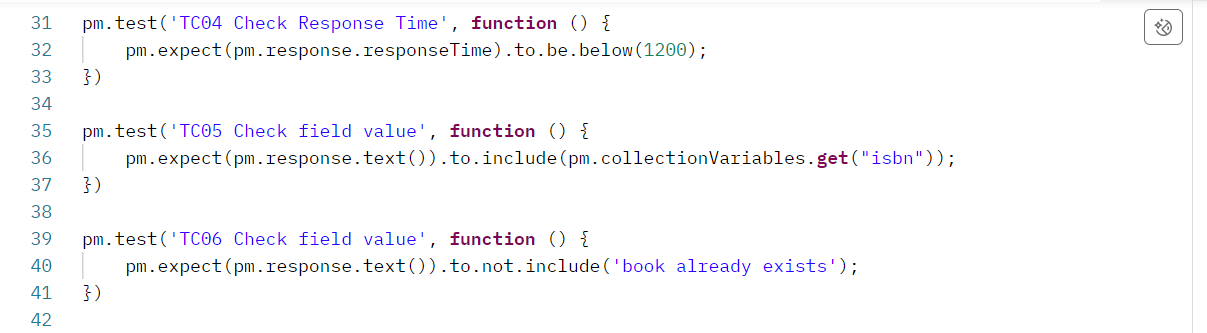
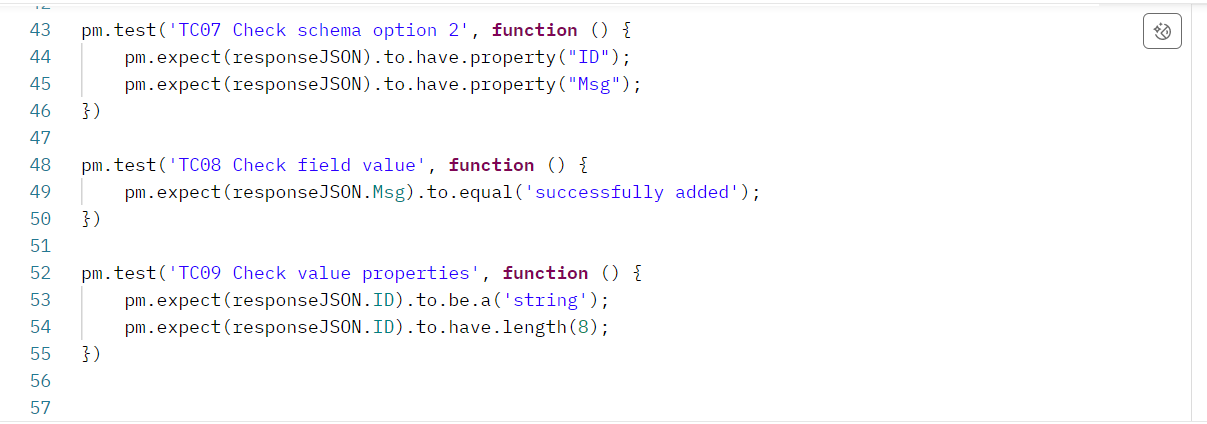




Add second ex:





Link to above collection:

<https://interstellar-comet-883543.postman.co/workspace/Oct-WorkSpace~7576195d-61cf-48a7-9633-a53c8990d524/collection/15821309-91e02f7a-8903-4579-bd0e-39c720b8780c?action=share&creator=15821309&active-environment=15821309-949e251f-d163-40ab-a781-67699236572e>