

# Dokumentacja projektu wykonanego w ramach zajęć Bazy danych I

Autor: Hadam Paweł

## 1. PROJEKT KONCEPCJI, ZAŁOŻENIA:

- Zdefiniowanie tematu projektu:

Założeniem mojego projektu było stworzenie aplikacji desktopowej pozwalającą na obsługę i zarządzanie bazą danych firmy przewożącej ludzi przy użyciu dowolnego pojazdu np. samochód, bus, autobus.

Aplikacja może być obsługiwana przez 3 typy użytkowników:

- Administrator,
- Użytkownik,
- Gość

- Analiza wymagań użytkownika:

W aplikacji znajdują się 3 poziomy dostępu:

- Administrator:
  - Login: admin, hasło: admin
  - Administrator ma dostęp do pełnej funkcjonalności, takich jak przegląd wszystkich widoków oraz dodawanie/usuwanie rekordów
- Użytkownik:
  - Użytkownik – tworzony przy rejestracji ma dostęp do wglądu na obecne kursy, zakup biletu na dowolnie wybrany kurs oraz ma ograniczony wgląd na pasażerów przy wpisanym kursie
- Gość:
  - Gość ma jedynie ograniczony wgląd na obecne kursy

- Zaprojektowanie funkcji:

- Dodanie nowych kursów pomiędzy dwoma miastami wraz z odległością między nimi, godzinami odjazdu oraz powrotu z pierwszego miasta, z danym kierowcą oraz pojazdem,
- Dodanie nowych kierowców,
- Dodanie nowych pojazdów,
- Rejestracji nowych użytkowników,
- Przegląd dodanych kursów, kierowców, pojazdów, użytkowników oraz pasażerów z biletami i bagażem na dany kurs,
- Zakup biletu przez użytkowników na dany kurs lub dodanie pasażera przez administratora,

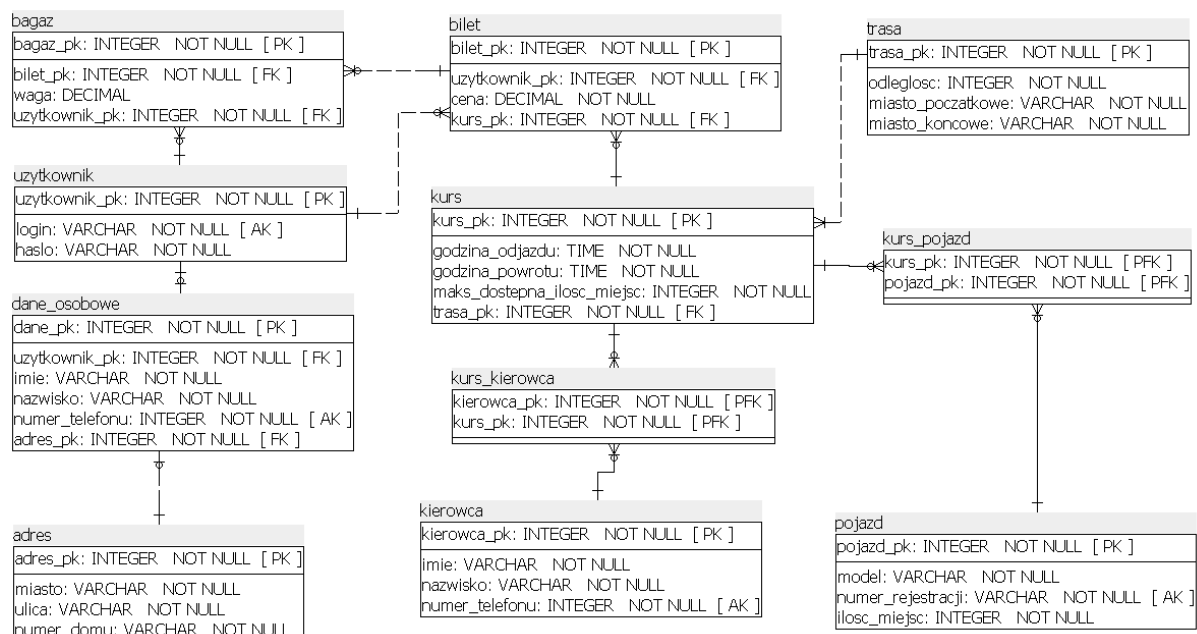
- Usuwanie rekordów z bazy przez administratora,
- Dodanie kolejnych kierowców oraz pojazdów na dany kurs,

## 2. PROJEKT DIAGRAMÓW (KONCEPTUALNY):

- Zdefiniowanie encji (obiektów) oraz ich atrybutów wraz z zdefiniowaniem kluczy:
  - bagaz:
    - bagaz\_pk – klucz główny encji
    - bilet\_pk – klucz obcy encji
    - waga – waga bagażu danego użytkownika na dany kurs
    - użytkownik\_pk – klucz obcy encji
  - użytkownik:
    - użytkownik\_pk – klucz główny encji
    - login – nazwa użytkownika aplikacji – wartość unikalna
    - hasło – hasło użytkownika aplikacji
  - dane\_osobowe:
    - dane\_pk – klucz główny encji
    - użytkownik\_pk – klucz obcy encji
    - imie – imię użytkownika aplikacji
    - nazwisko – nazwisko użytkownika aplikacji
    - numer\_telefonu – numer telefonu użytkownika aplikacji – wartość unikalna
    - adres\_pk – klucz obcy encji
  - adres:
    - adres\_pk – klucz główny encji
    - miasto – miejsce zamieszkania użytkownika
    - ulica – ulica na której mieszka użytkownik
    - numer\_domu – numer domu w którym mieszka użytkownik
  - bilet:
    - bilet\_pk – klucz główny encji
    - użytkownik\_pk – klucz obcy encji
    - cena – cena biletu liczona jako odległość \* 2 zł
    - kurs\_pk – klucz obcy encji
  - kurs:
    - kurs\_pk – klucz główny encji

- godzina\_odjazdu – godzina odjazdu z miasta początkowego
- godzina\_powrotu – godzina powrotu do miasta początkowego
- maks\_dostepna\_ilosc\_miejsc – ilość miejsc na dany kurs liczona jako suma ilości miejsc przydzielonych pojazdów
- trasa\_pk – klucz obcy encji
- kurs\_kierowca – tablica asocjacyjna:
  - kierowca\_pk – klucz główny i obcy encji
  - kurs\_pk – klucz główny i obcy encji
- kierowca:
  - kierowca\_pk – klucz główny encji
  - imie – imię kierowcy
  - nazwisko – nazwisko kierowcy
  - numer\_telefonu – numer telefonu kierowcy – wartość unikalna
- trasa:
  - trasa\_pk – klucz główny encji
  - odleglosc – odległość między miastami w encji
  - miasto\_poczatkowe – miasto początkowe trasy
  - miasto\_koncowe – miasto końcowe trasy
- kurs\_pojazd – tablica asocjacyjna:
  - kurs\_pk – klucz główny i obcy encji
  - pojazd\_pk – klucz główny i obcy encji
- pojazd:
  - pojazd\_pk – klucz główny encji
  - model – model pojazdu
  - numer\_rejestracji – tablica rejestracyjna pojazdu – wartość unikalna
  - ilosc\_miejsc – ilość dostępnych miejsc w danym pojeździe
- Zaprojektowanie relacji pomiędzy encjami:
  - Relacja bagaz-bilet – n:1, na jeden bilet można wziąć dowolną ilość bagażu

- Relacja bagaż-uzytkownik – n:1, dany użytkownik może mieć dowolną ilość bagażu
- Relacja uzytkownik-dane\_osobowe – 1:1 – dany użytkownik posiada unikalne dla siebie dane osobowe
- Relacja uzytkownik-bilet – 1:n – użytkownik może zakupić bilety na dowolną ilość kursów
- Relacja dane\_osobowe-adres – 1:1, użytkownik posiada jedno miejsce zamieszkania, oddzielone od siebie encje w celu lepszej normalizacji
- Relacja bilet-kurs: n:1 – dany kurs może posiadać wiele biletów
- Relacja kurs-trasa: n:1 – wiele kursów może posiadać daną trasę
- Relacja kurs-kierowca: n:m połączona tablicą asocjacyjną kurs\_kierowca, dany kierowca może być przydzielony na różne kursy, dany kurs może mieć również wielu kierowców
- Relacja kurs-pojazd: n:m połączona tablicą asocjacyjną kurs\_pojazd, dany pojazd może być używany w wielu kursach, dany kurs może mieć również wiele pojazdów



Rysunek 1 Diagram ER

### 3. PROJEKT LOGICZNY:

- Projektowanie tabel, kluczy, indeksów:  
Projekt schematu, tabel, kluczy, sekwencji a także widoków znajduje się w pliku init.sql. Plik ten dodaje także konto administratora aplikacji.
- Widoki:  
Widoki przedstawione są następującymi kwerendami:

```
CREATE OR REPLACE VIEW transport."trasy_view" AS
SELECT
    kurs.kurs_pk,
    trasa.trasa_pk,
    kurs.godzina_odjazdu,
    kurs.maks_dostepna_ilosc_miejsc AS wolne_miejsca,
    trasa.odleglosc AS km,
    trasa.miasto_poczkowe,
    trasa.miasto_koncowe,
    (trasa.odleglosc * 2) AS cena_biletu
FROM transport.kurs kurs
JOIN transport.trasa trasa ON kurs.trasa_pk = trasa.trasa_pk
ORDER BY miasto_poczkowe, godzina_odjazdu;

CREATE VIEW transport."uzytkownicy_view" AS
SELECT
    uzytkownik.uzytkownik_pk,
    dane.dane_pk,
    adres.adres_pk,
    uzytkownik.login,
    dane.imie,
    dane.nazwisko,
    dane.numer_telefonu,
    adres.miasto,
    adres.ulica,
    adres.numer_domu
FROM ((transport.uzytkownik uzytkownik
JOIN transport.dane_osobowe dane ON ((uzytkownik.uzytkownik_pk = dane.uzytkownik_pk)))
JOIN transport.adres ON ((dane.adres_pk = adres.adres_pk)))
ORDER BY uzytkownik.uzytkownik_pk;

CREATE OR REPLACE VIEW transport."pasazerowie_view" AS
SELECT
    uzytkownik.uzytkownik_pk,
    bilet.bilet_pk,
    bagaz.bagaz_pk,
    kurs.kurs_pk,
    dane_osobowe.imie,
    dane_osobowe.nazwisko,
    dane_osobowe.numer_telefonu,
    bagaz.waga
FROM transport.uzytkownik
JOIN transport.dane_osobowe ON uzytkownik.uzytkownik_pk = dane_osobowe.uzytkownik_pk
JOIN transport.bilet ON uzytkownik.uzytkownik_pk = bilet.uzytkownik_pk
LEFT JOIN transport.bagaz ON uzytkownik.uzytkownik_pk = bagaz.uzytkownik_pk
JOIN transport.kurs ON bilet.kurs_pk = kurs.kurs_pk;
```

- Słowniki danych:  
W projekcie znajdują się następujące słowniki:

bagaz		
nazwa	typ	dziedzina
bagaz_pk	INTEGER	liczby całkowite
bilet_pk	INTEGER	liczby całkowite
waga	DECIMAL	liczby zmiennoprzecinkowe podwójnej precyzji
uzytkownik_pk	INTEGER	liczby całkowite

uzytkownik		
nazwa	typ	dziedzina
uzytkownik_pk	INTEGER	liczby całkowite
login	VARCHAR	tekst
haslo	VARCHAR	liczby całkowite

dane_osobowe		
nazwa	typ	dziedzina
dane_pk	INTEGER	liczby całkowite
uzytkownik_pk	INTEGER	liczby całkowite
imie	VARCHAR	tekst
nazwisko	VARCHAR	tekst
numer_telefonu	INTEGER	liczby całkowite
adres_pk	INTEGER	liczby całkowite

adres		
nazwa	typ	dziedzina
adres_pk	INTEGER	liczby całkowite
miasto	VARCHAR	tekst
ulica	VARCHAR	tekst
numer_domu	VARCHAR	tekst

kierowca		
nazwa	typ	dziedzina
kierowca_pk	INTEGER	liczby całkowite
imie	VARCHAR	tekst
nazwisko	VARCHAR	tekst
numer_telefonu	INTEGER	liczby całkowite

kurs_kierowca		
nazwa	typ	dziedzina
kierowca_pk	INTEGER	liczby całkowite
kurs_pk	INTEGER	liczby całkowite

kurs		
nazwa	typ	dziedzina
kurs_pk	INTEGER	liczby całkowite
godzina_odjazdu	TIME	godziny:minuty:sekundy zegarowe
godzina_powrotu	TIME	godziny:minuty:sekundy zegarowe
maks_dostepna_ilosc_miejsc	INTEGER	liczby całkowite
trasa_pk	INTEGER	liczby całkowite

bilet		
nazwa	typ	dziedzina
bilet_pk	INTEGER	liczby całkowite
uzytkownik_pk	INTEGER	liczby całkowite
cena	DECIMAL	liczby zmiennoprzecinkowe podwójnej precyzji
kurs_pk	INTEGER	liczby całkowite

trasa		
nazwa	typ	dziedzina
trasa_pk	INTEGER	liczby całkowite
odleglosc	INTEGER	liczby całkowite
miasto_pocatkowe	VARCHAR	tekst
miasto_koncowe	VARCHAR	tekst

kurs_pojazd		
nazwa	typ	dziedzina
kurs_pk	INTEGER	liczby całkowite
pojazd_pk	INTEGER	liczby całkowite

pojazd		
nazwa	typ	dziedzina
pojazd_pk	INTEGER	liczby całkowite
model	VARCHAR	tekst
numer_rejestracji	VARCHAR	tekst
ilosc_miejsc	INTEGER	liczby całkowite

- Zaprojektowanie operacji na danych:

Projekt używał poniższych kwerend (znajdujących się również w pliku kwerendy.sql) używanych w obiektach typu Data Access Object:

```
INSERT INTO transport.adres (miasto, ulica, numer_domu) VALUES (?, ?, ?);
INSERT INTO transport.kurs (godzina_odjazdu, godzina_powrotu, maks_dostepna_ilosc_miejsc, trasa_pk) VALUES (?, ?, ?, ?);
INSERT INTO transport.kurs_kierowca (kierowca_pk, kurs_pk) VALUES (?, ?);
INSERT INTO transport.kurs_pojazd (kurs_pk, pojazd_pk) VALUES (?,?);
INSERT INTO transport.kierowca (imie, nazwisko, numer_telefonu) VALUES (?, ?, ?);
INSERT INTO transport.bagaz (bilet_pk, waga, uzytkownik_pk) VALUES (?, ?, ?);
INSERT INTO transport.dane_osobowe (uzytkownik_pk, imie, nazwisko, numer_telefonu, adres_pk) VALUES (?, ?, ?, ?, ?);
INSERT INTO transport.trasa (odleglosc, miasto_poczatkowe, miasto_koncowe) VALUES (?, ?, ?);
INSERT INTO transport.bilet (uzytkownik_pk, cena, kurs_pk) VALUES (?, ?, ?);
INSERT INTO transport.uzytkownik (login, haslo) VALUES (?, ?);
INSERT INTO transport.pojazd (model, numer_rejestracji, ilosc_miejsc) VALUES (?, ?, ?);
```

*Rysunek 2 Kwerendy dodające*

```
SELECT adres.adres_pk FROM transport.adres WHERE miasto = ? AND ulica = ? AND numer_domu = ?;
SELECT * FROM transport.trasy_view;
SELECT kurs.kurs_pk FROM transport.kurs WHERE kurs.godzina_odjazdu = ? AND kurs.godzina_powrotu = ? AND maks_dostepna_ilosc_miejsc = ? AND trasa_pk = ?;
SELECT * FROM transport.kurs WHERE kurs_pk = ?;
SELECT kierowca.kierowca_pk FROM transport.kierowca WHERE numer_telefonu = ?;
SELECT * FROM transport.kierowca LEFT JOIN transport.kurs_kierowca ON kierowca.kierowca_pk = kurs_kierowca.kierowca_pk;
SELECT trasa.trasa_pk FROM transport.trasa WHERE trasa.odleglosc = ? AND trasa.miasto_poczatkowe = ? AND trasa.miasto_koncowe = ?;
SELECT bilet.bilet_pk FROM transport.bilet WHERE uzytkownik_pk = ? AND kurs_pk = ?;
SELECT * FROM transport.pasazerowie_view WHERE kurs_pk = ?;
SELECT uzytkownik.uzytkownik_pk FROM transport.uzytkownik WHERE login = ?;
SELECT * FROM transport.uzytkownicy_view;
SELECT uzytkownik.uzytkownik_pk FROM transport.uzytkownik WHERE login = ? AND haslo = ?;
SELECT pojazd.ilosc_miejsc FROM transport.pojazd WHERE numer_rejestracji = ?;
SELECT pojazd.pojazd_pk FROM transport.pojazd WHERE numer_rejestracji = ?;
SELECT * FROM transport.pojazd LEFT JOIN transport.kurs_pojazd ON pojazd.pojazd_pk = kurs_pojazd.pojazd_pk;
```

*Rysunek 3 Kwerendy wyświetlające*

```
DELETE FROM transport.adres WHERE adres_pk = ?;
DELETE FROM transport.kurs WHERE kurs_pk = ?;
DELETE FROM transport.kurs_kierowca WHERE kierowca_pk = ?;
DELETE FROM transport.kurs_kierowca WHERE kurs_pk = ?;
DELETE FROM transport.kurs_pojazd WHERE pojazd_pk = ?;
DELETE FROM transport.kurs_pojazd WHERE kurs_pk = ?;
DELETE FROM transport.kierowca WHERE kierowca_pk = ?;
DELETE FROM transport.bagaz WHERE uzytkownik_pk = ? AND bilet_pk = ?;
DELETE FROM transport.dane_osobowe WHERE uzytkownik_pk = ? AND adres_pk = ?;
DELETE FROM transport.trasa WHERE trasa_pk = ?;
DELETE FROM transport.bilet WHERE uzytkownik_pk = ? AND kurs_pk = ?;
DELETE FROM transport.uzytkownik WHERE uzytkownik_pk = ?;
DELETE FROM transport.pojazd WHERE pojazd_pk = ?;
```

*Rysunek 4 Kwerendy usuwające*

Za pomocą logiki aplikacji powyższe kwerendy były używane do dodawania, aktualizacji, wyświetlania oraz usuwania niepotrzebnych nam już rekordów.



- Wyzwalacze

```
CREATE OR REPLACE FUNCTION kup_bilet()
  RETURNS TRIGGER AS $$
DECLARE
  w INTEGER;
BEGIN
  SELECT maks_dostepna_ilosc_miejsc
  INTO w
  FROM transport.kurs
  WHERE kurs_pk = NEW.kurs_pk;
  IF (w = 0)
  THEN RAISE 'Nie mozna kupic biletu! Brak miejsc!';
  ELSE
    UPDATE transport.kurs
    SET maks_dostepna_ilosc_miejsc = maks_dostepna_ilosc_miejsc - 1
    WHERE kurs_pk = NEW.kurs_pk;
    RETURN NEW;
  END IF;
END;
$$
LANGUAGE 'plpgsql';

CREATE TRIGGER kup_bilet_trigger
  BEFORE INSERT OR UPDATE
  ON transport.bilet
  FOR EACH ROW EXECUTE PROCEDURE kup_bilet();
```

Rysunek 5 Wyzwalacz aktualizujący ilość miejsc w kursie przy zakupie biletu i zapobiegający zakupie biletu jeżeli brak miejsc.

```
CREATE OR REPLACE FUNCTION anuluj_bilet()
  RETURNS TRIGGER AS $$
BEGIN
  UPDATE transport.kurs
  SET maks_dostepna_ilosc_miejsc = maks_dostepna_ilosc_miejsc + 1
  WHERE kurs_pk = OLD.kurs_pk;
  RETURN NULL;
END;
$$
LANGUAGE 'plpgsql';

CREATE TRIGGER anuluj_bilet_trigger
  AFTER DELETE
  ON transport.bilet
  FOR EACH ROW EXECUTE PROCEDURE anuluj_bilet();
```

Rysunek 6 Wyzwalacz aktualizujący ilość miejsc w kursie w przypadku usunięcia pasażera.

```

CREATE OR REPLACE FUNCTION dodaj_kierowca_bus()
  RETURNS TRIGGER AS $$
BEGIN
  UPDATE transport.kurs
  SET maks_dostepna_ilosc_miejsc = maks_dostepna_ilosc_miejsc + (SELECT ilosc_miejsc
                                                                FROM transport.pojazd
                                                                WHERE pojazd.pojazd_pk = NEW.pojazd_pk)
  WHERE kurs_pk = NEW.kurs_pk;
  RETURN NEW;
END;
$$
LANGUAGE 'plpgsql';

CREATE TRIGGER dodaj_kierowca_bus_trigger
  AFTER INSERT
  ON transport.kurs_pojazd
  FOR EACH ROW EXECUTE PROCEDURE dodaj_kierowca_bus();

```

Rysunek 7 Wyzwalacz aktualizujący liczbę miejsc w kursie w przypadku dodania pojazdu do kursu.

```

CREATE OR REPLACE FUNCTION usun_kurs_gdy_pasazer()
  RETURNS TRIGGER AS $$
DECLARE
  w INTEGER;
BEGIN
  SELECT COUNT(*)
  INTO w
  FROM transport.bilet
  WHERE kurs_pk = OLD.kurs_pk;
  IF (w > 0)
  THEN RAISE 'Nie mozna usunac kursu! Istnieje pasazer zapisany na ten kurs!';
  END IF;
  RETURN OLD;
END;
$$
LANGUAGE 'plpgsql';

CREATE TRIGGER usun_kurs_gdy_pasazer_trigger
  BEFORE DELETE
  ON transport.kurs
  FOR EACH ROW EXECUTE PROCEDURE usun_kurs_gdy_pasazer();

```

Rysunek 8 Wyzwalacz nie pozwalający usunąć kursu jeżeli są zapisani pasażerowie.

```

CREATE OR REPLACE FUNCTION usun_kurs_trasa()
| RETURNS TRIGGER AS $$
BEGIN
| DELETE FROM transport.trasa
| WHERE trasa.trasa_pk = OLD.trasa_pk;
| RETURN OLD;
END;
$$
LANGUAGE 'plpgsql';

CREATE TRIGGER usun_kurs_trasa_trigger
| AFTER DELETE
| ON transport.kurs
| FOR EACH ROW EXECUTE PROCEDURE usun_kurs_trasa();

```

Rysunek 9 Wyzwalacz usuwający trasę w przypadku usunięcia kursu.

```

CREATE OR REPLACE FUNCTION usun_admina()
| RETURNS TRIGGER AS $$
BEGIN
| IF OLD.login = 'admin'
| THEN
| RAISE 'Bład! Nie można usunąć administratora';
| END IF;
| RETURN OLD;
END;
$$
LANGUAGE 'plpgsql';

CREATE TRIGGER usun_adminaa_trigger
| BEFORE DELETE
| ON transport.uzytownik
| FOR EACH ROW EXECUTE PROCEDURE usun_admina();

```

Rysunek 10 Wyzwalacz nie pozwalający usunąć konta administratora.

```

CREATE OR REPLACE FUNCTION usun_kierowce()
  RETURNS TRIGGER AS $$
DECLARE
  w INTEGER;
BEGIN
  SELECT COUNT(*)
  INTO w
  FROM transport.kurs_kierowca
  WHERE kierowca_pk = OLD.kierowca_pk;
  IF (w > 0)
  THEN RAISE 'Nie mozna usunac kierowcy! Kierowca jest zapisany na kurs!';
  END IF;
  RETURN OLD;
END;
$$
LANGUAGE 'plpgsql';

CREATE TRIGGER usun_kierowce_trigger
  BEFORE DELETE
  ON transport.kierowca
  FOR EACH ROW EXECUTE PROCEDURE usun_kierowce();

```

Rysunek 11 Wyzwalacz nie pozwalający usunąć kierowcę jeżeli jest zapisany na kurs.

```

CREATE OR REPLACE FUNCTION usun_pojazd()
  RETURNS TRIGGER AS $$
DECLARE
  w INTEGER;
BEGIN
  SELECT COUNT(*)
  INTO w
  FROM transport.kurs_pojazd
  WHERE pojazd_pk = OLD.pojazd_pk;
  IF (w > 0)
  THEN RAISE 'Nie mozna usunac pojazdu! Pojazd jest zapisany na kurs!';
  END IF;
  RETURN OLD;
END;
$$
LANGUAGE 'plpgsql';

CREATE TRIGGER usun_pojazd_trigger
  BEFORE DELETE
  ON transport.pojazd
  FOR EACH ROW EXECUTE PROCEDURE usun_pojazd();

```

Rysunek 12 Wyzwalacz nie pozwalający usunąć pojazdu jeżeli jest zapisany na kurs.

```

CREATE OR REPLACE FUNCTION usun_uzytkownika_gdy_kurs()
  RETURNS TRIGGER AS $$
DECLARE
  w INTEGER;
BEGIN
  SELECT COUNT(*)
  INTO w
  FROM transport.bilet
  WHERE bilet.uzytkownik_pk = OLD.uzytkownik_pk;
  IF (w > 0)
  THEN RAISE 'Nie mozna usunac uzytkownika! Uzytkownik jest zapisany na kurs!';
  END IF;
  RETURN OLD;
END;
$$
LANGUAGE 'plpgsql';

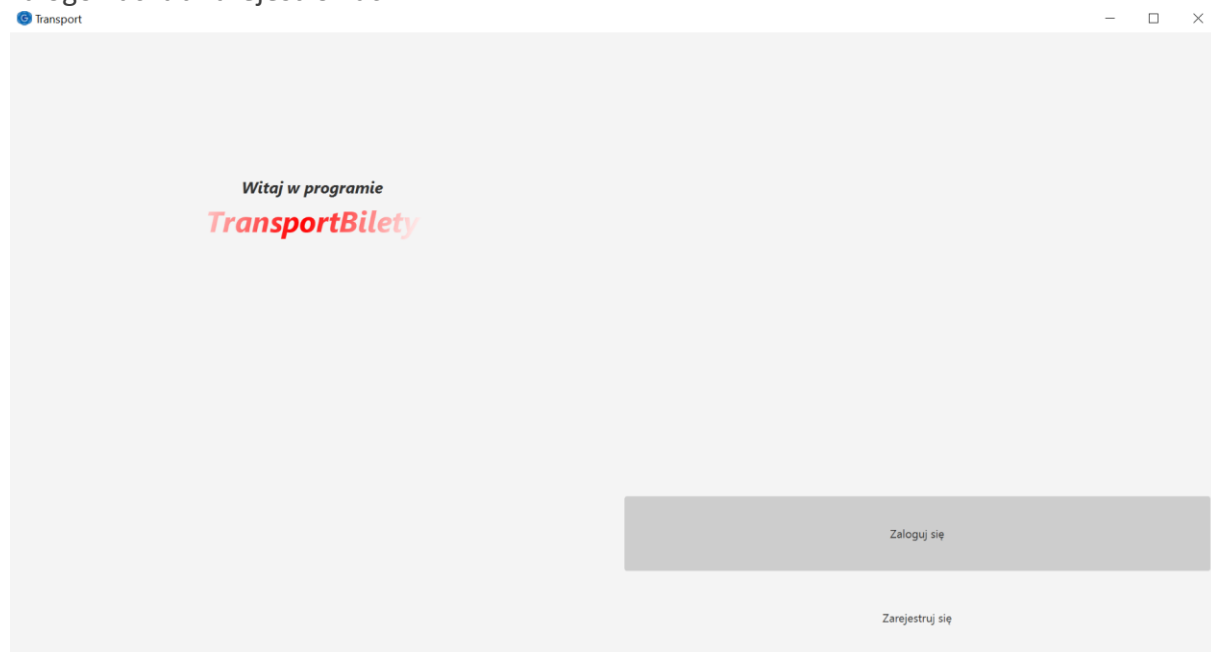
CREATE TRIGGER usun_uzytkownika_gdy_kurs_trigger
  BEFORE DELETE
  ON transport.uzytkownik
  FOR EACH ROW EXECUTE PROCEDURE usun_uzytkownika_gdy_kurs();

```

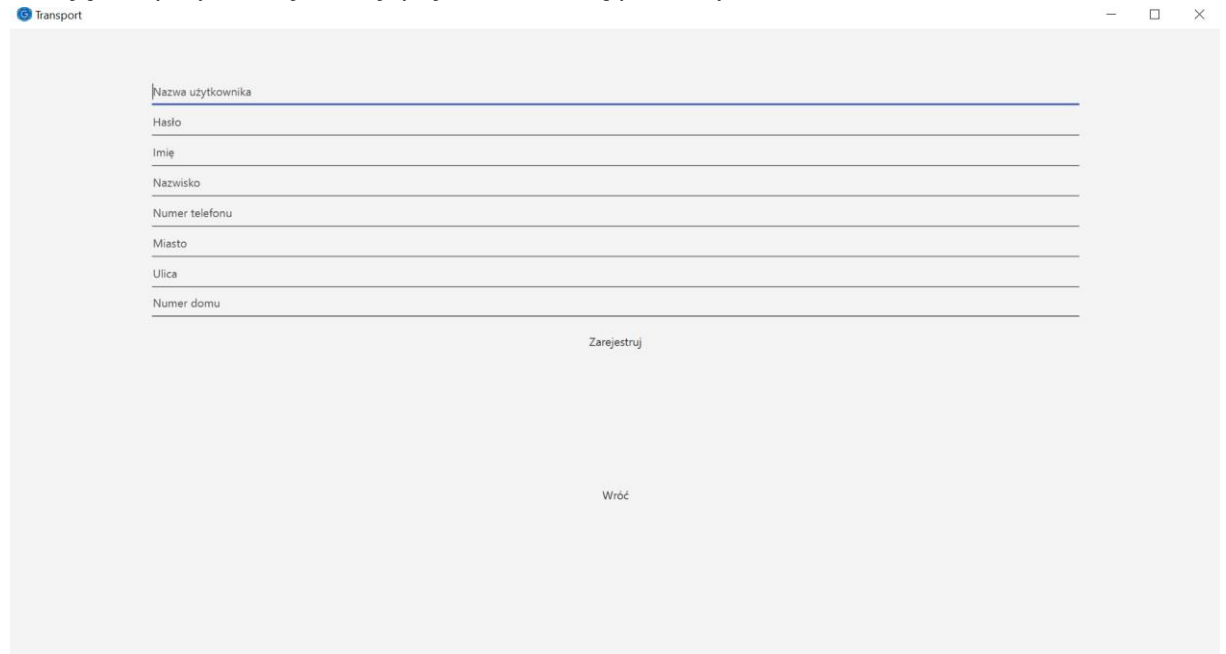
Rysunek 13 Wyzwalacz nie pozwalający usunąć użytkownika jeżeli jest zapisany na kurs.

## 4. PROJEKT FUNKCJONALNY:

W chwili uruchomienia aplikacji dostajemy okno powitalne pozwalające nam się zalogować lub zarejestrować:



Klikając na przycisk rejestracji pojawia nam się poniższy formularz:



Transport

Nazwa użytkownika

Hasło

Imię

Nazwisko

Numer telefonu

Miasto

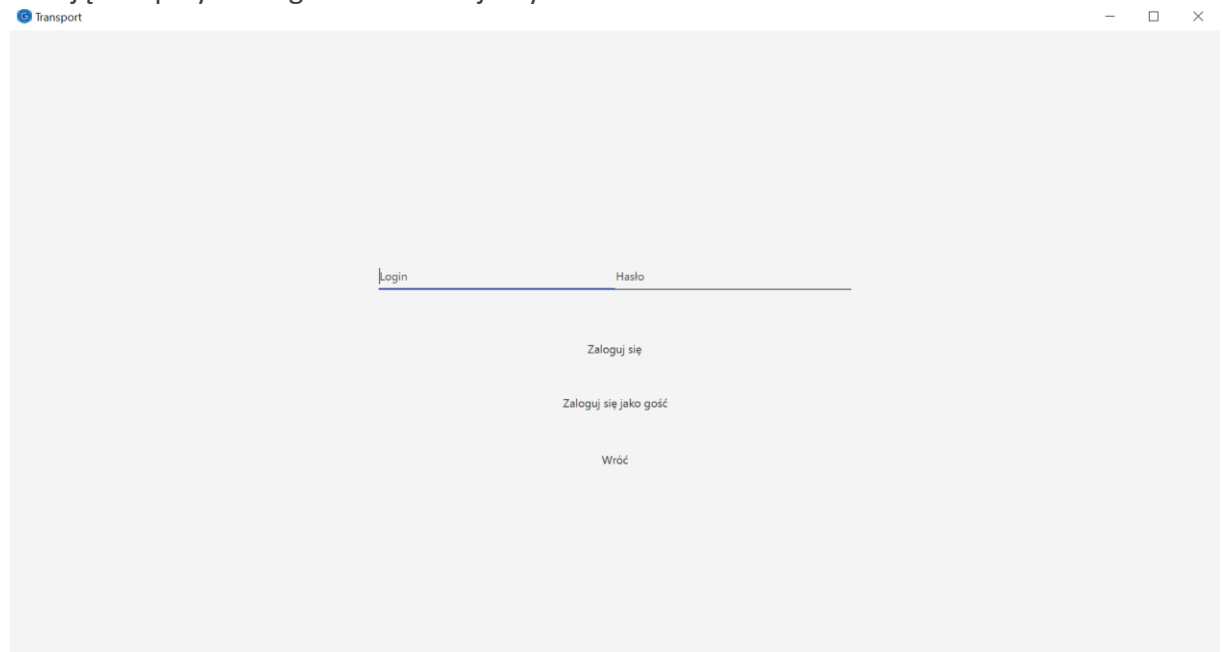
Ulica

Numer domu

Zarejestruj

Wróć

Klikając na przycisk logowania dostajemy:



Transport

Login

Hasło

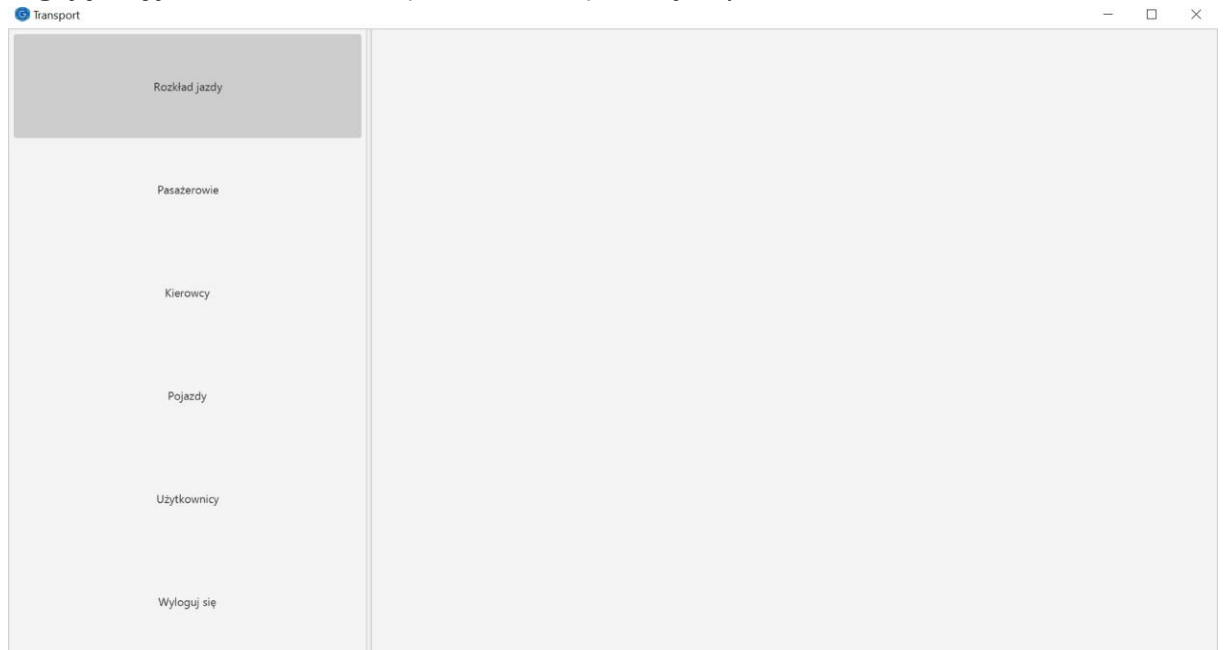
Zaloguj się

Zaloguj się jako gość

Wróć

Okno to pozwala nam się zalogować jako użytkownik z bazy danych lub jako gość.

Logując się jako administrator (admin:admin) dostajemy;

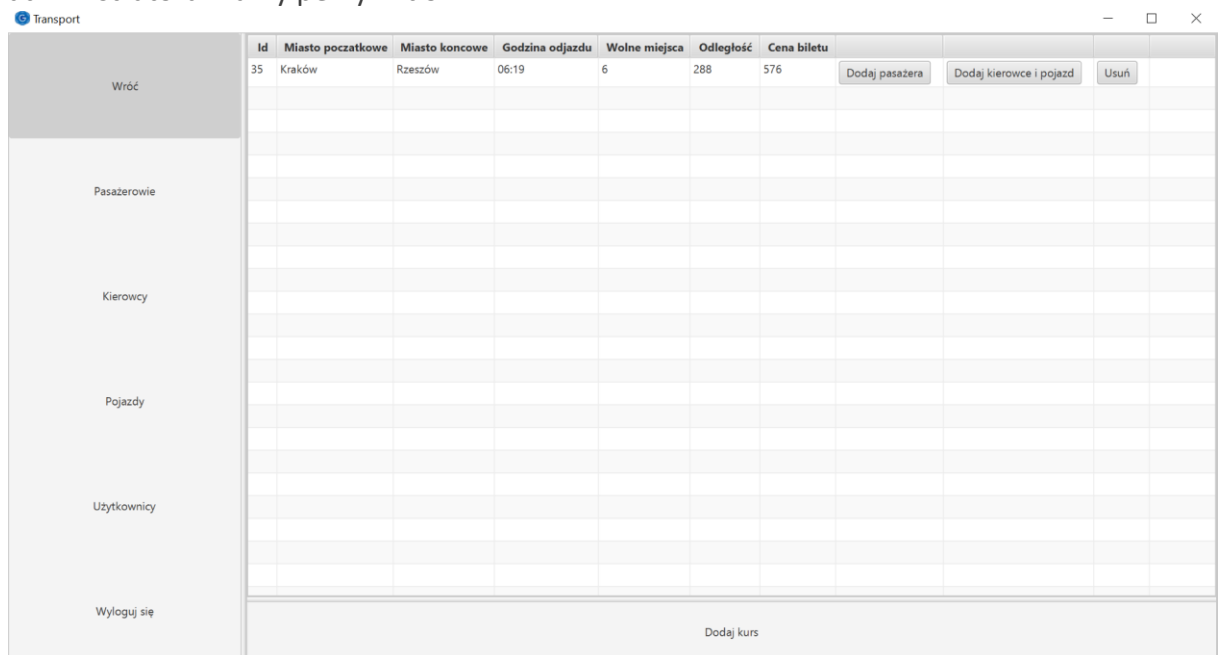


W przypadku logowania się jako standardowy użytkownik mamy ograniczone menu:



W przypadku zalogowania się jako gość:

Widoki rozkładu jazdy różnią się dla każdej grupy userów. W przypadku administratora mamy pełny widok:



Mamy możliwość dodania nowego kursu, użytkownika jako pasażera, dodania dodatkowego kierowcy wraz z pojazdem do istniejącego kursu. Grupa administratora pozwala nam też usunąć dany kurs.



Jako zwykły użytkownik:

[illegible]

Możemy jedynie kupić bilet przy użyciu przycisku dodaj pasażera (tutaj zalogowanego użytkownika).

Będąc gościem mamy możliwość jedynie zobaczyć obecne kursy:

Transport							
Wróć	Miasto początkowe	Miasto końcowe	Godzina odjazdu	Wolne miejsca	Odległość	Cena biletu	
	Kraków	Rzeszów	06:19	6	288	576	
Wyloguj się							

Widok pasażerów także się różni w zależności czy użytkownik jest administratorem czy standardowym użytkownikiem. Po wpisaniu id kursu oraz kliknięciu na przycisk szukaj dostajemy w przypadku administratora:



Widok kierowców:

[illegible]

Widok pojazdów:

[illegible]

Widok użytkowników:

[illegible]

**Formularze:**

## Formularz dodawania kursu:

Transport

Miasto początkowe

Miasto końcowe

Odległość między miastami

7:41 PM

8:41 PM

Numer telefonu kierowcy

Numer rejestracyjny pojazdu

Dodaj

Wróć

Oraz okno wybierania godziny:

7:41 PM

19:41

11121

102300132

92222214

821201918174

765

Dodaj

Wróć

Formularz dodawania pasażera jako administrator:

Transport

Id pasażera

Waga bagażu w kilogramach

Dodaj

Wróć

Formularz kupowania biletu jako standardowy użytkownik:

Transport


Waga bagażu w kilogramach

Czy chcesz kupić bilet na dany kurs?

Tak

Wróć

## Formularz dodawania do danego kursu dodatkowego kierowcę i pojazd:

 Transport


Numer telefonu kierowcy

Numer rejestracji pojazdu

Dodaj

Wróć

## Formularz dodający kierowcę:

 Transport

Imię kierowcy

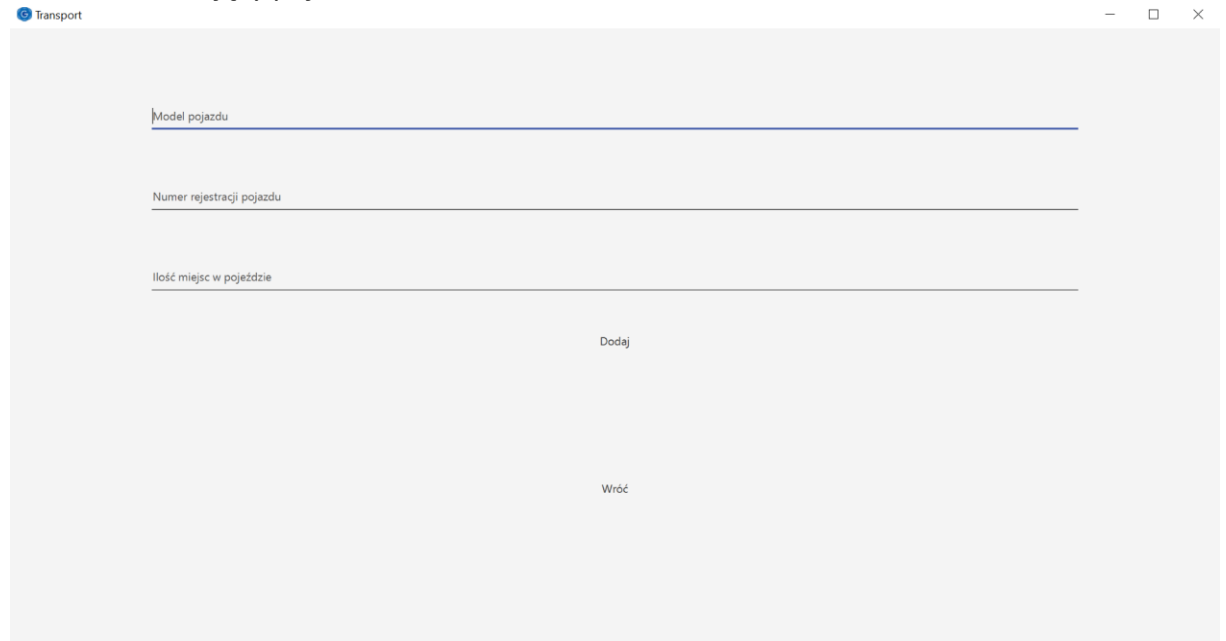
Nazwisko kierowcy

Numer telefonu kierowcy

Dodaj

Wróć

## Formularz dodający pojazd:



Model pojazdu

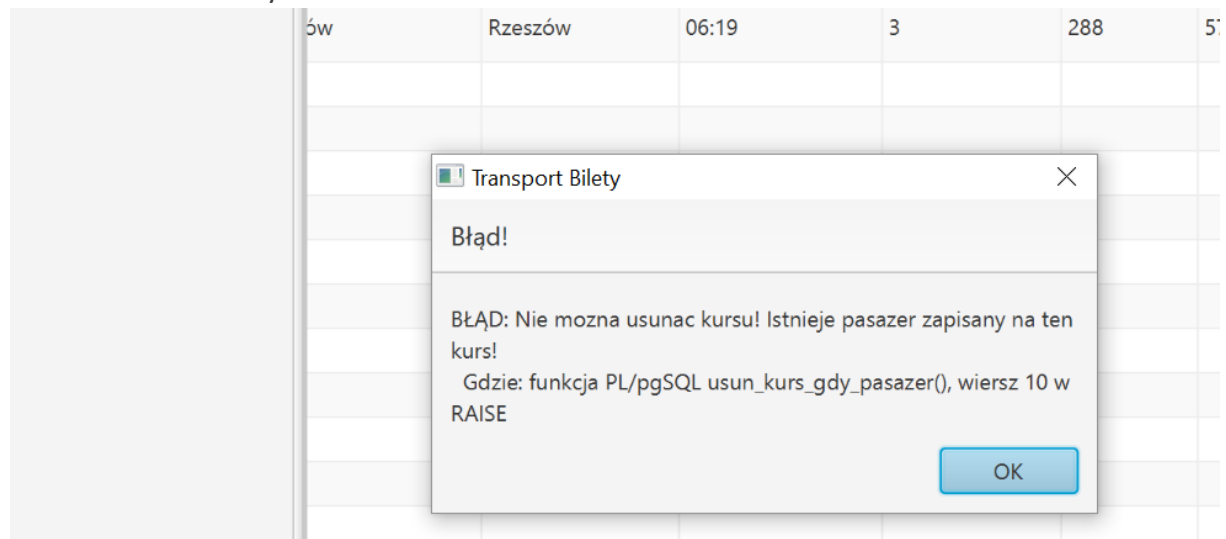
Numer rejestracji pojazdu

Ilość miejsc w pojeździe

Dodaj

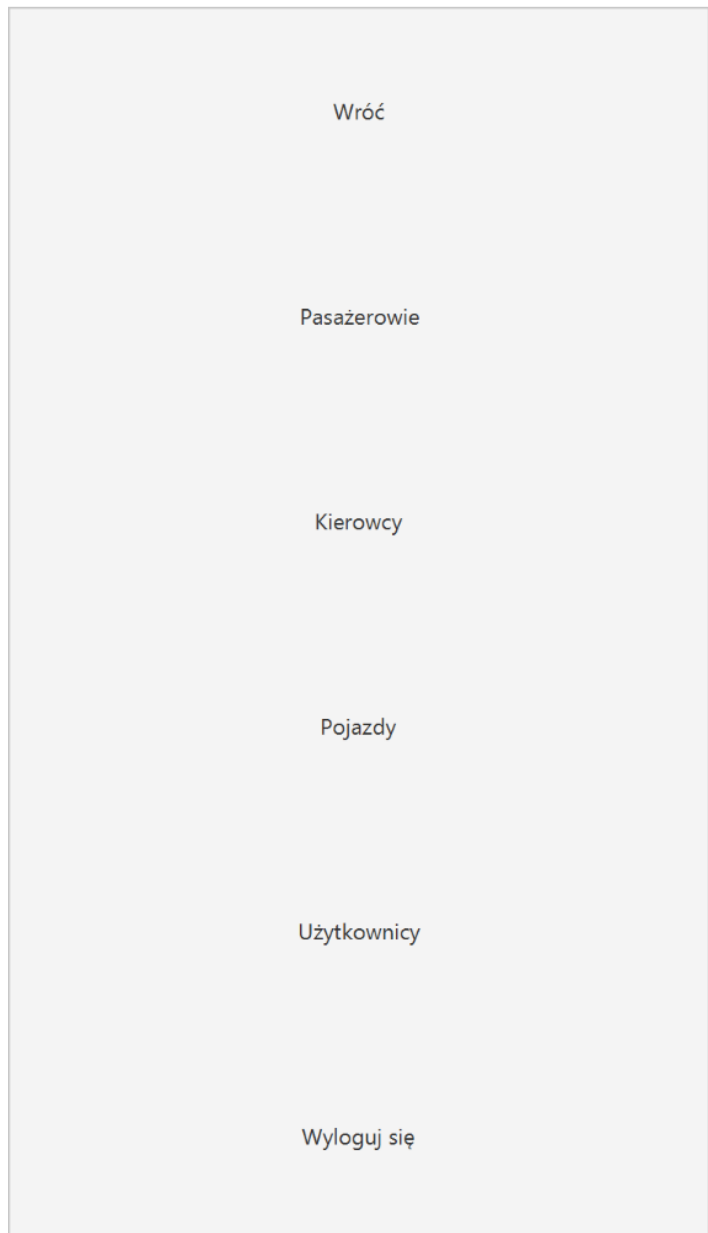
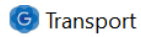
Wróć

W przypadku niedozwolonej operacji pojawia się okno informujące o błędzie wraz z komunikatem. Przykładowe okno:



Panel sterowania aplikacji znajdujące się po lewej stronie pozwala na szybkie przechodzenie między poszczególnymi widokami aplikacji. W przypadku wejścia w dany widok przycisk zmienia nazwę na wróć pozwalający powrót do głównego

ekranu:



## 5. DOKUMENTACJA:

W przypadku gdy w bazie nie ma schematu transport należy uruchomić skrypt init.sql. Pozwala on na stworzenie schematu, widoków, tabel, sekwencji i wyzwalaczy wraz z funkcjami. Skrypt ten też dodaje konto administratora. Resztę wprowadzanych danych wprowadzamy już ręcznie przy użyciu formularzy.

Konto administratora ma login: admin, hasło: admin.

Dokumentacja aplikacji nie była generowana z powodu możliwego zaciemnienia kodu tak naprawdę utrudniającego rozwijanie aplikacji.



Dokumentacja javadoc znajduje się w ./build/docs/javadoc Uważam, że struktura projektu (MVC oraz MVC dla widoku), nazwy klas, nazwy metod tłumaczą wystarczająco co autor ma na myśli i jednocześnie powołując się na książkę Czysty Kod. Podręcznik dobrego programisty Roberta C. Martina.

W projekcie użyłem bibliotek takich jak:

- ✓ JavaFX,
- ✓ Log4j w celu logowania działań,
- ✓ JFoenix dodającą komponenty wyglądu używane w aplikacji,
- ✓ JSch w celu tunelowania,
- ✓ Lombok w celu ograniczenia kodu „boilerplate”.

Link do aplikacji na GitHub: [KLIKNIJ!](#)