**What is REST?**

**RESTful** Web Services are basically REST Architecture based Web Services.

REST stands for **Re**presentational **S**tate **T**ransfer.

REST is a web standard based architecture and uses HTTP Protocol for data communication.

In REST Architecture everything is a resource. It is accessed by a common interface using HTTP standard methods.

RESTful web services are:

1. Light weight
2. Highly scalable
3. Maintainable

REST was first introduced by Roy Fielding in year 2000.

In REST architecture,

**REST Server** simply provides access to resources

**REST client** accesses and presents the resources.

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.

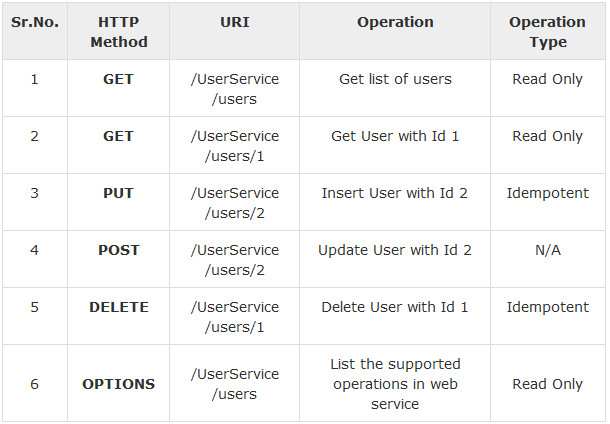
**HTTP Methods:**

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

1. A web service is a collection of open protocols and standards used for exchanging data between applications or systems.
2. 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 like inter-process communication on a single computer.
3. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.
4. 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.
5. 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.



**Jersey Framework** to create RESTful Web Services.

Jersey framework implements **JAX-RS 2.0** API, which is a standard specification to create RESTful Web Services

## Setup Jersey Framework Libraries

[https://jersey.java.net/download.htm](https://jersey.java.net/download.html)

Copy all the required jars from the below locations:

* \jaxrs-ri-2.17\jaxrs-ri\api
* \jaxrs-ri-2.17\jaxrs-ri\ext
* \jaxrs-ri-2.17\jaxrs-ri\lib

## 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.

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.

In [RESTful Web Services](https://www.tutorialspoint.com/restful/restful_first_application.htm) 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.

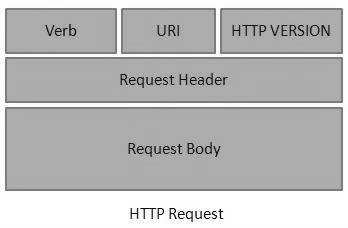
* **Understandability** − Both the Server and the Client should be able to understand and utilize the representation format of the resource.
* **Completeness** − Format should be able to represent a resource completely. For example, a resource can contain another resource. Format should be able to represent 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.

**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.

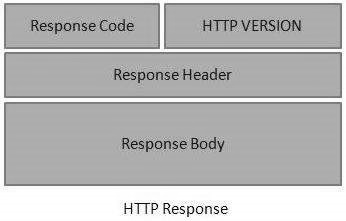
**HTTP Request**



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 key value pairs. For example, content length, content type, response date, server type, etc.
* **Response Body** − Response message content or Resource representation.

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>

## 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.

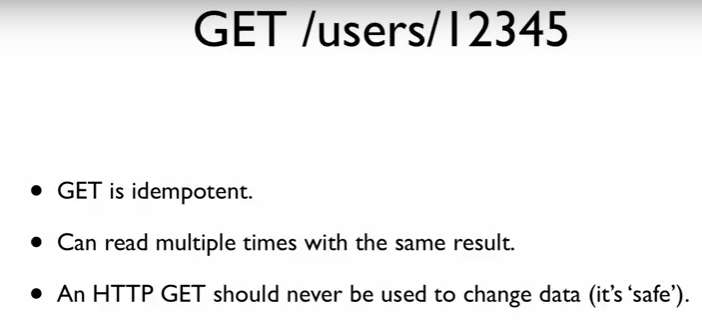
<http://localhost:8080/UserManagement/rest/UserService/getUser/1>

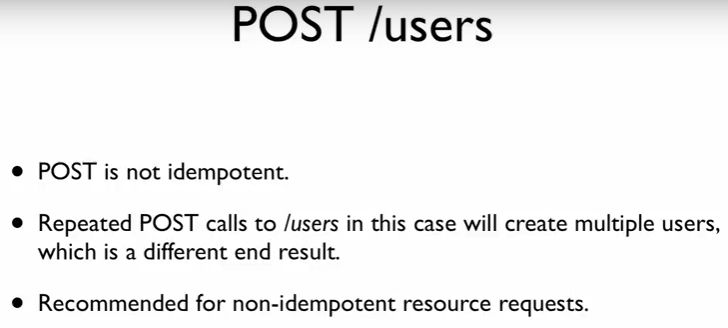
Following is an example of a good URI to fetch a user.

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

Here are important points to be considered:

* **GET** operations are read only and are safe.
* **PUT** and **DELETE** operations are **idempotent** means their result will always same no matter how many times these operations are invoked.
* **PUT** and **POST** operation are nearly same with the difference lying only in the result where **PUT** operation is **idempotent** and **POST** operation can cause different result.





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.

## 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 to configure a client's caching –

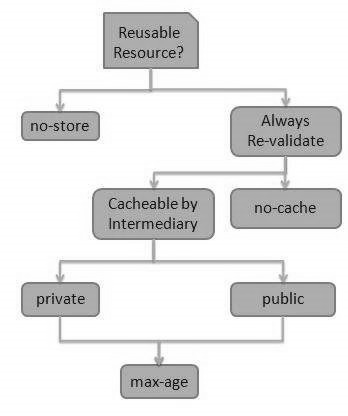
* 1. **Date**: Date and Time of the resource when it was created.
  2. **Last Modified:** Date and Time of the resource when it was last modified
  3. **Cache-Control:** Primary header to control caching.
  4. **Expires**: Expiration date and time of caching
  5. **Age**: Duration in seconds from when resource was fetched from the server

## Cache-Control Header

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 must 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.



Following are the best practices to be adhered to while designing a RESTful Web Service:

* **Validation** − Validate all inputs on the server. Protect your server against SQL or NoSQL injection attacks.
* **Session Based Authentication** − Use session based authentication to authenticate a user whenever a request is made to a Web Service method.
* **No Sensitive Data in the URL** − Never use username, password or session token in a URL, these values should be passed to Web Service via the POST method.
* **Restriction on Method Execution** − Allow restricted use of methods like GET, POST and DELETE methods. The GET method should not be able to delete data.
* **Validate Malformed XML/JSON** − Check for well-formed input passed to a web service method.
* **Throw generic Error Messages** − A web service method should use HTTP error messages like 403 to show access forbidden, etc.

|  |  |
| --- | --- |
| **HTTP Code** | **Description** |
| 200 | OK |
| 201 | Created |
| 204 | No Content |
| 304 | Not Modified |
| 400 | Bad Request |
| 401 | Unauthorized |
| 403 | Forbidden |
| 404 | Not Found |
| 409 | Conflict |
| 500 | Internal Server Error |

**JAX-RS** stands for JAVA API for RESTful Web Services.

JAX-RS is a JAVA based programming language API and specification to provide support for created RESTful Web Services.

**JAX-RS** uses annotations available from Java SE 5 to simplify the development of JAVA based web services creation and deployment.

It also provides supports for creating clients for RESTful Web Services.

**Jersey, a reference implementation of JAX-RS 2.0** by Oracle, in the [RESTful Web Services](https://www.tutorialspoint.com/restful/restful_first_application.htm).

|  |  |
| --- | --- |
| **Annotation** | **Description** |
| **@Path** | Relative path of the resource class/method. |
| **@GET** | HTTP Get request, used to fetch resource |
| **@PUT** | HTTP PUT request, used to create resource. |
| **@POST** | HTTP POST request, used to create/update resource. |
| **@DELETE** | HTTP DELETE request, used to delete resource. |
| **@HEAD** | HTTP HEAD request, used to get status of method availability |
| **@Produces** | States the HTTP Response generated by web service. For example, APPLICATION/XML, TEXT/HTML, APPLICATION/JSON etc. |
| **@Consumes** | States the HTTP Request type. For example, application/x-www-formurlencoded to accept form data in HTTP body during POST request. |
| **@PathParam** | Binds the parameter passed to the method to a value in path. |
| **@QueryParam** | Binds the parameter passed to method to a query parameter in the path |
| **@MatrixParam** | Binds the parameter passed to the method to a HTTP matrix parameter in path. |
| **@HeaderParam** | Binds the parameter passed to the method to a HTTP header. |
| **@CookieParam** | Binds the parameter passed to the method to a Cookie. |
| **@FormParam** | Binds the parameter passed to the method to a form value. |
| **@DefaultValue** | Assigns a default value to a parameter passed to the method |
| **@Context** |  |

**Interview Question and Answer:**

1. **What are webservices?**

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 like inter-process communication on a single computer.

1. **What are RESTful webservices?**

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 a service, provides resource representation such as JSON and set of HTTP Methods.

1. **What is messaging in RESTful webservices?**

A client sends a message in form of a HTTP Request and server responds in form of a HTTP Response. This technique is termed as Messaging. These messages contain message data and metadata i.e. information about message itself.

1. **What is caching?**

Caching refers to storing server response in client itself so that a client needs not to make server request for same resource again and again.

A server response should have information about how a caching is to be done so that a client caches response for a period or never caches the server response.