**Rest API Manual/Automation Testing**

* **Rest Assured Libraries are used to do the Rest API Automation Testing**
* **Postman tool is used for Manual API Testing**

**Communication between the Client and Server**

Back End (Server)

Programming Language: Java

Front End (Client)

Programming: Java

**Note**: Communication between the Client and Server when both Front End and Back End are developed with same language.

**There are two possible issues commonly industry faces**

1. One application is communicating with another application which are completely in different environments and having different repositories

**Example:** Websites like makemytrip.com, yatra.com need to communicate with different applications (Spicejet, indiGo etc.,) to perform the actions like for booking a flight ticket etc.,

**Https Request**

Back End (Server)

Spicejet.com

Programming: Java

Front End (Client)

makemytrip.com

Programming: .Net (C#) / Java

**Https Response**

Direct Communication is not possible between the Client and Server as they are hosted in the different environments with may be different programing Languages

1. Within the same application, there might be a chance of using the different languages (Angular Js-Type script) in Front End and different languages (Java) in the Back End. There might be direct communication is not possible between these (Front End and Back End) because of the Design Patterns.

Back End (Server)

Programming Language: Java

Front End (Client)

Programming: Angular JS – Type Script

Direct Communication is not possible between the Client and Server as they are developed with different programing Languages

**APIs can resolve the both above problems.**

* APIs are not language dependent, any language can use it.
* Front End (client application) will send a request to the API through http protocol. And this http protocol is independent of any language
* Information will be sent to the APIs in the JSON (or) XML formats.
  + JSON and XML doesn't belong to any language family. Every language can use these.
* APIs will serve the communication between two languages like below

Rest API

**Https Request Https Request**

Back End (Server)

SpiceJet.com

Programming: Java

Front End (Client)

Makemytrip.com/Spicejet.com

Programming: Angular JS – Type Script / .Net (C#) / Java

**Https Response Https Response**

**API Definition:**

API is an Interface between a client and server intended to simply the building of client-side software

* There are two different architectural designs available in the market to build APIs
  + SOAP APIs --> call as services
  + REST APIs --> actual APIs.
  + In general, everyone will say services (SOAP APIs) are same as REST APIs.
* REST APIs follows the light weighted protocol compared to the SOAP APIs.

**Differences between SOAP & REST:**

|  |  |  |
| --- | --- | --- |
| #S.No | SOAP | REST |
| 1 | Simple Object Access Protocol | Representational State Transfer |
| 2 | It is pretty much Old technology | It is latest technology |
| 3 | SOAP supports only XML format for Request and Response | REST supports Multiple formats like XML, JSON, Text, html etc., |
| 4 | It supports only Post request | It supports different types of requests like Post, Put, Get, Delete, Patch etc., |
| 5 | SOAP is a protocol | REST is Architectural Pattern  It is light weight protocol |
| 6 | SOAP cannot make use of REST since SOAP is a Protocol and REST is Architectural Pattern | REST can make use of SOAP as the underlying protocol for webservices, because REST is just architectural pattern |

**End Point / Base URI:**

It is an Address/ location where API is hosted on the server.

Http methods which are commonly used to communicate with REST APIs are:

* GET
* POST
* PUT
* DELETE

These are usually also called as CRUD operations on API. Here C is Create (POST), R is Retrieve (GET), U is Update (PUT) and D is Delete (DELETE)

**GET:**

* Get method/request is used to extract / retrieves the information from the given server using a given URI.
* While using GET request, it should only extract the data and there should not have any other impact on the data.
* No Payload/body required.
* Using **Query Parameters** can send input data in GET

**POST:**

* POST method/request is used to send the information to the server. For example customer information, file upload etc., using HTML forms
* Using **Form Parameters / Body Payload** can send input data in POST

**PUT:**

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

(OR)

* Updates all the existing information to the server using PUT method/request.

**DELETE:**

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

**RESOURCES:**

* Resources represent APIs/Collection which can be accessed from the server.

Example:

* Google.com/maps
* Google.com/search
* Google.com/Images

**PARAMETERS:**

* There are two ways to pass the parameters while sending the information to the server.
  + Path Parameters
  + Query Parameters

Path Parameters

* + Path parameters will be given in Base URI/End Point
  + Path Parameters are used to point to a specific collection of the resources.

Example:

<https://www.amazon.co.in/Orders/1127564>

Here Orders is the actual Resources/Collection and 1127564 is used as the path parameter to retrieve the specific order information

Query Parameters

* + Query parameters are used to Sort/Filter the resources
  + Query parameters are identified with **“?”**

Example:

<https://amazon.co.in/orders?sort_by=2/20/2020>

Here Orders resource will be filtered by the query parameter to get specific set of data.

Each and every query parameter will be separated with "&"

Example:

<https://www.google.com/search?client=opera&hs=vfy&ei=xn5SX_mxO-u-3LUP2KWs4A0&q=india&oq=india&gs_lcp=CgZwc3ktYWIQAzIFCAAQkQIyBAgAEEMyBwguELEDEEMyBwgAELEDEEMyBAgAEEMyBAgAEEMyBAgAEEMyBAgAEEMyCggAELEDEIMBEEMyCggAELEDEIMBEEM6BAgAEEc6CAgAELEDEIMBOgIIADoICC4QxwEQowI6BQguELEDUNHxoAtYhPegC2Cu-KALaABwBHgAgAFsiAH8A5IBAzEuNJgBAKABAaoBB2d3cy13aXrAAQE&sclient=psy-ab&ved=0ahUKEwj5rN63iNDrAhVrH7cAHdgSC9wQ4dUDCAw&uact=5>

**Note**: Path Parameters will be used with the specific sub resource, whereas Query Parameters will be used directly on resources/collection only.

* End Point request URL can be constructed as below
  + Base URI/resource/(Query/Path) Parameters

Combination of Domain, Path Parameter and Query Parameter is called as URL (Uniform Resource Identifier)

* 1. Difference between URL & URI is
     1. URL --> Uniform Resource Locator and It can have only domain name (or) it may have both domain name & path parameters (or) it may have domain name, path parameter & query parameter

<https://reqres>

<https://reqres.in/api>  
<https://reqres.in/api/users?page=2>

1. URI --> Uniform Resource Identifier --> It is a combination of Domain & Path Parameters (sometimes it may have Query Parameter also and it is optional)

**Headers/Cookies**

* Headers represent the meta-data associated with the API request and response
* In layman terms, we were sending additional parameters to process our request

Example: Authorization details

**Downloading POSTMAN**

* Official website: <https://www.postman.com>
* Download the Postman from the below URL by choosing appropriate OS and 32/64 bit
  + <https://www.postman.com/downloads/>

**New Collection Creation**

* In Postman, collection is nothing but a project. To create a New project, click on a New Collection link and provide the Collection Name and Description --> And click on Create --> New collection will be created in Postman

**To Add a New Request (GET/POST/PUT/DELETE)**

* Against to the collection name --> click on the three dot symbols (…) --> Click on Add Request --> Provide the **Request Name, Request Description** --> click on Save To <Collection Name> --> Request will be added in the collection.
* Click on Save button, then only changes made will be saved. After saving the request only appropriate tag name will be shown besides the Request Name in Collection
* To provide the Query Parameters, Click on Params tab and Enter the Key and Values. Which will reflect in the base URI
* Whenever any Payload given in the Request, provide the type of data which is provided. Click on Body 🡪 select **raw 🡪** click on Text and select the Json/XML appropriately. Under the Headers content-type should become **application/json** or **application/xml** respectively
* Once the Request is sent, if we get the Response code as **200 OK.** Then only we should confirm that Response is success. Otherwise there is some issue in the Response.

**Note:** In General **POST** is the superset, so POST APIs will be used instead of the DELETE/PUT requests, because same action can be performed by using the POST request.

**Library API Add Book (POST)**

This API will add a new Book into the server

**Complete URL:** http://216.10.245.166/Library/Addbook.php

**Base URL:** http://216.10.245.166

**Resource:** Library/Addbook.php

**Http Method:** POST

**Payload Body:**

{

"name":" Rest API Test Automation Learning ",

"isbn":"bcd",

"aisle":"227",

"author":"John foe"

}

**Response:**

{

"Msg": "successfully added",

"ID": "bcd227"

}

**Library API Get Book by Author Name (GET)**

This API will Retrieve all books information form the server based on Author Name

**Complete URL:** http://216.10.245.166/Library/GetBook.php?AuthorName=somename

**Base URL:** http://216.10.245.166

**Resource:** /Library/GetBook.php?AuthorName=somename

**Http Method:** GET

**Response:**

{

  "book\_name": "Rest API Test Automation Learning",

   "isbn": "bcd",

   "aisle": "227"

}

**Library API Get Book by ID (GET)**

This API will Retrieve a book information form the server based on ID

Note: Here ID value is combination of **isbn and aisle**

**Complete URL:** http://216.10.245.166/Library/GetBook.php?ID=someID

**Base URL:** http://216.10.245.166

**Resource:** Library/GetBook.php?ID=bcd227

**Http Method:** GET

**Response:**

{

  "book\_name": "Rest API Test Automation Learning",

   "isbn": "bcd",

   "aisle": "227"

}

**Library API Delete Book by ID (POST)**

This API will delete book information in the server based on ID

**Complete URL:** http://216.10.245.166/Library/DeleteBook.php

**Base URL:** http://216.10.245.166

**Resource:** :/Library/DeleteBook.php

**Http Method:** POST

**Payload body:**

{

"ID": "bcd227"

}

**Response:**

{

  "msg": "book is successfully deleted”

}

* Whenever Request Sent with all above Information, API will give a Response. In the API Response we generally Validate the
  1. Response Status Code
  2. Response Body
  3. Response Time
  4. Response Size
  5. Response Headers

**Http status codes:**

Status codes are divided into five categories

1. **1xx: Informational** - Communicates transfer protocol-level information
2. **2xx: Success -** Indicates that the client's request was accepted successfully
3. **3xx: Redirection -** Indicates that client must take some additional action in order to complete their request
4. **4xx: Client Error -** Indicates the error status codes at Clients
5. **5xx: Server Error -** Indicates the error status codes at servers

**Few Status Codes:**

1. **1xx: Informational**
   1. 100 - Continue
   2. 101 - Switching Protocol
   3. 102 - Processing
   4. 103 - Early Hints
2. **2xx: Success**
   1. 200 - OK - Indicates that request has succeeded
   2. 201 - Created - Indicates that request has succeeded and new resource has been created as a result
   3. 202 - Accepted - Indicates that request has been received but not completed yet
   4. 204 - No Content - The Server has fulfilled the request but does not return a response body
3. **3xx: Redirection**
   1. 300 - Multiple Choices - The request has more than one possible response. The user should choose one of them
   2. 301 - Moved Permanently - The URL of the requested resource has been changed permanently
   3. 302 - Found - The URL of the requested resource has been changed temporarily
4. **4xx: Client Error**
   1. 400 - Bad Request - The request could not understood by the server due to the incorrect syntax
   2. 401 - Unauthorized - Indicates that the request requires user authentication information
   3. 402 - Payment Required - Reserved for future use. It is aimed for using in the digital payment systems
   4. 403 - Forbidden - Unauthorized request, Client does not have access rights to the content
   5. 404 - Not Found - The Server cannot find the requested resource
5. **5xx: Server Error**
   1. 500 - Internal Server Error - The Sever encountered an unexpected condition which prevented it from fulfilling the request
   2. 501 - Not Implemented - The Http method not supported by the server and It cannot be handled
   3. 502 - Bad Gateway - The Sever got Invalid response while working as a Gateway to get a response needed to handle the request
   4. 503 - Service Unavailable - The Sever is not ready to handle the request
   5. 504 - Gateway Timeout - The Server is acting as a gateway and cannot get a response in time for a request

**Pre-Requisites before using Rest Assured API Automation**

* Java must be installed in the system.
* Any one of the IDE must be available (like Eclipse, Intellij idea, Netbeans etc.,)

**What is Rest Assured?**

Rest-Assured is a Java-based library that is used to test RESTful Web Services/API’s

(OR)

Rest Assured is a Java DSL (Domain-Specific Language) to test the Rest based APIs which were built in top of Http

* It supports POST, GET, PUT, DELETE, OPTIONS, PATCH, HEAD, … etc., requests and can be used to validate and verify the response of the requests.

**Downloading the Rest Assured API Jars**

Navigate to the <https://rest-assured.io/> 🡪 click on Downloads 🡪 Download the below packages

* 1. [rest-assured-4.3.1-dist.zip](http://dl.bintray.com/johanhaleby/generic/rest-assured-4.3.1-dist.zip)
  2. [json-path-4.3.1-dist.zip](http://dl.bintray.com/johanhaleby/generic/json-path-4.3.1-dist.zip)
  3. [xml-path-4.3.1-dist.zip](http://dl.bintray.com/johanhaleby/generic/xml-path-4.3.1-dist.zip)
  4. [json-schema-validator-4.3.1-dist.zip](http://dl.bintray.com/johanhaleby/generic/json-schema-validator-4.3.1-dist.zip)

Unzip these downloaded files and Rest Assured API jars will be available.

In Eclipse 🡪 Add the above downloaded Rest API Jars into the Eclipse IDE 🡪 Start the Rest API Automation

**How to add the Rest API Jars into the Eclipse IDE**

* Open the Project in Eclipse
* Right click on a project 🡪 select the Build Path and Configure Build Path
* Under the Libraries tab 🡪 Click on Add External Jars
* Browse the Jar files wherever located in the system (Add all the Jar files)
* Click Ok and Apply 🡪 all the Added Jar files will be reflected under the Referenced Libraries

Rest Assured API automation works on three principles

1. Given 🡪 To provide all input details
2. When 🡪 To submit the request (Usually Resource and Http method will fall under this category)
3. Then 🡪 To validate the response

**How to send static Json files (payload) directly into Post method of Rest API**

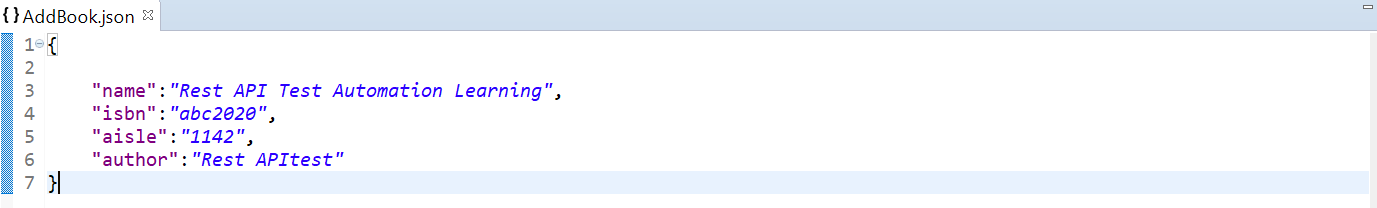
* Create a Json file inside the project (or) somewhere else
* First convert Json file data into byte streams using the below method

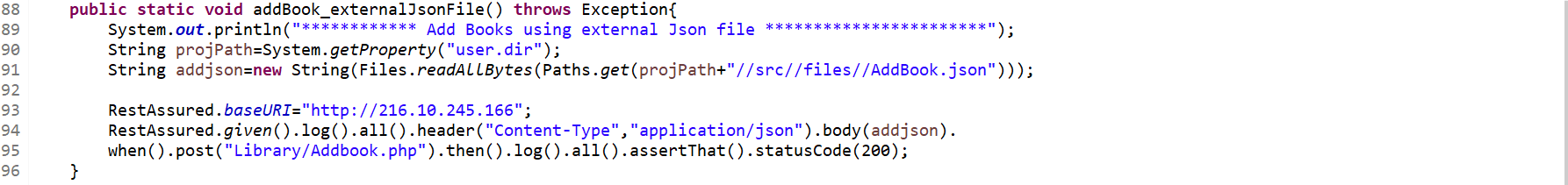
**Files.readAllBytes(Paths.get(<Json File Path>)))**

* And convert the byte stream into String Object as below

**String addjson=new String(Files.readAllBytes(Paths.get(<Json File Path>)));**

* Pass the above String Object inside body() method of Rest API





**Authentication Techniques**

**1. Basic Authentication**

**2. Token Based Authentication**

**3. OAuth2.0 Authentication**

**Basic Authentication:**

**---------------------**

**Basic authentication involves sending a base64-encoded username and password in the Authorization header.**

**public void testBasicAuthentication() {**

**RestAssured.baseURI = "https://api.example.com";**

**given()**

**.auth()**

**.basic("username", "password")**

**.when()**

**.get("/endpoint")**

**.then()**

**.statusCode(200);**

**}**

**Token Based Authentication:**

**---------------------------**

**Bearer token authentication involves including a token in the Authorization header.**

**public void testBearerTokenAuthentication() {**

**RestAssured.baseURI = "https://api.example.com";**

**given()**

**.auth()**

**.oauth2("your\_access\_token")**

**.when()**

**.get("/endpoint")**

**.then()**

**.statusCode(200);**

**}**

**OAuth2.0 Authentication:**

**------------------------**

**1. Obtain Authorization Code**

**The client (your application) redirects the user to the authorization server to obtain an authorization code.**

**2. Exchange Authorization Code for Access Token:**

**After the user grants permission, the client exchanges the obtained authorization code for an access token.**

**3. Use Access Token to Access Resources:**

**The client can use the access token to make authorized requests to the resource server (API).**

**1. Basic Authentication:**

**Basic authentication involves sending a base64-encoded username and password in the Authorization header.**

**public void testBasicAuthentication() {**

**RestAssured.baseURI = "https://api.example.com";**

**given()**

**.auth()**

**.basic("username", "password")**

**.when()**

**.get("/endpoint")**

**.then()**

**.statusCode(200);**

**}**

**2. Bearer Token Authentication:**

**Bearer token authentication involves including a token in the Authorization header.**

**public void testBearerTokenAuthentication() {**

**RestAssured.baseURI = "https://api.example.com";**

**given()**

**.auth()**

**.oauth2("your\_access\_token")**

**.when()**

**.get("/endpoint")**

**.then()**

**.statusCode(200);**

**}**

**3. OAuth 2.0 Authorization Code Grant Flow:**

**a) Obtain Authorization Code**

**The client (your application) redirects the user to the authorization server to obtain an authorization code.**

**b) Exchange Authorization Code for Access Token:**

**After the user grants permission, the client exchanges the obtained authorization code for an access token.**

**c) Use Access Token to Access Resources:**

**The client can use the access token to make authorized requests to the resource server (API).**

**import io.restassured.RestAssured;**

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

**import org.junit.jupiter.api.Test;**

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

**public class OAuth2Example {**

**// OAuth 2.0 endpoints**

**private static final String AUTH\_SERVER = "https://authorization-server.com";**

**private static final String TOKEN\_ENDPOINT = AUTH\_SERVER + "/token";**

**private static final String CLIENT\_ID = "your-client-id";**

**private static final String CLIENT\_SECRET = "your-client-secret";**

**private static final String REDIRECT\_URI = "https://your-redirect-uri.com";**

**private static final String SCOPE = "read write";**

**private static final String USERNAME = "your-username";**

**private static final String PASSWORD = "your-password";**

**private static final String API\_ENDPOINT = "https://api.example.com/data";**

**@Test**

**public void testOAuth2AuthorizationCodeGrant() {**

**// Step 1: Obtain Authorization Code**

**String authorizationCode = obtainAuthorizationCode();**

**// Step 2: Exchange Authorization Code for Access Token**

**String accessToken = exchangeAuthorizationCodeForAccessToken(authorizationCode);**

**// Step 3: Use Access Token to Access Resources**

**useAccessTokenToAccessResources(accessToken);**

**}**

**private String obtainAuthorizationCode() {**

**RestAssured.baseURI = AUTH\_SERVER;**

**// Redirect the user to the authorization server to obtain an authorization code**

**String authorizationCode =**

**given()**

**.redirects().follow(false)**

**.queryParam("response\_type", "code")**

**.queryParam("client\_id", CLIENT\_ID)**

**.queryParam("redirect\_uri", REDIRECT\_URI)**

**.queryParam("scope", SCOPE)**

**.get("/authorize")**

**.then()**

**.statusCode(302)**

**.extract()**

**.header("Location")**

**.split("code=")[1];**

**return authorizationCode;**

**}**

**private String exchangeAuthorizationCodeForAccessToken(String authorizationCode) {**

**// Exchange the obtained authorization code for an access token**

**Response response =**

**given()**

**.formParam("grant\_type", "authorization\_code")**

**.formParam("code", authorizationCode)**

**.formParam("client\_id", CLIENT\_ID)**

**.formParam("client\_secret", CLIENT\_SECRET)**

**.formParam("redirect\_uri", REDIRECT\_URI)**

**.when()**

**.post(TOKEN\_ENDPOINT);**

**String accessToken = response.jsonPath().getString("access\_token");**

**return accessToken;**

**}**

**private void useAccessTokenToAccessResources(String accessToken) {**

**// Use the obtained access token to access protected resources**

**RestAssured.baseURI = API\_ENDPOINT;**

**given()**

**.auth().oauth2(accessToken)**

**.when()**

**.get("/resource")**

**.then()**

**.statusCode(200);**

**}**