|  |  |  |
| --- | --- | --- |
|  | Dokumentacja projektu | **AI1** |
| **Autor** | Jakub Wojtowicz, 122980 | **Ocena** |
| **Kierunek, rok** | Informatyka, II rok, st. Stacjonarne (3,5-l) |  |
| **Temat projektu** | System do zamawiania usługi transportu mebli |  |

1. **Wstęp**

System do zamawiania usługi transportu mebli to aplikacja internetowa, która ma na celu ułatwienie procesu zamawiania i organizacji transportu mebli. Aplikacja skierowana jest do małych przedsiębiorstw lub osób które potrzebują przetransportować swoje meble. Działa jako platforma, na której zarejestrowani i zalogowani użytkownicy mogą zamawiać usługę transportu mebli z dodatkową usługą demontażu i montażu mebli oraz zarządzać zamówieniami, jak i swoim kontem użytkownika. Administratorzy mają pełną kontrolę nad systemem i mogą zarządzać wszystkimi zamówieniami (przypisywać samochód, oraz kierowcę, zmieniać status itp.), użytkownikami, usługami, samochodami i kierowcami.

1. **Narzędzia i technologie**

**NARZĘDZIA**

**•** Visual Studio Code (v.1.79 )

• XAMPP (v.3.3.0)

**SOFTWARE**

• PHP (v.8.2.4)

• Composer (v.2.5.7)

• Laravel (v.10.13.2)

• nodeJS (v.18.16.0)

**DODATKOWE BIBLIOTEKI**

• Axios (v.1.1.2)

• AlpineJS(v.3.4.2)

• Autoprefixer (v.10.4.2)

• Botstrap(v.5.3.0)

**Opis technologii**

* **Visual Studio Code**

Visual Studio Code (VS Code) to narzędzie programistyczne stworzone przez firmę Microsoft, które można używać na różnych platformach. Jest to środowisko programistyczne o otwartym kodzie źródłowym, które zyskało popularność ze względu na swoją prostotę, elastyczność i bogate funkcje.

VS Code oferuje wsparcie dla wielu języków programowania i frameworków, takich jak JavaScript, php, Java, HTML, CSS i wiele innych. Dzięki temu jest idealnym narzędziem do różnych typów projektów i środowisk programistycznych.

Jedną z kluczowych cech VS Code jest jego rozszerzalność. Można dostosować to środowisko programistyczne poprzez instalowanie różnych dodatków, które dodają nowe funkcje i usprawnienia. Społeczność programistów tworzy wiele takich rozszerzeń, które można znaleźć w galerii rozszerzeń Visual Studio Code.

Link do pobrania: [LINK](https://code.visualstudio.com/)

* **XAMPP**

XAMPP to łatwe i wygodne narzędzie, które pozwala na stworzenie lokalnego środowiska serwerowego na twoim komputerze. Dzięki wbudowanym komponentom, takim jak Apache, MySQL, PHP i Perl, XAMPP umożliwia tworzenie i testowanie aplikacji webowych bez potrzeby osobnej instalacji i konfiguracji każdego z tych elementów. Jest to idealne rozwiązanie dla programistów, którzy chcą pracować lokalnie nad swoimi aplikacjami przed ich wdrożeniem na serwerze produkcyjnym.

Link do pobrania: [LINK](https://www.apachefriends.org/download.html)

* **PHP**

PHP jest wszechstronnym językiem programowania, który umożliwia tworzenie dynamicznych stron internetowych i aplikacji webowych. Jest popularny ze względu na swoją prostotę nauki, bogate funkcje i wsparcie ze strony społeczności programistycznej.

Link do pobrania: [LINK](https://www.php.net/)

* **Composer**

Composer jest narzędziem służącym do zarządzania zależnościami w projektach PHP. Umożliwia łatwe instalowanie, aktualizowanie i usuwanie bibliotek, a także rozwiązywanie problemów z zależnościami. W społeczności PHP jest szeroko stosowany do skutecznego zarządzania zależnościami i przyspieszenia procesu tworzenia aplikacji PHP.

Link do pobrania: [LINK](https://getcomposer.org/)

* **Laravel**

Laravel to popularny i zaawansowany framework PHP, który ułatwia tworzenie skalowalnych i eleganckich aplikacji webowych. Zaprojektowany z myślą o programistach, Laravel oferuje wiele gotowych rozwiązań i narzędzi, które przyspieszają proces tworzenia aplikacji. Framework ten cechuje się czytelną składnią i łatwością nauki. Wykorzystuje konwencję ponad konfigurację, co oznacza, że wiele podstawowych ustawień jest już skonfigurowanych, co pozwala programistom skupić się na tworzeniu funkcjonalności. Laravel oferuje wiele funkcji, takich jak system routingu, zarządzanie sesjami, autoryzację, obsługę baz danych, buforowanie, konsolę Artisan do wykonywania poleceń wiersza poleceń, obsługę plików, testowanie jednostkowe i wiele innych. Można go zintegrować z popularnymi narzędziami front-endowymi, takimi jak Vue.js czy React, co ułatwia tworzenie interaktywnych interfejsów użytkownika. Aby zainstalować Laravel, należy upewnić się, że masz zainstalowane odpowiednie wersje Composera i PHP, a następnie użyć polecenia „composer create-project --prefer-dist laravel/laravel nazwa\_projektu”.

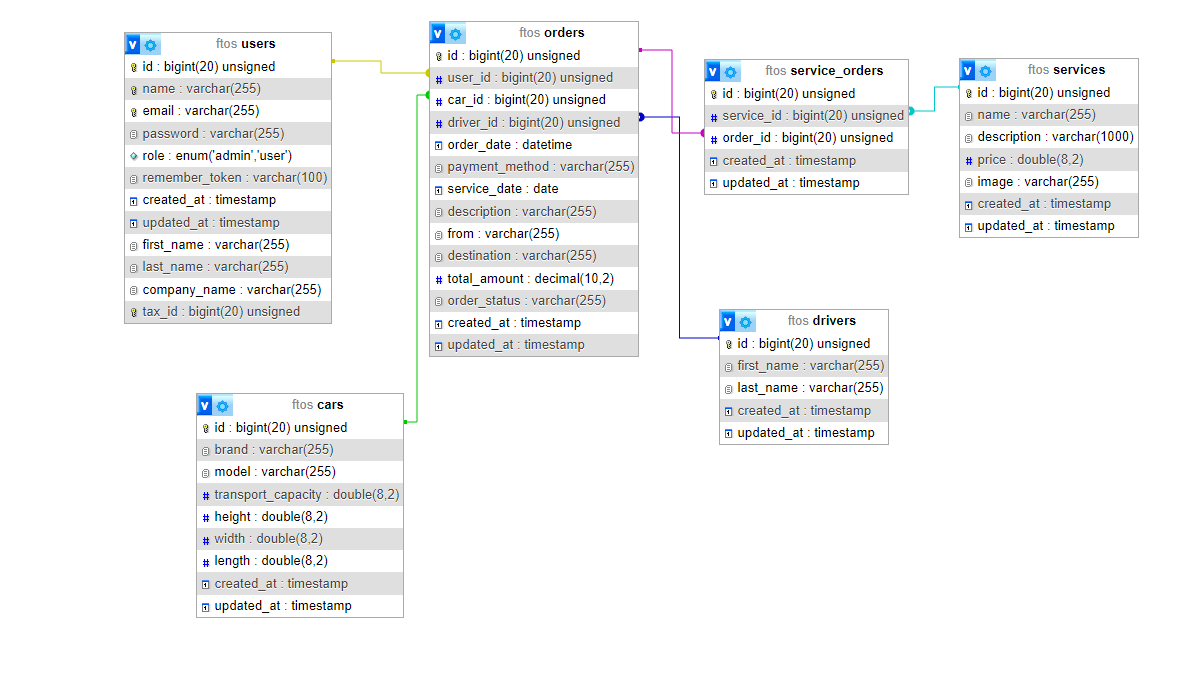
* **NodeJS**

Node.js to środowisko uruchomieniowe, które umożliwia uruchamianie kodu JavaScript poza przeglądarką. Dzięki swojej asynchronicznej i zdarzeniowej naturze, Node.js jest idealnym narzędziem do tworzenia skalowalnych aplikacji sieciowych. Posiada duże wsparcie społecznościowe i dostęp do wielu modułów, co daje programistom szerokie możliwości i efektywny sposób tworzenia zaawansowanych aplikacji serwerowych i narzędzi front-endowych.

Link do pobrania: [LINK](https://nodejs.org/en)

1. **Baza danych**

**ERD**

****

**Opis**

Ten model bazy danych zawiera strukturę tabel niezbędną do obsługi systemu zamówień dla sklepu internetowego. Składa się z kilku kluczowych tabel, takich jak "users" (użytkownicy), "cars" (samochody), "orders" (zamówienia), "drivers" (kierowcy), "services" (usługi) oraz tabela transferowa "service\_orders" (usługi - zamówienia).

Tabela "users" przechowuje informacje o użytkownikach, takie jak ich identyfikator, imię, nazwisko, adres e-mail, hasło, rola, data utworzenia i data aktualizacji, a także dodatkowe informacje, takie jak (opcjonalnie, dla firm) nazwa firmy i identyfikator podatkowy.

Tabela "cars" zawiera dane na temat samochodów firmy, takie jak ich identyfikator, marka, model, pojemność transportowa, wysokość, szerokość, długość, data utworzenia i data aktualizacji.

Tabela "orders" przechowuje informacje o zamówieniach, takie jak ich identyfikator, identyfikator użytkownika składającego zamówienie, identyfikator samochodu, identyfikator kierowcy obsługującego zamówienie, data zamówienia, metoda płatności, data wykonania usługi, opis, punkt odbioru ładunku, punkt docelowy, całkowita kwota zamówienia, status zamówienia, data utworzenia i data aktualizacji.

Tabela "drivers" zawiera dane na temat kierowców, takie jak ich identyfikator, imię, nazwisko, data utworzenia i data aktualizacji.

Tabela "services" przechowuje informacje o usługach, takie jak ich identyfikator, nazwa, opis, cena, obrazek, data utworzenia i data aktualizacji.

Tabela transferowa "service\_orders" łączy usługi i zamówienia, przechowując ich identyfikatory oraz datę utworzenia i aktualizacji.

Aby połączyć bazę danych z API, należy skonfigurować plik ".env" znajdujący się w głównym katalogu projektu. W tym pliku znajdują się ustawienia połączenia z bazą danych, takie jak typ bazy danych, adres serwera, port, nazwa bazy danych, nazwa użytkownika i hasło. W przypadku tego projektu, baza danych jest ustawiona na lokalny serwer bazodanowy XAMPP, a nazwa bazy danych to "ftos". Oto fragment pliku:

# typ bazy danych

DB\_CONNECTION=mysql

# adres serwera

DB\_HOST=127.0.0.1

# port z ktorego korzysta serwer

DB\_PORT=3306

# nazwa bazy z ktorej laravel korzysta

DB\_DATABASE=ftos

# nazwa uzytkownika

DB\_USERNAME=root

# haslo

DB\_PASSWORD=

Aktualnie projekt jest skonfigurowany do połączenia z lokalnym serwerem bazodanowym XAMPP. Aby poprawnie połączyć naszą stronę z bazą danych, musimy najpierw utworzyć odpowiednią bazę danych o nazwie "ftos". Jest to jedyna czynność, którą musimy wykonać przed zestawieniem połączenia. Po utworzeniu bazy danych, Strona wówczas będzie w stanie skutecznie komunikować się z nią i wykonywać operacje na tabelach.

**Modele i relacje**

Dzięki modelom relacji w bazie danych możemy łatwo tworzyć, modyfikować i usuwać rekordy, które są powiązane z danym rekordem. Oznacza to, że możemy bezproblemowo manipulować danymi w różnych tabelach, które są powiązane ze sobą. Dzięki temu możemy wygodnie zarządzać powiązanymi danymi i wprowadzać zmiany w systemie bez konieczności ręcznego manipulowania wieloma tabelami jednocześnie.

    // relacja tabeli orders z tabelą users

    public function user(): BelongsTo

    {

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

    }

    // relacja tabeli orders z tabelą services

    public function services(): BelongsToMany

    {

        return $this->belongsToMany(Service::class, 'service\_orders');

    }

Powyżej widzimy przykład stworzonych relacji z pliku Order.php. Widzimy tu relacje między tabelami orders, a users, oraz relację między tabelami orders, a services za pośrednictwem tabeli transferowej service\_orders.

**Migracje**

    public function up(): void

    {

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

            $table->id();

            $table->unsignedBigInteger("user\_id");

            $table->foreign("user\_id")->references("id")->on("users");

            $table->unsignedBigInteger("car\_id")->nullable();

            $table->foreign("car\_id")->references("id")->on("cars");

            $table->unsignedBigInteger("driver\_id")->nullable();

            $table->foreign("driver\_id")->references("id")->on("drivers");

            $table->datetime("order\_date");

            $table->string("payment\_method");

            $table->date("service\_date");

            $table->string("description");

            $table->string("from");

            $table->string("destination");

            $table->decimal("total\_amount", 10, 2)->default(0.00);

            $table->string("order\_status");

            $table->timestamps();

        });

    }

    /\*\*

     \* Reverse the migrations.

     \*/

    public function down(): void

    {

        Schema::table('orders', function (Blueprint $table) {

            $table->dropForeign(["user\_id"]);

            $table->dropForeign(["car\_id"]);

            $table->dropForeign(["driver\_id"]);

        });

        Schema::dropIfExists('orders');

    }

Powyżej widzimy przykład tworzenia migracji dla tabeli orders. W przedstawionej migracji dla tabeli "orders" tworzone są różne kolumny i ich typy danych. Oto opis tych kolumn:

Kolumna "id" jest automatycznie generowanym identyfikatorem dla każdego rekordu w tabeli. Jest to kolumna typu całkowitoliczbowego.

Kolumna "user\_id" jest kluczem obcym (foreign key) typu całkowitoliczbowego, która odnosi się do kolumny "id" w tabeli "users". W ten sposób nawiązane jest powiązanie między tabelą "orders" a "users".

Kolumna "car\_id" jest kluczem obcym typu całkowitoliczbowego, który jest opcjonalny (nullable). Jest powiązana z kolumną "id" w tabeli "cars".

Kolumna "driver\_id" jest kluczem obcym typu całkowitoliczbowego, który również jest opcjonalny. Jest powiązana z kolumną "id" w tabeli "drivers".

Kolumna "order\_date" jest kolumną daty i czasu, która przechowuje datę złożenia zamówienia.

Kolumna "payment\_method" jest kolumną typu tekstowego, w której przechowywana jest metoda płatności dla zamówienia.

Kolumna "service\_date" jest kolumną daty, która przechowuje datę planowanego wykonania usługi związanej z zamówieniem.

Kolumna "description" jest kolumną typu tekstowego, w której można umieścić opis zamówienia.

Kolumna "from" jest kolumną typu tekstowego, w której przechowywane jest miejsce początkowe zamówienia.

Kolumna "destination" jest kolumną typu tekstowego, która przechowuje miejsce docelowe zamówienia.

Kolumna "total\_amount" jest kolumną typu dziesiętnego, która przechowuje łączną kwotę zamówienia.

Kolumna "order\_status" jest kolumną typu tekstowego, która określa status zamówienia.

Kolumna "timestamps" to automatycznie generowane kolumny "created\_at" i "updated\_at", które przechowują daty utworzenia i ostatniej modyfikacji rekordu.

W sekcji "down" migracji usuwane są klucze obce dla kolumn "user\_id", "car\_id" i "driver\_id", a następnie cała tabela "orders" jest usuwana wraz z migracją w metodzie "dropIfExists".

**Seedery**

Seeder w Laravelu służy do automatycznego wypełniania bazy danych testowymi danymi. Pozwala na łatwe generowanie danych początkowych, takich jak rekordy w tabelach, co jest szczególnie przydatne podczas tworzenia środowiska developerskiego lub testowego.

Oto przykład seedera dla tabeli orders:

    public function run()

    {

        Schema::disableForeignKeyConstraints();

        Order::truncate();

        Order::insert([

            [

                'user\_id' => 1,

                'car\_id' => 1,

                'driver\_id' => 1,

                'order\_date' => \Carbon\Carbon::parse('2023-06-01 10:30:00'),

                'payment\_method' => 'Credit Card',

                'total\_amount' => '100.00',

                'order\_status' => 'Done',

                'service\_date' => \Carbon\Carbon::parse('2023-06-05'),

                'description' => 'Table 100x100x50',

                'from' => 'Rzeszow, Zalesie 123',

                'destination' => 'Rzeszow, Targowa 5',

                'created\_at' => Carbon::now(),

                'updated\_at' => Carbon::now(),

            ],

            [

                'user\_id' => 2,

                'car\_id' => 2,

                'driver\_id' => 3,

                'order\_date' => \Carbon\Carbon::parse('2023-06-01 10:30:00'),

                'payment\_method' => 'Cash',

                'total\_amount' => '150.00',

                'order\_status' => 'Done',

                'service\_date' => \Carbon\Carbon::parse('2023-06-03'),

                'description' => 'Sofa 300x100x100',

                'from' => 'Rzeszow, Podkarpacka 45',

                'destination' => 'Sanok, Kolejowa 10',

                'created\_at' => Carbon::now(),

                'updated\_at' => Carbon::now(),

            ],

        ]);

        Schema::enableForeignKeyConstraints();

    }

W tym seederze, wyłączane są ograniczenia kluczy obcych w bazie danych. Następnie, tabela "orders" jest czyszczona (wszystkie rekordy są usuwane). Następnie, dodawane są dwa nowe rekordy do tabeli "orders" zawierające różne dane, takie jak identyfikatory użytkownika, samochodu i kierowcy, daty zamówienia, metody płatności, kwoty, status zamówienia, daty usługi, opis, miejsce początkowe i docelowe. Na koniec, włączane są ponownie ograniczenia kluczy obcych w bazie danych. Funkcja Carbon używana w seederze służy do manipulacji datami i czasem w Laravelu, umożliwiając łatwe tworzenie i formatowanie dat oraz wykonywanie operacji takich jak dodawanie, odejmowanie lub porównywanie czasu.

1. **Uruchomienie aplikacji**

Aby uruchomić aplikację, należy upewnić się, że posiadamy zainstalowane wszystkie wymagane aplikacje w odpowiednich wersjach, takie jak PHP, Composer, Node.js i XAMPP. Jeśli którekolwiek z tych narzędzi nie jest zainstalowane, należy je zainstalować.

Proces uruchamiania aplikacji polega na wykonaniu następujących kroków:

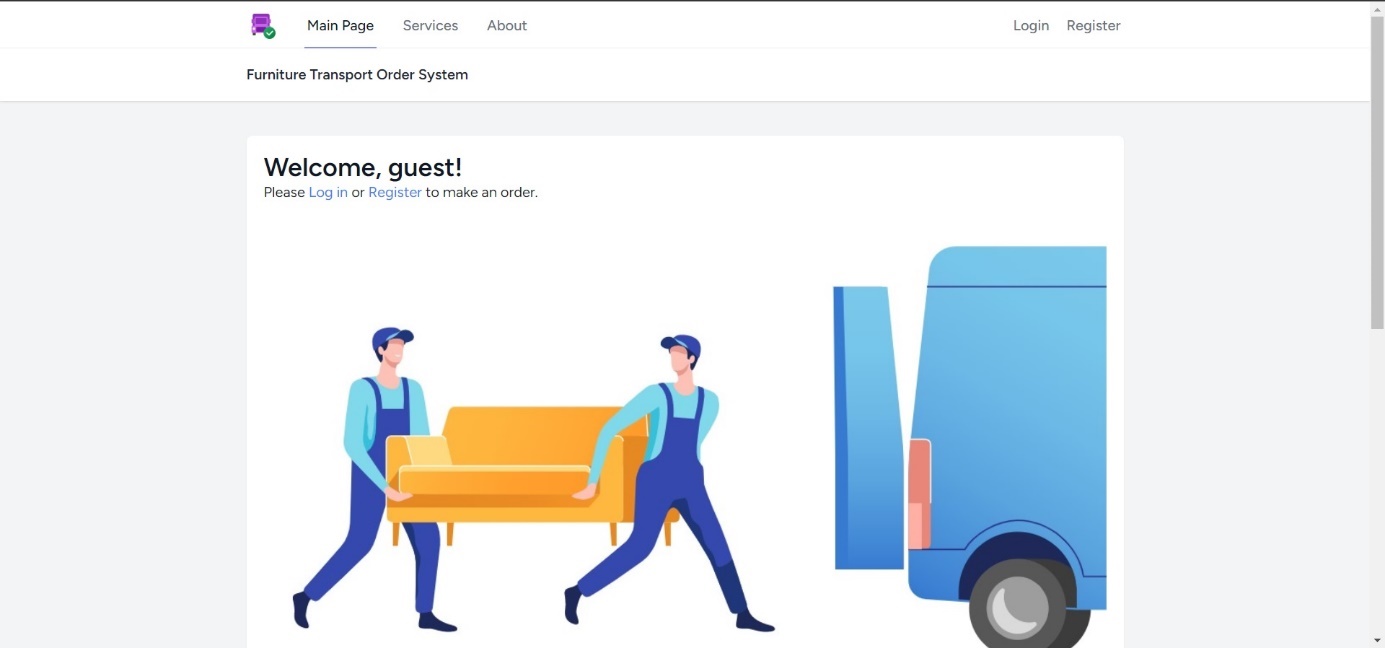
1. Pobierz repozytorium Laravela z GitHub za pomocą narzędzi do kontroli wersji, takich jak Git.
2. Przejdź do katalogu, w którym znajduje się pobrane repozytorium i uruchom terminal.
3. Wykonaj polecenie "composer install" w terminalu, aby zainstalować wszystkie zależności Laravela.
4. Wykonaj polecenie "npm install", oraz "npm run dev", aby zainstalować potrzebne pakiety JavaScript, oraz skompilować zasoby, które następnie będą gotowe do użycia w aplikacji.
5. Uruchom XAMPP i włącz serwery Apache i MySQL. Następnie otwórz panel administratora MySQL, klikając przycisk "admin".
6. W panelu administratora MySQL utwórz nową bazę danych o nazwie "ftos".
7. Skopiuj plik ".env.example" w katalogu Laravela i zapisz go jako ".env". Otwórz ten plik i skonfiguruj ustawienia połączenia z bazą danych zgodnie z ustawieniami XAMPP.
8. Powróć do terminala i wpisz polecenie "php artisan migrate:fresh". To spowoduje wykonanie migracji bazy danych.
9. Teraz wykonaj polecenie "php artisan db:seed", aby zapełnić bazę przykładowymi danymi
10. Po zakończeniu tych kroków aplikacja Laravel jest gotowa do uruchomienia.

W rezultacie aplikacja powinna być zainstalowana i gotowa do działania.

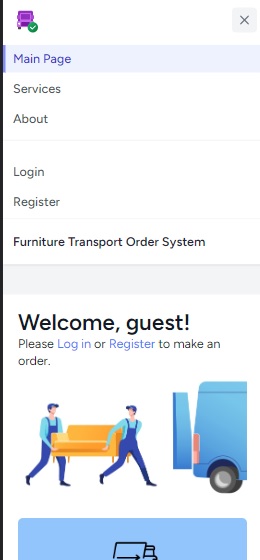
1. **Funkcjonalności aplikacji**

**Gość:**

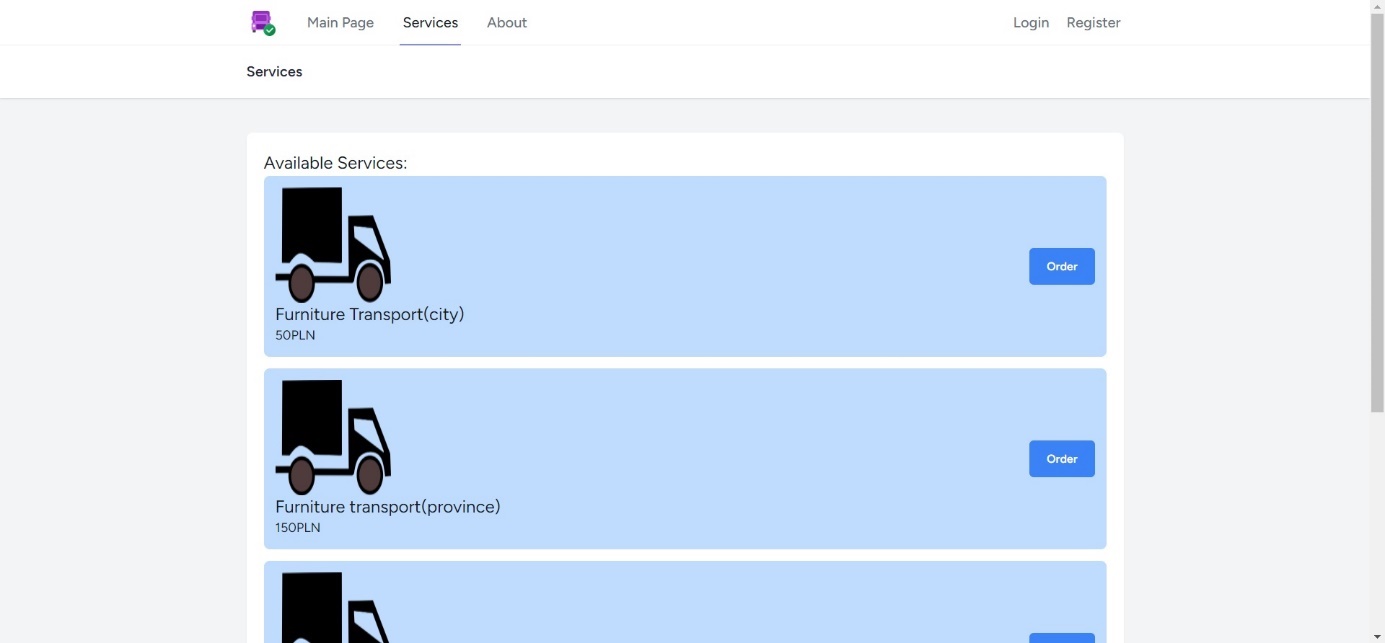
* Może przeglądać ogólnie dostępne zasoby, np. widoki: Main Page, Services, About

****

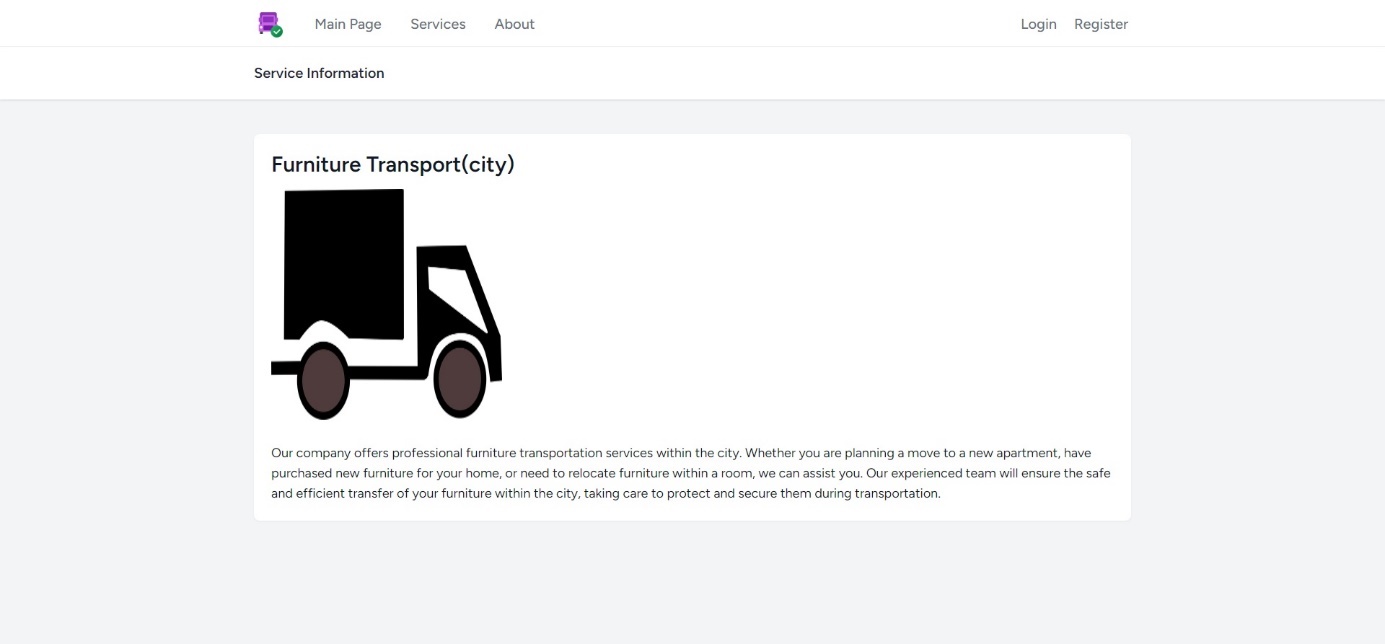
Strona główna aplikacji



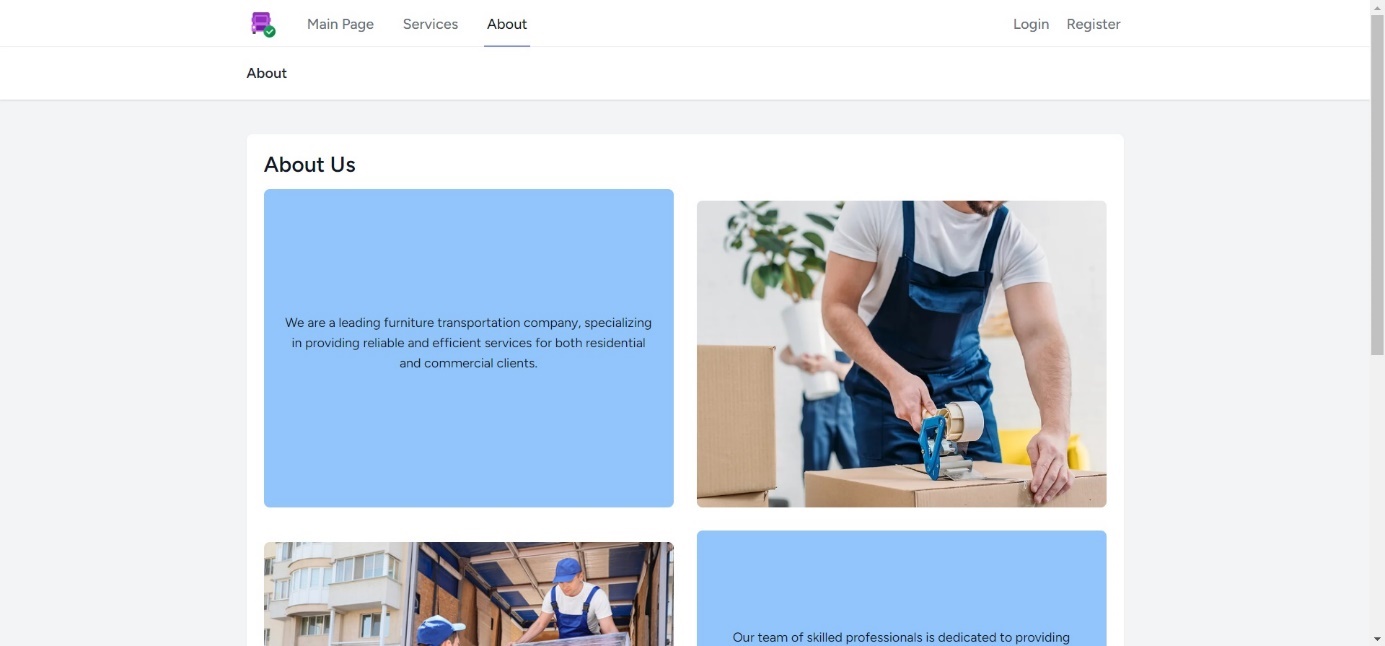
Strona główna z wysuwanym paskiem menu w formie mobilnej.



Zakładka z usługami dostępnymi na stronie



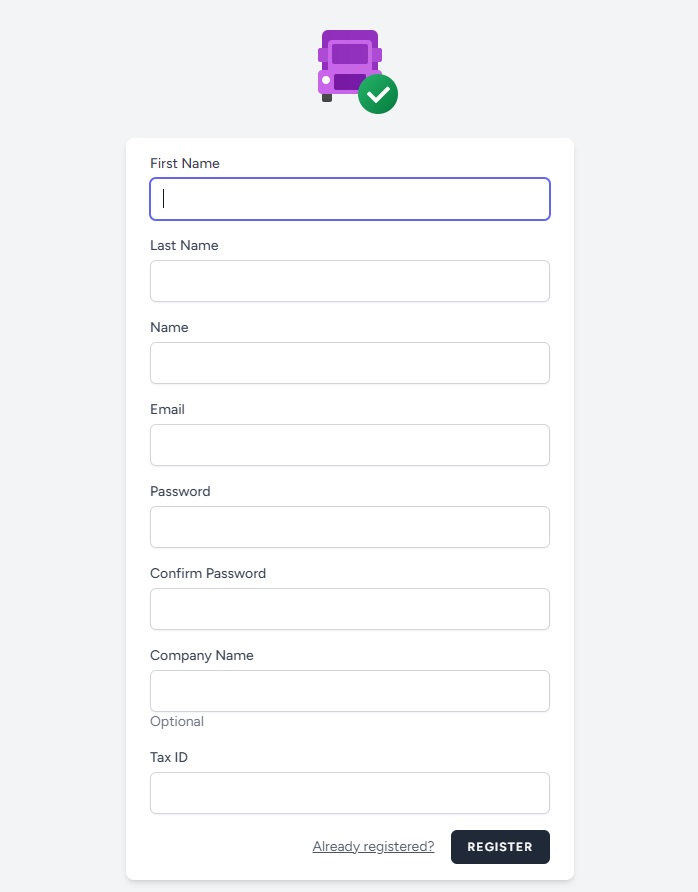
Po kliknięciu w jakąś usługę zostajemy przeniesieni na widok z większą ilością informacji o usłudze



Widok z informacjami „O nas”

**Rejestracja użytkownika:**

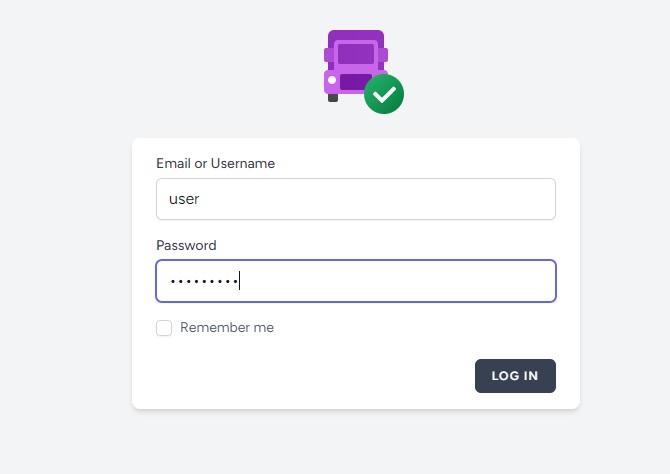
* + Aplikacja umożliwia nowym użytkownikom rejestrację, aby uzyskać dostęp do funkcjonalności.
  + Podczas rejestracji będą zbierane następujące dane:
    - Imię i nazwisko użytkownika
    - Nazwa użytkownika
    - Adres e-mail
    - Hasło
    - Nazwa firmy (opcjonalnie)
    - Numer identyfikacji podatkowej (NIP) i nazwa firmy (opcjonalnie, jeśli użytkownik jest przedsiębiorstwem)
  + Wymagania dotyczące hasła:
    - Hasło musi zawierać co najmniej 8 znaków.
    - Użytkownik musi potwierdzić hasło podczas rejestracji.



Widok rejestracji użytkownika

**Logowanie użytkownika:**

* + Zarejestrowani użytkownicy mogą zalogować się do aplikacji, używając swojego adresu e-mail (lub loginu) oraz hasła.



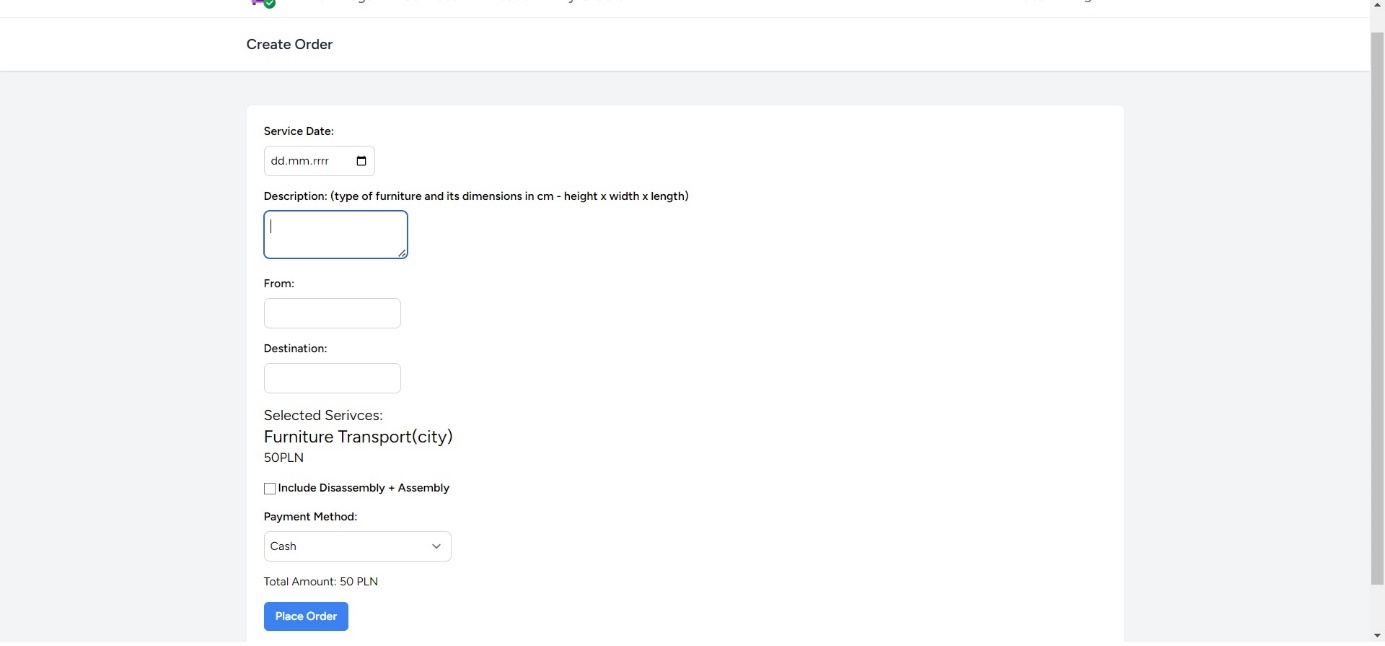
Widok logowania do aplikacji przez użytkownika „user”.

**Przeglądanie dostępnych usług:**

* + Użytkownicy mają możliwość przeglądania dostępnych usług transportu mebli (transport mebli w mieście, transport w województwie, transport krajowy, dodatkowo montaż i demontaż).

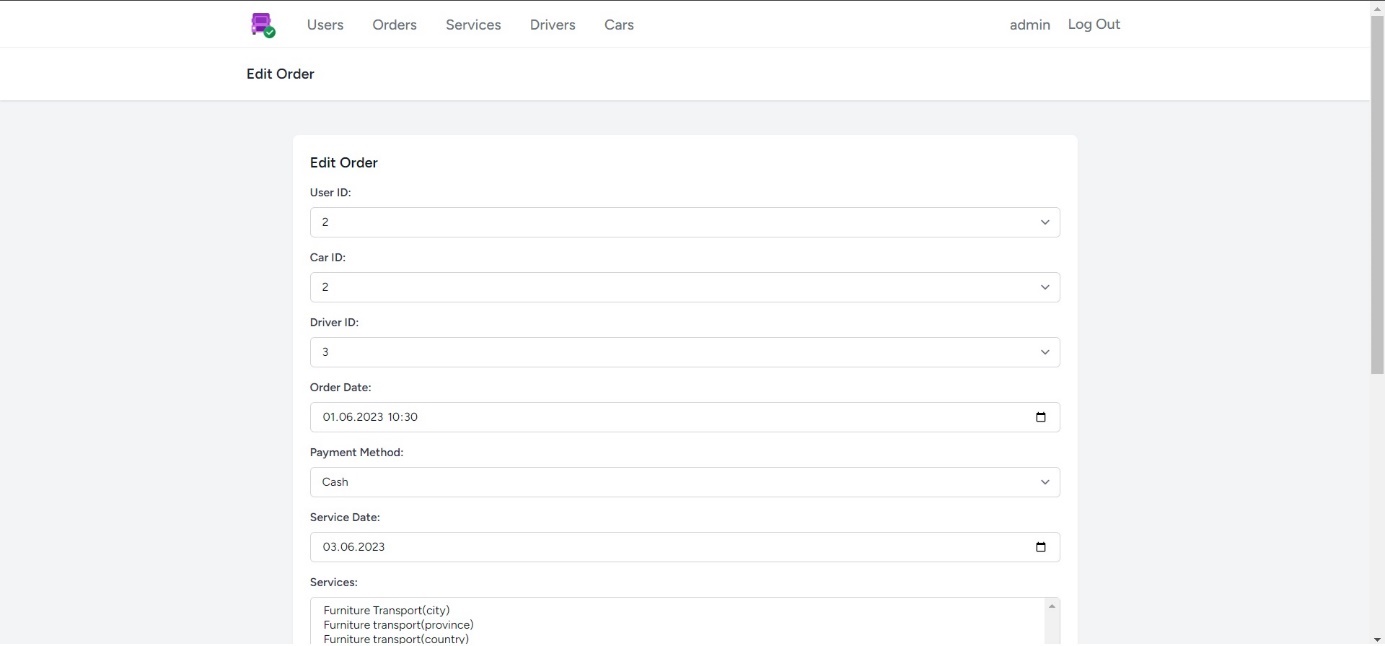
**Zamawianie usługi poprzez formularz:**

* + Zalogowani użytkownicy mogą zamówić wybraną usługę transportu mebli poprzez kliknięcie przycisku ‘Order’ w widoku z dostępnymi usługamioraz wypełnienie formularza zamówienia.
  + Formularz zawiera pola, takie jak:
    - Data usługi
* Opis mebli
* Adres odbioru mebli
* Adres dostawy mebli
* Wybrana uprzednio usługa
* Możliwość dodania do zamówienia usługi demontażu i montażu mebli
* Metoda płatności
* Pełna kwota zamówienia



**Potwierdzenie zamówienia:**

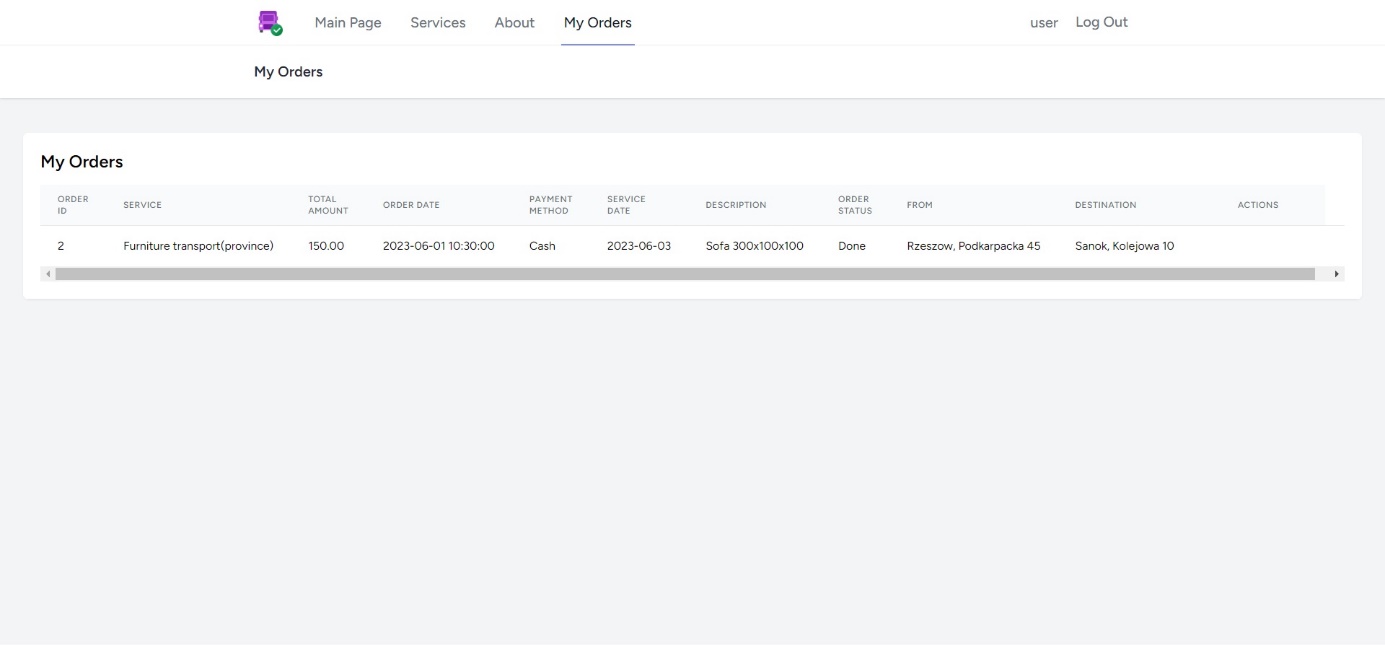
* Zamówienie jest przetwarzane przez administratora, który potwierdza dostępność usługi i ustala szczegóły transportu.



Widok edycji zamówienia przez konto administratora

**Zarządzanie zamówieniami:**

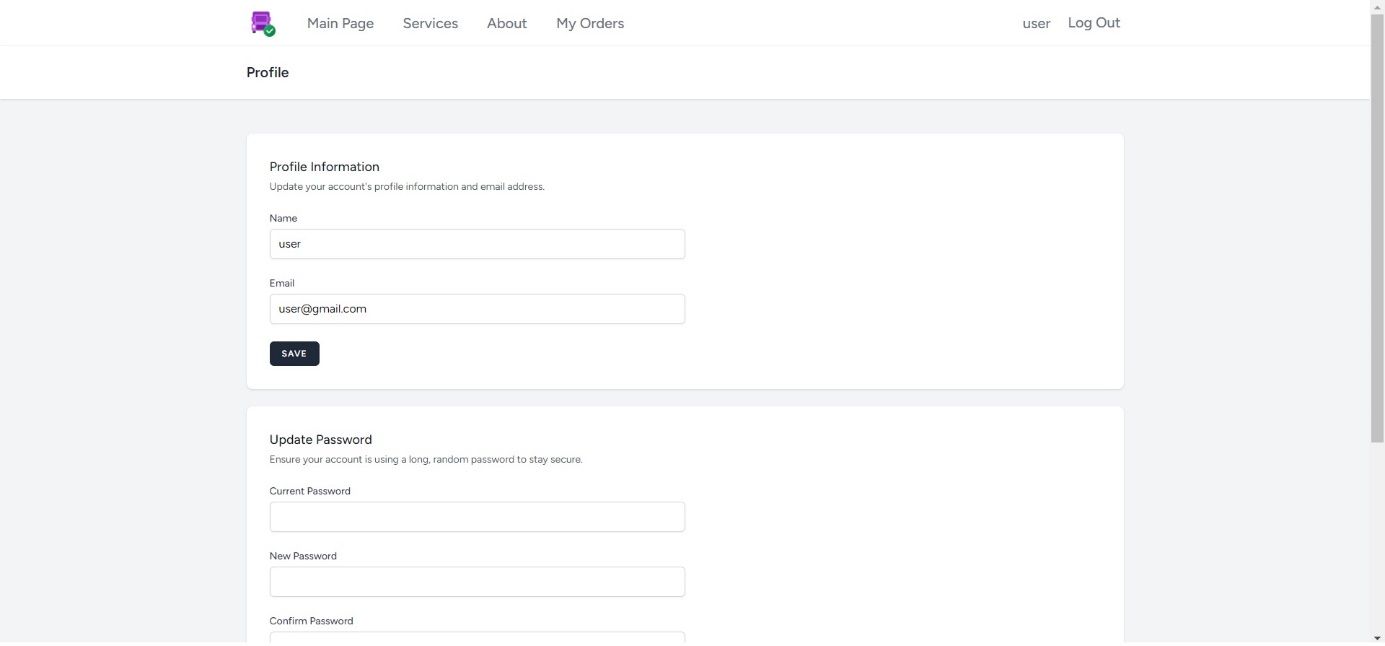
* + Zarejestrowani użytkownicy mają dostęp do panelu zarządzania swoimi zamówieniami.
  + Użytkownicy mogą sprawdzić status zamówienia, historia zamówień, informacje o dostawie itp.
  + Użytkownicy mogą anulować zamówienie jeszcze nie będące w trakcie realizacji lub anulować je.



Widok zamówień użytkownika ‘user’

**Zarządzanie kontem użytkownika:**

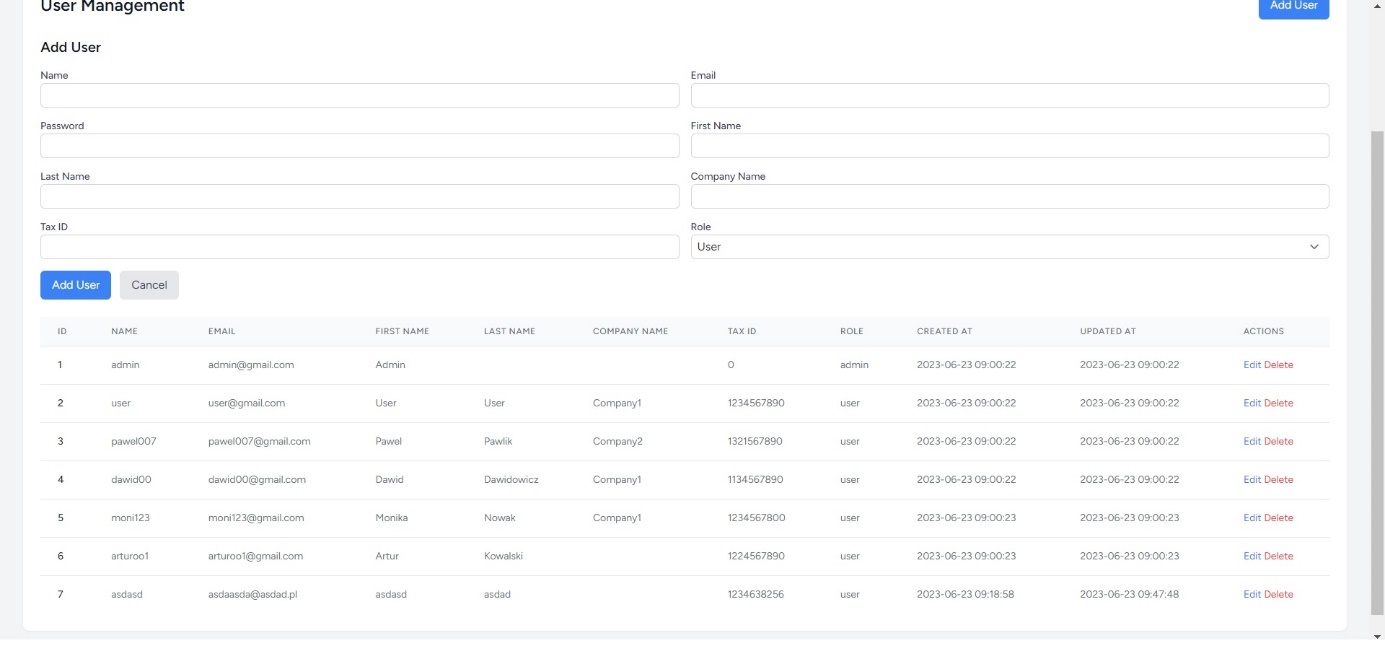
* Użytkownicy mogą zarządzać swoim kontem, edytując swoje dane osobowe.
* Istnieje możliwość zmiany hasła przez użytkownika, zmiana nazwy, adresu email.
* Możliwość usunięcia konta.



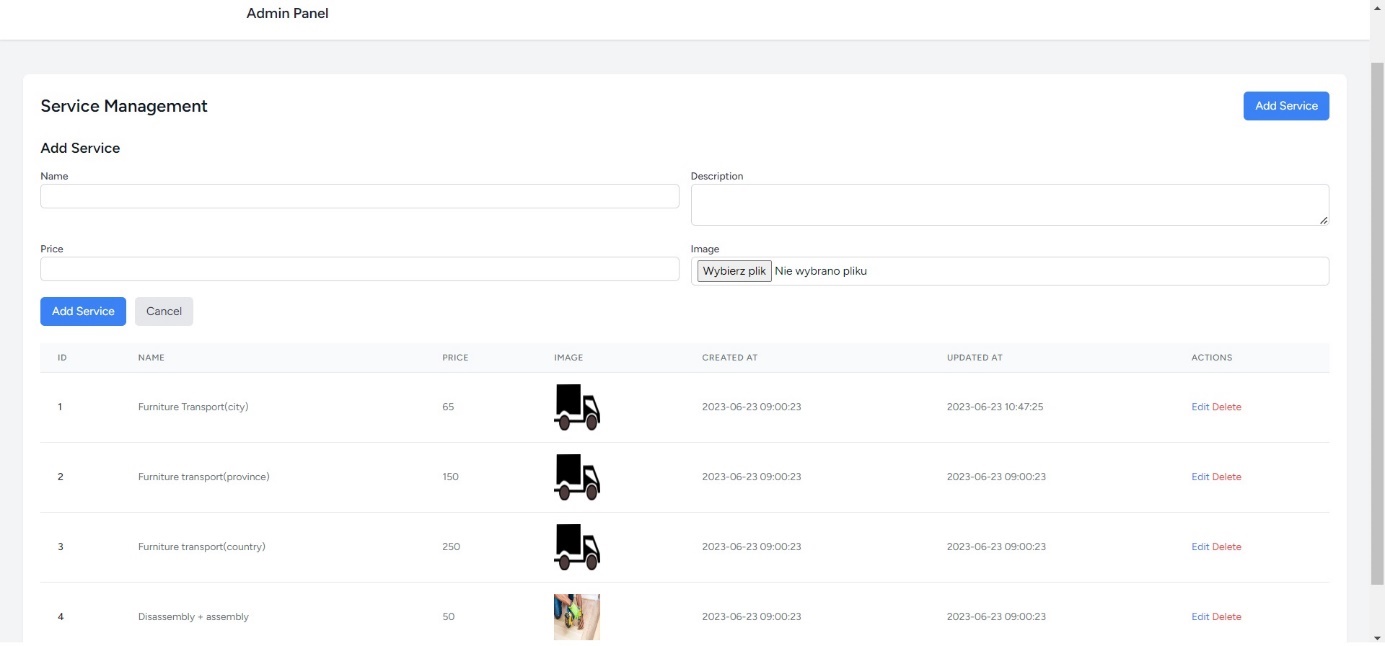
Widok panelu zarządzania swoimi danymi przez użytkownika

**Funkcje administratora:**

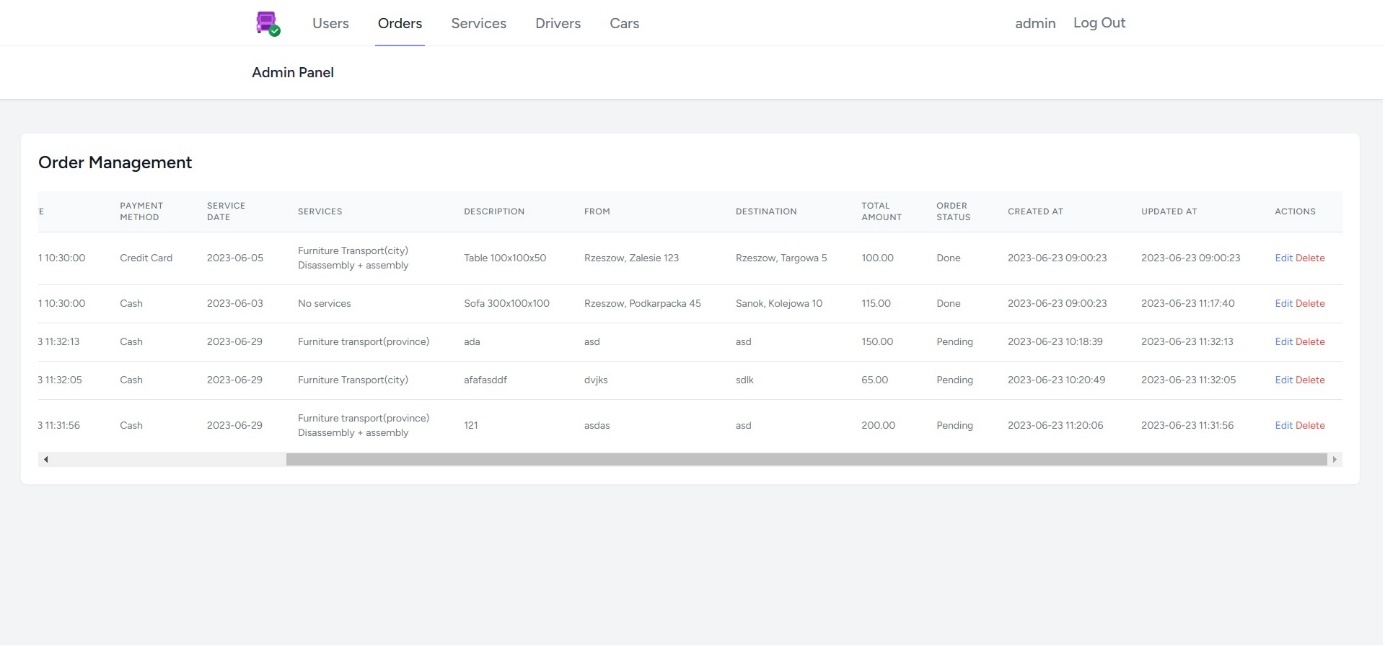
* Dodawanie, usuwanie i zarządzanie usługami, zamówieniami, kontami użytkowników, oraz samochodami i kierowcami.
* Przypisywanie konkretnych samochodów i kierowców do zamówień.



Widok zarządzania użytkownikami z panelu admina.



Widok zarządzania usługami z panelu admina.



Widok zarządzania zamówieniami z panelu admina.

* Zarządzanie samochodami i kierowcami odbywa się analogicznie do powyższych.

**Walidacja danych:**

Przykład walidacji danych dla dodawania użytkowników:

    public function store(StoreUserRequest $request)

    {

        User::create($request->all());

return redirect()->route('admin.users.index')->with('success', 'User added successfully');

    }

W UserController mamy funkcję store, która korzysta z StoreUserRequest, gdzie są przechowywane zasady walidacji danych dla tabeli użytkowników:

    public function authorize(): bool

    {

        return true;

    }

    /\*\*

     \* Get the validation rules that apply to the request.

     \*

     \* @return array<string, \Illuminate\Contracts\Validation\ValidationRule|array|string>

     \*/

    public function rules(): array

    {

        return [

            'name' => 'required|string|unique:users|max:255',

            'email' => 'required|email|unique:users|max:255',

            'password' => 'required|string|min:8',

            'first\_name' => 'required|string|max:255',

            'last\_name' => 'required|string|max:255',

            'company\_name' => 'nullable|string|max:255',

            'tax\_id' => 'required|numeric|digits:10|unique:users',

            'role' => 'required|in:user,admin',

        ];

    }

Zasady walidacji danych użytkowników określają wymagane pola i ich format, które muszą być spełnione podczas tworzenia lub aktualizowania rekordu użytkownika w systemie. Oto opis kilku tych zasad:

* 'name' - wymagane pole, które musi być unikalne w tabeli 'users' oraz mieć maksymalnie 255 znaków. Określa nazwę użytkownika.
* 'email' - wymagane pole, które musi być w formacie poprawnego adresu e-mail oraz unikalne w tabeli 'users'. Maksymalna długość to 255 znaków. Określa adres e-mail użytkownika.
* 'password' - wymagane pole, które musi być ciągiem znaków o minimalnej długości 8. Określa hasło użytkownika.
* 'tax\_id' - wymagane pole, które musi być liczbą o dokładnie 10 cyfrach i musi być unikalne w tabeli 'users'. Określa identyfikator podatkowy użytkownika.
* 'role' - wymagane pole, które musi mieć jedną z dwóch wartości: 'user' lub 'admin'. Określa rolę użytkownika w systemie.

1. **System kontroli wersji**

Projekt został zrealizowany z sytemem kontroli wersji Git.

Link do repozytorium: <https://github.com/jakubjw1/FurnitureTransportOrderSystemApp.git>