Developing and testing APIs using Postman

Overview

Postman, more than a utility, is a collaboration platform for developing APIs.

Normally, it is used as a way to quickly interact with existing APIs without having to code HTTP requests by hand.

It provides support for HTTP based APIs, including REST and GraphQL.

Postman also provides the ability to write tests and use Chai assertions, as seen on these Postman test examples.

With Postman, comes also a built-in (test) collection runner; it is also possible to execute tests from the outside, using a CLI tool named Newman.

Concepts

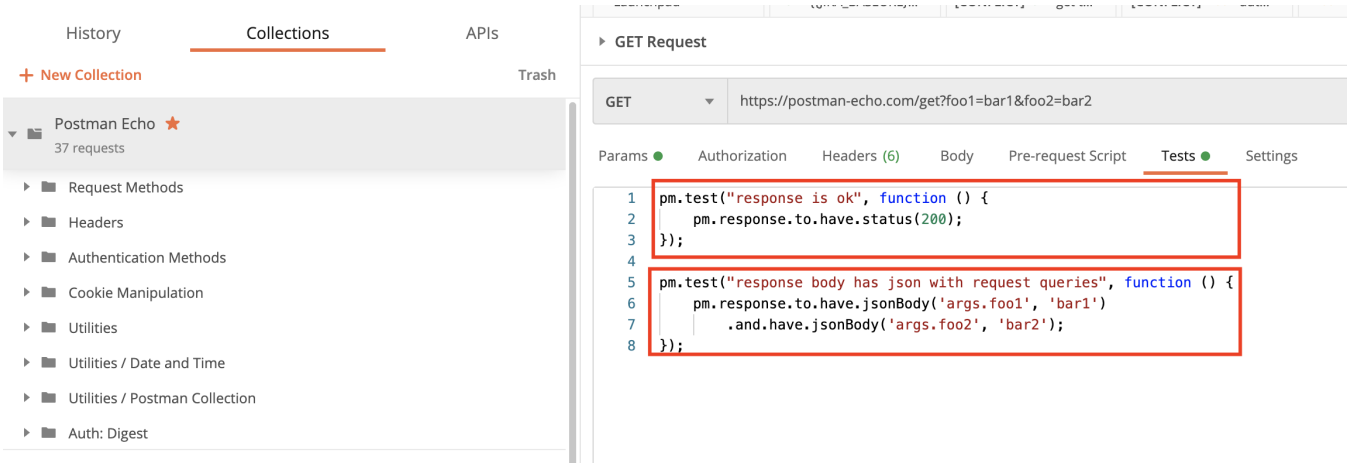
- request: an API request (e.g. HTTP POST on some URL, with some values)
  - authentication: authentication for the API request (e.g. HTTP basic auth, etc); can be defined at multiple levels and inherited
- collection: a way of grouping multiple requests
- folders within the collection: a way to better organize requests within the collection
- variables: can be defined at multiple levels (e.g. global, collection, environment, local, ...)
- test: a test; can be defined at request, folder or collection level
- pre-request script: some code execute before each test; can also be defined at request, folder or collection level
- environment: an abstraction of some test environment that describes a context for running the requests; it consists of one description plus a set of variables with their corresponding values

Implementing tests

Testing is achieved through the usage of scripts.

Tests can be implemented using Javascript and making use of Postman APIs/objects assisted by Chai assertions.

One or more tests can be defined at the request level, or even at the whole collection level.



Pre-request scripts may be useful as a means to initialize some data before the test or to implement some test setup code.

Variables can be defined at multiple levels and can be used to make maintenance easier; the sample applies to authentication, which can also make use of variables.

A test is defined by using "pm.test()".

example of a test that looks at the response's HTTP status code

```
pm.test("response is ok", function () {
    pm.response.to.have.status(200);
});
```

In Postman, quoting Postman documentation, *the **`pm`** object encloses all information pertaining to the script being executed and allows one to access a copy of the request being sent or the response received. It also allows one to get and set environment and global variables.*

Therefore, `pm` can be used to access the response, to perform assertions or even to make some requests.

Integrating with Xray

Integrating with Xray, in order to have visibility of API testing results in Jira, can be done by simply submitting automation results to Xray through the REST API or by using one of the available CI/CD plugins (e.g. for Jenkins).

This can be achieved using Newman and one of its reporters capable of generating a JUnit XML file.

Requirements

- Postman
- Newman
- newman-reporter-junitxray or newman-reporter-junitfull
- Xray Test Management Jenkins plugin (optional)

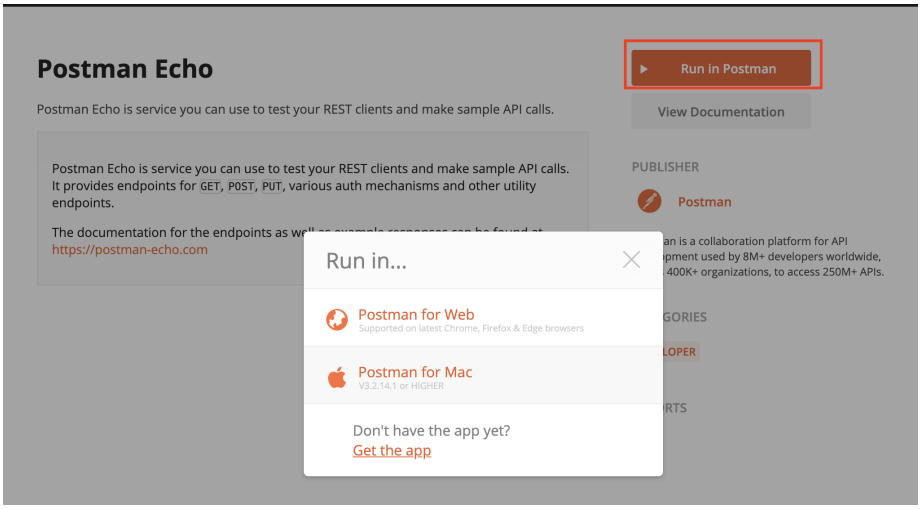
Example

Postman Echo API

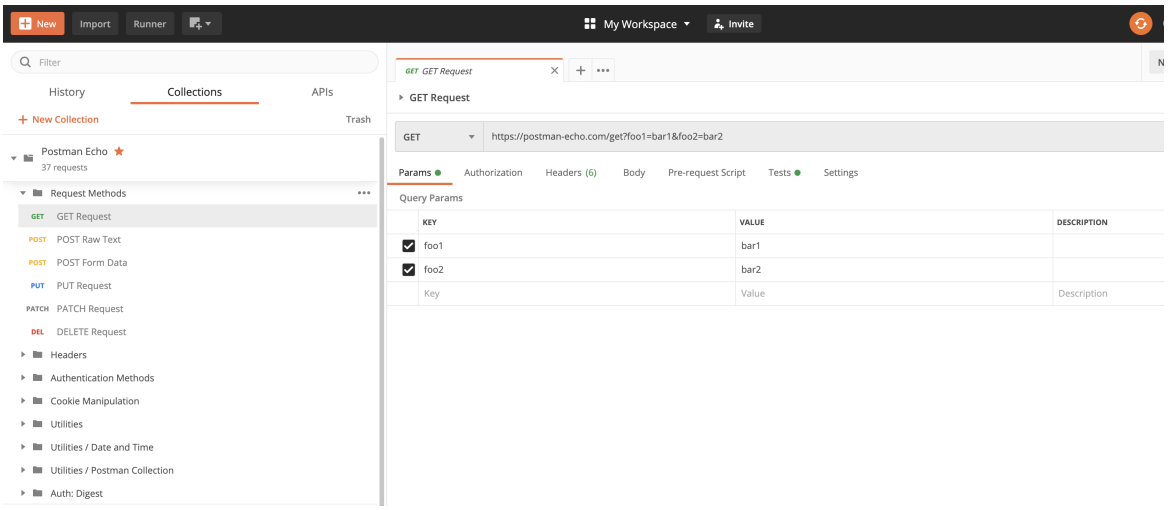
In this example, we're going to use Postman' sample Echo API as a way to showcase some tests and their integration with Xray.

The Postman Echo API provides a set of endpoints that we'll exercise.

We start by cloning an existing Postman collection from a template and importing it to Postman.

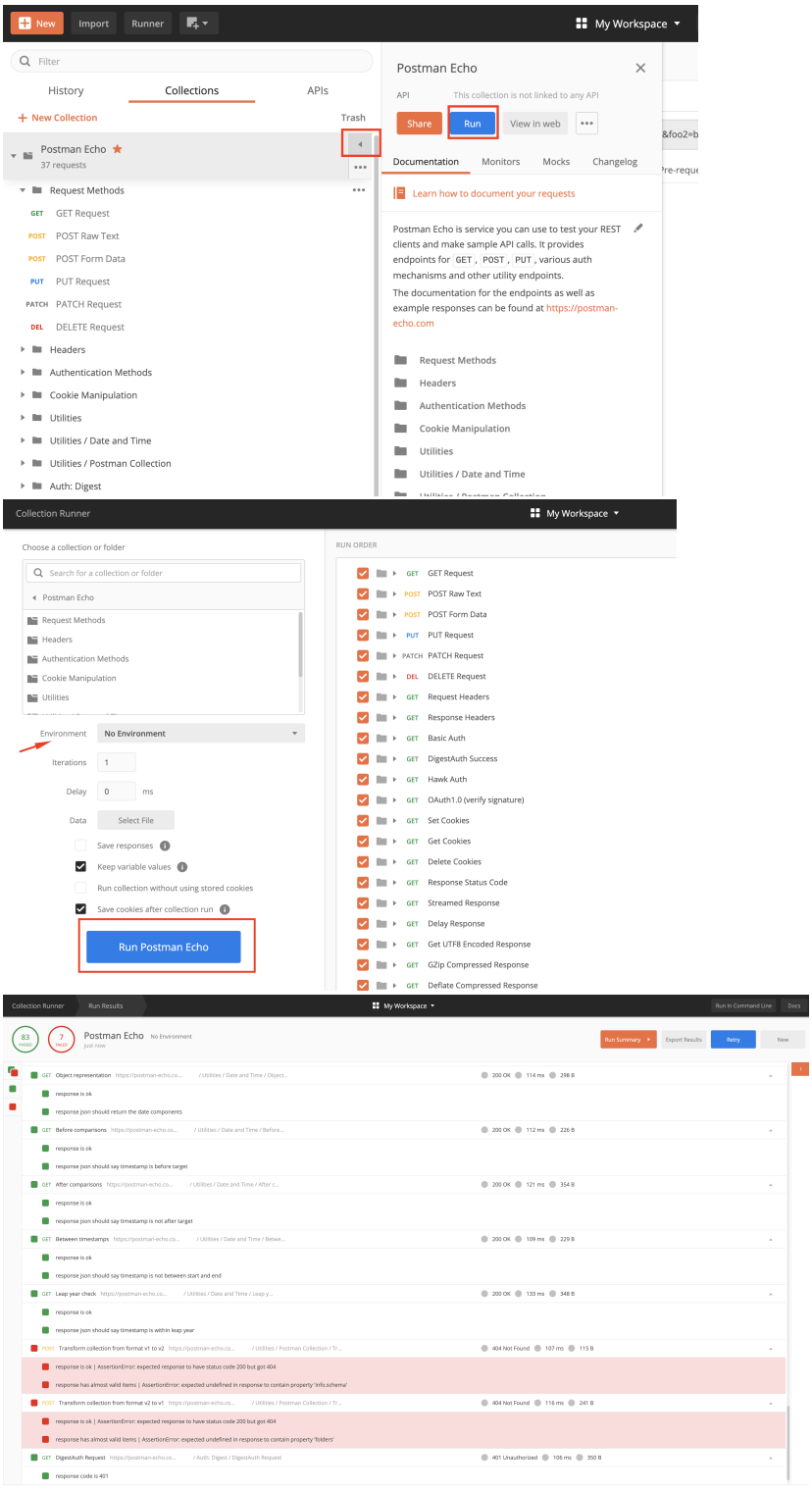


The collection contains a request per each endpoint, where each request has one or more tests.



In the previous example, we can see two tests: one for validating a successful HTTP request based on the status code and another that checks the response's JSON content.

The collection (or a subset of its tests) can be run using the Collection Runner.

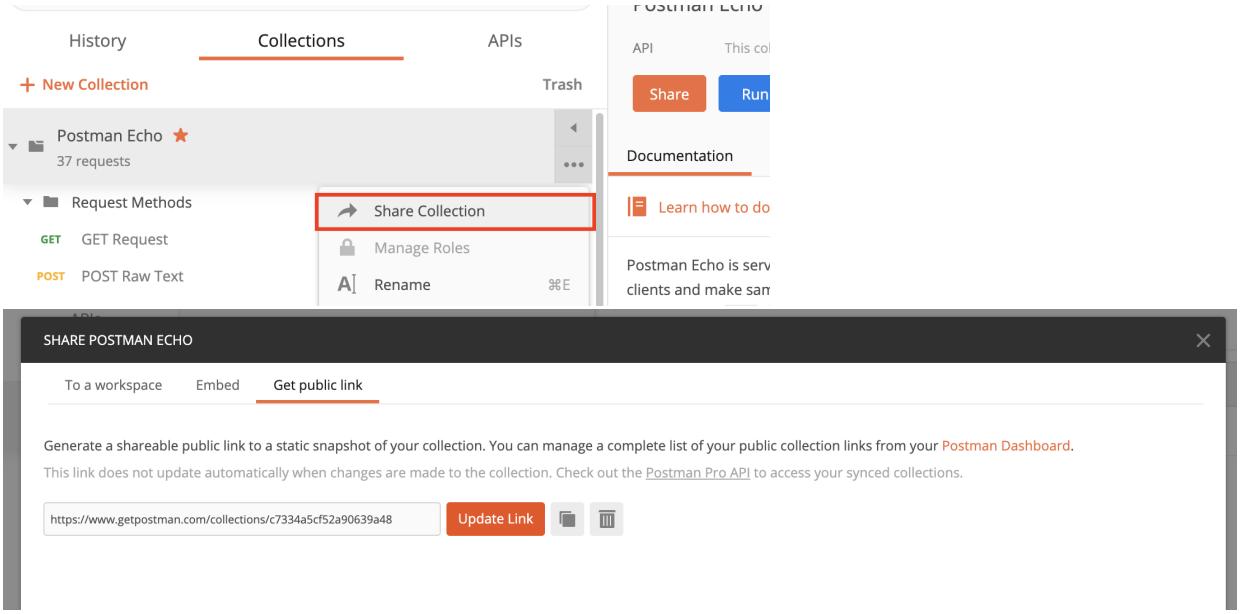


The runner shows the overall count for the number of passed and failed tests. We can also see the assertion error on failed tests; in this case, saving the response (setting the proper flag above) can help us better understand what is happening.

Running the tests can also be done from the command line or from within Jenkins (or any other CI/CD tool). This can be achieved using Newman.

In order to run Newman, we need to provide a path or a URL to our collection.

In this case, we'll obtain a public link to it.



Then we need to decide which Newman reporter to use. Newman provides a built-in JUnit reporter; however, better alternatives exist such as junitxray or junitfull.

Which Newman reporter should I use?

The standard Newman junit reporter produces <testcase> entries in the JUnit XML report that can be misleading as tests will be identified on the Postman test description, which can be similar between different tests (e.g. "response is ok").

Therefore, two alternative reporters arise: newman-reporter-junitxray and newman-reporter-junitfull

"newman-reporter-junitxray" (simply known as "junitxray"), will create <testcases> per each request, which in the end will lead to corresponding Test issues in Xray. This means that there won't be explicit visibility for each Postman test on that request, as they will be treated just as one.

"newman-reporter-junitfull" (simply known as "junitfull"), on the other hand, will produce one <testcase> per each Postman test, which will lead to the same number of corresponding Test issues in Xray.

If you aim just to have high-level overview of the request, then "junitxray" reporter will be preferable; otherwise, "junitfull" may be a better option.

	junit	junitxray	junitfull
tests	• 40 Tests (one per each PM test name /description)	• 37 Tests (one per request)	• 90 Tests (one per each PM test)
generic definition field	<collection>.<pm_test_description> "PostmanEcho.response is ok"	<collection>.<request_name> "PostmanEcho.Object representation"	<folder_path>/<request_name>. <pm_test_description> "Utilities / Date and Time / Object representation.response is ok"
notes	• leads to a collision of tests made for different requests • ignores folder path, which can lead to the collision of requests having the same name • one Test issue per each PM test	• ignores folder path, which can lead to the collision of requests having the same name • doesn't present the multiple PM tests • few Tests, one per each request	• can lead to many Test issues • one Test issue per each PM test, identified by the full (folder) path of the request

Installing Newman and its reporters

```
npm install -g newman
# install one of the following ones
npm install -g newman-reporter-junitxray
npm install -g newman-reporter-junitfull
```

Whenever running Newman, we can specify one or more reporters (if we want to), including a CLI friendly one.

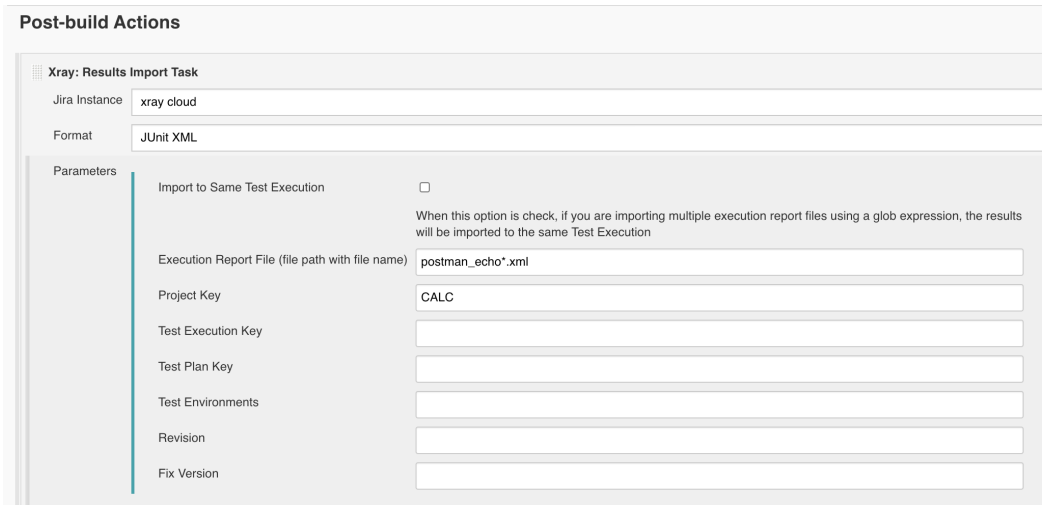
```
newman run https://www.getpostman.com/collections/c7334a5cf52a90639a48 -r 'cli,junitfull,junitxray' --reporter-junitfull-export postman_echo_junitfull.xml --reporter-junitxray-export postman_echo_junitxray.xml -n 1
```

If using Jenkins, we need to configure a build step to execute "newman" command.

Importing results is as easy as submitting them to the REST API with a POST request (e.g. curl), or by using one of the CI plugins available for free (e.g. Xray Jenkins plugin).

The following screenshots show the Jenkins configuration.

We could eventually fill/identify the Test Environment to associate to the Test Execution based on the Postman's Environment being used if it would make sense for us to analyze the results on a per-environment basis.



A Test Execution will be created containing results for all tests executed. Actually, in our specific case and only for demonstration purposes, two Test Executions would be created due to the fact that we're generating two JUnit XML files from the different Newman reporters.

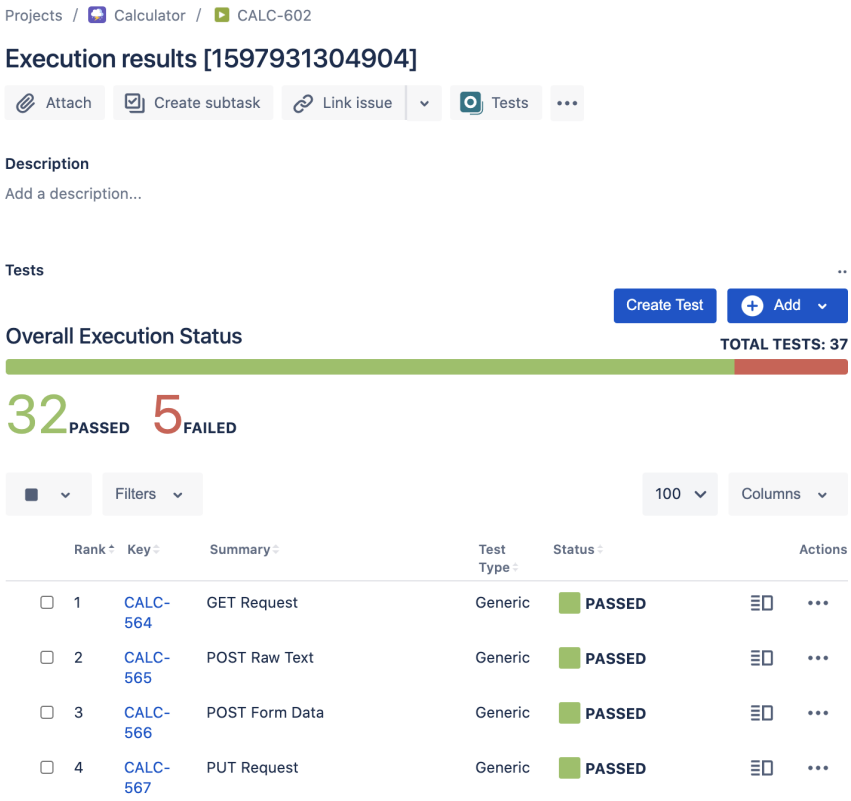
Unstructured (i.e. "Generic") Test issues will be auto-provisioned the first time you import the results, based on the identification of the test (see notes for possible Newman reporters above). The "Generic Definition" field on the Test issue is used as a way to uniquely identify the test.

Please note

Tests will be reused on subsequent result imports as long as you don't change what contributes to the calculation of the test's unique identifier (i.e. "Generic Definition" field); otherwise, new Tests will be auto-provisioned.

Therefore, and depending on the Newman reporter being used, if you change the Postman test description or the folder containing the test, it will lead to new Tests in Jira as Xray will consider them to be new.

In this example, we're looking at the Test Execution (and related Tests) created as a consequence of importing the JUnit XML report produced by the Newman reporter newman-reporter-junitxray.



Within the execution screen details, you can look at the Test Run details which include the duration, overall result, and also any eventual error message.

Calculator / Test Execution: CALC-602 / Test: CALC-593

Object representation

Return to Test Execution | Previous | Next | Execute with Exploratory App | Import Execution Results

Execution Status PASSED

Started On: 20/Aug/2020 02:48 PM

Finished On: 20/Aug/2020 02:48 PM

Assignee: Sérgio Freire

Executed By: Sérgio Freire

Test Environments: -

Versions: -

Revision: -

Comment Preview comment | **Execution Defects (0)** | **Execution Evidence (0)** Add Evidence

Execution Details

Test Description

None

Custom Fields

There are no Test Run Custom Fields defined.

Test Details

Test Type:	Generic
Definition:	PostmanEcho.Object representation

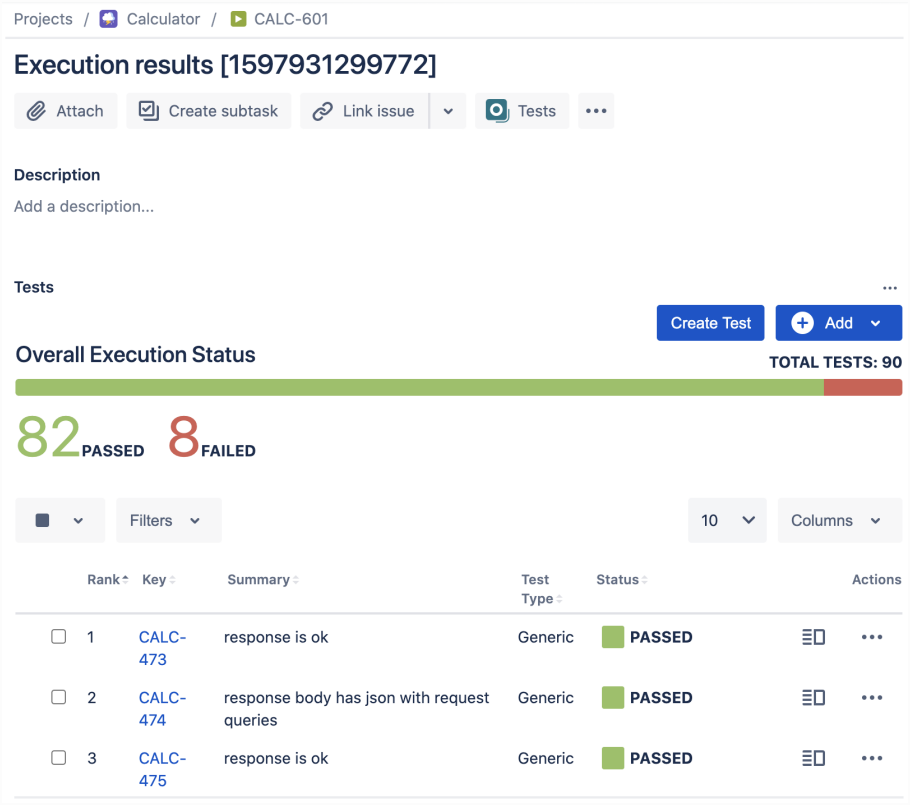
Results

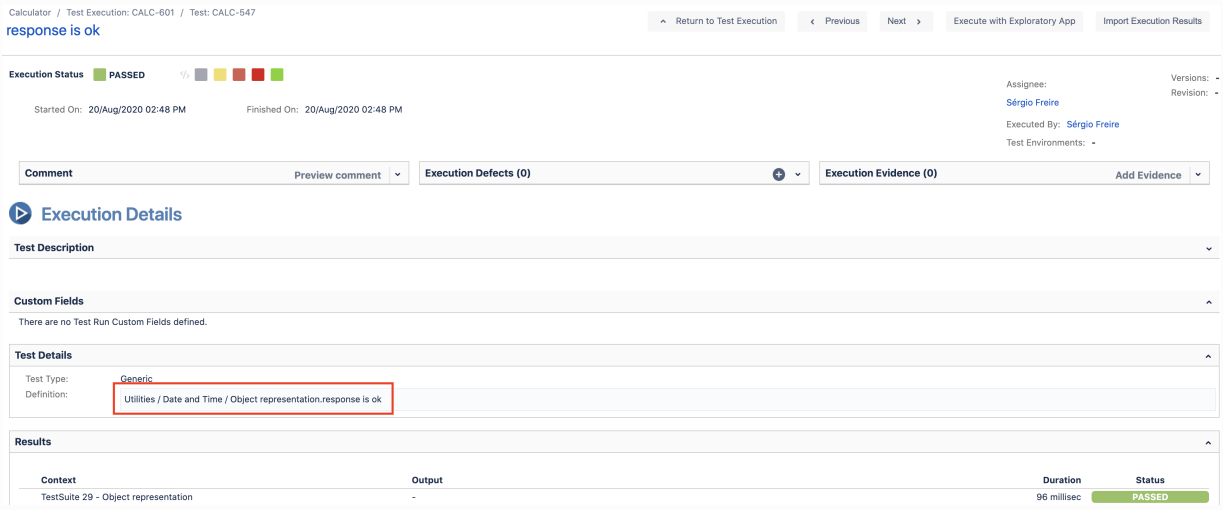
Context	Output	Duration	Status
TestSuite c8c16569-a771-4ad3-874c-9072351ff79a - Utilities / Date and Time	-	-	PASSED

What would be the results if I used "newman-reporter-junitfull"?

If you would use "newman-reporter-junitfull", you would obtain many more Test issues as seen ahead.

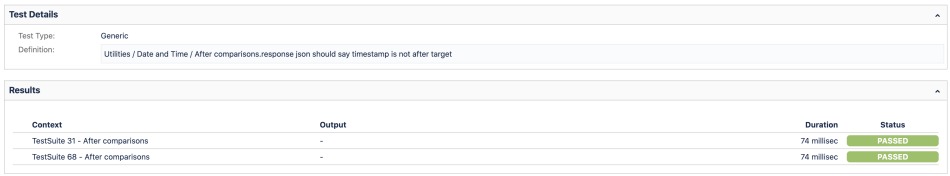
Some of these Tests would have the same Summary as it would be populated from Postman test's description.





Tips

- After importing results, you can link Test issues to existing requirements or user stories, so that you can track coverage on real-time directly on them
- You can map Postman's environment to Xray's Test Environment concept on Test Executions if you want to have visibility of the results on a per-environment basis
- Multiple iterations/executions can be linked to an existing Test Plan, whenever importing the results
- If you run the tests multiple times with "newman -n <number_of_iterations>" parameter, multiple entries will appear within the Results section of the Test Run execution screen details



References

- Postman
- Postman SDK
- Postman Echo API
- Using Newman
- newman-reporter-junitxray
- newman-reporter-junitfull
- Postman Quick Reference Guide