**API:**

**Application Programming Interface.** It is in JASON, XML etc..

**REST:**

**Representaional State Transfer**

It make a call from a client to server to get data back over the HTTP Protocol.

We get API s from Url.

**REST ASSURED:**

* Rest Assured is Open source Java Domain Specific Language (DSL)
* It is built on top of HTTP Client, Internally it use classes, methods, interfaces.
* It supports both XML and JSON format.

**REST- Assured Supports Other Requests:**

* POST
* GET : It will give you data
* PUT
* DELETE
* OPTIONS
* PATCH
* HEAD

**Pre-Conditions For REST-Assured:**

* Java
* Eclipse
* Maven
* Testng

**Dependencies:**

We have to add latest version of REST Assured dependency in pom.xml file

<dependency>

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

<artifactId>rest-assured</artifactId>

<version>3.0.6</version>

</dependency>

Then we have to get API.

**Sample Code:**

**import** org.testng.Assert;

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

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

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

**public** **class** Getdata {

@Test

**public** **void** TestResponseCode() {

Response rep=*get*("http://samples.openweathermap.org/data/2.5/weather?q=London,uk&appid=b6907d289e10d714a6e88b30761fae22");

**int** code = rep.getStatusCode();

System.***out***.println("Status of Code " +code);

Assert.*assertEquals*(code, 200);

}

@Test

**public** **void** Test() {

Response resp=*get*("http://samples.openweathermap.org/data/2.5/weather?q=London,uk&appid=b6907d289e10d714a6e88b30761fae22");

String data = resp.asString();

System.***out***.println("Data is" +data);

System.***out***.println("Responde time " +resp.getTime());

}

}

**In REST Assuresed we have different styles of writing test, Here we use BDD.**

**Intialization after “given”**

In BDD if you want set cookies, add authorization, parameters, or setting headers all we do after given sections.

In when, we get resource, or we consume resource.

**All actions after “When”**

Get,Post,put,delete,etc..

**Response after “Then”**

Validate status code, extract response, extract headers, cookies, extract response body.

**Methods (Requests): (In Eclipse StudentApp (RestAssured)**

**1) GET:**

**This Method Simply retrieves the data from server. No changes to server or resource.**

**EX:** Checking news, Searching on web ..

This action method we use for get data from file, its always after “When”.

Ex: .get(“/list”);

**Code:**

@Test

**public** **void** getstudentfrommed()

{

Response response3= *given*()

.param("programme", "Medicine")

.param("limit",1)

.when()

.get("/list");

System.***out***.println(response3.prettyPrint());

**2) POST:**

POST: This can create load on server if made multiple calls.

**This Method perform the changes to server, In simple words post always creates resources in server.**

**Ex: uploading pic, submitting application online .. etc..**

**It is NOT idempotent, so if you retrey the reqst N times, you will end up having N resources with N different URLs created on server.**

**Terminology:**

**“When U hit URI”**

**“When you send this Payload”**

**“When You make post reqst”**

**201 : Reqst submitted, It successes, It has created additional resource.**

**We can do this by differentways:**

* **By JSON object**
* **By Hash Map**
* **By File**
* **BY adding data into Body**

POST verb of HTTP protocol

***HTTP POST*** method is used to send data to the Server. Common places where you can find a ***POST***request is when you submit ***Form Data*** (HTML forms) on a web page. For e.g. submission of registration form of gmail. *POST*request usually result in changes on the Server like addition of new data or may be update to existing data.

The data that is sent to the server in a *POST* request is sent in the body of HTTP request. The type of body, XML, Json or some other format is defined by the ***Content-Type*** header. If a *POST* request contains Json data then the ***Content-Type*** header will have a value of ***application/json.***Similarly, for a POST request containing ***XML***the ***Content-Type*** header value will be ***application/xml***.

Let us understand testing a POST web service using a live example. We will take a look at a registration web service. Web service details are

|  |  |
| --- | --- |
| ***Endpoint*** | *http://restapi.demoqa.com/customer/register* |
| ***HTTP method type****:* | *POST* |
| ***Body***: | {     “FirstName” : “value”     “LastName” : “value”,     “UserName” : “value”,     “Password” : “value”,     “Email”        : “Value”   } |
| ***Success Response****:* | { “SuccessCode”: “OPERATION\_SUCCESS”, “Message”: “Operation completed successfully” } |
| ***Failure Response****:* | { “FaultId”: “User already exists”, “fault”: “FAULT\_USER\_ALREADY\_EXISTS” } |

**Don’t Forgot to add “Jackson dataBined” dependency.**

**In Code:**

**Create class for post**

**Write body for Jason/http**

**Create new package in src/main/java**

**Create new class(Pojio) in above package**

**Select Jason body**

**Right click source**

**Generate set setters getters**

**Select all fields, and select insertion point(after this seeters and getters will come)**

Get, Delete, Put will not create additional load on server while making multiple calls.

POST: This can create load on server if made multiple calls.

**3) DELETE:**

**This method simply deletes data or resources from the server.**

**Ex: Deleting files from cloud, delete pics from fb etc..**

**4) PUT:**

**This Method update the existing resources. In simple words this method can update the existing file on server.**

**If we do same operation 2 times it will over ride. It won’t create additional load to server.**