

Data driven testing

In this post, we are going to demonstrate, how you may use your data stored in excel files to perform data driven testing with the help of looping feature of vREST.

Here is the video tutorial for the same:

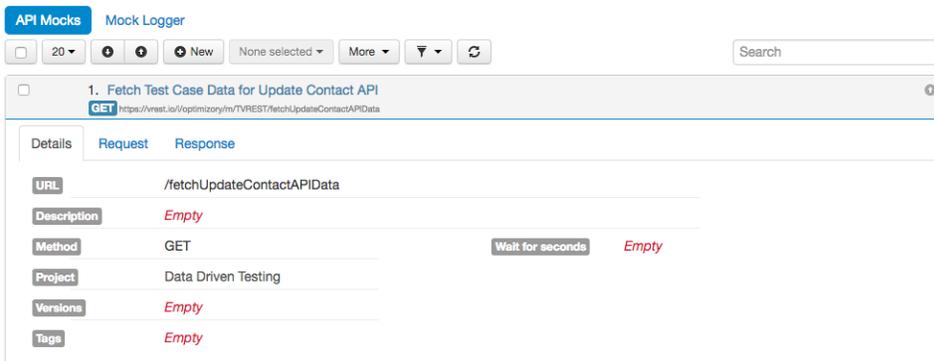
Context: We are defining test cases for a sample [Contacts Application](#). More specifically we are validating various scenarios of Update Contact API with the help of loop. And we are providing data to Update Contact API with the help of Mock server.

i For providing data with the help of CSV file, you will need to install vutil module. vutil module converts this CSV file into JSON format via REST API. For more information, please read [Fetch CSV data for data driven testing](#).

Steps:

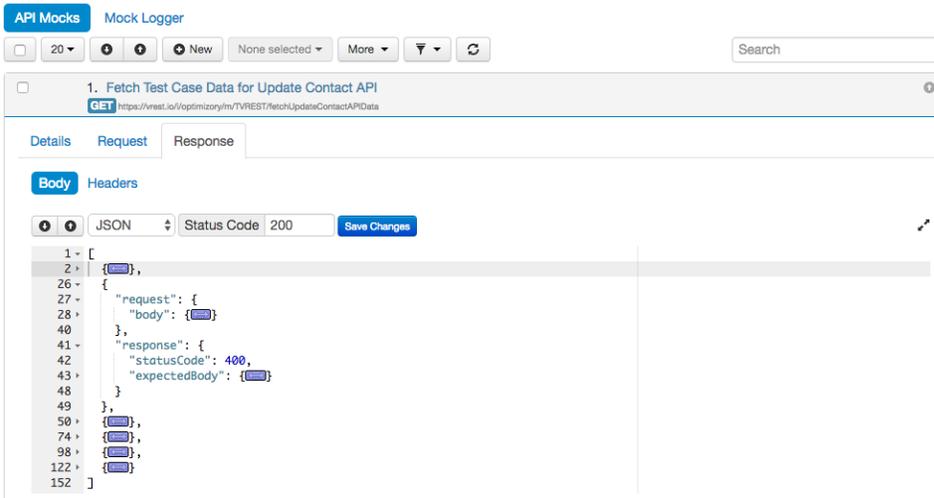
1. First fetch data with the help of a REST API

We are using Mock server to fetch data for our Update Contact API. You can extract test case data from any third party application with the help of a REST API.



The screenshot shows the vREST Mock Server interface. At the top, there are tabs for 'API Mocks' and 'Mock Logger'. Below the tabs, there are controls for '20' items, 'New', 'None selected', 'More', and a search bar. The main area displays a mock definition for a GET endpoint: '1. Fetch Test Case Data for Update Contact API' with the URL 'https://vrest.io/optimizory/m/TVREST/fetchUpdateContactAPIData'. The 'Request' tab is active, showing fields for 'URL' (/fetchUpdateContactAPIData), 'Description' (Empty), 'Method' (GET), 'Project' (Data Driven Testing), 'Versions' (Empty), and 'Tags' (Empty). There is a 'Wait for seconds' field set to 'Empty'.

We have defined the mock server response in the following format:



The screenshot shows the vREST Mock Server interface with the 'Response' tab active. The 'Body' section is expanded, showing a JSON response. The status code is set to 200. The JSON body is a list of objects, with the first object expanded to show its structure: 'request' and 'body' fields. The 'response' field is also expanded, showing 'statusCode': 400 and 'expectedBody': {}.

Now in Test Case's tab, fetch data using a separate test case as shown in snapshot below:

Test Suite "Test Suite 1"

- 1. Validate whether Add Contact API works with all valid details (POST)
- 2. Fetch Data from Mock Server (GET)
 - URL: `{{mockServerBaseURL}}/fetchUpdateContactAPIData`
 - Description: Empty
 - Method: GET
 - Project: Data Driven Testing
 - Versions: 1.0.0
 - Tags: Empty
 - Condition: Empty
 - Authorization: Empty
 - Wait for seconds: Empty

Now extract the whole response with the help of special path "\$". In the below snapshot, we have extracted the whole response in a variable named "data". We will use this variable as loop source.

Test Suite "Test Suite 1"

- 1. Validate whether Add Contact API works with all valid details (POST)
- 2. Fetch Data from Mock Server (GET)
 - Variables from API Response
 - Variables from Results ^{Beta}
 - New Delete
 - Help

Name	JSON Path / Utility Method
data	\$

 - Note:
 - For defining JSON path, follow the guide.
 - For defining Utility method, follow the guide.
 - These variables will be available in all subsequent test cases within a test run.

2. Create test case which is going to be part of loop

We are creating test case for "Update Contact API" as shown in snapshot below. We will discuss the various properties of this test case later.

Test Suite "Test Suite 1"

- 1. Validate whether Add Contact API works with all valid details (POST)
- 2. Fetch Data from Mock Server (GET)
- 3. Validate whether Update Contact API works with various permutations of data (PUT)
 - URL: `{{contactsAppBaseURL}}/contacts/{{contactId}}`
 - Description: Example of Data driven testing.
 - Method: PUT
 - Project: Data Driven Testing
 - Versions: 1.0.0
 - Tags: Empty
 - Condition: Empty
 - Loop source ^{Beta}: Empty
 - Authorization: Empty
 - Wait for seconds: Empty

3. Setup loop construct

Now expand the test case and setup loop by simply providing data for field "Loop Source" in "Details" sub-tab of the test case. We are defining loop source to the variable which we extracted from mock server response as shown in snapshot below.

b. Assertions

#	Source	Property	Comparison	Expected Value
1	Text Body		Call Default Validator	Show expected body
2	Status Code		Equal to number	{{data.\$.response.statusCode}}
3				

c. Expected Body

```
1 | \"{{data.$.response.expectedBody}}\"
```

5. Execution of test cases having loop constructs

Now if we execute our test cases, then loop construct will be executed 5 times as array length was 5. In our sample demonstration, one iteration in the loop has failed.

Test Suite "Test Suite 1"

Bar Chart demonstrating the failure of iteration

Loop Index

- 0 : Passed
- 1 : Passed
- 2 : Passed
- 3 : Passed
- 4 : Failed
- 5 : Passed

Result : Passed 7:09:38 pm, Dec 17, 2016

1. Text Body - Call Default Validator passed the assertion.

2. Status Code - equalToNumber - '400' was a number equal to number '400'.

6. That's it. We request you to provide your feedback around this functionality.