**10 Best Practices for Better RESTful API**

**RESTful API**:

Web APIs has become a very important topic in the last year. We at M-Way Solutions are working every day with different backend systems and therefore we know about the importance of a clean API design.

Typically, we use a RESTful design for our web APIs. The concept of REST is to separate the API structure into logical resources. There are used the HTTP methods GET, DELETE, POST and PUT to operate with the resources.

**10 best practices to design a clean RESTful API**:

These are 10 best practices to design a clean RESTful API:

1. **Use nouns but no verbs**
2. **GET method and query parameters should not alter the state**
3. **Use plural nouns**
4. **Use sub-resources for relations**
5. **Use HTTP headers for serialization formats**
6. **Use HATEOAS**
7. **Provide filtering, sorting, field selection and paging for collections**
8. **Version your API**
9. **Handle Errors with HTTP status codes**
10. **Allow overriding HTTP method**

**1) Use nouns but no verb**:

For an easy understanding use this structure for every resource:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Resource | GET  Red | POST  Create | PUT  Update | DELETE  delete |
| /cars | Returns a list of cares | Create a new car | Bulk update of cars | Delete all cars |
| /car/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 sub resources.

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**:

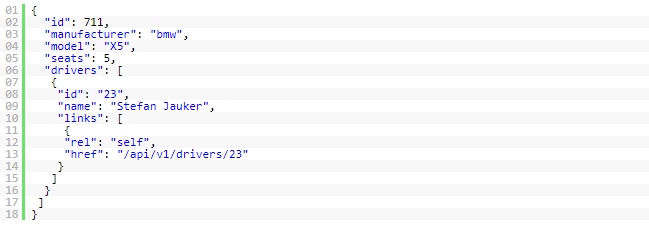
Both, client and server, need to know which format is used for the communication. The format has to be specified in the HTTP-Header.

Content-Type defines the request format.

Accept defines a list of acceptable response formats.

**6) Use HATEOAS**:

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



**HATEOAS Driven REST APIs**:

HATEOAS (Hypermedia as the Engine of Application State) is a constraint of the REST application architecture that keeps the RESTful style architecture unique from most other network application architectures. The term “hypermedia” refers to any content that contains links to other forms of media such as images, movies, and text.

This architectural style lets you use hypermedia links in the response contents so that the client can dynamically navigate to the appropriate resource by traversing the hypermedia links. This is conceptually the same as a web user navigating through web pages by clicking the appropriate hyperlinks in order to achieve a final goal.

For example, below given JSON response may be from an API like HTTP GET

{

"departmentId": 10,

"departmentName": "Administration",

"locationId": 1700,

"managerId": 200,

"links": [

{

"href": "10/employees",

"rel": "employees",

"type" : "GET"

}

]

}

Details about HATEOAS: <https://restfulapi.net/hateoas/>

**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 seats

**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=15&limit=5>; rel="next",

<https://blog.mwaysolutions.com/sample/api/v1/cars?offset=50&limit=3>; rel="last",

<https://blog.mwaysolutions.com/sample/api/v1/cars?offset=0&limit=5>; rel="first",

<https://blog.mwaysolutions.com/sample/api/v1/cars?offset=5&limit=5>; rel="prev",

**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 Status-Code element in a server response, is a 3-digit integer where the first digit of the Status-Code defines the class of response and the last two digits do not have any categorization role. There are 5 values for the first digit:

|  |  |
| --- | --- |
| **Serial Number** | **Code and Description** |
| 1 | **1xx: Informational**  It means the request has been received and the process is continuing. |
| 2 | **2xx: Success**  It means the action was successfully received, understood, and accepted. |
| 3 | **3xx: Redirection**  It means further action must be taken in order to complete the request. |
| 4 | **4xx: Client Error**  It means the request contains incorrect syntax or cannot be fulfilled. |
| 5 | **5xx: Server Error**  It means the server failed to fulfill an apparently valid request. |

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 amount of 10.

|  |  |
| --- | --- |
| **Status Code** | **Details** |
| **2xx: Successful** | |
| 200 - OK | Everything is working |
| 201-OK | New resource has been created |
| 204-OK | The resource was successfully deleted |
| **3xx: Redirection** | |
| 304  Not Modified | The client can use cached data |
| **4xx: Client Error** | |
| 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 |
| 402  Payment Required | You cannot use this code yet. |
| 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  Un-processable  Entity | Should be used if the server cannot process the entity, e.g. if an image cannot be formatted or mandatory fields are missing in the payload. |
| **5xx: Server Error** | |
| 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. |

**For Details about status code**:

<https://www.restapitutorial.com/httpstatuscodes.html>

**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.

**Link**:

<https://blog.mwaysolutions.com/2014/06/05/10-best-practices-for-better-restful-api/>

10 Best Practices for Better RESTful API