Kolegium Nauk Przyrodniczych
Uniwersytet Rzeszowski

Dokumentacja Projektu:

Komis Samochodowy

Przedmiot:

Bazy Danych

Wykonali:

Piotr Rojek, 125159

Maksymilian Przypek, 125155

Prowadzący: dr inż. Piotr Grochowalski

Rzeszów 2024

Spis treści

1. Wstęp

1.1. Cel projektu

Celem projektu jest stworzenie nowoczesnej platformy internetowej dla komisu samochodowego, która umożliwia użytkownikom wygodne i przejrzyste kupowanie, sprzedawanie oraz serwisowanie pojazdów. Platforma ta ma zapewnić użytkownikom łatwość przeglądania dostępnych samochodów, zarządzania swoimi pojazdami oraz korzystania z innych usług związanych z obrotem pojazdami. Projekt ma na celu uproszczenie procesu zakupu i sprzedaży samochodów oraz poprawienie jakości obsługi klienta poprzez zintegrowane narzędzia i nowoczesny interfejs użytkownika.

1.2. Zakres projektu

1.2.1. Interfejs użytkownika:

- Stworzenie nowoczesnego i intuicyjnego interfejsu umożliwiającego łatwe przeglądanie dostępnych pojazdów.
- Funkcje umożliwiające użytkownikom zakładanie kont, logowanie się, dodawanie i zarządzanie pojazdami.

1.2.2. Baza danych i zarządzanie pojazdami:

- Utworzenie bazy danych, która będzie regularnie aktualizowana, aby zapewnić bieżące informacje o pojazdach.
- Każdy pojazd będzie posiadał kartę z pełnymi informacjami, zdjęciami, specyfikacją, historią serwisową, ceną i aktualnym statusem.

1.2.3. Funkcjonalności dla użytkowników:

- Możliwość dodawania pojazdów, zmiany ich parametrów oraz zarządzania ich statusami (np. „W bazie”, „Na sprzedaż”, „W serwisie”, „Sprzedany”).
- Opcje zakupu pojazdów, które automatycznie aktualizują ich status i dostępność w systemie.

1.2.4. Panel administracyjny:

- Narzędzia do zarządzania całością systemu, w tym dodawanie i usuwanie pojazdów, zmiana ich statusów, przeglądanie historii transakcji i serwisów.
- Umożliwienie efektywnej organizacji pracy komisu i zapewnienie wysokiej jakości obsługi klienta.

2. Wymagania systemowe

2.1. Wymagania sprzętowe

2.1.1. Procesor:
Procesor z architekturą 64-bitową, co najmniej dwurdzeniowy (zalecany procesor czterordzeniowy lub lepszy).

2.1.2. Pamięć RAM:
Minimum 4 GB RAM (zalecane 8 GB lub więcej dla płynniejszej pracy).

2.1.3. Dysk twardy:
Co najmniej 5 GB wolnego miejsca na dysku (zalecane SSD dla lepszej wydajności).
Wolne miejsce potrzebne na instalację oprogramowania, baz danych oraz plików projektu.

2.2. Wymagania programowe

2.2.1. System operacyjny (jeden z poniższych):
- Windows 10/11 (64-bitowy).
- macOS 10.15 lub nowszy.
- Linux (dystrybucje takie jak Ubuntu 20.04 LTS lub nowsze).

2.2.2. Oprogramowanie:
- PHP w wersji 8.1 lub nowszej (wymagane dla Laravel 11).
- Composer, narzędzie do zarządzania zależnościami PHP.
- Laravel Installer lub możliwość instalacji Laravel za pośrednictwem Composer.
- PostgreSQL w wersji 12 lub nowszej (do zarządzania bazą danych).
- pgAdmin 4, narzędzie do zarządzania bazą danych PostgreSQL.

3. Instalacja i konfiguracja

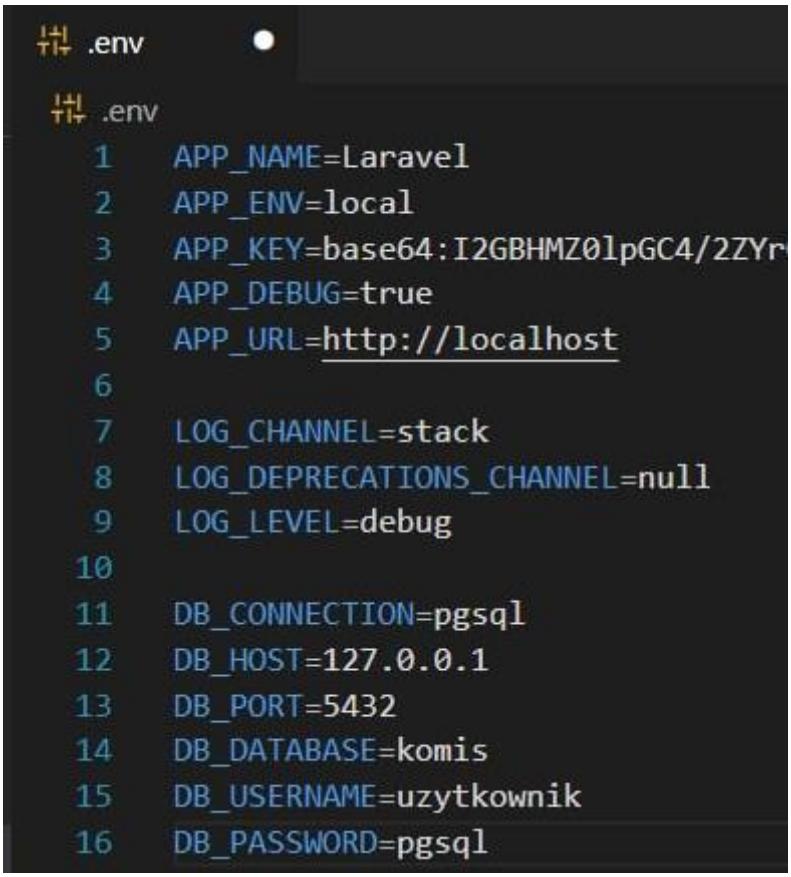
3.1. Instalacja Laravel 11

Laravel wymaga kilku narzędzi i bibliotek do działania:
- PHP >= 8.0.
- Composer (menedżer zależności dla PHP).
- Serwer WWW (np.: Apache, Nginx).
- Należy pobrać i zainstalować Composer ze strony getcomposer.org.
- Należy pobrać i zainstalować PostgresSQL oraz otworzyć program PGadmin4.

3.2. Uruchomienie serwera lokalnego

Wykonać komendę: composer install.

Następnie należy wykonać tylko MainSeeder.php aby baza danych poprawnie została zaimplementowana.

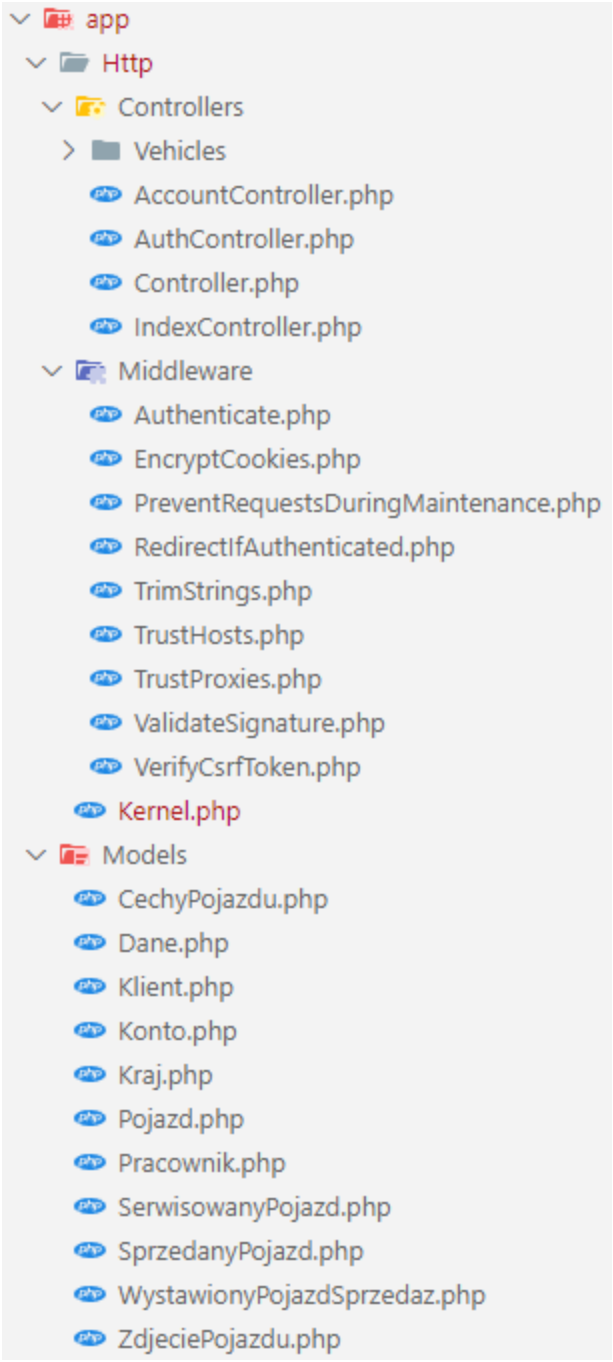


Należy wprowadzić prawidłowe dane w linii od 11-16 aby projekt działał prawidłowo.

Aby uruchomić server lokalny należy wprowadzić komendę: php artisan serve.

4. Struktura projektu

4.1. Struktura katalogów i plików



4.1.1. Controllers

W katalogu **Controllers** znajdują się kontrolery aplikacji, które zarządzają logiką odpowiedzi na żądania HTTP. Kontrolery grupują logikę związaną z konkretnymi funkcjonalnościami aplikacji.

- **AccountController.php** - Zarządza operacjami związanymi z kontami użytkowników, takimi jak rejestracja, edycja profilu, itp.
- **AuthController.php** - Odpowiada za autoryzację i autentykację użytkowników, zarządzanie logowaniem i wylogowaniem.
- **Controller.php** - Bazowy kontroler, z którego mogą dziedziczyć inne kontrolery. Zawiera wspólne metody i logikę.

- **IndexController.php** - Odpowiada za główną stronę aplikacji oraz podstawowe operacje związane z wyświetlaniem widoków.

4.1.2. Middleware

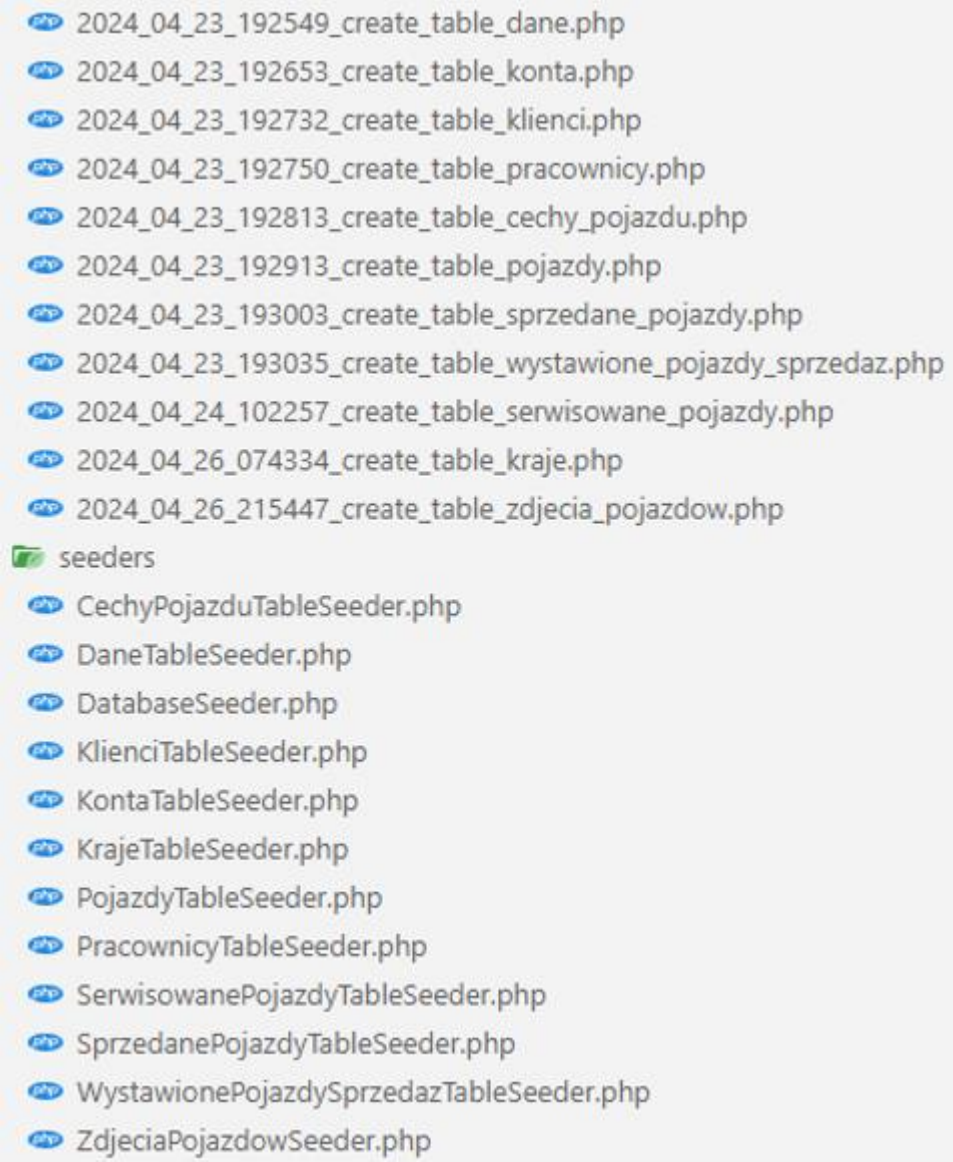
Katalog **Middleware** zawiera klasy pośredniczące, które mogą modyfikować żądania HTTP przed ich przekazaniem do kontrolerów oraz odpowiedzi przed ich wysłaniem do klienta.

- **Authenticate.php** - Middleware sprawdzający, czy użytkownik jest uwierzytelniony.
- **EncryptCookies.php** - Odpowiada za szyfrowanie ciasteczek.
- **PreventRequestsDuringMaintenance.php** - Blokuje dostęp do aplikacji podczas trybu konserwacji.
- **RedirectIfAuthenticated.php** - Przekierowuje uwierzytelnionych użytkowników do innej części aplikacji.
- **TrimStrings.php** - Usuwa białe znaki z początku i końca danych wejściowych.
- **TrustHosts.php** - Określa zaufane hosty, z których mogą pochodzić żądania.
- **TrustProxies.php** - Konfiguruje zaufane serwery proxy.
- **ValidateSignature.php** - Sprawdza ważność podpisów URL.
- **VerifyCsrfToken.php** - Chroni przed atakami CSRF (Cross-Site Request Forgery).
- **Kernel.php** - Główna klasa zarządzająca wszystkimi middleware'ami w aplikacji.

4.1.3. Models

Katalog **Models** zawiera modele, które reprezentują dane przechowywane w bazie danych oraz logikę związaną z tymi danymi.

- **CechyPojazdu.php** - Model reprezentujący cechy pojazdów.
- **Dane.php** - Model ogólny dla przechowywania różnych danych aplikacji.
- **Klient.php** - Model reprezentujący klienta komisu samochodowego.
- **Konto.php** - Model reprezentujący konto użytkownika.
- **Kraj.php** - Model reprezentujący kraj pochodzenia lub sprzedaży pojazdu.
- **Pojazd.php** - Model główny reprezentujący pojazd.
- **Pracownik.php** - Model reprezentujący pracownika komisu.
- **SerwisowanyPojazd.php** - Model reprezentujący pojazdy, które są serwisowane.
- **SprzedanyPojazd.php** - Model reprezentujący pojazdy, które zostały sprzedane.
- **WystawionyPojazdSprzedaz.php** - Model reprezentujący pojazdy wystawione na sprzedaż.
- **ZdjeciePojazdu.php** - Model reprezentujący zdjęcia pojazdów.



4.1.4. Migrations

Katalog **migrations** zawiera pliki migracji, które definiują strukturę bazy danych. Migracje pozwalają na tworzenie, modyfikowanie i usuwanie tabel oraz kolumn w bazie danych w sposób kontrolowany.

- **2014_10_12_000000_create_users_table.php** - Migracja tworząca tabelę **users** do przechowywania danych użytkowników.
- **2014_10_12_100000_create_password_reset_tokens_table.php** - Migracja tworząca tabelę **password_reset_tokens** do przechowywania tokenów resetowania haseł.
- **2019_08_19_000000_create_failed_jobs_table.php** - Migracja tworząca tabelę **failed_jobs** do przechowywania informacji o nieudanych zadaniach w kolejce.
- **2019_12_14_000001_create_personal_access_tokens_table.php** - Migracja tworząca tabelę **personal_access_tokens** do przechowywania tokenów dostępu osobistego.

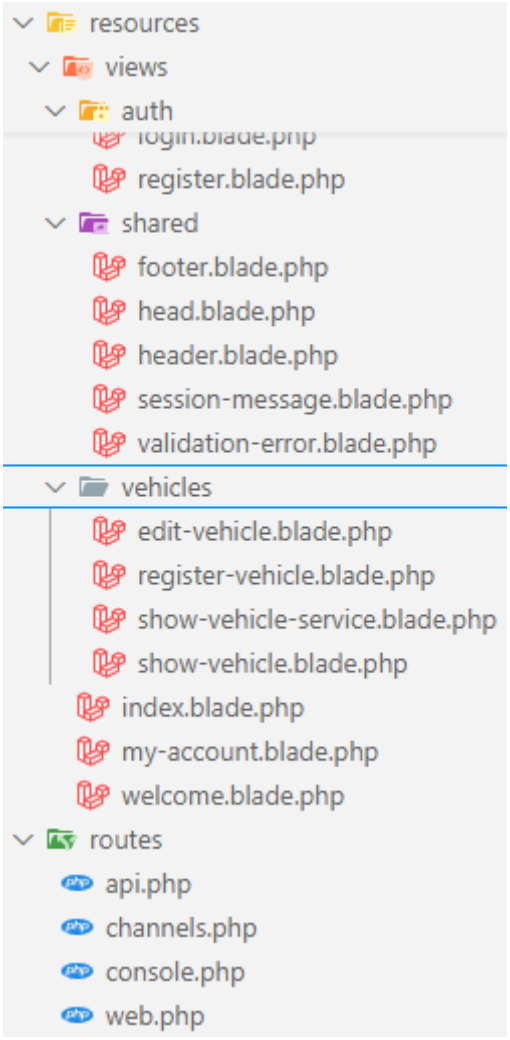
- **2024_04_23_192543_create_table_dane.php** - Migracja tworząca tabelę **dane** do przechowywania ogólnych danych aplikacji.
- **2024_04_23_192653_create_table_konta.php** - Migracja tworząca tabelę **konta** do przechowywania informacji o kontach użytkowników.
- **2024_04_23_192732_create_table_klienci.php** - Migracja tworząca tabelę **klienci** do przechowywania danych klientów.
- **2024_04_23_192750_create_table_pracownicy.php** - Migracja tworząca tabelę **pracownicy** do przechowywania danych pracowników.
- **2024_04_23_192813_create_table_cechy_pojazdu.php** - Migracja tworząca tabelę **cechy_pojazdu** do przechowywania cech pojazdów.
- **2024_04_23_192913_create_table_pojazdy.php** - Migracja tworząca tabelę **pojazdy** do przechowywania informacji o pojazdach.
- **2024_04_23_193003_create_table_sprzedane_pojazdy.php** - Migracja tworząca tabelę **sprzedane_pojazdy** do przechowywania danych o sprzedanych pojazdach.
- **2024_04_23_193035_create_table_wystawione_pojazdy_sprzedaz.php** - Migracja tworząca tabelę **wystawione_pojazdy_sprzedaz** do przechowywania danych o pojazdach wystawionych na sprzedaż.
- **2024_04_24_102257_create_table_serwisowane_pojazdy.php** - Migracja tworząca tabelę **serwisowane_pojazdy** do przechowywania informacji o pojazdach w serwisie.
- **2024_04_26_074334_create_table_kraje.php** - Migracja tworząca tabelę **kraje** do przechowywania informacji o krajach.
- **2024_04_26_215447_create_table_zdjecia_pojazdow.php** - Migracja tworząca tabelę **zdjecia_pojazdow** do przechowywania zdjęć pojazdów.

4.1.5. Seeders

Katalog **seeders** zawiera pliki seederów, które służą do wypełniania bazy danych danymi testowymi. Seedery pozwalają na szybkie tworzenie danych potrzebnych do testowania aplikacji.

- **CechyPojazduTableSeeder.php** - Seeder wypełniający tabelę **cechy_pojazdu** danymi testowymi dotyczącymi cech pojazdów.
- **DaneTableSeeder.php** - Seeder wypełniający tabelę **dane** ogólnymi danymi testowymi.
- **DatabaseSeeder.php** - Główny seeder uruchamiający pozostałe seedery, aby wypełnić całą bazę danych danymi testowymi.
- **KlienciTableSeeder.php** - Seeder wypełniający tabelę **klienci** danymi testowymi dotyczącymi klientów.
- **KontaTableSeeder.php** - Seeder wypełniający tabelę **konta** danymi testowymi dotyczącymi kont użytkowników.
- **KrajeTableSeeder.php** - Seeder wypełniający tabelę **kraje** danymi testowymi dotyczącymi krajów.
- **PojazdyTableSeeder.php** - Seeder wypełniający tabelę **pojazdy** danymi testowymi dotyczącymi pojazdów.
- **PracownicyTableSeeder.php** - Seeder wypełniający tabelę **pracownicy** danymi testowymi dotyczącymi pracowników.

- **SerwisowanePojazdyTableSeeder.php** - Seeder wypełniający tabelę **serwisowane_pojazdy** danymi testowymi dotyczącymi serwisowanych pojazdów.
- **SprzedanePojazdyTableSeeder.php** - Seeder wypełniający tabelę **sprzedane_pojazdy** danymi testowymi dotyczącymi sprzedanych pojazdów.
- **WystawionePojazdySprzedazTableSeeder.php** - Seeder wypełniający tabelę **wystawione_pojazdy_sprzedaz** danymi testowymi dotyczącymi pojazdów wystawionych na sprzedaż.
- **ZdjeciaPojazdowSeeder.php** - Seeder wypełniający tabelę **zdjecia_pojazdow** danymi testowymi dotyczącymi zdjęć pojazdów.



4.1.6. Resources/views

Katalog **views** zawiera szablony widoków Blade, które są wykorzystywane do renderowania stron internetowych aplikacji. Widoki Blade są częścią silnika szablonów Blade w Laravel, co umożliwia tworzenie dynamicznych i złożonych interfejsów użytkownika.

Katalog auth

Katalog **auth** zawiera widoki związane z autoryzacją i autentykacją użytkowników.

- **login.blade.php** - Widok strony logowania użytkownika.
- **register.blade.php** - Widok strony rejestracji nowego użytkownika.

Katalog shared
Katalog **shared** zawiera widoki wspólne dla wielu stron aplikacji, takie jak nagłówki, stopki i inne elementy wielokrotnego użytku.

- **footer.blade.php** - Widok stopki strony, która jest używana na różnych stronach aplikacji.
- **head.blade.php** - Widok zawierający elementy nagłówka HTML, takie jak meta tagi, linki do CSS itp.
- **header.blade.php** - Widok nagłówka strony, zawierający np. menu nawigacyjne.
- **session-message.blade.php** - Widok do wyświetlania wiadomości sesji (np.: komunikaty o błędach lub sukcesach).
- **validation-error.blade.php** - Widok do wyświetlania komunikatów o błędach walidacji formularzy.

Katalog vehicles

- Katalog **vehicles** zawiera widoki związane z zarządzaniem pojazdami w aplikacji.
- **edit-vehicle.blade.php** - Widok strony edycji informacji o pojeździe.
- **register-vehicle.blade.php** - Widok strony rejestracji nowego pojazdu.
- **show-vehicle-service.blade.php** - Widok szczegółów serwisowania pojazdu.
- **show-vehicle.blade.php** - Widok szczegółowych informacji o pojeździe.
- **index.blade.php** - Widok głównej strony zarządzania pojazdami.
- **my-account.blade.php** - Widok strony zarządzania kontem użytkownika.

4.1.7. Routes

Katalog **routes** zawiera pliki definiujące trasy aplikacji. Trasy określają, jakie kontrolery i metody są wywoływane w odpowiedzi na konkretne żądania HTTP.

- **api.php** - Plik definiujący trasy dla interfejsu API aplikacji. Zawiera trasy, które są dostępne pod ***/api/****.
- **channels.php** - Plik definiujący trasy dla kanałów komunikacyjnych w aplikacji, używany np. do powiadomień w czasie rzeczywistym.
- **console.php** - Plik definiujący konsolowe trasy i polecenia aplikacji. Zawiera definicje zadań do uruchomienia w konsoli.
- **web.php** - Plik definiujący trasy dla części webowej aplikacji. Zawiera trasy dostępne pod ***/***.

5. Baza danych

5.1. Diagram ERD (Entity-Relationship Diagram)

Diagram ERD (Entity-Relationship Diagram) jest graficzną reprezentacją struktury bazy danych. Przedstawia tabele (encje), kolumny (atrybuty) oraz relacje między nimi. Diagramy

ERD są kluczowe w procesie projektowania bazy danych, ponieważ umożliwiają wizualizację schematu danych oraz zrozumienie, jak różne elementy bazy danych są ze sobą powiązane.

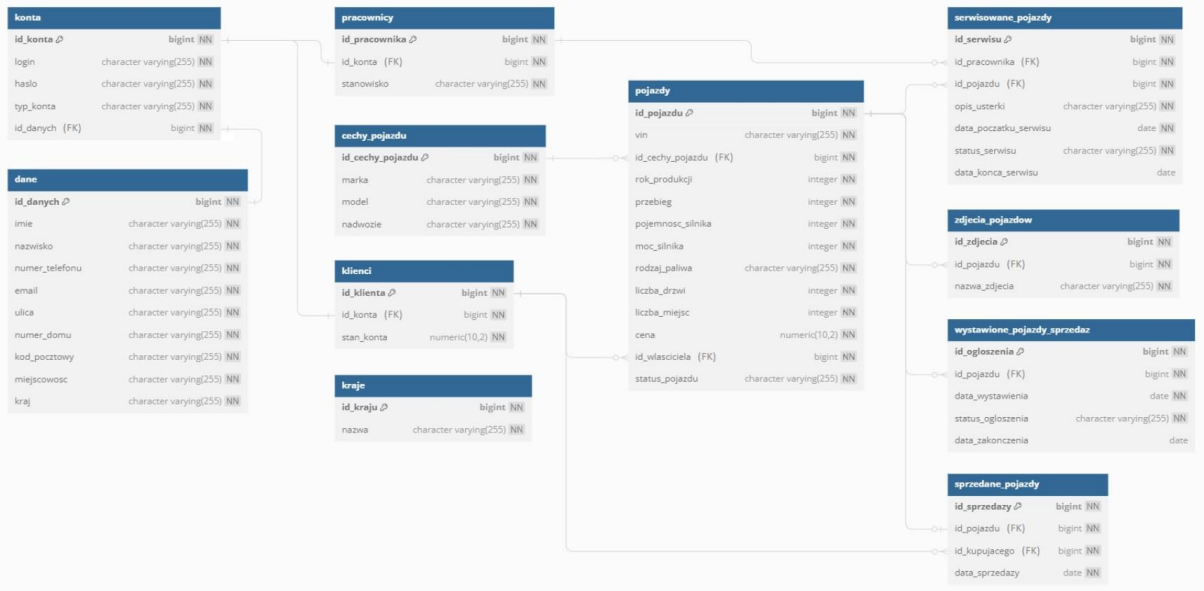


Diagram ERD przedstawiony w projekcie Komis Samochodowy pokazuje różne tabele, takie jak konta, dane, pracownicy, cechy_pojazdu, klienci, pojazdy, serwisowane_pojazdy, zdjecia_pojazdow, wystawione_pojazdy_sprzedaz oraz sprzedane_pojazdy. Relacje między tymi tabelami są również zdefiniowane, wskazując na sposób powiązania danych. Przykłady relacji obejmują:

- **Tabela konta** jest powiązana z tabelą dane poprzez klucz obcy id_danych.
- **Tabela pojazdy** ma relacje z tabelami cechy_pojazdu i zdjecia_pojazdow, gdzie id_cechy_pojazdu i id_pojazdu są kluczami obcymi.
- Reszta relacji zostanie opisana dokładnie w podpunkcie relacje między tabelami.

5.2. Opis tabel

5.2.1. Tabela konta

• ID_Konta - INT, PRIMARY KEY, NOT NULL
• Login - VARCHAR, NOT NULL
• Haslo - VARCHAR, NOT NULL
• Typ_Konta - VARCHAR, NOT NULL
• ID-danych - VARCHAR, FOREIGN KEY, NOT NULL

	id_konta [PK] bigint	login character varying (255)	haslo character varying (255)	typ_konta character varying (255)	id_danych bigint
1	1	konto1	$2y$12$qvX2wxJ.pXYLCeUcgYL11O1tHueppeYbQmOBQOdKo0AneRKYpS...	klient	1
2	2	konto2	$2y$12$wBCCWkIztAmd3Y5TZ7qD2OVm92yB56/gQ033vBQRdY8ZcKgvhK...	klient	2
3	3	konto3	$2y$12$aQlupLtIz5bTvrAFJ56mgulAi47yRCtmd9tKgfSeHPJroCOMs/0te	klient	3
4	4	konto4	$2y$12$U8GW1cUtECUl6KirASh57ecQXQG3yutx1.hVXWiDc7cCWaD7t56tq	klient	4
5	5	konto5	$2y$12$UMOrgavDq2dpL27F5q.bf.ueQNSnZcgLkUr6/WRpPCoHElbAjuVIG	pracownik	5
6	6	konto6	$2y$12$BcdzWKhhYOXS6QnumIo0dOgsSDYVDZ8I.C5DrEQI.OTLSKWI4/5R...	pracownik	6
7	7	konto7	$2y$12$6PEObJnLB9wfjJJAuv4.W.ZFBcKQZ/ZwwxP.y.EI.F/LCUIDoqsC6	klient	7
8	8	konto8	$2y$12$zJqx00.IaD3uthtblwpxR.CZS98t7xdA0qtkyFH1id7KYirsptPfO	klient	8
9	9	konto9	$2y$12$I6KNA3Eqp0xNxl9fME.EMuiUmL5SBh7znMQB3QuKtxkcMRxn8/J...	klient	9
10	10	konto10	$2y$12$esMSKcJQHkE1PUTaED.6cevWvj6vUwFBUjDqrR5Diwp8EPHU855r6	klient	10

5.2.2. Tabela dane

• PESEL - VARCHAR, PRIMARY KEY, NOT NULL
• Imie - VARCHAR, NOT NULL
• Nazwisko - VARCHAR, NOT NULL
• Numer_Telefonu - VARCHAR, NOT NULL
• Email - VARCHAR, NOT NULL
• Ulica - VARCHAR, NOT NULL
• Numer_Domu - VARCHAR, NOT NULL
• Kod_Pocztowy - VARCHAR, NOT NULL
• Miejscowosc - VARCHAR, NOT NULL
• Kraj - VARCHAR, NOT NULL

	hie character varying (255)	nazwisko character varying (255)	numer_telefonu character varying (255)	email character varying (255)	ulica character varying (255)	numer_domu character varying (255)	kod_pocztowy character varying (255)	miejscowosc character varying (255)	kraj character varying (255)
1	nna	Kowalska	123456789	anna.kowalska@example.com	Mickiewicza	12A	00-001	Warszawa	Polska
2	an	Nowak	987654321	jan.nowak@example.com	Długa	47B	00-002	Kraków	Polska
3	larta	Wiśniewska	123123123	marta.wisniewska@example.com	Krótka	3C	00-003	Łódź	Polska
4	arol	Lewandowski	321321321	karol.lewandowski@example.com	Polna	99	00-004	Gdańsk	Polska
5	gnieszka	Kaczmarek	456456456	agnieszka.kaczmarek@example.com	Zakątek	15	00-005	Poznań	Polska
6	obert	Kamiński	654654654	robert.kaminski@example.com	Szeroka	27	00-006	Wrocław	Polska
7	lagdalena	Zając	789789789	magdalena.zajac@example.com	Leśna	8	00-007	Szczecin	Polska
8	iotr	Kowal	987987987	piotr.kowal@example.com	Morska	64	00-008	Gdynia	Polska
9	orota	Mazur	111222333	dorota.mazur@example.com	Główna	4	00-009	Katowice	Polska
10	omasz	Klimek	333222111	tomasz.klimek@example.com	Parkowa	10	00-010	Rzeszów	Polska
11	lichalina	Kienska	444555666	michalina.kienska@example.com	Debowa	17F	00-007	Szczecin	Polska

5.2.3. Tabela pracownicy

• ID_Pracownika - INT, PRIMARY KEY, NOT NULL
• ID_Konta - INT, FOREIGN KEY, NOT NULL
• Stanowisko - VARCHAR, NOT NULL

	id_pracownika [PK] bigint	id_konta bigint	stanowisko character varying (255)
1	1	5	koordynator
2	2	6	admin
3	3	12	koordynator
4	4	14	koordynator

5.2.4. Tabela cechy_pojazdu

• ID_Cechy_Pojazdu - INT, PRIMARY KEY, NOT NULL
• Marka - VARCHAR, NOT NULL
• Model - VARCHAR, NOT NULL
• Nadwozie - VARCHAR, NOT NULL

	id_cechy_pojazdu [PK] bigint	marka character varying (255)	model character varying (255)	nadwozie character varying (255)
1	1	Toyota	Corolla	Sedan
2	2	Toyota	Corolla	Hatchback
3	7	Toyota	Yaris	Hatchback
4	8	Toyota	Camry	Sedan
5	9	Toyota	RAV4	SUV
6	10	Toyota	C-HR	SUV
7	11	Toyota	Highlander	SUV
8	12	Toyota	Sienna	Van
9	13	Toyota	Tacoma	Pickup
10	14	Toyota	Prius	Hatchback
11	15	Toyota	Avalon	Sedan
12	16	Honda	Civic	Sedan

5.2.5. Tabela pojazd

• VIN - VARCHAR, PRIMARY KEY, NOT NULL
• ID_Cechy_Pojazdu - INT, FOREIGN KEY, NOT NULL
• Rok_Produkcji - INT, NOT NULL
• Przebieg - INT, NOT NULL
• Pojemnosc_Silnika - INT, NOT NULL
• Moc_Silnika - INT, NOT NULL
• Rodzaj_Paliwa - VARCHAR, NOT NULL
• Liczba_Drzwi - INT, NOT NULL
• Liczba_Miejsc - INT, NOT NULL
• Cena - DECIMAL, NOT NULL
• ID_Wlasciciela - INT, FOREIGN KEY, NOT NULL
• Status_Pojazdu - VARCHAR, NOT NULL

	id_pojazdu [PK] bigint	vin character varying (255)	id_cechy_pojazdu bigint	rok_produkcji integer	przebieg integer	pojemnosc_silnika integer	moc_silnika integer	rodzaj_paliwa character varying (255)	liczba_drzwi integer	liczba_miejsc integer	cena numeric (10,2)	id_wlasciciela bigint	status_pojazdu character varyi
1	1	JM1BL1SF3A1278376	50	2010	126890	1800	140	benzyna	5	5	35000.00	3	W bazie
2	2	1FMCU9EG1AKB10553	165	2012	89371	2000	150	diesel	5	8	67000.00	2	W bazie
3	3	JTKJF5C77E3006451	180	2015	234581	2200	140	diesel	4	7	81500.00	2	W bazie
4	4	1B3LC56J68N136229	252	2009	265908	2000	114	benzyna + gaz	5	4	74000.00	3	W serwisie
5	5	3GCUKSEC1EG580856	42	2005	321876	1900	126	benzyna + gaz	5	5	29900.00	7	W serwisie
6	6	2G1WB55K569305562	63	2019	43707	3200	230	diesel	5	4	143000.00	10	W bazie
7	7	JM3KE4DY9E0335841	309	2012	156700	2600	198	benzyna	5	5	98000.00	9	W bazie
8	8	4A4AP3AU1FE003755	385	2004	358980	1400	110	diesel	4	5	34700.00	6	Na sprzedaż
9	9	1J8GN28K29W598211	201	2006	324097	2300	162	diesel + gaz	5	5	68500.00	5	Sprzedany
10	10	4T1SK12E5RU870281	125	2017	40980	3000	240	benzyna	2	2	109900.00	9	W bazie
11	11	2HJYK16586H589000	87	2020	13800	3200	312	diesel	2	2	99900.00	9	W bazie
12	12	2HJYK16586H589000	3	2010	210700	1000	87	benzyna + gaz	5	4	6700.00	4	Na sprzedaż

5.2.6. Tabela serwisowane_pojazdy

• ID_Serwisu - INT, PRIMARY KEY, NOT NULL
• ID_Pracownika - INT, FOREIGN KEY, NOT NULL
• Data_Poczatku_Serwisu - DATE, NOT NULL
• Status_Serwisu - VARCHAR, NOT NULL
• Data_Konta_Serwisu - DATE
• VIN - VARCHAR, FOREIGN KEY, NOT NULL

	id_serwisu [PK] bigint	id_pracownika bigint	id_pojazdu bigint	opis_usterki character varying (255)	data_poczatku_serwisu date	status_serwisu character varying (255)	data_konca_serwisu date
1	1	1	4	Zbita przednia szyba.	2023-04-19	W trakcie	[null]
2	2	3	5	Skrzynia biegów mi nie działa. Proszę o szybką naprawę...	2023-04-19	W trakcie	[null]
3	3	3	6	Proszę o wymianę świateł mijania.	2023-04-20	Zakończony	2023-04-24
4	4	4	7	Proszę o uzupełnienie płynu hamulcowego i chłodnicze...	2023-04-21	Zakończony	2023-04-23

5.2.7. Tabela wystawione_pojazdy_sprzedaz

• ID_Ogloszenia - INT, PRIMARY KEY, NOT NULL
• Data_Wystawienia - DATE, NOT NULL
• Status_Ogloszenia - VARCHAR, NOT NULL
• Data_Zakonczenia - DATE
• VIN - VARCHAR, FOREIGN KEY, NOT NULL

	id_ogloszenia [PK] bigint	id_pojazdu bigint	data_wystawienia date	status_ogloszenia character varying (255)	data_zakonczenia date
1	1	8	2023-04-07	W trakcie	[null]
2	2	9	2023-04-08	Zakończone	2023-04-22
3	3	11	2023-04-10	Zakończone	2023-04-15
4	4	12	2023-04-11	W trakcie	[null]

5.2.8. Tabela sprzedane_pojazdy

• ID_Sprzedazy - INT, PRIMARY KEY, NOT NULL
• Data_Sprzedazy - DATE, NOT NULL
• VIN - VARCHAR, FOREIGN KEY, NOT NULL
• ID_Kupujacego - INT, FOREIGN KEY, NOT NULL

	id_sprzedazy [PK] bigint	id_pojazdu bigint	id_kupujacego bigint	data_sprzedazy date
1	1	9	8	2023-04-22

5.2.9. Tabela klienci

• ID_klienta - INT, PRIMARY KEY, NOT NULL
• ID_konta - INT, FOREGIN KEY, NOT NULL
• Stan_Konta – VARCHAR, NOT NULL

	id_klienta [PK] bigint	id_konta bigint	stan_konta numeric (10,2)
1	1	1	109712.35
2	2	2	40971.45
3	3	3	89285.58
4	4	4	53894.20
5	5	7	23816.42
6	6	8	30013.50
7	7	9	4048.04
8	8	10	103592.74
9	9	11	4948.90
10	10	13	1379.33

5.2.10. Tabela kraje

• ID_kraju - VARCHAR, PRIMARY KEY, NOT NULL
• nazwa - VARCHAR, NOT NULL

	id_kraju [PK] bigint	nazwa character varying (255)
1	1	Afganistan
2	2	Albania
3	3	Algieria
4	4	Andora
5	5	Angola
6	6	Antigua i Barbuda
7	7	Arabia Saudyjska
8	8	Argentyna
9	9	Armenia
10	10	Australia
11	11	Austria
12	12	Azerbejdżan

5.2.11. Tabela zdjecia_pojazdow

• ID_zdjecia - INT, PRIMARY KEY, NOT NULL
•ID_pojazdu - INT, NOT NULL
• Nazwa_Zdjecia- VARCHAR, NOT NULL

5.3. Relacje między tabelami

5.3.1. Konto - Konto zawiera klucz obcy id_danych, który odnosi się do tabeli Dane. Dzięki temu model Konto posiada relację belongsTo z modelem Dane. To oznacza, że każde konto jest powiązane z dokładnie jednym rekordem z danych osobowych.

5.3.2. Klient - Klient ma klucz obcy id_konta, który odnosi się do tabeli Konta. Model Klient posiada zatem relację belongsTo z modelem Konto, oznaczając, że każdy klient jest powiązany z jednym konkretnym kontem.

5.3.3. Pojazd - Pojazd zawiera klucz obcy id_cechy_pojazdu, który odnosi się do tabeli CechyPojazdu. To oznacza, że każdy pojazd posiada zestaw cech opisany w modelu CechyPojazdu (relacja belongsTo). Dodatkowo, Pojazd posiada relację hasMany z modelem ZdjeciePojazdu, co oznacza, że pojazd może mieć wiele zdjęć.

5.3.4. Pracownik - Pracownik jest powiązany z tabelą Konta przez klucz obcy id_konta. Model Pracownik również posiada relację belongsTo do modelu Konto, co sugeruje, że każdy pracownik korzysta z jednego konta.

5.3.5. ZdjeciePojazdu - Model ZdjeciePojazdu jest powiązany z modelem Pojazd poprzez klucz obcy id_pojazdu. Oznacza to, że każde zdjęcie pojazdu jest przypisane do konkretnego pojazdu (relacja belongsTo).

5.3.6. SerwisowanyPojazd - Tabela serwisowane_pojazdy zawiera klucze obce id_pojazdu i id_pracownika, co wskazuje na relacje belongsTo do odpowiednio modeli Pojazd i Pracownik. To oznacza, że każdy serwisowany pojazd jest przypisany do konkretnego pojazdu i jest obsługiwany przez konkretnego pracownika.

5.3.7. SprzedanyPojazd i WystawionyPojazdSprzedaz - Oba modele posiadają klucz obcy id_pojazdu, który odnosi się do tabeli Pojazdy. Oba modele więc mają relację belongsTo do modelu Pojazd. SprzedanyPojazd dodatkowo ma klucz obcy id_kupujacego, co sugeruje relację z modelem Klient (choć model Klient nie jest jawnie zdefiniowany do takiej relacji w dostarczonych definicjach).

5.3.8. Tabela konta to miejsce, gdzie przechowywane są informacje niezbędne do logowania się oraz określenia roli użytkownika w systemie, takie jak login, hasło i typ konta. Każde konto jest powiązane z danymi osobowymi przechowywanymi w tabeli dane, gdzie zapisane są wszystkie kluczowe szczegóły o użytkownikach systemu, jak imię, nazwisko, adres, czy kontakt telefoniczny i emailowy.

5.3.9. Tabela klienci śledzi aktywności klientów na stronie, takie jak przeglądane pojazdy czy dokonane zakupy. Każdy klient jest związany z kontem użytkownika, co umożliwia identyfikację i personalizację obsługi.

5.3.10. Tabela pracownicy zawiera informacje o zatrudnionych w firmie, ich stanowiskach oraz przypisanym koncie użytkownika, co umożliwia zarządzanie uprawnieniami i odpowiedzialnościami związanymi z obsługą pojazdów i klientów.

5.3.11. Tabela cechy_pojazdu opisuje szczegóły każdego samochodu, takie jak marka, model, czy typ nadwozia. To ułatwia wyszukiwanie i filtrowanie pojazdów według konkretnych cech.

5.3.12. Tabela pojazdy przechowuje kompleksowe informacje o każdym samochodzie w komisie, włącznie z cechami pojazdu, właścicielem, stanem samochodu (na sprzedaż, sprzedany) oraz podstawowymi informacjami technicznymi jak rok produkcji, przebieg czy moc silnika. Tabela ta także zarządza relacjami z tabelą zdjecia_pojazdow, gdzie przechowywane są zdjęcia każdego pojazdu.

5.3.13. Tabela serwisowane_pojazdy odnosi się do pojazdów, które są aktualnie naprawiane. Zawiera informacje o odpowiedzialnym pracowniku, opisie usterki, a także statusie i czasie trwania serwisu.

5.3.14. Tabela wystawione_pojazdy_sprzedaz i tabela sprzedane_pojazdy odnoszą się do zarządzania procesem sprzedaży. Pierwsza z nich śledzi pojazdy wystawione na sprzedaż, ich status i daty ważności ogłoszenia, natomiast druga zapisuje dane o transakcjach sprzedaży, w tym identyfikator kupującego oraz datę sprzedaży pojazdu.

5.4. Migracje

5.4.1. create_table_dane

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('dane', function (Blueprint $table) {
            $table->id('id_danych');
            $table->string('imie');
            $table->string('nazwisko');
            $table->string('numer_telefonu');
            $table->string('email')->unique();
            $table->string('ulica');
            $table->string('numer_domu');
            $table->string('kod_pocztowy');
            $table->string('miejscowosc');
            $table->string('kraj');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('dane');
    }
};
```

Ta migracja tworzy tabelę **dane**, która przechowuje dane osobowe użytkowników. Tabela zawiera kolumny: **id_danych** (unikalny identyfikator), **imie**, **nazwisko**, **numer_telefonu**, **email** (musi być unikalny), **ulica**, **numer_domu**, **kod_pocztowy**, **miejscowosc** oraz **kraj**. Funkcja **up()** definiuje strukturę tabeli, a funkcja **down()** usuwa tabelę w przypadku cofania migracji.

5.4.2. create_table_konta

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('konta', function (Blueprint $table) {
            $table->id('id_konta');
            $table->string('login')->unique();
            $table->string('haslo');
            $table->string('typ_konta');
            $table->unsignedBigInteger('id_danych');
            $table->foreign('id_danych')->references('id_danych')->on('dane');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('konta');
    }
};
```

Ta migracja tworzy tabelę **konta**, która przechowuje informacje o kontach użytkowników. Tabela zawiera kolumny: **id_konta** (unikalny identyfikator), **login** (musi być unikalny), **haslo**, **typ_konta** oraz **id_danych** (klucz obcy do tabeli **dane**). Funkcja **up()** definiuje strukturę tabeli oraz relację z tabelą **dane**, a funkcja **down()** usuwa tabelę w przypadku cofania migracji.

5.4.3. create_table_klienci

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('klienci', function (Blueprint $table) {
            $table->id('id_klienta');
            $table->unsignedBigInteger('id_konta');
            $table->decimal('stan_konta', 10, 2);
            $table->foreign('id_konta')->references('id_konta')->on('konta');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('klienci');
    }
};
```

Ta migracja tworzy tabelę **klienci**, która przechowuje informacje o klientach. Tabela zawiera kolumny: **id_klienta** (unikalny identyfikator), **id_konta** (klucz obcy do tabeli **konta**) oraz **stan_konta** (saldo konta klienta). Funkcja **up()** definiuje strukturę tabeli oraz relację z tabelą **konta**, a funkcja **down()** usuwa tabelę w przypadku cofania migracji.

5.4.4. create_table_pracownicy

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('pracownicy', function (Blueprint $table) {
            $table->id('id_pracownika');
            $table->unsignedBigInteger('id_konta');
            $table->string('stanowisko');
            $table->foreign('id_konta')->references('id_konta')->on('konta');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('pracownicy');
    }
};
```

Ta migracja tworzy tabelę **pracownicy**, która przechowuje informacje o pracownikach. Tabela zawiera kolumny: **id_pracownika** (unikalny identyfikator), **id_konta** (klucz obcy do tabeli **konta**) oraz **stanowisko** (stanowisko pracownika). Funkcja **up()** definiuje strukturę tabeli oraz relację z tabelą **konta**, a funkcja **down()** usuwa tabelę w przypadku cofania migracji.

5.4.5. create_table_cechy_pojazdu

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('cechy_pojazdu', function (Blueprint $table) {
            $table->id('id_cechy_pojazdu');
            $table->string('marka');
            $table->string('model');
            $table->string('nadwozie');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('cechy_pojazdu');
    }
};
```

Ta migracja tworzy tabelę **cechy_pojazdu**, która przechowuje informacje o cechach pojazdów. Tabela zawiera kolumny: **id_cechy_pojazdu** (unikalny identyfikator), **marka**, **model** oraz **nadwozie**. Funkcja **up()** definiuje strukturę tabeli, a funkcja **down()** usuwa tabelę w przypadku cofania migracji.

5.4.6. create_table_pojazdy

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('pojazdy', function (Blueprint $table) {
            $table->id('id_pojazdu');
            $table->string('vin');
            $table->unsignedBigInteger('id_cechy_pojazdu');
            $table->integer('rok_produkcji');
            $table->integer('przebieg');
            $table->integer('pojemnosc_silnika');
            $table->integer('moc_silnika');
            $table->string('rodzaj_paliwa');
            $table->integer('liczba_drzwi');
            $table->integer('liczba_miejsc');
            $table->decimal('cena', 10, 2);
            $table->unsignedBigInteger('id_wlasciciela');
            $table->string('status_pojazdu');
            $table->foreign('id_cechy_pojazdu')->references('id_cechy_pojazdu')->on('cechy_pojazdu');
            $table->foreign('id_wlasciciela')->references('id_klienta')->on('klienci');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('pojazdy');
    }
};
```

Ta migracja tworzy tabelę **pojazdy**, która przechowuje informacje o pojazdach. Tabela zawiera kolumny: **id_pojazdu** (unikalny identyfikator), **vin**, **id_cechy_pojazdu** (klucz obcy do tabeli **cechy_pojazdu**), **rok_produkcji**, **przebieg**, **pojemnosc_silnika**, **moc_silnika**, **rodzaj_paliwa**, **liczba_drzwi**, **liczba_miejsc**, **cena**, **id_wlasciciela** (klucz obcy do tabeli **klienci**), oraz **status_pojazdu**. Funkcja **up()** definiuje strukturę tabeli oraz relacje z tabelami **cechy_pojazdu** i **klienci**, a funkcja **down()** usuwa tabelę w przypadku cofania migracji.

5.4.7. create_table_sprzedane_pojazdy

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('sprzedane_pojazdy', function (Blueprint $table) {
            $table->id('id_sprzedazy');
            $table->unsignedBigInteger('id_pojazdu');
            $table->unsignedBigInteger('id_kupujacego');
            $table->date('data_sprzedazy');
            $table->foreign('id_pojazdu')->references('id_pojazdu')->on('pojazdy');
            $table->foreign('id_kupujacego')->references('id_klienta')->on('klienci');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('sprzedane_pojazdy');
    }
};
```

Ta migracja tworzy tabelę **sprzedane_pojazdy**, która przechowuje informacje o sprzedanych pojazdach. Tabela zawiera kolumny: **id_sprzedazy** (unikalny identyfikator), **id_pojazdu** (klucz obcy do tabeli **pojazdy**), **id_kupujacego** (klucz obcy do tabeli **klienci**), oraz **data_sprzedazy**. Funkcja **up()** definiuje strukturę tabeli oraz relacje z tabelami **pojazdy** i **klienci**, a funkcja **down()** usuwa tabelę w przypadku cofania migracji.

5.4.8. create_table_wystawione_pojazdy_sprzedaz

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('wystawione_pojazdy_sprzedaz', function (Blueprint $table) {
            $table->id('id_ogloszenia');
            $table->unsignedBigInteger('id_pojazdu');
            $table->date('data_wystawienia');
            $table->string('status_ogloszenia');
            $table->date('data_zakonczenia')->nullable();
            $table->foreign('id_pojazdu')->references('id_pojazdu')->on('pojazdy');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('wystawione_pojazdy_sprzedaz');
    }
};
```

Ta migracja tworzy tabelę wystawione_pojazdy_sprzedaz, która przechowuje informacje o pojazdach wystawionych na sprzedaż. Tabela zawiera kolumny: id_ogloszenia (unikalny identyfikator), id_pojazdu (klucz obcy do tabeli pojazdy), data_wystawienia, status_ogloszenia, oraz data_zakonczenia (może być pusta). Funkcja up() definiuje strukturę tabeli oraz relację z tabelą pojazdy, a funkcja down() usuwa tabelę w przypadku cofania migracji.

5.4.9. create_table_serwisowane_pojazdy

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('serwisowane_pojazdy', function (Blueprint $table) {
            $table->id('id_serwisu');
            $table->unsignedBigInteger('id_pracownika');
            $table->unsignedBigInteger('id_pojazdu');
            $table->string('opis_usterki');
            $table->date('data_poczatku_serwisu');
            $table->string('status_serwisu');
            $table->date('data_konca_serwisu')->nullable();
            $table->foreign('id_pracownika')->references('id_pracownika')->on('pracownicy');
            $table->foreign('id_pojazdu')->references('id_pojazdu')->on('pojazdy');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('serwisowane_pojazdy');
    }
};
```

Ta migracja tworzy tabelę **serwisowane_pojazdy**, która przechowuje informacje o pojazdach w serwisie. Tabela zawiera kolumny: **id_serwisu** (unikalny identyfikator), **id_pracownika** (klucz obcy do tabeli **pracownicy**), **id_pojazdu** (klucz obcy do tabeli **pojazdy**), **opis_usterki**, **data_poczatku_serwisu**, **status_serwisu**, oraz **data_konca_serwisu** (może być pusta). Funkcja **up()** definiuje strukturę tabeli oraz relacje z tabelami **pracownicy** i **pojazdy**, a funkcja **down()** usuwa tabelę w przypadku cofania migracji.

5.4.10. create_table_kraje

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('kraje', function (Blueprint $table) {
            $table->id('id_kraju');
            $table->string('nazwa');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('kraje');
    }
};
```

Ta migracja tworzy tabelę **kraje**, która przechowuje informacje o krajach. Tabela zawiera kolumny: **id_kraju** (unikalny identyfikator) oraz **nazwa**. Funkcja **up()** definiuje strukturę tabeli, a funkcja **down()** usuwa tabelę w przypadku cofania migracji.

5.4.11. create_table_zdjecia_pojazdow

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('zdjecia_pojazdow', function (Blueprint $table) {
            $table->id('id_zdjecia');
            $table->unsignedBigInteger('id_pojazdu');
            $table->string('nazwa_zdjecia');
            $table->foreign('id_pojazdu')->references('id_pojazdu')->on('pojazdy')->onDelete('cascade');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('zdjecia_pojazdow');
    }
};
```

Ta migracja tworzy tabelę **zdjecia_pojazdow**, która przechowuje informacje o zdjęciach pojazdów. Tabela zawiera kolumny: **id_zdjecia** (unikalny identyfikator), **id_pojazdu** (klucz obcy do tabeli **pojazdy**) oraz **nazwa_zdjecia**. Funkcja **up()** definiuje strukturę tabeli oraz relację z tabelą **pojazdy** z ustawieniem usuwania kaskadowego, a funkcja **down()** usuwa tabelę w przypadku cofania migracji.

5.5. Seedery i dane testowe

Seedery są narzędziem w Laravelu, które umożliwiają wypełnienie bazy danych przykładowymi danymi. Są szczególnie przydatne podczas testowania aplikacji oraz przy tworzeniu środowisk deweloperskich, gdzie potrzebne są realistyczne dane do sprawdzenia funkcjonalności systemu. Seedery pozwalają na szybkie dodawanie dużej ilości danych do tabel, co ułatwia zarówno testowanie jak i demonstrację aplikacji. W projekcie Komis Samochodowy seedery są używane do wypełnienia tabel takimi danymi jak informacje o pojazdach, użytkownikach, transakcjach i wiele innych. Poniżej znajdują się informacje na temat seederów.

5.5.1. Cechy pojazdu

```php
public function run(): void
{
    $poczatkoweDane = [
        ['marka' => 'Toyota', 'model' => 'Corolla', 'nadwozie' => 'Sedan'],
        ['marka' => 'Toyota', 'model' => 'Corolla', 'nadwozie' => 'Hatchback'],
        ['marka' => 'Toyota', 'model' => 'Yaris', 'nadwozie' => 'Hatchback'],
        ['marka' => 'Toyota', 'model' => 'Camry', 'nadwozie' => 'Sedan'],
        ['marka' => 'Toyota', 'model' => 'RAV4', 'nadwozie' => 'SUV'],
        ['marka' => 'Toyota', 'model' => 'C-HR', 'nadwozie' => 'SUV'],
        ['marka' => 'Toyota', 'model' => 'Highlander', 'nadwozie' => 'SUV'],
        ['marka' => 'Toyota', 'model' => 'Sienna', 'nadwozie' => 'Van'],
        ['marka' => 'Toyota', 'model' => 'Tacoma', 'nadwozie' => 'Pickup'],
        ['marka' => 'Toyota', 'model' => 'Prius', 'nadwozie' => 'Hatchback'],
```

5.5.2. Dane

```php
$poczatkoweDane = [
    [
        'imie' => 'Anna',
        'nazwisko' => 'Kowalska',
        'numer_telefonu' => '123456789',
        'email' => 'anna.kowalska@example.com',
        'ulica' => 'Mickiewicza',
        'numer_domu' => '12A',
        'kod_pocztowy' => '00-001',
        'miejscowosc' => 'Warszawa',
        'kraj' => 'Polska',
    ],
```

5.5.3. Klienci

```php
public function run(): void
{
    $poczatkoweDane = [
        ['id_konta' => 1, 'stan_konta' => rand(0, 15000000) / 100],
        ['id_konta' => 2, 'stan_konta' => rand(0, 15000000) / 100],
        ['id_konta' => 3, 'stan_konta' => rand(0, 15000000) / 100],
        ['id_konta' => 4, 'stan_konta' => rand(0, 15000000) / 100],
        ['id_konta' => 7, 'stan_konta' => rand(0, 15000000) / 100],
        ['id_konta' => 8, 'stan_konta' => rand(0, 15000000) / 100],
        ['id_konta' => 9, 'stan_konta' => rand(0, 15000000) / 100],
```

5.5.4. Konta

```
public function run(): void
{
    $poczatkoweDane = [
        ['login' => 'konto1', 'haslo' => Hash::make('haslo1'), 'typ_konta' => 'klient', 'id_danych' => 1],
        ['login' => 'konto2', 'haslo' => Hash::make('haslo2'), 'typ_konta' => 'klient', 'id_danych' => 2],
        ['login' => 'konto3', 'haslo' => Hash::make('haslo3'), 'typ_konta' => 'klient', 'id_danych' => 3],
        ['login' => 'konto4', 'haslo' => Hash::make('haslo4'), 'typ_konta' => 'klient', 'id_danych' => 4],
        ['login' => 'konto5', 'haslo' => Hash::make('haslo5'), 'typ_konta' => 'pracownik', 'id_danych' => 5],
        ['login' => 'konto6', 'haslo' => Hash::make('haslo6'), 'typ_konta' => 'pracownik', 'id_danych' => 6],
        ['login' => 'konto7', 'haslo' => Hash::make('haslo7'), 'typ_konta' => 'klient', 'id_danych' => 7],
        ['login' => 'konto8', 'haslo' => Hash::make('haslo8'), 'typ_konta' => 'klient', 'id_danych' => 8],
        ['login' => 'konto9', 'haslo' => Hash::make('haslo9'), 'typ_konta' => 'klient', 'id_danych' => 9],
```

5.5.5. Kraje

```
public function run(): void
{
    $poczatkoweDane = [
        ['nazwa' => 'Afganistan'],
        ['nazwa' => 'Albania'],
        ['nazwa' => 'Algieria'],
        ['nazwa' => 'Andora'],
        ['nazwa' => 'Angola'],
        ['nazwa' => 'Antigua i Barbuda'],
        ['nazwa' => 'Arabia Saudyjska'],
        ['nazwa' => 'Argentyna'],
        ['nazwa' => 'Armenia'],
        ['nazwa' => 'Australia'],
```

5.5.6. Pojazdy

```
public function run(): void
{
    $poczatkoweDane = [
        [
            'vin' => 'JM1BL1SF3A1278376',
            'id_cechy_pojazdu' => 54,
            'rok_produkcji' => 2010,
            'przebieg' => 126890,
            'pojemnosc_silnika' => 1800,
            'moc_silnika' => 140,
            'rodzaj_paliwa' => 'benzyna',
            'liczba_drzwi' => 5,
            'liczba_miejsc' => 5,
            'cena' => 35000.00,
            'id_wlasciciela' => 3,
            'status_pojazdu' => 'W bazie'
        ],
```

5.5.7. Pracownicy

```php
public function run(): void
{
    $poczatkoweDane = [
        ['id_konta' => 5, 'stanowisko' => "koordynator"],
        ['id_konta' => 6, 'stanowisko' => "admin"],
        ['id_konta' => 12, 'stanowisko' => "koordynator"],
        ['id_konta' => 14, 'stanowisko' => "koordynator"],
    ];

    foreach ($poczatkoweDane as $dane) {
        Pracownik::firstOrCreate([
            'id_konta' => $dane['id_konta'],
        ], [
            'stanowisko' => $dane['stanowisko'],
        ]);
    }
}
```

5.5.8. Serwisowane Pojazdy

```php
$poczatkoweDane = [
    [
        'id_pracownika' => 1,
        'id_pojazdu' => 4,
        'opis_usterki' => 'Zbita przednia szyba.',
        'data_poczatku_serwisu' => Carbon::create(2023, 4, 19)->format('d.m.Y'),
        'status_serwisu' => 'W trakcie',
        'data_konca_serwisu' => null
    ],
```

5.5.9. Sprzedane pojazdy

```php
$poczatkoweDane = [
    [
        'id_pojazdu' => 9,
        'id_kupujacego' => 8,
        'data_sprzedazy' => Carbon::create(2023, 4, 22)->format('d.m.Y'),
    ],
];
```

5.5.10. Wystawione pojazdy na sprzedaż

```php
$poczatkoweDane = [
    [
        'id_pojazdu' => 8,
        'data_wystawienia' => Carbon::create(2023, 4, 7)->format('d.m.Y'),
        'status_ogloszenia' => 'W trakcie',
        'data_zakonczenia' => null
    ],
```

5.5.11. Zdjęcia pojazdów

```php
$poczatkoweDane = [
    ['id_pojazdu' => 1, 'nazwa_zdjecia' => 'Volkswagen_ID.4_1.png'],
    ['id_pojazdu' => 1, 'nazwa_zdjecia' => 'Volkswagen_ID.4_2.png'],
    ['id_pojazdu' => 1, 'nazwa_zdjecia' => 'Volkswagen_ID.4_3.png'],
    ['id_pojazdu' => 2, 'nazwa_zdjecia' => 'Renault_Kangoo_1.png'],
    ['id_pojazdu' => 2, 'nazwa_zdjecia' => 'Renault_Kangoo_2.png'],
    ['id_pojazdu' => 2, 'nazwa_zdjecia' => 'Renault_Kangoo_3.png'],
    ['id_pojazdu' => 3, 'nazwa_zdjecia' => 'Opel_Zafira_1.png'],
    ['id_pojazdu' => 3, 'nazwa_zdjecia' => 'Opel_Zafira_2.png'],
    ['id_pojazdu' => 3, 'nazwa_zdjecia' => 'Opel_Zafira_3.png'],
    ['id_pojazdu' => 4, 'nazwa_zdjecia' => 'Suzuki_GrandVitara_1.png'],
```

5.6. Funkcje w bazie danych

5.6.1. Funkcja pobierz_dane_pojazdu_plus_cechy_zdjecia

```sql
            -- Tworzenie tablicy JSON ze zdjęciami pojazdu
            (
                SELECT json_agg(json_build_object('nazwa_zdjecia', z.nazwa_zdjecia))
                FROM zdjecia_pojazdow z
                WHERE z.id_pojazdu = p.id_pojazdu
            ) AS zdjecia_pojazdu_data

        -- Łączenie tabel 'pojazdy' i 'cechy_pojazdu'
        FROM pojazdy p
        JOIN cechy_pojazdu c ON c.id_cechy_pojazdu = p.id_cechy_pojazdu
        WHERE p.id_pojazdu = id_pojazdu_param;

        -- Zakończenie transakcji
    EXCEPTION
        WHEN OTHERS THEN

            -- Obsługa błędów
            RETURN QUERY
            SELECT
                json_build_object('error', 'Wystąpił nieoczekiwany błąd podczas pobierania danych pojazdu') AS pojazd_data,
                NULL::JSON AS zdjecia_pojazdu_data;
    END;
END;
$$ LANGUAGE plpgsql;
```

Funkcja **pobierz_dane_pojazdu_plus_cechy_zdjecia** w PostgreSQL służy do pobierania szczegółowych danych o pojeździe oraz jego zdjęciach w formacie JSON. Przyjmuje jako parametr identyfikator pojazdu (**id_pojazdu_param**).

Działanie Funkcji

Funkcja rozpoczyna działanie w bloku **BEGIN**, gdzie wykonuje zapytanie SQL łączące tabele **pojazdy** i **cechy_pojazdu** na podstawie klucza **id_cechy_pojazdu**. Wynikiem jest obiekt JSON zawierający dane pojazdu, takie jak identyfikator, numer VIN, rok produkcji, przebieg, pojemność i moc silnika, rodzaj paliwa, liczba drzwi i miejsc, cena, identyfikator właściciela, status pojazdu oraz marka, model i nadwozie.

Dodatkowo, procedura tworzy tablicę JSON z nazwami zdjęć powiązanych z danym pojazdem, pobieranymi z tabeli **zdjecia_pojazdow**.

W przypadku wystąpienia błędu, blok **EXCEPTION** przechwytuje go i zwraca komunikat o błędzie w formacie JSON. Procedura kończy się zamknięciem transakcji, zwracając dwa pola: **pojazd_data** i **zdjecia_pojazdu_data**, które zawierają odpowiednio informacje o pojeździe oraz jego zdjęciach.

5.6.2. Funkcja sprawdz_wlasciciela_pojazdu

```
CREATE OR REPLACE FUNCTION sprawdz_wlasciciela_pojazdu(id_pojazdu_param INT, id_konta_param INT)
RETURNS BOOLEAN
AS $$
DECLARE

    -- Deklaracja zmiennych lokalnych
    typ_konta_ret VARCHAR;
    id_wlasciciela_ret INT;

-- Rozpoczęcie transakcji
BEGIN

    -- Pobranie typu konta
    SELECT typ_konta INTO typ_konta_ret
    FROM konta
    WHERE id_konta = id_konta_param;

    -- Sprawdzenie typu konta
    IF typ_konta_ret = 'pracownik' THEN
        RETURN FALSE;
    END IF;

    -- Pobranie ID właściciela pojazdu
    SELECT id_wlasciciela INTO id_wlasciciela_ret
    FROM pojazdy
    WHERE id_pojazdu = id_pojazdu_param;

    -- Zakończenie transakcji
    -- Zwracanie TRUE, jeśli ID właściciela pojazdu jest równe ID konta, w przeciwnym przypadku zwracanie FALSE
    RETURN (id_wlasciciela_ret = id_konta_param);
EXCEPTION
    WHEN OTHERS THEN

        -- Obsługa błędów
        -- Zwracanie FALSE w przypadku błędu
        RETURN FALSE;
END;
$$ LANGUAGE plpgsql;
```

Funkcja **sprawdz_wlasciciela_pojazdu** w PostgreSQL służy do sprawdzenia, czy dany użytkownik jest właścicielem pojazdu. Przyjmuje dwa parametry: identyfikator pojazdu (**id_pojazdu_param**) oraz identyfikator konta użytkownika (**id_konta_param**).

Działanie Funkcji

Funkcji rozpoczyna działanie od deklaracji zmiennych lokalnych **typ_konta_ret** i **id_wlasciciela_ret**, które będą przechowywać odpowiednio typ konta i identyfikator właściciela pojazdu.

W bloku **BEGIN** procedura wykonuje zapytanie SQL, które pobiera typ konta z tabeli **konta** na podstawie **id_konta_param**. Wynik jest przechowywany w zmiennej **typ_konta_ret**. Następnie procedura sprawdza, czy typ konta to 'pracownik'. Jeśli tak, procedura natychmiast zwraca **FALSE**, ponieważ pracownik nie może być właścicielem pojazdu.

Jeśli typ konta nie jest 'pracownik', procedura kontynuuje i wykonuje kolejne zapytanie SQL, które pobiera identyfikator właściciela pojazdu z tabeli **pojazdy** na podstawie **id_pojazdu_param**. Wynik jest przechowywany w zmiennej **id_wlasciciela_ret**.

Na końcu procedura porównuje identyfikator właściciela pojazdu z identyfikatorem konta użytkownika. Jeśli identyfikatory są równe, procedura zwraca **TRUE**, w przeciwnym razie zwraca **FALSE**.

W przypadku wystąpienia jakiegokolwiek błędu, blok **EXCEPTION** przechwytuje go, a procedura zwraca **FALSE**.

Procedura kończy się zamknięciem transakcji, zapewniając poprawne zwrócenie wartości logicznej określającej, czy użytkownik jest właścicielem pojazdu.

5.6.3. Funkcja pobierz_dane_pracownika

```
CREATE OR REPLACE FUNCTION pobierz_dane_pracownika(id_konta_param INT)
RETURNS TABLE(id_pracownika INT, stanowisko VARCHAR, czy_pracownik BOOLEAN)
AS $$

-- Rozpoczęcie transakcji
BEGIN

    -- Zwracanie zapytania pobierającego dane pracownika
    RETURN QUERY
    SELECT
        p.id_pracownika::INT,  -- id_pracownika = INT (pobranie i konwersja id_pracownika)
        p.stanowisko,          -- stanowisko = VARCHAR (pobranie stanowiska pracownika)
        TRUE AS czy_pracownik  -- czy_pracownik = TRUE
    FROM pracownicy p
    WHERE p.id_konta = id_konta_param;

    -- Sprawdzenie, czy nie znaleziono żadnego pracownika
    IF NOT FOUND THEN

        -- Zwrócenie wartości domyślnych, jeśli nie znaleziono żadnego pracownika
        RETURN QUERY
        SELECT
            NULL::INT AS id_pracownika,  -- id_pracownika = NULL
            NULL::VARCHAR AS stanowisko, -- stanowisko = NULL
            FALSE AS czy_pracownik;      -- czy_pracownik = FALSE
    END IF;

    -- Zakończenie transakcji
EXCEPTION
    WHEN OTHERS THEN

        -- Obsługa błędów
        RETURN QUERY
        SELECT
            NULL::INT AS id_pracownika,  -- id_pracownika = NULL
            NULL::VARCHAR AS stanowisko, -- stanowisko = NULL
            FALSE AS czy_pracownik;      -- czy_pracownik = FALSE
END;
$$ LANGUAGE plpgsql;
```

Funkcja **pobierz_dane_pracownika** w PostgreSQL służy do pobierania danych pracownika na podstawie identyfikatora konta. Przyjmuje identyfikator konta (**id_konta_param**) i zwraca tabelę z trzema kolumnami: identyfikatorem pracownika, stanowiskiem oraz flagą określającą, czy jest pracownikiem.
Funkcja rozpoczyna się od wykonania zapytania SQL, które pobiera **id_pracownika**, **stanowisko** i ustawia wartość **czy_pracownik** na **TRUE** z tabeli **pracownicy**, gdzie **id_konta** jest równy podanemu identyfikatorowi konta. Jeśli zapytanie nie zwróci żadnych wyników, procedura zwraca domyślne wartości: **NULL** dla **id_pracownika** i **stanowisko**, oraz **FALSE** dla **czy_pracownik**.
W przypadku wystąpienia błędu, procedura przechwytuje go i również zwraca domyślne wartości. Procedura kończy się zwróceniem wyników w postaci tabeli, zapewniając informację o tym, czy użytkownik jest pracownikiem oraz jego dane, jeśli są dostępne.

5.6.4. Funkcja pobierz_pojazdy_plus_cechy_zdjecia_w_serwisie

```
                'liczba_miejsc', p.liczba_miejsc,
                'cena', p.cena,
                'id_wlasciciela', p.id_wlasciciela,
                'status_pojazdu', p.status_pojazdu,
                'marka', c.marka,
                'model', c.model,
                'nadwozie', c.nadwozie
            )
            AS pojazd_data,

            -- Tworzenie tablicy JSON ze zdjęciami pojazdu
            (
                SELECT json_agg(json_build_object('nazwa_zdjecia', z.nazwa_zdjecia))
                FROM zdjecia_pojazdow z
                WHERE z.id_pojazdu = p.id_pojazdu
            )
            AS zdjecia_pojazdu_data

        -- Łączenie tabel 'pojazdy' i 'cechy_pojazdu'
        FROM pojazdy p
        JOIN cechy_pojazdu c ON c.id_cechy_pojazdu = p.id_cechy_pojazdu

        -- Warunek pobierający pojazdy z odpowiednim statusem
        WHERE p.status_pojazdu = 'W serwisie';

        -- Zakończenie transakcji
    EXCEPTION
        WHEN OTHERS THEN

            -- Obsługa błędów
            RETURN QUERY
            SELECT
                json_build_object('error', 'Wystąpił nieoczekiwany błąd podczas pobierania danych pojazdów') AS pojazd_data,
                NULL::JSON AS zdjecia_pojazdu_data;
    END;
END;
$$ LANGUAGE plpgsql;
```

Funkcja **pobierz_pojazdy_plus_cechy_zdjecia_w_serwisie** w PostgreSQL służy do pobierania szczegółowych danych o pojazdach, które są w serwisie, oraz ich zdjęciach w formacie JSON. Procedura nie przyjmuje żadnych parametrów i zwraca tabelę z dwoma kolumnami: **pojazd_data** i **zdjecia_pojazdu_data**.

Działanie Funkcji

Procedura rozpoczyna się od wykonania zapytania SQL w bloku **BEGIN**, który pobiera dane z tabel **pojazdy** i **cechy_pojazdu**, tworząc obiekt JSON zawierający szczegółowe informacje o pojeździe, takie jak identyfikator, numer VIN, rok produkcji, przebieg, pojemność i moc silnika, rodzaj paliwa, liczba drzwi i miejsc, cena, identyfikator właściciela, status pojazdu oraz marka, model i nadwozie.

Dodatkowo, procedura tworzy tablicę JSON zawierającą nazwy zdjęć powiązanych z danym pojazdem, pobieranych z tabeli **zdjecia_pojazdow**.

Procedura łączy tabele **pojazdy** i **cechy_pojazdu** na podstawie klucza **id_cechy_pojazdu** i filtruje wyniki, aby pobierać tylko pojazdy, których status jest równy 'W serwisie'.

W przypadku wystąpienia jakiegokolwiek błędu, blok **EXCEPTION** przechwytuje go i procedura zwraca komunikat o błędzie w formacie JSON. Procedura kończy się zwróceniem wyników w postaci tabeli z odpowiednimi danymi pojazdów oraz ich zdjęciami.

5.6.5. Funkcja zakup_pojazdu

```
SELECT * INTO dane_pojazdu_ret FROM pojazdy WHERE id_pojazdu = id_pojazdu_param;
IF dane_pojazdu_ret IS NULL THEN
    RETURN 'Nie znaleziono pojazdu.';
END IF;

-- Sprawdzenie, czy wystawiony pojazd jest własnością klienta
IF dane_pojazdu_ret.id_wlasciciela = id_konta_param THEN
    RETURN 'Pojazd jest już twoją własnością.';
END IF;

-- Sprawdzenie, czy wystawiony pojazd jest dostępny
IF NOT EXISTS (SELECT 1 FROM wystawione_pojazdy_sprzedaz WHERE id_pojazdu = id_pojazdu_param AND data_zakonczenia IS NULL) THEN
    RETURN 'Nie znaleziono wpisu wystawionego pojazdu.';
END IF;

-- Sprawdzenie, czy klient ma wystarczające środki
IF stan_konta_ret < dane_pojazdu_ret.cena THEN
    RETURN 'Nie masz wystarczających środków na koncie.';
END IF;

-- Aktualizacja stanu konta klienta
UPDATE klienci SET stan_konta = stan_konta_ret - dane_pojazdu_ret.cena WHERE id_konta = id_konta_param;

-- Aktualizacja statusu pojazdu
UPDATE pojazdy SET status_pojazdu = 'Sprzedany' WHERE id_pojazdu = id_pojazdu_param;

-- Rejestracja sprzedaży
INSERT INTO sprzedane_pojazdy (id_pojazdu, id_kupujacego, data_sprzedazy) VALUES (id_pojazdu_param, id_konta_param, NOW());

-- Aktualizacja statusu ogłoszenia
UPDATE wystawione_pojazdy_sprzedaz SET status_ogloszenia = 'Zakończone', data_zakonczenia = NOW() WHERE id_pojazdu = id_pojazdu_param;

-- Tworzenie nowego pojazdu w bazie na nowego właściciela
INSERT INTO pojazdy (vin, id_cechy_pojazdu, rok_produkcji, przebieg, pojemnosc_silnika, moc_silnika, rodzaj_paliwa, liczba_drzwi, licz
VALUES (dane_pojazdu_ret.vin, dane_pojazdu_ret.id_cechy_pojazdu, dane_pojazdu_ret.rok_produkcji, dane_pojazdu_ret.przebieg, dane_pojaz
RETURNING id_pojazdu INTO nowy_id_pojazdu_ret;

-- Kopiowanie zdjęć pojazdu do nowego pojazdu
FOR zdjecie_ret IN SELECT nazwa_zdjecia FROM zdjecia_pojazdow WHERE id_pojazdu = id_pojazdu_param LOOP
    INSERT INTO zdjecia_pojazdow (id_pojazdu, nazwa_zdjecia)
    VALUES (nowy_id_pojazdu_ret, zdjecie_ret.nazwa_zdjecia);
END LOOP;

-- Zakończenie transakcji
RETURN 'Pojazd został pomyślnie zakupiony.';
```

Funkcja **zakup_pojazdu** w PostgreSQL umożliwia użytkownikowi zakup pojazdu. Przyjmuje dwa parametry: identyfikator pojazdu (**id_pojazdu_param**) oraz identyfikator konta użytkownika (**id_konta_param**). Zwraca komunikat tekstowy informujący o wyniku operacji.
Funkcja najpierw sprawdza, czy konto użytkownika nie jest pracownikiem. Jeśli jest, zwraca komunikat o błędzie. Następnie pobiera stan konta klienta i dane pojazdu. Jeśli klient lub pojazd nie istnieją, zwraca odpowiedni komunikat o błędzie.
Jeżeli pojazd jest już własnością klienta lub nie jest dostępny na sprzedaż, procedura zwraca stosowne komunikaty. Sprawdza również, czy klient ma wystarczające środki na koncie. Jeśli nie, informuje o braku środków.
Procedura aktualizuje stan konta klienta, zmienia status pojazdu na "Sprzedany", rejestruje sprzedaż i aktualizuje status ogłoszenia. Tworzy nowy rekord pojazdu z nowym właścicielem i kopiuje zdjęcia pojazdu.
W przypadku sukcesu procedura zwraca komunikat "Pojazd został pomyślnie zakupiony". W razie błędu zwraca komunikat "Wystąpił problem, spróbuj ponownie później". Procedura zapewnia pełną obsługę zakupu pojazdu, weryfikując konto, środki i aktualizując odpowiednie dane.

5.6.6. Funkcja wyslij_do_serwisu

```
CREATE OR REPLACE FUNCTION wyslij_do_serwisu(id_pojazdu_param INT, opis_usterki_param TEXT)
RETURNS TEXT
AS $$
DECLARE

    -- Deklaracja zmiennych lokalnych
    id_pracownika_ret INT;
BEGIN

    -- Rozpoczęcie transakcji
    BEGIN

        -- Losowe wybranie pracownika
        SELECT id_pracownika INTO id_pracownika_ret
        FROM pracownicy
        ORDER BY RANDOM()
        LIMIT 1;

        -- Sprawdzenie, czy znaleziono pracownika
        IF id_pracownika_ret IS NULL THEN
            RETURN 'Nie znaleziono dostępnych pracowników.';
        END IF;

        -- Aktualizacja statusu pojazdu
        UPDATE pojazdy
        SET status_pojazdu = 'W serwisie'
        WHERE id_pojazdu = id_pojazdu_param;

        -- Rejestrowanie pojazdu w serwisie
        INSERT INTO serwisowane_pojazdy (id_pracownika, id_pojazdu, opis_usterki, data_poczatku_serwisu, status_serwisu,
        VALUES (id_pracownika_ret, id_pojazdu_param, opis_usterki_param, NOW(), 'W trakcie', NULL);

        -- Zakończenie transakcji
        RETURN 'Pojazd wysłany do serwisu pomyślnie.';
    EXCEPTION
        WHEN OTHERS THEN

            -- Obsługa błędów
            RETURN 'Wystąpił problem, spróbuj ponownie później.';
    END;
END;
$$ LANGUAGE plpgsql;
```

Funkcja **wyslij_do_serwisu** w PostgreSQL służy do wysyłania pojazdu do serwisu. Przyjmuje dwa parametry: identyfikator pojazdu (**id_pojazdu_param**) oraz opis usterki (**opis_usterki_param**). Zwraca komunikat tekstowy informujący o wyniku operacji.
Funkcja najpierw losowo wybiera pracownika z dostępnych w tabeli **pracownicy**. Jeśli nie znaleziono żadnego pracownika, procedura zwraca komunikat o braku dostępnych pracowników. Następnie aktualizuje status pojazdu na "W serwisie" w tabeli **pojazdy**.
Po zaktualizowaniu statusu pojazdu, procedura rejestruje pojazd w serwisie, wstawiając nowy rekord do tabeli **serwisowane_pojazdy** z odpowiednimi informacjami: identyfikatorem pracownika, identyfikatorem pojazdu, opisem usterki, datą rozpoczęcia serwisu oraz statusem serwisu ustawionym na "W trakcie".
W przypadku powodzenia, procedura zwraca komunikat "Pojazd wysłany do serwisu pomyślnie". W razie wystąpienia jakiegokolwiek błędu, procedura przechwytuje go i zwraca komunikat "Wystąpił problem, spróbuj ponownie później". Procedura zapewnia pełną obsługę procesu wysyłania pojazdu do serwisu, od wyboru pracownika po rejestrację serwisowania pojazdu.

5.6.7. Funkcja Zakończenia Serwisu Pojazdu

```
-- Rozpoczęcie transakcji
BEGIN

    -- Znalezienie ID pracownika na podstawie ID konta
    SELECT id_pracownika INTO id_pracownika_ret
    FROM pracownicy
    WHERE id_konta = id_konta_param;

    -- Sprawdzenie, czy znaleziono pracownika
    IF id_pracownika_ret IS NULL THEN
        RETURN 'Nie znaleziono odpowiedniego pracownika.';
    END IF;

    -- Znalezienie aktywnego wpisu serwisowego
    SELECT id_serwisu INTO id_serwisu_ret
    FROM serwisowane_pojazdy
    WHERE id_pojazdu = id_pojazdu_param
    AND id_pracownika = id_pracownika_ret
    AND data_konca_serwisu IS NULL
    ORDER BY data_poczatku_serwisu DESC, id_serwisu DESC
    LIMIT 1;

    -- Sprawdzenie, czy znaleziono wpis serwisowy
    IF id_serwisu_ret IS NULL THEN
        RETURN 'Nie znaleziono wpisu serwisowanego pojazdu.';
    END IF;

    -- Aktualizacja statusu pojazdu
    UPDATE pojazdy
    SET status_pojazdu = 'W bazie'
    WHERE id_pojazdu = id_pojazdu_param;

    -- Aktualizacja wpisu serwisowego
    UPDATE serwisowane_pojazdy
    SET status_serwisu = 'Zakończony', data_konca_serwisu = NOW()
    WHERE id_serwisu = id_serwisu_ret;

    -- Zakończenie transakcji
    RETURN 'Serwis pojazdu zakończony pomyślnie.';
EXCEPTION
    WHEN OTHERS THEN
```

Funkcja służy do zakończenia serwisowania pojazdu. Przyjmuje dwa parametry: identyfikator pojazdu oraz identyfikator konta użytkownika.
Funkcja rozpoczyna się od znalezienia pracownika na podstawie identyfikatora konta. Jeśli nie uda się znaleźć pracownika, procedura zwraca komunikat o braku odpowiedniego pracownika. Następnie wyszukuje aktywny wpis serwisowy dla danego pojazdu i pracownika. Aktywny wpis to taki, który nie ma ustawionej daty zakończenia serwisu. Jeśli nie znajdzie takiego wpisu, procedura zwraca komunikat o braku wpisu serwisowego.
Jeśli aktywny wpis serwisowy zostanie znaleziony, procedura aktualizuje status pojazdu na "W bazie" oraz oznacza wpis serwisowy jako zakończony, ustawiając datę zakończenia serwisu na bieżący czas. Na koniec procedura zwraca komunikat "Serwis pojazdu zakończony pomyślnie". W przypadku wystąpienia jakiegokolwiek błędu, procedura przechwytuje go i zwraca komunikat "Wystąpił problem, spróbuj ponownie później".
Funkcja ta umożliwia zakończenie serwisowania pojazdu, weryfikując pracownika, aktualizując status pojazdu i rejestrując zakończenie serwisu w sposób bezpieczny i z obsługą ewentualnych błędów.

5.6.8. Funkcja zakoncz_sprzedaz

```
CREATE OR REPLACE FUNCTION zakoncz_sprzedaz(id_pojazdu_param INT)
RETURNS TEXT AS $$
DECLARE

    -- Deklaracja zmiennych lokalnych
    id_ogloszenia_ret INT;
BEGIN

    -- Rozpoczęcie transakcji
    BEGIN

        -- Znalezienie aktywnego wpisu sprzedaży
        SELECT id_ogloszenia INTO id_ogloszenia_ret
        FROM wystawione_pojazdy_sprzedaz
        WHERE id_pojazdu = id_pojazdu_param AND data_zakonczenia IS NULL
        ORDER BY data_wystawienia DESC, id_ogloszenia DESC
        LIMIT 1;

        -- Sprawdzenie, czy znaleziono wpis sprzedaży
        IF id_ogloszenia_ret IS NULL THEN
            RETURN 'Nie znaleziono wpisu wystawionego pojazdu.';
        END IF;

        -- Aktualizacja statusu pojazdu
        UPDATE pojazdy
        SET status_pojazdu = 'W bazie'
        WHERE id_pojazdu = id_pojazdu_param;

        -- Aktualizacja wpisu sprzedaży
        UPDATE wystawione_pojazdy_sprzedaz
        SET status_ogloszenia = 'Zakończone', data_zakonczenia = NOW()
        WHERE id_ogloszenia = id_ogloszenia_ret;

        -- Zakończenie transakcji
        RETURN 'Sprzedaż pojazdu zakończona pomyślnie.';
    EXCEPTION
        WHEN OTHERS THEN

            -- Obsługa błędów
            RETURN 'Wystąpił problem, spróbuj ponownie później.';
    END;
END;
```

Funkcja **zakoncz_sprzedaz** w PostgreSQL służy do zakończenia sprzedaży pojazdu. Przyjmuje jeden parametr: identyfikator pojazdu (**id_pojazdu_param**). Procedura zwraca komunikat tekstowy informujący o wyniku operacji.
Funkcja rozpoczyna się od próby znalezienia aktywnego wpisu sprzedaży dla danego pojazdu, gdzie **data_zakonczenia** jest **NULL**. Zapytanie sortuje wyniki według daty wystawienia oraz identyfikatora ogłoszenia w kolejności malejącej i ogranicza wynik do jednego wpisu. Jeśli nie uda się znaleźć wpisu sprzedaży, procedura zwraca komunikat o braku wpisu.
Jeśli wpis sprzedaży zostanie znaleziony, procedura aktualizuje status pojazdu na "W bazie" w tabeli **pojazdy**. Następnie aktualizuje status ogłoszenia sprzedaży na "Zakończone" oraz ustawia datę zakończenia sprzedaży na bieżący czas.
W przypadku powodzenia, procedura zwraca komunikat "Sprzedaż pojazdu zakończona pomyślnie". W razie wystąpienia jakiegokolwiek błędu, blok **EXCEPTION** przechwytuje go i procedura zwraca komunikat "Wystąpił problem, spróbuj ponownie później".
Funkcja ta zapewnia bezpieczny sposób zakończenia sprzedaży pojazdu, aktualizując statusy zarówno pojazdu, jak i ogłoszenia sprzedaży, oraz rejestrując zakończenie transakcji w bazie danych.

5.6.9. Funkcja wystaw_pojazd_na_sprzedaz

```
CREATE OR REPLACE FUNCTION wystaw_pojazd_na_sprzedaz(id_pojazdu_param INT)
RETURNS TEXT
AS $$
BEGIN

    -- Rozpoczęcie transakcji
    BEGIN

        -- Aktualizacja statusu pojazdu
        UPDATE pojazdy
        SET status_pojazdu = 'Na sprzedaż'
        WHERE id_pojazdu = id_pojazdu_param;

        -- Sprawdzenie, czy aktualizacja powiodła się
        IF NOT FOUND THEN
            RETURN 'Wystąpił problem, spróbuj ponownie później.';
        END IF;

        -- Wstawienie nowego wpisu w tabeli sprzedaży pojazdów
        INSERT INTO wystawione_pojazdy_sprzedaz (id_pojazdu, data_wystawienia, status_ogloszenia,
        VALUES (id_pojazdu_param, NOW(), 'W trakcie', NULL);

        -- Zakończenie transakcji
        RETURN 'Pojazd wystawiony na sprzedaż pomyślnie.';
    EXCEPTION
        WHEN OTHERS THEN

            -- Obsługa błędów
            RETURN 'Wystąpił problem, spróbuj ponownie później.';
    END;
END;
$$ LANGUAGE plpgsql;
```

Funkcja **wystaw_pojazd_na_sprzedaz** w PostgreSQL służy do wystawienia pojazdu na sprzedaż. Przyjmuje jeden parametr: identyfikator pojazdu (**id_pojazdu_param**). Procedura zwraca komunikat tekstowy informujący o wyniku operacji.
Funkcja rozpoczyna się od aktualizacji statusu pojazdu na "Na sprzedaż" w tabeli **pojazdy** na podstawie podanego identyfikatora pojazdu. Następnie sprawdza, czy aktualizacja statusu powiodła się. Jeśli nie, zwraca komunikat o wystąpieniu problemu.
Jeśli aktualizacja statusu pojazdu jest udana, procedura wstawia nowy wpis do tabeli **wystawione_pojazdy_sprzedaz**, zawierający identyfikator pojazdu, datę wystawienia, status ogłoszenia ustawiony na "W trakcie" oraz **NULL** dla daty zakończenia.
W przypadku sukcesu procedura zwraca komunikat "Pojazd wystawiony na sprzedaż pomyślnie". Jeśli wystąpi jakikolwiek błąd, procedura przechwytuje go i zwraca komunikat "Wystąpił problem, spróbuj ponownie później".
Funkcja ta zapewnia bezpieczny sposób na wystawienie pojazdu na sprzedaż, aktualizując odpowiednie dane pojazdu i rejestrując nowe ogłoszenie sprzedaży.

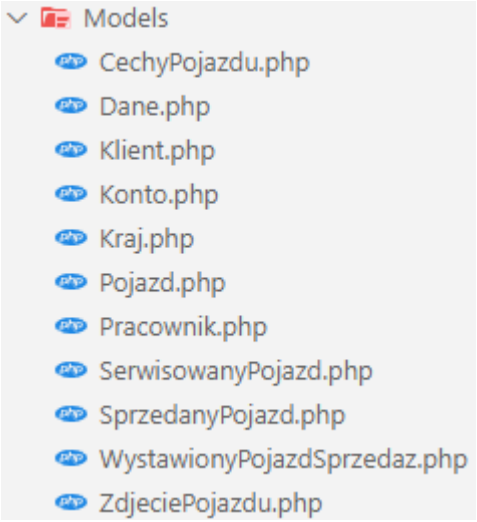
5.6.10. Funkcja usun_pojazd

```
CREATE OR REPLACE FUNCTION usun_pojazd(id_pojazdu_param INT)
RETURNS TEXT
AS $$
BEGIN

    -- Rozpoczęcie transakcji
    BEGIN

        -- Usuwanie powiązanych rekordów sprzedanych pojazdów
        DELETE FROM sprzedane_pojazdy WHERE id_pojazdu = id_pojazdu_param;

        -- Usuwanie powiązanych rekordów pojazdów w serwisie
        DELETE FROM serwisowane_pojazdy WHERE id_pojazdu = id_pojazdu_param;

        -- Usuwanie powiązanych rekordów wystawionych pojazdów na sprzedaż
        DELETE FROM wystawione_pojazdy_sprzedaz WHERE id_pojazdu = id_pojazdu_param;

        -- Usuwanie powiązanych rekordów zdjęć pojazdów
        DELETE FROM zdjecia_pojazdow WHERE id_pojazdu = id_pojazdu_param;

        -- Usuwanie pojazdu
        DELETE FROM pojazdy WHERE id_pojazdu = id_pojazdu_param;
        IF NOT FOUND THEN
            RETURN 'Nie znaleziono pojazdu do usunięcia.';
        END IF;

        -- Zakończenie transakcji
        RETURN 'Pojazd został pomyślnie usunięty.';
    EXCEPTION
        WHEN OTHERS THEN

            -- Obsługa wyjątków
            RETURN 'Wystąpił problem, spróbuj ponownie później.';
    END;
END;
$$ LANGUAGE plpgsql;
```

Funkcja **usun_pojazd** w PostgreSQL służy do usunięcia pojazdu oraz wszystkich powiązanych z nim rekordów z bazy danych. Przyjmuje jeden parametr: identyfikator pojazdu (**id_pojazdu_param**). Procedura zwraca komunikat tekstowy informujący o wyniku operacji.Procedura rozpoczyna się od usunięcia wszystkich powiązanych rekordów z innych tabel. Kolejno wykonuje operacje usuwania dla tabel: **sprzedane_pojazdy** - usunięcie rekordów powiązanych z danym pojazdem. **serwisowane_pojazdy** - usunięcie rekordów serwisowych powiązanych z danym pojazdem. **wystawione_pojazdy_sprzedaz** - usunięcie rekordów sprzedażowych powiązanych z danym pojazdem. **zdjecia_pojazdow** - usunięcie zdjęć powiązanych z danym pojazdem.Następnie procedura usuwa sam pojazd z tabeli **pojazdy**. Jeśli nie uda się znaleźć pojazdu do usunięcia, procedura zwraca komunikat "Nie znaleziono pojazdu do usunięcia".W przypadku powodzenia procedura zwraca komunikat "Pojazd został pomyślnie usunięty". Jeśli wystąpi jakikolwiek błąd, blok **EXCEPTION** przechwytuje go i procedura zwraca komunikat "Wystąpił problem, spróbuj ponownie później".Procedura zapewnia pełne usunięcie pojazdu z bazy danych wraz ze wszystkimi powiązanymi rekordami, co zapobiega pozostawieniu niepotrzebnych danych.

6. Modele

6.1. Struktura modeli



Struktura modeli w projekcie Laravel obejmuje różne klasy reprezentujące encje w bazie danych, które są wykorzystywane w aplikacji Komis Samochodowy. Każdy model odpowiada konkretnej tabeli w bazie danych i zawiera definicje relacji oraz logikę biznesową związaną z daną encją. Oto krótkie opisy poszczególnych modeli:

- **CechyPojazdu.php**: Model reprezentujący cechy pojazdu, takie jak marka, model i nadwozie. Zawiera definicje relacji z modelem **Pojazd**.
- **Dane.php**: Model przechowujący dane osobowe użytkowników, takie jak imię, nazwisko, adres i numer telefonu. Powiązany z modelem **Konto**.
- **Klient.php**: Model reprezentujący klienta, zawierający informacje o koncie klienta oraz stanie konta. Powiązany z modelem **Konto** i innymi modelami związanymi z transakcjami.
- **Konto.php**: Model zarządzający informacjami o koncie użytkownika, takimi jak login, hasło i typ konta (np. klient, pracownik). Jest podstawą do autoryzacji i zarządzania użytkownikami.
- **Kraj.php**: Model przechowujący informacje o krajach. Może być używany do powiązania z adresem w modelu **Dane**.
- **Pojazd.php**: Model reprezentujący pojazdy w systemie. Zawiera szczegółowe informacje o pojazdach, takie jak numer VIN, rok produkcji, przebieg, pojemność silnika, cena i status pojazdu. Powiązany z modelami **CechyPojazdu**, **ZdjeciePojazdu** i innymi.
- **Pracownik.php**: Model zarządzający danymi pracowników, w tym stanowiskiem i identyfikatorem konta. Powiązany z modelem **Konto**.
- **SerwisowanyPojazd.php**: Model reprezentujący pojazdy, które są serwisowane. Zawiera informacje o pracowniku serwisującym pojazd, opisie usterki, dacie rozpoczęcia i zakończenia serwisu oraz statusie serwisu.

- **SprzedanyPojazd.php**: Model przechowujący informacje o sprzedanych pojazdach, w tym dane kupującego, datę sprzedaży oraz powiązane pojazdy.
- **WystawionyPojazdSprzedaz.php**: Model reprezentujący pojazdy wystawione na sprzedaż. Zawiera informacje o dacie wystawienia, statusie ogłoszenia i dacie zakończenia ogłoszenia.
- **ZdjeciePojazdu.php**: Model przechowujący zdjęcia pojazdów. Powiązany z modelem **Pojazd**.

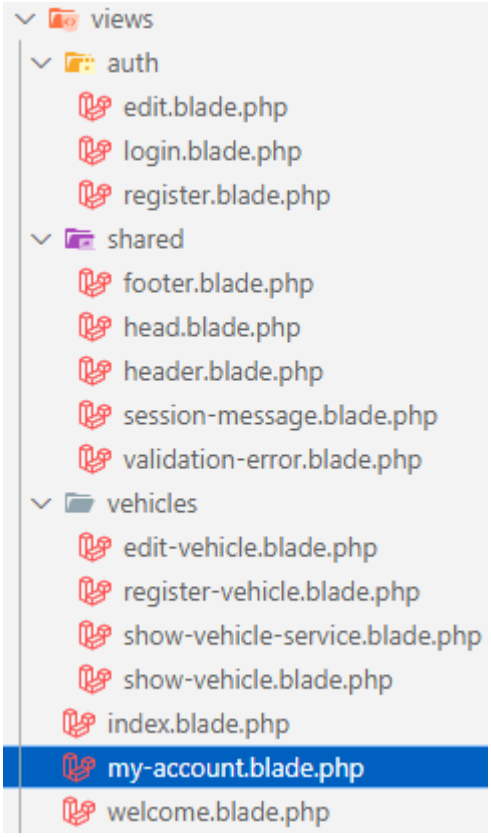
Każdy z tych modeli odgrywa istotną rolę w zarządzaniu danymi aplikacji i zapewnia integralność oraz spójność danych poprzez odpowiednio zdefiniowane relacje i logikę biznesową.

6.2. Relacje między modelami

- **Model CechyPojazdu** nie definiuje żadnych relacji z innymi modelami. Jest używany przez model **Pojazd** do powiązania cech pojazdu z danymi pojazdu.
- **Model Dane** jest powiązany z modelem **Konto** przez relację **belongsTo**. Każde konto ma swoje dane osobowe przechowywane w tej tabeli.
- **Model Klient** jest powiązany z modelem **Konto** przez relację **belongsTo**. Każdy klient ma jedno konto powiązane z jego danymi klienta.
- **Model Konto** jest powiązane z modelem **Dane** przez relację **belongsTo**. Każde konto użytkownika ma przypisane dane osobowe.
- **Konto** jest także powiązane z modelem **Klient** przez relację **hasOne**, co oznacza, że jedno konto może być powiązane z jednym klientem.
- **Konto** jest powiązane z modelem **Pracownik** przez relację **hasOne**, co oznacza, że jedno konto może być powiązane z jednym pracownikiem.
- **Model Kraj** Nie definiuje żadnych relacji z innymi modelami. Przechowuje informacje o krajach, które mogą być używane w innych tabelach.
- **Model Pojazd** ma relację **hasMany** z modelem **ZdjeciePojazdu**, co oznacza, że jeden pojazd może mieć wiele zdjęć.
- **Pojazd** jest powiązany z modelem **CechyPojazdu** przez relację **belongsTo**, co oznacza, że każdy pojazd ma przypisane cechy.
- **Model Pracownik** jest powiązany z modelem **Konto** przez relację **belongsTo**. Każdy pracownik ma jedno konto powiązane z jego danymi pracownika.
- **Model SerwisowanyPojazd** nie definiuje żadnych bezpośrednich relacji w kodzie, ale jest powiązany z modelami **Pracownik** i **Pojazd** przez klucze obce w tabeli **serwisowane_pojazdy**.
- **Model SprzedanyPojazd** nie definiuje żadnych bezpośrednich relacji w kodzie, ale jest powiązany z modelami **Pojazd** i **Klient** przez klucze obce w tabeli **sprzedane_pojazdy**.
- **Model WystawionyPojazdSprzedaz** nie definiuje żadnych bezpośrednich relacji w kodzie, ale jest powiązany z modelem **Pojazd** przez klucz obcy w tabeli **wystawione_pojazdy_sprzedaz**.
- **Model ZdjeciePojazdu** jest powiązany z modelem **Pojazd** przez relację **belongsTo**, co oznacza, że każde zdjęcie jest przypisane do jednego pojazdu.

7. Widoki

7.1. Struktura widoków



Katalog auth:
Ten katalog zawiera widoki związane z autoryzacją i rejestracją użytkowników. Znajdują się tu:

- **edit.blade.php**: Widok do edycji danych użytkownika.
- **login.blade.php**: Widok formularza logowania.
- **register.blade.php**: Widok formularza rejestracji.

Katalog shared:
Katalog ten zawiera wspólne komponenty, które mogą być wykorzystywane w różnych miejscach aplikacji. Są to:

- **footer.blade.php**: Widok stopki strony.
- **head.blade.php**: Widok sekcji **<head>** HTML, zawierający metadane i linki do zasobów.
- **header.blade.php**: Widok nagłówka strony.
- **session-message.blade.php**: Widok do wyświetlania komunikatów sesji, takich jak błędy czy potwierdzenia.
- **validation-error.blade.php**: Widok do wyświetlania błędów walidacji formularzy.

Katalog vehicles:
Katalog ten zawiera widoki związane z zarządzaniem pojazdami. Znajdują się tu:

- **edit-vehicle.blade.php**: Widok formularza edycji pojazdu.
- **register-vehicle.blade.php**: Widok formularza rejestracji nowego pojazdu.
- **show-vehicle-service.blade.php**: Widok szczegółów serwisu pojazdu.
- **show-vehicle.blade.php**: Widok szczegółów pojazdu.
- **index.blade.php**: Widok listy pojazdów.
- **my-account.blade.php**: Widok informacji o koncie użytkownika, prawdopodobnie z listą jego pojazdów i innymi danymi.

Katalog główny views:
W głównym katalogu **views** znajduje się także widok:

- **welcome.blade.php**: Widok strony powitalnej aplikacji, prawdopodobnie zawierający ogólne informacje o komisie samochodowym i podstawowe funkcje nawigacyjne.

8. Trasy (Routes)

8.1. Opis kluczowych tras

8.1.1. API Routes (api.php)
Plik ten definiuje trasy API, które są załadowane przez RouteServiceProvider i przypisane do grupy middleware api. Trasy w tym pliku są wykorzystywane do obsługi żądań API, takich jak pobieranie danych uwierzytelnionego użytkownika.

8.1.2. Broadcast Channels (channels.php)
Plik ten definiuje kanały nadawcze, które aplikacja obsługuje. Umożliwia autoryzację użytkowników do słuchania na określonych kanałach. Na przykład, kanały mogą być używane do komunikacji w czasie rzeczywistym między użytkownikami a serwerem.

8.1.3. Console Routes (console.php)
Ten plik definiuje trasy konsolowe, które umożliwiają definiowanie zadań do uruchomienia w konsoli. Trasy konsolowe pozwalają na tworzenie i wykonywanie poleceń Artisan, które mogą wykonywać różne zadania, takie jak wyświetlanie inspirujących cytatów.

8.1.4. Web Routes (web.php)
Plik ten definiuje trasy webowe, które są załadowane przez RouteServiceProvider i przypisane do grupy middleware web. Zawiera trasy związane z autoryzacją, kontem użytkownika i zarządzaniem pojazdami. Trasy te obsługują żądania HTTP związane z logowaniem, rejestracją, zarządzaniem kontem oraz operacjami na pojazdach, takimi jak edycja, rejestracja, sprzedaż i serwis.

9. Autoryzacja i uwierzytelnienie

9.1. Role i uprawnienia

9.1.1. Kontroler IndexController

```php
<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use App\Models\Pojazd;
use App\Models\Klient;
use App\Models\Pracownik;
use Illuminate\Support\Facades\Auth;

class IndexController extends Controller
{
    public function index()
    {
        $konto = Auth::user();
        $klient = null;
        $pracownik = null;
        $czyAdmin = false;
        $pojazdy = Pojazd::with('zdjecia', 'cechyPojazdu')->where('status_pojazdu', 'Na sprzedaż')->get();

        if (Auth::check()) {
            $czyAdmin = ($konto->typ_konta === 'pracownik');
            if ($czyAdmin) {
                $pojazdy = Pojazd::with('zdjecia', 'cechyPojazdu')->get();
                $pracownik = Pracownik::where('id_konta', $konto->id_konta)->first();
            } else {
                $klient = Klient::where('id_konta', $konto->id_konta)->first();
            }
        }

        return view('index', compact('pojazdy', 'czyAdmin', 'konto', 'klient', 'pracownik'));
    }
}
```

Kontroler **IndexController** jest kluczowym elementem projektu Laravel Komis Samochodowy, odpowiedzialnym za wyświetlanie strony głównej aplikacji. W kontrolerze tym znajduje się metoda **index**, która zarządza logiką wyświetlania pojazdów na sprzedaż oraz dostarcza dane o zalogowanym użytkowniku w zależności od jego typu konta - klienta lub pracownika.

Gdy metoda **index** jest wywoływana, najpierw pobiera zalogowanego użytkownika przy użyciu fasady **Auth**. Następnie inicjalizowane są zmienne **$klient**, **$pracownik** i **$czyAdmin** jako **null** lub **false**. Te zmienne będą później używane do przechowywania informacji o użytkowniku.

Kolejnym krokiem jest pobranie wszystkich pojazdów, które mają status "Na sprzedaż". Dane pojazdów są pobierane wraz z powiązanymi zdjęciami i cechami pojazdów. Jeśli użytkownik jest zalogowany, metoda sprawdza, czy jest on pracownikiem (administratorem). Jeśli tak, zmienna **$czyAdmin** jest ustawiona na **true**. Dla pracowników pobierane są wszystkie pojazdy, niezależnie od ich statusu, oraz dane pracownika powiązanego z kontem.

W przypadku, gdy zalogowany użytkownik nie jest pracownikiem, metoda zakłada, że jest on klientem, i pobiera odpowiednie dane klienta powiązane z jego kontem.

Na koniec metoda zwraca widok **index**, przekazując do niego zmienne: pojazdy, flagę administratora, dane konta, dane klienta i dane pracownika. Dzięki temu widok **index** może renderować stronę główną aplikacji z odpowiednimi informacjami dostosowanymi do rodzaju użytkownika, który przegląda stronę.

Kontroler **IndexController** zapewnia, że strona główna aplikacji jest dynamiczna i odpowiednio dostosowana do różnych typów użytkowników, umożliwiając im przeglądanie pojazdów oraz zarządzanie ich danymi w sposób zgodny z ich uprawnieniami.

9.2. Proces logowania i rejestracji

9.2.1. Kontroler AuthController

```
            'haslo.string' => 'Pole \'Hasło\' musi być ciągiem znaków.',
            'haslo.min' => 'Pole \'Hasło\' musi zawierać co najmniej 4 znaki.',
            'haslo.max' => 'Pole \'Hasło\' może zawierać maksymalnie 255 znaków.',
        ]);

        if (Auth::attempt(['login' => $credentials['login'], 'password' => $credentials['haslo']])) {
            $request->session()->regenerate();
            return redirect()->route('index')->with('success', 'Zalogowano pomyślnie.');
        } else {
            return back()->withErrors([
                'blad_logowania' => 'Podany \'Login\' lub \'Hasło\' są nieprawidłowe.',
            ])->withInput($request->only('login', 'haslo'));
        }
    }

    public function logout(Request $request)
    {
        Auth::logout();
        $request->session()->invalidate();
        $request->session()->regenerateToken();
        return redirect()->route('index');
    }

    public function redirectToHome()
    {
        return redirect()->route('index');
    }

    public function register()
    {
        if (Auth::check()) {
            return redirect()->route('index');
        }
        return view('auth.register');
    }

    public function registerValidate(Request $request)
    {
        $validatedData = $request->validate([
            // Dane użytkownika
            'imie' => 'required|string|max:255',
```

Kontroler **AuthController** jest odpowiedzialny za logikę rejestracji, logowania, wylogowania oraz edycji danych konta użytkownika w aplikacji Laravel Komis Samochodowy. Oto szczegółowy opis działania poszczególnych metod tego kontrolera:

Metoda **login** sprawdza, czy użytkownik jest już zalogowany. Jeśli tak, przekierowuje go na stronę główną (**index**). Jeśli użytkownik nie jest zalogowany, wyświetla formularz logowania.

Metoda **loginAuthenticate** obsługuje proces uwierzytelniania użytkownika. Przyjmuje dane z formularza logowania, waliduje je, a następnie próbuje uwierzytelnić użytkownika przy użyciu fasady **Auth**. Jeśli dane logowania są poprawne, sesja użytkownika jest regenerowana, a użytkownik zostaje przekierowany na stronę główną z komunikatem o pomyślnym logowaniu. W przypadku błędnych danych, użytkownik zostaje przekierowany z powrotem do formularza logowania z odpowiednim komunikatem o błędzie.

Metoda **logout** wylogowuje użytkownika, unieważnia sesję i regeneruje token sesji. Następnie użytkownik jest przekierowany na stronę główną.

Metoda **redirectToHome** po prostu przekierowuje użytkownika na stronę główną (**index**).

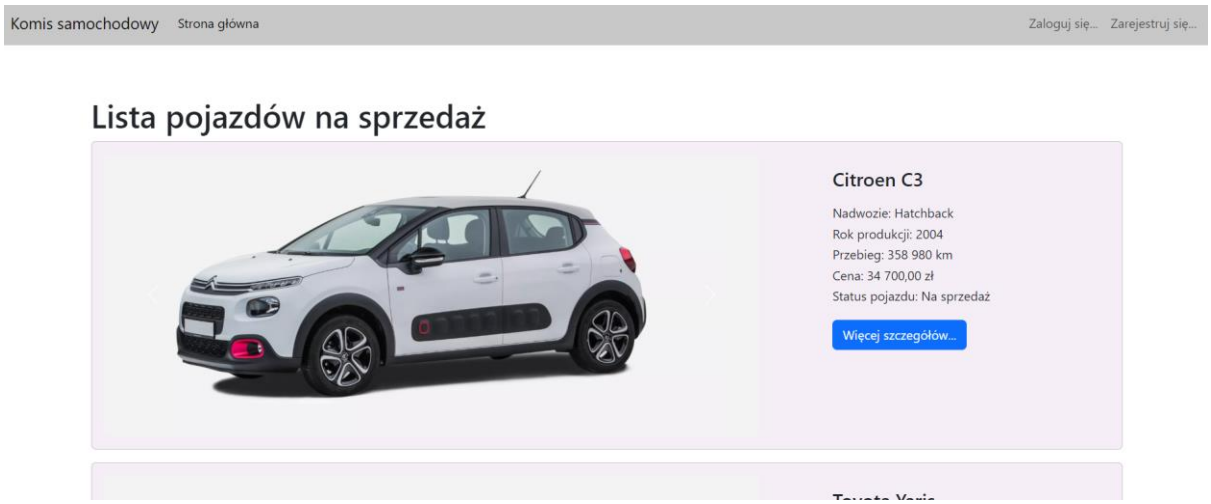
Metoda **register** sprawdza, czy użytkownik jest już zalogowany. Jeśli tak, przekierowuje go na stronę główną. W przeciwnym razie wyświetla formularz rejestracji.

Metoda **registerValidate** obsługuje proces rejestracji nowego użytkownika. Przyjmuje dane z formularza rejestracji, waliduje je, a następnie zapisuje dane użytkownika w bazie danych w ramach transakcji. Tworzy nowe rekordy w tabelach **dane**, **konta** i **klienci**. Jeśli rejestracja przebiegnie pomyślnie, użytkownik zostaje automatycznie zalogowany i przekierowany na stronę główną z komunikatem o pomyślnej rejestracji. W przypadku błędu, transakcja jest cofana, a użytkownik zostaje przekierowany z powrotem do formularza rejestracji z odpowiednim komunikatem o błędzie.

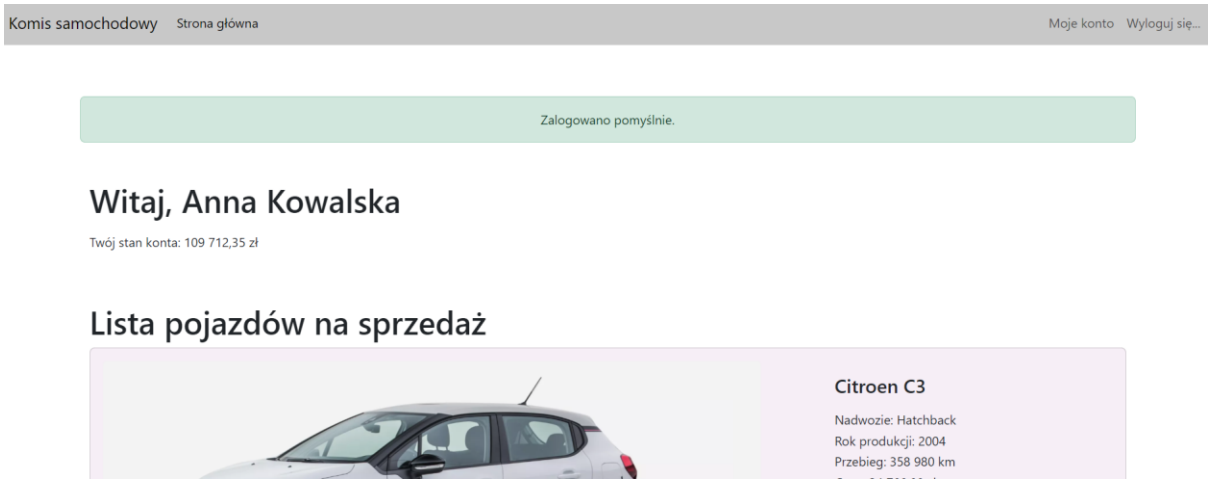
Metoda **edit** sprawdza, czy użytkownik jest zalogowany. Jeśli nie, przekierowuje go na stronę główną. Jeśli użytkownik jest zalogowany, pobiera jego dane oraz dane jego konta, a następnie wyświetla formularz edycji danych.

Metoda **editValidate** obsługuje proces aktualizacji danych użytkownika. Przyjmuje dane z formularza edycji, waliduje je, a następnie zapisuje zaktualizowane dane w bazie danych w ramach transakcji. Jeśli użytkownik wybrał opcję zmiany hasła, hasło jest również aktualizowane. Jeśli aktualizacja przebiegnie pomyślnie, użytkownik zostaje przekierowany na stronę swojego konta z komunikatem o pomyślnej aktualizacji. W przypadku błędu, transakcja jest cofana, a użytkownik zostaje przekierowany z powrotem do formularza edycji z odpowiednim komunikatem o błędzie.

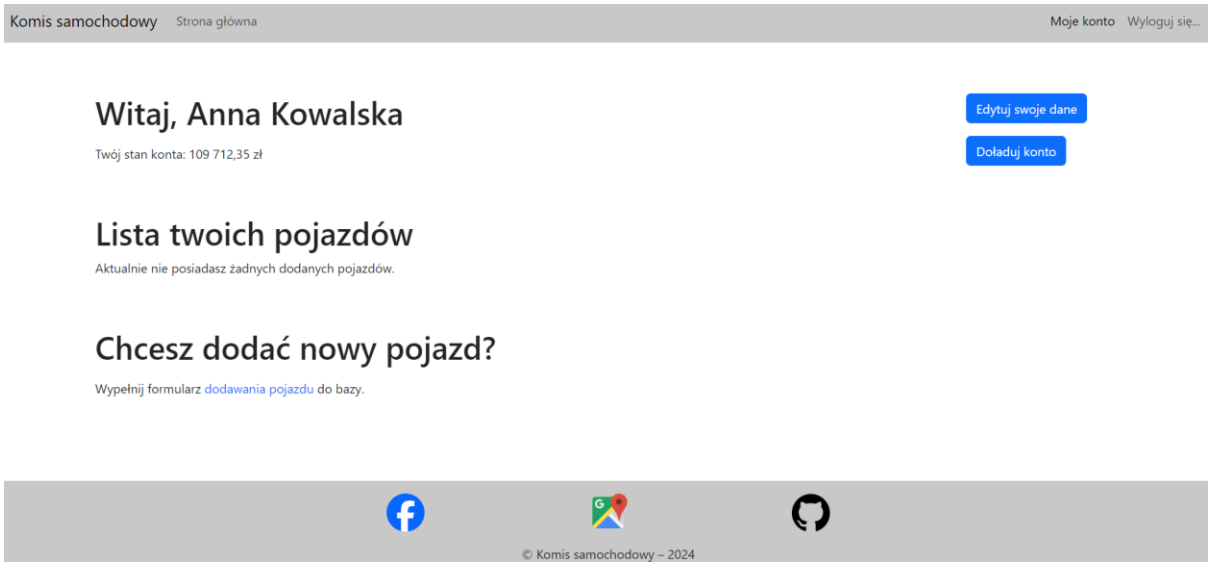
10. Testowanie i sprawdzenie poprawności działania aplikacji



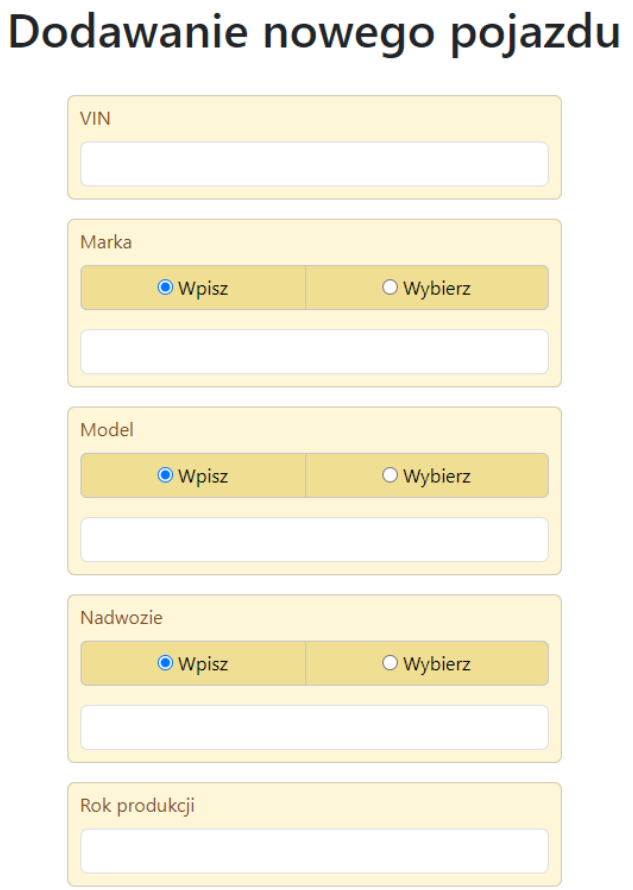
Niezalogowany użytkownik w górnej części ma pasek nawigacyjny oraz przyciski do rejestracji oraz logowania. Ma możliwość przeglądania ofert pojazdów wystawionych na sprzedaż.

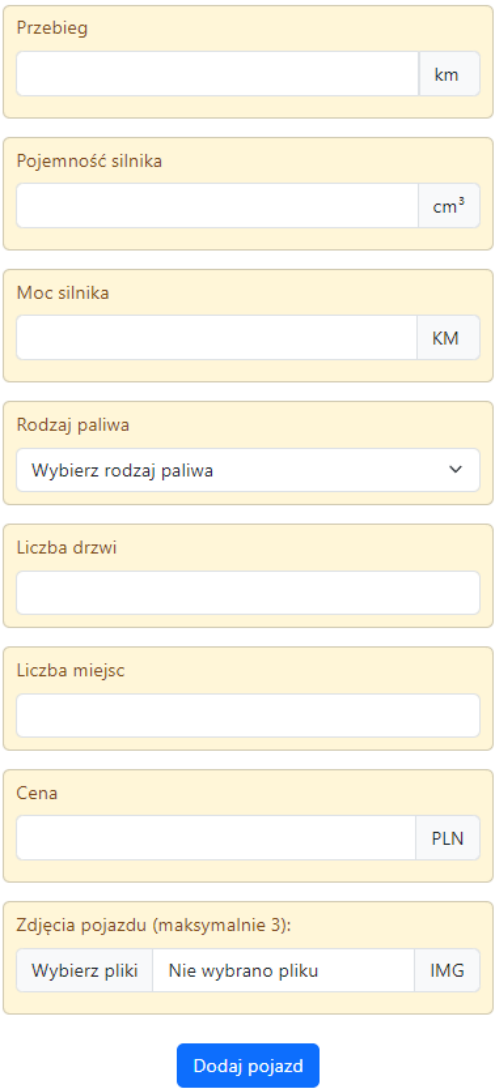


Zalogowany użytkownik w górnej części ma pasek nawigacyjny z odnośnikami do "Strona główna", "Moje konto" oraz "Wyloguj się". Na stronie wyświetlane jest powitanie zawierające imię i nazwisko użytkownika oraz informacja o stanie konta. Poniżej dostępna jest lista pojazdów wystawionych na sprzedaż, z możliwością przeglądania szczegółowych ofert.

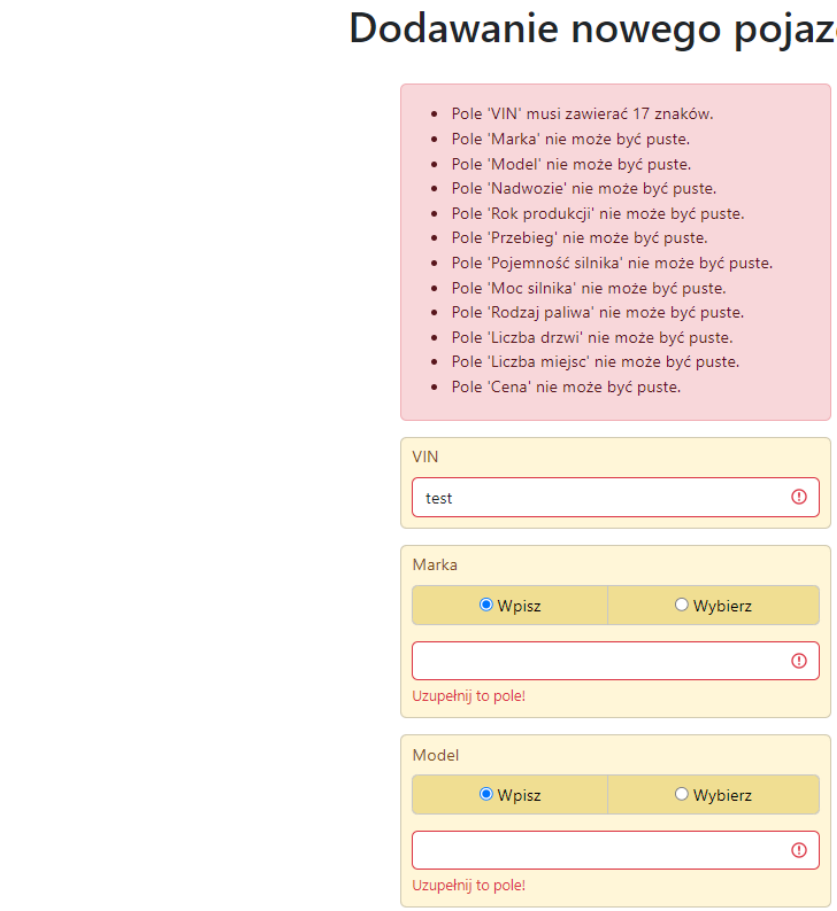


Zalogowany użytkownik w zakładce moje konto ma widoczny pasek nawigacyjny z odnośnikami do "Strona główna", "Moje konto" oraz "Wyloguj się". Wyświetlane jest powitanie z imieniem i nazwiskiem użytkownika oraz informacja o stanie konta. Na stronie dostępne są dwa przyciski: "Edytuj swoje dane" oraz "Doładuj konto". Poniżej znajduje się sekcja "Lista twoich pojazdów" z informacją, że aktualnie użytkownik nie posiada żadnych dodanych pojazdów. Na końcu strony znajduje się sekcja zachęcająca do dodania nowego pojazdu z linkiem do formularza dodawania pojazdu do bazy. Na dole strony znajdują się ikony prowadzące do mediów społecznościowych oraz informacje o prawach autorskich.

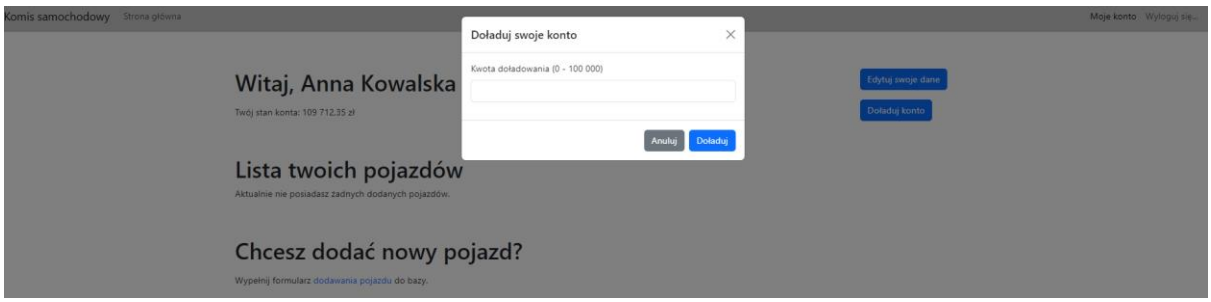




Formularz dodawania nowego pojazdu zawiera kilka pól do wypełnienia przez użytkownika. Na samej górze znajduje się pole do wpisania numeru VIN. Następnie są pola dotyczące marki, modelu, nadwozia, gdzie użytkownik może albo wpisać dane ręcznie, albo wybrać je z dostępnych opcji. Kolejne pola to rok produkcji, przebieg (wyrażony w kilometrach), pojemność silnika (w centymetrach sześciennych), moc silnika (w kilowatach) oraz rodzaj paliwa, który można wybrać z rozwijanej listy. Następnie użytkownik może wprowadzić liczbę drzwi i miejsc, cenę pojazdu (w złotych) oraz dodać zdjęcia pojazdu (maksymalnie trzy pliki). Na dole formularza znajduje się przycisk "Dodaj pojazd", który po kliknięciu przesyła wprowadzone dane do bazy danych. Całość formularza jest przejrzysta i ułatwia użytkownikowi dodanie nowego pojazdu do systemu.

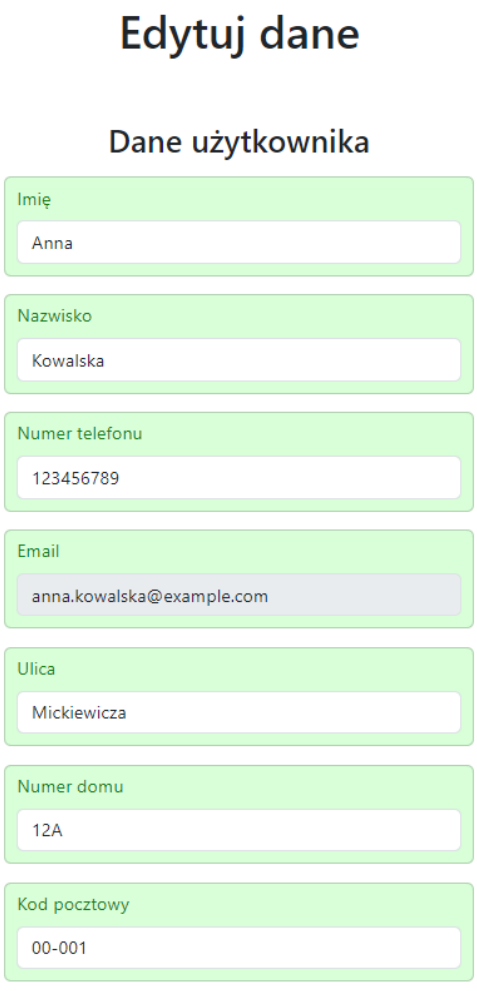


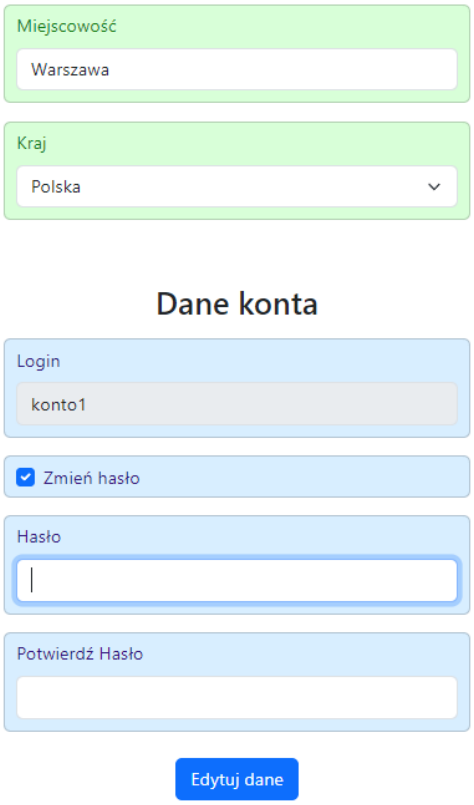
Formularz dodawania nowego pojazdu skutecznie waliduje dane wejściowe użytkownika. W przypadku wprowadzenia nieprawidłowych lub niekompletnych danych, formularz wyświetla komunikaty o błędach. Na samej górze formularza znajduje się czerwone pole z listą komunikatów, które wskazują, które pola są nieprawidłowo wypełnione lub pozostawione puste. Przykładowo, w przypadku pola "VIN" komunikat wskazuje, że musi ono zawierać 17 znaków. Ponadto, każde pole formularza, które zawiera błędne dane, jest podświetlone na czerwono i zawiera dodatkowy komunikat wskazujący, co należy poprawić. Przykłady obejmują pola takie jak "Marka" i "Model", które nie mogą być puste. System walidacji zapewnia, że użytkownik musi poprawić wszystkie błędy przed przesłaniem formularza, co zapobiega wprowadzeniu nieprawidłowych danych do bazy.



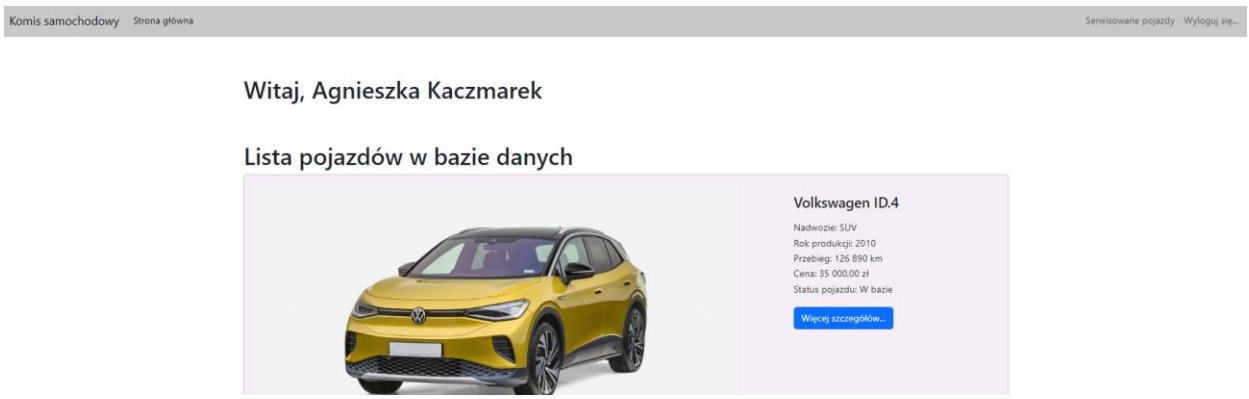
Po kliknięciu przycisku "Doładuj konto" pojawia się okno modalne z formularzem doładowania konta. Formularz zawiera jedno pole tekstowe, w którym użytkownik może wprowadzić kwotę doładowania w zakresie od 0 do 100 000 zł. Pod polem tekstowym znajdują się dwa przyciski: "Anuluj", który zamyka okno modalne bez dokonania zmian, oraz "Doładuj", który zatwierdza wprowadzone doładowanie i aktualizuje stan konta użytkownika.

Formularz doładowania konta skutecznie waliduje wprowadzane dane. W przypadku próby wprowadzenia kwoty doładowania większej niż dozwolony limit 100 000 zł, system wyświetla ostrzeżenie. Przykładowo, gdy użytkownik wprowadzi kwotę 200 000 000 zł, poniżej pola tekstowego pojawia się komunikat błędu z informacją "Wartość nie może być większa niż 100000". Formularz umożliwia anulowanie operacji za pomocą przycisku "Anuluj" lub potwierdzenie doładowania poprawną kwotą poprzez przycisk "Doładuj". Ta walidacja zapewnia, że użytkownik nie może wprowadzić nieprawidłowych danych, co zwiększa bezpieczeństwo i integralność procesu doładowania konta.

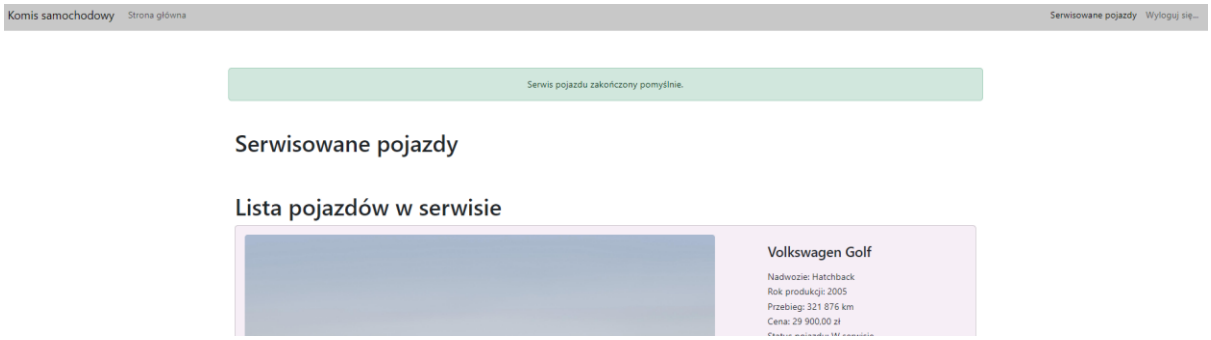


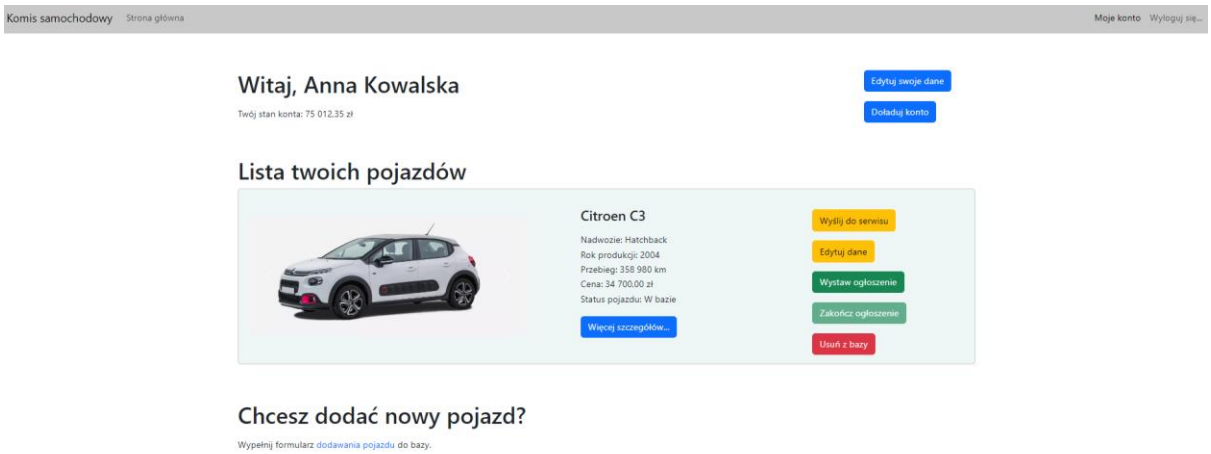


Formularz edycji danych użytkownika składa się z dwóch głównych sekcji: "Dane użytkownika" i "Dane konta". W sekcji "Dane użytkownika" użytkownik może edytować swoje imię, nazwisko, numer telefonu, adres email, ulicę, numer domu, kod pocztowy, miejscowość oraz kraj. Wszystkie te pola są obowiązkowe i mają zielone tło. W sekcji "Dane konta" znajduje się pole z loginem użytkownika oraz opcja zmiany hasła. Aby zmienić hasło, użytkownik musi zaznaczyć pole "Zmień hasło" i wprowadzić nowe hasło w dwóch polach: "Hasło" i "Potwierdź Hasło". Na dole formularza znajduje się przycisk "Edytuj dane", który po kliknięciu zapisuje wprowadzone zmiany w systemie. Formularz jest przejrzysty i intuicyjny, co ułatwia użytkownikowi aktualizację swoich danych osobowych i danych konta.

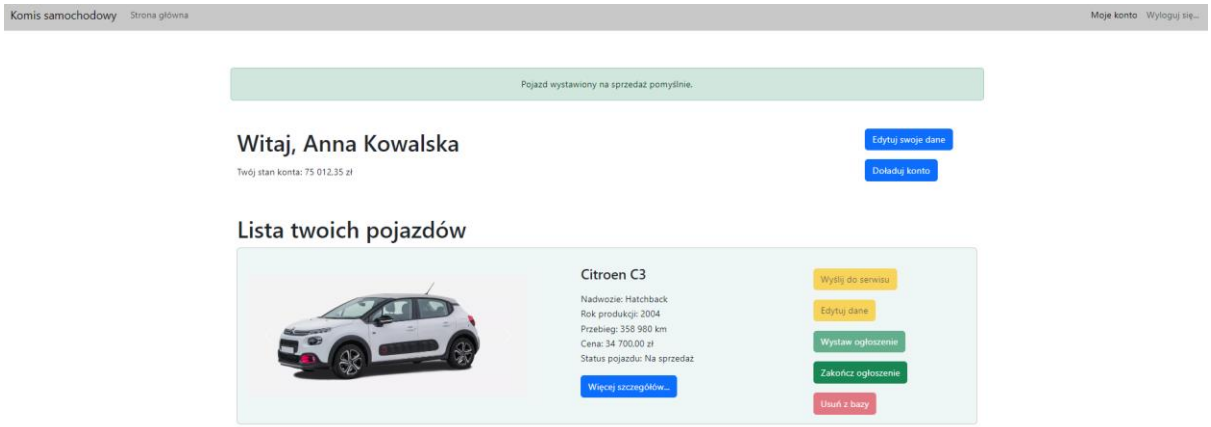


Po zalogowaniu pracownika, strona wyświetla sekcję powitalną z imieniem i nazwiskiem użytkownika oraz tytułem "Lista pojazdów w bazie danych". W górnej części znajduje się pasek nawigacyjny z odnośnikami do "Strona główna", "Serwisowane pojazdy" oraz "Wyloguj się". Lista pojazdów przedstawia szczegółowe informacje o dostępnych samochodach, takie jak nadwozie, rok produkcji, przebieg, cena oraz status pojazdu.

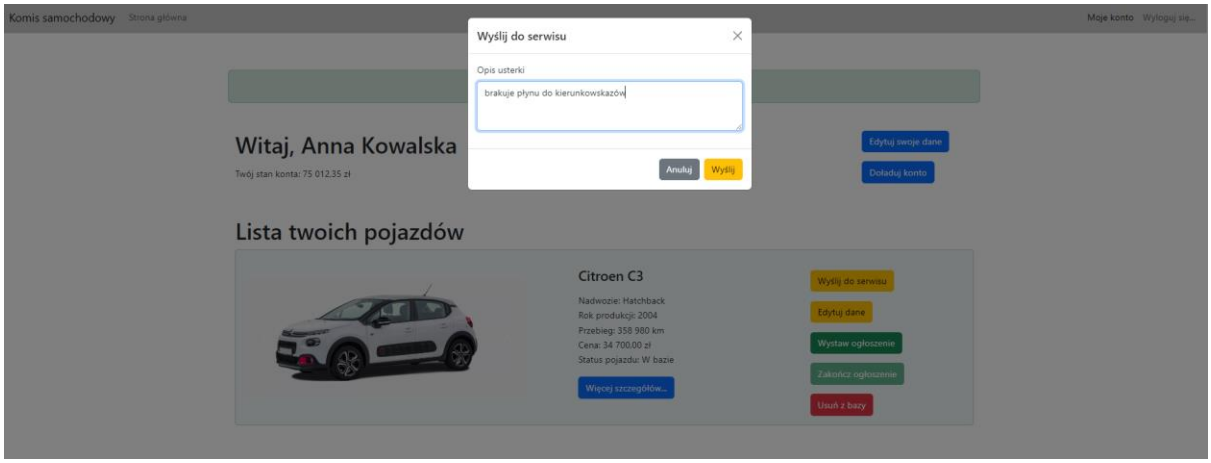




Po zakupie pojazdu klient ma możliwość wykonania kilku operacji. Może wysłać pojazd do serwisu, edytować dane pojazdu, wystawić ogłoszenie sprzedaży, zakończyć ogłoszenie oraz usunąć pojazd z bazy. Każda z tych opcji jest dostępna za pomocą odpowiednich przycisków obok szczegółowych informacji o pojeździe.



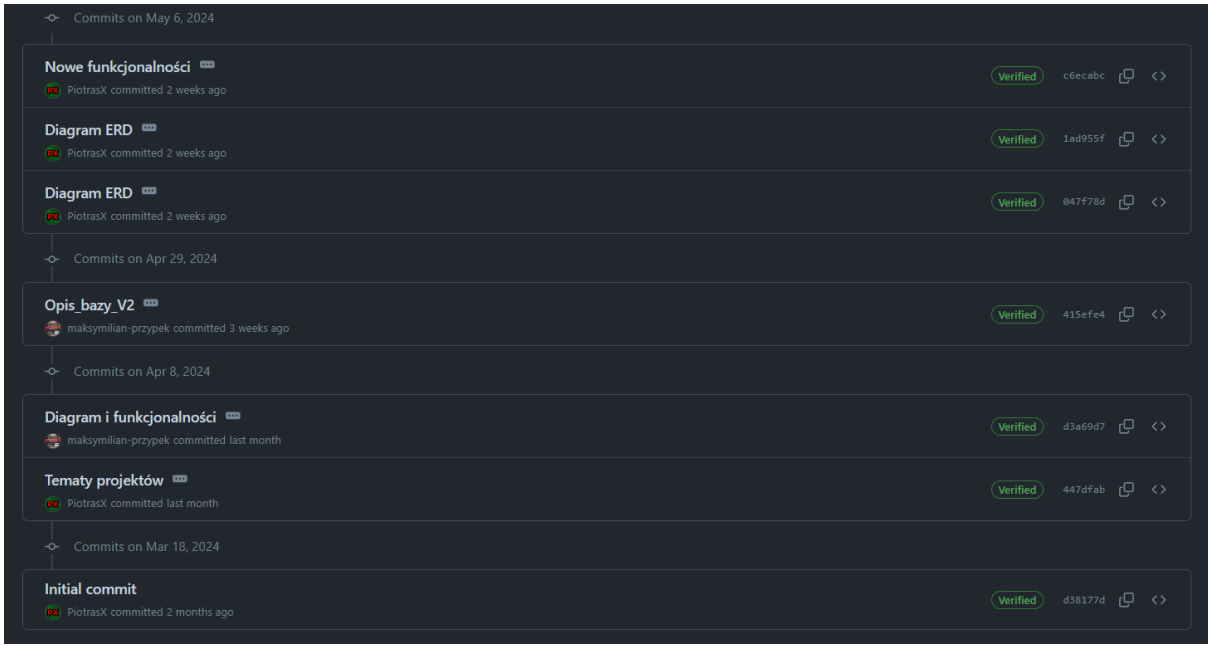
Po zakupie pojazdu klient ma możliwość wystawienia go ponownie na sprzedaż. W tym celu na stronie z listą pojazdów klienta dostępny jest przycisk "Wystaw ogłoszenie". Dodatkowo, klient może wysłać pojazd do serwisu, edytować dane pojazdu, zakończyć ogłoszenie oraz usunąć pojazd z bazy. Przy każdej operacji, w górnej części strony pojawia się zielony komunikat potwierdzający wykonanie danej akcji, na przykład "Pojazd wystawiony na sprzedaż pomyślnie". Strona zapewnia użytkownikowi łatwe zarządzanie pojazdem i jego statusami poprzez intuicyjny interfejs.



Po zakupie pojazdu klient ma możliwość wysłania go do serwisu. Po kliknięciu przycisku "Wyślij do serwisu" otwiera się okno modalne, w którym użytkownik może wprowadzić opis usterki. Przykładowo, użytkownik może wpisać informację "brakuje płynu do kierunkowskazów". Po wprowadzeniu opisu, użytkownik może zatwierdzić zgłoszenie, klikając przycisk "Wyślij" lub anulować operację, klikając "Anuluj". Ten proces umożliwia klientowi łatwe i szybkie zgłaszanie problemów technicznych z pojazdem bezpośrednio do serwisu.

11. Zarządzanie projektem

11.1. System kontroli wersji (Git)



12. Należy wykonać tylko MainSeeder.php aby baza danych poprawnie została zaimplementowana

12.1. Dane logowania dla Pracownika:

Konto 1:
Login: konto5
Hasło: haslo5

Konto 2:
Login: konto6
Hasło: haslo6

Konto 3:
Login: konto12
Hasło: haslo12

Konto 4:
Login: konto14
Hasło: haslo14

12.2. Dane logowania dla Klienta:

Konto 1:
Login: konto1
Hasło: haslo1

Konto 2:
Login: konto2
Hasło: haslo2

Konto 3:
Login: konto3
Hasło: haslo3

Konto 4:
Login: konto4
Hasło: haslo4

Konto 5:
Login: konto7
Hasło: haslo7

Konto 6:

Login: konto8
Hasło: haslo8

Konto 7:
Login: konto9
Hasło: haslo9

Konto 8:
Login: konto10
Hasło: haslo10

Konto 9:
Login: konto11
Hasło: haslo11

Konto 10:
Login: konto13
Hasło: haslo13