WebApplication- interaction is between user and application/system where there is a UI(user interface) using which user send requests to system.

WebService/Webbased API- interaction is between 2 systems where there is no UI(user interface).

**Rest API Automation**

**WebService**: Webservice is a piece of software/program using which

2 or more web applications interact with each other over the network.

Communication happens through XML request and response.

eg:

EMI Calculator – used by Wellsfargo CORE, used by LoanOriginationSystem

paypal, ups shipping service – amazon, flipkart for order processing.

**API**: It is software to software interface not user interface.

It is a program that connects two systems. API defines the methods for one program to interact with other.

API can webbased, library based, program based etc..

With APIs applications talk to each other without user knowledge:

eg: amazon talks to paypal and ups systems for order processing.

**All web services are APIs but not all APIS are webservices.**

**examples for APIs which are not Webbased:**

Java docs API

system calls are invoked using interrupts by linux kernel based API

Desktop applications like excel, word docs use VBA and COM based APIs

Webbased API:

REST API

**WebServices:**

Soap-

Rest API-

Xml RPC based services

**Soap**-

SimpleObjectAccessProtocol – It is XML based **protocol** to access webservices

It is platform and language independent – Device interoperability

It is independent of protocol- http, ftp, tcp, udp

Soap only depends on XML based messaging-

<Soap:envelope>

<soap:header>

<soap:body>

<soap:fault>

xml namespace definitions, operations, endpoint, port, message are all defined using a standard- WSDL –webservice definition language.

It also supports inbuilt security ws-security

It is heavyweight.

**Rest API**- It is an architecture style which mainly depends on http protocol.

Easy to learn and understand.

Instead of using complex mechanisms like RPC pr SOAP- REST says interact

between the applications using simple http methods and status codes.

REST is lightweight when compared to heavy soap based standards.

Platform and language independent.

We can use xml , JSON or plain text for data transfer.

JSON is most frequently used in REST APIs which is again lightweight data transport mode.

We can explore by analyzing URI’s.

URL:

<http://www.webservicex.net/globalweather.asmx>

getCities

getCountries

URI:

https://endPointname/apiName/apiVersion/resourcePath?parameters

[**http://api.amazon.com/**orders/6789](http://api.amazon.com/orders/6789) - get

[**http://api.amazon.com/**orders**/**createOrder](http://api.amazon.com/orders/createOrder) - post

[**https://graph.facebook.com**/bgolub?fields=id,name,picture](https://graph.facebook.com/bgolub?fields=id,name,picture)

**http://api.amazon.com**/search/v1/products?name=”laptop”

what is a resource and endpoint?

**http://api.wellsfargo.com /**v1/customers/3456/accounts/3257367888

**http://api.wellsfargo.com /**v1/customers/3456/accounts/withdraw

**http://api.wellsfargo.com /**v1/customers/3456/accounts/open

**http://api.wellsfargo.com /**v1/customers/3456/accounts/close

**http://api.wellsfargo.com /**v1/customers/3456/accounts/deposit

**http://api.wellsfargo.com /**v1/locations

**http://api.wellsfargo.com /**v1/accounts

**http://api.wellsfargo.com /**v1/loans

**http://api.wellsfargo.com /**v1/payments

* **resource names in API URI are represented as nouns and actions in resources are represented as verbs.**

**http methods:**

CRUD- post, get, update, delete

**get – Used for retrieval of a resource**

It do not have request body.

we can use query parameters in url to send request data bu this has limitation

like 2048 characters is max length of URI.

query data is exposed on uri – so not secure if you are querying using sensitive data.

**http://api.amazon.com**/search/v1/products?name=”laptop”

here- ?name=”laptop” is query parameter

It is safe(chances of server side resources being modified when you just retrieve data or do a read only operation is very less) and idempotent(response is same for n>0 requests).

But not all retrievals are done using get.

<https://api.walmart.com/search?query=laptop>

<https://api.walmart.com/account/electrode/api/signin?uname=”abc”&pwd=”dssdfd>” – this is not recomended

signin is secure action, so we use post even though its retrieval.

https://api.walmart.com/account/electrode/api/signin

**post:** Creating a resource

post will have request body/payload.

payload is actual content excluding headers.

Both request and response has payload which we call request payload and response payload.

It is secure because we don’t send request data as query parameters rather we send as body.

post is considered neither safe(chances of server side resources being modified when you create data using write operation is more) not idempotent.

post act as put if resource is already created

**put:** It is used for updating a resource

put can also act as post if resource do not exist already.

put also has request body

**delete:** delete a resource

It do not have request body

head: like get without response payload …with only header info

trace – used for debugging to echo the contents of http request back to requester

options- supported operations on this service

connect

get and head are safe

post, put, delete are not safe

get, head, put, delete- idempotent

**http status codes:**

100

200 –

201- post –resource created

204-No content- after successful delete

300

400-

400- bad request

401- unauthorized

403-client forbidden

404- resource not found

500-

500- internal server error

503- service is currently unavailable

**What to test in Rest API:**

RestAPIs do not have GUI.

All we deal with is request header, request payload,

**response headers and response payload.**

**FunctionalityTesting**-

verify response headers like statuscode, status message, accept-type,

verify actual response- key is present(assertTrue), value presence(assertTrue/assertNotNull),value comparisions, dataacount,

dataorder

**we can do happypath scenarios(smoke test cases), valid tests, boundary values, and invalid tests using different combinations in request data.**

**In invalid testing: verify error codes…**

eg- I am testing a post request- after testing request and response is fine

id is generated.

query database whether that id exists in db and query data is same as previous post request data.

or after post do a get and verify the get response is same as post request.

post- can we create more than one product with same id

can we create more than one product at same time

can we delete a product using post

can we get multiple products by passing comma separated product ids

can we update a product using post

**Authentication Testing:**

Basic

OAuth 1 or 2

**SecurityTesting:**

**EndPointTesting:**

**Load Testing:**

low level:

correct status code

expected header info – value/pattern match

payload having expected content

unsupported methods

wrong content types

malformed content

content with wrong structure

dependency test- one request is dependent on other response

**How to test:**

Manual**- Postman**

**SoapUI** is a popular tool which supports both SOAP and REST APIS automation.

Programming- in java-**HttpClient, RestAssured**

javascript- chakram

**Challenges:**

**Test data combination**

* **required, optional**

**In 2 weeks sprint if some requirement fields are added/deleted in between**

* **challenge to recombine the test cases within short duration**

**For Huge response /request and different combinations- like min length/max length, datatypes etc**

**RestAPI Automation using Java:**

1. **HttpClient**
2. **RestAssured**

**Steps to automate APIs:**

1. Create a new maven project and add all dependencies(HttpClient, org.json,TestNG,ApachePOI) in pom.xml
2. Write code to call an API and get the response
3. Parse the response
4. Add assertions

TestNG,WebDriver- WebAutomation

TestNG,HttpClient-APIAutomation

WebServices testing - SoapUI

Soap web services

Rest APIs

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tools Vs Programming

RestAPIs -

Create Order - http://amazon.api.com/orders - post

i have an order id -

request body

, headers

response body, headers

API Automation:

HttpClient

RestAssured

Chakram(javascript)

Amazon order---

payment gateway

submit--- welsfargo rest api - payment success- result is used by amazon

and displayed

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Need API Specifications -

URIS-

http://api.dev.walmartlabs.com/v1/search?apiKey={apiKey}&lsPublisherId={Your LinkShare Publisher Id}&query=ipod

http://api.qa.walmartlabs.com/v1/search?apiKey={apiKey}&lsPublisherId={Your LinkShare Publisher Id}&query=ipod

http://api.stage.walmartlabs.com/v1/search?apiKey={apiKey}&lsPublisherId={Your LinkShare Publisher Id}&query=ipod

http://api.walmartlabs.com/v1/search?apiKey={apiKey}&lsPublisherId={Your LinkShare Publisher Id}&query=ipod

request

response

required fields/optional fields

data types of fields- name is string, id is number, creation- Date, PayMentType- Enum

min length -

max length

2-50

1-40

valid testcases - 200, 201, 204

invalid testcases - 400, 401, 404, 403....

boundary values

TestCases-

Response- Headers, Payload

status code

contetnt type

required fields - key is present, value is not null, count of elements,

optional - but if key is present then value cannot be null

as it is optional - u can omit both key /value

authentiation test(security):

Basic

Oauth

miss tokens -

401 - unauthorized

performance testing-

security testing

Swagger Docs/Wiki links/ever notes/confluence/word docs/excel docs

{

"id":678

"name":"kavita"

"address":{"city":"hyd","zip":500047}

subjects:[

{"name":"java","duration":2},

{"name":"js","duration":3},

{"name":"selenium","duration":2}

]

}

List<NameValuePair> json = new ArrayList<NameValuePair>();

json.add(new BAsicNameValuePAir("id","434"));

json.add(new BAsicNameValuePAir("name","kavitha"));

post.setEntity(new UrlEncodedFormEntity(json));

jobj.get("id") = 678

jobj.get("name") = kavitha

jobj1 = obj.getJsonObject("address")

jobj1.get("city")=hyd

jobj2 = obj.getJsonArray("subjects")

jobj2.get(0).get("duration");

jobj2.get(1).get("duration");

jobj2.get(2).get("duration");

org.json

JsonPath

jsonsimple

resposne from server as streamobject which we convert to string

converting string to JSON object is Json Parsing

post can act as put when resource already exists and u want to update the resource

put can act as post if there is no resource - we want to create resource

pot.setEntity(HttpEntity obj)- we use to set body of post, put

HttpEntity has multiple implementations:

UrlEncodedFormEntity

StringEntity

RestAssured----