Dokumentacja projektu		AI1
Autor	Jakub Flis, 117794	**Data oddania**
Kierunek, rok	Informatyka, II rok, st. stacjonarne (3,5-l)	23.06.2022
Specjalizacja	–	
Grupa	LAB 1	**Ocena**
Temat projektu	*Witryna sprzedaży fotografii*	

Tematyka projektu:

Tematem jest strona służąca sprzedaży fotografii przez fotografów. Strona zawiera też zakładkę poświęconą fotografom, którzy współpracują ze stroną. Administrator strony po zalogowaniu może korzystać z interfejsu CRUD do zarządzania zdjęciami i fotografami. Użytkownik nie może się zalogować. Możliwe jest przeglądanie zdjęć, fotografów i informacji o nich. Użytkownik poza możliwością kliknięcia na zdjęcie zdjęcia w celu przejścia do widoku z powiększonym zdjęciem, nie ma możliwości interakcji ze stroną. Strona zatem ma charakter wyłącznie informacyjny.

Technologie wykonania:

Projekt został wykonany przy użyciu frameworka Laravel (https://laravel.com/docs/9.x/).

Wygląd strony został zrealizowany przy użyciu frameworka Bootstrap (https://getbootstrap.com/docs/5.2/getting-started/introduction/).

Baza danych MySQL.

Użyty został również git (https://en.wikipedia.org/wiki/Git).

Oprogramowanie serwerowe: XAMPP (Apache i MySQL) (https://www.apachefriends.org).

Środowisko programistyczne Visual Studio Code (https://code.visualstudio.com).

Źródło bezpłatnych obrazków: pixabay.com

Uruchomienie aplikacji:

composer install --no-interaction

php artisan storage:link

Uruchomić XAMPP, włączyć moduły Apache i MySQL. Utworzyć bazę danych projektai.

php artisan migrate

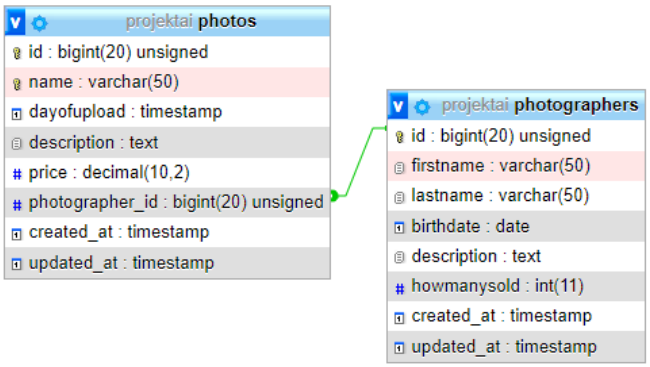
php artisan db:seed

php artisan serve

Przejść pod: http://127.0.0.1:8000 // dane logowania admina: jan@email.com, hasło: 1234.

Baza danych:

Składa się z dwóch tabel: fotografia i fotografowie. Każda fotografia ma przypisanego fotografa, użyty jest klucz obcy z tabeli fotografowie. Zdjęcie ma unikalną nazwę, datę dodania, opis i cenę. Fotograf ma imię, nazwisko, datę urodzenia, opis i liczbę sprzedanych zdjęć (liczba generowana losowo).



Połączenie z bazą danych: plik .env.

```
10
11    DB_CONNECTION=mysql
12    DB_HOST=127.0.0.1
13    DB_PORT=3306
14    DB_DATABASE=projektai
15    DB_USERNAME=root
16    DB_PASSWORD=
17
```

Zapisanie informacji o dodanym zdjęciu w bazie danych:

Photocontroller:

```php
public function create()
{
    return view('photos.create', [
        'photographers' => Photographer::all(),
        'today' => Carbon::now()->format('Y-m-d H:i:s')
    ]);
}

public function store(StorePhotoRequest $request)
{
    $input = $request->all();
    Photo::create($input);

    return redirect()->route('photos.index');
}
```

Seeding bazy:

```
PS C:\Users\Jamistrz\Desktop\FlisJakub_AI> php artisan db:seed
Seeding: Database\Seeders\UserSeeder
Seeded:  Database\Seeders\UserSeeder (117.03ms)
Seeding: Database\Seeders\PhotographerSeeder
Seeded:  Database\Seeders\PhotographerSeeder (59.25ms)
Seeding: Database\Seeders\PhotoSeeder
Seeded:  Database\Seeders\PhotoSeeder (27.03ms)
Database seeding completed successfully.
```

```php
public function run()
{
    Photo::truncate();
    Photo::upsert(
        [
            [
                'name' => 'Impresja', 'dayofupload' => Carbon::now()->format('Y-m-d H:i:s'), 'description' => 'Melancholijne kolory w najulotniejszej z chwil.',
                'price' => 19.99, 'photographer_id' => rand(1, 10)
            ],
            [
                'name' => 'Claude w kolebce', 'dayofupload' => Carbon::now()->format('Y-m-d H:i:s'), 'description' => 'Cud narodzin w humorze autora.',
                'price' => 60.00, 'photographer_id' => rand(1, 10)
            ],
            [
                'name' => 'Ambasada', 'dayofupload' => Carbon::now()->format('Y-m-d H:i:s'), 'description' => 'Kształt kształtowania kształtu polityki',
                'price' => 18.00, 'photographer_id' => rand(1, 10)
            ],
```

Autoryzacja:

W kontrolerach fotografa i fotografii:

Brak dostępu do widoków CRUDowych.

```php
public function __construct()
{
    $this->middleware('auth')->except(['index', 'show']);
}
```

Jeżeli użytkownik nie jest zalogowany nie widzi przycisków tabel.

```blade
<thead>
    <tr>
        <th scope="col">Tytuł zdjęcia</th>
        <th scope="col">Zdjęcie</th>
        <th scope="col">Data dodania</th>
        <th scope="col">Opis</th>
        <th scope="col">Nazwisko fotografa</th>
        <th scope="col">Cena</th>
        @if (Auth::check())
        <th scope="col"></th>
        <th scope="col"></th>
        @endif
    </tr>
</thead>
```

Walidacja:

Fotografia: np. cena musi spełniać regex (np. 5.00), łańcuch ceny musi mieć długość krótszą niż 6 (czyli cena maksymalnie milion), a id fotografa musi istnieć w tabeli fotografów.

```php
return [
    'name' => 'required|unique:photos|max:20',
    'description' => 'required|max:50',
    'price' => 'required|regex:/^\d+(\.\d{1,2})?$/|max:6',
    'photographer_id' => 'required|numeric|exists:photographers,id'
];
```

Przy fotografach data urodzenia musi być wcześniejsza niż obecna.

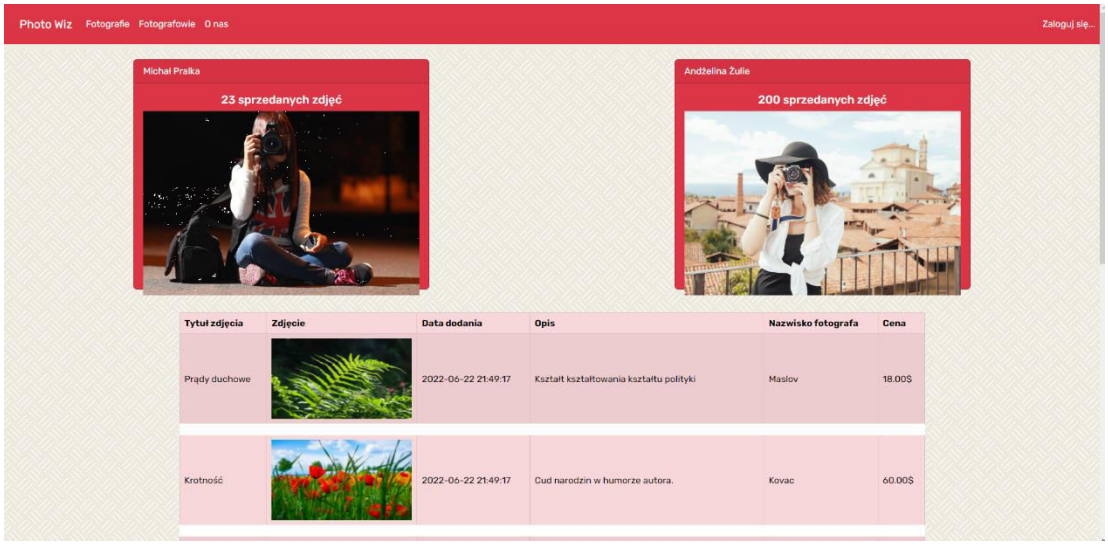
```php
return [
    'firstname' => 'required|max:20',
    'lastname' => 'required|max:20',
    'date_of_birth' => 'date_format:D.M.Y|before:today',
    'description' => 'required|max:511',
    'howmanysold' => 'required|numeric'
];
```

Trasy:

```
GET|HEAD        / ..... presentation.index
POST            _ignition/execute-solution ..... ignition.executeSolution › Spatie\LaravelIgnition › ExecuteSolutionController
GET|HEAD        _ignition/health-check ..... ignition.healthCheck › Spatie\LaravelIgnition › HealthCheckController
POST            _ignition/update-config ..... ignition.updateConfig › Spatie\LaravelIgnition › UpdateConfigController
GET|HEAD        about ..... presentation.about
GET|HEAD        api/user ..... generated::Q4G0iJ5De4zo9gQb
GET|HEAD        login ..... login › LoginController@login
POST            login ..... login.authenticate › LoginController@authenticate
GET|HEAD        logout ..... logout › LoginController@logout
GET|HEAD        photographers ..... photographers.index › PhotographerController@index
POST            photographers ..... photographers.store › PhotographerController@store
GET|HEAD        photographers/create ..... photographers.create › PhotographerController@create
GET|HEAD        photographers/{photographer} ..... photographers.show › PhotographerController@show
PUT|PATCH       photographers/{photographer} ..... photographers.update › PhotographerController@update
DELETE          photographers/{photographer} ..... photographers.destroy › PhotographerController@destroy
GET|HEAD        photographers/{photographer}/edit ..... photographers.edit › PhotographerController@edit
GET|HEAD        photos ..... photos.index › PhotoController@index
POST            photos ..... photos.store › PhotoController@store
GET|HEAD        photos/create ..... photos.create › PhotoController@create
GET|HEAD        photos/{photo} ..... photos.show › PhotoController@show
PUT|PATCH       photos/{photo} ..... photos.update › PhotoController@update
DELETE          photos/{photo} ..... photos.destroy › PhotoController@destroy
GET|HEAD        photos/{photo}/edit ..... photos.edit › PhotoController@edit
GET|HEAD        sanctum/csrf-cookie ..... generated::7yMX9IFeqF2jmKWQ › Laravel\Sanctum › CsrfCookieController@show
```

Przykładowe podstrony:

Podstrona z fotografami dla niezalogowanego użytkownika, na górze widoczni są dwaj losowi fotografowie.

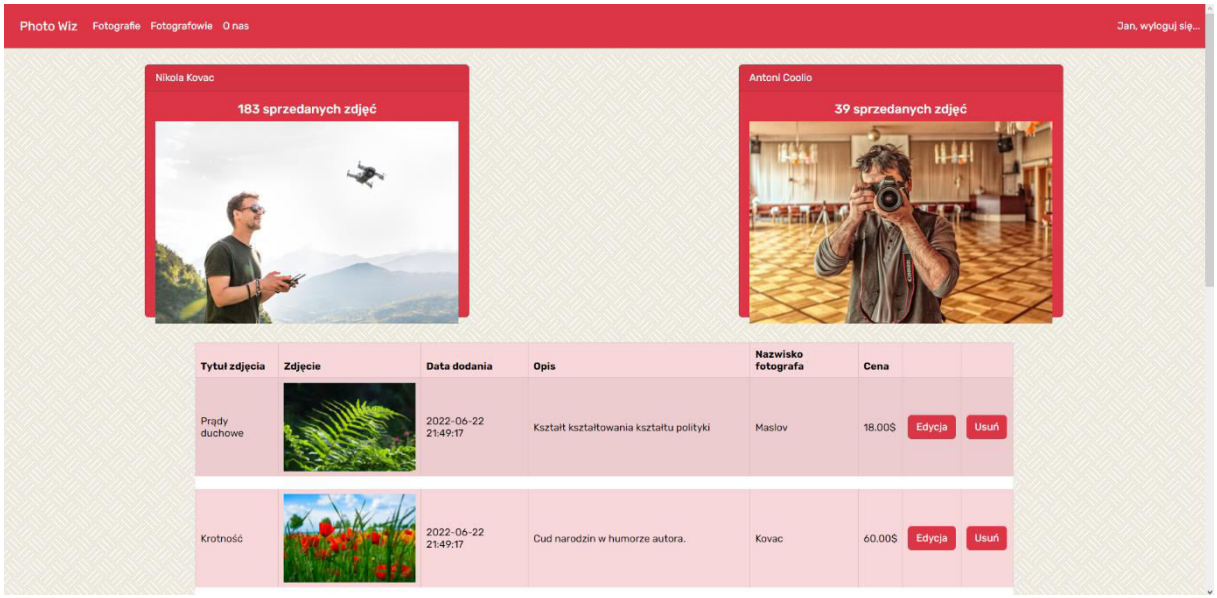


Losowanie unikalnych numerów fotografii przy użyciu tablicy i funkcji reindeksującej array_values:

```
<div class="container-fluid mt-5 position-relative">

    <div class="container-fluid m-2 d-flex justify-content-center">
        @php
        $max=20;
        $random_numbers=range(0,$max);

        $first=$random_numbers[rand(1, $max)];
        unset($random_numbers[array_search($first, $random_numbers, true)]);
        $random_numbers=array_values($random_numbers);

        $second=$random_numbers[rand(1, $max-1)];
        unset($random_numbers[array_search($second, $random_numbers, true)]);
        $random_numbers=array_values($random_numbers);

        $third=$random_numbers[rand(1, $max-2)];
        @endphp

        <img src="{{ asset('storage/img/'.'photo'.$first.'.jpg') }}" style="width: 20%; height: auto; margin-right: 5rem" alt="...">
        <img src="{{ asset('storage/img/'.'photo'.$second.'.jpg') }}" style="width: 20%; height: auto; margin-right: 5rem" alt="...">
        <img src="{{ asset('storage/img/'.'photo'.$third.'.jpg') }}" style="width: 20%; height: auto;" alt="...">
    </div>
    <a class="btn btn-danger position-absolute top-100 start-50 translate-middle" href="{{ route('photos.index') }}" role="button">Przeglądaj zdjęcia</a>
</div>
```

Po zalogowaniu:

Widoczne są przyciski CRUD.



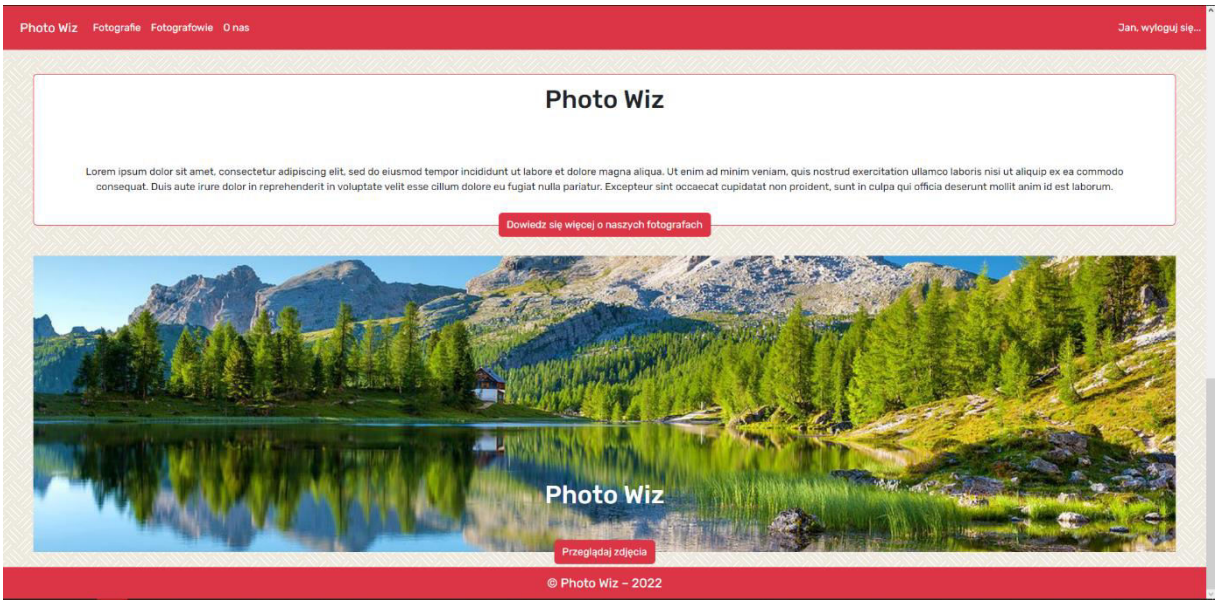
Widoczna jest paginacja – najpierw wyświetlane są nowsze zdjęcia.

Kod związany z paginacją (m. in. ukrywany jest przycisk następny gdy użytkownik jest na ostatniej podstronie):

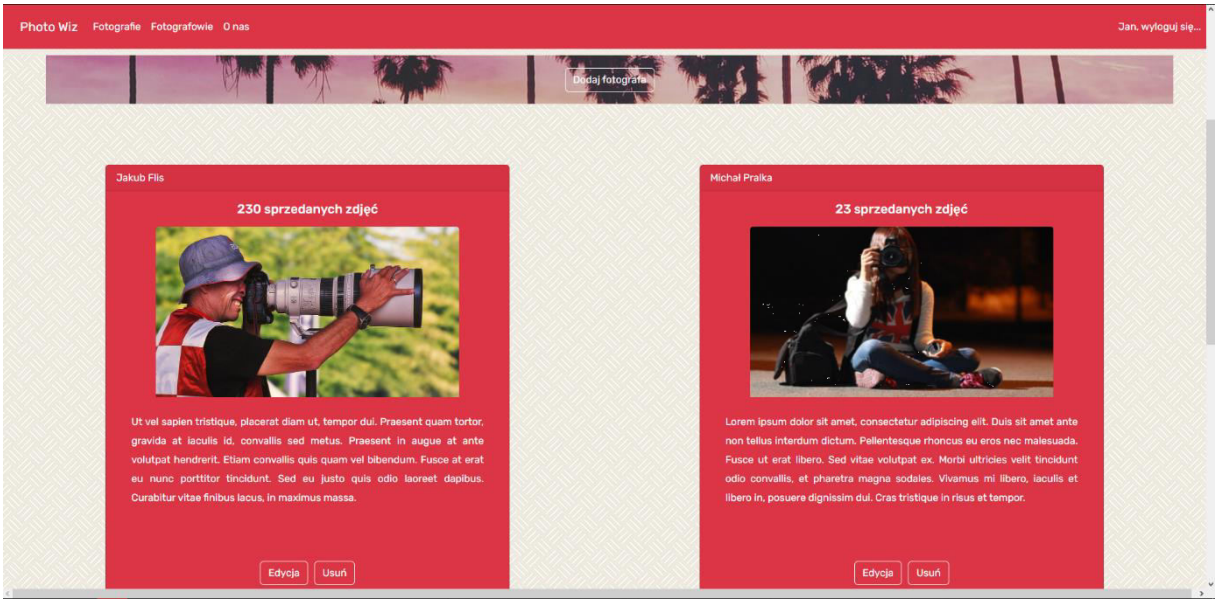
```
<nav aria-label="Page navigation example" class=" d-flex justify-content-center mt-4">
    <ul class="pagination">
        @if (!($photos->onFirstPage()))
        <li class="page-item" style="margin-right: 30px;">
            <a class="page-link bg-danger" style="padding: 10px 20px 10px 20px; color: white" href="{{$photos->previousPageUrl()}}" aria-label="Previous">
                <span aria-hidden="true">&laquo;</span>
            </a>
        </li>
        @endif

        @if (!($photos->onLastPage()))
        <li class="page-item">
            <a class="page-link bg-danger" style="padding: 10px 20px 10px 20px; color: white" href="{{$photos->nextPageUrl()}}" aria-label="Next">
                <span aria-hidden="true">&raquo;</span>
            </a>
        </li>
        @endif
    </ul>
</nav>
```

```
public function index() {
    return view('photos.index', [
        'photos' => DB::table('photos')->orderBy('id', 'DESC')->simplePaginate(7),
        'photographers' => Photographer::all(),
        'photographers_rand' => Photographer::all()->Random(2)
    ]);
}
```

Dolna część strony głównej.



Podstrona o fotografach.



Formularz edycji zdjęcia (id fotografa jest select boxem zawierającym istniejących fotografów z bazy).