Uniwersytet Rzeszowski

Wydział Nauk Ścisłych i Technicznych

Kamil Szopniewski

134976

FleetManagement – System do zarządzania flota pojazdów w firmie

Dokumentacja techniczna

Projekt zaliczeniowy przedmiotu Programowanie Obiektowe

Prowadzący: dr. inż. Wojciech Kozioł

Rzeszów, 2025 r

Spis treści

1. Wstęp

Niniejszy dokument przedstawia sprawozdanie techniczne z realizacji projektu pod tytułem: Fleetmanagement – System do zarządzania flotą pojazdów. Projekt powstał w ramach przedmiotu Programowanie Obiektowe na Uniwersytecie Rzeszowskim.

Cel Programu

Głównym celem projektu jest stworzenie aplikacji umożliwiającej efektywne zarządzanie flotą pojazdów, w tym ewidencję pojazdów, kierowców oraz przypisań pojazdów do kierowców na określone zadania lub okresy.

1.2 Funkcjonalność dla Pracowników

System udostępnia następujące funkcjonalności

- Zarządzanie pojazdami
    - Dodawanie nowych pojazdów do bazy danych z uwzględnieniem ich szczegółowych atrybutów
    - Przeglądanie wszystkich pojazdów
    - Edycja danych pojazdów
    - Wyświetlanie szczegółowych informacji i danym pojeździe
    - Usuwanie
- Zarządzanie kierowcami
    - Dodawanie nowych kierowców z ich danymi personalnymi i zawodowymi
    - Przegląd wszystkich kierowców
    - Edycja danych kierowców
    - Wyświetlanie szczegółowych informacji o wybranym kierowcy
    - Usuwanie
- Zarządzanie przypasaniami
    - Tworzenie nowych przypisań, łączących konkretny pojazd z konkretnym kierowcą na określony okres i zadanie
    - Ewidencja szczegółów przypisani
    - Przegląd listy wszystkich przypisań
    - Edycja istniejących przypisani
    - Wyświetlanie szczegółowych informacji o wybranym przypisaniu
    - Usuwanie

2. Architektura aplikacji

Projekt został zrealizowany w oparciu o podejście warstwowe, co sprzyja modularności i potencjałowi na rozbudowę

2.1 Warstwy aplikacji

Wyróżniono następujące warstwy

- Warstwa prezentacji: odpowiada za interakcje z uzrytkownikami. Zrealizowana jest za pomocą javaFX i polików FXML. Logika tej warstwy zanjduje się w klasach kontrolerów
- Warstwa dostępu do danych: zapewnia komunikacje z bazaą postgreSQL. Obejmuje klasy encji, które mapuja obiekty javy na tabele bazy danych oraz klasy DAO, które implementują operacje CRUD, kluczową role odgrywa tu framework HIbernate.

2.2 Użyte technologie

- Java (JDK 24): Język programowania użyty do stworzenia aplikacji.
- JavaFX (wersja 17.0.6): Biblioteka do tworzenia graficznego interfejsu użytkownika. Pliki FXML zostały użyte do deklaratywnego definiowania struktury GUI.
- Hibernate (wersja6.4.4.Final): Framework ORM (**Object-Relational Mapping**) do mapowania obiektowo-relacyjnego i zarządzania interakcjami z bazą danych. Konfiguracja w pliku hibernate.cfg.xml.
- PostgreSQL (wersja 42.7.6): Relacyjna baza danych używana do przechowywania danych aplikacji.
- Maven: Narzędzie do zarządzania zależnościami i budowania projektu

2.3 Struktura Projektu

Struktura katalogów i pakietów projektu została zorganizowana w następujący sposób:

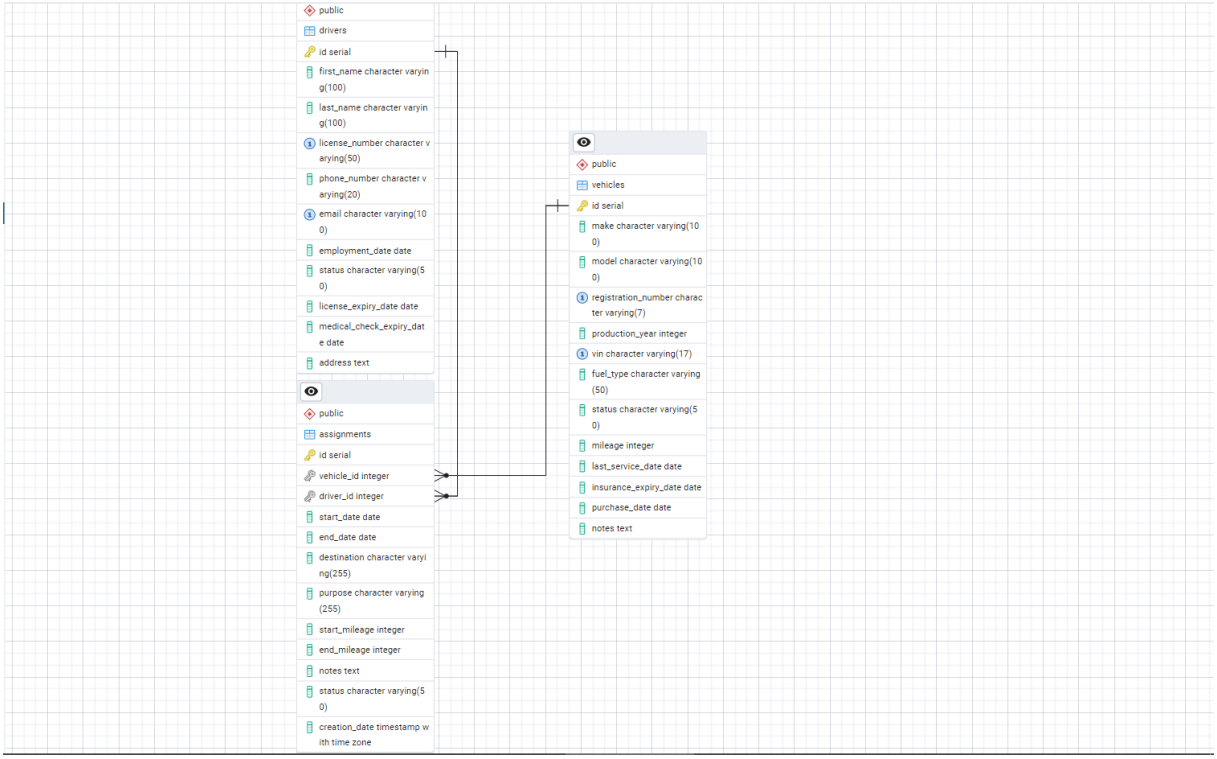
- src/main/java/com/example/fleetmanagement/:
    - MainApp.java: Główna klasa startowa aplikacji.
    - controller/: Klasy kontrolerów JavaFX.
    - model/: Klasy encji JPA.
    - dao/: Klasy Data Access Object.
    - util/: Klasy pomocnicze (np. HibernateUtil.java).
- src/main/resources/:
    - hibernate.cfg.xml: Plik konfiguracyjny Hibernate.
    - com/example/fleetmanagement/: Pliki FXML dla głównych widoków.

- com/example/fleetmanagement/Dialog/: Pliki FXML dla okien dialogowych.

- pom.xml: Plik konfiguracyjny Mavena.

3. Opis Bazy danych

3.1 Schemat bazy danych (Diagram ERD)



Rysunek 1. Diagram ERD. Źródło: pgadmin – projekt własny

Diagram przedstawia trzy główne tabele: vehicles, drivers i assignments, wraz z ich kolumnami i relacjami.

3.2 Opis tabel

- **Tabela: vehicles**
  Przechowuje szczegółowe informacje o pojazdach.
  - Id (SERIAL PRIMARY KEY): Identyfikator pojazdu.
  - Make (VARCHAR(100) NOT NULL): Marka.
  - Model (VARCHAR(100) NOT NULL): Model.
  - registration_number (VARCHAR(20) UNIQUE NOT NULL): Numer rejestracyjny.
  - production_year (INT): Rok produkcji.

- Vin (VARCHAR(17) UNIQUE): Numer VIN.
- fuel_type (VARCHAR(50)): Rodzaj paliwa.
- Status (VARCHAR(50) DEFAULT 'Dostępny'): Status pojazdu.
- Mileage (INT): Przebieg.
- last_service_date (DATE): Data ostatniego serwisu.
- insurance_expiry_date (DATE): Ważność ubezpieczenia.
- purchase_date (DATE): Data zakupu.
- Notes (TEXT): Notatki.

- **Tabela: drivers**
  Przechowuje informacje o kierowcach.
  - Id (SERIAL PRIMARY KEY): Identyfikator kierowcy.
  - first_name (VARCHAR(100) NOT NULL): Imię.
  - last_name (VARCHAR(100) NOT NULL): Nazwisko.
  - license_number (VARCHAR(50) UNIQUE NOT NULL): Numer prawa jazdy.
  - phone_number (VARCHAR(20)): Numer telefonu.
  - Email (VARCHAR(100) UNIQUE): Adres e-mail.
  - employment_date (DATE): Data zatrudnienia.
  - Status (VARCHAR(50) DEFAULT 'Aktywny'): Status kierowcy.
  - license_expiry_date (DATE): Ważność prawa jazdy.
  - medical_check_expiry_date (DATE): Ważność badań lekarskich.
  - Address (TEXT): Adres.
- **Tabela: assignments**
  Tabela przypisań pojazdów do kierowców.
  - Id (SERIAL PRIMARY KEY): Identyfikator przypisania.
  - vehicle_id (INT REFERENCES vehicles(id) ON DELETE SET NULL): ID pojazdu.
  - driver_id (INT REFERENCES drivers(id) ON DELETE SET NULL): ID kierowcy.
  - start_date (DATE NOT NULL): Data rozpoczęcia.
  - end_date (DATE): Data zakończenia.

- Destination (VARCHAR(255)): Cel (krótki opis).
- Purpose (VARCHAR(255)): Cel (szczegółowy opis).
- start_mileage (INT): Przebieg początkowy.
- end_mileage (INT): Przebieg końcowy.
- Notes (TEXT): Notatki do przypisania.
- Status (VARCHAR(50) DEFAULT 'Zaplanowane'): Status przypisania.
- creation_date (TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP): Data utworzenia rekordu.

3.3 Opis relacji

- Relacja jeden-do-wielu między vehicles a assignments (poprze vehicle_id).
- Relacja jeden-do-wielu między drivers a assignments (poprzez driver_id).
  Klucze obce używają ON DELETE SET NULL, co oznacza, że usunięcie pojazdu lub kierowcy spowoduje ustawienie odpowiedniego klucza obcego w powiązanych przypisaniach na NULL, ale nie usunie samego rekordu przypisania.

4. Funkcjonalność Aplikacji i Przepływ Danych

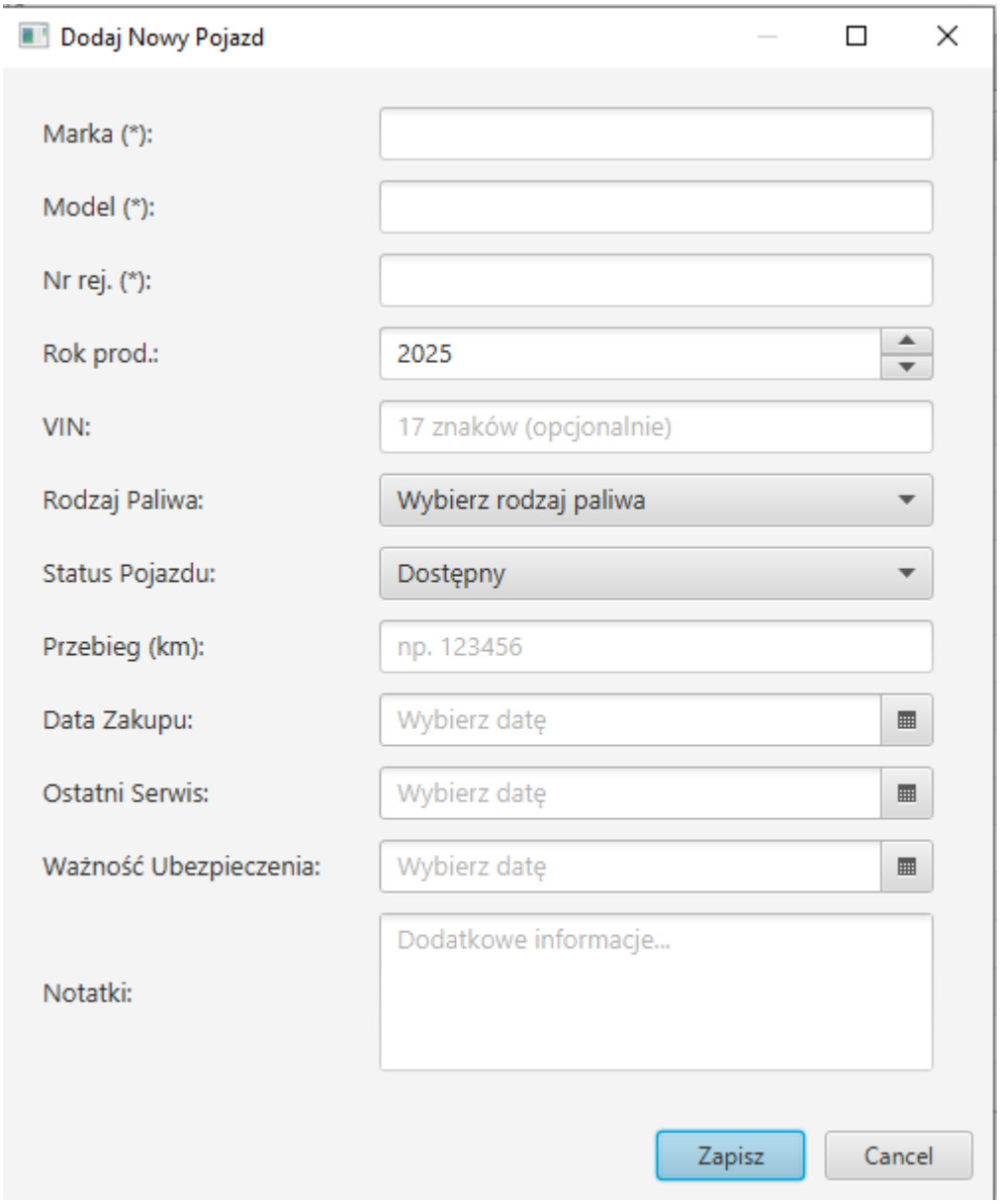
4.1 Główne Moduły i Ich Funkcjonalności

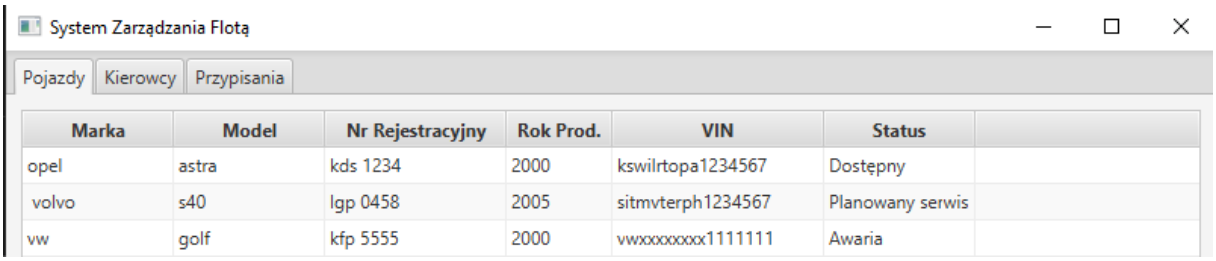
Aplikacja składa się z trzech głównych modułów dostępnych poprzez zakładki: Pojazdy, Kierowcy, Przypisania.

4.1.1 Moduł Pojazdów

- Wyświetlanie listy pojazdów w tabeli (VehicleView.fxml, VehicleController).
- Dodawanie nowego pojazdu poprzez okno dialogowe (VehicleDialog.fxml).
- Edycja danych wybranego pojazdu poprzez okno dialogowe.
- Wyświetlanie wszystkich szczegółów wybranego pojazdu w oknie dialogowym (tylko do odczytu).
- Usuwanie wybranego pojazdu (po potwierdzeniu).

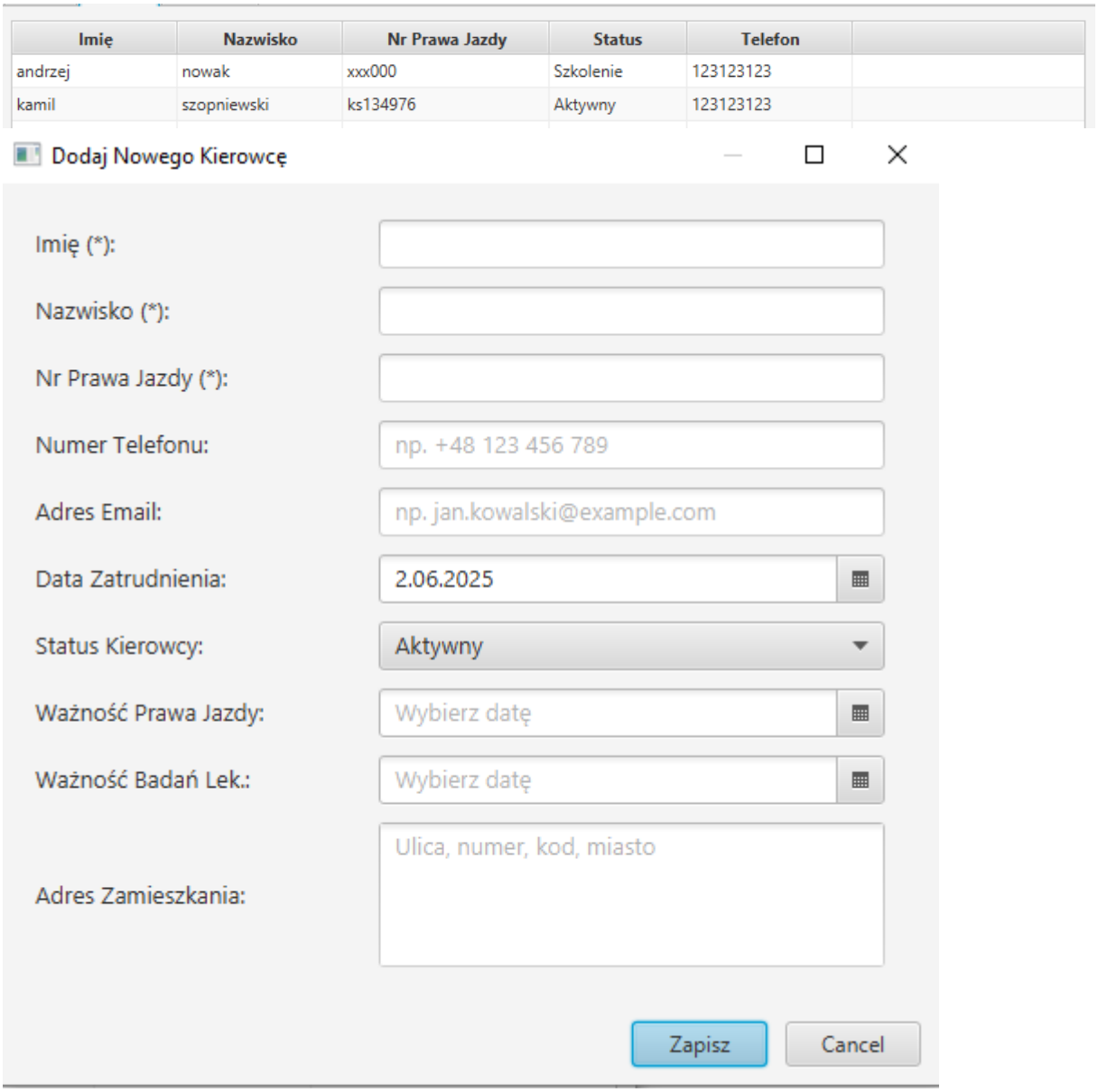
- Odświeżanie listy pojazdów.





4.1.2 Moduł Kierowców

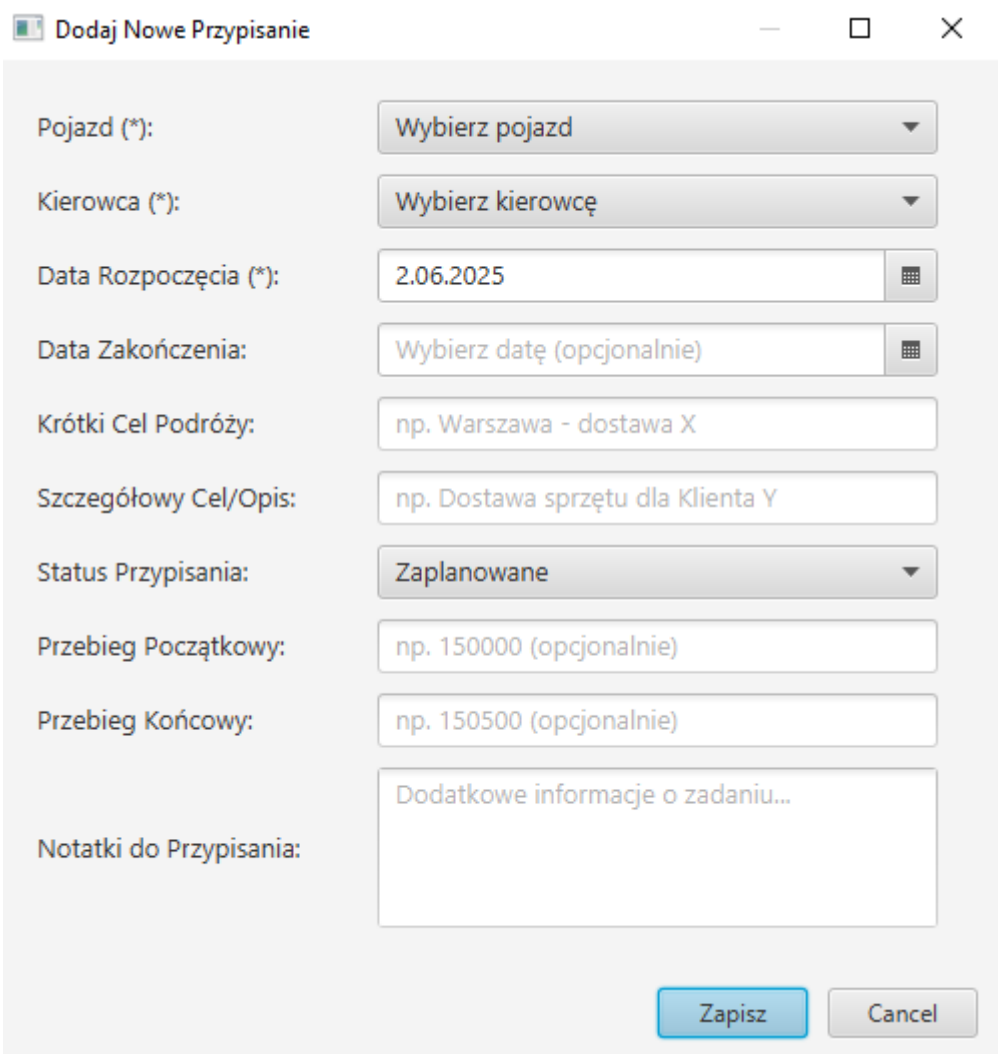
- Wyświetlanie listy kierowców (DriverView.fxml, DriverController).
- Funkcjonalności CRUD analogiczne do modułu pojazdów, realizowane przez DriverDialog.fxml.



4.1.3 Moduł Przypisań

- Wyświetlanie listy przypisań (AssignmentView.fxml, AssignmentController).
- Funkcjonalności CRUD realizowane przez AssignmentDialog.fxml.
- Formularz przypisania zawiera ComboBoxy do wyboru pojazdu i kierowcy, DatePickery oraz pola tekstowe.
- Zaawansowana walidacja danych, w tym sprawdzanie konfliktów terminów dla pojazdów i kierowców.

Pojazd	Kierowca	Od Kiedy	Do Kiedy	Cel (Krótki)	Status Zadania	Cel (Szczegóły)
volvo s40 (lgp 0458)	kamil szopniewski (ks134...	2025-06-02	2025-06-14	krakow	Zaplanowane	dostawa towaru do klient...
opel astra (kds 1234)	andrzej nowak (xxx000)	2025-06-02	2025-06-27	warszawa	Zaplanowane	dostawa do klienta x



5. Opis Wybranych Fragmentów Kodu Źródłowego

5.1 Zarządzanie Sesjami Hibernate (HibernateUtil.java)

```java
private static final SessionFactory sessionFactory = buildSessionFactory();

private static SessionFactory buildSessionFactory() {  1 usage  Szopex321
    try {
        //tworzy SessionFactory z hibernate.cfg.xml
        return new Configuration().configure().buildSessionFactory();
    } catch (Throwable ex) {
        System.err.println("Initial SessionFactory creation failed." + ex);
        throw new ExceptionInInitializerError(ex);
    }
}

public static SessionFactory getSessionFactory() {  16 usages  Szopex321
    return sessionFactory;
}

public static void shutdown() {  1 usage  Szopex321
    getSessionFactory().close();
}
```

Opis: Klasa HibernateUti jest kluczowa dla interakcji z bazą danych. Inicjalizuje SessionFactory na podstawie pliku hibernate.cfg.xml. Zapewnia singletonowy dostęp do SessionFactory oraz metodę shutdown() do poprawnego zamykania zasobów Hibernate.

5.2 Przykład klasy DAO (VehicleDao.java - metoda save i findAll)

```java
public void save(Vehicle vehicle) {  1 usage  Szopex321
    Transaction transaction = null;
    try (Session session = HibernateUtil.getSessionFactory().openSession()) {
        transaction = session.beginTransaction();
        session.persist(vehicle);
        transaction.commit();
    } catch (Exception e) {
        if (transaction != null) {
            transaction.rollback();
        }
        e.printStackTrace();
    }
}
```

```java
public List<Vehicle> findAll() {  2 usages  Szopex321
    try (Session session = HibernateUtil.getSessionFactory().openSession()) {
        return session.createQuery( s: "FROM Vehicle", Vehicle.class).list();
    } catch (Exception e) {
        e.printStackTrace();
        return List.of(); // Zwróć pustą listę w razie błędu
    }
}
```

Opis: Klasy DAO, takie jak VehicleDao, enkapsulują logikę dostępu do danych dla poszczególnych encji. Metoda save ilustruje zapis obiektu w transakcji. Metoda findAll pokazuje pobieranie wszystkich rekordów z tabeli przy użyciu HQL (Hibernate Query Language).

5.3 Obsługa okien dialogowych (VehicleController.handleAddVehicle())

```
Dialog<ButtonType> dialog = new Dialog<>();
dialog.setTitle("Dodaj Nowy Pojazd");
dialog.getDialogPane().setContent(dialogPaneContent);
ButtonType okButtonType = new ButtonType( s: "Zapisz", ButtonBar.ButtonData.OK_DONE);
dialog.getDialogPane().getButtonTypes().addAll(okButtonType, ButtonType.CANCEL);
dialog.setResizable(true);

Optional<ButtonType> result = dialog.showAndWait();
if (result.isPresent() && result.get() == okButtonType) {
    if (validateDialogInput(dialogPaneContent)) {
        Vehicle newVehicle = new Vehicle();
        updateVehicleFromDialog(newVehicle, dialogPaneContent);
        try {
            vehicleDao.save(newVehicle);
            loadVehicles(); // przeładowanie widoku listy
        } catch (Exception e) {
            showError( title: "Błąd dodawania pojazdu",  content: "Nie udało się dodać pojazdu.
            e.printStackTrace();
        }
```

Opis: Metody obsługi zdarzeń w kontrolerach, takie jak handleAddVehicle, są odpowiedzialne za otwieranie okien dialogowych (ładowanych z plików FXML), zbieranie danych od użytkownika, walidację tych danych, a następnie interakcję z warstwą DAO w celu utrwalenia zmian w bazie.

5.4 Walidacja danych (VehicleController.validateDialogInput()

```
if (makeField.getText() == null || makeField.getText().trim().isEmpty()) errorMessage.append("Marka jest wymagana.\n");
if (modelField.getText() == null || modelField.getText().trim().isEmpty()) errorMessage.append("Model jest wymagany.\n");
if (regNumberField.getText() == null || regNumberField.getText().trim().isEmpty()) errorMessage.append("Numer rejestracyjny jest wymagany.\n")
String vinTextVal = vinField.getText();
if (vinTextVal != null && !vinTextVal.trim().isEmpty() && vinTextVal.trim().length() != 17)
    errorMessage.append("Numer VIN musi mieć 17 znaków (lub pozostać pusty).\n");
if (statusComboBox.getValue() == null || statusComboBox.getValue().trim().isEmpty())
    errorMessage.append("Status pojazdu jest wymagany.\n");
String mileageTextVal = mileageField.getText();
if (mileageTextVal != null && !mileageTextVal.trim().isEmpty()) {
    try {
        int mileage = Integer.parseInt(mileageTextVal.trim());
        if (mileage < 0) errorMessage.append("Przebieg nie może być ujemny.\n");
    } catch (NumberFormatException e) { errorMessage.append("Przebieg musi być poprawną liczbą całkowitą.\n"); }
}
```

```
// Walidacja dat
LocalDate purchaseDate = purchaseDatePicker.getValue();
LocalDate lastServiceDate = lastServiceDatePicker.getValue();
LocalDate insuranceExpiryDateVal = insuranceExpiryDatePicker.getValue();
Integer productionYearVal = yearSpinner.getValue();

if (productionYearVal != null) {
    if (purchaseDate != null && purchaseDate.getYear() < productionYearVal) {
        errorMessage.append("Rok daty zakupu nie może być wcześniejszy niż rok produkcji.\n");
    }
    if (lastServiceDate != null && lastServiceDate.isBefore(LocalDate.of(productionYearVal,  month: 1,  dayOfMonth: 1))) {
        errorMessage.append("Data ostatniego serwisu nie może być wcześniejsza niż rok produkcji.\n");
    }
    if (insuranceExpiryDateVal != null && insuranceExpiryDateVal.isBefore(LocalDate.of(productionYearVal,  month: 1,  dayOfMonth: 1).plusDays( daysToAdd: 1)))
        errorMessage.append("Data ważności ubezpieczenia musi być późniejsza niż rok produkcji.\n");
    }
}
if (lastServiceDate != null && purchaseDate != null && lastServiceDate.isBefore(purchaseDate)) {
    errorMessage.append("Data ostatniego serwisu nie może być wcześniejsza niż data zakupu.\n");
}
```

Opis: Walidacja danych wejściowych jest kluczowym elementem zapewnienia ich poprawności i integralności. Realizowana jest w kontrolerach przed próbą zapisu lub aktualizacji danych. Obejmuje sprawdzanie wymaganych pól, formatów (np. VIN, e-mail), logicznych zależności między datami (np. data serwisu nie wcześniejsza niż rok produkcji) oraz unikanie konfliktów (np. podwójne przypisanie pojazdu/kierowcy w tym samym czasie).

6. Repozytorium GitHub

Cały kod źródłowy wraz z plikami projektu znajduję się na serwisie GitHub: https://github.com/Szopex321/fleetmanagement

7. Podsumowanie i Wnioski

Projekt "System Zarządzania Flotą Pojazdów" z powodzeniem realizuje założone cele, dostarczając funkcjonalną aplikację desktopową do zarządzania flotą. Zastosowanie technologii JavaFX, Hibernate i PostgreSQL pozwoliło na stworzenie stabilnego i rozszerzalnego rozwiązania.

Główne osiągnięcia:

- Stworzenie w pełni funkcjonalnego interfejsu GUI dla operacji CRUD na pojazdach, kierowcach i przypisaniach.
- Implementacja zaawansowanej walidacji danych, w tym sprawdzania unikalności i konfliktów biznesowych.
- Efektywne wykorzystanie Hibernate do mapowania obiektowo-relacyjnego i zarządzania transakcjami.
- Zastosowanie wzorca projektowego DAO do separacji logiki dostępu do danych.
- Wykorzystanie okien dialogowych dla intuicyjnego wprowadzania i edycji danych.

Wnioski i potencjalny rozwój:
Realizacja projektu stanowiła cenne doświadczenie w zakresie tworzenia aplikacji z użyciem nowoczesnych technologii Java. Projekt można dalej rozwijać, np. poprzez:

- Dodanie modułu raportowania.
- Implementację logowania użytkowników i zarządzania uprawnieniami.
- Wprowadzenie bardziej zaawansowanych opcji wyszukiwania i filtrowania.