

Spis treści

| | |
|--|----|
| 1. Wstęp..... | 2 |
| 2. Narzędzia i technologie..... | 3 |
| 2.1. Technologie Backendowe..... | 3 |
| 2.2. Baza Danych | 3 |
| 2.3. Środowisko Programistyczne i Narzędzia..... | 3 |
| 2.4. Technologie Frontendowe..... | 4 |
| 2.5. Kluczowe Biblioteki..... | 4 |
| 3. Baza danych..... | 5 |
| 3.1 Diagram ERD | 5 |
| 3.2 Opis tabel..... | 5 |
| 3.3 Opis powiązań między tabelami..... | 7 |
| 4. GUI..... | 8 |
| 4.1. Widok szczegółów przystanku i rozkładu jazdy..... | 8 |
| 4.2. Widok planowania trasy | 9 |
| 4.3. Widok panelu administracyjnego | 9 |
| 4.4 Dostępność i Użyteczność (RWD i Ułatwienia Dostępu) | 10 |
| 5. Uruchomienie aplikacji | 12 |
| 5.1. Wymagania wstępne..... | 12 |
| 5.2. Instalacja i konfiguracja | 12 |
| 5.3. Konfiguracja płatności Stripe | 13 |
| 5.4. Przykładowe dane logowania..... | 14 |
| 6. Funkcjonalność aplikacji..... | 14 |
| 6.1 Proces rejestracji i logowania użytkownika | 15 |
| 6.2 Przykładowy CRUD przeprowadzany przez administratora | 16 |
| 6.3 Zarządzanie użytkownikami przez administratora | 17 |
| 6.4 Przeglądanie ogólnodostępnych zasobów (bez logowania) | 19 |
| 6.5 Zarządzanie swoimi zasobami przez użytkownika aplikacji (bilety) | 20 |
| 6.6 Zarządzanie swoimi danymi przez użytkownika aplikacji..... | 22 |
| 6.7 Walidacja danych..... | 24 |
| 6.7.1 Przykłady walidacji w widokach | 24 |
| 6.7.2. Walidacja po stronie serwera (Server-Side Validation) | 28 |
| 6.7.3. Walidacja po stronie klienta (Client-Side Validation) | 30 |
| 6.7.4. Przykład praktyczny: Formularz Rejestracji/logowania | 30 |
| 6.8 Logika biznesowa..... | 32 |
| 7. Podsumowanie..... | 40 |

1. Wstęp

Tematem mojego projektu jest „System informacyjny dla miejskiej komunikacji autobusowej”. W założeniu powstać miała aplikacja webowa, której celem jest usprawnienie zarządzania transportem publicznym oraz dostarczenie pasażerom aktualnych i łatwo dostępnych informacji o połączeniach. System odwzorowuje kluczowe procesy związane z planowaniem i realizacją przejazdów, zarządzaniem ofertą biletową oraz ewidencją użytkowników.

Głównym założeniem projektu jest stworzenie zintegrowanej platformy, która służy zarówno pasażerom, jak i administratorom systemu. Użytkownicy aplikacji podzieleni są na dwie główne grupy o różnych uprawnieniach.

Zwykły użytkownik (pasażer) posiada dostęp do następujących funkcjonalności:

- Samodzielna rejestracja i logowanie w systemie.
- Zarządzanie swoim profilem (zmiana danych, hasła, zdjęcia profilowego).
- Przeglądanie dostępnych linii autobusowych, przystanków i szczegółowych rozkładów jazdy.
- Planowanie podróży poprzez wyszukiwanie połączeń między wybranymi przystankami.
- Zakup biletów elektronicznych z dostępnej oferty.
- Przeglądanie historii swoich transakcji i zakupionych biletów.

Administrator systemu otrzymuje rozszerzone uprawnienia, które pozwalają na kompleksowe zarządzanie całą platformą:

- Zarządzanie siatką połączeń: dodawanie i edycja linii autobusowych, przystanków oraz definiowanie kolejności przystanków na trasie.
- Konfiguracja rozkładów jazdy poprzez tworzenie kursów i przypisywanie im godzin odjazdów w zależności od typu dnia (np. dni robocze, weekendy).
- Zarządzanie ofertą biletową, w tym dodawanie nowych typów biletów, ustalanie ich cen, okresu ważności i statusu aktywności.
- Zarządzanie kontami użytkowników, w tym nadawanie uprawnień administracyjnych.
- Monitorowanie transakcji przejazdów w systemie.

Projekt obejmuje zatem cyfrowe odwzorowanie funkcjonowania operatora komunikacji miejskiej. Aplikacja ułatwia pasażerom codzienne korzystanie z transportu publicznego, a administratorom dostarcza scentralizowane narzędzie do efektywnego zarządzania całą infrastrukturą logistyczną i sprzedażową.

2. Narzędzia i technologie

Do realizacji projektu wykorzystano zbiór nowoczesnych, darmowych technologii opartych na licencji open-source, co zapewnia elastyczność i brak barier we wdrożeniu. Poniżej znajduje się szczegółowe omówienie kluczowych komponentów użytych do budowy aplikacji.

2.1. Technologie Backendowe

PHP

- Wersja: 8.2
- Opis: Główny język programowania po stronie serwera, na którym oparta jest cała logika biznesowa aplikacji. Jego nowoczesne funkcje pozwoliły na stworzenie wydajnego i bezpiecznego kodu.
- Licencja: PHP License (darmowa, open-source).
- Dokumentacja: <https://www.php.net/>

Laravel

- Wersja: 12.0
- Opis: Framework PHP wykorzystany jako szkielet aplikacji. Dostarcza gotowe komponenty do obsługi routingu, zapytań do bazy danych (ORM Eloquent), uwierzytelniania i wielu innych, co znacząco przyspieszyło proces tworzenia projektu.
- Licencja: MIT (darmowa, open-source).
- Dokumentacja: <https://laravel.com/docs/12.x>

2.2. Baza Danych

MySQL

- **Opis:** Popularny, open-source'owy system zarządzania relacyjnymi bazami danych (RDBMS). Został wybrany ze względu na swoją wydajność, niezawodność i szerokie wsparcie społeczności, co czyni go standardem w aplikacjach webowych.
- **Licencja:** GNU GPL v2.
- **Dokumentacja:** <https://www.mysql.com/>

2.3. Środowisko Programistyczne i Narzędzia

XAMPP

- **Opis:** Darmowy, wieloplatformowy pakiet oprogramowania, który posłużył jako lokalne środowisko serwerowe. Dostarczył wszystkie niezbędne komponenty do uruchomienia aplikacji: serwer WWW Apache, interpreter PHP oraz serwer bazy danych MySQL.
- **Licencja:** GNU General Public License.
- **Dokumentacja:** <https://www.apachefriends.org/pl/index.html>

PhpStorm

- **Opis:** Profesjonalne, zintegrowane środowisko programistyczne (IDE) od firmy JetBrains, dedykowane językowi PHP. Zostało wykorzystane ze względu na zaawansowane funkcje analizy kodu, refaktoryzacji, debugowania oraz integrację z narzędziami takimi jak Composer i systemy baz danych.
- **Licencja:** Komercyjna (z dostępnymi darmowymi licencjami edukacyjnymi).
- **Dokumentacja:** <https://www.jetbrains.com/phpstorm/>

Composer

- **Opis:** Menedżer zależności dla języka PHP. Służył do instalacji i zarządzania wszystkimi bibliotekami oraz frameworkiem Laravel.
- **Licencja:** MIT (darmowa, open-source).
- **Dokumentacja:** <https://getcomposer.org/>

2.4. Technologie Frontendowe

Blade

- **Opis:** Domyślny silnik szablonów frameworka Laravel. Pozwala na tworzenie dynamicznych widoków poprzez łączenie kodu HTML ze składnią PHP w prosty i czytelny sposób. Szablony Blade są kompilowane i buforowane, co zapewnia wysoką wydajność.
- **Licencja:** MIT (część frameworka Laravel).
- **Dokumentacja:** <https://laravel.com/docs/12.x/blade>

2.5. Kluczowe Biblioteki

Poniższe biblioteki zostały zainstalowane za pomocą Composera i stanowią integralną część projektu:

Guzzle HTTP Client

- **Opis:** Biblioteka do wykonywania zapytań HTTP. Może być wykorzystywana do komunikacji z zewnętrznymi API.
- **Licencja:** MIT.
- **Dokumentacja:** <https://docs.guzzlephp.org/>

Stripe PHP

- **Opis:** Oficjalna biblioteka PHP do integracji z systemem płatności Stripe. Umożliwia obsługę transakcji za bilety.
- **Licencja:** MIT.
- **Dokumentacja:** <https://stripe.com/docs/api?lang=php>

Laravel Lang

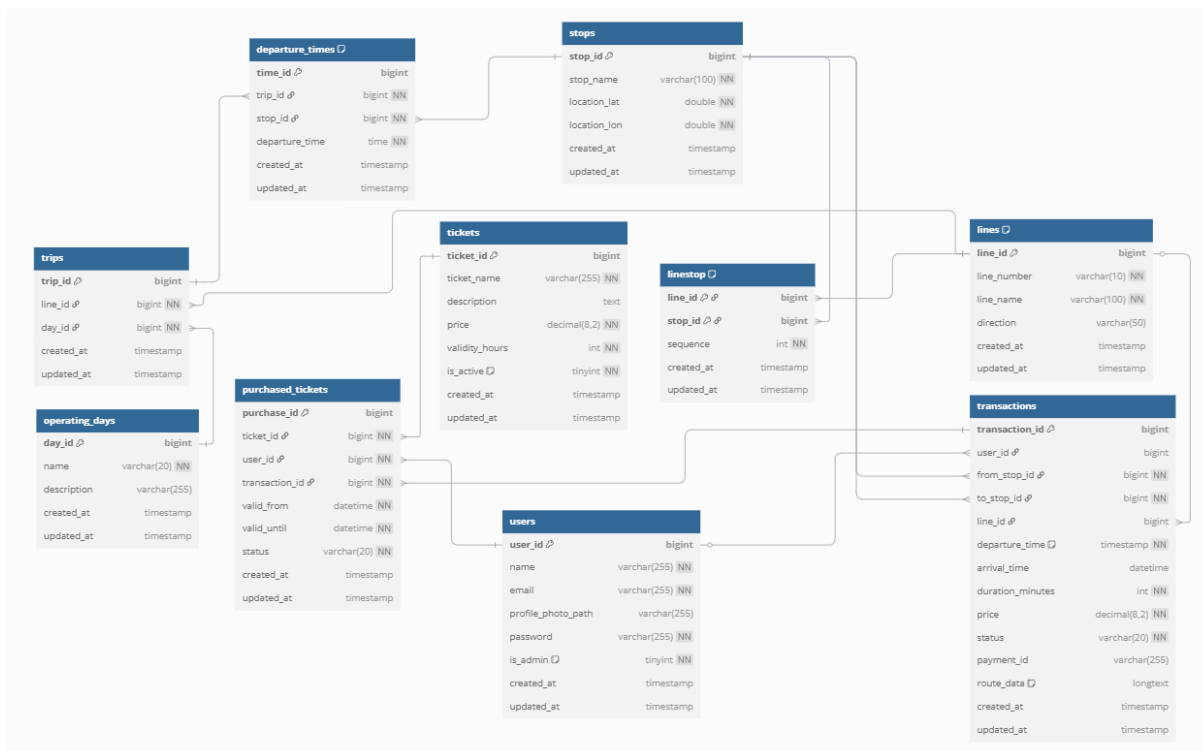
- **Opis:** Pakiet dostarczający gotowe tłumaczenia dla standardowych elementów frameworka Laravel, co ułatwiło internacjonalizację aplikacji.
- **Licencja:** MIT.
- **Dokumentacja:** <https://laravel-lang.com/>

PHPUnit

- **Opis:** Framework do testów jednostkowych dla PHP, używany do weryfikacji poprawności działania poszczególnych komponentów aplikacji.
- **Licencja:** BSD-3-Clause.
- **Dokumentacja:** <https://phpunit.de/>

3. Baza danych

3.1 Diagram ERD



Rysunek 1. Diagram ERD

3.2 Opis tabel

1. Tabela users (Użytkownicy)

Przechowuje dane kont użytkowników, takie jak identyfikator, dane logowania i informacje profilowe.

Unikalne: adres e-mail.

2. Tabela lines (Linie autobusowe)

Zawiera identyfikatory, numery i nazwy linii autobusowych oraz opcjonalny kierunek.

Unikalne: kombinacja numeru linii i kierunku.

3. **Tabela stops (Przystanki)**

Zawiera nazwy i współrzędne przystanków umożliwiające ich lokalizację na mapie.

Unikalne: nazwa przystanku.

4. **Tabela operating_days (Dni kursowania)**

Definiuje typy dni, według których obowiązują rozkłady jazdy.

Unikalne: nazwa typu dnia.

5. **Tabela tickets (Typy biletów)**

Przechowuje informacje o dostępnych typach biletów, w tym ceny, opisy i czas ważności.

Unikalne: nazwa biletu.

6. **Tabela trips (Kursy)**

Zawiera powiązania kursów z liniami i dniami kursowania.

Brak zdefiniowanych unikalnych kombinacji poza kluczami głównymi.

7. **Tabela linestop (Kolejność przystanków na linii)**

Reprezentuje relację między linią a przystankami wraz z określeniem ich kolejności.

Możliwa unikalność kombinacji linii, przystanku i kolejności.

8. **Tabela departure_times (Czas odjazdów)**

Przechowuje godziny odjazdów dla kursów z poszczególnych przystanków.

Unikalne: kombinacja kursu (trip_id), przystanku (stop_id) i godziny odjazdu.

9. **Tabela transactions (Transakcje przejazdów)**

Zawiera dane zrealizowanych przejazdów, w tym czasy, koszty, statusy i dane trasy.

Brak jednoznacznych wymagań unikalności poza kluczem głównym.

10. **Tabela purchased_tickets (Zakupione bilety)**

Rejestruje informacje o zakupionych i używanych biletach, powiązania z użytkownikami i transakcjami.

Brak jednoznacznych wymagań unikalności poza kluczem głównym.

3.3 Opis powiązań między tabelami

Model danych opiera się na zestawie relacji, które odwzorowują logikę funkcjonowania systemu komunikacji miejskiej. Kluczowe powiązania w bazie danych to:

1. Relacja między liniami a przystankami (N:M)

- Zależność pomiędzy tabelami **lines** i **stops** ma charakter wiele-do-wielu (N:M), ponieważ jedna linia obsługuje wiele przystanków, a jeden przystanek może być częścią wielu różnych linii.
- Relacja ta jest zrealizowana za pomocą tabeli pośredniczącej **linestop**. Zawiera ona klucze obce **line_id** i **stop_id**, a dodatkowa kolumna **sequence** definiuje dokładną kolejność przystanków na trasie danej linii.

2. Relacja między liniami, dniami kursowania a kursami (1:N)

- Tabela **trips** (kursy) jest centralnym punktem rozkładu jazdy. Jest ona powiązana z tabelą **lines** relacją jeden-do-wielu (1:N) poprzez klucz obcy **line_id**. Oznacza to, że jedna linia może mieć zdefiniowanych wiele kursów.
- Podobnie, tabela **trips** jest powiązana z tabelą **operating_days** (1:N) poprzez klucz **day_id**. Dzięki temu ten sam typ dnia (np. "Dni robocze") może być przypisany do wielu różnych kursów na różnych liniach, co umożliwia tworzenie odrębnych rozkładów jazdy na dni powszednie, weekendy czy święta.

3. Relacja między kursami, przystankami a czasami odjazdów (1:N)

- Każdy pojedynczy kurs z tabeli **trips** ma szczegółowy harmonogram odjazdów z poszczególnych przystanków, przechowywany w tabeli **departure_times**.
- Relacja jeden-do-wielu (1:N) łączy tabelę **trips** z **departure_times** za pomocą klucza **trip_id**. W tabeli **departure_times** znajduje się również klucz **stop_id**, wskazujący, którego przystanku dotyczy dany wpis.

4. Relacja między użytkownikami a transakcjami i biletami (1:N)

- Tabela **users** jest połączona z tabelami **transactions** i **purchased_tickets** relacjami jeden-do-wielu.
- Jeden użytkownik (**users**) może mieć wiele zarejestrowanych przejazdów (**transactions**). Pole **user_id** w tabeli **transactions** może być opcjonalne, co pozwala na rejestrowanie przejazdów anonimowych.
- Jeden użytkownik może również posiadać wiele zakupionych biletów (**purchased_tickets**).

5. Relacja między typami biletów a zakupionymi biletami (1:N)

- Tabela **tickets** (definiująca ofertę biletową) jest połączona z tabelą **purchased_tickets** relacją jeden-do-wielu (1:N) przez klucz **ticket_id**. Oznacza to, że jeden typ biletu (np. "Bilet 24-godzinny") może być zakupiony wielokrotnie przez różnych użytkowników.

6. Relacja między transakcjami a zakupionymi biletami (1:1 / 1:N)

- Tabela **purchased_tickets** jest opcjonalnie powiązana z tabelą **transactions** poprzez klucz obcy **transaction_id**. Pozwala to powiązać konkretny zakupiony bilet z transakcją przejazdu, podczas którego został on wykorzystany lub zakupiony.

Powyższe powiązania tworzą spójny model logiczny, który umożliwia efektywne zarządzanie siatką połączeń, rozkładami jazdy, ofertą biletową oraz historią przejazdów użytkowników.

4. GUI

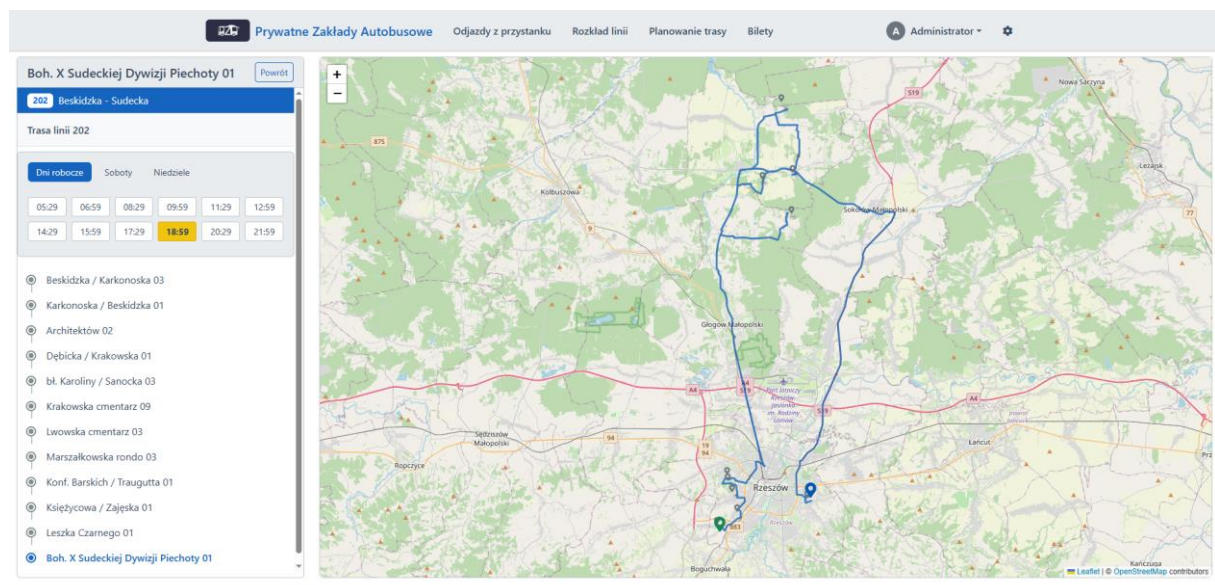
Poniżej zaprezentowano zrzuty ekranu oraz opisy trzech kluczowych widoków aplikacji, które ilustrują jej główne funkcjonalności.

4.1. Widok szczegółów przystanku i rozkładu jazdy

Widok ten jest jednym z podstawowych ekranów dla pasażera. Po lewej stronie ekranu znajduje się panel informacyjny. W jego nagłówku widnieje nazwa wybranego przystanku oraz przycisk pozwalający na powrót do listy wszystkich przystanków. Poniżej wyświetlana jest lista wszystkich linii autobusowych, które obsługują dany przystanek. Użytkownik może wybrać interesującą go linię.

Po dokonaniu wyboru, w panelu pojawia się szczegółowy rozkład jazdy dla tej linii, podzielony na zakładki: „Dni robocze”, „Soboty” i „Niedziele”. Godziny odjazdów wyświetlane są w formie siatki, a najbliższy nadchodzący odjazd jest wyróżniony. Dodatkowo, pod rozkładem prezentowana jest pełna trasa wybranej linii w formie listy przystanków, z wizualnym zaznaczeniem aktualnie przeglądanego.

Główną część ekranu (po prawej stronie) zajmuje interaktywna mapa, na której zaznaczona jest lokalizacja wybranego przystanku oraz pełna trasa aktywnej linii autobusowej.



Rysunek 2. Szczegóły przystanku i rozkład jazdy

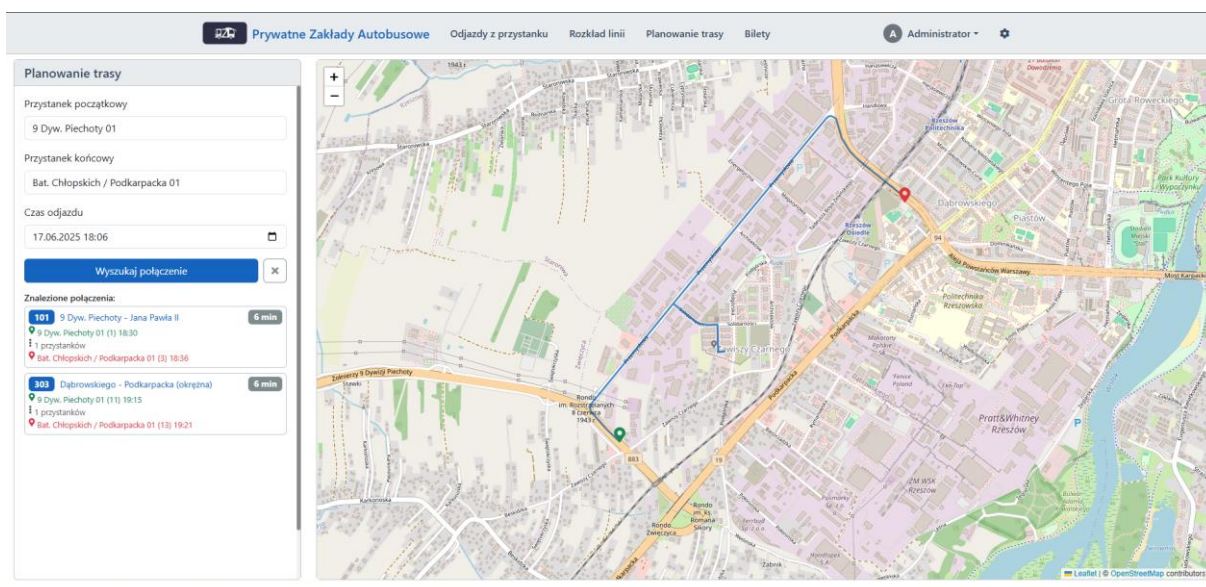
4.2. Widok planowania trasy

Ekran planowania trasy umożliwia użytkownikom wyszukiwanie połączeń między dwoma dowolnymi punktami w siatce komunikacyjnej. Po lewej stronie znajduje się formularz, w którym użytkownik musi uzupełnić:

- **Przystanek początkowy** – pole tekstowe z dynamicznymi podpowiedziami, ułatwiające znalezienie właściwego przystanku.
- **Przystanek końcowy** – pole działające na tej samej zasadzie co przystanek początkowy.
- **Czas odjazdu** – pole wyboru daty i godziny.

Po wyszukaniu, poniżej formularza wyświetlana jest lista proponowanych połączeń. Każda propozycja zawiera informacje o numerze i nazwie linii, szacowanym czasie przejazdu, godzinie odjazdu i przyjazdu, a także liczbie przystanków pośrednich. Użytkownik może kliknąć na wybraną propozycję, aby zobaczyć jej przebieg na mapie.

Po prawej stronie znajduje się mapa, która domyślnie wyświetla wszystkie przystanki w systemie. Po znalezieniu i wybraniu trasy, mapa wizualizuje przebieg połączenia, zaznaczając przystanek początkowy, końcowy oraz trasę przejazdu.



Rysunek 3. Widok planowania trasy

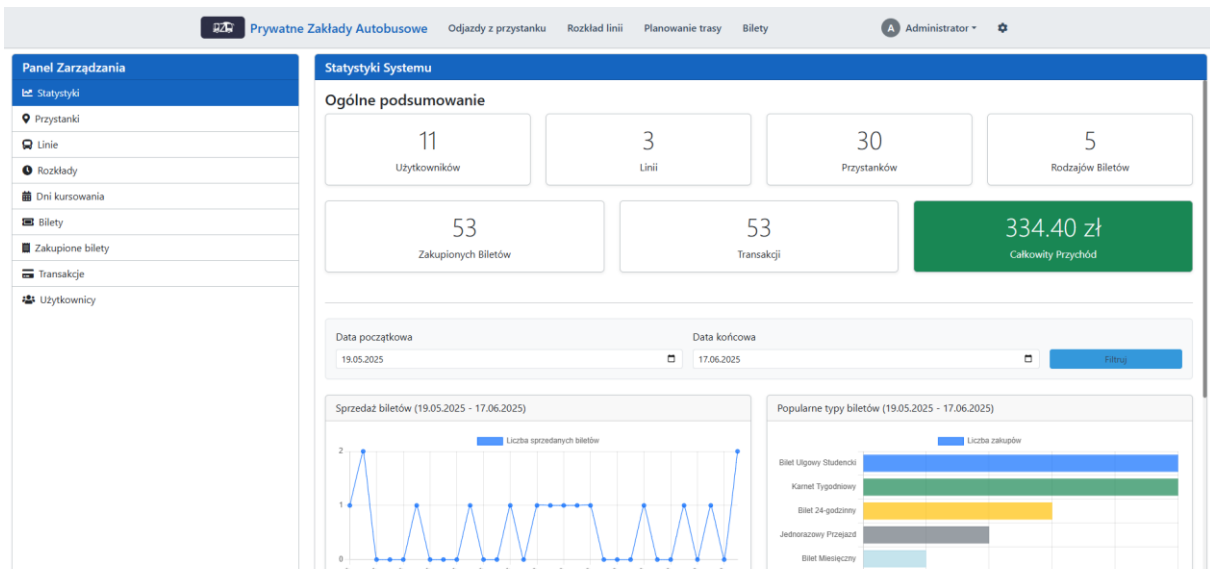
4.3. Widok panelu administracyjnego

Panel administracyjny jest centralnym miejscem do zarządzania całym systemem. Interfejs jest podzielony na dwie główne części. Po lewej stronie znajduje się stałe menu nawigacyjne, które umożliwia dostęp do poszczególnych modułów systemu. Każda pozycja w menu opatrzona jest ikoną oraz nazwą, np.:

- Statystyki
- Przystanki
- Linie

- Rozkłady
- Użytkownicy
- Bilety i transakcje

Po prawej, w głównej części ekranu, dynamicznie ładowana jest zawartość odpowiadająca wybranej z menu sekcji. Przykładowo, po kliknięciu w „Linie”, w tym miejscu pojawia się tabela z listą linii autobusowych oraz opcje pozwalające na ich dodawanie, edycję i usuwanie (operacje CRUD). Taka budowa interfejsu pozwala na szybkie i wygodne przełączanie się między zadaniami administracyjnymi bez potrzeby przeładowywania całej strony.



Rysunek 4. Widok panelu administracyjnego

4.4 Dostępność i Użyteczność (RWD i Ułatwienia Dostępu)

Projekt został zrealizowany z myślą o zapewnieniu szerokiej dostępności i komfortu użytkowania dla wszystkich pasażerów, w tym osób z niepełnosprawnościami wzroku. W tym celu zaimplementowano kilka kluczowych mechanizmów.

1. Responsywność (Responsive Web Design):

Interfejs aplikacji został zbudowany w oparciu o framework **Bootstrap 5**. Wykorzystanie jego systemu siatki (grid system: container, row, col-*) oraz responsywnych komponentów (np. nav, card) zapewnia, że układ strony dynamicznie dostosowuje się do rozmiaru ekranu. Aplikacja jest w pełni funkcjonalna i czytelna zarówno na dużych monitorach komputerowych, jak i na tabletach oraz smartfonach.

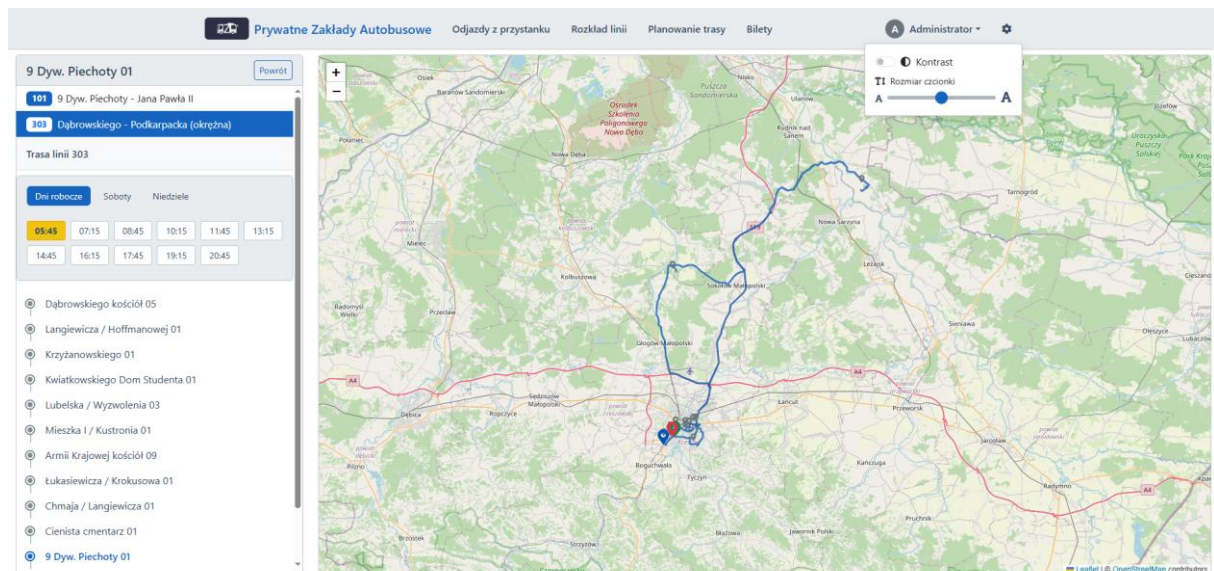
2. Kontrast i Czytelność:

Dobór kolorów w aplikacji został podyktowany wytycznymi **WCAG 2.1** na poziomie AA. Zapewniono, że stosunek kontrastu między tekstem a tłem wynosi co najmniej **4.5:1** dla standardowego tekstu, co gwarantuje jego czytelność dla osób słabowidzących.

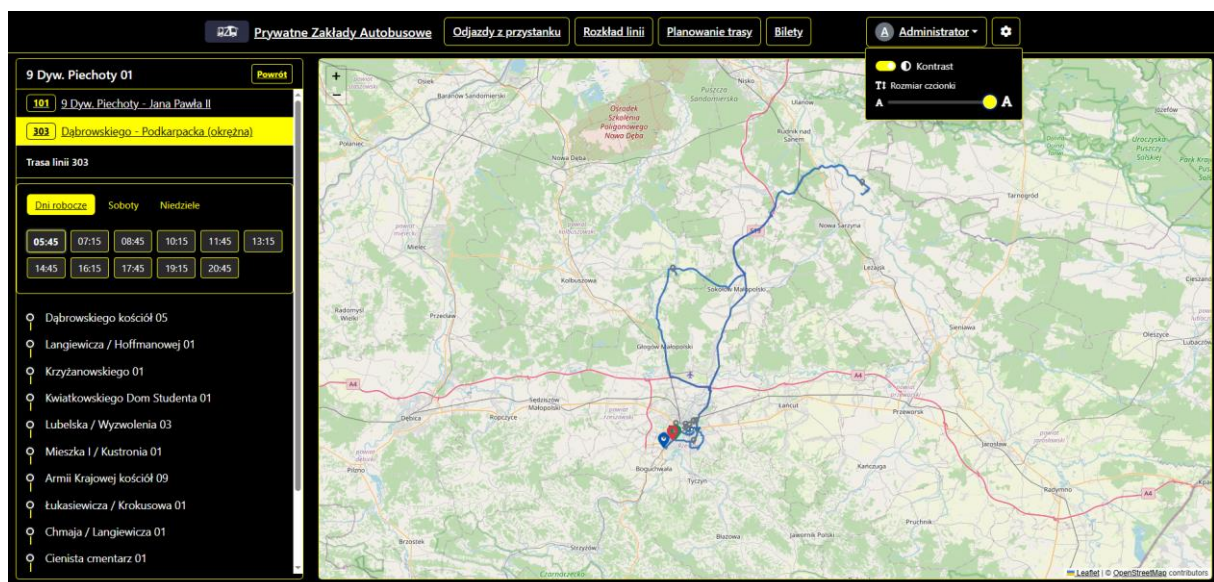
3. Dodatkowe Funkcjonalności Dostępności:

W interfejsie umieszczono dodatkowe narzędzia pozwalające na personalizację widoku:

- **Zmiana rozmiaru czcionki:** Użytkownik za pomocą dedykowanego suwaka może płynnie powiększać lub zmniejszać rozmiar tekstu na stronie, dostosowując go do swoich indywidualnych potrzeb.
- **Tryb wysokiego kontrastu:** Specjalny przełącznik (toggle button) umożliwia włączenie trybu wysokiego kontrastu, co jest standardowym ułatwieniem dla osób z poważnymi wadami wzroku.



Rysunek 5. Dostępność i Użyteczność



Rysunek 6. Dostępność i Użyteczność

5. Uruchomienie aplikacji

W tym rozdziale opisano kroki niezbędne do poprawnego skonfigurowania i uruchomienia projektu na lokalnym środowisku deweloperskim.

5.1. Wymagania wstępne

Przed przystąpieniem do instalacji należy upewnić się, że na komputerze zainstalowane jest następujące oprogramowanie:

1. **XAMPP** (lub inne środowisko dostarczające serwer Apache, MySQL i PHP)
 - Wymagana wersja **PHP: 8.2** lub nowsza.
 - Usługi **Apache** oraz **MySQL** muszą być uruchomione przed rozpoczęciem instalacji.
2. **Composer** – menedżer zależności dla PHP, dostępny globalnie w wierszu poleceń.
3. **Stripe CLI** – narzędzie wiersza poleceń od Stripe, wymagane do testowania funkcjonalności płatności.
 - Należy je pobrać z oficjalnej strony Stripe.
 - Po instalacji, należy jednorazowo autoryzować komputer, wykonując w terminalu komendę `stripe login` i postępując zgodnie z instrukcjami w przeglądarce.

5.2. Instalacja i konfiguracja

Poniżej przedstawiono dwie metody instalacji: automatyczną (zalecana) oraz manualną.

W głównym katalogu projektu znajduje się skrypt `start.bat`, który automatyzuje większość procesu instalacyjnego.

1. Rozpakuj archiwum z kodem źródłowym projektu w wybranej lokalizacji.
2. Uruchom pakiet XAMPP i upewnij się, że moduły Apache i MySQL są aktywne.
3. Uruchom plik `start.bat`. Skrypt wykona następujące czynności:
 - Sprawdzi, czy usługi XAMPP są uruchomione i utworzy bazę danych `bustimetable`.
 - Zainstaluje zależności PHP za pomocą `composer update`.
 - Skopiuje plik `.env.example` do `.env` i otworzy go w Notatniku. **W tym momencie należy uzupełnić klucze API Stripe (patrz sekcja 5.3)** oraz zweryfikować dane dostępowe do bazy danych.
 - Wygeneruje klucz aplikacji, uruchomi migracje z seederami i utworzy dowiązanie symboliczne.
 - Uruchomi serwer deweloperski (`php artisan serve`) oraz nasłuchiwanie webhooków Stripe (`stripe listen`).
 - Otworzy stronę aplikacji w domyślnej przeglądarce.

Jeśli instalacja automatyczna nie powiedzie się, należy wykonać poniższe kroki w terminalu w głównym katalogu projektu:

1. Rozpakuj kod źródłowy projektu i otwórz terminal w głównym folderze projektu.
2. Skopiuj plik konfiguracyjny: `copy .env.example .env`
3. Skonfiguruj plik `.env`: Otwórz plik i uzupełnij dane dostępowe do bazy danych oraz klucze Stripe (patrz [sekcja 5.3](#)).
4. Zainstaluj zależności PHP: `composer update`
5. Wygeneruj klucz aplikacji: `php artisan key:generate`
6. Uruchom migracje i seedery: `php artisan migrate --seed`
7. Utwórz dowiązanie symboliczne: `php artisan storage:link`
8. Uruchom serwer deweloperski: `php artisan serve`
9. Aplikacja będzie dostępna pod adresem <http://localhost:8000>.

5.3. Konfiguracja płatności Stripe

Aby funkcjonalność płatności działała poprawnie, należy skonfigurować połączenie z własnym kontem testowym Stripe.

1. Pobranie kluczy API:

- Zaloguj się do swojego panelu deweloperskiego Stripe (Stripe Dashboard).
- Przejdź do sekcji **Developers > API keys**.
- Skopiuj Publishable key (klucz publiczny) i wklej go do pliku `.env` jako wartość zmiennej `STRIPE_KEY`.
- Skopiuj Secret key (klucz tajny) i wklej go jako wartość `STRIPE_SECRET`.

2. Konfiguracja Webhooka:

- W panelu Stripe przejdź do **Developers > Webhooks**.
- Kliknij **Add an endpoint**.
- W polu Endpoint URL wpisz: `http://localhost:8000/stripe/webhook`
- W sekcji **Select events to listen to** wybierz zdarzenie `checkout.session.completed`.
- Po utworzeniu punktu końcowego, na stronie jego szczegółów znajdź sekcję **Signing secret**. Skopiuj ten klucz i wklej go do pliku `.env` jako wartość `STRIPE_WEBHOOK_SECRET`.

3. Uruchomienie nasłuchiwania (jeśli nie użyto skryptu start.bat):

- Po uruchomieniu serwera aplikacji (php artisan serve), otwórz **nowy terminal** i wykonaj komendę:

```
stripe listen --forward-to http://localhost:8000/stripe/webhook
```

- Ten proces musi być aktywny w tle, aby serwery Stripe mogły komunikować się z lokalną aplikacją.

5.4. Przykładowe dane logowania

Po poprawnym wykonaniu migracji z opcją --seed, w bazie danych zostaną utworzone przykładowe konta użytkowników:

- **Konto Administratora:**
 - **Login:** admin@example.com
 - **Hasło:** password123
- **Konto Zwykłego Użytkownika:**
 - Seeder tworzy również kilku losowych użytkowników. Można zalogować się na dowolne z tych kont, np.:
 - **Login:** clindgren@example.com
 - **Hasło:** password123

6. Funkcjonalność aplikacji

Aplikacja oferuje zestaw funkcjonalności zarówno dla zwykłych użytkowników, jak i administratorów. Poniżej przedstawiam główne możliwości wraz z wizualną prezentacją działania.

6.1 Proces rejestracji i logowania użytkownika

The screenshot shows the login interface of a web application. At the top, a navigation bar contains the company logo and name 'Prywatne Zakłady Autobusowe', followed by menu items: 'Odjazdy z przystanku', 'Rozkład linii', 'Planowanie trasy', and 'Bilety'. On the right side of the bar are links for 'Logowanie' and a settings icon. The main content area features a modal window with two tabs: 'Logowanie' (active) and 'Rejestracja'. The 'Logowanie' tab contains an 'Email' input field, a 'Hasło' (password) input field, and a blue 'Zaloguj' button. Below the button is a link: 'Nie masz jeszcze konta? [Zarejestruj się](#)'.

Rysunek 7. Logowanie użytkownika

Widok formularza logowania, umożliwiający autoryzację użytkownika za pomocą adresu e-mail oraz hasła. Po poprawnym uwierzytelnieniu użytkownik zostaje przekierowany do panelu z funkcjami zgodnymi z przypisaną rolą (zwykły użytkownik lub administrator). Mechanizm logowania obsługuje również komunikaty błędów w przypadku nieprawidłowych danych.

The screenshot shows the registration interface of the same web application. The navigation bar is identical to the one in the previous image. The modal window has the 'Rejestracja' tab selected. This tab contains input fields for 'Imię', 'Email', 'Hasło', and 'Potwierdź hasło'. A blue 'Zarejestruj' button is positioned below these fields. At the bottom of the modal, there is a link: 'Masz już konto? [Zaloguj się](#)'.

Rysunek 8. Rejestracja użytkownika

Formularz rejestracyjny służący do tworzenia nowego konta użytkownika. Wymaga podania unikalnego adresu e-mail, hasła oraz jego potwierdzenia. Zintegrowany system walidacji danych uniemożliwia rejestrację przy użyciu już zarejestrowanego adresu e-mail lub niespełniającego wymagań hasła.

6.2 Przykładowy CRUD przeprowadzany przez administratora

Prywatne Zakłady Autobusowe | Odjazdy z przystanku | Rozkład linii | Planowanie trasy | Bilety | Administrator

Dodaj przystanek

Nazwa przystanku

Szerokość geograficzna

Długość geograficzna

Wybierz lokalizację na mapie

Kliknij na mapę, aby ustawić współrzędne.

Zapisz Anuluj

Rysunek 9. Dodawanie przystanku

Panel administracyjny umożliwiający wprowadzenie nowego przystanku do systemu. Administrator podaje nazwę przystanku oraz współrzędne geograficzne (szerokość i długość geograficzna). System wymusza unikalność nazwy przystanku i sprawdza poprawność danych wejściowych.

Prywatne Zakłady Autobusowe | Odjazdy z przystanku | Rozkład linii | Planowanie trasy | Bilety | Administrator

Edytuj przystanek

Nazwa przystanku

Bat. Chłopskich / Podkarpacka 01

Szerokość geograficzna

50.023292

Długość geograficzna

21.980975

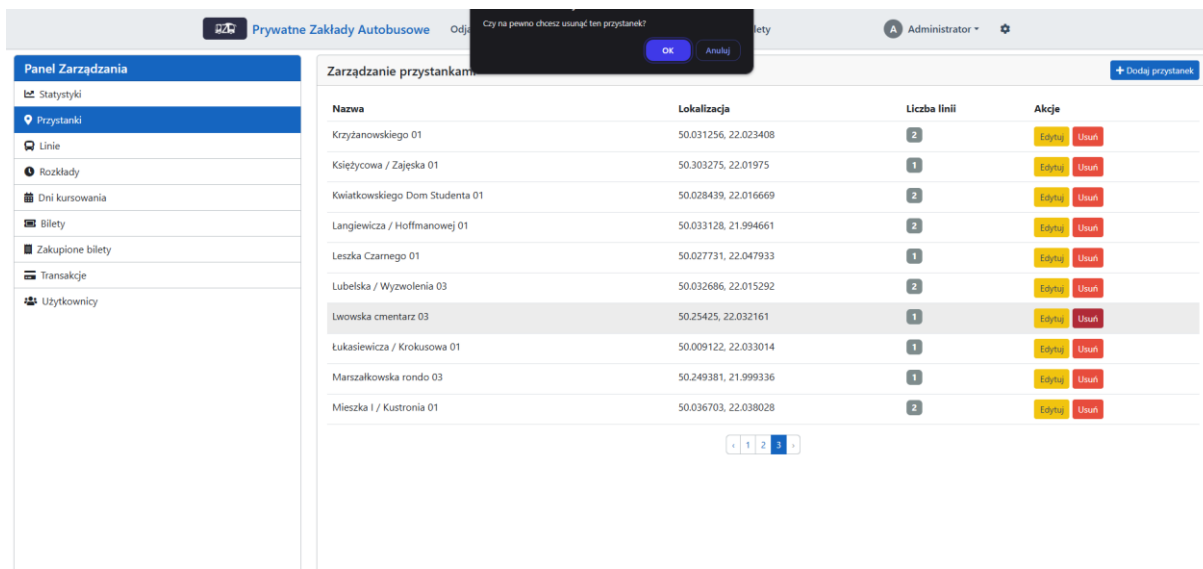
Wybierz lokalizację na mapie

Kliknij na mapę, aby ustawić współrzędne.

Zapisz zmiany Anuluj

Rysunek 10. Edycja przystanku

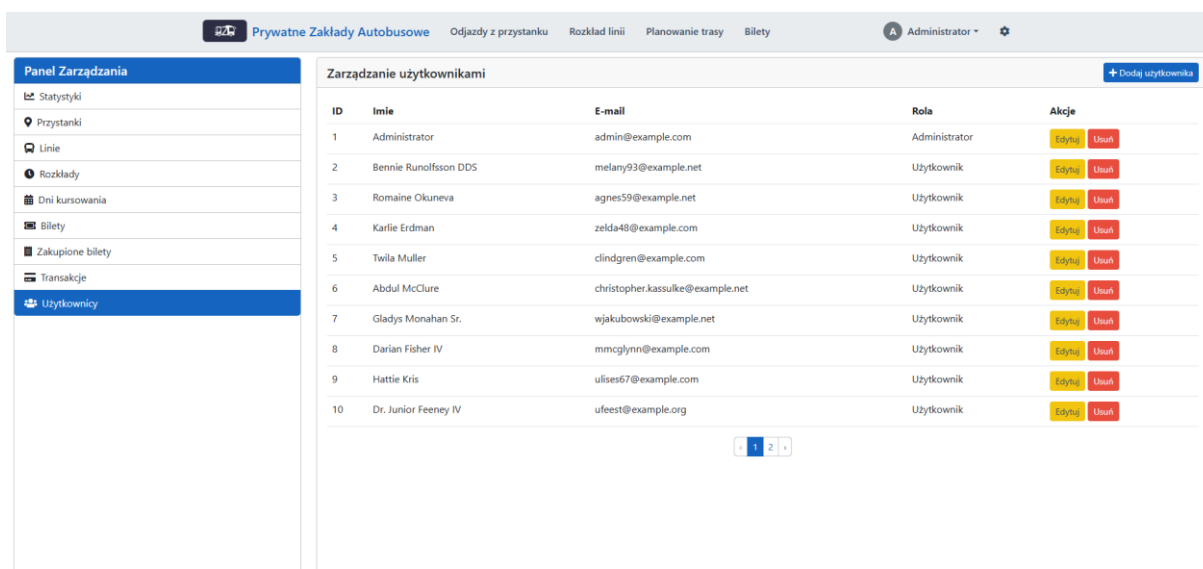
Widok edycji istniejącego przystanku z możliwością aktualizacji jego danych. Interfejs wspiera mechanizmy walidacji i zapobiega konfliktom nazw z innymi rekordami w bazie danych. Po zapisaniu zmian przystanek jest automatycznie aktualizowany w systemie.



Rysunek 11. Usuwanie przystanku

Funkcjonalność umożliwia trwałé usunięcie przystanku z bazy danych. System wymaga potwierdzenia operacji oraz obsługuje zabezpieczenia przed przypadkowym usunięciem danych. Operacja wpływa na powiązania w trasach i kursach.

6.3 Zarządzanie użytkownikami przez administratora



Rysunek 12. Przeglądanie użytkowników

Tabela użytkowników systemu dostępna z poziomu panelu administracyjnego. Umożliwia wyszukiwanie i podgląd podstawowych danych kont, takich jak rola czy e-mail.

Rysunek 13. Dodawanie użytkownika

Prywatne Zakłady Autobusowe

Odjazdy z przystanku

Rozkład linii

Planowanie trasy

Bilety

Administrator

Edytuj użytkownika

Zdjęcie profilowe (opcjonalnie)

Wybierz plik

Nie wybrano pliku

Maksymalny rozmiar 2MB. Dozwolone formaty: JPG, PNG.

Imię i nazwisko

Bernie Runolfsson DDS

Adres e-mail

melary93@example.net

Nowe hasło (pozostaw puste, aby nie zmieniać)

Minimum 8 znaków. Wypełnij tylko jeśli chcesz zmienić hasło.

Potwierdzenie nowego hasła

☐ Administrator

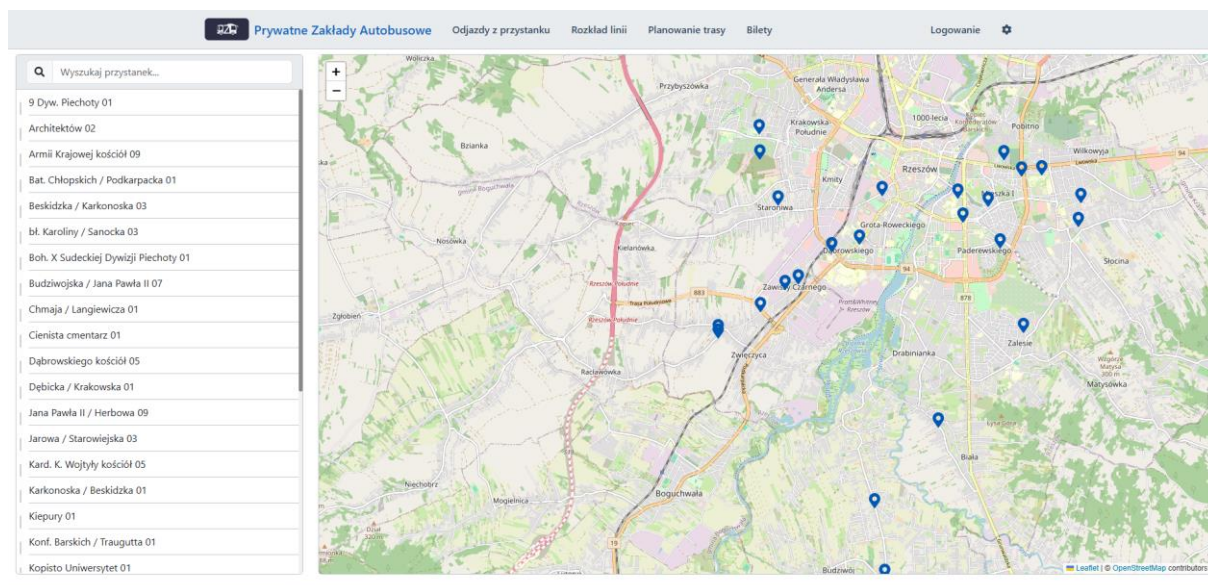
Zapicz zmiany

Anuluj

Rysunek 14. Edycja użytkownika

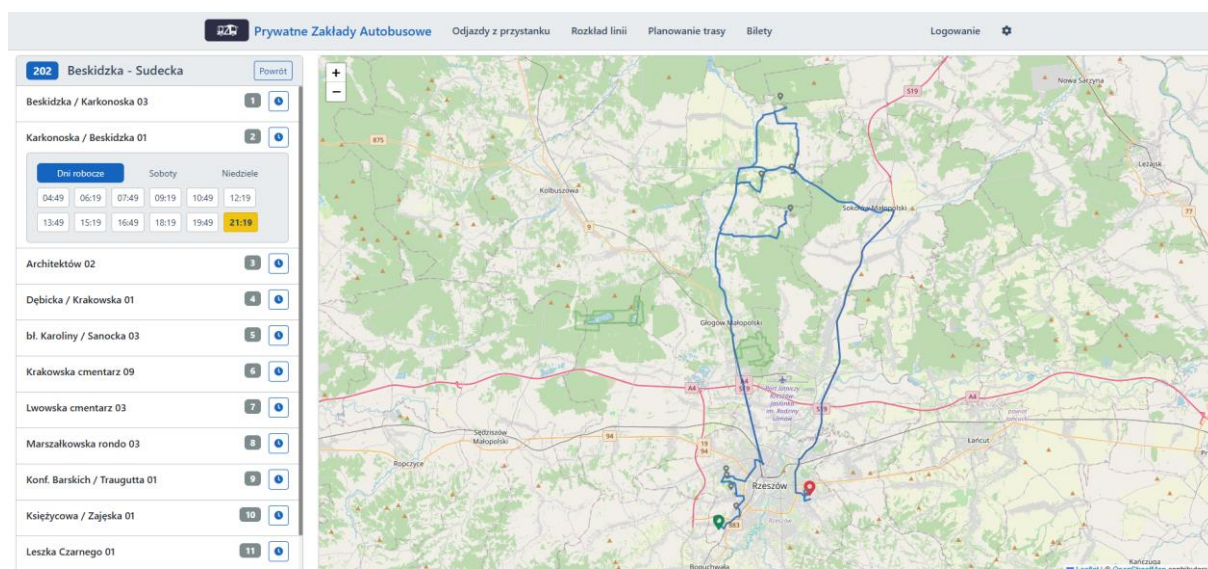
Interfejs umożliwiający modyfikację danych konta użytkownika, w tym zmianę roli oraz reset hasła. Edytowane dane są weryfikowane, a zmiany zapisywane bezpośrednio w bazie danych.

6.4 Przeglądanie ogólnodostępnych zasobów (bez logowania)



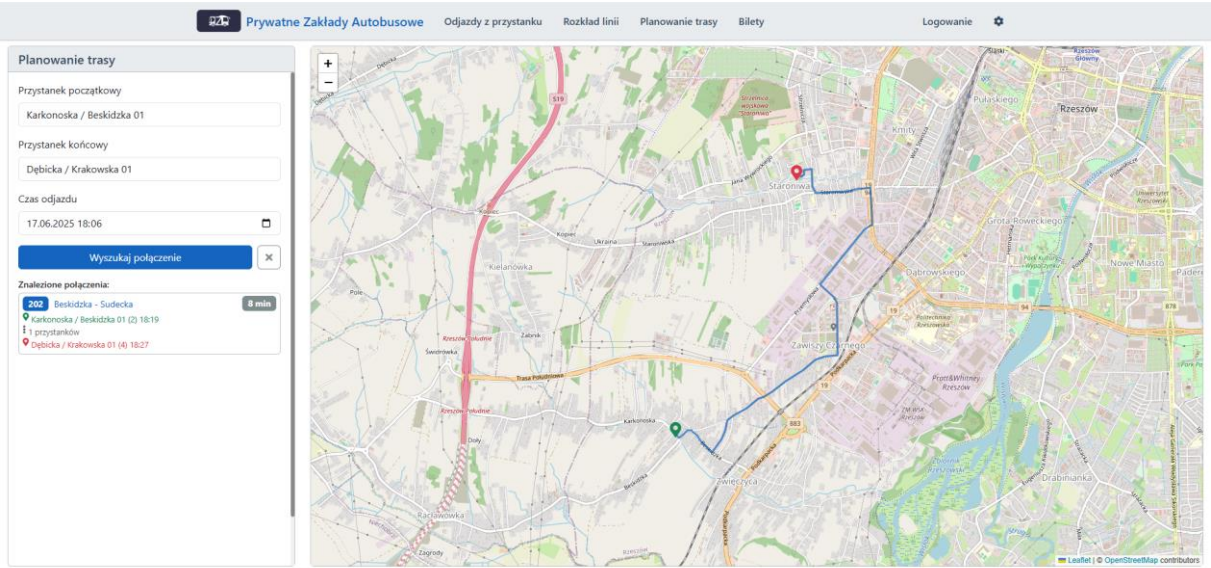
Rysunek 15. Odjazdy z przystanku.

Publiczny widok przedstawiający nadchodzące odjazdy z wybranego przystanku. Użytkownik ma możliwość szybkiego sprawdzenia godzin i tras bez potrzeby logowania. Dane są pobierane dynamicznie z systemu.



Rysunek 16. Rozkład linii

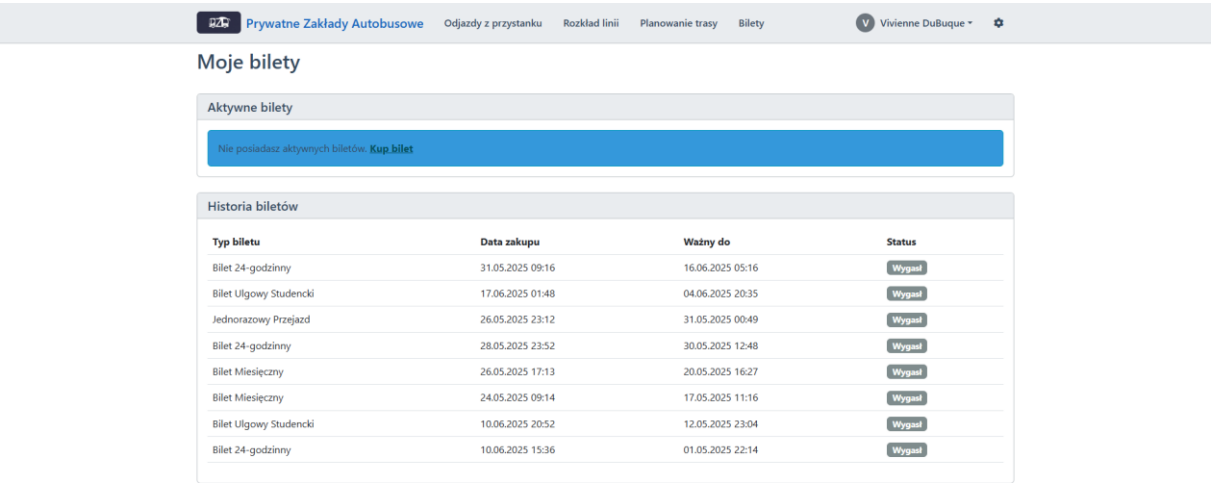
Szczegółowy rozkład jazdy dla wybranej linii autobusowej. Użytkownik może zapoznać się z trasą, godzinami kursów oraz przystankami pośrednimi. Rozkład dostosowuje się do wybranych dni kursowania.



Rysunek 17. Planowanie trasy

Moduł do planowania przejazdu między dwoma punktami. System uwzględnia aktualne kursy i pozwala wybrać optymalną trasę z uwzględnieniem przesiadek i godzin kursowania.

6.5 Zarządzanie swoimi zasobami przez użytkownika aplikacji (bilety)



Rysunek 18. Moje bilety (użytkownik)

Zakładka z listą aktualnych i archiwalnych biletów zakupionych przez zalogowanego pasażera. Widok zawiera informacje o statusie, ważności oraz powiązaniu z transakcjami.

PZK Prywatne Zakłady Autobusowe Odjazdy z przystanku Rozkład linii Planowanie trasy Bilety Vivienne DuBuque

Zakup biletu

Przystanek początkowy
Karkonoska / Beskidzka 01

Przystanek końcowy
Dębicka / Krakowska 01

Linia
Linia 202: Beskidzka - Sudecka

Rodzaj biletu
Bilet Ulgowy Studencki (2h)

Oblicz cenę i kup bilet

Rysunek 19. Kupowanie biletu przez użytkownika

Etap zakupu biletu, podczas którego użytkownik wybiera typ biletu z listy dostępnych wariantów. Widok zawiera informacje o cenie, czasie ważności i aktywności biletu.

PZK Prywatne Zakłady Autobusowe Odjazdy z przystanku Rozkład linii Planowanie trasy Bilety Vivienne DuBuque

Realizacja płatności

Bilet Ulgowy Studencki

Linia 202: Karkonoska / Beskidzka 01 → Dębicka / Krakowska 01

Dystans: 4.95 km
Szacowany czas przejazdu: 6 min

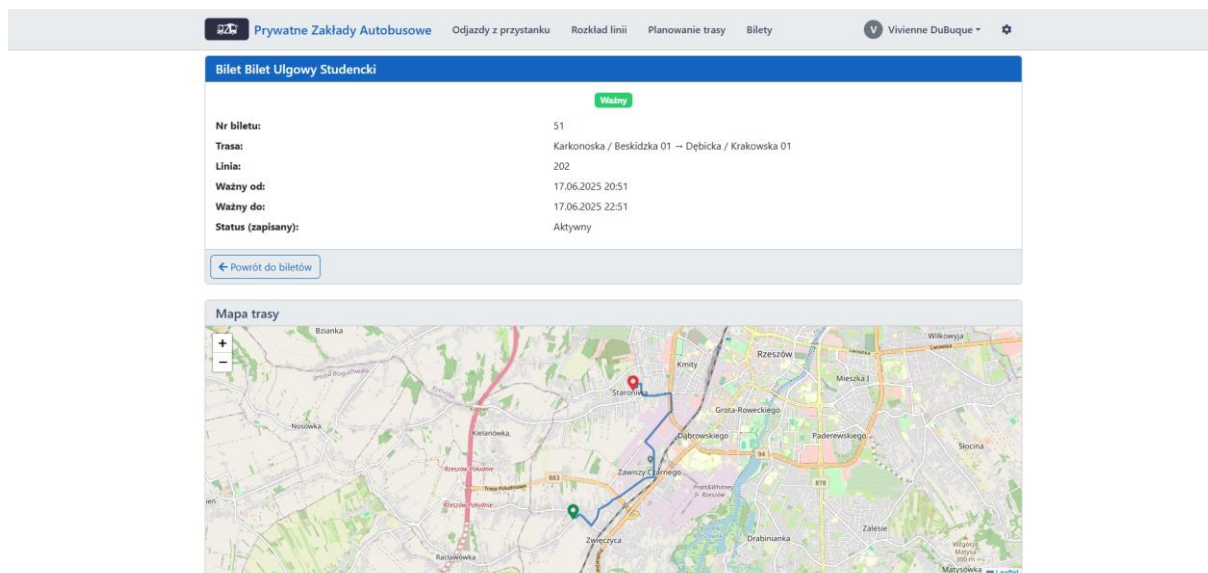
40.00 zł

Zapłać teraz

Anuluj

Rysunek 20. Kupowanie biletu c.d.

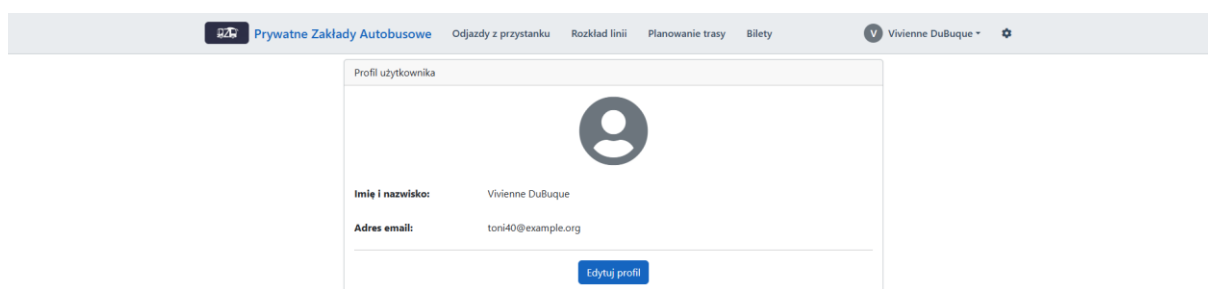
Kontynuacja procesu zakupu – użytkownik potwierdza dane zakupu i przechodzi do płatności. Po zatwierdzeniu operacja zostaje zapisana jako transakcja.



Rysunek 21. Szczegóły biletu

Widok szczegółowy zakupionego biletu, zawierający datę, okres ważności oraz informacje o statusie.

6.6 Zarządzanie swoimi danymi przez użytkownika aplikacji



Rysunek 22. Informacje o profilu

Panel użytkownika prezentujący dane konta, takie jak e-mail, imię i nazwisko oraz zdjęcie profilowe. Użytkownik ma możliwość przejścia do edycji.

Edytuj profil

Zdjęcie profilowe (opcjonalnie)

Wybierz plik avatar.jpg

Maksymalny rozmiar: 2MB. Dozwolone formaty: JPG, PNG.

Imię i nazwisko

Vivienne DuBuque

Adres email

toni40@example.org

Zmień hasło (opcjonalnie)

Obecne hasło

Wypełnij, jeśli chcesz zmienić hasło.

Nowe hasło

Potwierdź nowe hasło

Rysunek 23. Edycja profilu

Widok umożliwiający zmianę danych osobowych oraz aktualizację zdjęcia profilowego i hasła. Pola wejściowe są objęte mechanizmem walidacji.

Profil został zaktualizowany.

Profil użytkownika

Imię i nazwisko: Vivienne DuBuque

Adres email: toni40@example.org

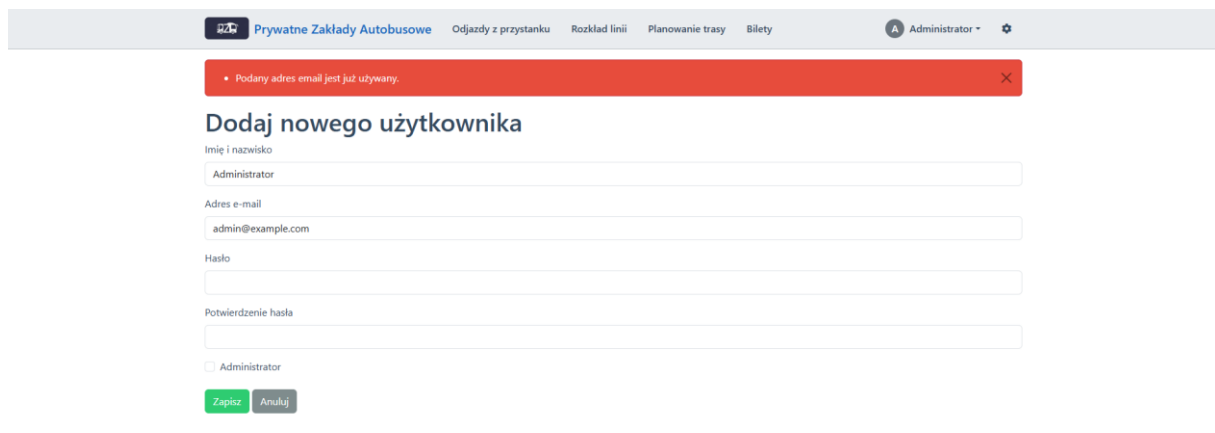
Edytuj profil

Rysunek 24. Wykonanie edycji profilu przez użytkownika

Zatwierdzenie wprowadzonych zmian i ich zapisanie w systemie. Interfejs prezentuje komunikaty potwierdzające powodzenie operacji.

6.7 Walidacja danych

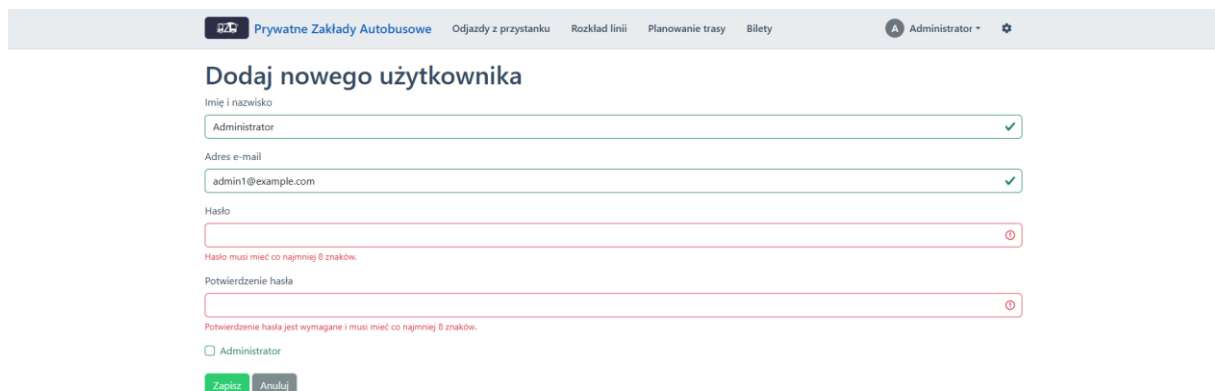
6.7.1 Przykłady walidacji w widokach



The screenshot shows a web application interface for adding a new user. At the top, there is a navigation bar with the logo 'PZK Prywatne Zakłady Autobusowe' and several menu items: 'Odjazdy z przystanku', 'Rozkład linii', 'Planowanie trasy', and 'Bilety'. The user is logged in as 'Administrator'. A red banner at the top of the form area displays the message: 'Podany adres email jest już używany.' (The provided email address is already in use). The form itself is titled 'Dodaj nowego użytkownika' and contains the following fields: 'Imię i nazwisko' (Name and surname) with the value 'Administrator', 'Adres e-mail' (Email address) with the value 'admin@example.com', 'Hasło' (Password), and 'Potwierdzenie hasła' (Confirm password). There is also a checkbox for 'Administrator' which is currently unchecked. At the bottom of the form are two buttons: 'Zapisz' (Save) and 'Anuluj' (Cancel).

Rysunek 25. Unique dla adresu email na backendzie (dodawanie użytkownika)

Backendowa walidacja unikalności adresu e-mail w bazie danych podczas dodawania konta. Zapobiega duplikacji użytkowników.



This screenshot shows the same 'Dodaj nowego użytkownika' form, but with validation feedback. The 'Imię i nazwisko' and 'Adres e-mail' fields now have green checkmarks on the right, indicating they are valid. The 'Hasło' (Password) and 'Potwierdzenie hasła' (Confirm password) fields have red borders and red error icons on the right. Below the password field, a red message states: 'Hasło musi mieć co najmniej 8 znaków.' (Password must be at least 8 characters long). Below the confirm password field, a red message states: 'Potwierdzenie hasła jest wymagane i musi mieć co najmniej 8 znaków.' (Confirm password is required and must be at least 8 characters long). The 'Administrator' checkbox remains unchecked, and the 'Zapisz' and 'Anuluj' buttons are still present at the bottom.

Rysunek 26. Długość hasła na frontendzie i backendzie (dodawanie użytkownika)

Walidacja długości hasła realizowana równoległe na warstwie frontendowej i backendowej. Wymuszony minimalny poziom złożoności.

Prywatne Zakłady Autobusowe

Odjazdy z przystanku

Rozkład linii

Planowanie trasy

Bilety

Administrator

Potwierdzenie pola hasło nie zgadza się.

Dodaj nowego użytkownika

Imię i nazwisko

Administrator

Adres e-mail

admin1@example.com

Hasło

Potwierdzenie hasła

☐ Administrator

Zapisz

Anuluj

Rysunek 27. Potwierdzenie hasła na backendzie (dodawanie użytkownika)

Kontrola zgodności pól „hasło” i „potwierdź hasło” po stronie serwera. Zapobiega zapisaniu niezgodnych danych.

Prywatne Zakłady Autobusowe

Odjazdy z przystanku

Rozkład linii

Planowanie trasy

Bilety

Administrator

Przystanek o tej nazwie już istnieje.

Dodaj przystanek

Nazwa przystanku

9 Dyw. Piechoty 01

Szerokość geograficzna

49.440896

Długość geograficzna

22.258301

Zapisz

Anuluj

Wybierz lokalizację na mapie

Kliknij na mapę, aby ustawić współrzędne.

Rysunek 28. Unique dla nazwy przystanku na backendzie (dodawanie przystanku)

Sprawdzenie unikalności nazwy przystanku w momencie dodawania nowej lokalizacji do systemu.

Prywatne Zakłady Autobusowe

Odjazdy z przystanku

Rozkład linii

Planowanie trasy

Bilety

A

Administrator

Kombinacja numeru linii i kierunku musi być unikalna.

Dodaj nową linię autobusową

Numer linii

101

Kombinacja numeru linii i kierunku musi być unikalna.

Nazwa linii

Kierunek (opcjonalnie)

Np. "Zajeźdźnia - Rynek" lub "Pętla"

Przystanki na trasie

| Przystanek | Kolejność | Akcje |
|--------------------|-----------|-----------------|
| 9 Dyw. Piechoty 01 | 1 | <div>Usuń</div> |
| Architektów 02 | 2 | <div>Usuń</div> |

Wybierz przystanek

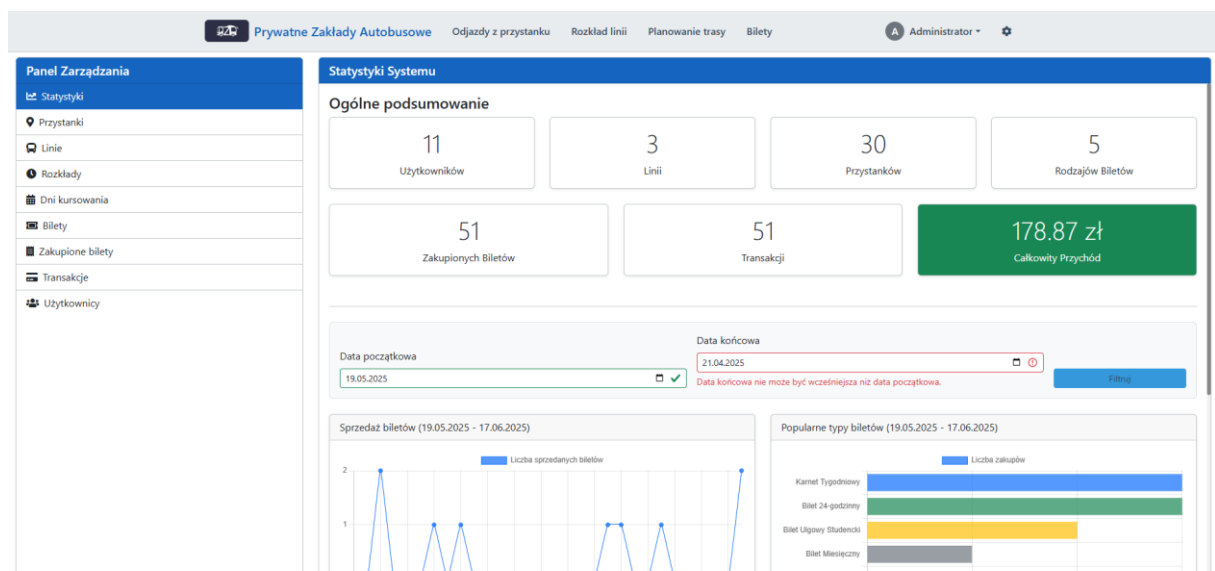
Dodaj przystanek

Zapisz linię

Anuluj

Rysunek 29. Unique dla linii i kombinacji na backendzie (dodawanie linii autobusowej)

Walidacja unikalności kombinacji numeru linii i kierunku – zapobiega duplikatom tras.



Rysunek 30. Obsługa daty na frontendzie i backendzie (statystyki w panelu administratora)

Synchronizacja formatu daty i jej zakresów między frontendem a backendem w widoku generowania statystyk.

Prywatne Zakłady Autobusowe Odjazdy z przystanku Rozkład linii Planowanie trasy Bilety Administrator

Nazwa jest już zajęta.

Dodaj nowy dzień kursowania

Nazwa dnia kursowania

Dni robocze

Opis

Np. "Dni robocze", "Soboty, niedziele i święta"

Zapisz Anuluj

Rysunek 31. Unique dla nazw dni kursowania (dodawanie dnia kursowania)

Kontrola unikalności nazw typów dni w bazie danych. System nie pozwala na zduplikowanie nazw takich jak „Dni robocze”.

Prywatne Zakłady Autobusowe Odjazdy z przystanku Rozkład linii Planowanie trasy Bilety Administrator

Dodaj nowy bilet

Nazwa biletu

Bilet Miesięczny

Opis

Opis biletu

Cena (zł)

-1

Cena jest wymagana i nie może być ujemna.

Ważność (godziny)

1.5

Ważność jest wymagana i musi wynosić co najmniej 1 godzinę (liczba całkowita).

☒ Aktywny

Zapisz Anuluj

Rysunek 32. Cena nieujemna i ważność jako liczba całkowita na backendzie (dodawanie biletu)

Walidacja atrybutów biletu: cena musi być większa lub równa zero, a ważność wyrażona jako liczba całkowita (liczba godzin).

6.7.2. Walidacja po stronie serwera (Server-Side Validation)

Sercem walidacji serwerowej są dedykowane klasy **Form Request**, znajdujące się w katalogu `app/Http/Requests/`. Każda z tych klas rozszerza `Illuminate\Foundation\Http\FormRequest` i jest odpowiedzialna za walidację danych przychodzących z konkretnych formularzy. Takie podejście pozwala na oddzielenie logiki walidacji od logiki kontrolerów, co czyni kod czystszy i łatwiejszym w utrzymaniu.

Przykładowe klasy **Form Request** wykorzystywane w projekcie to:

- `RegisterRequest` (formularz rejestracji)
- `LoginRequest` (formularz logowania)
- `StoreStopRequest` / `UpdateStopRequest` (zarządzanie przystankami)
- `StoreLineRequest` / `UpdateLineRequest` (zarządzanie liniami)
- `StoreUserRequest` / `UpdateUserRequest` (zarządzanie użytkownikami)
- `RouteSearchRequest` (wyszukiwarka połączeń)
- `StatisticsFilterRequest` (filtrowanie statystyk)

W metodzie `rules()` każdej z tych klas zdefiniowane są reguły walidacji. Aplikacja korzysta z szerokiego wachlarza wbudowanych reguł Lavela, co przedstawiono w poniższej tabeli.

| Kategoria | Reguła | Opis i Zastosowanie w Projekcie |
|-------------|---|---|
| Podstawowe | <code>required</code> | Wymusza, aby pole nie było puste. Stosowane m.in. w <code>LoginRequest</code> . |
| | <code>nullable</code> | Zezwala, aby pole było puste. Stosowane m.in. dla opcjonalnego pola <code>direction</code> w <code>StoreLineRequest</code> . |
| Typy danych | <code>string</code> , <code>integer</code> , <code>numeric</code> , <code>boolean</code> , <code>array</code> | Zapewnia, że dane mają oczekiwany typ (np. <code>numeric</code> dla współrzędnych geograficznych w <code>StoreStopRequest</code>). |
| | <code>email</code> | Sprawdza, czy wartość jest poprawnym adresem e-mail. Stosowane w <code>RegisterRequest</code> . |

| | | |
|-------------------|------------------------------|--|
| | date_format:format | Weryfikuje format daty i czasu (np. Y-m-d\TH:i w RouteSearchRequest). |
| | image, mimes | Weryfikuje, czy przesłany plik jest obrazem o dozwolonym rozszerzeniu (np. jpg, png dla profile_photo). |
| Rozmiar | min:wartość, max:wartość | Określa minimalną i maksymalną długość (dla ciągów znaków) lub wartość (dla liczb). Np. min:8 dla hasła w RegisterRequest. |
| Unikalność | unique:tabela,kolumna | Sprawdza, czy wartość jest unikalna w danej tabeli (np. unikalność adresu e-mail w tabeli users). |
| | Rule::unique()->ignore(\$id) | Wariant reguły unique stosowany przy aktualizacji, aby ignorować sprawdzanie unikalności dla edytowanego rekordu. |
| Relacje | exists:tabela,kolumna | Sprawdza, czy wartość (np. ID) istnieje w innej tabeli, co zapewnia integralność kluczy obcych. |
| | different:pole | Wymusza, aby wartość pola była inna niż wartość innego pola (np. przystanek końcowy musi być inny niż początkowy). |
| Hasła | confirmed | Wymaga, aby w formularzu znajdowało się pole password_confirmation o tej samej wartości co pole password. |

Dodatkowo, w niektórych klasach Form Request zaimplementowana jest metoda `messages()`. Pozwala ona na zdefiniowanie niestandardowych, bardziej przyjaznych dla użytkownika komunikatów błędów, które zastępują domyślne komunikaty Larela.

6.7.3. Walidacja po stronie klienta (Client-Side Validation)

Aby zapewnić użytkownikowi natychmiastową informację zwrotną i odciążyć serwer, aplikacja wykorzystuje również mechanizmy walidacji po stronie klienta:

- **Atrybuty HTML5:** Formularze używają standardowych atrybutów walidacyjnych HTML5, takich jak `required`, `minlength`, `maxlength`, a także typów pól (`type="email"`, `type="number"`, `type="password"`), które zapewniają podstawową walidację w przeglądarce.
- **Atrybut `novalidate`:** Większość formularzy posiada atrybut `novalidate`, który wyłącza domyślne dymki walidacyjne przeglądarki. Pozwala to na przejęcie pełnej kontroli nad sposobem wyświetlania błędów przez skrypty Bootstrapa i komunikaty serwerowe, co zapewnia spójny wygląd aplikacji.
- **Wyświetlanie błędów serwerowych:** Widoki Blade intensywnie korzystają z dyrektywy `@error('nazwa_pola')` ... `@enderror`. W przypadku błędu walidacji zwróconego przez serwer, dyrektywa ta dynamicznie dodaje do pola formularza klasę `.is-invalid` (co powoduje jego wizualne wyróżnienie na czerwono) oraz wyświetla stosowny komunikat w elemencie `<div class="invalid-feedback">`.
- **Walidacja JavaScript:** W bardziej złożonych formularzach, takich jak dodawanie linii z minimalną liczbą przystanków, zaimplementowano dodatkową logikę w JavaScript. Zapobiega ona wysłaniu formularza, jeśli podstawowe warunki nie są spełnione, jeszcze przed wysłaniem zapytania do serwera.

6.7.4. Przykład praktyczny: Formularz Rejestracji/logowania

Doskonałym przykładem synergii obu mechanizmów jest formularz rejestracji/logowania (`resources/views/auth/auth.blade.php`).

```
<form method="POST" action="{{ route('login') }}" class="needs-validation"
novalidate>
    @csrf
    <div class="mb-3">
        <label for="email" class="form-label">Email</label>
        <input type="email" class="form-control"
            id="email" name="email" value="{{ old('email') }}"
required>
        <div class="invalid-feedback">Podaj poprawny adres email.</div>
    </div>
    <div class="mb-3">
        <label for="password" class="form-label">Hasło</label>
        <input type="password" class="form-control"
            id="password" name="password" required>
        <div class="invalid-feedback">Hasło jest wymagane.</div>
    </div>
    <div class="d-grid">
        <button type="submit" class="btn btn-primary">Zaloguj</button>
    </div>
    <div class="mt-3 text-center">
        <p>Nie masz jeszcze konta? <a href="#" id="go-to-
register">Zarejestruj się</a></p>
```

```

        </div>
    </form>
</div>
<div class="tab-pane fade {{ request()->is('register') ? 'show active' : ''
}}" id="register-form" role="tabpanel" aria-labelledby="register-tab">
    <form method="POST" action="{{ route('register') }}" class="needs-
validation" novalidate>
        @csrf
        <div class="mb-3">
            <label for="name" class="form-label">Imię</label>
            <input type="text" class="form-control @error('name') is-
invalid @enderror"
                id="name" name="name" value="{{ old('name') }}" required
maxlength="255">
            <div class="invalid-feedback">Imię jest wymagane.</div>
        </div>
        <div class="mb-3">
            <label for="register_email" class="form-label">Email</label>
            <input type="email" class="form-control @error('email') is-
invalid @enderror"
                id="register_email" name="email" value="{{ old('email')
}}" required maxlength="255">
            <div class="invalid-feedback">Podaj poprawny adres email.</div>
        </div>
        <div class="mb-3">
            <label for="register_password" class="form-label">Hasło</label>
            <input type="password" class="form-control @error('password')
is-invalid @enderror"
                id="register_password" name="password" required
minlength="8">
            <div class="invalid-feedback">Hasło musi mieć co najmniej 8
znaków.</div>
        </div>
        <div class="mb-3">
            <label for="password_confirmation" class="form-label">Potwierdź
hasło</label>
            <input type="password" class="form-control"
                id="password_confirmation" name="password_confirmation"
required minlength="8">
            <div class="invalid-feedback">Potwierdzenie hasła jest wymagane
i musi mieć co najmniej 8 znaków.</div>
        </div>
        <div class="d-grid">
            <button type="submit" class="btn btn-
primary">Zarejestruj</button>
        </div>
        <div class="mt-3 text-center">
            <p>Masz już konto? <a href="#" id="go-to-login">Zaloguj
się</a></p>
        </div>
    </form>

```

W powyższym przykładzie:

- **Walidacja HTML5:** Pola name, email, password i password_confirmation mają atrybut required. Pole password ma dodatkowo minlength="8".
- **Wizualizacja błędów (Bootstrap):** Formularz posiada atrybut novalidate, co integruje go z systemem walidacji Bootstrapa, umożliwiając niestandardowe wyświetlanie komunikatów.

- **Integracja z serwerem (Blade):** Dyrektywa `@error('password')` sprawdza, czy z serwera (z klasy `RegisterRequest.php`) został zwrócony błąd dla pola hasła. Jeśli tak, pole `<input>` otrzymuje klasę `.is-invalid`, a w powiązonym bloku `.invalid-feedback` wyświetlany jest komunikat błędu, np. „Hasło musi mieć co najmniej 8 znaków”.

Dzięki takiemu podejściu, żaden formularz nie zostanie przetworzony przez logikę biznesową aplikacji, dopóki wprowadzone dane nie spełnią pełnych reguł bezpieczeństwa i spójności, określonych po stronie serwera.

Po omówieniu fundamentalnych mechanizmów, takich jak operacje CRUD i walidacja danych, można przejść do analizy najbardziej złożonego procesu w aplikacji. Poniżej przedstawiono szczegółowy, wieloetapowy opis logiki biznesowej stojącej za procesem zakupu biletu, który łączy w sobie interakcję na frontendzie, zaawansowane obliczenia oraz komunikację z usługami zewnętrznymi.

6.8 Logika biznesowa

Poniżej przedstawiono szczegółowy, wieloetapowy proces zakupu biletu na konkretną trasę. Jest to kluczowa funkcjonalność aplikacji, która łączy interakcję użytkownika na frontendzie, złożoną logikę obliczeniową w JavaScript, walidację i kalkulację po stronie serwera PHP oraz integrację z zewnętrznym systemem płatności Stripe.

Krok 1: Wybór trasy przez użytkownika (Frontend)

Proces rozpoczyna się od wyboru parametrów podróży przez użytkownika w dedykowanym formularzu.

- **Widok:** `resources/views/tickets/route-selection.blade.php`
- **Logika:** Użytkownik wybiera przystanek początkowy, końcowy oraz linię z dostępnych opcji. Pola te są dynamicznie aktualizowane dzięki JavaScript (np. po wybraniu przystanków ładowane są tylko te linie, które je obsługują). Na koniec użytkownik wybiera rodzaj biletu, co będzie miało wpływ na cenę bazową.

The image shows a web application interface for bus ticket booking. On the left, there is a form titled "Zakup biletu" (Buy ticket). It includes fields for "Przystanek początkowy" (Start stop) and "Przystanek końcowy" (End stop), a dropdown for "Linia" (Line) showing "Linia 202: Beskidzka - Sudecka", and a dropdown for "Rodzaj biletu" (Ticket type) showing "Bilet 24-godzinny (24h)". A blue button "Oblicz cenę i kup bilet" (Calculate price and buy ticket) is at the bottom of the form. On the right, there is a map of a city area with various bus routes and stops marked with blue pins. The map shows streets, green spaces, and a river. The application is titled "Prywatne Zakłady Autobusowe" (Private Bus Stations) in the header.

Rysunek 33. Formularz wyboru trasy i biletu

```
<form method="POST" action="{{ route('tickets.calculate') }}"
id="ticketForm" class="needs-validation" novalidate>
    @csrf
    <div class="mb-3">
        <label for="fromInput" class="form-label">Przystanek
początkowy</label>
        <div class="position-relative">
            <input type="text" class="form-control @error('from_stop_id')
is-invalid @enderror"
                id="fromInput" placeholder="Wpisz nazwę przystanku"
autocomplete="off" required>
            <input type="hidden" id="from_stop_id" name="from_stop_id"
value="{{ old('from_stop_id') }}" required>
            <div id="fromDropdown" class="dropdown-menu w-100"
style="display: none; max-height: 200px; overflow-y: auto;">
                @foreach($stops as $stop)
                    <div class="dropdown-item from-stop-item"
                        data-stop-id="{{ $stop->stop_id }}"
                        data-stop-name="{{ $stop->stop_name }}">
                        {{ $stop->stop_name }}
                    </div>
                @endforeach
            </div>
            <div class="invalid-feedback">Wybierz przystanek
początkowy.</div>
            @error('from_stop_id')
            <div class="invalid-feedback d-block">{{ $message }}</div>
            @enderror
        </div>
        <div class="mb-3">
            <label for="toInput" class="form-label">Przystanek końcowy</label>
            <div class="position-relative">
                <input type="text" class="form-control @error('to_stop_id') is-
invalid @enderror"
                    id="toInput" placeholder="Wpisz nazwę przystanku"
autocomplete="off" required>
                <input type="hidden" id="to_stop_id" name="to_stop_id"
value="{{ old('to_stop_id') }}" required>
                <div id="toDropdown" class="dropdown-menu w-100"
```

```

style="display: none; max-height: 200px; overflow-y: auto;">
    @foreach($stops as $stop)
        <div class="dropdown-item to-stop-item"
            data-stop-id="{{ $stop->stop_id }}"
            data-stop-name="{{ $stop->stop_name }}">
            {{ $stop->stop_name }}
        </div>
    @endforeach
</div>
<div class="invalid-feedback">Wybierz przystanek końcowy.</div>
</div>
@error('to_stop_id')
<div class="invalid-feedback d-block">{{ $message }}</div>
@enderror
</div>
<div class="mb-3">
    <label for="line_id" class="form-label">Linia</label>
    <select class="form-select @error('line_id') is-invalid @enderror"
id="line_id" name="line_id" disabled required>
        <option value="">Najpierw wybierz przystanki</option>
    </select>
    <div class="invalid-feedback">Wybierz linię.</div>
    @error('line_id')
    <div class="invalid-feedback d-block">{{ $message }}</div>
    @enderror
</div>
<div class="mb-3">
    <label for="ticket_id" class="form-label">Rodzaj biletu</label>
    <select class="form-select @error('ticket_id') is-invalid
@enderror" id="ticket_id" name="ticket_id" required>
        <option value="">Wybierz rodzaj biletu</option>
        @foreach($tickets as $ticket)
            <option value="{{ $ticket->ticket_id }}">{{ $ticket-
>ticket_name }} ({{ $ticket->validity_hours }}h)</option>
        @endforeach
    </select>
    <div class="invalid-feedback">Wybierz rodzaj biletu.</div>
    @error('ticket_id')
    <div class="invalid-feedback d-block">{{ $message }}</div>
    @enderror
</div>
<div class="d-grid">
    <button type="submit" class="btn btn-primary">Oblicz cenę i kup
bilet</button>
</div>
</form>

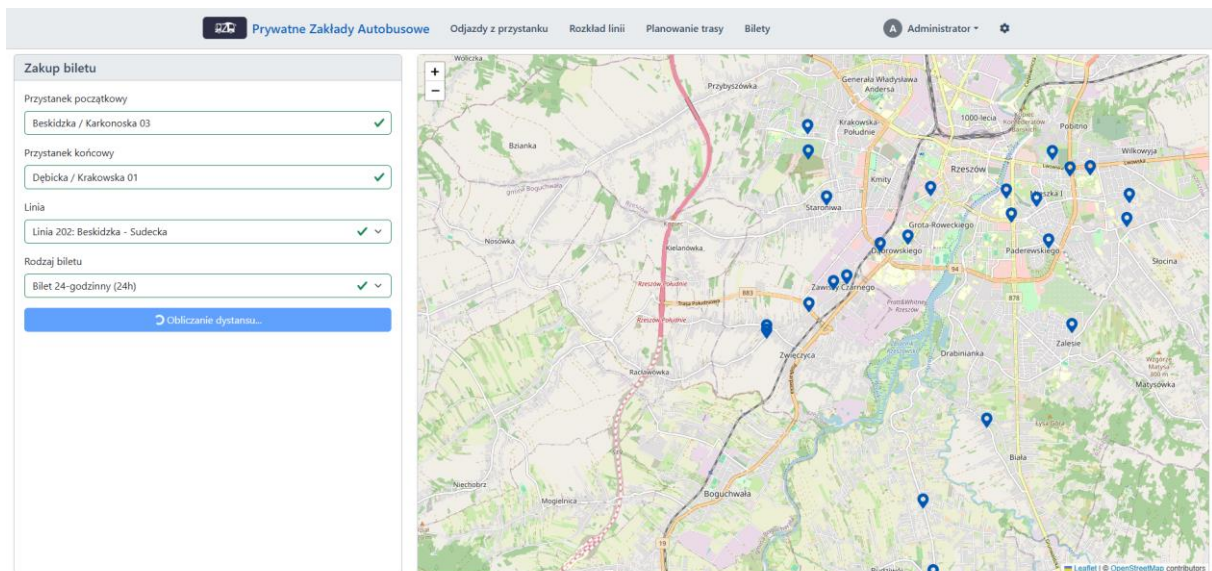
```

Krok 2: Obliczenie dystansu trasy (Frontend - JavaScript)

Po kliknięciu przycisku, formularz nie jest od razu wysyłany. Najpierw JavaScript próbuje obliczyć jak najdokładniejszy dystans trasy w tle, aby cena była bardziej precyzyjna.

- **Plik:** public/js/map.js
- **Logika:**
 1. Przechwytywane jest zdarzenie submit formularza.
 2. Wywoływana jest asynchroniczna metoda calculateRouteDistance().

3. Metoda ta wysyła zapytanie fetch do API (/api/lines/{lineId}/route-segment-stops), aby pobrać wszystkie przystanki na wybranym odcinku trasy.
4. Dla każdego segmentu trasy (odcinka między dwoma kolejnymi przystankami), skrypt próbuje obliczyć dystans drogowy za pomocą zewnętrznego serwisu **OSRM** (przez bibliotekę Leaflet Routing Machine).
5. **Logika awaryjna (Fallback):** Jeśli OSRM nie odpowie w ciągu 5,5 sekundy lub zwróci błąd, dystans dla danego segmentu jest obliczany jako linia prosta (przy użyciu formuły Haversine).
6. Dystanse wszystkich segmentów są sumowane.
7. Wynik (w kilometrach) jest wstawiany do ukrytego pola <input type="hidden" name="route_distance">.
8. Dopiero po pomyślnym obliczeniu dystansu, formularz jest programowo wysyłany na serwer.



Rysunek 34. Obliczanie dystansu

Krok 3: Walidacja i kalkulacja ceny (Backend - PHP)

Serwer otrzymuje dane z formularza wraz z obliczonym dystansem i przeprowadza finalną kalkulację.

- **Kontroler:** app/Http/Controllers/TicketController.php, metoda calculate()
- **Serwis:** app/Http/Services/TicketPriceService.php
- **Logika:**
 1. Żądanie jest walidowane za pomocą TicketCalculationRequest.

2. **Weryfikacja dystansu:** System sprawdza, czy dystans otrzymany z frontendu jest prawidłową wartością liczbową. Jeśli nie, lub jeśli go brakuje, serwer sam oblicza dystans trasy jako sumę linii prostych między przystankami (mechanizm fallback).
3. **Obliczenie czasu trwania:** Czas jest szacowany na podstawie liczby segmentów trasy i średniego czasu na segment.
4. **Obliczenie ceny:** Wywoływany jest serwis TicketPriceService, który kalkuluje ostateczną cenę na podstawie wzoru:
$$\text{Cena końcowa} = \text{Cena bazowa biletu} + (\text{Czas trwania} * \text{Stawka za minutę}) + (\text{Dystans} * \text{Stawka za kilometr})$$
5. Obliczone dane (cena, trasa, czas trwania) są zapisywane w sesji użytkownika.
6. Użytkownik jest przekierowywany do widoku podsumowania zamówienia.

```
public function calculate(TicketCalculationRequest $request)
{
    $fromStop = Stop::findOrFail($request->from_stop_id);
    $toStop = Stop::findOrFail($request->to_stop_id);
    $line = Line::findOrFail($request->line_id);
    $ticket = Ticket::findOrFail($request->ticket_id);

    $fromSequence = LineStop::where('line_id', $line->line_id)
        ->where('stop_id', $fromStop->stop_id)
        ->first()->sequence;

    $toSequence = LineStop::where('line_id', $line->line_id)
        ->where('stop_id', $toStop->stop_id)
        ->first()->sequence;

    if ($fromSequence >= $toSequence) {
        return back()->withErrors([
            'from_stop_id' => 'Wybrany kierunek podróży jest
nieprawidłowy.'
        ])->withInput();
    }

    $distanceInput = $request->input('route_distance');
    $distance = null;
    $ticketPriceService = app(TicketPriceService::class);

    if ($distanceInput !== null && $distanceInput !== '' &&
is_numeric(str_replace(',', '.', $distanceInput))) {
        $parsedDistance = floatval(str_replace(',', '.', $distanceInput));
        if ($parsedDistance > 0 || ($parsedDistance === 0.0 && $fromStop-
>stop_id === $toStop->stop_id)) {
            $distance = $parsedDistance;
        }
    }

    if ($distance === null) {
        $distance = $ticketPriceService->calculateRouteDistance(
            $fromStop->stop_id,
            $toStop->stop_id,
            $line->line_id
        );
    }
}
```

```

    }

    $avgTimePerSegment = $line->avg_time_per_segment_minutes ?? 3;
    $durationMinutes = ($toSequence - $fromSequence) * $avgTimePerSegment;
    if ($fromStop->stop_id === $toStop->stop_id) {
        $durationMinutes = 0;
    }
    $durationMinutes = max(0, $durationMinutes);

    $calculatedPrice = $ticketPriceService->calculatePrice(
        $durationMinutes,
        $distance,
        $ticket->ticket_id
    );

    $finalPrice = ceil($calculatedPrice);

    $routeData = [
        'from_stop' => [
            'id' => $fromStop->stop_id,
            'name' => $fromStop->stop_name,
            'sequence' => $fromSequence,
            'location_lat' => $fromStop->location_lat,
            'location_lon' => $fromStop->location_lon
        ],
        'to_stop' => [
            'id' => $toStop->stop_id,
            'name' => $toStop->stop_name,
            'sequence' => $toSequence,
            'location_lat' => $toStop->location_lat,
            'location_lon' => $toStop->location_lon
        ],
        'line' => [
            'id' => $line->line_id,
            'number' => $line->line_number,
            'name' => $line->line_name
        ],
        'distance' => round($distance, 2),
        'duration' => $durationMinutes
    ];

    session([
        'route_data' => $routeData,
        'ticket_id' => $ticket->ticket_id,
        'calculated_price' => $finalPrice
    ]);

    return redirect()->route('tickets.checkout-route');
}

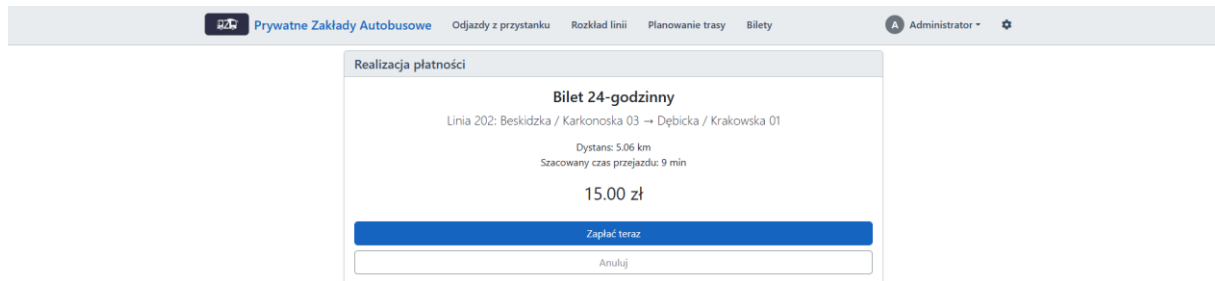
```

Krok 4: Finalizacja zakupu i inicjalizacja płatności (Backend)

Na tym etapie system przygotowuje wszystko do dokonania płatności.

- **Kontroler:** app/Http/Controllers/TicketController.php, metoda checkoutRoute()
- **Serwis:** app/Http/Services/TicketPurchaseService.php
- **Logika:**
 1. Dane o trasie i obliczona cena są pobierane z sesji.

2. W systemie tworzone są rekordy w tabelach transactions i purchased_tickets z początkowym statusem „oczekująca”.
3. Tworzona jest sesja płatności **Stripe** z ostateczną, obliczoną ceną.
4. Użytkownik jest przekierowywany do widoku podsumowania, gdzie widzi ostateczną kwotę i przycisk do przejścia na stronę płatności Stripe.

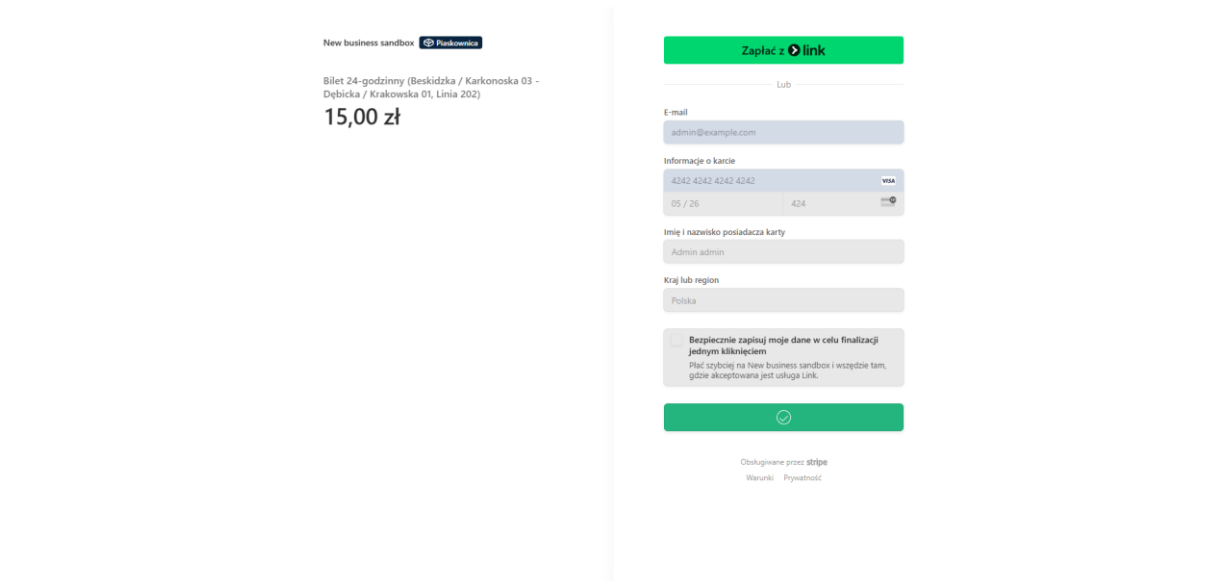


Rysunek 35. Podsumowanie zamówienia

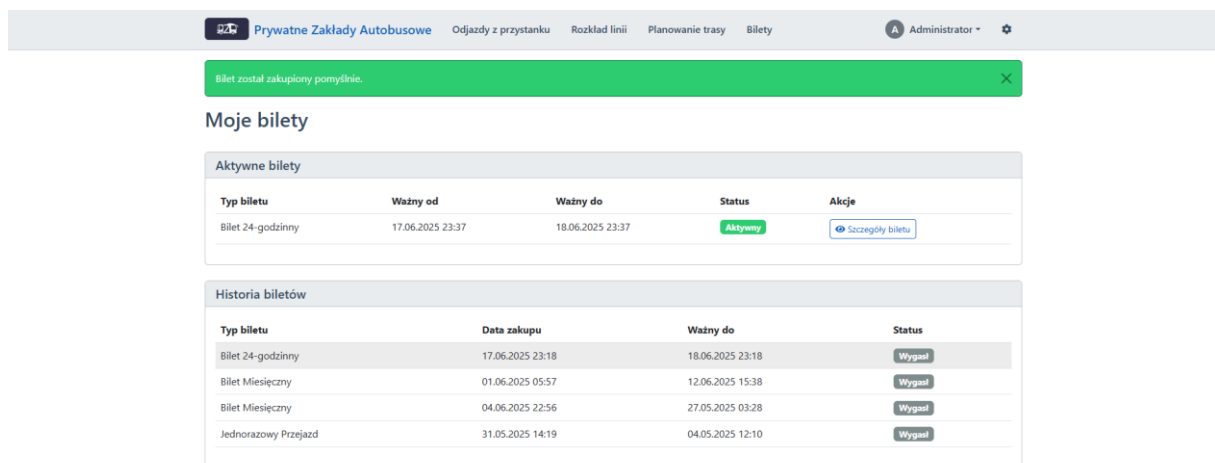
Krok 5: Potwierdzenie płatności i aktywacja biletu (Backend)

Ostatni krok to obsługa odpowiedzi z systemu płatności.

- **Kontroler:** app/Http/Controllers/StripeWebhookController.php
- **Logika:**
 1. Po udanej płatności, serwery Stripe wysyłają do aplikacji powiadomienie (webhook) o zdarzeniu checkout.session.completed.
 2. Kontroler webhooka odbiera to powiadomienie, weryfikuje jego autentyczność i znajduje powiązaną transakcję w bazie danych.
 3. Status transakcji jest aktualizowany na „zakończona”, a status zakupionego biletu na „aktywny”.
 4. Użytkownik, który w tym czasie został przekierowany ze Stripe na stronę sukcesu, widzi potwierdzenie pomyślnego zakupu. Bilet jest gotowy do użycia.



Rysunek 36. Udana transakcja stripe



Rysunek 37. Pomyślny zakup biletu

7. Podsumowanie

Przedstawiona dokumentacja opisuje projekt i realizację aplikacji webowej „Informator o rozkładzie autobusów”. Głównym celem projektu było stworzenie zintegrowanego systemu informacyjnego, który z jednej strony ułatwia pasażerom codzienne korzystanie z komunikacji miejskiej, a z drugiej dostarcza administratorom narzędzi do efektywnego zarządzania siatką połączeń, rozkładami jazdy oraz ofertą biletową.

Aplikacja została zbudowana w oparciu o nowoczesne technologie, z wykorzystaniem frameworka **Laravel 12** i języka **PHP 8.2** po stronie serwera oraz systemu szablonów **Blade** i **JavaScript** po stronie klienta. Baza danych **MySQL** zapewnia spójność i wydajność przechowywanych danych.

W ramach projektu zaimplementowano szereg kluczowych funkcjonalności, w tym:

- Dla **pasażerów**: wyszukiwarkę połączeń, przeglądanie rozkładów, planowanie trasy oraz zaawansowany system zakupu biletów online z dynamiczną kalkulacją ceny i integracją z systemem płatności **Stripe**.
- Dla **administratorów**: pełen zestaw operacji **CRUD** do zarządzania liniami, przystankami, kursami, typami biletów i użytkownikami, a także panel statystyczny wizualizujący kluczowe dane systemowe.

Szczególny nacisk położono na jakość kodu, bezpieczeństwo (m.in. przez rozbudowany system walidacji) oraz użyteczność, implementując zasady **Responsive Web Design** i dodatkowe funkcje dostępności, takie jak zmiana kontrastu i rozmiaru czcionki.

Całość tworzy profesjonalny i wyczerpujący dokument, który kompleksowo opisuje wszystkie aspekty zrealizowanego projektu, od założeń, przez technologie i architekturę, aż po szczegółowe działanie poszczególnych modułów.