Aplikacje internetowe 1 (AI1) – laboratorium nr 11

Laravel, REST API – realizacja kontrolera CRUD (cz. 1)

Początek laboratorium:

• pobrać na pulpit archiwum `Lab011_AI1_start.zip`, w którym umieszczony jest projekt startowy do wykonania zadań oraz rozpakować to archiwum,

• 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* lub rozszerzenia do *VSCode* „*DevDb*",

• do zadań wykorzystać *Postman'a* lub zainstalować rozszerzenie `humao.rest-client`

https://marketplace.visualstudio.com/items?itemName=humao.rest-client

Zadania (*Laravel*):

Zadanie 11.1:
Wyjaśnić zagadnienia z ostatniego zadania z poprzedniego laboratorium.
Zapoznać się z już wprowadzonymi modyfikacjami:
• wyłączenie *sesji*, *ciasteczek* oraz *tokenów CSRF* (w migracji *..._create_users_table.php* oraz *boostrap/app.php*),
• dodanie pliku *api.php*, w którym będą ustalane *ścieżki/endpoint'y API*,
• usunięcie *Laravel Sanctum* (na tym laboratorium nie potrzebne jest uwierzytelnianie, a na Lab013 będzie *JWT*),
• utworzenie *middleware'u* dodającego nagłówek *Accept: 'application/json'* do każdego żądania, co zapobiega zwracaniu widoków np. gdy wystąpi błąd.

(występują różnice pomiędzy Laravel'em 10.x a 12.x i nowszymi)

https://dev.to/grantholle/exploring-middleware-in-laravel-11-2e10
https://ma.ttias.be/disable-http-sessions-in-laravel-to-speed-up-your-api
https://stackoverflow.com/questions/78352481/prevent-laravel-11-from-setting-any-cookies-including-session-xsrf-cookies

https://laravel.com/docs/12.x/routing#api-routes
```
php artisan install:api
```

usunąć config/sanctum.php i migrację ..._personal_access_tokens_table_...
```
composer remove laravel/sanctum
```

https://dev.to/arxeiss/force-json-response-on-all-api-routes-in-laravel-29h
```
php artisan make:middleware ForceJsonResponse
```

(powyższe operacje/komendy zostały już wykonane)

Zadanie 11.2:
Otworzyć terminal *cmd (Command Prompt)* w *VSCode*.
Uruchomić serwer deweloperski *php* dla przy użyciu komendy `serve` *artisan'a*.

```
php artisan serve
```

Zadanie 11.3:
Otworzyć drugą kartę terminala *cmd* (*Command Prompt*) w *VSCode*.
Wykonać poniższe polecenie w celu wygenerowania nowego kontrolera z obecnymi funkcjami do przeprowadzania operacji CRUD na „*zasobie*", który nie będzie zawierał dwóch niepotrzebnych funkcji (wyjaśnić dlaczego są one niepotrzebne *).

https://laravel.com/docs/12.x/controllers#resource-controllers
https://laravel.com/docs/12.x/controllers#api-resource-routes

```
php artisan make:controller CountryController --api --resource
```

Zadanie 11.4:
Ustawić trasowanie dla kontrolera *CountryController* (pamiętając, że jest to *API* kontroler), odtąd będzie to w pliku *routes/api.php*.
Wyświetlić obecnie skonfigurowane trasowanie.

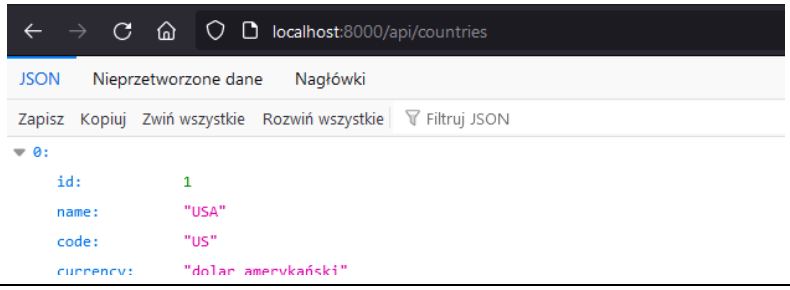
https://laravel.com/docs/12.x/controllers#api-resource-routes

```
Route::apiResource('countries', CountryController::class);
```

```
GET|HEAD        api/countries ....................... countries.index › CountryController@index
POST            api/countries ....................... countries.store › CountryController@store
GET|HEAD        api/countries/{country} ............... countries.show › CountryController@show
PUT|PATCH       api/countries/{country} ............ countries.update › CountryController@update
DELETE          api/countries/{country} .......... countries.destroy › CountryController@destroy
```

Zadanie 11.5:
Uzupełnić funkcję *index* kontrolera *CountryController* o zwracanie wszystkich krajów. Sprawdzić rezultat jej działania poprzez:
• przeglądarkę opartą o *Chromium*,
• przeglądarkę *Firefox*,
• wykonanie żądania *GET* w narzędziu *Postman*,
• plik *requesty.rest* do używania poprzez rozszerzenie *REST Client* w *VSCode* (kliknąć *Send Request*) lub samego *PhpStorm'a*.

```
public function index()
{
    $countries = DB::table('countries')->get();
    return $countries; // 200
}
```



Zadanie 11.6:

W następnym zadaniach przystąpić do realizacji funkcji *CountryController'a* według następujących założeń:

• formatem wymiany danych jest JSON: zwracany w *ciele odpowiedzi* (*Response body*), odczytywany z *ciała żądania* (*Request body*),

• dostępny jest *walidator* (*Validator*), który na podstawie ustalonych reguł walidacji określa poprawność obiektu: *Tak* lub *Nie* (*fails*) z informacją o błędach (*errors*),

• nieużywanie *...Resource*, *...Collection*,

• nieużywanie *...StoreRequest*, *...UpdateRequest*,

• nieużywanie *model binding'u* obiektu `$country` jako parametru funkcji, tylko `int $id`,

• nieużywanie funkcji *Eloquent'a* na modelu *Country*, tylko operacje na fasadzie `DB::...`,

Powyżej wymienione elementy będą wykorzystane na następnym laboratorium.

Na razie należy zaprogramować kontroler „*manualnie*" żeby wiedzieć jakie operacje w jakich sytuacjach mają zwracać odpowiednie kody odpowiedzi.

Nieużywanie:
https://laravel.com/docs/12.x/eloquent-resources
https://laravel.com/docs/12.x/routing#route-model-binding
https://laravel.com/docs/12.x/validation#creating-form-requests
https://laravel.com/docs/12.x/eloquent#refreshing-models

Zadanie 11.7:

Przystąpić do realizacji funkcji *CountryController'a* tak, aby były zaprogramowane według wcześniej ustalonych założeń oraz dokładnie według postaci opisanej poniżej (obsłużone sytuacje „*pozytywne*", zakończone sukcesem).

Następnie wykonać przykładowe *żądania HTTP* sprawdzające działanie tych funkcji. Ponadto spróbować dodać do bazy kraj z powierzchnią *null*.

```
CountryController:

        index():
                Pobrać wszystkie kraje z bazy danych.
                Zwrócić je jako tablicę JSON z obiektami i status 200.

        show(id_kraju):
                Pobrać kraj o id_kraju z bazy danych.
                Zwrócić ten kraj jako obiekt JSON i status 200.

        store(żądanie):
                Odczytać dane z ciała żądania (z JSONa) tzn. właściwości obiektu i ich wartości.
                Korzystając z danych dodać nowy kraj do bazy i uzyskać jego nowe id.
                Pobrać nowy kraj po id.
                Zwrócić ten kraj jako obiekt JSON i status 201.

        update(żądanie, id_kraju)
                Odczytać dane z ciała żądania (z JSONa) tzn. właściwości obiektu i ich wartości.
                Zaktualizować kraj o tym id_kraju danymi z ciała żądania.
                Pobrać kraj o tym id_kraju z bazy.
                Zwrócić ten kraj jako obiekt JSON i status 200.

        destroy(id_kraju):
                Usunąć kraj o tym id_kraju z bazy.
                Zwrócić status 204.
```

https://laravel.com/docs/12.x/queries#retrieving-all-rows-from-a-table
https://laravel.com/docs/12.x/queries#retrieving-a-single-row-column-from-a-table
https://laravel.com/docs/12.x/queries#auto-incrementing-ids
https://laravel.com/docs/12.x/queries#update-statements
https://laravel.com/docs/12.x/queries#delete-statements

https://laravel.com/docs/12.x/responses#json-responses

```
$inputs = [
        'name' => $request->input('name'),
        'code' => $request->input('code'),
        'currency' => $request->input('currency'),
        'area' => $request->input('area'),
        'language' => $request->input('language')
];

return response()->json($country, 201);

return response()->json(null, 204);
```

Zadanie 11.8:

Zmodyfikować istniejące funkcje *CountryController'a* tak, aby były zaprogramowane według wcześniej ustalonych założeń oraz poszerzone według postaci opisanej poniżej (dochodzi uwzględnienie sytuacji „*niepozytywnych*", zakończonych brakiem sukcesu, np. *nieodnalezienie obiektu*, *błąd walidacji*).
Następnie wykonać przykładowe *żądania HTTP* sprawdzające działanie tych funkcji, pod kątem sytuacji „*niepozytywnych*".

```
CountryController:

        index():
                Pobrać wszystkie kraje z bazy danych.
                Zwrócić je jako tablicę JSON z obiektami i status 200.

        show(id_kraju):
                Pobrać kraj o id_kraju z bazy danych, w celu sprawdzenia czy istnieje.
                Gdy nie istnieje:
                        Zwrócić status 404, oznajmujący o nieodnalezieniu tego kraju, czyli brak
                        możliwości jego zwrócenia.
                Gdy istnieje:
                        Zwrócić ten kraj jako obiekt JSON i status 200.

        store(żądanie):
                Odczytać dane z ciała żądania (z JSONa) tzn. właściwości obiektu i ich wartości.
                Ustalić reguły walidacji.
                Zwalidować dane.
                Gdy dane niepoprawne:
                        Zwrócić informację co było niepoprawne i status 400/422.
                Gdy dane poprawne:
                        Korzystając z danych dodać nowy kraj do bazy i uzyskać jego nowe id.
                        Pobrać nowy kraj po id.
                        Zwrócić ten kraj jako obiekt JSON i status 201.

        update(żądanie, id_kraju)
                Pobrać kraj o id_kraju z bazy danych, w celu sprawdzenia czy istnieje.
                Gdy nie istnieje:
                        Zwrócić status 404, oznajmujący o nieodnalezieniu tego kraju, czyli brak
                        możliwości jego aktualizacji.
                Gdy istnieje:
                        Odczytać dane z ciała żądania (z JSONa) tzn. właściwości obiektu i ….
                        Ustalić reguły walidacji.
                        Zwalidować dane.
                        Gdy dane niepoprawne:
                                Zwrócić informację co było niepoprawne i status 400/422.
                        Gdy dane poprawne:
                                Zaktualizować kraj o tym id_kraju danymi z ciała żądania.
                                Pobrać kraj o tym id_kraju z bazy.
                                Zwrócić ten kraj jako obiekt JSON i status 200.

        destroy(id_kraju):
                Pobrać kraj o id_kraju z bazy danych, w celu sprawdzenia czy istnieje.
                Gdy nie istnieje:
                        Zwrócić status 404, oznajmujący o nieodnalezieniu tego kraju, czyli brak
                        możliwości jego usunięcia.
                Gdy istnieje:
                        Usunąć kraj o tym id_kraju z bazy.
                        Zwrócić status 204.
```

https://laravel.com/docs/12.x/validation#manually-creating-validators

```
$rules = [
    'name' => 'required|string|unique:countries,name,|max:50',
    'code' => 'required|string|unique:countries,code,|max:3',
    'currency' => 'required|string|max:30',
    'area' => 'required|integer|min:0',
    'language' => 'required|string|max:50',
];

use Illuminate\Support\Facades\Validator;

$validator = Validator::make($inputs, $rules);
if($validator->fails()) {
        return response($validator->errors(), 422); // lub 400
}

$rules = [
   'name' => 'required|unique:countries,name,'.$id.'|max:50',
   'code' => 'required|string|unique:countries,code,'.$id.'|max:3',
   'currency' => 'required|string|max:30',
   'area' => 'required|integer|min:0',
   'language' => 'required|string|max:50',
];
```

Zadanie 11.9: *

Zmodyfikować istniejące funkcje *CountryController'a* tak, aby były zaprogramowane według wcześniej ustalonych założeń oraz poszerzone według postaci opisanej poniżej (dochodzi zastąpienie zwracania obiektów *bazodanowych* na rzecz *przekształconych* obiektów np. z tylko *potrzebnymi polami*, *w odpowiednim formacie*). Wykluczyć ze zwracanych obiektów:

`"created_at": null,`
`"updated_at": null`

Następnie wykonać przykładowe *żądania HTTP* sprawdzające działanie tych funkcji, pod kątem nowej postaci *JSON'ów*.

```
CountryController:

        index():
                Pobrać wszystkie kraje z bazy danych i umieścić je w kolekcji.
                Utworzyć nową kolekcję, do której dodać kolejno przekształcone kraje.
                Zwrócić tą kolekcję jako tablicę JSON z obiektami i status 200.

        show(id_kraju):
                Pobrać kraj o id_kraju z bazy danych, w celu sprawdzenia czy istnieje.
                Gdy nie istnieje:
                        Zwrócić status 404, oznajmujący o nieodnalezieniu tego kraju, czyli brak
                        możliwości jego zwrócenia.
                Gdy istnieje:
                        Przekształcić kraj z postaci bazodanowej na postać przekształconą.
                        Zwrócić ten przekształcony kraj jako obiekt JSON i status 200.

        store(żądanie):
                Odczytać dane z ciała żądania (z JSONa) tzn. właściwości obiektu i ich wartości.
                Ustalić reguły walidacji.
                Zwalidować dane.
                Gdy dane niepoprawne:
                        Zwrócić informację co było niepoprawne i status 400/422.
                Gdy dane poprawne:
                        Korzystając z danych dodać nowy kraj do bazy i uzyskać jego nowe id.
                        Pobrać nowy kraj po id.
                        Przekształcić kraj z postaci bazodanowej na postać przekształconą.
                        Zwrócić ten przekształcony kraj jako obiekt JSON i status 201.

        update(żądanie, id_kraju)
                Pobrać kraj o id_kraju z bazy danych, w celu sprawdzenia czy istnieje.
                Gdy nie istnieje:
                        Zwrócić status 404, oznajmujący o nieodnalezieniu tego kraju, czyli brak
                        możliwości jego aktualizacji.
                Gdy istnieje:
                        Odczytać dane z ciała żądania (z JSONa) tzn. właściwości obiektu i ….
                        Ustalić reguły walidacji.
                        Zwalidować dane.
```

```
                        Gdy dane niepoprawne:
                                Zwrócić informację co było niepoprawne i status 400/422.
                        Gdy dane poprawne:
                                Zaktualizować kraj o tym id_kraju danymi z ciała żądania.
                                Pobrać kraj o tym id_kraju z bazy.
                                Przekształcić kraj z postaci bazo… na postać przekształconą.
                                Zwrócić ten przekształcony kraj jako obiekt JSON i status 200.

public function index()
{
        $countries = DB::table('countries')->get();
        $countriesCollection = [];
        foreach ($countries as $country) {
                $countriesCollection[] = [
                        'id' => $country->id,
                        'code' => $country->code,
                        'currency' => $country->currency,
                        'area' => $country->area,
                        'language' => $country->language,
                ];
        }

        return $countriesCollection; // 200
}

$countryResource = [
        'id' => $country->id,
        'code' => $country->code,
        'currency' => $country->currency,
        'area' => $country->area,
        'language' => $country->language,
];

return $countryResource; // 200
```

W innych *framework'ach* byłoby to jak zwracanie *DTO*:
https://learn.microsoft.com/en-us/aspnet/web-api/overview/data/using-web-api-with-entity-framework/part-5
https://nullpointerexception.pl/mapowanie-obiektow-w-aplikacji-spring

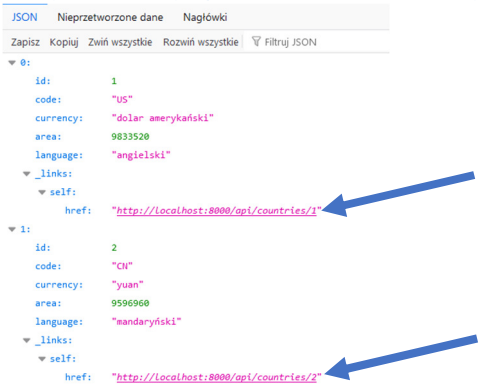
W *Laravel'u* nie ma wbudowanej obsługi *DTO*, ale w zastępstwie są *…Resource* i *…Collection* (następne laboratorium).
https://wendelladriel.com/blog/data-transfer-objects-in-laravel-why-and-how

Zadania (*Laravel, cd.*):

Zadanie 11.10: *

Zapoznać się z *3 poziomem dojrzałości API – Hypermedia Controls, HATEOAS.*
Jako prosty przykład zmodyfikować **`$countriesCollection`**, tak aby każdy kraj posiadał odnośnik do siebie:

https://web.archive.org/web/20211127044624/https://jakzostacprogramista.net/2020/11/25/rest-api-model-dojrzalosci-richardsona



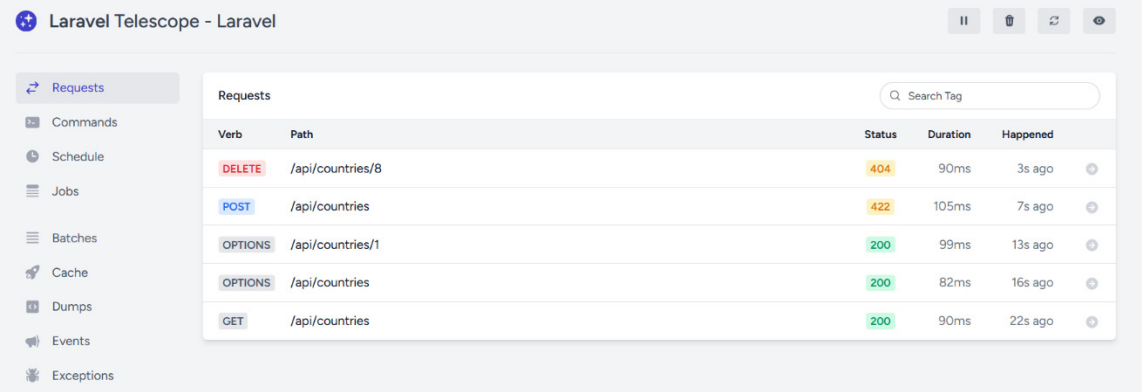
Zadanie 11.11: *

Zainstalować pakiet *Telescope*, oraz wykonać migrację dla tabel potrzebnych do przechowywania danych *Telescope'a*.
Wykonać kilka dowolnych żądań (do *api/countries/...*).
Przejść pod poniższy adres. Zapoznać się z zakładkami i ich zawartością:
• *Requests*,
• *Commands*,
• *Exceptions*,
• *Models*,
• *Queries*.

https://laravel.com/docs/12.x/telescope

```
composer require laravel/telescope
php artisan telescope:install
php artisan migrate
```

http://localhost:8000/telescope



* – 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