## What is REST architecture?

REST stands for REpresentational State Transfer. REST is web standards based architecture and uses HTTP Protocol. It revolves around resource 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 2000.

In REST architecture, a 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 representation to represent a resource like text, JSON, XML. JSON is the most popular one.

### HTTP methods

Following four HTTP methods are commonly used in REST based architecture.

* **GET** − Provides a read only access to a resource.
* **POST** − Used to create a new resource.
* **DELETE** − Used to remove a resource.
* **PUT** − Used to update a existing resource or create a new resource.

## Introduction to 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 webservices uses 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.

## Creating RESTFul Webservice

In next chapters, we'll create a webservice say user management with following functionalities −

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr.No.** | **URI** | **HTTP Method** | **POST body** | **Result** |
| 1 | /UserService/users | GET | empty | Show list of all the users. |
| 2 | /UserService/addUser | POST | JSON String | Add details of new user. |
| 3 | /UserService/getUser/:id | GET | empty | Show details of a user. |

## Restful Web Services

Restful Web Services is a **stateless client-server** architecture where web services are resources and can be identified by their URIs.

REST Client applications can use HTTP GET/POST methods to invoke Restful web services. REST doesn’t specify any specific protocol to use, but in almost all cases it’s used over HTTP/HTTPS. When compared to SOAP web services, these are lightweight and doesn’t follow any standard. We can use XML, JSON, text or any other type of data for request and response.

### Java RESTful Web Services API

Java API for RESTful Web Services (JAX-RS) is the Java API for creating REST web services. JAX-RS uses annotations to simplify the development and deployment of web services. JAX-RS is part of JDK, so you don’t need to include anything to use it’s annotations.

### Restful Web Services Annotations

Some of the important JAX-RS annotations are:

* @Path: used to specify the relative path of class and methods. We can get the URI of a webservice by scanning the Path annotation value.
* @GET, @PUT, @POST, @DELETE and @HEAD: used to specify the HTTP request type for a method.
* @Produces, @Consumes: used to specify the request and response types.
* @PathParam: used to bind the method parameter to path value by parsing it.

### Restful Web Services and SOAP

1. SOAP is a protocol whereas REST is an architectural style.
2. SOAP server and client applications are tightly coupled and bind with the WSDL contract whereas there is no contract in REST web services and client.
3. Learning curve is easy for REST when compared to SOAP web services.
4. REST web services request and response types can be XML, JSON, text etc. whereas SOAP works with XML only.
5. JAX-RS is the Java API for REST web services whereas JAX-WS is the Java API for SOAP web services.

### REST API Implementations

There are two major implementations of JAX-RS API.

1. **Jersey**: [Jersey](https://jersey.github.io/) is the reference implementation provided by Sun. For using Jersey as our JAX-RS implementation, all we need to configure its servlet in web.xml and add required dependencies. Note that JAX-RS API is part of JDK not Jersey, so we have to add its dependency jars in our application.
2. **RESTEasy**: [RESTEasy](https://resteasy.github.io/) is the JBoss project that provides JAX-RS implementation.

### Java Restful Web Services Tutorial

Let’s see how easy to create Restful web service using Jersey and then RESTEasy. We will be exposing following methods over HTTP and use Chrome Postman extension to test these.

|  |  |  |
| --- | --- | --- |
| URI | HTTP Method | Description |
| /person/{id}/getDummy | GET | Returns a dummy person object |
| /person/add | POST | Adds a person |
| /person/{id}/delete | GET | Delete the person with ‘id’ in the URI |
| /person/getAll | GET | Get all persons |
| /person/{id}/get | GET | Get the person with ‘id’ in the URI |

The key elements of a RESTful implementation are as follows:

1. **Resources** – The first key element is the resource itself. Let assume that a web application on a server has records of several employees. Let's assume the URL of the web application is **http://demo.guru99.com**. Now in order to access an employee record resource via REST, one can issue the command **http://demo.guru99.com/employee/1** - This command tells the web server to please provide the details of the employee whose employee number is 1.
2. **Request Verbs** - These describe what you want to do with the resource. A browser issues a GET verb to instruct the endpoint it wants to get data. However, there are many other verbs available including things like POST, PUT, and DELETE. So in the case of the example **http://demo.guru99.com/employee/1** , the web browser is actually issuing a GET Verb because it wants to get the details of the employee record.
3. **Request Headers** – These are additional instructions sent with the request. These might define the type of response required or the authorization details.
4. **Request Body** - Data is sent with the request. Data is normally sent in the request when a POST request is made to the REST web service. In a POST call, the client actually tells the web service that it wants to add a resource to the server. Hence, the request body would have the details of the resource which is required to be added to the server.
5. **Response Body** – This is the main body of the response. So in our example, if we were to query the web server via the request **http://demo.guru99.com/employee/1** , the web server might return an XML document with all the details of the employee in the Response Body.
6. **Response Status codes** – These codes are the general codes which are returned along with the response from the web server. An example is the code 200 which is normally returned if there is no error when returning a response to the client.

**Restful Methods**

The below diagram shows mostly all the verbs (POST, GET, PUT, and DELETE) and an example of what they would mean.

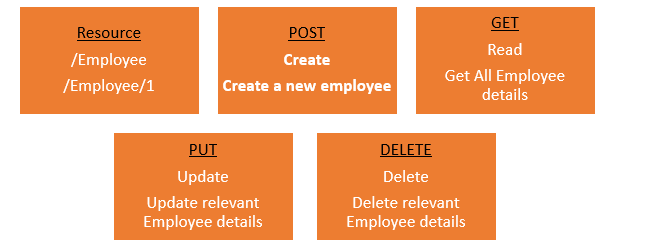
Let's assume that we have a RESTful web service is defined at the location. **http://demo.guru99.com/employee** . When the client makes any request to this web service, it can specify any of the normal HTTP verbs of GET, POST, DELETE and PUT. Below is what would happen If the respective verbs were sent by the client.

1. **POST** – This would be used to create a new employee using the RESTful web service
2. **GET** - This would be used to get a list of all employee using the RESTful web service
3. **PUT** - This would be used to update all employee using the RESTful web service
4. **DELETE** - This would be used to delete all employee using the RESTful web service

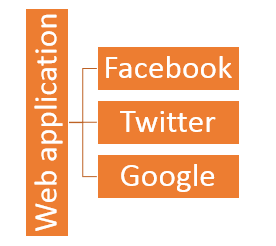
Let's take a look from a perspective of just a single record. Let's say there was an employee record with the employee number of 1.

The following actions would have their respective meanings.

1. **POST** – This would not be applicable since we are fetching data of employee 1 which is already created.
2. **GET** - This would be used to get the details of the employee with Employee no as 1 using the RESTful web service
3. **PUT** - This would be used to update the details of the employee with Employee no as 1 using the RESTful web service
4. **DELETE** - This is used to delete the details of the employee with Employee no as 1

[](https://www.guru99.com/images/3-2016/032316_0816_RESTfulWebS1.png)

Facebook, Twitter, and Google expose their functionality in the form of Restful web services. This allows any client application to call these web services via REST.

[](https://www.guru99.com/images/3-2016/032316_0816_RESTfulWebS2.png)

1. The event of Devices – Nowadays, everything needs to work on[Mobile](https://www.guru99.com/mobile-testing.html)devices, whether it be the mobile device, the notebooks, or even car systems.

Can you imagine the amount of effort to try and code applications on these devices to talk with normal web applications? Again Restful API's can make this job simpler because as mentioned in point no 1, you really don't need to know what is the underlying layer for the device.

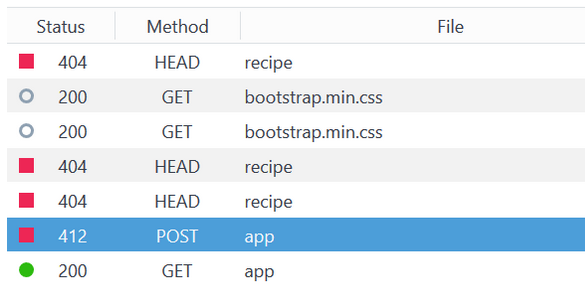
1. Finally is the event of the Cloud – Everything is moving to the cloud. Applications are slowly moving to cloud-based systems such as in Azure or Amazon. Azure and Amazon provide a lot of API's based on the Restful architecture. Hence, applications now need to be developed in such a way that they are made compatible with the Cloud. So since all Cloud-based architectures work on the REST principle, it makes more sense for web services to be programmed on the REST based architecture to make the best use of Cloud-based services.

# HTTP Status Codes for REST API

[](https://metamug.com/article/status-codes-for-rest-api.html)

by [Deepak Mishra](https://metamug.com/article/status-codes-for-rest-api.html)

Last Updated: Oct 11, 2018



[//](https://metamug.com/article/status-codes-for-rest-api.html): # ()

### Introduction

HTTP Status codes help categorize the response. Sometimes they are self-explanatory (e.g. 404) and sometimes they are backed by information (e.g. 201). A REST API MUST implement these status codes well to convey the right information to its clients. Correct status codes help client app developers handle responses better. Here will focus more on 4XX Status Codes for communicating errors with the rest clients.

### 2XX Content

Browsers generate a success indicator for the 2XX status code. So 2XX status codes should be used specify a successful request.

* **201 Created** - Used for POST request to create resource.
* **202 Accepted** - Request accepted by server, but cannot respond immediately.
* **203 Non-Authoritative Information** - Retrieve information expected from 202 request.
* **204 No Content** - Response doesn't have a payload.

When the server has a long operation to be performed, it responds with 202. But since REST is stateless, it cannot respond to the request later. The client, therefore after a certain interval, requests again for the same resource and gets the data intended for request with 202 response. This follow-up request is replied with status 203 by the server.

### 3XX Redirect

The REST API developer should maintain old resources, in case he is migrating to new ones. 3XX series codes are displayed as errors in browser console and should be used to indicate resource relocation.

* **301 Moved Permanently** - Server changed the URI and asking the client to use a new URI.
* **302 Found** - Server wants to retain old URI, providing an alternate URI. Since [Cool URIs don't change](https://www.w3.org/Provider/Style/URI)
* **304 Not Modified** - Server instructing the client to use its cached results. 304 is considered a redirect because the server is redirecting the client to its own cache for the response and not to another URI.

### 4XX Status Codes (Client Errors)

When the client makes a mistake, the server should notify the client of 4XX error. The most popular being 404. When the server cannot find the resource the client requested. Browsers show errors in their console for 4XX series, even when they necessarily are errors. For example, when the resource is deleted, the server SHOULD return 410 instead of 200 stating that the resource has been deleted.

* **400 Bad Request** - Request is missing some critical information. Some important Header.
* **401 Unauthorised** - Client doesn't have valid credentials/token (Authentication).
* **402 Payment Required** - Paid Service
* **403 Forbidden** - Client has requested a forbidden resource. This error can be sent even when the client has correct credentials. (Authorization).
* **404 Not Found** - Resource does not exist on the server.
* **405 Method not Found** - Request method (HTTP Verb) sent by the client is not supported.
* **406** - Accept Header sent by the client is not supported.
* **408 request timeout** - Server didn't receive a complete request from the client
* **409 Conflict** - Client attempting to create a duplicate record, which is not allowed.
* **410 Gone** - The requested resource has been deleted.
* **411 Length Required** - Server will not accept the request without the Content-Length Header.
* **412 Precondition Failed** - The server understands the request, but the format of the request is incorrect.
* **415 unsupported media type** - Server doesnt understand the payload format. e.g Server parses xml, json but payload contains yml. Content-Type or Content-Encoding may also mislead the server and it will send 415 error without checking payload.
* **419 Too Many Requests** - Server is unable to handle further requests temporarily.
* **422 Unprocessable Entity** - Request body cannot be parsed.
* **429 Too Many Requests** - Used for [rate limiting](https://blog.apisyouwonthate.com/what-is-api-rate-limiting-all-about-1819a390ab06).

Server throws 400 when the user is sending an invalid request. Something that's not even HTTP request. For example, the client is sending just plaintext, with no reference to method or protocol. It throws 422 when Content-Type header says application/json, but XML is being sent.

There are several other 4XX status codes that can be used as well like **407 proxy authentication required**, **416 requested range not satisfiable**, **417 expectation failed** but they are less common.

These client codes are most common and best used by REST APIs to convey client errors. There are other status codes in 4XX series and you can roll out your own for something specific.

### 5XX Server Errors

* **500 Internal Server Error** is the generic server error which the server shouldn't send. Because it gives very little information to the REST client about the error on the server, we only use it when we don't know the cause of the error. But the server is taking the responsibility of the error.

# SOAP vs REST Web Services

There are many differences between SOAP and REST web services. The important 10 differences between SOAP and REST are given below:

|  |  |  |
| --- | --- | --- |
| **No.** | **SOAP** | **REST** |
| 1) | SOAP is a **protocol**. | REST is an **architectural style**. |
| 2) | SOAP stands for **Simple Object Access Protocol**. | REST stands for **REpresentational State Transfer**. |
| 3) | SOAP **can't use REST** because it is a protocol. | REST **can use SOAP** web services because it is a concept and can use any protocol like HTTP, SOAP. |
| 4) | SOAP **uses services interfaces to expose the business logic**. | REST **uses URI to expose business logic**. |
| 5) | **JAX-WS** is the java API for SOAP web services. | **JAX-RS** is the java API for RESTful web services. |
| 6) | SOAP **defines standards**to be strictly followed. | REST does not define too much standards like SOAP. |
| 7) | SOAP **requires more bandwidth** and resource than REST. | REST **requires less bandwidth** and resource than SOAP. |
| 8) | SOAP **defines its own security**. | RESTful web services **inherits security measures** from the underlying transport. |
| 9) | SOAP **permits XML** data format only. | REST **permits different** data format such as Plain text, HTML, XML, JSON etc. |
| 10) | SOAP is **less preferred** than REST. | REST **more preferred** than SOAP. |