RESTful Web Services Tutorial with REST API Example

What is Restful Web Services?

Restful Web Services is a lightweight, maintainable, and scalable service that is built on the REST architecture. Restful Web Service, expose API from your application in a secure, uniform, stateless manner to the calling client. The calling client can perform predefined operations using the Restful service. The underlying protocol for REST is HTTP. REST stands for REpresentational State Transfer.

In this REST API tutorial, you will learn-

RESTful Key Elements

REST Web services have really come a long way since its inception. In 2002, the Web consortium had released the definition of WSDL and SOAP web services. This formed the standard of how web services are implemented.

In 2004, the web consortium also released the definition of an additional standard called RESTful. Over the past couple of years, this standard has become quite popular. And is being used by many of the popular websites around the world which include Facebook and Twitter.

REST is a way to access resources which lie in a particular environment. For example, you could have a server that could be hosting important documents or pictures or videos. All of these are an example of resources. If a client, say a web browser needs any of these resources, it has to send a request to the server to access these resources. Now REST services defines a way on how these resources can be accessed.

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 services, 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 services. In a POST call, the client actually tells the REST web services 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 RESTful API 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 REST API example of what they would mean.

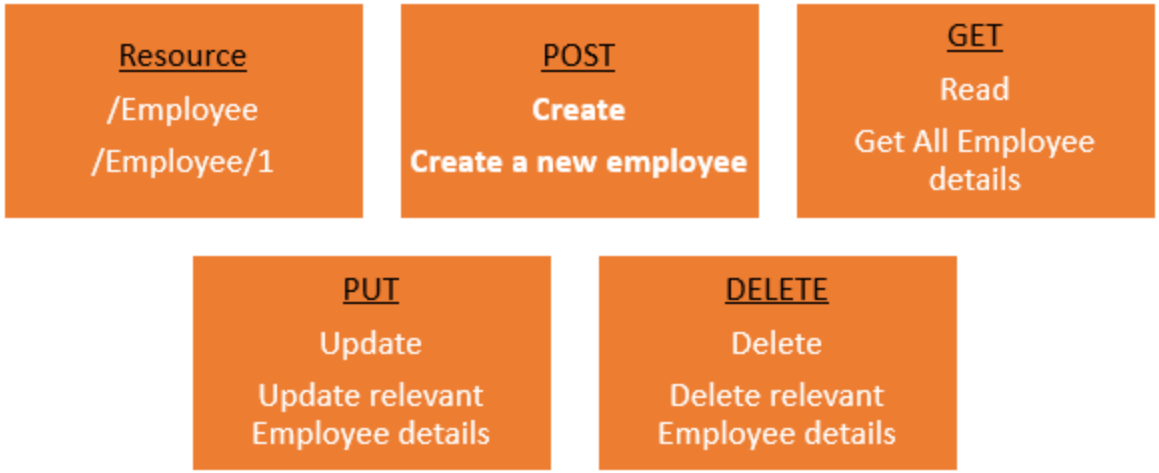
Let's assume that we have a RESTful web service is defined at the location. **http://demo.guru99.com/employee** . When the client makes any request to this web service, it can specify any of the normal HTTP verbs of GET, POST, DELETE and PUT. Below is what would happen If the respective verbs were sent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

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



Why Restful

Restful mostly came into popularity due to the following reasons:

1. Heterogeneous languages and environments – This is one of the fundamental reasons which is the same as we have seen for SOAP (/soap-simple-object-access-protocol.html) as well.

- It enables web applications that are built on various programming languages to communicate with each other

- With the help of Restful services, these web applications can reside on different environments, some could be on Windows, and others could be on Linux.

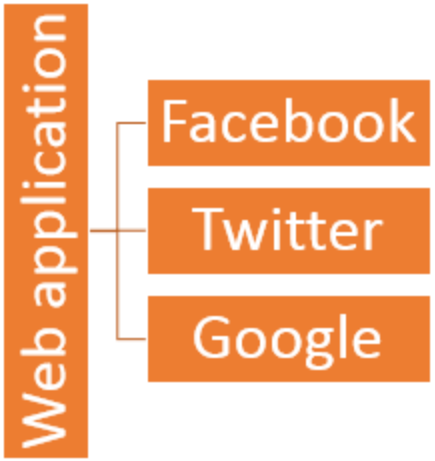
But in the end, no matter what the environment is, the end result should always be the same that they should be able to talk to each other. Restful web services offer this flexibility to applications built on various programming languages and platforms to talk to each other.

The below picture gives an example of a web application which has a requirement to talk to other applications such Facebook, Twitter, and Google.

Now if a client application had to work with sites such as Facebook, Twitter, etc. they would probably have to know what is the language Facebook, Google and Twitter are built on, and also on what platform they are built on.

Based on this, we can write the interfacing code for our web application, but this could prove to be a nightmare.

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.



2. The event of Devices – Nowadays, everything needs to work on Mobile (/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.

3. 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 services based architecture to make the best use of Cloud-based services.

Restful Architecture

An application or architecture considered RESTful or REST-style has the following characteristics

1. State and functionality are divided into distributed resources – This means that every resource should be accessible via the normal HTTP commands of GET, POST, PUT, or DELETE. So if someone wanted to get a file from a server, they should be able to issue the GET request and get the file. If they want to put a file on the server, they should be able to either issue the POST or PUT request. And finally, if they wanted to delete a file from the server, they an issue the DELETE request.
2. The architecture is client/server, stateless, layered, and supports caching –

- Client-server is the typical architecture where the server can be the web server hosting the application, and the client can be as simple as the web browser.
- Stateless means that the state of the application is not maintained in REST.

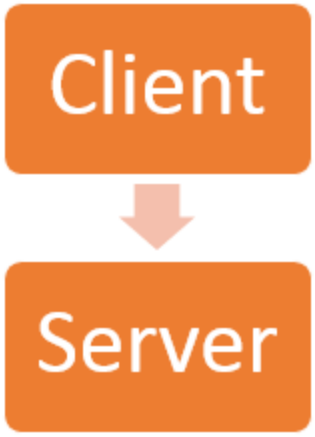
For example, if you delete a resource from a server using the DELETE command, you cannot expect that delete information to be passed to the next request.

In order to ensure that the resource is deleted, you would need to issue the GET request. The GET request would be used to first get all the resources on the server. After which one would need to see if the resource was actually deleted.

RESTFul Principles and Constraints

The REST architecture is based on a few characteristics which are elaborated below. Any RESTful web service has to comply with the below characteristics in order for it to be called RESTful. These characteristics are also known as design principles which need to be followed when working with RESTful based services.

1. **RESTFul Client-Server**



This is the most fundamental requirement of a REST based architecture. It means that the server will have a RESTful web service which would provide the required functionality to the client. The client send's a request to the web service on the server. The server would either reject the request or comply and provide an adequate response to the client.

2. **Stateless**

The concept of stateless means that it's up to the client to ensure that all the required information is provided to the server. This is required so that server can process the response appropriately. The server should not maintain any sort of information between requests from the client. It's a very simple independent question-answer sequence. The client asks a question, the server answers it appropriately. The client will ask another question. The server will not remember the previous question-answer scenario and will need to answer the new question independently.

3. **Cache**

The Cache concept is to help with the problem of stateless which was described in the last point. Since each server client request is independent in nature, sometimes the client might ask the server for the same request again. This is even though it had already asked for it in the past. This request will go to the server, and the server will give a response. This increases the traffic across the network. The cache is a concept implemented on the client to store requests which have already been sent to the server. So if the same request is given by the client, instead of going to the server, it would go to the cache and get the required information. This saves the amount of to and fro network traffic from the client to the server.

4. **Layered System**

The concept of a layered system is that any additional layer such as a middleware layer can be inserted between the client and the actual server hosting the RESTFul web service (The middleware layer is where all the business logic is created. This can be an extra service created with which the client could interact with before it makes a call to the web service.). But the introduction of this layer needs to be transparent so that it does not disturb the interaction between the client and the server.

5. Interface/Uniform Contract

This is the underlying technique of how RESTful web services should work. RESTful basically works on the HTTP web layer and uses the below key verbs to work with resources on the server

- POST - To create a resource on the server
- GET - To retrieve a resource from the server
- PUT - To change the state of a resource or to update it
- DELETE - To remove or delete a resource from the server