What are assertions?

Assertion statmts are used to validate the response.It can be done in two ways:

Method1: either use testng or junit assertions

Steps:

* Static imports

**import** org.testng.Assert; 🡪for testng framework

**import** static org.junit.Assert.\*; 🡪for juintframework

* **Assertion stamt**

@Test

**public** **void** testngassertion() {

System.***out***.println("to print values of a paticular attribute");

String status=*given*().queryParam("api-key", "tdW5dbL1gMyF14kzX0dlLMeiekGphMdl").queryParam("format", "json").when()

.get("/lists/names").then().extract().path("status");

Assert.*assertEquals*(status, "OK");

System.***out***.println("Result: to print values of a paticular attribute"+ status);

}

Method 2: using HamcrestMatchers

Hamcrest is a framework that assists writing software tests in the Java programming language. It supports creating customized assertion matchers ('Hamcrest' is an anagram of 'matchers'), allowing match rules to be defined declaratively. These matchers have uses in unit testing frameworks such as JUnit and jMock and testng.

How to use?

* Static import

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

* **maven dependency:**

<!-- https://mvnrepository.com/artifact/org.hamcrest/hamcrest-all -->

<dependency>

<groupId>org.hamcrest</groupId>

<artifactId>hamcrest-all</artifactId>

<version>1.3</version>

<scope>test</scope>

</dependency>

* Assertion statmt:

E.g:1

@Test

**public** **void** restassuredassertion() {

System.***out***.println("to print values of a paticular attribute");

*given*().queryParam("api-key", "tdW5dbL1gMyF14kzX0dlLMeiekGphMdl").queryParam("format", "json").when()

.get("/lists/names").then().body("status", *equalTo*("ok"));

// .body("status",*equalToIgnoringCase*("ok"));

Here,we can directly assert inside .body() method.and no need of extracting the data.

*equalTo*("ok") method is available within hamcrest library.

E.g:2 Check singlevalue in arraylist:

@Test

**public** **void** extractarrayvalues() {

System.***out***.println("to check single values in array");

*given*().queryParam("api-key", "tdW5dbL1gMyF14kzX0dlLMeiekGphMdl").queryParam("format", "json").when()

.get("/lists/names").then().body("results.list\_name",*hasItem*(*equalToIgnoringCase*("Paperback NONfiction")));

//here, results🡪array name

.list\_name🡪attribute inside array

"results.list\_name"🡪collects the values of all the list\_name in the array and stores in a list,since it is a String,use “ ”.

hasItem🡪checks for the specified value from the list.

E.g:2 Check more than one value in arraylist:

@Test

**public** **void** extractmultiplearrayvalues() {

System.***out***.println("to check multiple values in array");

*given*().queryParam("api-key", "tdW5dbL1gMyF14kzX0dlLMeiekGphMdl").queryParam("format", "json").when()

.get("/lists/names").then().body("results.list\_name",*hasItems*("Paperback Nonfiction","Hardcover Fiction"));

}

Adding Multiple Assertions:

E.g:3

@Test

**public** **void** checkhashkeyvalues() {

System.***out***.println("to check hashkeyvalues in array");

*given*().queryParam("api-key", "tdW5dbL1gMyF14kzX0dlLMeiekGphMdl").queryParam("format", "json").when()

.get("/lists/names").then().body("results[1]",*hasKey*("list\_name") );

}

E.g:4

@Test

**public** **void** extracthahmapvalues() {

System.***out***.println("to check hashmap values");

*given*().queryParam("api-key", "tdW5dbL1gMyF14kzX0dlLMeiekGphMdl").queryParam("format", "json").when()

.get("/lists/names").then().body("results.findAll{it.list\_name=='Paperback Advice'}",*hasItems*(*hasValue*("2008-06-08")) );

}

E.g:5

To check key-value pair :

.body("results.findAll{it.newest\_published\_date=='2017-01-29'}",*hasItems*(*hasEntry*("list\_name\_encoded","young-adult-paperback"))

Multiple Assertions:

Instead of writing single assertion stmt in each test case,we can write multiple assertions in a single test case using multiple .body() stmts.

E.g:6

@Test

**public** **void** multipleassertions() {

System.***out***.println("to perform multiple assertions");

*given*().queryParam("api-key", "tdW5dbL1gMyF14kzX0dlLMeiekGphMdl").queryParam("format", "json").when()

.get("/lists/names").then().body("results.findAll{it.newest\_published\_date=='2017-01-29'}",*hasItems*(*hasEntry*("list\_name\_encoded","young-adult-paperback")))

.body("results.findAll{it.list\_name=='Paperback Advice'}",*hasItems*(*hasValue*("2008-06-08")) )

.body("results[1]",*hasKey*("list\_name") );

Logical Assertions:

@Test

**public** **void** logicalassertions() {

System.***out***.println("to perform logical assertions");

*given*().queryParam("api-key", "tdW5dbL1gMyF14kzX0dlLMeiekGphMdl").queryParam("format", "json").when()

.get("/lists/names").then()

.body("num\_results", *equalTo*(55))

.body("num\_results", *lessThan*(56)

.body("num\_results", *greaterThanOrEqualTo*(55));}

Hard Assertions:

Take the eg:here,assertion does not match in stmt 2 and stmt4

@Test

**public** **void** hardassertions() {

System.***out***.println("to check hard assertions");

*given*().queryParam("api-key", "tdW5dbL1gMyF14kzX0dlLMeiekGphMdl").queryParam("format", "json").when()

.get("/lists/names").then()

.body("results[0].list\_name", *equalTo*("Combined Print and E-Book Fiction"))

.body("results[0].display\_name", *equalTo*("Combined Print and E-Book Fiction"))

.body("results[0].newest\_published\_date", *equalTo*("2019-07-21"))

.body("results[0].updated", *equalTo*("WEEKLYs"));}

Here,assertion is stopped with second stmt without checking for 3rd and 4 th stmt.so,it does not list out assertion failure for 4th stmt.

To overcome this and to list out assertion failure for individual stmt,soft assert is used

o/p: JSON path results[0].display\_name doesn't match.

Expected: Combined Print and E-Book Fiction

Actual: Combined Print & E-Book Fiction

//displays only 2nd assertion failure

Soft Asserts:

It can be done by writing all assertion stmt in single body.in this case,it lists out all the assertion failure without stopping with the first failure.

@Test

**public** **void** softassertions() {

System.***out***.println("to check soft assertions");

*given*().queryParam("api-key", "tdW5dbL1gMyF14kzX0dlLMeiekGphMdl").queryParam("format", "json").when()

.get("/lists/names").then()

.body("results[0].list\_name", *equalTo*("Combined Print and E-Book Fiction"),

"results[0].display\_name", *equalTo*("Combined Print and E-Book Fiction"),

"results[0].newest\_published\_date", *equalTo*("2019-07-21"),

"results[0].updated", *equalTo*("WEEKLYs"));}

Here,all stmts are written in single .body()

o/p:

java.lang.AssertionError: 2 expectations failed.

JSON path results[0].display\_name doesn't match.

Expected: Combined Print and E-Book Fiction

Actual: Combined Print & E-Book Fiction

JSON path results[0].updated doesn't match.

Expected: WEEKLYs

Actual: WEEKLY

Assertion is not stopped with first failure

Asserting on complete json response

What is json assert library?

Used to write unit tests in less code in Rest interfaces.

Used to compare strings.

How does it work?

Test cases are written such that two strings are compared with each other.

Expected result is obtained from get call and returns json object.

Actual result is given in string format,and JSON assert converts them to json object and compared with expected one.

JSON assert :

1.add json assert dependency.

<dependency>

<groupId>org.skyscreamer</groupId>

<artifactId>jsonassert</artifactId>

<version>1.5.0</version>

<scope>test</scope>

</dependency>

2.copy json object completely in a text document🡪save it🡪copy +paste the txt file root of the project.

3.get the expected value from the file in string format.

String expectedvalue=**new** String(Files.*readAllBytes*(Paths.*get*(System.*getProperty*("user.dir")+File.***separator***+"jsonobject.txt")));

//reads the file in string format

//user.dir🡪to access root of the project

4.get the actual value as string from getcall()

String actualvalue=*given*()

.when().get("/list").asString();

5.Assertion:

It can be done using testng or junit assertion also.

Assert.*assertEquals*(expectedvalue, actualvalue);

o/p: junit.framework.ComparisonFailure: expected:<...":1,"firstName":"Ver[r]non","lastName":"Har...> but was:<...":1,"firstName":"Ver[]non","lastName":"Har...>

Result will not be clear telling which part of string is mismatched.to overcome this,we can use json assert library.

How to use json assertlibrary for assertion?

@Test

**public** **void** getstudentdetails() **throws** IOException, JSONException {

String expectedvalue = **new** String(

Files.*readAllBytes*(Paths.*get*(System.*getProperty*("user.dir") + File.***separator*** + "jsonobject.txt")));

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

String actualvalue = *given*().when().get("/list").asString();

System.***out***.println("actual value is" + actualvalue);

// Assert.assertEquals(expectedvalue, actualvalue);

JSONAssert.*assertEquals*(expectedvalue, actualvalue, JSONCompareMode.***LENIENT***);

}o/p: java.lang.AssertionError: [email=egestas.rhoncus.Proin@massaQuisqueporttitor.org].firstName

Expected: Verrnon

got: Vernon

It clearly shows the mismatch data.

6.JSONCompare Modes:

Linent and Strict mode

For.e.g.consider,we have changed the order of data in the input file.

Lineant mode:test will pass

Strict mode: test will fail because JSONAssert lib also checks for the order of data.

@Test

**public** **void** assertionwithstrictmode() **throws** IOException, JSONException {

System.***out***.println("to assert with different order in expected value");

String expextedvalue = **new** String(

Files.*readAllBytes*(Paths.*get*(System.*getProperty*("user.dir") + File.***separator*** + "difforder.txt")));

String actualvalue = *given*().when().get("/list").asString();

JSONAssert.*assertEquals*(expextedvalue, actualvalue, JSONCompareMode.***STRICT***);

}

o/p: FAILED: assertionwithstrictmode

java.lang.AssertionError: [0].email

Expected: faucibus.orci.luctus@Duisac.net

got: egestas.rhoncus.Proin@massaQuisqueporttitor.org

; [0].firstName

Expected: Murphy

got: Vernon

; [0].id

Expected: 2

got: 1

; [0].lastName

Expected: Holmes

got: Harper

; [1].email

Here,

i/p file order: id:2,id:1

actual order:id:1,id:2

so,JSONAssert lib checks order also and fails test

Suggestion: if we are just comparing/validating data,it is better to use lineant mode.if we use strict mode,test case will be fragile.

Root path in Rest asseured: