**5 Basic REST API Design Guidelines**

As soon as we start working on an API, design issues arise. A robust and strong design is a key factor for API success. A poorly designed API will indeed lead to misuse or–even worse–no use at all by its intended clients (application developers).

Creating and providing a state of the art API requires taking into account:

* RESTful API principles as described in the literature ([Roy Fielding](https://en.wikipedia.org/wiki/Roy_Fielding), [Martin Fowler](https://en.wikipedia.org/wiki/Martin_Fowler), HTTP specification, etc.).
* The API practices of the Web Giants like Google, Microsoft, Facebook, PayPal and other big companies.

Designing a REST API raises questions and issues for which there is no universal answer. REST best practices are still being debated and consolidated, which is what makes this job fascinating.

Here are the 5 basic design guidelines that [make a RESTful API](http://www.restcase.com):

1. Resources (URIs)
2. HTTP methods
3. HTTP headers
4. Query parameters
5. Status codes

Let's go over each one and explain a bit.

## I. Resources (URIs)

### Names and Verbs

To describe your resources, use concrete names and not action verbs.   
For decades, computer scientists used action verbs in order to expose services in an RPC way, for instance: getUser(1234)createUser(user)deleteAddress(1234)

By contrast, the RESTful approach is to use:   
GET /users/1234 POST /users (with JSON describing a user in the body)DELETE /addresses/1234

### URI case

When it comes to naming resources in a program, there are 3 main types of case conventions: CamelCase, snake\_case, and spinal-case. They are just a way of naming the resources to resemble natural language while avoiding spaces, apostrophes, and other exotic characters. This habit is universal in programming languages where only a finite set of characters is authorized for names.

### **CamelCase**

CamelCase has been popularized by the Java language. It intends to emphasize the beginning of each word by making the first letter uppercase. E.g. camelCase, currentUser, etc. Aside from debates about its readability, its main drawback is to be ineffective in contexts which are not case sensitive.

### **snake\_case**

snakecase has been widely used for years by C programmers, and more recently in Ruby. Words are separated by underscores “”, thus letting a compiler or an interpreter understand it as a single symbol, but also allowing readers to separate words fluently. However, its popularity has decreased due to a lot of abuses in C programs with over-extended or too short names. Unlike camel case, there are very few contexts where snake case is not usable. Examples: snakecase, currentuser, etc.

### **spinal-case**

spinal-case is a variant of snake case which uses hyphens “-” to separate words. The pros and cons are quite similar to those of snake case, with the exception that some languages do not allow hyphens in symbol names (for variable, class, or function naming). You may find it referred to as lisp-case because it is the usual way to name variables and functions in lisp dialects. It is also the traditional way of naming folders and files in UNIX and Linux systems. Examples: spinal-case, current-user, etc.

According to RFC3986, URLs are “case sensitive” (except for the scheme and the host).   
In practice, though, a sensitive case may create dysfunctions with APIs hosted on a Windows system.

It is recommended to use spinal-case (which is highlighted by RFC3986), this case is used by Google, PayPal and other big companies.

## II HTTP Methods

As stated earlier, one of the key objectives of the REST approach is using HTTP as an application protocol in order to avoid shaping a homemade API. Hence, we should systematically use HTTP verbs to describe what actions are performed on the resources and facilitate the developer’s work handling recurrent CRUD operations (create, read, update and delete).

The following methods are usually observed:

### GET (read)

The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.

### HEAD

Same as GET but transfers the status line and header section only.

### POST (write)

A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.

### PUT (update)

Replaces all current representations of the target resource with the uploaded content.

### DELETE (delete)

Removes all current representations of the target resource given by a URI.

### OPTIONS

Describes the communication options for the target resource.

## III. HTTP Headers

HTTP header fields provide required information about the request or response, or about the object sent in the message body.

There are 4 types of HTTP message headers:

1. **General Header**: these header fields have general applicability for both request and response messages.
2. **Client Request Header**: these header fields have applicability only for request messages.
3. **Server Response Header**: these header fields have applicability only for response messages.
4. **Entity Header**: these header fields define meta information about the entity-body or, if no BODY is present, about the resource identified by the request.

## IV Query Parameters

### Paging

It is necessary to anticipate the paging of your resources in the early design phase of your API. It is indeed difficult to foresee precisely the progression of the amount of data that will be returned. Therefore, we recommend paginating your resources with default values when they are not provided by the calling client, for example with a range of values [0-25].

### Filtering

Filtering consists in restricting the number of queried resources by specifying some attributes and their expected values. It is possible to filter a collection on several attributes at the same time and to allow several values for one filtered attribute.

### Sorting

Sorting the result of a query on a collection of resources. A sort parameter should contain the names of the attributes on which the sorting is performed, separated by a comma.

### Searching

A search is a sub-resource of a collection. As such, its results will have a different format than the resources and the collection itself. This allows us to add suggestions, corrections, and information related to the search. Parameters are provided the same way as for a filter, through the query-string, but they are not necessarily exact values, and their syntax permits approximate matching.

## V. Status Codes

It is very important that as a RESTful API, you make use of the proper HTTP Status Codes, especially when [mocking RESTful API](http://www.restcase.com).

The mostly used status codes:

**200 – OK**: Everything is working.

**201 – CREATED**: New resource has been created.

**204 – NO CONTENT**: The resource was successfully deleted (no response body).

**304 – NOT MODIFIED**: The date returned is cached data (data has not changed).

**400 – BAD REQUEST**: The request was invalid or cannot be served. The exact error should be explained in the error payload, i.e., „The JSON is not valid“.

**401 – UNAUTHORIZED**: The request requires user authentication.

**403 – FORBIDDEN**: The server understood the request, but is refusing it or the access is not allowed.

**404 – NOT FOUND**: There is no resource behind the URI.

**500 – INTERNAL SERVER ERROR**: API developers should avoid this error. If an error occurs in the global catch blog, the stack trace should be logged and not returned as response.

## Summary

REST is not something new. REST is a return to the Web the way it was before the age of the big application server, through its emphasis on the early Internet standards, URI, and HTTP.

Resource modeling requires a careful consideration based on the business needs, technical considerations (clean design, maintainability, etc.) and cost-benefit analysis of various approaches discussed earlier so that the API design brings out the best API consumer interaction experience.

We went over the basics of [designing and developing RESTful API](http://www.restcase.com). These guidelines hopefully will help in creating clean, easy to use, and understandable APIs.