

Tworzenie Aplikacji Bazodanowych

Raport z Projektu

Autobusy

Prowadzący projekt: dr inż. Łukasz Wyciślik

Kierunek: Informatyka SSI, semestr 6

Sekcja: S14

Autorzy:

Rafał Klinowski

Jakub Cisowski

Kacper Nitkiewicz

Paweł Kabza

Michał Chyla

Temat projektu

Tematem projektu jest stworzenie aplikacji bazodanowej – systemu obsługi przewoźnika komunikacji miejskiej.

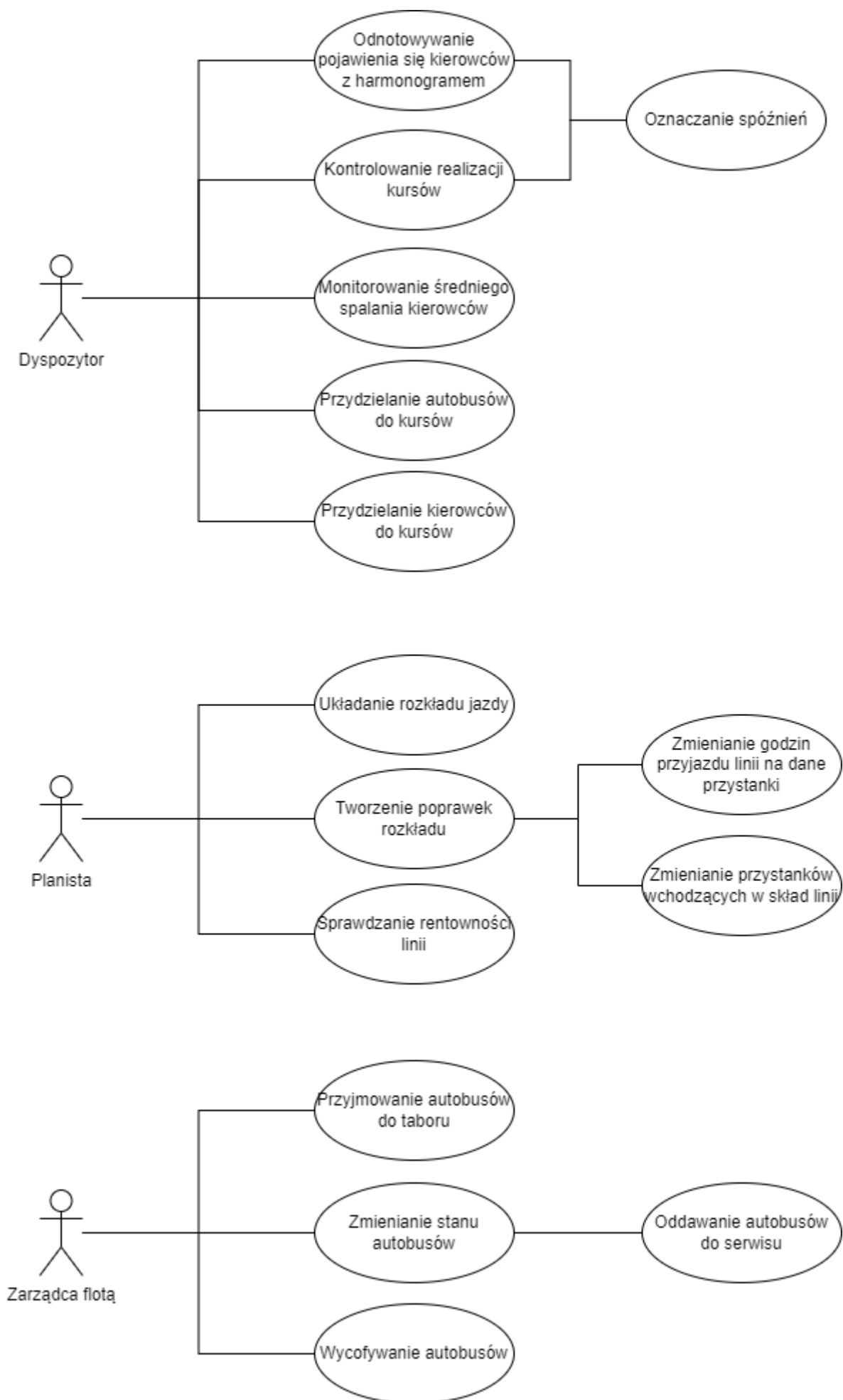
Aplikacja ma umożliwiać: definiowanie nowych przystanków i linii autobusowych, tworzenie rozkładu jazdy, zatrudnianie nowych kierowców, przyjmowanie nowych pojazdów do taboru, oddawanie autobusów do serwisu oraz przypisywanie kierowców, pojazdów i linii do kursów autobusowych. Co więcej, w aplikacji ma być możliwość weryfikacji pracy kierowców – możliwość oznaczania spóźnień, sprawdzanie średniego spalania paliwa – jak również monitorowania rentowności linii na podstawie ilości skasowanych biletów i spalonego paliwa.

Wszystkie wymienione powyżej aspekty mają być dynamiczne – oznacza to, że każdy element powinno dać się w każdym momencie zmienić przez odpowiednio do tego uprawnioną osobę. Przykładowo, po dodaniu do bazy nowego pojazdu, w każdym momencie powinna być możliwość zmiany jego stanu, numeru rejestracyjnego, oddania go do serwisu, itd.

Szczegółowy opis systemu

Pierwszym krokiem przy projektowaniu aplikacji było stworzenie odpowiednich diagramów – diagramu klas oraz przypadków użycia.

Diagram przypadków użycia prezentuje opisane wyżej informacje. W systemie znajduje się trzech aktorów – Dyspozytor, odpowiedzialny za zatrudnianie i monitorowanie kierowców, jak również przydzielanie ich do konkretnych przejazdów (więcej na ten temat w dalszej części raportu); Planista, odpowiedzialny za układanie rozkładu jazdy, dodawanie nowych linii oraz tworzenie poprawek rozkładu (takich jak: modyfikacja godzin przyjazdu, trasy, częstotliwości kursów); oraz Zarządca flotą, odpowiedzialny za przyjmowanie nowych pojazdów oraz monitorowanie ich stanu technicznego.



Rysunek 1 Diagram przypadków użycia.

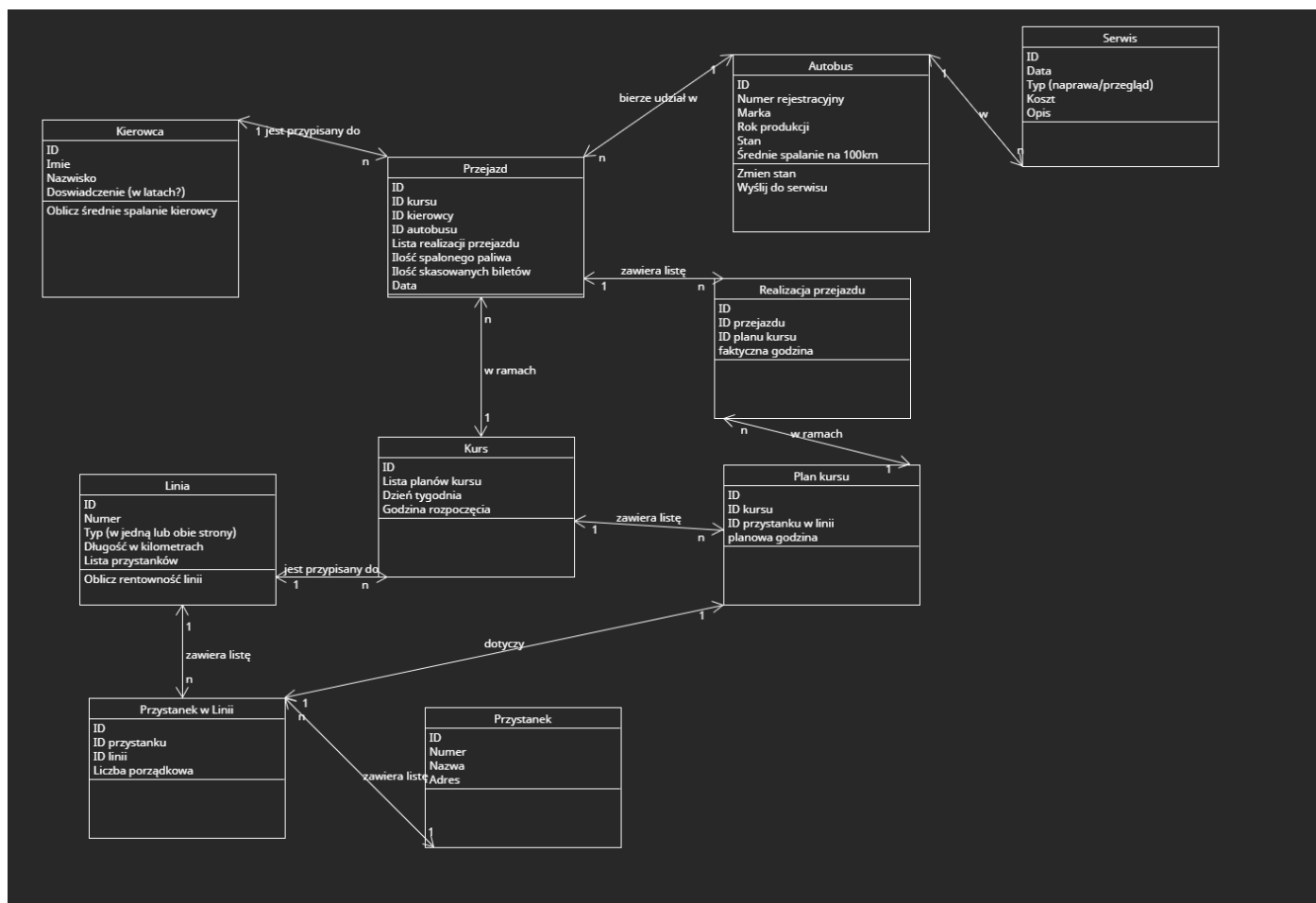
Diagram klas uległ zmianie kilkakrotnie w trakcie implementacji projektu. Było to związane z rozszerzeniem funkcjonalności aplikacji o dodatkowe klasy.

Podstawowymi jednostkami w systemie są klasy: Kierowca, Autobus oraz Przystanek. Klasa Serwis reprezentuje naprawę danego autobusu.

Klasa PrzystanekWLinii to sposób na połączenie danego przystanku z daną linią. Ponieważ jeden przystanek może być odwiedzany przez autobusy wielu linii, wówczas między Przystankiem a Przystankiem w Linii zastosowano relację 1:N. Analogicznie, Linia zawiera wiele przystanków – ponownie relacja 1:N.

Kurs jest nieco bardziej konkretną klasą niż Linia, jednak nadal jest to w pełni abstrakcyjna jednostka. Podczas gdy Linia zawiera definicję przystanków wraz z odpowiednią kolejnością, Kurs jest przypisany do konkretnego dnia i godziny. Ponadto, Kurs zawiera wiele obiektów typu PlanKursu – one zawierają z kolei dany Przystanek w Linii wraz z konkretną godziną planowego przyjazdu na ten przystanek.

Przejazd jest konkretną realizacją Kursu, odbywającym się w konkretny dzień o konkretnej godzinie z udziałem odpowiedniego kierowcy i autobusu. Przejazd zawiera dodatkowo informacje takie jak: ilość skasowanych biletów, ilość spalonego paliwa. Realizację przejazdu można kontrolować za pomocą klasy RealizacjaPrzejazdu – można w niej oznaczać rozbieżności w godzinach między planowym a faktycznym przejazdem na dane przystanki.



Rysunek 2 Ostateczna wersja diagramu klas.

Dla przykładu, zdefiniujmy jeden przejazd. Zaczynamy od przystanków:

<i>Id przystanku</i>	Numer	Nazwa	Adres
1	11	Pierwszy	Pierwsza 11
2	22	Drugi	Druga 22
3	33	Trzeci	Trzecia 33

Następnie definiujemy linię, która odwiedza przystanki w kolejności: Pierwszy > Drugi > Trzeci.

<i>Id linii</i>	Numer	Typ	Długość w km	Przystanki
1	50	W jedną stronę	13.32	{ Pierwszy, Drugi, Trzeci }

Kolejny krok to stworzenie kursu dla tej linii wraz z odpowiednimi planami kursu:

<i>Id kursu</i>	Numer linii	Dzień tygodnia	Godzina rozpoczęcia
1	50	Poniedziałek	8:00

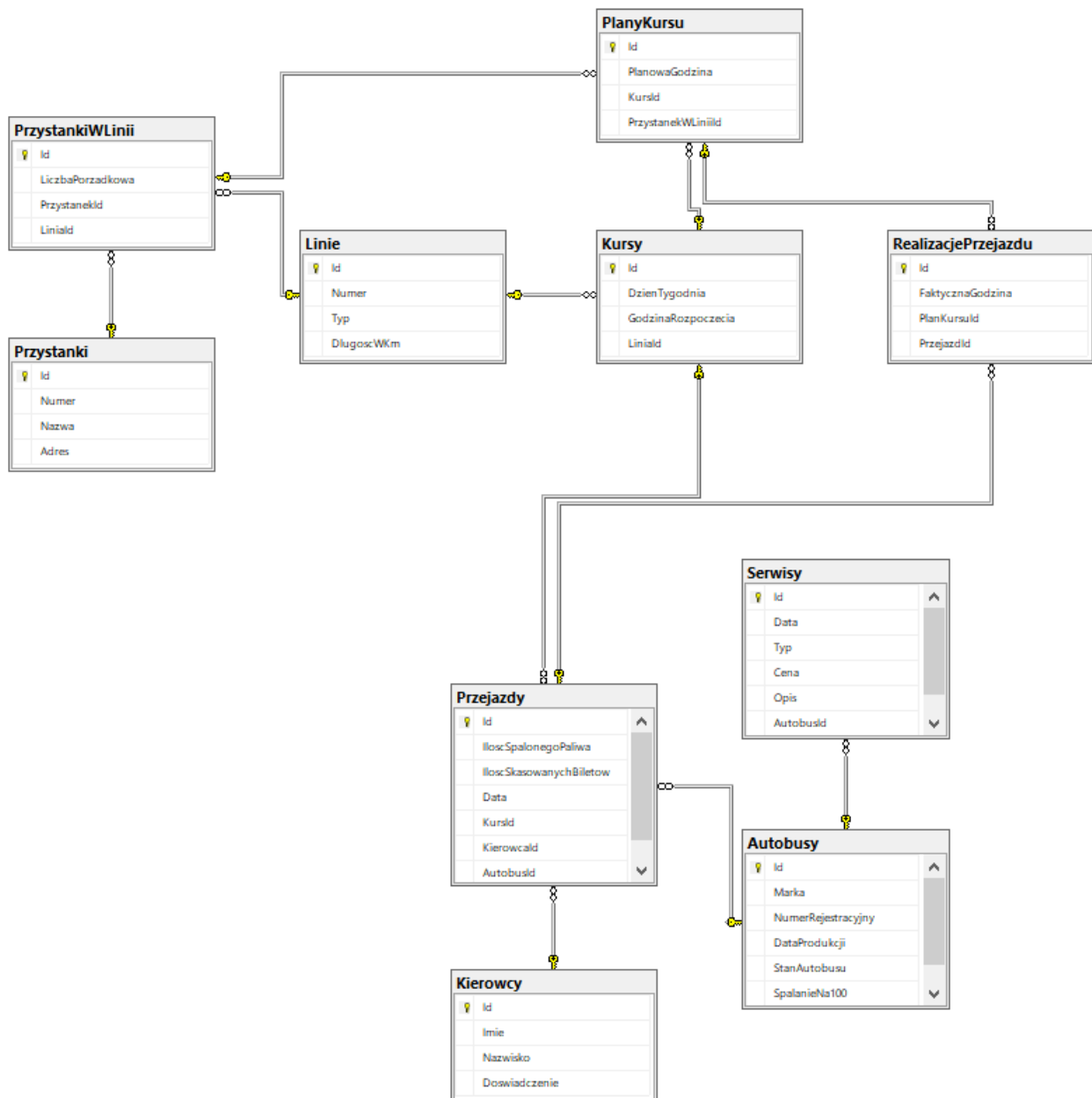
<i>Id planu kursu</i>	Przystanek	Id kursu	Planowa godzina
1	Pierwszy	1	8:05
2	Drugi	1	8:13
3	Trzeci	1	8:22

Ostatnim krokiem jest stworzenie konkretnego przejazdu dla tego kursu:

<i>Id przejazdu</i>	Id kursu	Data	Kierowca	Autobus
1	1	16.06.2022 8:00	{ Jan Kowalski }	{ ABC-DEFG }

Obiekty klas pośrednich (PrzystanekWLinii, RealizacjaPrzejazdu) zostaną stworzone i połączone relacjami automatycznie.

Ostatnim z diagramów jest wygenerowany przez system zarządzania bazą danych diagram przedstawiający tabele wraz z ich relacjami. Ten diagram zawiera dokładne nazwy poszczególnych kolumn (pól) w każdej z tabeli.



Rysunek 3 Diagram relacji między tabelami w bazie danych, wygenerowany automatycznie przez SQL Server Management Studio.

Opis użytej technologii

Aplikacja została napisana w języku C# w wersji frameworku .NET 6.0.

Interfejs użytkownika powstał w technologii WPF (Windows Presentation Foundation) i został napisany w XAML. Dodatkowo, w celach polepszenia go wizualnie, wykorzystano bibliotekę MaterialDesign:

<https://github.com/MaterialDesignInXAML/MaterialDesignInXamlToolkit>

Do komunikacji z bazą danych skorzystano z Entity Framework Core. Framework pomógł mapować rekordy z bazy danych na obiekty w aplikacji, jak również w znaczący sposób uprościł komunikację z bazą danych: <https://github.com/dotnet/efcore>

Do generacji raportów wykorzystano bibliotekę QuestPDF, pozwalającą w łatwy sposób generować pliki PDF bezpośrednio w aplikacji: <https://www.questpdf.com/>

Ponadto, w celu wyświetlania pomocy dla użytkownika wykorzystano bibliotekę CefSharp:
<https://cefsharp.github.io/>

Baza danych

Baza danych jest stworzona z wykorzystaniem systemu Microsoft SQL Server. Zarówno Entity Framework jak i wykorzystane środowisko programistyczne w łatwy sposób współpracują z tego typu bazą danych.

Dla uproszczenia, baza danych jest lokalna i znajduje się na komputerze użytkownika. Więcej informacji można znaleźć w sekcji [Tworzenie bazy danych](#).

Zawartość bazy danych można przeglądać na przykład za pomocą programu Microsoft SQL Server Management Studio (SSMS) – pozwala on na otwarcie lokalnego pliku zawierającego bazę danych i przeglądanie poszczególnych tabeli, ich zawartości, diagramów bazy danych i skryptów.

Skrypt DDL tworzący bazę danych został wygenerowany automatycznie przez Microsoft SQL Server Management Studio na podstawie utworzonych relacji i definicji tabeli.

Wygląd i działanie aplikacji

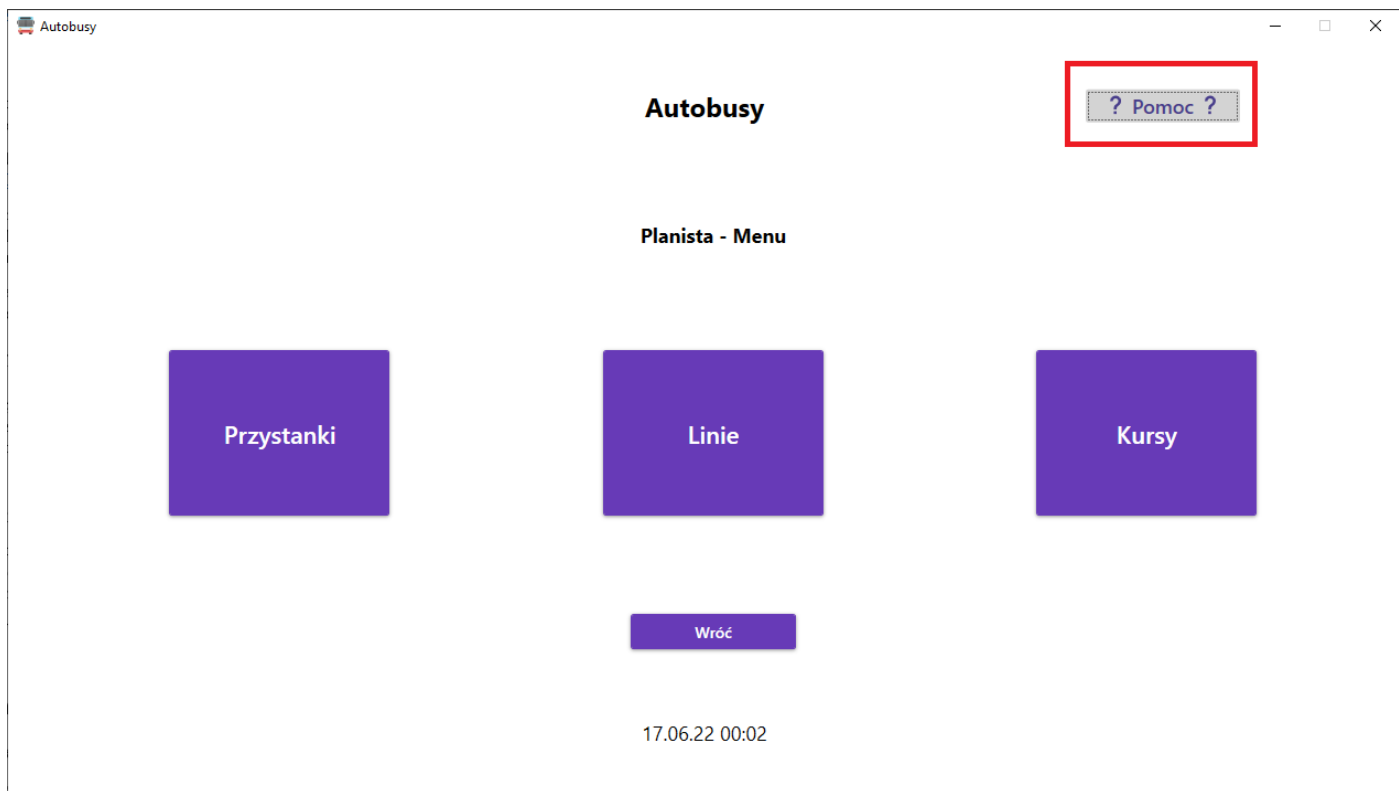
Po uruchomieniu aplikacji znajdujemy się w głównym menu, z którego możemy wybrać czyją odpowiedzialnością chcemy się zajmować – Dyspozytora, Planisty czy Zarządcy flotą. Każda z tych osób ma dostęp do kilku ekranów, na których może modyfikować, dodawać i usuwać odpowiednie elementy.

Funkcjonalności dostępne w aplikacji pokrywają się z tymi opisanymi na diagramie przypadków użycia – sekcja [Szczegółowy opis systemu](#).

Dokładne działanie każdej kontrolki oraz możliwości na każdym z ekranów są opisane w pliku HelpReference, dostępnym wewnątrz aplikacji lub jako załącznik do tego raportu.

Pomoc dla użytkownika

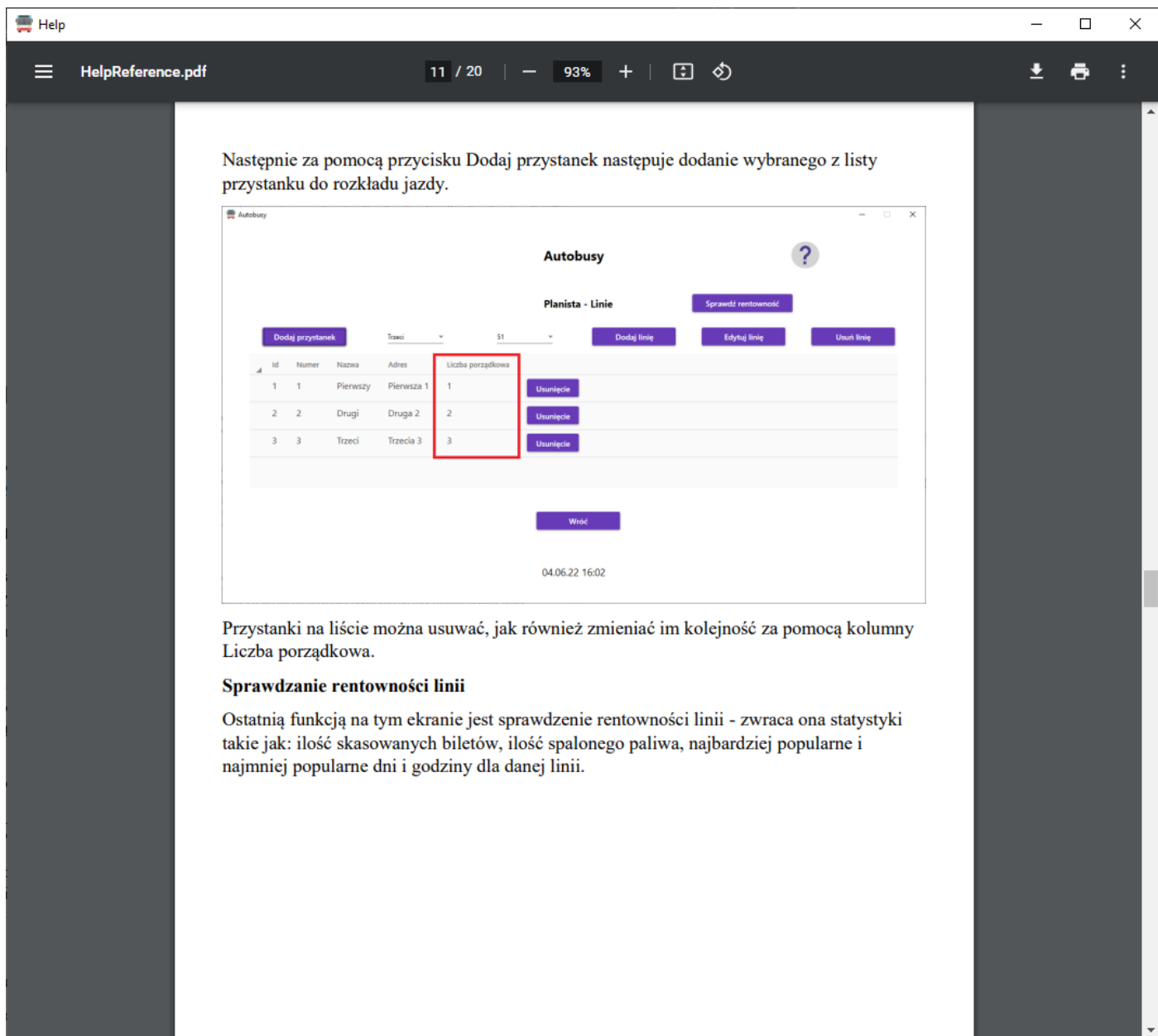
Aplikacja posiada wbudowany system „Help” – pomocy dla użytkownika. W menu głównym, po naciśnięciu przycisku „Pomoc”, wyświetlone zostanie osobne okno zawierające opis aplikacji wraz z poszczególnymi ekranami i opcjami dostępnymi dla każdego z aktorów.



Rysunek 4 Lokalizacja przycisku "Pomoc" w menu głównym.

Wewnątrz pliku pomocy znajduje się wyjaśnienie poszczególnych funkcjonalności na każdym z ekranów; procesu dodawania nowej linii, kursu, przejazdu; jak również objaśnienie korzystania z poszczególnych kontroltek, takich jak `DatePicker` – wybór daty.

Ponieważ plik pomocy zawiera istotne informacje na temat korzystania z aplikacji, jest on dołączony jako załącznik do tego raportu – sekcja [Załącznik. Pomoc dla użytkownika](#).



Rysunek 5 Fragment okna z pomocą dla użytkownika wewnątrz aplikacji.

Generowanie raportów

Poza funkcjonalnością każdego z aktorów, w aplikacji jest możliwość generowania raportów w dowolnej kombinacji trzech dziedzin:

- Informacje o kierowcach – lista kierowców wraz z ilością ich spóźnień, średnim spóźnieniem oraz średnim spalaniem
- Informacje o liniach – szczegółowe informacje o liniach wraz z kursami dla każdej linii oraz ich rentownością na podstawie ilości skasowanych biletów i ilości spalonego paliwa
- Informacje o bazie danych – ogólne informacje dotyczące ilości rekordów w poszczególnych tabelach bazy danych

Autobusy - Raport, 31.05.2022 13.08

Raport zawiera: Informacje o kierowcach Informacje o liniach
Statystyki bazy danych

Id linii: 1

Numer linii: 1

Typ linii: JednaStrona

Długość linii: 12,00

Id	Dzień tygodnia	Godzina	Ilość przejazdów	Ilość sprzedanych biletów	Ilość spalonego paliwa
13	Poniedziałek	12:00	1	133	12

Rysunek 6 Fragment dokumentu zawierającego raport na temat linii.

Kluczowe fragmenty kodu

Dostęp do danych - repozytorium

W celu dostępu do bazy danych w aplikacji, zaimplementowaliśmy wzorzec projektowy **repozytorium**. Zawiera on generyczne operacje CRUD (Create, Read, Update, Delete), które można wykorzystać w dowolny sposób do wykonania działań na bazie danych.

Co ważniejsze, repozytorium enkapsuluje dostęp do bazy danych – w pozostałych miejscach kodu, dostęp ten odbywać się może wyłącznie poprzez repozytorium, nie ma możliwości bezpośredniego nawiązania połączenia z bazą danych i pobierania z niej danych. Daje to dodatkową warstwę zabezpieczeń i pozwala nam kontrolować przepływ danych między aplikacją a bazą danych.

Podstawą repozytorium w projekcie jest interfejs `IGenericRepository<TEntity>`, który definiuje następujące metody:

```

public interface IGenericRepository<TEntity>
{
    TEntity GetById(int id, params Expression<Func<TEntity, object>>[] includes);

    TEntity GetFirst(Func<TEntity, bool> predicate, params Expression<Func<TEntity, object>>[] includes);

    List<TEntity> List(params Expression<Func<TEntity, object>>[] includes);

    List<TEntity> List(Func<TEntity, bool> predicate, params Expression<Func<TEntity, object>>[] includes);

    void Add(TEntity entity);

    void Delete(TEntity entity);

    void Update(TEntity entity);

    void UpdateMany(IEnumerable<TEntity> entities);
}

```

natomiast konkretna klasa, która zapewnia implementację powyższych metod, to `DatabaseRepository`. Klasa ta implementuje również interfejs `IDisposable` w celu automatycznego zamykania połączenia z bazą danych gdy nie jest ono już potrzebne.

Dla przykładu, aby pobrać z bazy danych wszystkie przystanki, możemy napisać:

```

private readonly List<Przystanek> _przystanki;

using (var repo = new DatabaseRepository<Przystanek>(new AutobusyContext()))
{
    _przystanki = repo.List();
}

```

Repozytorium zadziała dla każdego typu danych znajdującego się w bazie danych (w aplikacji wszystkie takie klasy posiadają pole `Id`), a to, co konkretnie i w jaki sposób jest pobierane jest precyzowane za pomocą parametrów funkcji.

Tworzenie bazy danych

Baza danych zostanie automatycznie stworzona przy uruchomieniu aplikacji, jeżeli jeszcze nie istnieje. Jest za to odpowiedzialna funkcja udostępniona przez Entity Framework – `EnsureCreated` – wywoływana przy starcie aplikacji:

```

if (!db.Database.CanConnect())
{
    db.Database.EnsureCreated();
}

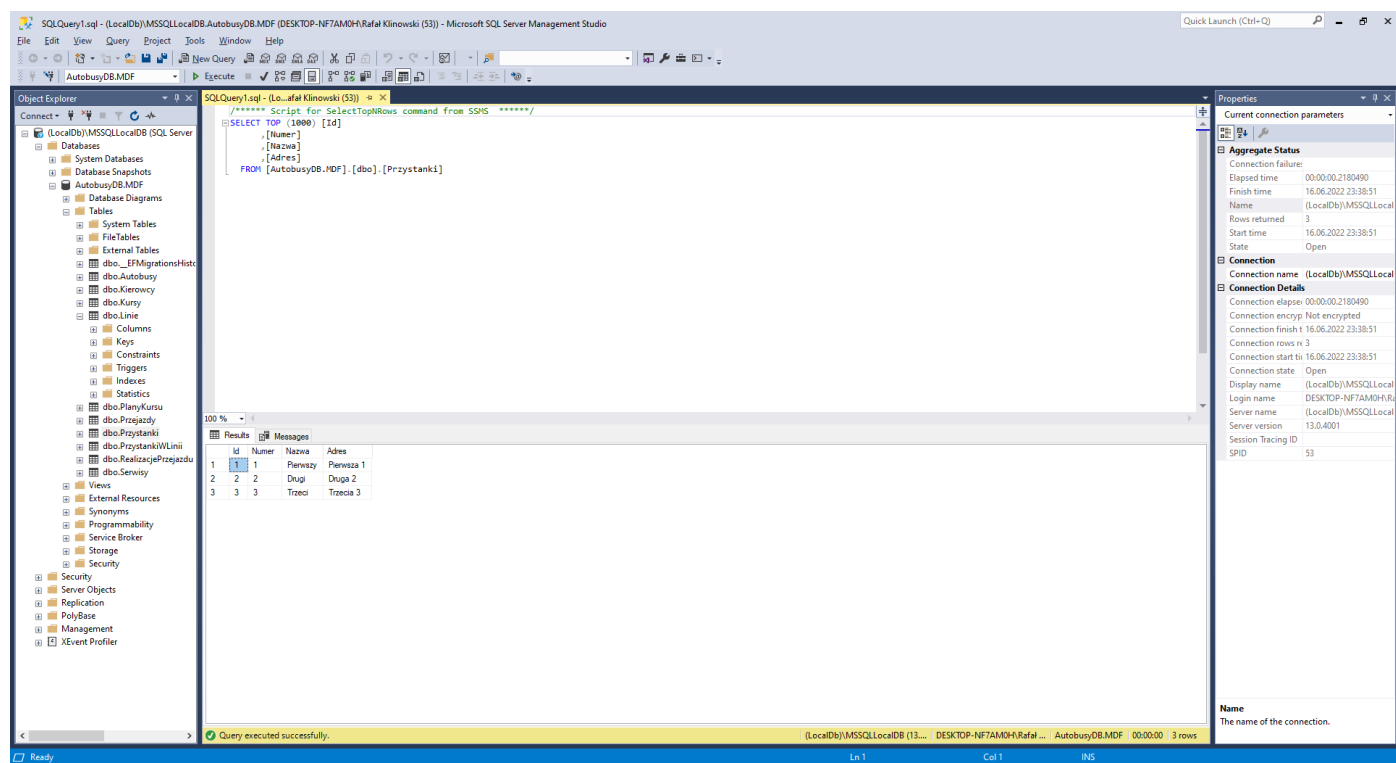
```

Domyślna lokalizacja bazy danych na dysku to:

`C:\Użytkownicy\{Aktualnie zalogowany użytkownik}\AutobusyDB.MDF.mdf`

Plik bazy danych może zostać uruchomiony przez środowisko Microsoft SQL Server Management Studio, w którym można podejrzeć zawartość poszczególnych tabeli, jak również wygenerować diagramy lub skrypty związane z bazą danych. Środowisko umożliwia również bezpośrednie wykonywanie skryptów na

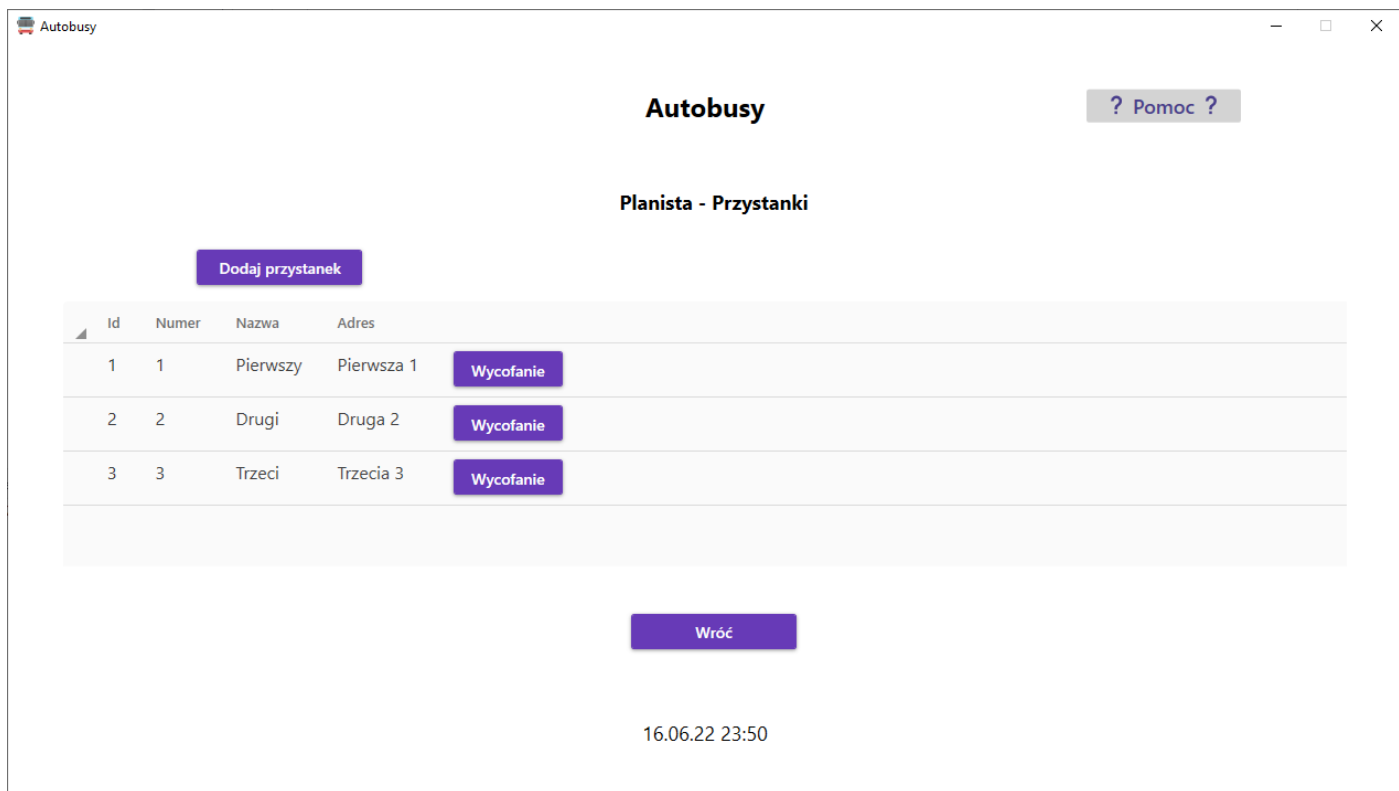
bazie danych, nie jest to jednak zalecane – wszystkie operacje bazodanowe powinny odbywać się przy pomocy aplikacji.



Rysunek 7 Widok wszystkich rekordów w tabeli Przystanki w środowisku SSMS.

Wyświetlanie danych na ekranie

Aplikacja składa się z wielu ekranów, jednak wyświetlanie danych jest przeprowadzane analogicznie na każdym z nich. W tym celu wykorzystujemy kontrolkę **DataGrid**, pozwalającą w łatwy sposób wyświetlić dane w formie tabeli z podziałem na kolumny (poszczególne pola encji), ale również umożliwiającą edycję poszczególnych rekordów w czasie rzeczywistym.



Rysunek 8 Przykład sposobu, w jaki wyświetlane są dane w aplikacji - poszczególne przystanki są rekordami w tabeli. Zarówno Nazwy, jak i Adresy, można edytować bezpośrednio na tym ekranie.

Pierwszym krokiem jest pobranie danych z bazy danych – sposób ten jest opisany w sekcji [Dostęp do danych – repozytorium](#).

Następnie, przy definicji kolumn `DataGrid`'u, przeprowadzamy wiązanie („Binding”) na poszczególne pola klas. Na przykład, kolumna odpowiedzialna za adres przystanku jest zdefiniowana następująco:

```
<DataGridTextColumn Binding="{Binding Adres}" Header="Adres" />
```

Ostatnią częścią jest ustawienie `DataContext` ekranu na pobraną z bazy danych kolekcję:

```
this.DataContext = _przystanki;
```

Połączenie danych w ten sposób pozwala nie tylko na dynamiczne, bezpośrednie wyświetlanie obiektów pobranych z bazy danych, ale również na dowolną modyfikację ich właściwości (poza `Id`, które jest w trybie tylko do odczytu).

Co więcej, wyświetlane na interfejsie dane można sortować za pomocą wiersza nagłówkowego każdej z kolumn w tabeli:

	Id	↓ Numer	Nazwa	Adres	
	3	3	Trzeci	Trzecia 3	Wycofanie
	2	2	Drugi	Druga 2	Wycofanie
	1	1	Pierwszy	Pierwsza 1	Wycofanie

Rysunek 9 Przystanki posortowane malejąco po Numerze.

Przekazywanie danych pomiędzy oknami

W przypadku wykorzystania przez aplikację dodatkowych okien (na przykład przy definicji nowego kursu), dane są przekazywane za pomocą parametrów wejściowych. Po otwarciu takiego okna, główne okno aplikacji przestaje reagować na wejście użytkownika tak długo, aż dodatkowe okno zostanie zamknięte.

Całość kodu źródłowego

Kod źródłowy aplikacji wraz z dokumentacją (diagramami, skryptami) znajduje się w repozytorium GitHub: https://github.com/JakubCisowski/2022_TAB_S14_NITKIEWICZ

Podział obowiązków

Tworząc projekt podzieliliśmy część z obowiązków – oczywiście, każda osoba pracowała nad każdym aspektem projektu, jednak niektórymi zajmowała się bardziej od innych. Poniżej znajduje się lista przedstawiająca podział obowiązków w naszej sekcji:

- Jakub Cisowski – interfejs użytkownika, pliki XAML
- Rafał Klinowski – obsługa zdarzeń, logika interfejsu
- Kacper Nitkiewicz – dostęp do bazy danych, repozytorium
- Paweł Kabza i Michał Chyla – encje, konfiguracja tabeli, połączenie z bazą danych

Podsumowanie, wnioski

Podczas tworzenia relacji między encjami po stronie aplikacji skorzystaliśmy z Fluent API. Jest to alternatywne podejście, udostępniane przez Entity Framework, pozwalające na niejako „ręczną” definicję właściwości każdego z pól wewnątrz klas, jak również relacji między klasami. W pierwszej wersji aplikacji korzystaliśmy z adnotacji, jednak Fluent API udostępnia znacznie większą dowolność i posiada mniej ograniczeń dotyczących tworzenia relacji.

Interfejs użytkownika został napisany w języku XAML, który jest szeroko wykorzystywany przez aplikacje pisane w Windows Presentation Foundation. Pozwala to na jasne rozdzielenie interfejsu od logiki – interfejs znajduje się w plikach XAML, natomiast logika (między innymi: dostęp do bazy danych, zdarzenia odpowiedzialne za naciśnięcie przycisku, itd.) znajduje się w plikach źródłowych C#. XAML pozwala na łatwe definiowanie właściwości każdej z kontrolek, takich jak: rozmiar, położenie, styl, tekst, i tym podobne.

W aplikacji wybraliśmy system zarządzania bazą danych Microsoft SQL Server, gdyż Entity Framework pracuje z nim natywnie, a więc znacząco ułatwiony jest dostęp do bazy danych, jej stworzenie czy połączenie z nią klas znajdujących się w aplikacji. Ponadto, instalacja i konfiguracja systemu jest bardzo prosta, oraz korzystanie z niego daje nam dodatkową korzyść – Linq2Sql. Za pomocą funkcji LINQ

(Language-Integrated Query) języka C# możemy bezpośrednio odwoływać się do tabeli w bazie danych tak, jak gdyby były to kolekcje w aplikacji. W związku z tym implementacja operacji CRUD w repozytorium nie sprawiała żadnych trudności.

W aplikacji, połączenie z bazą danych jest ciągle otwierane i zamykane – nie dopuszczamy nigdy do sytuacji, w której połączenie to jest otwarte przez długi czas, na przykład przez cały cykl życia programu w systemie – ponieważ długotrwałe podtrzymywanie połączenia z bazą danych:

- wykorzystuje zasoby komputera
- uniemożliwia innym wątkom lub aplikacjom w systemie dostęp do zasobu
- komunikacja nie zostanie samoistnie zerwana przez środowisko uruchomieniowe, ponieważ połączenie z bazą danych nie jest zarządzane przez .NET

Źródła

- Ikona aplikacji: Vectors Market - Flaticon (<https://www.flaticon.com/free-icons>)
- ORM: Entity Framework Core - <https://github.com/dotnet/efcore>
- Interfejs użytkownika: Material Design - <https://github.com/MaterialDesignInXAML/MaterialDesignInXamlToolkit>
- Dodatkowe biblioteki: CefSharp.WPF - <https://github.com/cefsharp/CefSharp>
- Dokumentacja Entity Framework Core: <https://www.entityframeworktutorial.net/>
- Narzędzia do tworzenia diagramów: <https://app.diagrams.net/>
<http://www.umletino.com/umletino.html>
- Microsoft SQL Server Management Studio (SSMS): <https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16>

Załącznik. Pomoc dla użytkownika

Pomoc jest dostępna również wewnątrz aplikacji.

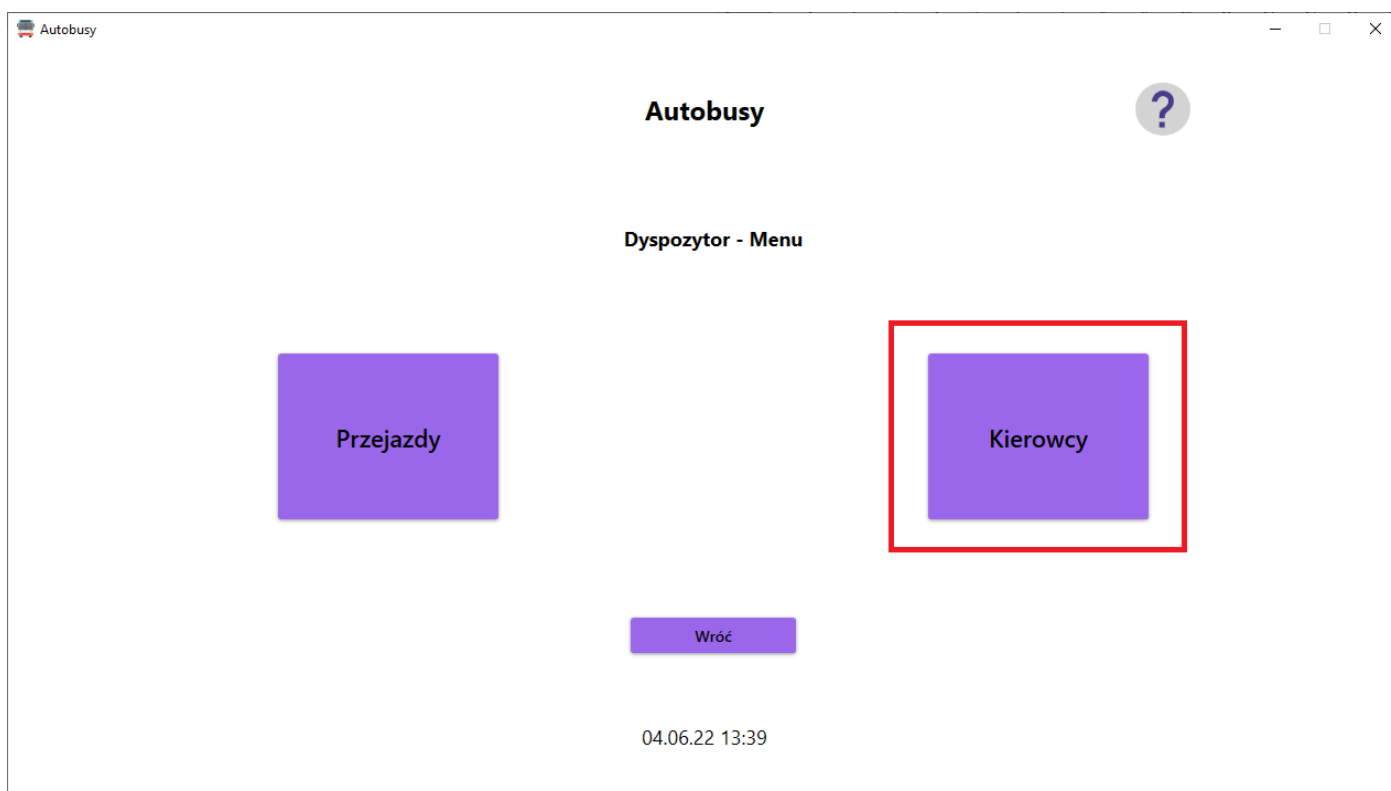
Dyspozytor

Odpowiedzialność dyspozytora jest podzielona na dwie części:

1. Kierowcy - na tym ekranie jest możliwe dodawanie nowych kierowców (przycisk Dodaj kierowcę) oraz uzupełniania danych osobowych zatrudnionych kierowców (imię, nazwisko, doświadczenie). Jest również możliwość sprawdzenia średniego spalania dla każdego z kierowców. Jeżeli osoba ta nie brała jeszcze udziału w żadnych przejazdach, stosowna informacja zostanie wyświetlona na ekranie.

Dodawanie kierowcy

Dodawanie kierowców i modyfikacja ich danych osobowych odbywa się w sekcji Dyspozytor -> Kierowcy.



Kierowcy już znajdujący się w bazie danych są wyświetleni na liście. Po naciśnięciu przycisku „Dodaj kierowcę”, zostanie do listy dodany nowy, pusty rekord.

Autobusy

?

Autobusy

Dyspozytor - Kierowcy

Dodaj kierowcę

Id	Imię	Nazwisko	Doświadczenie		
1	Jan	Kowalski	3	Średnie spalanie	Średnie spóźnienia
2	Jakub	Nowak	12	Średnie spalanie	Średnie spóźnienia
0			0	Średnie spalanie	Średnie spóźnienia

Wróć

04.06.22 13:40

Jest następnie możliwość uzupełnienia danych osobowych – imię, nazwisko, doświadczenie (liczba całkowita).

Początkowo Id (które jest w trybie tylko do odczytu) zostanie ustawione na 0, jednak po wstawieniu kierowcy do bazy danych Id zostanie zaktualizowane.

Sprawdzanie statystyk kierowców

Dla każdego kierowcy jest możliwość sprawdzenia średniego spalania oraz średniego spóźnienia. Jeżeli dany kierowca brał udział w co najmniej jednym przejeździe, wówczas jego średnie spalanie (w litrach na kilometr) oraz średnie spóźnienie (w minutach; jest to średnie spóźnienie na przystanek, a nie całego przejazdu) zostanie wyświetlone w osobnym oknie.

Średni czas spóźnienia kierowcy

Średni czas spóźnienia kierowcy Jan Kowalski wynosi: 14,28 minut.

OK

W przeciwnym wypadku, na ekran zostanie wyświetlona informacja o braku danych.

Średnie spalanie kierowcy

Brak danych o spalaniu dla kierowcy Jan Kowalski.

OK

2. Przejazdy - na tym ekranie jest możliwe tworzenie nowych przejazdów na podstawie kursów. Po wybraniu kursu (reprezentującego teoretyczny rozkład jazdy - układaniem zajmuje się Planista) jest możliwe sprecyzowanie dokładnej daty odbycia, kierowcy, autobusu, a po przejeździe możliwe jest uzupełnienie dodatkowych informacji takich jak: ilość skasowanych biletów, ilość spalonego paliwa. Do stworzenia przejazdu konieczne jest wybranie kursu, w ramach którego dany przejazd się odbywa. Możliwe jest również podejrzenie linii tego kursu, wraz z kolejnością odwiedzania przystanków.

Dodawanie przejazdu

Aby dodać nowy przejazd, konieczne jest najpierw dodanie Linii i Kursu (tym zajmuje się Planista).

W sekcji Dyspozytor -> Przejazdy możliwe jest wybranie linii oraz kursu w ramach linii, dla którego chcemy dodać nowy przejazd. Linia musi zostać wybrana jako pierwsza, aby w drugiej liście rozwijalnej pojawiły się kursy dla tej linii.

The screenshot shows a web application window titled "Autobusy". Inside, the main heading is "Autobusy" with a help icon (question mark) to its right. Below this, the section is labeled "Dyspozytor - Przejazdy". There are two dropdown menus: "Linia" (highlighted with a red box) and "Kurs" (also highlighted with a red box). To the right of these is a purple button labeled "Dodaj przejazd". Below the dropdowns is a table with the following columns: "Id", "Dzień tygodnia", "Data", "Ilość skasowanych biletów", "Ilość spalonego paliwa", "Kierowca", "Autobus", and "Spóźnienia". The table body is currently empty. At the bottom center, there is a purple button labeled "Wróć" and a timestamp "04.06.22 14:12".

Po wybraniu kursu, na liście pojawią się znajdujące się w bazie danych przejazdy. Nowy przejazd można dodać za pomocą przycisku „Dodaj przejazd”.

Autobusy

Autobusy

Dyspozytor - Przejazdy

1

Poniedziałek 00

Dodaj przejazd

Id	Dzień tygodnia	Data	Ilość skasowanych biletów	Ilość spalonego paliwa	Kierowca	Autobus	Spóźnienia
3	Poniedziałek	29.05.2022	133	12	Wybierz	Wybierz	Odnacz

Wróć

04.06.22 14:14

Aktualizowanie parametrów przejazdu

Po stworzeniu przejazdu należy ustawić datę odbycia przejazdu oraz wybrać kierowcę i autobus.

Datę wybieramy za pomocą kontrolki znajdującej się w kolumnie „Data”.

2022

Sun, May 29

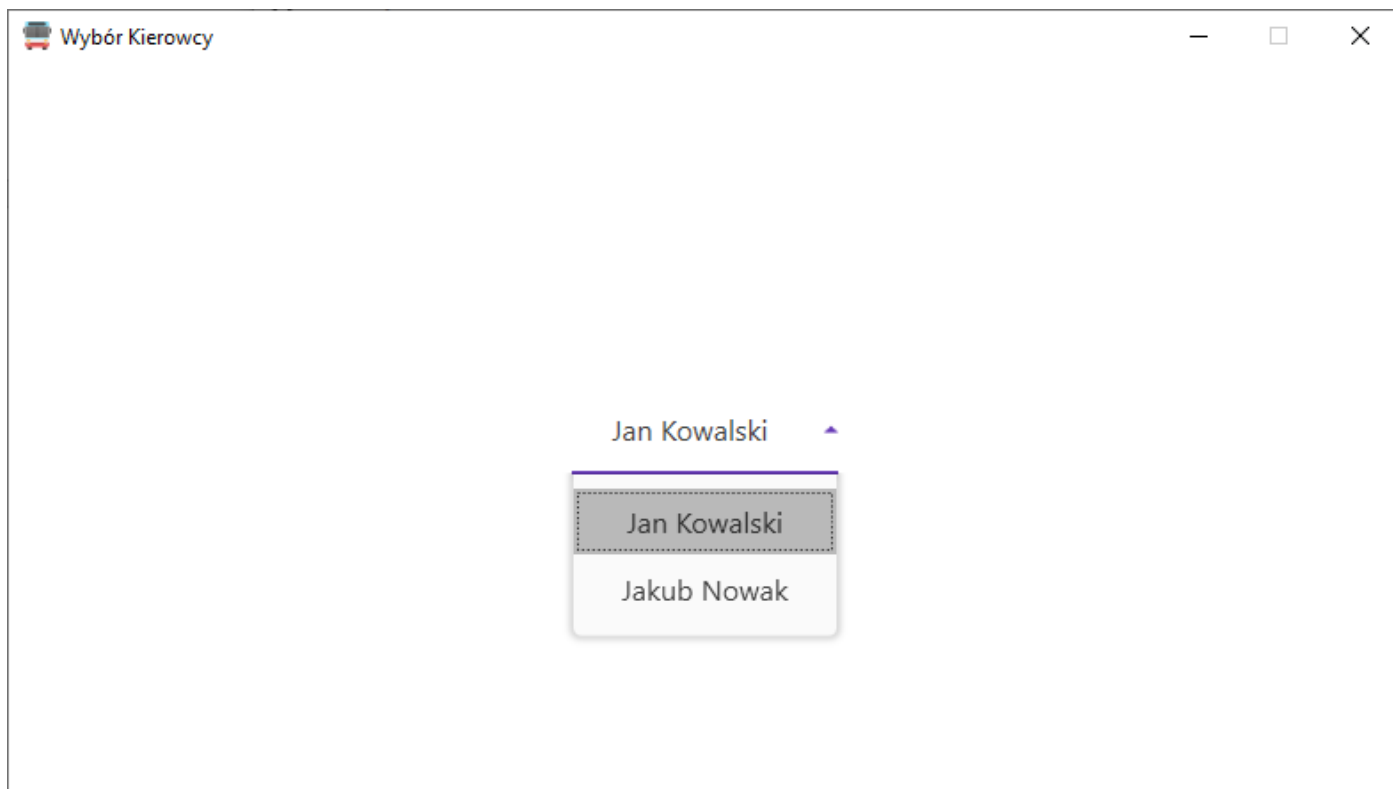
May 2022

< >

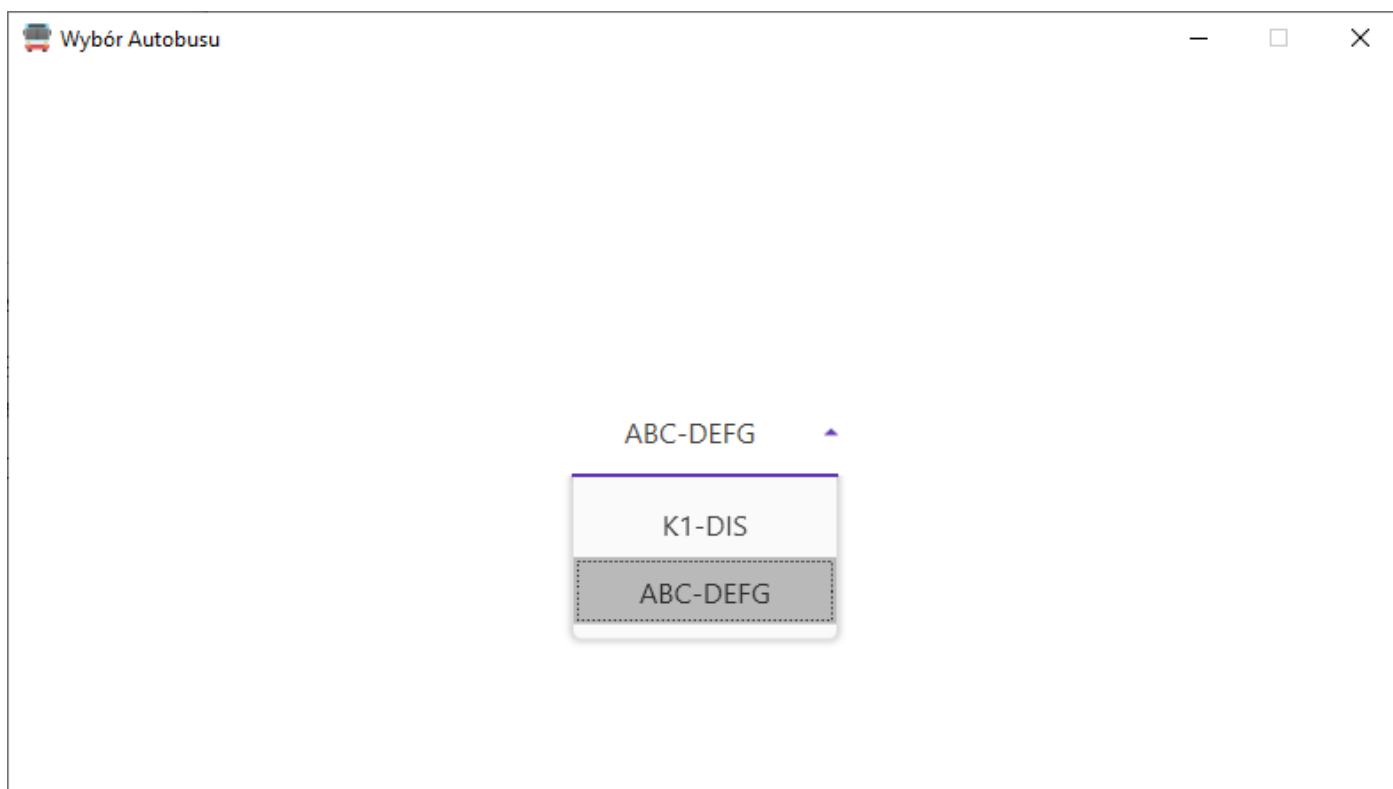
P	W	Ś	C	P	S	N
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

Domyślnie wyświetlane są dni w obecnym miesiącu. Kliknięcie w górną część kontrolki pozwala na wybór miesiąca i/lub roku.

Kierowcę wybieramy za pomocą przycisku „Wybierz” w kolumnie „Kierowca”. Po naciśnięciu go, w osobnym oknie pojawi się lista rozwijalna z kierowcami – należy wybrać jeden rekord z listy i zamknąć okno.




Analogicznie, naciśnięcie przycisku „Wybierz” w kolumnie „Autobus” pozwoli na wybór pojazdu.



Ponadto, jest możliwość wpisania dodatkowych danych dotyczących przejazdu: ilości skasowanych biletów, ilości spalonego paliwa, oraz spóźnień. Dane te mogą być uzupełnione również w późniejszym czasie.

- Ilość skasowanych biletów – liczba całkowita
- Ilość spalonego paliwa – oznacza ilość paliwa spaloną w trakcie całego przejazdu; jest to liczba zmiennoprzecinkowa, którą należy wpisać ze znakiem ‘.’ (kropka) zamiast przecinka, np.: **12.7**.
- Spóźnienia – po naciśnięciu przycisku „Odznacz” w kolumnie „Spóźnienia”, w osobnym oknie pojawi się lista przystanków z planowymi godzinami. Dla każdego przystanku jest możliwa edycja godziny, reprezentującej faktyczny czas przyjazdu kierowcy na przystanek w ramach tego przejazdu


SpóźnieniaWindow

Spóźnienia

Data: 29.05.2022 22:24:03, linia: 1

Id	Numer przystanku	Nazwa przystanku	Faktyczna godzina przyjazdu
0	1	Pierwszy	10:01 PM
0	2	Drugi	10:05 PM
0	3	Trzeci	10:08 PM

Godzinę ustala się za pomocą specjalnej kontrolki. Najpierw należy na zegarze wybrać godzinę, a następnie, po zatwierdzeniu – minuty.

11:58^{PM}

0

5

10

15

20

25

30

35

40

45

50

55

AM

PM

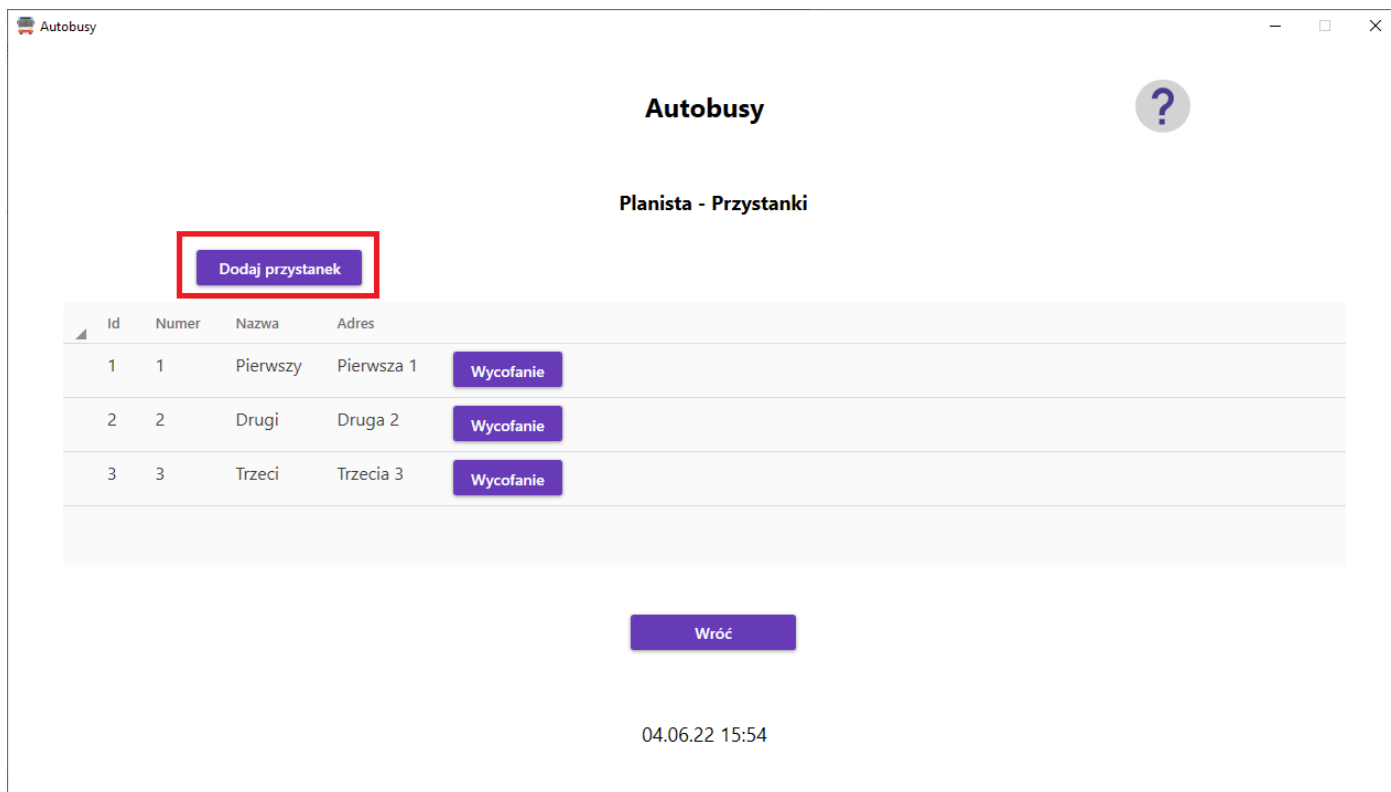
Planista

Odpowiedzialność planisty jest podzielona na trzy części:

1. Przystanki - na tym ekranie jest możliwe dodawanie nowych przystanków, edycja istniejących i ich usuwanie.

Dodawanie przystanku

Za pomocą przycisku Dodaj przystanek, do listy dodawany jest nowy, następnie jest możliwość edycji jego parametrów bezpośrednio na liście.



Usuwanie przystanków

Przycisk Wycofanie usuwa dany przystanek.

-
2. Linie - na tym ekranie definiowana jest linia - podstawowa jednostka podziału.

Tworzenie nowej linii

Pierwszym krokiem jest stworzenie nowej linii za pomocą przycisku Dodaj linię - w osobnym oknie nastąpi wpisanie parametrów takich jak numer linii czy jej typ. Jest możliwość również edycji i usuwania istniejących linii.

LiniaWindow

Id

Numer

Typ linii

Długość

51

JednaStrona

13

Długość linii (w kilometrach) jest liczbą zmiennoprzecinkową i należy ją wpisać ze znakiem ‘.’ (kropka). Id (tylko do odczytu) zostanie automatycznie uzupełnione po zapisaniu do bazy danych.

Dodawanie przystanków do linii

Kolejnym krokiem jest wybór z listy rozwijalnej: najpierw linii (po numerze), a następnie przystanku (po nazwie).

Autobusy

Autobusy

Dodaj przystanek

Przystanek

Pierwszy

Linia

51

Planista - Linie

Dodaj linię

Edytuj linię

Usuń linię

Sprawdź rentowność

Id	Numer	Nazwa	Adres	Liczba porządkowa
----	-------	-------	-------	-------------------

Wróć

04.06.22 16:01

Następnie za pomocą przycisku Dodaj przystanek następuje dodanie wybranego z listy przystanku do rozkładu jazdy.

RentownoscWindow				
Ilość kursów	Ilość spalonego paliwa	Ilość skasowanych biletów	Popularne dni	Niepopularne dni
0	Brak danych	Brak danych	Brak danych	Brak danych

3. Kursy - na tym ekranie definiowane są kursy na podstawie linii, reprezentujące rozkład jazdy autobusu.

Dodawanie kursu

Na początku należy z listy rozwijalnej wybrać linię (po numerze linii). Po wciśnięciu przycisku Dodaj kurs do listy zostanie dodany nowy kurs - należy wybrać dzień tygodnia (NIE konkretną datę – ta informacja znajduje się w Przejeździe) i godzinę rozpoczęcia.

Autobusy

?

Planista - Kursy

Dodaj kurs

1

Numer linii

Id	Dzień tygodnia	Godzina		
13	Poniedziałek	12:00 AM	Edycja	Usunięcie

Wróć

04.06.22 16:10

Definiowanie godzin przyjazdu na poszczególne przystanki

Po wciśnięciu przycisku Edycja, wyświetli się nowe okno, w którym widnieje lista zawierająca przystanki w takiej kolejności, jaka była ustalona w panelu Linie. Dla każdego przystanku z listy można ustawić godzinę, o której autobus ma pojawić się na danym przystanku.

Definiowanie Kursu

Planista - Kursy

Dodaj plan kursu

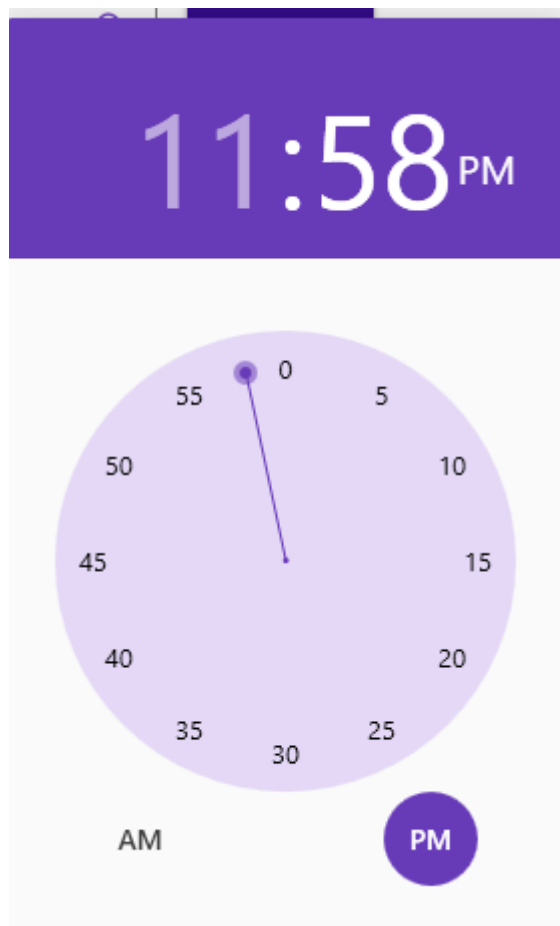
13 - Poniedziałek - 00:00:00

Id	Numer przystanku	Nazwa przystanku	Godzina	
3	1	Pierwszy	10:01 PM	Usunięcie
4	2	Drugi	10:05 PM	Usunięcie
5	3	Trzeci	10:08 PM	Usunięcie

Na ekranie tym widnieją informacje o kursie, dla którego aktualnie definiowane są przystanki i godziny – identyfikator kursu, dzień tygodnia i godzina rozpoczęcia.

Kolumna „Id” reprezentuje identyfikator danego planu kursu, tylko do odczytu – nie Id przystanku.

Godzinę ustala się za pomocą specjalnej kontrolki. Najpierw należy na zegarze wybrać godzinę, a następnie, po zatwierdzeniu – minuty.



Jest tu również możliwość usuwania przystanków lub dodawania nowych – w ten sposób niektóre kursy mają inną trasę niż wynikałoby z definicji linii.

Usuwanie kursu

Przycisk Usunięcie na głównym ekranie usuwa dany kurs wraz z całym rozkładem jazdy dla niego.

Zarządca Flotą

Odpowiedzialnością Zarządcy Flotą jest kontrola stanu autobusów, przyjmowanie nowych do taboru oraz oddawanie ich do serwisu.

Dodawanie autobusów

Za pomocą przycisku Dodaj autobus jest możliwe dodanie do listy nowego pojazdu - konieczne jest następnie wypełnienie wszystkich kolumn reprezentujących parametry autobusu.

Autobusy

Zarządca flotą

Dodaj autobus

Serwisy

Id	Marka	Numer Rejestracyjny	Data Produkcji	Spalanie na 100	Stan Autobusu		
1	Audi	K1-DIS	01.01.2020	9.70	Dobry	Wycofanie	Serwis
2	Ford	ABC-DEFG	29.05.2014	11.00	Dobry	Wycofanie	Serwis

Wróć

04.06.22 15:30

WAŻNE: spalanie pojazdu na 100km, będące liczbą zmiennoprzecinkową, należy podać używając kropki a nie przecinka! (na przykład: 9.7).

Stan autobusu można w każdym momencie zmienić korzystając z listy rozwijalnej.

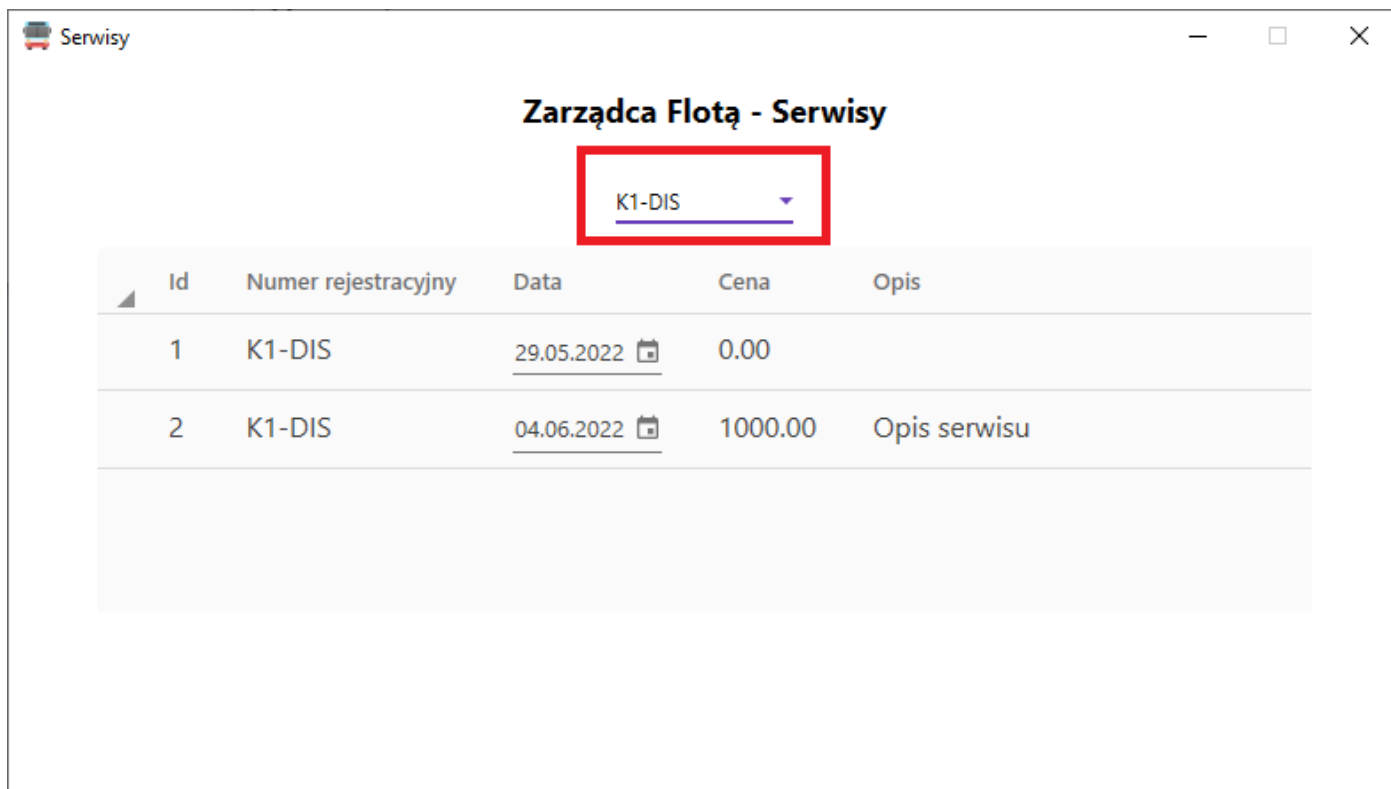
Usuwanie autobusów

Za pomocą przycisku Wycofanie możliwe jest usunięcie wybranego pojazdu.

Serwisy

Przycisk Serwis służy do oddania autobusu do serwisu - w ten sposób, stan pojazdu automatycznie zmienia się na 'W serwisie', a w bazie danych zostanie utworzony nowy Serwis z danym pojazdem.

Parametry serwisu można modyfikować w osobnym oknie, po naciśnięciu przycisku „Serwisy”. Z listy rozwijalnej należy wybrać interesujący nas pojazd, aby przeglądać jego serwisy.



Ponownie, cena naprawy jest liczbą zmiennoprzecinkową i musi zostać podana ze znakiem ‘.’ (kropka).

Generacja raportów

Raporty można wygenerować z poziomu każdego użytkownika.

Na ekranie głównym aplikacji należy kliknąć przycisk „Generacja raportów”.



W osobnym oknie możemy wybrać, jakie informacje mają zostać zawarte w raporcie.

☒ Informacje o kierowcach☐ Informacje o liniach☒ Informacje o bazie danych**Generuj!**

Można wybrać dowolną (lub wszystkie) z trzech opcji:

- Informacje o kierowcach – tabelaryczne zestawienie zatrudnionych kierowców, wraz z ilością ich przejazdów, średnim spalaniem (na kilometr trasy), ilością spóźnień oraz średnim spóźnieniem (w minutach)
- Informacje o liniach – tabelaryczne zestawienie wszystkich linii wraz z kursami w ramach każdej linii. Zostaną wyświetlone podstawowe informacje o linii (numer, typ, długość) oraz dla każdego kursu: dzień tygodnia, godzina, ilość przejazdów, ilość sprzedanych biletów oraz ilość spalonego paliwa (w ramach wszystkich przejazdów w ramach danego kursu)
- Informacje o bazie danych – statystyki dotyczące ilości rekordów w każdej z tabeli bazy danych

Raport zostanie wygenerowany w formie pliku PDF, w folderze, w którym znajduje się aplikacja. Format nazwy będzie następujący: „Autobusy raport - (data) (godzina).pdf”.

Baza danych

Aplikacja korzysta z lokalnej bazy danych SQL Server LocalDB.

Tworzenie i lokalizacja bazy danych

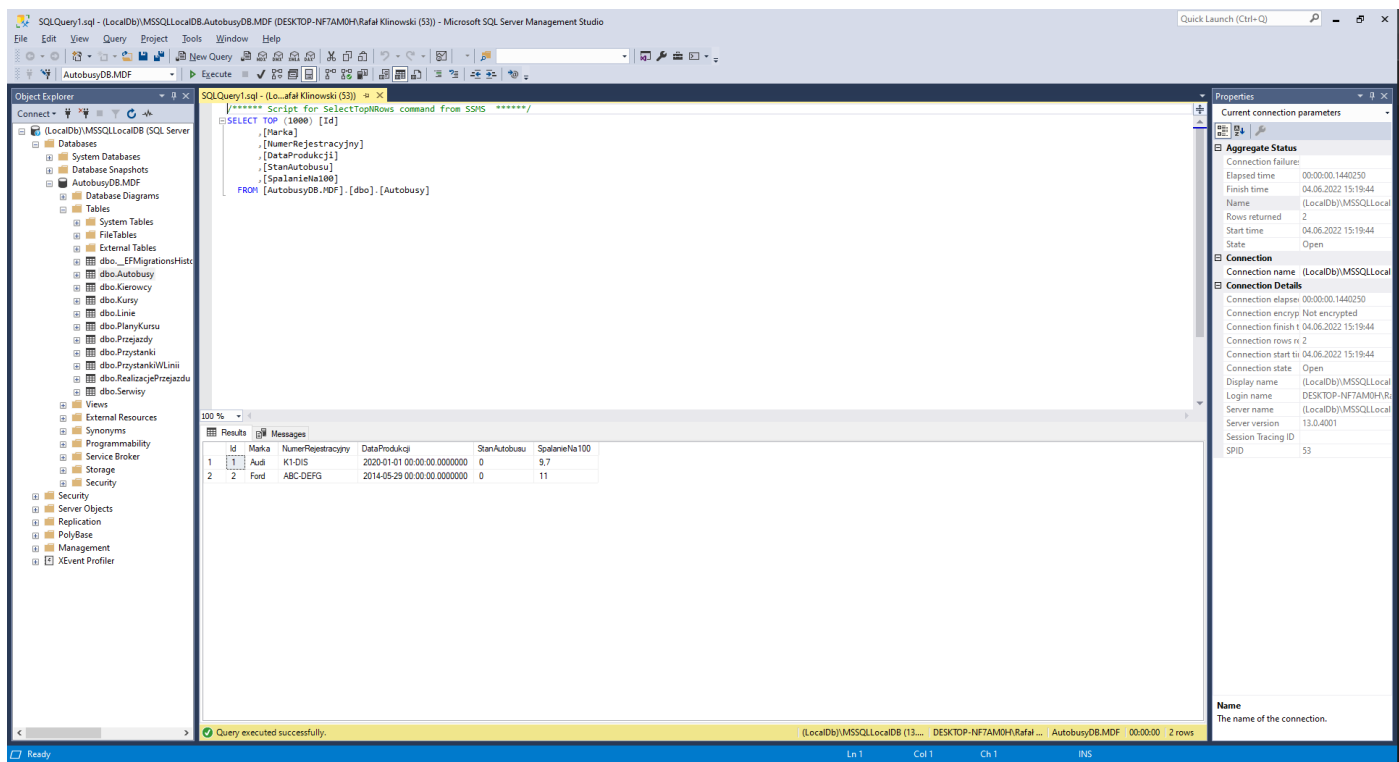
Przy pierwszym uruchomieniu aplikacji, jeżeli baza danych nie istnieje, zostanie automatycznie stworzona.

Domyślna ścieżka: C:\Użytkownicy\ (Aktualnie zalogowany użytkownik) \AutobusyDB.MDF.mdf.

Zawartość bazy danych

Jest możliwość podejrzenia zawartości bazy danych za pomocą programu SSMS (SQL Server Management Studio) od firmy Microsoft. Po połączeniu z lokalną bazą danych i wybraniu z listy pliku AutobusyDB.MDF.mdf, możliwe jest zobaczenie schematu, poszczególnych tabeli oraz ich zawartości.

Zrzut ekranu przedstawiający tabele w bazie danych oraz przykładową zawartość jednej z nich:



Dostęp do bazy danych

Dostęp do bazy danych odbywa się w aplikacji za pomocą frameworku Entity Framework Core oraz wzorca projektowego Repozytorium.

Entity Framework Core

Jest to ORM, który łączy się z lokalną bazą danych oraz automatycznie mapuje encje z bazy danych na obiekty .NET. Framework znacząco ułatwia komunikację z bazą danych i poruszanie się po tabelach.

Repozytorium

Jest to wzorec projektowy używany najczęściej przy komunikacji z zewnętrznymi zasobami, m.in. bazami danych. Repozytorium to generyczna klasa zawierająca metody reprezentujące operacje CRUD:

- Get
- List
- Add
- Update
- Remove

Za pomocą tych metod, po stworzeniu obiektu i przekazaniu odpowiedniego typu jako parametr, możliwe jest odwołanie się do bazy danych, a jednocześnie wywołanie dodatkowych funkcji bezpośrednio przy operacji odczytu/zapisu. Jest to swego rodzaju opakowanie dostępu do bazy danych, zwiększające bezpieczeństwo i zmniejszające ilość wykonywanych, generycznych operacji.