

# Swagger Editor, Laravel REST API – sortowanie, filtrowanie, paginacja

Początek laboratorium:

- przejść pod adres <https://editor.swagger.io/>,
- usunąć całą zawartość, wkleić zawartość pliku *Actors API Lab007 start.yaml*,
- pobrać na pulpit archiwum *Lab007\_PAB\_start.zip*, w którym umieszczony jest projekt startowy do wykonania zadań oraz rozpakować to archiwum,
- ~~przejsć do rozpakowanego folderu oraz w przypadku posiadania innych ustawień niż domyślne (np. połączenia z bazą), wykonać ich zmianę w .env.example oraz start....,~~
- uruchomić skrypt *start.bat* (Windows, 2x kliknięciem) lub *start.sh* (inne systemy, przez polecenie `bash start.sh`),
- wyświetlić zawartość bazy danych *SQLite* z pliku *database.sqlite* za pomocą np. *DBeaver'a*.

Zadania (Swagger, Laravel, sortowanie):

- <https://github.com/NationalBankBelgium/REST-API-Design-Guide/wiki>

### Zadanie 7.1:

Zapoznać się z „przewodnikiem projektowania” REST API opracowanego przez jednego z współtwórców REST API Narodowego Banku Belgii, opisującym podjęte przez nich decyzje oraz wybrane sposoby/strategie/konwencje.

#### **REST**

REST API Design Goals, REST Constraints, REST Resources, REST Resources Design Workflow, REST Resources Naming, REST Resources URIs, REST Resources Parameters, REST Resources Single items and collections, REST Resources Relations, REST Resources Many to many Relations, REST Resources Relations expansion, REST Resources Actions, REST API Versioning, REST API Documentation

#### **HTTP**

HTTP Methods, HTTP Status Codes, HTTP Status Codes Success (2xx), HTTP Status Codes Redirection (3xx), HTTP Status Codes Client Error (4xx), HTTP Status Codes Server Error (5xx), Media types

#### **CRUD Operations**

CRUD About, CRUD Create Single item, CRUD Retrieve Single item, CRUD Retrieve Collection, CRUD Update Single item, CRUD Delete Single item

#### **Filtering**

Filtering About, Filtering Using styles, Filtering Using includes, Filtering Using excludes

#### **Sorting**

Sorting About, Sorting Metadata, Sorting Example

#### **Searching**

Searching About, Searching Local search, Searching Scoped search, Searching Global search, Searching Advanced search, Searching Wildcards, Searching Formatting

### Zadanie 7.2:

Przejsć do *Swagger Editor*.

Dla *endpoint'u* pobrania wszystkich aktorów zaprojektować *parametr ciągu zapytania* dotyczący *sortowania* wyników według wielu *kolumn*, określenia porządków: *rosnąco*, *malejąco*. Nazwy kolumn zapisać jako *camelCase*.

<https://github.com/NationalBankBelgium/REST-API-Design-Guide/wiki/Sorting-About>

<https://github.com/NationalBankBelgium/REST-API-Design-Guide/wiki/REST-Resources-Parameters#query-string>

<https://swagger.io/docs/specification/describing-parameters/#query-parameters>

<https://swagger.io/docs/specification/serialization>

The screenshot shows the Swagger Editor interface for the `/actors` endpoint. The endpoint is a GET request that returns all actors. Under the 'Parameters' section, there is a query parameter named `sort` of type `array[string]`. The description for `sort` is 'Define the order in which the items in a response should be returned'. Below the description, a list of available values is shown: `firstName$ASC`, `firstName$DESC`, `lastName$ASC`, and `lastName$DESC`. The `sort` parameter is currently set to `firstName$ASC`.

### Zadanie 7.3:

Przejsć do kontrolera *ActorController*, zaimplementować *sortowanie* (podobnie według poniższego źródła), tak, aby jego funkcjonowanie odbywało się zgodnie z napisaną wcześniej dokumentacją.

<https://laraveldaily.com/tip/sorting-your-api-results>

<https://laravel.com/docs/11.x/strings#method-snake-case>

### Zadanie 7.4:

Poprzez **Try it out** sprawdzić działanie *endpoint'u* dla różnych wartości parametru `sort`.

[http://localhost:8000/api/actors?sort=lastName\\$ASC](http://localhost:8000/api/actors?sort=lastName$ASC)

[http://localhost:8000/api/actors?sort=birthday\\$DESC](http://localhost:8000/api/actors?sort=birthday$DESC)

[http://localhost:8000/api/actors?sort=firstName\\$ASC,lastName\\$ASC](http://localhost:8000/api/actors?sort=firstName$ASC,lastName$ASC)

The screenshot shows the Swagger Editor interface for the `/actors` endpoint. The endpoint is a GET request that returns all actors. Under the 'Parameters' section, there is a query parameter named `sort` of type `array[string]`. The description for `sort` is 'Define the order in which the items in a response should be returned'. Below the description, a list of available values is shown: `firstName$ASC`, `firstName$DESC`, and `lastName$ASC`. The `sort` parameter is currently set to `firstName$ASC,lastName$ASC`. The 'Try it out' button is visible in the top right corner. Below the parameters section, there is an 'Execute' button and a 'Clear' button. The 'Responses' section is also visible, showing the curl command and the request URL.

### Zadanie 7.5:

Jakie ograniczenie występuje obecnie w *sortowaniu*? Zmodyfikować dokumentację w celu usunięcia tego ograniczenia.

GET /actors Returns all actors

Parameters

Cancel

Name	Description
sort	Define the order in which the items in a response should be returned
array[string] (query)	height\$ASC
	lastName\$ASC
	Add string item

Execute Clear

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8000/api/actors?sort=height%24ASC,lastName%24ASC' \
-H 'accept: application/json'
```

Request URL

```
http://localhost:8000/api/actors?sort=height%24ASC,lastName%24ASC
```

### Zadania (Swagger, Laravel, filtrowanie):

#### Zadanie 7.6:

W jaki sposób, w poniższych źródłach, określone są operacje *filtrowania* i *wyszukiwania*?

<https://github.com/NationalBankBelgium/REST-API-Design-Guide/wiki/Filtering-About>  
<https://github.com/NationalBankBelgium/REST-API-Design-Guide/wiki/Searching-Local-search>  
<https://uxpickle.com/search-vs-filter-in-sofware/>

#### Zadanie 7.7:

Dla *endpoint*'u pobierania wszystkich aktorów zaprojektować *parametry ciągu zapytania* dotyczące *filtrowania* wyników według wielu *kolumn*.

*Filtrowanie* ma umożliwiać wybór pożądanych obiektów, według narzuconych wartości w danych kolumnach.

Nazwy *parametrów* zapisać jako *camelCase*.

firstName Filter actors by their first name

string (query) firstName

lastName Filter actors by their last name

string (query) lastName

birthday Filter actors by their birthday

string(\$date) (query) birthday

height Filter actors by their height

integer(\$int32) (query) height

### Zadanie 7.8:

Przejsć do kontrolera *ActorController*, zaimplementować *filtrowanie*, tak, aby jego funkcjonowanie odbywało się zgodnie z napisaną wcześniej dokumentacją.

```
$query->where(...);
```

### Zadanie 7.9:

Poprzez **Try it out** sprawdzić działanie *endpoint'u* dla różnych parametrów dotyczących *filtrowania* oraz różnych wartości.  
Jednocześnie użyć także parametr *sort*.

<http://localhost:8000/api/actors?firstName=Ben>

<http://localhost:8000/api/actors?firstName=Ben&lastName=Whishaw>

[http://localhost:8000/api/actors?sort=height\\$ASC&firstName=Ben](http://localhost:8000/api/actors?sort=height$ASC&firstName=Ben)

[http://localhost:8000/api/actors?sort=height\\$ASC,birthday\\$ASC&firstName=Ben](http://localhost:8000/api/actors?sort=height$ASC,birthday$ASC&firstName=Ben)

#### Curl

```
curl -X 'GET' \
  'http://localhost:8000/api/actors?sort=height%24ASC,birthday%24ASC&firstName=Ben' \
  -H 'accept: application/json'
```

#### Request URL

```
http://localhost:8000/api/actors?sort=height%24ASC,birthday%24ASC&firstName=Ben
```

#### Server response

##### Code

##### Details

200

##### Response body

```
{
  "data": [
    {
      "id": 24,
      "first_name": "Ben",
      "last_name": "Stiller",
      "birthday": "1965-11-30",
      "height": 170
    },
    {
      "id": 21,
      "first_name": "Ben",
      "last_name": "Miller",
      "birthday": "1966-02-24",
      "height": 175
    },
    {
      "id": 18,
      "first_name": "Ben",
      "last_name": "Whishaw",
      "birthday": "1980-10-14",
      "height": 175
    }
  ]
}
```



Download

## Zadania (Swagger, Laravel, paginacja):

### Zadanie 7.10:

Zapoznać się z następującymi zagadnieniami dotyczącymi *paginacji*:

- kiedy i do czego jest potrzebna *paginacja*?,
- *paginacja* typu „*limit*” i „*offset*”,
- *metadane paginacji*,
- gotowa implementacja *paginacji* w *framework'u Laravel*.

<https://github.com/NationalBankBelgium/REST-API-Design-Guide/wiki/Pagination-About>  
<https://github.com/NationalBankBelgium/REST-API-Design-Guide/wiki/Pagination-Rules-and-metadata>  
<https://github.com/NationalBankBelgium/REST-API-Design-Guide/wiki/Pagination-Example>

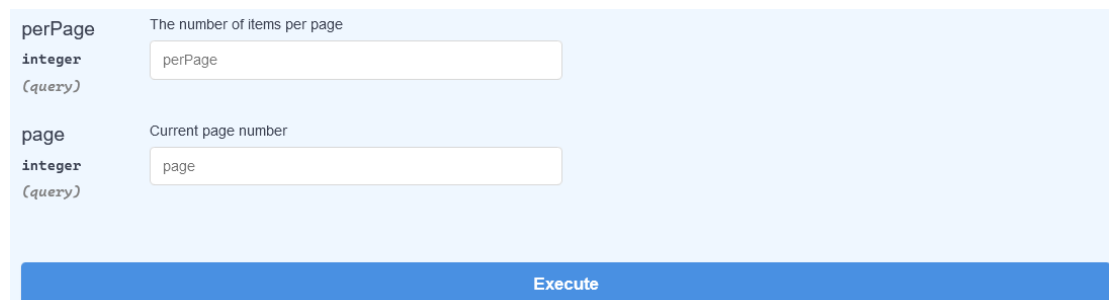
<https://laravel.com/docs/11.x/pagination>  
<https://laravel.com/docs/11.x/pagination#paginating-query-builder-results>  
<https://laravel.com/docs/11.x/pagination#simple-pagination>  
<https://laravel.com/docs/11.x/pagination#appending-query-string-values>  
<https://laravel.com/docs/11.x/pagination#converting-results-to-json>

### Zadanie 7.11:

Dla *endpoint'u* pobierania wszystkich aktorów zaprojektować *parametry ciągu zapytania* dotyczące *paginacji*.

Należy ustalić podział na „*strony*” o danej liczbie obiektów (np. 10) oraz wybór konkretnej strony (według jej numeru 1 – ...).

Nazwy *parametrów* zapisać jako *camelCase*.



The screenshot shows a Swagger UI form for pagination parameters. It has two input fields: 'perPage' with the description 'The number of items per page' and 'page' with the description 'Current page number'. Both fields are labeled as 'integer (query)'. Below the fields is a blue 'Execute' button.

### Zadanie 7.12:

Przejsć do kontrolera *ActorController*, ustawić *paginację*, tak, aby jej funkcjonowanie odbywało się zgodnie z napisaną wcześniej dokumentacją.

```
$actors = $request->filled('perPage') ?  
    $query->simplePaginate($request->input('perPage'))->withQueryString() : $query->get();  
return new ActorCollection($actors);
```

### Zadanie 7.13:

Poprzez ***Try it out*** sprawdzić działanie *endpoint'u* dla parametrów dotyczących *paginacji*. Jednocześnie użyć także parametr *sort* oraz parametry *filtrowania*.

<http://localhost:8000/api/actors?perPage=10>  
<http://localhost:8000/api/actors?perPage=10&page=2>  
[http://localhost:8000/api/actors?sort=height\\$ASC&perPage=10&page=1](http://localhost:8000/api/actors?sort=height$ASC&perPage=10&page=1)  
[http://localhost:8000/api/actors?sort=height\\$ASC&firstName=Ben&perPage=2&page=1](http://localhost:8000/api/actors?sort=height$ASC&firstName=Ben&perPage=2&page=1)

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8000/api/actors?firstName=Ben&perPage=2&page=1' \
-H 'accept: application/json'
```

Request URL

```
http://localhost:8000/api/actors?firstName=Ben&perPage=2&page=1
```

Server response

Code	Details
200	<div><div>Response body</div><div><pre>{   "data": [     {       "id": 18,       "first_name": "Ben",       "last_name": "Whishaw",       "birthday": "1980-10-14",       "height": 175     },     {       "id": 21,       "first_name": "Ben",       "last_name": "Miller",       "birthday": "1966-02-24",       "height": 175     }   ],   "links": {     "first": "http://localhost:8000/api/actors?firstName=Ben&amp;perPage=2&amp;page=1",     "last": null,     "prev": null,     "next": "http://localhost:8000/api/actors?firstName=Ben&amp;perPage=2&amp;page=2"   } }</pre></div><div><div>Download</div></div></div>

- ✳ – zadania/podpunkty do samodzielnego dokończenia/wykonania,
- ✳ – zadania/podpunkty dla zainteresowanych.

Po zakończonym laboratorium należy skasować wszystkie pobrane oraz utworzone przez siebie pliki z komputera w sali laboratoryjnej.

Wersja pliku: v1.0

Inne: ✳