Katedra Systemów Informacyjnych
Uniwersytet Morski w Gdyni

Informatyka, Aplikacje internetowe i mobilne, semestr 5

Projektowanie serwisów internetowych

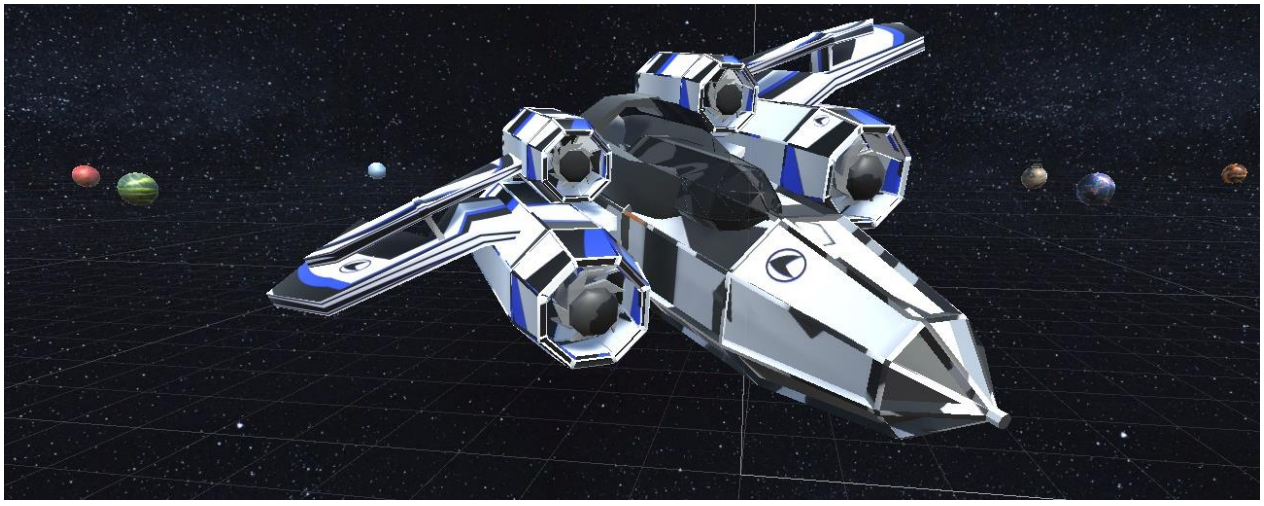
Laboratorium nr 8 i 9

Tworzenie zaawansowanych aplikacji WWW z wykorzystaniem frameworka PHP Laravel

Na poprzednich zajęciach przedstawione zostały podstawy frameworka Laravel. Niniejszy przewodnik stanowi uzupełnienie wiedzy o Laravelu o funkcje wykorzystywane w aplikacjach produkcyjnych. Zostanie to pokazane częściowo na bazie pewnej tajemniczej aplikacji.

1. Wstęp do aplikacji

W 2150 roku podczas eksploracji drogi mlecznej statkiem kosmicznym, w wyniku promieniowania kosmicznego uszkodził się główny komputer podkładowy statku. Nowym zadaniem jest napisanie aplikacji w Laravelu, która umożliwi dalszy bezpieczny lot. Od jej stabilności zależą dalsze losy załogi, a więc w aplikacji nie mogą pojawić się żadne kody błędu zaczynające się od 5! (Na przykład 500).



2. Utworzenie nowego projektu laravel na serwerze foka.

Utwórz w katalogu public_html nowy projekt za pomocą poniższego polecenia composer. Projekt aplikacji będzie dostępny pod nazwę katalogu Space2150. W efekcie końcowym powinna się pojawić odpowiednia struktura Space2150 zawierające odpowiednie katalogi i pliki frameworka Laravel.

composer create-project --prefer-dist laravel/laravel Space2150

Po utworzeniu projektu zadbaj o odpowiednie uprawnienia dla charakterystycznych katalogów niezbędnych do działania aplikacji za pomocą polecenia chmod.

Nie zapomnij także o zadbaniu o bezpieczeństwo katalogu Space2150 tak, aby po wywołaniu adresu: https://foka.umg.edu.pl/~s[nr_albumu]/Space2150 nie było dostępu do listy katalogów i plików.

Katedra Systemów Informacyjnych
Uniwersytet Morski w Gdyni

3. Przygotowanie zasobów bazodanowych oraz konfiguracji połączenia z bazą/schematem w projekcie Laravel

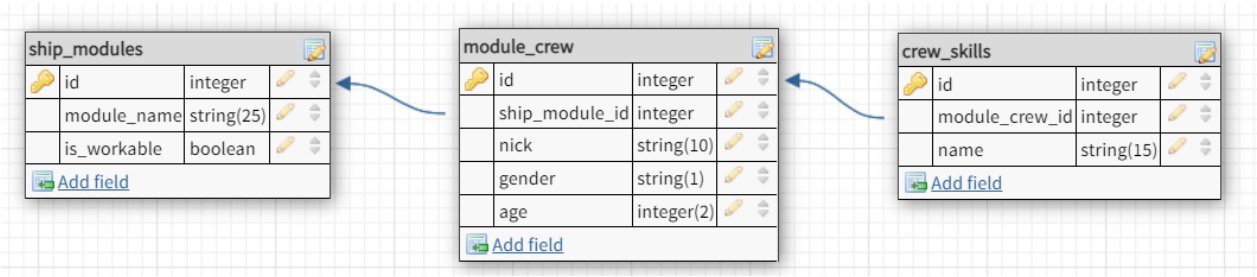
a) Załóżmy, że struktury bazodanowe z danymi będą przechowywane w twoje bazie danych Postgresql na serwerze foka w nowym schemacie o nazwie DBSpace2150. Dlatego przygotuj ten schemat w swojej bazie danych na serwerze foka.

b) w pliku .env nie zapomnij ustawić połączenia z pgsql, odpowiedniego portu, nazwy bazy, użytkownika oraz nowej wartości DBSchema wskazującej na DBSpace2150, zaś w pliku config/database.php dla pgsql nie zapomnij ustawić parametru
'search_path' => env('DB_SCHEMA', 'forge'),

c) W pliku ***.env*** ustaw wartość zmiennej APP_URL na
APP_URL=***https://foka.umg.edu.pl/~s[nr_albumu]/Space2150/public***
Dzięki tej zmienne będzie można odwołać się w kodzie do katalogu głównego aplikacji np. za pomocą: ***<?=config('app.url'); ?>*** co ułatwi bardzo budowanie menu aplikacji oraz odwołania do tras routingu aplikacji.

4. Schemat bazy danych projektu Space2150

Poglądowy schemat relacyjnej bazy danych dla naszej aplikacji zaprojektowany za pomocą narzędzia https://app.dbdesigner.net

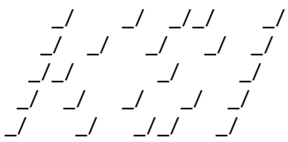


Z powodu wizualnego brak przedstawienia pewnych wymogów poniżej przedstawione te tabele wraz stosownym komentarzem informującym o założeniach i wymaganiach projektowych.

```
ship_modules
- id (auto increment)
- module_name        łańcuch tekstowy, min 3 znaki, max. 25 znaków, wartość unikatowa
- is_workable        boolean

module_crew
- id (auto increment)
- ship_module_id     całkowita liczba dodatnia / klucz obcy (FK) do id z tabeli ship_modules
- nick               pseudonim -> łańcuch tekstowy min. 3 znaki,  max.10 znaków, wartość unikatowa
- gender             1 znak  np.  F - Female, M - Male, N - Non-Binary
- age                całkowita liczba dodatnia (od 18 do 85 lat)

crew_skills
- id (auto increment)
- module_crew_id     całkowita liczba dodatnia / klucz obcy (FK) do id z tabeli module_crew
- name               łańcuch tekstowy, min 3 znaki, max. 15 znaków, wartość unikatowa
```


Katedra Systemów Informacyjnych
Uniwersytet Morski w Gdyni

5. Przygotowanie tabel migracji

Przygotuj tabele migracji dla trzech tabel dla przedstawionego schematu bazy danych w poprzednim punkcie. Wygenerowanie tablicy migracji polega na wydaniu odpowiedniego polecenia php artisan.
Na przykład tabeli migracji dla tabeli ship_modules

php artisan make:migration create_ship_modules_table

Utwórz wstępne tabele migracji dla pozostałych tabel: module_crew, crew_skills
Podczas uzupełniania wygenerowanych definicji tabel migracji zaprojektuj odpowiednie typy danych dla poszczególnych atrybutów/kolumn. Pomoże w tym poniższa dokumentacja

https://laravel.com/docs/10.x/migrations#creating-columns

oraz uwzględnij powiązanie tabel ze sobą za pomocą kluczy obcych

https://laravel.com/docs/10.x/migrations#foreign-key-constraints

Przykład definicji klucza obcego dla tabeli module_crew:

```
$table->integer('ship_module_id')->unsigned();
$table->foreign(ship_module_id')->references('id')->on('ship_modules');
```

Inny przykład definicji pole - łańcuch tekstowy, unikatowy

```
$table->string('module_name')->unique();
```

Uruchomienie migracji – w celu utworzenia tabel wraz z innymi elementami po stronie pgsql

Wykonaj polecenie do tworzenia w bazie danych migracji. Sprawdź, czy pojawiły się w pgsql w schemacie DBSpace2150 odpowiednie tabele bazodanowe.

php artisan migrate

6. Tworzenie modeli z wykorzystaniem Eloquent

Laravel zawiera w sobie potężne narzędzie Eloquent umożliwiające mapowanie obiektowo-relacyjne (ORM), dające ogromne możliwości, przy minimalnym wysiłku ze strony projektanta/programisty.

Podczas korzystania z Eloquent każda tabela bazy danych ma odpowiedni „Model", który jest używany do interakcji z tą tabelą. Oprócz pobierania rekordów z tabeli bazy danych, modele Eloquent umożliwiają również wstawianie, aktualizowanie i usuwanie rekordów z tabeli.

https://laravel.com/docs/10.x/eloquent

W celu jego użycia należy stworzyć modele odpowiadające tworzonym tabelom za pomocą odpowiedniego polecenia

php artisan make:model NazwaModelu

Wszystkie tworzone modele przechowywane są w katalogu App\Models.

Konwencja tworzenia modeli w odwołaniu do wcześniej utworzonych tabel migracji.

Dobrym przyzwyczajeiniem jest nazywanie modeli w powiązaniu do nazwa tabel. Na przykład jeśli tabela nazywa się moja_tabela lub mojatabela do nazwa modelu powinna brzmieć MojaTabela.

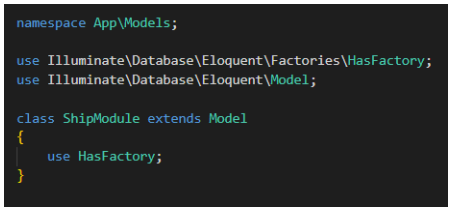
Przykładowo dla tabeli ship_modules odpowiedni model można stworzyć za pomocą

php artisan make:model ShipModules

Nowy plik *ShipModules.php* zostanie stworzony w katalogu *App\Models*. Na początku plik ten zawiera tylko ogólną konstrukcję modelu i model ten nie jest powiązany na starcie z żadną tabelą bazodanową pomimo użytej nazwy samego modelu.

Wiersz zawierający *use HasFactory* wskzuje, że można do przygotować dane testowe dla określonego modelu, a wtedy podczas uruchomienia aplikacji (przy wdrażaniu) będą one dostępne. Na tym etapie można zakomentować użycie HasFactory.

Katedra Systemów Informacyjnych
Uniwersytet Morski w Gdyni



```php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class ShipModule extends Model
{
    use HasFactory;
}
```

Przedstawiony model należy uzupełnić o kilka elementów: stałe opisujące kolumny, nazwa tabeli powiązanej z modelem, określenie klucza głównego (Primary Key), pola, które mogą być wypełniane masowo, Inne metody -> np. metoda za pomocą której można pobierać załogantów danego modułu statku (użycie hasMany).
W celu poinformowania, że podczas używania modelu dla operacji insert, update nie rejestrujemy znaczników czasu tych operacji to należny w modelu zadeklarować publiczną składową timestamps z ustawioną wartością na false: public $timestamps = false;
Poniżej zaproponowano dwa warianty klasy modelu ShipModules na podstawie, których można przygotować w dalszej części laboratorium pozostałe niezbędne modele.

```php
// wariant 1
class ShipModules extends Model
{
// jeśli nie prowadzimy informacji timestamps
// to należy zadeklarować to w modelu
public $timestamps = false;
//Stałe opisujące dostępne kolumny:
public const FIELD_ID = 'id';
public const FIELD_MODULE_NAME= 'module_name';
public const FIELD_IS_WORKABLE = 'is_workable';

//Nazwa tabeli powiązanej z modułem
protected $table = 'ship_modules';

//Klucz główny
protected $primaryKey = self::FIELD_ID;

//Pola, które mogą być wypełniane masowo
protected $fillable = [
        self::FIELD_MODULE_NAME,
        self::FIELD_IS_WORKABLE,
];

//Przy pomocy tej metody będzie można pobierać
// załogantów danego modułu statku
public function moduleCrew()
   {
```

```php
// wariant 2
class ShipModules extends Model
{
// jeśli nie prowadzimy informacji timestamps
// to należy zadeklarować to w modelu
public $timestamps = false; /

//Nazwa tabeli powiązanej z modułem
protected $table = 'ship_modules';

//Klucz główny
protected $primaryKey = 'id';

//Pola, które mogą być wypełniane masowo
protected $fillable = ['module_name','is_workable'];

// dodatkowa metoda odczytująca załogantów
// dla danego  modułu
 public function moduleCrew()
   {
                return $this->hasMany(ModuleCrew::class);
    }

}
```

Wygeneruj modele dla pozostałych tabel. Uzupełnij je o nazwy tabel; klucz podstawowy, pola, które należy wypełniać i ewentualnie o jakieś inne metody.

Informacyjnie – podczas korzystania z modeli np. w kontrolerach warto pamiętać o pewnych metodach z których można skorzystać

Metoda all() - odczytanie wszystkich modułów (wszystkie atrybuty):

$allshipmodules = ShipModules::all();

Lista sprawnych modułów statku (wszystkie atrybuty)

$myShipModules = ShipModules::all()->where('is_workable', 1);

Pobranie tylko id i module_name dla wszystkich modułów statku

$myShipModules = ShipModules::get(['id','module_name']);

Metoda find() - Znalezienie modułu statku o nr id=3

$myshipmodule = ShipModules::find(3);

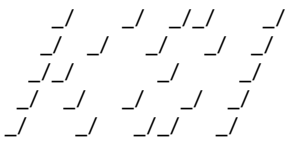
Więcej: https://larainfo.com/blogs/laravel-find-method-example

Inne przykłady użycia metod select + where + get

https://www.itsolutionstuff.com/post/laravel-eloquent-where-query-examplesexample.html

Inne przydatne zaawansowane podzapytania:

https://laravel.com/docs/10.x/eloquent#advanced-subqueries


Katedra Systemów Informacyjnych
Uniwersytet Morski w Gdyni

7. Przygotowanie danych w tabelach (użycie seederów w Laravel)

Wstęp - Przemierzając przestrzeń kosmiczną czasem może przytrafić się anomalia resetująca komputery statku. Mając kod programu można go szybko uruchomić ponownie, ale wypełnienie danych w bazie jest już bardziej czasochłonne. Problem ten może zostać rozwiązany przy użyciu Seederów, wypełniających bazę podstawowymi rekordami. W celu stworzenia Seedera dla ShipModules można wykorzystać polecenie:

php artisan make:seeder ShipModulesSeeder

W katalogu database/seeders zostanie utworzony plik o nazwie ShipModulesSeeder.php

W klasie ShipModulesSeeder uzupełnij metodę run w taki sposób, aby dodać 3 rekordy tzn. działające 2 moduły engines, lights oraz niedziałający moduł air_condition.

Zwróć uwagę, że w metodzie run() użyto klasy modelu ShipModules (aby podać nazwy pól poprzez odczytanie publicznych atrybutów const z klasy ShipModules) oraz klasa DB z metodą insert.

W celu użycia klasy ShipModules oraz DB należy je dodać za pomocą „use” przed definicją klasy ShipModulesSeeder:

```
use Illuminate\Support\Facades\DB;
use App\Models\ShipModules;
```

Dla wariantu 1:

```
class ShipModulesSeeder extends Seeder
{
    public function run()
    {
  DB::table('ship_modules')->insert([
            [ShipModules::FIELD_MODULE_NAME=> 'engines', ShipModules::FIELD_IS_WORKABLE =>true],
            [ShipModules::FIELD_MODULE_NAME=> 'lights', ShipModules::FIELD_IS_WORKABLE =>true],
            [ShipModules::FIELD_MODULE_NAME=> 'air_condition', ShipModules::FIELD_IS_WORKABLE =>false],
        ]);
    }
}
```

Dla wariantu 2:

```
class ShipModulesSeeder extends Seeder
{
    public function run()
    {
     DB::table('ship_modules')->insert([
            ['module_name'=>'engines', 'is_workable' =>true],
            ['module_name'=>'lights', 'is_workable' =>true],
            ['module_name'=>'air_condition', 'is_workable' =>false],
        ]);
    }
}
```

Uruchomienie seedera dla klasy ShipModulesSeeder

Z pomocą poniższego polecenia uruchom Seedera (czyli proces dołożenia rekordów) dla wskazanej klasy ShipModulesSeeder:

php artisan db:seed --class= ShipModulesSeeder

sprawdź z PgAdmin4, czy w tabeli ship_modules pojawiły się powyższe 3 rekordy.

Uwaga:

ponawianie powyższego polecenia seedera dla klasy ShipModulesSeeder powoduje powielanie tych samych rekordów!

Informacja dodatkowa

nie trzeba tych poleceń wykonywać!

do uruchomienia Seederów wykorzystać można różne polecenia:

uruchomienie wszystkich seederów umieszczonych w database/seeders:

php artisan db:seed

odświeżenie (wyczyszczenie) całej bazę/schematu, ponowienie migracji oraz seederów od nowa:

php artisan migrate:fresh –seed

Seederami można również wypełniać bazę losowymi danymi. Więcej informacji na temat seederów znajdziesz na stronie: https://laravel.com/docs/10.x/seeding

W kolejnym kroku należy stworzyć seedery tworzące minimum 3 rekordy dla tabel: module_crew, crew_skills i uruchomić wszystkie seedery (ewentualnie migracje) od nowa (w przypadku powielenia błędnie rekordów oraz niezgodności wartości klucza zewnętrznego wskazującego na klucz podstawowy z innej tabeli). W wyniku prawidłowych działań w katalogu database/seeders powinny pojawić się dwa pliki: ModuleCrewSeeder.php, CrewSkillsSeeder.php zawierające odpowiednie klasy i uzupełnioną metodę run(). Z powodu tego, że pomiędzy tabelami istnieją zależności to uruchamianie seederów za pomocą artisan powinno następować według kolejności: ShipModulesSeeder, ModuleCrewSeeder, CrewSkillsSeeder. Sprawdź w PgAdmin4, czy w schemacie DBSpace2150 pojawiły się 3 rekordy w każdej tabeli.

8. Utworzenie kontrolerów dla poszczególnych modeli

Utworzenie kontrolera dla istniejącego już modelu można uzyskać za pomocą przykładowego polecenia:

```
php artisan make:controller ModelNameController --resource
```

Na przykład utworzenie kontrolera dla modelu ShipModules można uzyskać za pomocą polecenia:

```
php artisan make:controller ShipModulesController  --resource
```

Dokonaj edycji pliku app/Http/Controllers/ShipModulesController.php i zauważ, że w definicji tego kontrolera zostały przygotowane metody index, create, store, show, edit, update, destroy wraz z odpowiednimi komentarzami w jakim celu możemy te metody wykorzystać. Oczywiście jest to pewna propozycja na starcie. Zawsze można dodać własne metody.

Zanim przejdziemy do edycji i tworzenia kontrolerów dla pozostałych modeli warto na potrzeby testowania utworzyć podstawowy kontroler HomeController.

Katedra Systemów Informacyjnych
Uniwersytet Morski w Gdyni

9. Utworzenie podstawowego kontrolera na potrzeby strony startowej / domowej aplikacji Space2150

W celu utworzenia podstawowego kontrolera wykonaj w katalogu Space2150 polecenie:

```
php artisan make:controller HomeController --resource
```

w którym użycie -- resource powoduje wygenerowanie metod: index() itd. podobnie jak dla kontrolera modelu. W metodzie index() HomeController ustaw, że metoda ta zwraca widok home:

```
return view('home');
```

Czynności organizacyjne:
Utwórz w katalogu resource/view widok home.blade.php. Podobnie jak podczas poprzedniego laboratorium warto tak zorganizować widok home tak, aby korzystał z przygotowanych w podkatalogu partials z odpowiednich podwidoków head.blade.php (nagłówek widoku) oraz navi.blade.php (menu nawigacyjne aplikacji uwzględniające dla trzech tabel dostęp do odpowiedniej funkcjonalności (lista rekordów z tabeli, dodawanie nowego rekordu itp.). Natomiast podobnie jak poprzednio wgraj do katalogu public plik styles.css (taki jak na poprzednim laboratorium), czyli arkusz styli z którego możesz skorzystać podczas działania swojej aplikacji.

10. Organizacja logiczna katalogów dla widoków

W celu zbudowania przejrzystej aplikacji warto zaplanować pewną strukturę katalogów zawierające widoki dla konkretnych modeli / klas.
Propozycja - przykładowo można w katalogu resources/view utworzyć podkatalog shipmodules, który będzie zawierał utworzone w kolejnych krokach poniższe widoki z odpowiednim przeznaczeniem:

Nazwa widoku	Plik	Opis przeznaczenia
list	list.blade.php	Widok w formie listy modułów w ujęciu tabelarycznego layoutu
add	add.blade.php	Widok/Nowy formularz umożliwiający dodanie nowego modułu statku
edit	edit.blade.php	Widok/Formularz pozwalający aktualizację modułu statku
show	show.blade.php	Widok pokazujący dane w formularzu /tylko do odczytu/ modułu statku w celu potwierdzenia usunięcia tego egzemplarza
message	message.blade.php	Widok wyświetlający dla przekazanego parametru z kontrolera odpowiedni komunikat błędu lub potwierdzenie poprawnego wykonania czynności.

11. Podstawowy routing aplikacji

Ustaw podstawowy routing w swojej aplikacji
Dokonaj edycji pliku routes/web.php i ustaw, że głównym widokiem zamiast welcome jest teraz widok home.

```
Route::get('/', function () {
    return view('home');
});
```

Katedra Systemów Informacyjnych
Uniwersytet Morski w Gdyni

12. Wyświetlenie listy modułów statku

W celu wyświetlenia listy modułów statku można oprogramować metodę index() w kontrolerze ShipModulesController. Nie będziemy korzystać tak jak poprzednio z klasy DB tylko z klasy Modelu ShipModules oraz metod dostępnych w tej klasie. I tak w celu zwrócenia listy modułów statku w metodzie index() można skorzystać z metody all() dla klasy ShipModules, a następnie wynik zmiennej $myShipModules zwracający wszystkie rekordy z tabeli ship_modules można przekazać jako odpowiedni parametr do nowego widoku shipmodules.list, który trzeba będzie w kolejnym kroku utworzyć.

```php
public function index()
{
    // read all Ship Modules
    $myShipModules = ShipModules::all();
    // return view with ship_modules parameters
    return   view('shipmodules.list', ['ship_modules' => $myShipModules,]);
}
```

Jednak metoda index nie musi być przez ciebie użyta. Zawsze możesz utworzyć metodę list(), a w niej wykonać takie samo wywołanie odpowiedniego widoku z parametrem.

Utworzenie widoku shipmodules/list.blade.php

W katalogu shipmodules utwórz plik o nazwie list.blade.php. W tworzonym nowym widoku list uwzględnij odpowiednie widoki head oraz navi z podkatalogu partials. Wykorzystaj pętlę @foreach do wyświetlenia elementów id, module_name, is_workable dla każdego rekordu z wykorzystaniem przekazanego parametru $ship_modules. W przypadku problemów zajrzyj do zbudowanego już mechanizmu widoku prezentującego listę książek (w aplikacji Laravel_ksiazki).

Utworzenie routingu w celu dostępu do widoku / listy modułów statku

W celu przetestowania działania widoku do pliku routes/web.php dodaj trasę do funkcjonalności / listy modułów statku:

```php
// Ship modules list
Route::get('/shipmodules/list', [ShipModulesController::class,'index']);
```

Przetestowanie działania listy modułów statku:

Przy założeniu, że w menu nawigacyjnym (widok navi) dla linku opisanym jako: „Ship modules" umieszczono adres <a href="***<?=config('app.url'); ?>/shipmodules/list***">Ship modules list przetestuj działanie aplikacji w zakresie zwracania listy modułów statku.

Używanie ***<?=config('app.url'); ?>*** jako głównego URL-a aplikacji bardzo ułatwia zgodność wywołania elementów menu, ale także tras routing do metod danego z kontrolera

Ship modules list

Id	Module name	Is Workable
1	engines	true
2	lights	true

Katedra Systemów Informacyjnych
Uniwersytet Morski w Gdyni

13. Dodanie nowego modułu statku z uwzględnieniem walidacji

Dodanie nowego modułu statku wiąże się np. z wykorzystaniem metody create w kontrolerze ShipModulesController oraz odpowiedniego nowego widoku add.blade.php w katalogu resources/view/shipmodules.

Uzupełnienie metody create w kontrolerze ShipModulesController:

```php
public function create()
{
    // create new form to add ship module
    return   view('shipmodules.add');
}
```

Trasa routing z metodą get dla wywołania metody create

Dopisz w web.php trasę routingu:

```php
Route::get('/shipmodules/add', [ShipModulesController::class,'create']);
```

Dodaj lub uzupełnij w menu nawigacyjnym -> plik navi.blade.php element menu z linkiem wskazującym adres dodania nowego modułu.

Utwórz zawartość pliku add.blade.php podbnie jak poniżej:

```blade
<!DOCTYPE html>
<html>
@include('partials.head')
<body>
  @include('partials.navi')
  <div id="zawartosc">
  <h2>Add ship module</h2>
  <form class="form-inline" action ="<?=config('app.url'); ?>/shipmodules/save" method = "post" >
  @csrf
    <p>
      <label for="module_name">Module Name:</label>
      <input id="module_name" name="module_name" size="25" required>
    </p>
    <p>
      <label for="is_workable">Is Workable:</label>

    <input type="radio" name="is_workable" id="is_workable" value=1 checked  required>
    <label for="is_workable">True</label>
    <input type="radio" name="is_workable" id="is_workable" value=0  required>
    <label for="is_workable">False</label>
    </p>
      <p><button type="submit" class="btn btn-primary mb-2">Save</button></p>
  </form>
  <p>
  @if ($errors->any())
    <div class="alert alert-danger">
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif
</p>
</div>
</body>
</html>
```

Formularz ten posiada akcję APP_URL/shipmodules/save, dla której w web.php należy zdefiniować trasę (dla post) wskazując na metodę store w kontrolerze ShipModulesController.

Dodatkowo pod formularzem umieszczono instrukcję @if, której zadaniem jest odczytanie błędów walidacji, która używana jest w metodzie store(Request $request) kontrolera. Jeśli nie spełnimy wymogów walidacji to nie uda nam się zapisać danych i otrzymamy odpowiedni komunikat wyświetlony pod formularzem.

Dlatego w kolejnym kroku zdefiniuj trasę routingu post /shipmodules/save wskazującą na metodę store:

```
Route::post('/shipmodules/save', [ShipModulesController::class,'store']);
```

Uzupełnij metodę store w kontrolerze ShipModulesController o następującą zawartość:

```
public function store(Request $request)
{

    $validated = $request->validate([
        'module_name' => 'required|min:3|max:25|unique:ship_modules',
        'is_workable' => 'required',
    ]);

    if ($validated)
    {
      // create new ShipModules
      $mod_ship = new ShipModules();
      // prepare data from request
      $mod_ship->module_name = $request->module_name;
      $mod_ship->is_workable = $request->is_workable;
      // save to database
      $mod_ship->save();
        // if OK then return to Ship Modules List
      return  redirect('/shipmodules/list');

    }
}
```

Zauważ, że korzystamy z mechanizmu walidacji dostępnego w klasie Request. Wszystkie reguły które powinny być spełnione rozdzielane są znakiem: „|". Co oznaczają poszczególne zapisy:

- min: 3 – oznacza min 3 znaki,
- max: 25 – oznacza max 25 znaków,
- unique:ship_modules – oznacza unikatowość module_name w obrębie tabeli ship_modules,
- required – wymagane.

Jeśli walidacja jest spełniona (wartość True) to przechodzimy do tworzenia nowego obiektu klasy modelu ShipModules:

$mod_ship = new ShipModules();

Następnie odczytujemy za pomocą $request odpowiednie wartości przekazane z formularza i przypisujemy je odpowiednim atrybutom z klasy ShipModule. Całość zakańczamy zapisem za pomocą metody save(). Po wykonaniu tych czynności aplikacja przekierowuje nas (za pomocą redirect) do listy modułów.

Przetestuj działanie dodawania nowego modułu!

14. Edycja danych modułu statku

W celu przygotowania mechanizmu edycji można w widoku shipmodules/list.blade.php obok kolumny is_workable dołożyć dodatkową kolumnę Edit, a także już przyszłościowo Delete. W kolumnach tych należy podczas działania pętli #foreach budować odpowiednie adresy URL w celu wykonania operacji Edit oraz Del.

W widoku list dla dołożone kolumny edit dodaj odpowiedni adres URL, a także przyszłościowo w kolumnie Del odpowiedni adres URL usuwania modułów. Mile widziany byłby następujący widok:

Ship modules list

Id	Module name	Is Workable	Edit	Del
2	lights	true	Edit	Del
4	power	true	Edit	Del

Katedra Systemów Informacyjnych
Uniwersytet Morski w Gdyni

W celu uzyskania takiego widoku uwzględnij poniższy kawałek kodu HTML/PHP w pliku list.blade.php:

```
<td><a href="<?=config('app.url'); ?>/shipmodules/edit/{{$el->id}}">Edit</a></td>
<td><a href="<?=config('app.url'); ?>/shipmodules/show/{{$el->id}}">Del</a></td>
```

W kontrolerze ShipModulesController uzupełnij kod dla metody edit:

```
public function edit($id)
{
   // find Ship Module by Id
    $myShipModule = ShipModules::find($id);
    // check counter
    if ($myShipModule == null)
    {
        $error_message = "Ship module id=".$id." not find";
    return view('shipmodules.message',['message'=>$error_message,'type_of_message'=>'Error']);
    }
    if ($myShipModule->count() > 0)
    return    view('shipmodules.edit', ['shipmodule' => $myShipModule,]);

}
```

Zauważ, że jeśli nie zostanie znaleziony rekord to zmienna $myShipModule uzyskuje wartość null. Wówczas skorzystaj ze specjalnego widoku shipmodules.message wyświetlającego komunikat z błędem. Oczywiście w celu przetestowania taki widok message.blade.php należy zbudować.

Kolejnym sprawdzaniem metody edit jest sprawdzenie, czy licznik count() z otrzymanego wyniku metody find jest większy od zera, a zatem odnaleziony został rekord – wówczas zwracamy odpowiedni widok shipmodules.edit z przekazaniem parametru shipmodule w celu wykorzystania go w widoku edit.blade.php, który należy podobnie zbudować jak add.blade.php z tą różnicą, że w tym widoku trzeba przypisać odpowiednie wartości elementom HTML korzystając z przekazanego parametru $shipmodule.

Kod fragmentu (nie cały) widoku edit.blade.php może mieć postać:

```
<!DOCTYPE html>
<html>
@include('partials.head')
<body>
  @include('partials.navi')
  <div id="zawartosc">
  <h2>Edit module</h2>
  <form class="form-inline" action ="<?=config('app.url'); ?>/shipmodules/update/{{$shipmodule->id}}" method = "post" >
    @csrf
    <p>
      <label for="module_name">Id:</label>
      <input id="id" name="id" value="{{$shipmodule->id}}" readonly>
    </p>
    <p>
      <label for="module_name">Module Name:</label>
      <input id="module_name" name="module_name" value="{{$shipmodule->module_name}}" size="25" required>
    </p>
    <p>
      <label for="is_workable">Is Workable:</label>

    <input type="radio" name="is_workable" id="is_workable" value=1  @if ($shipmodule->is_workable) checked @endif required>
    <label for="is_workable">True</label>
    <input type="radio" name="is_workable" id="is_workable" value=0  @if (!($shipmodule->is_workable)) checked @endif required>
    <label for="is_workable">False</label>
    </p>
      <p><button type="submit" class="btn btn-primary mb-2">Update</button></p>
  </form>
  <p>
  @if ($errors->any())
```

Zauważ, że w akcji formularza umieszczono APP_URL wskazujący na wykonanie akcji ***update*** dla odpowiedniego id.

Podobnie analizowane są błędy walidacji jak w przypadku dodawania modułu statku.

Zwróć uwagę na skorzystanie z @if w ustalaniu wartości checked dla input type=radio.

Należy jeszcze dopisać odpowiednią trasę get i post do pliku web.php:

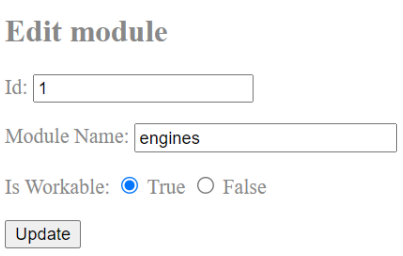
```php
// Edit and update module
Route::get('/shipmodules/edit/{id}', [ShipModulesController::class,'edit']);
Route::post('/shipmodules/update/{id}', [ShipModulesController::class,'update']);
```

Propozycja fragmentu kodu metody update w kontrolerze ShipModulesController:

```php
public function update(Request $request, $id)
{
    // validation
    //

    if ($validated)
    {
      // create new ShipModules
      $mod_ship = ShipModules::find($id);
      // prepare data from request
      if ($mod_ship !=null )
      {
      $mod_ship->module_name = $request->module_name;
      $mod_ship->is_workable = $request->is_workable;
      // save to database
      $mod_ship->save();
      // if OK then return to Ship Modules List
      return  redirect('/shipmodules/list');
      }
      else
      {
        $error_message = "Ship module id=$id not find";
        return view('shipmodules.message',['message'=>$error_message,'type_of_message'=>'Error']);
      }
    }
}
```

Na tym etapie przetestuj działanie edycji. Widok edycji przykładowego modułu:



Po wykonaniu aktualizacji aplikacja powinna przekierować nas do listy modułów

Katedra Systemów Informacyjnych
Uniwersytet Morski w Gdyni

15. Usuwanie danych

W celu usunięcia danych chcielibyśmy najpierw pokazać (show) użytkownikowi widok zawierający częściowe dane (np. id, module_name) jakie chcemy usunąć, a dopiero po podjęciu decyzji zostanie usunięty ten rekord przez metodę destroy w kontrolerze ShopModulesController.

Na potrzeby usuwania będą nam potrzebne odpowiednie trasy routingu get i post

```
// delete module
Route::get('/shipmodules/show/{id}', [ShipModulesController::class,'show']);
Route::post('/shipmodules/delete/{id}', [ShipModulesController::class,'destroy']);
```

Propozycja kodu metody show w kontrolerze ShipModulesController.

```
public function show($id)
{
    // find Ship Module by Id
    $myShipModule = ShipModules::find($id);
    // check counter
    if ($myShipModule == null)
    {
        $error_message = "Ship module id=$id not find";
    return view('shipmodules.message',['message'=>$error_message,'type_of_message'=>'Error']);
    }
    if ($myShipModule->count() > 0)
    return    view('shipmodules.show', ['shipmodule' => $myShipModule,]);

}
```

Opis: Po znalezieniu rekordu jeśli nie jest to wartość null zostanie pokazany widok show.blade.php

Jeśli otrzymamy null to zwracamy komunikat błędu za pomocą znanego już widoku.

Propozycja kodu pokazujące fragmentaryczne dane modułu statku w celu potwierdzenie usunięcia.

```
<!DOCTYPE html>
<html>
@include('partials.head')
<body>
  @include('partials.navi')
  <div id="zawartosc">
   <h2>Confirmation - Delete Id: {{$shipmodule->id}}</h1>
  <form class="form-inline" action ="<?=config('app.url'); ?>/shipmodules/delete/{{$shipmodule->id}}" method = "post" >
      @csrf

    <p>
      <label for="module_name">Id:</label>
      <input id="id" name="id" value="{{$shipmodule->id}}" readonly>
    </p>
    <p>
      <label for="module_name">Module Name:</label>
      <input id="module_name" name="module_name" value="{{$shipmodule->module_name}}" size="25" readonly required>
    </p>
       <p><button type="submit" class="btn btn-primary mb-2">Delete</button></p>
  </form>
</div>
</body>
</html>
```

Zwróć uwagę, że w formularzu tym jest użyta akcja shipmodules/delete/odczytany_id_z_parametru.

Akcja delete zgodnie z routingiem wskazuje na metodę destroy().

Widok show dla wybranego modułu o id=5:

Propozycja metody destroy w kontrolerze ShipModulesController.

```php
public function destroy($id)
{
    // prepare data from request
    $mod_ship = ShipModules::find($id);
      if ($mod_ship != null )
      {
        // delete shipmodule
        $mod_ship->delete();
        return  redirect('/shipmodules/list');
      }
      else {
        $error_message = "Delete Ship module id=$id not find";
        return view('shipmodules.message',['message'=>$error_message,'type_of_message'=>'Error']);
      }
}
```

Zwróć uwagę na metodę delete() służącą do usunięcia egzemplarza/elementu klasy z modelu ShipModules, a tym samym pociąga to za sobą usunięcie także z bazy danych.

Przetestuj działanie metody destroy: Wybierz z menu aplikacji listę modułów, wybierz wskazany moduł do usunięcia. Sprawdź, czy pojawi się odpowiednie potwierdzenie usuwania (z ang. Confirmation). Potwierdź usuwanie!

16. Utworzenie pozostałych kontrolerów i widoków

Należy stworzyć osobne kontrolery dla pozostałych modeli ***CrewSkills, ModuleCrew*** oraz zapewnić podstawowe operacje CRUD podobnie jak dla ShipModules. Nie zapomnij przygotować odpowiednich podkatalogów katalogu view: ***modulecrew*** oraz ***crewskills*** w których będą umieszczone odpowiednie widoki.

Katedra Systemów Informacyjnych
Uniwersytet Morski w Gdyni

17. Dodatkowa funkcjonalność

Wykorzystaj metodę *moduleCrew();* która została zadeklarowana w pierwszym modelu ShipModules w celu dopisania / modyfikacji kodu tak, aby po kliknięciu w nazwę modułu (z poziomu widoku dostępnego w menu „Ship Modules List") pojawiła się lista załogantów przypisanych do obsługi tego modułu.
Można zdefiniować dodatkową metodę w kontrolerze ShipModules, która w celu odnalezienia załogantów odczyta wartości z modułu/klasy ModuleCrew.

```
// find crew by id module
$crew = ModuleCrew::all()->where('ship_module_id',$id);
```

Do tego oczywiście należy przygotować odpowiedni widok w katalogu shipmodules oraz odpowiedni routing tras dla metody get.

W efekcie końcowym chcielibyśmy zobaczyć listę nick'ów załogantów obsługujących/serwisujących moduł o wybranej nazwie (np. lights)

List of crew members servicing the module lights

Id	Nick
2	Cosmo
3	Vera

18. Bezpieczeństwo aplikacji

Kto może dodawać, edytować i usuwać rekordy?
W celu zapewnienia bezpieczeństwa danych podobnie jak na poprzednim laboratorium:

- wykorzystaj pakiet Breeze (wcześniej wykonaj kopię pliku web.php zawierającego zdefiniowane trasy routingu),
- Dołóż do aplikacji opcję zalogowania i wylogowania,
- Dodaj do aplikacji mechanizm, że tylko zalogowany użytkownik o loginie np. tylko admin będzie miał dostęp do funkcjonalności dodawania, edycji, usuwania dla wszystkich 3 tabel w projekcie.

Podsumowanie

To dopiero początek kosmicznej przygody z Laravelem, którego możliwości są wielkie niczym przestrzeń kosmiczna… ☺

Sprawozdanie

Przypominamy, że materiał ten jest przeznaczony na zajęcia laboratoryjne 8 i 9.

Po zajęciach nr 9 w Sprawerze trzeba wysyłać link do swojej aplikacji **Space2150** posadowionej na serwerze foka.