**Application Programming Interface: -**

**API stand for Application Programming Interface.**

**Multiplication**

**Subtraction**

ition

**Addition**

**System 1**

A = 10; b= 20;

A= 30; b= 40;

A= 2; b= 4

From **our class** we called these methods for our use. It is designed by someone else either person/ company.

**Our Class**

**System 1**

Sends Response  
 {XML, JSON, HTML}

Authentication Key/ Token required

Sends Request  
{XML, JSON, HTML}

**System 2**

**Calculator.jar** (Someone created this jar file for our use)

**API’s** are nothing but some services/functionalities created by another person/company for use. But, they didn’t share their code. If anyone want to use that service/functionality, then it can be use sending request to that API’s in proper format designed by the API developers.

**System 1** will interact with **System2. System 1** will send request in any format to access API’s. The format will be decided by **System 2** either it may **JSON or XML**. Proper authentication will have needed to access API’s of **System 2**. Once request send successfully then they will get response from **System 2.**

**API’s** are mostly used when two systems are interacting with each other. **API’s** hides the business logic from rest of the world.

**Selenium also worked on Rest API’s.**

**Web Services: -**

**MMT** will received the information from user   
and coverts it into **JSON/XML** format known as  
 **JSON Payload/XML Request.** It depends on  
in which format Other Systems (API’s) allowed   
to request/response. **These type of API’s are called  
Web Services.** Here, we are communicating over **HTTP** without  
JAR file. So, these are called as **Web Services**. Everyone have their  
own security and they are not bothered about other systems  
securities interacting with them.

Generates response and send back to MMT

Send details for payment to Axis Bank API’s

**MMT will display received data in either JSON/XML into proper UI understandable by User.**

**Emirates will send response back to MMT {  
“PRICE” : 50,000**

**“TIME”: “6 AM”**

**}  
RESPONSE JSON**

**Sends request either in XML/JSON**

**USER**

**User send details to MMT on their web site/app. For Ex: -**

**{  
“FROM” : “PUNE”**

**“TO”: “JOHANNESBURG**

**“FROM DATE”: “02-SEP-2019”**

**“RETURN DATE”: “12-SEP-2019”**

**“PASSENGERS”:01**

**}**

**MAKE MY TRIP**

1. **Authentication** should be there to access API’s/Web Services   
   of other parties. It may be either **Session/Token ID’s**.
2. **Username/Password** for user’s accessing websites.
3. **Sending Request either in XML/JSON.**
4. **Method Type** like **HTTP – CRUD** operations like **(Create, Retrieve, Update, Delete)**
5. **GET Calls.**

Note: - Over the network/HTTP calls are known as **Web Services**. By using Jar files locally are known as **API’s**. Some JAR files are OPEN SOURCE. We can change internal logic of that file also.

**There are 2 types of API’s: - 1) Simple Object Access Protocol (SOAP) 2) Rest API’s.**

|  |  |
| --- | --- |
| **SOAP** | **REST** |
| **SOAP** is a protocol | **REST** is an architectural type |
| **SOAP** stands for Simple Object Access Protocol | **REST** stands for Representational State Transfer |
| **SOAP** can’t use REST because it a protocol | **REST** can use SOAP because it’s a concept like HTTP |
| **JAX-WS** is the JAVA API for SOAP Web Services | **JAX-RS** is the JAVA API for RESTful Web Services |
| It requires **more bandwidth and resources** | It requires **less bandwidth and resources** |
| It permits **XML** data only | It permits data in **Plain Text, JSON, XML** etc. |
| It is **less preferred** than REST | REST **more preferred** than SOAP |

**C R U D** stands for **CREATE-POST, RETRIEVE-GET, UPDATE-PUT, DELETE-DELETE**

**HTTP Request: -  
1) URL 2) Headers 3)Payload**

**HTTP Response: -  
1) Status 2) Response Payload 3) Message**

**Account Information of any bank is present here**

**S1 – Client  
Consumer**

**S2 – Server  
Producer**

**Oracle DB**

**URI** is a combination of **URL & QUERY PARAMETER.**

1. Create an account on server – **POST** call. We need to pass JSON/XML payload to create account. **POST** call is used to create new entity.
2. Get an account by accNo/accName – **GET call.** Here we may pass URI, Path Parameter, Query Parameter, Headers etc. **GET** call is used to retrieve any information.
3. Update an existing entity – **PUT** call. We can Create and Update an entity using **PUT** call.
4. Delete an existing entity – **DELETE** call. Here, we pass **JSON/XML** payload.

**Status Response Code:** -

1. OK – 200
2. CREATED – 201
3. BAD REQUEST – 400
4. UNAUTHORISED - 401
5. FORBIDDEN - 403
6. NOT FOUND – 404
7. INTERNAL SERVER ERROR – 500

**Different HTTP methods are: - GET, POST, PUT, DELETE**

**POSTMAN, SOUP UI, JMETER are used to call API’s.**

**URL (End Point URL) + Service URL = URI. “?” represents Query Parameter**

**SWAGGER** is a API document where we can test our API’s.

**Automation of REST API: -**

To automate REST API, we need following Maven Dependencies: -

**httpClient and httpCore: -** To connect to the REST API and perform various operations.

**Json:** - To parse received output from JSON to simple readable format. (Use Maven Dependency 20160212)

**testNG: -** To create test cases.

**Jackson-core and Jackson-databind** maven dependencies (2.6.3) for Marshaling and UnMarshaling purpose.

**JSON (JavaScript Object Notation):** - When two systems communication with each other than they used JSON for communication purpose.

**HTTP Response**

**HTTP Request**

**Server**

**Client**

**{  
 “empid”: 1,  
“empname”: “Tom”,  
“salary”:250,  
“address”:”Pune”**

**}**

**JSON Schema**

On Server Side all data is present. But that data may present in Java Classes, NODE JS Classes, PHP Objects etc. But user cannot understand these objects. Hence, we convert these objects in JSON format which is **light weight, easy to understand.** These conversion is done at Server Side. **OBJ -> JSON (Server Side).**

On Client Side we get data in JSON. So, depending on devices/methods the **JSON** object is converted **OBJECT** and it is displayed to user on proper UI. **JSON -> OBJ (Client Side).**

There are various API’s which convert **OBJ -> JSON** (known as **Marshaling**) and **JSON -> OBJ (**known as **UnMarshaling).** The various API’s are: -  
**A)Google GSON B)Jackson C)Simple JSON D)JSON SOUP etc.**

**OBJ** is basically called as **POJO (Plain Old Java Object).**