

Swagger Editor, Laravel REST API – zasoby zagnieżdżone, relacja o-t-m, płytkie zagnieżdżanie

Początek laboratorium:

- przejść pod adres <https://editor.swagger.io/>,
- usunąć całą zawartość, wkleić zawartość pliku *Hotels API Lab004 start.yaml*,
- zapoznać się z zawartością pliku oraz wygenerowaną zawartością (strona prawa).

Zadania (Swagger Editor):

- blog o projektowaniu REST API: <https://www.mscharhag.com/p/rest-api-design>

Zadanie 4.1:

Zapoznać się z artykułem dotyczącym ścisłej relacji *one-to-many*.

Jako przykład przedstawiono powiązanie pomiędzy *artykułami* i *komentarzami*.

Przedstawiony jest także sposób zarządzania komentarzami będącymi zasobem zagnieżdżonym artykułów.

<https://www.mscharhag.com/api-design/rest-one-to-many-relations>

Tightly coupled relations

Zadanie 4.2:

W kolejnych zadaniach zaprojektować i napisać ścieżki (*endpoint'y*) operacji *CRUD'owych* dla *pokoji hotelowych* (zarządzanie *pokojami* danego *hotelu* z jego poziomu, dany *hotel* ma wiele *pokoji*, dany *pokój* znajduje się tylko w danym *hotelu* – ścisła relacja *o-t-m*):

- projektować API według wymagań 2 poziomu modelu dojrzałości Richardsona,
- stosować dobre praktyki odnośnie projektowania REST API i API internetowych,
- uwzględnić wszystkie elementy jak w zadaniu 3.3 z poprzedniego laboratorium,
- dostosować API pod używanie definicji modeli przesyłanych z/do Laravel'a.

–

Zadanie 4.3:

Zaprojektować i napisać *endpoint* dla operacji pobrania wszystkich pokoi (danego hotelu).

GET

/hotels/{hotelId}/rooms Returns all rooms of the hotel



Zadanie 4.4:

Zaprojektować i napisać ścieżkę (*endpoint*) dla operacji pobrania danego pokoju (z danego hotelu).

Napisać (w *components* → *schemas*) definicję parametru ścieżki *roomId* do wykorzystywania w tym i niektórych następnych zadaniach.

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

Common Parameters for Various Paths

Zadanie 4.5:

Zaprojektować i napisać ścieżkę (*endpoint*) dla operacji utworzenia nowego pokoju (w danym hotelu).

–

Zadanie 4.6:

Zaprojektować i napisać ścieżkę (*endpoint*) dla operacji aktualizacji danych danego pokoju (w danym hotelu).

–

Zadanie 4.7:

Zaprojektować i napisać ścieżkę (*endpoint*) dla operacji usunięcia danego pokoju (z danego hotelu = całkowite).

–

Zadanie 4.8:

Wykonać eksport dokumentacji:

- File → Save as YAML, który zapisać jako *Hotels API Lab004.yaml*,
- File → Convert and save as JSON, który zapisać jako *Hotels API Lab004.json*.

–

Kontynuacja laboratorium:

- pobrać na pulpit archiwum *Lab004_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 (*Laravel nested resources*):

- <https://laravel.com/docs/11.x/eloquent-relationships#one-to-many>
- <https://laravel.com/docs/11.x/controllers#restful-nested-resources>

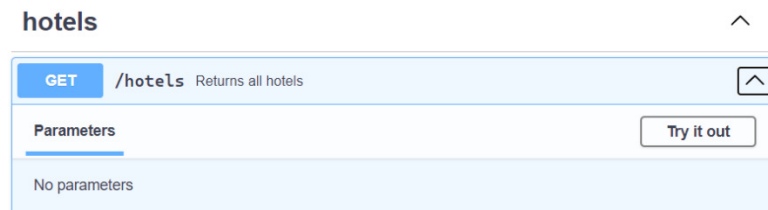
Zadanie 4.9:

Otworzyć terminal *cmd* (*Command Prompt*) w *VSCode*. Następnie uruchomić aplikację.

```
php artisan serve
```

Zadanie 4.10:

Za pomocą **Try it out** endpoint'ów hoteli (bez pokoi) wykonać kilka możliwych rodzajów żądań (z przewidywanym sukcesem, z przewidywanym niepowodzeniem: brak odnalezienia zasobu, błędy walidacji), w tym dodać dwa hotele np. *Mariott* i *Europejski*.



Zadanie 4.11:

Wygenerować kontroler typu API z operacjami CRUD na zasobie dla modelu *Room*, wraz z request'ami.

Utworzyć resource'y *RoomResource* i *RoomCollection*.

Ustawić ścieżki jako *nested resource* (pokoje są zasobem zagnieżdżonym hoteli).

Wyświetlić ścieżki aplikacji.

<https://laravel.com/docs/11.x/controllers#api-resource-routes>

```
php artisan make:controller RoomController --model=Room --resource --requests --api
```

```
php artisan make:resource RoomResource
```

```
php artisan make:resource RoomCollection
```

<https://laravel.com/docs/11.x/controllers#restful-nested-resources>

```
php artisan route:list
```

Zadanie 4.12:

Wykonać następujące etapy programowania kontrolera CRUD pokoi, tak, aby jego działanie pokrywało się 1:1 z napisaną wcześniej dokumentacją w *.yaml*:

- uzupełnić funkcję `toArray()` w *RoomResource.php*,
- uzupełnić funkcję `index()` w *RoomController.php*, tak żeby zwracała pokoje danego hotelu,
- uzupełnić funkcję `show()` w *RoomController.php*, tak żeby zwracała dany pokój danego hotelu,
- uzupełnić reguły walidacji dla żądań w *StoreRoomRequest.php*,
- uzupełnić reguły walidacji dla żądań w *UpdateRoomRequest.php* (te same),
- uzupełnić funkcję `store()` w *RoomController.php*, tak żeby dodawała (do bazy) nowy pokój danego hotelu,
- uzupełnić funkcję `update()` w *RoomController.php*, tak żeby aktualizowała dany pokój danego hotelu,
- uzupełnić funkcję `destroy()` w *RoomController.php*, tak żeby usuwała (całkiem z bazy) dany pokój danego hotelu.

Używać operacje na metodzie `rooms()` modelu *Hotel*.

<https://laravel.com/docs/11.x/eloquent-resources#data-wrapping>

<https://laravel.com/docs/11.x/eloquent-resources#preserving-collection-keys>

```
public $preserveKeys = true;  
public static $wrap = null;
```

```

return [
    'id' => $this->id,
    'number' => $this->number,
    'name' => $this->name,
    'occupation' => $this->occupation,
    'price' => $this->price,
];

return [
    'number' => 'required|integer|min:1|max:999',
    'name' => 'sometimes|nullable|string|min:2|max:30',
    'occupancy' => 'required|integer|min:1|max:4',
    'price' => 'required|decimal:0,2|min:0',
];

```

<https://laravel.com/docs/11.x/eloquent-relationships#the-create-method>

Zadanie 4.13: *

Za pomocą **Try it out** endpoint'ów *pokojów hoteli* wykonać wszystkie możliwe rodzaje żądań (z przewidywanym sukcesem, z przewidywanym niepowodzeniem: brak odnalezienia zasobu, błędy walidacji), w tym:

- dodać nowy pokój hotelu *Mariott*: 232 bez nazwy, 3os., 900zł,
- dodać nowy pokój hotelu *Mariott*: 444 Lounge, 4os., 1200zł,
- pobrać wszystkie pokoje hotelu *Mariott*,
- pobrać pokój hotelu *Mariott* z pierwszej pauzy,
- zaktualizować pokój hotelu *Mariott* z drugiej pauzy, zmieniając cenę na 1500zł,
- pokój hotelu *Mariott* z pierwszej pauzy.

–

Zadanie 4.14: *

Uzupełnić (ręcznie) plik *lab4.rest*, w którym napisać wszystkie możliwe żądania (oraz ich wszystkie możliwe rezultaty) do *Hotels API: hoteli i ich pokojów* (np. inne hotele z Warszawy i pokoje z ich oferty).

Rozszerzenie *REST Client* dla VSCode lub wbudowana obsługa *.rest* w *PhpStorm*.

–

Zadanie 4.15: *

Zaproponować nową funkcję w modelu *Room*, której użyciem zastąpić powtarzający się fragment w funkcjach *show*, *update*, *destroy* w kontrolerze *RoomController*.

–

Zadanie 4.16: *

Ustawić ścieżki dotyczące *pokoji* na *Shallow Nesting*.

Przekształcić *RoomController* dzięki temu, że samo [unikatowe] *id* pokoju pozwala na jego identyfikację do operacji np. pobrania tego pokoju, edycji jego danych, usunięcia go.

<https://laravel.com/docs/11.x/controllers#shallow-nesting>

Zadanie 4.17: *

Zaktualizować dokumentację, aby była dostosowana do nowej postaci *endpoint'ów*, po zmianach w poprzednim zadaniu.

Wykonać eksport dokumentacji, który zapisać jako *Hotels API Lab004 SN.yaml*.

–

- ✱ – 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: *

<https://guides.rubyonrails.org/routing.html#shallow-nesting>

<https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/>