Step-2: Variables, karate-config, API requests, Assertions

In this article, I will be talking about variables and simple API requests with Karate Framework. In the first part, I talked about introducing Karate and did a simple setup. If you have not read the first article, you can check **it out here.** If you don't have any previous experiences with BDD, you can refer to this **article**about what the BDD is and its usage. Now, let's continue with the variables in Karate.

Variables

You can define the variables with the **def** keyword in the feature file directly. The structure should be a **def** keyword followed by a variable name and a value. It is like defining variables in any programming language. Here is the one simple usage of the variables.

```
Scenario: this scenario is for defining variables
 * def firstVariable = 12
 * def secondVariable = 'cakes'
 * print 'firstVariable -> ' + firstVariable, 'secondVariable
```

Output of the above scenario is:

```
10:42:56.259 [ForkJoinPool-1-worker-1] INFO  com.intuit.karate
```

In the example above, I have shown you the simple usage of the variables. But in the API testing, you need to define a complex **JSON** object and do your operations with this JSON object. Now, let's create a JSON object in the feature file and print it.

```
Scenario: defining JSON object and print it
    Given def jsonObject =
    """
        [
          {
            "name": "jack",
            "phone" : 15435667788
          },
          {
            "name": "jennie",
            "phone" : 13443567234
          }
        ]
      """
    * print jsonObject[1].name, jsonObject[1].phone
```

And here is the console output:

```
10:45:05.097 [ForkJoinPool-1-worker-1] INFO com.intuit.karate
```

As you see, it is convenient and practical. You are not responsible for any operations back other than writing Gherkin syntax.

Karate-config.js

You will need to have some files that you can write for centralized data. You can use the karate-config.js file to write your global variables like base URL, environment, etc. Here is the template of the **karate-config.js** file:

```
function fn() {
 var env = karate.env; // get system property 'karate.env'
 karate.log('karate.env system property was:', env);
 if (!env) {
   env = 'dev';
 }
 var config = {
   env: env,
   myVarName: 'hello karate',
   baseUrl: 'https://www.kloia.com/'
 }
```

```
    if (env == 'dev') {
      // customize
      // e.g. config.foo = 'bar';
    } else if (env == 'e2e') {
      // customize
    }
    return config;
  }
```

This is a simple JS function. You can define the variables in the config as a JSON format ("key: value" structure) and use this variable name directly in the feature file. Also, you can define variables based on the environment that you may want to specify different values. In the example below, you will see a basic usage of the karate-config.js file above.

```
Scenario: using karate-config
 * print baseUrl
 * print myVarName
```

And the console output is:

```
11:04:08.750 [ForkJoinPool-1-worker-1] INFO  com.intuit.karate
11:04:08.753 [ForkJoinPool-1-worker-1] INFO  com.intuit.karate
```

API basics

Briefly, the goal of doing API testing is sending requests to some services and matching the response, and validating the result. So I can say that your first task is sending a request, and the second one is to **verify the response**. Let's see what you need to know for sending requests, how to send a request on Karate, then learn the assertions later.

Base URL

You can define the **base URL** in Karate with the <**url**> keyword. Just write the **url** then **base URL** after that.

```
Given url 'https://www.kloia.com/'
```

Path Params

After you define the URL, you need to define a **path** to send a request. You can handle path parameters with the **path** keyword in Karate.

```
And path 'blog'
```

Query Params

Some endpoints present **query parameters** to search for specific keywords. Here is the usage of the query params in Karate:

```
And path 'blog'
And param search = 'karatePosts'
```

GET Request

You can read the data from the API with a **get** request. Here is the simple get request with Karate.

```
Scenario: Get request
 Given url 'https://www.kloia.com/'
 And path 'blog', 'karate'
 When method GET
 Then status 200
```

Did you notice the last line of the test? In Karate, you can easily verify the status code of the response with the **status** keyword. I will explain the rest of the assertions below, but you can use the example above for status verification right now.

I want to show you this GET request with **Java** and **Ruby** so that you can compare all of them and see how Karate is more practical than others.

```
@Test
    public void testSomeTest(){
        baseURI = "https://www.kloia.com/";

        Response response = given().
                                pathParam("key", "karate").
                            when().
                                get("/blog/{key}");

        assertEquals(response.statusCode(), 200);
    }
```

Java simple get example

```
def testSimple
  api_url = "https://www.kloia.com/"
  path = "blog"
  resp = HTTParty.get(api_url + path, headers: { })
  resp.code.should == 200
end
```

Ruby simple get example

As you see the example above, if we compare Karate and other languages, we can easily say that Karate's coding is very straightforward.

POST Request

You can create new data on the services with a **post** request. In Karate, you can use the **request** keyword to attach the request body to your request and **method post** for sending requests.

```
Scenario: Post request
 * def user =
   """
   {
     "name": "Jane",
     "username": "jane_pool",
     "phone": 12546758485
     }
   }
   """

 Given url 'https://www.kloia.com/'
 And request user
 When method POST
 Then status 201
```

PUT Request

You can use the **put method** just like a post request and update the data.

```
Scenario: Put request
 * def user =
   """
   {
     "name": "Jane",
     "username": "jane_forest",
     "phone": 12546758485
     }
   }
   """

 Given url 'https://www.kloia.com/'
 And path 'user', 129
 And request user
```

```
  When method PUT
  Then status 200
```

DELETE Request

You can remove data with a **delete request**, and the **method delete** keyword will handle this request.

```
Scenario: Delete request
 Given url 'https://www.kloia.com/'
 And path 'user', 129
 When method DELETE
 Then status 204
```

As you see in the examples above, you can easily send your API requests with the Karate framework without doing anything other than writing your test scripts with Gherkin syntax.

Assertions

As a tester, you are responsible for verifying responses and making related assertions. I have shown you one of the assertions, **status verification**, but there are many Karate Framework assertions available. You can handle all verifications with **'match'** keywords. I will show you some of them now.

```
Scenario: Matchers examples
 * def jsonBody =
 """
{
 "category": {
    "id": 1,
    "name": "cats"
 },
 "name": "kitty",
 "photoUrls": [
   "www.kitty.com"
  ],
 "tags": [
   {
     "id": 0,
     "name": "sweet"
   }
  ],
 "status": "available"
}
 """
 Given url 'https://petstore.swagger.io/v2'
 And path 'pet'
 And request jsonBody
 When method POST
 Then status 200
 And match response.category == jsonBody.category
 And match response.id == '#present'
 And match response.name == jsonBody.name
 And match responseHeaders['Date'] == '#notnull'
 And match each response.tags == { 'id': '#number', 'name': '#
```

'#present' is for verifying data is present or not.

'#notnull' is for checking if the data is null or not. You can also use '#null'

'#number', '#string' will verify the data types.

each will loop all the responses values if the response includes an array. So you can easily verify all of the values in the response body.

It is just like talking to someone. Quite easy. These matching examples are only some examples of assertions; more matches are featured in **Karate** to use in your test scripts.

This article introduced the simple usage of Karate. The following article will discuss more complex operations like **reading data files, callers, and scenario outline structures**.

Stay on the line and keep learning...

Resources: **https://github.com/intuit/karate**

Share:

Selcuk Temizsoy

Experienced Software Test Automation Engineer working in different industries. Skilled in Agile, Performance Test and Test Automation. Software Test Consultant at kloia