REST API :

Questions :

1. What is a Web Services?
2. What is an API? What is the difference between Web Service and API?
3. What is the difference between SOAP and REST?
4. What is a Resource and endpoint?

**What is a Web Services?**

**Web Services** : A web Service is any piece of software/ program that makes it available over the internet and uses a standard XML messaging system.

**Web Service** is any piece of SW/program which helps to connect two systems over the internet and it uses XML messaging system.

**Web Service** is a piece of SW/program that can be accessed by other programs over the Web.

Examples: EMI Calculator WS used by many many banks, paypal acct,ups services can be used by many online shopping’s like AMAZON,flipkart etc..

**What is an API?**

**API**: Anything /Program that connects two system is called as an API.

Api’s can be Web base, library base ,program bases etc.

**All Web services are API’s , but all Api’s cant not be Web Services.**

If API is based on web and using http communicating between two systems are called as Web based API’s. Ex. REST API

API’s which are not web services are: Java docs(which are library bases), System calls(invoked using interrupts by Linux kernel based API),etc…

**Types of Web Services:**

RPC: Remote Procedure Call

SOAP: Simple Object Access Protocol

REST: Representational State Transfer

**SOAP**: It is XML based protocol to access Web services.

These are language independent and platform independent.-device interoperability.

Soap web services only depends on XML based messaging.

Independent of protocol which is can be used whether it could be Http, Ftp, tcp. UDP protocols..

**Soap messaging request/response looks like:**

It will have <soap:envelope> which has <soap:header> and <soap:body>

<soap:envelope> -- mandatory

<soap:header> -- optional

<soap:body> -- mandatory

<soap:error> -- optional –in case of error scenario we have it in soap response

It is all XML based, using **envelope** we are able to communicate between the systems.

why we call soap as a **protocol**, because it need to bind to WSDL file its strictly like a standard.

WSDL - wizdel standard definition Language which defines namespace, type of **operations** that are supported in WS, endpoints, post messages.It also supports **inbuilt** security called **WS**-**Security,** error handling.

Every **request or response is embedded within soap envelope making it heavy** weight even for simple data transfers.

User should have prior understanding on how WSDL works before learning SOAP

web services.

Soap based URL ends with WSDL.

**Rest** : IT IS An architectural style mainly depends depends on stateless, client-server, cacheable communications protocol – http protocol.

Instead of using complex mechanism like RPC, soap it says interact between the applications using simple http. Rest is light weight when compared to Heavy Soap based Standard, no need to follow WSDL.

Rest is also language and platform independent.

Rest supports multiple formats like, XML, JSON, html and plaintext file too.

Mainly xml and json are used to transport the data. and json is frequently used in REST Api ‘s which is again leight weight in data transport mode.

Rest can be easily used in presence of firewalls and it can be made secure by

different external authentication mechanisms.

Rest is easy to learn and no expensive tools required.

Rest API’s are faster.

Ex. https://developers.facebook.com/docs/graph-api

<https://graph.facebook.com/bgolub?fields=id,name,picture>

https://www.googleapis.com/books/v1/mylibrary/bookshelves

https: secured protocol

[www.googleapis.com](http://www.googleapis.com): endpoint : from where we are getting the resource

books: Api Name

bgolub?fields=id,name,picture: Resource

sometime we do have version number, given immediately after the apiname.

Rest API URI examples:

<http://talentscreen.api.com/candidates>

<http://talentscreen.api.com/subjects>

endpoint- talentscreen.api.com

resources- candidates,subjects

Ex. <https://www.welsfargo.com/v1/customers/56777/accounts/77777/withdraw>

Ex. <https://www.welsfargo.com/v1/customers/56777/accounts/77777/open>

Ex. <https://www.welsfargo.com/v1/customers/56777/accounts/77777/deposit>

resource – All the resources are represented as noun, actions performed on resources are called as verbs. In above examples customer is the main resource and account is the sub **resources** which are represented as **nouns** and withdraw, open and deposits are the **actions** which represented as **verbs**.

URL is subset of URI:

URI – url+ resource

A url can always be URI i but not all URI’s are URL’s

**HTTP Methods:** Get,Post,Put,Delete(Crud Operations), Head,Trace, options, connect..

Get: it do not have Request Body. We can use query parameters in url to send request data bit this has some limitations like 2048 characters is the maximum length of URI. Whatever the query data which is exposed on the URI is not safe (Secure) if you are using sensitive data.

It is safe and idempotent. ( get method are mostly read only operations, hence chances of being modifying the data on server side are less hence get is Safe)

Idempotents means.. when we send more than one resource and if we get the same response for (n>0) are called as Idempotent.but not all retrievals are not done using GET.

http://www.walmart.com/search/query?laptop

We have Safe and Idempotent methods. safe means secure

## What is HTTP?

The Hypertext Transfer Protocol (HTTP) is designed to enable communications between **clients and servers.**

HTTP works as a request-response protocol between a client and server.

A web browser may be the client, and an application on a computer that hosts a web site may be the server.

Example: A client (browser) submits an HTTP request to the server; then the server returns a response to the client. The response contains status information about the request and may also contain the requested content.

Two commonly used methods for a request-response between a client and server are: GET and POST.

* **GET** - Requests data from a specified resource
* **POST** - Submits data to be processed to a specified resource

## The GET Method

**Note that the query string (name/value pairs) is sent in the URL of a GET request:**

/test/demo\_form.php**?name1=value1&name2=value2**

**Some other notes on GET requests:**

* GET requests can be cached
* GET requests remain in the browser history
* GET requests can be bookmarked
* GET requests should never be used when dealing with sensitive data
* GET requests have length restrictions
* GET requests should be used only to retrieve data

## The POST Method

**Note that the query string (name/value pairs) is sent in the HTTP message body of a POST request:**

POST /test/demo\_form.php HTTP/1.1  
Host: w3schools.com  
**name1=value1&name2=value2**

**Some other notes on POST requests:**

* POST requests are never cached
* POST requests do not remain in the browser history
* POST requests cannot be bookmarked
* POST requests have no restrictions on data length
* The following table lists some other HTTP request methods:

|  |  |
| --- | --- |
| **Method** | **Description** |
| HEAD | Same as GET but returns only HTTP headers and no document body |
| PUT | Uploads a representation of the specified URI |
| DELETE | Deletes the specified resource |
| OPTIONS | Returns the HTTP methods that the server supports |
| CONNECT | Converts the request connection to a transparent TCP/IP tunnel |

## You can divide HTTP methods into two main categories safe and idempotent. Safe methods are HTTP methods that do not modify the resource e.g. [a GET request  is safe](http://javarevisited.blogspot.com/2012/03/get-post-method-in-http-and-https.html) because it doesn't modify the resource you are requesting e.g. data of a Book. Another safe HTTP method is HEAD, which doesn't change the resource representation on the Server, but all other HTTP methods e.g. POST, PUT, or DELETE are non-safe. Coming to idempotent methods, they are HTTP methods which can be called multiple times and they will produce the same result. They are considered the safe option to update a resource on the Server. Some examples of idempotent HTTP methods are GET, PUT, and PATCH. No matter how many times you call them, they will produce the same result with same URI.

## What is Safe Methods in HTTP

These are HTTP methods which don't change the resource on the server side. For example using a GET or a HEAD request on a resource URL should NEVER change the resource. Safe methods can be **cached**and **prefetched**without any repercussions or side-effect to the resource . Here is an example of safe method

GET /order/123 HTTP/1.1

This will retrieve the order with orderId 123. No matter how many times you execute this method, the order in the server will not be modified or impacted. That's why [GET method](http://java67.blogspot.com/2014/08/difference-between-post-and-get-request.html) is a safe method.

## What are Idempotent Methods in HTTP

These are methods which are safe from multiple calls i.e. they produce same result irrespective of how many times you call them. They change the resource in Server every time you call them but the end result is always same. Maths is good place to explain idempotent methods, consider the following example:

int i = 30; // idempotent

i++; // not idempotent

Here the assignment operation is idempotent, no matter how many times you execute this statement, i will always be 4. The**second example is not idempotent**. Executing this 10 times will result in a different outcome as when running 5 times. Since both examples are changing the value of i, both are non-safe methods.  
  
Idempotency is an important thing while building a fault-tolerant RESTful API. Idempotency is also the reason of [why should you use PUT over POST to update a resource in REST](http://java67.blogspot.com/2015/09/top-10-restful-web-service-interview-questions-answers.html).   
  
For example, suppose a client wants to update a resource through POST. Since **POST is not an idempotent method**, calling it multiple times may result in incorrect updates.  
  
In the real world its quietly likely that your POST request may timeout, what will happen to the resource that. Is the resource actually updated? Does the timeout happened during sending the request to the server, or the response to the client?  
  
Can we safely retry again, or do we need to figure out first what has happened with the resource? By using idempotent methods like PUT, you don't have to answer this question, but we can safely resend the request until we actually get a response back from the server.

### Summary

Here is nice overview of which HTTP methods are safe and Idempotent:

* GET is both Safe and Idempotent.
* HEAD is also both safe and idempotent.
* OPTIONS is also safe and idempotent.
* PUT is not safe but idempotent.
* DELETE is not safe but idempotent.
* POST is neither safe nor idempotent.
* PATCH is also neither safe nor idempotent.