Instytut Informatyki
Uniwersytet Rzeszowski

Przedmiot: Bazy danych

Dokumentacja Techniczna Systemu Wypożyczalni Samochodów "Rent-A-Car OS"

Autor: Artur Jurkowski | Mikołaj Kopacz

134916 | 134926

Opis: System obsługi i zarządzania działalnością wypożyczalni samochodów z wykorzystaniem bazy danych PostgreSQL oraz aplikacji webowej Python Streamlit.

Link do strony:

https://projektbazydanych-tkiczs4cek4rkutnibup8v.streamlit.app/

Rzeszów 2026

1. Specyfikacja tematu projektu

1.1 Geneza problemu

Współczesny rynek wynajmu pojazdów, mimo rosnącej cyfryzacji, w sektorze małych i średnich przedsiębiorstw (MŚP) często opiera się na rozwiązaniach doraźnych. Wypożyczalnie korzystają z zeszytów rezerwacji, rozproszonych plików tekstowych lub arkuszy kalkulacyjnych, które nie są ze sobą zintegrowane.

Taki model zarządzania prowadzi do krytycznych problemów operacyjnych:

- **Brak integralności danych:** Informacje o klientach i flocie są niespójne (np. literówki w numerach VIN, dublowanie kartotek klientów).
- **Konflikty rezerwacji (Overbooking):** Brak centralnej walidacji dostępności aut w czasie rzeczywistym skutkuje przypisaniem jednego pojazdu dwóm klientom.
- **Ryzyko prawne:** Wydawanie pojazdów bez ważnych przeglądów technicznych z powodu braku automatycznych powiadomień.
- **Utrudniona analityka:** Brak narzędzi do szybkiej oceny rentowności floty i efektywności pracowników.

1.2 Założenia projektowe

Celem projektu było stworzenie zintegrowanego systemu "Rent-A-Car OS", który eliminuje "papierologię" i automatyzuje kluczowe procesy.

Główne filary systemu to:

1. **Architektura "Thick Database":** Przeniesienie logiki biznesowej (walidacja, obliczenia) do warstwy bazy danych PostgreSQL (procedury składowane, triggery, constraints).
2. **Centralizacja:** Wszystkie dane (CRM, Flota, Finanse) w jednej relacyjnej strukturze.
3. **Dostępność:** Interfejs webowy (Streamlit) dostępny przez przeglądarkę, niewymagający instalacji po stronie klienta.
4. **Bezpieczeństwo:** System ról (Menadżer/Pracownik) oraz mechanizmy zapobiegające usuwaniu danych historycznych.

---

2. Analiza wymagań i narzędzia

2.1 Środowisko technologiczne

Projekt zrealizowano w architekturze klient-serwer, wykorzystując nowoczesne i wydajne narzędzia:

- **Warstwa Danych (Backend):** PostgreSQL 15+ (obsługa transakcji, procedury PL/pgSQL).
- **Warstwa Aplikacji (Middleware/Frontend):** Python 3.8+ z frameworkiem Streamlit.
- **Środowisko programistyczne:** PyCharm / VS Code.

2.2 Kluczowe biblioteki

W pliku `requirements.txt` zdefiniowano zależności niezbędne do działania systemu:

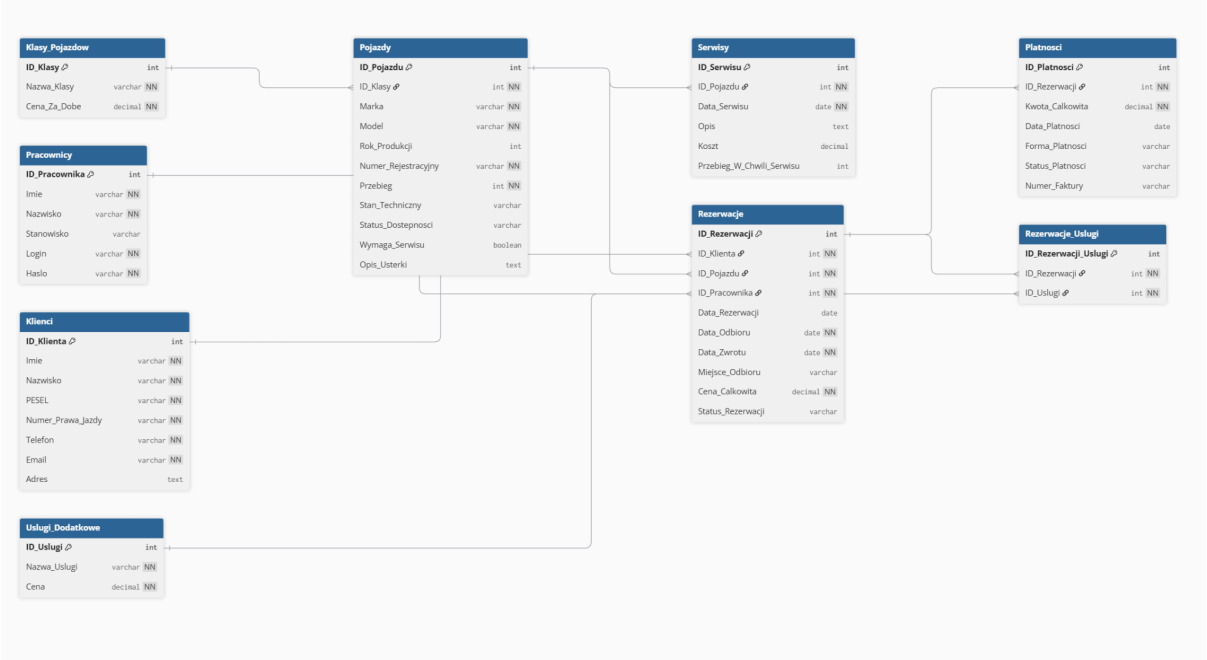
- `streamlit` – framework do budowy interfejsu graficznego (GUI) w modelu webowym.
- `psycopg2-binary` – sterownik bazy danych PostgreSQL, zapewniający bezpieczne (parametryzowane) zapytania SQL.
- `pandas` – biblioteka do przetwarzania wyników zapytań SQL do postaci ramek danych (DataFrames), co ułatwia ich prezentację w tabelach i na wykresach.
- `fpdf` – biblioteka służąca do dynamicznego generowania plików PDF (potwierdzenia rezerwacji).

---

3. Projekt Bazy Danych (Struktura Tabel)

Baza danych została zaprojektowana w trzeciej postaci normalnej (3NF). Składa się z **9 tabel**, których szczegółowa specyfikacja znajduje się poniżej.

3.1 Schemat ERDDiagram obrazuje relacje między encjami. Centralną tabelą jest `Rezerwacje`, łącząca klienta, pojazd i pracownika



Rys. 1. Schemat relacji bazy danych.

3.2 Opis relacji między tabelami (Logika ERD)

Diagram ERD (Entity Relationship Diagram) przedstawia strukturę powiązań między encjami systemu. W projekcie zastosowano ***więzy integralności referencyjnej*** *typu `NOT NULL` dla kluczy obcych, co wymusza spójność danych na poziomie bazy.*

Poniżej scharakteryzowano najważniejsze relacje:

1. ***Klasy_Pojazdow (1) — (N) Pojazdy***
    - *Typ: Jeden do Wielu (Wymagana).*
    - *Opis: Każdy pojazd w systemie musi być przypisany do dokładnie jednej klasy (np. "Premium", "Ekonomiczna"). Usunięcie klasy jest niemożliwe, jeśli są do niej przypisane jakiekolwiek pojazdy.*
2. ***Klienci (1) — (N) Rezerwacje***
    - *Typ: Jeden do Wielu (Wymagana).*
    - *Opis: Każda rezerwacja musi wskazywać na konkretnego klienta z bazy. Jeden klient może posiadać wiele rezerwacji (historia wypożyczeń).*
3. ***Pojazdy (1) — (N) Rezerwacje***

    - *Typ: Jeden do Wielu (Wymagana).*
    - *Opis: Rezerwacja dotyczy konkretnego, fizycznego pojazdu. System uniemożliwia stworzenie rezerwacji "w powietrzu" (bez przypisanego auta).*
4. ***Pracownicy (1) — (N) Rezerwacje***
    - *Typ: Jeden do Wielu (Wymagana).*
    - *Opis: Każda transakcja musi posiadać "opiekuna" – pracownika, który ją utworzył w systemie. Pozwala to na pełną audytowalność działań i ocenę efektywności personelu.*
5. ***Pojazdy (1) — (N) Serwisy***
    - *Typ: Jeden do Wielu (Opcjonalna od strony serwisu).*
    - *Opis: Pojazd może mieć wiele wpisów w historii napraw. Wpis serwisowy nie może istnieć bez wskazania pojazdu, którego dotyczy.*
6. ***Rezerwacje (1) — (N) Platnosci***
    - *Typ: Jeden do Wielu (Wymagana, Kaskadowa).*
    - *Opis: Do rezerwacji generowana jest płatność. Zastosowano klauzulę `ON DELETE CASCADE` – usunięcie lub anulowanie rezerwacji powoduje automatyczne usunięcie powiązanego rekordu płatności, co zapobiega niespójnościom finansowym.*
7. ***Rezerwacje (M) — (N) Uslugi_Dodatkowe***
    - *Typ: Wiele do Wielu.*
    - *Opis: Realizowana przez tabelę łączącą `Rezerwacje_Uslugi`. Jedna rezerwacja może zawierać wiele dodatków (np. GPS i Fotelik), a ta sama usługa z cennika może być wybierana wielokrotnie przez różnych klientów.*

3.3 Wyjaśnienie notacji graficznej

Na diagramie zastosowano notację "Crow's Foot" z uwzględnieniem wymagalności:

- ***Linia ciągła prostopadła (||)** przy tabeli nadrzędnej oznacza relację obowiązkową (np. rezerwacja **musi** mieć klienta).*
- ***Symbol "kurzej stopki"** oznacza liczność "wiele" (np. jeden klient może mieć **wiele** rezerwacji).*

3.4 Opis szczegółowy tabel (DDL)

Poniżej przedstawiono definicje tabel w języku SQL. W modelu fizycznym zastosowano klauzulę `NOT NULL` dla kluczy obcych, aby wymusić spójność danych (np. każda rezerwacja musi być przypisana do konkretnego klienta i pojazdu).

3.4.1 Tabela Klasy_Pojazdow

Słownik definiujący segmenty aut. Wydzielenie ceny do tej tabeli pozwala na globalną zmianę stawek dla całej grupy aut.

SQL
```
CREATE TABLE model.Klasy_Pojazdow (
```

```
    ID_Klasy SERIAL PRIMARY KEY,
    Nazwa_Klasy VARCHAR(50) UNIQUE NOT NULL,
    Cena_Za_Dobe DECIMAL(10, 2) NOT NULL
);
```

3.4.2 Tabela Pracownicy

Zawiera dane personelu. Kolumna `Login` jest unikalna, co jest fundamentem systemu logowania.

```
SQL
CREATE TABLE model.Pracownicy (
    ID_Pracownika SERIAL PRIMARY KEY,
    Imie VARCHAR(50) NOT NULL,
    Nazwisko VARCHAR(50) NOT NULL,
    Stanowisko VARCHAR(50),
    Login VARCHAR(50) UNIQUE NOT NULL,
    Haslo VARCHAR(50) NOT NULL
);
```

3.4.3 Tabela Klienci

Baza CRM. Zastosowano kluczowe ograniczenia `UNIQUE` dla numeru PESEL oraz Prawa Jazdy, aby zapobiec dublowaniu klientów.

```
SQL
CREATE TABLE model.Klienci (
    ID_Klienta SERIAL PRIMARY KEY,
    Imie VARCHAR(50) NOT NULL,
    Nazwisko VARCHAR(50) NOT NULL,
    PESEL VARCHAR(11) UNIQUE NOT NULL,
    Numer_Prawa_Jazdy VARCHAR(20) UNIQUE NOT NULL,
    Telefon VARCHAR(15) NOT NULL,
    Email VARCHAR(100) NOT NULL,
    Adres TEXT
);
```

3.4.4 Tabela Pojazdy

Ewidencja floty. Relacja z tabelą klas jest wymagana (`NOT NULL`). Constraint `CHECK` na polu `Status_Dostepnosci` wymusza jeden z trzech stanów: 'Dostępny', 'Wypożyczony', 'W serwisie'.

```
SQL
```

```
CREATE TABLE model.Pojazdy (
    ID_Pojazdu SERIAL PRIMARY KEY,
    ID_Klasy INT REFERENCES model.Klasy_Pojazdow(ID_Klasy) NOT NULL,
    Marka VARCHAR(50) NOT NULL,
    Model VARCHAR(50) NOT NULL,
    Rok_Produkcji INT,
    Numer_Rejestracyjny VARCHAR(20) UNIQUE NOT NULL,
    Przebieg INT NOT NULL,
    Stan_Techniczny VARCHAR(50),
    Status_Dostepnosci VARCHAR(20) CHECK (Status_Dostepnosci IN ('Dostępny',
'Wypożyczony', 'W serwisie')),
    Wymaga_Serwisu BOOLEAN DEFAULT FALSE,
    Opis_Usterki TEXT DEFAULT NULL
);
```

3.4.5 Tabela Rezerwacje

Kluczowa tabela transakcyjna. Wymusza istnienie powiązań z klientem, pojazdem i pracownikiem. Zawiera walidację logiczną dat: data zwrotu musi być późniejsza lub równa dacie odbioru.

```
SQL
CREATE TABLE model.Rezerwacje (
    ID_Rezerwacji SERIAL PRIMARY KEY,
    ID_Klienta INT REFERENCES model.Klienci(ID_Klienta) NOT NULL,
    ID_Pojazdu INT REFERENCES model.Pojazdy(ID_Pojazdu) NOT NULL,
    ID_Pracownika INT REFERENCES model.Pracownicy(ID_Pracownika) NOT NULL,
    Data_Rezerwacji DATE DEFAULT CURRENT_DATE,
    Data_Odbioru DATE NOT NULL,
    Data_Zwrotu DATE NOT NULL,
    Miejsce_Odbioru VARCHAR(100),
    Cena_Calkowita DECIMAL(10, 2) NOT NULL,
    Status_Rezerwacji VARCHAR(20) CHECK (Status_Rezerwacji IN ('Potwierdzona',
'Anulowana', 'Zakończona', 'W trakcie')),
    CHECK (Data_Zwrotu >= Data_Odbioru)
);
```

3.4.6 Tabela Serwisy

Rejestr napraw i przeglądów. Każdy wpis serwisowy musi dotyczyć konkretnego pojazdu (`NOT NULL`).

```
SQL
CREATE TABLE model.Serwisy (
    ID_Serwisu SERIAL PRIMARY KEY,
    ID_Pojazdu INT REFERENCES model.Pojazdy(ID_Pojazdu) NOT NULL,
```

```
    Data_Serwisu DATE NOT NULL,
    Opis TEXT,
    Koszt DECIMAL(10, 2),
    Przebieg_W_Chwili_Serwisu INT
);
```

3.4.7 Tabela Uslugi_Dodatkowe

Słownik usług płatnych extra (np. GPS, fotelik).

```
SQL
CREATE TABLE model.Uslugi_Dodatkowe (
    ID_Uslugi SERIAL PRIMARY KEY,
    Nazwa_Uslugi VARCHAR(100) NOT NULL,
    Cena DECIMAL(10, 2) NOT NULL
);
```

3.4.8 Tabela Rezerwacje_Uslugi

Tabela łącząca (wiele-do-wielu), pozwalająca przypisać wiele usług dodatkowych do jednej rezerwacji. Zastosowano `ON DELETE CASCADE` dla rezerwacji.

```
SQL
CREATE TABLE model.Rezerwacje_Uslugi (
    ID_Rezerwacji_Uslugi SERIAL PRIMARY KEY,
    ID_Rezerwacji INT REFERENCES model.Rezerwacje(ID_Rezerwacji) ON DELETE
CASCADE NOT NULL,
    ID_Uslugi INT REFERENCES model.Uslugi_Dodatkowe(ID_Uslugi) NOT NULL,
    UNIQUE(ID_Rezerwacji, ID_Uslugi)
);
```

3.4.9 Tabela Platnosci

Ewidencja wpływów finansowych z walidacją formy płatności. Płatność jest ściśle powiązana z rezerwacją – usunięcie rezerwacji usuwa płatność (`ON DELETE CASCADE`).

```
SQL
CREATE TABLE model.Platnosci (
    ID_Platnosci SERIAL PRIMARY KEY,
    ID_Rezerwacji INT REFERENCES model.Rezerwacje(ID_Rezerwacji) ON DELETE
CASCADE NOT NULL,
    Kwota_Calkowita DECIMAL(10, 2) NOT NULL,
    Data_Platnosci DATE DEFAULT CURRENT_DATE,
```

```
    Forma_Platnosci VARCHAR(20) CHECK (Forma_Platnosci IN ('Gotówka', 'Karta',
'Przelew')),
    Status_Platnosci VARCHAR(20) CHECK (Status_Platnosci IN ('Oczekująca',
'Zrealizowana', 'Anulowana')),
    Numer_Faktury VARCHAR(50)
);
```

---

4. Logika Biznesowa (Backend SQL)

Kluczowe operacje na danych zostały zamknięte w procedurach składowanych (Stored Procedures), co zapewnia bezpieczeństwo i spójność.

4.1 Kluczowe procedury i funkcje

A. Procedura `sp_dodaj_klienta` Bezpieczne dodawanie klienta. Sprawdza duplikaty (PESEL/Prawo Jazdy) przed wykonaniem `INSERT`. W przypadku błędu rzuca wyjątek `RAISE EXCEPTION`.

B. Funkcja `ZnajdzDostepnePojazdy` (Algorytm anty-overbooking) Najważniejsza funkcja systemu. Sprawdza, czy w zadanym przedziale dat auto nie ma kolizji z inną rezerwacją.

```
SQL
CREATE OR REPLACE FUNCTION ZnajdzDostepnePojazdy(...) RETURNS TABLE (...) AS
$$
BEGIN
    RETURN QUERY SELECT ... FROM Pojazdy p
    WHERE p.Status_Dostepnosci != 'W serwisie'
      AND NOT EXISTS (
          SELECT 1 FROM Rezerwacje r
          WHERE r.ID_Pojazdu = p.ID_Pojazdu
          AND r.Status_Rezerwacji IN ('Potwierdzona', 'W trakcie')
          AND daterange(r.Data_Odbioru, r.Data_Zwrotu, '[]') && daterange(p_data_od,
p_data_do, '[]')
      );
END;
$$ LANGUAGE plpgsql;
```

C. Procedura `sp_usun_klienta` Zabezpiecza przed usunięciem klienta posiadającego aktywne (niezwrócone) auto. Jeśli klient ma zakończone rezerwacje, jego dane w tabeli

`Rezerwacje` są anonimizowane (`ID_Klienta = NULL`) przed usunięciem rekordu z tabeli głównej.

---

5. Implementacja Aplikacji (Kod Źródłowy Python)

Warstwa aplikacji została podzielona na dwa główne moduły: `db.py` (obsługa bazy) oraz `app.py` (interfejs użytkownika). Poniżej znajduje się analiza najważniejszych fragmentów kodu.

5.1 Moduł komunikacji z bazą (`src/db.py`)

Ten plik pełni rolę "mostu" między aplikacją a bazą danych. Separuje logikę interfejsu od logiki zapytań SQL.

A. Funkcja `get_connection()` Odpowiada za nawiązanie bezpiecznego połączenia z bazą PostgreSQL. Wykorzystuje bibliotekę `psycopg2`.

Python
```
def get_connection():
    config = get_db_config()
    # Nawiązanie połączenia z parametrami (host, user, password, dbname)
    return psycopg2.connect(**config)
```

B. Funkcja `run_command()` (Obsługa transakcji) Uniwersalna funkcja do wykonywania operacji modyfikujących dane (`INSERT`, `UPDATE`, `CALL`). Kluczowym elementem jest obsługa transakcyjności:

- `conn.commit()` – zatwierdza zmiany, jeśli operacja się powiodła.
- `conn.rollback()` – cofa zmiany w przypadku błędu, co chroni bazę przed niespójnością.

C. Funkcja `check_login()` Bezpieczna weryfikacja użytkownika. Używa zapytań parametryzowanych (`%s`), co **chroni aplikację przed atakami SQL Injection**.

Python
```
def check_login(username, password):
    sql = "SELECT ID_Pracownika... FROM Pracownicy WHERE Login=%s AND Haslo=%s"
    # Parametry są przekazywane jako krotka (tuple), sterownik sam je escapuje
    df = run_query(sql, (username, password))
    if not df.empty:
        return df.iloc[0].to_dict()
    return None
```

5.2 Moduł interfejsu użytkownika (`src/app.py`)

Aplikacja zbudowana w oparciu o framework **Streamlit**, który pozwala na tworzenie reaktywnych interfejsów webowych.

A. Zarządzanie Sesją (`st.session_state`) Aplikacja jest bezstanowa, dlatego wykorzystano obiekt `session_state` do przechowywania informacji o zalogowanym użytkowniku oraz stanie kreatora rezerwacji.

```python
if "logged_in" not in st.session_state:
    st.session_state["logged_in"] = False
# ...
if not st.session_state["logged_in"]:
    # Wyświetl ekran logowania
    with st.form("login_form"):
        # ... logika logowania
```

B. Generowanie PDF (`create_pdf_confirmation`) System wykorzystuje bibliotekę **FPDF** do dynamicznego tworzenia dokumentów potwierdzenia rezerwacji. Funkcja tworzy obiekt PDF, dodaje nagłówki, dane klienta i pojazdu, a następnie zwraca ciąg bajtów gotowy do pobrania przez użytkownika.

```python
def create_pdf_confirmation(klient_info, auto_info, ...):
    pdf = FPDF()
    pdf.add_page()
    pdf.set_font("Arial", "B", 16)
    pdf.cell(0, 10, clean_text("Potwierdzenie Rezerwacji"), ln=True, align="C")
    # ... dodawanie kolejnych linii tekstu
    return bytes(pdf.output())
```

Potwierdzenie Rezerwacji

Data: 2026-01-25

Obsluga: Adam

DANE KLIENTA:

Klient: Klimek Andrzej (55102855561)

POJAZD:

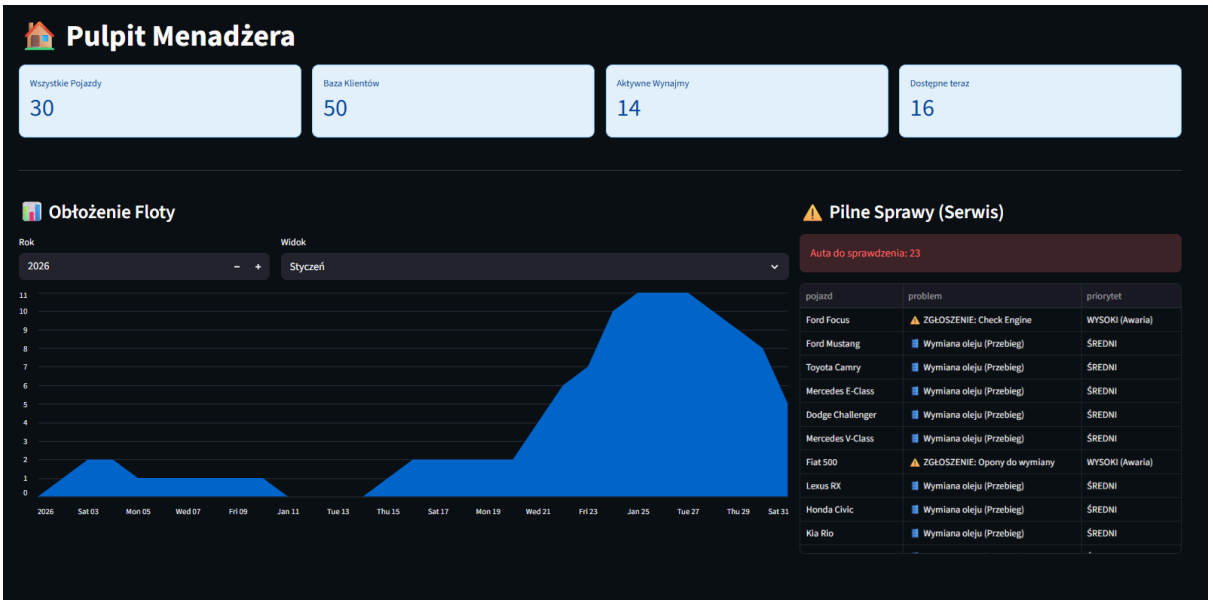
Toyota Yaris (WA 10001)

TERMIN I KOSZT:

2026-01-25 - 2026-01-28

Razem: 270.00 PLN

C. Wizualizacja danych (Pulpit Menadżera) Aplikacja wykorzystuje komponenty `st.metric` do wyświetlania KPI oraz `st.area_chart` do rysowania wykresów obłożenia. Dane do wykresów są pobierane bezpośrednio z funkcji analitycznych SQL (`get_monthly_occupancy`).



5. Realizacja Zapytań Problemowych (Kod SQL i Analiza)

Poniżej przedstawiono implementację 10 kluczowych problemów analitycznych rozwiązanych za pomocą zaawansowanego SQL.

5.1. Implementacja mechanizmów CRUD (Na przykładzie tabeli Klienci)

W systemie zastosowano podejście oparte na procedurach składowanych, aby odseparować warstwę aplikacji od bezpośrednich operacji na tabelach. Poniżej przedstawiono implementację czterech podstawowych operacji dla encji `Klienci`.

A. Create (Dodawanie danych) – Procedura `sp_dodaj_klienta` Procedura ta odpowiada za bezpieczne wprowadzenie nowego klienta. Przed wykonaniem instrukcji `INSERT` następuje walidacja logiczna – system sprawdza, czy w bazie nie istnieje już osoba o takim samym numerze PESEL lub Prawa Jazdy.

Kod SQL:

```
CREATE OR REPLACE PROCEDURE sp_dodaj_klienta(
    p_imie VARCHAR, p_nazwisko VARCHAR, p_pesel VARCHAR,
    p_nr_prawa VARCHAR, p_telefon VARCHAR, p_email VARCHAR, p_adres VARCHAR
) LANGUAGE plpgsql AS $$
BEGIN
    -- Walidacja unikalności PESEL
    IF EXISTS (SELECT 1 FROM Klienci WHERE PESEL = p_pesel) THEN
        RAISE EXCEPTION 'Błąd: Klient o podanym numerze PESEL (%) już istnieje!', p_pesel;
    END IF;
    -- Walidacja unikalności Prawa Jazdy
    IF EXISTS (SELECT 1 FROM Klienci WHERE Numer_Prawa_Jazdy = p_nr_prawa) THEN
        RAISE EXCEPTION 'Błąd: Numer Prawa Jazdy (%) jest już przypisany do innego klienta!', p_nr_prawa;
    END IF;
```

```
    INSERT INTO Klienci (Imie, Nazwisko, PESEL, Numer_Prawa_Jazdy, Telefon, Email,
Adres)

    VALUES (p_imie, p_nazwisko, p_pesel, p_nr_prawa, p_telefon, p_email, p_adres);

END;

$$;
```


```
    INSERT INTO Klienci (Imie, Nazwisko, PESEL, Numer_Prawa_Jazdy, Telefon, Email,
Adres)

    VALUES (p_imie, p_nazwisko, p_pesel, p_nr_prawa, p_telefon, p_email, p_adres);

END;

$$;
```


B. Read (Odczyt danych) – Funkcja `fn_pobierz_klientow` Funkcja zwraca tabelę z listą klientów. Posiada opcjonalny parametr `p_id`. Jeśli zostanie podany, funkcja zwraca jednego klienta, jeśli jest `NULL` – zwraca wszystkich, sortując ich alfabetycznie.

Kod SQL:

```
CREATE OR REPLACE FUNCTION fn_pobierz_klientow(p_id INT DEFAULT NULL)

RETURNS TABLE (ID_Klienta INT, Imie VARCHAR, Nazwisko VARCHAR, PESEL
VARCHAR,

        Nr_Prawa_Jazdy VARCHAR, Telefon VARCHAR, Email VARCHAR, Adres TEXT)

LANGUAGE plpgsql AS $$

BEGIN

    RETURN QUERY

    SELECT k.ID_Klienta, k.Imie, k.Nazwisko, k.PESEL, k.Numer_Prawa_Jazdy, k.Telefon,
k.Email, k.Adres

    FROM Klienci k

    WHERE p_id IS NULL OR k.ID_Klienta = p_id
```

```
    ORDER BY k.Nazwisko, k.Imie;
END;
$$;
```

C. Update (Aktualizacja danych) – Procedura `sp_aktualizuj_klienta` Procedura wykorzystuje funkcję `COALESCE`, co pozwala na elastyczną edycję. Jeśli przekażemy `NULL` w parametrze, stara wartość w bazie pozostanie bez zmian. Dodatkowo procedura sprawdza, czy przy zmianie PESELU nie następuje konflikt z innym użytkownikiem.

Kod SQL:

```
CREATE OR REPLACE PROCEDURE sp_aktualizuj_klienta(
    p_id INT, p_imie VARCHAR, p_nazwisko VARCHAR, p_pesel VARCHAR,
    p_nr_prawa VARCHAR, p_telefon VARCHAR, p_email VARCHAR, p_adres VARCHAR
) LANGUAGE plpgsql AS $$
BEGIN
    -- Sprawdzenie czy nowy PESEL nie należy już do kogoś innego
    IF p_pesel IS NOT NULL AND EXISTS (SELECT 1 FROM Klienci WHERE PESEL = p_pesel AND ID_Klienta != p_id) THEN
        RAISE EXCEPTION 'Błąd: Podany PESEL należy już do innego klienta!';
    END IF;

    UPDATE Klienci
    SET Imie = COALESCE(p_imie, Imie),
        Nazwisko = COALESCE(p_nazwisko, Nazwisko),
        PESEL = COALESCE(p_pesel, PESEL),
        Numer_Prawa_Jazdy = COALESCE(p_nr_prawa, Numer_Prawa_Jazdy),
        Telefon = COALESCE(p_telefon, Telefon),
        Email = COALESCE(p_email, Email),
```

```
    Adres = COALESCE(p_adres, Adres)
  WHERE ID_Klienta = p_id;
END;
$$;
```

D. Delete (Usuwanie danych) – Procedura `sp_usun_klienta` Najbardziej złożona procedura CRUD. Realizuje więzy integralności i politykę bezpieczeństwa danych:

1. Sprawdza, czy klient ma aktywne (niezwrócone) wypożyczenia – jeśli tak, blokuje usunięcie.
2. Sprawdza historię – jeśli klient wypożyczał auto w ciągu ostatniego roku, system blokuje usunięcie ze względu na wymogi archiwizacji (symulacja RODO/wymogów księgowych).
3. Jeśli warunki są spełnione, anonimizuje dane w tabeli `Rezerwacje` (ustawia `ID_Klienta` na `NULL`) i usuwa rekord z tabeli `Klienci`.

Kod SQL:

```
CREATE OR REPLACE PROCEDURE sp_usun_klienta(p_id INT)
LANGUAGE plpgsql AS $$
DECLARE
  v_ostatnia_data DATE;
  v_aktywne INT;
BEGIN
  -- Sprawdzenie aktywnych wypożyczeń
  SELECT COUNT(*) INTO v_aktywne FROM Rezerwacje
  WHERE ID_Klienta = p_id AND Status_Rezerwacji NOT IN ('Zakończona', 'Anulowana');

  IF v_aktywne > 0 THEN
    RAISE EXCEPTION 'Błąd: Nie można usunąć klienta, który ma aktywne rezerwacje!';
  END IF;
```

```
    -- Sprawdzenie historii (RODO/Retencja danych)

    SELECT MAX(Data_Zwrotu) INTO v_ostatnia_data FROM Rezerwacje WHERE
ID_Klienta = p_id;


    IF v_ostatnia_data IS NOT NULL AND v_ostatnia_data > (CURRENT_DATE - INTERVAL
'1 year') THEN

        RAISE EXCEPTION 'Błąd: Ostatnie wypożyczenie było mniej niż rok temu. Dane
muszą zostać zachowane.';

    END IF;


    -- Anonimizacja historii i usunięcie klienta

    UPDATE Rezerwacje SET ID_Klienta = NULL WHERE ID_Klienta = p_id;

    DELETE FROM Klienci WHERE ID_Klienta = p_id;

END;

$$;
```

5.2. Realizacja kluczowych pytań problemowych (Algorytmika w SQL)

W projekcie zdefiniowano 10 kluczowych problemów biznesowych, które rozwiązano za pomocą dedykowanych funkcji i procedur składowanych w języku PL/pgSQL. Takie podejście ("Thick Database") odciąża aplikację kliencką i zapewnia spójność obliczeń.

1. Algorytm sprawdzania dostępności pojazdów (`raporty.ZnajdzDostepnePojazdy`)

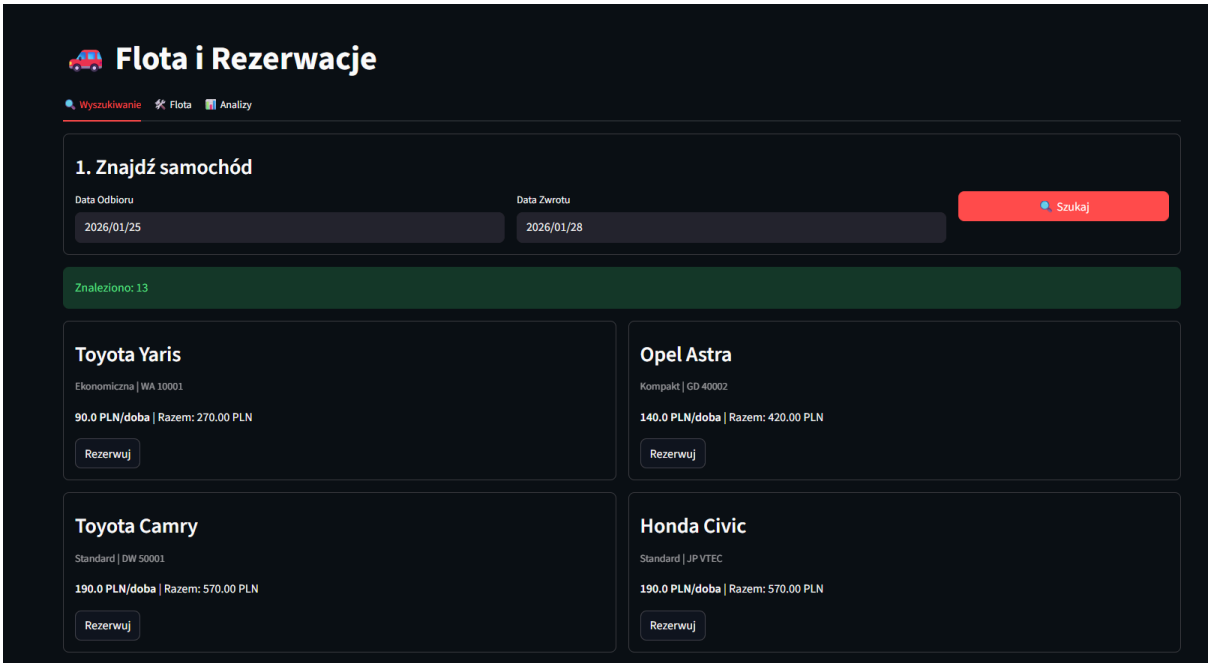
Problem: System musi wykluczyć pojazdy, które w zadanym terminie mają już potwierdzoną rezerwację, przebywają w serwisie lub mają zgłoszoną usterkę techniczną. Algorytm:

1. **Pobierz wszystkie pojazdy, które nie mają statusu "W serwisie" i flagi `Wymaga_Serwisu`.**
2. **Dla każdego pojazdu sprawdź tabelę `Rezerwacje`.**

3. **Użyj operatora nakładania się zakresów (&&) na typach `daterange`, aby wykryć kolizję daty żądanej z datami istniejących rezerwacji.**
4. **Zwróć tylko te auta, dla których nie znaleziono kolizji ("NOT EXISTS").**

SQL

```
CREATE OR REPLACE FUNCTION raporty.ZnajdzDostepnePojazdy(p_data_od DATE,
p_data_do DATE, p_klasa_id INT DEFAULT NULL)

RETURNS TABLE (ID_Pojazdu INT, Marka VARCHAR, Model VARCHAR, Nr_Rej
VARCHAR, Cena DECIMAL, Klasa VARCHAR) AS $$

BEGIN

   RETURN QUERY

   SELECT

      p.ID_Pojazdu, p.Marka, p.Model, p.Numer_Rejestracyjny,

      kp.Cena_Za_Dobe, kp.Nazwa_Klasy

   FROM model.Pojazdy p

   JOIN model.Klasy_Pojazdow kp ON p.ID_Klasy = kp.ID_Klasy

   WHERE p.Status_Dostepnosci != 'W serwisie'

    AND p.Wymaga_Serwisu = FALSE

    AND (p_klasa_id IS NULL OR p.ID_Klasy = p_klasa_id)

    AND NOT EXISTS (

       SELECT 1 FROM model.Rezerwacje r

       WHERE r.ID_Pojazdu = p.ID_Pojazdu

       AND r.Status_Rezerwacji IN ('Potwierdzona', 'W trakcie')

       AND daterange(r.Data_Odbioru, r.Data_Zwrotu, '[]') && daterange(p_data_od,
p_data_do, '[]')

    );

END;

$$ LANGUAGE plpgsql;
```



2. Raport Finansowy z analizą wzrostu (`raporty.RaportPrzychodow`)

Problem: Menadżer potrzebuje zestawienia miesięcznego z informacją o zmianie procentowej przychodów względem poprzedniego miesiąca (Month-over-Month).
Algorytm:

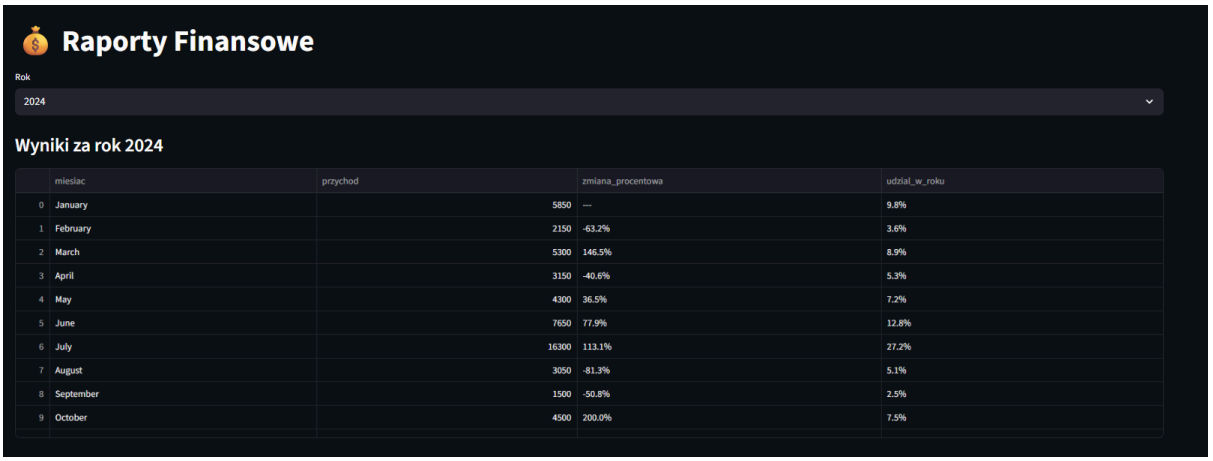
1. **Wygeneruj serię dat dla wszystkich 12 miesięcy roku (nawet tych bez przychodu).**
2. **W pętli oblicz sumę wpłat (`Status_Platnosci = 'Zrealizowana'`) dla każdego miesiąca.**
3. **Porównaj bieżący przychód z przychodem z poprzedniej iteracji pętli, aby obliczyć % wzrostu lub spadku.**
4. **Oblicz udział procentowy danego miesiąca w całkowitym rocznym przychodzie.**

SQL

```
CREATE OR REPLACE FUNCTION raporty.RaportPrzychodow(p_rok INT)

RETURNS TABLE (Miesiac TEXT, Przychod DECIMAL, Poprzedni_Miesiac DECIMAL, Zmiana_Procentowa TEXT, Udzial_W_Roku TEXT) AS $$

DECLARE

    rec RECORD;

    v_roczna_suma DECIMAL;

    v_poprzedni_przychod DECIMAL := 0;

    v_aktualny_przychod DECIMAL;
```

```
    v_zmiana DECIMAL;
BEGIN
    SELECT SUM(Kwota_Calkowita) INTO v_roczna_suma
    FROM model.Platnosci
    WHERE EXTRACT(YEAR FROM Data_Platnosci) = p_rok
      AND Status_Platnosci = 'Zrealizowana';

    v_roczna_suma := COALESCE(v_roczna_suma, 1);

    FOR rec IN
        SELECT EXTRACT(MONTH FROM m)::INT as m_num, TO_CHAR(m, 'Month') as
m_nazwa
        FROM generate_series(MAKE_DATE(p_rok, 1, 1), MAKE_DATE(p_rok, 12, 1),
INTERVAL '1 month') m
    LOOP
        SELECT COALESCE(SUM(Kwota_Calkowita), 0) INTO v_aktualny_przychod
        FROM model.Platnosci
        WHERE EXTRACT(YEAR FROM Data_Platnosci) = p_rok
          AND EXTRACT(MONTH FROM Data_Platnosci) = rec.m_num
          AND Status_Platnosci = 'Zrealizowana';

        Miesiac := rec.m_nazwa;
        Przychod := v_aktualny_przychod;
        Poprzedni_Miesiac := v_poprzedni_przychod;

        IF v_poprzedni_przychod = 0 THEN Zmiana_Procentowa := '---';
        ELSE
```

```
            v_zmiana := ((v_aktualny_przychod - v_poprzedni_przychod) /
v_poprzedni_przychod) * 100;

            Zmiana_Procentowa := ROUND(v_zmiana, 1) || '%';

        END IF;


        Udzial_W_Roku := ROUND((v_aktualny_przychod / v_roczna_suma * 100), 1) ||
'%';

        v_poprzedni_przychod := v_aktualny_przychod;

        RETURN NEXT;

    END LOOP;

END;

$$ LANGUAGE plpgsql;
```



3. Ranking Klientów VIP - Analiza RFM (`raporty.RankingKlientowVIP`)

Problem: Firma potrzebuje zidentyfikować nie tylko tych, którzy wydali dużo (Monetary), ale też tych, którzy wracają często (Frequency) i byli niedawno (Recency).
Algorytm:

1. **Oblicz globalne maksimum wydatków i liczby wizyt w całej bazie (punkt odniesienia).**
2. **Dla każdego klienta oblicz wynik (score 1-4) dla każdego z trzech wymiarów (Recency, Frequency, Monetary).**
3. **Połącz wyniki w kod RFM (np. "431").**

4. **Na podstawie kodu przypisz segment marketingowy (np. "💎 Absolutny Champion" dla najlepszych, "💤 Ryzyko odejścia" dla dawno niewidzianych).**

SQL

```
CREATE OR REPLACE FUNCTION raporty.RankingKlientowVIP(top_n INT)

RETURNS TABLE (Klient TEXT, Wydatki DECIMAL, RFM_Kod TEXT,
Segment_Marketingowy TEXT) AS $$

DECLARE

   rec RECORD;

   v_r_score INT; v_f_score INT; v_m_score INT;

   v_max_wydatki DECIMAL; v_max_wizyty INT;

BEGIN

   SELECT MAX(sum_wydatki), MAX(cnt_wizyty) INTO v_max_wydatki, v_max_wizyty

   FROM (

      SELECT SUM(p.Kwota_Calkowita) as sum_wydatki, COUNT(r.ID_Rezerwacji) as
cnt_wizyty

      FROM model.Rezerwacje r

      JOIN model.Platnosci p ON r.ID_Rezerwacji = p.ID_Rezerwacji

      WHERE p.Status_Platnosci = 'Zrealizowana' GROUP BY r.ID_Klienta

   ) sub;


   v_max_wydatki := COALESCE(v_max_wydatki, 1);

   v_max_wizyty := COALESCE(v_max_wizyty, 1);


   FOR rec IN

      SELECT k.ID_Klienta, (k.Imie || ' ' || k.Nazwisko) as nazwa,

         MAX(r.Data_Rezerwacji) as ost_data, COUNT(r.ID_Rezerwacji) as wizyty,
SUM(p.Kwota_Calkowita) as kwota

      FROM model.Klienci k
```

```
        JOIN model.Rezerwacje r ON k.ID_Klienta = r.ID_Klienta
        JOIN model.Platnosci p ON r.ID_Rezerwacji = p.ID_Rezerwacji
        WHERE p.Status_Platnosci = 'Zrealizowana'
        GROUP BY k.ID_Klienta ORDER BY kwota DESC LIMIT top_n
    LOOP
        v_m_score := CEIL((rec.kwota / v_max_wydatki) * 4);
        v_f_score := CEIL((rec.wizyty::DECIMAL / v_max_wizyty) * 4);

        IF rec.ost_data > CURRENT_DATE - INTERVAL '3 month' THEN v_r_score := 4;
        ELSIF rec.ost_data > CURRENT_DATE - INTERVAL '6 month' THEN v_r_score :=
3;
        ELSIF rec.ost_data > CURRENT_DATE - INTERVAL '12 month' THEN v_r_score :=
2;
        ELSE v_r_score := 1; END IF;

        Klient := rec.nazwa; Wydatki := rec.kwota;
        RFM_Kod := v_r_score::TEXT || v_f_score::TEXT || v_m_score::TEXT;

        IF v_m_score = 4 AND v_f_score >= 3 THEN Segment_Marketingowy := '💎
Absolutny Champion';
        ELSIF v_m_score >= 3 THEN Segment_Marketingowy := '💰 Wieloryb';
        ELSIF v_f_score >= 3 THEN Segment_Marketingowy := '🔄 Lojalny bywalec';
        ELSIF v_r_score = 1 THEN Segment_Marketingowy := '💤 Ryzyko odejścia';
        ELSE Segment_Marketingowy := '🙂 Standardowy'; END IF;
        RETURN NEXT;
    END LOOP;
END;
$$ LANGUAGE plpgsql;
```

Ranking VIP

klient	wydatki	rfm_kod	segment_marketingowy
Rafał Zakrzewski	10500	124	Wieloryb (Dużo wydaje)
Jakub Krawczyk	9200	244	Absolutny Champion
Michał Jankowski	6750	243	Wieloryb (Dużo wydaje)
Ewelina Kaczmarek	6050	243	Wieloryb (Dużo wydaje)
Karolina Jóźwiak	5750	133	Wieloryb (Dużo wydaje)

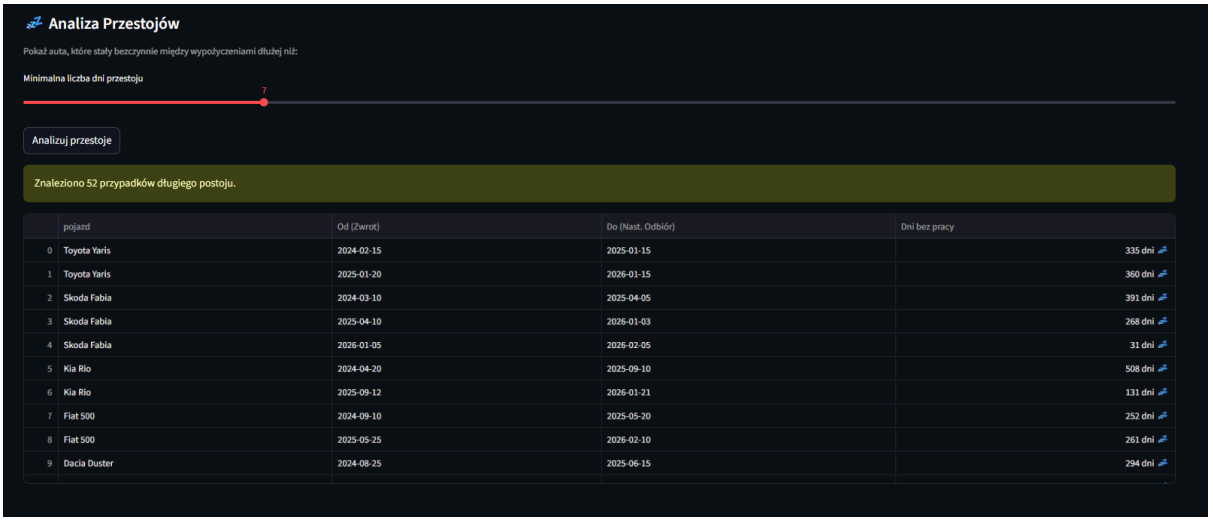
4. Analiza Przestojów Floty (`raporty.AnalizaPrzestojow`)

Problem: Identyfikacja pojazdów, które "kurzą się" na parkingu (mają długie przerwy między wynajmami). Algorytm:

1. **Dla każdego pojazdu pobierz listę rezerwacji posortowaną chronologicznie.**
2. **Użyj funkcji okna `LEAD()`, aby w bieżącym wierszu uzyskać datę następnego wypożyczenia.**
3. **Oblicz różnicę dni: `Data_Następnego_Odbioru` - `Data_Zwrotu`.**
4. **Jeśli różnica jest większa niż limit (np. 7 dni), dodaj auto do raportu przestojów.**

```
SQL

CREATE OR REPLACE FUNCTION raporty.AnalizaPrzestojow(min_dni_przerwy INT)

RETURNS TABLE (Pojazd VARCHAR, Data_Zwrotu DATE, Data_Nastepnego_Odbioru
DATE, Dni_Przestoju INT) AS $$

DECLARE

    r RECORD;

BEGIN

    FOR r IN

        SELECT p.Marka || ' ' || p.Model AS auto, rez.Data_Zwrotu AS data_konca,

            LEAD(rez.Data_Odbioru) OVER (PARTITION BY p.ID_Pojazdu ORDER BY
rez.Data_Odbioru) AS data_start_next
```

```
    FROM model.Pojazdy p JOIN model.Rezerwacje rez ON p.ID_Pojazdu =
rez.ID_Pojazdu

    WHERE rez.Status_Rezerwacji != 'Anulowana'

  LOOP

    IF r.data_start_next IS NOT NULL THEN

      Dni_Przestoju := (r.data_start_next - r.data_konca);

      IF Dni_Przestoju >= min_dni_przerwy THEN

        Pojazd := r.auto; Data_Zwrotu := r.data_konca; Data_Nastepnego_Odbioru :=
r.data_start_next;

          RETURN NEXT;

      END IF;

    END IF;

  END LOOP;

END;

$$ LANGUAGE plpgsql;
```



5. Automatyczna Prognoza Serwisowa (`raporty.PrognozaSerwisowa`)

Problem: System musi proaktywnie informować o konieczności serwisu na podstawie przebiegu LUB zgłoszeń usterek od kierowców. Algorytm:

1. **Pobierz aktualny przebieg pojazdu oraz przebieg przy ostatnim serwisie.**
2. **Sprawdź flagę `Wymaga_Serwisu` (zgłoszenia manualne).**
3. **Ustal priorytet:**
    - **WYSOKI: Jeśli zgłoszono usterkę manualnie.**
    - **ŚREDNI: Jeśli różnica przebiegów przekracza limit (np. 15 000 km).**
    - **NISKI: Jeśli zbliża się do limitu (zostało mniej niż 1000 km).**

SQL

```
CREATE OR REPLACE FUNCTION raporty.PrognozaSerwisowa(limit_km_serwisu INT DEFAULT 15000)

RETURNS TABLE (Pojazd VARCHAR, Problem VARCHAR, Priorytet TEXT, Szacowana_Data DATE) AS $$

DECLARE

    rec RECORD;

BEGIN

    FOR rec IN

        SELECT p.ID_Pojazdu, p.Marka, p.Model, p.Przebieg, p.Wymaga_Serwisu, p.Opis_Usterki,

            COALESCE(MAX(s.Przebieg_W_Chwili_Serwisu), 0) as ost_serwis_km

        FROM model.Pojazdy p LEFT JOIN model.Serwisy s ON p.ID_Pojazdu = s.ID_Pojazdu

        WHERE p.Status_Dostepnosci != 'W serwisie' GROUP BY p.ID_Pojazdu

    LOOP

        Pojazd := rec.Marka || ' ' || rec.Model; Szacowana_Data := CURRENT_DATE;

        IF rec.Wymaga_Serwisu THEN

            Problem := '⚠️ ZGŁOSZENIE: ' || COALESCE(rec.Opis_Usterki, 'Brak opisu'); Priorytet := 'WYSOKI (Awaria)'; RETURN NEXT;

        ELSIF (rec.Przebieg - rec.ost_serwis_km) >= limit_km_serwisu THEN

            Problem := '🛢️ Wymiana oleju (Przebieg)'; Priorytet := 'ŚREDNI'; RETURN NEXT;

        ELSIF (rec.Przebieg - rec.ost_serwis_km) >= (limit_km_serwisu - 1000) THEN
```

```
            Problem := '⏳ Wkrótce przegląd'; Priorytet := 'NISKI'; RETURN NEXT;
        END IF;
    END LOOP;
END;
$$ LANGUAGE plpgsql;
```

6. Generowanie Historii w JSON (`raporty.PobierzHistorieKlientaJSON`)

Problem: Optymalizacja przesyłania danych do frontendu. Zamiast wysyłać wiele wierszy, baza powinna zwrócić jeden obiekt zawierający całą historię. Algorytm:

1. Pobierz dane klienta.
2. Pobierz wszystkie rezerwacje tego klienta.
3. Użyj `json_agg` do stworzenia tablicy JSON z historią.
4. Użyj `json_build_object` do zapakowania całości w jeden dokument.

SQL

```
CREATE OR REPLACE FUNCTION raporty.PobierzHistorieKlientaJSON(p_id_klienta
INT)

RETURNS JSON AS $$

BEGIN

   RETURN (

      SELECT json_build_object(

         'klient_id', p_id_klienta,

         'imie_nazwisko', (SELECT Imie || ' ' || Nazwisko FROM model.Klienci WHERE
ID_Klienta = p_id_klienta),

         'historia', COALESCE((

            SELECT json_agg(json_build_object(

               'pojazd', p.Marka || ' ' || p.Model,

               'termin', r.Data_Odbioru || ' do ' || r.Data_Zwrotu,

               'koszt', r.Cena_Calkowita,

               'status', r.Status_Rezerwacji

            ) ORDER BY r.Data_Odbioru DESC)

            FROM model.Rezerwacje r JOIN model.Pojazdy p ON r.ID_Pojazdu =
p.ID_Pojazdu

            WHERE r.ID_Klienta = p_id_klienta

         ), '[]'::json)

      )

   );

END;

$$ LANGUAGE plpgsql;
```

7. Analiza Obłożenia Floty w Czasie (`raporty.OblozenieRoczne`)

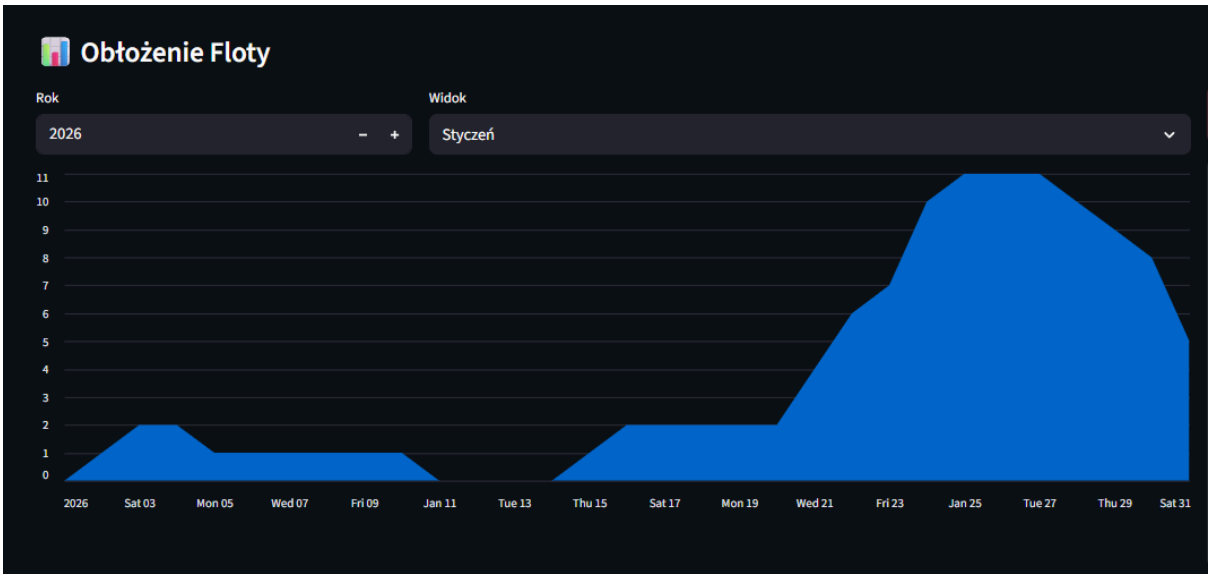
Problem: Wyświetlenie wykresu zajętości aut dzień po dniu, uwzględniając dni bez rezerwacji. Algorytm:

1. **Wygeneruj kalendarz dla całego roku (`generate_series` co 1 dzień).**
2. **Połącz (`LEFT JOIN`) kalendarz z rezerwacjami.**
3. **Warunek złączenia: Czy data kalendarzowa mieści się w przedziale `Data_Odbioru` - `Data_Zwrotu`?**
4. **Zlicz liczbę aktywnych rezerwacji dla każdego dnia.**

SQL

```
CREATE OR REPLACE FUNCTION raporty.OblozenieRoczne(p_rok INT)

RETURNS TABLE (Dzien DATE, Liczba_Aut INT) AS $$

BEGIN

    RETURN QUERY

    SELECT kalendarz.data::DATE, COUNT(r.ID_Rezerwacji)::INT

    FROM generate_series(MAKE_DATE(p_rok, 1, 1), MAKE_DATE(p_rok, 12, 31),
INTERVAL '1 day') AS kalendarz(data)

    LEFT JOIN model.Rezerwacje r ON r.Status_Rezerwacji != 'Anulowana'

    AND kalendarz.data BETWEEN r.Data_Odbioru AND r.Data_Zwrotu

    GROUP BY kalendarz.data ORDER BY kalendarz.data;
```

```
END;

$$ LANGUAGE plpgsql;
```



8. Ocena Efektywności Pracowników (`raporty.EfektywnoscPracownikow`)

Problem: Ranking sprzedawców i automatyczna ocena ich wyników. Algorytm:

1. **Oblicz średni obrót przypadający na jednego pracownika w całej firmie.**
2. **Dla każdego pracownika zsumuj wartość jego rezerwacji.**
3. **Porównaj wynik pracownika ze średnią:**
    - **Wynik > 120% średniej -> "⭐ Lider Sprzedaży".**
    - **Wynik < 50% średniej -> "⚠️ Poniżej normy".**
    - **W przeciwnym razie -> "✅ W normie".**

SQL

```
CREATE OR REPLACE FUNCTION raporty.EfektywnoscPracownikow()

RETURNS TABLE (Pracownik VARCHAR, Obrót DECIMAL, Ocena VARCHAR) AS $$

DECLARE

    v_srednia DECIMAL; rec RECORD;

BEGIN
```

```
    SELECT AVG(s.suma) INTO v_srednia FROM (SELECT SUM(Cena_Calkowita) as
suma FROM model.Rezerwacje GROUP BY ID_Pracownika) s;

    FOR rec IN

        SELECT (p.Imie || ' ' || p.Nazwisko) as osoba,
COALESCE(SUM(r.Cena_Calkowita),0) as total

        FROM model.Pracownicy p LEFT JOIN model.Rezerwacje r ON p.ID_Pracownika =
r.ID_Pracownika

        GROUP BY p.ID_Pracownika ORDER BY total DESC

    LOOP

        Pracownik := rec.osoba; Obrót := rec.total;

        IF rec.total > (v_srednia * 1.2) THEN Ocena := '⭐ Lider Sprzedaży';

        ELSIF rec.total < (v_srednia * 0.5) THEN Ocena := '⚠️ Poniżej normy';

        ELSE Ocena := '✅ W normie';

        END IF;

        RETURN NEXT;

    END LOOP;

END;

$$ LANGUAGE plpgsql;
```

9. Statusy Klientów / Retention (`raporty.StatusKlientow`)

Problem: Identyfikacja klientów, którzy przestali korzystać z usług (Churn) oraz tych, którzy aktualnie posiadają auto. Algorytm:

1. **Dla każdego klienta znajdź datę ostatniego zwrotu auta.**
2. **Oblicz liczbę dni, jakie upłynęły od tego momentu.**
3. **Przypisz status:**
    - **Dni < 0 -> "🔵 Ma aktywne wypożyczenie" (Data zwrotu jest w przyszłości).**
    - **< 30 dni -> "🟢 Aktywny".**
    - **< 365 dni -> "🟠 Uśpiony".**
    - **365 dni -> "🔴 Utracony".**

SQL

```
CREATE OR REPLACE FUNCTION raporty.StatusKlientow()

RETURNS TABLE (Klient VARCHAR, Dni_Temu INT, Status TEXT) AS $$

DECLARE

   rec RECORD; v_ostatni DATE; v_diff INT;

BEGIN

   FOR rec IN SELECT ID_Klienta, Imie, Nazwisko FROM model.Klienci LOOP

      SELECT MAX(Data_Zwrotu) INTO v_ostatni FROM model.Rezerwacje WHERE ID_Klienta = rec.ID_Klienta;

      Klient := rec.Imie || ' ' || rec.Nazwisko;

      IF v_ostatni IS NULL THEN Dni_Temu := NULL; Status := 'Nowy / Brak Historii';

      ELSE

         v_diff := (CURRENT_DATE - v_ostatni); Dni_Temu := v_diff;


         IF v_diff < 0 THEN -- Obsługa aktywnego wypożyczenia

            Status := '🔵 Ma aktywne wypożyczenie'; Dni_Temu := 0;

         ELSIF v_diff < 30 THEN Status := '🟢 Aktywny (Super)';

         ELSIF v_diff < 90 THEN Status := '🟡 Aktywny';

         ELSIF v_diff < 365 THEN Status := '🟠 Uśpiony';
```

```
            ELSE Status := '🔴 Utracony';
            END IF;
        END IF;
        RETURN NEXT;
    END LOOP;
END;
$$ LANGUAGE plpgsql;
```

Statusy

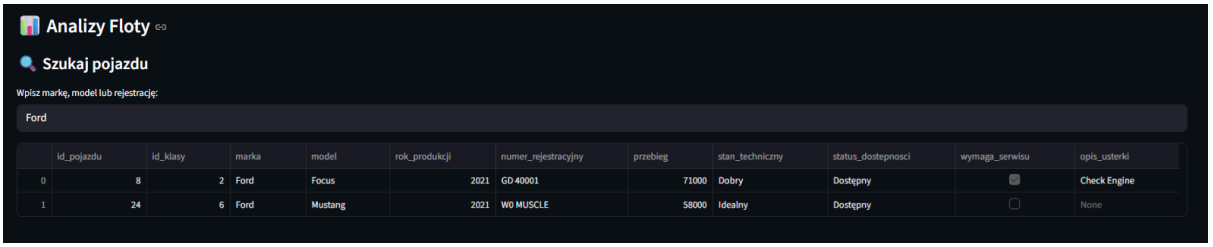
klient	dni_temu	status
Jan Kowalski	20	Aktywny (Super)
Anna Nowak	15	Aktywny (Super)
Piotr Zieliński	3	Aktywny (Super)
Kasia Wiśniewska	21	Aktywny (Super)
Marek Wójcik	502	Utracony
Tomasz Kamiński	467	Utracony
Magda Lewandowska	423	Utracony
Paweł Dąbrowski	390	Utracony
Monika Szymańska	370	Utracony
Krzysztof Woźniak	-5	Aktywny (Super)

10. Inteligentna Wyszukiwarka Pojazdów (`api.SzukajPojazdu`)

Problem: Wyszukiwanie pojazdów odporne na wielkość liter, z priorytetyzacją wyników (Ranking). Algorytm:

1. **Pobierz frazę od użytkownika i wyczyść białe znaki (`TRIM`).**
2. **Szukaj frazy w kolumnach: Marka, Model, Rejestracja używając `ILIKE` (ignoruje wielkość liter).**
3. **Sortowanie (Ranking trafności):**
    - **Priorytet 1: Dokładne dopasowanie rejestracji.**
    - **Priorytet 2: Dokładne dopasowanie marki.**
    - **Priorytet 3: Marka zaczyna się od frazy.**
    - **Priorytet 4: Reszta wyników.**

```sql
CREATE OR REPLACE FUNCTION api.SzukajPojazdu(fraza TEXT)
RETURNS SETOF model.Pojazdy AS $$
DECLARE
    r model.Pojazdy%ROWTYPE;
    v_clean TEXT; v_exact TEXT;
BEGIN
    v_exact := TRIM(fraza);
    v_clean := '%' || v_exact || '%';

    FOR r IN
        SELECT * FROM model.Pojazdy p
        WHERE p.Marka ILIKE v_clean OR p.Model ILIKE v_clean OR
p.Numer_Rejestracyjny ILIKE v_clean
        ORDER BY
            CASE
                WHEN p.Numer_Rejestracyjny ILIKE v_exact THEN 0
                WHEN p.Marka ILIKE v_exact THEN 1
                WHEN p.Marka ILIKE (v_exact || '%') THEN 2
                ELSE 3
            END ASC, p.Marka, p.Model
    LOOP
        RETURN NEXT r;
    END LOOP;
END;
$$ LANGUAGE plpgsql;
```



---

7. Instrukcja Instalacji

- **Środowisko:** Zainstaluj `uv`, Python 3.8+ oraz bazy PostgreSQL i MongoDB.
- **Zależności:** `uv pip install -r requirements.txt`
- **Konfiguracja:**
  - Ustaw hasła do PostgreSQL w pliku `src/db.py`.
  - Upewnij się, że URI do MongoDB w `src/db.py` lub `src/app.py` jest poprawne.
- **Baza PostgreSQL (SQL):**
  - Struktura: `psql -f setup.sql`
  - Dane: `psql -f dane.sql`
- **Uruchomienie Aplikacji:** `uv run streamlit run src/app.py`

7.1 Architektura Połączenia i Dostęp Zdalny

System "Rent-A-Car OS" został zaprojektowany w modelu **cloud-ready**, co oznacza pełną separację warstwy aplikacji od warstwy danych. Umożliwia to zdalny dostęp do bazy danych z dowolnego miejsca, co jest kluczowe dla rozproszonej struktury firmy (wiele oddziałów wypożyczalni).

1. Konfiguracja Połączenia (`db.py`)

Aplikacja wykorzystuje hybrydowy mechanizm łączenia z bazą PostgreSQL, zaimplementowany w module `src/db.py`. System automatycznie wykrywa środowisko uruchomieniowe:

- **Środowisko Lokalne (Development):** Jeśli aplikacja nie wykryje sekretów chmurowych, domyślnie łączy się z lokalną instancją PostgreSQL (`localhost:5432`). Jest to bezpieczne rozwiązanie do testów i rozwoju oprogramowania.
- **Środowisko Zdalne (Production):** W przypadku wdrożenia na serwer (np. Streamlit Community Cloud), aplikacja pobiera dane uwierzytelniające z bezpiecznego magazynu sekretów (`st.secrets`).

Kod odpowiedzialny za to przełączanie:

```
def get_db_config():
    try:
        # Próba pobrania bezpiecznych danych z chmury
        return st.secrets["postgres"]
    except Exception:
        # Fallback do konfiguracji lokalnej
        return {
            "dbname": "wypozyczalnia_db",
            "user": "postgres",
            "password": "admin",
            "host": "localhost",
            "port": "5432",
        }
```

2. Bezpieczeństwo Dostępu Zdalnego

Aby zapewnić bezpieczeństwo danych przesyłanych przez sieć publiczną, wdrożono następujące mechanizmy:

1. **SSL/TLS (Secure Sockets Layer):** Połączenie z bazą danych jest szyfrowane, co zapobiega podsłuchiwaniu transmisji (Man-in-the-Middle). Biblioteka `psycopg2` domyślnie wspiera tryb `sslmode='require'`.
2. **Zarządzanie Sekretami:** Hasła do bazy danych nigdy nie są przechowywane jawnym tekstem w kodzie źródłowym (hardcoded). W środowisku produkcyjnym są wstrzykiwane jako zmienne środowiskowe.
3. **Biała Lista IP (Whitelisting):** Dostęp do portu 5432 bazy danych jest ograniczony zaporą sieciową (Firewall), akceptującą połączenia wyłącznie z adresów IP serwera aplikacji.

3. Schemat Komunikacji

[Użytkownik (Przeglądarka)]

| HTTPS

v

[Serwer Aplikacji (Python Streamlit)]

| Szyfrowane połączenie TCP/IP (Port 5432)

v

[Serwer Bazy Danych (PostgreSQL)]

(Procedury Składowane / Widoki)

Takie podejście pozwala na skalowanie systemu – w razie potrzeby baza danych może zostać przeniesiona na wydajniejszy serwer w chmurze (np. AWS RDS, Azure Database for PostgreSQL lub Neon.tech) bez konieczności modyfikacji kodu aplikacji, jedynie poprzez zmianę parametrów konfiguracyjnych.

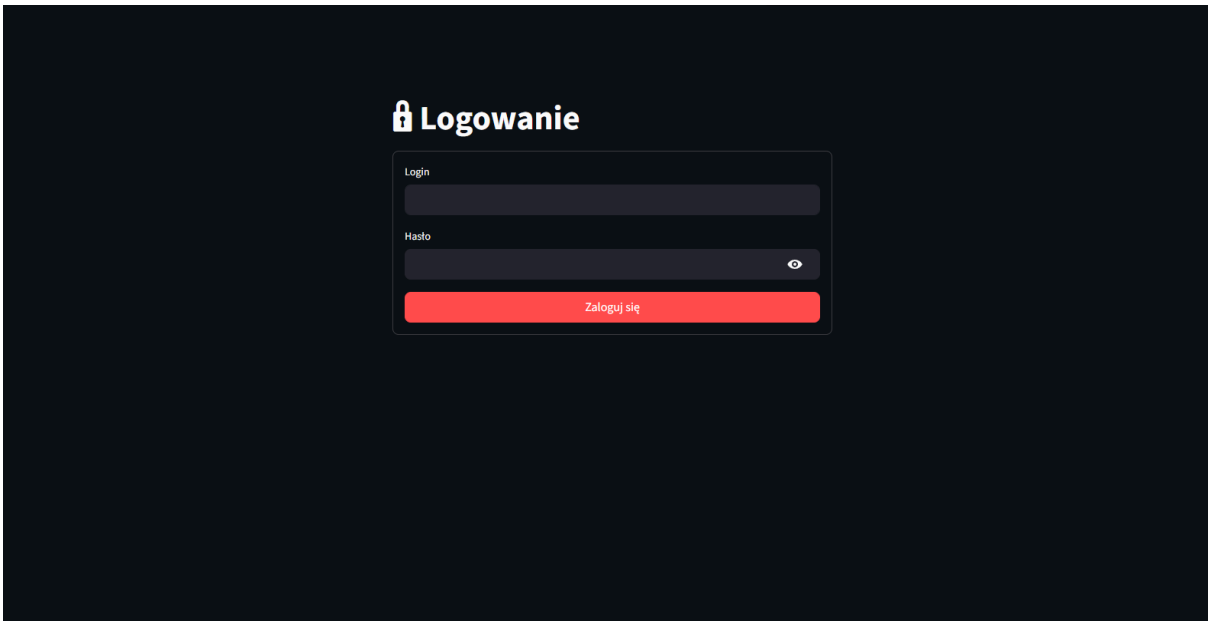
8. Przewodnik po Interfejsie Użytkownika

Aplikacja podzielona jest na panel boczny (nawigację) oraz główne okno robocze.

8.1. Ekran Logowania

Pierwszy ekran widoczny po uruchomieniu aplikacji.

- **Pole "Login":** Miejsce na wpisanie nazwy użytkownika (np. `admin`).
- **Pole "Hasło":** Miejsce na hasło (ukryte znaki).
- **Przycisk "Zaloguj się":** Weryfikuje dane w bazie `Pracownicy`. W przypadku sukcesu przenosi do Pulpitu.
- **Domyślne loginy (login | hasło):**
    - **Menadżer -** admin | admin
    - **Pracownik -** ewa | ewa



8.2. Menu Boczne (Sidebar)

Dostępne przez cały czas pracy z aplikacją.

- **Informacja o użytkowniku:** Wyświetla imię i rolę zalogowanej osoby.
- **Sekcja "Sekcje" (Radio Button):** Pozwala przełączać się między modułami:
    - 🏠 Pulpit
    - 🚗 Flota & Rezerwacje
    - 👥 Klienci
    - 💰 Finanse
    - 💼 Pracownicy (widoczne tylko dla Menadżera)
- **Przycisk "Wyloguj":** Kończy sesję i wraca do ekranu logowania.

- **Przycisk "Reset":** Czyści pamięć podręczną sesji (np. formularze rezerwacji).

8.3. Moduł: Pulpit (Dashboard)

Ekran startowy pełni rolę centrum dowodzenia (BI - Business Intelligence), prezentując kluczowe wskaźniki w czasie rzeczywistym.

- **Kluczowe KPI (Metrics):** Cztery kafelki wyświetlające na żywo:
    - Liczbę wszystkich pojazdów.

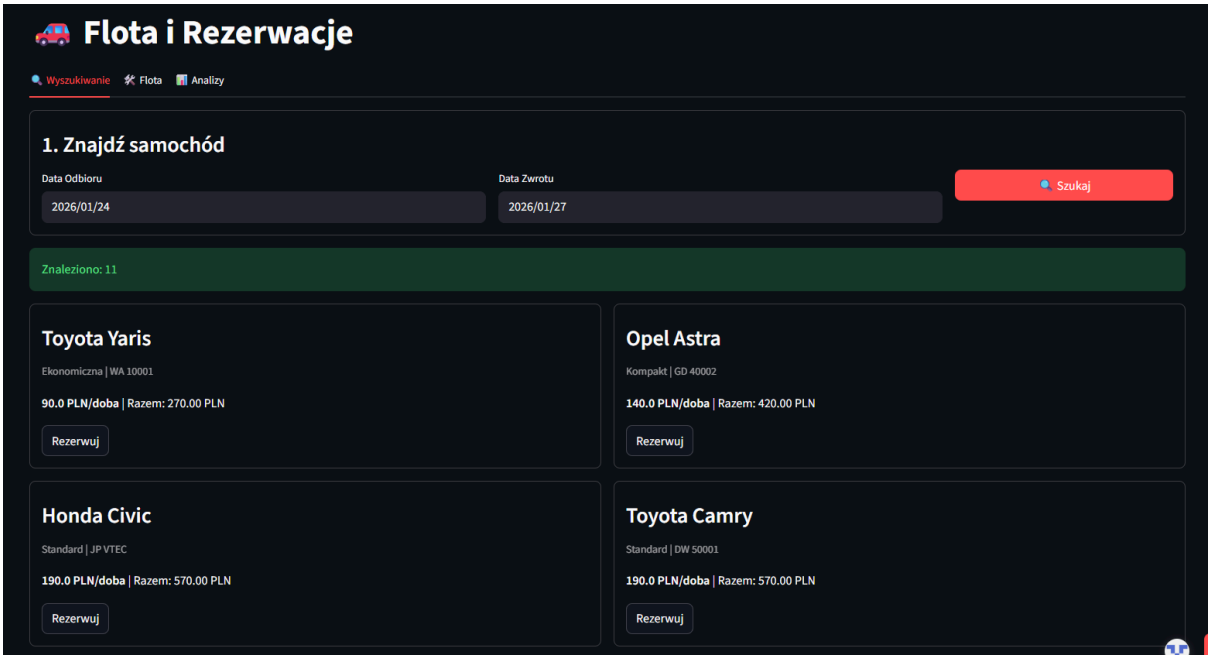
    - Wielkość bazy klientów.
    - Liczbę aktywnych rezerwacji.
    - Liczbę dostępnych "od ręki" aut (obliczaną dynamicznie).
- **Wykres Obłożenia (Area Chart):** Interaktywny wykres liniowo-obszarowy, wizualizujący zajętość floty w skali roku (dane pobierane z funkcji OblozenieRoczne). Pozwala zidentyfikować sezonowość.
- **Panel "Pilne Sprawy" (Alert System):** Czerwona sekcja ostrzegawcza po prawej stronie. Automatycznie wyświetla listę pojazdów, które:
    - Wymagają wymiany oleju (na podstawie limitu km).
    - Mają zgłoszone, nienaprawione awarie. Dane pochodzą z procedury PrognozaSerwisowa.

8.4. Moduł: 🚗 Flota & Rezerwacje

Moduł ten stanowi serce systemu operacyjnego wypożyczalni. Został podzielony na trzy funkcjonalne zakładki, obsługujące proces wynajmu, zarządzanie stanem technicznym oraz analitykę.

Zakładka 1: 🔍 Wyszukiwanie (Proces Rezerwacji)

- **Filtrowanie:** Pola "Data Odbioru" i "Data Zwrotu" pozwalają określić ramy czasowe najmu.
- **Wyszukiwarka:** Przycisk "🔍 Szukaj" uruchamia procedurę `ZnajdzDostepnePojazdy`, która weryfikuje dostępność aut, wykluczając te będące w serwisie lub zajęte w podanym terminie.
- **Wyniki:** Lista dostępnych pojazdów prezentowana jest w formie kart, zawierających markę, model, klasę, cenę za dobę oraz całkowity koszt wynajmu.

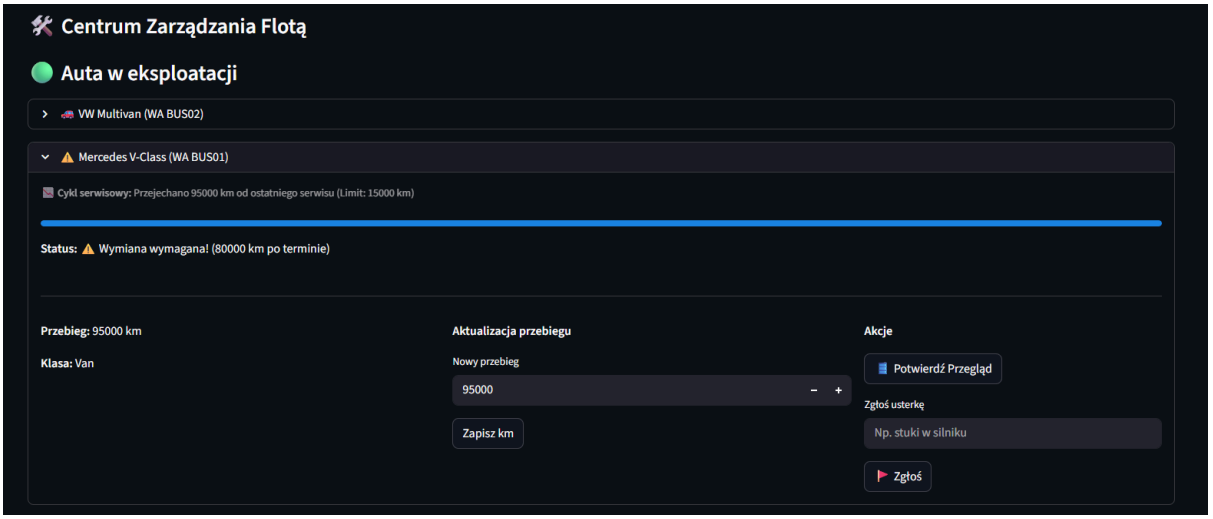


- **Formularz Rezerwacji:** Po kliknięciu "Rezerwuj" system umożliwia:
    - Wybranie istniejącego klienta z listy rozwijanej (Searchable Selectbox).
    - Rejestrację nowego klienta "w locie" (wprowadzenie danych osobowych).

- **Finalizacja:** Przycisk "Potwierdź" zapisuje rezerwację w bazie. Po sukcesie generowany jest plik PDF z potwierdzeniem (Przycisk "Pobierz PDF").

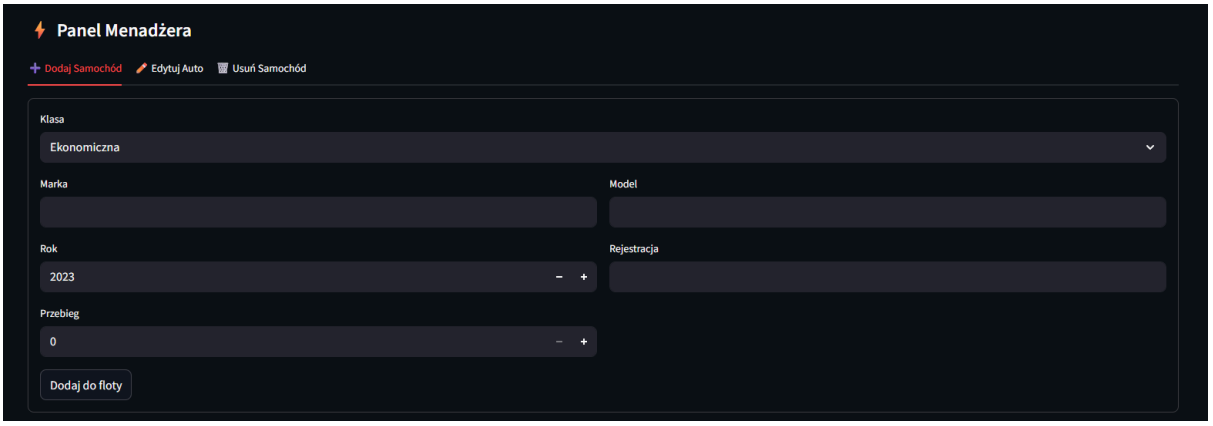
Zakładka 2: 🛠️ Flota (Centrum Zarządzania) Zamiast statycznej tabeli, zastosowano interaktywny panel zarządzania cyklem życia pojazdu.

- **Sekcja "🟢 Auta w eksploatacji":**
    - Wyświetla listę sprawnych pojazdów w formie rozwijanych paneli.
    - **Monitoring Serwisowy:** Każde auto posiada pasek postępu ("Progress Bar") pokazujący zużycie oleju/limit kilometrów do przeglądu (na podstawie limitu 15 000 km).
    - **Szybkie Akcje:** Pracownik może zaktualizować przebieg, zgłosić usterkę (przycisk "🚩 Zgłoś") lub potwierdzić wykonanie przeglądu olejowego (reset licznika).
    - **Wysyłka do serwisu:** W przypadku awarii, jednym kliknięciem można zmienić status auta na "W serwisie".



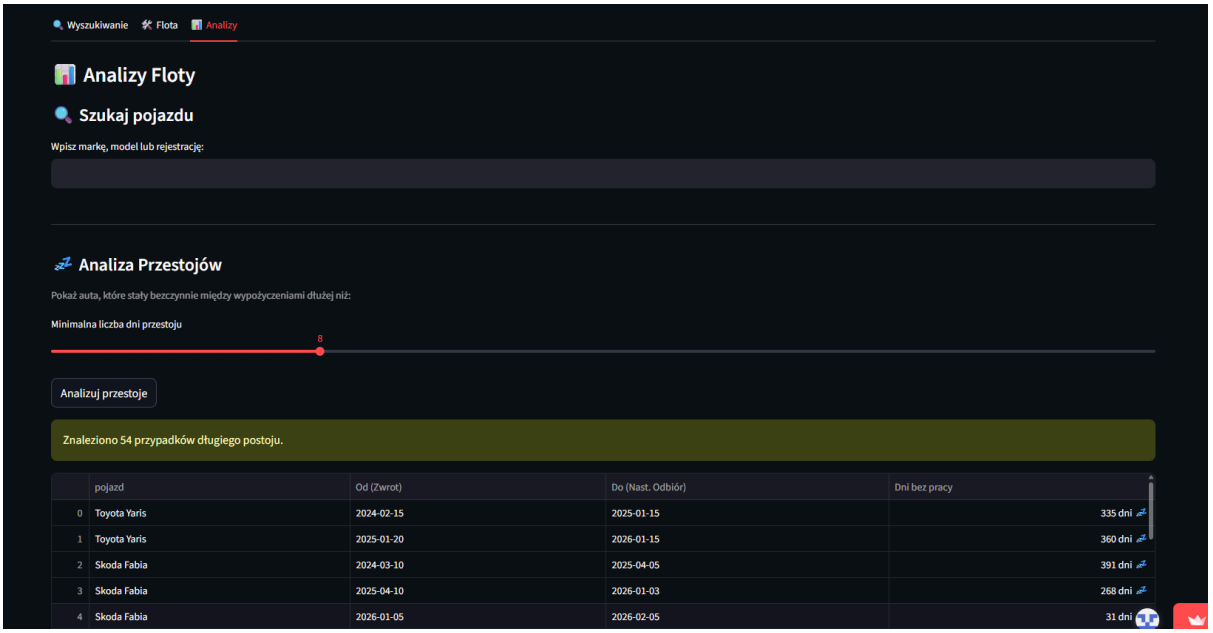
- **Sekcja "🔴 Warsztat":**
  - Lista pojazdów wyłączonych z ruchu.
  - Wyświetla powód naprawy/opis usterki.
  - Przycisk **"✅ Naprawione"**: Przywraca auto do floty dostępnej i rejestruje wpis w historii serwisowej.

- **Panel Menadżera (Widoczny tylko dla uprawnionych):**
  - **Dodaj Samochód:** Formularz wprowadzania nowego pojazdu do floty.
  - **Edytuj Auto:** Umożliwia korektę danych technicznych (np. zmiana nr rejestracyjnego, marki, modelu) istniejących pojazdów.
  - **Usuń Samochód:** Trwałe usunięcie pojazdu z bazy (zabezpieczone przed usunięciem aut z historią wypożyczeń).



Zakładka 3: 📊 Analizy

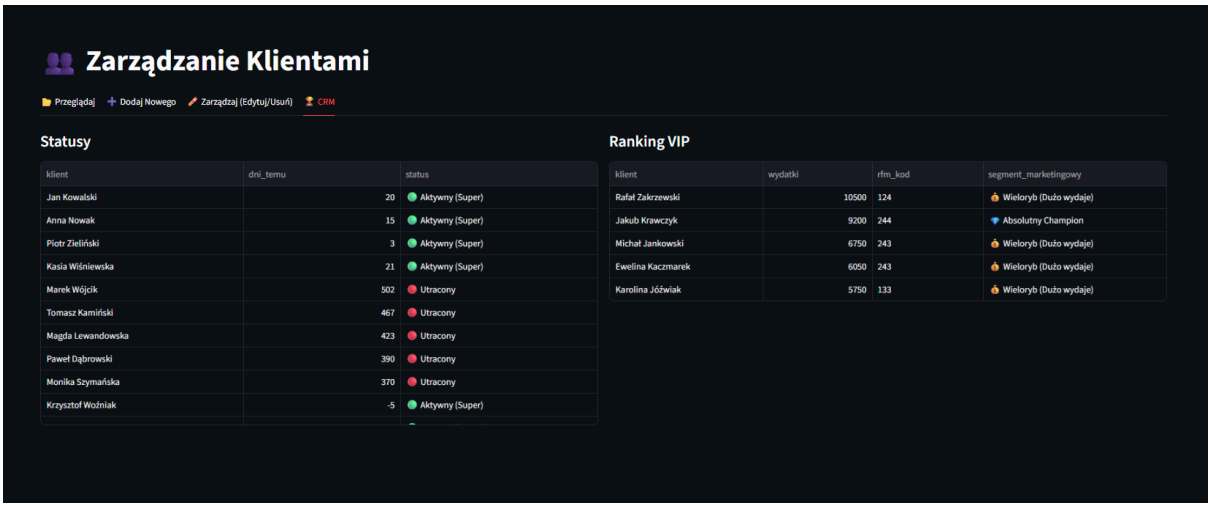
- **Inteligentna Wyszukiwarka:** Pole tekstowe obsługiwane przez funkcję SQL `SzukajPojazdu`, pozwalające znaleźć auto po fragmencie marki, modelu lub rejestracji.
- **Analiza Przestojów:** Narzędzie dla logistyki. Suwak pozwala zdefiniować "Minimalną liczbę dni przestoju", a system (funkcja `AnalizaPrzestojow`) zwraca listę aut, które nie zarabiają od dłuższego czasu.



8.5. Moduł: Klienci (CRM)

Moduł ten wykracza poza zwykłą kartotekę, oferując funkcje zarządzania relacjami z klientem (CRM). Podzielony jest na cztery zakładki:

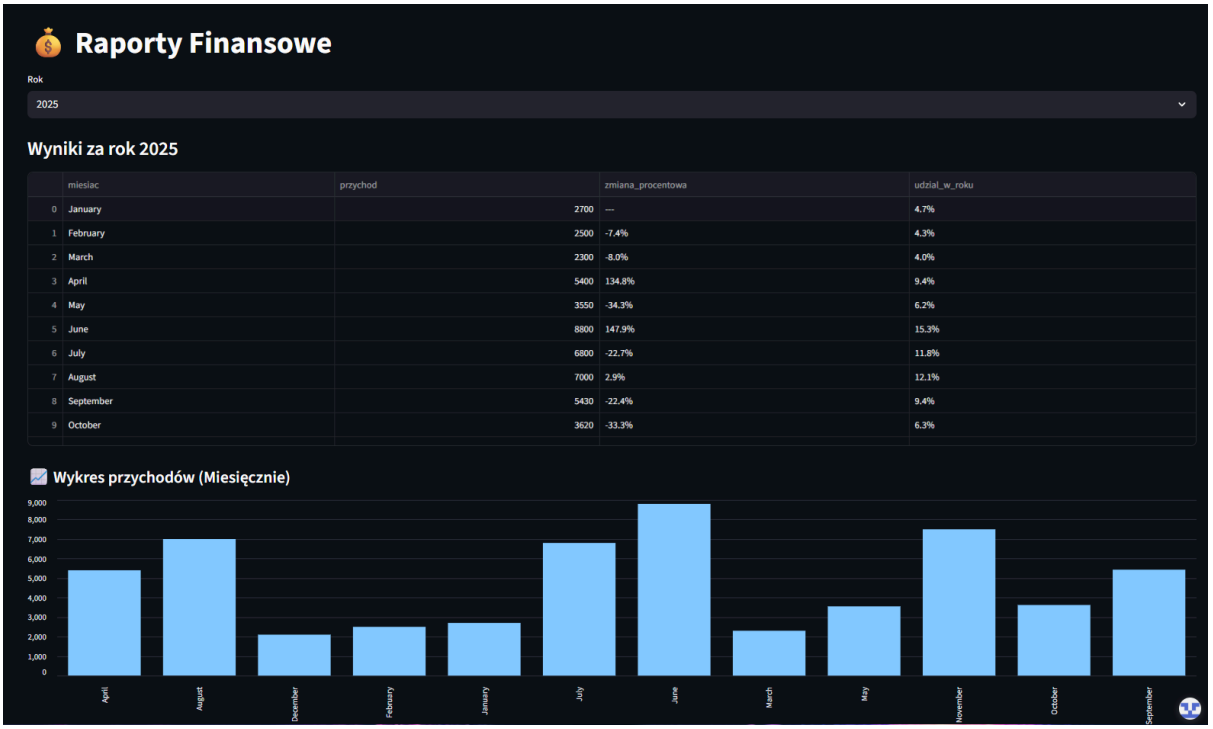
1. 📂 **Przeglądaj:** Tabela z listą klientów oraz unikalna funkcja **"Pobierz JSON Historii"**. Pozwala ona wygenerować kompletny dossier klienta (dane + historia wypożyczeń) w formacie JSON, gotowym do eksportu do systemów zewnętrznych.
2. ➕ **Dodaj Nowego:** Formularz rejestracyjny z walidacją PESEL i unikalności Prawa Jazdy (obsługiwaną przez procedurę sp_dodaj_klienta).
3. ✏️ **Zarządzaj:** Umożliwia edycję danych oraz bezpieczne usuwanie klientów (z uwzględnieniem blokady RODO dla aktywnych wypożyczeń).
4. 🏆 **CRM:** Panel analityczny wyświetlający:
    - **Statusy Klientów:** Automatyczna klasyfikacja (Aktywny / Uśpiony / Utracony) na podstawie daty ostatniego wynajmu.
    - **Ranking VIP:** Tabela wyróżniająca najlepszych klientów algorytmem RFM (Segmenty: "Wieloryb", "Lojalny", "Ryzyko odejścia").



8.6. Moduł: Finanse

Moduł raportowy służący do kontroli przepływów pieniężnych firmy.

- **Wybór Okresu:** Menu pozwalające wybrać rok obrotowy.
- **Tabela Analityczna:** Wyświetla nie tylko sumę przychodów, ale także **dynamikę zmian (Month-over-Month)** oraz procentowy udział danego miesiąca w wyniku rocznym. Obliczenia te wykonywane są bezpośrednio w bazie danych (funkcja RaportPrzychodow).
- **Wizualizacja:** Wykres słupkowy (Bar Chart) obrazujący trendy przychodowe w poszczególnych miesiącach.



8.7. Moduł: Pracownicy (Panel Administratora)

Dostępny wyłącznie dla użytkowników o roli "Menadżer". Służy do zarządzania kadrami i oceniania wyników.

- **Zakładka "Efektywność":**
  - Wyświetla automatyczny ranking sprzedawców.
  - System SQL (`EfektywnoscPracownikow`) analizuje sumę sfinalizowanych rezerwacji dla każdego pracownika i porównuje ją ze średnią firmową, przyznając ocenę (np. "⭐ Lider Sprzedaży" lub "⚠️ Poniżej normy").
  - Wykres słupkowy wizualizuje wyniki zespołu.
- **Zakładka "Konta":**
  - Pozwala dodawać nowych pracowników, nadawać im uprawnienia (Sprzedawca/Serwisant/Menadżer) oraz usuwać konta z bazy.

---

9. Podsumowanie

Projekt "Rent-A-Car OS" to kompletne, gotowe do wdrożenia rozwiązanie klasy ERP dla małych wypożyczalni.

Najważniejsze osiągnięcia:

1. **Pełna cyfryzacja:** System obsługuje cykl życia pojazdu od zakupu, przez wynajmy i serwisy, aż po sprzedaż.
2. **Bezpieczeństwo architektury:** Separacja logiki (SQL) od prezentacji (Python) oraz użycie zapytań parametryzowanych zapewnia odporność na błędy i ataki.
3. **Wsparcie decyzji:** Dashboardy menadżerskie i raporty finansowe dostarczają wiedzy niezbędnej do rozwoju biznesu.
4. **Użyteczność:** Generowanie gotowych do druku plików PDF znacznie przyspiesza pracę obsługi klienta.

---

10. Bibliografia

1. Dokumentacja PostgreSQL 15 – Procedury składowane i typy danych.
2. Dokumentacja Streamlit – Zarządzanie stanem aplikacji i komponenty UI.
3. Dokumentacja PyFPDF – Tworzenie dokumentów PDF w Pythonie.
4. Materiały wykładowe z przedmiotu Bazy Danych (Uniwersytet Rzeszowski).