

# Testing APIs using Karate DSL



## What you'll learn

- [Prerequisites](#)
- [Integrating](#)
- [Defining](#) tests using Karate DSL
  - [Run](#) the test and push the test report to Xray
  - [Validate](#) in Jira that the test results are available
    - [JUnit XML results](#)
  - [Jira UI](#)

## • [Tips](#)

## • [References](#)

### Source-code for this tutorial

- code is available in [GitHub](#)

## Overview

Karate is an open-source tool to combine API test-automation, mocks, performance and UI automation in one framework. The BDD syntax popularized by Cucumber is language-neutral, and accessible for non-programmers. Assertions and HTML reports are built-in, and you can run tests in parallel.

## Prerequisites

For this example we will use Karate DSL, that has available a Maven archetype that will build the skeleton of the project.

The Karate Maven archetype will create the `pom.xml`, recommended directory structure, sample test and [JUnit 5](#) runner.

We will need:

- Access to a [demo site](#) that we aim to test
- Maven environment with [JUnit 5](#)

To start using Karate DSL please follow the [Get Started](#) documentation.

The test consists in validating the listing operation of the API from the [demo site](#) and a second one to create and fetch the created user to validate the success.

By default we see 5 files being created, one that will hold the logging configurations, called *logback-test.xml*

#### logback-test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %
msg%n</pattern>
        </encoder>
    </appender>

    <appender name="FILE" class="ch.qos.logback.core.FileAppender">
        <file>target/karate.log</file>
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %
msg%n</pattern>
        </encoder>
    </appender>

    <logger name="com.intuit" level="DEBUG"/>

    <root level="info">
        <appender-ref ref="STDOUT" />
        <appender-ref ref="FILE" />
    </root>

</configuration>
```

A second file that will have the Karate configurations (*karate-config.js*) regarding the environments, it will allow the definitions of variables per environment or to define actions to be executed in different environments:

#### karate-config.js

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

For this example we will not change the above files. We still have 3 other files that were created, one called *ExamplesTest.java* that is a special Java class that will allow the execution in parallel of the tests defined in Karate (you can find more information [here](#)). In this class we have added a method `outputJUnitXml(true)` in the runner to enable the Junit report to be generated in the output.

The final version of the class is below.

### ExamplesTest.java

```
package examples;

import com.intuit.karate.Results;
import com.intuit.karate.Runner;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

class ExamplesTest {

    @Test
    void testParallel() {
        Results results = Runner.path("classpath:examples")
            .outputJunitXml(true)
            .parallel(2);
        assertEquals(0, results.getFailCount(), results.
            getErrorMessages());
    }

}
```

Karate supports JUnit 5 and the advantage is that you can have multiple methods in a test-class. Notice that in the below class we use the `@Karate.Test` tag that will identify this method as a test.

In here we are defining what is the test case we want to execute, in this case we are saying that we want to execute the *"DummyUsers"* feature.

### DummyUsersRunner.java

```
package examples.users;

import com.intuit.karate.junit5.Karate;

class DummyUsersRunner {

    @Karate.Test
    Karate testDummyUsers() {
        return Karate.run("DummyUsers").relativeTo(getClass());
    }

}
```

The final file is the feature file where the tests are defined, although it has similarities with Cucumber, you will see that there is a staggering difference, in this case there is no code behind that you need to define, the notation defined here will be handled directly by Karate. Notice that Json is supported by default and there are some keywords that will trigger actions, check the Karate documentation for more information.

For our example we have defined two scenarios, one to get all dummy users and then fetch the first user by id and another that will create a user and fetch it to validate its creation.

## dummyusers.feature

Feature: sample karate test script

Background:

```
* url 'http://dummy.restapiexample.com/api/v1/'
```

Scenario: get all dummy users and then get the first user by id

Given path 'employees'

When method get

Then status 200

```
* def first = response.data[0]
```

Given path 'employee', first.id

When method get

Then status 200

Scenario: create a dummy user and then get it by id

```
* def user =
```

```
  """
```

```
  {
```

```
    "name": "Karate Test User",
```

```
    "salary": "3000",
```

```
    "age": "35",
```

```
  }
```

```
  """
```

Given path 'create'

And request user

When method post

Then status 200

```
* def id = response.data.id
```

```
* print 'created id is: ', id
```

Given path 'employee',id

When method get

Then status 200

And match response contains {status:success}

Let us go over some specificities of the above code to make it more clear.

First notice that we are using Gherkin language with extra definitions, we have a *Feature* with two *Scenarios*, one *Background* common to both *Scenarios*, where we have defined the default url to be used.

In the scenarios we are using Gherkin language (using the Given-When-Then keywords) and, as Gherkin supports [catch-all symbol '\\*'](#), each time you want to use a script inline prefix it with '\*'.

In the first scenario we are performing a *GET* from the default url (defined in the background) plus what is defined in the *path* and validating that we receive a HTTP 200. Then we extract from the response the first entry of the data element and save it in a variable *first*.

Still in the same test we are performing the last *HTTP GET*, now to the url plus '*employee*' adding the value of the variable *first* in the query string and validating that we get an *HTTP 200*.

The second scenario is a little more complicated as we are performing a *POST* request with a user object in the *BODY* and then extracting the user id to perform a *GET* with it and check if the user was created with success.

Once the code is implemented it can be executed with the following command, that will execute all tests present:

```
mvn test
```

If you need to filter the execution in the command line you can use some filters, as you can see we are defining that we will look into Karate tags and skip the tests with the `@skipme` tag. We are also defining that we will use the `ExamplesTest` runner (JUnit5 parallel executor) but only for tests in the `dummyusers` feature as we can see below:

```
mvn test "-Dkarate.options=--tags ~@skipme classpath:examples/DummyUsers
/dummyusers.feature" -Dtest=ExamplesTest
```

The results are immediately available in the terminal

```
feature: classpath:examples/DummyUsers/dummyusers.feature
scenarios: 2 | passed: 2 | failed: 0 | time: 3.797s

11:01:12.624 [main] INFO com.intuit.karate.Suite - <pass> feature 1 of 1 (0 remaining) classpath:examples/DummyUsers/dummyusers.feature
Karate version: 1.1.0

elapsed: 5.05 | threads: 1 | thread time: 3.76
Features: 1 | skipped: 0 | efficiency: 0.74
Scenarios: 2 | passed: 2 | failed: 0

HTML report: (paste into browser to view) | Karate version: 1.1.0
file:///Users/cristiancunha/Documents/karatetutorial/target/karate-reports/karate-summary.html

[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 5.258 s - in examples.ExamplesTest
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 3.347 s
[INFO] Finished at: 2022-01-20T11:01:13Z
[INFO]
```

Karate also generates an HTML report that have detailed information on the tests results as we can below:

Summary   Top   Feature: examples/DummyUsers/dummyusers.feature   sample karate test script		
2	0	
Scenario: [1] get all dummy users and then get the first user by id		
1	0	1000
2	0	1000
3	0	1000
4	0	1000
5	0	1000
6	0	1000
7	0	1000
8	0	1000
9	0	1000
10	0	1000
11	0	1000
12	0	1000
13	0	1000
14	0	1000
15	0	1000
16	0	1000
17	0	1000
18	0	1000
19	0	1000
20	0	1000
21	0	1000
22	0	1000
23	0	1000
24	0	1000
25	0	1000
26	0	1000
27	0	1000
28	0	1000
29	0	1000
30	0	1000
31	0	1000
32	0	1000
33	0	1000
34	0	1000
35	0	1000
36	0	1000
37	0	1000
38	0	1000
39	0	1000
40	0	1000
41	0	1000
42	0	1000
43	0	1000
44	0	1000
45	0	1000
46	0	1000
47	0	1000
48	0	1000
49	0	1000
50	0	1000
51	0	1000
52	0	1000
53	0	1000
54	0	1000
55	0	1000
56	0	1000
57	0	1000
58	0	1000
59	0	1000
60	0	1000
61	0	1000
62	0	1000
63	0	1000
64	0	1000
65	0	1000
66	0	1000
67	0	1000
68	0	1000
69	0	1000
70	0	1000
71	0	1000
72	0	1000
73	0	1000
74	0	1000
75	0	1000
76	0	1000
77	0	1000
78	0	1000
79	0	1000
80	0	1000
81	0	1000
82	0	1000
83	0	1000
84	0	1000
85	0	1000
86	0	1000
87	0	1000
88	0	1000
89	0	1000
90	0	1000
91	0	1000
92	0	1000
93	0	1000
94	0	1000
95	0	1000
96	0	1000
97	0	1000
98	0	1000
99	0	1000
100	0	1000
101	0	1000
102	0	1000
103	0	1000
104	0	1000
105	0	1000
106	0	1000
107	0	1000
108	0	1000
109	0	1000
110	0	1000
111	0	1000
112	0	1000
113	0	1000
114	0	1000
115	0	1000
116	0	1000
117	0	1000
118	0	1000
119	0	1000
120	0	1000
121	0	1000
122	0	1000
123	0	1000
124	0	1000
125	0	1000
126	0	1000
127	0	1000
128	0	1000
129	0	1000
130	0	1000
131	0	1000
132	0	1000
133	0	1000
134	0	1000
135	0	1000
136	0	1000
137	0	1000
138	0	1000
139	0	1000
140	0	1000
141	0	1000
142	0	1000
143	0	1000
144	0	1000
145	0	1000
146	0	1000
147	0	1000
148	0	1000
149	0	1000
150	0	1000
151	0	1000
152	0	1000
153	0	1000
154	0	1000
155	0	1000
156	0	1000
157	0	1000
158	0	1000
159	0	1000
160	0	1000
161	0	1000
162	0	1000
163	0	1000
164	0	1000
165	0	1000
166	0	1000
167	0	1000
168	0	1000
169	0	1000
170	0	1000
171	0	1000
172	0	1000
173	0	1000
174	0	1000
175	0	1000
176	0	1000
177	0	1000
178	0	1000
179	0	1000
180	0	1000
181	0	1000
182	0	1000
183	0	1000
184	0	1000
185	0	1000
186	0	1000
187	0	1000
188	0	1000
189	0	1000
190	0	1000
191	0	1000
192	0	1000
193	0	1000
194	0	1000
195	0	1000
196	0	1000
197	0	1000
198	0	1000
199	0	1000
200	0	1000
201	0	1000
202	0	1000
203	0	1000
204	0	1000
205	0	1000
206	0	1000
207	0	1000
208	0	1000
209	0	1000
210	0	1000
211	0	1000
212	0	1000
213	0	1000
214	0	1000
215	0	1000
216	0	1000
217	0	1000
218	0	1000
219	0	1000
220	0	1000
221	0	1000
222	0	1000
223	0	1000
224	0	1000
225	0	1000
226	0	1000
227	0	1000
228	0	1000
229	0	1000
230	0	1000
231	0	1000
232	0	1000
233	0	1000
234	0	1000
235	0	1000
236	0	1000
237	0	1000
238	0	1000
239	0	1000
240	0	1000
241	0	1000
242	0	1000
243	0	1000
244	0	1000
245	0	1000
246	0	1000
247	0	1000
248	0	1000
249	0	1000
250	0	1000
251	0	1000
252	0	1000
253	0	1000
254	0	1000
255	0	1000
256	0	1000
257	0	1000
258	0	1000
259	0	1000
260	0	1000
261	0	1000
262	0	1000
263	0	1000
264	0	1000
265	0	1000
266	0	1000
267	0	1000
268	0	1000
269	0	1000
270	0	1000
271	0	1000
272	0	1000
273	0	1000
274	0	1000
275	0	1000
276	0	1000
277	0	1000
278	0	1000
279	0	1000
280	0	1000
281	0	1000
282	0	1000
283	0	1000
284	0	1000
285	0	1000
286	0	1000
287	0	1000
288	0	1000
289	0	1000
290	0	1000
291	0	1000
292	0	1000
293	0	1000
294	0	1000
295	0	1000
296	0	1000
297	0	1000
298	0	1000
299	0	1000
300	0	1000
301	0	1000
302	0	1000
303	0	1000
304	0	1000
305	0	1000
306	0	1000
307	0	1000
308	0	1000
309	0	1000
310	0	1000
311	0	1000
312	0	1000
313	0	1000
314	0	1000
315	0	1000
316	0	1000
317	0	1000
318	0	1000
319	0	1000
320	0	1000
321	0	1000
322	0	1000
323	0	1000
324	0	1000
325	0	1000
326	0	1000
327	0	1000
328	0	1000
329	0	1000
330	0	1000
331	0	1000
332	0	1000
333	0	1000
334	0	1000
335	0	1000
336	0	1000
337	0	1000
338	0	1000
339	0	1000
340	0	1000
341	0	1000
342	0	1000
343	0	1000
344	0	1000
345	0	1000
346	0	1000
347	0	1000
348	0	1000
349	0	1000
350	0	1000
351	0	1000
352	0	1000
353	0	1000
354	0	1000
355	0	1000
356	0	1000
357	0	1000
358	0	1000
359	0	1000
360	0	1000
361	0	1000
362	0	1000
363	0	1000
364	0	1000
365	0	1000
366	0	1000
367	0	1000
368	0	1000
369	0	1000
370	0	1000
371	0	1000
372	0	1000
373	0	1000
374	0	1000
375	0	1000
376	0	1000
377	0	1000
378	0	1000
379	0	1000
380	0	1000
381	0	1000
382	0	1000
383	0	1000
384	0	1000
385	0	1000
386	0	1000
387	0	1000
388	0	1000
389	0	1000
390	0	1000
391	0	1000
392	0	1000
393	0	1000
394	0	1000
395	0	1000
396	0	1000
397	0	1000
398	0	1000
399	0	1000
400	0	1000
401	0	1000
402	0	1000

## JUnit Report

```
<testsuite failures="0" name="examples/DummyUsers/dummyusers.feature"
skipped="0" tests="2" time="4.30768"><testcase classname="examples.
DummyUsers.dummyusers" name="[1:6] get all dummy users and then get the
first user by id" time="3.102637"><system-out>* url 'http://dummy.
restapiexample.com/api/v1/' ..... passed
Given path 'employees'
..... passed
When method get
..... passed
Then status 200
..... passed
* def first = response.data[0]
..... passed
Given path 'employee', first.id
..... passed
When method get
..... passed
Then status 200
..... passed
</system-out></testcase>
<testcase classname="examples.DummyUsers.dummyusers" name="[2:17] create a
dummy user and then get it by id" time="1.205043"><system-out>* url
'http://dummy.restapiexample.com/api/v1/' .....
passed
* def user =
..... passed
Given url 'http://dummy.restapiexample.com/api/v1/create'
..... passed
And request user
..... passed
When method post
..... passed
Then status 200
..... passed
* def id = response.data.id
..... passed
* print 'created id is: ', id
..... passed
Given path id
..... passed
</system-out></testcase>
</testsuite>
```

If you have more than one feature file there will be one JUnit report per feature file.



A new version of Karate is about to be released where the testcase name will not have the order of the scenario and line.

A release candidate with the change is already available for you to experiment: <https://search.maven.org/artifact/com.intuit.karate/karate-core/1.2.0.RC4/jar>.

With the next official release the next step will not be needed and can be skipped.

Notice that the JUnit report generated with Karate joins, in the name of the testcase, the order of the scenario and line: "[1:6]" concatenated with the testcase name. In Xray we are using the testcase path+name to uniquely identify the test each time the result is uploaded, in this case if the line changes (due to some edition of the file thus changing the line of the code) Xray will create a new test (with this new name) instead of uploading the results to the previously created one.

We advise you to use the tool available in <https://github.com/bitcoder/junit-processor> to remove the characters from the testcase name, this tool have a patch exactly to remove that from the JUnit report generated. To use it you just have to run the following command:

```
junit-processor -p 1 examples.DummyUsers.dummyusers.xml
```

This will produce a new file called "junit-new.xml" that you can use to upload to Xray.

---

## Integrating with Xray

As we saw in the above example, where we are producing a JUnit report with the result of the tests, we need to import those results to your Jira instance, this can be done by simply submitting automation results to Xray through the REST API, by using one of the available CI/CD plugins (e.g. for Jenkins) or using the Jira interface to do so.

---

### API

#### API

Once you have the report file available you can upload it to Xray through a request to the [REST API endpoint for JUnit](#), and for that the first step is to follow the instructions in [v1](#) or [v2](#) (depending on your usage) to obtain the token we will be using in the subsequent requests.

#### Authentication

The request made will look like:

```
curl -H "Content-Type: application/json" -X POST --data '{ "client_id":  
"API_KEY", "client_secret": "API_SECRET" }' https://xray.cloud.getxray.app  
/api/v2/authenticate
```

The response of this request will return the token to be used in the subsequent requests for authentication purposes.

#### JUnit XML results

Once you have the token we will use it in the API request with the definition of some common fields on the Test Execution, such as the target project, project version, etc.

```
curl -H "Content-Type: text/xml" -X POST -H "Authorization: Bearer  
$token" --data @"junit-new.xml" https://xray.cloud.getxray.app/api/v2  
/import/execution/junit?projectKey=XT&testPlanKey=XT-351
```

With this command we are creating a new Test Execution in the referred Test Plan with a generic summary and two tests with a summary based on the test name.

Projects / Xray Tutorials / XT-351

### tutorial-java-karate

Attach Create subtask Link issue Tests ...

Description  
Add a description...

Tests  
Tests Test Executions  
Add Tests Create Test Execution View on board

Overall Execution Status All Environments, final status

2 PASSED TOTAL TESTS: 2

Filters 10 Columns

Key	Summary	Assignee	#Test Executions	Dataset	Latest Status	Actions
XT-356	get all dummy users and then get the first user by id		1		PASSED	⌵ ...
XT-357	create a dummy user and then get it by id		1		PASSED	⌵ ...

Prev 1 Next Total 2 issues

## Jira UI

### Jira UI

1

Create a Test Execution for the tests that you have

Projects / Xray Tutorials / XT-351

### tutorial-java-karate

Attach Create subtask Link issue Tests ...

Description  
Add a description...

Tests  
Tests Test Executions  
Add Tests Create Test Execution View on board

Overall Execution Status All Environments, final status

2 PASSED TOTAL TESTS: 2

Filters 10 Columns

Key	Summary	Assignee	#Test Executions	Dataset	Latest Status	Actions
XT-356	get all dummy users and then get the first user by id		1		PASSED	⌵ ...
XT-357	create a dummy user and then get it by id		1		PASSED	⌵ ...

Prev 1 Next Total 2 issues

2

Fill in the necessary fields and press "Create"

#### Create planned Test Execution

Project  
Xray Tutorials

Summary  
Test Execution for Test Plan XT-351

Assignee  
Cristiano Cunha

Choose a user to assign the Test Execution

File Version/s  
Select...

Test Environment  
Select...

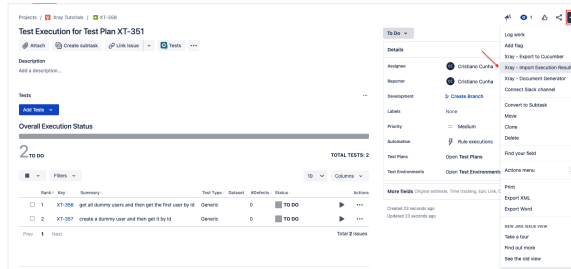
☒ Go to Test Execution

Create Cancel

3

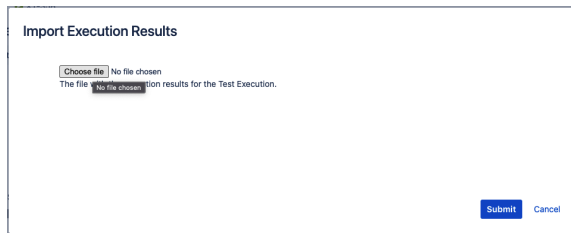
Open the Test Execution and import the JUnit report





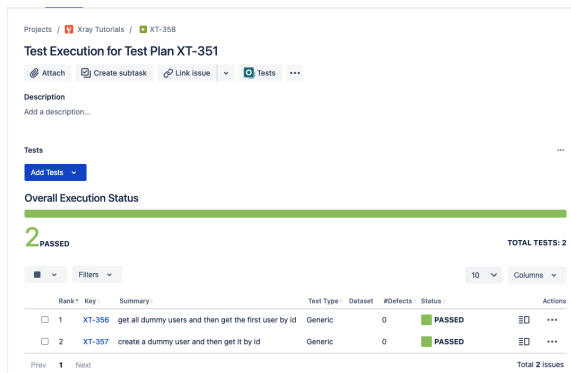
4

Choose the results file and press "Import"

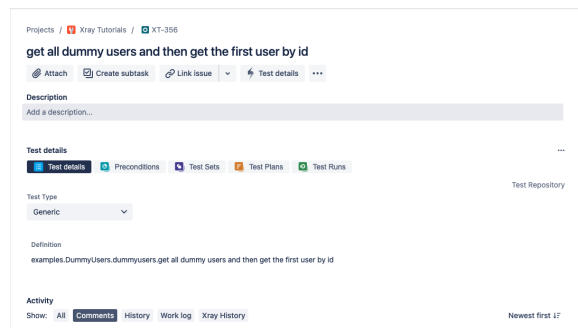


5

The Test Execution is now updated with the test results imported



Tests implemented will have a corresponding Test entity in Xray. Once results are uploaded, Test issues corresponding to the tests are auto-provisioned, unless they already exist.



Xray uses a concatenation of the suite name and the test name as the the unique identifier for the test.

In Xray, results are stored in a Test Execution, usually a new one. The Test Execution contains a Test Run per each test that was executed using playwright-test runner.

Detailed results, including logs and exceptions reported during execution of the test, can be seen on the execution screen details of each Test Run, accessible through the *Execution details*:

As we can see here:

## Tips

- after results are imported, in Jira Tests can be linked to existing requirements/user stories, so you can track the impacts on their coverage.
- results from multiple builds can be linked to an existing Test Plan, to facilitate the analysis of test result trends across builds.
- results can be associated with a Test Environment, in case you want to analyze coverage and test results per environment later on. A Test Environment can be a testing stage (e.g. dev, staging, pre-prod, prod) or an identifier of the device/application used to interact with the system (e.g. browser, mobile OS).

## References

- <https://karatelabs.github.io/karate/>
- <https://github.com/karatelabs/karate>
- <http://dummy.restapiexample.com/>
- <https://github.com/bitcoder/junit-processor>