**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.

**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

All videos are from <https://www.youtube.com/watch?v=OM4mr3PKgcQ&list=PLUDwpEzHYYLvLZX_QEGTNolPvNADXid0I&pp=iAQB>

Note – All Programs are at [E:\Study\API\_Testing](file:///E:\Study\API_Testing) location

Get Request 🡪

GET Request Approach 1 (without using Response variable)

package SDET;

import static io.restassured.RestAssured.\*;

import static org.hamcrest.Matchers.\*;

import org.testng.annotations.Test;

public class GET\_Method {

*@Test*

void get\_request() {

*given*()

.contentType("ContentType.JSON")

.when()

.get("https://reqres.in/api/users?page=1")

.then()

.statusCode(200)

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

.body("data[0].first\_name", *equalTo*("George"))

.log().all(); // to see the entire response body in console

}

}

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.

GET Request Approach 2 🡪

**Asserting the response by storing it in a Response variable**

Here we do not require *.then()* section

By storing the response in a variable we can perform multiple validations with help of condition statements, looping etc

Video - <https://www.youtube.com/watch?v=5fWDqLFbJnA>

package SDET;

import org.testng.Assert;

import org.testng.annotations.Test;

import io.restassured.response.Response;

import static io.restassured.RestAssured.\*;

public class Response\_Validation {

*@Test*

void getUsers()

{

Response res =

*given*()

.when()

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

// Printing the complete response body

System.***out***.println(res.getBody().asString()); // it prints the body in one line without indentation

System.***out***.println(res.jsonPath().prettify()); // it prints the same body in readable format

// Asserting Status Code using res.getStatusCode()

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

Assert.*assertEquals*(res.getStatusCode(), 200);

// Asserting Headers using res.getHeader(), need to pass header key as parameter

Assert.*assertEquals*(res.getHeader("Content-Type"), "application/json; charset=utf-8");

// Asserting the Response Body using res.jsonPath().get()

String name = res.jsonPath().get("data[0].first\_name").toString();

System.***out***.println("Name of user is " + name);

}

}

*Parcing the JSON Response 🡪 Traversing through json response to get required field using JSONObject.*

**Printing all the users from Response**

// printing all users from the response

JSONObject jo = new JSONObject(res.asString()); // converting response to JSON object

for (int i=0; i<jo.getJSONArray("data").length(); i++)

{

String first\_name = jo.getJSONArray("data").getJSONObject(i).get("first\_name").toString();

System.***out***.println(first\_name);

}

**Checking Whether specific user is present in response or not**

// Checking whether specific user is present in the response or not

boolean flag = false;

JSONObject jo1 = new JSONObject(res.asString());

for (int i=0; i<jo1.getJSONArray("data").length(); i++)

{

if (jo1.getJSONArray("data").getJSONObject(i).get("first\_name").equals("George"))

{

flag = true;

break;

}

}

Assert.*assertEquals*(flag, true);

**Finding the total sum of IDs**

// Finding sum of total ID numbers

int total\_id = 0;

for (int i=0; i<jo.getJSONArray("data").length(); i++)

{

String id = jo.getJSONArray("data").getJSONObject(i).get("id").toString();

total\_id = total\_id + Integer.*parseInt*(id);

}

System.***out***.println("Sum of total IDs is " + total\_id);

**POST Request**

There are multiple ways to pass data while creating POST request.

* + using HashMap
  + using org.json
  + using POJO (Plain Old Java Object), mostly used
  + using external json file

1. Creating POST request using HashMap 🡪

Request to be sent through Postman :

{

"name": "morpheus",

"job": "leader",

“skills”: “C”, “C++”, “Java”

}

package SDET;

import static io.restassured.RestAssured.\*;

import static org.hamcrest.Matchers.\*;

import java.util.HashMap;

import org.testng.annotations.Test;

public class POST\_HashMap {

*@Test*

void POST\_Using\_HashMap() {

HashMap data = new HashMap();

// JSONObject data = new JSONObject(); this is for approach 2

data.put("name", "VJ");

data.put("job", "Rocks");

String courseArr[] = {"C", "C++", "Java"};

data.put("skills", courseArr);

*given*()

.contentType("application/json")

.body(data)

// .body(data.toString()) this is for approach2, JSONObject

.when()

.post("https://reqres.in/api/users")

.then()

.statusCode(201)

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

.body("name", *equalTo*("VJ"))

.body("job", *equalTo*("Rocks"))

.body("skills[0]", *equalTo*("C"))

.log().all();

}

}

1. Creating POST request using org.json 🡪

It is exactly same as above request, just couple of points are different.

* Instead of creating HashMap, we have to create JSONObject.
* In the given section while giving body parameter, we have to send it by converting it to String.

1. Creating POST request using POJO 🡪

* In POJO approach, we have to create a separate class with getters & setters for all the variables.
* Then create the object of this above class in request class & access the setters method to set the variable values. Rest all other things are same.

POJO Class

package SDET;

public class POJO\_Class {

String name;

String job;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getJob() {

return job;

}

public void setJob(String job) {

this.job = job;

}

}

POJO POST Request

package SDET;

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

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

import org.testng.annotations.Test;

public class POST\_POJO {

*@Test*

void POST\_POJO\_Request() {

POJO\_Class data = new POJO\_Class();

data.setName("VJ");

data.setJob("Rocks");

*given*()

.contentType("application/json")

.body(data)

.when()

.post("https://reqres.in/api/users")

.then()

.statusCode(201)

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

.body("name", *equalTo*("VJ"))

.body("job", *equalTo*("Rocks"))

.log().all();

}

}

1. Using external json file 🡪

External ‘body.Json’ file

{

"name": "VJ",

"job": "Rocks"

}

package SDET;

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

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

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileReader;

import org.json.JSONObject;

import org.json.JSONTokener;

import org.testng.annotations.Test;

public class POST\_ExternalJson {

*@Test*

void ExternalJson\_Request() throws FileNotFoundException {

File f = new File(".\\body.json");

FileReader fr = new FileReader(f);

JSONTokener jt = new JSONTokener(fr);

JSONObject data = new JSONObject(jt);

*given*()

.contentType("application/json")

.body(data.toString())

.when()

.post("https://reqres.in/api/users")

.then()

.statusCode(201)

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

.body("name", *equalTo*("VJ"))

.body("job", *equalTo*("Rocks"))

.log().all();

}

}

Parameterizing Path & Query Parameter 🡪

Note : While sending the request, we need to refer parameterized path parameter only, Query parameter will go automatically with the request, no need to separately mention it.

package SDET;

import static io.restassured.RestAssured.\*;

import org.testng.annotations.Test;

public class Path\_And\_Query\_Parameter {

*@Test*

void PassPathQueryParams() {

// URL -> https://reqres.in/api/users?page=2&id=5

*given*()

.pathParam("myPath", "users") // path parameter

.queryParam("page", "2") // Query parameter

.queryParam("id", 5) // Query parameter

.when()

.get("https://reqres.in/api/{myPath}")

.then()

.statusCode(200)

.log().all();

}

}

Cookies 🡪

Note : Cookies value always keeps on changing. So for every GET request execution, you will get new cookie value.

To get the cookies, we have to store response in a Response variable & then use methods like res.getCookie(cookie\_name) & res.getCookies()

package SDET;

import static io.restassured.RestAssured.\*;

import java.util.Map;

import org.testng.annotations.Test;

import io.restassured.response.Response;

public class CookiesDemo {

*@Test*

void cookiesTest() {

Response res =

*given*()

.when()

.get("https://www.google.com/");

// Getting single cookie info

String cookie\_value = res.getCookie("AEC");

System.***out***.println("AEC Cookie value is " + cookie\_value);

// Getting all cookies info

Map<String, String> cookies\_info = res.getCookies();

System.***out***.println(cookies\_info.keySet()); // printing all cookies names

for (String k : cookies\_info.keySet()) // Printing all cookie name & values

{

String cookie = res.getCookie(k);

System.***out***.println("Value of " + k + " cookie is " + cookie);

}

}

}

Headers 🡪

Unlike cookies as some of the headers values remains same, we can use .header method in .then() section to validate

.then()

.header("Server", "gws");

To print all the header, we can simply write .log().headers() at the end in .then() section.

OR we can store them in response variable & then print.

void getHeaders() {

Response res =

*given*()

.when()

.get("https://www.google.com");

// Getting single header info

String header\_value = res.getHeader("Content-Type");

System.***out***.println(header\_value);

// Getting all Headers info

Headers myHeaders = res.getHeaders();

for (Header h : myHeaders)

{

System.***out***.println(h.getName() + " Header's value is " + h.getValue());

}

}

Logs 🡪

* If we want to print only response body in the logs then use ‘.log().body()’ in .then() section at the end.
* If we want to print only cookies in the logs then use ‘.log().cookies()’ in .then() section at the end.
* If we want to print only headers in the logs then use ‘.log().headers()’ in .then() section at the end.
* If we want to print everything in the logs then use ‘.log().all()’ in .then() section at the end.

**Authorization**

Authentication – checking user is valid or not by checking credentials.

Authorization - Checking access permissions / privileges

Types of authentication 🡪

* Basic
* Digest
* Preemptive
* Bearer token, most widely used
* OAuth 2.0
* API Key

All authentications should be provided in .auth() method under .given() section

1. **Basic / Digest / Preemptive Authentication**

URL - <https://postman-echo.com/basic-auth>

Username – postman

Password - password

package Authentication;

import static io.restassured.RestAssured.\*;

import static org.hamcrest.Matchers.\*;

import org.testng.annotations.Test;

public class BasicAuth {

*@Test*

void testBasicAuth() {

*given*()

.auth().basic("postman", "password")

// .auth().digest("postman", "password") for digest authentication

// .auth().preemptive().basic("postman", "password") for preemptive authentication

.when()

.get("https://postman-echo.com/basic-auth")

.then()

.statusCode(200)

.body("authenticated", *equalTo*(true))

.log().all();

}

}

Internal algorithm for basic, digest & preemptive are different. From user point of view they are almost same as we provide username & password for all of these types of authentication.

1. **Bearer Token**

Here we provide previously generated token as bearer token. We need to provide this token under request Headers section.

package Authentication;

import static io.restassured.RestAssured.\*;

import org.testng.annotations.Test;

public class BearerToken {

*@Test*

void testBearerToken() {

String token = "ghp\_5SqWgpoc1G7orVJIEUSLYc2wqLYgzB1TD9Gv";

*given*()

.headers("Authorization", "Bearer " + token)

// .headers("Authorization", "Bearer ghp\_5SqWgpoc1G7orVJIEUSLYc2wqLYgzB1TD9Gv") Token can be directly provided as we can see in Postman Headers section

.when()

.get("https://api.github.com/user/repos")

.then()

.statusCode(200)

.log().all();

}

}

1. **OAuth 2.0**

package Authentication;

import static io.restassured.RestAssured.\*;

import org.testng.annotations.Test;

public class OAuth2 {

*@Test*

void testOAuth2() {

*given*()

.auth().oauth2("ghp\_5SqWgpoc1G7orVJIEUSLYc2wqLYgzB1TD9Gv")

.when()

.get("https://api.github.com/user/repos")

.then()

.statusCode(200)

.log().all();

}

}

1. **API Key Authentication**

package Authentication;

import static io.restassured.RestAssured.\*;

import org.testng.annotations.Test;

public class ApiKeyAuth {

*@Test*

void testApiKeyAuth() {

*given*()

.queryParam("appid", "564a5e3c2372daab6adfb0230ef10b76") //appid is Api Key

.when()

.get("https://api.openweathermap.org/data/2.5/forecast/daily?q=Delhi&cnt=7&units=metric")

.then()

.statusCode(200)

.log().all();

}

}

Instead of sending the complete URL in given section, we can cut it down to path parameter & multiple query parameters.

void testApproach2() {

*given*()

.pathParam("myPath", "data/2.5/forecast/daily")

.queryParam("q", "Delhi")

.queryParam("cnt", "7")

.queryParam("units", "matric")

.queryParam("appid", "564a5e3c2372daab6adfb0230ef10b76")

.when()

.get("https://api.openweathermap.org/{myPath}")

.then()

.statusCode(200)

.log().all();

}

Generating the access token & passing it runtime 🡪

Note : In below program, ‘formParam’ are different than the parameters we send with normal request.

package RuntimeAccessToken;

import org.testng.Assert;

import org.testng.annotations.Test;

import io.restassured.response.Response;

import static io.restassured.RestAssured.\*;

public class OAuth2 {

*@Test*

void runtimeOAuth2() {

Response res =

*given*()

.formParam("client\_id", "TestAppVJ")

.formParam("client\_secret", "d5f278eb2383d382e4298ad3dbf6eaf2")

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

.when()

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

System.***out***.println(res.jsonPath().prettify());

String access\_token = res.jsonPath().get("access\_token").toString();

System.***out***.println("Token for authentication is " + access\_token);

Response res1 =

*given*()

.auth().oauth2(access\_token)

.when()

.post("http://coop.apps.symfonycasts.com/api/4624/chickens-feed");

System.***out***.println("Status code " + res1.getStatusCode());

Assert.*assertEquals*(res1.getStatusCode(), 200);

System.***out***.println(res1.getBody().asString());

}

}

* 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());

}

}

**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);

}

**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;