|  |  |
| --- | --- |
| **AI1** | Dokumentacja projektu |
| **Autor** | Rafał Liszcz, 125140 |
| **Kierunek, rok** | Informatyka, II rok, st. stacjonarne (3,5-l) |
| **Temat  projektu** | Informator o rozkładzie autobusów |

Spis treści

[Wstęp 3](#_Toc167791055)

[Narzędzia i technologie 3](#_Toc167791056)

[Baza danych 4](#_Toc167791057)

[GUI 5](#_Toc167791058)

[Uruchomienie aplikacji 7](#_Toc167791059)

[Funkcjonalności aplikacji 7](#_Toc167791060)

[Wybrany proces/logika biznesowa: 12](#_Toc167791061)

[Walidacja danych: 15](#_Toc167791062)

[Podsumowanie: 24](#_Toc167791063)

## Wstęp

Aplikacja „Rozkład autobusowy” ma na celu umożliwienie użytkownikom wyszukiwania oraz zarządzania rozkładami autobusów. Użytkownicy mogą wyszukiwać przejazdy według miast wyjazdu i przyjazdu, daty oraz godziny wyjazdu. Administratorzy mają dodatkowo możliwość dodawania, edytowania oraz usuwania rozkładów autobusowych.

## Narzędzia i technologie

W projekcie wykorzystano następujące technologie:

* **PHP (wersja 8.X)** – język programowania używany do tworzenia logiki aplikacji.
* **Laravel (wersja 11.x)** – framework PHP ułatwiający tworzenie aplikacji webowych.
* **Blade** – silnik szablonów wykorzystywany w Laravelu.
* **MySQL** – system zarządzania bazą danych używany do przechowywania danych aplikacji.
* **Bootstrap** – framework CSS używany do stylizacji interfejsu użytkownika.
* **JavaScript** – język programowania używany do dynamicznej interakcji na stronie.

Wszystkie wymienione technologie są dostępne bezpłatnie. Dokumentacja i miejsca pobierania:

* [PHP](https://www.php.net/docs.php)
* [Laravel](https://laravel.com/docs)
* [MySQL](https://dev.mysql.com/doc/)
* [Bootstrap](https://getbootstrap.com/docs/4.1/getting-started/introduction/)
* [JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript)

## Baza danych

**Schemat ERD:**

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

**Opis zawartości bazy danych:**

* **Users** – tabela przechowująca informacje o użytkownikach (id, name, email, password, role\_id).
* **Cities** – tabela przechowująca informacje o miastach (id, name, img, zip\_code).
* **Schedules** – tabela przechowująca informacje o rozkładach jazdy (id, departure\_time, arrival\_time, ticket\_price, route\_id, bus\_number).
* **Routes** – tabela przechowująca informacje o trasach (id, distance).
* **Stops** – tabela przechowująca informacje o przystankach (id, name).

**Opis powiązań pomiędzy tabelami:**

Jeden użytkownik może być administratorem lub zwykłym użytkownikiem.

Jeden rozkład jazdy należy do jednej trasy.

Trasa może mieć wiele przystanków.

Miasta są powiązane z rozkładami jazdy przez tabele pośrednie.

## GUI

**Widok 1: Strona główna**

Obraz zawierający tekst, niebo, chmura, zrzut ekranu

Opis wygenerowany automatycznie

**Widok 2: Wyszukiwanie przejazdów**

Obraz zawierający tekst, zrzut ekranu, Strona internetowa, oprogramowanie

Opis wygenerowany automatycznie**Widok 3: Dodawanie nowego przejazdu**

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, numer

Opis wygenerowany automatycznie

## Uruchomienie aplikacji

**Kroki konfiguracyjne:**

Uruchomić XAMPP oraz Włączyć Apache oraz MySql.

Otworzyć Projekt w VSCode.

W przypadku posiadania innych ustawień niż domyślne (np. połączenia z bazą), wykonać ich zmianę w .env.example oraz start.bat lub start.sh.

W Folderze Projektu Uruchomić skrypt start.bat (Windows, 2x kliknięciem) lub start.sh (inne systemy, przez polecenie bash start.sh).

Otworzyć terminal cmd (Command Prompt) w VSCode.

Uruchom serwer lokalny za pomocą komend php artisan serve.

## Funkcjonalności aplikacji

**Logowanie i rejestracja**

**Logowanie:** Formularz logowania z walidacją danych (email, hasło). Przykładowe loginy i hasła:

Email: Jan@email.com, Hasło: 1234 🡨 Admin

Email: marta@email.com, Hasło: 1234 🡨 Użytkownik

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, komputer

Opis wygenerowany automatycznie

**Rejestracja:** Formularz rejestracji z walidacją danych (nazwa użytkownika, email, hasło, potwierdzenie hasła).

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, wyświetlacz

Opis wygenerowany automatycznie

**CRUD przeprowadzany przez administratora**

**Dodawanie nowego przejazdu:** Administrator może dodawać nowe przejazdy poprzez formularz.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, wyświetlacz

Opis wygenerowany automatycznie

**Usuwanie Przejazdu:** Administrator może usuwać przejazdy poprzez Kliknięcie w przycisk Delete.

Obraz zawierający tekst, zrzut ekranu, Strona internetowa, Reklama internetowa

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Strona internetowa, oprogramowanie

Opis wygenerowany automatycznie

**Edycja Przejazdu:** Administrator może edytować przejazdy poprzez formularz.

**Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu, oprogramowanie, wyświetlacz

Opis wygenerowany automatycznie**

**Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie**

**Przeglądanie ogólnodostępnych zasobów**

**Filtrowanie zasobów:** Wszyscy mogą filtrować przejazdy według miasta wyjazdu, miasta przyjazdu, daty i godziny wyjazdu.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Strona internetowa

Opis wygenerowany automatycznie

Obraz zawierający zrzut ekranu, tekst, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

## Wybrany proces/logika biznesowa:

**Wyszukiwanie przejazdów:**

Proces wyszukiwania przejazdów polega na filtrowaniu dostępnych rozkładów jazdy na podstawie kryteriów wprowadzonych przez użytkownika. Kryteria te mogą obejmować miasto wyjazdu, miasto przyjazdu, datę oraz godzinę wyjazdu. Poniżej znajduje się implementacja tego procesu:

**Pobieranie danych z żądania:**

$departureCity = $request->input('departure\_city');

    $arrivalCity = $request->input('arrival\_city');

    $departureDate = $request->input('departure\_date');

    $departureTime = $request->input('departure\_time');

**Funkcja pobiera dane z obiektu Request**, które zostały przekazane przez użytkownika za pomocą formularza wyszukiwania. Te dane to:

* **departure\_city**: Miasto wyjazdu.
* **arrival\_city:** Miasto przyjazdu.
* **departure\_date:** Data wyjazdu.
* **departure\_time:** Godzina wyjazdu.

**Inicjalizacja zapytania do bazy danych:**

    $query = Schedule::query();

Tworzy początkowe zapytanie do bazy danych na modelu Schedule, które później będzie modyfikowane w zależności od podanych kryteriów.

**Dodawanie warunków do zapytania:**

**Miasto wyjazdu:**

 if ($departureCity) {

        $query->whereHas('departureCities', function ($query) use ($departureCity) {

        $query->where('city\_id', $departureCity);

      });

 }

Jeśli użytkownik podał miasto wyjazdu, zapytanie zostanie rozszerzone o warunek, który sprawdza, czy departureCities (relacja w modelu Schedule) zawiera podane city\_id.

**Miasto przyjazdu:**

if ($arrivalCity) {

        $query->whereHas('arrivalCities', function ($query) use ($arrivalCity)

{

        $query->where('city\_id', $arrivalCity);

        });

}

Jeśli użytkownik podał miasto przyjazdu, zapytanie zostanie rozszerzone o warunek, który sprawdza, czy arrivalCities (relacja w modelu Schedule) zawiera podane city\_id.

**Data wyjazdu:**

if ($departureDate) {

        $query->whereDate('departure\_time', $departureDate);

    }

Jeśli użytkownik podał datę wyjazdu, zapytanie zostanie rozszerzone o warunek, który sprawdza, czy departure\_time jest równy podanej dacie.

**Godzina wyjazdu:**

if ($departureTime) {

        $query->whereTime('departure\_time', '>=', $departureTime);

}

Jeśli użytkownik podał godzinę wyjazdu, zapytanie zostanie rozszerzone o warunek, który sprawdza, czy departure\_time jest większy lub równy podanej godzinie.

**Sprawdzanie, czy żadne kryteria nie zostały podane:**

if (!$departureCity && !$arrivalCity && !$departureDate && !$departureTime) {

        $schedules = collect();

    } else {

        $schedules = $query->with('departureCities', 'arrivalCities', 'route')->get();

    }

Jeśli żadne kryteria nie zostały podane, funkcja zwraca pustą kolekcję. W przeciwnym razie wykonuje zapytanie do bazy danych, pobierając wyniki wraz z powiązanymi miastami wyjazdu, przyjazdu oraz trasą.

**Paginacja wyników:**

$schedules = $query->with('departureCities', 'arrivalCities', 'route')->paginate(3)->appends([

        'departure\_city' => $departureCity,

        'arrival\_city' => $arrivalCity,

        'departure\_date' => $departureDate,

        'departure\_time' => $departureTime,

    ]);

Wyniki są paginowane, aby na jednej stronie wyświetlać tylko trzy rekordy. Funkcja appends dodaje parametry wyszukiwania do paginacji, aby zachować je w URL podczas przeglądania kolejnych stron wyników.

**Pobieranie wszystkich miast:**

$cities = City::all();

Pobiera wszystkie miasta z bazy danych, aby mogły być wyświetlone w formularzu wyszukiwania.

**Zwracanie widoku z wynikami wyszukiwania:**

return view('index', compact('schedules', 'cities', 'departureCity', 'arrivalCity', 'departureDate', 'departureTime'));

**Funkcja zwraca widok index z przekazanymi danymi:**

schedules: Wyniki wyszukiwania.

cities: Lista wszystkich miast.

departureCity, arrivalCity, departureDate, departureTime: Kryteria wyszukiwania wprowadzone przez użytkownika.

## Walidacja danych:

Walidacja danych to kluczowy element każdej aplikacji, który zapewnia, że wprowadzone przez użytkowników informacje są poprawne i zgodne z wymaganiami systemu. W projekcie „Rozkład autobusowy” zastosowano walidację w różnych miejscach, aby zapewnić integralność danych i poprawność działania aplikacji. Poniżej przedstawiono szczegółowy opis walidacji danych dla różnych funkcjonalności.

**1. Rejestracja użytkownika**

Walidacja danych podczas rejestracji użytkownika odbywa się w metodzie register w kontrolerze AuthController. Obejmuje ona sprawdzenie, czy nazwa użytkownika, email i hasło spełniają określone wymagania.

public function register(Request $request)

    {

        $validatedData = $request->validate([

            'name' => 'required|max:255',

            'email' => 'required|email|unique:users',

            'password' => 'required|confirmed',

        ]);

        $user = User::create([

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

            'email' => $validatedData['email'],

            'password' => Hash::make($validatedData['password']),

            'role\_id' => 2,

        ]);

        Auth::login($user);

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

    }

**name:** Wymagane, maksymalnie 255 znaków.

**email:** Wymagane, musi być unikalny w tabeli users, musi mieć format emaila.

**password:** Wymagane, musi być potwierdzone (pole password\_confirmation).

 <form method="POST" action="{{ route('register') }}" class="needs-validation" novalidate>

                    @csrf

                    <div class="form-group mb-2">

                        <label for="name" class="form-label">Nazwa użytkownika</label>

                        <input id="name" name="name" type="text" class="form-control @if ($errors->first('name')) is-invalid @endif" value="{{ old('name') }}" required>

                        <div class="invalid-feedback">Nazwa użytkownika jest wymagana!</div>

                    </div>

                    <div class="form-group mb-2">

                        <label for="email" class="form-label">Email</label>

                        <input id="email" name="email" type="email" class="form-control @if ($errors->first('email')) is-invalid @endif" value="{{ old('email') }}" required>

                        <div class="invalid-feedback">Nieprawidłowy email!</div>

                    </div>

                    <div class="form-group mb-2">

                        <label for="password" class="form-label">Hasło</label>

                        <input id="password" name="password" type="password" class="form-control @if ($errors->first('password')) is-invalid @endif" required>

                        <div class="invalid-feedback">Hasło jest wymagane!</div>

                    </div>

                    <div class="form-group mb-2">

                        <label for="password\_confirmation" class="form-label">Potwierdź hasło</label>

                        <input id="password\_confirmation" name="password\_confirmation" type="password" class="form-control @if ($errors->first('password\_confirmation')) is-invalid @endif" required>

                        <div class="invalid-feedback">Potwierdzenie hasła jest wymagane!</div>

                    </div>

                    <div class="text-center mt-4 mb-4">

                        <input class="btn btn-primary" type="submit" value="Zarejestruj się">

                    </div>

                </form>

**Walidacja danych po stronie klienta i servera:**

Obraz zawierający tekst, zrzut ekranu, Czcionka, design

Opis wygenerowany automatycznie

**2. Logowanie użytkownika**

Walidacja danych podczas logowania użytkownika odbywa się w metodzie authenticate w kontrolerze AuthController. Obejmuje ona sprawdzenie, czy email i hasło są podane.

public function authenticate(Request $request)

    {

        $credentials = $request->validate([

            'email' => ['required', 'email'],

            'password' => ['required'],

        ]);

        if (Auth::attempt($credentials)) {

            $request->session()->regenerate();

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

        }

        return back()->withErrors([

            'email' => 'The provided credentials do not match our records.',

        ])->onlyInput('email');

    }

email: Wymagane, musi mieć format emaila.

password: Wymagane.

 <form method="POST" action="{{ route('login.authenticate') }}" class="needs-validation" novalidate>

                    @csrf

                    <div class="form-group mb-2">

                        <label for="email" class="form-label">Email</label>

                        <input id="email" name="email" type="text" class="form-control @if ($errors->first('email')) is-invalid @endif" value="{{ old('email') }}">

                        <div class="invalid-feedback">Nieprawidłowy email!</div>

                    </div>

                    <div class="form-group mb-2">

                        <label for="continent" class="form-label">Hasło</label>

                        <input id="password" name="password" type="password" class="form-control @if ($errors->first('password')) is-invalid @endif">

                        <div class="invalid-feedback">Nieprawidłowe hasło!</div>

                    </div>

                    <div class="text-center mt-4 mb-4">

                        <input class="btn btn-primary" type="submit" value="Wyślij">

                    </div>

                </form>

Walidacja po stronie klienta i servera:

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

**3. Dodawanie nowego przejazdu**

Walidacja danych podczas dodawania nowego przejazdu odbywa się w metodzie store kontrolera ScheduleController i wykorzystuje klasę StoreScheduleRequest.

 public function rules(): array

    {

        return [

            'bus\_number' => 'required|string|max:255',

            'distance' => 'required|numeric|min:0',

            'departure\_city' => 'required|string|max:255',

            'arrival\_city' => 'required|string|max:255',

            'departure\_time' => 'required|date|date\_format:Y-m-d\TH:i',

            'arrival\_time' =>

'required|date|after:departure\_time|date\_format:Y-m-d\TH:i',

            'ticket\_price' => 'required|numeric|min:0',

            'departure\_city\_img' =>

'nullable|image|mimes:jpeg,png,jpg,gif|max:2048',

            'arrival\_city\_img' =>

'nullable|image|mimes:jpeg,png,jpg,gif|max:2048',

            'stops' => 'array',

            'stops.\*' => 'string|max:255',

        ];

    }

bus\_number: Wymagane, tekst, maksymalnie 255 znaków.

distance: Wymagane, liczba, minimum 0.

departure\_city: Wymagane, tekst, maksymalnie 255 znaków.

arrival\_city: Wymagane, tekst, maksymalnie 255 znaków.

departure\_time: Wymagane, data w formacie Y-m-d\TH:i.

arrival\_time: Wymagane, data po departure\_time, format Y-m-d\TH:i.

ticket\_price: Wymagane, liczba, minimum 0.

departure\_city\_img: Opcjonalne, obrazek w formatach jpeg, png, jpg, gif, maksymalnie 2048 KB.

arrival\_city\_img: Opcjonalne, obrazek w formatach jpeg, png, jpg, gif, maksymalnie 2048 KB.

stops: Tablica, elementy muszą być tekstem, maksymalnie 255 znaków.

 <div class="form-group mb-2">

            <label for="bus\_number" class="form-label">Numer Autobusu</label>

            <input type="text" name="bus\_number" id="bus\_number" class="form-control @if ($errors->first('bus\_number')) is-invalid @endif" required maxlength="255">

            <div class="invalid-feedback">Numer autobusu jest wymagany i nie może przekraczać 255 znaków.</div>

        </div>

        <div class="form-group mb-2">

            <label for="distance" class="form-label">Dystans (km)</label>

            <input type="number" step="0.01" name="distance" id="distance" class="form-control @if ($errors->first('distance')) is-invalid @endif" required min="0">

            <div class="invalid-feedback">Dystans jest wymagany i musi być liczbą nieujemną.</div>

        </div>

        <div class="form-group mb-2">

            <label for="departure\_city" class="form-label">Miasto Wyjazdu</label>

            <input type="text" name="departure\_city" id="departure\_city" class="form-control @if ($errors->first('departure\_city')) is-invalid @endif" required maxlength="255">

            <div class="invalid-feedback">Miasto wyjazdu jest wymagane i nie może przekraczać 255 znaków.</div>

            <input type="file" name="departure\_city\_img" class="form-control mt-2">

        </div>

        <div class="form-group mb-2">

            <label for="arrival\_city" class="form-label">Miasto Przyjazdu</label>

            <input type="text" name="arrival\_city" id="arrival\_city" class="form-control @if ($errors->first('arrival\_city')) is-invalid @endif" required maxlength="255">

            <div class="invalid-feedback">Miasto przyjazdu jest wymagane i nie może przekraczać 255 znaków.</div>

            <input type="file" name="arrival\_city\_img" class="form-control mt-2">

        </div>

        <div class="form-group mb-2">

            <label for="departure\_time" class="form-label">Godzina Odjazdu</label>

            <input type="datetime-local" name="departure\_time" id="departure\_time" class="form-control @if ($errors->first('departure\_time')) is-invalid @endif" required>

            <div class="invalid-feedback">Godzina odjazdu jest wymagana.</div>

        </div>

        <div class="form-group mb-2">

            <label for="arrival\_time" class="form-label">Godzina Przyjazdu</label>

            <input type="datetime-local" name="arrival\_time" id="arrival\_time" class="form-control @if ($errors->first('arrival\_time')) is-invalid @endif" required>

            <div class="invalid-feedback">Godzina przyjazdu jest wymagana i musi być późniejsza niż godzina odjazdu.</div>

        </div>

        <div class="form-group mb-2">

            <label for="ticket\_price" class="form-label">Cena</label>

            <input type="number" step="0.01" name="ticket\_price" id="ticket\_price" class="form-control @if ($errors->first('ticket\_price')) is-invalid @endif" required min="0">

            <div class="invalid-feedback">Cena biletu jest wymagana i musi być liczbą nieujemną.</div>

        </div>

        <div class="form-group mb-2">

            <label for="stops" class="form-label">Przystanki (Przystanki muszą być oddzielone przecinkiem)</label>

            <input type="text" name="stops[]" id="stops" class="form-control @if ($errors->first('stops')) is-invalid @endif" required>

            <div class="invalid-feedback">Przystanki są wymagane.</div>

        </div>

**Walidacja po stronie klienta i servera:**



4. Aktualizacja przejazdu

Walidacja danych podczas aktualizacji przejazdu odbywa się w metodzie update kontrolera ScheduleController i wykorzystuje klasę UpdateScheduleRequest.

public function rules(): array

    {

        return [

            'bus\_number' => 'required|string|max:255',

            'distance' => 'required|numeric|min:0',

            'departure\_city' => 'required|string|max:255',

            'arrival\_city' => 'required|string|max:255',

            'departure\_time' => 'required|date|date\_format:Y-m-d\TH:i',

            'arrival\_time' =>

'required|date|after:departure\_time|date\_format:Y-m-d\TH:i',

            'ticket\_price' => 'required|numeric|min:0',

            'departure\_city\_img' =>

'nullable|image|mimes:jpeg,png,jpg,gif|max:2048',

            'arrival\_city\_img' =>

'nullable|image|mimes:jpeg,png,jpg,gif|max:2048',

            'stops' => 'array',

            'stops.\*' => 'string|max:255',

        ];

    }

Walidacja jest taka sama jak przy dodawaniu nowego przejazdu.

        <div class="form-group mb-2">

            <label for="bus\_number" class="form-label">Numer Autobusu</label>

            <input type="text" name="bus\_number" id="bus\_number" class="form-control @if ($errors->first('bus\_number')) is-invalid @endif" value="{{ $schedule->bus\_number }}" required maxlength="255">

            <div class="invalid-feedback">Numer autobusu jest wymagany i nie może przekraczać 255 znaków.</div>

        </div>

        <div class="form-group mb-2">

            <label for="distance" class="form-label">Dystans (km)</label>

            <input type="number" step="0.01" name="distance" id="distance" class="form-control @if ($errors->first('distance')) is-invalid @endif" value="{{ $schedule->route->distance }}" required min="0">

            <div class="invalid-feedback">Dystans jest wymagany i musi być liczbą nieujemną.</div>

        </div>

        <div class="form-group mb-2">

            <label for="departure\_city" class="form-label">Miasto Wyjazdu</label>

            <input type="text" name="departure\_city" id="departure\_city" class="form-control @if ($errors->first('departure\_city')) is-invalid @endif" value="{{ $schedule->departureCities->first()->name }}" required maxlength="255">

            <div class="invalid-feedback">Miasto wyjazdu jest wymagane i nie może przekraczać 255 znaków.</div>

            <input type="file" name="departure\_city\_img" class="form-control mt-2">

        </div>

        <div class="form-group mb-2">

            <label for="arrival\_city" class="form-label">Miasto Przyjazdu</label>

            <input type="text" name="arrival\_city" id="arrival\_city" class="form-control @if ($errors->first('arrival\_city')) is-invalid @endif" value="{{ $schedule->arrivalCities->first()->name }}" required maxlength="255">

            <div class="invalid-feedback">Miasto przyjazdu jest wymagane i nie może przekraczać 255 znaków.</div>

            <input type="file" name="arrival\_city\_img" class="form-control mt-2">

        </div>

        <div class="form-group mb-2">

            <label for="departure\_time" class="form-label">Godzina Odjazdu</label>

            <input type="datetime-local" name="departure\_time" id="departure\_time" class="form-control @if ($errors->first('departure\_time')) is-invalid @endif" value="{{ $schedule->departure\_time }}" required>

            <div class="invalid-feedback">Godzina odjazdu jest wymagana.</div>

        </div>

        <div class="form-group mb-2">

            <label for="arrival\_time" class="form-label">Godzina Przyjazdu</label>

            <input type="datetime-local" name="arrival\_time" id="arrival\_time" class="form-control @if ($errors->first('arrival\_time')) is-invalid @endif" value="{{ $schedule->arrival\_time }}" required>

            <div class="invalid-feedback">Godzina przyjazdu jest wymagana i musi być późniejsza niż godzina odjazdu.</div>

        </div>

        <div class="form-group mb-2">

            <label for="ticket\_price" class="form-label">Cena</label>

            <input type="number" step="0.01" name="ticket\_price" id="ticket\_price" class="form-control @if ($errors->first('ticket\_price')) is-invalid @endif" value="{{ $schedule->ticket\_price }}" required min="0">

            <div class="invalid-feedback">Cena biletu jest wymagana i musi być liczbą nieujemną.</div>

        </div>

        <div class="form-group mb-2">

            <label for="stops" class="form-label">Przystanki (Przystanki muszą być oddzielone przecinkiem)</label>

            <input type="text" name="stops[]" id="stops" class="form-control @if ($errors->first('stops')) is-invalid @endif" value="{{ implode(', ', $schedule->route->stops->pluck('name')->toArray()) }}" required>

            <div class="invalid-feedback">Przystanki są wymagane.</div>

        </div>

**Walidacja po stronie klienta i servera:**

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

Podsumowując, walidacja danych w projekcie „Rozkład autobusowy” zapewnia integralność danych i poprawność działania aplikacji, chroniąc przed wprowadzeniem nieprawidłowych informacji.

## Podsumowanie:

Dokumentacja projektu „Rozkład autobusowy” przedstawia kompleksowy opis procesu tworzenia aplikacji, która umożliwia zarządzanie i przeglądanie rozkładów jazdy autobusów. Dzięki wykorzystaniu Laravel 11.x oraz silnika Blade, aplikacja jest modularna, łatwa do rozszerzenia i utrzymania. MySQL zapewnia solidne podstawy do przechowywania danych, a Bootstrap oraz JavaScript zapewniają interaktywność i responsywność interfejsu użytkownika. Walidacja danych jest kluczowym elementem projektu, zapewniającym integralność danych i bezpieczeństwo aplikacji. Implementacja różnych funkcjonalności, takich jak logowanie, rejestracja, zarządzanie zasobami oraz filtrowanie wyników, pokazuje praktyczne zastosowanie najlepszych praktyk programistycznych.