Karate

Karate maven versions

<!-- https://mvnrepository.com/artifact/com.intuit.karate/karate-core -->

<dependency>

<groupId>com.intuit.karate</groupId>

<artifactId>karate-core</artifactId>

<version>1.4.0</version>

</dependency>

Karate maven test cases

<!-- https://mvnrepository.com/artifact/com.intuit.karate/karate-junit5 -->

<dependency>

<groupId>com.intuit.karate</groupId>

<artifactId>karate-junit5</artifactId>

<version>1.4.0</version>

<scope>test</scope>

</dependency>

Generate project

mvn archetype:generate -DarchetypeGroupId=com.intuit.karate -DarchetypeArtifactId=karate-archetype -DarchetypeVersion=0.9.5 -DgroupId=com.mycompany –DartifactId=KarateUdemy

### Reference links

For assertions

<https://priyankab85.medium.com/cheat-sheet-for-karate-assertions-and-matching-d248383546e0>

GIT hub links- sample project

<https://github.com/bondar-artem/angular-realworld-example-app/tree/node18compatible>

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

<https://github.com/karatelabs/karate/blob/master/karate-demo/src/test/java/demo/schema/schema.feature>

karate with kafka- it is clearly we have samples here

<https://github.com/karatelabs/karate-examples/tree/main/kafka>

<https://github.com/Sdaas/karate-kafka>

For karate project template

<https://github.com/karatelabs/karate-template>

For karate sample projects

<https://github.com/karatelabs/karate-examples>

<https://github.com/karatelabs/karate-examples/blob/main/README.md>

<https://github.com/karatelabs/karate-todo/>

java links

<https://github.com/karatelabs/karate#java-api>

Live application urls

<https://angular.realworld.io/>

1. to login  
   <https://api.realworld.io/api/users/login>

{

    "user": {

        "email": "vv.manideeep1@gmail.com",

        "password": "MANIdeep@123"

    }

}

login cred- [vv.manideep1@gmail.com](mailto:vv.manideep1@gmail.com) pass- MANIdeep@123

1. to Fire post request

https://api.realworld.io/api/articles

<https://conduit.productionready.io/api/tags>

https://api.realworld.io/api/tags

https://api.realworld.io/api/articles?limit=10&offset=10

Karate keywords

Karate have many keywords to identify all those keywords are present in below

Remember after Gherkin keywords (Given, when, Then) we have to use these karate keywords

Exl:- Given url – here url is karate keyword

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

HTTP- for http module below are the keywords

url | path | request | method | status | soap action | retry until | response

Keywords in Assert module

match == | match != | match contains | match contains only | match contains any | match contains deep | match contains only deep | match !contains | match each | match each contains deep | match header | Fuzzy Matching | Schema Validation | contains short-cuts

Maven build path setting

Generally when u build all the java files will be compiled to .class files, but these feature files from src/test/resources

Can not be compiled hence these files are generally not moved to target folder, but when u keep the below entry those

Feature Files will be moved to target folder.

If u don’t do this, feature files will not be added to target folder & classpath

Only after doing this, feature files will be moved to target folder and these files will be recognized

<https://github.com/karatelabs/karate#folder-structure>

<build>

<testResources>  
 <testResource>  
 <directory>src/test/java</directory>  
 <excludes>  
 <exclude>\*\*/\*.java</exclude>  
 </excludes>  
 </testResource>  
 </testResources>  
</build>

Gradle settings

sourceSets {

test {

resources {

srcDir file('src/test/java')

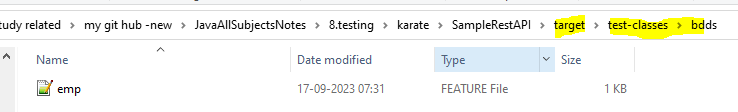
exclude '\*\*/\*.java'

}

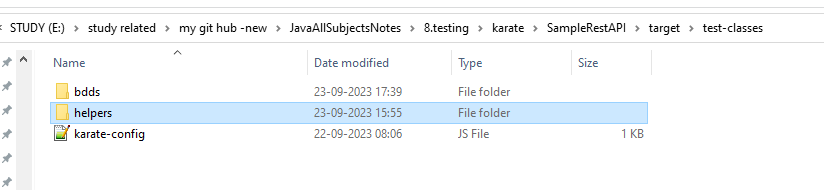
}

}

See here as per below image, all the feature files are moved to target due to above step



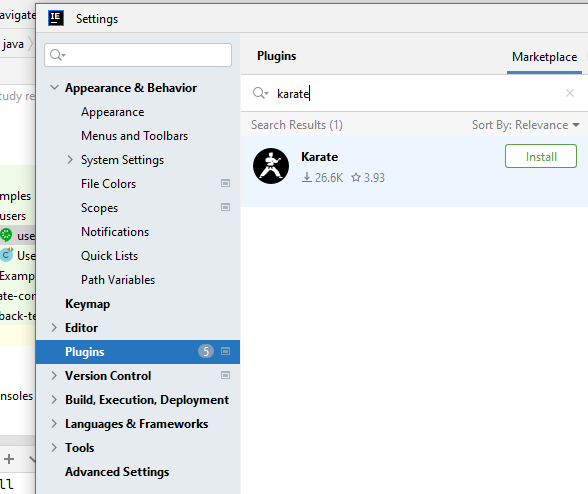
Remember all ur json files, test classes, all resources required to run BDD should come under this “test-classes” folder



Install karate plugin

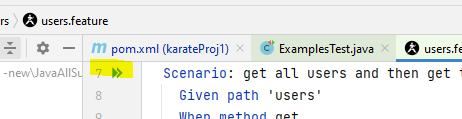
Install karate and gherkin plugins in intellj,

To install plugin🡪 go to file🡪 settings🡪 plugins🡪 search for all those plugins



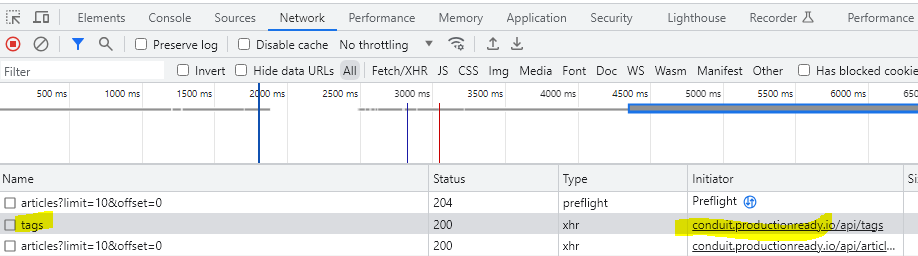
.feature files are karate test files

Run feature file- install all the required gherkin and karate plugins

here in the picture click left side play button to run all

the scenarios or which ever scenario u want

Finding the urls that get data



Xhr (xml http request ) are the request that fetch data from server

Feature files

#### Remember

Remember while writing scenario files ,

* 1 feature will have many scenarios, without feature keyword we cannot write scenario’s at all, so in every

Feature file writing feature keyword is mandatory

* after given/when/then keywords immediately first always we have to write karate keywords like url.. only

, after those only we have to write data

Ex:-

|  |  |
| --- | --- |
| When method Get | This will work fine, because after BDD keyword When , we have karate keyword method |
| When Get method | This will throw error, because karate keyword method, must always be first, not at last |

* if u write keyword at last then it wont detect the keyword

ex:- Given url ‘some url’ -- here url is a keyword

* Background is the one which is going to be called each and every method
* While sending data in request body , we should enclose in single quotes as like below

And request { "user" : "mani", "password" : "santu" }

* To fetch the values from karate-config.js and all then we should use “#(some variable name)”, this is same like in spring

We fetch in property like “${variable name}”

* In karate prefer single quotes
* We can set the values to global object called “config” based on the environment value passed form –Dkarate.env=”dev”

And we can fetch using “#(variable name defined in karate-config.js file)”

While Asserting

1. Enclosing ‘#(any variable)’ in single quotes-

|  |  |
| --- | --- |
| single quotes lo hash of  ‘#(any var name)’ | double quotes lo dollar of  “${}” |

And match response[9].empname == '#(ename)'

By default, we should enclose #() in single quotes, same like in spring “${}” – double quotes lo dollar

While asserting , when we are using #() inside triple quotes, we don’t need to enclose #() again in single quote

#### Famous errors

1. “no step-definition method match found for: staus 200”

When u get no step definition found, means some syntax u wrote wrong,

Even it could be space issue

Ex: def var= ‘mani’ 🡪 here if u see there is no space between var and =

**Ex:2** Then staus 200

Here in above status , T is missing hence it thrown “no step-definition method match found for: staus 200”

1. Dangerous space issues

\* def featureResponse = call read ('classpath:bdds/reUsableGlobalFeature.feature')

If there is any space issues , it wont even tell the error, it will simply wont call the feature at all,

We will think of build issues… feature file also would have been already present in target/test-classes

It wont even tell the error also properly

**So make sure no spaces between that**

|  |  |
| --- | --- |
| Issue in code | Root cause-solution |
| def var= ‘mani’ | No space between r&= |
| \* def credentials= read('cla...ntials.json') | Issue- no step definition found – means here no space between **s and =**  Solution:- put some space after s and before =  As **def credentials =** |
|  |  |
|  |  |

Invoking feature files from java

1. We can invoke feature files from java api /junit

<https://github.com/karatelabs/karate/blob/master/karate-demo/src/test/java/demo/java/JavaApiTest.java>

import com.intuit.karate.Runner;

import java.util.HashMap;

import java.util.Map;

import org.junit.jupiter.api.BeforeAll;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.\*;

@Test

void testCallingClasspathFeatureFromJava() {

Map<String, Object> args = new HashMap();

args.put("name", "World");

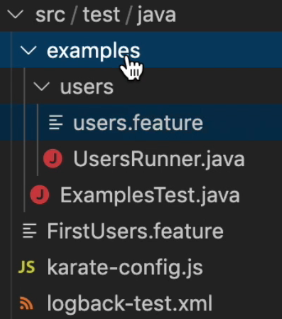
Map<String, Object> result = Runner.runFeature("classpath:demo/java/from-java.feature", args, true);

assertEquals("Hello World", result.get("greeting"));

}

U can even pass the values to feature files

1. Have a global runner which will run all the feature files present in the current package

1) This main runner is also like spring component scan, it will scan only those classes

present in current package and sub packages.

Here also ExamplesTest is the main runner which will look for the feature files present in

The current package and sub packages, here it will look for the feature files like

“users.feature” not “FirstUsers.feature” because this file is outside

1. Mvn test// same like junit, when u run mvn test all the test files will be executed

In the same flow all those feature files will be executed

1. Via intellij plugin also u can run the individual scenario or entire feature file- like running junit in intellij

Running only specific feature using tags

@ignore is the predefined tag

Way1:- running via junit – here @ is very very important

@Karate.Test  
public Karate testSpecificTags() {  
 return Karate.*run*().tags("@mani").relativeTo(getClass());  
}

Ensure scenario is already annotated with @anyname

@mani

Scenario: hit PLain Get method url

Given path '/tags'

Now only scenarios or features annotated with /tags only will get executed

Way2:- using maven command

mvn test –Dkarate.options=”--tags @mani”

Reports-results

U will find all the reports under target🡪 karate-reports🡪 src.test.java…html

Sample feature files

Generally one feature will have lot of scenarios

Remember while writing scenario files ,

* first always we have to write keyword after given/when/then , after those only we have to write data
* if u write keyword at last then it wont detect the keyword

ex:- Given url ‘some url’ -- here url is a keyword

1. Calling GET method without data

Scenario: test url working or not  
 Given url 'https://conduit.productionready.io/api/tags'  
 When method Get is invoked  
 Then status 200 is returned back from the server

1. Calling GET method with data- by passing the path parameters as per below url

<https://api.realworld.io/api/articles?limit=10&offset=10>

way1 – passing parameters as object

Scenario: hit url with query paramers

Given params {limit:*10*, offset:*10*}

Given url *'https://api.realworld.io/api/articles'*

When Get method

Then status *200*

**Using AND operator**

**Here we can pass data as json object**

Scenario: hit url with query paramers  
Given params {limit: 10 , offset:10}  
 And url 'https://api.realworld.io/api/articles'  
When method Get  
Then status 200

Using BACKGROUND and path keywords

Background keyword is used to write common statements in top, so that they will be executed every time for every scenario

‘path’ keyword is extension to url, here we kept base url in background and remaining subpath under

‘Path’ keyword

<https://api.realworld.io/api/articles?limit=10&offset=10>

<https://api.realworld.io/api/tags>

Feature: Api testing  
 Background: Define base url  
 Given url 'https://api.realworld.io/api'  
  
 Scenario: hit url with query parameters  
 Given path '/articles'  
 And params {limit:10, offset:10}  
 When method Get  
 Then status 200  
  
 Scenario: hit PLain Get method url  
 Given path '/tags'  
 When method Get  
 Then status 200

Assertions

Calling a post method with headers

https://api.realworld.io/api/users/login

login cred- [vv.manideep1@gmail.com](mailto:vv.manideep1@gmail.com) pass- MANIdeep@123

First we have to login to this url and get the token from result, then we have to pass that token as header for all post requests

Fire a post request to get the token

https://api.realworld.io/api/users/login

{

    "user": {

        "email": "vv.manideeep1@gmail.com",

        "password": "MANIdeep@123"

    }

}

Second post request number 2

<https://api.realworld.io/api/articles>

{

    "article": {

        "tagList": [],

        "title": "PAvan kalyan is hero ",

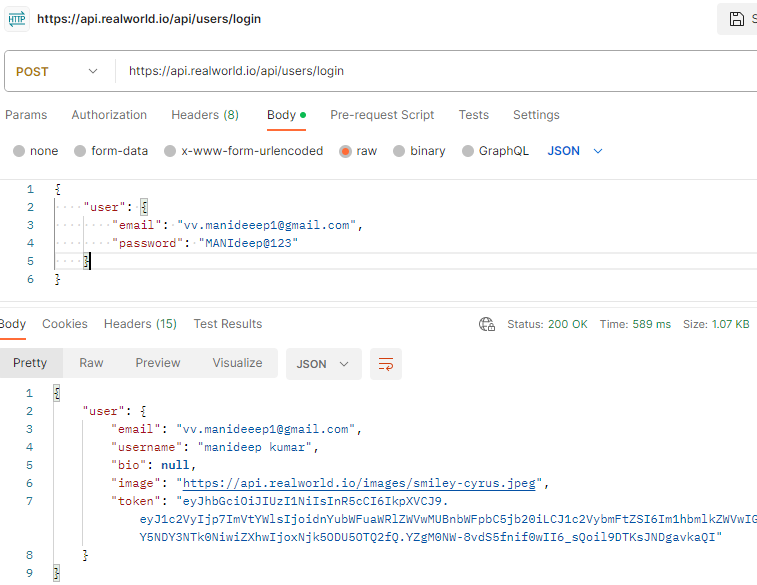
        "description": "chandrababu arrest in AP",

        "body": "chandrababu is a golden person"

    }

}

Add a header named “Authorization” and value “Token <your token from previous step>”



Related BDD script

Here we will call a post method with header and create a variable to store the output of the REST response

There should not be any spacing gaps in between = and all

Feature:demo the post method with header  
  
 Background: cfg the base url  
 Given url 'https://api.realworld.io/api'  
  
  
 Scenario: login to get the token and pass that token as header to publish the article  
 Given path '/users/login'  
 And request { "user": { "email": "vv.manideeep1@gmail.com", "password": "MANIdeep@123" } }

#when json request is big, then we should enclose that in triple double quotes “”” <ur json obj> “””  
 When method Post  
 Then status 200  
 \* def token = response.user.token  
  
 Given header Authorization = 'Token '+ token  
 Given path '/articles'  
 And request {"article": {"tagList": [],"title": "sai in hero","description": "chiru in hero","body": "cn"}}  
 When method post  
 Then status 201  
 And match response.article.title == 'sai in hero'

Calling a delete method

#delete method demo, by passing json as object

Scenario: Delete method demo

Given path '/deleteEmp'

And request { "empno": 11 }

And header Authorization = token

When method Delete

Then status 200

Profiles / env var

We can activate profiles using the command as mvn test –Dkarate.env=”dev”

for each and every environment url is different, so we will configure all those same like spring

In all environments **config** json object is available, we can prepare this object in karate-config.js as below

For every environment we can set the values to the json config object, we can fetch values from config object using “#()”

As Same like in springs “${}”

1. Sample from karate-config.js

* Here we will set values to config object,
* var **config** = {  
   apiUrl : 'http://localhost:8080/employeeApi/'  
   }  
   if (env == 'dev') {  
   config.usernameFromKarateJs = 'mani'  
   config.passwordFromKarateJs = 'santu'  
   } else if (env == 'e2e') {  
  config.usernameFromKarateJs = 'maniv1'  
   config.passwordFromKarateJs = 'santuv1'  
   }



1. The values set to config object can be fetched in feature files using “#()”, here variable called

“usernameFromKarateJs” is configured in karate-config.js file, we fetched that var in feature file using “#()”

Feature: hit my local api methods  
 Background: cfg base url  
 Given url apiUrl  
 And path '/login'  
 And request { "user":"#(usernameFromKarateJs)", "password":"santu" }  
 When method Post  
 Then status 200  
 \* def token = response.token

Calling one feature from another feature file

@Before vs @BeforeEach

If u want to execute some code before all features, then write those statements in js file



Re-usable feature

------------

Feature: writing a reusable feature  
Scenario: Reusable feature -login to get the token  
 Given url 'http://localhost:8080/employeeApi/'  
 And path 'login'  
 *# we can load the json credentials from json file instead of  
# And request { "user" : "mani", "password" : "santu" }* \* def credentials = read('classpath:/bdds/json/credentials.json')  
 And request credentials  
 When method Post  
 Then status 200  
 \* def token = response.token  
 \* print 're-usable feature executed successfully'  
 Scenario: call java method and get name of employee  
 \* def EmpFakerClass = Java.type('helpers.EmpFaker')  
 \* def empObj = new EmpFakerClass ();  
 \* def empName = empObj.getEmployeeName()  
 \* print 'reusable scenario exec successfully and sending empnname as' + empName

Feature which called above feature

----------------

Feature: hit my local api methods  
 Background: demo for @Before and @BeforeEach  
 *# when we used call method, since we kept this in background, this will be executed for every scenario* \* def featureResponse = call read('classpath:bdds/reUsableGlobalFeature.feature')  
  
 *#as we used CALLONCE() method,eventhough we used placed this in background,this method will be cached and  
 # this method will be executed only once  
# \* def featurecalledOneTime = callonce read('classpath:bdds/reUsableGlobalFeature.feature')* \* def ename = featureResponse.empName  
 Scenario: s1  
 \* print 'scenario 1 exec and received name as '+ ename  
 Scenario: s2  
 \* print 'scenario 2 exec and received name as '+ ename

Creating variables in feature files & using them there itself

Feature: Create and use the variables in feature files  
 Background:  
 *#this apiurl is a variable created in js file* Given url apiUrl  
 And path '/login'  
 And request  
 """  
 {  
 "user": "mani",  
 "password": "santu"  
 }  
 """  
 When method Post  
 Then status 200  
 \* def token = response.token  
  
 Scenario: create variable and use it  
 Given path '/saveEmp'  
 *# here we created a variable and we will use these variables using #()* And def userData = { "tcsempnum": 2000639, "tcsempname": "Manideep tata", "tcsmarks": 222 }  
 And header Authorization = token  
 And request  
 """  
 {  
 "empno": #(userData.tcsempnum),  
 "empname": #(userData.tcsempname),  
 "marks": #(userData.tcsmarks)  
 }  
 """  
 When method post  
 Then status 200  
 And match response[7] contains {"empname": #(userData.tcsempname)}

Configure global variables in js

The lines placed in this file will be executed only once for all the feature files

If u want to execute some code before all features, then configure in js file

Given path '/saveEmp'  
And header Authorization = token

If u see above, we may have requirements to pass the headers to each and every request

Instead of passing in each request, we can set those headers in karate-config.js itself

As like below only

var accesstoken = karate.callSingle('classpath: bdds/empCrud.feature',config).token  
karate.configure('headers',{Authorization : 'token'+accesstoken})

Note: above is not working for now in my intellij

when this got executed for first time, it will configure those headers and pass those in each and every request



Assertions

Keywords in Assert module

Here like in junit, instead of “assert” we have to use “match” keyword

match == | match != | match contains | match contains only | match contains any | match contains deep | match contains only deep | match !contains | match each | match each contains deep | match header | Fuzzy Matching | Schema Validation | contains short-cuts

<https://api.realworld.io/api/articles?limit=10&offset=10>

**Response is   
{**"articles": **[{},{} …10 such objects] ,** "articlesCount": **197}**

this will give 10 json objects

<https://api.realworld.io/api/tags>

Response is

**{**"tags": **[**"welcome","implementations","introduction","codebaseShow","ipsum","qui","cupiditate","et","quia","deserunt"**]}**

Examples

1. Asserting the size of array and
2. Asserting the exact value

Feature: Api testing  
 Background: Define base url  
 Given url 'https://api.realworld.io/api'

Scenario: hit url with query parameters https://api.realworld.io/api/articles?limit=10&offset=1  
 Given path '/articles'  
 And params {limit:10, offset:10}  
 When method Get  
 Then status 200  
 *# this returns json whose key is articles and value is array of objects whose size is 10* And match response.articles == '#[10]'  
 *# this is asserting a field* And match response.articlesCount == 197  
  
Scenario: hit PLain Get method url  
 Given path '/tags'  
 When method Get  
 Then status 200  
 *#the below asserts tags value is having 10 objects* And match response.tags == '#[10]'  
 And match response.tags contains ['welcome']  
 And match response.tags contains 'ipsum'  
 And match response.tags !contains 'orayya'  
 *# asserting the output datatype is array* And match response.tags =='#array'  
 *# loop and assert each element as string*And match each response.tags == '#string'

And match response[7] contains {"empname": #(userData.tcsempname)}

Note: Same like in junit as we have “assert” keyword, here we have “match” keyword

1. Asserting an array value – if the output is array of string as below

Sample output:- [ "manideep", "santu", "sailu", "charan", "ramadevi", "ranga rao" ]

Scenario: Assert emp names  
 Given path '/getEmpNames'  
 When method Get  
 Then status 200  
 And match response contains ["ranga rao"]

And match response !contains ["anushka"]

1. Asserting the data types

When response is array of strings, then we can assert response as array data type

Sample output:- [ "manideep", "santu", "sailu", "charan", "ramadevi", "ranga rao" ]

Refer this file for complete feature



##### Asserting arrays

Feature: assertions  
 Background: cfg base url  
 *#this base url is already configured in the js file*Given url apiUrl  
 Scenario: asserting array  
 Given path '/getEmpNames'  
 When method Get  
 Then status 200  
 *# here the output of the endpoint is like --> [ "manideep", "ramadevi", "ranga rao" ]  
 # Asserting the array count as 6* And match response == '#[6]'  
 *#Here response is array of strings, Assert the output array as array data type* And match response == '#array'  
 *# here response is array of strings, so assert each element in array as String data type* And match each response == '#string'  
 *# assert 1 element data as manideep* And match response contains "manideep"  
 And match response !contains "ramadevi1"  
 *# asserting 2 elements - here in this case both elements must be present* And match response contains ['manideep',"ramadevi"]  
 *# asserting any one in the list, means here we gave 2 ,but actual response should contain atleast one ele what is present here* And match response contains any ["manideep","eleWhichIsNotPresent-Luxembourg"]  
 *# to assert all the elements in the arry and it should not have any elements beyond it* And match response contains only [ "santu", "manideep","sailu", "charan", "ramadevi", "ranga rao" ]  
 *# asserting first element as manideep* And match response[0] == "manideep"  
 *# assert that first element must not be santu* And match response[0] != "santu"

*#sample output of that end point /getStudentNames is  
 #[  
 # { "studentName": "manideep", "marks": 101, "addrs": { "cityname": "luxemborg", "countryName": "europe" }, "gender": "male" },  
 # { "studentName": "santoor", "marks": 201, "addrs": { "cityname": "hyderabad", "countryName": "india" }, "gender": "female" },  
 # { "studentName": "charan", "marks": 301, "addrs": { "cityname": "chicago", "countryName": "America" }, "gender": "male" },  
 # { "studentName": "sai", "marks": 100, "addrs": { "cityname": null, "countryName": "America" }, "gender": "male" } ]  
 # ]* Scenario: Asserting array with inner objects  
 Given path '/getStudents'  
 When method Get  
 Then status 200  
 *# this api will give us array of 4 objes , assert arrray size or count* And match response == '#[4]'  
 *#assert first ele json value as string* And match response[0].studentName == '#string'  
 *#assert each object student name data type as string* And match each response..studentName == '#string'  
 And match response[0].studentName == 'manideep'  
 *#iterate through all objects of an array and atleast one obj it should contain 100 marks  
 # response[0] means 0th o index object, here response[\*] means every element inside an array- response[\*].marks gives [101,201,301,100]* And match response[\*].marks contains 100  
 *# among 4 objects in that array 1 obj value can be null,* And match response[\*].addrs.cityname contains null  
 *# matching inner objects with doubledot  
 # we can also write above statement as* And match response..cityname contains null  
 *# we want to compare each and every to male, if we use response[\*].gender-- this will give array of genders we cant compare array with single ele  
 # so to compare EACH AND EVERY ele we should use each operator whereas the response[\*].gender value is ["male","male","male","male"]* And match each response..gender == "male"  
 *# take gender list and it should not have any 'female' at all, here response[\*].gender -- will gives us the gender list* And match response[\*].gender contains "male"  
 *# take the gender list and it should contain male also ,* And match response[\*].gender contains "male"

1. Enclosing ‘#(any variable)’ in single quotes-

|  |  |
| --- | --- |
| single quotes lo hash of  ‘#(any var name)’ | double quotes lo dollar of  “${}” |

And match response[9].empname == '#(ename)'

By default, we should enclose #() in single quotes, same like in spring “${}” – double quotes lo dollar

While asserting , when we are using #() inside triple quotes, we don’t need to enclose #() again in single quote

Fuzzy matching

Fuzzy matching means – just comparing the type,

We won’t compare the output values only types

The supported markers are the following:

| **Marker** | **Description** |
| --- | --- |
| #ignore | Skip comparison for this field even if the data element or JSON key is present |
| #null | Expects actual value to be null, and the data element or JSON key *must* be present |
| #notnull | Expects actual value to be not-null |
| #present | Actual value can be any type or *even* null, but the key *must* be present (only for JSON / XML, see below) |
| #notpresent | Expects the key to be **not** present at all (only for JSON / XML, see below) |
| #array | Expects actual value to be a JSON array |
| #object | Expects actual value to be a JSON object |
| #boolean | Expects actual value to be a boolean true or false |
| #number | Expects actual value to be a number |
| #string | Expects actual value to be a string |
| #uuid | Expects actual (string) value to conform to the UUID format |
| #regex STR | Expects actual (string) value to match the regular-expression 'STR' (see examples above) |
| #? EXPR | Expects the JavaScript expression 'EXPR' to evaluate to true, see [self-validation expressions](https://github.com/karatelabs/karate#self-validation-expressions) below |
| #[NUM] EXPR | Advanced array validation, see [schema validation](https://github.com/karatelabs/karate#schema-validation) |
| #(EXPR) | For completeness, [embedded expressions](https://github.com/karatelabs/karate#embedded-expressions) belong in this list as well |

Example

*#sample output of that end point /getStudentNames is  
#[  
# { "studentName": "manideep", "marks": 101, "addrs": { "cityname": "luxemborg", "countryName": "europe" }, "gender": "male" },  
# { "studentName": "santoor", "marks": 201, "addrs": { "cityname": "hyderabad", "countryName": "india" }, "gender": "female" },  
# { "studentName": "charan", "marks": 301, "addrs": { "cityname": "chicago", "countryName": "America" }, "gender": "male" },  
# { "studentName": "sai", "marks": 100, "addrs": { "cityname": null, "countryName": "America" }, "gender": "male" } ]  
# ]*

Scenario: Fuzzy matching  
 *#refer above for output of this end point* Given path '/getStudents'  
 When method Get  
 Then status 200  
 *#both response..marks and response[\*].marks are same and gives array as output* And match response..marks contains 100  
 And match response[\*].marks contains 100  
 *#asserting each and every element as number datatype, here the values for marks are always number no nulls hence we used single hash* And match each response..marks == '#number'  
 *## match each and every objects cityname as string,but some objects have null , in that case we shold use ..* And match each response..cityname == '##string'  
 *# this says match each and every obj's cityname must be string, some can be null , so used double hash* And match each response[\*].cityname == '##string'

Schema valiadation

<https://github.com/karatelabs/karate#schema-validation>

<https://github.com/karatelabs/karate/blob/master/karate-core/src/test/java/com/intuit/karate/core/schema-like.feature>

\* def oddSchema = { price: '#string', status: '#? \_ < 3', ck: '##number', name: '#regex[0-9X]' }

\* def isValidTime = read('time-validator.js')

When method get

Then match response ==

"""

{

id: '#regex[0-9]+',

count: '#number',

odd: '#(oddSchema)',

data: {

countryId: '#number',

countryName: '#string',

leagueName: '##string',

status: '#number? \_ >= 0',

sportName: '#string',

time: '#? isValidTime(\_)'

},

odds: '#[] oddSchema'

}

"""

Example 2

Scenario: schemavalidation  
 \* def validator = read('classpath:bdds/time-validator.js')  
 Given path 'getAllEmployees'  
 When method Get  
 Then status 200  
 And match each response[\*] ==  
 """  
 {  
 "empno": '#number',  
 "empname": '#string',  
 "marks": '#number',  
 "doj": "#? validator(\_)"  
}  
 """

--related java function

<https://github.com/karatelabs/karate/blob/master/karate-core/src/test/java/com/intuit/karate/core/schema-like-time-validator.js>

function fn(s) {  
 var SimpleDateFormat = Java.type("java.text.SimpleDateFormat");  
 var sdf = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss");  
 try {  
 sdf.parse(s).time;  
 return true;  
 } catch(e) {  
 karate.log('\*\*\* invalid date string:', s);  
 return false;  
 }  
}

Calling java static & instance methods

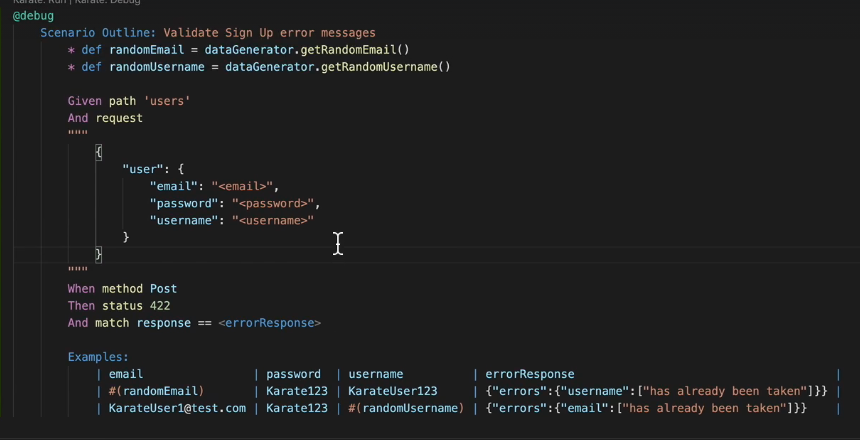


Feature: calling a java methods  
 Background: cfg base url and call static and instance method  
 Given url apiUrl

*# load the java class*

\* def faker = Java.type('helpers.EmpFaker')  
 *#here assume faker as class name  
  
 # here getLoginUserName() is a static method* \* def username = faker.getLoginUserName()  
  
 *# creating obj of class and call instance method* \* def obj = new faker()  
 \* def loginPass = obj.getLoginPassword()  
  
 And path '/login'  
 And request  
 """  
 {  
 "user":#(username),  
 "password":#(loginPass)  
 }  
 """  
 When method Post  
 Then status 200  
 And def token = response.token  
  
 Scenario: calling java instance and static methods  
 Given path '/saveEmp'  
 *# obj is object of that class, since getEmpNumber() is instance method we should call with obj reference* \* def enumber = obj.getEmpNumber()  
 \* def ename = obj.getEmployeeName()  
 *# here faker is class var reference name, so we can call with class name* \* def emarks = faker.getEmployeeMarks()  
 And header Authorization = token  
 And request  
 """  
 {  
 "empno": #(enumber),  
 "empname": #(ename),  
 "marks": #(emarks),  
 "doj" : "2023-09-22T07:33:53.647"  
 }  
 """  
 When method post  
 Then status 200  
 And match response[9].empname == '#(ename)'

Exec same scenario with diff i/p



Here if u see, the scenario outline, same scenario will be executed multiple number of times with diff different inputs

Which are provided under examples



Scenario Outline: run same scenario with multiple examples  
 Given path '/saveEmp'  
 And header Authorization = token  
 *# here we are calling java instance method to get the random employee name* And def ename = obj.getEmployeeName()  
 And request  
 *## here the values for <eno>, <emarks> will be given by examples section* """  
 {  
  
 "empno": "<eno>",  
 "empname": #(ename),  
 "marks": "<emarks>"  
 }  
 """  
 When method post  
 Then status 200  
 And match response[<index>].empname == '#(ename)'  
 And match response[<index>].empno == <eno>  
*# here now we gave 4 sets of data, now same scenario will be executed 4 times with all below inputs* Examples:  
 |eno | emarks |index  
 |7 |208 |6  
 |8 |209 |7  
 |9 |210 |8  
 |10 |211 |9

Reading data from files



Many times, we will have big json data , instead of hard coding that json in feature files,

We can keep that json in file and fetch it from feature file

Feature: demo to pull data from json files  
 Background: cfg base url and load data from feature files  
 Given url apiUrl  
 And path '/login'  
 *#here to login we will get the json from this json file instead of hard coding here* \* def credentials = read('classpath:bdds/json/credentials.json')  
 \* def payload = read ('classpath:bdds/json/employee.json')  
  
 *# since the same json will be loaded from file everytime we run the test case, here we can override with dynamic values  
  
 # load the java class* \* def empFakerClass = Java.type('helpers.EmpFaker')  
 *# create obj for this class to call instance methods* \* def empObj = new empFakerClass()  
  
 *# modify the payload with the random value generated by java method* \* set payload.empname = empObj.getEmployeeName()  
  
 And request credentials  
 When method post  
 Then status 200  
 And def token = response.token  
  
 Scenario: read json data from file & call java methods to modify it  
 Given path '/saveEmp'  
 And header Authorization = token  
 And request payload  
 When method post  
 Then status 200  
 *# assert the response with payload data with what we sent* And match response[10].empname == payload.empname

Printing values

1. Using karate.log(‘some log’) – generally this is used in javascript functions
2. Using print keyword

And def token = response.token  
 \* print token

\* print 'size after savings is ' +noOfEmpsBeforeSaving

Here print is the main keyword, we can use + operator also

Scenario: read json data from file & call java methods to modify it  
  
Given path '/saveEmp'  
 \* set payload.empno = noOfEmpsBeforeSaving +1  
 And header Authorization = token  
 And request payload  
 When method post  
 Then status 200  
 *# assert the response with payload data with what we sent* \* print 'size after savings is ' +noOfEmpsBeforeSaving  
 \* def nv = noOfEmpsBeforeSaving  
 \* print 'var nv value is '+nv  
 And match response['#(nv)'].empname == payload.empname



Debugging tips

1. 1st identify which line it came, if u cant find line number ,

Comment some piece of code and then try running the code and uncomment other piece, slowly narrow done

And find the line number

1. Don’t afraid if error come, read it carefully and find mistake

Assignment



Feature: assingments bro  
 Background: cfg base url and token generation  
 *#this url is configured in js file* Given url apiUrl  
 And path '/login'  
  
 *#read() is a java method , so to pass file name as string we should enclose in single quoted  
 # here we have placed credentials in json and we are loading to send as part of POST request* \* def credentials = read ('classpath:/bdds/json/credentials.json')  
 And request credentials  
 When method post  
 Then status 200  
 \* def token = response.token  
 \* print 'received token as -->' + token  
  
 Scenario: save new emp with existing modified data  
 Given path '/getAllEmployees'  
 When method Get  
 Then status 200  
 \* def resultObj = response[response.length -1]  
 \* print 'received result as '+ resultObj.empname  
 \* def existingCount = response.length  
  
 *#now we have to take the previous response and modify it and save it to the list by hitting POST request* Given path '/saveEmp'  
 *# fetch and modify values of previous object* \* def newEmpno = resultObj.empno +1  
 \* def newEmpMarks = resultObj.marks +1  
 *# to generate emp name we will call java class instance method and get a random name* \* def EmpFakerClass = Java.type('helpers.EmpFaker')  
 \* def empObj = new EmpFakerClass()  
 \* def newEmpName = empObj.getEmployeeName()  
  
 *# now with all above values we will modify the above json object* \* set resultObj.empno = newEmpno  
 \* set resultObj.marks = newEmpMarks  
 \* set resultObj.empname = newEmpName  
  
 And request resultObj  
 And header Authorization = token  
 When method post  
 Then status 200  
 \* print 'new count of employees is ' + response.length  
 And match response[10].empname == '#(newEmpName)'  
*# And match response.length == '20'*