**Http methods Best Practices**

First step is to identify the resources, such as photos, people (and others) that the service handles. Next, determine the best way to map the resource behavior to HTTP methods. But mapping operations is easy once you know the characteristics of the HTTP methods.

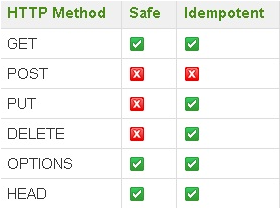
**Safety**

An HTTP method is considered safe if the request made with it does not cause any side effects and does not change the state of the resource. For example, the operation of retrieving a photo is safe, because the operation does not imply any update on the photo.

**Idempotent**

If you get the same response no matter how many times you request it, the method is said to be idempotent. So, you can consider a repetitive update operation on an existing photo as idempotent, if executed with the same parameters, the result is the same every time the operation is performed.

But not all methods are safe or idempotent. Check the table below.



### HTTP Methods

HTTP methods are at the heart of REST. Clients, like browsers, applications using the APIs of Facebook or Google, or any other clients rely on them. There is some confusion on how and when to use them, therefore we are going to cover them below.

#### **GET**

We already know that GET must be safe and idempotent. Web clients know this as well and do not expect side effects when repeating GET requests. Use GET only for retrieving the representation of resources or other information.

#### **POST**

POST is most commonly used for creating resources. For this purpose, the body of the request must contain the representation of the resource. You can also find it useful in other situations, i.e. when updating several resources at once, on tunneling requests, or when avoiding browser limitations (like query size).

#### **PUT**

Because POST is not idempotent, avoid using it when updating a resource and use PUT instead, as shown below (this is a common mistake). Though you may return a body in the response, it is not required by the HTTP 1.1 specification.

#### **DELETE**

As you can understand from its name, this method is used for deleting resources. The interesting thing is that you might not always be able to implement it as idempotent. If DELETE is idempotent, your service has to be aware of all the deleted resources and return the same HTTP status, for example 204 No Content, even if the resource has been previously deleted.

However, there might be other restrictions, i.e. security policies, that will require your service to return the 404 Not Found status instead.

#### **OPTIONS**

Use it to query the server about the HTTP methods available for resource manipulation or to ping the service. Be aware that the request does not contain a body, but it’s free to return the representation of a resource in the response.

#### **HEAD**

Use this method to look for a particular resource on the server. It usually returns the same headers as GET. The HTTP specification requires that nor request neither response contain a body.