WEB Services :

* A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system. XML is used to encode all communications to a web service. For example, a client invokes a web service by sending an XML message, then waits for a corresponding XML response. As all communication is in XML, web services are not tied to any one operating system or programming language--Java can talk with Perl; Windows applications can talk with Unix applications.
* Web services are self-contained, modular, distributed, dynamic applications that can be described, published, located, or invoked over the network to create products, processes, and supply chains. These applications can be local, distributed, or web-based. Web services are built on top of open standards such as TCP/IP, HTTP, Java, HTML, and XML.
* Web services are XML-based information exchange systems that use the Internet for direct application-to-application interaction. These systems can include programs, objects, messages, or documents.
* A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.

To summarize, a complete web service is, therefore, any service that:

* Is available over the Internet or private (intranet) networks
* Uses a standardized XML messaging system
* Is not tied to any one operating system or programming language
* Is self-describing via a common XML grammar
* Is discoverable via a simple find mechanism

## Components of Web Services

The basic web services platform is XML + HTTP. All the standard web services work using the following components

* SOAP (Simple Object Access Protocol)
* UDDI (Universal Description, Discovery and Integration)
* WSDL (Web Services Description Language)

All these components have been discussed in the [Web Services Architecture](https://www.tutorialspoint.com/webservices/web_services_architecture.htm)chapter.

## How Does a Web Service Work?

A web service enables communication among various applications by using open standards such as HTML, XML, WSDL, and SOAP. A web service takes the help of:

* XML to tag the data
* SOAP to transfer a message
* WSDL to describe the availability of service.

You can build a Java-based web service on Solaris that is accessible from your Visual Basic program that runs on Windows.

You can also use C# to build new web services on Windows that can be invoked from your web application that is based on JavaServer Pages (JSP) and runs on Linux.

## Example

Consider a simple account-management and order processing system. The accounting personnel use a client application built with Visual Basic or JSP to create new accounts and enter new customer orders.

The processing logic for this system is written in Java and resides on a Solaris machine, which also interacts with a database to store information.

The steps to perform this operation are as follows:

* The client program bundles the account registration information into a SOAP message.
* This SOAP message is sent to the web service as the body of an HTTP POST request.
* The web service unpacks the SOAP request and converts it into a command that the application can understand.
* The application processes the information as required and responds with a new unique account number for that customer.
* Next, the web service packages the response into another SOAP message, which it sends back to the client program in response to its HTTP request.
* The client program unpacks the SOAP message to obtain the results of the account registration process.

https://www.tutorialspoint.com/restful/restful\_introduction.htm

**RESTful** Web Services are basically REST Architecture based Web Services. In REST Architecture everything is a resource. RESTful web services are light weight, highly scalable and maintainable and are very commonly used to create APIs for web-based applications.

Web services based on REST Architecture are known as RESTful Web Services. These web services use HTTP methods to implement the concept of REST architecture. A RESTful web service usually defines a URI (Uniform Resource Identifier), which is a service that provides resource representation such as JSON and a set of HTTP Methods.

## What is REST?

REST stands for **RE**presentational **S**tate **T**ransfer. REST is a web standards based architecture and uses HTTP Protocol for data communication. It revolves around resources where every component is a resource and a resource is accessed by a common interface using HTTP standard methods. REST was first introduced by Roy Fielding in year 2000.

In REST architecture, a REST Server simply provides access to resources and the REST client accesses and presents the resources. Here each resource is identified by URIs/ Global IDs. REST uses various representations to represent a resource like Text, JSON and XML. JSON is now the most popular format being used in Web Services.

### HTTP Methods

The following HTTP methods are most commonly used in a REST based architecture.

* **GET** − Provides a read only access to a resource.
* **PUT** − Used to create a new resource.
* **DELETE** − Used to remove a resource.
* **POST** − Used to update an existing resource or create a new resource.
* **OPTIONS** − Used to get the supported operations on a resource.

## RESTFul Web Services

A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.

Web services based on REST Architecture are known as RESTful Web Services. These web services use HTTP methods to implement the concept of REST architecture. A RESTful web service usually defines a URI (Uniform Resource Identifier), which is a service that provides resource representation such as JSON and a set of HTTP Methods.

## What is a Resource?

REST architecture treats every content as a resource. These resources can be Text Files, Html Pages, Images, Videos or Dynamic Business Data. REST Server simply provides access to resources and REST client accesses and modifies the resources. Here each resource is identified by URIs/ Global IDs. REST uses various representations to represent a resource where Text, JSON, XML. The most popular representations of resources are XML and JSON.

### Representation of Resources

A resource in REST is a similar Object in Object Oriented Programming or is like an Entity in a Database. Once a resource is identified then its representation is to be decided using a standard format so that the server can send the resource in the above said format and client can understand the same format.

For example, in [RESTful Web Services - First Application](https://www.tutorialspoint.com/restful/restful_first_application.htm) chapter, a user is a resource which is represented using the following XML format −

<user>

<id>1</id>

<name>Mahesh</name>

<profession>Teacher</profession>

</user>

The same resource can be represented in JSON format as follows −

{

"id":1,

"name":"Mahesh",

"profession":"Teacher"

}

### Good Resources Representation

REST does not impose any restriction on the format of a resource representation. A client can ask for JSON representation whereas another client may ask for XML representation of the same resource to the server and so on. It is the responsibility of the REST server to pass the client the resource in the format that the client understands.

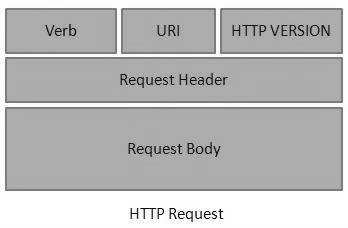
Following are some important points to be considered while designing a representation format of a resource in RESTful Web Services.

* **Understandability** − Both the Server and the Client should be able to understand and utilize the representation format of the resource.
* **Completeness** − Format should be able to represent a resource completely. For example, a resource can contain another resource. Format should be able to represent simple as well as complex structures of resources.
* **Linkablity** − A resource can have a linkage to another resource, a format should be able to handle such situations.

However, at present most of the web services are representing resources using either XML or JSON format.

RESTful Web Services make use of HTTP protocols as a medium of communication between client and server. A client sends a message in form of a HTTP Request and the server responds in the form of an HTTP Response. This technique is termed as Messaging. These messages contain message data and metadata i.e. information about message itself. Let us have a look on the HTTP Request and HTTP Response messages for HTTP 1.1.

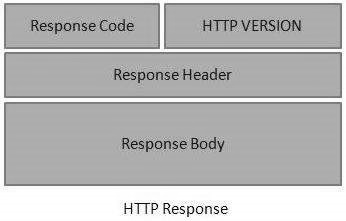
## HTTP Request



An HTTP Request has five major parts −

* **Verb** − Indicates the HTTP methods such as GET, POST, DELETE, PUT, etc.
* **URI** − Uniform Resource Identifier (URI) to identify the resource on the server.
* **HTTP Version** − Indicates the HTTP version. For example, HTTP v1.1.
* **Request Header** − Contains metadata for the HTTP Request message as key-value pairs. For example, client (or browser) type, format supported by the client, format of the message body, cache settings, etc.
* **Request Body** − Message content or Resource representation.

## HTTP Response



An HTTP Response has four major parts −

* **Status/Response Code** − Indicates the Server status for the requested resource. For example, 404 means resource not found and 200 means response is ok.
* **HTTP Version** − Indicates the HTTP version. For example HTTP v1.1.
* **Response Header** − Contains metadata for the HTTP Response message as keyvalue pairs. For example, content length, content type, response date, server type, etc.
* **Response Body** − Response message content or Resource representation.

### Example

As we have explained in the [RESTful Web Services - First Application chapter](https://www.tutorialspoint.com/restful/restful_first_application.htm), let us put http://localhost:8080/UserManagement/rest/UserService/users in the POSTMAN with a GET request. If you click on the Preview button which is near the send button of Postman and then click on the Send button, you may see the following output.



Here you can see, the browser sent a GET request and received a response body as XML.

Addressing refers to locating a resource or multiple resources lying on the server. It is analogous to locate a postal address of a person.

Each resource in REST architecture is identified by its URI (Uniform Resource Identifier). A URI is of the following format −

<protocol>://<service-name>/<ResourceType>/<ResourceID>

Purpose of an URI is to locate a resource(s) on the server hosting the web service. Another important attribute of a request is VERB which identifies the operation to be performed on the resource. For example, in [RESTful Web Services - First Application](https://www.tutorialspoint.com/restful/restful_first_application.htm) chapter, the URI is **http://localhost:8080/UserManagement/rest/UserService/users** and the VERB is GET.

## Constructing a Standard URI

The following are important points to be considered while designing a URI −

* **Use Plural Noun** − Use plural noun to define resources. For example, we've used users to identify users as a resource.
* **Avoid using spaces** − Use underscore (\_) or hyphen (-) when using a long resource name. For example, use authorized\_users instead of authorized%20users.
* **Use lowercase letters** − Although URI is case-insensitive, it is a good practice to keep the url in lower case letters only.
* **Maintain Backward Compatibility** − As Web Service is a public service, a URI once made public should always be available. In case, URI gets updated, redirect the older URI to a new URI using the HTTP Status code, 300.
* **Use HTTP Verb** − Always use HTTP Verb like GET, PUT and DELETE to do the operations on the resource. It is not good to use operations name in the URI.

### Example

Following is an example of a poor URI to fetch a user.

http://localhost:8080/UserManagement/rest/UserService/getUser/1

Following is an example of a good URI to fetch a user.

http://localhost:8080/UserManagement/rest/UserService/users/1

As per the REST architecture, a RESTful Web Service should not keep a client state on the server. This restriction is called Statelessness. It is the responsibility of the client to pass its context to the server and then the server can store this context to process the client's further request. For example, session maintained by server is identified by session identifier passed by the client.

RESTful Web Services should adhere to this restriction. We have seen this in the [RESTful Web Services - Methods](https://www.tutorialspoint.com/restful/restful_methods.htm) chapter, that the web service methods are not storing any information from the client they are invoked from.

**Consider the following URL −**

https://localhost:8080/UserManagement/rest/UserService/users/1

If you hit the above url using your browser or using a java based client or using Postman, result will always be the User XML whose Id is 1 because the server does not store any information about the client.

<user>

<id>1</id>

<name>mahesh</name>

<profession>1</profession>

</user>

## Advantages of Statelessness

Following are the benefits of statelessness in RESTful Web Services −

* Web services can treat each method request independently.
* Web services need not maintain the client's previous interactions. It simplifies the application design.
* As HTTP is itself a statelessness protocol, RESTful Web Services work seamlessly with the HTTP protocols.

## Disadvantages of Statelessness

Following are the disadvantages of statelessness in RESTful Web Services −

* Web services need to get extra information in each request and then interpret to get the client's state in case the client interactions are to be taken care of.

Caching refers to storing the server response in the client itself, so that a client need not make a server request for the same resource again and again. A server response should have information about how caching is to be done, so that a client caches the response for a time-period or never caches the server response.

Following are the headers which a server response can have in order to configure a client's caching −

|  |  |
| --- | --- |
| **Sr.No.** | **Header & Description** |
| 1 | **Date**  Date and Time of the resource when it was created. |
| 2 | **Last Modified**  Date and Time of the resource when it was last modified. |
| 3 | **Cache-Control**  Primary header to control caching. |
| 4 | **Expires**  Expiration date and time of caching. |
| 5 | **Age**  Duration in seconds from when resource was fetched from the server. |

Cache-Control Header

Following are the details of a Cache-Control header −

|  |  |
| --- | --- |
| **Sr.No.** | **Directive & Description** |
| 1 | **Public**  Indicates that resource is cacheable by any component. |
| 2 | **Private**  Indicates that resource is cacheable only by the client and the server, no intermediary can cache the resource. |
| 3 | **no-cache/no-store**  Indicates that a resource is not cacheable. |
| 4 | **max-age**  Indicates the caching is valid up to max-age in seconds. After this, client has to make another request. |
| 5 | **must-revalidate**  Indication to server to revalidate resource if max-age has passed. |

Best Practices

* Always keep static contents like images, CSS, JavaScript cacheable, with expiration date of 2 to 3 days.
* Never keep expiry date too high.
* Dynamic content should be cached for a few hours only.

What is a Payload : A rest Request/Response has header and payload(body)..if it is request body then it is called as Request Payload and response as Response Payload.

How do you set a payload request when it comes to post Request?

SoupUI

Web services : Web service is a piece of software that makes itself available over the internet and uses a standard XML massaging systems.

Web service is a piece of software or program that can be accessed by other programs over the web.

Types of web services :

1. RPC Remote Procedure Call (any call which is made outside …..

2. SOAP – Simple Object Access Protocol

3. REST – Representaional State Transfer

API : Anything which able to connect two devices/systems is called as an API. Application program interface (**API**) is a set of routines, protocols, and tools for building software applications. An **API** specifies how software components should interact.

API are library based or web based, but web services are only the web based.

All the web services are the API’s but not all the API are need to be web services. API ‘s are the super set of web services.

Ex. Amazon site – payment Gateway –simple api call which helping to connect two systems…this also can be web services….

Library based API --- java api.., docs api-( set of classes)

Kernel api – connecting C lang.

Protocols are different for all web services.

What QA will do with Web Services:

1. Going through the specifications docs ( docs given by the project lead) to understand functionality of web services
2. Analyse the input parameters to be passed and what response you are expecting
3. We don’t have any UI based(look and feel) for an appln, so its all the programs based and even the response is also available in either in XML or Json.
4. We will be testing the actual response by adding ASSERTIONS.

Assertions are nothing but conditions , checking actual response = expected response.

SOUPUI : Soap is the web service and **SoupUi is a tool to test web services**. Inititially it is developed to test the soap web services , but later versions supports both soap and REST web services….and this is mainly used for testing the web services. Tool is written in JAVA language.

Developers develops the Web Services and testers need to make sure these web services are working fine or not. They are called by multiple applications so we need to test the web services.

Soupui tool is developed by smart Bear company

First need to create the new project--- soupui project / rest project

For soupui project we need a WSDL …URL will be given by the company for testing. We can save the project anywhere in the system and we can import later from the dir and it will create the new project.

Import local project

Import packed project

Import composite project --- depending on the team or project we are working

SOAP is called as a protocol, because it follows WSDL…

WSDL is the web service definition language…it defines operations,name space ,messages ,prototypes , parameters which are used by web services .

Binding : binding the operations

Endpoint : from where we get the WSDL

Root elements in the emvelope ..data xml format

Rest is an architectural style

MEDHA NOTES :

|  |
| --- |
|  |
| hat is API? |
|  |  |
|  | Webservices: |
|  |  |
|  | What is a webservice? |
|  |  |
|  | Webservice is a piece of software/function that can be accessed by other programs over the web. |
|  |  |
|  |  |
|  | RPC |
|  | SOAP |
|  | REST API's |
|  |  |
|  |  |
|  | EmiCalculator - written a program in java which takes some inputs and gives you required output |
|  |  |
|  | inputs- amount, tenure |
|  | calc based on interest rate, tenure and amt |
|  | output- EMI |
|  |  |
|  | public Integer emiCalc(int amount, int tenure){ |
|  | //calc |
|  | return emi; |
|  | } |
|  |  |
|  |  |
|  | Citi |
|  |  |
|  | wellsfargo |
|  | chase |
|  |  |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | How Qa test webservices? |
|  | 1. Go through the specifications(docs) to understand functionality of webservice. |
|  | 2. Analyse input parameters to be passed and what response you are expecting. |
|  | 3. Functionalty Testing - You will test actual response by adding assertions. |
|  | What kind of assertions: |
|  | status code |
|  | header information - content type, content length |
|  | actual elements |
|  | actual data |
|  | data order |
|  | count of data |
|  | valid/invalid |
|  | boundary condtions |
|  |  |
|  | 4. securityTesting |
|  | 5. LoadTesting |
|  |  |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  |  |
|  | SoapUI - It is tool to test webservices.It was initially developed to test SOAP based websercvices |
|  | but latest SoapUI tool versions support testing both Soap and Rest webservices. |
|  |  |
|  | The tool itself is written in Java language. |
|  | \*\*\*\*\*\*\* |
|  |  |
|  | assertion is statement that makes sure actual = expected in programs or in tools. |
|  |  |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | Soap websrvices: |
|  |  |
|  | Soap is a protocol. It follows WSDL standards. |
|  | WSDL- webservice definition language. |
|  | WSDL - defines operations, namespace, messages, porttypes,parmaters which are used by webservice. |
|  |  |
|  | Soap webservices trasfers data (request and response) through xml format only |
|  | All the request and response parameters are binded inside soap envelope. |
|  | <soap:envelope namespace> |
|  | <soap:header/> |
|  | <soap:body> |
|  | <country>US</coubntry> |
|  | </soap:body> |
|  | </soap:envelope> |
|  |  |
|  | It has inbuilt ws security. |
|  |  |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | Rest APi's |
|  |  |
|  | Rest is an architecture. |
|  | It uses http methods to define its operations. |
|  |  |
|  | Mainly CRUD operations are concerned in Rest apis: |
|  |  |
|  | Api to create user - post |
|  | update the user details - put |
|  | delete the user - delete |
|  | retrieve the user - get |
|  | http methods: get, put, post, delete, trace, head, options |
|  |  |
|  | facebook apis: |
|  | http://facebook.api.com/users - post |
|  | register with fb: create user account |
|  | name |
|  | mailid |
|  | adress |
|  | dob |
|  | pwd |
|  |  |
|  | response- id is created for user |
|  |  |
|  | update profile : |
|  |  |
|  |  |
|  | http://facebook.api.com/users/id=2030 - put |
|  | city |
|  | zipcode |
|  |  |
|  |  |
|  | details updated succesfully |
|  |  |

SECOND CLASS:

SoapUI- Tool to test both Soap and Rest webservices.

TestSuite- Has set of TestCases

TestSteps in TestCase:

1. SoapRequest – which is used to create request for given webservice and get the response

2. JDBCRequest- To test DB w.r.to changes as per webservice functionality.

Eg- TestStep1-creates customer in db, we can verify using JDBC test step.

3.Properties and propertyTransfer

Properties- is to declare and load properties

Property transfer- to transfer properties from step to other step.

Useful to load request properties from external file.

Also if we have dependent webservices- O/p of one webservice is in/pp to other webservice.

Eg- 1.User should login to application- sessionId is created in response

2.User should track orderNumber –

What/How to test in webservices-Soap or Rest – both:

FunctionalityTesting: SoapUI tool or programitically

Statuscode,statusmessage

HeaderInformation- content type,contentlength

ActualResponse- element is present or not, count of data,

Order of data, element value, mandatory parameters

Set request with differnet combinations -Valid and invalid input data, boundary conditions.

EndPointTesting:

<http://qa.cdyne.com/ip2geo/ip2geo.asmx?wsdl>

<http://dev.cdyne.com/ip2geo/ip2geo.asmx?wsdl>

<http://ws1.cdyne.com/ip2geo/ip2geo.asmx?wsdl>

<http://ws2.cdyne.com/ip2geo/ip2geo.asmx?wsdl>

LoadTesting – SoapUI, Jmeter

SecurityTesting – SoapUI, programmatically

THIRD CLASS :

|  |
| --- |
|  |
| RestAPI: |
|  |  |
|  | crud operations: |
|  |  |
|  | get - retrieval - say u pass order id and get order details |
|  | get wil not have request body |
|  | post - creation of resource- give order detials - order gets created with new id |
|  | post will have request body |
|  | put - updating the resource - to update existing order id details |
|  | delete - delete the resource - delete the order with order id |
|  |  |
|  | what to test: |
|  | statuscode |
|  | contetnt type, length - headers |
|  | actual response |
|  |  |
|  | Different AUthentication types: |
|  | Auth - to secure the api's |
|  | Basic - user eneters uname and pwd , token id is generated which is to be persisted for further requests eg: talentscreen |
|  | OAuth - key and access tokens are generated as per api's - eg:twitter apis |
|  | public api's - api key - eg:google |
|  |  |
|  |  |
|  | Load Testing - performance of the api |
|  | tps - threads per second |
|  | bps - bytes per second |
|  |  |
|  | Threads- no fo concurrent/parallel user requests |
|  |  |
|  | Strategy- simple, burst, variance |
|  |  |
|  | TEstDelay -Time in ms b/n each batch of requests |
|  | Random - random factor for test load |
|  | eg: test delay -400, rf =0.5 |
|  | delay is distributd b/n 400\*0.5 - |
|  | so delay will be 200-400 |
|  |  |
|  |  |
|  |  |
|  | o/p: |
|  |  |
|  | min- shoertest time step has taken to execute(ms) |
|  | max- longest time step has taken to execute(ms) |
|  | avg- avg time step has taken to execute(ms) |
|  | cnt - no of time sstep has been executed |
|  | err- assertion Errors |
|  |  |
|  | Security Testing - SoapUI tool supports in built |
|  |  |
|  |  |
|  |  |
|  |  |
|  | SoapMock service: used when you are not able to user actual service |
|  |  |
|  | https://www.soapui.org/soap-mocking/service-mocking-overview.html |
|  | Data Driven Testing- running same test step with different set of data |
|  | SoapUIPRo- DataSource and DataSourceLoop step for data driven testing |
|  |  |