**Rest Assured: is a Java based DSL (Domain Specific Language) or library that can be used to test RESTful web services. REST Assured** library behaves like a headless Client to access **Rest** web services.

**HTTP REQUEST:** is a packet (binary data) of information sent from one computer to another computer communicating something.

**HTTP REQUEST** contains **Request header** and **Request body**

|  |  |  |
| --- | --- | --- |
| **Request header** | **Request Line** | **URL followed by parameters** |
| **Request method Token [ GET, POST,….]** |
| **Request body** | **additional information to server to process the request in a form of json or xml file.** | |

**HTTP RESPONSE:** isa packet of information sent by server to the client in response to the earlier Request made by client.

HTTP RESPONSE contain

|  |  |  |  |
| --- | --- | --- | --- |
| **Status Line** | **HTTP protocol version** | **Status Code** | **Reason Phrase** |
| **Response header** |  | **Status code** | |
| **Content-type [ xml or json], Content-Length** | |
|  |  | **Additional information** | |
| **Response body** | **Resource data that was requested [ Resource]** | | |

**REST Assured Test:**

* Create a new Maven project:
* add below **REST Assured dependency** in **pom.xml**

<!-- https://mvnrepository.com/artifact/io.rest-assured/rest-assured -->

<dependency>

<groupId>io.rest-assured</groupId>

<artifactId>rest-assured</artifactId>

<version>3.0.0</version>

<scope>test</scope>

</dependency>

* **RestAssured** is **class** to set up a request with **base URI**
* **IO.restassured.RestAssured** is a **class** to create **HTTP requests** against **base URI,** supports **different HTTP method** types **(GET,POST,PUT,PATH UPDATE,HEAD** and **OPTIONS).** Retrieves the **Response** from Server helps validate Responses.
* Every Request in Rest Assured library is represented by an **interface** **RequestSpecification**
* Io.restAssured.response.Response interface represents a Response returned from server. Different method can be called on Response object to get different parts of response.
* New java file, add below code

**public** **class** FirstRest{

@Test

**public** **void** GetWeatherDetails() {

// Specify the base URL to the RESTful web service

**RestAssured.baseURI = "http://restapi.demoqa.com/utilities/weather/city";**

// Get the RequestSpecification of the request that you want to be sent to the server.

// The server is specified by the BaseURI that we have specified in the above step.

**RequestSpecification httpRequest = RestAssured.given();**

// Make a request to the server by specifying the method Type and the method URL.

// This will return the Response from the server. Store the response in a variable.

**Response response = httpRequest.request(Method.GET, "/Hyderabad");**

// Now let us print the body of the message to see what response. we have received from the server

**String resBody = response.getBody().asString();**

**System.out.println("Response Body is => " + resBody);**

}

}

* **How to Validate Response Status using Rest Assured**

**Functional testing of Web Services** involves **verifying** the **Response returned** from various **End points.**

Some of the important **Test Verifications are**

1. **Response Status**
2. **Response Header**
3. **Response Body**

**1. Validate Response Status Code**

@Test

public void GetWeatherDetails() {

RestAssured.baseURI = "http://restapi.demoqa.com/utilities/weather/city";

RequestSpecification httpRequest = RestAssured.given();

Response response = httpRequest.get("/Hyderabad");

// Get the status code from the Response. In case of a successful interaction with the web service, we should get a status code of 200.

int statusCode = response.getStatusCode();

// Assert that correct status code is returned.

Assert.assertEquals(statusCode /\*actual value\*/, 200 /\*expected value\*/, "Correct status code returned");

}

* **How to validate an Error Status Code**

@Test

public void GetWeatherDetailsInvalidCity()

{

RestAssured.baseURI = "http://restapi.demoqa.com/utilities/weather/city";

RequestSpecification httpRequest = RestAssured.given();

Response response = httpRequest.get("/78789798798");

int statusCode = response.getStatusCode();

Assert.assertEquals(statusCode /\*actual value\*/, 200 /\*expected value\*/, "Correct status code returned");

}

* **Verify the Status Code returned by providing invalid city name**

@Test

public void GetWeatherDetailsInvalidCity()

{

RestAssured.baseURI = "http://restapi.demoqa.com/utilities/weather/city";

RequestSpecification httpRequest = RestAssured.given();

Response response = httpRequest.get("/78789798798");

int statusCode = response.getStatusCode();

Assert.assertEquals(statusCode /\*actual value\*/, 200 /\*expected value\*/, "Correct status code returned");

}

* **How to validate Response Status Line**

@Test

public void GetWeatherDetailsInvalidCity()

{

RestAssured.baseURI = "http://restapi.demoqa.com/utilities/weather/city";

RequestSpecification httpRequest = RestAssured.given();

Response response = httpRequest.get("/78789798798");

int statusCode = response.getStatusCode();

Assert.assertEquals(statusCode /\*actual value\*/, 200 /\*expected value\*/, "Correct status code returned");

}

* **How to make a POST Request using Rest Assured**

**POST Method** is used to **send data** to **server**. The **data** that is **sent to server** in a **POST request** is sent in the **body of HTTP request**. The **type of body, xml, json** or some other format **if a POST request contains** **json data** then **Content-Type** **header** will have a value of application/json, for a **POST request containing xml** the **Content-Type header** value will be **application/xml**.

* **Step 1. Create a Request pointing to the Service Endpoint**

RestAssured.baseURI ="http://restapi.demoqa.com/customer";

RequestSpecification request = RestAssured.given();

* **Step 2:** In order to **create a json object** in the **code** we need to **add** simple **JSON library** from mvn repository [ below dependency for pom.xml].

<!-- https://mvnrepository.com/artifact/com.googlecode.json-simple/json-simple -->

<dependency>

<groupId>com.googlecode.json-simple</groupId>

<artifactId>json-simple</artifactId>

<version>1.1.1</version>

</dependency>

* **To create a json request which contains all the fields**

**JSONObject** is a class that is present in **org.json.simple**package. This class is a programmatic representation of a ***JSON*** string.  Take a look at the **Request JSON** above for our test web service, you will notice that there are multiple nodes in the **Json**. Each node can be added using the ***JSONObject.put(String, String)***method. Once you have added all the nodes you can get the String representation of ***JSONObject*** by calling***JSONObject.toJSONString()*** method.

// JSONObject is a class that represents a Simple JSON.

// We can add Key - Value pairs using the put method

JSONObject requestParams = new JSONObject();

requestParams.put("FirstName", "Virender");

requestParams.put("LastName", "Singh");

requestParams.put("UserName", "simpleuser001");

requestParams.put("Password", "password1");

requestParams.put("Email",  "someuser@gmail.com");

* **Step 3: To add json body in the request and send the Request**

In this step we will simply add the **JSON** String to the body of the **HTTP Request** and make sure that the ***Content-Type*** header field has a value of ***application/json***.

you can put the **JSON** string in the body using the method called ***RequestSpecification.body(JsonString).***This method lets you updated the content of ***HTTP Request***Body. However, if you call this method multiple times the body will be updated to the latest **Json** String.

// Add a header stating the Request body is a JSON

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

// Add the Json to the body of the request

request.body(requestParams.toJSONString());

// Post the request and check the response

Response response = request.post("/register");

* **Step 4; Validate the Response**

Once we get the response back, all we have to do is validate parts of the response.

int statusCode = response.getStatusCode();

Assert.assertEquals(statusCode, "201");

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

Assert.assertEquals("Correct Success code was returned", successCode, "OPERATION\_SUCCESS");

* **The complete code for above:**

|  |  |
| --- | --- |
|  | @Test  public void RegistrationSuccessful()  {  RestAssured.baseURI ="http://restapi.demoqa.com/customer";  RequestSpecification request = RestAssured.given();  JSONObject requestParams = new JSONObject();  requestParams.put("FirstName", "Virender"); // Cast  requestParams.put("LastName", "Singh");  requestParams.put("UserName", "sdimpleuser2dd2011");  requestParams.put("Password", "password1");  requestParams.put("Email", "sample2ee26d9@gmail.com");  request.body(requestParams.toJSONString());  Response response = request.get("/register");  int statusCode = response.getStatusCode();  System.out.println("The status code recieved: " + statusCode);  System.out.println("Response body: " + response.body().asString());  } |