AI1	Dokumentacja projektu
Autor	Jakub Opar, 125149
Kierunek, rok	Informatyka, II rok, st. stacjonarne (3,5-l)
Temat projektu	*Portal z ogłoszeniami kupna i sprzedaży*

Spis treści

1. Wstęp

Informacje ogólne

Sprzedam.pl to aplikacja internetowa gdzie można wystawiać ogłoszenia kupna i sprzedaży różnych rzeczy zaczynając od koszulki czy kubka kończąc na samochodach lub komputerach. Użytkownik aplikacji będzie miał możliwość przeglądania ofert, zalogowany użytkownik może dodawać, edytować i usuwać swoje oferty, a administrator ponadto ma możliwość zarządzania użytkownikami oraz ich ofertami. Podczas przeglądania ofert użytkownik może obejrzeć większą liczbę ofert za pomocą przycisku „Pokaż więcej" lub skorzystać z wyszukiwarki.

Założenia aplikacji

- Aplikacja realizuje operacje CRUD
- Użytkownik ma możliwość przeglądania ogólnie dostępnych zasobów
- Użytkownik ma możliwość zarejestrowania się i zalogowania
- Użytkownik ma możliwość zarządzania swoimi ofertami (CRUD)
- Użytkownik ma możliwość zarządzania obrazkami
- Aplikacja jest zabezpieczona pod kątem wykonywania niedozwolonych operacji
- Dane wprowadzane do formularzy są walidowane
- Strona posiada swoje widoki błędów
- Aplikacja posiada schludny i przyjazny dla użytkownika interfejs graficzny
- Administrator ma możliwość zarządzania zasobami w szerszym zakresie
- Można wyszukiwać oferty za pomocą wyszukiwarki
- Można na stronie pokazać więcej ofert za pomocą przycisku „Więcej ofert"

2. Informacje techniczne

Programy

Podczas tworzenia aplikacji wykorzystywano IDE Visual Studio Code 1.89.1 , Xampp v3.3.0 oraz system zarządzania pakietami Composer 2.7.6. Każde z tych programów jest darmowe.

Visual Studio Code 1.89.1:

Pobieranie:

https://code.visualstudio.com/updates/v1_89

Dokumentacja:

https://code.visualstudio.com/docs

Licencja:

https://code.visualstudio.com/License

Xampp v3.3.0:

Pobieranie:

https://www.apachefriends.org/pl/download.html

Dokumentacja:

https://www.apachefriends.org/docs/

Licencja:

https://www.apachefriends.org/pl/about.html

Composer 2.7.6:

Pobieranie:

https://getcomposer.org/download/

Dokumentacja:

https://getcomposer.org/doc/

Licencja:

https://github.com/composer/composer/blob/main/LICENSE

Technologie

Aplikacja jest oparta na framewarku Laravel 11.9.2 z wersją php 8.0.30. Widoki są zrobione z wykorzystaniem Bootstrapa v5.3. Ponadto aplikacja używa biblioteki charts.js 4.4.1. Serwer xampp wykorzystuje MariaDB w wersji 10.4.28. Każda z tych technologii jest darmowa.

Laravel 11.9.2:

Pobieranie:

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

Dokumentacja:

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

Licencja:

https://github.com/translation/laravel/blob/master/LICENSE

php 8.0.30:

Pobieranie:

https://windows.php.net/download#php-8.3

Dokumentacja:

https://www.php.net/docs.php

Licencja:

https://www.php.net/license/index.php

Bootstrap v5.3:

Pobieranie:

https://getbootstrap.com/docs/5.3/getting-started/download/

Dokumentacja:

https://getbootstrap.com/docs/5.3/getting-started/introduction/

Licencja:

https://github.com/twbs/bootstrap/blob/main/LICENSE

Chart.js 4.4.1:

Pobieranie:

https://www.chartjs.org/docs/latest/getting-started/installation.html

Dokumentacja:

https://www.chartjs.org/docs/latest/

Licencja:

https://github.com/chartjs/Chart.js/blob/master/LICENSE.md

MariaDB 10.4.28 (MySQL):

Pobieranie:

https://mariadb.com/kb/en/mariadb-10-4-28-release-notes/

Dokumentacja:

https://mariadb.com/kb/en/documentation/

Licencja:

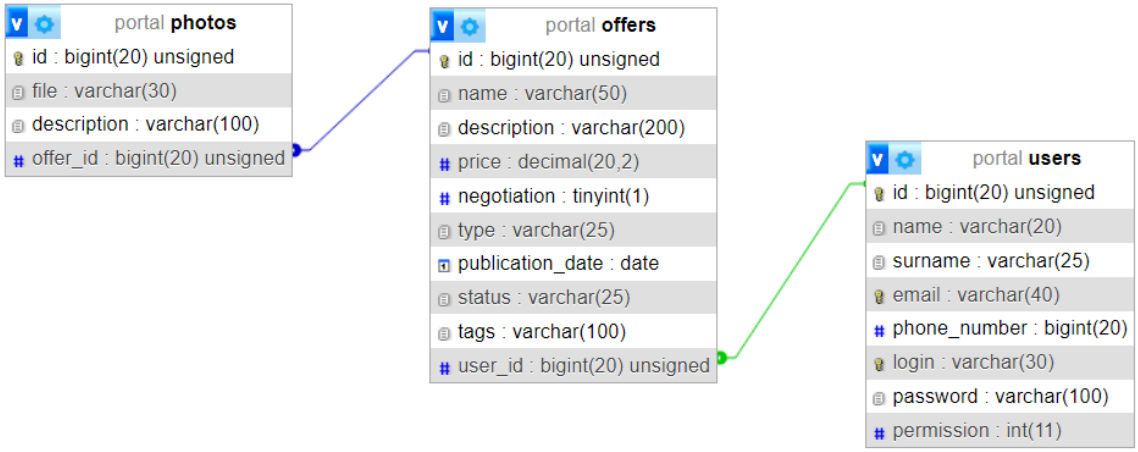
https://mariadb.com/kb/en/mariadb-licenses/

3. Baza danych

Opis ogólny

Baza danych portal składa się z trzech tabel: photos, offers oraz users. Główną z nich jest tabela offers, w której przechowywane są informacje o ofertach znajdujących się w aplikacji. Do oferty są przypisane zdjęcia, które znajdują się w tabeli photos. Tabela ta zawiera nazwę pliku, opis zdjęcia oraz identyfikator oferty do której są przypisane. Oferta również posiada informacje jaki użytkownik ją dodał. W tabeli users znajdują się informacje o użytkowniku oraz jego dane logowania.

Diagram ERD



Rysunek 3.1. Diagram ERD.

Rysunek 3.1 przedstawia diagram związków encji na którym są opisane tabele oraz relacje między nimi.

Tabele

Tabela: offers

Nazwa Kolumny	Typ	Właściwości	Opis
id	bigint	Długość maksymalna: 20 Unikalny Klucz podstawowy	Unikalny identyfikator dla oferty
name	varchar	Długość maksymalna: 50	Nazwa oferty
description	varchar	Długość maksymalna: 200	Opis oferty
price	decimal	Długość maksymalna: 20 Dwa miejsca po przecinku Nie może być ujemna	Cena oferty
negotiation	tinyint	Długość maksymalna: 1	Informacja czy można negocjować cenę oferty czy nie. 1 jeśli można 0 jeśli nie
type	varchar	Długość maksymalna: 25	Informacja o typie oferty. Czy to jest sprzedaż czy kupno
publication_date	date	Nie może przyjmować daty przed 2024 rokiem	Informacja o dacie publikacji
status	varchar	Długość maksymalna: 25	Informacja o statusie oferty, czy jest aktualna, zarezerwowana czy zakończona
tags	varchar	Długość maksymalna: 100	Tagi użytkownika, pomogające w znalezieniu oferty
user_id	bigint	Długość maksymalna: 20 Klucz obcy	Identyfikator użytkownika który dodał ofertę

Tabela: photos

Nazwa Kolumny	Typ	Właściwości	Opis
id	bigint	Długość maksymalna: 20 Unikalny Klucz podstawowy	Unikalny identyfikator dla zdjęcia
file	varchar	Długość maksymalna: 30	Nazwa pliku zdjęcia. Sam plik jest przechowywany w storage
description	varchar	Długość maksymalna: 100	Opis zdjęcia
offer_id	bigint	Długość maksymalna: 20 Klucz obcy	Identyfikator oferty do której jest przypisane zdjęcie

Tabela: Users

Nazwa kolumny	Typ	Właściwości	Opis
id	bigint	Długość maksymalna: 20 Unikalny Klucz podstawowy	Unikalny identyfikator dla użytkownika
name	varchar	Długość maksymalna: 20	Imię użytkownika
surname	varchar	Długość maksymalna: 25	Nazwisko użytkownika
email	varchar	Długość maksymalna: 40 Unikalny	Adres email użytkownika
Phone_number	bigint	Długość maksymalna: 9	Numer telefonu użytkownika
login	varchar	Długość maksymalna: 30 Unikalny	Login użytkownika
password	varchar	Długość maksymalna: 100 Zahashowany	Hasło użytkownika
permission	int	Długość maksymalna: 1	Poziom uprawnien użytkownika

Opis relacji

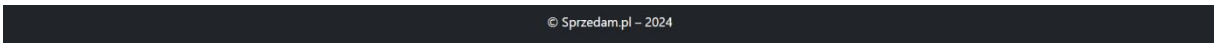
Tabela 1	Tabela 2	Typ relacji
offers	photos	Jeden do wiele
users	offers	Jeden do wiele

Jeden użytkownik może mieć wiele ofert i jedna oferta może mieć wiele zdjęć.

4. Opis warstwy użytkowej

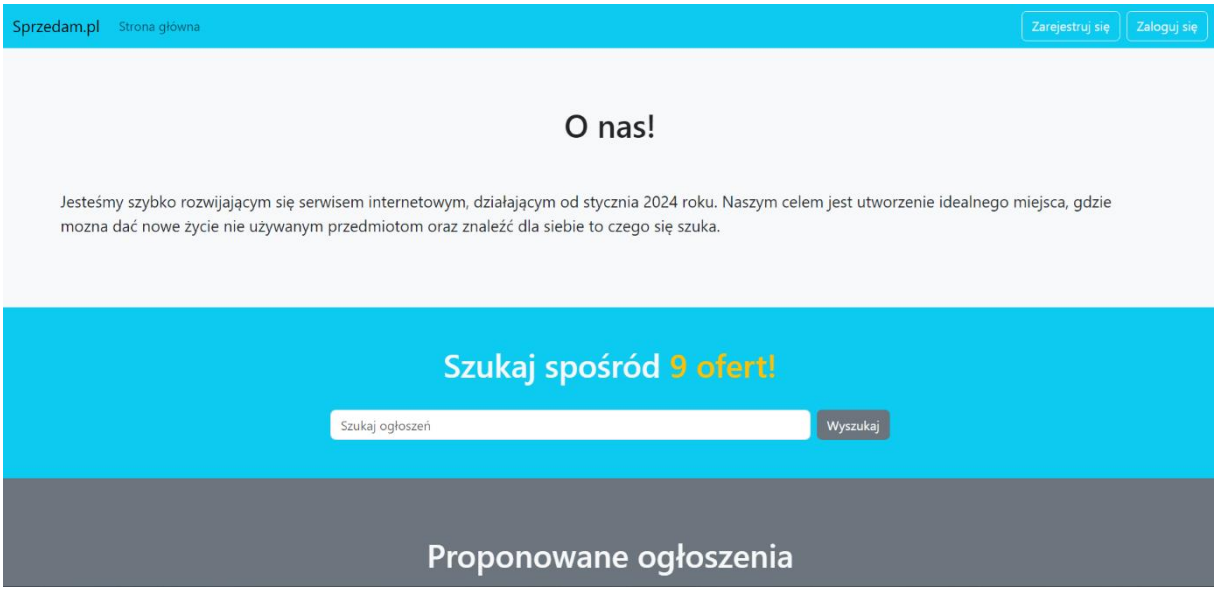
Elementy stron

Rysunek 4.1. Nawigacja.

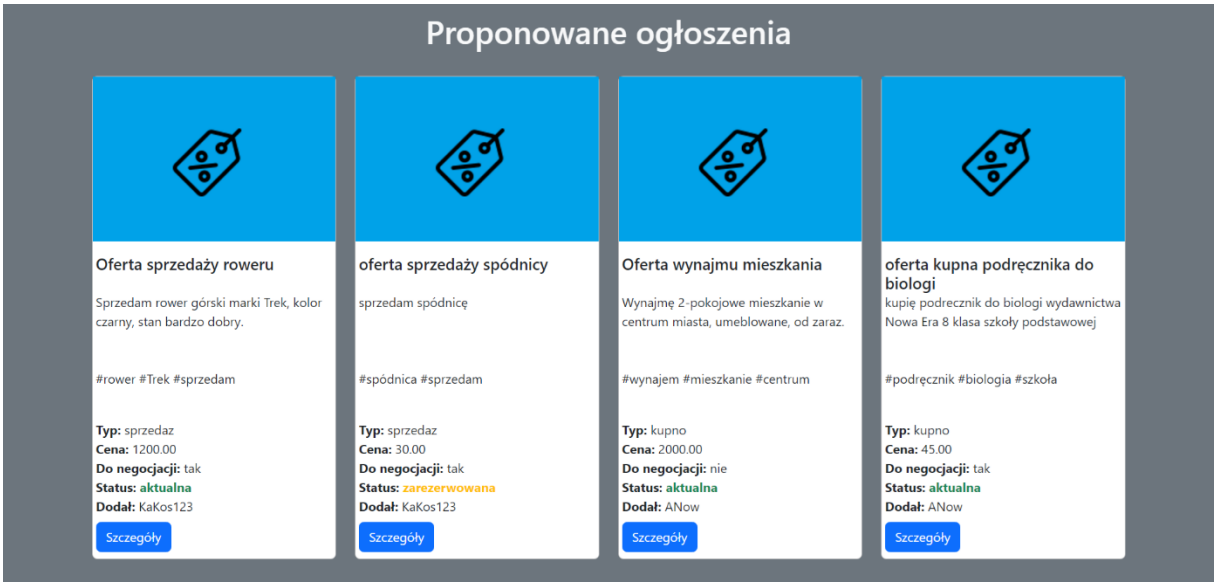


Rysunek 4.2. Stopka.

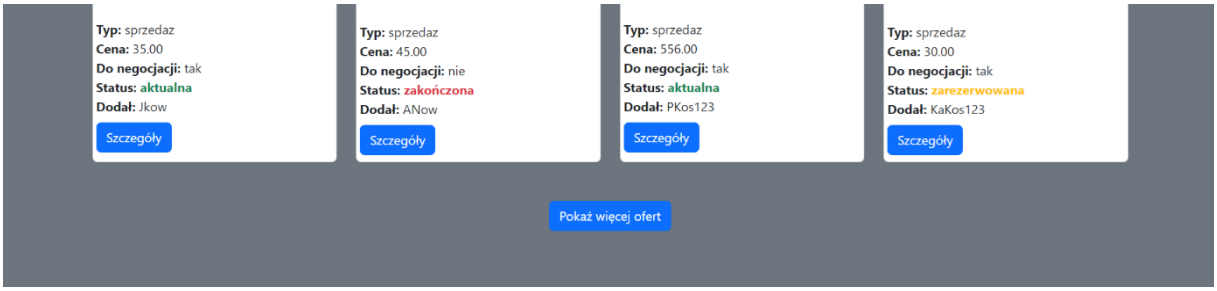
Strona główna



Rysunek 4.3. Góra strony głównej.



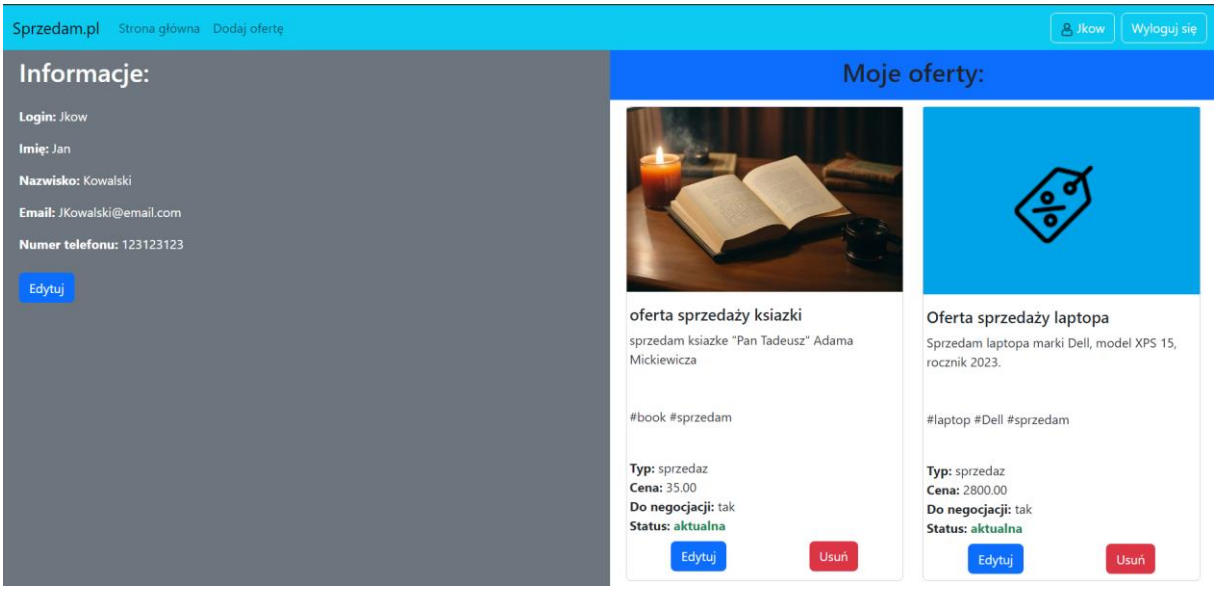
Rysunek 4.4. Sekcja proponowanych ofert na stronie głównej.



Rysunek 4.5. Przycisk do pokazanie większej ilości ofert.

Rysunki 4.3 i 4.4 pokazują wygląd strony głównej. Znajduje się na niej wyszukiwarka pozwalająca na przeszukiwanie ofert (Więcej o niej w opisach funkcjonalności) oraz wyżej informacja ile ofert ogólnie jest zapisanych w aplikacji. Poniżej znajduje się sekcja z proponowanymi ogłoszeniami które są wybierane losowo. Aby zobaczyć więcej ofert można użyć opcji „Pokaż więcej ofert” pokazanej na rysunku 4.5 (Więcej o niej w opisach funkcjonalności). W karcie każdej z ofert znajduje się przycisk „Szczegóły” kierujący na widok z większą liczbą informacji o danej ofercie.

Panel użytkownika

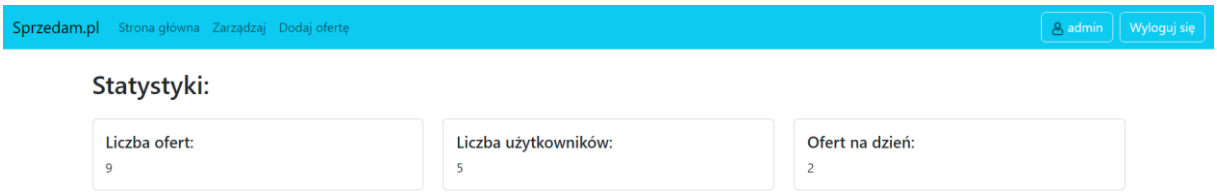


Rysunek 4.6. Panel użytkownika.

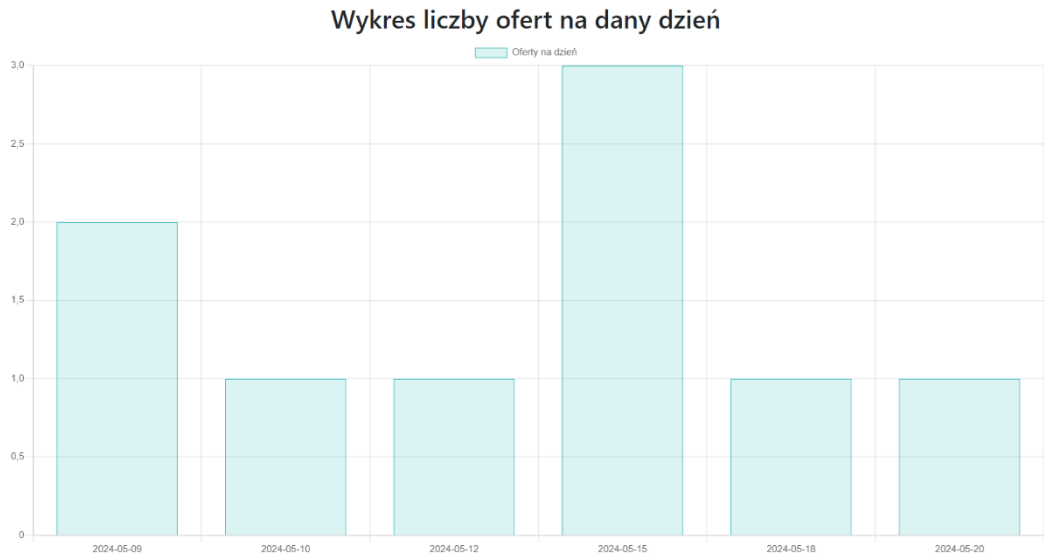
Na rysunku 4.6 jest przedstawiony panel zalogowanego użytkownika. Jest on podzielony na dwie kolumny. Na lewej pokazane są dane użytkownika które może poza loginem zmienić po kliknięciu przycisku „Edytuj". W prawej kolumnie za to pokazane są oferty danego użytkownika, gdzie użytkownik ma możliwość edycji ich lub usunięcia.

Również kiedy jest się zalogowanym na konto użytkownika w pasku nawigacyjnym pokazuje się dodatkowa opcja „Dodaj ofertę", która służy do dodawania ofert.

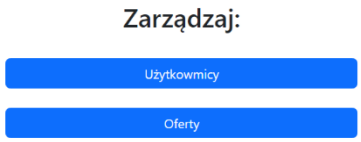
Panel administratora



Rysunek 4.7. Liczbowe statystyki w panelu administratora.



Rysunek 4.8. Wykres w panelu administratora.



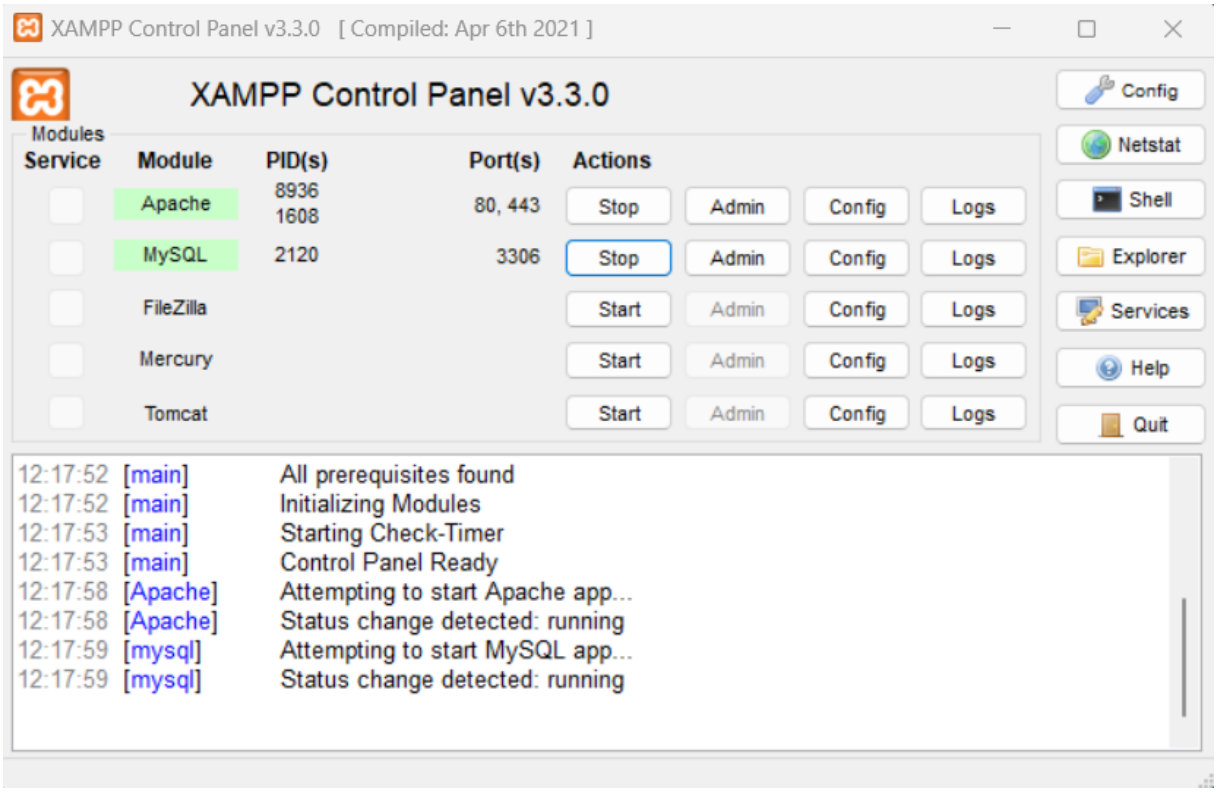
Rysunek 4.9. Panel wyboru tabeli do zarządzania w panelu administratora.

Rysunki od 4.7 do 4.9 pokazują elementy panelu administratora. Na pierwszym z nim są pokazane karty ze statystykami które są wyliczane (więcej o nich w opisie funkcjonalności). Drugim elementem jest wykres liczby ofert na dzień (o nim również więcej w opisach funkcjonalności) oraz trzecim elementem jest panel wyboru tabel, obydwa przyciski kierują na odpowiadającą im tabelę.

W przypadku bycia zalogowanym na konto administratora poza opcją „Dodaj ofertę” pojawia się również opcja „Zarządzaj”, która kieruje na panel administratora który jest w tym podpunkcie opisywany.

5. Uruchamianie aplikacji

Wymagane programy



Rysunek 5.1. Poprawnie uruchomiony Xampp z odpowiednimi usługami

Przed uruchomieniem aplikacji należy uruchomić serwer Xampp i uruchomić na nim usługi Apache i MySQL tak jak zrobione to jest na rysunku 5.1. Również należy pamiętać aby mieć zainstalowany na komputerze composer oraz Visual Studio Code.

Jeśli się nie posiada tych narzędzi to linki do pobrania ich są w rozdziale 2 dokumentacji.

```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=
APP_DEBUG=false
APP_TIMEZONE=UTC
APP_URL=http://localhost

APP_LOCALE=pl
APP_FALLBACK_LOCALE=en
APP_FAKER_LOCALE=pl_PL

APP_MAINTENANCE_DRIVER=file
APP_MAINTENANCE_STORE=database

BCRYPT_ROUNDS=12

LOG_CHANNEL=stack
LOG_STACK=single
LOG_DEPRECATIONS_CHANNEL=null
LOG_LEVEL=debug

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=portal
DB_USERNAME=root
DB_PASSWORD=

SESSION_DRIVER=database
SESSION_LIFETIME=120
SESSION_ENCRYPT=false
SESSION_PATH=/
SESSION_DOMAIN=null

BROADCAST_CONNECTION=log
FILESYSTEM_DISK=local
QUEUE_CONNECTION=database
```

Rysunek 5.2. Klik konfiguracyjny .env.example

Rysunek 5.2 zawiera informacje dotyczące konfiguracji aplikacji.

```
APP_LOCALE=pl
APP_FALLBACK_LOCALE=en
APP_FAKER_LOCALE=pl_PL
```

Te linie odpowiadają za ustawienie języka aplikacji. Jest to język polski, a jako drugorzędny język ustawiony jest język angielski.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=portal
DB_USERNAME=root
DB_PASSWORD=
```

Te linie odpowiadają za konfigurację połączenia z bazą danych. Wybrana jest tutaj baza danych mysql. Użytkownik bazy danych jest domyślny, a host bazy danych to localhost.

Skrypt stawiający aplikację i bazę danych

Aplikacja > start.bat

```
@echo off

%systemDrive%\xampp\mysql\bin\mysql -uroot -e "CREATE DATABASE IF NOT EXISTS portal;"

if %errorlevel% neq 0 (
    msg %username% "Nie udało się utworzyć bazy danych."
    exit /b %errorlevel%
)

if exist storage\app\public\photos (
    rmdir /s /q storage\app\public\photos
    mkdir storage\app\public\photos
)

xcopy /s /i /y "storage\app\public\photos-example\*" "storage\app\public\photos\"

php -r "copy('.env.example', '.env');"

call composer install

call php artisan migrate:fresh --seed

call php artisan key:generate

call php artisan storage:link

call php artisan serve

start http://127.0.0.1:8000

code .

```

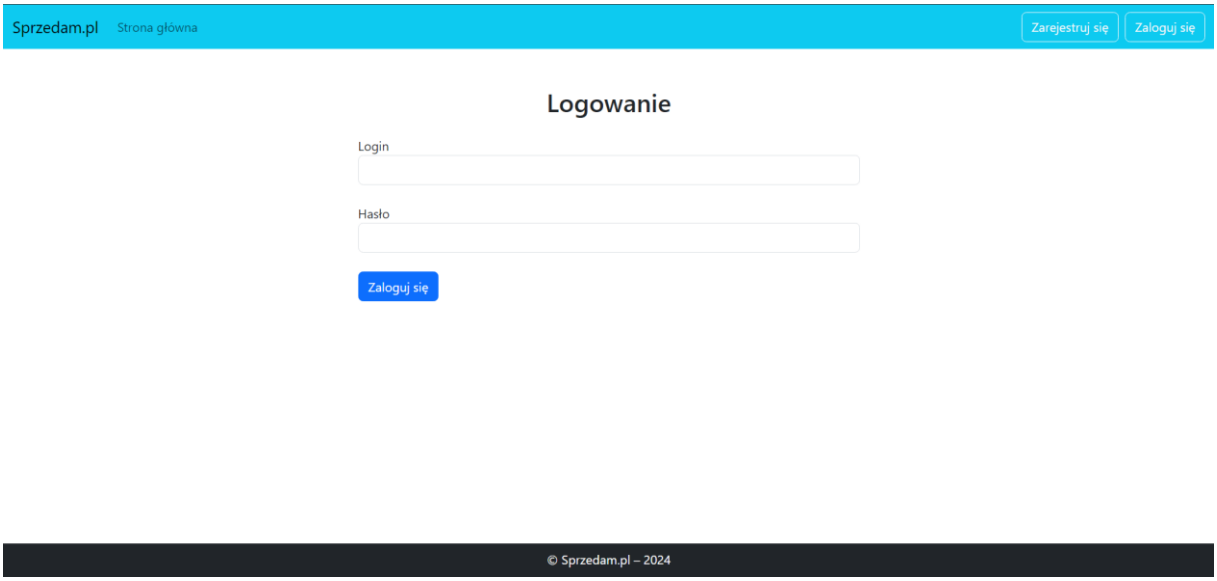
Rysunek 5.3. Kod skryptu stawiającego aplikacje i bazę danych.

Powyższy skrypt odpowiada za uruchomienie aplikacji i postawienie bazy danych. Odbywa się to w następujących etapach:

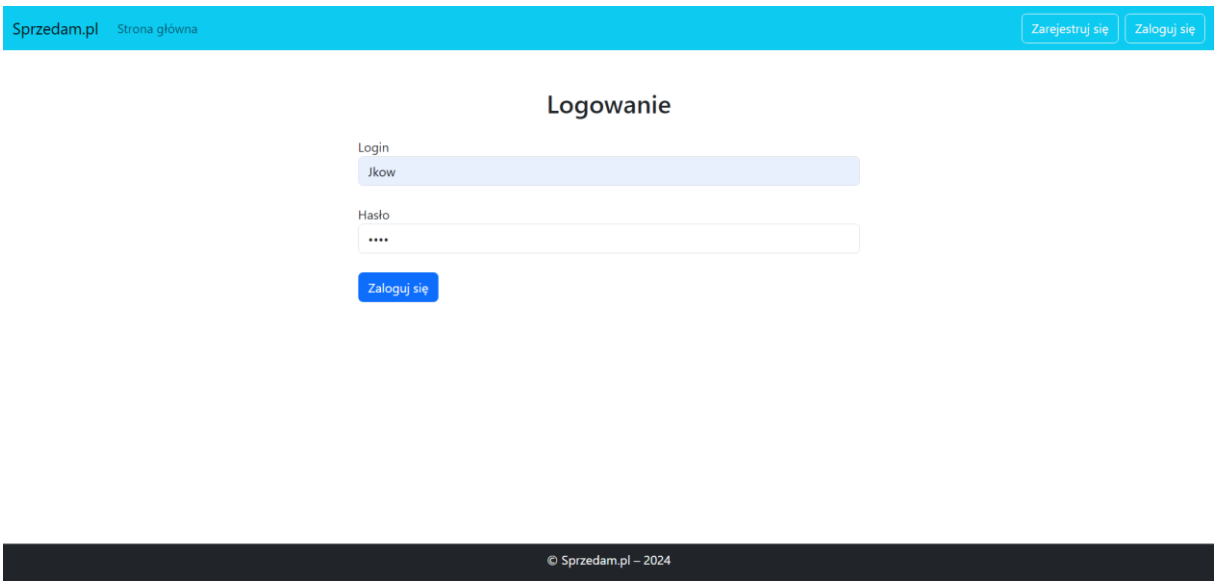
1. Utworzenie bazy danych „Portal” jeśli nie istnieje.
2. W przypadku gdy nie udało się utworzyć bazy danych program zwraca odpowiednią informację i skrypt kończy działanie.
3. Czyszczenie zawartości katalogu photos, który zawiera zdjęcia wstawione przez użytkowników.
4. Kopiowanie zdjęć z katalogu photos-example do katalogu photos. Znajdują się w nim przykładowe zdjęcia wykorzystywane przez przykładowe dane zasiane przez seeder.
5. Przygotowywanie pliku .env poprzez skopiowanie zawartości pliku .env-example zawierającego konfigurację aplikacji.
6. Instalacja zależności composera.
7. Odświeżenie migracji wraz z ponownym zasianiem przykładowych danych.
8. Wygenerowanie klucza aplikacji
9. Utworzenie symbolicznego linku do storage
10. Uruchomienie serwera aplikacji
11. Otworzenie domyślnej przeglądarki z wpisanym adresem 127.0.0.1:8000.
12. Otworzenie projektu w Visual Studio Code.

6. Funkcjonalności

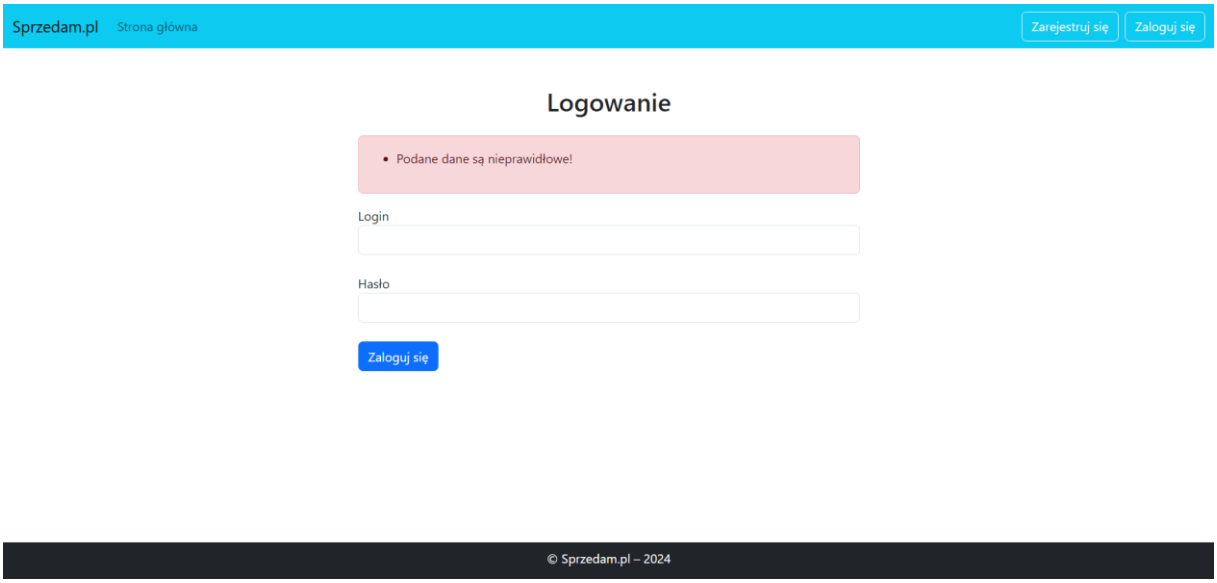
Logowanie i rejestracja



Rysunek 6.1. Widok logowania.



Rysunek 6.2. Widok logowania z wprowadzonymi danymi.



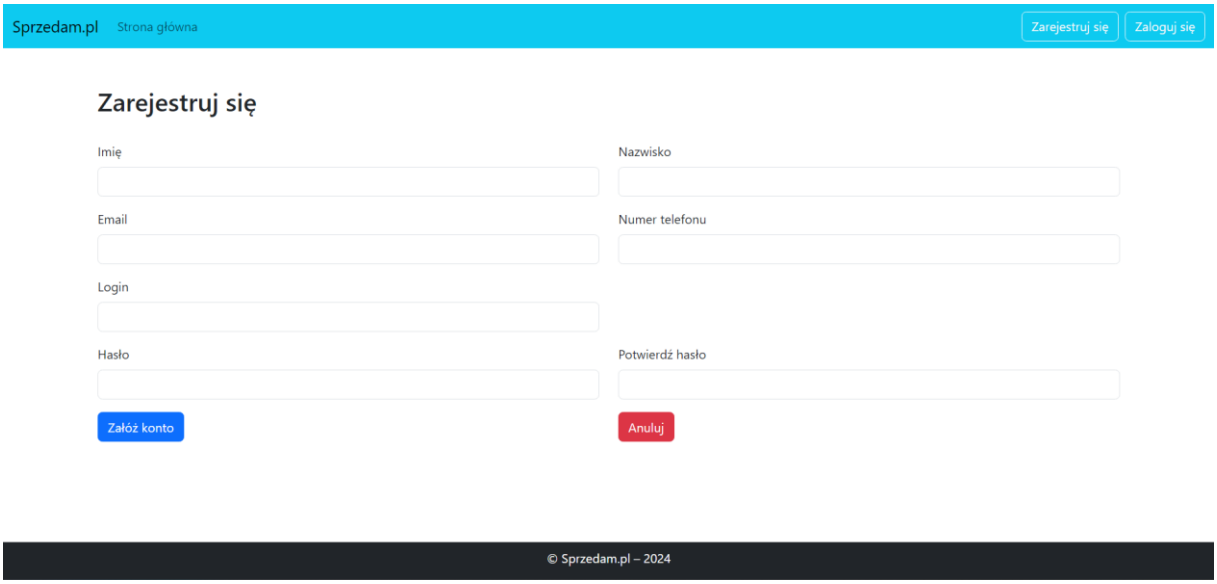
Rysunek 6.3. Widok logowania z komunikatem błędu.

```php
public function login(Request $request)
{
    if (Gate::allows('is-logged-in')) {
        return back();
    }

    $credentials = $request->only('login', 'password');

    if (Auth::attempt($credentials)) {
        $request->session()->regenerate();

        return redirect()->intended(route('mainPage'));
    }

    return back()->withErrors([
        'login' => 'Podane dane są nieprawidłowe!',
    ]);
}
```

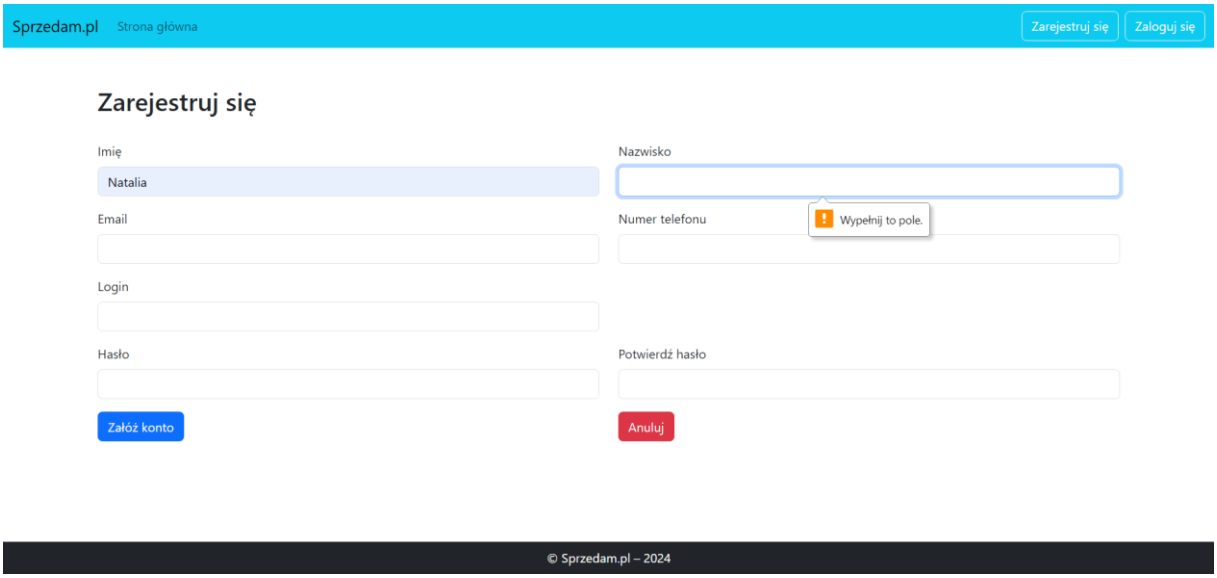
Rysunek 6.4. Kod funkcji na logowanie się

Widok logowania posiada dwa pola do wprowadzania danych tak jak pokazano na rysunku 6.1: Login i hasło. W przypadku wprowadzenia danych jak na rysunku 6.2 następuje zalogowanie użytkownika. W przypadku kiedy dane nie były poprawne zostaje wyrzucony komunikat o błędzie. (patrz rysunek 6.3)

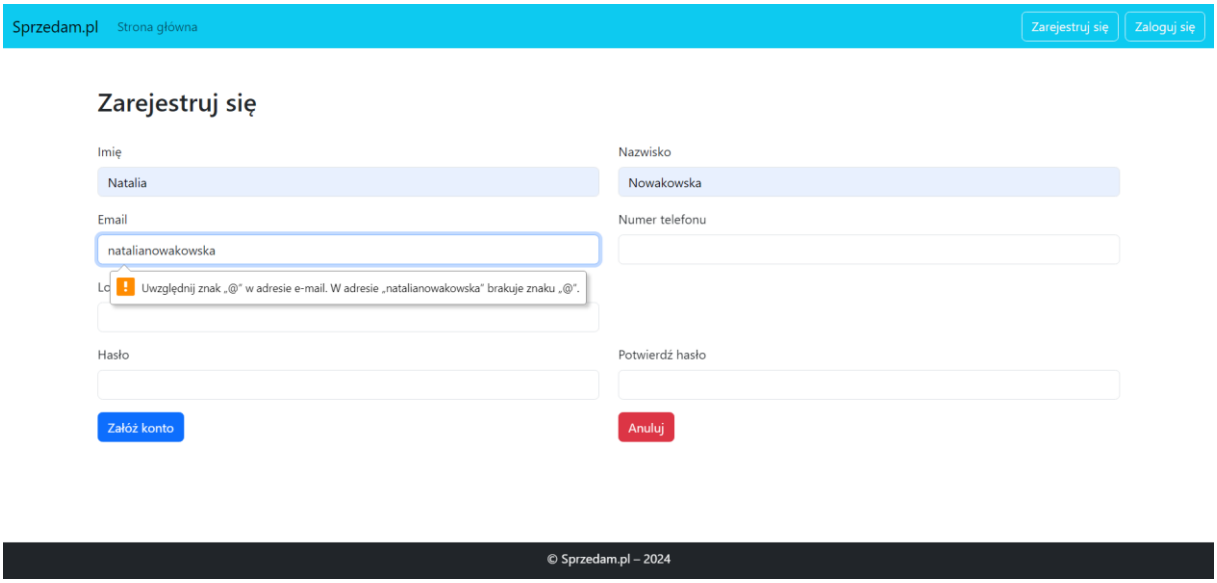
Funkcja login pokazana na rysunku 6.4 przyjmuje jeden argument w postaci requestu. Na początku funkcja sprawdza czy użytkownik jest już zalogowany. Jak jest to następuje powrót na stronę z której przyszedł na trasę logowania. Jeśli nie jest zalogowany to z request zostają pobrane wprowadzone przez użytkownika login i hasło. Następnie następuje próba zalogowania się. Jeśli się udało użytkownik zostaje przekierowany na stronę główną i sesja z requestu zostaje wygenerowana na nowo, a jeśli nie to zostaje przekierowany na stronę logowania się wraz z komunikatem o błędzie.



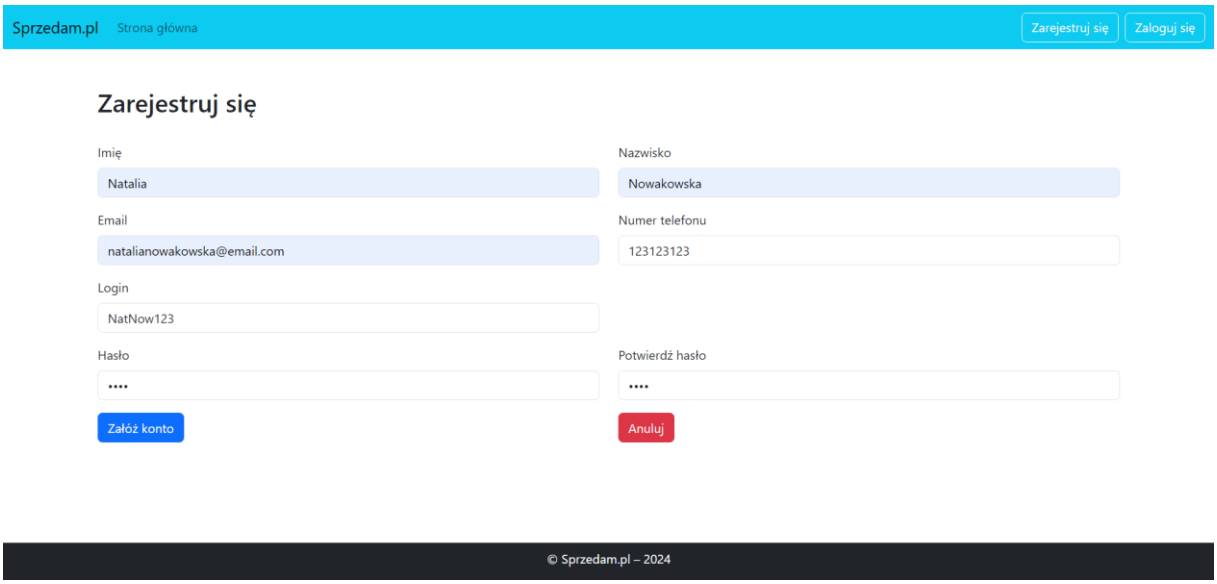
Rysunek 6.5. Strona rejestracji.



Rysunek 6.7. Brak wprowadzenia wymaganych danych.



Rysunek 6.8. Błędne podanie adresu email.



Rysunek 6.9. Poprawnie podane dane.

```php
public function register(RegisterRequest $request)
{
    if (Gate::allows('is-logged-in')) {
        abort(403);
    }
    try {
        $input = $request->validated();
        $input['permission'] = 2;
        User::create($input);

        return redirect()->route('loginPage');
    } catch (\Illuminate\Validation\ValidationException $e) {
        return redirect()->back()
            ->withErrors($e->errors())
            ->withInput();
    }
}
```

Rysunek 6.10. Funkcja odpowiadająca za rejestracje.

```php
class RegisterRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     */
    public function authorize(): bool
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array<string, \Illuminate\Contracts\Validation\ValidationRule|array<mixed>|string>
     */
    public function rules(): array
    {
        return [
            'name' => 'required|string|max:20',
            'surname' => 'required|string|max:25',
            'email' => [
                'required',
                'string',
                'email',
                'max:40',
                'unique:users,email,',
                'regex:/^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/'
            ],
            'phone_number' => 'required|numeric|min:9|max:9',
            'login' => 'required|string|max:30|unique:users,login',
            'password' => 'required|string|max:100',
            'commitPassword' => 'required|string|max:100|same:password'
        ];
    }
}
```

Rysunek 6.11. Reguły walidacji w RegisterRequest.

Na rysunku 6.5 przedstawiona jest strona z formularzem do rejestracji. W przypadku wprowadzenia poprawnych danych jak na rysunku 6.9 następuje poprawne zarejestrowanie się. W przypadku kiedy któreś wymagane pole zostało pominięte lub dane w nie wprowadzone były nie poprawne to następuje wyświetlenie odpowiedniego komunikatu jak na rysunkach 6.7 i 6.8.

Funkcja przedstawiona na rysunku 6.10 przyjmuje argument RegisterRequest który posiada zasady walidacji przedstawione na rysunku 6.11. Oznaczają one, że:

- Pole name jest wymagane i musi być ciągiem znaków o maksymalnej długości 20 znaków.
- Pole surname również jest wymagane i musi być ciągiem znaków, a maksymalna długość wynosi 25 znaków.
- Adres e-mail, reprezentowany przez pole email, jest obowiązkowy i musi być poprawnym ciągiem znaków, spełniającym standardy adresu e-mail. Maksymalna długość tego pola to 40 znaków. Ponadto adres e-mail musi być unikalny, a jego format musi odpowiadać określonemu wyrażeniu regularnemu, które sprawdza poprawność adresu e-mail.
- Pole phone_number jest wymagane i musi być liczbą o dokładnie 9 cyfrach.
- Pole login również jest wymagane, musi być ciągiem znaków o maksymalnej długości 30 znaków i musi być unikalne.
- Pole password jest obowiązkowe i musi być ciągiem znaków o maksymalnej długości i 100 znaków.
- Pole commitPassword jest również wymagane, musi być ciągiem znaków o tej samej maksymalnej długości co password, a dodatkowo jego wartość musi być identyczna jak wartość pola password.

Najpierw następuje sprawdzenie czy użytkownik nie jest obecnie zalogowany. Jeśli jest następuje wyrzucenie błędu 403. Następnie sprawdzana jest poprawność wprowadzonych danych, jeśli nie są poprawne zostaje wyrzucony ValidationException i następuje przekierowanie na stronę rejestracji wraz z komunikatami o błędach. Jeśli dane są poprawne to wartość pola permission zostaje ustawiona domyślnie na „2” co oznacza, że zarejestrowany użytkownik ma uprawnienia użytkownika. (Więcej o uprawnieniach w podrozdziale o użytkownikach) Na koniec następuje utworzenie nowego użytkownika i przekierowanie na stronę logowania.

```
public function logout(Request $request)
{
    Auth::logout();

    $request->session()->invalidate();
    $request->session()->regenerateToken();

    return redirect('/');
}
```

Rysunek 6.12. Funkcja odpowiadająca za wylogowanie się.

Rysunek 6.12 przedstawia funkcję odpowiadającą za wylogowanie się użytkownika. Przyjmuje ona jeden argument czyli request. Najpierw ona wyloguje użytkownika, a później następuje unieważnienie bieżącej sesji użytkownika i wygenerowanie nowego tokenu csrf. Na koniec odbywa się przekierowanie na stronę główną.

Testowe dane logowania

Administrator:

Login: admin

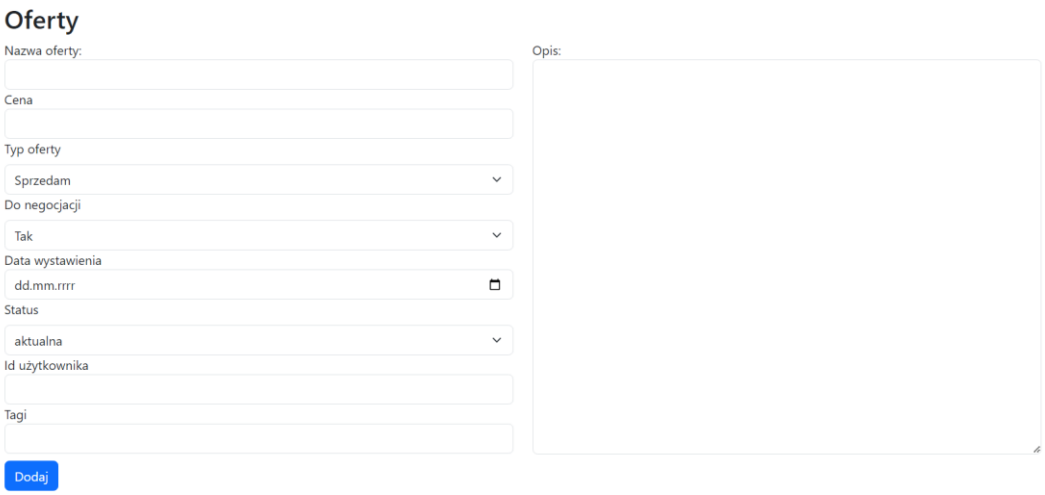
Hasło: admin

Użytkownik:

Login: Jkow

Hasło: 1234

CRUD (Create, update, delete)

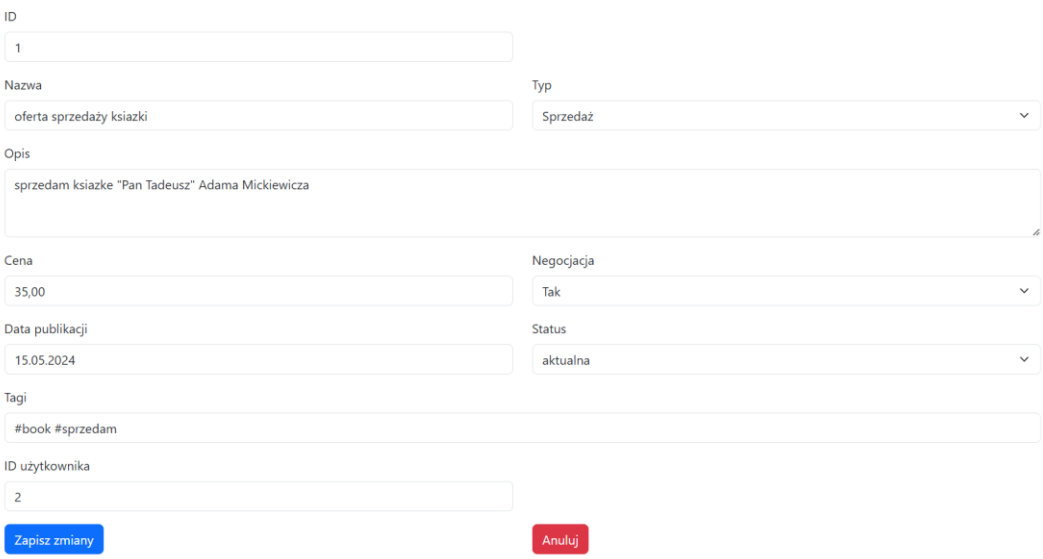


Rysunek 6.13. Formularz do dodawania ofert przez administratora.

#	Nazwa	Opis	Cena	Negocjacja	Typ	Status	Tagi	Id_użytkownika		
1	oferta sprzedaży ksiazki	sprzedam ksiazke "Pan Tadeusz" Adama Mickiewicza	35.00	tak	sprzedaz	aktualna	#book #sprzedam	2	Edytuj	Delete
2	oferta sprzedaży koszulki	sprzedam koszulkę	45.00	nie	sprzedaz	zakończona	#tshirt #sprzedam	3	Edytuj	Delete
3	oferta sprzedaży telefonu	sprzedam telefon marki Samsung	556.00	tak	sprzedaz	aktualna	#telefon #Samsung #sprzedam	4	Edytuj	Delete
4	oferta sprzedaży spódnicy	sprzedam spódnicę	30.00	tak	sprzedaz	zarezerwowana	#spódnica #sprzedam	5	Edytuj	Delete
5	oferta kupna podręcznika do biologi	kupię podrecznik do biologi wydawnictwa Nowa Era 8 klasa szkoły podstawowej	45.00	tak	kupno	aktualna	#podręcznik #biologia #szkoła	3	Edytuj	Delete
6	Oferta sprzedaży laptopa	Sprzedam laptopa marki Dell, model XPS 15, rocznik 2023.	2800.00	tak	sprzedaz	aktualna	#laptop #Dell #sprzedam	2	Edytuj	Delete
7	Oferta wynajmu mieszkania	Wynajmę 2-pokojowe mieszkanie w centrum miasta, umeblowane, od zaraz.	2000.00	nie	kupno	aktualna	#wynajem #mieszkanie #centrum	3	Edytuj	Delete
8	Oferta sprzedaży samochodu	Sprzedam samochód marki Toyota, model Corolla, rok produkcji 2022, przebieg 20 000 km.	45000.00	tak	sprzedaz	aktualna	#samochod #Toyota #sprzedam	4	Edytuj	Delete
9	Oferta sprzedaży roweru	Sprzedam rower górski marki Trek, kolor czarny, stan bardzo dobry.	1200.00	tak	sprzedaz	aktualna	#rower #Trek #sprzedam	5	Edytuj	Delete

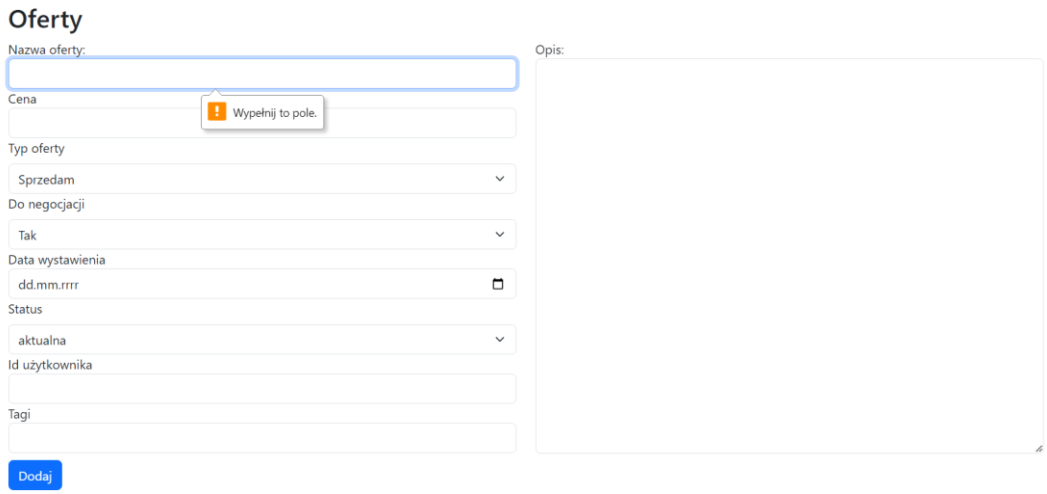
Rysunek 6.14. Tabelka wyświetlająca wszystkie oferty.

Rysunki 6.13 i 6.14 przedstawiają elementy widoku panelu do zarządzania ofertami przez administratora. W górnej części znajduje się formularz do dodawania oferty, a na dole tabela wyświetlająca wszystkie oferty. Każda z nich z boku ma dwie opcje „edytuj" i „usuń". Opcja edytuj przekierowuje na stronę odpowiadającą za edycję danej oferty (Która będzie opisywana w dalszej części dokumentacji), a opcja usuń usuwa daną ofertę z bazy danych.



Rysunek 6.15. Formularz do edycji oferty.

Rysunek 6.15 przedstawia formularz do edycji oferty. Jest on wypełniony aktualnymi danymi danej oferty. Pole „id" i „Id użytkownika" są tylko do odczytu.



Rysunek 6.16. Komunikat walidacji mówiący, że dane pole jest wymagane.

```php
public function store(CreateOfferRequest $request)
{
    if (Gate::denies('access-admin')) {
        abort(403);
    }

    try {
        $validatedData = $request->validated();
        Offer::create($validatedData);

        return redirect()->route('offerIndex');
    } catch (\Illuminate\Validation\ValidationException $e) {
        return redirect()->back()
            ->withErrors($e->errors())
            ->withInput();
    }
}
```

Rysunek 6.17. Funkcja dodawania oferty przez administratora.

```php
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class CreateOfferRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     */
    public function authorize(): bool
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array<string, \Illuminate\Contracts\Validation\ValidationRule|array<mixed>|string>
     */
    public function rules(): array
    {
        return [
            'name' => 'required|string|max:50',
            'description' => 'nullable|string|max:200',
            'price' => 'required|numeric|min:0|max:999999999',
            'negotiation' => 'required|boolean|in:0,1',
            'type' => 'required|string|in:sprzedam,kupie',
            'publication_date' => 'required|date|after_or_equal:2024-01-01',
            'status' => 'required|string|in:aktualna,zarezerwowana,zakończona',
            'tags' => 'nullable|string|max:100',
            'user_id' => 'required|exists:users,id',
        ];
    }
```

Rysunek 6.18 Zasady walidacji CreateOfferRequest.

Funkcja przedstawiona na rysunku 6.17 przyjmuje jako argument CreateOfferRequest w którym są zawarte reguły walidacji mówiące że:

- Pole name jest wymagane i musi być ciągiem znaków o maksymalnej długości 50 znaków.
- Pole description jest opcjonalne, ale jeśli jest obecne, musi być ciągiem znaków o maksymalnej długości 200 znaków.
- Pole price jest obowiązkowe i musi być liczbą, której wartość mieści się w przedziale od 0 do 999999999.
- Pole negotiation jest wymagane i musi być wartością logiczną (boolean), przyjmującą jedną z wartości 0 lub 1.
- Pole type jest obowiązkowe i musi być ciągiem znaków, który może przyjmować jedną z dwóch wartości: "sprzedam" lub "kupie".
- Pole publication_date jest wymagane i musi być datą, która jest równa lub późniejsza niż 1 stycznia 2024 roku.
- Pole status jest wymagane i musi być ciągiem znaków, który przyjmuje jedną z trzech wartości: "aktualna", "zarezerwowana" lub "zakończona".
- Pole tags jest opcjonalne, ale jeśli jest obecne, musi być ciągiem znaków o maksymalnej długości 100 znaków.

- Pole user_id jest wymagane i musi istnieć w tabeli users, co oznacza, że musi odnosić się do istniejącego użytkownika.

Na początku sprawdzane jest czy użytkownik wykonujący dodanie oferty ma uprawnienia administratora. Jeśli ich nie ma następuje wyrzucenie błędu 403. Następnie następuje sprawdzenie czy podane dane są poprawne. Jeśli są następuje dodanie nowej oferty i przekierowanie na stronę zarządzania ofertami, a jeśli nie są następuje wyrzucenie wyjątku ValidationException i następuje przekierowanie na stronę do zarządzania ofertami wraz z komunikatem o błędach.

```
public function show($id)
{
    if (Gate::denies('access-admin')) {
        abort(403);
    }
    $offer = Offer::findOrFail($id);
    return view('AdminPages.offerEditA', ['offer' => $offer]);
}
```

Rysunek 6.19. Funkcja do wyświetlania oferty o danym id do edycji.

Funkcja przedstawiona na rysunku 6.19 przyjmuje argument „id" czyli identyfikator oferty. Najpierw sprawdzane jest czy użytkownik ma uprawnienia administratora. Jeśli ma to następuje wyświetlenie formularza do edycji oferty wraz z jej danymi. Jeśli dana oferta nie istnieje zostaje wyrzucony błąd 404.

```php
public function update(UpdateOfferRequest $request, $id)
{
    if (Gate::denies('access-admin')) {
        abort(403);
    }

    try {
        $offer = Offer::findOrFail($id);
        $input = $request->all();
        $offer->update($input);

        return redirect()->route('offerIndex');
    } catch (\Illuminate\Validation\ValidationException $e) {
        return redirect()->back()
            ->withErrors($e->errors())
            ->withInput();
    }
}
```

Rysunek 6.20. Funkcja aktualizacji oferty.

Aplikacja > app > Http > Requests > UpdateOfferRequest.php > UpdateOfferRequest > rules

```php
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class UpdateOfferRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     */
    public function authorize(): bool
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array<string, \Illuminate\Contracts\Validation\ValidationRule|array<mixed>|string>
     */
    public function rules(): array
    {
        return [
            'name' => 'required|string|max:50',
            'description' => 'nullable|string|max:200',
            'price' => 'required|numeric|min:0|max:999999999',
            'negotiation' => 'required|boolean|in:0,1',
            'type' => 'required|string|in:sprzedam,kupie',
            'status' => 'required|string|in:aktualna,zarezerwowana,zakończona',
            'tags' => 'nullable|string|max:100',
        ];
    }
}

```

Rysunek 6.21. Reguły walidacji w UpdateOfferRequest.

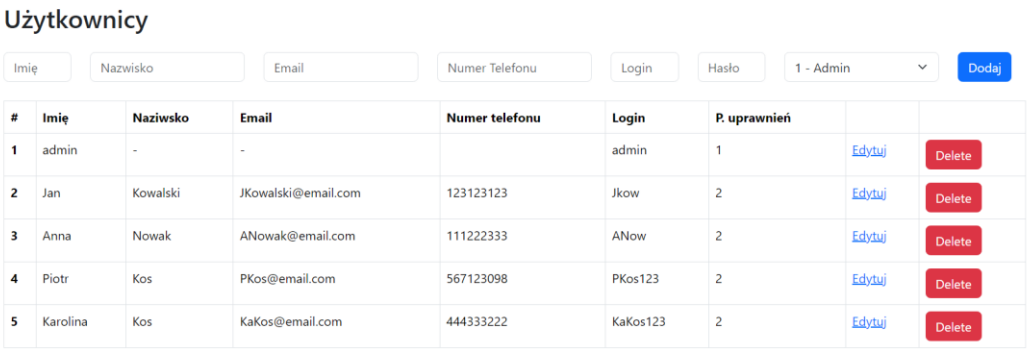
Na rysunku 6.20 przedstawiona jest funkcja update która przyjmuje dwa argumenty: UpdateOfferRequest oraz „id" czyli identyfikator oferty. Request ten posiada zasady walidacji analogiczne do CreateOfferRequest (Patrz strona 27). Na początku sprawdzane jest czy użytkownik ma uprawnienia administratora, jeśli nie ma zostaje wyrzucony błąd 403, a jeśli oferta nie istnieje zostaje wyrzucony błąd 404. Następnie zostają pobrane dane z żądania, zostaje wykonana aktualizacja oferty i następuje przekierowanie na stronę do zarządzania ofertami.

```
public function destroy($id)
{
    if (Gate::denies('access-admin')) {
        abort(403);
    }
    $offer = Offer::findOrFail($id);

    $offer->photo()->delete();

    $offer->delete();

    return $this->index();
}
```

Rysunek 6.22. Przedstawia funkcję usuwania oferty.

Funkcja przedstawiona na rysunku 6.22 odpowiada za usuwanie oferty z bazy danych. Przyjmuje ona jeden argument „id" czyli identyfikator oferty. Najpierw sprawdzane jest czy użytkownik posiada uprawnienia administratora. Jeśli nie ma jest wyrzucany błąd 403. Następnie następuje znalezienie oferty o danym id. Jeśli jej nie znaleziono zostaje wyrzucony błąd 404. Jeśli ją znaleziono to najpierw usuwaną są wszystkie zdjęcia przypisane do danej oferty, a później zostaje usunięta sama oferta. Na koniec następuje powrót na stronę do zarządzania ofertami.

Zarządzanie użytkownikami przez administratorów



Rysunek 6.23. Widok panelu do zarządzania użytkownikami przez administratora.

Rysunek 6.23 przedstawia panel do zarządzania użytkownikami przez administratora. Jego działanie jest analogiczne do panelu do zarządzania ofertami. (Patrz strona 24).

Dodawanie użytkowników do bazy danych również działa analogicznie i również posiada swoje zasady walidacji które są analogiczne do tych przy rejestracji. (Patrz strona 23).

Uprawnienia użytkowników

Użytkownik może mieć uprawnienia:

1 – uprawnienia administratora:

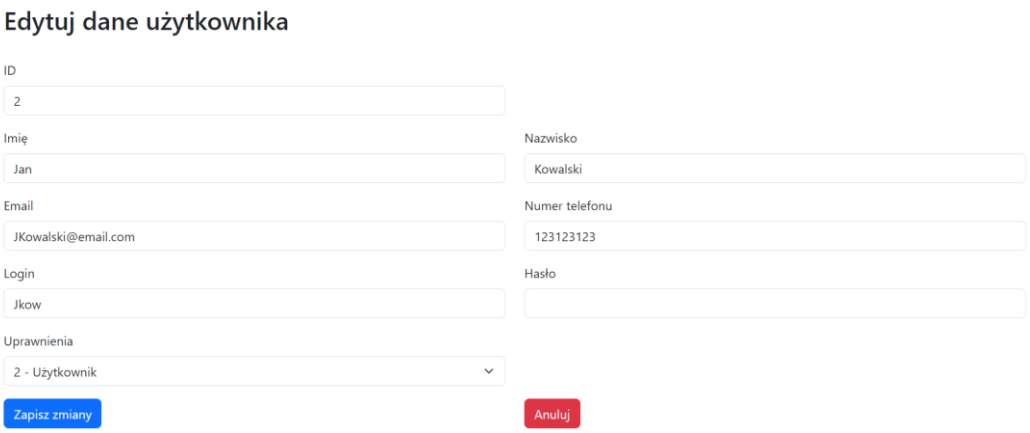
- Dostęp do panelu administratora
- Możliwość zarządzania wszystkimi ofertami z panelu administratora
- Możliwość zarządzania użytkownikami
- Możliwość dodawania ofert
- Możliwość zarządzania swoimi ofertami z panelu użytkownika
- Możliwość przeglądania ogólnie dostępnych zasobów

2 – uprawnienia użytkownika:

- Możliwość dodawania ofert
- Możliwość zarządzania swoimi ofertami
- Możliwość przeglądania ogólnie dostępnych zasobów

Użytkownik niezalogowany:

- Możliwość przeglądania ogólnie dostępnych zasobów



Rysunek 6.24. Formularz do edycji użytkownika.

Rysunek 6.24 przedstawia formularz do edycji użytkownika. Działa on analogicznie do formularza edycji ofert (Patrz strona 25) z różnicą, że pole hasło działa tak, że jak coś się wpisze w nie następuje zmiana hasła, a jak pozostawi je się puste to hasło się nie zmienia. Pole „id" jest tylko do odczytu.

```php
public function destroy($id)
{
    if (Gate::denies('access-admin')) {
        abort(403);
    }
    DB::transaction(function () use ($id) {
        $user = User::with('offers.photo')->findOrFail($id);
        foreach ($user->offers as $offer) {
            $offer->photo()->delete();

            $offer->delete();
        }

        $user->delete();
    });
    return $this->index();
}
```

Rysunek 6.25. Funkcja usuwania użytkownika.

Rysunek 6.25 przedstawia funkcję odpowiadającą za usuwania użytkownika z bazy danych. Przyjmuje ona jeden argument w postaci „id" czyli identyfikatora użytkownika. Najpierw ona sprawdza czy użytkownik ma uprawnienia administratora. Jeśli ich nie ma to jest wyrzucany błąd 403. Następnie rozpoczyna się transakcja która usuwa wszystkie zdjęcia przypisane do ofert użytkownika, a później

same oferty, a na końcu usuwany jest użytkownik. Później następuje wyświetlenie panelu do zarządzania użytkownikami.

Przeglądanie ogólnodostępnych zasobów

Rysunek 6.26. Fragment strony głównej z ogólnodostępnymi zasobami.

Panel ten wyświetla losowo 4 oferty. Posiada on dwie możliwości przeglądania ofert. Pierwsza to za pomocą wyszukiwarki a druga jest za pomocą przycisku „Pokaż więcej ofert”.

```php
public function MainPageindex(Request $request)
{
    $AllCount = Offer::count();
    session()->put('AllCount', $AllCount);

    $amount = $request->session()->get('amount', 4);

    $offers = Offer::with(['photo', 'user'])->inRandomOrder()->take($amount)->get();

    return view('index', ['offers' => $offers, 'AllCount' => $AllCount]);
}
```

Rysunek 6.27. Funkcja ładowania strony głównej.

Funkcja przedstawiona na rysunku 6.27 odpowiada za wyświetlanie się strony głównej oraz załadowanie na niej ofert. Przyjmuje ona jeden argument request. Najpierw pobiera ona ogólną liczbę ofert i umieszcza ją w sesji. Następnie z sesji pobierana jest zmienna „amount” i losuje tą ilość ofert ile ona wynosi do wyświetlenia. Jeśli w sesji nie ma tej zmiennej ustawionej to domyślnie pobierana jest wartość 4. Na końcu funkcja zwraca widok głównej strony z podaniem do niej wylosowanych ofert i ogólnej liczby wszystkich ofert.

```php
public function search(Request $request)
{
    $query = $request->input('query');

    $offers = Offer::where('name', 'like', "%$query%")
        ->orWhere('tags', 'like', "%$query%")
        ->get();

    $AllCount = Offer::count();

    return view('index', ['offers' => $offers, 'AllCount' => $AllCount]);
}
```

Rysunek 6.28. Funkcja wyszukiwarki ofert.

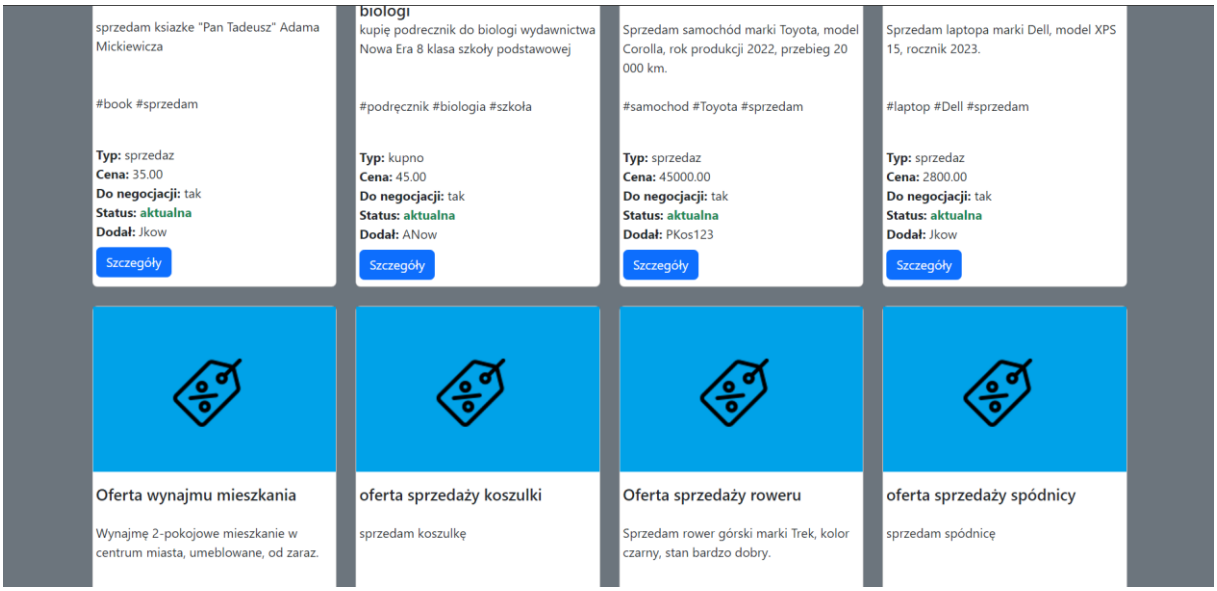
Rysunek 6.28 przedstawia funkcję która realizuje funkcje wyszukiwania ofert. Przyjmuje ona jeden argument czyli request. Najpierw ta funkcje z request pobiera dane z pola wyszukiwarki. Później wykonuje zapytanie do bazy danych wyszukujące oferty w których nazwach lub tagach zawarty jest ciąg znaków wpisany w wyszukiwarkę. Później pod zmienną allCount zapisywana jest ogólna liczba ofert i na końcu zwracany jest widok strony głównej z wynikami wyszukiwania. Przykład poniżej.

Rysunek 6.29. Wynik wyszukiwania „#book”

```php
public function showMoreOffers(Request $request)
{
    $AllCount = $request->session()->get('AllCount');
    if (!$request->session()->has('amount')) {
        $request->session()->put('amount', 4);
    }
    $count = $request->session()->get('amount');

    if ($count >= $AllCount) {
        return redirect()->route('mainPage')->with('status', 'Brak większej ilości ofert.');
    } else {
        $request->session()->increment('amount', 4);

        $offers = Offer::with('photo')->inRandomOrder()->take($count + 4)->get();

        return redirect()->route('mainPage', ['offers' => $offers]);
    }
}
```

Rysunek 6.30. Funkcja pokazująca więcej losowych ofert.

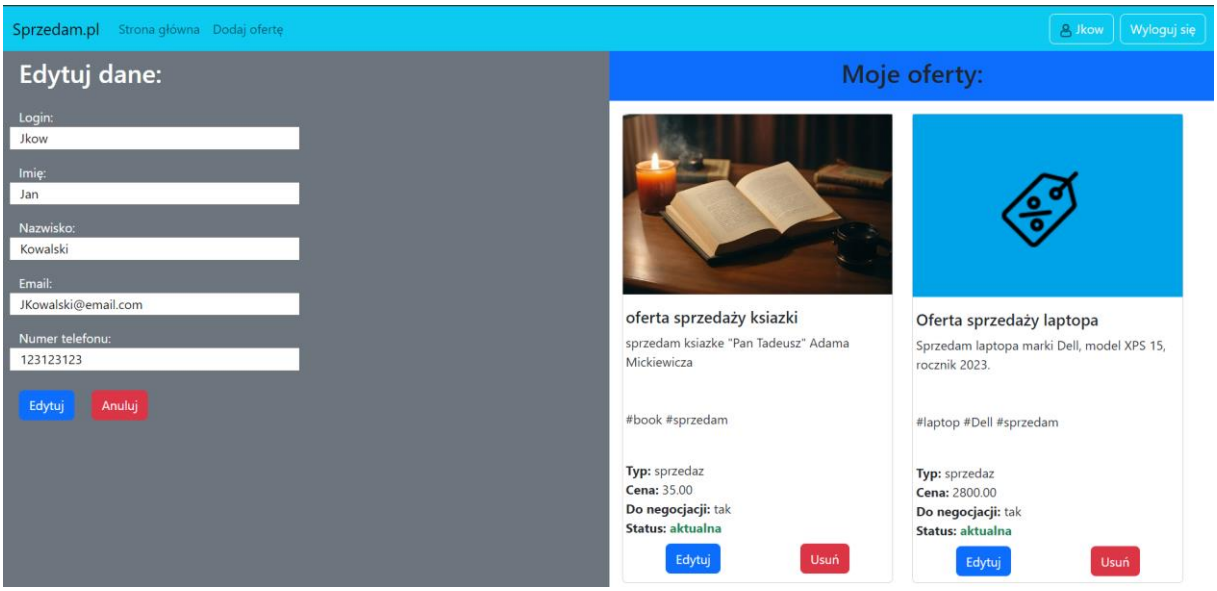
Funkcja pokazana na rysunku 6.30 odpowiada za pokazanie więcej losowych ofert na stronie. Przyjmuje ona jeden argument request. Najpierw z sesji pobiera się liczbę ogólną ofert w bazie danych. Następnie sprawdzane jest czy jest ustawiona zmienna „amount” odpowiadająca wylosowanej liczbie ofert. Jeśli jej nie ma jest ustawiana. Następnie do zmiennej „count” pobierana jest wartość „amount” z sesji. Później sprawdzane jest czy ilość ofert wyświetlonych jest większa bądź równa ogólnej ilości ofert. Jeśli jest to następuje powrót na stronę główną z wyświetleniem komunikatu, że nie ma więcej ofert do wyświetlenia. W przeciwnym wypadku zmienna „amount” w sesji jest zwiększana o 4. I następuje losowanie większej ilości ofert. Na koniec następuje przekierowanie na stronę główną z przekazaniem zmiennej „offers”.



Rysunek 6.31. Wynik wciśnięcia przycisku „Pokaż więcej ofert”.

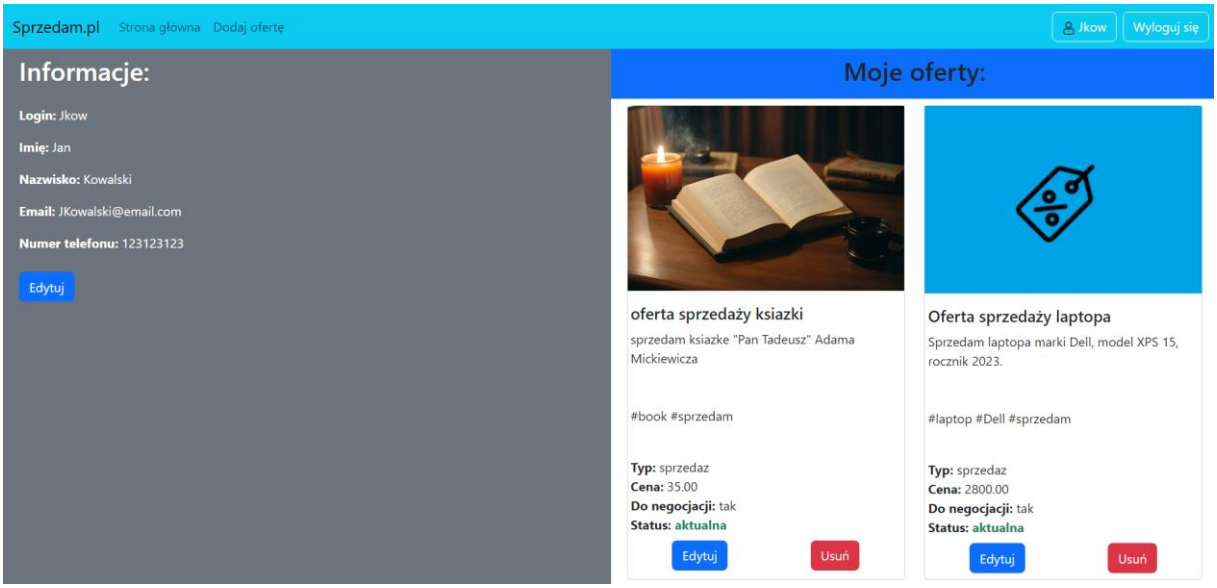
Rysunek 6.32. Komunikat o braku większej ilości ofert do wyświetlenia.

Zarządzanie swoimi danymi przez użytkownika



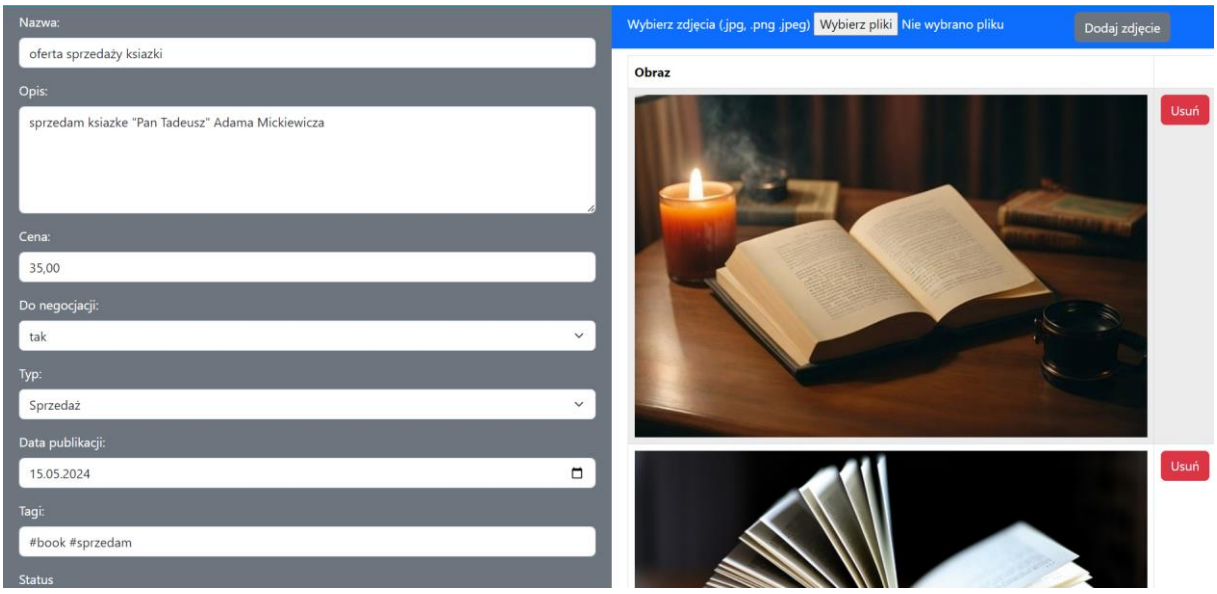
Rysunek 6.33. Panel edycji danych użytkownika.

Rysunek 6.33 przedstawia formularz edycji danych użytkownika. Działa on analogicznie jak formularz do dodawania użytkowników przez administratora z tą różnicą, że liczba danych jaką może edytować jest mniejsza. Pole login jest tylko do odczytu.



Rysunek 6.34. Panel użytkownika.

Jak było wspomniane w rozdziale opisującym warstwę użytkową (Patrz rozdział 4) po prawej stronie użytkownik ma panel do zarządzania ofertami. Każda oferta posiada opcje edycji i usunięcia.



Rysunek 6.35. Panel do edycji oferty.

Rysunek 6.35 przedstawia panel do edycji oferty. Jest on podzielony na dwie kolumny. Po lewej znajduje się formularz z aktualnymi danymi oferty które można edytować. Działa on analogicznie jak formularz do edycji ofert administratora z nałożonymi ograniczeniami. Reguły walidacji pozostają bez zmian.

Po prawej stronie za to znajduje się tabela ze zdjęciami które są przypisane do tej oferty. Po prawej stronie w każdym wierszu znajduje się przycisk usuń który usuwa dane zdjęcie.

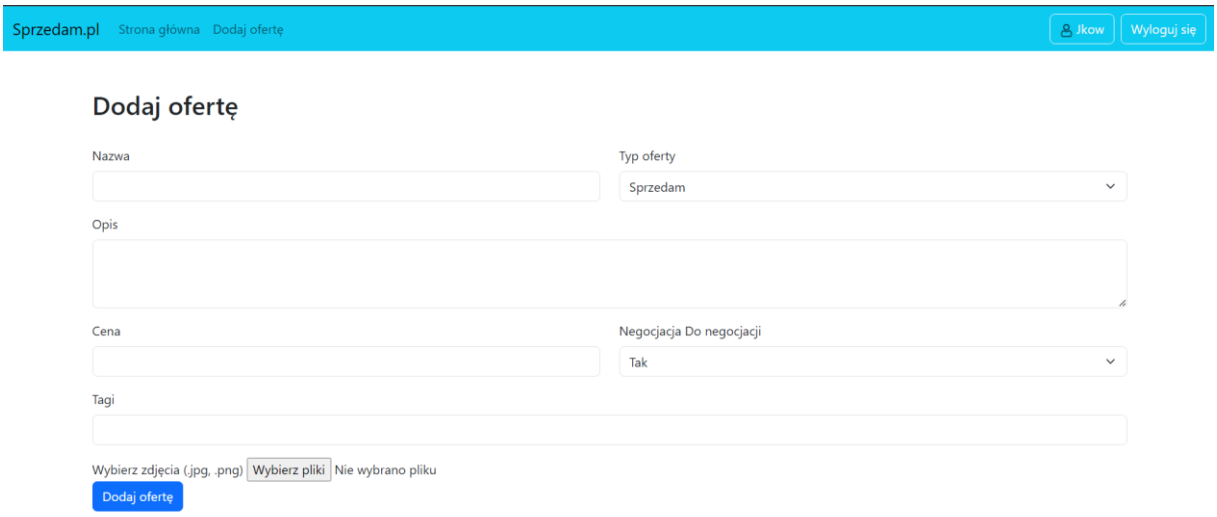
Nad tabelą znajduje się formularz do dodawania zdjęć. Przyjmuje on pliki o rozszerzeniach .jpg .png .jpeg. Może on przyjmować kilka zdjęć na raz.

```php
public function rules(): array
{
    return [
        'photos' => 'nullable|array',
        'photos.*' => 'image|mimes:jpeg,png,jpg|max:2048',
    ];
}
```

Rysunek 6.36. Reguły walidacji dla dodawania zdjęć.

Na rysunku 6.36 są przedstawione reguły walidacji dodawania zdjęć w formularzu. Oznaczają one że:

- Dodanie zdjęcia jest opcjonalne
- Można dodać kilka zdjęć naraz
- Każde zdjęcie musi być w formacie .jpg, .png lub .jpeg
- Maksymalny rozmiar zdjęcia to 2MB



Rysunek 6.37. Formularz do dodawania oferty przez użytkownika.

Rysunek 6.37 przedstawia formularz do dodania oferty przez użytkownika działa on analogicznie do formularza dodawania ofert przez administratora z kilkoma różnicami. Po pierwsze ma nałożone ograniczenia w postaci że użytkownikowi data się ustawia automatycznie i nie może edytować id użytkownika który dodał tą ofertę. Za to formularz posiada dodatkowo możliwość wgrania zdjęć do oferty, czego administrator nie ma.

```php
public function storeByUser(CreateOfferByUserRequest $request)
{
    if (Gate::denies('is-logged-in')) {
        abort(401);
    }

    try {
        $user = Auth::user();
        $input = $request->validated();
        $input['user_id'] = $user->id;
        $input['status'] = 'aktualna';
        $input['publication_date'] = Carbon::now();

        $offer = Offer::create($input);

        if ($request->hasFile('photos')) {
            foreach ($request->file('photos') as $photo) {
                $filename = $this->generateRandomString() . '.' . $photo->getClientOriginalExtension();
                $photo->storeAs('public/photos', $filename);

                Photo::create([
                    'file' => $filename,
                    'description' => '',
                    'offer_id' => $offer->id,
                ]);
            }
        }

        return redirect()->route('profile', $user);
    } catch (\Illuminate\Validation\ValidationException $e) {
        return redirect()->back()
            ->withErrors($e->errors())
            ->withInput();
    }
}
```

Rysunek 6.38. Funkcja dodająca ofertę przez użytkownika.

```php
private function generateRandomString($length = 10)
{
    return substr(str_shuffle(str_repeat($x = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ', ceil
    ($length / strlen($x)))), 1, $length);
}
```

Rysunek 6.39. Funkcja generująca losową nazwę pliku.

Na rysunkach 6.38 oraz 6.39 są przedstawione funkcje które odpowiadają za dodanie oferty przez użytkownika oraz wgranie pliku do storage z losowo wygenerowaną nazwą. Reguły walidacji są analogiczne jak dla dodawania oferty przez administratora oraz dodawania zdjęć.

Przyjmuje ona jako argument obiekt żądania CreateOfferByUserRequest. Na początku sprawdza, czy użytkownik jest zalogowany. Jeśli nie, wyrzuca błąd 401. Następnie, uzyskuje aktualnie zalogowanego użytkownika oraz sprawdza czy podane dane są poprawne. Uzupełnia dodatkowe pola takie jak identyfikator użytkownika, status oraz data publikacji, a następnie tworzy nową ofertę w bazie danych przy użyciu tych danych. Jeśli żądanie zawiera pliki zdjęć, funkcja przechodzi przez wszystkie przesłane zdjęcia, generuje losową nazwę pliku przy pomocy funkcji generateRandomString oraz zapisuje zdjęcia w określonej lokalizacji na serwerze. Po zapisaniu plików tworzy wpisy w bazie danych dla tych zdjęć, łącząc je z wcześniej utworzoną ofertą. Na końcu, w przypadku pomyślnego wykonania, następuje przekierowanie do profilu użytkownika. Jeśli podane dane były nie poprawne, użytkownik jest przekierowywany z powrotem z komunikatami o błędach oraz zachowaniem wprowadzonych danych.