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

GIT hub links- sample project

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

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

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

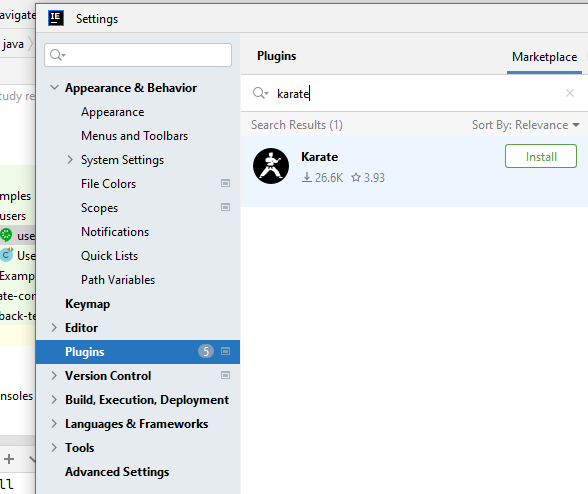
Keywords in Assert module

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

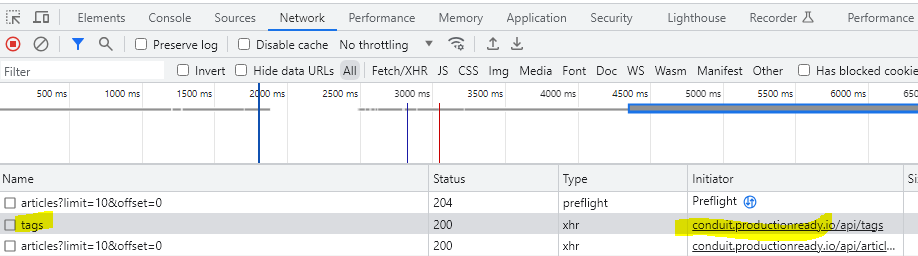
Install karate plugin

Install karate and gherkin plugins in intellj,

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



Finding the urls that get data

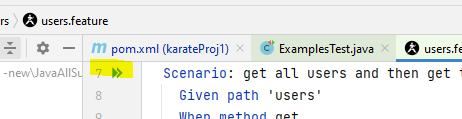


Xhr (xml http request ) are the request that fetch data from server

Feature files

.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

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");

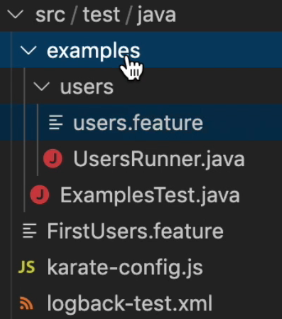
Map<String, Object> result = Runner.runFeature("classpath:demo/java/from-java.feature", args, true);

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

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

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

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'

Calling a post method

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 ",

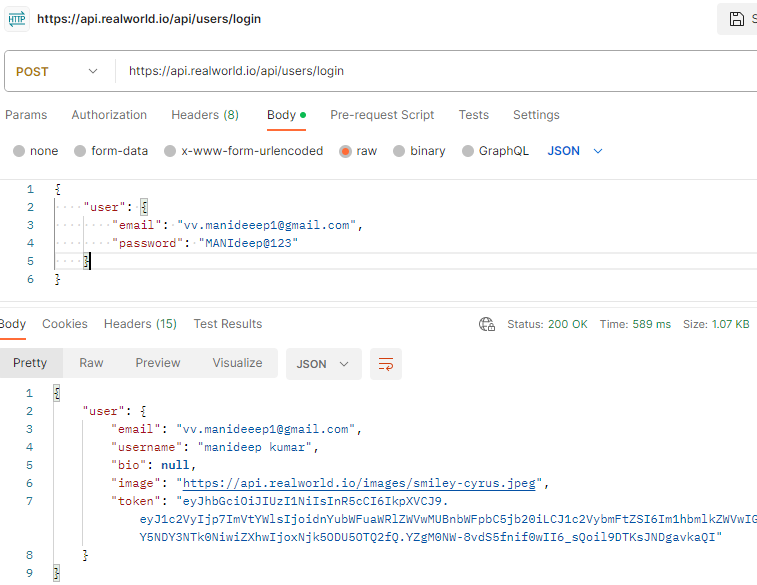
        "description": "chandrababu arrest in AP",

        "body": "chandrababu is a golden person"

    }

}

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



Related BDD script

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 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'