

POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

PROJEKT Z BAZ DANYCH

System obsługi biletów dla firmy oferującej loty pasażerskie.

Termin zajęć: Wtorek, 13:15–14:45

AUTOR/AUTORZY:

Krzysztof Jopek

Indeks: 241406

E-mail: 241406@student.pwr.edu.pl

Hubert Knaziak

Indeks: 241320

E-mail: 241320@student.pwr.edu.pl

PROWADZĄCY ZAJĘCIA:

dr inż. Roman Ptak, W4/K9

Wrocław, 2019 r.

Spis treści:

1. Wstęp.....	4
1.1. Cel projektu	4
1.2. Zakres projektu.....	4
2. Analiza wymagań	4
2.1. Opis działania i schemat logiczny systemu	4
2.2. Wymagania funkcjonalne	4
2.2.1. Diagram przypadków użycia.....	5
2.2.2. Scenariusze przypadków użycia.....	5
2.3. Wymagania нефункционалне	11
2.4. Przyjęte założenia projektowe.....	11
3. Projekt systemu	11
3.1. Projekt bazy danych	11
3.1.1. Analiza rzeczywistości i uproszczony model konceptualny	11
3.1.2. Model logiczny i normalizacja	12
3.1.3. Model fizyczny i ograniczenia integralności danych	13
3.1.4. Inne elementy schematu – mechanizmy przetwarzania danych	13
3.1.5. Projekt mechanizmów bezpieczeństwa na poziomie bazy danych.....	13
3.2. Projekt aplikacji użytkownika	14
3.2.1. Architektura aplikacji i diagramy projektowe	14
3.2.2. Interfejs graficzny i struktura menu	14
3.2.3. Projekt wybranych funkcji systemu	15
3.2.4. Metoda podłączania do bazy danych – integracja z bazą danych	20
3.2.5. Projekt zabezpieczeń na poziomie aplikacji.....	20
4. Implementacja systemu baz danych	21
4.1. Tworzenie tabel i definiowanie ograniczeń // inna czcionka	21
4.2. Implementacja mechanizmów przetwarzania danych.....	22
4.3. Implementacja uprawnień i innych zabezpieczeń	22
4.4. Testowanie bazy danych na przykładowych danych.....	23
5. Implementacja i testy aplikacji.....	25
5.1. Instalacja i konfigurowanie systemu	25
5.2. Instrukcja użytkowania aplikacji.....	25
5.3. Testowanie opracowanych funkcji systemu	26
5.4. Omówienie wybranych rozwiązań programistycznych.....	28

5.4.1. Implementacja interfejsu dostępu do bazy danych	28
5.4.2. Implementacja wybranych funkcjonalności systemu	30
5.4.3. Implementacja mechanizmów bezpieczeństwa	33
6. Podsumowanie i wnioski.....	35
Literatura	35
Spis rysunków	35

1. Wstęp

1.1. Cel projektu

Celem projektu jest zaprojektowanie i implementację aplikacji umożliwiającej dostęp do bazy danych, która będzie umożliwiać zakup biletów na loty obsługiwane przez daną firmę przewoźową.

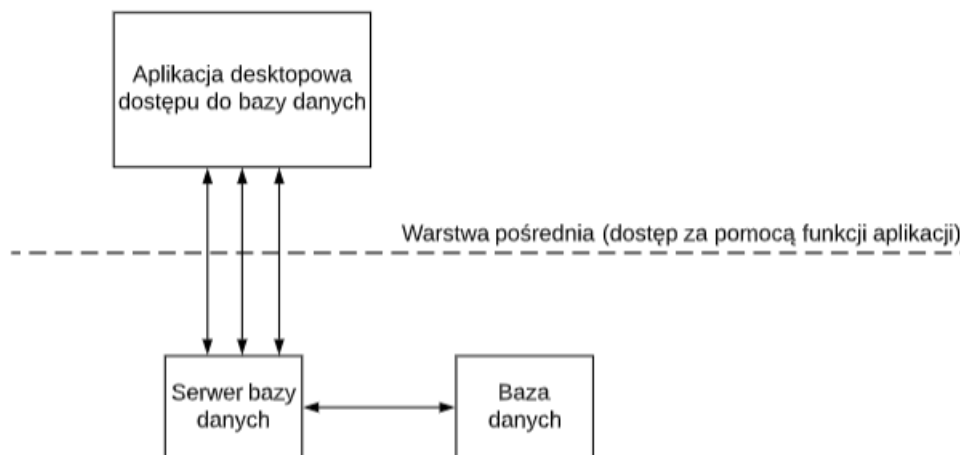
1.2. Zakres projektu

Zakres pracy obejmuje zaprojektowanie bazy danych obsługiwanej przez serwer bazodanowy Microsoft SQL Server. W utworzonej bazie zostały wprowadzone tabele i widoki realizujące odpowiednie zapytania zadawane do bazy z poziomu aplikacji. Ta z kolei została napisana w języku Java. Zaimplementowany został interfejs graficzny za pomocą którego zachodzi interakcja z użytkownikiem. Ponadto został zaimplementowany system zabezpieczeń aplikacji w postaci logowania.

2. Analiza wymagań

2.1. Opis działania i schemat logiczny systemu

Aplikacja pozwala na sprzedaż usługi lotu samolotem pasażerskim w międzynarodowej firmie zajmującej się świadczeniem usług lotniczych połączeń międzynarodowych. Aplikacja ma umożliwić zdalny zakup biletu na lot z bazy lotów przypisanej lokalnej placówce firmy. Dostęp do bazy danych będzie możliwy z poziomu aplikacji desktopowej. System będzie umożliwiał wykonywanie określonych operacji w zależności od rodzaju użytkownika (np. pracownik(administrator) – zarządzanie połączeniami, zarządzanie kontami i pasażerami; klient – rejestracja w systemie, kupowanie biletów, wyszukiwanie połączeń).



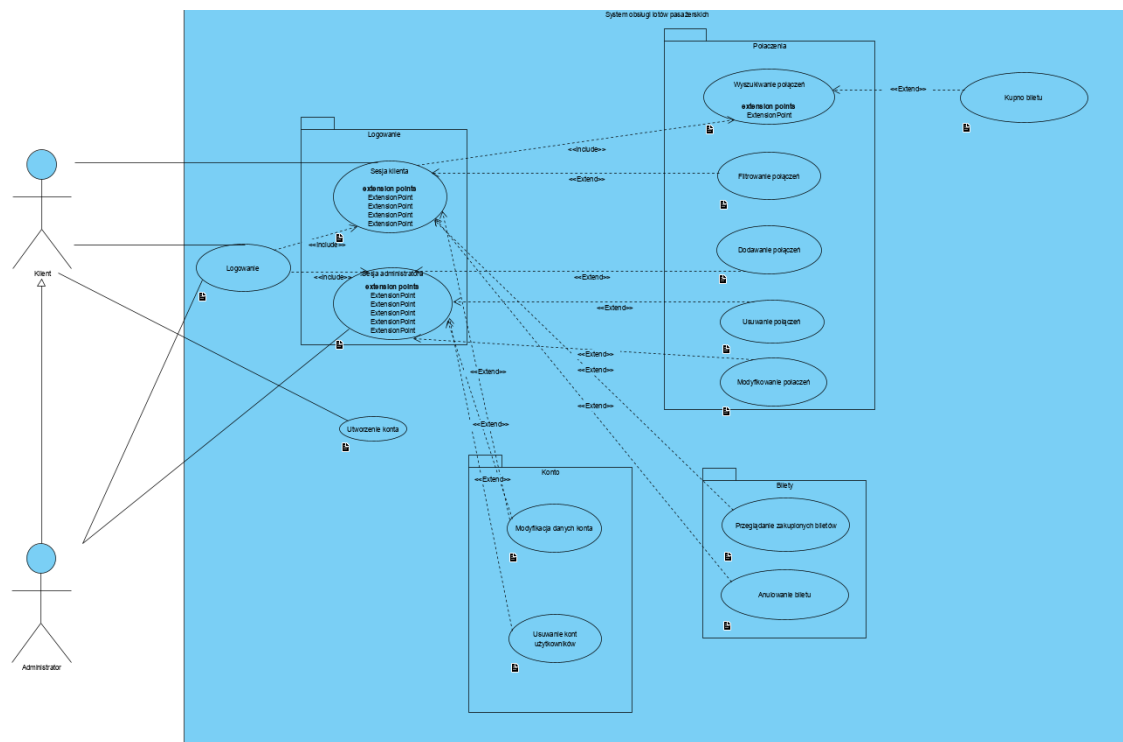
Rysunek 1. Schemat logiczny systemu

2.2. Wymagania funkcjonalne

- Administrator wprowadza dane lotu: numer lotu, miejsce wylotu, miejsce przylotu, data wylotu, klasa usługi, numer miejsca siedzącego, numer lotu.
- Administrator może dodawać nowe połączenia oraz modyfikować i usuwać dane już istniejących połączeń.
- Administrator może usuwać konta użytkowników.

- Administrator posiada wszystkie funkcjonalności klienta
- Klient może wyszukać lot spośród listy obecnie dostępnych.
- Klient może kupić usługę lotu spośród listy obecnie dostępnych.
- Klient może przeszukać zawartość listy według możliwych parametrów: miejsce wylotu, miejsce przylotu, data odlotu.
- Klient może założyć konto z loginem i hasłem, modyfikować dane osobowe, usuwać swoje konto oraz może rezerwować lot na swoje dane osobowe przypisane do konta.
- Klient może anulować wybrany zakupiony bilet.
- Klient ma dostęp do informacji o zakupionych biletach.

2.2.1. Diagram przypadków użycia



Rysunek 2. Diagram przypadków użycia

2.2.2. Scenariusze przypadków użycia

Poniżej znajdują się kolejne scenariusze przypadków użycia dla każdego przypadku użycia.

• Logowanie

OPIS

CEL: Logowanie się na konto w celu możliwości interakcji z programem

WS: Inicjalizacja poprzez uruchomienie programu

WK: Wprowadzenie poprawnych danych logowania bądź zamknięcie okna dialogowego

PRZEBIEG

1. Uruchomienie programu
2. Wypełnienie formularza logowania poprawnymi danymi
3. Potwierdzenie logowania klawiszem "Zaloguj"

• **Utworzenie konta**

OPIS

CEL: Utworzenie konta dla nowego użytkownika

WS: Inicjalizacja poprzez uruchomienie programu i wybranie odpowiedniej opcji na ekranie startowym

WK: Podanie następujących, poprawnych danych nowego użytkownika: imię, nazwisko, adres zamieszkania, adres email, numer telefonu, login, hasło

PRZEBIEG

1. Wybranie opcji "Założ konto" na ekranie startowym
2. Wypełnienie formularza poprawnymi danymi
3. Potwierdzenie operacji poprzez kliknięcie przycisku: "Potwierdź"

• **Sesja klienta**

OPIS

CEL: Obsługa programu za pośrednictwem konta klienta

WS: Inicjalizacja poprzez uruchomienie programu i zalogowanie się na konto klienta

WK: Wylogowanie się użytkownika z konta

PRZEBIEG

1. Wybieranie możliwych opcji w menu dialogowym
2. Po wykonaniu zamierzonych działań należy wylogować się z sesji.

• **Sesja administratora**

OPIS

CEL: Obsługa programu przy pomocy konta o uprawnieniach administratorskich

WS: Zalogowanie się na konto administratora

WK: Wylogowanie się z konta

PRZEBIEG

1. Wybranie odpowiednich opcji z menu dialogowego
2. Po wprowadzeniu zmian wylogowanie się z konta

• Modyfikacja danych konta

OPIS

CEL: Modyfikacja danych konta użytkownika

WS: Posiadanie odpowiednich uprawnień do edycji danych konta (klient może edytować tylko swoje dane, administrator może edytować dane wszystkich użytkowników). Ponadto należy wybrać odpowiednią opcję z ekranu menu użytkownika : "Edytuj dane konta"

WK: Wprowadzenie zmian w danych wyświetlanych na formularzu

PRZEBIEG

1. Wybranie opcji "Edytuj dane konta" na ekranie głównym aplikacji
2. Modyfikacja danych wyświetlanych na formularzu
3. Zatwierdzenie danych przyciskiem "Zatwierdź"

• Usuwanie kont użytkowników

OPIS

CEL: Usunięcie konta danego użytkownika

WS: Uruchomienie programu, otwarcie sesji administratora, wybranie odpowiedniej opcji w menu głównym

WK: Wpisanie danych usuwanego konta do formularza i zatwierdzenie zmian

PRZEBIEG:

1. Przy pomocy konta administratora wybranie w menu głównym opcji: "Usuń konto"
2. Wprowadzenie do formularza danych usuwanego konta
3. Potwierdzenie zmian przyciskiem: "Zatwierdź"

• Przeglądanie zakupionych biletów

OPIS

CEL: Podgląd danych zakupionego przez użytkownika biletu

WS: Uruchomienie programu, otwarcie sesji klienta, wybranie zakupionego biletu w odpowiednim oknie

WK: Wybranie przez użytkownika opcji "Wstecz"

PRZEBIEG

1. Wybranie przez użytkownika opcji "Podgląd" w oknie z zakupionymi biletami
2. Przegląd wyświetlanych w formularzu danych przez klienta
3. Zamknięcie wyświetlanego okna przyciskiem "Wstecz"

• Anulowanie biletu

OPIS

CEL: Anulowanie biletu zakupionego przez użytkownika

WS: Uruchomienie programu, otwarcie sesji klienta, wybranie wybranego biletu w odpowiednim oknie

WK: Po wybraniu biletu wciśnięcie przycisku : "Zrezygnuj"

PRZEBIEG

1. Wybranie zakupionego biletu za pośrednictwem odpowiedniego formularza
2. Wciśnięcie przycisku "Zrezygnuj"
3. Potwierdzenie wykonywanej operacji za pośrednictwem dodatkowego, wyświetlanego okna dialogowego

• Wyszukiwanie połączeń

OPIS

CEL: Wyszukanie lotu o znanym kodzie ID

WS: Inicjalizacja programu, otwarcie sesji klienta, wybranie odpowiedniej opcji w menu głównym

WK: Wybranie przycisku "Wstecz" na wyświetlanym formularzu

PRZEBIEG

1. Wybranie opcji "Wyszukaj połączenie" w menu głównym
2. Wprowadzenie do formularza numeru ID wyszukiwanego lotu

3. Wybranie jednej z opcji na wyświetlanym oknie dialogowym

• Filtrowanie połączeń

OPIS

CEL: Filtrowanie listy wyświetlanych lotów

WS: Inicjalizacja programu, otwarcie sesji klienta, wybranie odpowiedniej opcji w oknie wyświetlanych lotów

WK: Zamknięcie okna odpowiednim przyciskiem

PRZEBIEG

1. Wybranie przez klienta opcji "Filtruj loty" w menu głównym
2. Wpisanie do formularza odpowiednich danych do filtrowania listy
3. Zamknięcie wyświetlanego okna przyciskiem "Wstecz"

• Dodawanie połączeń

OPIS

CEL: Wprowadzenie nowego lotu do systemu

WS: Inicjalizacja programu, otwarcie sesji administratora, wybranie odpowiedniej opcji z menu głównego

WK: Zatwierdzenie wprowadzonych danych

PRZEBIEG

1. Wybranie opcji "Dodaj lot" w menu głównym
2. Wprowadzenie danych nowego lotu do formularza
3. Zatwierdzenie wprowadzanych zmian przyciskiem "Zatwierdź"

• Usuwanie połączeń

OPIS

CEL: Usunięcie danego lotu z bazy danych

WS: Uruchomienie programu, otwarcie sesji administratora, wybranie odpowiedniej opcji z menu głównego

WK: Zatwierdzenie operacji po wprowadzeniu danych usuwanego lotu

PRZEBIEG

1. Wybranie opcji "Usuń lot" z menu głównego
2. Wprowadzenie do formularza danych usuwanego lotu
3. Potwierdzenie wprowadzanych zmian przyciskiem "Zatwierdź"

• Modyfikowanie połączeń

OPIS

CEL: Modyfikacja danych lotu

WS: Uruchomienie programu, otwarcie sesji administratora, wybranie odpowiedniej opcji w menu głównym

WK: Zatwierdzenie wprowadzonych modyfikacji do formularza

PRZEBIEG

1. Wybranie opcji "Modyfikuj dane lotu"
2. Wprowadzenie do formularza numeru ID modyfikowanego lotu
3. Modyfikacja danych wyświetlanych na formularzu 4. Zatwierdzenie modyfikacji poprzez kliknięcie przycisku "Zatwierdź"

• Kupno biletu

OPIS

CEL: Zakup usługi lotu samolotem pasażerskim

WS: Inicjalizacja programu, otwarcie sesji klienta, wyszukanie lotu po odpowiednim numerze ID, wybranie odpowiedniej opcji na wyświetlanym oknie dialogowym

WK: Potwierdzenie zakupu usługi w oknie dialogowym

PRZEBIEG

1. Wpisanie do formularza numeru ID lotu
2. Po wyświetleniu się lotu wybranie opcji "Zakup usługę"
3. Zatwierdzenie wykonywanej operacji w dodatkowym oknie dialogowym

2.3. Wymagania niefunkcjonalne

- Aplikacja jest aplikacją desktopową, która ma połączenie z lokalną bazą danych,
- Aplikacja posiada przejrzysty i łatwy w obsłudze interfejs graficzny,
- Dodawanie nowych lotów może być dokonywane jedynie przez uprawnione osoby,
- Z jednego konta na jeden lot można zarezerwować maksymalnie 5 miejsc,
- Kupowanie, anulowanie rezerwacji lotów oraz aktualizowanie danych osobowych jest dokonywane przez klientów,
- Aplikacja powinna działać na systemach operacyjnych: Windows, Linux,
- Pojemność bazy danych powinna mieścić około 100 000 rekordów (bilety lotnicze),
- Oprogramowanie powinno być w stanie obsłużyć jednocześnie 1000 użytkowników,
- Technologia użyta do implementacji bazy danych – Microsoft SQL Server,
- Technologia użyta do napisania oprogramowania: Java.

2.4. Przyjęte założenia projektowe

W ramach projektu przyjęto, że aplikacja użytkownika będzie desktopowa, z własnym interfejsem graficznym. Zostanie ona wykonana obiektowo wykorzystując w tym celu biblioteki Swing, środowisko Idea IntelliJ oraz język Java. Korzystać ona będzie z relacyjnej bazy danych utworzonej na serwerze Microsoft SQL Server. Przepływ danych i ich przetwarzanie będzie się wykonywać z wykorzystaniem zapytań i procedur języka SQL. Wszystkie operacje można wykonać z poziomu aplikacji, w której zostanie też zaprezentowany wynik danego zapytania, oparty oczywiście na wspomnianej wcześniej bazie.

3. Projekt systemu

3.1. Projekt bazy danych

3.1.1. Analiza rzeczywistości i uproszczony model konceptualny

Opis zasobów ludzkich

Klient przy utworzeniu nowego konta wypełnia formularz, w którym podaje swoje dane osobowe oraz ustala login i hasło do konta. Klient loguje się do systemu, po zalogowaniu ma dostęp do listy dostępnych najbliższych lotów. Klient może kupić bilet na lot poprzez zaznaczenie danego lotu, oraz kliknięcie przycisku „Kup”. Po zatwierdzeniu klient może obserwować swoje bilety w zakładce „Moje Loty”. Klient może w zakładce „Moje loty” anulować zakup biletów. W zakładce „Ustawienia” klient może zmodyfikować swoje dane osobowe przypisane do konta.

Przepisy

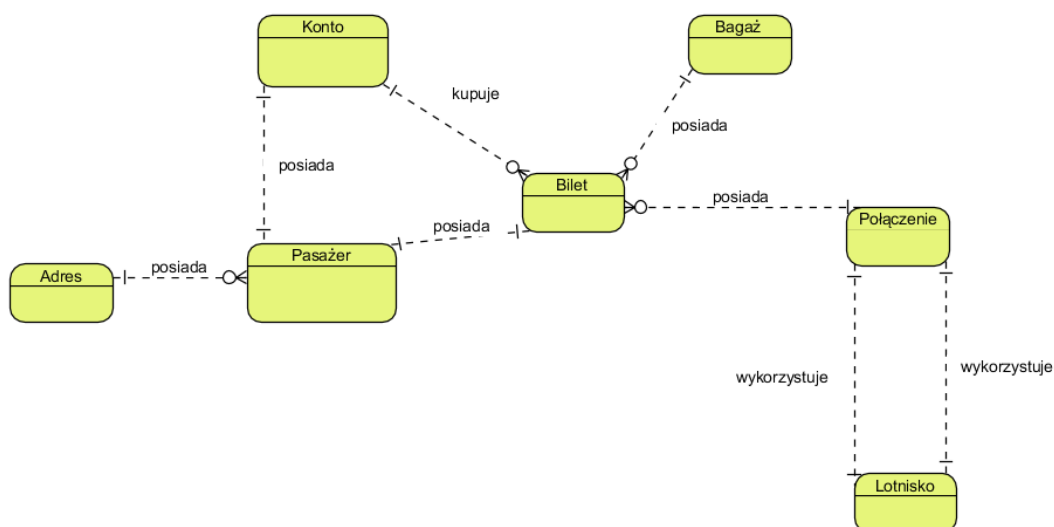
Pracownik ponosi odpowiedzialność za poprawność danych – odpowiada materialnie za niezgodność danych ze stanem właściwym. Klient ponosi odpowiedzialność za poprawność swoich danych osobowych i ponosi odpowiedzialność prawną za ich fałszowanie.

Dane techniczne

Klient może przeglądać dane połączenia za pośrednictwem aplikacji desktopowej. Zakłada się, że klientów jednocześnie przeglądających połączenia lotnicze może być ponad 1000 oraz firma może zawierać kilka tysięcy danych o klientach, kilkaset danych połączeń oraz kilka tysięcy danych o

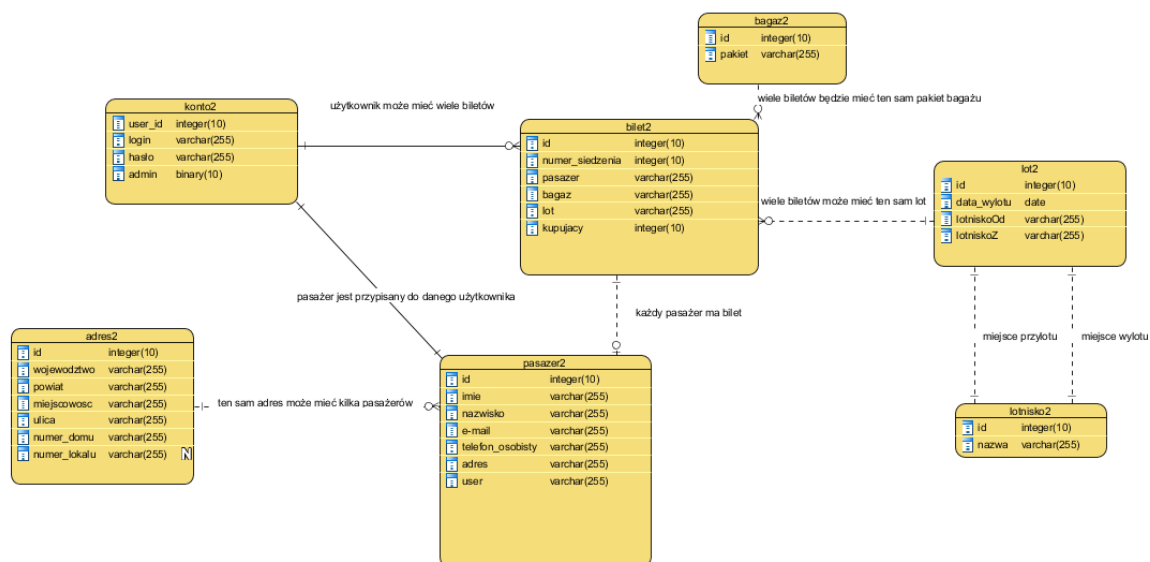
biletach. Firma składa się z jednej placówki w konkretnym miejscu na terenie kraju (szczegółowe dane dołączone są do umowy). Zaleca się stosowanie technologii Java.

Model konceptualny



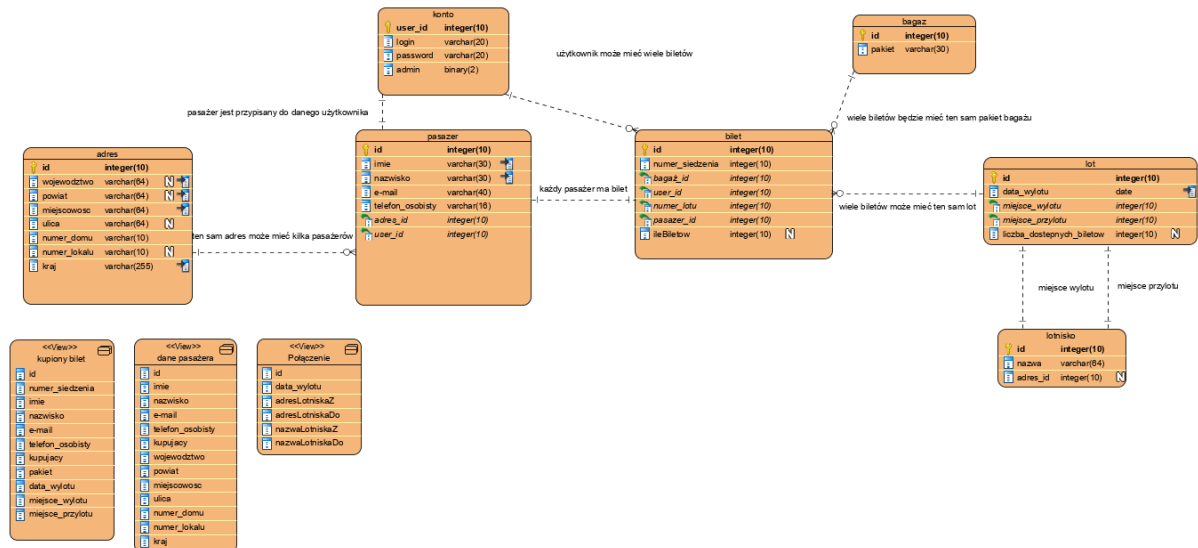
Rysunek 3. Uproszczony model konceptualny związków encji

3.1.2. Model logiczny i normalizacja



Rysunek 4. Model logiczny

3.1.3. Model fizyczny i ograniczenia integralności danych



Rysunek 5. Model fizyczny bazy danych wraz z ograniczeniami

Jak widać na załączonym wyżej rysunku, wszystkie dane w tabelach posiadają ograniczenia. Ponadto rekordy z pierwszej linii każdej z nich są kluczami podstawowymi. Niektóre z nich (jak pasazer_id w encji Bilet) są ponadto kluczami obcymi.

3.1.4. Inne elementy schematu – mechanizmy przetwarzania danych

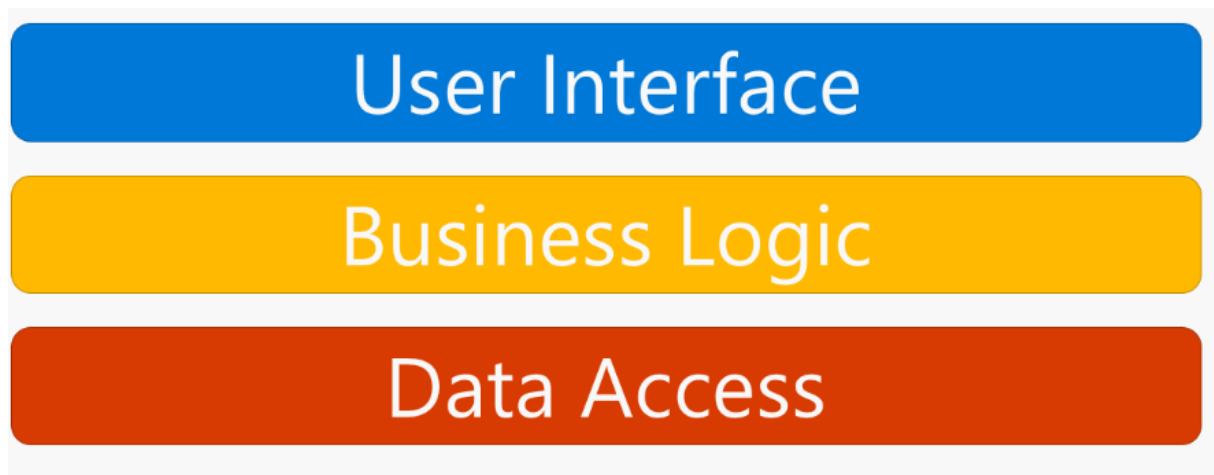
W ramach projektu bazy danych powstanie w niej widoki, które będą przedstawiać informacje zbiorcze z kilku tabel, np. bilet wraz ze wszystkimi danymi gotowymi do wydruku czy wszystkie dane pasażera wraz z adresem zamieszkania. Natomiast w ramach projektu nie jest przewidziane użycie wyzwalaczy.

3.1.5. Projekt mechanizmów bezpieczeństwa na poziomie bazy danych

Bezpieczeństwo zapewniać będzie logowanie się do bazy na różnych użytkowników: administrator oraz klient. Administrator ma pełny dostęp do wszystkich tabel, natomiast klient jedynie do tabeli lotów, pasażera oraz biletów.

3.2. Projekt aplikacji użytkownika

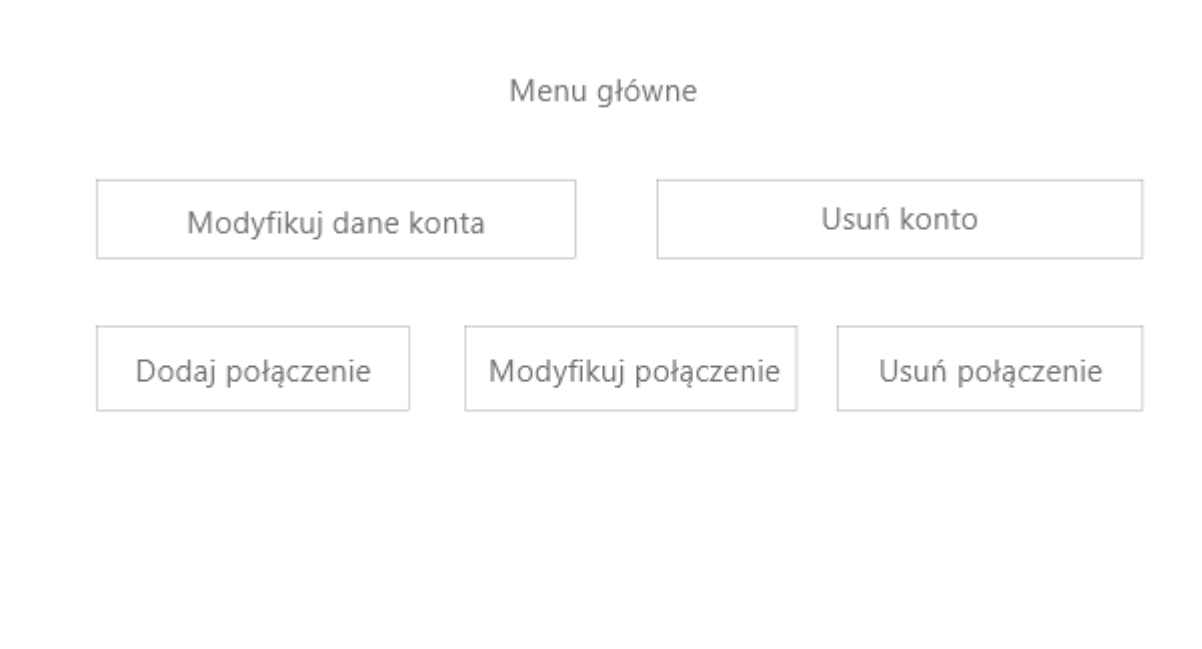
3.2.1. Architektura aplikacji i diagramy projektowe



Rysunek 6. Rysunek przedstawiający architekturę trzywarstwową aplikacji.

Aplikacja będzie miała architekturę trzywarstwową: warstwa interfejsu użytkownika (UI), warstwa logiki biznesowej (BL) oraz warstwa dostępu do danych (DA). Korzystając z tej architektury, użytkownicy wysyłają żądania za pośrednictwem warstwy UI, które współdziałają tylko z warstwą BL. Z kolei warstwa BL może wywoływać warstwę DA dla żądań dostępu do danych.

3.2.2. Interfejs graficzny i struktura menu



Rysunek 7. Menu główne – Administrator

Menu główne

Modyfikuj dane konta

Wyszukaj połączenie

Pokaż listę połączeń

Pokaż zakupione bilety

Rysunek 8. Menu główne – Klient

3.2.3. Projekt wybranych funkcji systemu

Dodawanie połączenia

Data wylotu: Data przylotu:

Miejsce wylotu:

Miejsce przylotu:

Bagaż:

Zatwierdź

Rysunek 9. Dodawanie połączenia – Administrator

Start

Logowanie

Założ konto

Rysunek 10. Ekran powitalny

Zakupione bilety:

Nr.	Data wylotu:	Data przylotu:	Miejsce wylotu:	Miejsce przylotu:	Bagaż:	Anuluj

Rysunek 11. Lista kupionych biletów

Lista połączeń

Nr.	Data wylotu:	Data przylotu:	Miejsce wylotu:	Miejsce przylotu:	Bagaż:	Zakup

Rysunek 12. Lista połączeń

Logowanie

Login:

Hasło:

Zaloguj

Rysunek 13. Logowanie

Modyfikowanie połączenia

Data wylotu:

Data przylotu:

Miejsce wylotu:

Miejsce przylotu:

Bagaż:

Zatwierdź

Rysunek 14. Modyfikowanie połączenia - Administrator

Usuwanie konta

Nr ID konta:

Zatwierdź

Rysunek 15. Usuwanie konta – Administrator

Usuwanie połączenia

Nr ID połączenia:

Zatwierdź

Rysunek 16. Usuwanie połączenia – Administrator

Wyszukiwanie połączenia:

Data wylotu: Data przylotu:

Miejsce wylotu:

Miejsce przylotu:

Bagaż:

Zatwierdź

Rysunek 17. Wyszukiwanie połączenia

Zakładanie konta

Imię:

Nazwisko:

Email:

Telefon:

Kraj:

Województwo:

Powiat:

Miejscowość:

Ulica:

Nr. domu:

Nr.lokalu:

Login:

Hasło:

Zatwierdź

Rysunek 18. Zakładanie konta

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

Podłączenie do bazy danych MS SQL Server zostało wykonane za pomocą sterownika Microsoft SQL Server 2019 JDBC Driver.

3.2.5. Projekt zabezpieczeń na poziomie aplikacji

Bezpieczeństwo zapewnić będzie logowanie się do aplikacji za pomocą loginu oraz hasła, które są przechowywane w bazie danych. Po poprawnej weryfikacji danych użytkownik otrzyma dostęp do aplikacji.

4. Implementacja systemu baz danych

Implementacja i testy bazy danych w wybranym systemie zarządzania bazą danych.

4.1. Tworzenie tabel i definiowanie ograniczeń // inna czcionka

Tabele zostały utworzone za pomocą komendy CREATE TABLE.

```
CREATE TABLE Konto(  
UserID int IDENTITY(1,1) PRIMARY KEY,  
Login nvarchar(20) NOT NULL ,  
Password nvarchar(20) NOT NULL ,  
Admin binary(2) NOT NULL  
);
```

```
CREATE TABLE Adres(  
AdresID int IDENTITY(1,1) PRIMARY KEY,  
Kraj nvarchar(30) NOT NULL,  
Wojewodztwo nvarchar(30),  
Powiat nvarchar(30),  
Miejscowosc nvarchar(30) NOT NULL ,  
Ulica nvarchar(30),  
Numer_domu nvarchar(30) NOT NULL,  
Numer_lokalu nvarchar(30),  
);
```

```
CREATE TABLE Pasazer(  
PasazerID int IDENTITY(1,1) PRIMARY KEY,  
Imie nvarchar(30) NOT NULL,  
Nazwisko nvarchar(30) NOT NULL,  
Email nvarchar(40) NOT NULL,  
Telefon_osobisty nvarchar(16) NOT NULL,  
AdresID int FOREIGN KEY REFERENCES Adres(AdresID),  
UserID int FOREIGN KEY REFERENCES Konto(UserID),  
);
```

```
CREATE TABLE Bagaz(  
BagazID int IDENTITY(1,1) PRIMARY KEY,  
Pakiet nvarchar(30) NOT NULL ,  
);
```

```
CREATE TABLE Lotnisko(  
LotniskoID int IDENTITY(1,1) PRIMARY KEY,  
Nazwa nvarchar(30) NOT NULL ,  
AdresID int FOREIGN KEY REFERENCES Adres(AdresID)  
);
```

```
CREATE TABLE Lot(  
LotID int IDENTITY(1,1) PRIMARY KEY,  
Data_wylotu date NOT NULL ,  
Miejsce_wylotu int,  
Miejsce_przylotu int,  
);
```

```
CREATE TABLE Bilet(
BiletID int IDENTITY(1,1) PRIMARY KEY,
Numer_siedzenia int NOT NULL ,
BagazID int FOREIGN KEY REFERENCES Bagaz(BagazID),
UserID int FOREIGN KEY REFERENCES Konto(UserId),
Numer_lotu int FOREIGN KEY REFERENCES Lot(LotID),
PasazerID int FOREIGN KEY REFERENCES Pasazer(PasazerID),
Status nvarchar(30)
);
```

4.2. Implementacja mechanizmów przetwarzania danych

Widoki zostały stworzone za pomocą komendy CREATE VIEW.

```
CREATE VIEW DanePasazera AS
SELECT p.PasazerID, p.Imie, p.Nazwisko, p.Email, p.Telefon_osobisty, p.UserID AS Numer_Konta,
a.Kraj,
a.Wojewodztwo, a.Powiat, a.Miejscowosc,a.Ulica, a.Numer_domu, a.Numer_lokalu
FROM Pasazer p JOIN Adres a ON p.AdresID = a.AdresID;
```

```
CREATE VIEW Przeloty AS
SELECT l.LotID, l.Data_Wylotu, lotnWy.Nazwa AS Miejsce_wylotu, lotnPrzy.Nazwa AS
Miejsce_przylotu
FROM Lot l FULL JOIN Lotnisko lotnWy ON l.Miejsce_wylotu = lotnWy.LotniskoID FULL JOIN
Lotnisko lotnPrzy ON l.Miejsce_przylotu = lotnPrzy.LotniskoID ;
```

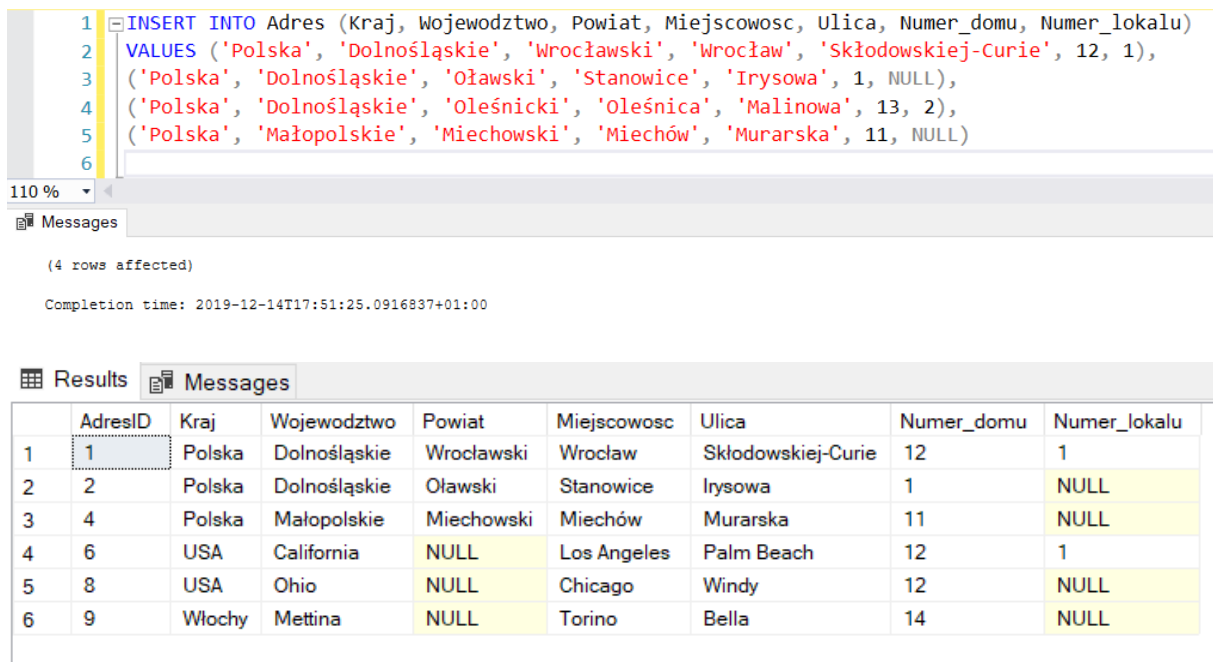
4.3. Implementacja uprawnień i innych zabezpieczeń

Uprawnienia zostały nadane za pomocą komendy GRANT.

```
USE FlightStore
GO
GRANT ALL PRIVILEGES ON Bilet TO Admin
GRANT ALL PRIVILEGES ON Lot TO Admin
GRANT ALL PRIVILEGES ON Pasazer TO Admin
GRANT ALL PRIVILEGES ON Lotnisko TO Admin
GRANT ALL PRIVILEGES ON Konto TO Admin
GRANT ALL PRIVILEGES ON Adres TO Admin
GRANT ALL PRIVILEGES ON Bagaz TO Admin
```

```
USE FlightStore
GO
GRANT SELECT, INSERT, DELETE ON Bilet TO Klient
GRANT SELECT, INSERT, UPDATE ON Pasazer TO Klient
GRANT SELECT, INSERT, UPDATE ON Adres TO Klient
GRANT SELECT, INSERT, UPDATE ON Konto TO Klient
GRANT SELECT ON Lot TO Klient
```

4.4. Testowanie bazy danych na przykładowych danych



The screenshot shows a SQL query window with an INSERT statement and its results. The query inserts four rows into the 'Adres' table. The results table shows the inserted data with columns: AdresID, Kraj, Wojewodztwo, Powiat, Miejscowosc, Ulica, Numer_domu, and Numer_lokalu.

```
1 INSERT INTO Adres (Kraj, Wojewodztwo, Powiat, Miejscowosc, Ulica, Numer_domu, Numer_lokalu)
2 VALUES ('Polska', 'Dolnośląskie', 'Wrocławski', 'Wrocław', 'Skłodowskiej-Curie', 12, 1),
3 ('Polska', 'Dolnośląskie', 'Oławski', 'Stanowice', 'Irysowa', 1, NULL),
4 ('Polska', 'Dolnośląskie', 'Oleśnicki', 'Oleśnica', 'Malinowa', 13, 2),
5 ('Polska', 'Małopolskie', 'Miechowski', 'Miechów', 'Murarska', 11, NULL)
6
```

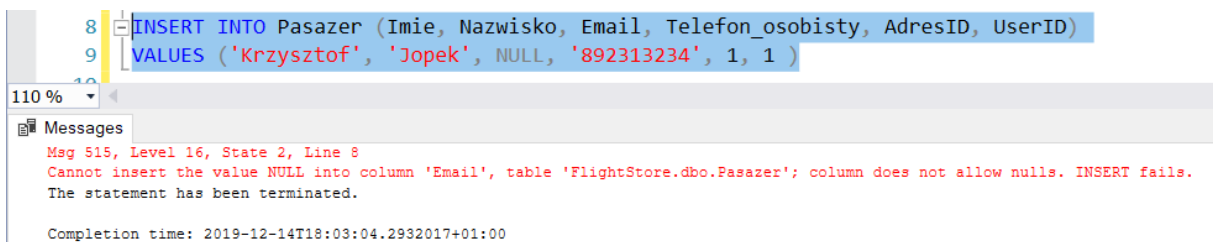
Messages

(4 rows affected)

Completion time: 2019-12-14T17:51:25.0916837+01:00

	AdresID	Kraj	Wojewodztwo	Powiat	Miejscowosc	Ulica	Numer_domu	Numer_lokalu
1	1	Polska	Dolnośląskie	Wrocławski	Wrocław	Skłodowskiej-Curie	12	1
2	2	Polska	Dolnośląskie	Oławski	Stanowice	Irysowa	1	NULL
3	4	Polska	Małopolskie	Miechowski	Miechów	Murarska	11	NULL
4	6	USA	California	NULL	Los Angeles	Palm Beach	12	1
5	8	USA	Ohio	NULL	Chicago	Windy	12	NULL
6	9	Włochy	Mettina	NULL	Torino	Bella	14	NULL

Rysunek 19. Poprawne wstawienie rekordów do tabeli "Adres"



The screenshot shows a SQL query window with an INSERT statement that fails because it tries to insert a NULL value into the 'Email' column, which is NOT NULL. The error message is displayed in the Messages pane.

```
8 INSERT INTO Pasazer (Imie, Nazwisko, Email, Telefon_osobisty, AdresID, UserID)
9 VALUES ('Krzysztof', 'Jopek', NULL, '892313234', 1, 1)
```

Messages

Msg 515, Level 16, State 2, Line 8
Cannot insert the value NULL into column 'Email', table 'FlightStore.dbo.Pasazer'; column does not allow nulls. INSERT fails.
The statement has been terminated.

Completion time: 2019-12-14T18:03:04.2932017+01:00

Rysunek 20. Test wstawienia wartości NULL do kolumny NOT NULL zakończona niepowodzeniem.



The screenshot shows a SQL query window with a DELETE statement that successfully removes the record with AdresID = 3. The Messages pane shows the success message. The table 'Adres' is displayed with the remaining records.

```
4
5 DELETE FROM Adres WHERE AdresID = 3;
```

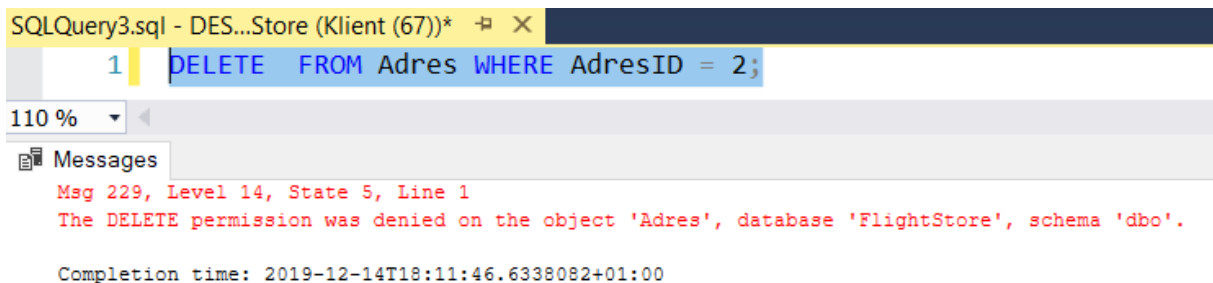
Messages

(1 row affected)

Completion time: 2019-12-14T18:07:58.2556448+01:00

AdresID	Kraj	Wojewodztwo	Powiat	Miejscowosc	Ulica	Numer_do...	Numer_lok...
1	Polska	Dolnośląskie	Wrocławski	Wrocław	Skłodowski...	12	1
2	Polska	Dolnośląskie	Oławski	Stanowice	Irysowa	1	NULL
4	Polska	Małopolskie	Miechowski	Miechów	Murarska	11	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Rysunek 21. Poprawne usuwanie rekordów z tabeli "Adres"



The screenshot shows a SQL query window with a DELETE statement that fails because the user does not have the necessary permissions to delete from the 'Adres' table. The error message is displayed in the Messages pane.

```
1 DELETE FROM Adres WHERE AdresID = 2;
```

Messages

Msg 229, Level 14, State 5, Line 1
The DELETE permission was denied on the object 'Adres', database 'FlightStore', schema 'dbo'.

Completion time: 2019-12-14T18:11:46.6338082+01:00

Rysunek 22. Test usunięcia rekordu z tabeli "Adres" zakończona niepowodzeniem.

	BiletID	Numer_sie...	Imie	Nazwisko	Email	Telefon_os...	Numer_Ku...	Pakiet	Data_wylotu	MiejsceWyl...	MiejcePrzyl...
▶	401	1	Krzysztof	Jopek	krzysiek@w...	898324123	2	Podręczny	2019-12-17	Los Angeles	Chicago
	402	1	Krzysztof	Jopek	krzysiek@w...	898324123	2	Podręczny	2019-12-17	Los Angeles	Chicago
	403	1	Krzysztof	Jopek	krzysiek@w...	898324123	2	Podręczny	2019-12-17	Los Angeles	Chicago
	404	1	Krzysztof	Jopek	krzysiek@w...	898324123	2	Podręczny	2019-12-17	Los Angeles	Chicago
	405	1	Krzysztof	Jopek	krzysiek@w...	898324123	2	Podręczny	2019-12-17	Los Angeles	Chicago
	406	1	Krzysztof	Jopek	krzysiek@w...	898324123	2	Podręczny	2019-12-17	Los Angeles	Chicago
	407	1	Krzysztof	Jopek	krzysiek@w...	898324123	2	Podręczny	2019-12-17	Los Angeles	Chicago
	408	1	Krzysztof	Jopek	krzysiek@w...	898324123	2	Podręczny	2019-12-17	Los Angeles	Chicago
	409	1	Krzysztof	Jopek	krzysiek@w...	898324123	2	Podręczny	2019-12-17	Los Angeles	Chicago
	410	1	Krzysztof	Jopek	krzysiek@w...	898324123	2	Podręczny	2019-12-17	Los Angeles	Chicago
	411	1	Krzysztof	Jopek	krzysiek@w...	898324123	2	Podręczny	2019-12-17	Los Angeles	Chicago
	412	1	Krzysztof	Jopek	krzysiek@w...	898324123	2	Podręczny	2019-12-17	Los Angeles	Chicago
	413	1	Krzysztof	Jopek	krzysiek@w...	898324123	2	Podręczny	2019-12-17	Los Angeles	Chicago

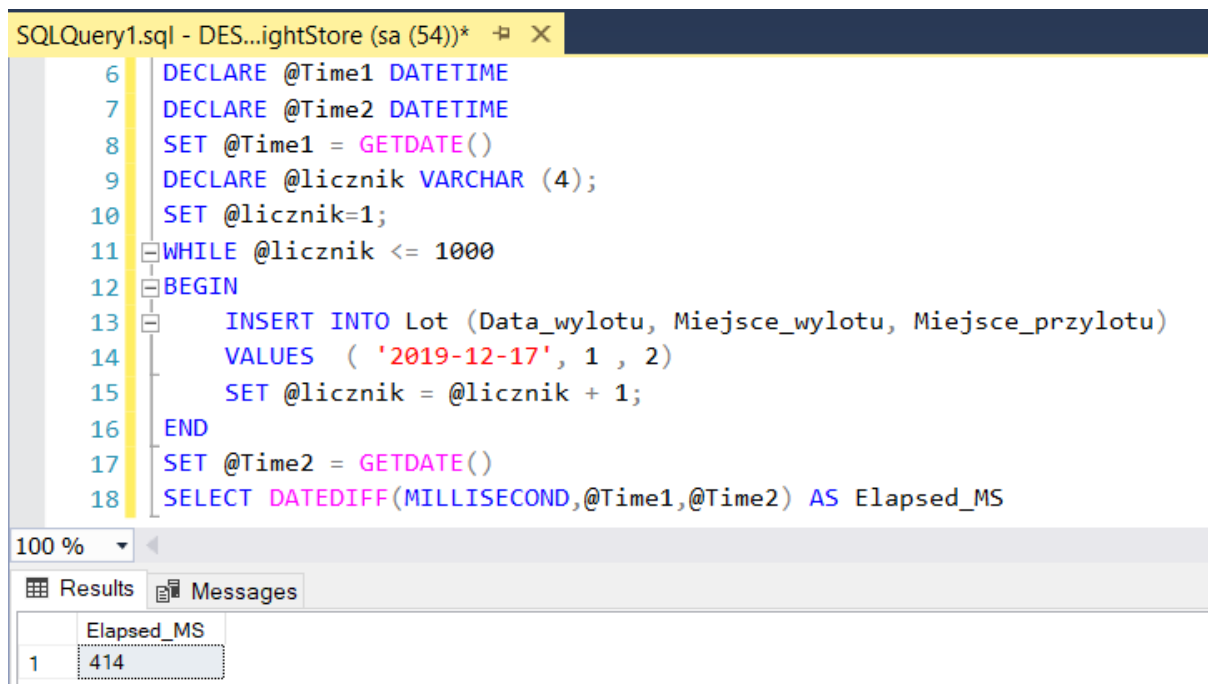
Rysunek 23. Widok "Bilety" po wstawieniu 100 jednakowych biletów.

	PasazerID	Imie	Nazwisko	Email	Telefon_os...	Numer_Ko...	Kraj	Wojewodzt...	Powiat	Miejscowosc	Ulica	Numer_do...	Numer_lok...
▶	2	Krzysztof	Jopek	krzysiek@w...	898324123	2	Polska	Dolnośląskie	Wrocławski	Wrocław	Skłodowski...	12	1
	3	Anna	Żarczyńska	anna@op.pl	982341234	4	Polska	Dolnośląskie	Oławski	Stanowice	Irysowa	1	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Rysunek 24. Widok "Dane Pasażera" wraz z przykładowymi pasażerami.

	11	12	13	14	15	16	17	18	19	20	21	22	23	24
		DECLARE	@Time1	DATETIME										
		DECLARE	@Time2	DATETIME										
		SET	@Time1	=	GETDATE()									
		DECLARE	@licznik	VARCHAR	(3);									
		SET	@licznik	=	1;									
		WHILE	@licznik	<=	100									
		BEGIN												
		INSERT	INTO	Bilet	(Numer_siedzenia, BagazID, UserID, Numer_lotu, PasazerID)									
		VALUES	(1, 1, 1, 1, 2)											
		SET	@licznik	=	@licznik + 1;									
		END												
		SET	@Time2	=	GETDATE()									
		SELECT	DATEDIFF	(MILLISECOND, @Time1, @Time2) AS Elapsed_MS										
100 %														
Results		Messages		Client Statistics										
		Elapsed_MS												
1		23												

Rysunek 25. Test wydajnościowy dla wstawiania 100 nowych biletów.



```
SQLQuery1.sql - DES...ightStore (sa (54))* X
6 DECLARE @Time1 DATETIME
7 DECLARE @Time2 DATETIME
8 SET @Time1 = GETDATE()
9 DECLARE @licznik VARCHAR (4);
10 SET @licznik=1;
11 WHILE @licznik <= 1000
12 BEGIN
13     INSERT INTO Lot (Data_wylotu, Miejsce_wylotu, Miejsce_przylotu)
14     VALUES ( '2019-12-17', 1 , 2)
15     SET @licznik = @licznik + 1;
16 END
17 SET @Time2 = GETDATE()
18 SELECT DATEDIFF(MILLISECOND,@Time1,@Time2) AS Elapsed_MS
```

100 %

Results Messages

	Elapsed_MS
1	414

Rysunek 26. Test wydajnościowy dla wstawiania 1000 nowych lotów.

5. Implementacja i testy aplikacji

5.1. Instalacja i konfigurowanie systemu

W celu uruchomienia oraz poprawnego działania aplikacji użytkownik potrzebuje:

- zainstalowanej wirtualnej maszyny Java – Java Runtime Environment w wersji 11 lub nowszej,
- zainstalowany system zarządzania bazą danych Microsoft SQL Server wraz z postawionym serwerem oraz utworzoną bazą danych posiadającą wymagane tabele oraz zawarte w nich dane z wygenerowanego wcześniej skryptu,
- system operacyjny Windows 10

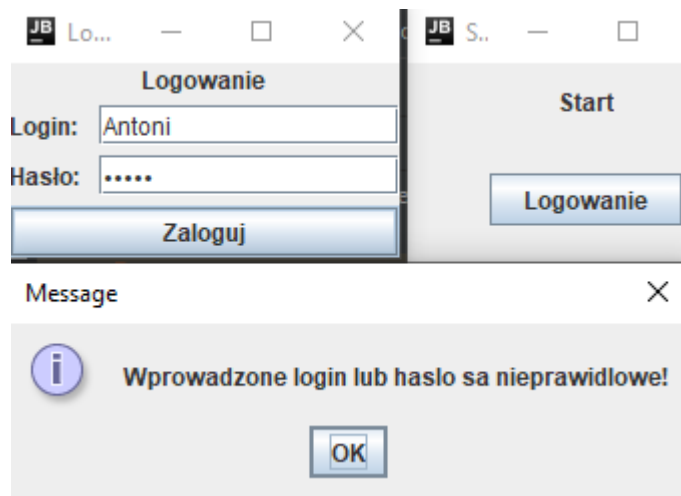
5.2. Instrukcja użytkownika aplikacji

Użytkownik, aby wejść do systemu, musi się zalogować wprowadzając swój login oraz hasło, a następnie nacisnąć przycisk logowania. Jeżeli użytkownik nie posiada swojego hasła, może je utworzyć przechodząc do okienka tworzenia nowego konta i wypełniając formularz swoimi danymi.

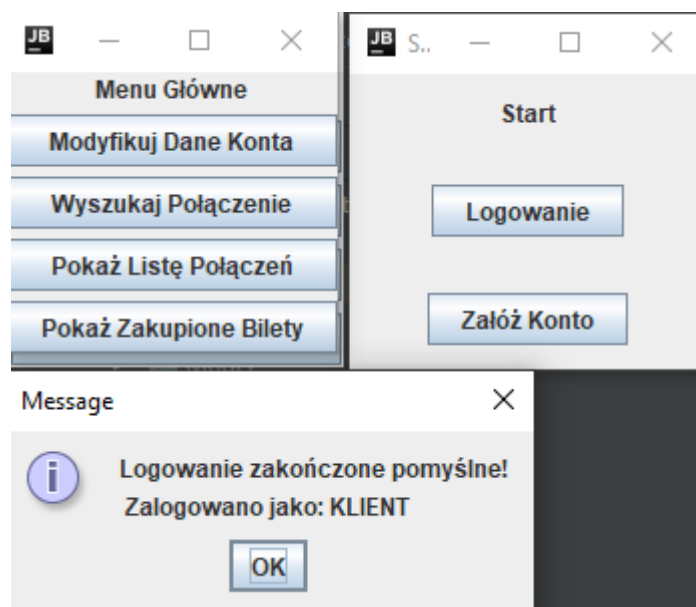
Po zalogowaniu do systemu, jeżeli użytkownik jest klientem, ma możliwość przeglądania aktywnych lotów poprzez przycisk <Pokaż listę połączeń> oraz wybór jednego z nich w celu kupienia biletu. Aby kupić bilet klient musi wybrać pakiet bagażu na dany lot, następnie nacisnąć przycisk <Kupuj>. Po zatwierdzeniu, bilet zostaje dodany do listy kupionych biletów wraz z danymi pasażera, lotu oraz bagażu, którą klient może przeglądać w osobnym okienku naciskając przycisk <Pokaż zakupione bilety> w menu głównym. Klient może również bilet wyszukać naciskając przycisk <Wyszukaj połączenie>, a następnie wpisując odpowiednie parametry. Ponadto klient może edytować swoje dane osobowe w osobnym formularzu naciskając przycisk <Modyfikuj dane konta>.

Jeżeli użytkownik jest administratorem, oprócz uprawnień klienckich, ma możliwość dodawania, modyfikowania oraz usuwania lotów, modyfikowania oraz usuwania kont klientów.

5.3. Testowanie opracowanych funkcji systemu



Rysunek 27. Nieudane logowanie.



Rysunek 28. Udana logowanie.

JB Lista Połączeń

Lista Połączeń					
ID	Data_wylotu	Miejsce_wylotu	Miejsce_przylotu	Bagaż	Zakup
4	2020-04-14	Berlin	Londyn	MAŁY	KUP TERAZ
5	2021-04-20	Berlin	Paryż	MAŁY	KUP TERAZ
6	2012-01-12	Paryż	Berlin	MAŁY	KUP TERAZ
7	2021-04-20	Berlin	Paryż	MAŁY	KUP TERAZ
8	2021-04-20	Londyn	Paryż	MAŁY	KUP TERAZ

Rysunek 29. Lista połączeń.

JB Lista Połączeń

Lista Połączeń					
ID	Data_wylotu	Miejsce_wylotu	Miejsce_przylotu	Bagaż	Zakup
4	2020-04-14	Berlin	Londyn	MAŁY	KUP TERAZ
5	2021-04-20	Berlin	Paryż	ŚREDNI	KUP TERAZ
6	2012-01-12	Paryż	Berlin	MAŁY	KUP TERAZ
7	2021-04-20	Berlin	Paryż	MAŁY	KUP TERAZ
8	2021-04-20	Londyn	Paryż	MAŁY	KUP TERAZ

Rysunek 30. Kupowanie biletu.

JB Modyfikowanie Danych Konta

Modyfikacja Danych Konta

Imię: Nazwisko:

Email: Telefon:

Kraj: Województwo:

Powiat: Miejscowość:

Ulica:

Nr.domu: Nr.lokalu:

Login: Hasło:

Message

Dane zostały zmodyfikowane!

Rysunek 31. Modyfikacja danych konta.

Wyszukiwanie Połączenia

Data wylotu: 2021-04-20

Miejsce wylotu: Londyn

Miejsce Przylotu: Paryż

Zatwierdź

Rysunek 32. Wyszukiwanie połączenia.

Lista Połączeń

ID	Data_wylotu	Miejsce_wylotu	Miejsce_przylotu	Bagaż	Zakup
8	2021-04-20	Londyn	Paryż	MAŁY	KUP TERAZ

Rysunek 33. Lista wyszukiwanych połączeń.

Zakupione Bilety

ID	Data_wylotu	Miejsce_wylotu	Miejsce_przylotu	Numer_Siedze...	Bagaż	Anulowanie
5	2021-04-20	Berlin	Paryż	1	ŚREDNI	ANULUJ
8	2021-04-20	Londyn	Paryż	1	MAŁY	ANULUJ
8	2021-04-20	Londyn	Paryż	2	MAŁY	ANULUJ
7	2021-04-20	Berlin	Paryż	1	MAŁY	ANULUJ
6	2012-01-12	Paryż	Berlin	1	DUŻY	ANULUJ
8	2021-04-20	Londyn	Paryż	3	MAŁY	ANULUJ

Rysunek 34. Lista zakupionych biletów.

5.4. Omówienie wybranych rozwiązań programistycznych

5.4.1. Implementacja interfejsu dostępu do bazy danych

Aplikacja łączy się z bazą danych za pomocą interfejsu JDBC (Java DataBase Connectivity) ze sterownikiem do Microsoft SQL Server. Połączenie, rozłączenie z bazą oraz wysyłanie zapytań jest wykonywane za pomocą metod z klasy SQLUtilities: **Connect(String URL)**, **Disconnect()**, **ExecuteNonQuery(PreparedStatement p)** oraz **ExecuteQuery(PreparedStatement sql)**;

```

public class SQLUtilities {
    //public static final String URL = "jdbc:sqlserver://localhost;databaseName=FlightStore;user=Admin;password=admin;";
    public static Connection connection;
    public SQLUtilities(){ }
    public static boolean Connect(String URL){
        try{
            Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
            connection = DriverManager.getConnection(URL);
            return true;
        }
        catch(ClassNotFoundException ex){
            return false;
        }
        catch(SQLException ex){
            System.out.println(ex);
            return false;
        }
    }
    public static boolean Disconnect(){
        try{
            connection.close();
            return true;
        }
        catch(SQLException ex){
            return false;
        }
    }
    public static int ExecuteNonQuery(PreparedStatement preparedStatement){
        try{
            return preparedStatement.executeUpdate();
        }
        catch(SQLException ex){
            return -1;
        }
    }
    public static ResultSet ExecuteQuery(PreparedStatement sql){
        try{
            return sql.executeQuery();
        }
        catch(SQLException ex){
            Logger.getLogger(SQLUtilities.class.getName()).log(Level.SEVERE, "msg: null, ex);
            return null;
        }
    }
}

```

Rysunek 35. Klasa SQLUtilities

5.4.2. Implementacja wybranych funkcjonalności systemu

```
public static void dodajBilet(String [] dane){
    String URL = "jdbc:sqlserver://localhost;databaseName=FlightStore;user=Admin;password=admin;";
    try {
        SQLUtilities.Connect(URL);
        int bagazID = 0, userID = 0, pasazerID = 0, iloscSiedzen = 0;

        String zapytanie = "SELECT BagazID FROM Bagaz WHERE Pakiet = ?";
        PreparedStatement s = SQLUtilities.connection.prepareStatement(zapytanie);
        s.setString(1, dane[1]);
        ResultSet r = SQLUtilities.ExecuteQuery(s);
        if(r.next()) bagazID = r.getInt(1);

        zapytanie = "SELECT UserID FROM Konto WHERE Login = ?";
        s = SQLUtilities.connection.prepareStatement(zapytanie);
        s.setString(1, Aplikacja.Login);
        r = SQLUtilities.ExecuteQuery(s);
        if(r.next()) userID = r.getInt(1);

        zapytanie = "SELECT PasazerID FROM Pasazer WHERE UserID = ?";
        s = SQLUtilities.connection.prepareStatement(zapytanie);
        s.setInt(1, userID);
        r = SQLUtilities.ExecuteQuery(s);
        if(r.next()) pasazerID = r.getInt(1);

        zapytanie = "SELECT COUNT(*) FROM Bilet WHERE Numer_Lotu = ?";
        s = SQLUtilities.connection.prepareStatement(zapytanie);
        s.setInt(1, Integer.parseInt(dane[0]));
        r = SQLUtilities.ExecuteQuery(s);
        if(r.next()) iloscSiedzen = r.getInt(1) + 1;

        String zapytanie1 = "INSERT INTO Bilet (Numer_siedzenia, Status, BagazID, UserID, Numer_lotu, PasazerID) VALUES (?, ?, ?, ?, ?, ?)";
        s = SQLUtilities.connection.prepareStatement(zapytanie1);
        s.setInt(1, iloscSiedzen);
        s.setString(2, "ZAKUPIONY");
        s.setInt(3, bagazID);
        s.setInt(4, userID);
        s.setInt(5, Integer.parseInt(dane[0]));
        s.setInt(6, pasazerID);
        SQLUtilities.ExecuteNonQuery(s);
        JOptionPane.showMessageDialog(null, "Bilet został zakupiony!");
        SQLUtilities.connection.close();
    } catch (SQLException ex) {
        JOptionPane.showMessageDialog(null, ex);
    }
}
```

Rysunek 36. Metoda dodajBilet() dodająca nowy bilet do bazy danych.

```

public static void usunKonto(int ID){
    String URL = "jdbc:sqlserver://localhost;databaseName=FlightStore;user=Admin;password=admin;";
    try {
        SQLUtilities.Connect(URL);
        String zapytanie = "SELECT UserID, login FROM Konto WHERE UserID = ?";
        PreparedStatement s = SQLUtilities.connection.prepareStatement(zapytanie);
        s.setInt( parameterIndex: 1,ID);
        ResultSet r = SQLUtilities.ExecuteQuery(s);
        if(r.next()) {
            int odczyt1 = r.getInt( columnIndex: 1);
            String odczyt2 = r.getString( columnIndex: 2);
            if (odczyt1 == ID) {
                zapytanie = "DELETE FROM Konto WHERE UserID = ?";
                s = SQLUtilities.connection.prepareStatement(zapytanie);
                s.setInt( parameterIndex: 1, ID);
                int potwierdzenie = JOptionPane.showConfirmDialog( parentComponent: null, message: "Czy na pewno usunąć rekord?", title: "Potwierdzenie", JOptionPane.YES_NO_OPTION);
                if(potwierdzenie == 1){
                    SQLUtilities.connection.close();
                    return;
                }
                SQLUtilities.ExecuteNonQuery(s);
                JOptionPane.showMessageDialog( parentComponent: null, message: "Usunięto konto o loginie:" + odczyt2);
            }
            else{
                JOptionPane.showMessageDialog( parentComponent: null, message: "Wprowadzony numer ID nie istnieje!");
            }
        }
        else {
            JOptionPane.showMessageDialog( parentComponent: null, message: "Wprowadzony numer ID nie istnieje!");
        }
        SQLUtilities.connection.close();
    }
    catch(SQLException ex){
        JOptionPane.showMessageDialog( parentComponent: null, ex);
    }
}

```

Rysunek 37. Metoda usunKonto(int ID) usuwająca konto z bazy o podanym numerze ID.

```

public static String[] uzupełnij(int numerID){
    String URL = "jdbc:sqlserver://localhost;databaseName=FlightStore;user=Admin;password=admin;";
    String [] dane1 = null;
    try {
        SQLUtilities.Connect(URL);
        String zapytanie = "SELECT Data_wylotu, Miejsce_wylotu, Miejsce_przylotu FROM Lot WHERE LotID = ?";
        PreparedStatement s = SQLUtilities.connection.prepareStatement(zapytanie);
        s.setInt( parameterIndex: 1, numerID);
        ResultSet r = SQLUtilities.ExecuteQuery(s);
        if (r.next()) {
            Date data = r.getDate( columnIndex: 1);
            DateFormat format = new SimpleDateFormat( pattern: "yyyy-MM-dd");
            String dataString = format.format(data);
            String [] dane = {dataString, r.getString( columnIndex: 2), r.getString( columnIndex: 3)};
            SQLUtilities.connection.close();
            return dane;
        }
        else{
            JOptionPane.showMessageDialog( parentComponent: null, message: "Błąd przy odczycie danych z bazy");
            SQLUtilities.connection.close();
        }
    }
    catch(SQLException ex){
        JOptionPane.showMessageDialog( parentComponent: null, ex);
    }
    return dane1;
}

```

Rysunek 38. Metoda uzupełnij(int ID) wypełniająca formularz do edycji danymi połączenia o wybranym numerze ID.

```

public static void modyfikujPolaczenie(Date data, int [] dane){

    String URL = "jdbc:sqlserver://localhost;databaseName=FlightStore;user=Admin;password=admin;";
    try {
        SQLUtilities.Connect(URL);
        String zapytanie = "UPDATE Lot SET Data_wylotu = ? , Miejsce_wylotu = ?, Miejsce_przylotu = ? WHERE LotID = ?";
        PreparedStatement s = SQLUtilities.connection.prepareStatement(zapytanie);
        s.setDate( parameterIndex: 1,data);
        s.setInt( parameterIndex: 2, dane[1]);
        s.setInt( parameterIndex: 3, dane[2]);
        s.setInt( parameterIndex: 4, dane[0]);
        SQLUtilities.ExecuteNonQuery(s);

        JOptionPane.showMessageDialog( parentComponent: null, message: "Dane zostały pomyslnie wprowadzone do bazy danych");
        SQLUtilities.connection.close();
        MenuGlowneAdmin.frameModyfikujPolaczenie.setVisible(false);
    }
    catch(SQLException ex){
        JOptionPane.showMessageDialog( parentComponent: null, ex);
    }
}

```

Rysunek 39. Metoda modyfikujPolaczenie(Date data, int[] dane) wprowadza nowe dane do wybranego istniejącego lotu w bazie.


```

public static String [] podajDane(){

String URL = "jdbc:sqlserver://localhost;databaseName=FlightStore;user=Admin;password=admin;";
String [] dane1 = new String[1];
try {
    int rozmiar = 0;
    SQLUtilities.Connect(URL);
    // String zapytanie = "SELECT LotID, Data_wylotu, Miejsce_wylotu, Miejsce_przylotu FROM Lot";
    String liczba = "SELECT count(*) FROM Lot";
    PreparedStatement l = SQLUtilities.connection.prepareStatement(liczba);
    ResultSet wynik = SQLUtilities.ExecuteQuery(l);
    if(wynik.next()) rozmiar = wynik.getInt( "columnIndex: 1");
    rozmiar = rozmiar * 4;
    String [] dane = new String[rozmiar];

    String zapytanie = "SELECT * FROM Lot";
    PreparedStatement s = SQLUtilities.connection.prepareStatement(zapytanie);
    ResultSet r = SQLUtilities.ExecuteQuery(s);
    int i = 1;
    Date data;
    DateFormat format = new SimpleDateFormat( "pattern: yyyy-MM-dd");
    ResultSet temp;
    while(r.next()) {
        dane[i-1] = String.valueOf(r.getInt( "columnLabel: LotID"));
        i++;
        data = r.getDate( "columnLabel: Data_wylotu");
        dane[i-1] = format.format(data);
        i++;
        String zapytanie1 = "SELECT Nazwa FROM Lotnisko WHERE LotniskoID = ?";
        s = SQLUtilities.connection.prepareStatement(zapytanie1);
        s.setInt( "parameterIndex: 1, r.getInt( "columnLabel: Miejsce_wylotu"));
        temp = SQLUtilities.ExecuteQuery(s);
        if(temp.next()) dane[i-1] = temp.getString( "columnIndex: 1");
        // dane[i-1] = String.valueOf(r.getInt("Miejsce_wylotu"));
        i++;
        zapytanie1 = "SELECT Nazwa FROM Lotnisko WHERE LotniskoID = ?";
        s = SQLUtilities.connection.prepareStatement(zapytanie1);
        s.setInt( "parameterIndex: 1, r.getInt( "columnLabel: Miejsce_przylotu"));
        temp = SQLUtilities.ExecuteQuery(s);
        if(temp.next()) dane[i-1] = temp.getString( "columnIndex: 1");
        //dane[i-1] = String.valueOf(r.getInt("Miejsce_przylotu"));
        i++;
    }
    SQLUtilities.connection.close();
    return dane;
}
catch(SQLException ex){
    JOptionPane.showMessageDialog( "parentComponent: null, ex);
}
return dane1;
}

```

Rysunek 40. Metoda podajDane() zwracająca listę wszystkich połączeń dostępnych w bazie danych.

5.4.3. Implementacja mechanizmów bezpieczeństwa

W aplikacji zostało zaimplementowane logowanie do systemu z weryfikacją loginu oraz hasła. Za pomocą zapytania SQL program szuka czy w bazie danych w tabeli „Konto” jest rekord o podanych przez użytkownika parametrach. Jeżeli tak, zostaje przydzielony dostęp do systemu. W innym przypadku zostaje wyświetlony odpowiedni komunikat o niepoprawnych danych. W bazie danych ustawiono uprawnienia dla konta administratora oraz klienta za pomocą polecenia GRANT w języku SQL. Aplikacja posiada metodę sprawdzającą czy aktywny użytkownik jest administratorem i ma uprawnienia do wykonania danej czynności.

```

public static boolean zaloguj(String login, String haslo){

    String URL = "jdbc:sqlserver://localhost;databaseName=FlightStore;user=Klient;password=klient;";
    try {
        SQLUtilities.Connect(URL);
        String zapytanie = "SELECT login, password FROM Konto WHERE login = ? AND password = ?";
        PreparedStatement s = SQLUtilities.connection.prepareStatement(zapytanie);
        s.setString( parameterIndex: 1, login);
        s.setString( parameterIndex: 2, haslo);
        ResultSet r = SQLUtilities.ExecuteQuery(s);
        if(r.next()){
            String odczyt1 = r.getString( columnIndex: 1);
            String odczyt2 = r.getString( columnIndex: 2);
            if(odczyt1.equals(login) && odczyt2.equals(haslo)) return true;
        }
        SQLUtilities.connection.close();
        return false;
    }
    catch (SQLException ex){
        JOptionPane.showMessageDialog( parentComponent: null, ex);
    }
    return false;
}

```

Rysunek 41. Metoda `zaloguj(String login, String haslo)` logująca do systemu.

```

public static boolean sprawdzCzyAdmin(String login, String haslo){

    String URL = "jdbc:sqlserver://localhost;databaseName=FlightStore;user=Klient;password=klient;";
    try {
        SQLUtilities.Connect(URL);
        String zapytanie = "SELECT admin FROM Konto WHERE login = ? AND password = ?";
        PreparedStatement s = SQLUtilities.connection.prepareStatement(zapytanie);
        s.setString( parameterIndex: 1, login);
        s.setString( parameterIndex: 2, haslo);
        ResultSet r = SQLUtilities.ExecuteQuery(s);
        if(r.next()) {
            String odczyt = r.getString( columnIndex: 1);
            if (odczyt.equals("1")) return true;
        }
        SQLUtilities.connection.close();
    }
    catch(SQLException ex){
        JOptionPane.showMessageDialog( parentComponent: null, ex);
    }
    return false;
}

```

Rysunek 42. Metoda `sprawdzCzyAdmin(String login, String haslo)` sprawdzająca czy użytkownik jest administratorem.

6. Podsumowanie i wnioski

W wyniku prac powstała aplikacja łącząca się z bazą danych i realizująca podstawowe wymagania sprzedaży biletów lotniczych. Pozwala ona wykonywać operacje opisane w projekcie. Dzięki zastosowaniu relacyjnej bazy danych możemy przechowywać wszelkie informacje dotyczące użytkowników, pasażerów, połączeń czy biletów, a zastosowana aplikacja pozwala łatwo edytować jej zawartość dzięki czemu ułatwia administrowanie systemu.

Literatura

- [1] Strona internetowa: <https://www.sqlservertutorial.net/>.
- [2] Garcia-Molina H., Ullman J.D., Widom J., *Systemy baz danych. Kompletny podręcznik*. Wydanie II, Prentice Hall, New Jersey, 2011.
- [3] Strona internetowa: <http://roman.ptak.staff.iiar.pwr.wroc.pl/>.
- [4] Strona internetowa: <http://it.dth.pl/grant-oraz-revoke-kurs-jezyka-sql-mysql-cz-1/>.

Spis rysunków

Rysunek 1. Schemat logiczny systemu	4
Rysunek 2. Diagram przypadków użycia.....	5
Rysunek 3. Uproszczony model conceptualny związków encji.....	12
Rysunek 4. Model logiczny	12
Rysunek 5. Model fizyczny bazy danych wraz z ograniczeniami.....	13
Rysunek 6. Rysunek przedstawiający architekturę trzywarstwową aplikacji.	14
Rysunek 7. Menu główne – Administrator.....	14
Rysunek 8. Menu główne – Klient.....	15
Rysunek 9. Dodawanie połączenia – Administrator	15
Rysunek 10. Ekran powitalny	16
Rysunek 11. Lista kupionych biletów	16
Rysunek 12. Lista połączeń.....	17
Rysunek 13. Logowanie	17
Rysunek 14. Modyfikowanie połączenia - Administrator.....	18
Rysunek 15. Usuwanie konta – Administrator.....	18
Rysunek 16. Usuwanie połączenia – Administrator.....	19
Rysunek 17. Wyszukiwanie połączenia	19
Rysunek 18. Zakładanie konta	20
Rysunek 19. Poprawne wstawienie rekordów do tabeli "Adres"	23

Rysunek 20. Test wstawienia wartości NULL do kolumny NOT NULL zakończona niepowodzeniem.	23
Rysunek 21. Poprawne usuwanie rekordów z tabeli "Adres"	23
Rysunek 22. Test usunięcia rekordu z tabeli "Adres" zakończona niepowodzeniem.	23
Rysunek 23. Widok "Bilety" po wstawieniu 100 jednakowych biletów.	24
Rysunek 24. Widok "Dane Pasażera" wraz z przykładowymi pasażerami.	24
Rysunek 25. Test wydajnościowy dla wstawiania 100 nowych biletów.	24
Rysunek 26. Test wydajnościowy dla wstawiania 1000 nowych lotów.	25
Rysunek 27. Nieudane logowanie.	26
Rysunek 28. Udana logowanie.	26
Rysunek 29. Lista połączeń.	27
Rysunek 30. Kupowanie biletu.	27
Rysunek 31. Modyfikacja danych konta.	27
Rysunek 32. Wyszukiwanie połączenia.	28
Rysunek 33. Lista wyszukanych połączeń.	28
Rysunek 34. Lista zakupionych biletów.	28
Rysunek 35. Klasa SQLUtilities.	29
Rysunek 36. Metoda dodajBilet() dodająca nowy bilet do bazy danych.	30
Rysunek 37. MetodausunKonto(int ID) usuwająca konto z bazy o podanym numerze ID.	31
Rysunek 38. Metoda uzupełnij(int ID) wypełniająca formularz do edycji danychi połączenia o wybranym numerze ID.	31
Rysunek 39. Metoda modyfikujPolaczenie(Date data, int[] dane) wprowadza nowe dane do wybranego istniejącego lotu w bazie.	32
Rysunek 40. Metoda podajDane() zwracająca listę wszystkich połączeń dostępnych w bazie danych.	33
Rysunek 41. Metoda zaloguj(String login, String haslo) logująca do systemu.	34
Rysunek 42. Metoda sprawdzCzyAdmin(String login, String haslo) sprawdzająca czy użytkownik jest administratorem.	34