System Zarządzania i Ewidencji Warsztatu

Zaimplementowana część aplikacji

Zaimplementowano cały planowany system z wyjątkiem funkcjonalności generowania rozliczeń okresowych przez administratora.

Zmiany w stosunku do założeń projektowych

- Administrator posiada wszystkie uprawnienia mechanika oraz część własnych uprawnień. Zmiana ta jest wynikiem dostosowania programu do potrzeb bardzo małych firm.

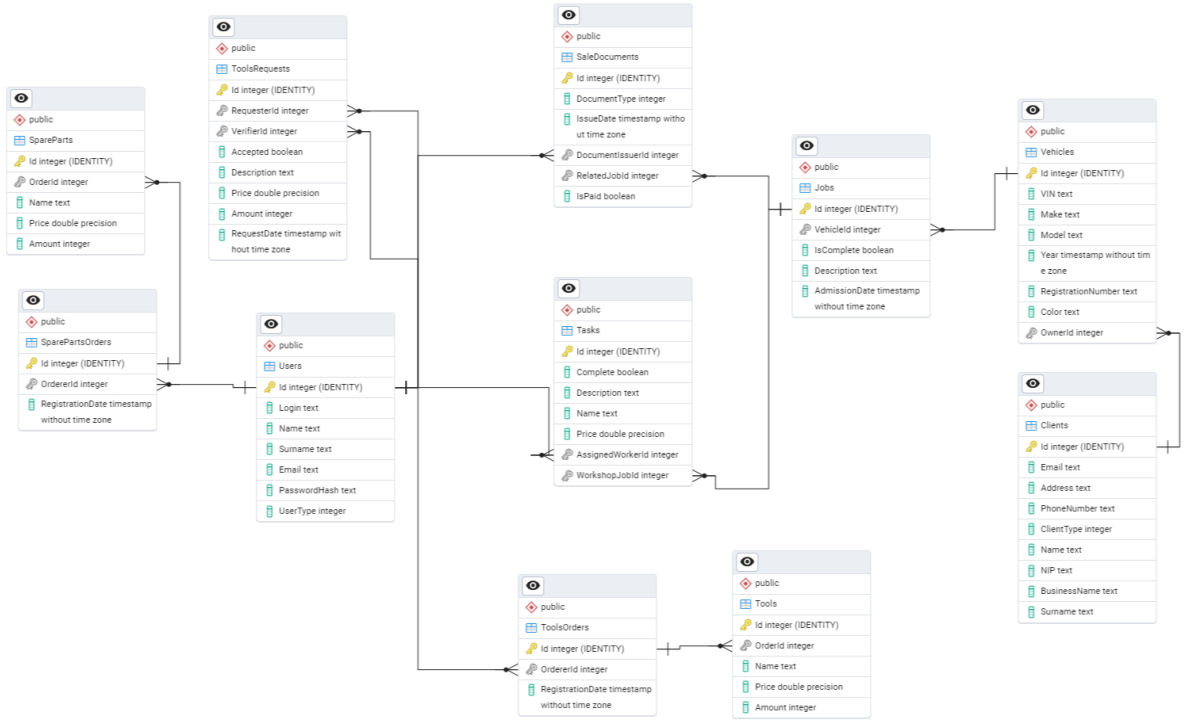
- „Resetuj hasło” zostało zamienione na „Zmień hasło”, ponieważ ta opcja jest bardziej uniwersalna i pozwala administratorom lepiej zarządzać użytkownikami.

- Brak wylogowania się, ponieważ system uwierzytelniania oparty na JWT nie wymaga funkcji wylogowywania, co zdecydowano się pominąć.

- Dokument sprzedaży jest połączony z całym zleceniem, a nie z usługami, ponieważ takie podejście wydaje się bardziej odpowiednie w dłuższej perspektywie.

- Pominięcie poszczególnych metod typu „SprawdźCzyOpłacony” na rzecz implementacji metod zwracających dane o obiekcie z uwzględnieniem szczegółowych DTO, ponieważ te metody już dostarczają odpowiednie dane, a DTO zapewniają kontrolę nad tym, aby nie ujawniać zbyt wielu lub wrażliwych informacji, jak hashe haseł.

Schemat zaimplementowanej Bazy Danych



Specyfikacja struktury aplikacji

Aplikacja została stworzona w oparciu o framework .NET (Web Core) w wersji 9.0 i jest kompilowana do formatu binarnego. W zależności od środowiska uruchomieniowego, może być kontenerowana za pomocą technologii Docker. W celu ułatwienia jej uruchomienia w tej formie, dołączony został plik compose, który umożliwia powtarzalne uruchamianie aplikacji oraz zapewnia dostęp do zmiennych środowiskowych.

Aplikacja została podzielona na dwie główne części: backend oraz frontend. Struktura backendu jest podzielona na szereg katalogów. W folderze głównym znajdują się pliki, które opisują strukturę projektu z perspektywy programistycznej, zmienne uruchomieniowe projektu, plik compose, katalog z skompilowanymi wersjami aplikacji oraz główny katalog projektu „SZEW”.

W katalogu głównym umieszczono również folder certs, zawierający certyfikaty poświadczeń, a także całą strukturę kodu, podzieloną na poszczególne katalogi:

- Controllers – kontrolery przetwarzania zapytań API
- Data – powiązanie danych z bazy z obiektami.

- DTO – pliki opisujące sposób przekazywania danych do obiektu
- Helper- plik opisujący mapowania pomiędzy poszczególnymi klasami
- Interfaces – interfejsy dla funkcji poszczególnych klas
- Migrations – automatycznie generowane struktury migracji bazy danych
- Models – poszczególne klasy występujące w systemie
- Repository – implementacje interfejsów poszczególnych klas

Działanie SI

Komunikacja pomiędzy frontendem a backendem realizowana jest za pomocą REST API, a dane wymieniane są przy użyciu formatu JSON. Proces autentykacji użytkownika polega na podaniu loginu oraz hasła, które przesyłane są w bezpiecznym połączeniu SSL. Hasło jest następnie porównywane z jego zahashowaną wersją przechowywaną w bazie danych. Jeśli dane są poprawne, system generuje i zwraca JSON Web Token, który służy do dalszej weryfikacji oraz identyfikacji użytkownika. Aplikacja oferuje szereg endpointów, z których każdy odpowiada za określoną funkcjonalność. W zależności od uprawnień użytkownika (Administrator vs Mechanik), dostęp do niektórych z nich może być ograniczony. Poniżej znajduje się lista wszystkich dostępnych endpointów:

POST /api/Auth/login

GET /api/SaleDocument

POST /api/SaleDocument

GET /api/SaleDocument/{id}

GET /api/SaleDocument/exists/{id}

PUT /api/SaleDocument/{SaleDocumentId}

DEL /api/SaleDocument/{saleDocumentId}

GET /api/SparePart

POST /api/SparePart

GET /api/SparePart/{id}

PUT /api/SparePart/{sparePartId}

DEL /api/SparePart/{sparePartId}

GET /api/SparePartsOrder/{id}/exists

GET /api/Tool

POST /api/Tool

GET /api/Tool/{id}

PUT /api/Tool/{toolId}

DEL /api/Tool/{toolId}

GET /api/ToolsOrder

POST /api/ToolsOrder

PUT /api/ToolsOrder/{id}

DEL /api/ToolsOrder/{id}

GET /api/ToolsRequest

POST /api/ToolsRequest

GET /api/ToolsRequest/{id}

GET /api/ToolsRequest/exists/{id}

PUT /api/ToolsRequest/verify/{requestId}

DEL /api/ToolsRequest/{requestId}

GET /api/User

POST /api/User

GET /api/User/{id}

GET /api/User/{id}/exists

GET /api/User/{id}/type

GET /api/User/profile

PUT /api/User/{UserId}

DEL /api/User/{UserId}

PUT /api/User/{UserId}/changepassword

GET /api/Vehicle

POST /api/Vehicle

GET /api/Vehicle/{id}

GET /api/Vehicle/{vin}

GET /api/Vehicle/{id}/exists

PUT /api/Vehicle/{vehicleId}

DEL /api/Vehicle/{vehicleId}

GET /api/WorkshopClient

POST /api/WorkshopClient

GET /api/WorkshopClient/{id}

PUT /api/WorkshopClient/{id}

DEL /api/WorkshopClient/{id}

GET /api/WorkshopClient/{id}/vehicles

GET /api/WorkshopClient/{id}/type

GET /api/WorkshopClient/{id}/exists

GET /api/WorkshopJob

POST /api/WorkshopJob

GET /api/WorkshopJob/{id}

GET /api/WorkshopJob/{id}/exists

PUT /api/WorkshopJob/{workshopJobId}

DEL /api/WorkshopJob/{workshopJobId}

GET /api/WorkshopTask

POST /api/WorkshopTask

GET /api/WorkshopTask/{id}

GET /api/WorkshopTask/{id}/exists

PUT /api/WorkshopTask/{workshopTaskId}

DEL /api/WorkshopTask/{workshopTaskId}

Endpointy wystawiane są przez serwer web Kestrel, który wchodzi w skład .NET 9.

Używane pakiety NUGET:

- AutoMapper 13.0.1
- BCrypt.Net-Next 4.0.3
- Microsoft.AspNetCore.Authentication.JwtBearer 9.0.0
- Microsoft.AspNetCore.OpenApi 9.0.0
- Microsoft.EntityFrameworkCore 9.0.0
- Microsoft.EntityFrameworkCore.Tools 9.0.0
- Microsoft.VisualStudio.Azure.Containers.Tools.Targets 1.21.0
- Npgsql.EntityFrameworkCore.PostgreSQL 9.0.2
- Scalar.AspNetCore 1.2.55

Do wizualizacji endpointów wykorzystujemy Open API, a do przedstawienia ich w formie graficznej – Scalar’a.

Do mapowania między klasami używamy narzędzia AutoMapper, w którym definiujemy profile mapowania w folderze Helper.

W celu zwiększenia bezpieczeństwa systemu przechowywane są jedynie hashe haseł, co realizujemy za pomocą pakietu BCrypt (warto rozważyć również implementację Argon2 ID jako alternatywę).

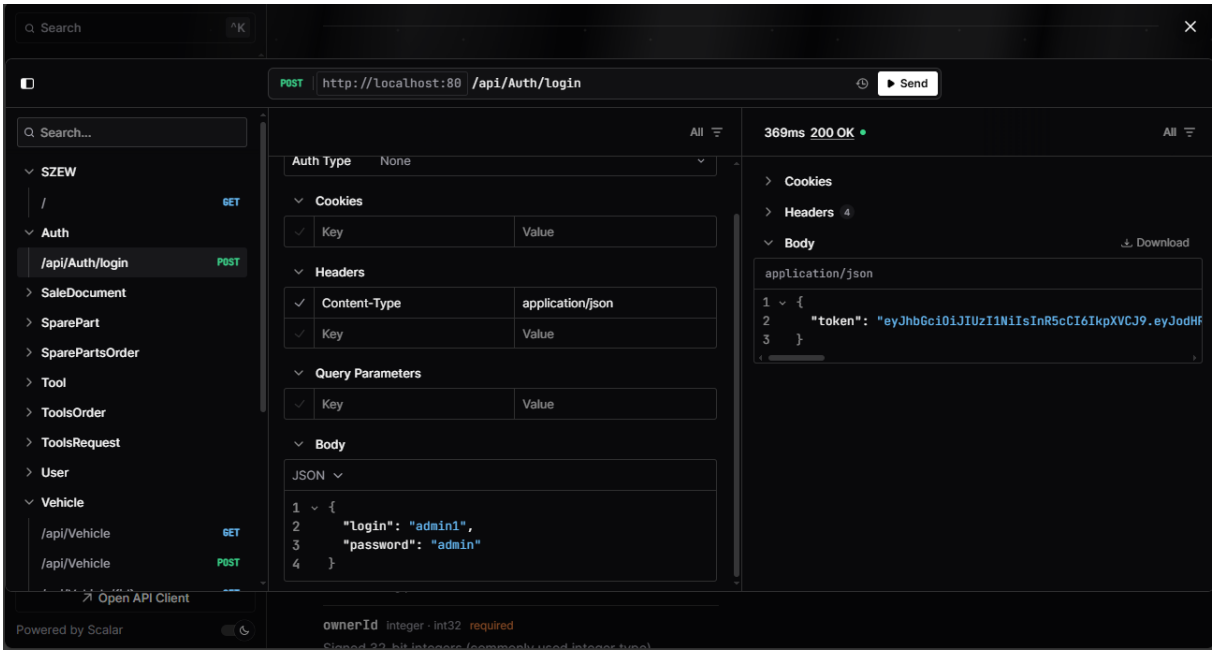
Wybór bazy danych padł na PostgreSQL, który został wybrany ze względu na otwarte środowisko, wysoką wydajność oraz korzystne koszty użytkowania. Do integracji z tą bazą danych używamy pakietu Npgsql, stanowiącego most pomiędzy PostgreSQL a Entity Framework od Microsoftu. Baza danych znajduje się poza systemem, a w systemie konfigurowana jest jedynie łączność z bazą, przy pomocy parametrów takich jak adres hosta, port, nazwa bazy danych, nazwa użytkownika oraz hasło (tzw. connection string).

Umiejscowienie aplikacji

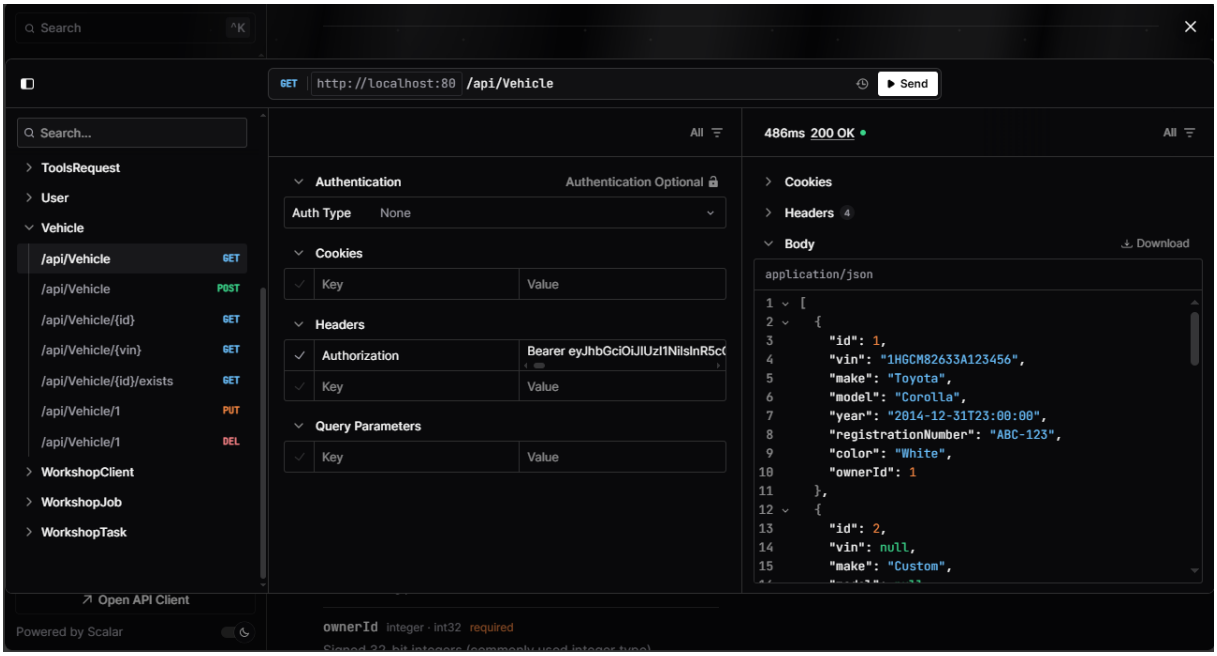
Aplikacja została zaprojektowana z myślą o obsłudze wielu środowisk uruchomieniowych. Niemniej jednak, w celu ułatwienia testowania jej funkcjonalności, zostanie zaprezentowana w wersji lokalnej

Skrócona instrukcja obsługi

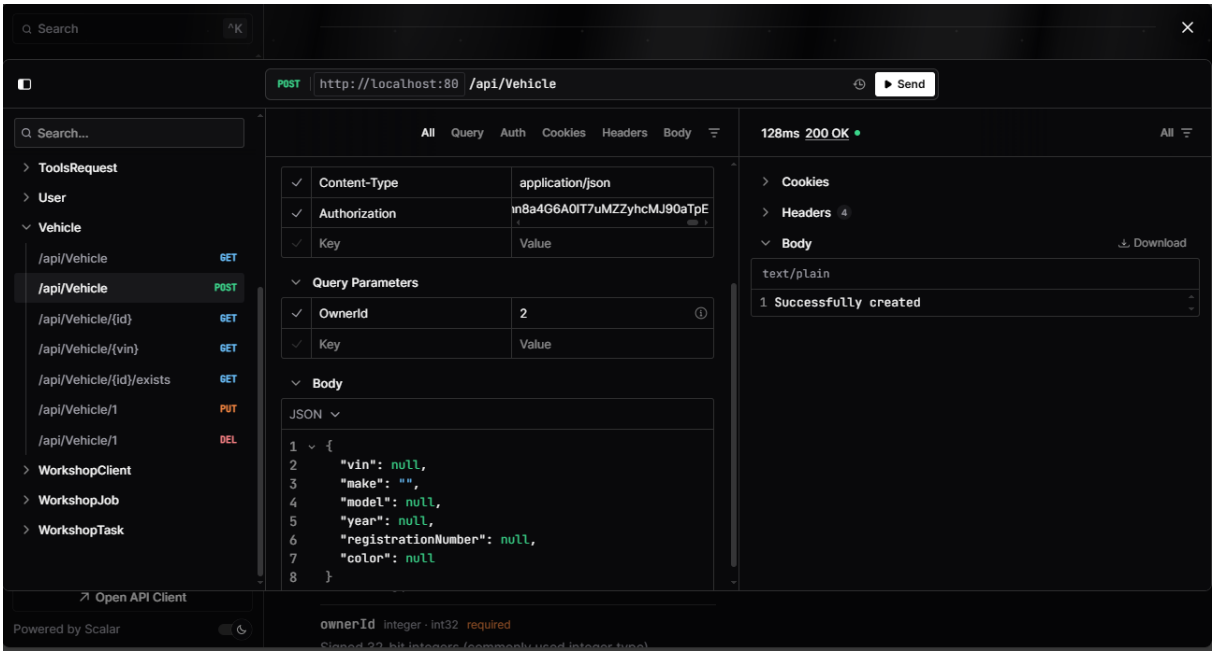
1. Uruchamiamy Visual Studio jako Administrator
2. Uruchamiamy bazę danych
    a. Wykorzystanie pgAdmin
        i. Uruchamiamy aplikację pgAdmin
        ii. Włączamy bazę danych
    b. Wykorzystanie dockera
        i. Uruchamiamy Docker Desktop
        ii. Wybieramy ikonkę compose i uruchamiamy aplikację
3. Jeżeli pierwszy raz uruchamiamy aplikację, w wierszu poleceń, w lokalizacji “/SZEW/SZEW/” musimy użyć “dotnet run testdata”, aby umieścić przykładowe dane w bazie danych.
    a. Aby odświeżyć zestaw danych w bazie wpisujemy “dotnet run testdata forced”
4. Włączamy aplikację poprzez wpisanie `dotnet run` w konsoli w lokalizacji “SZEW/SZEW/” lub włączając zielony przycisk “play”. Możemy również wybrać czy chcemy mieć szyfrowane połączenie (https) lub zwykłe (http)
5. Włączamy przeglądarkę internetową i wpisujemy frazę http://localhost/scalar/v1 (jeżeli włączyliśmy program w kroku 3 w trybie http) lub https://localhost/scalar/v1 (jeżeli włączyliśmy program w kroku 3 w trybie https)
6. Po lewej stronie ekranu wyświetli nam się nawigator. Rozwijamy listę “Auth”, klikamy w “/api/Auth/login”. Po prawej stronie ekranu pojawi nam się okienko ze strukturą z poleceniem “curl”. W tym okienku klikamy przycisk “▷ Test Request”
7. Po kliknięciu wyświetli nam się okno z do wprowadzenia odpowiednich danych. Zjeżdżamy niżej i na samym dolę widzimy sekcję “Body” w formacje JSON. W polu “login” wpisujemy “admin1”, a w polu “password” wpisujemy “admin”.
8. Po kliknięciu przycisku “▷ Send” który znajduje się obok paska z adresem, po niewielkim czasie oczekiwania wyświetli nam się format token dostępu w formacje json.



9. Kopiujemy token i wklejamy go gdzieś w bezpieczne miejsce np. w notatniku tak, aby nigdzie się nam nie zgubił. W przeciwnym razie musielibyśmy ponowić kroki od 6 do 8, aby wygenerować go ponownie.
10. Jeżeli dalej znajdujemy się w tym samym okienku co w poprzednim punkcie, to w lewym górnym rogu “pod paskiem Search” mamy ikonkę która rozwinie nam pozostałe funkcjonalności z których możemy skorzystać. Dzięki temu rozwiązaniu będzie nam dużo łatwiej przemieszczać się między funkcjonalnościami aplikacji.
11. W naszej aplikacji możemy pobierać informacje z bazy danych (GET), zamieszczać nowe rekordy w bazie (POST), edytować rekordy (PUT), oraz je usuwać (DEL).
12. Po lewej stornie przechodzimy np. do modułu “Vehicle” i rozwijamy zakładkę.
13. Wybieramy “/api/Vehicle” z polem “GET”.
14. W headers w polu “Key” wpisujemy “Authorization”, a obok w polu “Value” wpisujemy “Bearer “ i wklejamy nasz wcześniej wygenerowany token. Ostateczna wersja powinna wyglądać następująco “Bearer <Token>”
15. Następnie klikamy “▷ Send” i po prawej stronie wyświetlą nam się dane pobrane z bazy w formacje json.

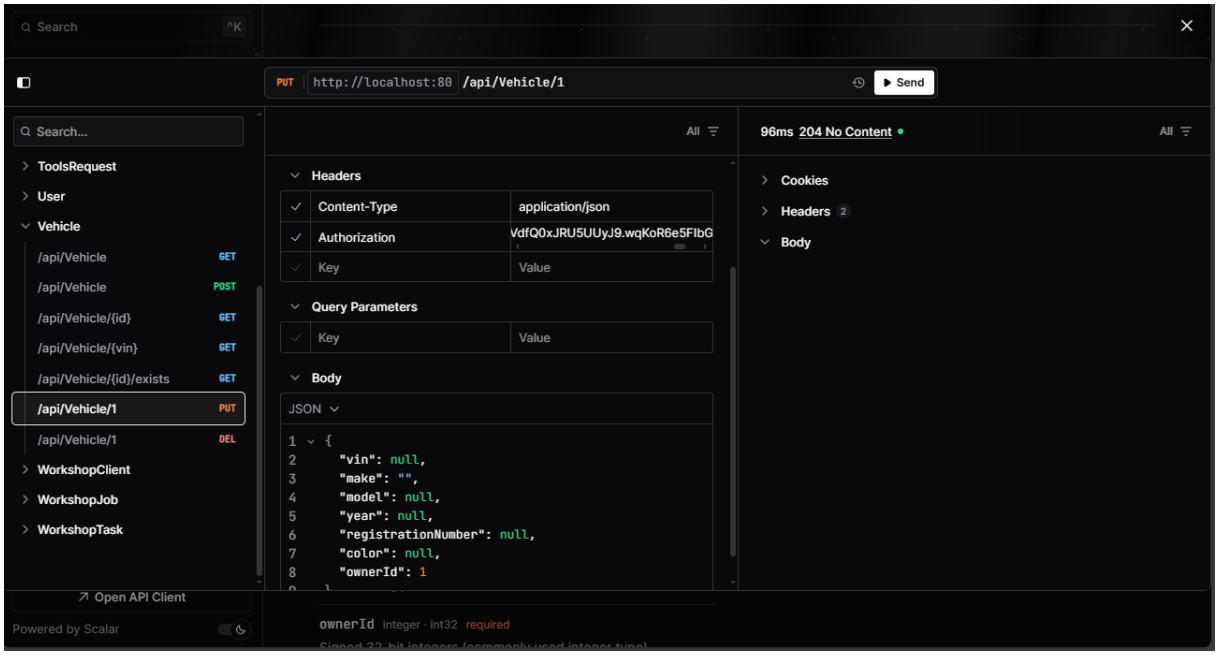


16. Aby dodać dane do bazy, klikamy “/api/Vehicle” z polem “POST”.
17. Powtarzamy punkt 14 w nowym wierszu w Headers (nie usuwamy “Content-Type” i “application/json”).
18. W tym przypadku uzupełniamy również “OwnerId” w sekcji “Query Parameters”, oznaczający do jakiej osoby ma być przypisany pojazd.
19. W polu “Body” uzupełniamy wszystkie dane i klikamy “▷ Send”.
20. Jeżeli poprawnie wprowadzimy dane, to powinna wyświetlić się nam informacja “Successfully created”

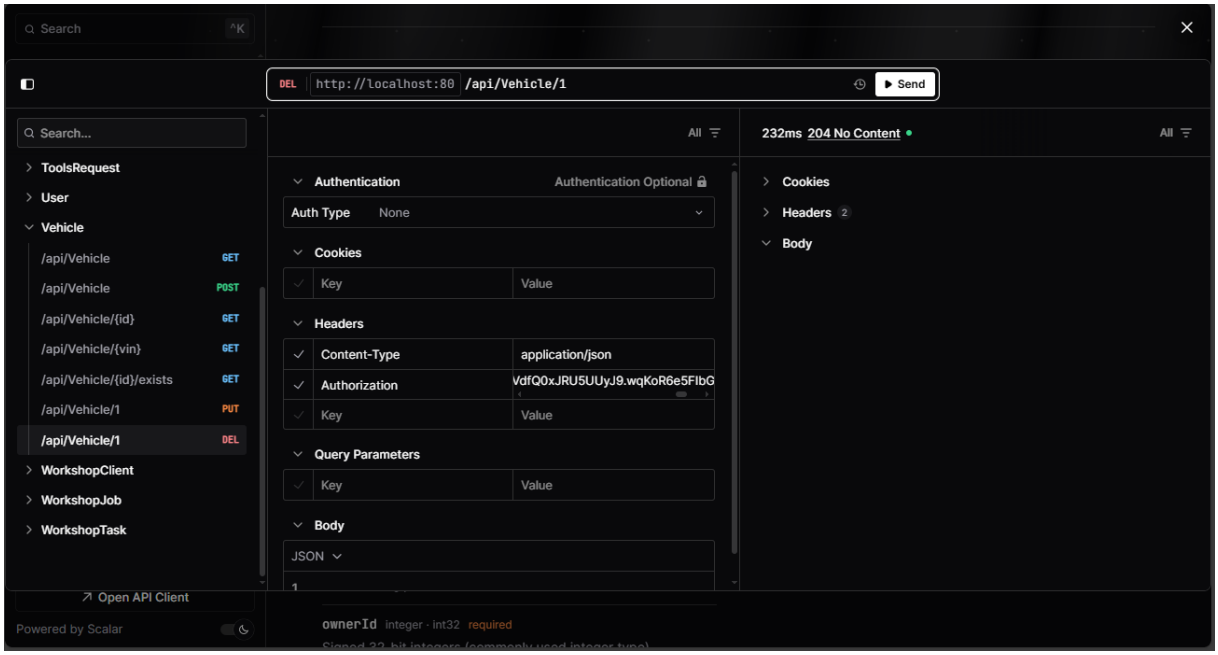


21. Aby edytować rekord w bazie, klikamy “/api/Vehicle” z polem “PUT”.
22. Powtarzamy punkt 17
23. U góry w adresie edytujemy zamieniamy “{vehicleId}” na interesujące nas id w bazie np. 1

24. W polu “Body” uzupełniamy wszystkie dane i klikamy “▷ Send”.
25. Jeżeli operacja przebiegła poprawnie, to pod przyciskiem “▷ Send”, powinna wyświetlić się zielonka kropka



26. Aby usunąć rekord z bazy, klikamy “/api/Vehicle” z polem “DEL”.
27. Powtarzamy punkt 17
28. U góry w adresie edytujemy zamieniamy “{vehicleId}” na interesujące nas id w bazie np. 1 i klikamy “▷ Send”.
29. Jeżeli operacja przebiegła poprawnie, to pod przyciskiem “▷ Send”, powinna wyświetlić się zielonka kropka



30. W taki sposób możemy sprawdzać działanie innych modułów.