**Getting Started**

This document explains how to efficiently work with the Postman API framework in your project to perform basic tests on the APIs available by configuring it. It assumes that you are familiar with the Postman Data Driven testing method.

**Before** **you** **start**

You will need access to the following:

* [Postman](https://www.postman.com/downloads/) installed in your system along with Java
* Basic to intermediate knowledge in Excel
* [Node.js and Npm package](https://nodejs.org/en/download) installed in your system.
* [Newman package](https://learning.postman.com/docs/collections/using-newman-cli/installing-running-newman/) installed in your system to generate csv and html reports.

**Interacting with the configured framework**

The framework is setup in a way where the user does not have to use the Postman tool extensively. All the basic info required to pass a request can be configured using the csv file which will be imported since the framework follows a data driven method.

1. Each of the APIs to be configured must be input as an entry in the csv import file.
2. Every row in the import file denotes an instance of an API run.
3. The framework developed can read the following data straight from the import file and assert it into the framework:
   1. Test Scenario ID
   2. Test Scenario Description
   3. Test Case ID
   4. Test Case Description
   5. API Method Type (GET, POST, PUT, PATCH, DELETE, etc....)
   6. Website URL
   7. Expected Response Time of the request to be run.
   8. Expected Status Code of the request to be run.
   9. Headers to be added to the request.
   10. Query Parameters to be added to the URL.
   11. Request Payload.
   12. Values to be stored and chained [A custom method must be written in the test script to fetch the desired value(s)]
   13. Expected values in Response Data after the request is run.
4. Currently, the authorization can be parsed from the input file. But the framework supports only the Bearer token authentication as of now.
5. *The framework is available to be* [*downloaded as a collection.*](https://propelinc-my.sharepoint.com/:f:/p/hariharan/EphMgFf0N11CvKT3HuRhlx8B_XYOfFZygeqXYxBLrsD1yQ?e=l26Gcb)

**Understanding the framework**

The framework facilitates the general validations performed during API testing. Depending on the API method type, a certain set of tests are run to validate the API run.

The validations that are performed validate the API method type along with the expected status code and response time. And based on the request type and the user input, request body validation is also performed.

The tests to be run for each of the iterations in the input file will be performed only if the response code of the corresponding API run matches with the expected status code value given by the user.

**Working with the CSV import file**

As mentioned before, each of the row entry is taken as an iteration of an API run in the test Postman runner window. Also, the data being input in the import file must follow a certain format so that the framework can parse the values without any error(s) and/or mismatches. The data format to be followed for each of the columns available in the import sheet can be referred below:

**1. Test Scenario ID**: The value/data to be input in this field does not need to follow any format. The data given in this field will be displayed in the postman test runner window as part of the test description to enable traceability between the test(s) executed and the test case(s) available in the import sheet.

*Example: TS001, TS01, S01, etc....*

**2. Test Scenario Description:** The value/data to be input in this field does not need to follow any format. The data given in this field will be displayed in the postman test runner window as part of the test description to enable traceability between the test(s) executed and the test case(s) available in the import sheet.

*Example: “Verify the API to generate the authentication token with valid and invalid request payload”*

**3. Test Case ID:** The value/data to be input in this field does not need to follow any format. The data given in this field will be displayed in the postman test runner window as part of the test description to enable traceability between the test(s) executed and the test case(s) available in the import sheet.

*Example: TC001, TC01, C01, etc....*

**4. Test Scenario Description:** The value/data to be input in this field does not need to follow any format. The data given in this field will be displayed in the postman test runner window as part of the test description to enable traceability between the test(s) executed and the test case(s) available in the import sheet.

*Example: “Validate the response time, status code, and responseBody of the POST api request which generates the auth token for the given set of input data”*

For the values of the Test Scenario ID, Test Scenario Description, Test Case ID, and Test Case Description, the test runner window will display the test description as follows,

A picture containing text, screenshot, font, algebra

Description automatically generated

**5.** A picture containing text, screenshot, font, line

Description automatically generated**Priority:** The value/data to be input in this field does not need to follow any format. This is to signify the priority just for the user’s knowledge.

*Example: High, Medium, Low*

**6.** A screenshot of a computer

Description automatically generated with low confidence**API Call Type:** This column data is used to read and pass the API method type for the corresponding API run instance in the framework. This column facilitates the user to explicitly set a request run type directly from the import sheet. Some of the accepted data values are as follows:

POST, GET, PUT, PATCH, DELETE, etc...

**7. A screenshot of a computer

Description automatically generated with medium confidenceUrl:** This column facilitates the user to explicitly set a request URL directly from the import sheet.

*Example: {{baseUrl}}/api/auth/login*

**8. Expected Response Time:** The user can input the expected/ideal response time for the request that is to be run. If the request response time is below the given value, the test pertaining to the response time will pass, as shown below.



Else, the test pertaining to the response time validation will fail, as shown below.



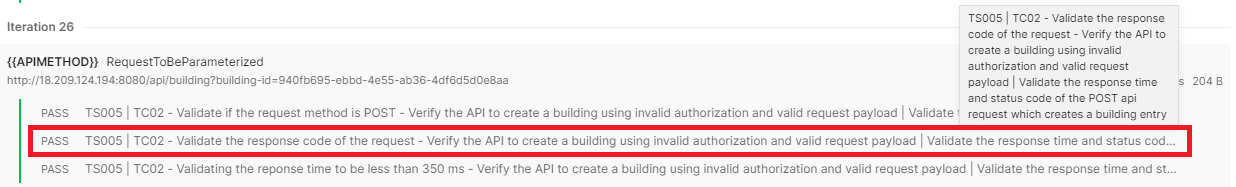
The valid input data in this column are all the positive integers, and the value given will be measured in milliseconds(ms).

*A screenshot of a computer

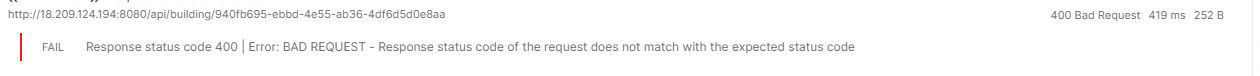
Description automatically generated with low confidence*

*Example: In the given case, if the corresponding request’s response time is less than 350ms**then the test corresponding to the response time will pass, and if the request’s response time is more than 350ms**then the test will fail*

**9. Expected Status Code:** The user can input the expected/ideal Status code for the request that is to be run. If the request status code matches the given value, the test pertaining to the response status code will pass, as shown below.



Else, the test pertaining to the response time validation will fail, as shown below.



The valid input data in this column are all the positive integers, such as 200, 201, 400, 401, 404…

If the user enters the expected status code to be *201 Created,* and the response code of the corresponding API run is *201 Created*. The “*Created”* data entered as part of the expected status code cannot be read by the framework, hence an error will be thrown.

*A picture containing text, screenshot, font, design

Description automatically generated*

*Example: In the given case, if the corresponding request’s status code is then the test corresponding to the response time will pass, and if the request’s response time is more than 350ms**then the test will fail*

**10. Headers:** The user can input the headers which should be passed as part of the request that is to be run. Always make sure to follow the following format,

* ":" colon after the key is necessary while creating expected results, also no additional spaces are required.
* While entering two or more key-value pairs as part of the headers, the entries following the first pair should be entered in the next line “↵”.

*Example:*

A picture containing text, font, screenshot, white

Description automatically generatedkey1:value1,  
key2:value2,  
key3:value3

The key(s) and value(s) given under the Headers column for request passed, the key-value pair(s) are visible as part of the response as displayed below,



**11. Query Parameters:** The user can input the query parameters which should be passed as part of the request URL that is to be run. Always make sure to follow the following format,

* ":" colon after the key is necessary while creating expected results, also no additional spaces are required.
* While entering two or more key-value pairs as part of the headers, the entries following the first pair should be entered in the next line “↵”.

*Example:*

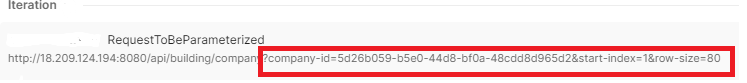
A screenshot of a computer

Description automatically generated with low confidencekey1:value1,

key2:value2,

key3:value3

The key(s) and value(s) given under the Query Parameters column for request passed, the key-value pair(s) are added to the request URL as displayed below,

****

**12. Request Payload:** The user can input the request payload directly in this column which will be passed as part of the request that is to be run. There are no special format(s) to follow for the data to be entered in this column, the format which is used to send the payload cannot be changed from one API to another.

For example, if the user chooses to send the request payload using raw-JSON format, then all the APIs in the import file will send the payload data in the raw-JSON format. This must be noted while passing the request payload using the framework.

**13. Values to be stored and chained:** This column can be used by the user in case one or more values should be stored in a variable from the request response if the key-value pair is available as a root object in the response JSON.

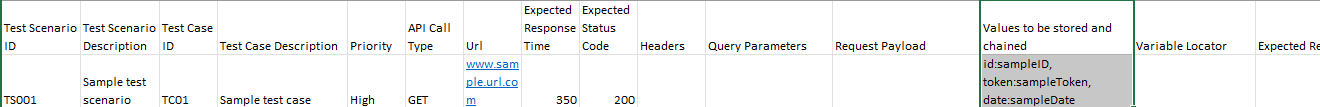
*Example:* If the key-value pair to be stored is “total\_pages”, then the user can simply enter “total\_pages:This key displays the total number of pages in the response” value under the Values to be stored and chained column. Note that the part displayed after the “:” is to give a brief description on the key-value to be fetched.A screenshot of a computer code

Description automatically generated

Note: If the key to be located is nested under the root node, then the user must enter the JSON path to locate the key under the Variable Locator column.

The value located for the keys given are stored in variables, The variable names are assigned using the following data:

1. Test scenario ID
2. Test Case ID
3. Number of the key to be found



For the example given above, The variable name which stores the value for the **first** key (id) will be assigned as *“valueFetched\_****1****\_TS001\_TC01”*, the variable name which stores the value for the **second** key (token) will be assigned as *“valueFetched\_****2****\_TS001\_TC01”*, and the variable name which stores the value for the third key (date) will be assigned as *“valueFetched\_3\_TS001\_TC01”.*

**14. Variable Locator:** If the key-value pair is not available under the root object of the response JSON, the user can provide the inputdata to the JSON path from the base. To easily get the JSON path for key(s) in a nested JSON, we can use tools like [JSON Path Finder](https://jsonpathfinder.com/).A screenshot of a computer code

Description automatically generated

**Note:** If the key to be found is available withing the root object of the response JSON, the user need not use this column.

**x.module\_set[0].module\_dto[0].id**

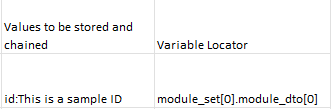
Then, the value to be entered under the Variable Locator column is as follows:

**module\_set[0].module\_dto[0]**

The succeeding part of the JSON path (“**.id**“) should be given as input in the previous column Values to be stored and chained as follows:

id:This is a sample ID

In the above case for **x.module\_set[0].module\_dto[0].id**, the input should be as follows



In short, the part of the JSON locator which succeeds after the root is given as input.

Note: Please make sure that the “.” Is not entered at the beginning of input data

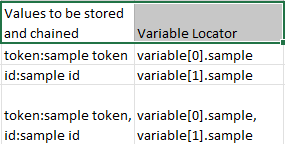
If multiple key values must be stored for an iteration, the values under the Variable Locator column should be displayed as follows:

JsonPath[0].value,

JsonPath[1].value,

JsonPath[2].value

The values have to be separated by commas “,” and each of the next entries must be input in the next line.



For the first entry in the above example, the JSON path to locate the key is x.token.variable[0].sample, and the path is entered in the data file as shown above

*Example:* If the key-value pair to be located from the response body is "first\_name": "Lindsay", the key-value pair in this case is nested under the root node. In this case, the path of the key is found to be ***x.data[1].first\_name***. “x” denotes the root node. The user should enter the values displayed after the root node excluding the period “.”

The value to be entered under the Variable Locator column would be ***data[1]*** and the input under the Values to be stored and chained column would be ***first\_name:To fetch the first name from the request response***

**15. Expected Response Data:** This column can be used by the user to validate if one or more values are available as part of the response body.

*Note:* Currently, the data given under the column must follow a key-value pair format as displayed below

key1:value1,

key2:value2,

key3:value3

Here, value1, value2, and value3 will be taken as reference and the framework will check if the mentioned value(s) are available as part of the response body.

**Generating reports using newman integration**

Newman is a command-line collection runner, which enables to run the collection and report generation. The user can run a collection by exporting the collection folder along with the environment variables and the test data or by generating an API key by following the steps given below

If the user is *exporting* the collection, environment variables along with the test data, the collection can be run as follows:

newman run fileUploadPostmanCollection.json -d importFileData.csv -e environmentVariablesExport -r html,csv

[Please refer to this website](https://learning.postman.com/docs/collections/using-newman-cli/newman-options/) to get familiarized with some of the useful commands along with [html](https://www.npmjs.com/package/newman-reporter-html) and [csv](https://www.npmjs.com/package/newman-reporter-csv) reports generation that can be used in the newman to run the collection and generate the reports efficiently.