

Politechnika Śląska

Wydział Matematyk Stosowanej

Kierunek Informatyka



**Politechnika
Śląska**

Gliwice, 07.07.2020

Programowanie obiektowe i graficzne

projekt zespołowy

"Salon samochodowy"

Skład zespołu projektowego:

Kamil Szpak gr. lab. 3H

Anna Dzierżawa gr. lab. 3H

Paweł Magdziarz gr. lab. 3H

Artur Giza gr. lab. 3H

1. Opis projektu.

Jest to aplikacja bazodanowa pozwalająca prowadzić salon samochodowy. Łączy się ona zdalnie z serwerem MySQL, co zapewnia współdzielony dostęp do bazy danych.

Do aplikacji logują się pracownicy poprzez indywidualny login i hasło. Aplikacja zapewnia następujące funkcjonalności:

- Dodaj/Edytuj/Usuń Pracownika (właściciel)
- Dodaj/Edytuj/Usuń pojazd w salonie
- Nowa sprzedaż
- Moje sprzedaże
- Sprawdź Modele
- Statystyki pracowników (właściciel)

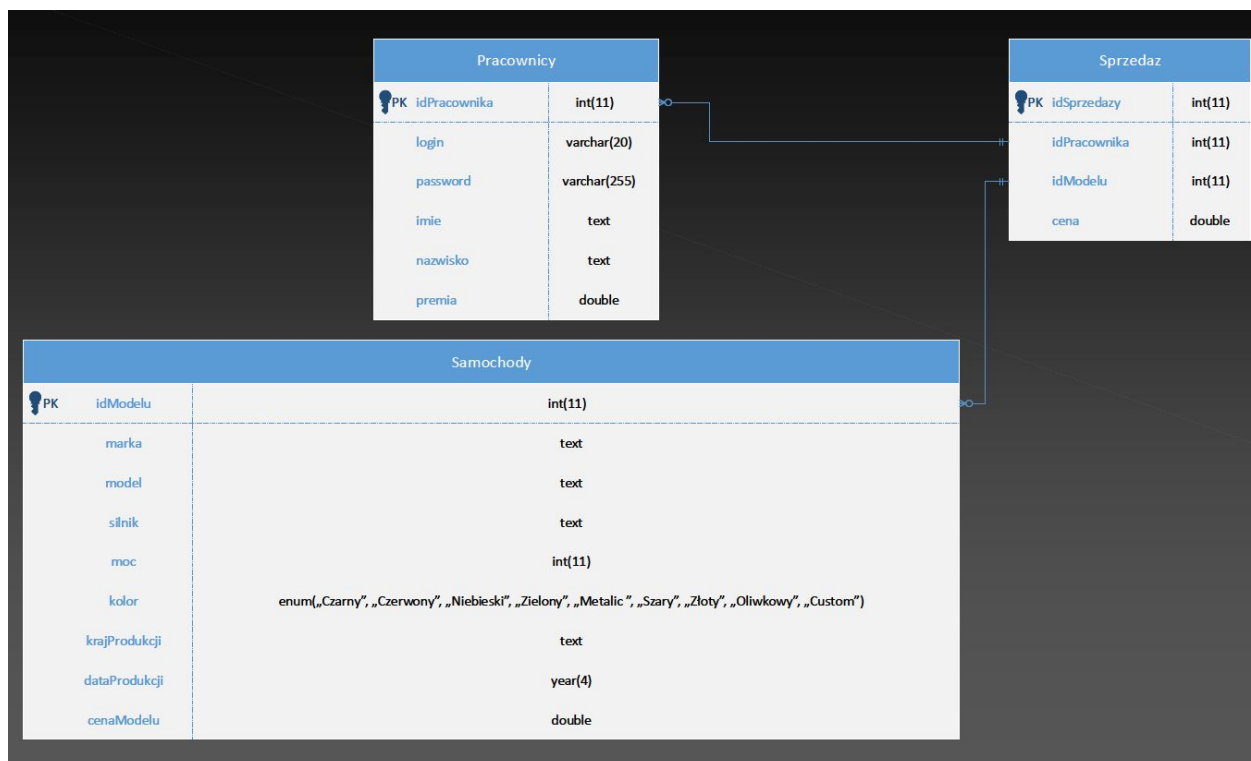
Ponadto każdy z użytkowników posiada zakładkę Konto gdzie może dokonać zmiany aktualnego hasła, uzyskać informację na temat własnego konta, czy się z niego wylogować.

2. Wymagania.

By móc korzystać z aplikacji potrzebujemy komputer wraz z zainstalowanym systemem Windows oraz z dostępem do internetu, dzięki któremu będziemy mogli połączyć się z bazą danych znajdującą się na serwerze MySQL.

3. Przebieg realizacji.

Schemat bazy danych



Opis bazy danych

W projekcie wykorzystywana jest baza danych salonu samochodowego, która zawiera informacje dotyczące pracowników, sprzedawanych modeli pojazdów oraz każdej sprzedaży dokonanej w tym salonie. Baza składa się z 3 tabel: Pracownicy, Samochody, Sprzedaż.

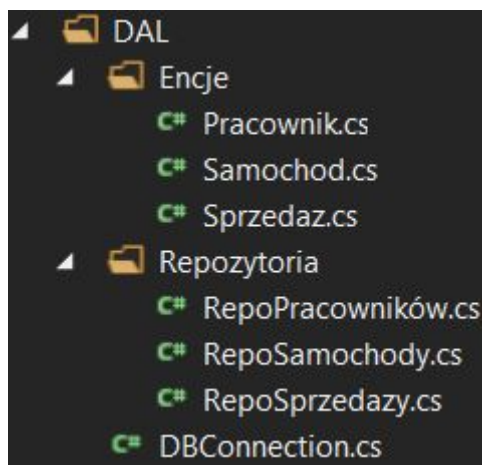
Baza umożliwia dostęp do informacji:

- **Tabela Pracownicy:** Dane pracownika, jego indywidualny login oraz hasło umożliwiające dostęp do aplikacji, numer identyfikacyjny pracownika, premie.
- **Tabela Samochody:** Nazwa marki i modelu samochodu, jego dane techniczne, numer identyfikacyjny pojazdu, kraj i data produkcji oraz cena.
- **Tabela Sprzedaż:** Cena oraz numer identyfikacyjny danej transakcji, numer identyfikacyjny sprzedawcy.

Założenia:

- Możliwość dodawania kolejnych pracowników i usuwania ich oraz tworzenie dla nich danych logowania.
- Możliwość stworzenia listy pojazdów dostępnych do nabycia w salonie, edytowanie ich oraz usuwanie. Pojazdy posiadają specyfikację techniczną, dane produkcyjne oraz cenę.
- Możliwość archiwizowania sprzedaży. Dzięki temu możemy prowadzić statystykę sprzedaży wśród pracowników oraz wyliczać dla nich dodatkowe premie. Premia do wynagrodzenia stanowi 2% ceny od sprzedanego samochodu.

Implementacja **warstwy DAL** odpowiedzialnej za obsługę bazy danych (MySQL):



Zawiera ona kolejne 2 warstwy: **Encje** oraz **Repozytoria**

Encje reprezentują tabele znajdujące się w bazie danych dlatego stworzone zostały 3 klasy Pracownika, Samochód oraz Sprzedaż. Każda z nich posiada zmienne odpowiadające wartościom jakie zostały nadane podczas tworzenia poszczególnych encji.

- **Klasa Pracownik** posiada konstruktor umożliwiający stworzenie obiektu wczytanego z bazy danych. Ponadto występują tam dwa kolejne konstruktory. Jeden z nich tworzy nieobecny jeszcze w bazie obiekt, któremu zostaje przypisany odpowiedni numer ID na podstawie listy wszystkich obiektów. Dla nowo powstałych obiektów ID będzie o 1 większy od poprzedniego wykorzystanego. Trzeci konstruktor umożliwia nam kopiowanie obiektu. Klasa ta posiada metodę **Equals()**, która służy do sprawdzania czy w liście pracowników znajduje się obiekt, który chcemy dodać do bazy. Metoda **ToInsert()** odpowiada za generowanie stringa składającego się z danych pracownika. Jest wykorzystywana przy dodawaniu go do bazy danych.
- **Klasa Samochód** została pozbawiona metody sprawdzającej Equals(), ponieważ w salonie mogą być sprzedawane auta o identycznych specyfikacjach.
- **Klasa Sprzedaz** została pozbawiona metody sprawdzającej Equals() oraz konstruktora kopiującego.

Repozytoria zawierają metody odpowiedzialne za pobieranie obiektów i usuwanie z bazy, dodawanie ich do niej oraz edytowanie dla Pracowników, Samochodów i Sprzedaży.

Klasa DBConnection odpowiada za przygotowanie połączenia z bazą danych. Konstruktor domyślny **DBConnection()** pozyskuje nazwę serwera (IP) na której znajduje się baza danych salonu, login użytkownika, hasło za pomocą którego powinno być wykonane połączenie, nazwę bazy do zainicjowania połączenia oraz numer portu wykorzystywany do komunikacji serwerem MYSQL.

Projekt został zrealizowany z wykorzystaniem wzorca projektowego

MVVM (Model - View - ViewModel).

Poszczególne warstwy prezentują się następująco:

- Model

W tej warstwie pobierane i przetwarzane są informacje z bazy danych. Dane pobierane są do kolekcji za pomocą metod zaimplementowanych w warstwie DAL. W warstwie Model znajdziemy także logikę biznesową oraz implementacje metod odpowiedzialnych za:

- Wyszukiwanie po ID samochodu, pracownika oraz pojedynczej sprzedaży
- Dodawanie pracowników, samochodów, sprzedaży do bazy po uprzednim sprawdzeniu czy poszczególne rekordy już znajdują się w bazie
- Edycję i usuwanie poszczególnych rekordów z bazy
- Przypisywanie ID z bazy danych na podstawie ID z listy w programie (ListView w warstwie widoku)

Implementacja każdego typu z ww. metod jest do siebie bardzo zbliżona, różnice stanowi miejsce, z którego pobierane są dane (różne tabele)

- View

Pierwszym widokiem, który użytkownik zobaczy po uruchomieniu aplikacji jest LoginScreen.xaml. Widokiem głównym, w którym umieszczone zostały zakładki przenoszące nas do poszczególnych paneli, jest MainWindow.xaml. W centralnej części tego widoku znalazł się DataTemplate, który wypełniany jest innymi widokami (About.xaml, AddCar.xaml, Selling.xaml) po kliknięciu w dany przycisk w rozwijanym panelu z zakładkami.

W prawym górnym rogu MainWindow.xaml znalazł się PopUp Box. Niektóre z widoków pojawiających się po wybraniu jednej z opcji zawartych w PopUp Boxie nie wypełniają DataTemplate, lecz pojawiają się jako osobne okienka (ChangePassword.xaml, AboutAcc.xaml).

Logika odpowiedzialna za przełączanie się pomiędzy poszczególnymi widokami zawarta jest w odpowiadających klasach z warstwy ViewModel.

- ViewModel

W tej warstwie znalazły się właściwości i komendy niezbędne do prawidłowego działania po połączeniu z warstwą widoku. Każdy widok posiada obsługującą go klasę w warstwie

ViewModel. Klasy w tej warstwie są do siebie bardzo zbliżone budową i różnią się jedynie właściwościami oraz komendami w zależności od obsługiwanego widoku.

Opis klasy w warstwie ViewModel na podstawie klasy AddCarVM:

Klasa dziedziczy po ViewModelBase, będącą klasą bazową w ViewModelu. Konstruktor klasy AddCarVM przyjmuje obiekt modelu i wywołuje metodę, która odpowiada za stworzenie kolekcji samochodów znajdujących w bazie (pobierane są marka i model).

```

public AddCarVM(Model model)
{
    this.model = model;
    ZaladujSamochodyDoListy();
}

Odwołania: 4
public void ZaladujSamochodyDoListy()
{
    Samochody = model.Samochody;
    samochodyLista = new ObservableCollection<string>();
    foreach (var samochod in Samochody)
    {
        samochodyLista.Add($"{samochod.Marka} {samochod.ModelPojazdu}");
    }
}

```

Komendy znajdujące się w klasie odpowiedzialne są za uzupełnianie TextBoxów informacjami na temat danego pojazdu oraz obsługę pojazdów w bazie (dodawanie, edytowanie, usuwanie).

Implementacja komendy na podstawie DodajSamochod:

```

private ICommand dodajSamochod = null;
Odwołania: 0
public ICommand DodajSamochod
{
    get
    {
        if (dodajSamochod != null)
            return dodajSamochod;
        dodajSamochod = new RelayCommand(
            arg =>
            {
                var samochod = new Samochod(Marka, modelPojazdu, Silnik, Kolor, KrajProdukcji, rokProdukcji, Cena, Moc);
                if (!model.DodajSamochod(samochod))
                    return;
                ZaladujSamochodyDoListy();
                ClearAll();
                OnPropertyChanged(nameof(samochodyLista));
                OnPropertyChanged(nameof(marka));
                OnPropertyChanged(nameof(modelPojazdu));
                OnPropertyChanged(nameof(cena));
                OnPropertyChanged(nameof(moc));
                OnPropertyChanged(nameof(kolor));
                OnPropertyChanged(nameof(rokProdukcji));
                OnPropertyChanged(nameof(krajProdukcji));
                OnPropertyChanged(nameof(silnik));
                System.Windows.MessageBox.Show($"{Marka} {ModelPojazdu} został dodany do bazy!");
            },
            arg => (Marka != "") && (ModelPojazdu != "") && (Silnik != "") && (Kolor != "") && (KrajProdukcji != "") && (RokProdukcji != "") && (Moc != 0) && (Cena != 0)
        );
        return dodajSamochod;
    }
}

```

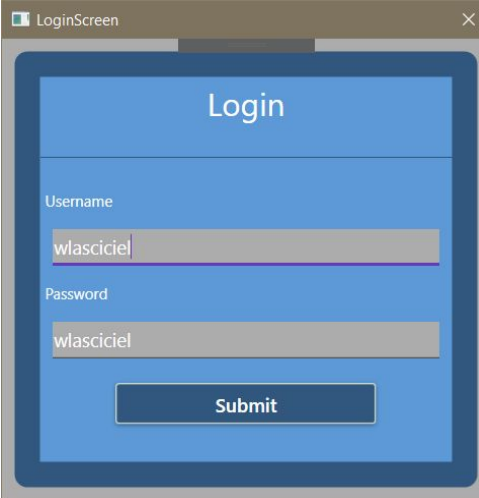
Dodawanie samochodu odbywa się z wykorzystaniem metody RelayCommand zaimplementowanej w klasie bazowej. Jako pierwszy argument podane są polecenia do zrealizowania. Tworzymy tam nowy obiekt klasy Samochod o właściwościach podanych w TextBoxach. Następnie z wykorzystaniem metody DodajSamochod, przyjmującej jako argument obiekt klasy Samochod odbywa się właściwe dodawanie rekordu do bazy.

Następnie wywoływana jest metoda ZaladujSamochodyDoListy, która znalazła się także w konstruktorze. Umożliwia to “odświeżenie” kolekcji od razu po dodaniu pojazdu. W kolejnych etapach odbywa się “czyszczenie”, czyli ustawianie jako puste, TextBoxów oraz wywoływane są metody zgłaszające zmiany we właściwościach podanych jako argumenty.

Drugim argumentem, jaki przyjmuje RelayCommand, są warunki, które muszą zostać spełnione aby komenda mogła zostać wywołana. W tym przypadku pola przyjmujące wartość typu *string* nie mogą być puste, a pola przyjmujące wartości liczbowe muszą być większe od zera.

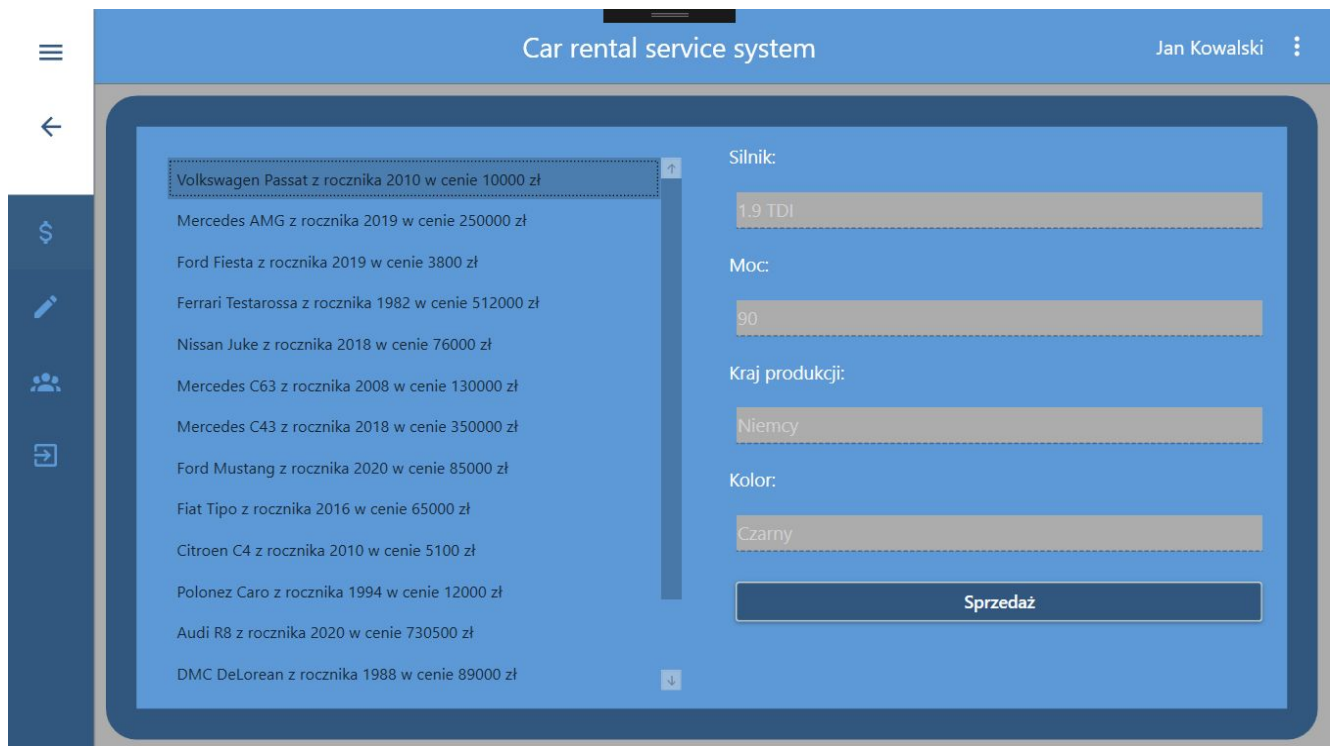
4. Instrukcja użytkowania.

Po uruchomieniu aplikacji pojawia się okno logowania. Należy podać przypisane do pracownika lub właściciela dane logowania.

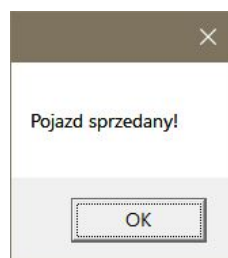


The screenshot shows a window titled "LoginScreen" with a close button (X) in the top right corner. Inside the window is a blue rectangular area with the word "Login" at the top. Below "Login" are two input fields. The first field is labeled "Username" and contains the text "własciciel". The second field is labeled "Password" and also contains the text "własciciel". Below these fields is a dark blue button with the word "Submit" in white text.

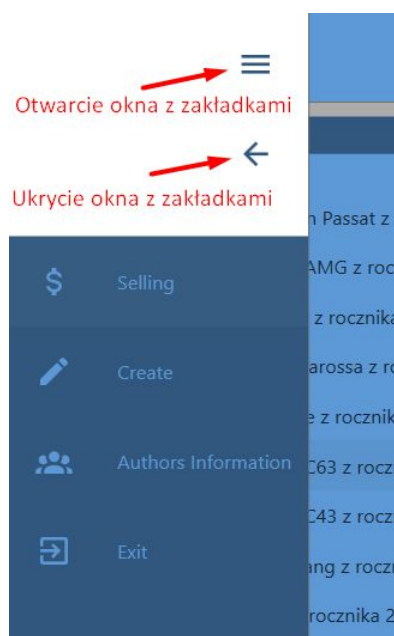
Po zalogowaniu zostanie wyświetlona automatycznie jedna z zakładek o nazwie **“Selling”**, w której to dokonujemy sprzedaży dowolnego modelu znajdującego się w bazie danych. Aby sprzedać pojazd musimy go wybrać z listy dostępnych samochodów i zatwierdzić transakcję przyciskiem **“Sprzedaż”**. Każdy obiekt z listy posiada **nazwę marki, model, rok produkcji** oraz **cenę**. Aby uzyskać informację o **silniku, mocy, kraju pochodzenia** oraz **kolorze** pojazdu musimy wybrać interesujący nas samochód.



Po pomyślnym dokonaniu sprzedaży zostanie zwrócona informacja:



Aby korzystać z innych funkcjonalności jakie dostarcza nam aplikacja musimy rozwinąć **okno nawigacji** i wybrać interesującą nas **zakładkę**:



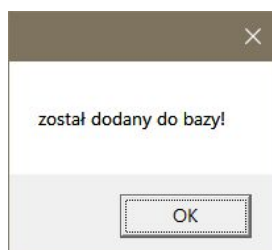
Znajdują się tam 4 dostępne opcje pozwalające nam na:

- Wyżej omówioną sprzedaż.
- Modyfikację pojazdów dostępnych w salonie.
- Uzyskanie informacji o twórcach aplikacji i wykonanych przez nich zadań.
- Zamknięcie aplikacji.

Zakładka Create umożliwia nam **wprowadzenie nowych pojazdów** do salonu, **modyfikację** aktualnie dostępnych, a także **usunięcie** wyprzedanych modeli.

Dodawanie pojazdu dokonujemy poprzez uzupełnienie rubryk informacyjnych dotyczących samochodu jaki chcemy wprowadzić do sprzedaży. Po wprowadzeniu ich zatwierdzamy cały proces przyciskiem **“Dodaj”**. Dodany model ukaże się na końcu listy wyświetlonej po prawej stronie programu.

Marka:	Model:
<input type="text" value="Opel"/>	<input type="text" value="Astra"/>
Silnik:	Moc:
<input type="text" value="1.7"/>	<input type="text" value="135"/>
Kraj produkcji:	Data produkcji:
<input type="text" value="Włochy"/>	<input type="text" value="2014"/>
Kolor:	Cena:
<input type="text" value="Czarny"/>	<input type="text" value="57400"/>
<input type="button" value="Edytuj"/>	<input type="button" value="Usuń"/>
<input type="button" value="Dodaj"/>	

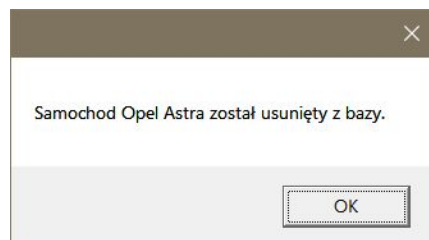


Marka:	Model:
Opel	Astra
Silnik:	Moc:
1.7	135
Kraj produkcji:	Data produkcji:
Włochy	2014
Kolor:	Cena:
Czarny	57400

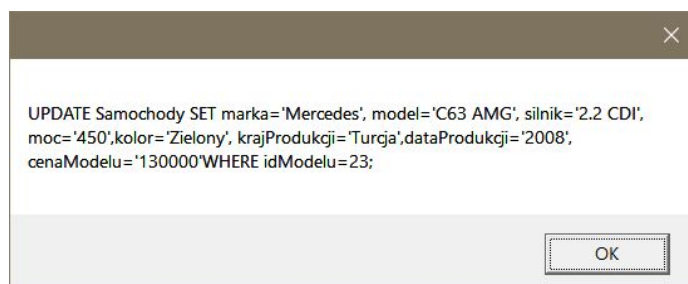
Edytuj Usuń Dodaj

- Ford Fiesta
- Ferrari Testarossa
- Nissan Juke
- Mercedes C63
- Mercedes C43
- Ford Mustang
- Fiat Tipo
- Citroen C4
- Polonez Caro
- Audi R8
- DMC DeLorean
- Opel Astra

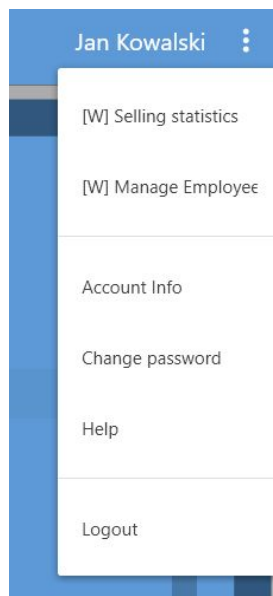
Jeśli chcemy **usunąć pojazd** wystarczy go wybrać z listy i zatwierdzić operację przyciskiem **“Usuń”**. Po usunięciu pojazdu z salonu pojawi się komunikat:



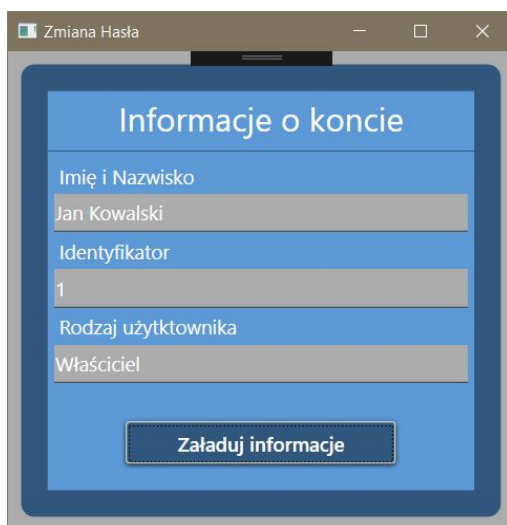
W celu **edycji pojazdu** należy wybrać interesujący nas model z listy. Następnie możemy dokonać dowolnych zmian wśród informacji wyświetlanych w rubrykach po lewej stronie okna. Zmiany zatwierdzamy przyciskiem **“Edytuj”**. Pomyślnie zakończona operacja edycji zostanie zakomunikowana:



W prawym górnym rogu możemy zauważyć **imię i nazwisko aktualnie zalogowanego użytkownika**, natomiast obok tych danych znajdują się **pop up box** zawierający następujące funkcjonalności:



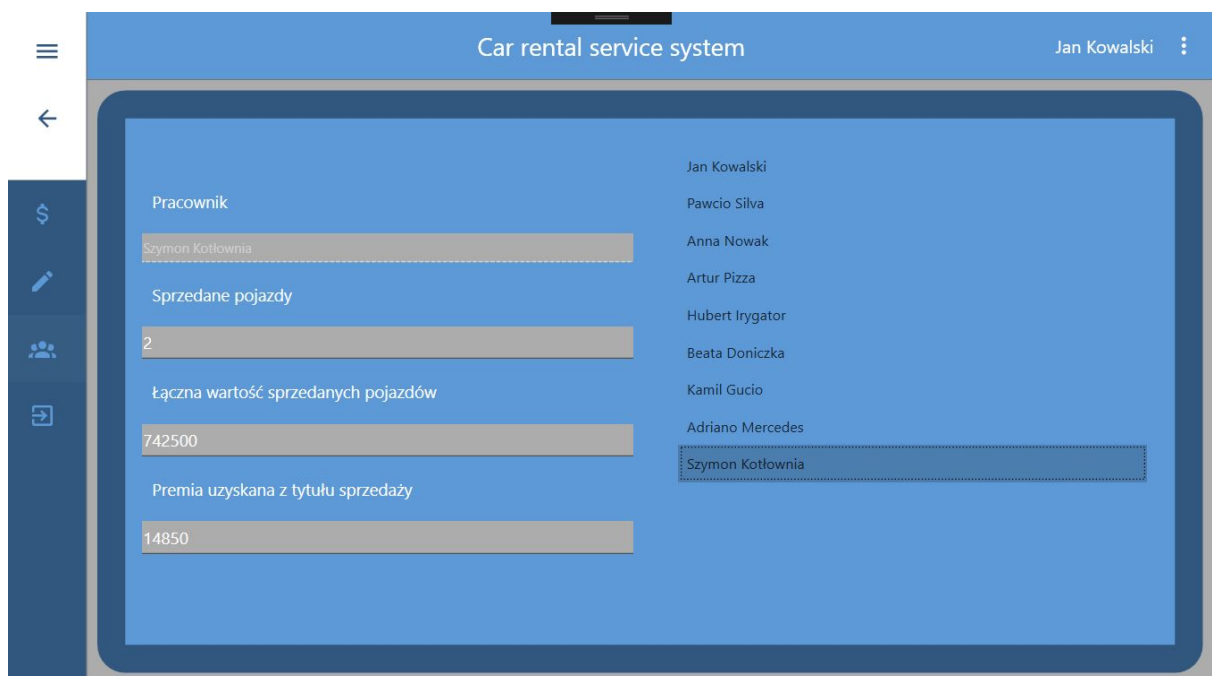
- **Informacja o koncie** (Aby wyświetlić dane o aktualnie zalogowanym użytkowniku musimy wpierw załadować dane poprzez przycisk “**Załaduj informacje**”).



- **Zmian hasła.**



- **Help** wyświetla informacje o autorach aplikacji.
- **Logout** wylogowuje aktualnie zalogowanego użytkownika i przenosi do okna logowania.
- **[W]Selling statistics** jest opcją dostępną **tylko dla Właściciela** (użytkownik o id = 1) i przedstawia statystyki sprzedaży pracowników salonu.



- **[W]Manage Employee** jest opcją dostępną **tylko dla Właściciela** i pozwala na zatrudnianie, zwalnianie pracowników oraz edytowanie ich danych służących do logowania.

Jeśli zwykły pracownik będzie próbował skorzystać z 2 ostatnich funkcjonalności zostanie poinformowany o braku uprawnień.

5. Podsumowanie i wnioski

W trakcie realizacji zadania projektowego stworzyliśmy program, który mógłby okazać się pomocny podczas prowadzenia salonu samochodowego. W pełni zrealizowaliśmy założone wcześniej cele. Podczas implementacji nie napotkaliśmy dużej liczby problemów. Praca nad projektem pozwoliła nam wykorzystać w szerszej praktyce wzorec MVVM, przy okazji utrwalając poznane w trakcie semestru cechy programowania obiektowego. Udało się nam także stworzyć większą aplikację pracując w grupie, co może okazać się bardzo przydatne w przyszłości. Funkcjonalność naszej aplikacji można w przyszłości poszerzyć np. o automatyczne kasowanie pojazdu z bazy w momencie sprzedaży lub o bardziej przejrzysty i nowoczesny interfejs użytkownika.

6. Dodatek - udokumentowanie wykorzystania systemu kontroli wersji.

Excluding merges, **4 authors** have pushed **51 commits** to master and **51 commits** to all branches. On master, **0 files** have changed and there have been **0 additions** and **0 deletions**.



1 Release published by 1 person

Published **v1.0 Release** 15 minutes ago

