POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

Projekt z Baz danych

Bazodanowy system rezerwacji biletów dla przewoźnika kolejowego

AUTORZY:

Sebastian Łągiewski
Radosław Mucha
Łukasz Zatorski

PROWADZĄCY ZAJĘCIA:

Dr inż. Robert Wójcik, W4/K-9

OCENA PRACY:

Wrocław 2018

SPIS TREŚCI

SPIS RYSUNKÓW

1. Wstęp

Prezentowany projekt baz danych został opracowany w trakcie realizowania zajęć projektowych z przedmiotu Bazy Danych 2. W trakcie pracy projektowej stworzona została przykładowa baza danych przewoźnika kolejowego oraz towarzysząca jej aplikacja do obsługi. Przechowywane są w tej bazie informacje na temat rezerwacji miejsc w wagonach, kursów pociągów oraz rozkładu jazdy.

1.1. Cel projektu

Celem projektu z baz danych było stworzenie oraz implementacja aplikacji bazodanowej umożliwiającej dostęp do lokalnej bazy danych przeznaczonej do obsługi systemu rezerwacji przewoźnika kolejowego.

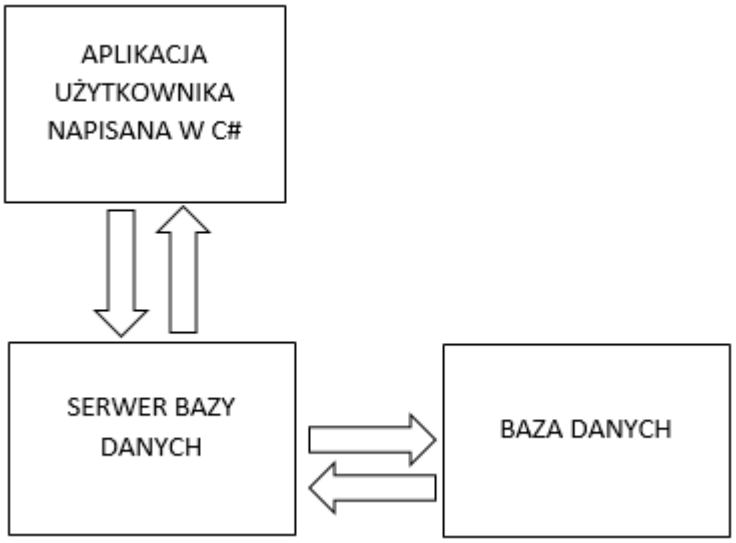
1.2. Zakres projektu

System rezerwacji będzie obsługiwany za pośrednictwem serwera bazy danych Microsoft SQL Server. Interfejs użytkownika zostanie zrealizowany w postaci aplikacji obiektowej napisanej w języku C# i technologii .Net.

2. Analiza wymagań

System umożliwiać będzie zarządzanie bazą rezerwacji przewoźnika kolejowego. Dostęp do bazy będzie posiadać sprzedawca biletów. Sprzedawca będzie miał możliwość przeglądania rezerwacji, w szczególności: stację początkową i końcową, numer wagonu i miejsca, numer pociągu, datę oraz godzinę wyjazdu. Sprzedawca w każdej chwili będzie mógł sprawdzić także dane klienta. W takim przypadku baza danych będzie zawierać także numer id klienta oraz id transakcji, datę i godzinę dokonania zakupu.

2.1. Opis działania i schemat logiczny systemu



Rys. 1 Schemat komunikacji i struktura systemu

Baza danych łączy się dwukierunkowo w serwerem baz danych, a aplikacja porozumiewa się tak samo z serwerem.

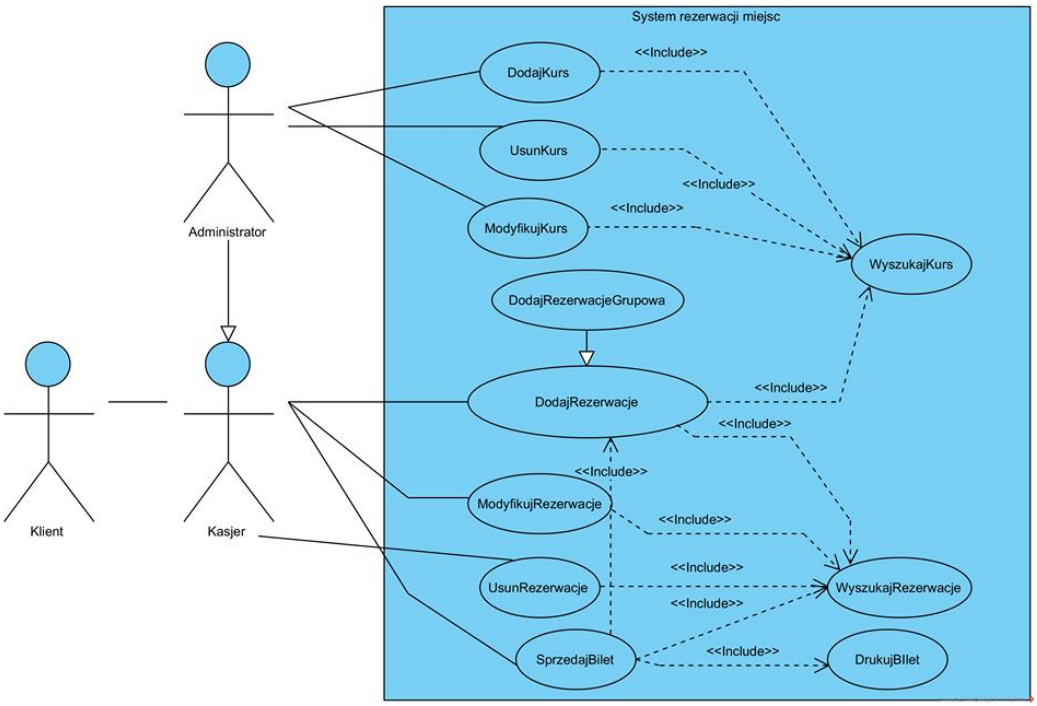
2.2. Wymagania funkcjonalne

Projekt posiada następujące wymagania funkcjonalne:

- System zapewnia kompleksową obsługę klienta
- Kasjer może uzyskać rezerwację
- Kasjer może sprawdzić rezerwację danego klienta
- Kasjer może sprawdzić dostępne pociągi
- Kasjer może sprawdzić, czy są wolne miejsca w pociągu
- Kasjer może założyć nowe konto klienta w systemie
- Administrator może przeglądać rezerwacje pasażerów
- Administrator może wprowadzić do systemu nowe pociągi

2.2.1. Diagram przypadków użycia

Diagram przypadków użycia dla tego projektu został stworzony w programie Visual Paradigm.



Rys. 2 Diagram przypadków użycia

2.2.2. Scenariusze wybranych przypadków użycia

Zarzadzanie kursami pociągów

PU DodajKurs
Opis
Cel: Dodanie nowego kursu.
WS: Brak
WK: Dodanie kursu o podanych atrybutach obowiązkowych: nr. pociągu, cena biletu, ilość miejsc, miejsce początkowe, miejsce docelowe, godzina odjazdu, godzina przybycia do stacji końcowej.
Przebieg:
1. Należy podać wymienione wyżej atrybuty kursu
2. Należy wywołać ***PU WyszukajKurs***, sprawdzić czy kurs o podanych danych już istnieje. Jeśli istnieje,
należy zakończyć PU. W przeciwnym wypadku należy dodać kurs do BD.

PU ModyfikujKurs
Opis
Cel: Modyfikacja danego kursu.
WS: Brak

WK: Podanie kursu o danym identyfikatorze.
Przebieg:
1. Należy podać ID kursu.
2. Należy wywołać ***PU WyszukajKurs***, sprawdzić czy kurs o podanym ID istnieje. Jeśli istnieje, można zmodyfikować dany kurs lub wywołać ***PU UsunKurs*** w celu usunięcia kursu, w przeciwnym wypadku należy zakończyć PU.

PU UsunKurs
Opis
Cel: Usunięcie kursu z BD
WS: Jest wywoływany z ***PU ModyfikujKurs***, jeśli dany kurs zostanie znaleziony za pomocą ***PU WyszukajKurs.***
WK: Usunięcie kursu o podanych do PU danych.
Przebieg:
1. Kurs o podanych danych zostaje usunięty.

PU WyszukajKurs
Opis
Cel: Poszukiwanie kursu.
WS: Brak
WK: Podanie kursu o zadanych danych lub komunikat o braku takowego kursu.
Przebieg:
1. Szukanie przebiega według danych podanych do przypadku użycia.
2. Jeśli istnieje kurs o podanych danych, jest on zwracany, w przeciwnym wypadku zwracana jest informacja o jego braku.

Zarzadzanie rezerwacjami

PU DodajRezerwacje
Opis
Cel: Dodanie nowej rezerwacji.
WS: Brak
WK: Dodanie rezerwacji o podanych atrybutach: kurs, data, miejsce, dane klienta.
Przebieg:
1. Należy podać atrybuty rezerwacji.
2. Należy wywołać ***PU WyszukajKurs***, sprawdzic czy kurs o podanym ID istnieje.
3. Jeśli istnieje, należy wywołać ***PU WyszukajRezerwacje***, sprawdzić czy na dane miejsce nie ma już rezerwacji.
4. Jeśli warunki są spełnione, dodajemy nową rezerwację, w przeciwnym wypadku należy zakończyć PU.

PU DodajRezerwacjeGrupowa
Opis
Cel: Dodanie kilku rezerwacji na jeden kurs.
WS: Brak

WK: Dodanie rezerwacji o podanych atrybutach: kurs, data, miejsce, dane klienta.
Przebieg:
1. Należy podać atrybuty rezerwacji - najpierw kurs, data i dane klienta zamawiającego; następnie wybrać odpowiednią ilość miejsc.
2. Należy wywołać ***PU WyszukajKurs***, sprawdzić czy kurs o podanym konkretnym numerze ID istnieje.
3. Jeśli istnieje, należy wywołać ***PU WyszukajRezerwacje***, sprawdzić czy na dane miejsca nie ma już rezerwacji.
4. Jeśli warunki są spełnione, dodajemy nową rezerwację, w przeciwnym wypadku należy zakończyć PU.

PU ModyfikujRezerwacje
Opis
Cel: Modyfikacja danej rezerwacji.
WS: Brak
WK: Modyfikacja lub usunięcie rezerwacji o danym identyfikatorze.
Przebieg:
1. Podanie ID rezerwacji.
2. Należy wywołać ***PU WyszukajRezerwacje***, sprawdzić czy rezerwacja o podanym ID istnieje.
3. Jeśli istnieje, można zmodyfikować daną rezerwacje lub wywołać ***PU UsunRezerwacje*** w celu usunięcia rezerwacji, w przeciwnym wypadku należy zakończyć PU.

PU UsunRezerwacje
Opis
Cel: Usuniecie rezerwacji z BD.
WS: Jest wywoływany z ***PU ModyfikujRezerwacje***, jesli dana rezerwacja zostanie znaleziona za pomocą ***PU WyszukajRezerwacje.***
WK: Usunięcie rezerwacji o podanych do PU danych.
Przebieg:
1. Rezerwacja o podanych danych zostaje usunięta.

PU WyszukajRezerwacje
Opis
Cel: Poszukiwanie rezerwacji.
WS: Brak
WK: Podanie rezerwacji o zadanych danych lub komunikat o braku takowej rezerwacji.
Przebieg:
1. Szukanie przebiega według danych podanych do przypadku użycia (nr. rezerwacji lub imienia i nazwiska osoby rezerwującej).
2. Jeśli istnieje rezerwacja o podanych danych, zwracana jest rezerwacja, w przeciwnym wypadku zwracana jest informacja o jej braku.

Sprzedaż oraz drukowanie biletów

PU SprzedajBilet
Opis
Cel: Sprzedaż biletu.
WS: Brak
WK: Dodanie nowego biletu do rezerwacji.
Przebieg:
1. Należy podać dane potrzebne do zakupu biletu: imię i nazwisko kupującego oraz jeśli potrzeba numer miejsca, id kursu, nr. rezerwacji.
2. Jeśli dokonano wcześniejszej rezerwacji to 3., w przeciwnym wypadku wywołujemy ***PU DodajRezerwacje.***
3. Należy wywołać ***PU WyszukajRezerwacje*** w celu jej identyfikacji.
4. Jeśli się ona powiedzie wywołujemy ***PU DrukujBilet***, w przeciwnym wypadku należy zakończyć PU.

PU DrukujBilet
Opis
Cel: Generacja oraz drukowanie biletu.
WS: Jest wywoływany z ***PU SprzedajBilet***, jeśli rezerwacja zostanie potwierdzona przez ***PU WyszukajRezerwacje.***
WK: Wygenerowanie biletu.
Przebieg:
1. Generacja biletu.

2.3. Wymagania niefunkcjonalne

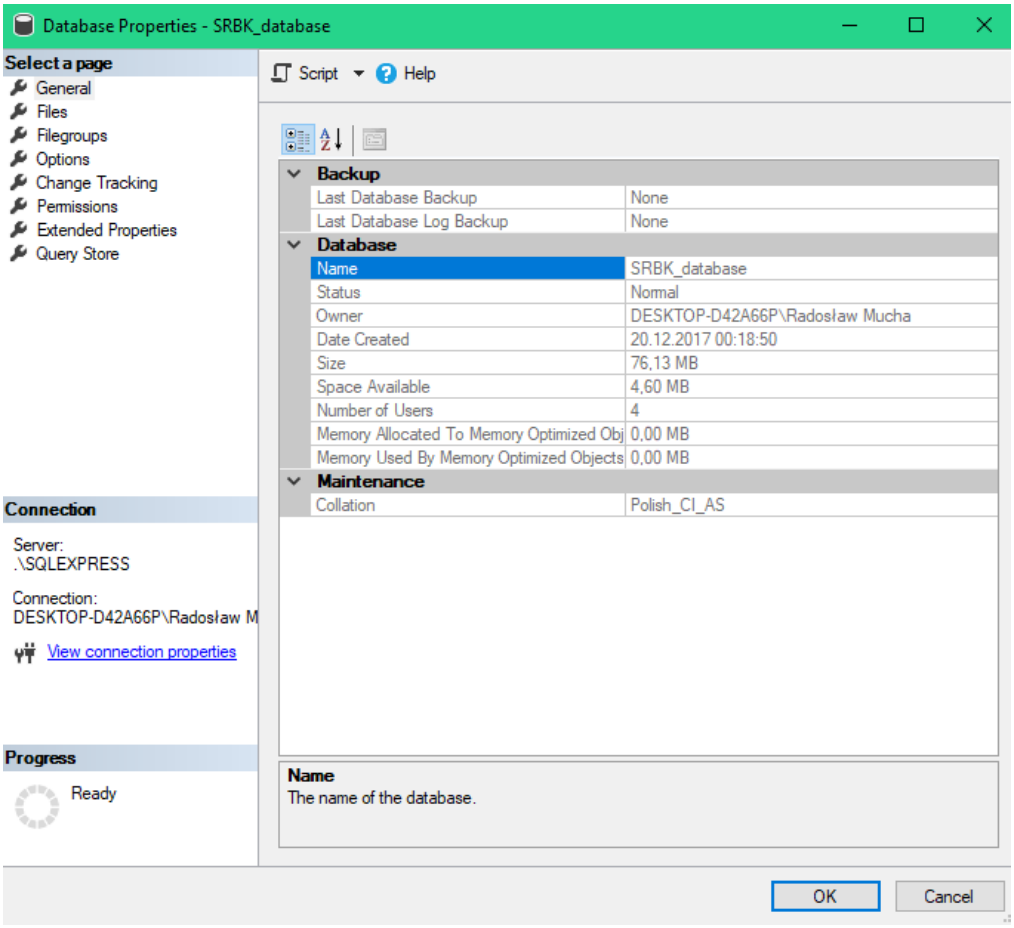
Projekt posiada następujące wymagania niefunkcjonalne:
- Aplikacja powinna mieć nowoczesny, spójny oraz intuicyjny interfejs.
- Aplikacja powinna zapewnić pełne bezpieczeństwo danych.
- Przejrzysty interfejs wyboru miejsca (do rezerwacji)
- Rozdzielenie aplikacji na dwa poziomy (poziom rezerwacji i poziom zarządzania)
- Szybkość i prostota działania
- Różne poziomy uprawnień (klient, administrator)

2.3.1. Wykorzystywane technologie i narzędzia

System rezerwacji jest obsługiwany za pośrednictwem serwera bazy danych Microsoft SQL Server i aplikacji serwera Microsoft SQL Express 2014. Interfejs użytkownika zostanie zrealizowany w postaci aplikacji obiektowej napisanej w języku C# za pomocą oprogramowania Microsoft Visual Studio 2017.

2.3.2. Wymagania dotyczące rozmiaru bazy danych

Baza danych wraz z wprowadzeniem dwóch kursów oraz danych dziesięciu pasażerów potrzebuje około 80MB. Wraz z dodawaniem kolejnych klientów systemu oraz kursów rozmiar bazy danych będzie ulegał stopniowemu zwiększeniu.



Rys. 3 Właściwości bazy danych

2.3.3. Wymagania dotyczące bezpieczeństwa systemu

Dostęp do bazy danych musi być odpowiednio zabezpieczony. Ma to na celu ograniczenie dostępu do wrażliwych danych osobowych klientów, bazy danych rezerwacji oraz pojedynczych kursów. Dostęp do zarządzania kursami będzie posiadał tylko administrator dzięki indywidualnemu loginowi oraz hasłu. Hasło będzie posiadał również kasjer.

3. Projekt systemu

Punkt sprawozdania „Projekt systemu” będzie zawierał opis słowny oraz diagramy pomocne do opisania projektu.

3.1. Projekt bazy danych

W projekcie zostały zaimplementowane następujące tabele wraz z ograniczeniami.

CUSTOMERS - Tabela przechowuje dane klientów.

- ID (*int(4)*) – przechowuje id klienta. Nie może być ono puste.
- NAME (*varchar(50)*) – przechowuje imię użytkownika o maksymalnej długości 50 znaków. Pole to nie może być puste. Nie można stosować znaków specjalnych oraz cyfr.
- SURNAME (*varchar(50)*) – przechowuje nazwisko użytkownika o maksymalnej długości 50 znaków. Pole to nie może być puste. Nie można stosować znaków specjalnych oprócz „-” oraz cyfr.
- ADDRESS (*varchar(70)*) – przechowuje ulicę, nr domu oraz mieszkania, jeśli jest taka potrzeba. Jego maksymalna długość to 70 znaków. Można stosować znaki specjalne „ . ”, „ / ”, „ - ”. Pole to nie może być puste.
- CITY (*varchar(50)*) – przechowuje miejscowość użytkownika o maksymalnej długości 50 znaków. Można stosować jeden znak specjalny: „ – ”. Pole to nie może być puste.
- ZIP_CODE (*varchar(6)*) – przechowuje kod pocztowy użytkownika w formacie x1x2- x3x4x5, gdzie xi reprezentuje liczbę. Pole to nie może być puste.
- PHONE_NUMBER (*varchar(11)*) – przechowuje numer telefonu w postaci łańcucha znaków. Jego długość równa jest 11. Pole to może przechowywać tylko cyfry (z 2 liczbowym numerem kierunkowym na początku). Pole to nie może być puste.
- EMAIL (*varchar(50)*) – zawiera email użytkownika. Można stosować znaki specjalne przed znakiem „ @ ”, zaś po nim można stosować „ –” oraz musimy przynajmniej raz użyć „ .”. Pole to nie może być puste.

STATIONS - Tabela przechowuje listę stacji.

- ID (*int*) – przechowuje id stacji. Nie może być ono puste.

• NAME (*varchar(50)*) – przechowuje nazwę stacji, reprezentującą jej lokalizację. Może zawierać maksymalnie 50 znaków, w tym litery oraz znak ‘-‘. Nie może być puste oraz musi być unikalne.

VISITS - Tabela ta przechowuje dane odwiedzin danego pociągu dla danego miasta.
• ID (*int*) – przechowuje id odjazdu. Nie może być ono puste.
• STATION_ID (*int*) – przechowuje id stacji, z której pociąg wyrusza. Nie może być ono puste.
• COURSE_ID (*int*) – przechowuje id kursu, którego dotyczy dany odjazd. Nie może być ono puste.
• ORDER (*int*) – przechowuje kolejność odwiedzenia danej stacji dla danego kursu. Nie może być ono puste. Cyfra 0 będzie oznaczała stację początkową.
• AVAILABLE_SEATS (*int*) – przechowuje ilość wolnych miejsc na danym etapie kursu. Nie może być puste.
• DATE (*time*) – przechowuje godzinę i minutę odwiedzenia miasta w postaci HH:MM. Nie może być ono puste.

RESERVATIONS - Tabela to przechowuje informacje rezerwacji danego klienta.
• ID (*int*) – przechowuje id rezerwacji. Nie może być ono puste.
• CUSTOMER_ID (*int*) – przechowuje id klienta, który dokonał rezerwacji. Nie może być ono puste.
• PRICE (*smallmoney*) – przechowuje cenę dla danej rezerwacji. Nie może być ono puste.
• DATE (*date*) – przechowuje dzień, na który ważny jest bilet, w postaci DD:MM:YYYY. Pole to nie może być puste.

SEATS - Tabela ta przechowuje informacji o miejscach na poszczególnych etapach kursu, które można zarezerwować.
• ID (*int*) – przechowuje id miejsca. Nie może być ono puste.
• NUMBER (*int*) – przechowuje numer miejsca
• COURSE_ID (*int*) – przechowuje id kursu do którego należy dane miejsce. Nie może być ono puste.

• VISIT_ID (*int*) – przechowuje stację końcową danego etapu kurs, dla którego zostało zarezerwowane miejsce. Może być NULL – dla wolnego miejsca.
• RESERVATION_ID (*int*) – przechowuje id rezerwacji dla danego miejsca. Może mieć wartość NULL jeśli miejsce jest wolne.

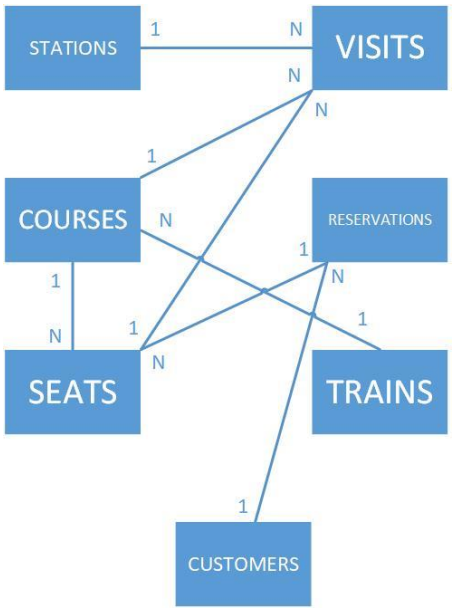
COURSES - Tabela ta przechowuje informacje o kursach.
• ID (*int*) – przechowuje id kursu. Nie może być ono puste.
• TRAIN_ID (*int*) – przechowuje id pociągu, którego dotyczy dany kurs. Nie może być puste.

TRAINS - Tabela ta przechowuje informacje o pociągach.
• ID (*int*) – przechowuje id pociągu. Nie może być ono puste.
• NAME (*varachar(50)*) – przechowuje nazwę pociągu. Nie może być puste oraz musi być unikalne.
• MODEL (*varchar(50)*) – przechowuje model danego pociągu. Nie może być puste.

WIDOK - SHOW_TRAINS - Tabela ta przechowuje informacje o pociągach.
• ID pociągu (*int*) – przechowuje id pociągu. Nie może być ono puste.
• Nazwa pociągu (*varachar(50)*) – przechowuje nazwę pociągu. Nie może być puste oraz musi być unikalne.
• Model pociągu (*varchar(50)*) – przechowuje model danego pociągu. Nie może być puste.

3.1.1. Uproszczony model konceptualny

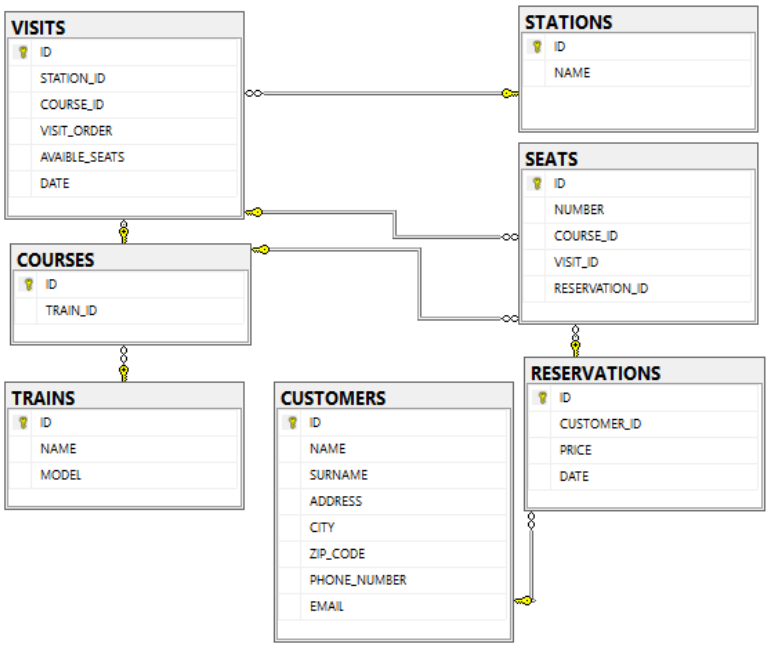
Model konceptualny został stworzony zgodnie z notacją Chana.



Rys. 4 Model konceptualny projektowanej bazy danych

3.1.2. Model logiczny i normalizacja

Model logiczny bazy danych przedstawiony na poniższym obrazku został wygenerowany przez program do zarządzania bazami danych – MS SQL Management Studio.



Rys. 5 Model logiczny projektowanej bazy danych

3.1.3. Inne elementy schematu – mechanizmy przetwarzania danych

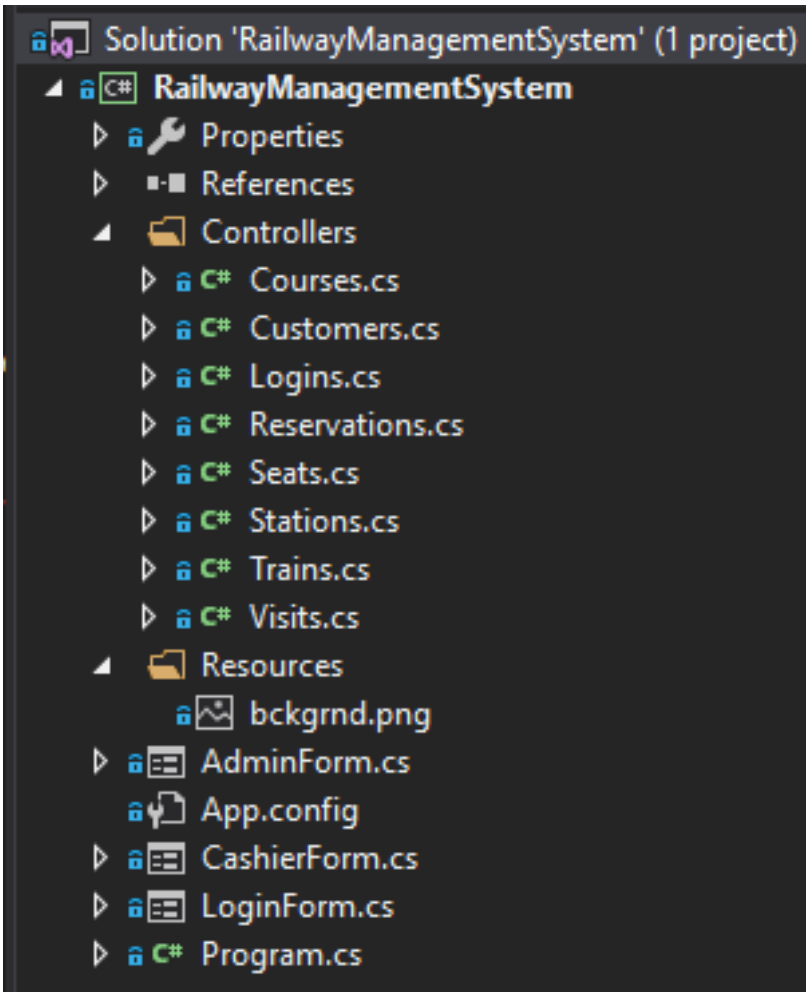
Mechanizmy przetwarzana danych w naszym projekcie zostały przedstawione w podpunkcie *3.2.3. Projekt wybranych funkcji systemu*.

3.2. Projekt aplikacji użytkownika

Ten punkt sprawozdania odnosi się do projektowania aplikacji. Przedstawia głównie kody skryptów oraz programów wchodzących w skład projektu.

3.2.1. Architektura aplikacji i diagramy projektowe

Aplikacja jest tworzona w języku C#, za pomocą Microsoft Visual Studio 2017. Poniższy screenshot pokazuje wszystkie pliki projektu.



Rys. 6 Składniki aplikacji

3.2.2. Interfejs graficzny i struktura menu

Interfejs graficzny jest tworzony za pomocą narzędzi wbudowanych w środowisko Visual Studio. Aplikacja została stworzona tak, aby korzystanie z niej było jak najbardziej intuicyjne oraz wygodne dla użytkowników, czy to kasjera, czy administratora systemu. Struktura menu programu oraz funkcje zostały pokazane w podpunkcie *4.2.1. Obsługa menu*.

3.2.3. Projekt wybranych funkcji systemu

Funkcje systemu obrazują przypadki użycia. Dokładny opis znajduje się w podpunkcie *2.2.2. Scenariusze wybranych przypadków użycia*.

3.2.4. Metoda podłączania do bazy danych – integracja z bazą danych

Metoda podłączenia do bazy danych oraz jej integracja zostały opisane w punkcie 4.2.3. *Implementacja interfejsu dostępu do bazy danych*.

3.2.5. Projekt zabezpieczeń na poziomie aplikacji

Dostęp do aplikacji zostanie zabezpieczony przez wybór loginu oraz podanie hasła.

4. Implementacja systemu

System został zaimplementowany zgodnie z założeniami projektu.

4.1. Realizacja bazy danych

Baza danych została stworzona za pomocą środowiska Microsoft SQL.

4.1.1. Tworzenie tabel i definiowanie ograniczeń

W naszym projekcie występuje 8 różnych tabel. Poniższy zrzut ekranu przedstawia kod odpowiedzialny za stworzenie tabeli STATIONS wraz z ograniczeniami:

```
CREATE TABLE [dbo].[STATIONS](
    [ID] [int] IDENTITY(1,1) NOT NULL,
    [NAME] [varchar](50) NOT NULL,
 CONSTRAINT [PK_STATIONS] PRIMARY KEY CLUSTERED
(
    [ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS =
ON) ON [PRIMARY]
) ON [PRIMARY]
```

4.1.2. Implementacja mechanizmów przetwarzania danych

Funkcje systemu bazy danych oparliśmy na widokach, funkcjach oraz procedurach SQL.

Pierwszy kod przedstawia mechanizm, który ma za zadanie podać informację o kursach między dwoma wybranymi miastami.

```
CREATE FUNCTION [dbo].[SHOW_COURSES_WITH_AB](@stacja_a VARCHAR(50),
@stacja_b VARCHAR(50))
RETURNS @kursy TABLE
(
COURSE_ID int primary key NOT NULL
)
AS
BEGIN
-- Pobranie kursów które zawierają stację a
declare @kursy_a as TABLE(COURSE_ID int, VISIT_ORDER int)
INSERT INTO @kursy_a (COURSE_ID, VISIT_ORDER) (
SELECT COURSE_ID, VISIT_ORDER
FROM VISITS
INNER JOIN STATIONS ON VISITS.STATION_ID = STATIONS.ID
WHERE STATIONS.NAME = @stacja_a)
-- Pobieranie kursów które zawierają stację a i stację b
```

```
declare @kursy_b as TABLE(COURSE_ID int, VISIT_ORDER int)
INSERT INTO @kursy_b (COURSE_ID, VISIT_ORDER) (
SELECT COURSE_ID, VISIT_ORDER
FROM VISITS
INNER JOIN STATIONS ON VISITS.STATION_ID = STATIONS.ID
WHERE COURSE_ID = ANY(SELECT COURSE_ID FROM @kursy_a) AND
STATIONS.NAME = @stacja_b)
--Sprawdzenie czy kolejność jest odpowiednia
INSERT INTO @kursy (COURSE_ID) (
SELECT [@kursy_b].COURSE_ID
FROM @kursy_b
INNER JOIN @kursy_a ON [@kursy_b].COURSE_ID = [@kursy_a].COURSE_ID
WHERE [@kursy_b].VISIT_ORDER > [@kursy_a].VISIT_ORDER
)
return
END
```

Kolejny kod to procedura. W tym przypadku procedura ma za zadanie wyświetlenie dostępnych siedzeń.

```
CREATE PROCEDURE [dbo].[SHOW_AVAIBLE_SEATS]
@COURSE_ID INT,
@STATION_A VARCHAR(70),
@STATION_B VARCHAR(70)
AS
BEGIN
DECLARE @FIRST_STATION INT = (
SELECT VISITS.VISIT_ORDER
FROM VISITS INNER JOIN STATIONS
ON VISITS.STATION_ID = STATIONS.ID
WHERE STATIONS.NAME = @STATION_A AND VISITS.COURSE_ID =
@COURSE_ID)
DECLARE @LAST_STATION INT = (
SELECT VISITS.VISIT_ORDER
FROM VISITS INNER JOIN STATIONS
ON VISITS.STATION_ID = STATIONS.ID
WHERE STATIONS.NAME = @STATION_B AND VISITS.COURSE_ID =
@COURSE_ID)
SELECT SIEDZENIA.SIEDZENIE
FROM (
SELECT SEATS.NUMBER AS SIEDZENIE, COUNT(SEATS.NUMBER) AS LICZBA
FROM SEATS INNER JOIN VISITS
ON SEATS.VISIT_ID = VISITS.ID
WHERE VISITS.VISIT_ORDER > (@FIRST_STATION - 1) AND
VISITS.VISIT_ORDER < (@LAST_STATION + 1) AND SEATS.COURSE_ID =
@COURSE_ID AND SEATS.RESERVATION_ID is NULL
GROUP BY SEATS.NUMBER) AS SIEDZENIA
WHERE SIEDZENIA.LICZBA = @LAST_STATION - @FIRST_STATION + 1;
END
```

Ostatnim użytym typem mechanizmu były widoki, czyli wirtualne tabele. Poniższy widok odpowiada za wyświetlanie trasy pociągu.

```
SELECT      dbo.COURSES.ID, dbo.TRAINS.NAME AS POCIĄG,
    (SELECT TOP (1) dbo.STATIONS.NAME
    FROM dbo.VISITS INNER JOIN
    dbo.STATIONS ON dbo.VISITS.STATION_ID = dbo.STATIONS.ID
    WHERE (dbo.VISITS.COURSE_ID = dbo.COURSES.ID)
    ORDER BY dbo.VISITS.VISIT_ORDER) AS PIERWSZA_STACJA,
    (SELECT TOP (1) dbo.STATIONS.NAME
    FROM dbo.VISITS INNER JOIN
```

```
    dbo.STATIONS ON dbo.VISITS.STATION_ID = dbo.STATIONS.ID
    WHERE (dbo.VISITS.COURSE_ID = dbo.COURSES.ID)
    ORDER BY dbo.VISITS.VISIT_ORDER DESC) AS OSTATNIA_STACJA
FROM dbo.COURSES INNER JOIN
dbo.TRAINS ON dbo.COURSES.TRAIN_ID = dbo.TRAINS.ID
```

4.2. Realizacja elementów aplikacji

Program został napisany w języku C# za pomocą środowiska Microsoft Visual Studio 2017. Aplikacja wyposażona jest w dwa oddzielne interfejsy. Wybór danego trybu następuje podczas procesu logowania do aplikacji. W zależności od użytkownika, program może się otworzyć w trybie kasjera lub administratora.

4.2.1. Walidacja i filtracja

Funkcja walidacji w naszym projekcie została użyta podczas dodawania danych nowego klienta. Wtedy sprawdzana jest poprawność wpisywania danych, np. numeru telefonu.

```
var customerData = new Customers.CustomerData(
textBoxNewCustomerName.Text.All(char.IsLetter) && textBoxNewCustomerName.Text.Length < 51 ? textBoxNewCustomerName.Text : SetDataIncorrect(),
textBoxNewCustomerSurname.Text.Length < 51 && textBoxNewCustomerSurname.Text.Any(char.IsLetter) && !textBoxNewCustomerSurname.Text.Any(char.IsDigit) &&
textBoxNewCustomerAddress.Text.Any(char.IsLetter) && textBoxNewCustomerAddress.Text.Length < 71 && textBoxNewCustomerAddress.Text.Contains(" ") && textBo
textBoxNewCustomerCity.Text.All(char.IsLetter) && textBoxNewCustomerCity.Text.Length < 51 ? textBoxNewCustomerCity.Text : SetDataIncorrect(),
!textBoxNewCustomerZipCode.Text.Any(char.IsLetter) && textBoxNewCustomerZipCode.Text.Length == 6 && textBoxNewCustomerZipCode.Text.Contains('-') ? textBo
textBoxNewCustomerPhoneNumber.Text.All(char.IsDigit) && textBoxNewCustomerPhoneNumber.Text.Length == 11 && !textBoxNewCustomerPhoneNumber.Text.Contains('
textBoxNewCustomerEmail.Text.Contains('@') && textBoxNewCustomerEmail.Text.Contains('.') && !textBoxNewCustomerEmail.Text.Contains(" ") && textBoxNewCust
```

Rys. 7 Użycie walidacji w projekcie.

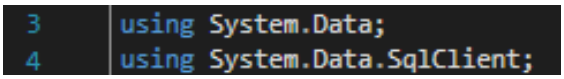
Filtrację możemy znaleźć podczas wyszukiwania danego klienta z poziomu kasjera oraz administratora.

```
private void textBoxSearch_TextChanged(object sender, EventArgs e)
{
    string name = textBoxSearchByName.Text;
    string surname = textBoxSearchBySurname.Text;
    string email = textBoxSearchByEmail.Text;
    string phoneNumber = textBoxSearchByPhoneNumber.Text;

    DataTable dataTable = (DataTable)dataGridViewCustomers.DataSource;

    string rowFilter = string.Format("Imię LIKE '{0}%'", name);
    rowFilter += string.Format("AND Nazwisko LIKE '{0}%'", surname);
    rowFilter += string.Format("AND Email LIKE '{0}%'", email);
    rowFilter += string.Format("AND [Nr. tel.] LIKE '{0}%'", phoneNumber);

    dataTable.DefaultView.RowFilter = rowFilter;
}
```

Rys. 8 Użycie filtracji w projekcie

4.2.2. Implementacja interfejsu dostępu do bazy danych

Język C# zawiera w sobie zestaw gotowych bibliotek umożliwiających łączenie i obsługę różnych baz danych.



Rys. 9 Użycie bibliotek w projekcie

Są to providery dla baz SQL. Aby automatycznie połączyć się z konkretną bazą danych SQL, potrzebne jest wykorzystanie następującego kodu:

```
var machineName = Environment.MachineName;
sqlConnection = new SqlConnection("Data Source="+ machineName + "\\SQLEXPRESS; database=SRBK_database;Trusted_Connection=yes");
```

Rys. 10 Tworzenie nowego obiektu SQL Connection

4.2.3. Implementacja wybranych funkcjonalności systemu

Przykładowym wywołaniem procedury jest metoda AddReservations w klasie Reservations, która wywołuje procedurę ADD_RESERVATIONS. Dodaje ona do systemu bazodanowego nową rezerwację, na podstawie podanych przez kasjera danych. Wywołanie to, jak i wszystkie inne w zaimplementowanej aplikacji, jest odporne na ataki typu SQL Injection.

```
public static bool AddReservations(SqlConnection sqlConnection, string customerId,
                string price, string courseId, string stationA, string stationB, string seatNumber)
{
    try
    {
        sqlConnection.Open();
        string command = $"EXEC ADD_RESERVATIONS " +
                        $"@customerID, @price, @courseID, @stationA, @stationB, @seatNumber";
        SqlCommand sqlCommand = new SqlCommand(command, sqlConnection);
        sqlCommand.Parameters.AddWithValue("@customerID", customerId);
        sqlCommand.Parameters.AddWithValue("@price", price);
        sqlCommand.Parameters.AddWithValue("@courseID", courseId);
        sqlCommand.Parameters.AddWithValue("@stationA", stationA);
        sqlCommand.Parameters.AddWithValue("@stationB", stationB);
        sqlCommand.Parameters.AddWithValue("@seatnNumber", seatNumber);
        sqlCommand.ExecuteNonQuery();

    }
    catch
    {
        Debug.WriteLine("Błąd zapytania do bazy danych!");
        return false;
    }
    finally
    {
        sqlConnection.Close();
    }
    return true;
}
```

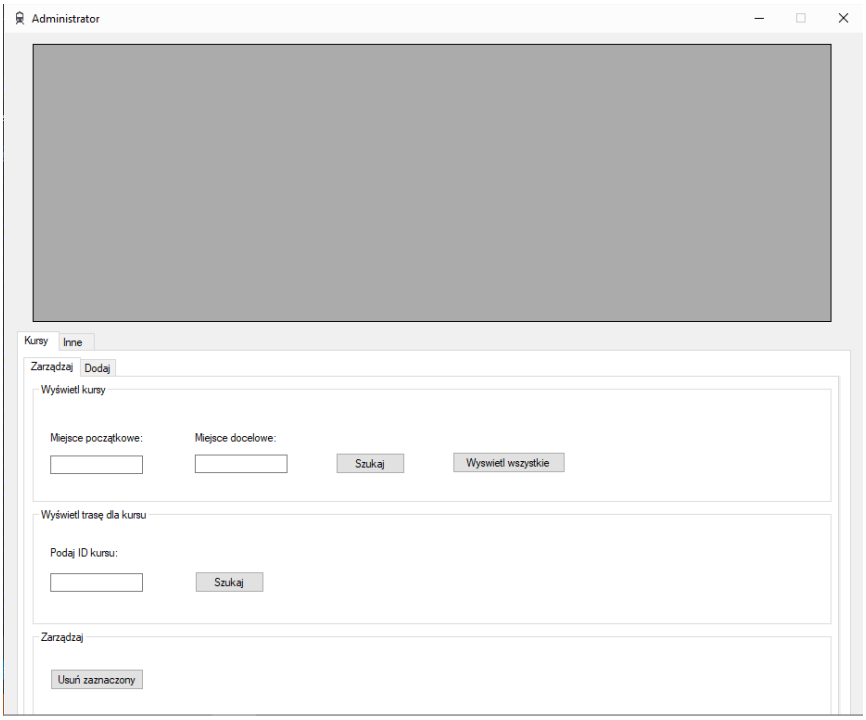
Rys. 11 Przykład wywołania procedury ADD_RESERVATIONS

4.2.4. Implementacja mechanizmów bezpieczeństwa

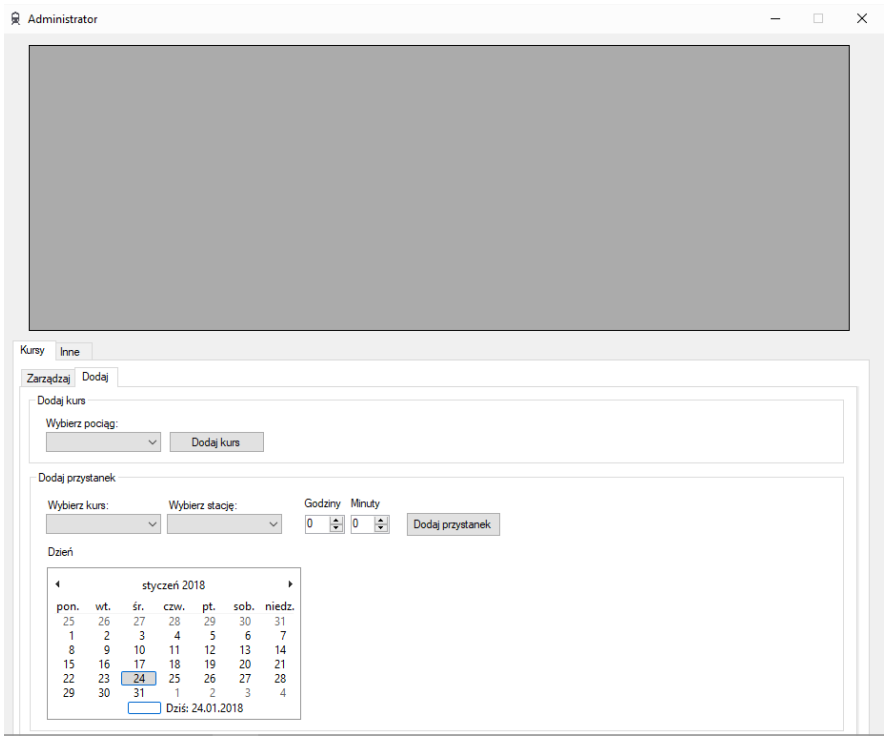
Mechanizmy bezpieczeństwa dostępu do aplikacji zostały opisane w *5.3. Testowanie mechanizmów bezpieczeństwa*.

5. Testowanie systemu

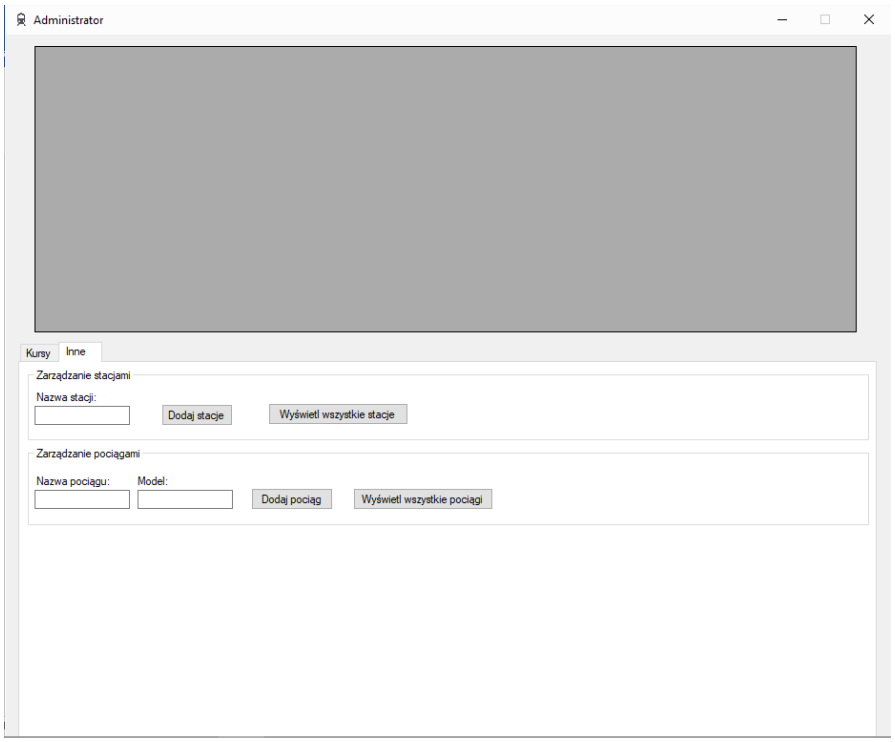
Po zalogowaniu się na konto administratora, użytkownik ma możliwość dodania nowych kursów oraz zarządzania już istniejącymi. W zakładce „inne” użytkownik ma możliwość zarządzania stacjami w systemie.



Rys. 12 Ekran aplikacji w trybie admin – zarządzanie kursami

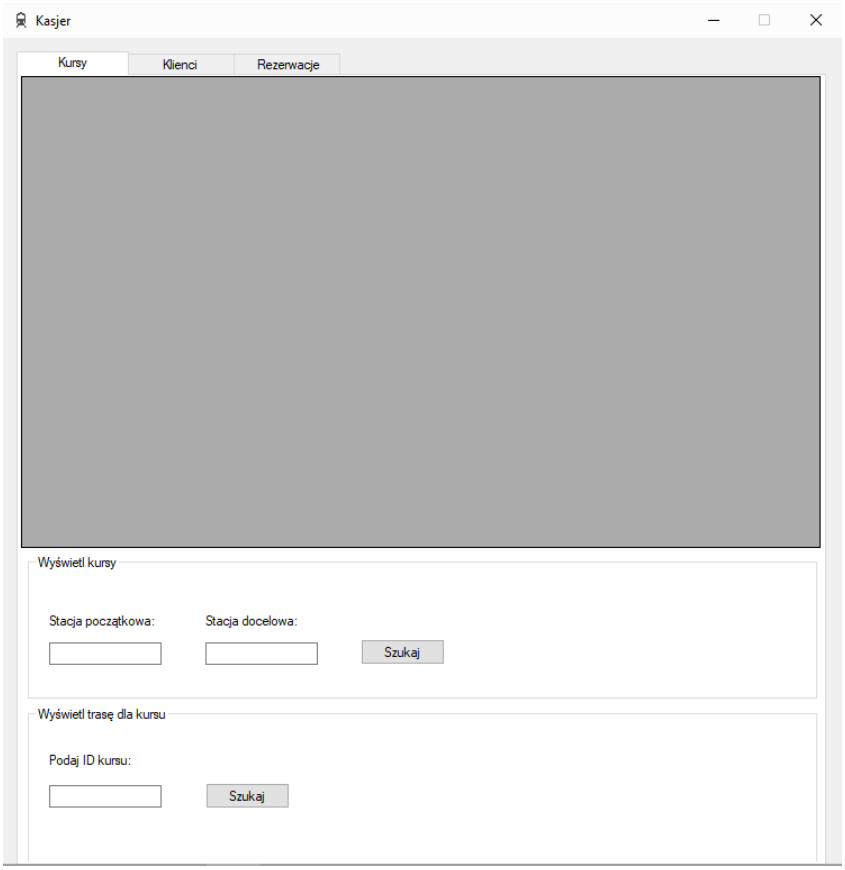


Rys. 13 Ekran aplikacji w trybie admin – dodawanie kursu

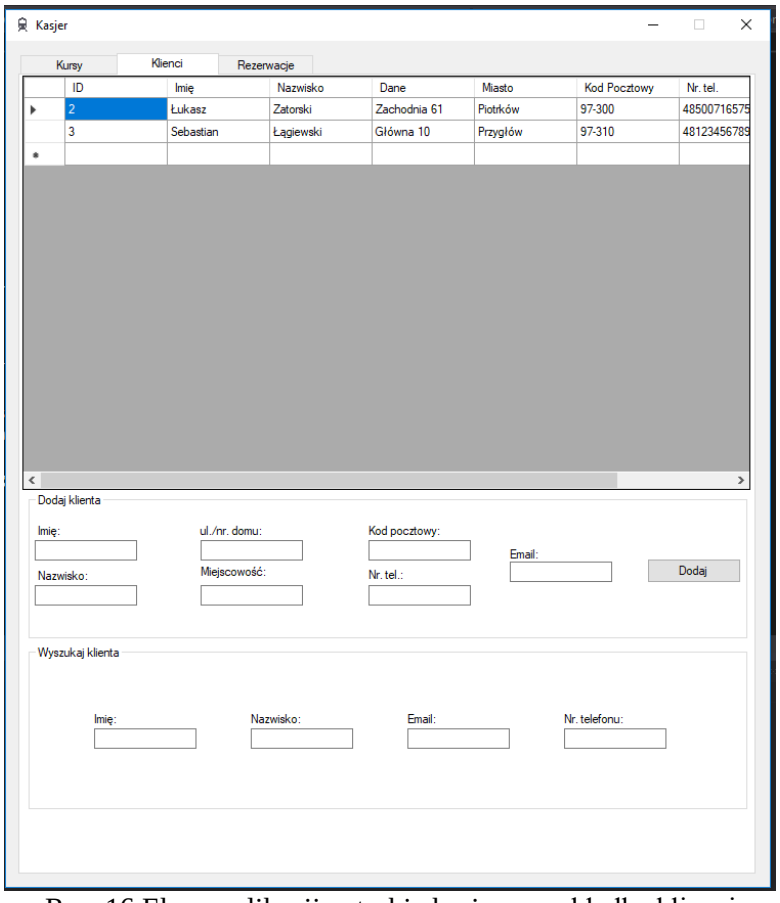


Rys. 14 Ekran aplikacji w trybie admin – zakładka inne

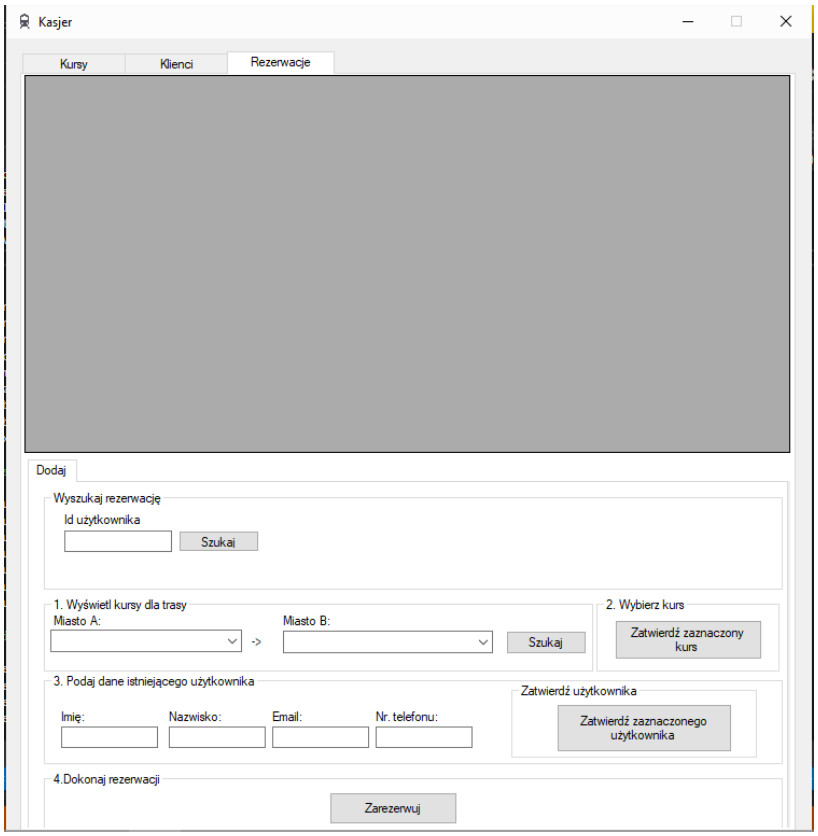
Po zalogowaniu się na konto kasjera, użytkownik ma możliwość dodania i wyświetlenia wszystkich kursów dostępnych w systemie oraz wyszukiwanie ich po ID połączenia. Dodatkowo kasjer ma możliwość dodania klienta do bazy wraz z informacjami osobowymi, typu: adres, numer telefonu, email etc. Może także wyszukiwać po danych takich jak imię, nazwisko, numer telefonu. Najważniejszą funkcją jest dodanie rezerwacji w systemie. Użytkownik wybiera dane połączenie, stację początkową, stację końcową oraz dane klienta. Po tym może zatwierdzić transakcję i wprowadzić ją do bazy danych.



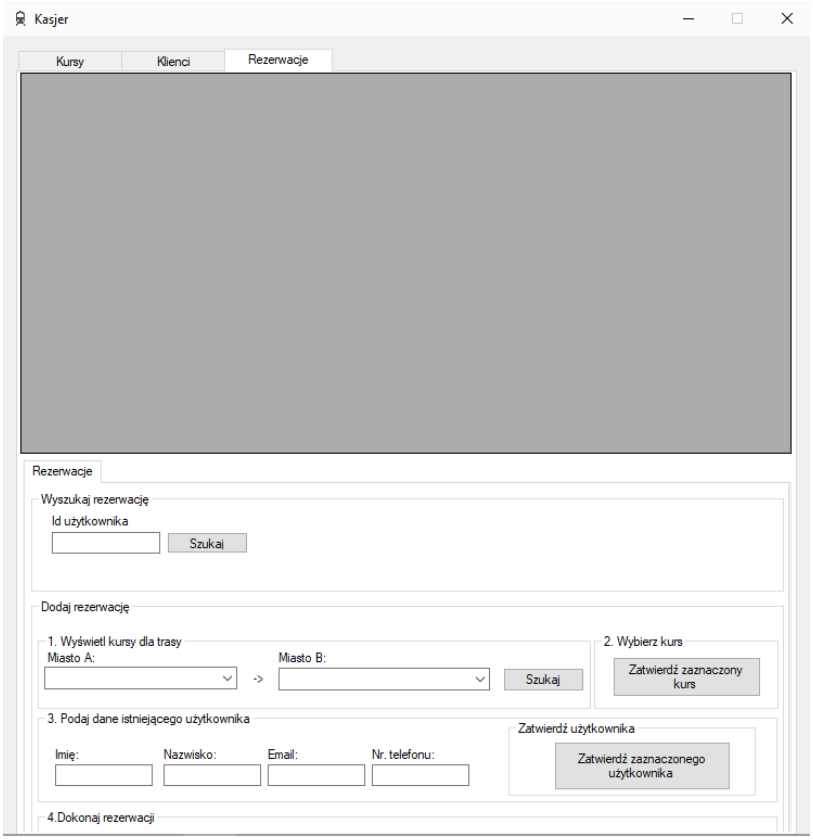
Rys. 15 Ekran aplikacji w trybie kasjera – zakładka kursy



Rys. 16 Ekran aplikacji w trybie kasjera – zakładka klienci



Rys. 17 Ekran aplikacji w trybie kasjera – zakładka rezerwacje, funkcja wyszukiwania rezerwacji



Rys. 18 Ekran aplikacji w trybie kasjera – zakładka rezerwacja, funkcja dodawania rezerwacji

5.1. Instalacja i konfigurowanie systemu

Aplikacja dostarczona jest do klienta z rozszerzeniem .exe, natomiast bazę danych należy zainstalować osobno wg. następujących kroków:

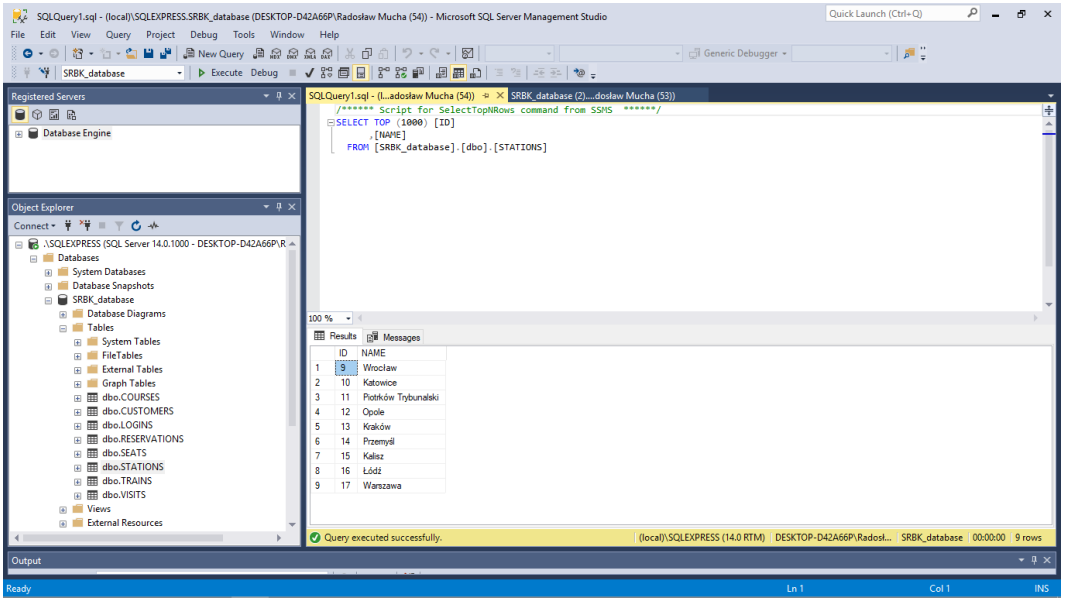
1. Pobranie MS SQL Server 2014 Express
2. Instalacja z domyślnymi ustawieniami
3. Zalogowanie się do serwera
4. Uruchomienie skryptu *SRBK_database.sql* przez dwukrotne kliknięcie LPM
5. Wykonanie skryptu (naciśnięcie przycisku z czerwonym wykrzyknikiem i napisem "*Execute*")

Po tych czynnościach program oraz baza danych są gotowe do użycia.

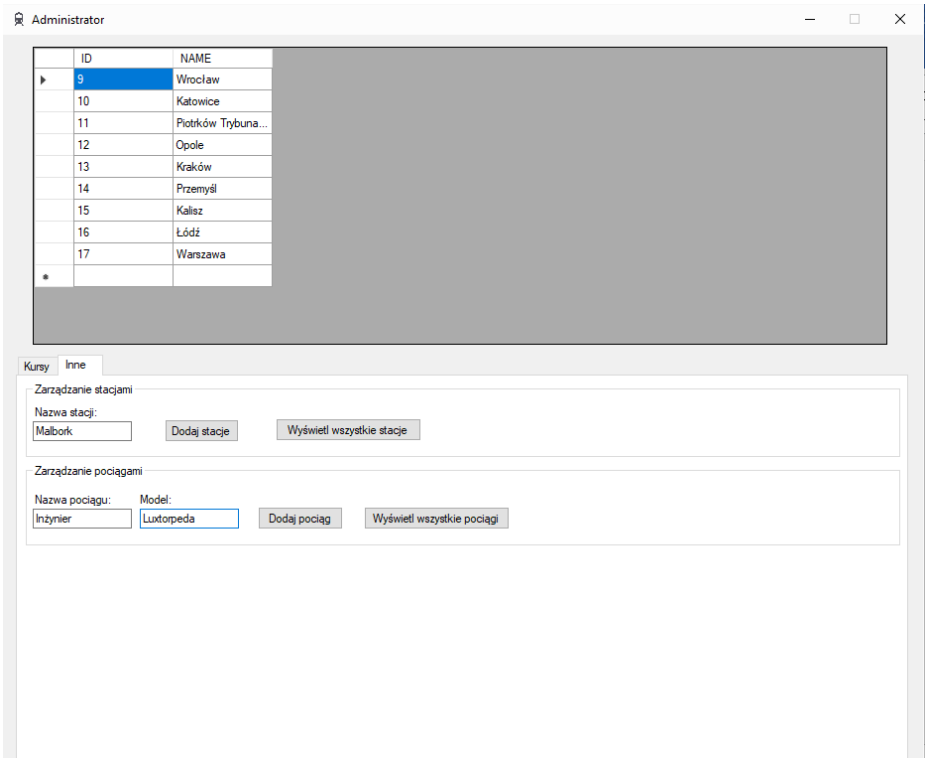
5.2. Testowanie opracowanych funkcji systemu

5.2.1. Testowanie dodawania pozycji do bazy danych

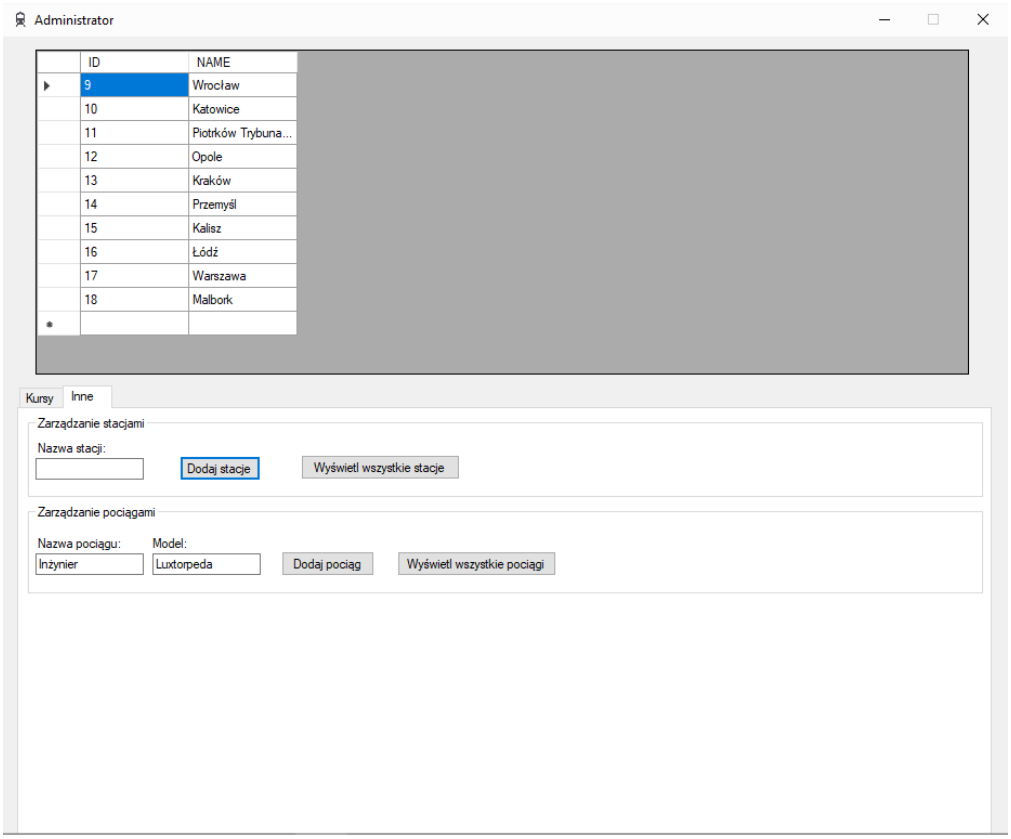
W trybie administratora mamy możliwość dodania kolejnych stacji do bazy danych:



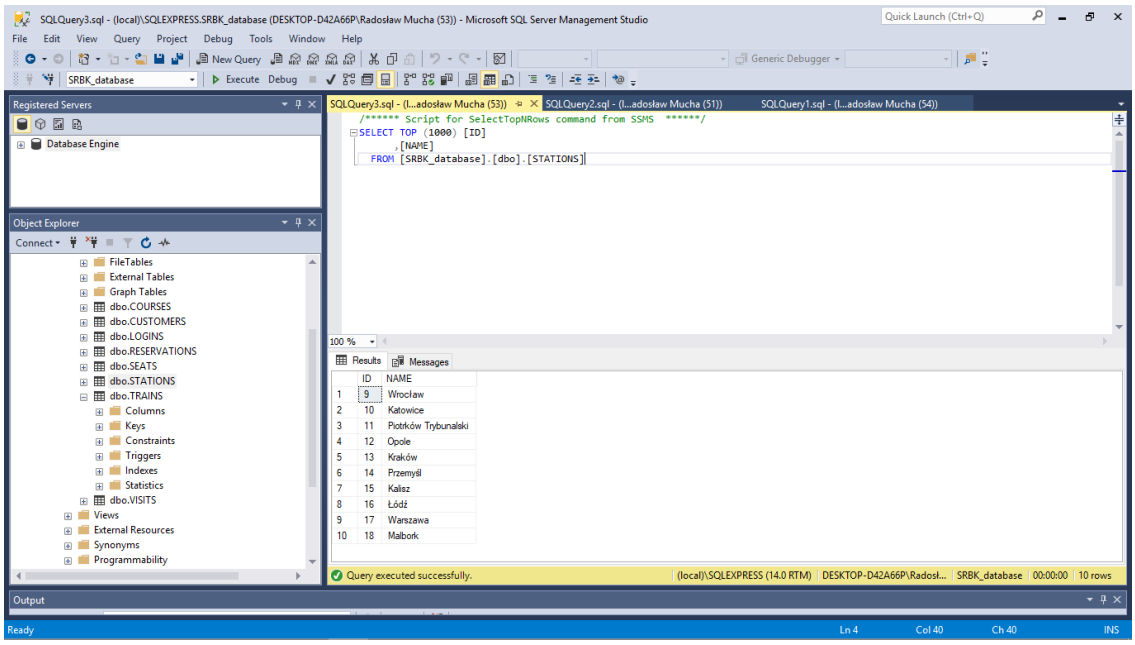
Rys. 19 Ekran aplikacji SQL Management Studio przed dodaniem nowej stacji



Rys. 20 Ekran aplikacji w trybie administratora – dodawanie nowej stacji oraz modelu pociągu

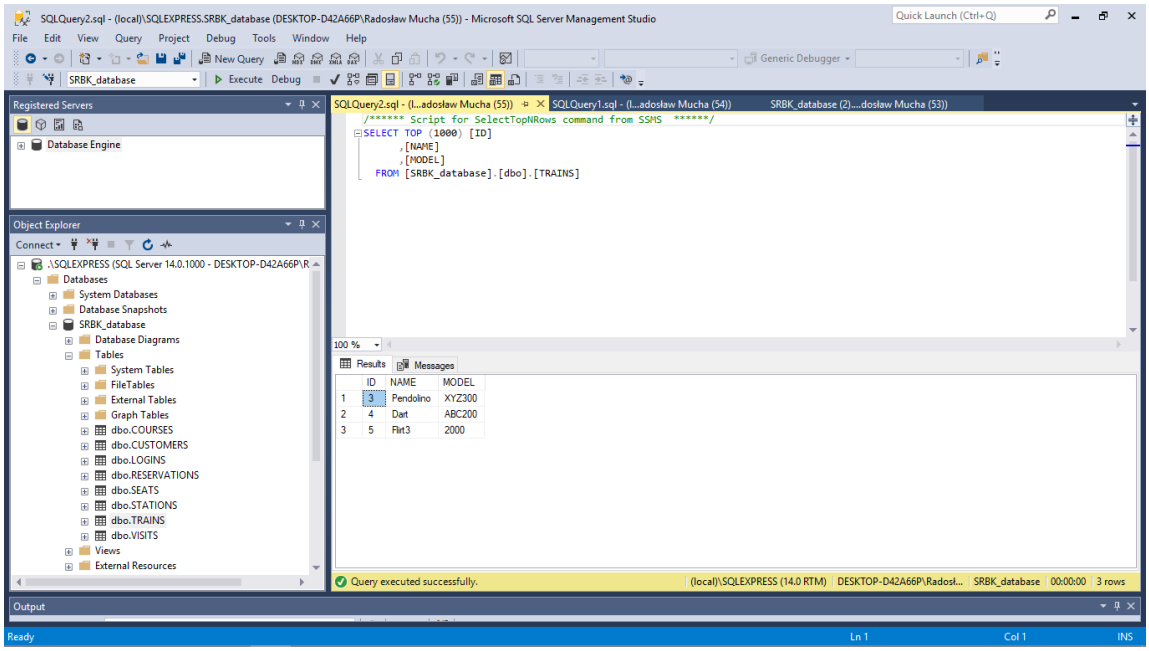


Rys. 21 Ekran aplikacji w trybie administratora – wynik działania aplikacji po dodaniu stacji „Malbork”

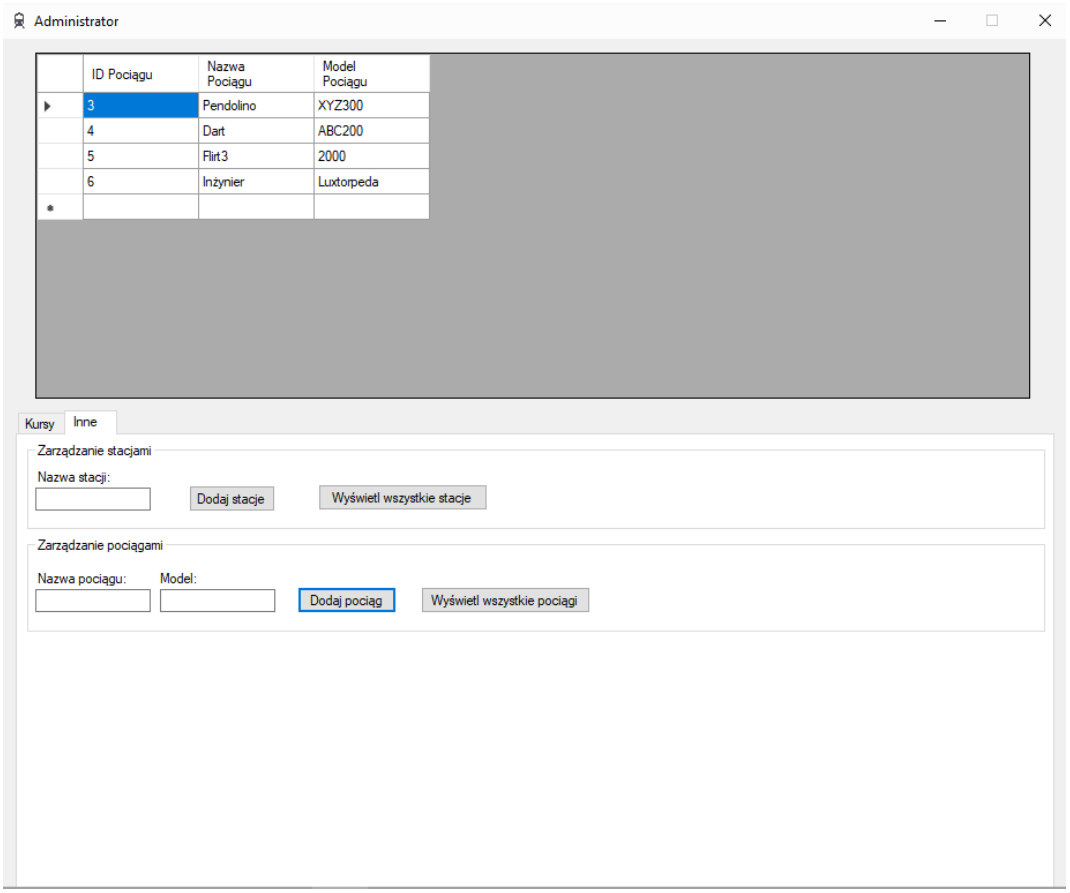


Rys. 22 Ekran aplikacji SQL Management Studio po dodaniu nowej stacji Malbork

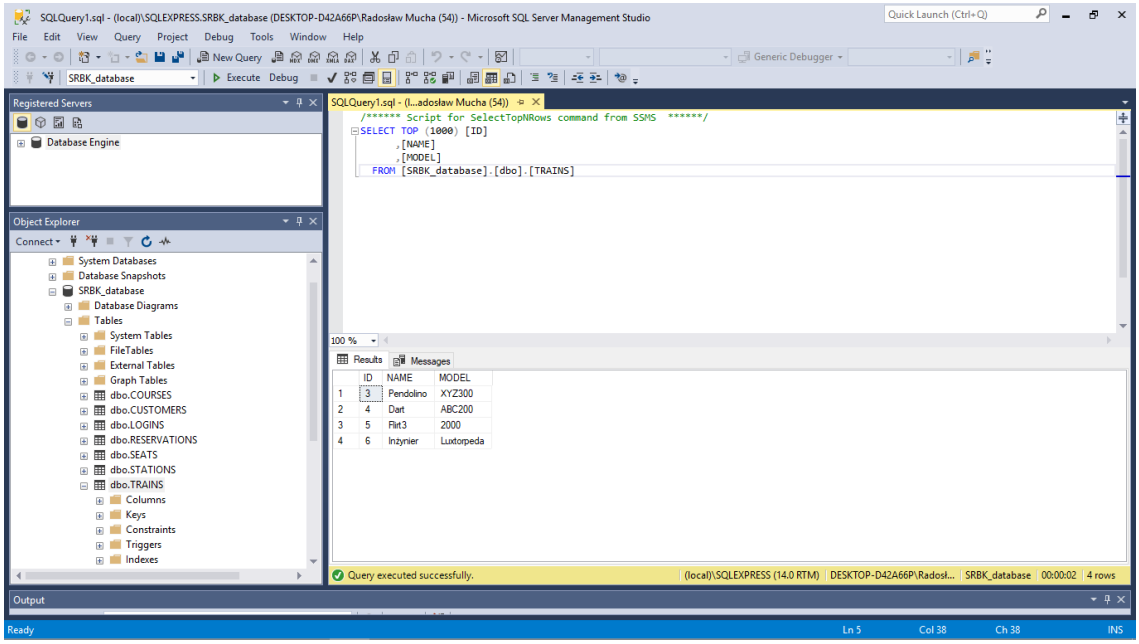
Następnym testem było sprawdzenie w trybie administratora możliwości dodania kolejnych modeli pociągów:



Rys. 23 Ekran aplikacji SQL Managment Studio przed dodaniem nowego pociągu



Rys. 24 Ekran aplikacji w trybie administratora – wynik działania aplikacji po dodaniu nowego modelu pociągu „Luxtorpeda”

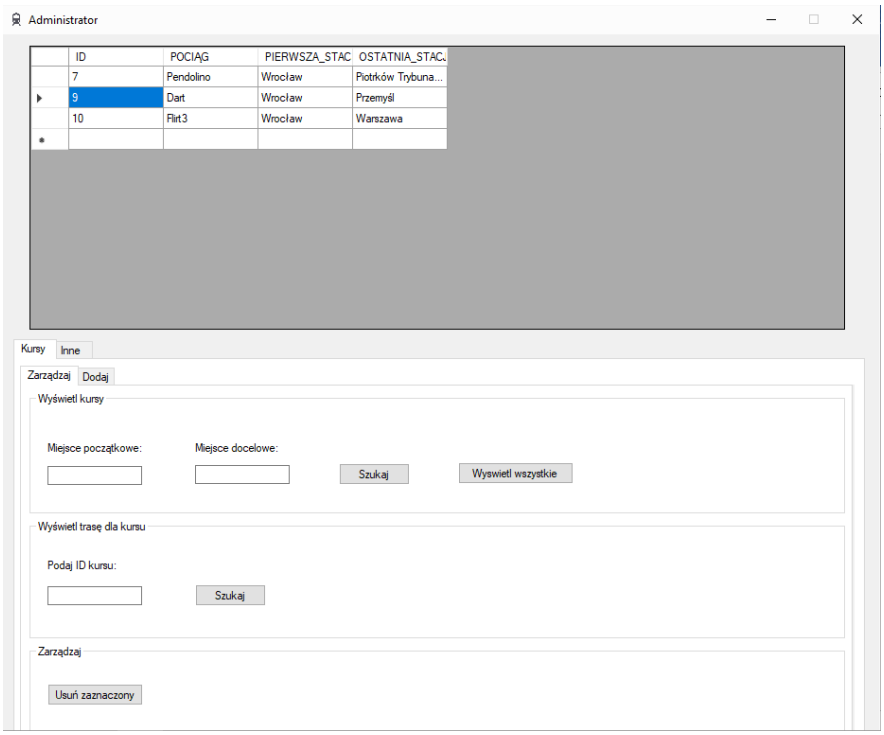


Rys. 25 Ekran aplikacji SQL Managment Studio po dodaniu nowego modelu pociągu „Luxtorpeda”

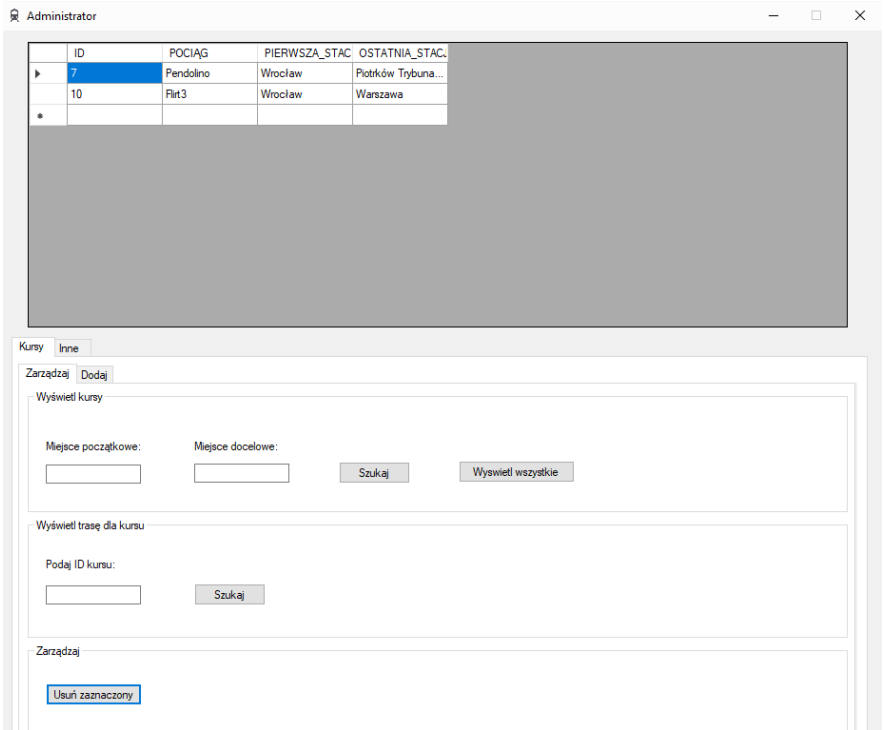
Jak można zauważyć, program prawidłowo dodał do bazy danych nową stację oraz model pociągu.

5.2.2.Testowanie usuwania pozycji z bazy danych

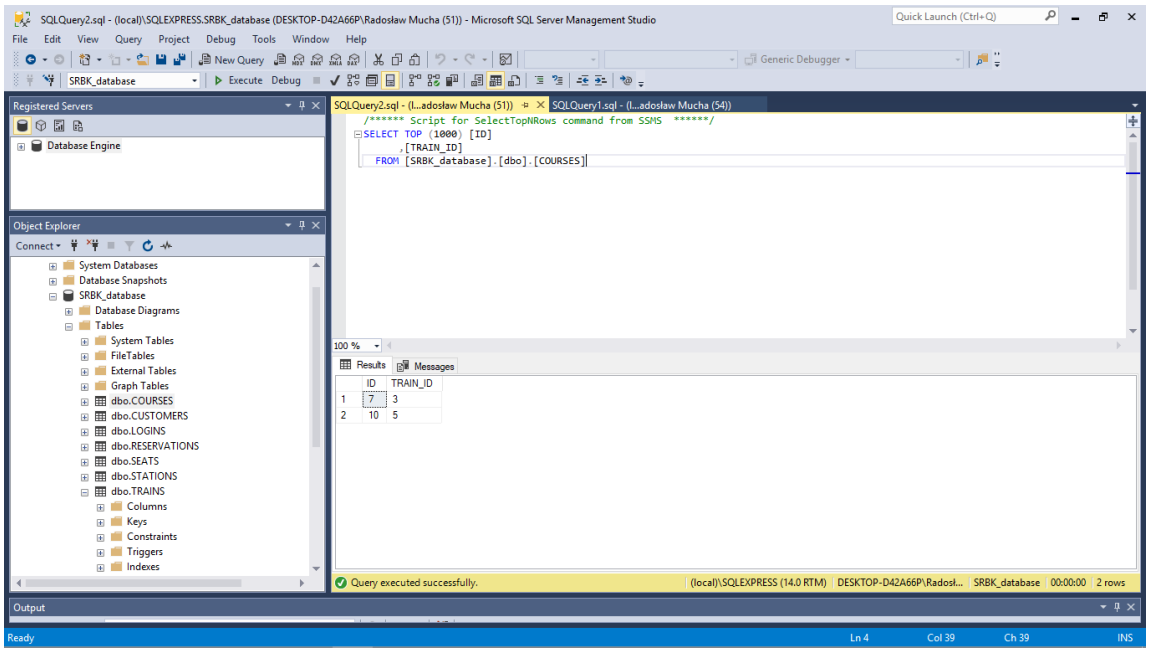
W trybie administratora mamy możliwość usunięcia konkretnego kursu z systemu rezerwacji.



Rys. 26 Ekran aplikacji w trybie administratora – wyświetlanie dostępnych kursów



Rys. 27 Ekran aplikacji w trybie administratora – wyświetlanie dostępnych kursów po usunięciu

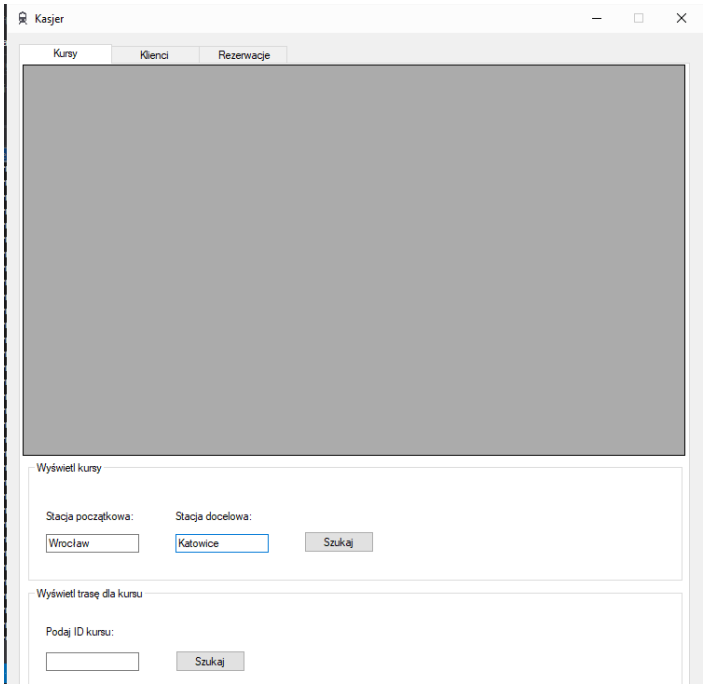


Rys. 28 Ekran aplikacji w trybie administratora – wyświetlanie dostępnych kursów po usunięciu wybranego kursu

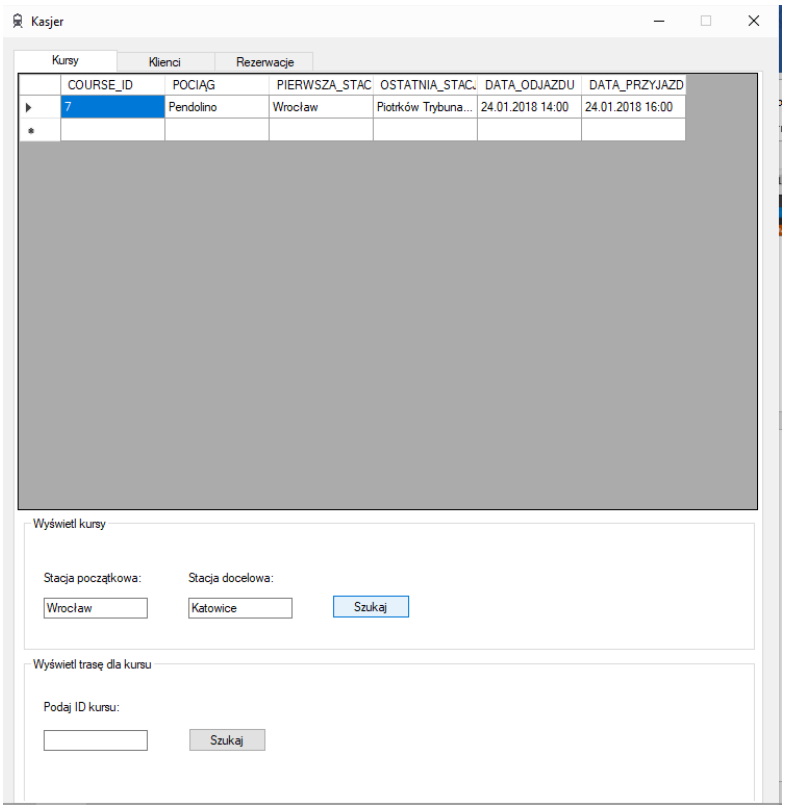
Program poprawnie usunął z bazy danych wybrany przez użytkownika kurs.

5.2.3.Testowanie wyświetlania pozycji z bazy danych

Aplikacja ma także możliwość wyszukiwania dostępnych kursów pomiędzy dwoma wybranymi miastami.



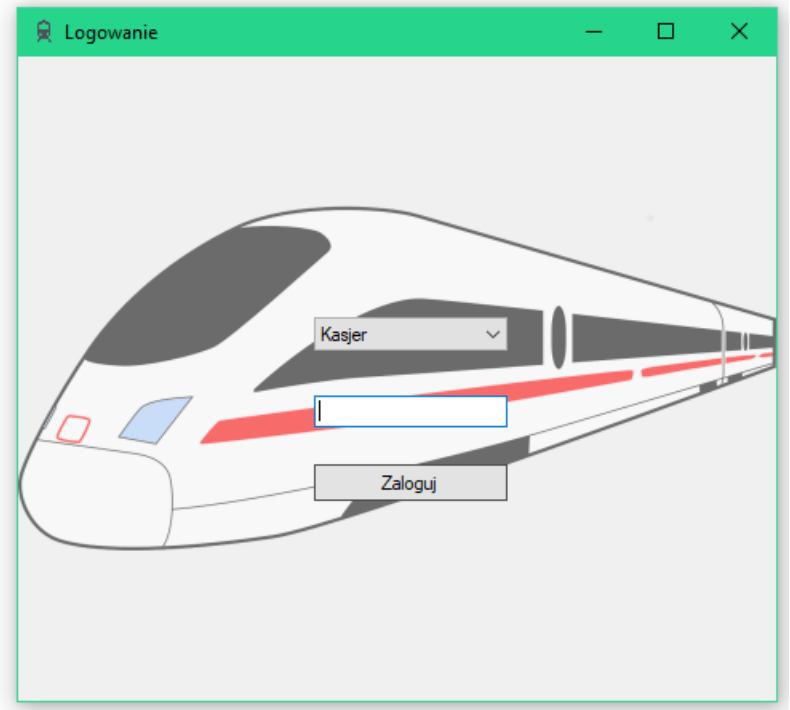
Rys. 29 Ekran aplikacji w trybie administratora – zakładka kursy



Rys. 30 Ekran aplikacji w trybie administratora – wyświetlanie dostępnych kursów

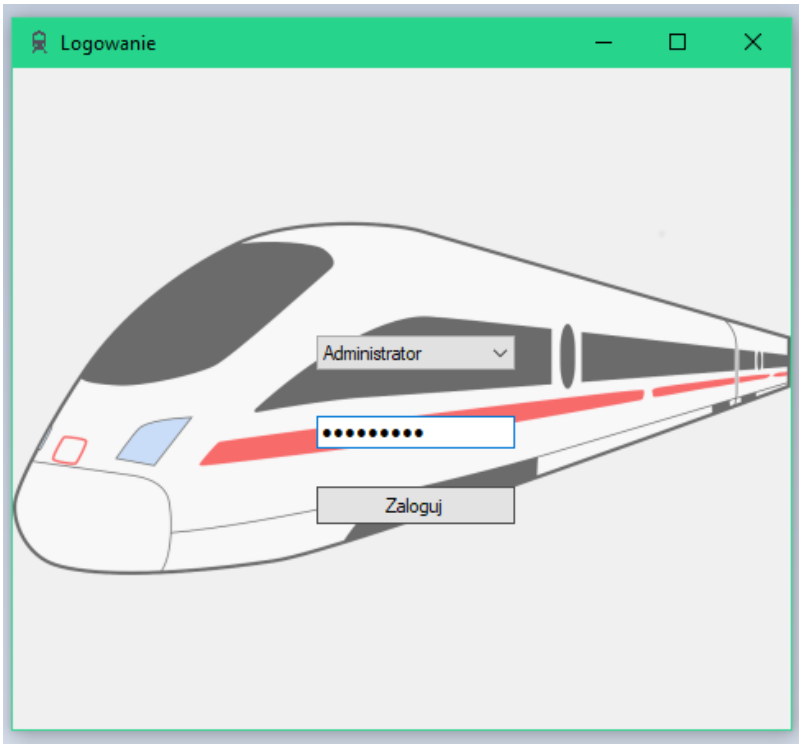
5.3. Testowanie mechanizmów bezpieczeństwa

Po uruchomieniu aplikacji użytkownik ma możliwość wybrania loginu oraz typu widoku aplikacji, w zależności od funkcji (kasjer/administrator). Każdy użytkownik systemu ma przydzielony inny login oraz różne hasło. Hasła użytkowników są szyfrowane przez algorytm AES wykorzystany w naszym projekcie. Dzięki temu mamy zapewniony wysoki poziom bezpieczeństwa w dostępie do bazy danych.



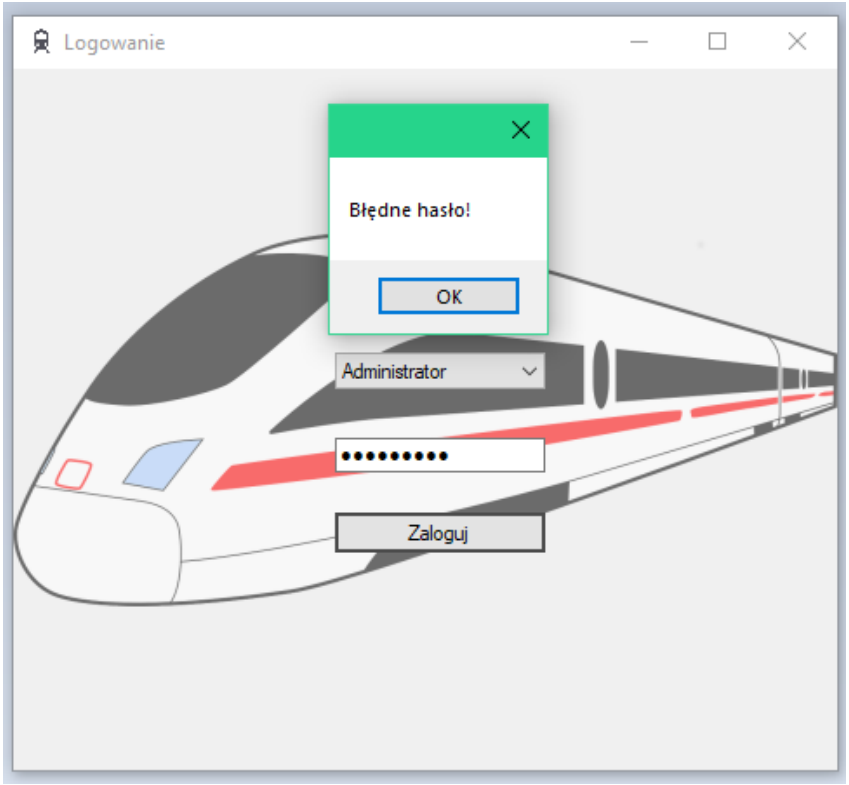
Rys. 31 Ekran logowania do aplikacji

Dostęp do aplikacji jest chroniony hasłem. Podczas wprowadzania, znaki wpisywanego hasła są maskowane.



Rys. 32 Efekt maskowania hasła

Wprowadzenie nieprawidłowych danych powoduje wyświetlenie komunikatu „błąd danych”, co zapobiega nieautoryzowanemu dostępowi do systemu rezerwacji.



Rys. 33 Błąd po wprowadzeniu nieprawidłowego hasła

5.4. Wnioski z testów

Aplikacja oraz bazy danych działają prawidłowo, zgodnie z początkowymi założeniami. Nie zauważono błędów w działaniu aplikacji oraz bazy danych.

6. Podsumowanie

Wszystkie założenia projektu postawione na początku dokumentu zostały zrealizowane. Sam projekt okazał się wymagającym zadaniem. Podczas prac nad aplikacją oraz bazą danych napotkaliśmy wiele problemów, których rozwiązanie pochłonęło nam wiele czasu. Dzięki temu poszerzyliśmy naszą wiedzę oraz nabraliśmy doświadczenia z zakresu baz danych, języka SQL oraz programowania obiektowego. Wiadomości oraz wiedzę zdobytą podczas realizacji projektu będziemy mogli wykorzystać w przyszłości.

Literatura

[1] Górski J., *Inżynieria oprogramowania w projekcie informatycznym*, Mikom, Warszawa, 2000
[2] Pelikant A., *MS SQL Server. Zaawansowane metody programowania*, Helion, Gliwice, 2014.
[3] Beynon-Davies P., *Systemy baz danych*, WNT, Warszawa, 2000.
[4] Garcia-Molina H., Ullman J.D., Widom J., *Systemy baz danych. Kompletny podręcznik. Wydanie II,* Prentice Hall, New Jersey, 2011.
[5] Posadas M., Tajniki C# i .NET Framework. *Wydajne aplikacje dzięki zaawansowanym funkcjom języka C# i architektury .NET*, Packt Publishing, 2017