\_/ \_/ \_/\_/\_/\_/ \_/

\_/ \_/ \_/ \_/ \_/ \_/\_/ \_/ \_/

\_/ \_/ \_/ \_/

\_/ \_/\_/\_/\_/\_/ \_/

Katedra Systemów Informacyjnych

Uniwersytet Morski w Gdyni

**Niniejszy plik jest poradnikiem do Laravela, składającym się z czterech części:**

1. Instalacja Laravela na serwerze Foka - strona 2.
2. Krok po kroku stworzenie pierwszej aplikacji w Laravelu na serwerze Foka - strona 6.
3. Opcjonalna konfiguracja środowiska lokalnego - strona 16.
4. Wiedza dopełniająca, do stworzenia bardziej zaawansowanych aplikacji - strona 18.

dr hab. Ewa Ratajczak-Ropel

mgr Ireneusz Meyer

lic. Krzysztof Sadowski

\_/ \_/ \_/\_/\_/\_/ \_/

\_/ \_/ \_/ \_/ \_/ \_/\_/ \_/ \_/

\_/ \_/ \_/ \_/

\_/ \_/\_/\_/\_/\_/ \_/

Katedra Systemów Informacyjnych

Uniwersytet Morski w Gdyni

**Krok 0**

Informacje startowe:

Podczas instalacji Laravela korzystać będziemy ze zdalnego repozytorium. Należy pamiętać, że kod źródłowy tego frameworka może ulegać zmianie, a co za tym idzie wykorzystane w tym przewodniku obrazy/zrzuty ekranów mogą się zdezaktualizować.

$LOGNAME - wartość zmiennej środowiskowej po zalogowaniu się za pomocą SSH.

Sprawdzenie tej wartości zmiennej:

$ echo $LOGNAME

**Krok 1 – Połączenie się z foka za pomocą ssh**

Połącz się z serwerem foka.umg.edu.pl za pomocą połączenia ssh (np. narzędzie putty)

**Krok 2 – Instalacja własnego framework laravel na swoim koncie – na serwerze foka**

Sklonuj najnowszy kod źródłowy Laravela z oficjalnego repozytorium git do swojego system lokalnego. Wystarczy wykonać następujące polecenie, aby to zrobić:

$ cd

***$ cd public\_html***

jeśli nie masz tego folderu, możesz go utworzyć:

$ mkdir public\_html

i powtórz poprzednie polecenie cd

$ git clone <https://github.com/laravel/laravel.git>

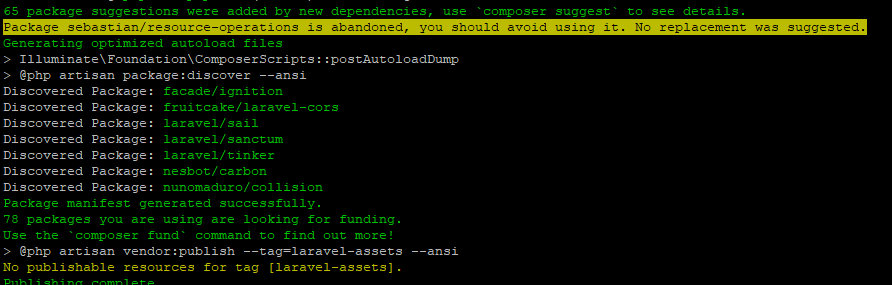
**$ ls –la**



Przejdź do katalogu laravel i korzystając z narzędzia composera przeprowadź instalację laravela

***$ cd laravel***

***$ composer install***



# Krok 3 – Ustawienie uprawień

W celu prawidłowego działania frameworka laravel należy ustawić odpowiednie uprawnienia. W tym celu wykonaj poniższe polecenia

$ cd ..

$ pwd Przykład: /home/s99999/public\_html

$ chmod 755 –R laravel

$ chmod 777 –R laravel/storage

W katalogu laravel skopiuj lub przenieś plik o nazwie .env.example na plik o nazwie .env.

***$ cd /home/$LOGNAME/public\_html/laravel***

Zmiana nazwy: ***$ mv .env.example .env*** lub skopiowanie: ***$ cp .env.example .env***

# Krok 4 – Ustaw Encryption Key

Następnie wygeneruj klucz szyfrowania liczb losowych base64, który jest używany przez usługę szyfrowania Illuminate.

***$ php artisan key:generate***

Application key set successfully.

Dokonaj edycji pliku konfiguracyjnego .env i zaktualizuj wymagane ustawienia. Upewnij się również, że APP\_KEY jest prawidłowo ustawiony$ nano .env

APP\_NAME=Laravel

APP\_ENV=local

APP\_KEY=base64:Wer9JfZHN4qYQt9i8Hn1hLt8LWPeT3urzdI0hVqfzJM=

APP\_DEBUG=true

**APP\_URL=http://foka.umg.edu.pl/~s[nr\_albumu]/laravel**

# Krok 5 – Ustawienia bazy danych PGSQL

Utwórz schemat o nazwie ***laravel*** w swojej bazie pgsql na serwerze foka.umg.edu.pl za pomocą pgadmin4: https://foka.umg.edu.pl/pgadmin4

Następnie w pliku ***.env*** ustaw poniższe wartości:

DB\_CONNECTION=pgsql

DB\_HOST=127.0.0.1 or localhost

DB\_PORT=5432

DB\_DATABASE=s[nr\_albumu] s99999

DB\_SCHEMA=laravel

DB\_USERNAME=s99999 # Twój login na foka

// Uwaga - w oryginalnym pliku nie ma DB\_SCHEMA

**DB\_PASSWORD**=secret\_password – hasło przydzielone na zajęciach

Dokonaj edycji pliku ***config/database.php***i ustaw odpowiednie

'pgsql' => [

'driver' => 'pgsql',

'url' => env('DATABASE\_URL'),

'host' => env('DB\_HOST', '127.0.0.1'),

'port' => env('DB\_PORT', '5432'),

'database' => env('DB\_DATABASE', 'forge'),

'username' > env('DB\_USERNAME', 'forge'),

'password' => env('DB\_PASSWORD', ''),

'charset' => 'utf8',

'prefix' => '',

'prefix\_indexes' => true,

# usuń ten wiersz 'schema' => 'public',

**# Zamiast niego wprowadź poniższy:**

**'schema' => env('DB\_SCHEMA', 'forge'),**

'sslmode' => 'prefer',

],

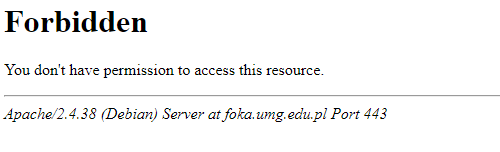
# Krok 6 Bezpieczeństwo frameworka Laravel

W katalogu laravel utwórz plik .htaccess z zawartością:

**Options -Indexes**

Sprawdź teraz adres działanie adresu URL:

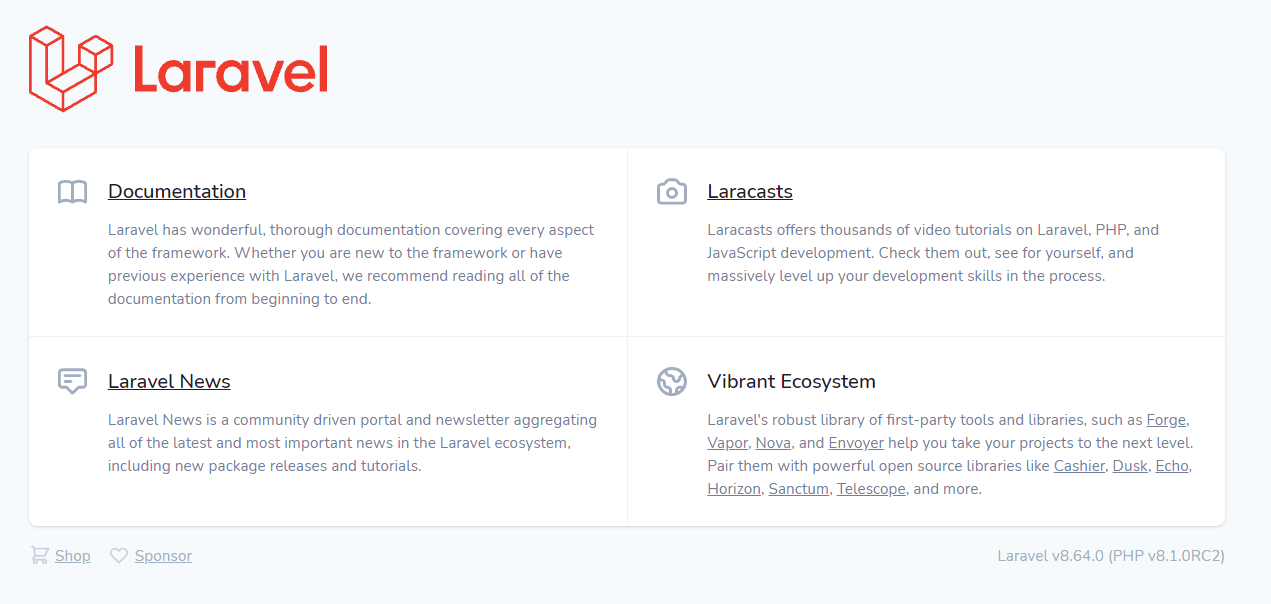
[https://foka.umg.edu.pl/~s\_[nr+albumu]/laravel](https://foka.umg.edu.pl/~s_%5bnr+albumu%5d/laravel)

l

# Krok 7 - Sprawdzenie działania domyślnego widoku „Welcome”

**Otwórz w przeglądarce adres URL:**

**https://foka.umg.edu.pl/~s\_[nr+albumu]/laravel/public/**

****

# Krok 8 – Test komunikacji Laravela z PostgreSQL

Sprawdź za pomocą polecenia ***pwd***, czy bieżący katalog to jest

***/home/$LOGNAME/public\_html/laravel***

Na koniec sprawdź, czy możesz komunikować się z bazą danych PostgreSQL za pośrednictwem Laravel:

***$ php artisan migrate:install***

****

Potem możesz sprawdzić za pomocą PGADMIN4, że w twojej bazie danych PGSQL, a dokładnie w schemacie laravel pojawiał się tabela migration nie zawierająca jeszcze żadnych danych.

\_/ \_/ \_/\_/\_/\_/ \_/

\_/ \_/ \_/ \_/ \_/ \_/\_/ \_/ \_/

\_/ \_/ \_/ \_/

\_/ \_/\_/\_/\_/\_/ \_/

Katedra Systemów Informacyjnych

Uniwersytet Morski w Gdyni

# Krok 11 - Utworzenie pierwszej prostej aplikacji w Laravelu

1. Utwórz pliki z migracjami, dzięki którym Laravel sam stworzy potrzebne tabele w bazie danych. Poprzez SSH wprowadź polecenie *„php artisan make:migration ksiazki”, a następnie „php artisan make:migration kategorie”. Dzięki temu zostaną utworzone pliki* (ścieżka: database-> migrations-> nazwa migracji)*, których treść możesz wypełnić poniższym kodem:*

Następnie użyj polecenia „php artisan migrate”, a później sprawdź w pgadmin4 czy tabele się utworzyły.

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

class Kategorie extends Migration

{

public function up()

{

Schema::create('kategorie', function (Blueprint $table) {

$table->id();

$table->string('opis');

$table->timestamps();

});

}

public function down()

{

Schema::dropIfExists('kategorie');

}

}

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

class Ksiazki extends Migration

{

public function up()

{

Schema::create('ksiazki', function (Blueprint $table) {

$table->id();

$table->string('tytul');

$table->double('id\_kategorii');

$table->timestamps();

});

}

public function down()

{

Schema::dropIfExists('ksiazki');

}

}

1. Stwórz potrzebne widoki w folderze resources->views. Dla fragmentów, które się powtarzają można stworzyć folder partials w views.

Widoki w views->partials:

1. head.blade.php Tutaj możemy od razu podpiąć bootstrapa pod nasz projekt.

<head>

<title>Books example</title>

<meta charset="UTF-8">

<link rel="stylesheet" href="{{ URL::asset('styles.css') }}" />

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</head>

1. navi.blade.php

<nav class="navbar navbar-expand-lg navbar-light bg-light">

<div class="collapse navbar-collapse" id="navbarNavAltMarkup">

<div class="navbar-nav">

<a class="nav-item nav-link active" href="./">Strona domowa<span class="sr-only">(current)</span></a>

<a class="nav-item nav-link" href="./ksiazki">Książki</a>

<a class="nav-item nav-link" href="./dodaj\_kategorie">Dodaj kategorię</a>

<a class="nav-item nav-link" href="./dodaj\_ksiazke">Dodaj książkę</a>

<a class="nav-item nav-link" href="./autoryzacja">Autoryzacja</a>

</div>

</div>

</nav>

Widoki w views:

1. strona\_dodaj\_kategorie.blade.php

<!DOCTYPE html>

<html>

@include('partials.head')

<body>

@include('partials.navi')

<form class="form-inline" action ="./dodaj\_kategorie" method = "post" >

@csrf

<label for="tytul">Opis kategorii</label>

<textarea class="form-control" id="tytul" name="opis" rows="2"></textarea>

<button type="submit" class="btn btn-primary mb-2">Dodaj</button>

</form>

</body>

</html>

1. strona\_dodaj\_ksiazke.blade.php

<!DOCTYPE html>

<html>

@include('partials.head')

<body>

@include('partials.navi')

<form class="form-inline" action ="./dodaj\_ksiazke" method = "post" >

@csrf

<label for="tytul">Tytuł książki</label>

<textarea class="form-control" id="tytul" name="tytul" rows="2"></textarea>

@foreach($kategorie as $kategoria)

<input class="form-check-input" type="radio" name="id\_kategorii" id="id\_kategorii" value={{$kategoria->id}} checked>

<label for="id\_kategorii">{{$kategoria->opis}}</label>

@endforeach

<button type="submit" class="btn btn-primary mb-2">Dodaj</button>

</form>

</body>

</html>

1. strona\_domowa.blade.php

<!DOCTYPE html>

<html>

@include('partials.head')

<body>

@include('partials.navi')

<h1>Hello UMG!</h1>

</body>

</html>

1. strona\_ksiazki.blade.php

<!DOCTYPE html>

<html>

@include('partials.head')

<body>

@include('partials.navi')

<table class="table table-striped">

<thead>

<tr>

<th scope="col">Tytuł</th>

<th scope="col">Kategoria</th>

</tr>

</thead>

<tbody>

@foreach($ksiazki as $ksiazka)

<tr>

<td>{{$ksiazka->tytul}}</td>

<td>{{$ksiazka->opis}}</td>

</tr>

@endforeach

</tbody>

</table>

</body>

</html>

e) wylogowano.blade.php

<!DOCTYPE html>

<html>

@include('partials.head')

<body>

@include('partials.navi')

<h1>Wylogowano.</h1>

</body>

</html>

1. *Stwórz kontroler sterujący ruchem w aplikacji.*

Polecenie: „php artisan make:controller PodstawowyKontroler” utworzy plik w ścieżce app->Http->Controllers. Na górze pliku dodaj „use Illuminate\Support\Facades\DB;”, aby umożliwić mu operacje na bazie danych oraz „use Illuminate\Support\Facades\Auth;”, aby umożliwić operacje na autoryzacji. Następnie dodaj metody sterujące ruchem w aplikacji.

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\DB;

use Illuminate\Support\Facades\Auth;

class PodstawowyKontroler extends Controller

{

public function zwrocStroneDomowa()

{

return view('strona\_domowa');

}

public function zwrocStroneKsiazki()

{

$ksiazkiZBazy = DB::table('ksiazki')->leftJoin('kategorie',

'ksiazki.id\_kategorii', '=',

'kategorie.id') -> get();

return view('strona\_ksiazki', [

'ksiazki' => $ksiazkiZBazy,

]);

}

public function zwrocStroneDodajKategorie()

{

return view('strona\_dodaj\_kategorie');

}

public function dodajKategorie(Request $request)

{

$opisZFormularza = $request->opis;

DB::table('kategorie')->insert([

'opis' => $opisZFormularza,

]);

return redirect('./dodaj\_ksiazke');

}

public function zmienStanAutoryzacji()

{

if(auth()->check()){

$user = auth()->user();

Auth::logout();

return view('wylogowano');

}

else{

return redirect('/register');

}

}

public function zwrocStroneDodajKsiazke()

{

$kategorieZBazy = DB::table('kategorie')-> get();

return view('strona\_dodaj\_ksiazke', [

'kategorie' => $kategorieZBazy,

]);

}

public function dodajKsiazke(Request $request)

{

$tytulZFormularza = $request->tytul;

$idKategoriiZFormularza = $request->id\_kategorii;

DB::table('ksiazki')->insert([

'tytul' => $tytulZFormularza,

'id\_kategorii' => doubleval($idKategoriiZFormularza),

]);

return redirect('/ksiazki');

}

}

1. *Podepnij funkcje kontrolera, pod ścieżki URL w pliku web.php, znajdującym się w folderze routes.*

<?php

use App\Http\Controllers\PodstawowyKontroler;

use Illuminate\Support\Facades\Route;

Route::get('/', [PodstawowyKontroler::class,'zwrocStroneDomowa']);

Route::get('/ksiazki', [PodstawowyKontroler::class,'zwrocStroneKsiazki']);

Route::get('/dodaj\_kategorie', [PodstawowyKontroler::class,'zwrocStroneDodajKategorie'])->middleware('auth');

Route::post('/dodaj\_kategorie', [PodstawowyKontroler::class,'dodajkategorie']);

Route::get('/dodaj\_ksiazke', [PodstawowyKontroler::class,'zwrocStroneDodajKsiazke'])->middleware('auth');

Route::post('/dodaj\_ksiazke', [PodstawowyKontroler::class,'dodajKsiazke']);

Route::get('/autoryzacja', [PodstawowyKontroler::class,'zmienStanAutoryzacji']);

require \_\_DIR\_\_.'/auth.php';

1. Dodaj autoryzację, a następnie edytuj widoki login i register w views->auth

*„composer require laravel/breeze –dev”*

*„php artisan breeze:install”*

*„php artisan migrate”*

<!DOCTYPE html>

<html>

@include('partials.head')

<body>

@include('partials.navi')

<x-guest-layout>

<x-auth-card>

<x-slot name="logo">

</x-slot>

<!-- Session Status -->

<x-auth-session-status class="mb-4" :status="session('status')" />

<!-- Validation Errors -->

<x-auth-validation-errors class="mb-4" :errors="$errors" />

<form method="POST" action="{{ route('login') }}">

@csrf

<!-- Email Address -->

<div>

<x-label for="email" :value="\_\_('Email')" />

<x-input id="email" class="block mt-1 w-full" type="email" name="email" :value="old('email')" required autofocus />

</div>

<!-- Password -->

<div class="mt-4">

<x-label for="password" :value="\_\_('Password')" />

<x-input id="password" class="block mt-1 w-full"

type="password"

name="password"

required autocomplete="current-password" />

</div>

<!-- Remember Me -->

<div class="block mt-4">

<label for="remember\_me" class="inline-flex items-center">

<input id="remember\_me" type="checkbox" class="rounded border-gray-300 text-indigo-600 shadow-sm focus:border-indigo-300 focus:ring focus:ring-indigo-200 focus:ring-opacity-50" name="remember">

<span class="ml-2 text-sm text-gray-600">{{ \_\_('Remember me') }}</span>

</label>

</div>

<div class="flex items-center justify-end mt-4">

@if (Route::has('password.request'))

<a class="underline text-sm text-gray-600 hover:text-gray-900" href="{{ route('password.request') }}">

{{ \_\_('Forgot your password?') }}

</a>

@endif

<x-button class="ml-3">

{{ \_\_('Log in') }}

</x-button>

</div>

</form>

</x-auth-card>

</x-guest-layout>

</tbody>

</table>

</body>

</html>

<!DOCTYPE html>

<html>

@include('partials.head')

<body>

@include('partials.navi')

<x-guest-layout>

<x-auth-card>

<x-slot name="logo">

</x-slot>

<!-- Validation Errors -->

<x-auth-validation-errors class="mb-4" :errors="$errors" />

<form method="POST" action="{{ route('register') }}">

@csrf

<!-- Name -->

<div>

<x-label for="name" :value="\_\_('Name')" />

<x-input id="name" class="block mt-1 w-full" type="text" name="name" :value="old('name')" required autofocus />

</div>

<!-- Email Address -->

<div class="mt-4">

<x-label for="email" :value="\_\_('Email')" />

<x-input id="email" class="block mt-1 w-full" type="email" name="email" :value="old('email')" required />

</div>

<!-- Password -->

<div class="mt-4">

<x-label for="password" :value="\_\_('Password')" />

<x-input id="password" class="block mt-1 w-full"

type="password"

name="password"

required autocomplete="new-password" />

</div>

<!-- Confirm Password -->

<div class="mt-4">

<x-label for="password\_confirmation" :value="\_\_('Confirm Password')" />

<x-input id="password\_confirmation" class="block mt-1 w-full"

type="password"

name="password\_confirmation" required />

</div>

<div class="flex items-center justify-end mt-4">

<a class="underline text-sm text-gray-600 hover:text-gray-900" href="{{ route('login') }}">

{{ \_\_('Already registered?') }}

</a>

<x-button class="ml-4">

{{ \_\_('Register') }}

</x-button>

</div>

</form>

</x-auth-card>

</x-guest-layout>

</tbody>

</table>

</body>

</html>

1. *Utwórz plik ze stylami „styles.css” w folderze public:*

p{

background-color: red;

}

h1{

width: 100%;

text-align: center;

}

\_/ \_/ \_/\_/\_/\_/ \_/

\_/ \_/ \_/ \_/ \_/ \_/\_/ \_/ \_/

\_/ \_/ \_/ \_/

\_/ \_/\_/\_/\_/\_/ \_/

Katedra Systemów Informacyjnych

Uniwersytet Morski w Gdyni

**Instalacja i konfiguracja środowiska**

**(Cały punkt NIE dotyczy komputerów w sali laboratoryjnej!!!):**

1. Instalacja Microsoft Visual C++ Redistributable:

- Pobierz i zainstaluj odpowiadającą Ci wersję, na przykład x64:

<https://docs.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist?view=msvc-160>

<https://windows.php.net/downloads/releases/php-8.0.11-Win32-vs16-x64.zip>

1. Instalacja php w środowisku Windows:

- Na początku stwórz folder na dysku, w którym umieścisz pliki z archiwum PHP

- Pobierz odpowiadającą Ci wersję, na przykład:

<https://windows.php.net/downloads/releases/php-8.0.11-Win32-vs16-x64.zip>

1. Ustawienie zmiennej środowiskowej:

- Skopiuj ścieżkę do folderu z plikami z archiwum PHP

- W wyszukiwarce Windows wpisz frazę „zmienne środowiskowe”, a następnie wybierz wynik „Edytuj zmienne środowiskowe”

- W sekcji „Zmienne użytkownika” wybierz zmienną „Path”, a następnie kliknij edytuj

- W nowym oknie kliknij przycisk „Nowy”

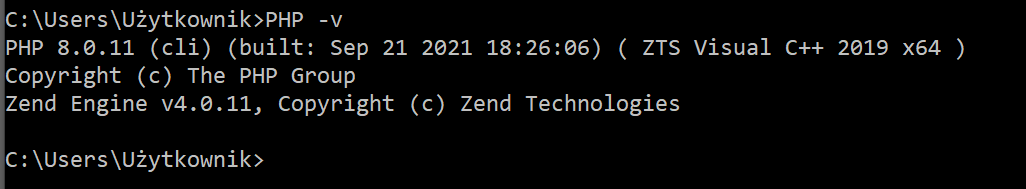
- Wklej skopiowaną wcześniej ścieżkę, a następnie kliknij ok

- Nowe okno powinno się zamknąć

- Kliknij ok w pozostałych dwóch oknach

1. Weryfikacja zmiennej środowiskowej:

– W wyszukiwarce Windows wpisz frazę „cmd”, wybierz wynik „Wiersz poleceń” a po otwarciu nowego okna wpisz „PHP -v”, Efekt powinien być jak poniżej:



1. Konfiguracja PHP:  
   - W folderze z plikami z archiwum PHP odnajdź plik „php.ini-development”, skopiuj go, a nazwę kopi zmień na „php.ini”, a następnie otwórz go poprzez ulubiony edytor tekstowy.

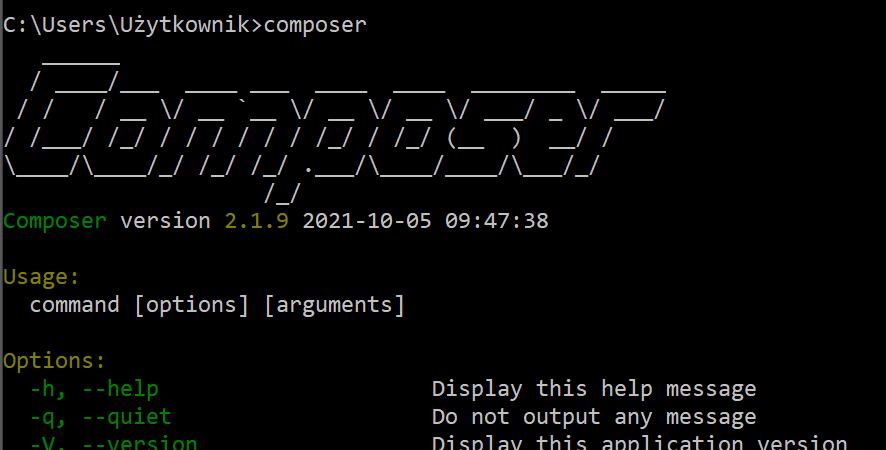
- Możesz aktywować dyrektywy usuwając „ ; ” przed nimi. Aktywuj następujące dyrektywy:

* extension=fileinfo
* extension=pdo\_pgsql
* extension=pgsql

1. Instalacja Composera system zarządzania pakietami Composer.

- Pobierz i zainstaluj: <https://getcomposer.org/Composer-Setup.exe>

- Wykonaj podpunkt „c)” tym razem wpisując frazę „composer” zamiast „PHP -v”. Efekt powinien być jak poniżej:



\_/ \_/ \_/\_/\_/\_/ \_/

\_/ \_/ \_/ \_/ \_/ \_/\_/ \_/ \_/

\_/ \_/ \_/ \_/

\_/ \_/\_/\_/\_/\_/ \_/

Katedra Systemów Informacyjnych

Uniwersytet Morski w Gdyni

**Aplikacja WWW w 10 prostych i szybkich krokach.**

**Do jej zbudowania można wykorzystać dowolne IDE (zalecane są IDE z terminalami, dzięki czemu nie trzeba będzie odnajdywać ścieżki projektu w cmd. Czerwone komentarze powinny być zamienione na własne wartości. Pamiętaj o zachowaniu czarnych znaków interpunkcyjnych.):**

1. Utworzenie projektu:
2. Uruchom terminal cmd, a następnie przejdź przy jego pomocy do folderu, w którym planujesz przechowywać projekt
3. Wprowadź komendę, zmieniając ją zgodnie z oczekiwaną nazwą projektu:

„composer create-project --prefer-dist laravel/laravel <NAZWA\_PROJEKTU>”

1. Aby aktywować serwer wprowadź polecenie "php artisan serve”
2. Uruchom w przeglądarce <http://localhost:8000>
3. Pod podanym adresem znajduje się twoja aplikacja, zmiany w kodzie będą aktualizować się automatycznie
4. Serwer wyłączyć można poprzez wciśnięcie kombinacji przycisków na klawiaturze ctrl+c w oknie cmd
5. Konfiguracja bazy danych w projekcie:
6. W Laravelu bazy danych (celowo w liczbie mnogiej) konfigurować można w pliku „database.php”, w folderze „config” ale bezpieczniejszym rozwiązaniem jest utworzenie osobnego pliku .env dla każdego środowiska, np. inne testowe, a inne produkcyjne.
7. Odnajdź plik „env.example”, a następnie skopiuj go, a kopię nazwij „.env” (Z kropką przed env). W niej zdefiniuj następujące wartości:

DB\_CONNECTION=

DB\_HOST=

DB\_PORT=

DB\_DATABASE=

DB\_USERNAME=

DB\_PASSWORD=

1. Model
2. Tabele możesz zdefiniować poprzez system do zarządzania bazą danych, np. PostgreSQL lub za pomocą mechanizmu migracji w Laravelu. Opisujemy je w kodzie, a następnie poprzez jedną komendę Laravel tworzy tabele automatycznie w bazie. Tabele definiuje się poprzez migrację, aby przykładową migrację stworzyć należy w cmd, w ścieżce projektu wprowadzić komendę:

*„php artisan make:migration <NAZWA TABELI> ”*

Następnie odnajdź plik w projekcie (ścieżka: database-> migrations-> nazwa migracji). Nazwa migracji składać się będzie z daty jej stworzenia oraz nazwy tabeli. W metodzie up można zdefiniować tabelę oraz kolumny do stworzenia, a w metodzie down efekt do cofnięcia. Przykładowy kod:

Wprowadź polecenie „php artisan migrate” a tabele powinny pojawić się w bazie.

Laravel udostępnia wiele możliwości pracy z bazami danych. Wśród nich wyróżnić można fasadę DB i jej poniższe funkcje (wykorzystanie ich zostanie przedstawione w podpunkcie 5.e ):

*public function up()*

*{*

*Schema::create('<NAZWA\_TABELI>', function (Blueprint $table) {*

*$table->id();*

*$table->string('<NAZWA\_KOLUMNY>');*

*$table->double('<NAZWA\_KOLUMNY\_2>');*

*$table->timestamps();*

*});*

*}*

*public function down()*

*{*

*Schema::dropIfExists('<NAZWA\_TABELI>');*

*}*

1. Wczytanie danych:

*$<NAZWA\_ZMIENNEJ> = DB::table('<NAZWA\_TABELI>’)-> get();*

1. Sortowanie danych:

*$<NAZWA\_ZMIENNEJ> = DB::table('<NAZWA\_TABELI>’)->orderBy('<NAZWA\_KOLUMNY>', 'desc')->get();*

1. Łączenie tabel:

*$<NAZWA\_ZMIENNEJ> = DB::table(‘<NAZWA\_TABELI>')*

*->rightJoin('<NAZWA\_TABELI\_2>',*

*‘<NAZWA\_TABELI>.<KOLUMNA\_TABELI\_1>’, '=',*

*‘<NAZWA\_TABELI\_2>.<KOLUMNA\_TABELI\_2>’)->get();*

1. Warunki:

*$<NAZWA\_ZMIENNEJ> = DB::table('<NAZWA\_TABELI>')->*

*where('<NAZWA\_KOLUMNY>', '=', '<WARTOŚĆ>’)->*

*where('<NAZWA\_KOLUMNY\_2>', '<', '<WARTOŚĆ>')->get();*

1. Dodawnie wierszy:

*DB::table('<NAZWA\_TABELI>')->insert([*

*'<NAZWA\_KOLUMNY\_1>’ => '<WARTOŚĆ\_1>’,*

*‘<NAZWA\_KOLUMNY\_2>' => '<WARTOŚĆ\_2>’,*

*]);*

1. Modyfikacja wierszy:

*DB::table('<NAZWA\_TABELI>')*

*->where(‘<NAZWA\_KOLUMNY\_1>’, '=','<WARTOŚĆ>')*

*->update([*

*'<NAZWA\_KOLUMNY\_2>' => '<WARTOŚĆ\_2>',*

*]);*

1. Usuwanie:

*DB::table('<NAZWA\_TABELI>')*

*->where('<NAZWA\_KOLUMNY>', '=','<WARTOŚĆ>')*

*->delete();*

1. I wiele innych 😉
2. Widok
3. Do warstwy widoku wykorzystuje się pliki „<NAZWA\_PLIKU>.blade.php” , które tworzyć można w folderze resources->views->layouts. Pliki css oraz js możemy umieścić w folderze public, a następnie importować je poprzez „<link rel="stylesheet" href="{{ URL::asset('<NAZWA\_PLIKU>.css') }}" />” w kodzie.
4. Pliki blade pozwalają na wszystko, na co pozwala html, a do tego mają parę innych super mocy 😉   
   - Możliwość stworzenia elementów, które można wielokrotnie wykorzystać. Np. możemy stworzyć w folderze layouts folder „partials”, a w nim dowolne pliki blade, których ciało chcielibyśmy wklejać do innych. Następnie możemy je przywoływać poprzez *„@include('partials.<NAZWA\_PLIKU>’)”* w kodzie.
5. Dla bezpieczeństwa do każdego formularza powinno się wpisać dyrektywy @csrf, dzięki czemu Laravel będzie chronił aplikację przed atakami następującymi poprzez wysyłanie formularzy z innych witryn.
6. Do zmiennych można odwoływać się w strukturze html poprzez podwójne nawiasy klamrowe, na przykład: <p>Hello {{ $<NAZWA\_ZMIENNEJ> }}</p>
7. Możemy również wykorzystywać instrukcje warunkowe czy pętle:

*@if (1==2)*

*<p>Hi</p>*

*@elseif (1==3)*

*<h1>HI</h1>*

*@else*

*<h2>Hello</h2>*

*@endif*

@for ($x = 0; $x < 10; $x++)

<p> Aktualna runda: {{ $i }} </p>

@endfor

<ul>

@foreach($parking as $car)

<li class="example\_class">{{ $car->mark }}</li>

@endforeach

</ul>

1. W linkach do innych stron aplikacji należy podawać linki do ścieżek zdefiniowanych w kontrolerach, a nie plików.
2. Kontroler
3. Ruchem między stronami na witrynie powinny sterować kontrolery. Na początku może być jeden, a wraz z rozwojem witryny będą pojawiały się kolejne. Nowy kontroler można stworzyć przy pomocy polecenia „php artisan make:controller <NAZWA\_KONTROLERA>”, a on pojawi się w app->Http->Controllers
4. W kontrolerach możemy tworzyć metody przekierowujące lub zwracające widoki.

*public function showMyPrettySite()*

*{*

*return view('<NAZWA\_PLIKU\_BLADE\_PHP\_BEZ\_ROZSZERZENIA>');*

*}*

1. Ponadto kontrolery mogą przyjmować dane z formularzy

*public function showSiteWithFormParams(Request $request)*

*{*

*return view('<JAK\_WYŻEJ>', [*

*'<NAZWA\_ZMIENNEJ>' => $request->input('<NAME\_Z\_FORMULARZA>'),*

*'<INNA\_NAZWA\_ZMIENNEJ>' => $request->input('<INNE\_NAME\_Z\_FORMULARZA>'),*

*]);*

*}*

1. Oraz z adresu URL

public function showSiteWithUrlParams($<NAZWA\_ZMIENNEJ\_1\_Z\_URL>, $<NAZWA\_ZMIENNEJ\_2\_Z\_URL>)

{

return view(‘*<JAK\_WYŻEJ>*', [

‘<NAZWA\_ZMIENNEJ\_1 >’ => <NAZWA\_ZMIENNEJ\_1\_Z\_URL>,

'<NAZWA\_ZMIENNEJ\_2 >'=><NAZWA\_ZMIENNEJ\_2\_Z\_URL>

]);

}

1. W metodach kontrolera można pracować na bazach danych na przykład:

*public function showSIteWIthDBData()*

*{*

*DB::table('<NAZWA\_TABELI>’)->insert([*

*’<KOLUMNA>’ => ‘<WARTOŚĆ>’,*

*‘<INNA\_KOLUMNA>’ => '<INNA\_WARTOSC>',*

*]);*

*$<NAZWA\_ZMIENNEJ>= DB::table(‘<NAZWA\_TABELI>’)->get();*

*return view('*‘*<JAK\_WYŻEJ>*'*', ['<NAZWA\_ZMIENNEJ ' => $<NAZWA\_ZMIENNEJ]);*

*}*

1. Routingi

W folderze routes znajduje się plik web.php, w którym możemy łączyć wpisywane w URL ścieżki z odpowiednimi kontrolerami oraz ścieżkami. Po „Route::” umieszczamy metodę np. post lub get, a następnie ścieżkę, po której wpisaniu w URL zostanie zwrócony widok z klasy podanej w nawiasie kwadratowym, a dokładnie jej metody przywołanej po „::”. Na przykład:

*Route::get(‘/’, [<NAZWA\_KONTROLERA>::class,’<METODA\_KONTROLERA\_BEZ\_()>’]);*

Przykład argumentów w URL(zostaną przekazane do argumentów metod):

*Route::get('/home/{*<NAZWA\_ZMIENNEJ\_1\_Z\_URL*}/{‘*<NAZWA\_ZMIENNEJ\_2\_Z\_URL’ *}', [<NAZWA\_KONTROLERA>::class,'<METODA\_KONTROLERA\_BEZ\_()>']);*

Przykład przyjmowania formularza:

Route::post('/form', [*<NAZWA\_KONTROLERA>:*:class,'*<METODA\_KONTROLERA\_BEZ\_()>*']);

1. Autoryzacja

Implementacja autoryzacji w Laravelu została niesamowicie zautomatyzowana. Wystarczy wprowadzić kilka komend:

*composer require laravel/breeze* --dev

*php artisan breeze:install*

*php artisan migrate*

^--(instalcja Laravel Breeze oraz sworzenie w bazie potrzebnych tabel)

Następnie w web.php do routingów możemy dopisać „->middleware('auth');” przy routingach do których dostęp mają mieć tylko zalogowani użytkownicy, a logować się można poprzez http://127.0.0.1:8000/login

Na przykład: *Route::get('/pretty\_site', [MyController::class, 'showMyPrettySite*’*])-> middleware('auth');*

W kontrolerach można również sprawdzać, czy użytkownik jest zalogowany oraz pobierać dane użytkownika:

*if(auth()->check()){*

*$user = auth()->user();*

*}*

*return redirect('/login');*

Autoryzację można wyłączyć poprzez zmianę w pliku web.php z Auth::routes() na Auth::routes(['register' => false]);

1. Testy

W naszej implementacji nie pisaliśmy testów automatycznych, dlatego należy przeprowadzić testy manualne, dokładnie sprawdzając czy aplikacja działa.

1. Usprawnienia

W wyniku testów może się okazać, że musimy poprawić lub możemy usprawnić naszą aplikację.

1. Refaktoryzacja

Stworzenie aplikacji do dopiero początek, w kolejnym kroku powinno się przeprowadzić jej refaktoryzację, czyli przeprowadzenie zmian we wprowadzonym kodzie tak, aby zachować jego najwyższą jakość (Ten proces nie zmienia funkcjonalności aplikacji).