

API TESTING USING KARATE FRAMEWORK

Author: *Sai Santhosh Marupaka*

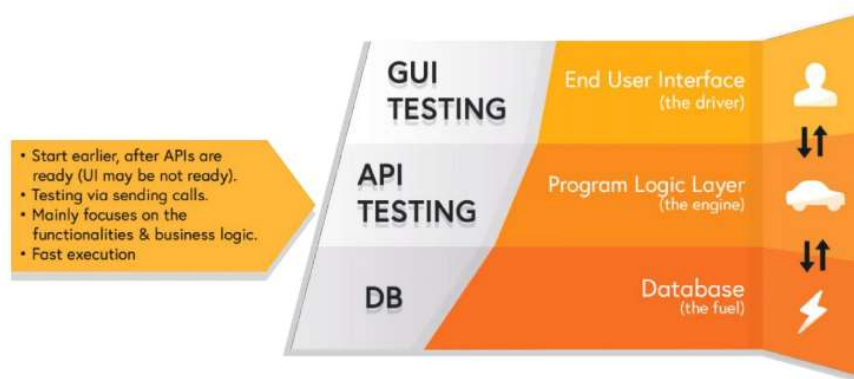
Table of Contents

Abstract	3
What is API Testing?	3
Introduction.....	4
Industry trends on API Test Automation.....	4
API Automation Testing Challenges and solution	4
So, why use Karate?	5
Key differences between Karate vs Traditional BDD	6
How do I get started with Karate Framework?	6
How to run the tests?	10
Advanced usage of Karate:	11
Test Execution Reports	13
Observations	14
Limitations:	14
Case Study – Refinitiv - SWIFT NEWS	15
Problem:.....	15
Solution:.....	15
Benefits	15
Appendix:	15

Abstract

What is API Testing?

API stands for Application Programming Interface. Typically, an API is used to facilitate an interaction between two different applications by using any means of communication. When these API(s) are used over the web networks, we call them as “Web Services”. In any application, API(s) are developed to communicate with a database or run some core business logic. Hence, it is important for a QE to test the API(s) for maximum test coverage or discover any defects. The below picture demonstrates where API testing fits in.



There are many types of API testing like Unit level, functional, security etc., but Functional API testing is more important for an application, as it primarily focused on the functionality of the API. In a typical functional testing, a test case includes verification of HTTP response code, validation of response content and error codes in case if API returns any error.

There are primarily two types of WebServices/API protocols.

1. REST [**R**epresentational **S**tate **T**ransfer]
2. SOAP [**S**imple **O**bject **A**ccess **P**rotocol]

REST is a simple and lightweight architecture and whereas SOAP is a heavy weight due to its extensive used standards and XML.

Introduction

Industry trends on API Test Automation

There are many tools out there in the market, which are quite popular for API Automation. Few are listed below:

1. SOAPUI
2. PostMan
3. RestAssured
4. Karate Framework

SOAPUI is a very popular tool for testing the APIs. It is predominantly used to test the SOAP web services.

PostMan is free and open source tool which helps you to test the APIs. It is available as plugin and as a desktop tool.

RestAssured, is an open source java based client library that is used to test RESTful web services. This library has methods, which can be called directly for fetching responses in JSON/XML format or any other supported formats.

Karate Framework, is an open source API automation framework based on Cucumber. It is built on top of Cucumber-JVM and allows you can run tests and generate reports like any standard java project. Any technical or non-technical testers can write tests in BDD style without knowing the code. Gherkin syntax like 'Given, When and Then' statements are used to create test scenarios.

BDD approach brings in collaboration between Business stakeholders, QA team and developers.

The best part is, this framework allows you to write and execute API test without writing a single line of code.

API Automation Testing Challenges and solution

What tool(s) should I use for automation of web services?

Not all tools are created equal. For example, **SOAPUI** is a desktop-based tool primarily used to test SOAP web services and it has no adequate support to run the tests via CI/CD or on the cloud. However, there are few options, which you can achieve cloud testing with SOAP but those are commercial based.

With **Rest Assured**,

- You need to keep implementing the test code aka step-definition methods as your functionality grows and may get tedious to maintain when it comes to dependency injection.

- There is no direct support to run any dynamic data driven tests.
- No direct support for parallel execution of tests.
- No option of running the setup or configuration steps only once. A typical step in a Background block of cucumber runs before each scenario
- No good test execution reports.

With Karate Framework, there is no compilation and programming knowledge required. All of the above limitations with RestAssured can be overcome with Karate.

So, why use Karate?

Karate framework helps to get rid of boilerplate code, which involves in writing API tests to make web services calls and validate the response. Its syntax is concise, clear and versatile. It has native support for both JSON and XML files i.e you do not need to write any additional parsers.

- Easily written in Gherkin syntax and even non-programmers can write tests.
- Comprehensive support for different flavors of HTTP calls like SOAP, HTTP/HTTPS, Websocket, FTP, HTTP proxy, JMS/MQ, JDBC [DB connections, Oracle/DB2]
- Powerful assertion techniques.
- Scripts are plain-text, require no compilation step or IDE, and teams can collaborate using Git / standard SCM
- It helps to construct most complex request-response operations with minimal effort.
- Can run all test scenarios in parallel and cut down the execution time.
- Supports schema validations and validates all payload values in one-step. This is the biggest limitation of REST assured.
- Supports File upload/multipart support, which is a partial/buggy in REST Assured.
- Supports Retry support i.e built-in feature where you can specify a condition to be polled.
- Supports to fetch the response from requests and use them as payload in subsequent web service calls.
- Built-in support for environment switching (qa, dev, pre-prod).
- Allows you to execute the scenarios on the cloud like Azure/AWS.
- You can also write custom code in java and use them in feature files
- Allows you to write the javascript code in a feature file
- Supports excellent reporting plugins like cucumber-reporting.
- Allows you to use the scenarios as a Gatling performance tests.
- Great documentation on github.

Key differences between Karate vs Traditional BDD

Karate Vs Traditional BDD

Less Code == Less Scripting Errors

Karate

```
Scenario: create, retrieve a pet
  Given url 'http://mysite.io/v2/pet'
  And request { name: 'Wolf' }
  When method post
  Then status 200
  And match response ==
    { id: '#number', name: 'Wolf' }
```

and ... that's all :)

Cucumber

```
Scenario: create, retrieve a pet
  Given pet endpoint is up
  And request where name is 'Wolf'
  When we send post request
  Then status is 200
  And returned JSON with id, name
```

+ steps
+ business logic
+ pojos

Karate Vs Traditional BDD

Simpler Code == Less Scripting Errors

Karate

```
Scenario: create, retrieve a pet
  Given url 'http://mysite.io/v2/pet'
  And request { name: 'Wolf' }
  When method post
  Then status 200
  And match response ==
    { id: '#number', name: 'Wolf' }
```

REST-assured

```
public void createPetTest() {
  given()
    .contentType(ContentType.JSON)
    .body("{\"name\": \"Wolf\"}")
    .when()
    .post("http://mysite.io/v2/pet")
    .then().statusCode(200)
    .body("id",
      Matchers.instanceOf(Integer.class))
    .body("name", equalTo("Wolf"));
}
```

How do I get started with Karate Framework?

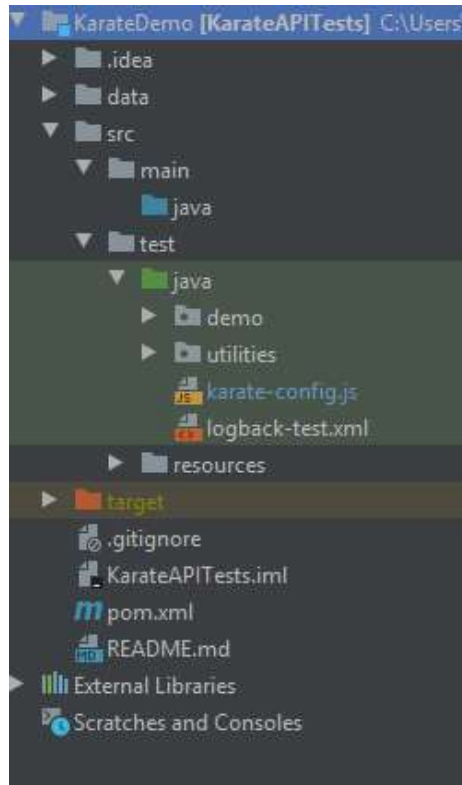
Karate supports Maven and Gradle. For simplicity, let us take maven archetype to generate a project

```
mvn archetype:generate -DarchetypeGroupId=com.intuit.karate -
DarchetypeArtifactId=karate-archetype -DarchetypeVersion=0.9.0 -
DgroupId=com.refintiv.demo -DartifactId=KarateTestDemo
```

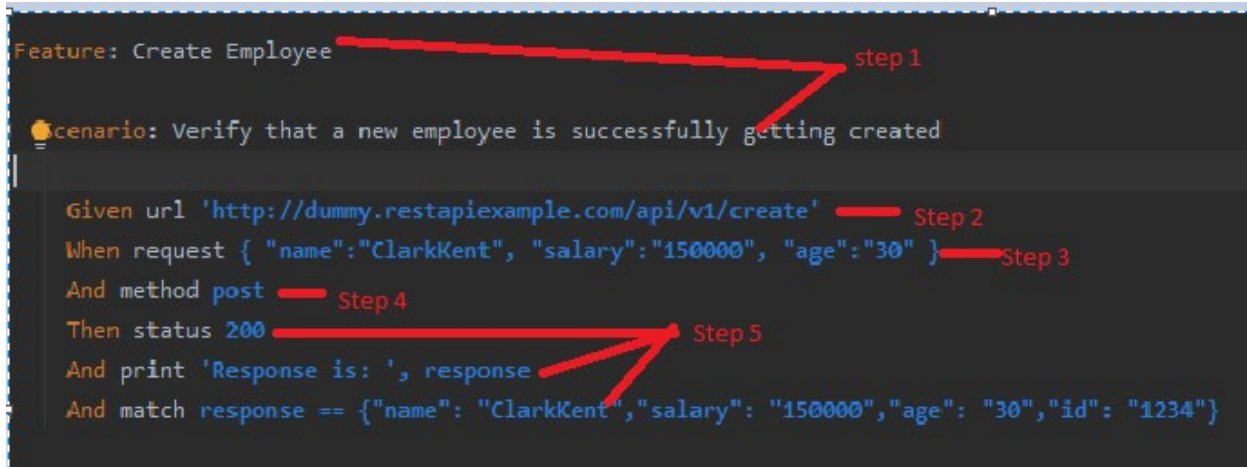
The directory layout is standard for Java Maven / Cucumber-JVM projects and as per the karate docs, it is recommended to use either Eclipse or IntelliJ IDEA for developing Karate tests.

Both IDEs offer support for JUnit and Cucumber, which Karate can leverage not only for editing but also for running tests. However, it is recommended to use IntelliJ IDEA as it provides a better syntax highlight and intellisense.

This how typical framework structure looks like after running the above mvn command. Please note that few folders like demo, utilities are created separately.



A simple example scenario looks like:



```

Feature: Create Employee

Scenario: Verify that a new employee is successfully getting created

  Given url 'http://dummy.restapiexample.com/api/v1/create'
  When request { "name": "ClarkKent", "salary": "150000", "age": "30" }
  And method post
  Then status 200
  And print 'Response is: ', response
  And match response == {"name": "ClarkKent", "salary": "150000", "age": "30", "id": "1234"}
  
```

The screenshot shows a Karate test scenario. Red arrows point from labels to specific parts of the code: 'step 1' points to the Feature and Scenario lines; 'Step 2' points to the 'Given url' line; 'Step 3' points to the 'When request' line; 'Step 4' points to the 'And method post' line; and 'Step 5' points to the 'Then status' line.

Step 1: Specifying the feature and scenario description [i.e a typical cucumber format]

Step 2: Setting up the endpoint URL in Given statement.

Step 3: Specifying the request payload in JSON format that need to be sent to the service.

Step 4: Specifying the type of HTTP request i.e POST here

Step 5: Validate the status code and match the response content from the service exactly matches with expected content. Print response statement, prints the response to the console.

Few other basic level test scenarios:

In the screenshot below, there are 2 tests which runs a HTTP GET method to check if the booking-Id response is correct and not null.

Karate framework supports **hamcrest** matchers & assertions. If you look at the 'Then' statement in the below scenarios, the response is being validated with matchers.

Background:

```
* url 'https://restful-booker.herokuapp.com'
```

Scenario: Verify that the `getbooking id` is working

```
Given path 'booking'
```

```
When method GET
```

```
Then status 200
```

```
* assert response!=null
```

```
And match $ contains {"bookingid":1}
```

Scenario: Verify that the `getbooking id` is working and contains our expected resp

```
Given path 'booking'
```

```
When method GET
```

```
Then status 200
```

```
And match $ == {"bookingid": "#notnull"}
```

Feature: sample karate test script

Background:

```
* url 'https://jsonplaceholder.typicode.com'
```

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

```
Given path 'users'
```

```
When method get
```

```
Then status 200
```

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

```
Given path 'users', first.id
```

```
When method get
```

```
Then status 200
```

All steps are quite concise. The Java implementation is essentially hidden from the tester (unless, of course, they want to look into the framework's lower levels). Furthermore, the scenario [screenshot above], shows how to use data from one response as the input for a second request. This is one of the excellent features available in this framework

- Given steps build requests
- When steps make request calls
- Then steps validate responses
- Catch-all steps (*) provide additional directives, like setting variables

How to run the tests?

As you see there is no `step_definition` code written for these scenarios. The easiest way to run all the tests without any additional configuration is through command line "**mvn test**". Karate uses maven surefire plugin under the hood and this plugin look for the java class file, which has 'test' keyword as either prefix or suffix.

```
C:\Users\...teframework\KarateDemo>mvn test
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.san.demo:KarateAPITests >-----
[INFO] Building KarateAPITests 1.0-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ KarateAPITests ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Users\...teframework\KarateDemo\src\main\resources
[INFO]
```

You can also run any feature file or specific cucumber tags via mvn command. For example, here is the command that runs a specific feature file

```
mvn test -Dcucumber.options="src/test/features/com/perspecsys/salesforce/featurefiles/Account.feature"
```

```
mvn test -Dcucumber.options="--tags @TestTag" [ to run feature file which has specific tags]
```

A test runner class can also be used to run specific feature file.

```
package demo;

import cucumber.api.CucumberOptions;
import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(
    features={"src/test/resources/features/belly.feature"},
    glue = "steps" //whatever the name of your steps package is
)
public class RunCukesTest
{
}

}
```

How to run test scenario in parallel and configure cucumber-reporting plugin?

```

public class TestKarateRunnerParallel {

    @Test
    public void testParallel() {
        System.setProperty("karate.env", "demo"); // ensure reset if other tests (e.g. mock) had set env in CI
        Results results = Runner.parallel(getClass(), threadCount: 5);
        generateReport(results.getReportDir());
        assertTrue(results.getErrorMessages(), condition: results.getFailCount() == 0);
    }

    public static void generateReport(String karateOutputPath) {
        Collection<File> jsonFiles = FileUtils.listFiles(new File(karateOutputPath), new String[] {"json"}, recursive: true);
        List<String> jsonPaths = new ArrayList(jsonFiles.size());
        jsonFiles.forEach(file -> jsonPaths.add(file.getAbsolutePath()));
        Configuration config = new Configuration(new File(pathname: "target"), projectName: "demo");
        ReportBuilder reportBuilder = new ReportBuilder(jsonPaths, config);
        reportBuilder.generateReports();
    }
}

```

Advanced usage of Karate:

Data Driven Test: One of the great options in Karate framework is, it allows to call the csv/json test data file directly under the Examples section of Scenario Outline i.e. the user is not required to give all the test data combinations in a feature file.

```

Feature: Demo on Karate Framework for Swift News for retrieving data through token authorization

Background:
  * url edsBaseUrl
  * header accept = 'application/json'

@EDSToken
Scenario Outline: Get request scenario with token generation
  Given path '/data/news/beta1/headlines'
  * def authToken = Java.type("Refinitiv.utilities.TestData").getTokenId()
  * print authToken
  * headers { Authorization: '#{authToken}' }
  And param query = '<param>'
  When method get
  Then status 200
  And match each response.data contains { storyId: '#notnull' }
  * assert response!=null
  # For demo purpose we are pausing the scenario to let the token to expire.
  * eval java.lang.Thread.sleep(15000)

Examples:
  | read('classpath:resources/data/EDSTokenTestData.csv') |

```

Dynamic Data Driven:

A complex dynamic data driven example, which reuses the feature file. In the below, the scenario outline calls a reusable feature file and runs it for various test combinations.

```
Feature: Data driven with Dynamic params

Background: Dynamic params

Scenario Outline: Verify that the user can create various booking ids with dynamic params
    Given path ''
    * def result = call read('resthelper_reusable.feature') { firstName:<firstName>,lastName:<lastName>}
    * match result.response.booking contains {totalprice: "#number"}

Examples:


| firstName | lastName |
|-----------|----------|
| Clark     | Kent     |
| Oliver    | Queen    |


```

In the Reusable feature file [screenshot below], the test data JSON file is being read and replaces the firstName & lastName with the values from the caller.

```
Feature: Create a booking Id
Background: Booking id creation
    * url 'https://restful-booker.herokuapp.com'
    * def payload = read('../data/testdata.json')

Scenario: Verify that the user can create a booking
    Given path 'booking'
    * payload.firstname = firstName
    * payload.lastname = lastName
    # * print payload
    And header Content-Type = 'application/json'
    * header Accept = "application/json"
    And request payload
    When method POST
    Then status 200
    * assert response!=null
    And results=response
```

```
{
  "firstname" : "#(fname)",
  "lastname" : "#(lname)",
  "totalprice" : 1134,
  "depositpaid" : true,
  "bookingdates" : {
    "checkin" : "2019-01-01",
    "checkout" : "2019-01-01"
  },
  "additionalneeds" : "lunch"
}
```

Call Java code in feature file:

The below example shows you how to call java code in a feature file.

'Java.type('utilities.StringUtil').isEmailValid(emailText)' takes emailText as an argument and validates if the email address is valid via java method and returns a boolean value.

Java code

```
package utilities;

public class StringUtil {
    // the method should be static here if you want to call this inside a scenario
    public static boolean isEmailValid(String text){
        return text.matches( regex: "^[A-Za-z0-9]+.[A-Za-z0-9]+@[A-Za-z0-9]+(?:de|com|in)");
    }
}
```

```
# Call Java code from a feature file
Feature: Java code from a feature file

@demo
Scenario: Get Number from response
    Given url 'https://reqres.in/api/users/2'
    When method GET
    Then status 200
    * def emailText = response.data.email
    * def flag = Java.type("utilities.StringUtil").isEmailValid(emailText)
    And assert flag==true
```

Test Execution Reports

Karate can generate JUnit reports by default but other reporting plugins like cucumber reports are also possible to integrate.

The below default JUnit report shows step-by-step log for each scenario. Full requests and responses are automatically logged which is quite helpful for debugging.



Test Suite Navigation

of failed tests: 0/16 (0.00%)

of skipped tests: 0/16 (0.00%)

of passed tests: 16/16 (100.00%)

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16		

Scenario Outline: [1.1.18] Verify that the user can fetch weather info of various US Cities

Test 1 : " url 'https://api.openweathermap.org/data/2.5/'	0.010169
Test 2 : Given path 'weather'	0.008712
Test 3 : " param zip = 94105+ 'us'	0.018041
Test 4 : " param appId = apiKey	0.000071
Test 5 : And header Accept = "application/json"	0.004842
Test 6 : When method GET	1.649361

```

19:13:35.319 request:
1 > GET https://api.openweathermap.org/data/2.5/weather?zip=94105%2Cus&appid=37fff23f32f904bacd4bc9078c7c38f
1 > Accept: application/json
1 > Accept-Encoding: gzip,deflate
1 > Connection: Keep-Alive
1 > Host: api.openweathermap.org
1 > User-Agent: Apache-HttpClient/4.5.5 (Java/11.0.3)

19:13:36.179 response time in milliseconds: 856.54
1 < 200
1 < Access-Control-Allow-Credentials: true
1 < Access-Control-Allow-Methods: GET, POST
1 < Access-Control-Allow-Origin: *
1 < Connection: keep-alive
1 < Content-Length: 545
1 < Content-Type: application/json; charset=utf-8
1 < Date: Thu, 12 Dec 2019 13:43:35 GMT
1 < Server: openresty
1 < X-Cache-Key: /data/2.5/weather?zip=94105%2Cus
1 < "coord":{"lon":-122.39,"lat":37.79},"weather":[{"id":500,"main":"Rain","description":"light rain","icon":"10n"},
{"id":701,"main":"Mist","description":"mist","icon":"50n"}],"base":{"stations","main":
{"temp":287.34,"feels_like":286.85,"temp_min":285.15,"temp_max":289.26,"pressure":1025,"humidity":100,"visibility":1287,"wind":
{"speed":2.6,"deg":190},"rain":{"1h":0.4},"clouds":{"all":90},"dt":1576158215,"sys":
{"type":1,"id":4321,"country":"US","sunrise":1576163743,"sunset":1576198261},"timezone":-28800,"id":0,"name":"San Francisco","cod":200}

Test 7 : Then status 200
Test 8 : " assert response!=null

```

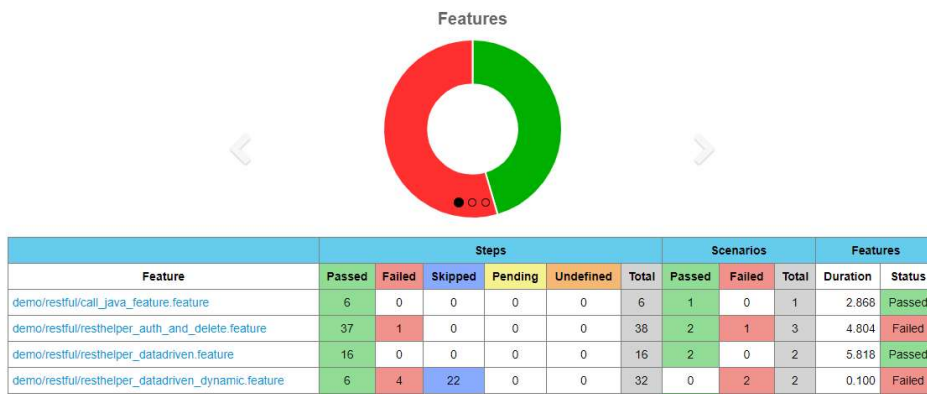
Scenario Outline: [1.2.19] Verify that the user can fetch weather info of various US Cities

Cucumber Report

Features Tags Steps Failures

Features Statistics

The following graphs show passing and failing statistics for features



Observations

- Any scenario that involves complex authorization token generation that expires every few minutes, Karate framework may not support well. A customer reusable utility is written to generate the auth token. This custom utility can be used in any standard java project as well.

Limitations:

Although there are no limitations noticed with Karate, but here are few issues/bugs with Karate framework.

<https://github.com/intuit/karate/issues>

Case Study – Refinitiv - SWIFT NEWS

SWIFT NEWS team had few issues with current API test framework and took 3~4 days to have an API test framework up & running.

Problem:

- Token generation, which expires every 5 minutes, is a bit challenging with the existing API framework.
- Takes at-least 6~8 hours to implement a simple or medium type functional test script.

Solution:

The team was able to solve the aforementioned problems with Karate framework easily. Using Karate,

- It was quick to bring an API framework along with simple & complex test script in a day.
- Took ~ 1.5 hours to write a complex test scenario like auth-token generation.

Benefits

Average Performance Execution Metrics Existing vs Karate Framework

60% faster with Karate framework.

Scenario	# of Tests	Execution Time with existing Framework (in secs)			Execution Time with Karate Framework (in secs)				
		Run 1	Run 2	Average	Run 1	Run 2	Run 3	Run 4	Average
Scenario 1	5	14.28	13	13.64	10.77	8.23	11.04	8.92	9.74
Scenario 2	26	50.74	45	47.87	44.52	25.26	22.59	22.85	28.8

80% effort savings is observed with Karate framework

	Complex Test Case	Comments
Standard Java Project	8 hours	A tester has spent approximately 8 hours of time to code/develop a complex test scenario
Karate Framework	1.5 hour	A tester has spent 1+ hour's time to complete a complex TC with Karate

Appendix:

Sources Referred:

<https://automationpanda.com/2018/12/10/testing-web-services-with-karate/>

<https://github.com/intuit/karate>