AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY

WYDZIAŁ FIZYKI I INFORMATYKI STOSOWANEJ
KIERUNEK STUDIÓW: INFORMATYKA STOSOWANA

BAZY DANYCH 1

Dokumentacja

Projekt aplikacji GUI dla wypożyczalni samochodów

zrealizował
Przemysław Ryś

Kraków, 19 Stycznia 2025

1 Projekt koncepcji, założenia

1. Zdefiniowanie tematu projektu

Projekt zakłada opracowanie bazy danych w PostgreSQL, wspierającej działanie aplikacji wypożyczalni samochodowej. Aplikacja ma umożliwiać klientom rezerwację samochodów, a pracownikom sprawne zarządzanie flotą i obsługą płatności. Administrator ma z kolei prosty wgląd do danych zawartych w poszczególnych tabelach oraz widok raportów. Widok rozszerzają funkcjolanlości filtrowania oraz sortowania danych.

2. Analiza wymagań użytkownika

Baza danych obejmuje funkcje takie jak rejestrowanie nowych użytkowników, zarządzanie danymi o pojazdach oraz obsługa wynajmu oraz płatności. Możliwe jest również generowanie raportów, przy korzystaniu z odpowiednich widoków po stronie bazy danych. Tworzeni użytkownicy z poziomu aplikacji są klientami.

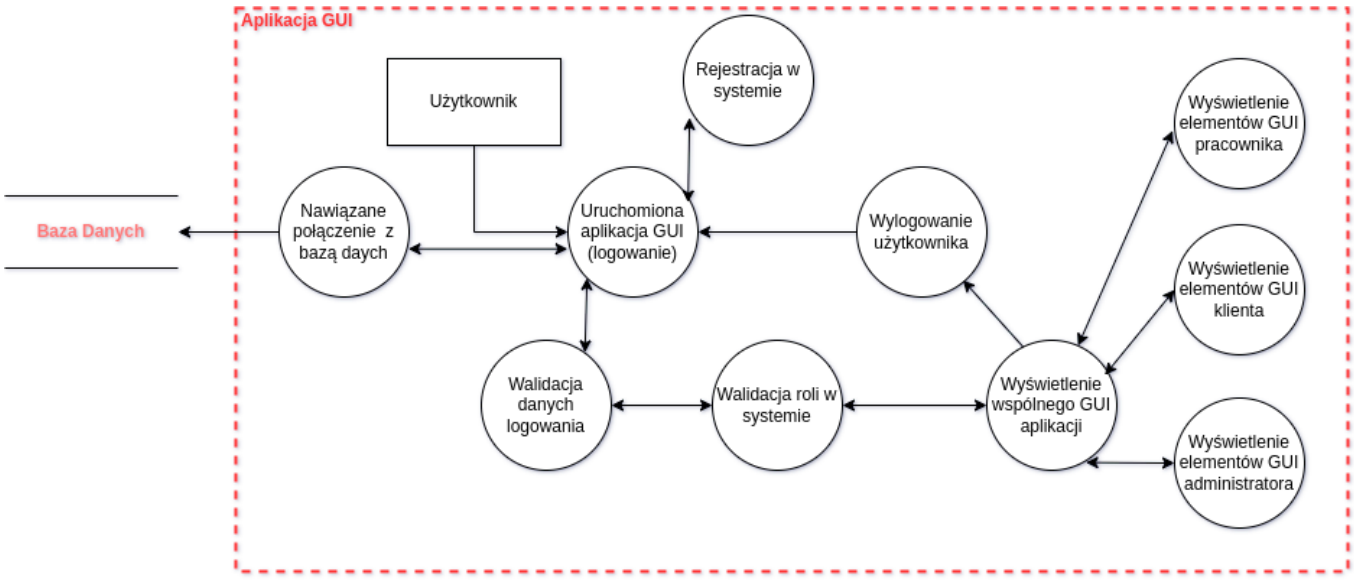
3. Zaprojektowanie funkcji

Podstawowe funkcje bazy to m.in. dodawanie i usuwanie pojazdów, filtrowanie i sortowanie danych w tabelach, a także wypożyczanie samochodów. Ważnymi elementami są także rejestrowanie płatności oraz możliwość tworzenia raportów dotyczących przychodów i historii wynajmów.

2 Projekt diagramów (konceptualny)

4. Budowa i analiza diagramu przepływu danych (DFD)

Diagram DFD (Data Flow Diagram) to graficzna reprezentacja przepływu danych w systemie, ukazująca procesy przetwarzania, magazyny danych oraz strumienie danych. Służy do modelowania procesów biznesowych i analizowania wymiany danych w systemie. Jego główne elementy to procesy, przepływy danych, magazyny danych i źródła lub odbiorcy. DFD ułatwia projektowanie i dokumentowanie systemów, zapewniając przejrzystość ich działania. Diagram DFD projektu znajduje się na obrazie 1.



Rys. 1: Diagram DFD przedstawiający przepływ danych w projekcie.

5. Zdefiniowanie encji (obiektów) oraz ich atrybutów

Uwaga, w odwoływaniu się do tabel projektu bazy należy umieszczać nazwę schematu przed ich nazwą, czyli np:

```
CREATE SCHEMA projekt_bd1;
...
SELECT * FROM projekt_bd1.cars;
```

Listing 1: Utworzony schemat w bazie danych.

Podstawowymi encjami w bazie danych są:

Encja: users

- **user_id** → Klucz główny
- email → E-mail
- password → Hasło
- role → Rola użytkownika
- status → Status konta
- created_at → Data utworzenia konta

Encja: cars

- **car_id** → Klucz główny
- make → Marka
- model → Model
- year → Rok produkcji
- license_plate → Numer rejestracyjny
- daily_rate → Stawka dzierżawy dzienna
- vin → Numer identyfikacyjny pojazdu
- status → Status pojazdu
- fuel_type → Rodzaj paliwa
- insurance_status → Status ubezpieczenia
- seat_count → Liczba miejsc
- color → Kolor
- type → Typ pojazdu

Encja: customers

- **customer_id** → Klucz główny
- first_name → Imię
- last_name → Nazwisko
- address → Adres
- phone_number → Numer telefonu
- email → E-mail
- **user_id** → Klucz obcy tabeli users
- active_rental → Flaga czy wypożyczenie aktywne

Encja: rentals

- **rental_id** → Klucz główny
- **customer_id** → Klucz obcy do tabeli customer
- **car_id** → Klucz obcy do tabeli cars
- rental_date → Data wypożyczenia
- return_date → Data zwrotu

Encja: employees

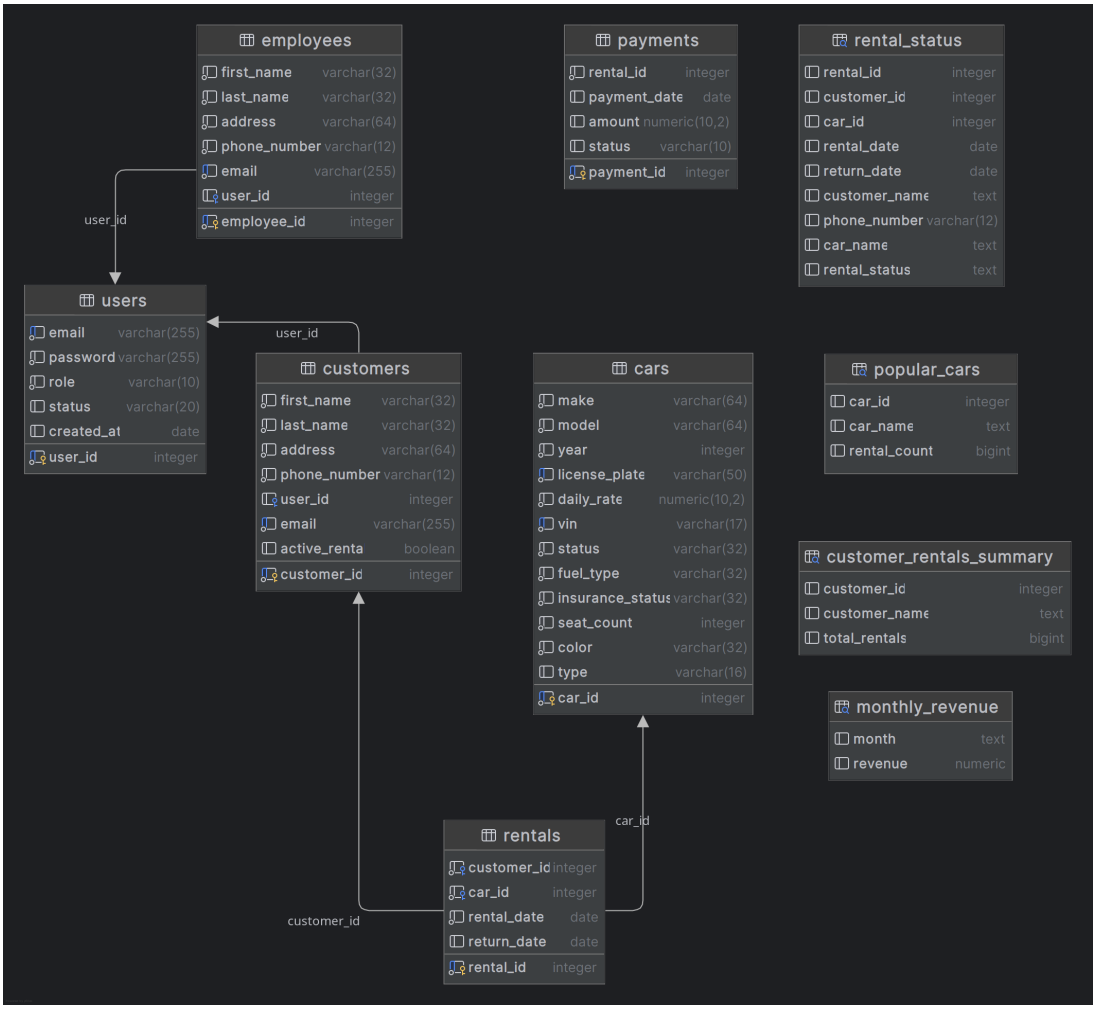
- **employee_id** → Klucz główny
- first_name → Imię
- last_name → Nazwisko
- address → Adres
- phone_number → Numer telefonu
- email → E-mail
- **user_id** → Klucz obcy tabeli users

Encja: payments

- **payment_id** → Klucz główny
- **rental_id** → Klucz obcy do tabeli rentals
- payment_date → Data płatności
- amount → Kwota
- status → Status płatności

6. Zaprojektowanie relacji pomiędzy encjami:

Diagram ERD (Entity-Relationship Diagram) to graficzna reprezentacja struktury bazy danych, która ilustruje encje oraz relacje między nimi. Służy do projektowania baz danych i umożliwia zrozumienie powiązań między różnymi elementami systemu. Kluczowymi składnikami diagramu są encje (reprezentujące obiekty lub pojęcia), atrybuty (ich cechy) oraz relacje (powiązania między encjami). Jest on niezbędny w procesie tworzenia, analizy i dokumentacji baz danych. Diagram projektu przedstawiony został na obrazie 2.



Rys. 2: Diagram ERD przedstawiający strukturę bazy danych.

3 Projekt logiczny

7. Projektowanie tabel, kluczy, indeksów

Aby aplikacja mogła działać, musimy stworzyć strukturę naszej bazy danych, odpowiedni kod w języku SQL to realizujący znajduje się poniżej, natomiast szczegółowy opis encji jest w kolejnym punkcie dotyczącym słowników danych.

```sql
CREATE SCHEMA projekt_bd1;
SET SEARCH_PATH TO projekt_bd1;

----------    TABLES    --------------------

-- Tabela reprezentująca encję użytkowników
CREATE TABLE IF NOT EXISTS projekt_bd1.users (
    user_id SERIAL PRIMARY KEY, -- Klucz główny
    email VARCHAR(255) UNIQUE NOT NULL CHECK (email LIKE '%@%'), -- Email z ograniczeniem
    password VARCHAR(255) NOT NULL, -- Hasło
    role VARCHAR(10) CHECK (role IN ('customer', 'employee', 'admin')) NOT NULL, -- Rola użytkownika
    status VARCHAR(20) DEFAULT 'active', -- Status konta
    created_at DATE DEFAULT CURRENT_DATE -- Data utworzenia konta
);

-- Tabela reprezentująca encję samochodów
CREATE TABLE IF NOT EXISTS projekt_bd1.cars (
    car_id SERIAL PRIMARY KEY, -- Klucz główny
    make VARCHAR(64) NOT NULL, -- Marka
    model VARCHAR(64) NOT NULL, -- Model
    year INT NOT NULL, -- Rok produkcji
    license_plate VARCHAR(50) UNIQUE NOT NULL,  -- Numer rejestracyjny
    daily_rate DECIMAL(10, 2) NOT NULL, -- Stawka dzierżawy dzienna
    vin VARCHAR(17) NOT NULL, -- Numer identyfikacyjny pojazdu
    status VARCHAR(32) NOT NULL CHECK (status IN ('available', 'rented')), -- Status pojazdu
    fuel_type VARCHAR(32) NOT NULL CHECK (fuel_type IN ('petrol', 'electric', 'diesel', 'hybrid')), -- Rodzaj paliwa
    insurance_status VARCHAR(32) NOT NULL DEFAULT 'uninsured' CHECK (insurance_status IN ('insured', 'uninsured', 'expired')), -- Status ubezpieczenia
    seat_count INT NOT NULL, -- Liczba miejsc
    color VARCHAR(32) NOT NULL, -- Kolor
    type VARCHAR(16) CHECK (type IN ('coupe', 'hatchback', 'cabriolet', 'estate', 'sedan', 'suv', 'van')), -- Typ pojazdu
    CONSTRAINT unique_vin UNIQUE (vin) -- Unikalny numer VIN
);

-- Tabela reprezentująca encję klientów
CREATE TABLE IF NOT EXISTS projekt_bd1.customers (
    customer_id SERIAL PRIMARY KEY, -- Klucz główny
    first_name VARCHAR(32) NOT NULL, -- Imię
    last_name VARCHAR(32) NOT NULL, -- Nazwisko
    address VARCHAR(64) NOT NULL, -- Adres
    phone_number VARCHAR(12) NOT NULL, -- Numer telefonu
    email VARCHAR(255) UNIQUE NOT NULL CHECK (email LIKE '%@%'), -- Email z ograniczeniem
    user_id INT, -- Obce odniesienie do tabeli users
    active_rental BOOLEAN DEFAULT FALSE, -- Flaga określająca aktywne wynajęcie
    CONSTRAINT fk_user_id FOREIGN KEY (user_id) REFERENCES users(user_id) -- Klucz obcy
);

-- Tabela reprezentująca relację (między klientami a samochodami) wypożyczenia
CREATE TABLE IF NOT EXISTS projekt_bd1.rentals (
```

```
    rental_id SERIAL PRIMARY KEY, -- Klucz główny
    customer_id INT NOT NULL REFERENCES projekt_bd1.customers(customer_id), -- klucz obcy z
    tabeli customer
    car_id INT NOT NULL REFERENCES projekt_bd1.cars(car_id), -- klucz obcy z tabeli cars
    rental_date DATE NOT NULL DEFAULT CURRENT_DATE, -- Data wypożyczenia
    return_date DATE DEFAULT NULL -- Data zwrotu
);

-- Tabela reprezentująca encję pracowników
CREATE TABLE IF NOT EXISTS projekt_bd1.employees (
    employee_id SERIAL PRIMARY KEY, -- Klucz główny
    first_name VARCHAR(32) NOT NULL, -- Imię
    last_name VARCHAR(32) NOT NULL, -- Nazwisko
    address VARCHAR(64) NOT NULL, -- Adres
    phone_number VARCHAR(12) NOT NULL, -- Numer telefonu
    email VARCHAR(255) UNIQUE NOT NULL CHECK (email LIKE '%@%'), -- Email z ograniczeniem
    user_id INT, -- Klucz obcy z tabeli users
);
ADD CONSTRAINT fk_user_id FOREIGN KEY (user_id) REFERENCES projekt_bd1.users(user_id);

-- Tabela reprezentująca encję płatności
CREATE TABLE IF NOT EXISTS projekt_bd1.payments (
    payment_id SERIAL PRIMARY KEY, -- Klucz główny
    rental_id INT NOT NULL, -- Klucz obcy z tabeli rentals
    payment_date DATE, -- Data płatności
    amount DECIMAL(10, 2), -- Kwota
    status VARCHAR(10) CHECK (status IN ('paid', 'unpaid')), -- Status płatności
    CONSTRAINT fk_rental FOREIGN KEY (rental_id) REFERENCES rentals (rental_id)
        ON DELETE CASCADE
);

----------     WIDOKI     --------------------

-- Widok, który już podałem, łączy dane o wypożyczeniach, klientach i samochodach
CREATE OR REPLACE VIEW projekt_bd1.rental_status AS
SELECT
    r.rental_id,
    r.customer_id,
    r.car_id,
    r.rental_date,
    r.return_date,
    c.first_name || ' ' || c.last_name AS customer_name,
    c.phone_number AS phone_number,
    ca.make || ' ' || ca.model AS car_name,
    -- Jeśli return_date jest NULL, status to "Wypożyczony", w przeciwnym razie "Zwrócony"
    CASE
        WHEN r.return_date IS NULL THEN 'Wypożyczony'
        ELSE 'Zwrócony'
        END AS rental_status
FROM
    projekt_bd1.rentals r
        JOIN
    projekt_bd1.customers c ON r.customer_id = c.customer_id
        JOIN
    projekt_bd1.cars ca ON r.car_id = ca.car_id;
```

```sql
-- Widok, który podsumowuje wypożyczenia klientów
CREATE OR REPLACE VIEW projekt_bd1.customer_rentals_summary AS
SELECT
    c.customer_id,
    c.first_name || ' ' || c.last_name AS customer_name,
    COUNT(r.rental_id) AS total_rentals
FROM projekt_bd1.rentals r
JOIN projekt_bd1.customers c ON r.customer_id = c.customer_id
GROUP BY c.customer_id, c.first_name, c.last_name
ORDER BY total_rentals DESC;

--  Widok, który podsumowuje popularność samochodów
CREATE OR REPLACE VIEW projekt_bd1.popular_cars AS
SELECT
    c.car_id,
    c.make || ' ' || c.model AS car_name,
    COUNT(r.rental_id) AS rental_count
FROM projekt_bd1.rentals r
         JOIN projekt_bd1.cars c ON r.car_id = c.car_id
GROUP BY c.car_id, c.make, c.model
ORDER BY rental_count DESC;

-- Widok, który podsumowuje przychody miesięczne
CREATE OR REPLACE VIEW projekt_bd1.monthly_revenue AS
SELECT
    TO_CHAR(r.rental_date, 'YYYY-MM') AS month,  -- Konwertujemy datę na tekstowy format YYYY-MM
    SUM((r.return_date - r.rental_date) * c.daily_rate) AS revenue
FROM projekt_bd1.rentals r
         JOIN projekt_bd1.cars c ON r.car_id = c.car_id
GROUP BY TO_CHAR(r.rental_date, 'YYYY-MM')
ORDER BY month;

-- SELECT * FROM projekt_bd1.customer_rentals_summary;
-- SELECT * FROM projekt_bd1.popular_cars;
-- SELECT * FROM projekt_bd1.monthly_revenue;

----------    TRIGGERY    --------------------

-- Wyzwalacz sprawdzający daty wypożyczenia i zwrotu:
-- Zapewnia, że data zwrotu nie może być wcześniejsza niż data wypożyczenia.
CREATE OR REPLACE FUNCTION validate_rental_dates()
    RETURNS TRIGGER AS $$
BEGIN
    IF NEW.return_date < NEW.rental_date THEN
        RAISE EXCEPTION 'Data zwrotu (%), nie może być wcześniejsza niż data wypożyczenia (%)',
            NEW.return_date, NEW.rental_date;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trg_validate_rental_dates
    BEFORE INSERT OR UPDATE ON projekt_bd1.rentals
    FOR EACH ROW
EXECUTE FUNCTION validate_rental_dates();
```

```sql
-- Funkcja triggera
CREATE OR REPLACE FUNCTION create_payment_trigger()
    RETURNS TRIGGER AS $$
DECLARE
    rental_duration INT;
    daily_rate DECIMAL(10, 2);
BEGIN
    IF NEW.return_date IS NOT NULL AND OLD.return_date IS NULL THEN
        -- Obliczanie różnicy dni + 1
        rental_duration := (NEW.return_date - NEW.rental_date) + 1;

        -- Pobranie daily_rate z tabeli cars
        SELECT c.daily_rate
        INTO daily_rate
        FROM projekt_bd1.cars c
        WHERE c.car_id = NEW.car_id;

        -- Tworzenie rekordu w tabeli payments bez  payment_date
        INSERT INTO projekt_bd1.payments (rental_id, amount, status)
        VALUES (NEW.rental_id, rental_duration * daily_rate, 'unpaid');
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Dodanie triggera do tabeli rentals (pierwszy trigger)
CREATE TRIGGER after_return_date_set
    AFTER UPDATE OF return_date
    ON projekt_bd1.rentals
    FOR EACH ROW
EXECUTE FUNCTION create_payment_trigger();

-- Funkcja triggera dla aktualizacji payment_date
CREATE OR REPLACE FUNCTION update_payment_date_trigger()
    RETURNS TRIGGER AS $$
BEGIN
    -- Sprawdzenie, czy status zmienia się na 'paid'
    IF NEW.status = 'paid' AND OLD.status = 'unpaid' THEN
        -- Aktualizacja payment_date na aktualną datę
        UPDATE projekt_bd1.payments
        SET payment_date = NOW()
        WHERE rental_id = NEW.rental_id AND status = 'unpaid';  -- Tylko dla rekordów o
    statusie 'unpaid'
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Dodanie triggera do tabeli payments (drugi trigger)
CREATE TRIGGER after_status_change_to_paid
    AFTER UPDATE OF status
    ON projekt_bd1.payments
    FOR EACH ROW
EXECUTE FUNCTION update_payment_date_trigger();
```

Listing 2: Zawartości pliku inicjującego strukturę tabel, widoków oraz triggerów.

```sql

-- Wstawianie danych do tabeli users
INSERT INTO projekt_bd1.users (email, password, role) VALUES
    ('prys@poczta', 'admin', 'admin'),
        ('mpawlik@poczta', 'haslo', 'employee'),
    ('jkowalski@poczta', 'haslo', 'customer');

-- Wstawianie danych do tabeli customers
INSERT INTO projekt_bd1.customers (first_name, last_name, address, phone_number, email)
    VALUES ('Andrzej', 'Nowak','Krakow', '+48111222333', 'pnowak@poczta');

UPDATE projekt_bd1.customers
    SET user_id = users.user_id
    FROM users
    WHERE customers.email = users.email;

-- Wstawianie danych do tabeli employees
INSERT INTO projekt_bd1.employees (first_name, last_name, address, phone_number, email)
    VALUES ('Bogdan', 'Grabowski','Krakow', '+48666777888', 'bgrabowski@poczta');

INSERT INTO projekt_bd1.employees (first_name, last_name, address, phone_number, email)
    VALUES ('Przemysław', 'Ryś','Krakow', '+48111000111', 'prys@poczta');

UPDATE projekt_bd1.employees
    SET user_id = users.user_id
    FROM users
    WHERE employees.email = users.email;

-- Wstawianie pełnych danych dla wszystkich samochodów
INSERT INTO projekt_bd1.cars (make, model, year, license_plate, daily_rate, vin, status,
    fuel_type, insurance_status, seat_count, color, type)
VALUES
    ('Audi', 'A4', 2015, 'KR12345', 20.00, '1HGCM82633A123456', 'available', 'petrol', '
    insured', 5, 'black', 'sedan'),
    ('Porsche', 'Panamera Turbo E-hybryda', 2024, 'KR23455', 50.00, '2HGCM82633A123456', '
    available', 'hybrid', 'insured', 5, 'orange', 'sedan'),
    ('Ford', 'F-150', 2021, 'KR33455', 35.00, '2HGCM82633A323456', 'available', 'petrol', '
    insured', 5, 'red', 'van'),
    ('Hyundai', 'IONIQ 5', 2022, 'KR33555', 32.00, '2HYCM82633A323446', 'available', 'electric
    ', 'insured', 5, 'beige', 'suv'),
    ('Mercedes-Benz', 'C63s AMG E', 2023, 'KR44444', 60.00, '2HYCG72633A363444', 'available',
    'petrol', 'insured', 5, 'gray', 'sedan'),
    ('Subaru', 'WRX', 2022, 'KR43355', 40.00, '2HGDM82633A321455', 'available', 'petrol', '
    insured', 5, 'orange', 'sedan'),
    ('Volkswagen', 'ID.7', 2024, 'KR23455', 35.00, '2BGNM82677A333456', 'available', 'electric
    ', 'insured', 5, 'gray', 'sedan'),
    ('Volkswagen', 'ID.7', 2024, 'KR44455', 35.00, '2BGNM82677A333456', 'available', 'electric
    ', 'insured', 5, 'gray', 'sedan'),
    ('BMW', 'M3 E46 Coupe', 2002, 'KR98275', 50.00, '2BGNM03677K733416', 'available', 'petrol'
    , 'insured', 4, 'gray', 'coupe'),
    ('Ford', 'Focus ST', 2012, 'KR22275', 35.00, '2BGNM036A1K73G416', 'available', 'diesel', '
    insured', 5, 'orange', 'hatchback'),
    ('Jaguar', 'F-Type Convertible', 2021, 'KR21574', 75.00, '2AGGM026A1F73G416', 'available',
     'petrol', 'insured', 2, 'white', 'cabriolet'),
    ('Mercedes-Benz', 'C-Class Estate', 2022, 'KR21363', 50.00, '2HCDM026A1F74G416', '
    available', 'diesel', 'insured', 5, 'blue', 'estate'),
    ('Tesla', 'Model Y', 2021, 'KR27763', 75.00, '2HCYM025A1F74R4T6', 'available', 'electric',
     'insured', 5, 'gray', 'suv'),
    ('Volkswagen', 'Transporter T6', 2016, 'KR42363', 50.00, '2HCDM026F1G44G416', 'available',
```

```
    'diesel', 'insured', 9, 'gray', 'van'),
46  ('Mercedes-Benz', 'Vito E-Cell ', 2011, 'KR24677', 60.00, '2HCDM026T3W34G496', 'available'
    , 'electric', 'insured', 2, 'white', 'van'),
47  ('Kia', 'Sorento', 2021, 'KR82679', 80.00, '2HCSM02ST3I34G496', 'available', 'petrol', '
    insured', 7, 'blue', 'suv');
```

Listing 3: Kod SQL dodający dane do bazy.

8.Słowniki danych

Atrybut	Typ danych	Ograniczenia	Dziedzina / Przykład
user_id	SERIAL	PRIMARY KEY	Unikalny identyfikator użytkownika
email	VARCHAR(255)	UNIQUE, NOT NULL, CHECK (email LIKE '%@%')	Adres e-mail (np. user@example.com)
password	VARCHAR(255)	NOT NULL	Hasło użytkownika (np. '12345')
role	VARCHAR(10)	CHECK (role IN ('customer', 'employee', 'admin')), NOT NULL	Rola użytkownika (customer, employee, admin)
status	VARCHAR(20)	DEFAULT 'active'	Status użytkownika (np. 'active')
created_at	DATE	DEFAULT CURRENT_DATE	Data utworzenia konta (np. '2025-01-19')

Tab. 1: Słownik danych dla encji `users`

Atrybut	Typ danych	Ograniczenia	Dziedzina / Przykład
car_id	SERIAL	PRIMARY KEY	Unikalny identyfikator pojazdu (np. 1)
make	VARCHAR(64)	NOT NULL	Marka pojazdu (np. 'Audi')
model	VARCHAR(64)	NOT NULL	Model pojazdu (np. 'A4')
year	INT	NOT NULL	Rok produkcji (np. 2015)
license_plate	VARCHAR(50)	UNIQUE, NOT NULL	Numer rejestracyjny (np. 'KR12345')
daily_rate	DECIMAL(10, 2)	NOT NULL	Stawka dzierżawy dzienna (np. 20.00)
vin	VARCHAR(17)	NOT NULL	Numer VIN (np. '1HGCM82633A123456')
status	VARCHAR(32)	NOT NULL, CHECK (status IN ('available', 'rented'))	Status pojazdu (np. 'available')
fuel_type	VARCHAR(32)	NOT NULL, CHECK (fuel_type IN ('petrol', 'electric', 'diesel', 'hybrid'))	Rodzaj paliwa (np. 'petrol')
insurance_status	VARCHAR(32)	NOT NULL DEFAULT 'uninsured', CHECK (insurance_status IN ('insured', 'uninsured', 'expired'))	Status ubezpieczenia (np. 'insured')
seat_count	INT	NOT NULL	Liczba miejsc (np. 5)
color	VARCHAR(32)	NOT NULL	Kolor pojazdu (np. 'black')
type	VARCHAR(16)	CHECK (type IN ('coupe', 'hatchback', 'cabriolet', 'estate', 'sedan', 'suv', 'van'))	Typ pojazdu (np. 'sedan')

Tab. 2: Słownik danych dla encji `cars`

Atrybut	Typ danych	Ograniczenia	Dziedzina / Przykład
customer_id	SERIAL	PRIMARY KEY	Unikalny identyfikator klienta (np. 1)
first_name	VARCHAR(32)	NOT NULL	Imię klienta (np. 'Jan')
last_name	VARCHAR(32)	NOT NULL	Nazwisko klienta (np. 'Kowalski')
address	VARCHAR(64)	NOT NULL	Adres klienta (np. 'Warszawa, ul. XYZ')
phone_number	VARCHAR(12)	NOT NULL	Numer telefonu (np. '123456789')
email	VARCHAR(255)	UNIQUE, NOT NULL, CHECK (email LIKE '%@%')	Adres e-mail (np. 'user@example.com')
user_id	INT	FOREIGN KEY (user_id) REFERENCES users(user_id)	Identyfikator użytkownika powiązanego z klientem (np. 2)
active_rental	BOOLEAN	DEFAULT FALSE	Flaga aktywnego wynajmu (np. 'FALSE')

Tab. 3: Słownik danych dla encji `customers`

Atrybut	Typ danych	Ograniczenia	Dziedzina / Przykład
rental_id	SERIAL	PRIMARY KEY	Unikalny identyfikator wypożyczenia (np. 1)
customer_id	INT	NOT NULL, FOREIGN KEY (customer_id) REFERENCES customers(customer_id)	Identyfikator klienta (np. 2)
car_id	INT	NOT NULL, FOREIGN KEY (car_id) REFERENCES cars(car_id)	Identyfikator pojazdu (np. 5)
rental_date	DATE	NOT NULL, DEFAULT CURRENT_DATE	Data wypożyczenia (np. '2025-01-19')
return_date	DATE	DEFAULT NULL	Data zwrotu pojazdu (np. '2025-01-25') lub NULL

Tab. 4: Słownik danych dla encji `rentals`

9. Analiza zależności funkcyjnych i normalizacja tabel (dekompozycja do 3NF ewentualnie BCNF)

Normalizacja do 3NF:

Tabele w bazie danych są już w 3NF. Wszystkie atrybuty zależą tylko od klucza głównego lub kluczy obcych, nie ma zależności częściowych ani przejściowych.

Atrybut	Typ danych	Ograniczenia	Dziedzina / Przykład
employee_id	SERIAL	PRIMARY KEY	Unikalny identyfikator pracownika (np. 1)
first_name	VARCHAR(32)	NOT NULL	Imię pracownika (np. 'Jan')
last_name	VARCHAR(32)	NOT NULL	Nazwisko pracownika (np. 'Kowalski')
address	VARCHAR(64)	NOT NULL	Adres zamieszkania (np. 'Warszawa, ul. 3 Maja 12')
phone_number	VARCHAR(12)	NOT NULL	Numer telefonu (np. '123456789')
email	VARCHAR(255)	UNIQUE, NOT NULL, CHECK (email LIKE '%@%')	Adres email (np. 'jan.kowalski@example.com')
user_id	INT	FOREIGN KEY (user_id) REFERENCES users(user_id)	Identyfikator użytkownika z tabeli users (np. 2)

Tab. 5: Słownik danych dla encji `employees`

Atrybut	Typ danych	Ograniczenia	Dziedzina / Przykład
payment_id	SERIAL	PRIMARY KEY	Unikalny identyfikator płatności (np. 1)
rental_id	INT	FOREIGN KEY (rental_id) REFERENCES rentals(rental_id) ON DELETE CASCADE	Identyfikator wypożyczenia, klucz obcy z tabeli rentals (np. 2)
payment_date	DATE	NULLABLE	Data płatności (np. '2025-01-19')
amount	DECIMAL(10, 2)	NULLABLE	Kwota płatności (np. 100.00)
status	VARCHAR(10)	CHECK (status IN ('paid', 'unpaid'))	Status płatności ('paid', 'unpaid')

Tab. 6: Słownik danych dla encji `payments`

Dekonstrukcja tabel:

Nie ma potrzeby dalszej dekompozycji, ponieważ już spełnia zasady.

Normalizacja do 3NF:

Tabele w bazie danych są już w 3NF. Wszystkie atrybuty zależą tylko od klucza głównego lub kluczy obcych, nie ma zależności częściowych ani przejściowych.

10. Denormalizacja struktury tabel

Nie jest konieczna, ponieważ struktura bazy danych jest prosta i nie korzysta z wielu instrukcji typu join.

11. Zaprojektowanie operacji na danych

Odpowiednie operacje aplikacji są wykonywane w kodzie z wykorzystaniem funkcji cursor z modułu **psycopg2**, główne operacje pobierania danych zawarte są w pliku **"database/models.py"**.

4 Projekt funkcjonalny

12. Interfejsy do prezentacji, edycji i obsługi danych

Interfejs użytkownika

Po uruchomieniu programu wyświetla się okno logowania z możliwością rejestracji nowego klienta. Interfejs aplikacji GUI składa się z zakładek, które prezentują różne widoki użytkownikowi w zależności od jego roli. Dane pochodzące z bazy danych prezentowane są w postaci tabel w odpowiednich zakładach. Administrator i pracownik mają wspólne zakładki samochodów oraz wypożyczeń, niemniej pracownik ma dodatkowe funkcjonalności których nie posiada administrator to znaczy dodawanie/usuwanie samochodu, zmienianie jego statusu ubezpieczenia oraz obsługa płatności, jeśli samochód został zwrócony. Administrator ma natomiast zakładkę dla raportów i wszystkih danych w bazie za pomocą odpowiednich zakładek. Klient z kolei posiada widok gdzie są obrazy odpowiednich samochodów, ma również widok tabeli historii wypożyczeń. Jest również widok przedstawiający aktualne wypożyczenie z kluczowymi informacjami oraz możliwością zwrotu.

Mechanizmy edycji danych

Pracownik może zmieniać status płatności, ubezpieczenia oraz dodawać bądź usuwać samochody z floty. Rejestracja nowego użytkownika wprowadza do systemu jego dane. Wypożyczanie i zwracanie również tworzy dane poprzez odpowiednie triggery.

13. Wizualizacja danych

Raporty są w postaci odpowiednich tabel reprezentujących widoki raportów po stronie bazy danych.

14. Zdefiniowanie panelu sterowania aplikacji

Panel sterowania poza zestawem zakładek, który jest specyficzny dla określonej roli użytkownika składa się z przycisku wyloguj który zamyka otwarte okno dla roli i otwiera okno logowania. Jest również przycisk zmiany kolorów aplikacji (biały/ciemno-szary) oraz kółko powiadamiające o stanie połączenia: zielone - połączenie aktywne, czerwone - połączenie przerwane.

Możliwość zmiany bazy danych (PostgreSQL), która łączy się z GUI odbywa się w pliku **config/database_config.json**, tam można zmienić kluczowe parametry połączenia.

15. Makropolecenia

Makropolecenia aplikacji są zapytaniami SQL zapisanymi w kodzie języka python, tak jak i cała aplikacja (biblioteka **pyqt6**). Kwerendy języka SQL wykonywane są natomiast poprzez bibliotekę **psycopg2**. Kwerendy w tej bibliotece są wykonywane poprzez obiekt cursor. Poniższy kod zwraca informacje o użytkowniku z bazy danych.

```
_connection = psycopg2.connect(**DATABASE_CONFIG, cursor_factory=
   RealDictCursor)
cursor = _connection.cursor()
query = """
    SELECT user_id, email, role, password
    FROM projekt_bd1.users
    WHERE email = %s AND password = %s
"""
cursor.execute(query, (email, password))
user = cursor.fetchone()
```

Listing 4: Przykład użycia cursora.

W taki sam sposób zaimplementowane zostały inne funkcjonalności takie jak pobieranie, edycja, dodawanie lub usuwanie danych z bazy. Odpowiednie kwerendy są zaimplementowane w funkcjach obsługujących przyciski i formularze.

5 Dokumentacja

16. Wprowadzanie danych

Wprowadzanie danych odbywa się na poziomie uwierzytelniania i jednoczesnej autoryzacji według roli przypisanej dla użytkownika po stronie bazy danych. Predefiniowane role dla systemu stanowią: 'admin', 'employee' oraz 'customer'. Dwie pierwsze są przypisane bezpośrednio na poziomie bazy danych kiedy dodawani są pracownicy (administrator to też pracownik), natomiast rejestracja nowego użytkowanika, którą umożliwia przycisk w oknie logowanie domyślnie przypisuje nowemu użytkownikowi role klienta tj. 'customer'. Admin posiada wgląd do wszystkich danych.

Dane (na tym poziomie wykonania aplikacji) pracownik może: dodawać bądź usuwać samochód z bazy, lub zmieniając jego status (jeśli jest np. w naprawie, lub ma nieważne ubezpieczenie). Może ponadto zatwierdzać płatności, które obsługują odpowiednie triggery po stronie bazy.

Klient, przy pomocy przycisków wypożyczenia oraz zwrotu samochodu edytuje odpowiednie pola w bazie danych pośrednio, wypożyczając tworzy nowy rekord w tabeli wypożyczeń, płatność obsługiwana jest w momencie zwrotu, sam samochód zmienia status na wypożyczony i nie jest już obecny w widoku głównym dla innych klietów.

17. Dokumentacja użytkownika

Admin

Panel administratora ma widok odpowiednich zakładek **Raporty, Wypożyczenia, Użytkownicy, Klienci, Pracownicy, Samochody**. Każda sekcja poza raportami została rozszerzona o możliwość filtrowania danych oraz sortowania ich według odpowiedniego atrybutu.

Główna sekcja raportów na podstawie predefiniowanych widoków po stronie bazy danych umożliwa wygenerowanie:

- Podsumowania wypożyczeń klientów
- Podsumowania popularności samochodów
- Podsumowania przychodów miesięcznych

Pracownik

Panel pracownika ma widok odpowiednich zakładek **Samochody, Wypożyczenia, Płatności**. Każda sekcja poza raportami została rozszerzona o możliwość filtrowania danych oraz sortowania ich według odpowiedniego atrybutu. W sekcji **Samochody** są przyciski odpowiedzialne za dodawanie oraz usuwanie samochodu z bazy danych. W oknie **Płatności** jest przycisk otwierający okno formularza dla operacji zatwierdzania płatności

Klient

Panel klienta ma widok odpowiednich zakładek **Samochody, Wypożyczenia, FAQ, Historia**. Okno wypożyczenia ma inny layout dla klientów, ponieważ przdstawia tylko aktualne wypożyczenie, za pomocą odpowiedniego widoku dane wypożyczenia są wypisywane na ekranie, poniżej natomiast znajduje się przycisk umożliwiający zwrot samochodu, logika aplikacji pod spodem wykonuje odpowiednie operacje aktualizujące baze. Klient ma również wgląd do historii wypożyczeń. Ważne, aktualnie wypożyczenie aktualizuje się po wylogowaniu i ponownym zalogowaniu.

18. Opracowanie dokumentacji technicznej

Dokumentacja techniczna jest w odpowiednich plikach projektu, każda metoda posiada wyjaśnienie tego jak działa, to znaczy jakie dane przyjmuje oraz jakie zwraca.

Niezbędne jest zainstalowanie odpowiednich zależności dla uruchomienia projektu

```
python -m venv venv
source venv/bin/activate

pip install -r requirements.txt
```

Listing 5: Instalacja zależności dla projektu.

5.1 19. Wykaz literatury

- **Psycopg2** – Oficjalna dokumentacja biblioteki Psycopg2, która służy do łączenia i pracy z bazą danych PostgreSQL w Pythonie: `https://www.psycopg.org/docs/`
- **PyQt6** – Oficjalna dokumentacja PyQt6, biblioteki umożliwiającej tworzenie graficznych interfejsów użytkownika (GUI) z wykorzystaniem Qt w Pythonie: `https://riverbankcomputing.com/software/pyqt/intro`

6 Projekt

Link do projektu: `https://github.com/Krzemon/Car-Rental-App`