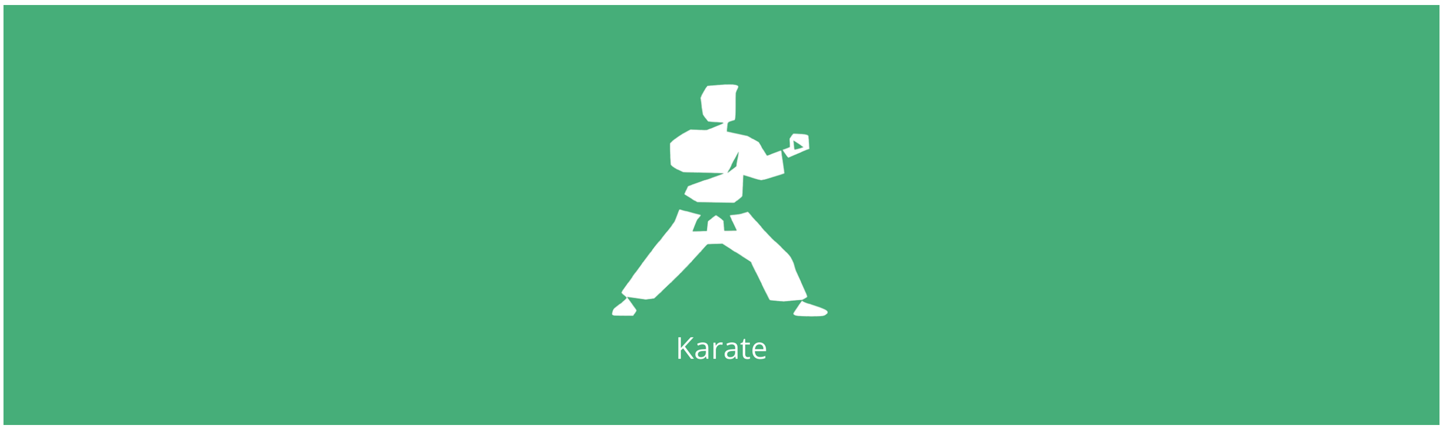


Test automation with Karate (I)

Karate is an open-source testing framework that simplifies API test automation in a way that makes it quick, clear and easy to maintain. This article will go over its main features when it's useful, and common use cases.

What makes it appealing?

Karate is the only open-source tool to combine API test-automation, mocks, performance-testing and even UI automation into a single, unified framework. It uses the Gherkin syntax, made popular by Cucumber, which is language-neutral, easy to use even for non-programmers and is centered on Behavior Driven Development (BDD).

On top of that, it is incredibly versatile when it comes to testing. Powerful JSON and XML assertions are built-in and can run tests in parallel to increase speed.

Main features

- Setup is simple, fast, and straightforward
- BDD is unified in one file. Defining the steps in other locations isn't necessary.
- Tests are simple, concise, legible, and easy to maintain.
- Offers the possibility to use Java classes for complex utilities or help features that facilitate debugging and maintainability.
- Advanced programming skills are not required. Common API actions are implemented in natural language.

Privacy

- Simple implementation of complex assertions.
- Native support for JSON and XML assertions.
- Allows for reading of CSV files (Data-Driven Testing).
- Built-in JavaScript engine.
- Imports CSV or JSON files with information. In Cucumber, the variables are static in a table.
- Complete documentation and examples.
- Websocket support.
- Complete and highly visual HTML reports.
- The following languages and tools are compatible with Karate:

Where can you use it?

Typical use cases

Karate is a great option when you want an automated testing suite for REST services and what you need isn't excessively complex.

Its simplicity and ease of use make it ideal when you want access to a full toolkit of tests that you can use even if you're not a programming whizz as well as something that uses common language that can be understood by both the technical and business teams. At the same time, this allows the tests to serve as functional API documentation, upon which the tests are launched (see the Living Documentation concept).

Where not to use

In cases where a complex data or environment setup is needed for the tests to run properly, it may not be a good option. That's not because Karate is unable to perform more complicated actions, but when you do, some of its best qualities like clarity and simplicity are lost.

does it work?

Privacy

- Java 8+
- Maven or Gradle
- Eclipse or IntelliJ IDE

All the examples that we'll review, and many more, can be found in this repository.

Initial setup

For a Maven project, you'll need to add the following dependencies:

```
<dependency>
    <groupId>com.intuit.karate</groupId>
    <artifactId>karate-apache</artifactId>
    <version>0.9.4</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>com.intuit.karate</groupId>
    <artifactId>karate-junit4</artifactId>
    <version>0.9.4</version>
    <scope>test</scope>
</dependency>
```

If it's a Gradle project, add the following dependencies to the build.gradle file:

```
testCompile 'com.intuit.karate: karate-junit4: 0.9.4'
testCompile 'com.intuit.karate: karate-apache: 0.9.4'
```

Another way to install on a new project is to use the artifact provided by Maven, running the following command in a terminal:

```
mvn archetype:generate \
-DarchetypeGroupId=com.intuit.karate \
-DarchetypeArtifactId=karate-archetype \
-DarchetypeVersion=0.6.2 \
-DgroupId=org.example \
-DartifactId=karate-example
```

Main use cases

To apply the test cases, we'll use the test website https://reqres.in/, which provide various endpoints upon which GET, PUT, POST and DELETE requests can be made.

Privacy

To design the tests, create as many .feature files as needed. In these files, write the collection of scenarios that will be used to test a specific feature.

The basic specification of the tests is based on the Gherkin language:

```
Scenario:
 Given
 When
 And
 Then
```

To review the main Karate features, we'll go over some of the most common use cases:

GET

The test consists of getting a list of the system users and checking to see if the response is correct.

```
Feature: Get Tests on reqres.in
Scenario: Get users list
  Given url  'https://reqres.in' + '/api/users' + '?page=2'
  When method GET
  Then status 200
```

As you can see, there are constant definitions like the URLs that are going to be used more than once. They can be taken out and defined for their use throughout the feature in the Background section. For this, we use the keyword "def" in Karate.

```
Feature: Get Tests on reqres.in
  Background:
  * def urlBase = 'https://reqres.in'
  * def usersPath = '/api/users'
  Scenario: Get users list
     Given url  urlBase + usersPath + '?page=2'
     When method GET
     Then status 200
```

This test can be extended by adding additional checks on the content of the response.

Privacy

```
Feature: Get Tests on reqres.in
  Background:
  * def urlBase = 'https://reqres.in'
  * def usersPath = '/api/users'
  Scenario: Get users list and check value in field
     Given url urlBase + usersPath
     When method GET
     Then status 200
     And match $..first_name contains 'Emma'
     And match $..id contains '#notnull'
```

To make the assertions, use the keyword “match.” By doing so, you can check if an expression is evaluated as *true* by using the built-in JavaScript engine.

The main operators used in these kinds of expressions are:

- match ==
- match !=
- match contains, match contains only, match contains any, and match !contains

The “$” expression is equivalent to the body of the call response. So basically, “$” is the “response.” When you add the two periods (..) as shown in the example, it will search for this characteristic at any level of depth in the response body.

Different markers can be used for generic checks, such as:

#ignore: Ignore the field.

#null: The value should be null.

#notnull: The value should not be null.

#array: The value should be a JSON array.

#object: The value should be a JSON object.

#boolean: The value should be true or false.

er: The value should be a number.

Privacy

#uuid: The value should coincide with the UUID format.

#regex: The value coincides with a regular expression.

#? EX: The Javascript EX expression should evaluate as true.

POST

This test verifies the creation of a user, which is why we should add "BODY" to the request.

Here, we'll see how to run it by also making use of the "Scenario Outline" concept, which puts parameters around both the petition and the expected results in each case.

This method of running tests is extremely useful when we want to run the same feature under different scenarios.

```
Feature: Login and register Tests on reqres.in
  Background:
  * def urlBase = 'https://reqres.in'
  * def loginPath = '/api/login'
Scenario Outline:   As a <description>, I want to get the corresponding
    Given url  urlBase + loginPath
    And request { 'email': <username> , 'password': <password> }
    When method POST
    * print response
    Then response.status == <status_code>
    Examples:
         |username                |password     | status_code    | descr
         |'eve.holt@reqres.in'    |'cityslicka' | 200            | valid
         |'eve.holt@reqres.in'    |null         | 400            | inval
```

In this test definition, you can see several elements worth highlighting like:

- Substitution parameters or placeholders delimited by < >.
- Table of test examples.

the full response to know the attributes that can be later checked, it's included in the definition and will have the following console output:

```
08:12:19.298 [ForkJoinPool-1-worker-1] INFO com.intuit.karate - [print]
  "token": "QpwL5tke4Pnpja7X4"
}
```

PUT

The following test updates the value of a user attribute and checks to see if it has been properly modified.

```
Feature: Post/Put/Patch/Delete Tests on reqres.in
Background:
* def urlBase = 'https://reqres.in'
* def usersPath = '/api/users'
Scenario: Put user
Given url  urlBase + usersPath + '/2'
And request { name: 'morpheus updated',job: 'leader updated' }
When method PUT
Then status 200
And match $.name == 'morpheus updated'
```

In this definition, you can verify that the JSON language is native in Karate. That means that you can insert the definition in any part, without having to worry about closing it or "escaping" with quotation marks. Besides that, you can see that you don't even have to put quotation marks in the keys, since the analyzer takes care of processing them.

Just as JSON structures are used as input data, you can also use XML.

REUSING FEATURES

Let's imagine that after an entity creation test (POST), the object needs to be deleted and this feature will be required more than once. Here, it's convenient to reuse features.

To do this, we'll create two reusable features – one for the creation and one for elimination. Since this isn't a test in itself, it should be annotated with @ignore so Privacy en you run the tests, they are ignored.

```
  Background:
  * def urlBase = 'https://reqres.in'
  * def usersPath = '/api/users'
  Scenario: Post user Data-Driven
     Given url  urlBase + usersPath
     And request { name: '#(name)',job: '#(job)' }
     When method POST
     Then status 201
```

```
@ignore
Feature: Reusable Delete Tests on reqres.in [delete.feature]
  Background:
  * def urlBase = 'https://reqres.in'
  * def usersPath = '/api/users'
  Scenario: Delete user
     * def path = urlBase + usersPath +'/' + id
     Given  url path
     When method DELETE
     Then status 204
```

Now, we'll look at an example where both are used via the "call" command.

```
Feature: Reusable Tests on reqres.in
  Scenario: call post and delete with reusable features and delete with
  * table users
  |name     |job        |
  |'max'    |'tester1'  |
  * def result = call read('post2.feature') users
  * def id = result[0].response.id
  * table ids
  |id|
  |id|
  * def res = result.responseStatus == 201 ? {} : karate.call('delete.f
```

Conditional operations are another feature that we can see. In this example, the deletion will only be carried out if the response code from the previous POST operation is successful.

RUNNING THE TESTS

To run the tests, define a Java class where you'll configure the routine as well as the location of the features.

```
package examples;
: com.intuit.karate.KarateOptions;
```

Privacy

```
@KarateOptions(features = "classpath:examples", tags = "~@ignore")
class ExamplesRunner {
}
```

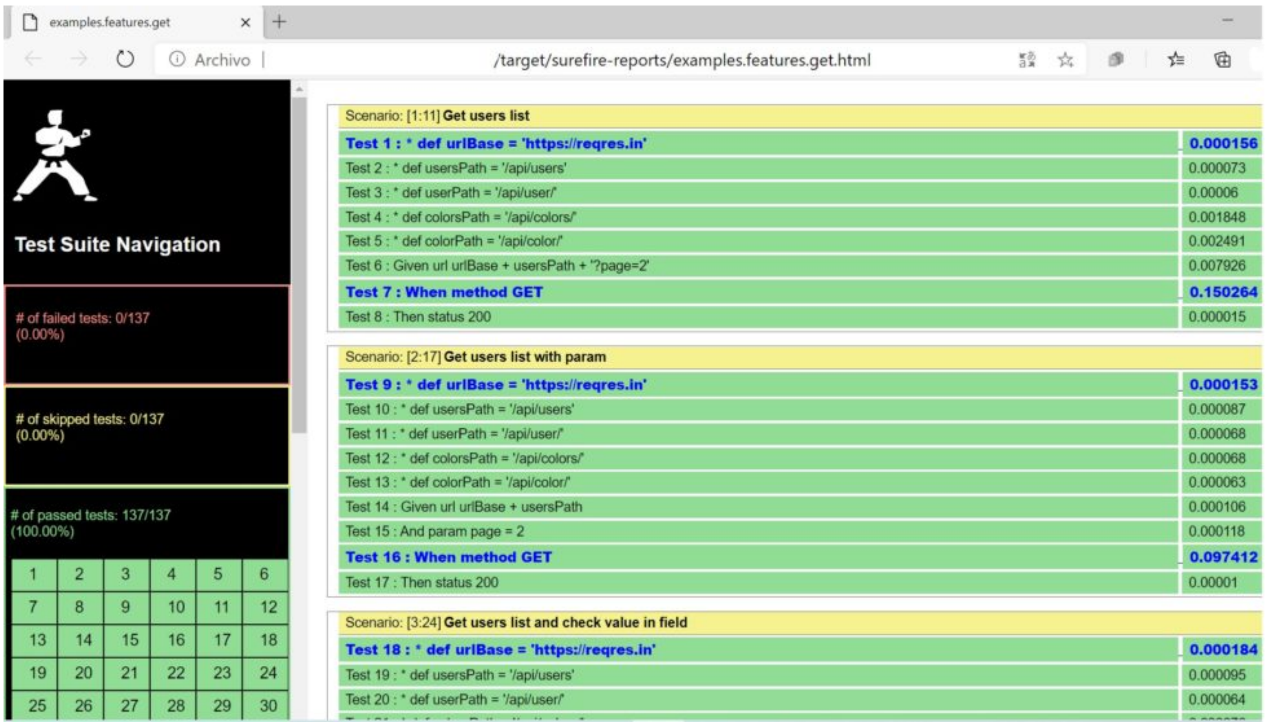
Here, Karate expects the existence of a “karate-config.js” file, with the configuration variables. It’s here that you can define variables like the environment or JavaScript reusables.

There are two ways to launch the execution: from our IDE or by command line. In the second case, you can run the following Maven command:

```
mvn clean test -Dtest=ExamplesRunnerv
```

REPORTS

When the tests are run, a report will be generated for each feature. The report proposed by Karate is in HTML format and its structure is as follows:



# of failed tests: 0/137 (0.00%)

# of skipped tests: 0/137 (0.00%)

# of passed tests: 137/137 (100.00%)

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30

Scenario: [1:11] **Get users list**

Test 1 : * def urlBase = 'https://reqres.in'	**0.000156**
Test 2 : * def usersPath = '/api/users'	0.000073
Test 3 : * def userPath = '/api/user/'	0.00006
Test 4 : * def colorsPath = '/api/colors/'	0.001848
Test 5 : * def colorPath = '/api/color/'	0.002491
Test 6 : Given url urlBase + usersPath + '?page=2'	0.007926
Test 7 : When method GET	**0.150264**
Test 8 : Then status 200	0.000015

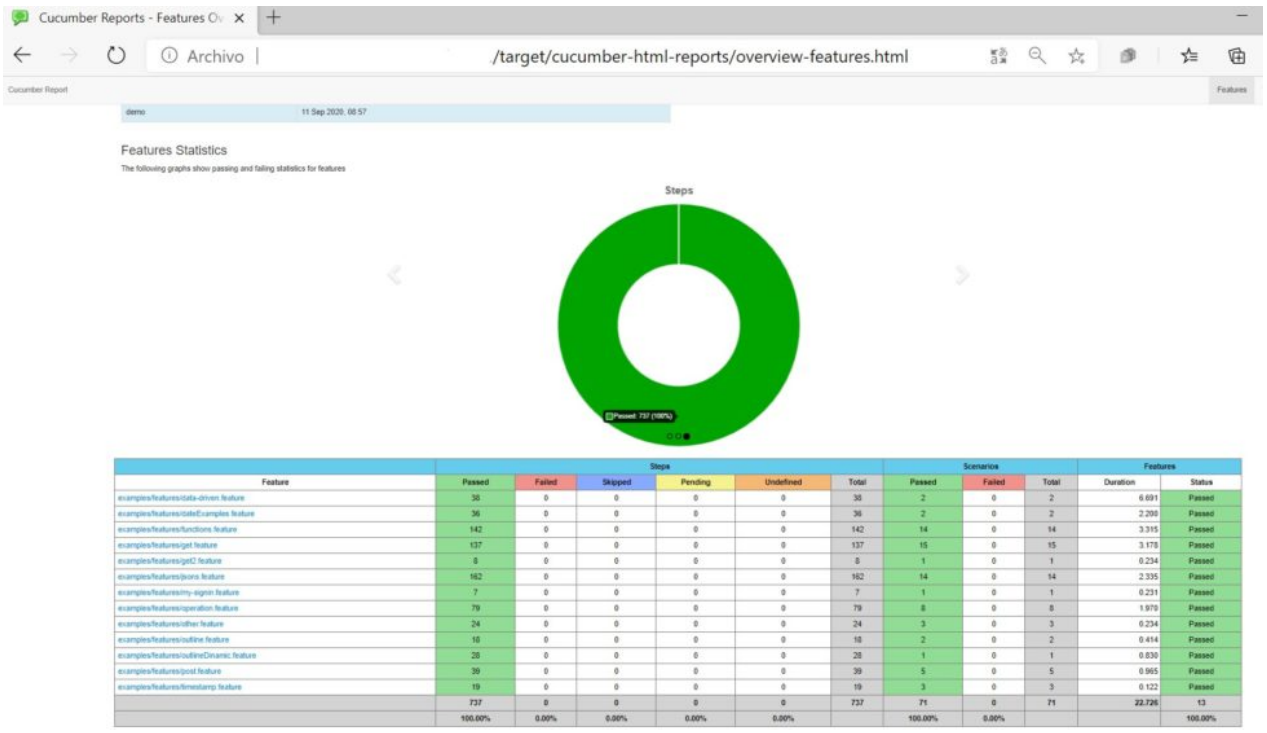
Scenario: [2:17] **Get users list with param**

Test 9 : * def urlBase = 'https://reqres.in'	**0.000153**
Test 10 : * def usersPath = '/api/users'	0.000087
Test 11 : * def userPath = '/api/user/'	0.000068
Test 12 : * def colorsPath = '/api/colors/'	0.000068
Test 13 : * def colorPath = '/api/color/'	0.000063
Test 14 : Given url urlBase + usersPath	0.000106
Test 15 : And param page = 2	0.000118
Test 16 : When method GET	**0.097412**
Test 17 : Then status 200	0.00001

Scenario: [3:24] **Get users list and check value in field**

Test 18 : * def urlBase = 'https://reqres.in'	**0.000184**
Test 19 : * def usersPath = '/api/users'	0.000095
Test 20 : * def userPath = '/api/user/'	0.000064

Privacy


Cucumber Reports - Features Ov
Archivo
/target/cucumber-html-reports/overview-features.html
Cucumber Report
Features
demo
11 Sep 2020, 08:57
Features Statistics
The following graphs show passing and failing statistics for features
Steps
Passed: 737 (100%)

	Steps						Scenarios			Features	
Feature	Passed	Failed	Skipped	Pending	Undefined	Total	Passed	Failed	Total	Duration	Status
examples/features/data-driven.feature	38	0	0	0	0	38	2	0	2	6.691	Passed
examples/features/dateExamples.feature	36	0	0	0	0	36	2	0	2	2.200	Passed
examples/features/functions.feature	142	0	0	0	0	142	14	0	14	3.315	Passed
examples/features/get.feature	137	0	0	0	0	137	15	0	15	3.178	Passed
examples/features/get2.feature	8	0	0	0	0	8	1	0	1	0.234	Passed
examples/features/jsons.feature	162	0	0	0	0	162	14	0	14	2.335	Passed
examples/features/my-signin.feature	7	0	0	0	0	7	1	0	1	0.231	Passed
examples/features/operation.feature	79	0	0	0	0	79	8	0	8	1.970	Passed
examples/features/other.feature	24	0	0	0	0	24	3	0	3	0.234	Passed
examples/features/outline.feature	18	0	0	0	0	18	2	0	2	0.414	Passed
examples/features/outlineDinamic.feature	28	0	0	0	0	28	1	0	1	0.830	Passed
examples/features/post.feature	39	0	0	0	0	39	5	0	5	0.965	Passed
examples/features/timestamp.feature	19	0	0	0	0	19	3	0	3	0.122	Passed
	737	0	0	0	0	737	71	0	71	22.726	13
	100.00%	0.00%	0.00%	0.00%	0.00%		100.00%	0.00%			100.00%

These are Karate's main features, but not its only ones. Within the demo repository, you'll find several features that we haven't mentioned in this article but could come in handy.

Code repository

You can find and download the code in this repository.

Final thoughts

Advantages

One of the key advantages of this tool is that it is open-source, so it undergoes continuous development and has a wide range of examples and documentation for a multitude of test cases.

On top of that, given its integration with Java, its basic features can be extended.

Another advantage is the possibility of running simultaneous tests, which means your tests can be done in just a fraction of the time that it would take to run them one after the other.

Privacy

Cucumber. This could help people looking for consistency across reports.

Finally, being able to create reusable features between scenarios is another important advantage. It's the ideal solution for login flows or utility classes.

Downsides

As we mentioned earlier, in situations where you need a complex preparation of environment or test data for the test to run properly, this may not be your best option. That's because, even though it can integrate with other Java classes and has a built-in JavaScript engine to implement simple functions, from our point of view, this way of working does not bode well for the maintainability of the test suite.

Conclusions

One of the aspects that make BDD enriching for projects is that the business team can write in Gherkin or collaborate with the development team to define the tests. In the case that the business side is open to getting involved in these kinds of tasks, this tool aids communication by using a high-level definition language.

Using a high-level language to define the tests also allows them to serve as the functional documentation for the API itself. That's definitely a win-win.

So, overall, from a Quality Assurance point of view, Karate will cover most of your needs. It really is an excellent, simple and effective tool for conducting web service integration tests.

Sources:

https://github.com/intuit/karate

https://apiumhub.com/es/tech-blog-barcelona/karate-framework-testeo-apis/

https://stackshare.io/karate-dsl/alternatives

https://hackernoon.com/yes-karate-is-not-true-bdd-698bf4a9be39

Privacy

Country

Spain

DKCs

Quality Assurance

Back to list Next Post >