Aplikacje internetowe 1 (AI1) – laboratorium nr 12

Laravel, REST API – realizacja kontrolera CRUD (cz. 2)

Początek laboratorium:

• w *XAMPP* uruchomić *Apache* oraz *MySQL*, następnie przejść do *phpMyAdmin*,

• pobrać na pulpit archiwum `Lab012_AI1_start.zip`, w którym umieszczony jest projekt startowy do wykonania zadań oraz rozpakować to archiwum,

• przejść 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*),

• zainstalować program *Postman* oraz pominąć zakładanie konta: *lightweight API client*,

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

Zadania (*Laravel*):

Zadanie 12.1: Zapoznać się z następującymi zagadnieniami: • *Eloquent: API Resources* (*…Resource*, *…Collection*), • *Route Model Binding*, • *Form Request Validation* (reguły walidacji pól z *JSON'ów* tak jak wcześniej z formularzy), • *Refreshing Models*, • *Working With Validated Input*.
https://laravel.com/docs/11.x/eloquent-resources https://laravel.com/docs/11.x/routing#route-model-binding https://laravel.com/docs/11.x/validation#creating-form-requests https://laravel.com/docs/11.x/eloquent#refreshing-models https://laravel.com/docs/11.x/validation#working-with-validated-input
Zadanie 12.2: Otworzyć terminal *cmd (Command Prompt)* w *VSCode*. Uruchomić serwer deweloperski *php* dla przy użyciu komendy `serve` *artisan'a*. W przeglądarce przejść do *Telescope* oraz pozostawić go otwartym.
`php artisan serve` http://localhost:8000/telescope/requests

Zadanie 12.3:

Otworzyć drugą kartę terminala *cmd* (*Command Prompt*) w *VSCode*.
Wykonać poniższe polecenie w celu wygenerowania *CountryResource*, który pozwoli na przekształcenie *kraju* w postaci bazodanowej na obiekt *JSON'owy*.
Uzupełnić funkcję *toArray* w celu dostosowania postaci zwracanego *JSON'a*: właściwości *created_at* i *updated_at* nie będą występowały.
Użyć tego *resource'a* w funkcjach *CountryController'a*, czyli zastąpić obecne przekształcenia.

https://laravel.com/docs/11.x/eloquent-resources

```
php artisan make:resource CountryResource
```

```
return [
    'id' => $this->id,
    'name' => $this->name,
    'code' => $this->code,
    'currency' => $this->currency,
    'area' => $this->area,
    'language' => $this->language,
    //'created_at' => $this->created_at,
    //'updated_at' => $this->updated_at,
];
```

```
use App\Http\Resources\CountryResource;
```

```
new CountryResource($country)
```

GET http://localhost:8000/api/countries/1

Zadanie 12.4:

Utworzyć *CountryCollection* (*resource* typu *kolekcja*), który pozwoli na przekształcenie kolekcji *krajów* w postaci bazodanowej na *json'ową* tablicę obiektów (uwzględniając postać z poprzedniego zadania).
Użyć tego *resource'a* w funkcji *index*, czyli zastąpić obecne przekształcenia.

https://laravel.com/docs/11.x/eloquent-resources#generating-resource-collections

```
php artisan make:resource CountryCollection
```

https://laravel.com/docs/11.x/eloquent-resources#resource-collections

```
new CountryCollection($countries)
```

GET http://localhost:8000/api/countries

Zadanie 12.5:

Zastąpić obecne sprawdzenia czy dany kraj istnieje za pomocą mechanizmu *Route Model Binding*. Wyjaśnić działanie tego mechanizmu.

https://laravel.com/docs/11.x/routing#route-model-binding

```
public function ...(int $id) → public function ...(Country $country)
```

GET http://localhost:8000/api/countries/100

Zadanie 12.6:
Wygenerować *StoreCountryRequest* oraz *UpdateCountryRequest*, do których przenieść reguły walidacji.
W *authorize* zmienić *false* na *true*.
Następnie usunąć użycie *Validator'a* i zmienić dotychczasowe parametry *Request* na *...CountryRequest*.
Spróbować dodać do bazy nowy kraj z nieprawidłowymi danymi.

https://laravel.com/docs/11.x/validation#creating-form-requests

```
php artisan make:request StoreCountryRequest
php artisan make:request UpdateCountryRequest

$this->country->id

public function ...(Request $request)
↓
public function ...(StoreCountryRequest $request)
public function ...(UpdateCountryRequest $request)
```

POST http://localhost:8000/api/countries

```
{
    "name": "Polska",
}
```

Zadanie 12.7:
Zastąpić odczytywanie danych z pól ciała żądania na uwzględnianie tylko tych, które przeszły *walidację*.

https://laravel.com/docs/11.x/validation#working-with-validated-input

```
$inputs = [...];
↓
$inputs = $request->validated();
```

Zadanie 12.8:
Zastąpić użycie operacji na fasadzie *DB* na użycie operacji *Eloquenta* na modelu *Country*.

https://laravel.com/docs/11.x/validation#working-with-validated-input

```
Country::all()
Country::create($inputs)
```

https://laravel.com/docs/11.x/eloquent#refreshing-models

```
$country->update($inputs);
$country = $country->refresh();

$country->delete();
```

Zadanie 12.9:
„*Skrócić*" kod funkcji *CountryController'a*, tak aby w funkcjach:
• *index*, *store* i *show* była max. 1 linijka kodu,
• *update* i *delete* były max. 2 linijki kodu.
Napisać komentarze w linijkach, gdzie istnieje szansa wystąpienia kodów *HTTP 4XX*.

–

Zadanie 12.10:
Za pomocą programu *Postman* wykonać następujące żądania, o ile nie były wykonane do tej pory:
• pobranie wszystkich krajów,
• pobranie kraju o `id` równym 2,
• pobranie kraju o `id` równym 49,
• dodanie nowego kraju (z podaniem prawidłowych danych),
• dodanie nowego kraju (z podaniem nieprawidłowych danych np. brakującą nazwą, ujemną powierzchnią, pustą nazwą),
• edycja danych kraju (z podaniem prawidłowych danych, istniejącego),
• edycja danych kraju (z podaniem prawidłowych danych, nieistniejącego),
• edycja danych kraju (z podaniem nieprawidłowych danych, analogicznie),
• edycja danych kraju (z podaniem kodu już zajętego przez inny kraj),
• usunięcie kraju (istniejącego),
• usunięcie kraju (nieistniejącego),
• usunięcie kraju, w którym odbywa/ją się wycieczka/i.

Od tej pory należy umieć następujące kwestie:
• jaka *metoda HTTP* będzie wykorzystana,
• nagłówki *Accept* i *content-type* oraz wartość *application/json*,
• pod jaką ścieżkę/URL należy wykonać żądanie,
• co ma zawierać ciało żądania, czy jest wymagane,
• jaki jest oczekiwany status *HTTP* (*2XX*/*4XX*),
• co będzie zawierać ciało odpowiedzi,
• na którym poziomie dojrzałości jest obecne tu *REST API*.

–

Zadania (*Laravel, cd.*):

Zadanie 12.11: *
Wykonać poniższą komendę.
Ustawić *routing* dla nowego kontrolera. ~~Wyczyścić *cache* w razie 404 na */api/trips*.~~
Następnie analogicznie do zadań 12.4 – 12.5 wygenerować i uzupełnić *TripResource* oraz *TripCollection*.
Analogicznie do obecnej postaci *CountryController'a* uzupełnić funkcje *index* oraz *show* w *TripController*.

```
php artisan make:controller TripController --api --resource --model=Trip --requests
```

~~`php artisan route:clear`~~

Zadanie 12.12: *
Uzupełnić *CountryResource* o dołączenie kolekcji wycieczek dla każdego kraju, jeśli są „*załadowane*". Uzupełnić o ładowanie „*wstępne*" powiązanych wycieczek:

```
    //'created_at' => $this->created_at,
    //'updated_at' => $this->updated_at,
    'trips' => TripResource::collection($this->whenLoaded('trips')),
];
```

```
public function index()
{
    return new CountryCollection(Country::with('trips')->get());
}

public function show(Country $country)
{
    return new CountryResource($country->loadMissing('trips'));
}
```

Zadanie 12.13: *

Uzupełnić *StoreTripRequest* oraz *UpdateTripRequest*.
Analogicznie do obecnej postaci *CountryController'a* uzupełnić funkcje *store*, *update* oraz *delete* w *TripController*.
Następnie wykonać przykładowe żądania *POST*, *PUT*, *DELETE* dotyczące wycieczek odnoszące się do powyższych funkcji.

–

Zadanie 12.14: *

Nawiązując do ostatniego podpunktu z zadania 12.10 zaproponować rozwiązanie problemu przy usuwaniu krajów, w których odbywają się wycieczki.
Zwrócić odpowiedni kod odpowiedzi informujący o błędzie wynikającym z winy użytkownika.

–

Zadanie 12.15: *

Zapoznać się z terminem „*paginacja*".
Ustawić *paginację* w funkcji *index* w *TripController*, tak aby zwracała po 3 rekordy w jednej odpowiedzi („na jednej *stronie*").
Sprawdzić działanie mechanizmu, np. przejście na stronę nr 2 (wycieczki o `id` 4-6).

https://laravel.com/docs/11.x/pagination

```
public function index()
{
    return new TripCollection(Trip::paginate(3));
}
```

GET http://localhost:8000/api/trips
GET http://localhost:8000/api/trips?page=2

Zadanie 12.16: *

Wyjaśnić różnicę pomiędzy dwoma sposobami aktualizacji obiektów: *PUT* i *PATCH*. W *Laravel'u* routing dla funkcji *update* jest ustawiony domyślnie dla obu z nich.

–

Zadanie 12.17: *

Dostosować odpowiednio reguły walidacji w *UpdateTripRequest* na potrzeby obsługi tego typu żądań (*PUT* vs *PATCH*) przy aktualizacji danych wycieczki.

https://laravel.com/docs/11.x/validation#validating-when-present

Zadanie 12.18: *
Dodać funkcję *bulkStore* do *TripController'a* realizującą operację typu *batch (bulk) insert* (*zbiorcze wstawianie*), pozwalającej na dodanie do bazy wielu wycieczek przekazanych w ciele żądania *POST* jako tablicy obiektów *JSON*.
Dodać walidację dla zawartości tablicy.
Napisać i wykonać przykładowe *żądanie* dodające dwie nowe wycieczki na raz.

```
Route::post('trips/bulk', [TripController::class, 'bulkStore']);

php artisan make:request BulkStoreTripRequest

https://laravel.com/docs/11.x/validation#validating-nested-array-input

Trip::insert($request->validated());

[
 {
  ...
 },
 {
  ...
 }
]
```

* – 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