**RestAssured**

REST Api or RESTful Api is Representational State Transfer.

* RestAssured is an Java library that is designed to test/automate REST api. It is available in jar files.
* It supports only REST services and can’t support SOAP services.

For installation <https://www.youtube.com/watch?v=yDdBOspPp_c&t=844s>

* For RestAssured testing, we use Maven project as it downloads all the necessary jar files by just mentioning them in dependencies of pom.xml file.
* Dependencies that we need to add in pom.xml for Rest api testing.
  + RestAssured
  + json-schema-validator
  + json-path
  + xml-path
  + java-hamcrest
  + hamcrest-core
  + hamcrest-library
  + TestNG
  + Json-simple
  + Apache POI
* All these dependencies (code) can be downloaded from maven repository official site.
* APIs are available in the form of JAR files.

**Examples using TDD approach 🡪**

For requesting data from server. GET response 🡪

**package** asd;

**import** org.testng.annotations.Test;

**import** io.restassured.RestAssured;

**import** io.restassured.response.Response;

**import** io.restassured.specification.RequestSpecification;

**import** junit.framework.Assert;

**public** **class** getRe {

@Test

**public** **void** weatherData() {

// Request object creation

RequestSpecification re = RestAssured.*given*();

// Response object

Response response = re.get("http://restapi.demoqa.com/utilities/weather/city/Pune");

// Printing response

String responseBody = response.getBody().asString();

System.***out***.println(responseBody);

}

}

Steps for any RestAssured API testing 🡪

* Create request object (RequestSpecification)
* Create response object
* Extract response body from it

For posting on server, POST request 🡪

RequestSpecification re1 = RestAssured.*given*();

re1.header("Content-Type", "application/json");

JSONObject json = **new** JSONObject();

json.put("id", "25");

json.put("title", "VJ");

json.put("author", "Vijay");

re1.body(json.toJSONString());

re1.post("http://localhost:3000/posts");

For deleting something from server, DELETE request 🡪

RequestSpecification re2 = RestAssured.*given*();

re2.delete("http://localhost:3000/posts/21");//need to provide exact URL

For modifying existing data, PUT request (same as POST just write PUT request with modified data)🡪

RequestSpecification re3 = RestAssured.*given*();

re1.header("Content-Type", "application/json");

JSONObject json = **new** JSONObject();

json.put("id", "25");

json.put("title", "VJ");

json.put("author", "Vijay");

re3.body(json.toJSONString());

re3.put("http://localhost:3000/posts/25");

Different Validation points in RestAssured 🡪

* Status code validation
* Status line validation
* Validating Headers
* Success code validation
* Validating node values from JSON response
* Basic Preemptive Authentication

Note – For more details, please check compete programs

RequestSpecification re = RestAssured.*given*();

Response response = re.get("http://restapi.demoqa.com/utilities/weather/city/Pune");

// Status code validation

**int** statusCode = response.getStatusCode();

System.***out***.println("Status code is " + statusCode);

Assert.*assertEquals*(200, statusCode);

// Status Line validation

String statusLine = response.getStatusLine();

System.***out***.println("Status line is " + statusLine);

Assert.*assertEquals*("HTTP/1.1 200 OK", statusLine);

// Success Code validation

String successCode = response.jsonPath().get("successCode");

Assert.*assertEquals*("OPERATION\_SUCCESS", successCode);

// Validating Headers

String contentType = response.header("Content-Type"); // ‘Content-Type’ is one of the headers

Assert.*assertEquals*("application/xml; charset=UTF-8", contentType);

// Validating node value

JsonPath jsonpath = response.jsonPath();

Assert.*assertEquals*(jsonpath.get("Temperature"), "31 Degree celsius");

// Basic Authentication

PreemptiveBasicAuthScheme authscheme = **new** PreemptiveBasicAuthScheme();

authscheme.setUserName("ToolsQA");

authscheme.setPassword("TestPassword");

RestAssured.*authentication*=authscheme;

**Authentication in REST\_Api**

* Authentication is a process to prove that you are authentic person or not.
* Rest Assured supports several authentication schemes like OAuth, digest, certificate, form and preemptive basic authentication.
* Basic Authentication can be classified as Preemptive and Challenged.
* In Preemptive Authentication, along with login request credentials (username and password) are also send. So it reduces one extra call from server.
* This is not the case with Challenged Authentication.
* Mostly Preemptive Authentication is used.

@Test

**public** **void** test5() {

**int** code = RestAssured.*given*()

.auth().preemptive()

.basic("ToolsQA", "TestPassword")

.when()

.get("http://restapi.demoqa.com/authentication/CheckForAuthentication")

.getStatusCode();

System.***out***.println("Response code from server is " + code);

}

**RestAssured with BDD approach using Cucumber**

Dependencies that we need to add in pom.xml for BDD approach

* + rest-assured
  + json-schema-validator
  + json-path
  + xml-path
  + testng
  + java-hamcrest
  + hamcrest-core
  + hamcrest-library
* Complete structure is divided in three parts.
  + given()
    - set cookies
    - add authorization
    - add parameters
    - set headers info
  + when()
    - sending request (like Get, Post, Put, Delete)
  + then()
    - validate status code
    - extract response
    - extract headers
    - response body

Complete project is at C:\Users\VJ\Documents\API\_BDD

All videos are from <https://www.youtube.com/watch?v=n3UITFRJ9KU&list=PLig60kSF-huhMszQmp6GBA77hwnw0rJEm&index=3>

Get Request 🡪

**import** org.testng.annotations.Test;

**import** **static** io.restassured.RestAssured.\*;

**import** **static** org.hamcrest.Matchers.\*;

**import** **static** org.junit.Assert.*assertThat*;

**public** **class** getWeatherDetails {

@Test

**public** **void** getRequest() {

*given*()

.when()

.get("https://reqres.in/api/users?page=2")

.then()

.statusCode(200)

.statusLine("HTTP/1.1 200 OK")

.header("Content-Type", "application/json; charset=utf-8")

.assertThat()

.body("RestResponse.result.name", equalTo("India")); // single line validation, taken as example

.body("RestResponse.result.name", hasItems("India", "US", "France"); // multiple content validation

}

}

Note – For single item validation from response body we use ‘equalsTo()’ method whereas for multiple item validation we use ‘hasItems()’ method.

* While validating the response body, we can use ‘JSONPathFinder’ chrome extension.

POST Request 🡪

* Create a static map object inside a class
* Create @BeforeMethod and add data in map object.
* Inside the same above method define RestAssured.baseURI and RestAssured.basePath
* Create a @Test method having all given, when, then section
* Make sure given section contains contentType as ‘application/json’ and body as above created map object.

**public** **class** PostRequestExample {

**public** **static** HashMap *map1* = **new** HashMap();

@BeforeMethod

**public** **void** Data() {

*map1*.put("name", "VJ");

*map1*.put("job", "searching");

RestAssured.*baseURI* = "https://reqres.in/api";

RestAssured.*basePath* = "/users"; // provide path or query parameter

}

@Test

**public** **void** PostRequest() {

*given*()

.contentType("application/json")

.body(*map1*)

.when()

.post()

.then()

.statusCode(201);

.log().all(); // to see the generated logs in console

}

}

PUT Request 🡪

* While creating Put / Post request, we need to provide content type and body in the given section.
* PUT request is same as POST request, only difference is in the when we need to call PUT method instead of POST method.

DELETE Request 🡪

**public** **class** DeleteRequestExample {

@Test

**public** **void** DeleteRequest() {

RestAssured.*baseURI* = "https://reqres.in/api";

RestAssured.*basePath* = "users/2";

*given*()

.when()

.delete()

.then()

.statusCode(200)

.statusLine("HTTP/1.1 200 OK")

.log().all();

}

}

Storing the response and validating it 🡪

* We need to create Response object by calling .extract().response() method
* All the given, when, then should be written inside this Response object.

@Test

**public** **void** extractResponse() {

Response response =

*given*()

.when()

.get("https://reqres.in/api/users?page=2")

.then()

.statusCode(200)

.extract().response();

String jsonAsString = response.asString();

Assert.*assertEquals*(jsonAsString.contains("michael.lawson@reqres.in"), **true**);

}

Extracting node value from the response 🡪

{

"spec": {

"groups": [

{

"name": "book",

"title": "classic-books:1.0.2"

},.......

]

}

}

* response.body().jsonPath().get("spec.groups[i].title");

Sending parameters and headers with request 🡪

* We need to use .param() and .header() method in the given section.

**public** **class** ParamAndHeaders {

@Test

**public** **void** getRequest() {

*given*()

.param("MyName", "VJ")

.header("MyHeader", "Indian")

.when()

.get("https://reqres.in/api/users?page=2")

.then()

.statusCode(200);

}

}

**OAuth 2.0 Generating access token at runtime 🡪**

Access token is used for once only. So instead of generating it manually, we can write a program to generate it at runtime and use it.

Logic is to access the endpoint which is used for requesting access token and then extract token from its response body and use it in your regular request.

Video reference <https://www.youtube.com/watch?v=GMuFZMBZLg4&list=WL&index=5&t=202s>

Complete project is at C:\Users\VJ\Documents\API\_BDD\APITesting\_UsingBDD\src\test\java\RestAssuredTests\OAuth2.java

**public** **class** OAuth2 {

@Test

**public** **void** test1() {

Response rest = RestAssured

.*given*()

.formParam("client\_id", "Chick\_Feed")

.formParam("client secret", "b0084eb95a49cd77e46afd7f9f817950")

.formParam("grant\_type", "client\_credentials")

.post("http://coop.apps.symfonycasts.com/token");

System.***out***.println(rest.jsonPath().get("access\_token"));

String token = rest.jsonPath().get("access\_token");

Response rest1 = RestAssured

.*given*()

.auth()

.oauth2(token)

.post("http://coop.apps.symfonycasts.com/api/1529/chickens-feed");

System.***out***.println("Status code is" + rest1.getStatusCode());

System.***out***.println("Message " + rest1.getBody().asString());

}

}