10 Best Practices for Better RESTful API

1. Use nouns but no verbs

For an easy understanding use this structure for every resource:

Resource	GET read	POST create	PUT update	DELETE
/cars	Returns a list of cars	Create a new car	Bulk update of cars	Delete all cars
/cars/711	Returns a specific car	Method not allowed (405)	Updates a specific car	Deletes a specific car

Do not use verbs:

```
/getAllCars
/createNewCar
/deleteAllRedCars
```

2. GET method and query parameters should not alter the state

Use **PUT, POST** and **DELETE** methods instead of the **GET** method to alter the state.
Do not use **GET** for state changes:

```
GET /users/711?activate or
GET /users/711/activate
```

3. Use plural nouns

Do not mix up singular and plural nouns. Keep it simple and use only plural nouns for all resources.

```
/cars instead of /car
/users instead of /user
/products instead of /product
/settings instead of /setting
```

4. Use sub-resources for relations

If a resource is related to another resource use subresources.

```
GET /cars/711/drivers/ Returns a list of drivers for car 711
GET /cars/711/drivers/4 Returns driver #4 for car 711
```

5. Use HTTP headers for serialization formats

Both, client and server, need to know which format is used for the communication. The format has to be specified in the HTTP-Header.

Content-Type defines the request format.
Accept defines a list of acceptable response formats.

6. Use HATEOAS

Hypermedia **a**s **t**he **E**ngine **o**f **A**pplication **S**tate is a principle that hypertext links should be used to create a better navigation through the API.

```
{
  "id": 711,
  "manufacturer": "bmw",
  "model": "X5",
  "seats": 5,
  "drivers": [
   {
    "id": "23",
    "name": "Stefan Jauker",
    "links": [
     {
     "rel": "self",
     "href": "/api/v1/drivers/23"
    }
   ]
  }
 ]
}
```

7. Provide filtering, sorting, field selection and paging for collections

Filtering:

Use a unique query parameter for all fields or a query language for filtering.

```
GET /cars?color=red Returns a list of red cars
GET /cars?seats<=2 Returns a list of cars with a maximum of 2 s
```

Sorting:

Allow ascending and descending sorting over multiple fields.

```
GET /cars?sort=-manufactorer,+model
```

This returns a list of cars sorted by descending manufacturers and ascending models.

Field selection

Mobile clients display just a few attributes in a list. They don’t need all attributes of a resource. Give the API consumer the

ability to choose returned fields. This will also reduce the network traffic and speed up the usage of the API.

```
GET /cars?fields=manufacturer,model,id,color
```

Paging

Use limit and offset. It is flexible for the user and common in leading databases. The default should be limit=20 and offset=0

```
GET /cars?offset=10&limit=5
```

To send the total entries back to the user use the custom HTTP header: X-Total-Count.

Links to the next or previous page should be provided in the HTTP header link as well. It is important to follow this link header values instead of constructing your own URLs.

```
Link: <https://blog.mwaysolutions.com/sample/api/v1/cars?offset
<https://blog.mwaysolutions.com/sample/api/v1/cars?offset=50&li
<https://blog.mwaysolutions.com/sample/api/v1/cars?offset=0&lim
<https://blog.mwaysolutions.com/sample/api/v1/cars?offset=5&lim
```

8. Version your API

Make the API Version mandatory and do not release an unversioned API. Use a simple ordinal number and avoid dot notation such as 2.5.

We are using the url for the API versioning starting with the letter „v“

```
/blog/api/v1
```

9. Handle Errors with HTTP status codes

It is hard to work with an API that ignores error handling. Pure returning of a HTTP 500 with a stacktrace is not very helpful.

Use HTTP status codes

The HTTP standard provides over 70 status codes to describe the return values. We don’t need them all, but

there should be used at least a mount of 10.

200 – OK – Eyerything is working
201 – OK – New resource has been created
204 – OK – The resource was successfully deleted

304 – Not Modified – The client can use cached data

400 – Bad Request – The request was invalid or cannot be served. The exact error should be explained in the error payload. E.g. „The JSON is not valid“
401 – Unauthorized – The request requires an 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.
422 – Unprocessable Entity – Should be used if the server cannot process the enitity, e.g. if an image cannot be formatted or mandatory fields are missing in the payload.

500 – Internal Server Error – API developers should avoid this error. If an error occurs in the global catch blog, the stracktrace should be logged and not returned as response.

Use error payloads

All exceptions should be mapped in an error payload. Here is an example how a JSON payload should look like.

```
{
  "errors": [
    {
     "userMessage": "Sorry, the requested resource does not exist",
     "internalMessage": "No car found in the database",
     "code": 34,
     "more info":
"http://dev.mwaysolutions.com/blog/api/v1/errors/12345"
    }
  ]
}
```

10. Allow overriding HTTP method

Some proxies support only **POST** and **GET** methods. To support a RESTful API with these limitations, the API needs a way to override the HTTP method.

Use the custom HTTP Header **X-HTTP-Method-Override** to overrider the POST Method.