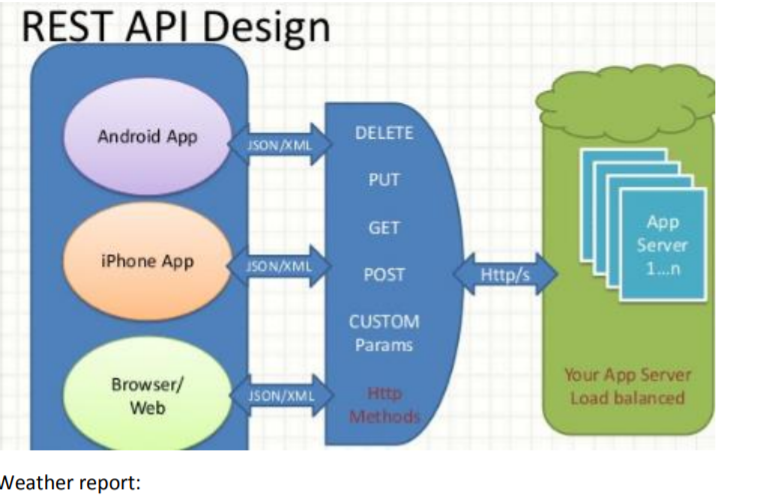
**Getting Started with REST API Testing**

A REST API defines a set of operations where developers can perform requests and receive responses via HTTP protocol



REST stands for Representational state transfer

• Because REST API’s use HTTP, they can be used by practically any programming language

• It acts as medium to propagate communication between the client and server applications on the World Wide Web

• Stateless – No client data is stored on the server between requests and session state is stored on the client.

• http is the transport protocol for REST

Examples : The Twitter REST API

Facebook REST API (deprecating)

Google Translate REST API

Flickr REST API

Dropbox REST API

**The key principles of REST are as follows:**

• Represent everything with a unique ID; a URI

• Stateless communication

• Make use of standard HTTP methods such as GET, POST, DELETE, and PUT

**GET-**

* The GET method is used to extract information from the given server using a given URI.
* While using GET request, it should only extract data and should have no other effect on the data.
* No Payload/Body required

**POST-**

* A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.
* To replace existing data or creating new data.
* Parameters should be sent in JSON/XML format
* Payload/Request body required

**PUT-** Replaces all current representations of the target resource with the uploaded content.

**DELETE-** Removes all current representations of the target resource given by a URI.

Resources can have multiple representations

**End point:** The term **endpoint** is focused on the **URL** that is used to make a request.

**End point can contains**

* Base URL
* Resources
* Parameters

Example: BaseURL/Resources?Parameters

Header and Cookies:

Google Place search api:

[https://maps.googleapis.com/maps/api/place/nearbysearch/*output*?*parameters*](https://maps.googleapis.com/maps/api/place/nearbysearch/output?parameters)

https://maps.googleapis.com/maps/api/place/nearbysearch/json?location=-33.8670522,151.1957362&radius=1500&type=restaurant&keyword=cruise&key=***YOUR\_API\_KEY***

BaseURL: https://maps.googleapis.com

Resources: maps/api/place/nearbysearch

Parameters: *parameters*

where output may be either of the following values:

* json (recommended) indicates output in JavaScript Object Notation (JSON)
* xml indicates output as XML

**JSON Path Expression Tester:** [**https://jsonpath.curiousconcept.com/**](https://jsonpath.curiousconcept.com/)

**Get Message Example:**

package basicsOfRestAPI;

import io.restassured.RestAssured;

import io.restassured.http.ContentType;

import static io.restassured.RestAssured.given;

import static org.hamcrest.Matchers.equalTo;

import org.testng.annotations.Test;

public class GetMessage {

@Test

public void getPlaceAPI()

{

// TODO Auto-generated method stub

//BaseURL or Host

RestAssured.baseURI="https://maps.googleapis.com";

given().

param("location","-33.8670522,151.1957362").

param("radius","1500").

param("key","AIzaSyCWg2j-SAAUKIy8rTVK\_dPlw7z14kRL7Io").

when().

get("/maps/api/place/nearbysearch/json").

then().assertThat().statusCode(200).and().contentType(ContentType.JSON).and().

body("results[0].name",equalTo("Sydney")).and().

body("results[0].place\_id", equalTo("ChIJP3Sa8ziYEmsRUKgyFmh9AQM")).and().

header("Server","scaffolding on HTTPServer2");

/\*header("dfd","fsdfds").

cookie("dsfs","csder").

body()\*/

//Status code of the response

//Content type

//Body

//Header responses

}

}

**POST Request**

In Post Request we need to send the data to the server in the form of Body

Parameters Types:

* Path Parameter
* Query Parameter
* Header Parameter

**Difference between Path Parameter and Query Parameter:**

**@QueryParam** is used to access key/value pairs in the query string of the URL (the part after the ?). For example in the url http://example.com?q=searchterm, you can use @QueryParam("q") to get the value of q.

**@PathParam** is used to match a part of the URL as a parameter. For example in an url of the form http://example.com/books/{bookid}, you can use @PathParam("bookid") to get the id of a book.

**package** basicsOfRestAPI;

**import** org.testng.annotations.BeforeTest;

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

**import** files.Payload;

**import** files.Resources;

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

**import** io.restassured.RestAssured;

**import** io.restassured.http.ContentType;

**import** io.restassured.path.json.JsonPath;

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

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

**import** java.io.FileInputStream;

**import** java.io.IOException;

**import** java.util.Properties;

**public** **class** PostMessage {

Properties prop=**new** Properties();

@BeforeTest

**public** **void** getData() **throws** IOException

{

FileInputStream fis=**new** FileInputStream("C:\\Users\\1024812\\Desktop\\Selenium\\RestAPIJavaProject\\src\\files\\env.properties");

prop.load(fis);

//prop.get("HOST");

}

@Test

**public** **void** PostMessage() {

//Task 1- Grab the response

RestAssured.*baseURI*= prop.getProperty("HOST");

Response res=*given*().

queryParam("key",prop.getProperty("KEY")).

body(Payload.*getPOSTData*()).

when().

post(Resources.*placePOSTData*()).

then().assertThat().statusCode(200).and().contentType(ContentType.***JSON***).and().

body("status",*equalTo*("OK")).

extract().response();

// Task 2- Grab the Place ID from response

String responseString=res.asString();

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

JsonPath js= **new** JsonPath(responseString);

String placeid=js.get("place\_id");

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

//Task 3 place this place id in the Delete request

*given*().

queryParam("key","AIzaSyDIQgAh0B4p0SdyYkyW8tlG-y0yJMfss5Y").

body("{"+

"\"place\_id\": \""+placeid+"\""+

"}").

when().

post("/maps/api/place/delete/json").

then().assertThat().statusCode(200).and().contentType(ContentType.***JSON***).and().

body("status",*equalTo*("OK"));

}

}

XML Post Message:

package basicsOfRestAPI;

import io.restassured.RestAssured;

import io.restassured.http.ContentType;

import io.restassured.path.xml.XmlPath;

import io.restassured.response.Response;

import static io.restassured.RestAssured.given;

import static org.hamcrest.Matchers.equalTo;

import java.io.IOException;

import java.nio.file.Files;

import java.nio.file.Paths;

import org.testng.annotations.Test;;

public class XMLPostMessage {

//public static void main(String[] args) {

@Test

public void Test() throws IOException{

String postData=GenerateStringFromResource("xml path of the body");

RestAssured.baseURI="https://maps.googleapis.com";

Response res=given().

queryParam("key", "AIzaSyCWg2j-SAAUKIy8rTVK\_dPlw7z14kRL7Io").

body(postData).

when().

post("/maps/api/place/add/xml").

then().assertThat().statusCode(200).and().contentType(ContentType.XML).and().

extract().response();

//convert the raw data into string

String respon=res.asString();

System.out.println(respon);

//convert the response into xml format

XmlPath x=new XmlPath(respon);

//System.out.println(x.get("PlaceResponse.place\_id"));

}

//convert the xml into string

public static String GenerateStringFromResource(String path) throws IOException{

return new String(Files.readAllBytes(Paths.get(path)));

}

}

Reusable methods

public static XmlPath rawToXML(Response r)

{

String respon=r.asString();

XmlPath x=new XmlPath(respon);

return x;

}

public static JsonPath rawToJson(Response r)

{

String respon=r.asString();

JsonPath x=new JsonPath(respon);

return x;

}

Traverse JSON

**package** basicsOfRestAPI;

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

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

**import** io.restassured.RestAssured;

**import** io.restassured.http.ContentType;

**import** io.restassured.path.json.JsonPath;

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

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

**import** files.\*;

**public** **class** TraversingJSON {

@Test

**public** **void** extractingNamesAPI()

{

// **TODO** Auto-generated method stub

//BaseURL or Host

RestAssured.*baseURI*="https://maps.googleapis.com";

Response res=*given*().

param("location","-33.8670522,151.1957362").

param("radius","500").

param("key","AIzaSyDIQgAh0B4p0SdyYkyW8tlG-y0yJMfss5Y").log().all().

when().

get("/maps/api/place/nearbysearch/json").

then().assertThat().statusCode(200).and().contentType(ContentType.***JSON***).and().

body("results[0].name",*equalTo*("Sydney")).and().

body("results[0].place\_id", *equalTo*("ChIJP3Sa8ziYEmsRUKgyFmh9AQM")).and().

header("Server","pablo").log().body().

extract().response();

JsonPath js= ReusableMethod.*rawToJson*(res);

**int** count=js.get("results.size()");

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

**for**(**int** i=0;i<count;i++)

{

// System.out.println(js.get("results["+i+"].name"));

//System.out.println();

//System.out.println(js.get("results["+i+"].name"));

}

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

/\*header("dfd","fsdfds").

cookie("dsfs","csder").

body()\*/

//Status code of the response

//Content type

//Body

//Header responses

}

}

**Advance Payload creation strategies:**

Dynamically build JSON payload with external data inputs

Parameterize the API tests with multiple data sets

How to send static JSON files(Payload) directly into Post method of Rest Assured

Feed JSON Payload from using excel using Hashmap.

**Dynamically build JSON payload with external data inputs:**

In this, instead of passing the static payload , we are sending the data externally i.e. in the payload class file .

To convert the JSON into String use the below URL, Enter the json in the edit box and click escape.

JSON Formatter: <https://www.freeformatter.com/json-escape.html#ad-output>

package libraryAPIExampleTests;

import static org.hamcrest.Matchers.equalTo;

import org.testng.annotations.Test;

import files.Payload;

import files.Resources;

import files.ReusableMethod;

import io.restassured.RestAssured;

import io.restassured.http.ContentType;

import io.restassured.path.json.JsonPath;

import io.restassured.response.Response;

import static io.restassured.RestAssured.given;

public class DynamicPostRequest {

@Test

public void AddBook() {

RestAssured.baseURI="http://216.10.245.166";

Response res=given().

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

body(Payload.Addbook("qwerty","1234")).

when().

post("/Library/Addbook.php").

then().assertThat().statusCode(200).

extract().response();

JsonPath js=ReusableMethod.rawToJson(res);

String id=js.get("ID");

System.out.println(id);

}

}

**package** files;

**public** **class** Payload {

**public** **static** String getPOSTData()

{

String b="{"+

"\"location\":{"+

"\"lat\":-33.8669710,"+

"\"lng\":151.1958750"+

"},"+

"\"accuracy\":50,"+

"\"name\":\"Google Shoes!\","+

"\"phone\_number\":\"(02) 9374 4000\","+

"\"address\":\"48 Pirrama Road, Pyrmont, NSW 2009, Australia\","+

"\"types\":[\"shoe\_store\"],"+

"\"website\":\"http://www.google.com.au/\","+

"\"language\":\"en-AU\""+

"}:";

**return** b;

}

**public** **static** String Addbook(String ISBN, String AISLE) {

String book="{\r\n\r\n\"name\":\"Learn Appium Automation with Java\",\r\n\"isbn\":\""+ISBN+"\",\r\n\"aisle\":\""+AISLE+"\",\r\n\"author\":\"John foe\"\r\n}\r\n ";

**return** book;

}

}

**Parameterize the API tests with multiple data sets**

BY using TestNG dataProvider we can parameterize the tests

**package** libraryAPIExampleTests;

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

**import** org.testng.annotations.DataProvider;

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

**import** files.Payload;

**import** files.Resources;

**import** files.ReusableMethod;

**import** io.restassured.RestAssured;

**import** io.restassured.http.ContentType;

**import** io.restassured.path.json.JsonPath;

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

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

**public** **class** DynamicPostRequest {

**@Test(dataProvider="AddMultipleBooks")**

**public** **void** AddBook**(String isbn, String aisle)** {

RestAssured.*baseURI*="http://216.10.245.166";

Response res=*given*().

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

**body(Payload.*Addbook*(isbn,aisle)).**

when().

post("/Library/Addbook.php").

then().assertThat().statusCode(200).

extract().response();

JsonPath js=ReusableMethod.*rawToJson*(res);

String id=js.get("ID");

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

}

@DataProvider(name="AddMultipleBooks")

**public** Object[][] getBooksData() {

**return** **new** Object[][] { {"qazwsx","0997"} , {"qazedc","0879"} , {"qazrfv","0654"} };

}

}

**How to send static JSON files(Payload) directly into Post method of Rest Assured**

If we have static JSON data in our test case then we can use this method.

Save the JSON data in external file with .json extension

{

"name":"Learn Appium Automation with Java",

"isbn":"bcd",

"aisle":"227",

"author":"John foe"

}

**package** libraryAPIExampleTests;

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

**import** java.io.IOException;

**import** java.nio.file.Files;

**import** java.nio.file.Paths;

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

**import** files.Payload;

**import** files.ReusableMethod;

**import** io.restassured.RestAssured;

**import** io.restassured.path.json.JsonPath;

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

**public** **class** HandlingStaticJsonInputData {

@Test

**public** **void** AddBook() **throws** IOException {

RestAssured.*baseURI*="http://216.10.245.166";

Response res=*given*().

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

body(*GenerateStringFromResource*("C:\\Users\\1024812\\Desktop\\Rest API Software and Notes\\AddBookStaticPayload.json")).

when().

post("/Library/Addbook.php").

then().assertThat().statusCode(200).

extract().response();

JsonPath js=ReusableMethod.*rawToJson*(res);

String id=js.get("ID");

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

}

//convert the xml/json into string

**public** **static** String GenerateStringFromResource(String path) **throws** IOException{

**return** **new** String(Files.*readAllBytes*(Paths.*get*(path)));

}

}

Automating Jira Rest API:

Server: B0QE-2BG5-GKG0-V6IX

Username: narayana.10209

Password: Automation@009

<http://localhost:9090/secure/WelcomeToJIRA.jspa>.

The preferred authentication methods for the JIRA REST APIs are [OAuth](https://developer.atlassian.com/display/JIRADEV/JIRA+REST+API+Example+-+OAuth+authentication) and [HTTP basic authentication](https://developer.atlassian.com/display/JIRADEV/JIRA+REST+API+Example+-+Basic+Authentication) (when using SSL).

JIRA itself uses cookie-based authentication in the browser.

<https://developer.atlassian.com/server/jira/platform/cookie-based-authentication/>

Cookie Bases Authentication: using this we can authenticate ourself to create any issues in jira.

This shows you how to allow REST clients to authenticate themselves using [cookies](https://en.wikipedia.org/wiki/HTTP_cookie). This is one of three methods that you can use for authentication against the Jira REST API; the other two are[Basic authentication](https://developer.atlassian.com/server/jira/platform/basic-authentication) and [OAuth](https://developer.atlassian.com/server/jira/platform/oauth).

1)Create a POST Request using the below details manually in POSTMAN tool.

URI : <http://localhost:9090/rest/auth/1/session>

Here <http://localhost:9090> is the server url and rest/auth/1/session is the resource.

Since it is a post request we need to send the body

{ "username": "narayana.10209", "password": "Automation@009" }

Pass the below header

Key: Content-type

Value: application/json

2)Click on Send

3)output will be in the below form (session value will change)

{

"session": {

"name": "JSESSIONID",

"value": "D140C63D6D323B93F931FA9118A19517"

},

"loginInfo": {

"loginCount": 3,

"previousLoginTime": "2019-02-12T15:29:49.086+0530"

}

}

Create an issue in Jira

Post request: <http://localhost:9090/rest/api/2/issue>

Here below highlighted key is your Jira Project Key

Body:

{

"fields": {

"project": {

**"key": "RES"**

},

"summary": "Rest API Manual First Issue",

"description": "first defect creating using postman",

"issuetype":{

"name": "Bug"

}

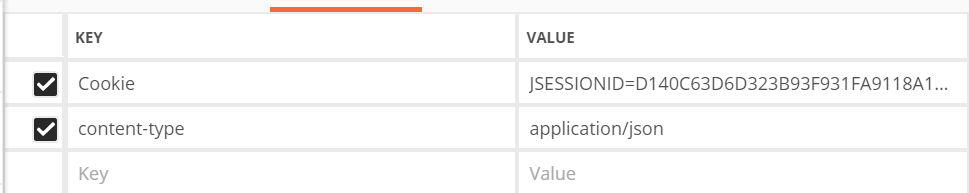
}

}

Headers:

Key: Cookie

Value: use the session value from the earlier request (JSESSIONID= D140C63D6D323B93F931FA9118A19517)



Click on Send

Response will be

{

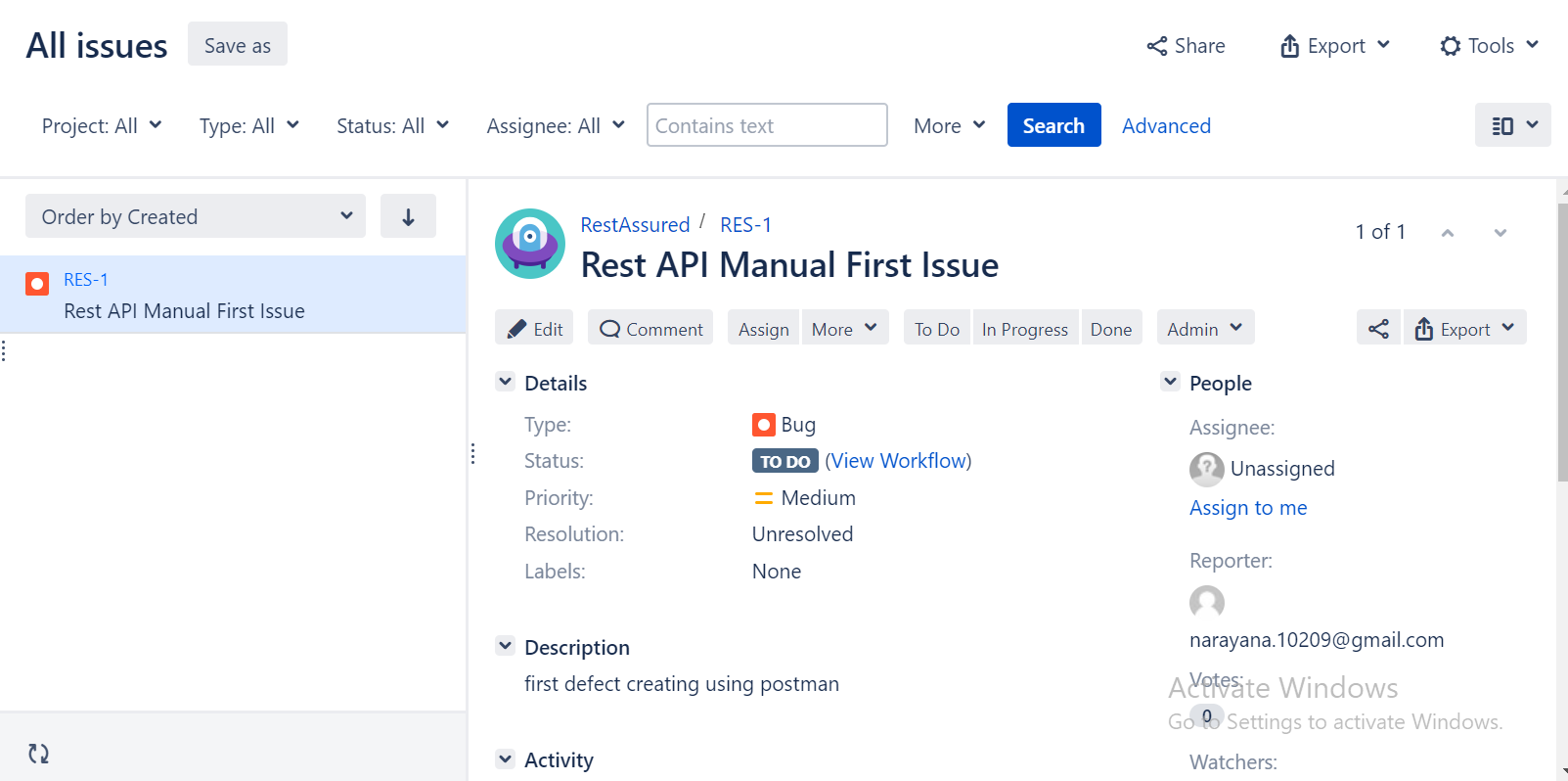
"id": "10000",

"key": "RES-1",

"self": "http://localhost:9090/rest/api/2/issue/10000"

}

Here id is the Jira issue id



Automating Jira API:

1)Authenticate yourself by creating session id

2)using the above session id create an issue with the mandatory fields.

1)Creating session id:

package jiraAPI;

import java.io.FileInputStream;

import java.io.IOException;

import java.util.Properties;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.Test;

import files.ReusableMethod;

import io.restassured.RestAssured;

import io.restassured.path.json.JsonPath;

import io.restassured.response.Response;

import static io.restassured.RestAssured.given;

public class CreateSessionIDUsingPOSTRequest {

Properties prop=new Properties();

@BeforeTest

public void getData() throws IOException

{

FileInputStream fis=new FileInputStream("C:\\Users\\1024812\\Desktop\\Selenium\\RestAPIJavaProject\\src\\files\\env.properties");

prop.load(fis);

//prop.get("HOST");

}

@Test

public void createSessionID() {

//Creating a session to authenticate

RestAssured.baseURI=prop.getProperty("JiraHost");

Response resp=given().

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

body("{ \"username\": \"narayana.10209\", \"password\": \"Automation@009\" }").

when().

post("/rest/auth/1/session").

then().statusCode(200).

extract().response();

JsonPath js=ReusableMethod.rawToJson(resp);

String sessionid=js.get("session.value");

System.out.println(sessionid);

//Create an issue

}

}

**Creating an issue in Jira(POST Request):**

**package** jiraAPI;

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

**import** files.ReusableMethod;

**import** io.restassured.RestAssured;

**import** io.restassured.path.json.JsonPath;

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

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

**public** **class** CreateIssueInJira {

@Test

**public** **void** createIssue() {

RestAssured.*baseURI*=”http://localhost:9090”;

Response res=*given*().

Header(“Content-Type”,”application/json”).

Header(“Cookie”,”JSESSIONID=”+ReusableMethod.*getJiraSessionID*()).

Body(“{\r\n” +

“ \r\n” +

“ \”fields\”: {\r\n” +

“ \”project\”: {\r\n” +

“ \r\n” +

“ \”key\”: \”RES\”\r\n” +

“ },\r\n” +

“ \r\n” +

“ \”summary\”: \”Rest API Manual First Issue\”,\r\n” +

“ \”description\”: \”first defect creating using postman\”,\r\n” +

“ \”issuetype\”:{\r\n” +

“ \”name\”: \”Bug\”\r\n” +

“ \r\n” +

“ }\r\n” +

“ \r\n” +

“}\r\n” +

“}”).

When().

Post(“rest/api/2/issue”).

Then().statusCode(201).

Extract().response();

JsonPath js=ReusableMethod.*rawToJson*(res);

String JiraID=js.get(“id”);

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

}

}

**Twitter API Testing Workflow**

First question we need to ask dev team

what type of Autentication its is?

Google map-Key

**Twitter uses**[OAuth](http://oauth.net/)**to provide authorized access to its API.**

<https://developer.twitter.com/en/docs/basics/authentication/overview/oauth>

**OAuth:** An **open protocol** to allow **secure authorization** in a **simple** and **standard** method from web, mobile and desktop applications.

OAuth is a simple way to publish and interact with protected data. It's also a safer and more secure way for people to give you access

**OAuth Parameters :**

Consumer Key

Consumer Secret

Token

Token Secret

**How to get Twitter OAuth Parameters?**

Create Twitter Account

Login to Twitter and Create Application from which you want to connect to Twitter –

<https://apps.twitter.com/>

**Explore Twitter API’s available**

<https://developer.twitter.com/en/docs/api-reference-index>

Twitter uses [OAuth 1.0a](https://oauth.net/core/1.0a/) and [OAuth2](https://oauth.net/2/) to provide authorized access to the API.

Twitter API Authentication Model

Application-only authentication: Oauth2 (bearer token)

Application-only authentication is a form of authentication where an application makes API requests on its own behalf, without the user context. This method is for developers that just need read-only to access public information.

Example: If a developer is trying to view Tweets or lists that are publically available, then they just have to use application-only authentication.

To use this method, you need to use a [**bearer token**](https://developer.twitter.com/en/docs/basics/authentication/guides/bearer-tokens). You can generate a bearer token by passing your consumer key and secret through the [POST oauth2 / token](https://developer.twitter.com/en/docs/basics/authentication/api-reference/token) endpoint.

API calls using app-only authentication are rate limited per API method at the app level.  Not all API methods support application-only authentication because some methods require a user context (for example, a Tweet can only be created by a logged-in user, so user context is required for that operation).  Not all API methods support application-only authentication because some methods require a user context (for example, a Tweet can only be created by a logged-in user, so user context is required for that operation).

Application-user authentication: Oauth 1a (access token for user context)

The user authentication method of authentication allows an authorized app to act on behalf of the user, as the user.

Example: if a developer wanted to build a feature that would allow a user to post Tweets through their platform using the statuses/update endpoint, the developer would have to use user authentication to get permission from the user to post a Tweet on their behalf.

In other words, a signed request identifies an application’s identity in addition to the identity accompanying granted permissions of the end-user the application is making API calls on behalf of, represented by the user’s [**access token**](https://developer.twitter.com/en/docs/basics/authentication/guides/access-tokens).

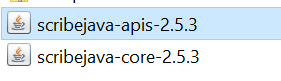
User authentication requires the consumer key and secret from your Twitter app and the access token and access token secret retrieved through the [3-legged-oauth](https://developer.twitter.com/en/docs/basics/authentication/overview/3-legged-oauth) process with the user that the developer is acting on the behalf of.  Not all API methods support application-user authentication.

How do you authenticate using oAuth in RestAssured API?

By using OAuth Method which accepts four parameters ConsumerKey, ConsumerSecret, Token,TokenSecret.

**Create Rest Request with OAuth 1.0 authentication:**

**Add the below jars before executing the tests**



[**https://mvnrepository.com/artifact/com.github.scribejava/scribejava-core/2.5.3**](https://mvnrepository.com/artifact/com.github.scribejava/scribejava-core/2.5.3)

[**https://mvnrepository.com/artifact/com.github.scribejava/scribejava-apis/2.5.3**](https://mvnrepository.com/artifact/com.github.scribejava/scribejava-apis/2.5.3)

Create and Delete Tweet in Twitter:

package twitterAPI;

import static io.restassured.RestAssured.given;

import org.testng.annotations.Test;

import io.restassured.RestAssured;

import io.restassured.path.json.JsonPath;

import io.restassured.response.Response;

public class CreateAndDeleteTweet {

String ConsumerKey="1HxV3RV10qvboQ4hkGcdYoCkK";

String ConsumerSecret="UvxP0uQS930tncArpncKw4YNSMIKDlrnxMF5GZvu36iVzHABhQ";

String Token="421068780-kZgXQr3GImZoEUoUKZiKKXJajKXHxcKhZO77wlAP";

String TokenSecret="xrSeD4EhBz5ubmLWKyzVb7t12CkpRQm3wVtKywkMFFcxz";

String id;

@Test

public void createTweet()

{

RestAssured.baseURI="https://api.twitter.com/1.1/statuses";

Response res=given().

**auth().oauth(ConsumerKey, ConsumerSecret, Token, TokenSecret).**

queryParam("status", "I am tweeting from Rest API Automation using Java").

when().

post("/update.json").

then().extract().response();

String response=res.asString();

System.out.println(response);

JsonPath js=new JsonPath(response);

id=js.get("id").toString();

System.out.println("Created Tweet ID is " + id);

// JsonPath js=ReusableMethod.rawToJson(res);

String latesttweet=js.get("text").toString();

System.out.println("Created Tweet Message is " +latesttweet);

}

@Test

public void E2E() {

createTweet();

RestAssured.baseURI="https://api.twitter.com/1.1/statuses";

Response res= given().auth().oauth(ConsumerKey, ConsumerSecret, Token, TokenSecret)

.when().post("/destroy/"+id+".json").then().extract().response();

String response =res.asString();

System.out.println(response);

JsonPath js= new JsonPath(response);

//System.out.println(js.get("text"));

System.out.println("Tweet which got deleted with automation is below");

System.out.println(js.get("text").toString());

System.out.println(js.get("truncated").toString());

}

}

**Maven Project**

Rest Assured Dependencies in POM:

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>6.1.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.11</version>

<scope>test</scope>

</dependency>

**<dependency>**

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

**<artifactId>rest-assured</artifactId>**

**<version>3.0.2</version>**

**<scope>test</scope>**

**</dependency>**

**<dependency>**

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

**<artifactId>json-path</artifactId>**

**<version>3.0.2</version>**

**</dependency>**

**<dependency>**

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

**<artifactId>xml-path</artifactId>**

**<version>3.0.2</version>**

**</dependency>**

**<dependency>**

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

**<artifactId>json-schema-validator</artifactId>**

**<version>3.0.2</version>**

**<scope>test</scope>**

**</dependency>**

**JSON PATH EXAMPLES**

**package basicsOfGoogleRestAPI;**

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

**import files.ReusableMethod;**

**import io.restassured.RestAssured;**

**import io.restassured.http.ContentType;**

**import io.restassured.path.json.JsonPath;**

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

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

**public class GetRequestGoogleDirectionsAPI {**

**@Test**

**public void googleDirections() {**

**//https://maps.googleapis.com**

**///maps/api/directions/json?origin=Disneyland&destination=Universal+Studios+Hollywood&key=YOUR\_API\_KEY**

**RestAssured.baseURI="https://maps.googleapis.com";**

**Response res=given().**

**queryParam("origin", "Disneylant").**

**queryParam("destination", "Universal+Studios+Hollywood").**

**queryParam("key","AIzaSyCWg2j-SAAUKIy8rTVK\_dPlw7z14kRL7Io").**

**when().**

**get("/maps/api/directions/json").**

**then().assertThat().statusCode(200).and().contentType(ContentType.JSON).**

**extract().response();**

**String response=res.asString();**

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

**//JSON Path Expression Tester: https://jsonpath.curiousconcept.com/**

**JsonPath js=ReusableMethod.rawToJson(res);**

**int count=js.get("geocoded\_waypoints.size()");**

**System.out.println("size of the geocoded\_waypoints is " + count);**

**for(int i=0;i<count;i++)**

**{**

**System.out.println(js.get("geocoded\_waypoints["+i+"]"));**

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

**//System.out.println(js.get("results["+i+"].name"));**

**}**

**String status=js.get("geocoded\_waypoints[0].geocoder\_status");**

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

**String Placeid=js.get("geocoded\_waypoints[0].place\_id");**

**System.out.println("Place id is " + Placeid);**

**String type=js.get("geocoded\_waypoints[0].types[0]");**

**System.out.println("Type 1 is " + type);**

**String northeastlongitude=js.get("routes[0].bounds.northeast.lat").toString();**

**System.out.println("North East Longitude is " + northeastlongitude);**

**String copyright=js.get("routes[0].copyrights");**

**System.out.println("Copy Right is " + copyright);**

**String distancetext=js.get("routes[0].legs[0].distance.text");**

**System.out.println("Distance Text is " + distancetext);**

**String end\_address=js.get("routes[0].legs[0].end\_address");**

**System.out.println("End Address is " + end\_address);**

**String PolylinePoint=js.get("routes[0].legs[0].steps[2].polyline.points");**

**System.out.println("Polyline point is " + PolylinePoint);**

**String travel\_mode=js.get("routes[0].legs[0].steps[2].travel\_mode");**

**System.out.println("Travel Mode is " + travel\_mode);**

**}**

**}**