**Best practices for Rest API**

1. **Use nouns not verb**
   1. Noun :- /cars, /cars/711, /cars/711/drivers **√**
   2. Verbs :- /getAllCars, /createNewCar  **X**
2. **Use only plural noun to keep it simple**
   1. /cars instead of /car
3. **Get method & Query parameter should not alter the state**
   1. GET /users/711/activate  **X**
   2. GET /users/711?activate **X**
4. **Use sub-resources for relations**
   1. GET /cars/711/drivers/ **[**Returns a list of drivers for car 711**]**
   2. GET /cars/711/drivers/4 **[**Returns driver #4 for car 711**]**
5. **Use HTTP headers for serialization formats**
   1. Both, client and server, need to know which format is used for the communication. The format has to be specified in the HTTP-Header.
   2. Content-Type defines the request format.
   3. Accept defines a list of acceptable response formats.
6. **Version you API**
   1. Make the API Version mandatory and do not release an unversioned API.
   2. Use a simple ordinal number and avoid dot notation such as 2.5.

**/blog/api/v1**

1. **Provide Filtering, Sorting, Field Selection and Paging capability for collection**
   1. **Filtering**
      1. 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*

* 1. **Sorting**
     1. 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.

* 1. **Field selection**
     1. 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

* 1. **Paging**
     1. Use limit and offset. The default should be limit=20 and offset=0

GET /cars?offset=10&limit=5

* + 1. **HTTP header: X-Total-Count**
       1. To send the total entries back to the user use the custom HTTP header: X-Total-Count.
       2. 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.
       3. 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",

1. **Handle Errors with HTTP status codes**
   1. 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.
      1. 200 – OK – Eyerything is working
      2. 201 – OK – New resource has been created
      3. 204 – OK – The resource was successfully deleted
      4. 304 – Not Modified – The client can use cached data
      5. 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“
      6. 401 – Unauthorized – The request requires an user authentication
      7. 403 – Forbidden – The server understood the request, but is refusing it or the access is not allowed.
      8. 404 – Not found – There is no resource behind the URI.
      9. 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.
      10. 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.
   2. **Use error payloads**
      1. 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"

}

]

}

1. **Use HATEOAS**
   1. **H**ypermedia **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"

}

]

}

]

}