Sprawozdanie

Kurs: Tworzenie aplikacji bazodanowych

Temat: System informatyczny wspomagający obsługę floty przedsiębiorstwa

Data: 24.06.2025

Autorzy: Krzysztof Klecha (lider sekcji)
Przemysław Dąbrowski
Paweł Dziergas
Piotr Mastalerz

Spis treści

Zawartość

1. Treść zadania

Celem projektu było stworzenie aplikacji wspomagającej zarządzanie flotą pojazdów w firmie.

W skład systemu miały wchodzić:

- relacyjna baza danych przechowująca informacje obecne w systemie
- backend, pracujący nieustannie na serwerze, realizujący logikę aplikacji oraz udostępniający REST API
- aplikacja kliencka umożliwiająca interakcję z systemem

2. Analityka

2.1. Wymagania funkcjonalne

Aplikacja powinna umożliwiać wykonywanie następujących czynności:

- Przechowywanie informacji o pracownikach firmy,
- Przechowywanie informacji o posiadanych samochodach,
- Zapisywanie informacji o kosztach utrzymania pojazdów takich jak tankowanie czy serwis,
- Dokumentacja przeglądów technicznych pojazdów,
- Rezerwowanie pojazdów dla pracowników na użytek służbowy lub prywatny,
- Przeglądanie historii zdarzeń takich jak rezerwacje, przeglądy czy serwisy.

2.2. Wymagania niefunkcjonalne

Poza wyżej zdefiniowanymi wymaganiami funkcjonalnymi, system powinien również spełniać wymagania niefunkcjonalne:

- Posiadać responsywny interfejs użytkownika,
- Zapewniać bezpieczeństwo dostępu do danych (autoryzacja użytkowników),
- Przejrzysta struktura kodu i dokumentacja API,
- Modułowość komponentów aplikacji.

2.3. Wybrane technologie

Na podstawie analizy wymagań funkcjonalnych i niefunkcjonalnych podjęto decyzję o wyborze nowoczesnych, popularnych i dobrze wspieranych technologii, za pomocą których zamierzaliśmy realizować projekt.

2.3.1. Technologia bazy danych

MariaDB

Powody:

- Dobra skalowalność,
- MariaDB jest open-source i posiada duże wsparcie społeczności,
- pełne wsparcie standardu SQL,
- dobra integracja z entity framework.

2.3.2. Technologia Backendu

.NET, Swagger, Entity Framework Core

Powody:

- entity framework core, jako ORM umożliwia dostęp do bazy danych
- Swagger umożliwia automatyczne generowanie dokumentacji API oraz ułatwia testowanie

2.3.3. Technologia Frontendu

React, Typescript

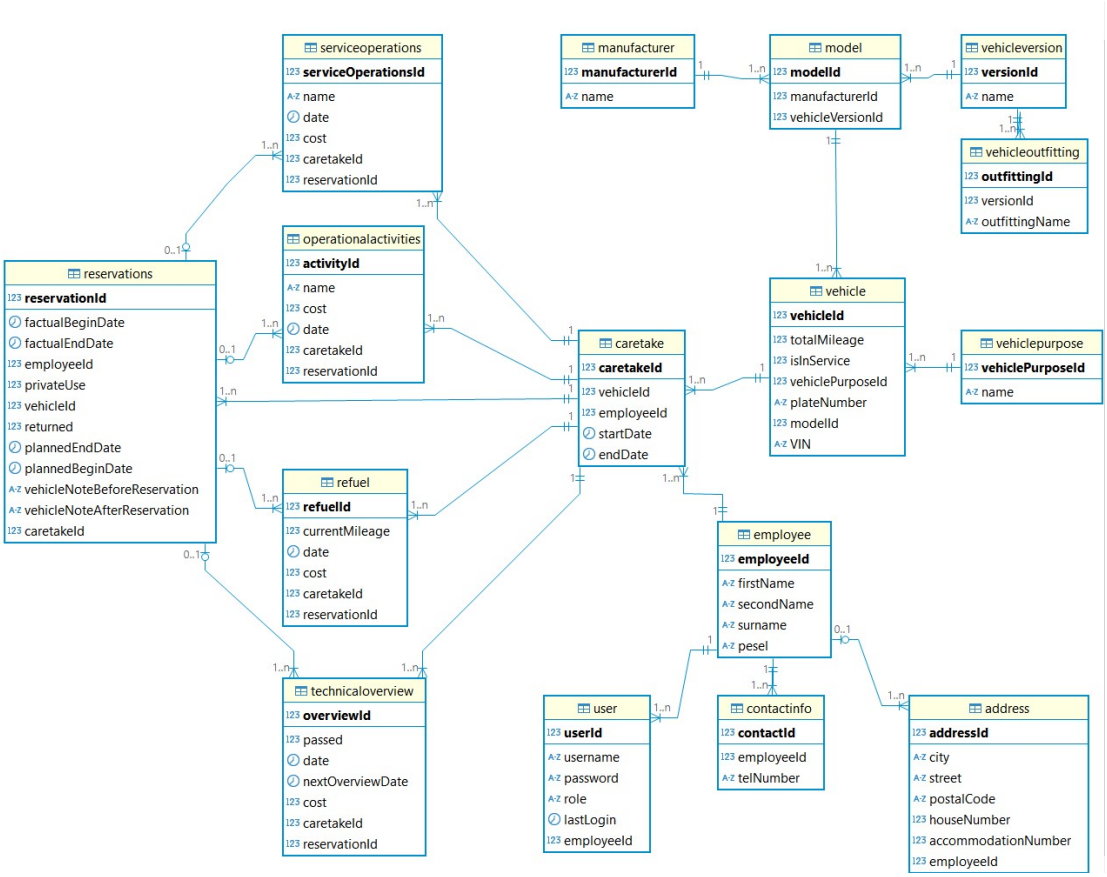
Powody:

- doskonała współpraca z REST API,
- popularność, skutkująca dobrą dostępnością materiałów i wzorów,
- możliwość dostępu z przeglądarki.

3. Architektura systemu

3.1. Baza danych

System wykorzystuje relacyjną bazę danych SQL (Maria DB).



Rys. 1 – schemat bazy danych

Taka struktura bazy danych umożliwia:

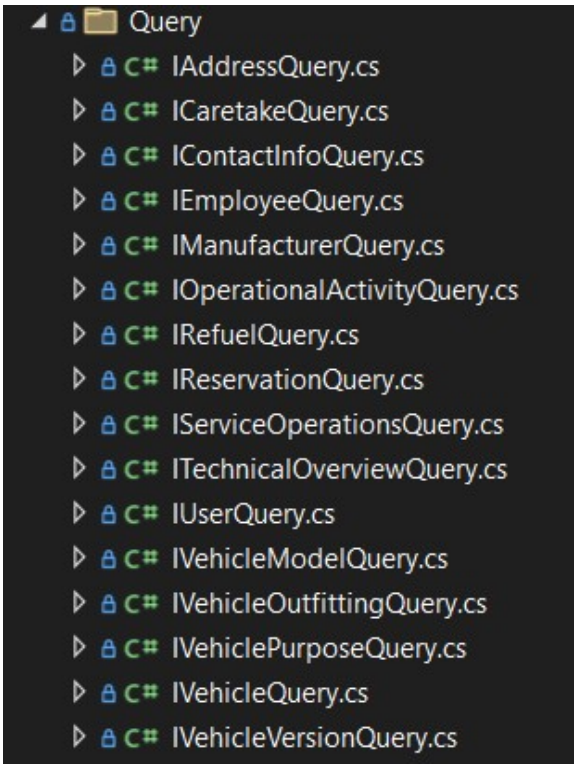
- śledzenie, który pracownik jest odpowiedzialny za którą czynność,
- przeglądanie historii operacji,
- minimalizacje rozmiaru danych na dysku.

3.2. Backend

Część serwerowa systemu została zaimplementowana w technologii .NET. Backend odpowiada za logikę biznesową, komunikację z bazą danych oraz udostępnienie interfejsu API dla aplikacji frontendowej.

Centralny plik konfiguracyjny Program.cs zawiera definicje usług oraz konfigurację połączenia z bazą danych przy pomocy Entity Framework Core i providera Pomelo.EntityFrameworkCore.MySql (dla MariaDB). Aplikacja obsługuje także Swaggera, który automatycznie generuje dokumentację dostępnych endpointów REST API.

Wszystkie zapytania do bazy danych realizowane są przez odpowiednio wydzielone klasy implementujące interfejsy IQuery. Przykładowe interfejsy to IRefuelQuery, IVehicleQuery, IServiceOperationsQuery i inne.

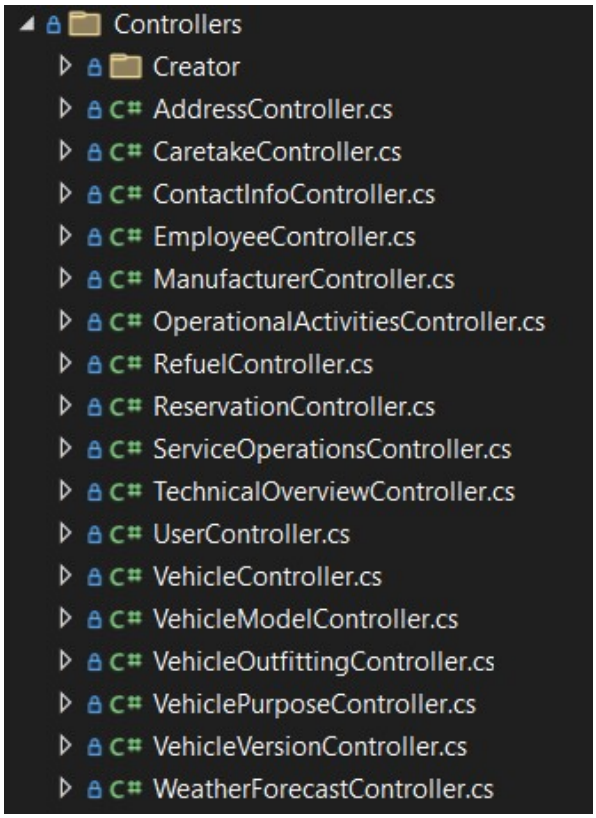


Rys. 2 - interfejsy

Backend obsługuje różne typy operacji:

- GET – do pobierania danych (np. historii serwisu),
- PUT – do zapisu nowych danych (np. zapis tankowania),
- POST – do aktualizacji już istniejących danych (np. zapis daty końca rezerwacji),
- DELETE – do usuwania wpisów.

Struktura kontrolerów (np. RefuelController.cs) zapewnia przejrzystość kodu oraz umożliwia łatwą rozbudowę systemu. Dzięki podzieleniu na fragmenty, komponenty są osobno testowalne i łatwe do utrzymania.



Rys. 3 - kontrolery

3.3. Frontend

Aplikacja frontendu została utworzona przy użyciu biblioteki React z wykorzystaniem TypeScript dla zwiększenia wygody oraz czytelności kodu. Za przekierowywanie na strony zawierające konkretne komponenty aplikacji odpowiada „router” z biblioteki react.

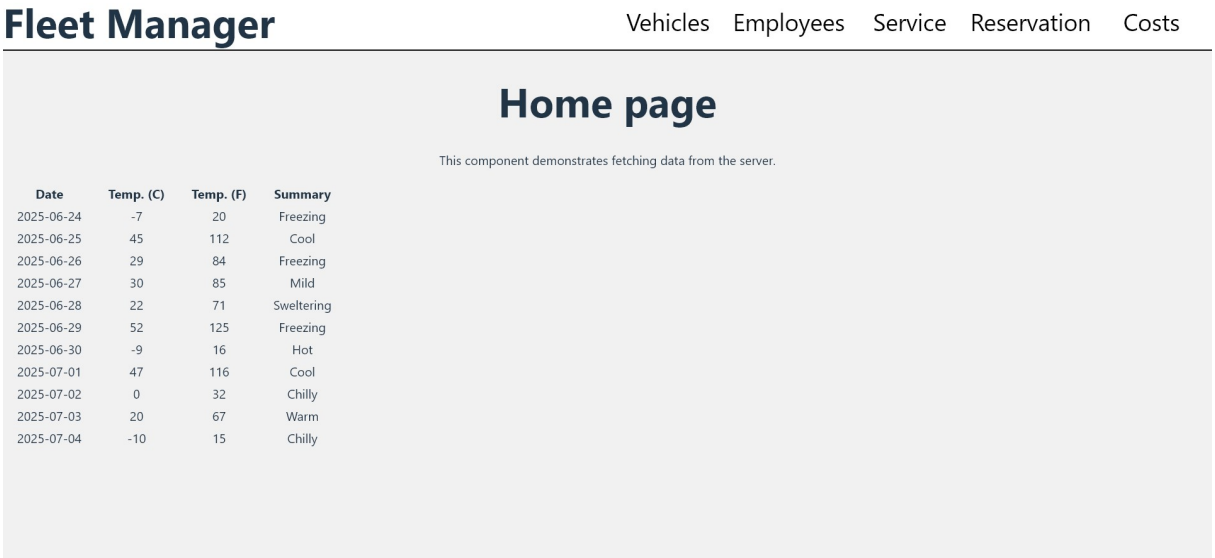
Plik App.tsx definiuje strukturę aplikacji. Interfejs jako pierwszą część zawiera zawsze pasek nawigacyjny „Navbar” na górze strony. Pozwala on na przenoszenie się między poszczególnymi kartami. Jako drugą część zawiera jedną ze stron wybranych przez użytkownika.

Rys. 4 – „Navbar”

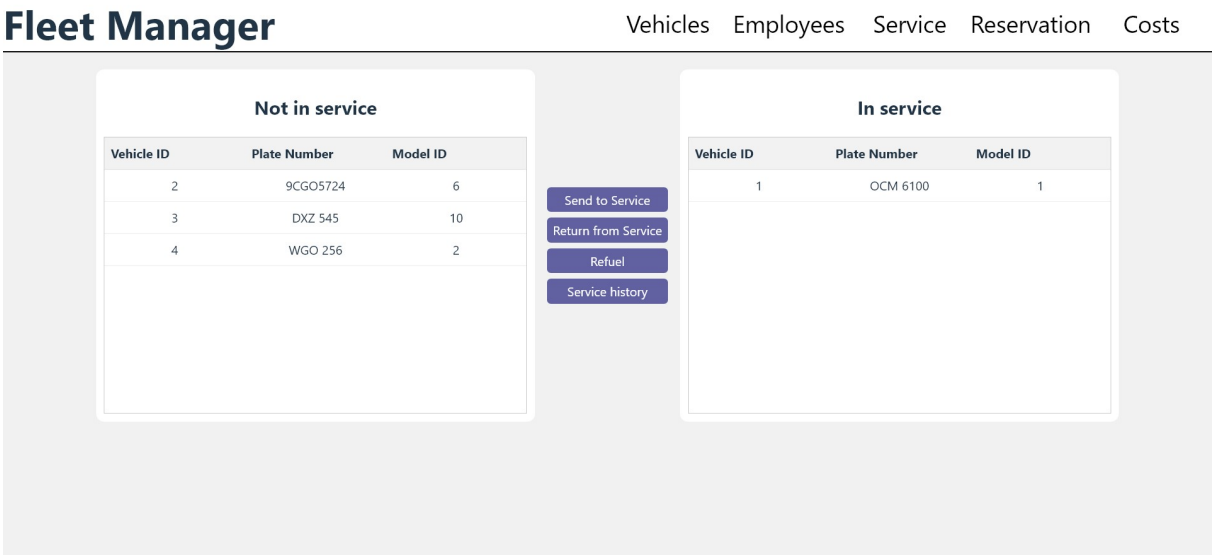
Użytkownik, poprzez widoczny powyżej pasek nawigacyjny może wybrać jedną z pięciu stron:

- Vehicles
- Employees
- Service
- Reservation
- Costs

Dwie przykładowe strony:



Rys. 5 – strona domyślna „Home”



Rys. 6 – strona obsługująca serwisy pojazdów „Service”

3.4. Komunikacja

Frontend i backend komunikują się poprzez protokół HTTP/HTTPS, wykorzystując REST API. Przykładowo, komponent Service.tsx umożliwia użytkownikowi zarejestrowanie tankowania pojazdu. W tym celu użytkownik wybiera pojazd z listy i wprowadza dane tankowania w formularzu. Dane są następnie przesyłane do backendu za pomocą zapytania PUT na endpoint /api/refuel/create.

Rysunki 7 i 8 pokazują przykład komunikacji frontendu z backendem.

```
37      [HttpPut("create")]
38      public async Task<IActionResult> CreateRefuelAsync(Refuel model)
39      {
40          try
41          {
42              await query.CreateRefuelAsync(model);
43              return Ok();
44          }
45          catch (Exception ex)
46          {
47              var msg = exCreator.ConstructErrorMessage("put", model, ex);
48              return StatusCode(500, msg);
49              throw;
50          }
51      }
```

Rys 7. – fragment pliku „RefuelController.cs”

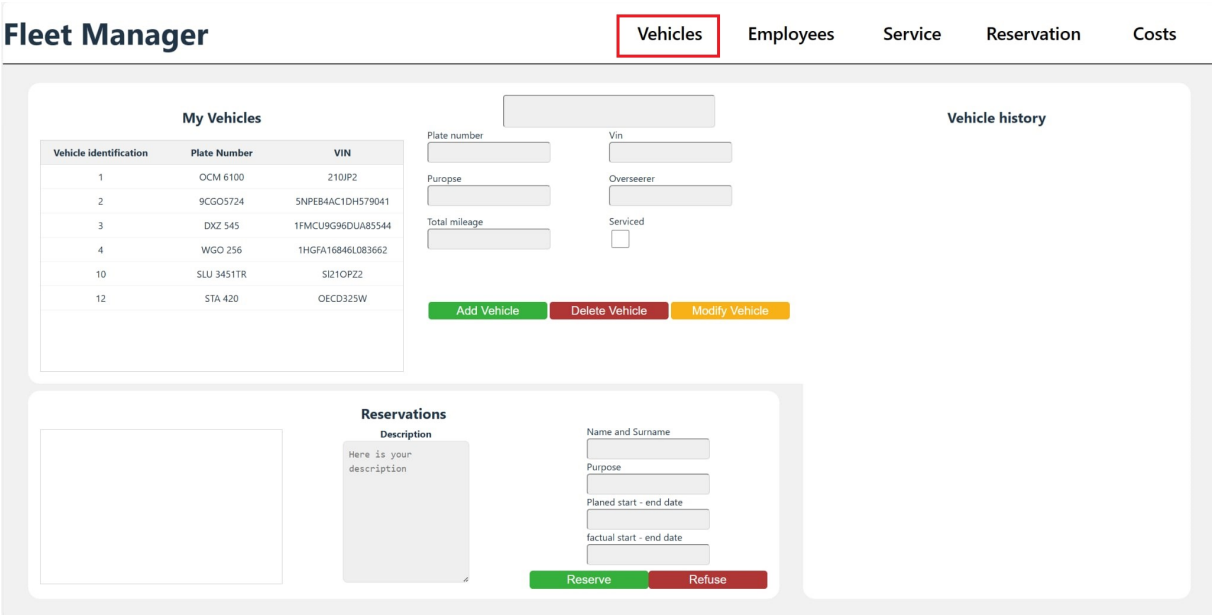
```
147         await fetch(`/api/refuel/create`, {
148             method: 'PUT',
149             headers: { 'Content-Type': 'application/json' },
150             body: JSON.stringify(payload)
151         });
```

Rys. 8 – fragment pliku „Service.tsx”

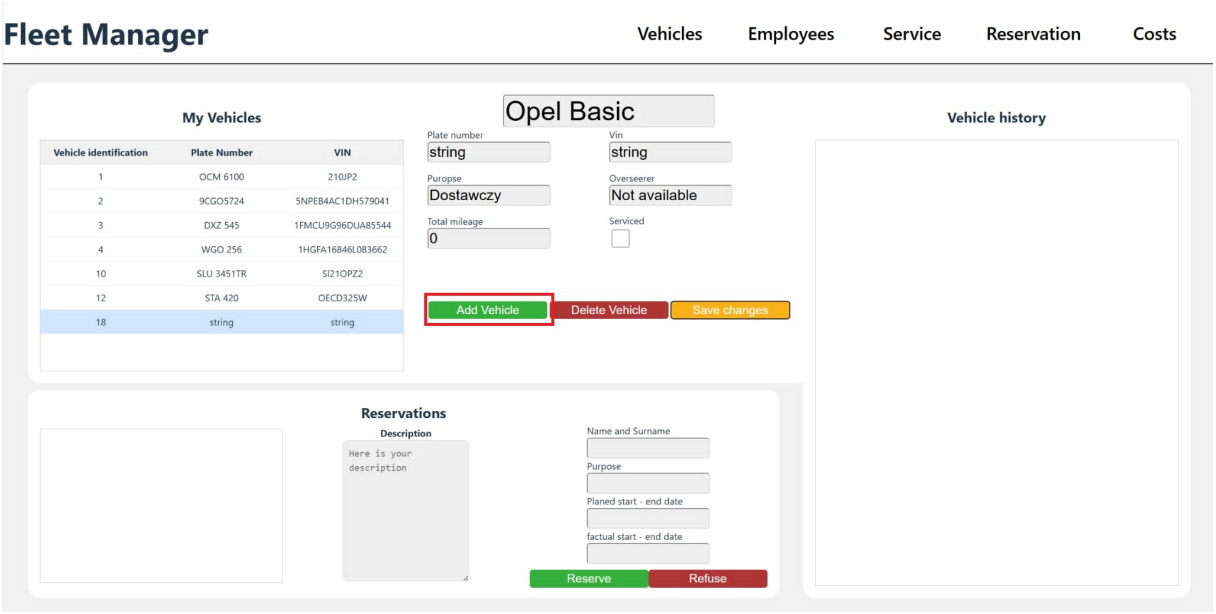
4. Instrukcja dla użytkownika - (Use Cases)

4.1. Dodawanie pojazdu

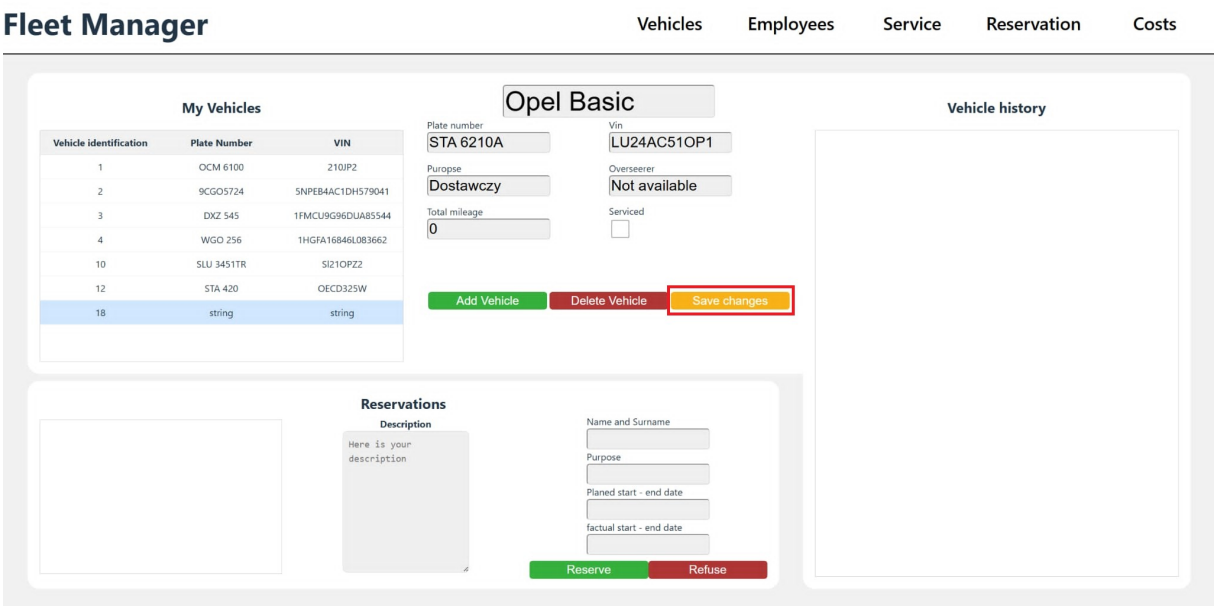
1. Użytkownik przechodzi do zakładki „Vehicles”



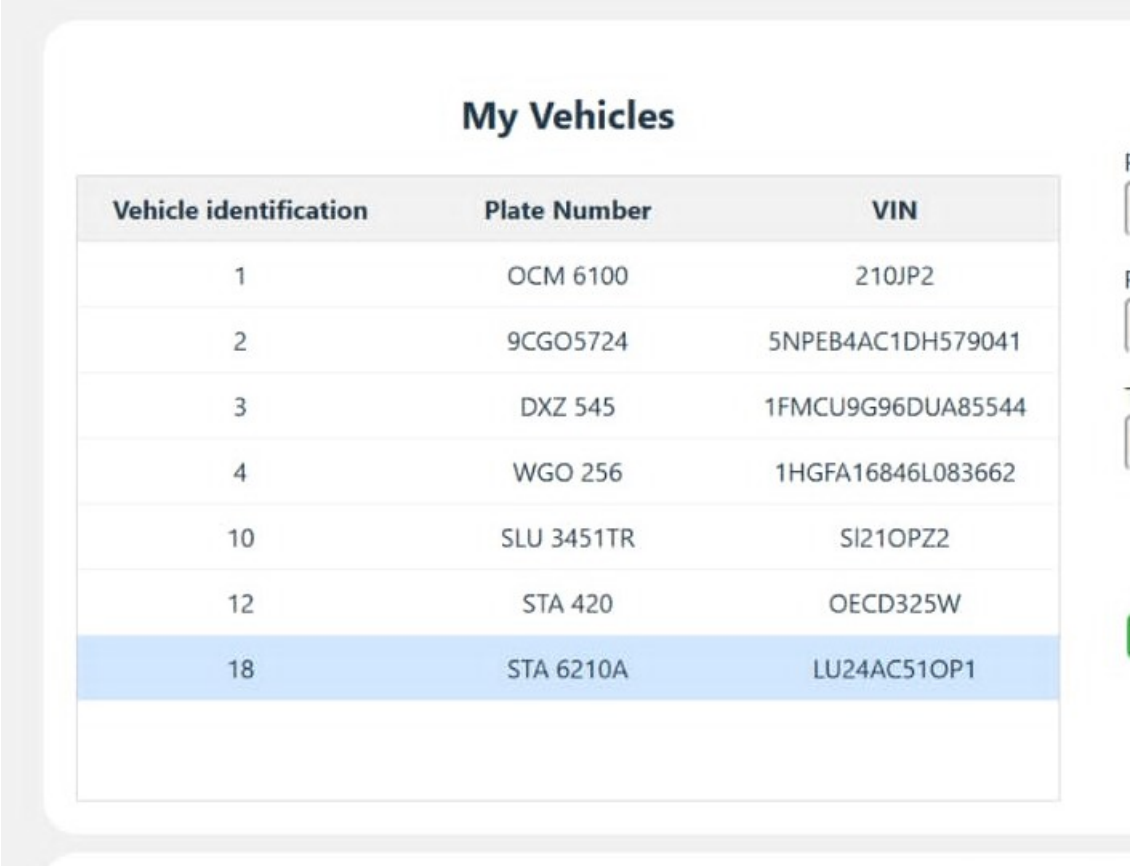
2. Użytkownik klika „Add Vehicle”



3. Użytkownik edytuje potrzebne pola, takie jak rejestracja i numer VIN, po czym klika „save changes”



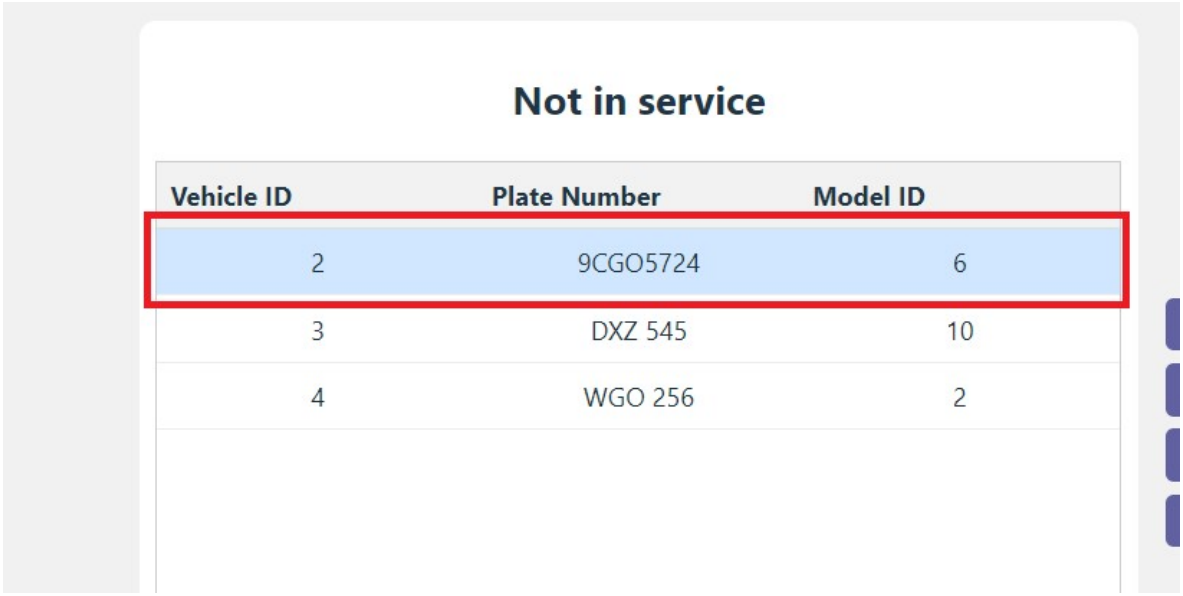
4. Pojazd jest dodany i zapisany w odpowiedniej tabeli



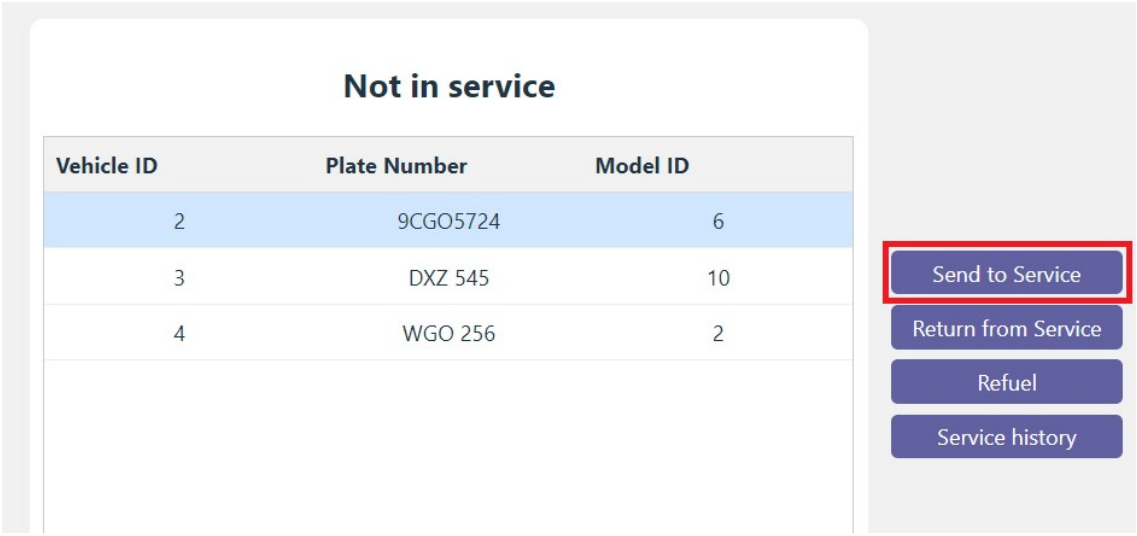
4.2. Dodawanie serwisu pojazdu

1. Użytkownik przechodzi do zakładki „Service”

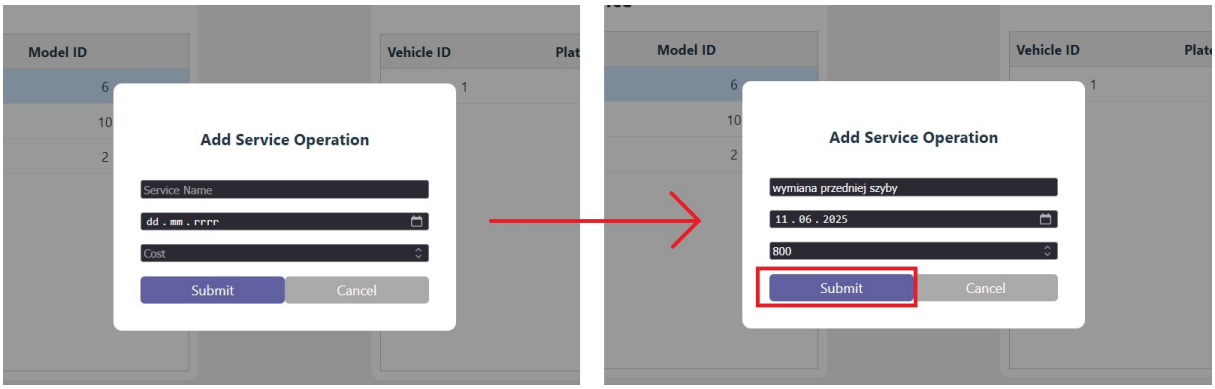
2. Zaznacza pojazd, który chce oddać do serwisu



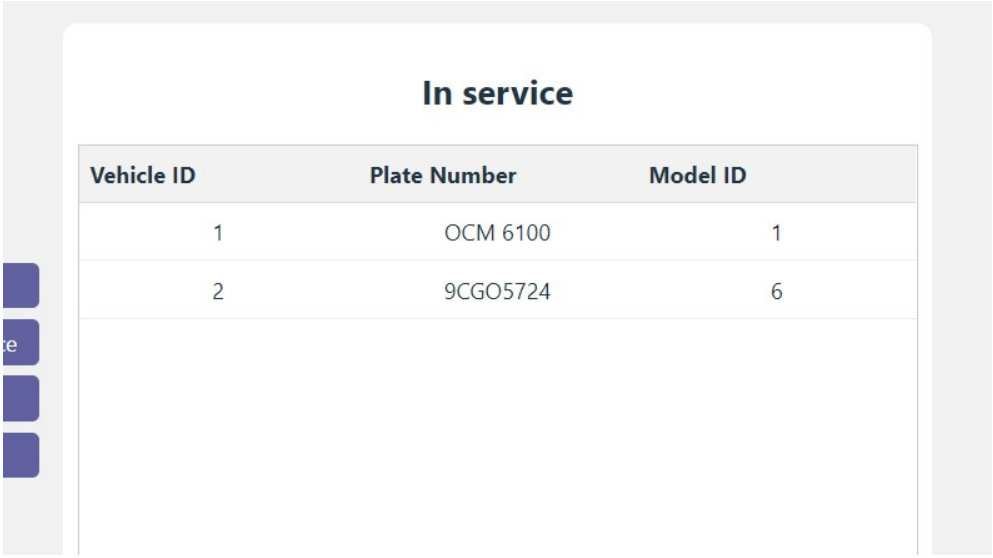
3. Klika przycisk „send to service”



4. Wypełnia dane w wyświetlonym okienku i naciska przycisk „submit”



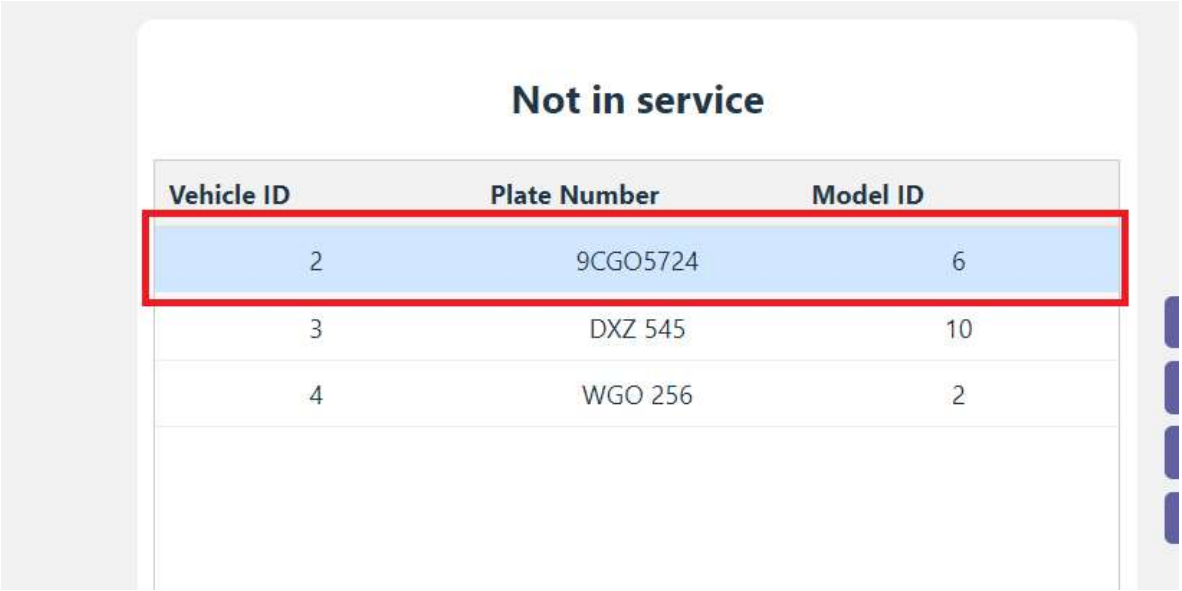
5. Pojazd pojawia się w tabeli „In service”



4.3. Wyświetlanie historii serwisowej pojazdu

1. Użytkownik przechodzi do zakładki „Service”

2. Zaznacza pojazd, którego historię serwisu chce zobaczyć



3. Klika przycisk „service history”

4. Wyświetla się historia czynności serwisowych wykonanych dla wybranego pojazdu

4.4. Sortowanie chronologiczne

Warto wspomnieć, że w wielu miejscach stosowana jest tabela „sortable table”. Umożliwia to sortowanie po dowolnej kolumnie. Najważniejszą z kolumn, po których można sortować jest data – kliknięcie „date” w dowolnej tabeli poskutkuje posortowaniem chronologicznym.

Service history of vehicle 1

service ID	name of service	date ▲	cost	caretakeId
1	wymiana rozrządu na zamiennik	2025-01-01	1000	1
8	regeneracja rozrządu	2025-02-05	1500	1
2	naprawa koła przedniego	2025-06-06	2000	1
5	wymiana rozrządu na oryginalnych częściach	2025-06-20	5680	1

5. Wnioski

Projekt pozwolił na praktyczne wykorzystanie wiedzy z zakresu tworzenia aplikacji bazodanowych. Stworzenie kompletnej aplikacji umożliwiło zapoznanie się z:

- pełnym cyklem projektowania aplikacji (od analizy do wdrożenia),
- integracją różnych technologii (np. .NET, SQL, React),
- organizacją warstwowej architektury,
- obsługą API i dokumentowaniem go z użyciem Swaggera,
- dobrymi praktykami projektowania baz danych i interfejsów użytkownika.

Projekt realizuje założony cel - stworzenie aplikacji wspierającej zarządzanie flotą pojazdów w firmie. Dzięki wykorzystaniu ostrożnie wybranych technologii system jest:

- Modułowy – łatwo rozwijać o nowe funkcjonalności,
- Przenośny – działa zarówno lokalnie, jak i na serwerach produkcyjnych,
- Czytelny zarówno pod względem kodu, jak i struktury API,
- przyjazny dla użytkownika i intuicyjny

Zaprojektowana architektura umożliwia dalszą rozbudowę, na przykład dodanie możliwości zarządzania całą siecią placówek, a nie tylko jedną.