|  |  |  |
| --- | --- | --- |
| Dokumentacja projektu | | **AI1** |
| **Autor** | Kamil Krukar, 122947 | **Ocena** |
| **Kierunek, rok** | Informatyka, II rok, st. stacjonarne (3,5-l) |  |
| **Temat projektu** | *Forum o muzeach i zabytkach* |

**Spis treści**

[**1.** **Wstęp** 2](#_Toc136971591)

[**2.** **Narzędzia i technologie** 3](#_Toc136971592)

[**3.** **Baza danych** 4](#_Toc136971593)

[**4.** **Projekt GUI** 13](#_Toc136971594)

[**5.** **Uruchomienie aplikacji** 15](#_Toc136971595)

[**6.** **Funkcjonalności aplikacji** 16](#_Toc136971596)

# **Wstęp**

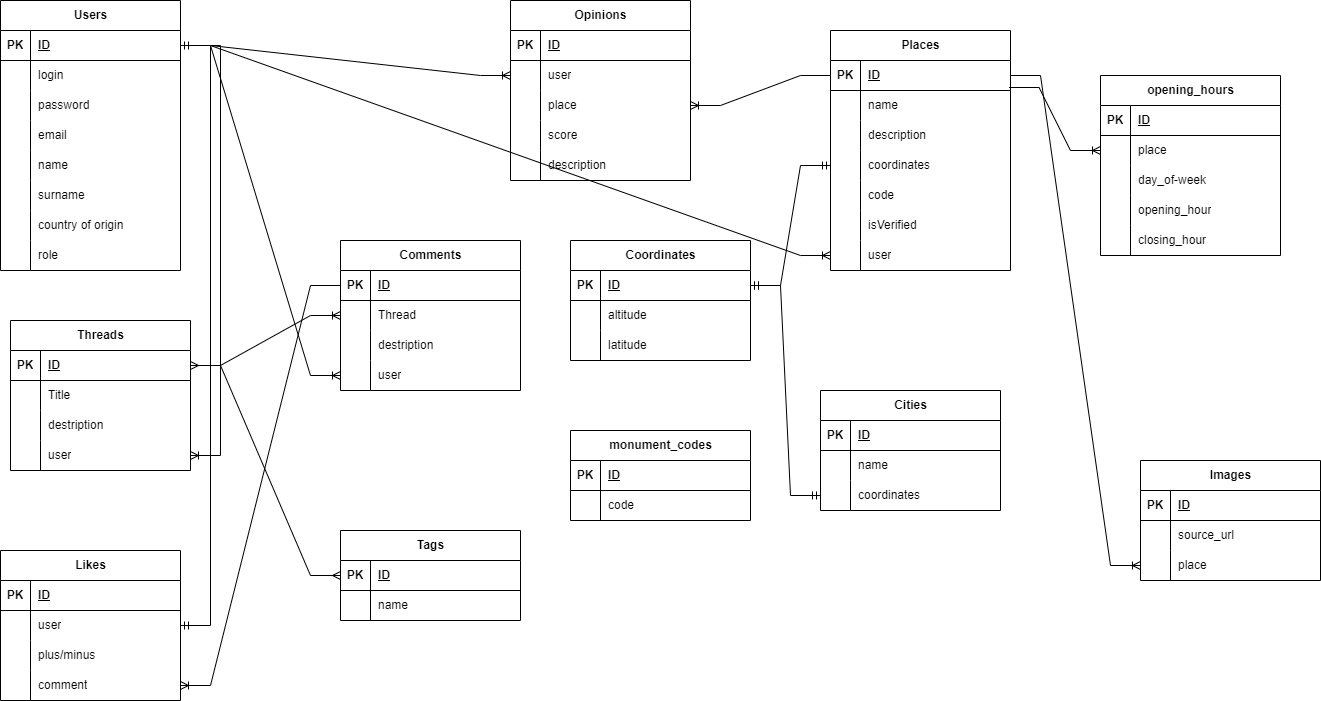
Forum o muzeach i zabytkach to aplikacja internetowa napisana głównie w języku php, przy pomocy frameworka Laravel. Funkcjonalność aplikacji dzieli się niejako na dwie główne części: Pierwsza to faktyczne miejsce na dyskusje, klasyczne forum internetowe, druga część to spis miejsc wartych odwiedzin takich jak muzea czy zabytki. Użytkownik ma możliwość dodawania otagowanego postu, następnie wszyscy pozostali użytkownicy mają możliwość wyszukania postu użytkownika, dodania komentarza po dym postem, a do dodanych komentarzy jest możliwość wyrażenia prostej opinii znanej powszechnie w internecie pod postacią systemu like/dislike. Druga część aplikacji umożliwia klientom przeglądanie wszystkich zabytków dostępnych na stronie, jak i wyszukiwanie miejsc bliskich szukanemu miastu. Po znalezieniu interesującego miejsca możemy zobaczyć jego godziny otwarcia zobaczyć zdjęcia dodane przez właściciela miejsca oraz ludzi wyrażających opinie o tym miejscu. Oczywiście jest też możliwość dodania opinii w popularnym formacie gwiazdkowym. Mamy też możliwość dodania własnego muzeum, albo ulubionego zabytku ze swojej okolicy za podaniem numeru NIP w przypadku muzea, a w przypadku zabytku, indywidualnego kodu zabytku. Program weryfikuje istnienie takiego miejsca i jeśli istnieje automatycznie na stronie nowo powstałego miejsca pojawaia się napis: „to miejsce zostało zweryfikowane”. W przypadku nowych zabytków i nowych muzeuów których program nie jest w stanie zweryfikować nadal istnieje możliwość nadania takiego statusu przez administratora, co więcej jest on w stanie odebrać ten status.

# **Narzędzia i technologie**

1. PHP 8.1 – Język programowania skryptowego do tworzenia stron i aplikacji internetowych. (Licencja: PHP License 3.01) ([PHP: Documentation](https://www.php.net/docs.php))
2. Laravel 10 – Framework dla PHP, umożliwiający szybsze i efektywniejsze tworzenie aplikacji internetowych. (Licencja: MIT) ([Installation - Laravel - The PHP Framework For Web Artisans](https://laravel.com/docs/10.x))
3. Laravel Starter Kit Breeze – Starter kit od Laravela, oferujący podstawowy szkielet aplikacji z autentykacją użytkowników. ([Starter Kits - Laravel - The PHP Framework For Web Artisans](https://laravel.com/docs/10.x/starter-kits))
4. Git 2.37.1.windows.1 – Narzędzie do kontroli wersji, umożliwiające efektywne zarządzanie i śledzenie zmian w kodzie. (Licencja: GNU General Public License version 2) ([Git (git-scm.com)](https://git-scm.com/))
5. Faker – Biblioteka PHP do generowania fałszywych danych do testowania. (Licencja: MIT) (<https://github.com/FakerPHP/Faker>)
6. Node.js v18.13.0 – Platforma do tworzenia aplikacji sieciowych na serwerze, używana do obsługi JavaScript po stronie serwera. (Licencja: MIT) ([Node.js (nodejs.org)](https://nodejs.org/en))
7. Rządowy rejestr zabytków – spis kodów polskich zabytków nieruchomych(<https://nid.pl/zasoby/rejestr-zabytkow-zasoby/>)
8. Bootstrap 5.0.0-beta2 – biblioteka front-end -owa z gotowymi elementami wyglądu strony([https://getbootstrap.com/docs/5.3/getting-started/introduction/)](https://getbootstrap.com/docs/5.3/getting-started/introduction/)

# **Baza danych**

**Schemat ERD:**



## Powiązania:

Baza danych dla projektu składa się z 11 głównych tabel stanowiących główną strukturę projektu, oraz 4 tabel będących składowymi frameworka Laravel, oraz 1 tabelą bez żadnych powiązań przechowująca kody zabytków.

Tabela **users:** przechowuje dane niezbędne do zalogowania i rejestracji, użytkownik może dodać wiele komentarzy, polubień, wątków oraz miejsc.

Tabela **threads:** przechowuje tytuł posta, opis, id użytkownika który go dodał. Wątek może mieć jednego właściciela, jest powiązany relacją wiele do wielu z tabelą tags. Jeden wątek może mieć wiele komentarzy.

Tabela **tags:** przechowuje tylko nazwę taga. Jest powiązana relacją wiele do wielu z tabelą threads.

Tabela **threads\_threads:** przechowuje klucze obce tabeli tags i threads realizując powiązanie. Wiele wątków może być otagowane np. militariami.

Tabela **likes:** przechowuje informacje o tym kto i do jakiego komentarza dodał polubienie oraz czy jest ono pozytywne czy negatywne. Użytkownik może polubić komentarz tylko raz.

Tabela **comments:** przechowuje informacje o tym do jakiego posta jest on dodany, jaki używtkownik go dodał, no i oczywiście treść. Jeden komentarz może mieć wiele polubień.

Tabela **opinons:** podobna do comments, przechowuje informacje o użytkowniku który dodał opinie, do jakiego miejsca, score, czyli wartość od 1 do 5 i treść opinii.

Tabela **coordinates:** przechowuje współrzędne geograficzne dla miejsc i miast.

Tabela **places:** ma w sobie informacje takie jak nazwa miejsca, opis, współrzędne geograficzne, godziny otwarcia, kod NIP lub zabytku, informacje o tym czy to miejsce jest zweryfikowane i opcjonalną informację o tym jaki użytkownik dodał to miejsce.

Tabela **cities:** przechowuje współrzędne geograficzne i nazwę miasta.

Tabela **opening\_hours:** ma kolumny przechowujące informacje dla miejsca o tym kiedy jest otwarcie a kiedy zamknięcie w poszczególnych dniach tygodnia.

Tabela **images:** przechowuje klucz obcy miejsca do którego zostało dodane zdjęcie oraz ścierzkę do pliku ze zdjęciem.

Tabela **monument\_codes:** przechowuje kody zabytków.

Pozostałe tabele to personal\_access\_tokens, password\_reset\_tokens, migrations, failed\_jobs – tabele te są również zawarte w bazie danych i są dostarczone przez framework Laravel.

## Konfiguracja:

Połączenie z bazą danych w pliku .env:

DB\_CONNECTION=mysql

DB\_HOST=127.0.0.1

DB\_PORT=3306

DB\_DATABASE=laravel

DB\_USERNAME=root

DB\_PASSWORD=

## Migracje:

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Na fragmencie ekranu przedstawionym powyżej widać wszystkie migracje niezbędne do powstania tabel służących funckjonalnościom aplikacji jak i frameworka laravel. Ułożyłem je w ten sposób żeby nie otrzymywać błędów o nieisniejących tabelach, w przypadku gdy realizowane jest powiazanie do nieistniejącej jeszcze tabeli. Polecenie php artisan migrate uruchamia migracje kolejno od góry tworząc na początku tabele bez kluczy obcych zapobiegając błędom.

**Przykładowa migracja:**

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

return new class extends Migration

{

    /\*\*

     \* Run the migrations.

     \*/

    public function up(): void

    {

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

            $table->id();

            $table->string('name')->unique();

            $table->text('description');

            $table->foreignId('coordinates\_id')->constrained('coordinates');

            $table->string('code')->unique();

            $table->boolean('verified');

            $table->foreignId('user\_id')->nullable()->default(null)->constrained('users');

            $table->timestamps();

        });

    }

    /\*\*

     \* Reverse the migrations.

     \*/

    public function down(): void

    {

        Schema::dropIfExists('places');

    }

};

W tym fragmencie kodu migracji tworzona jest tabela "places" w bazie danych. Poniżej znajduje się krótkie wyjaśnienie każdej kolumny:

* $table->id() - Tworzy kolumnę "id" jako unikalny identyfikator dla każdego rekordu w tabeli "places". Ta kolumna jest automatycznie inkrementowana i służy jako klucz główny tabeli.
* $table->string('name')->unique() - Tworzy kolumnę "name" jako tekstową kolumnę, w której przechowywane będą nazwy miejsc. Metoda unique() definiuje, że wartości w tej kolumnie muszą być unikalne.
* $table->text('description') - Tworzy kolumnę "description" jako tekstową kolumnę, w której przechowywane będą opisy miejsc.
* $table->foreignId('coordinates\_id')->constrained('coordinates') - Tworzy kolumnę "coordinates\_id" jako klucz obcy (foreign key), który jest powiązany z kluczem głównym tabeli "coordinates". Ta kolumna przechowuje identyfikator koordynatów związanych z danym miejscem.
* $table->string('code')->unique() - Tworzy kolumnę "code" jako tekstową kolumnę, w której przechowywane będą kody miejsc. Metoda unique() definiuje, że wartości w tej kolumnie muszą być unikalne.
* $table->boolean('verified') - Tworzy kolumnę "verified" jako logiczną kolumnę, która przechowuje informację o tym, czy dane miejsce zostało zweryfikowane.
* $table->foreignId('user\_id')->nullable()->default(null)->constrained('users') - Tworzy kolumnę "user\_id" jako klucz obcy (foreign key), który jest powiązany z kluczem głównym tabeli "users". Ta kolumna przechowuje identyfikator użytkownika, który utworzył dane miejsce. Metoda nullable() definiuje, że ta kolumna może przyjmować wartość null, a metoda default(null) ustawia domyślną wartość tej kolumny na null.
* $table->timestamps() - Tworzy kolumny "created\_at" i "updated\_at", które automatycznie przechowują datę i czas utworzenia oraz ostatniej aktualizacji rekordu.

W skrócie, powyższy kod migracji tworzy tabelę "places" z różnymi kolumnami, które przechowują informacje o nazwie, opisie, koordynatach, kodzie, weryfikacji oraz informacje o użytkowniku, który utworzył dane miejsce.

## Mapowanie relacji w modelach:

W projekcie zastosowano mapowanie relacji przy użyciu ORM Eloquent.

Wykorzystywane są funkcje takie jak:

->belongsToMany

->belongsTo

->hasOne

->hasMany

## Przykład Modelu i powiązań:

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

use App\Models\Comments;

class Threads extends Model

{

    use HasFactory;

    protected $fillable = ['Title', 'user\_id', "description"];

    public function user()

    {

        return $this->belongsTo(User::class);

    }

    public function tags()

    {

        return $this->belongsToMany(Tags::class, 'tags\_threads', 'thread\_id', 'tag\_id');

    }

    public function comments()

    {

        return $this->hasMany(Comments::class, 'thread\_id');

    }

}

1. **Powiązanie "user":**

belongsTo(User::class): Definiuje, że ten wątek należy do jednego użytkownika.

Powiązanie typu "belongsTo" oznacza, że wątek jest zależny od innego modelu (w tym przypadku od modelu "User").

Powiązanie jest ustalane na podstawie klucza obcego "user\_id" w tabeli "threads" i klucza głównego "id" w tabeli "users".

1. **Powiązanie "tags":**

belongsToMany(Tags::class, 'tags\_threads', 'thread\_id', 'tag\_id'): Definiuje, że ten wątek może być powiązany z wieloma tagami.

Powiązanie typu "belongsToMany" oznacza, że wątek może mieć wiele tagów, a tagi mogą być przypisane do wielu wątków.

Powiązanie jest realizowane poprzez tabelę pośredniczącą "tags\_threads", która przechowuje informacje o powiązaniu między tagami a wątkami.

Klucz obcy "thread\_id" w tabeli "tags\_threads" wskazuje na identyfikator wątku, a klucz obcy "tag\_id" wskazuje na identyfikator tagu.

1. **Powiązanie "comments":**

hasMany(Comments::class, 'thread\_id'): Definiuje, że ten wątek może mieć wiele komentarzy.

Powiązanie typu "hasMany" oznacza, że wątek może mieć wiele powiązanych komentarzy.

Powiązanie jest ustalane na podstawie klucza obcego "thread\_id" w tabeli "comments" i klucza głównego "id" w tabeli "threads".

## Seedy:

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Większość seederów jest zrealizowanych przy pomocy fakera tak jak w przypadku poniżej:

class CommentsSeeder extends Seeder

{

    public function run()

    {

        $faker = Faker::create();

        $threads = Threads::pluck('id')->toArray();

        $users = User::pluck('id')->toArray();

        for ($i = 1; $i <= 50; $i++) {

            $comment = Comments::create([

                'thread\_id' => $faker->randomElement($threads),

                'description' => $faker->paragraph,

                'user\_id' => $faker->randomElement($users),

            ]);

        }

    }

}

Seeder dodaje 50 komentarzy do istniejących threadów od istniejących użytkowników.

'thread\_id' => $faker->randomElement($threads) – losuje do jakiego wątku będzie należał komentarz.

'description' => $faker->paragraph – tworzy przykładowy paragraf lorem ipsum.

Inne seedery są ustawione „na sztywno”:

    public function run()

    {

        $faker = Faker::create();

        $museumsData = [

            [

                'name' => 'Wieliczka Salt Mine',

                'latitude' => 49.9834,

                'longitude' => 20.0523,

            ],

            [

                'name' => 'Auschwitz-Birkenau State Museum',

                'latitude' => 50.0349,

                'longitude' => 19.1796,

            ],

//(…)

            [

                'name' => 'Wrocław Market Square',

                'latitude' => 51.1107,

                'longitude' => 17.0326,

            ],

        ];

        foreach ($museumsData as $museumData) {

            $coordinates = Coordinates::create([

                'latitude' => $museumData['latitude'],

                'longitude' => $museumData['longitude'],

            ]);

            Places::create([

                'name' => $museumData['name'],

                'description' => $faker->text(500),

                'coordinates\_id' => $coordinates->id,

                'code' => Str::random(6),

                'verified' => 0

            ]);

        }

    }

W pierwszej kolejności dodajemy współrzędne geograficzne miejsca a potem uzupełniamy pola tabeli place i przypisujemy te współrzędne.

Ostatnią formą seedera jest czytanie z pliku CSV:

    public function run()

    {

        $csvFile = database\_path('seeders/zestawienie\_zabytki\_nieruchome.csv');

        $file = fopen($csvFile, 'r');

        if ($file !== false) {

            $header = fgetcsv($file); // Read the header row

            $inspireIds = [];

            // Find the index of the "code" column

            $inspireIdIndex = array\_search('code', $header);

            while (($row = fgetcsv($file)) !== false) {

                $inspireId = $row[$inspireIdIndex];

                $inspireIds[] = $inspireId;

            }

            fclose($file);

            foreach ($inspireIds as $inspireId) {

                DB::table('monument\_codes')->insert([

                    'code' => $inspireId,

                    // Add other columns if needed

                ]);

            }

        }

    }

Otwieramy i odczytujemy plik CSV, szukamy kolumny o nazwie code i pobieramy wszystkie rekordy oraz zapisujemy do bazy danych.

# **Projekt GUI**

Projekt posiada jednolity design, natomiast wykożystuje wiele elementów bootstrapa takich jak tabele czy karty. Jest również w pełni resposywna i działa poprawnie na urządzeniach mobilnych.

## Strona główna:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

## Widok miejsc:

Obraz zawierający tekst, oprogramowanie, Czcionka, numer

Opis wygenerowany automatycznie

## Widok postu:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Strona internetowa

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

# **Uruchomienie aplikacji**

1. Uruchomienie projektu wymaga PHP, Composer, Node.js i NPM. Instrukcja instalacji tych narzędzi znajduje się w dokumentacji Laravel:

[Installation - Laravel - The PHP Framework For Web Artisans](https://laravel.com/docs/10.x" \l "your-first-laravel-project)

1. Wypakuj katalog z projektem po czym przejdź do wypakowanego katalogu używając terminala (Win + R, ‘cmd.exe’), w tym celu użyj komendy cd „sciezka\_do\_katalogu”
2. Uruchom polecenie composer install
3. Uruchom polecenie npm install
4. Uruchom polecenie php artisan storage:link
5. Uruchom polecenie php artisan migrate --seed
6. Uruchom polecenie php artisan serve
7. Uruchom przeglądarkę i przejdź do adresu URL serwera deweloperskiego (domyślnie jest to <http://127.0.0.1:8000>). Adres ten wyświetla się w terminalu.

**Dane logowania administratora:**

**Email:** [**admin@example.com**](mailto:admin@example.com)

**Hasło: adminpassword**

# **Funkcjonalności aplikacji**

Żeby lepiej zrozumieć działanie aplikacji podzielę funkcjonalności na 3 części:

1. Funkcjonalności gościa.
2. Funkcjonalności usera, gdzie funkcjonalności usera to wszystkie wymienione w pierwszym punkcie, oraz te dodane w punkcie drugim.
3. Funkcjonalności admina, gdzie funkcjonalności admina to wszystkie wymienione w drugim punkcie, oraz te dodane w punkcie trzecim.

## 1.Funkcjonalności gościa

## Logowanie się i rejestracja:

Te funkcjonalności zostały dodane automatycznie przez technologię breeze, widoki zostały ostylizowane adekwatnie do wyglądu aplikacji:

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, linia, Czcionka

Opis wygenerowany automatycznie

## Filtracja i przeglądanie postów za pomocą searchbara z autocompleate, bądź panelu z najpopularniejszymi tagami.

Obraz zawierający tekst, oprogramowanie, Oprogramowanie multimedialne, zrzut ekranu

Opis wygenerowany automatycznie

Na zrzucie powyżej widać w jaki sposób user może znaleźć wądki które go interesują. Tagi są ułożone malejąco ze względu na ilość postów jakie dotyczą tego taga, po kliknięciu zostajemy przeniesieni do widoku wszystkich wątków oznaczonych klikniętym tagiem.

        // wyszukanie 12 najpopularniejszych tagów

        $tagsWithMostThreads = Tags::withCount('threads')

            ->orderBy('threads\_count', 'desc')

            ->limit(12)

            ->get();

        $cities = Cities::all();

        //rozpoznanie czy id jest podane czy nie w przypadku klikniecia taga

        if ($id) {

            $selectedTag = Tags::findOrFail($id);

            $threads = $selectedTag->threads;

        } else {

            $selectedTag = null;

            $threads = Threads::latest()->get();

        }

        //... i w przypadku searchbara

        $selectedTag = $request->input('tag');

        $tag = $tags->firstWhere('name', $selectedTag);

Żeby mój autocompleate działał samodzielnie i nie powielać domyślnego autocompleate musiałem go wyłączyć:

<form action="{{ route('tags.search') }}" method="GET" class="d-flex justify-content-end" autocomplete="off" >

Kod Javascript odpowiedzialny za edytowany autocompleate prezentuje się tak:

    // Pobieranie nazw tagów i miast

    var cities = {!! json\_encode($cities->pluck('name')) !!};

    var countries = {!! json\_encode($tag->pluck('name')) !!};

    // Funkcja do obsługi funkcjonalności automatycznego uzupełniania pól tekstowych

    function autocomplete(inp, arr) {

        var currentFocus;

        // Wywołanie funkcji po wpisaniu tekstu w pole tekstowe

        inp.addEventListener("input", function (e) {

            var val = this.value;

            closeAllLists();

            if (!val) {

                return false;

            }

            currentFocus = -1;

            var a = document.createElement("DIV");

            a.setAttribute("id", this.id + "autocomplete-list");

            a.setAttribute("class", "autocomplete-items");

            this.parentNode.appendChild(a);

            // Iteracja przez elementy i sprawdzanie dopasowań

            for (var i = 0; i < arr.length; i++) {

                if (arr[i].substr(0, val.length).toUpperCase() == val.toUpperCase()) {

                    var b = document.createElement("DIV");

                    b.innerHTML = "<strong>" + arr[i].substr(0, val.length) + "</strong>";

                    b.innerHTML += arr[i].substr(val.length);

                    b.innerHTML += "<input type='hidden' value='" + arr[i] + "'>";

                    // Obsługa kliknięcia na element

                    b.addEventListener("click", function (e) {

                        inp.value = this.getElementsByTagName("input")[0].value;

                        closeAllLists();

                    });

                    a.appendChild(b);

                }

            }

        });

        // Obsługa klawiatury

        inp.addEventListener("keydown", function (e) {

            var x = document.getElementById(this.id + "autocomplete-list");

            if (x) x = x.getElementsByTagName("div");

            if (e.keyCode == 40) {

                currentFocus++;

                addActive(x);

            } else if (e.keyCode == 38) {

                currentFocus--;

                addActive(x);

            } else if (e.keyCode == 13) {

                e.preventDefault();

                if (currentFocus > -1) {

                    if (x) x[currentFocus].click();

                }

            }

        });

        // Pokazuje pasujące elementy

        function addActive(x) {

            if (!x) return false;

            removeActive(x);

            if (currentFocus >= x.length) currentFocus = 0;

            if (currentFocus < 0) currentFocus = x.length - 1;

            x[currentFocus].classList.add("autocomplete-active");

        }

        // Usuwa aktywność z elementów

        function removeActive(x) {

            for (var i = 0; i < x.length; i++) {

                x[i].classList.remove("autocomplete-active");

            }

        }

        // Zamyka wszystkie listy z uzupełnieniami

        function closeAllLists(elmnt) {

            var x = document.getElementsByClassName("autocomplete-items");

            for (var i = 0; i < x.length; i++) {

                if (elmnt != x[i] && elmnt != inp) {

                    x[i].parentNode.removeChild(x[i]);

                }

            }

        }

        // Zamyka listy po kliknięciu na inne elementy

        document.addEventListener("click", function (e) {

            closeAllLists(e.target);

        });

    }

    // Inicjalizacja funkcjonalności autocomplete dla pól tekstowych

    autocomplete(document.getElementById("cityInput"), cities);

    autocomplete(document.getElementById("tagInput"), countries);

Zaprezentowany kod dodaje autocompleate do obu pól tekstowych, miast i tagów.

## Przeglądanie konkretnego posta i komentarzy:

Obraz zawierający tekst, zrzut ekranu, Czcionka, dokument

Opis wygenerowany automatycznie

Od góry wyświetlają się:

* Tagi dotyczące tego posta
* Tytuł posta
* Kto go dodał
* Jaka jest jego treść
* Komentarze
  + Kto dodał komentarz
  + Kiedy
  + Jaka jest treść
  + Ile ma likeów

Funkcja przekazujące wszystkie te informacje do widoku:

    public function index($id)

    {

        $thread = Threads::findOrFail($id);

        $comments = $thread->comments;

        $user = auth()->user();

        $likes = Likes::all();

        $likesJson = $likes->toJson();

        $tags = $thread->tags;

        $tag = Tags::all();

        $cities = Cities::all();

        return view('threads.index', compact('thread', 'comments', 'likes', 'likesJson', 'tags', 'cities','tag'));

    }

**Przykładowy sposób wyświetlania(thread):**

        <div class="card">

            <div class="card-body">

                <h5 class="card-title">{{ $thread->title }}</h5>

                <h6 class="card-subtitle mb-2 text-muted">Posted by: {{ $thread->user->name }}</h6>

                <p class="card-text">{{ $thread->description }}</p>

            </div>

        </div>

## Przeglądanie wszystkich miejsc lub poprzez wpisanie do searchbara nazwy miasta.

Obraz zawierający tekst, oprogramowanie, Czcionka, Strona internetowa

Opis wygenerowany automatycznie

public function search(Request $request)

    {

// Funkcja obliczająca odległość między dwoma punktami na podstawie ich współrzędnych geograficznych

        function calculateDistance($lat1, $lon1, $lat2, $lon2, $unit = 'km')

        {

// $lat1, $lon1 - szerokość i długość geograficzna pierwszego punktu

// $lat2, $lon2 - szerokość i długość geograficzna drugiego punktu

// $unit - jednostka odległości (domyślnie kilometry)

// Promień Ziemi w kilometrach lub milach

            $earthRadius = ($unit === 'km') ? 6371 : 3959;

            $latDelta = deg2rad($lat2 - $lat1);

            $lonDelta = deg2rad($lon2 - $lon1);

            $a = sin($latDelta / 2) \* sin($latDelta / 2) + cos(deg2rad($lat1)) \* cos(deg2rad($lat2)) \* sin($lonDelta / 2) \* sin($lonDelta / 2);

            $c = 2 \* atan2(sqrt($a), sqrt(1 - $a));

            $distance = $earthRadius \* $c;

            return $distance;

        }

// Pobieranie wartości pola wejściowego 'search' z żądania użytkownika

        $search = $request->input('search');

// Wyszukiwanie miasta na podstawie podanej nazwy

        $city = Cities::where('name', $search)->first();

// Sprawdzenie, czy miasto zostało znalezione. W przeciwnym razie przekierowanie z błędem

        if (!$city) {

            return redirect()->back()->with('error', 'Miasto nie znalezione');

        }

// Pobranie współrzędnych geograficznych miasta

        $cityCoordinates = $city->coordinates;

        $cityLatitude = $cityCoordinates->latitude;

        $cityLongitude = $cityCoordinates->longitude;

// Inicjalizacja pustej tablicy na przechowywanie miejsc pasujących do kryteriów wyszukiwania

        $filteredPlaces = [];

// Pobranie wszystkich miejsc

        $places = Places::all();

// Iteracja przez miejsca i obliczenie odległości

        foreach ($places as $place) {

            $placeCoordinates = $place->coordinates;

            $placeLatitude = $placeCoordinates->latitude;

            $placeLongitude = $placeCoordinates->longitude;

// Obliczenie odległości między miastem a miejscem przy użyciu funkcji calculateDistance()

            $distance = calculateDistance($cityLatitude, $cityLongitude, $placeLatitude, $placeLongitude);

// Sprawdzenie, czy odległość mieści się w zakresie 50 (w jednostce odległości, np. kilometrach)

            if ($distance <= 50) {

// Dodanie miejsca do tablicy filteredPlaces, jeśli spełnia kryterium odległości

                $filteredPlaces[] = $place;

            }

        }

// Pobranie wszystkich tagów i miast

        $tag = Tags::all();

        $cities = Cities::all();

        return view('cities.index', compact('filteredPlaces', 'city', 'tag', 'cities'));

    }

Matematycznie rzecz biorąc odległość jest wyliczana ze wzoru:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

W momencie wyszukania miasta pobierane są koordynaty szukanego miasta a następnie wyświetlane są wszystkie miejsca w promieniu 50 km od szukanego miasta.

## Wyświetlanie konkretnego miejsca przeglądanie iformacji, dodanych zdjęć i opinii:

Obraz zawierający tekst, zrzut ekranu, woda, oprogramowanie

Opis wygenerowany automatycznie

Zdjęcia wyświetlane są w karuzeli, podane godziny wyświetlamy niepodane pomijamy.

## 2.Funkcjonalności usera

## Polubienia komentarzy

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, Czcionka, zrzut ekranu, linia

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

Zaimplementowałem to według schematu:

Policz i pokaż różnicę pozytywnych likeów i negatywnych likeów.

Sprawdź czy obecnie zalogowany user zostawil like pod którymś komentarzem otwieranego posta. Jeśli TAK:

TAK: Sprawdź czy jest pozytywny. Jeśli TAK:

TAK: ustaw status guzika like na „liked”

NIE: ustaw status guzika diske na „disliked”

Jeśli ktoś kliknie przycisk Like:

Sprawdź czy przycisk like jest już wciśnięty

Jeśli jest:

Usuń z bazy danych rekord tego like-a

Aktualizuj ilość polubien

Zmien stan guzika

Jeśli nie jest:

Sprawdź czy guzik disklike jest kliknięty

Jeśli jest:

Usuń negatywny like

Aktualizuj ilość polubien

Dodaj nowy pozytywny like

Zmien stan obu guzików

Jeśli nie jest:

Dodaj like

Aktualizuj ilość polubien

Zmien stan guzika

Część z rzeczy jest realizowana po stronie serwera, część po stronie klienta, ale najczęściej po obu na raz.

Część kodu po stronie serwera:

public function like(Request $request)

{

    $commentId = $request->input('commentId');

    $userId = auth()->user()->id;

    // Sprawdzenie, czy użytkownik już polubił lub niepolubił komentarz

    $like = Likes::where('comment\_id', $commentId)

        ->where('user\_id', $userId)

        ->first();

    if ($like) {

        // Użytkownik już polubił lub niepolubił komentarz, więc aktualizujemy polubienie

        $like->isPositive = true;

        $like->save();

    } else {

        // Użytkownik nie polubił ani nie niepolubił komentarza, więc tworzymy nowe polubienie

        $like = new Likes();

        $like->comment\_id = $commentId;

        $like->user\_id = $userId;

        $like->isPositive = true;

        $like->save();

    }

    // Zwracamy zaktualizowaną liczbę polubień dla konkretnego komentarza

    $likeCount = Comments::find($commentId)

        ->like()

        ->where('isPositive', true)

        ->count();

    return response()->json([

        'likeCount' => $likeCount,

    ]);

}

public function unlike(Request $request)

{

    $commentId = $request->input('commentId');

    $userId = auth()->user()->id;

    // Znajdujemy rekord polubienia komentarza przez użytkownika

    $like = Likes::where('comment\_id', $commentId)

        ->where('user\_id', $userId)

        ->first();

    if ($like) {

        // Usuwamy polubienie

        $like->delete();

    }

    // Zwracamy zaktualizowaną liczbę polubień dla konkretnego komentarza

    $likeCount = Comments::find($commentId)

        ->like()

        ->where('isPositive', true)

        ->count();

    return response()->json([

        'likeCount' => $likeCount,

    ]);

}

Dla dislikeów analogicznie.

**Część po stronie klienta:**

const likeButtons = document.querySelectorAll('.like-btn');

const dislikeButtons = document.querySelectorAll('.dislike-btn');

const commentCards = document.querySelectorAll('.comment-cards .card');

function handleLikeButtonClick(event) {

    const commentId = event.target.dataset.commentId;

    const liked = event.target.dataset.liked === 'true';

    const dislikeButton = event.target.parentElement.querySelector('.dislike-btn');

    const disliked = dislikeButton.dataset.disliked === 'true';

    const likesContainer = event.target.parentElement.querySelector('.likes-container span');

    let likePoints = parseInt(likesContainer.textContent);

    if (!liked) {

        // Wysłanie żądania AJAX w celu zwiększenia liczby polubień dla komentarza

        axios

            .post('/comments/like', { commentId })

            .then((response) => {

                // Po udanej odpowiedzi, zaktualizuj liczbę polubień i zmień styl przycisku

                event.target.classList.remove('btn-primary');

                event.target.classList.add('btn-success');

                event.target.innerText = 'Polubiono';

                likePoints++;

                likesContainer.textContent = likePoints;

                event.target.dataset.liked = 'true';

                if (disliked) {

                    dislikeButton.classList.remove('btn-danger');

                    dislikeButton.classList.add('btn-secondary');

                    dislikeButton.innerText = 'Niepolubione';

                    likePoints++;

                    likesContainer.textContent = likePoints;

                    dislikeButton.dataset.disliked = 'false';

                }

            })

            .catch((error) => {

                console.log(error);

            });

    } else {

        // Wysłanie żądania AJAX w celu zmniejszenia liczby polubień dla komentarza

        axios

            .post('/comments/unlike', { commentId })

            .then((response) => {

                // Po udanej odpowiedzi, zaktualizuj liczbę polubień i zmień styl przycisku

                event.target.classList.remove('btn-success');

                event.target.classList.add('btn-primary');

                event.target.innerText = 'Polub';

                likePoints--;

                likesContainer.textContent = likePoints;

                event.target.dataset.liked = 'false';

            })

            .catch((error) => {

                console.log(error);

            });

    }

}

**Dodawanie komentarzy:**

Jeśli jesteśmy zalogowani wyświetla się prosty panel:

Obraz zawierający tekst, biały, linia, zrzut ekranu

Opis wygenerowany automatycznie

Po wypełnieniu możemy dodać komentarz przyciskiem submit.

**Kod wykonujący insert:**

    public function store(Request $request)

    {

        $request->validate([

            'description' => 'required|max:1000'

        ]);

        $comment = new Comments();

        $comment->thread\_id = $request->input('thread\_id');

        $comment->user\_id = Auth::id();

        $comment->description = $request->input('description');

        $comment->save();

        return redirect()->back()->with('success', 'Comment added successfully');

    }

## Dodawanie nowego posta:

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Po kliknięciu add thread wyświetla się panel gdzie możemy uzupełnić niezbędne dane do dodanie posta, gdy nie istnieje wymieniony w polu tag zostaje utworzony i automatycznie zostaje przypisany do tego watku.

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

public function store(Request $request)

{

    // Walidacja danych formularza

    $validatedData = $request->validate([

        'title' => 'required',

        'description' => 'required',

        'tags' => 'required',

    ]);

    // Przetwarzanie tagów

    $tagNames = explode(',', $validatedData['tags']);

    $tags = [];

    foreach ($tagNames as $tagName) {

        $tagName = trim($tagName);

        // Sprawdzenie, czy tag już istnieje

        $tag = Tags::where('name', $tagName)->first();

        if (!$tag) {

            // Tag nie istnieje, tworzenie nowego

            $tag = new Tags;

            $tag->name = $tagName;

            $tag->save();

        }

        $tags[] = $tag->id;

    }

    // Tworzenie nowego wątku

    $thread = new Threads;

    $thread->title = $validatedData['title'];

    $thread->description = $validatedData['description'];

    $thread->user\_id = Auth::id();

    $thread->save();

    // Przypisanie tagów do wątku

    foreach ($tags as $tagId) {

        $tagThread = new TagsThreads;

        $tagThread->tag\_id = $tagId;

        $tagThread->thread\_id = $thread->id;

        $tagThread->save();

    }

    // Przekierowanie lub wykonanie dodatkowych akcji, jeśli potrzebne

    return redirect()->route('threads.index', ['id' => $thread->id]);

}

## Dodanie nowego miejsca:

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

Pola do Opening hours są obowiązkowe. Musimy podać nazwę miejsca, zaznaczyć czy jest to zabytek czy muzeum, jeśli jest to zabytek podajemy kod zabytku, a jeśli jest to muzeum trzeba podać NIP firmy. Jeśli kod zabytku jest zgodny z istniejącym kodem z bazy danych miejsce jest od razu zweryfikowane. Następnie trzeba podać szerokość i wysokość geograficzną, koniecznym jest też dodanie opisu. Opcjonalnie możemy dodać godziny otwarcia miejsca oraz zdjęcie które będzie wyświetlane na karcie miejsca.

public function store(Request $request)

{

    try {

        // Tworzenie współrzędnych

        $coordinates = Coordinates::create([

            'latitude' => $request->input('latitude'),

            'longitude' => $request->input('longitude'),

        ]);

        // Tworzenie miejsca

        $code = $request->input('monument\_code');

        $monumentCode = Monument\_codes::where('code', $code)->first();

        $verified = $monumentCode ? 1 : 0;

        $place = Places::create([

            'name' => $request->input('name'),

            'coordinates\_id' => $coordinates->id,

            'code' => $code,

            'description' => $request->input('description'),

            'verified' => $verified,

        ]);

        // Przetwarzanie podanych godzin otwarcia

        $submittedOpeningHours = $request->input('opening\_hours');

        $daysOfWeek = [

            'monday' => 1,

            'tuesday' => 2,

            'wednesday' => 3,

            'thursday' => 4,

            'friday' => 5,

            'saturday' => 6,

            'sunday' => 7,

        ];

        foreach ($submittedOpeningHours as $day => $hours) {

            $openingTime = $hours['opening\_time'];

            $closingTime = $hours['closing\_time'];

            $dayOfWeek = $daysOfWeek[$day];

            // Tworzenie rekordu godzin otwarcia

            $openingHour = new OpeningHour([

                'place\_id' => $place->id,

                'day\_of\_week' => $dayOfWeek,

                'opening\_time' => $openingTime,

                'closing\_time' => $closingTime,

            ]);

            $openingHour->save();

        }

        // Sprawdzenie, czy zostało przesłane zdjęcie

        if ($request->hasFile('photo')) {

            $image = $request->file('photo');

            // Zapisanie pliku w folderze storage

            $imagePath = $image->store('photos', 'public');

            $imageFileName = 'photos/' . basename($imagePath);

            // Tworzenie rekordu obrazu

            $image = new Images([

                'source\_url' => $imageFileName,

                'places\_id' => $place->id,

            ]);

            $image->save();

        }

        // Przekierowanie na stronę sukcesu lub wykonanie dodatkowych akcji

        return redirect()->route('places');

    } catch (QueryException $exception) {

        $errorMessage = 'Wystąpił błąd podczas zapisywania miejsca.';

        if ($exception->getCode() === '23000') {

            $errorMessage = 'Miejsce o takiej samej nazwie już istnieje.';

        }

        return redirect()->back()->with('error', $errorMessage)->withInput();

    }

}

## Dodawanie opinii o miejscu:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Strona internetowa

Opis wygenerowany automatyczniePo zalogowaniu jest też możliwość wyrażenia opinii o miejscu jak widać na zrzucie powyżej, możemy też dodać zdjęcie które będzie się wyświetlać w karuzeli na początku strony.

public function addOpinion(Request $request)

{

    // Walidacja danych formularza

    $validatedData = $request->validate([

        'place\_id' => 'required|numeric',

        'score' => 'required|numeric|between:1,5',

        'description' => 'required|string',

    ]);

    // Tworzenie nowej opinii

    $opinion = new Opinions();

    $opinion->place\_id = $validatedData['place\_id'];

    $opinion->user\_id = Auth::id();

    $opinion->score = $validatedData['score'];

    $opinion->description = $validatedData['description'];

    // Sprawdzenie czy zostało przesłane zdjęcie

    if ($request->hasFile('photo')) {

        $image = $request->file('photo');

        // Zapisanie pliku w folderze storage

        $imagePath = $image->store('photos', 'public');

        $imageFileName = 'photos/' . basename($imagePath);

        // Tworzenie rekordu obrazu

        $image = new Images([

            'source\_url' => $imageFileName,

            'places\_id' => $opinion->place\_id,

        ]);

        $image->save();

    }

    // Zapisanie opinii

    $opinion->save();

    // Przekierowanie lub zwrócenie odpowiedzi

    return redirect()->back()->with('success', 'Opinion added successfully.');

}

## 3.Funkcjonalności admina

## Usuwanie postów

Obraz zawierający tekst, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie

Pojawia się możliwość usunięcia posta, zwykła operacja crud

    public function destroy($id)

    {

        // Find the thread by ID

        $thread = Threads::findOrFail($id);

        if (!auth()->user()->role === 1) {

            return redirect()->back()->with('error', 'You are not authorized to delete this thread.');

        }

        // Delete the thread

        $thread->delete();

        // Redirect the user to the threads index page

        return redirect()->route('root')->with('success', 'Thread deleted successfully.');

    }

## Panel edycji użytkowników

Obraz zawierający tekst, zrzut ekranu, numer, oprogramowanie

Opis wygenerowany automatycznie

W tym miejscu admin może edytować użytkowników oraz ich usuwać.

Obraz zawierający tekst, zrzut ekranu, numer, linia

Opis wygenerowany automatycznie

## Edycja i usuwanie opinii:

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

## Dodanie i usunięcie weryfikacji miejsca:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie