Collegium Witelona Uczelnia Państwowa w Legnicy
Wydział Nauk Technicznych i Ekonomicznych
Kierunek: Informatyka

Projekt z przedmiotu Projektowanie systemów baz danych

Temat: System zarządzania wypożyczalnią samochodów.

Autorzy
Mateusz Bogacz-Drewniak, nr. indeksu: 44491
Paweł Kruk, nr. indeksu: 43845

Grupa: 1(2)

Prowadzący przedmiot
mgr inż. Roman Hojniak

Legnica, 2025

Spis treści

1 Koncepcja **3**
- 1.1 Cel projektu bazy danych . . . 3
- 1.2 Opis dziedziny przedmiotowej . . . 3
- 1.3 Założenia wstępne . . . 5

2 Specyfikacja wymagań systemu **6**
- 2.1 Użytkownicy systemu oraz ich role . . . 6
  - 2.1.1 Niezalogowani użytkownicy . . . 6
  - 2.1.2 Zalogowani klienci (Użytkownicy) . . . 6
  - 2.1.3 Administratorzy . . . 6
- 2.2 Wymagania funkcjonalne . . . 8
  - 2.2.1 Moduł autentykacji . . . 8
  - 2.2.2 Moduł zarządzania użytkownikami . . . 8
  - 2.2.3 Moduł zarządzania samochodami . . . 8
  - 2.2.4 Moduł wypożyczeń . . . 8
  - 2.2.5 Moduł historii zmian . . . 9
- 2.3 Wymagania niefunkcjonalne . . . 9
  - 2.3.1 Wymagania dotyczące użyteczności . . . 9
  - 2.3.2 Wymagania dotyczące wydajności . . . 9
  - 2.3.3 Wymagania dotyczące bezpieczeństwa . . . 9
  - 2.3.4 Wymagania dotyczące niezawodności . . . 9

3 Model danych **10**
- 3.1 Schemat bazy danych . . . 10
- 3.2 Opis poszczególnych encji . . . 11
  - 3.2.1 Auta . . . 11
  - 3.2.2 AutaZdj . . . 11
  - 3.2.3 Miasta . . . 11
  - 3.2.4 Użytkownicy . . . 12
  - 3.2.5 Admin . . . 12
  - 3.2.6 Wypozyczenie . . . 12
  - 3.2.7 CzarnaLista . . . 13
  - 3.2.8 HistoriaZmian . . . 13
- 3.3 Model MVC . . . 14
  - 3.3.1 Model (Models) . . . 14
  - 3.3.2 Widok (Template) . . . 14
  - 3.3.3 Kontroler (View / Controller) . . . 15

4 Interfejs Użytkownika **16**
- 4.1 Opis GUI . . . 16
  - 4.1.1 Panel publiczny . . . 16
  - 4.1.2 Panel użytkownika . . . 17
  - 4.1.3 Panel administracyjny . . . 19
- 4.2 Implementacja . . . 21
  - 4.2.1 Definicje modeli . . . 21
  - 4.2.2 Repozytoria danych . . . 21
  - 4.2.3 Serwisy biznesowe . . . 22

4.2.4 Widoki (Controllers) 22
4.2.5 Konfiguracja URL 23
4.2.6 Walidacja i bezpieczeństwo 24

5 Słownik pojęć **25**

1 Koncepcja

1.1 Cel projektu bazy danych

Celem projektu jest stworzenie systemu zarządzania wypożyczalnią samochodów, który umożliwi efektywne zarządzanie flotą pojazdów, obsługę klientów oraz monitorowanie procesu wypożyczeń. System ma na celu usprawnienie procesów biznesowych związanych z wypożyczaniem pojazdów, umożliwienie użytkownikom przeglądania dostępnych samochodów, dokonywania rezerwacji oraz zarządzania swoimi wypożyczeniami.

Główne cele systemu:

- Zapewnienie platformy do zarządzania flotą pojazdów
- Udostępnienie klientom możliwości przeglądania i wypożyczania samochodów
- Zapewnienie efektywnego systemu rezerwacji i kontroli dostępności pojazdów
- Monitorowanie wypożyczeń i kontrola terminów zwrotu
- Zarządzanie klientami, w tym możliwość blokowania nieuczciwych użytkowników
- Śledzenie i rejestrowanie zmian dokonywanych w systemie dla celów audytu i kontroli

1.2 Opis dziedziny przedmiotowej

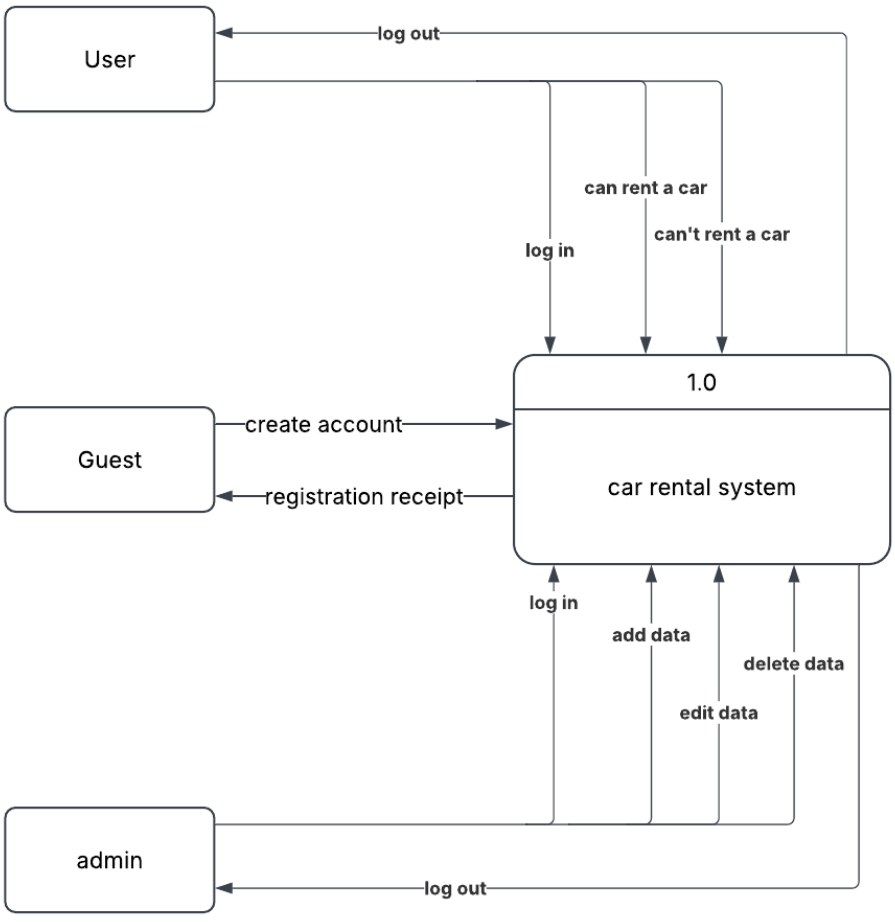
Wypożyczalnia samochodów to firma usługowa, która udostępnia pojazdy klientom na określony czas za odpowiednią opłatą. Proces wypożyczenia obejmuje kilka kluczowych etapów:

1. **Rejestracja klienta** - nowy użytkownik musi zarejestrować się w systemie podając swoje dane osobowe i adresowe
2. **Przeglądanie dostępnych pojazdów** - użytkownik przegląda dostępne samochody wraz z ich specyfikacjami
3. **Proces wypożyczenia** - wybór samochodu, określenie dat wypożyczenia, potwierdzenie
4. **Zarządzanie wypożyczeniem** - monitorowanie terminów, przedłużanie, zwrot

System obsługuje dwa główne typy użytkowników: klientów (osoby wypożyczające) oraz administratorów (pracowników wypożyczalni). Administratorzy mają dostęp do szerszych funkcjonalności, takich jak zarządzanie flotą, dodawanie nowych pojazdów, przeglądanie wszystkich wypożyczeń oraz zarządzanie czarną listą klientów.

Kluczowymi elementami dziedziny są:

- **Samochody** - pojazdy dostępne do wypożyczenia wraz z ich parametrami
- **Klienci** - osoby korzystające z usług wypożyczalni
- **Wypożyczenia** - rekordy dotyczące wypożyczonych pojazdów, dat i osób wypożyczających
- **Administratorzy** - osoby zarządzające systemem
- **Czarna lista** - lista klientów z ograniczonym dostępem do usług



Rysunek 1: Proces wypożyczenia samochodu

1.3 Założenia wstępne

Projekt systemu został oparty na następujących założeniach:

1. **Platforma technologiczna**:
    - Framework Django (Python)
    - Relacyjna baza danych (PostgreSQL)
    - Interfejs webowy (HTML, CSS, JavaScript)
    - Konteneryzacja (Docker)
2. **Struktura danych**:
    - System przechowuje dane o samochodach, klientach, administratorach, wypożyczeniach i blokadach
    - Każdy samochód posiada swoją specyfikację i zdjęcia
    - Klienci posiadają dane osobowe oraz adresowe
    - Wypożyczenia zawierają informacje o datach, użytkowniku i wypożyczonym pojeździe
3. **Funkcjonalność**:
    - Dwupoziomowy system uprawnień (administrator/użytkownik)
    - Mechanizm autentykacji i autoryzacji
    - Zarządzanie dostępnością pojazdów
    - System blokowania nieuczciwych klientów
    - Zarządzanie zdjęciami pojazdów
4. **Interfejs**:
    - Responsywny interfejs użytkownika
    - Oddzielne panele dla klientów i administratorów
    - Przejrzysty katalog pojazdów z filtrowaniem
    - Intuicyjny system rezerwacji

2 Specyfikacja wymagań systemu

2.1 Użytkownicy systemu oraz ich role

System obsługuje trzy główne typy użytkowników:

2.1.1 Niezalogowani użytkownicy

Uprawnienia:

- Przeglądanie strony głównej
- Rejestracja w systemie
- Logowanie do systemu

2.1.2 Zalogowani klienci (Użytkownicy)

Uprawnienia:

- Przeglądanie dostępnych samochodów
- Wyświetlanie szczegółów pojazdów
- Wypożyczanie samochodów
- Przeglądanie własnych wypożyczeń
- Edycja własnych danych osobowych

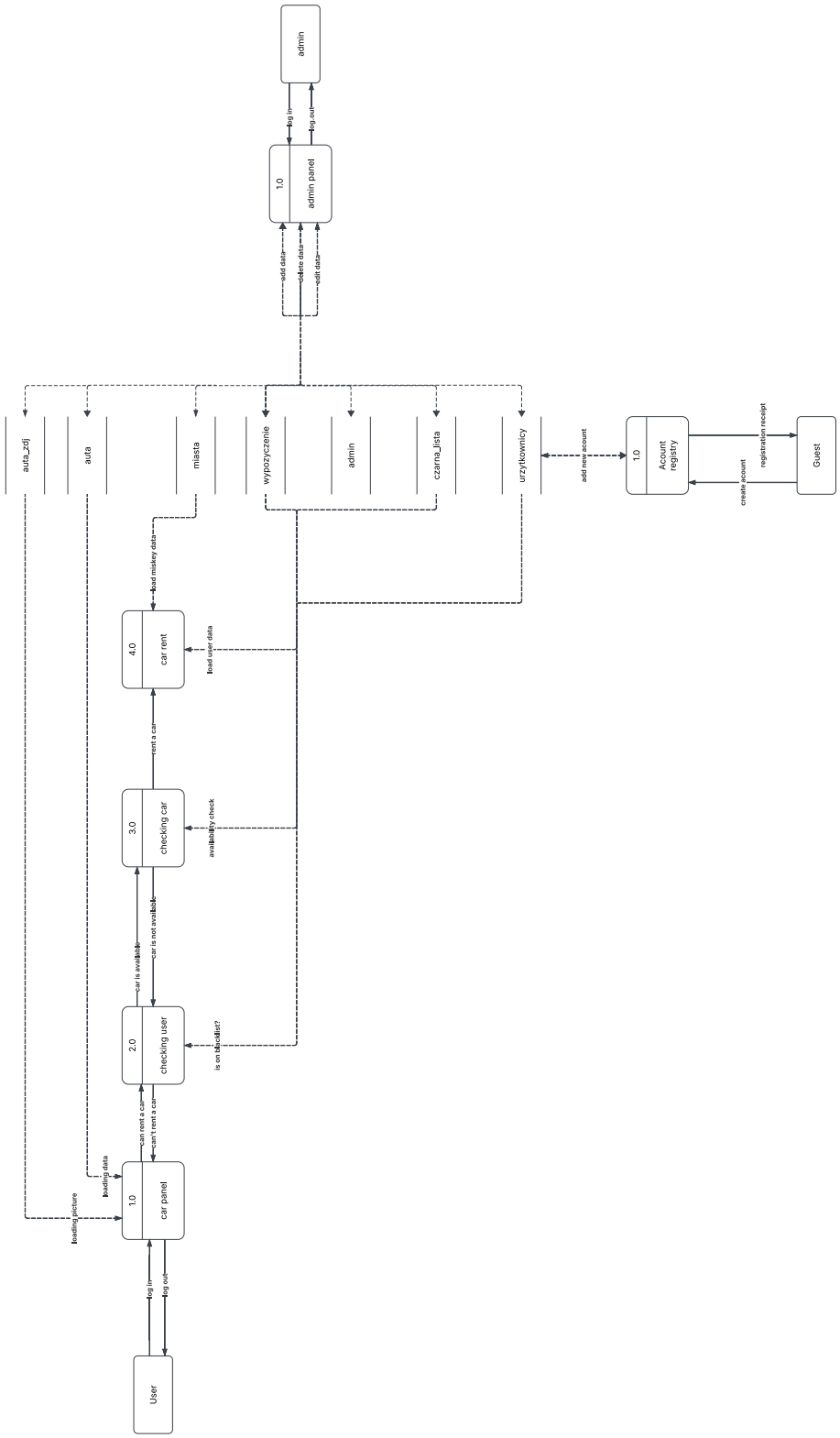
Ograniczenia:

- Brak dostępu do panelu administracyjnego
- Brak możliwości modyfikacji danych samochodów
- Brak możliwości wypożyczenia jeśli użytkownik jest na czarnej liście

2.1.3 Administratorzy

Uprawnienia:

- Zarządzanie flotą samochodów (dodawanie, edycja, usuwanie)
- Zarządzanie danymi pojazdów (w tym zdjęcia)
- Zarządzanie wypożyczeniami
- Zarządzanie użytkownikami
- Dodawanie i usuwanie użytkowników z czarnej listy
- Zarządzanie innymi administratorami
- Zarządzanie adresami



Rysunek 2: Diagram przypadków użycia systemu

2.2 Wymagania funkcjonalne

2.2.1 Moduł autentykacji

- F1.1: System umożliwia rejestrację nowych użytkowników
- F1.2: System zapewnia logowanie dla użytkowników i administratorów
- F1.3: System umożliwia wylogowanie z systemu
- F1.4: System weryfikuje uprawnienia użytkowników do dostępu do funkcji

2.2.2 Moduł zarządzania użytkownikami

- F2.1: System umożliwia administratorowi przeglądanie wszystkich użytkowników
- F2.2: System umożliwia administratorowi dodawanie nowych użytkowników
- F2.3: System umożliwia administratorowi edycję danych użytkowników
- F2.4: System umożliwia administratorowi usuwanie użytkowników
- F2.5: System umożliwia dodawanie użytkowników do czarnej listy
- F2.6: System umożliwia usuwanie użytkowników z czarnej listy
- F2.7: System umożliwia administratorowi zarządzanie adresami użytkowników

2.2.3 Moduł zarządzania samochodami

- F3.1: System umożliwia administratorowi dodawanie nowych samochodów
- F3.2: System umożliwia administratorowi edycję danych samochodów
- F3.3: System umożliwia administratorowi usuwanie samochodów
- F3.4: System umożliwia administratorowi zarządzanie zdjęciami samochodów
- F3.5: System wyświetla szczegóły wybranego samochodu
- F3.6: System umożliwia administratorowi zarządzanie kolejnością zdjęć samochodów

2.2.4 Moduł wypożyczeń

- F4.1: System umożliwia klientom wypożyczanie dostępnych samochodów
- F4.2: System weryfikuje dostępność samochodu przed wypożyczeniem
- F4.3: System weryfikuje czy klient nie jest na czarnej liście
- F4.4: System umożliwia administratorowi przeglądanie wszystkich wypożyczeń
- F4.5: System umożliwia administratorowi dodawanie, edycję i usuwanie wypożyczeń
- F4.6: System wyświetla historię wypożyczeń dla każdego samochodu

2.2.5 Moduł historii zmian

- F5.1: System rejestruje wszystkie operacje wykonywane na danych (INSERT, UPDATE, DELETE)
- F5.2: System umożliwia administratorowi przeglądanie historii zmian
- F5.3: System umożliwia filtrowanie historii zmian według różnych kryteriów
- F5.4: System umożliwia eksport historii zmian do formatu CSV

2.3 Wymagania niefunkcjonalne

2.3.1 Wymagania dotyczące użyteczności

- NF1.1: System posiada intuicyjny interfejs użytkownika
- NF1.2: Interfejs systemu jest responsywny i dostosowany do różnych urządzeń
- NF1.3: System informuje użytkowników o statusie wykonywanych operacji (komunikaty sukcesu, błędu)
- NF1.4: System obsługuje standardowe formaty zdjęć (JPG, PNG, GIF, BMP)

2.3.2 Wymagania dotyczące wydajności

- NF2.1: System odpowiada na zapytania użytkownika w czasie krótszym niż 2 sekundy
- NF2.2: System obsługuje jednoczesne wypożyczanie tego samego samochodu przez wielu użytkowników
- NF2.3: System efektywnie zarządza zasobami serwera (pamięć, CPU)

2.3.3 Wymagania dotyczące bezpieczeństwa

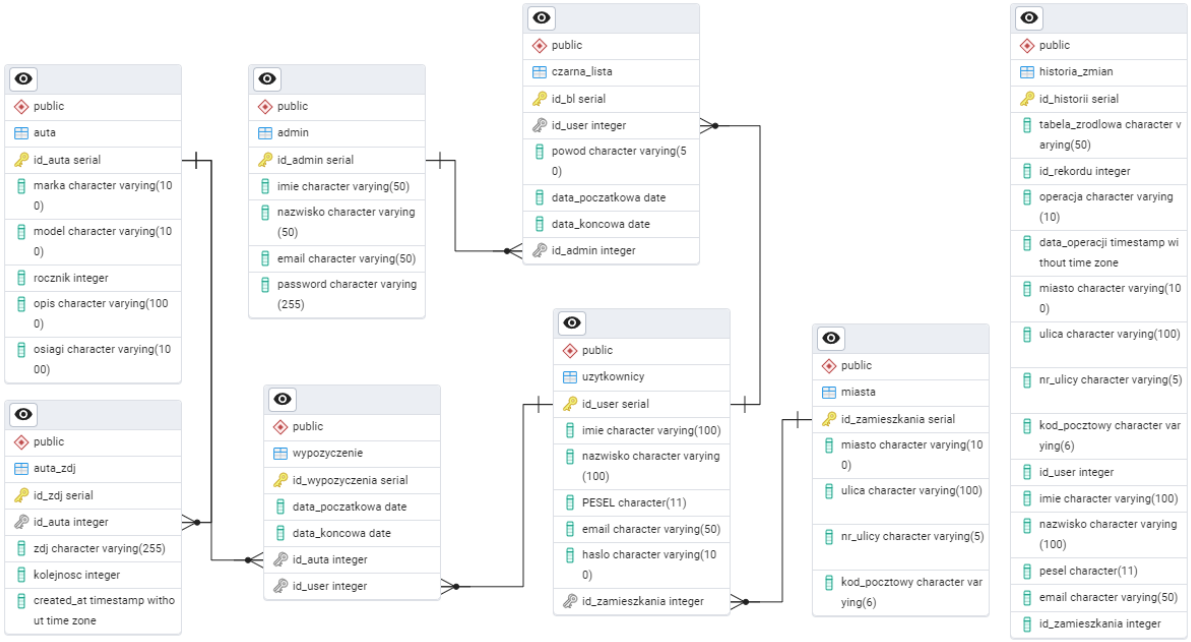
- NF3.1: Hasła użytkowników są przechowywane w formie zahaszowanej
- NF3.2: System zabezpiecza przed nieautoryzowanym dostępem do panelu administracyjnego
- NF3.3: System zabezpiecza dane przed atakami typu SQL Injection i Cross-Site Scripting
- NF3.4: System wymaga uwierzytelnienia do dostępu do funkcji klienta i administratora

2.3.4 Wymagania dotyczące niezawodności

- NF4.1: System jest dostępny 24/7 z wyjątkiem zaplanowanych przerw konserwacyjnych
- NF4.2: System obsługuje błędy w sposób elegancki, dostarczając użytkownikowi informacje o problemie
- NF4.3: System posiada procedury tworzenia kopii zapasowych danych

3 Model danych

3.1 Schemat bazy danych



Rysunek 3: Diagram relacji encji (ERD) bazy danych

Relacje między tabelami:

1. Użytkownik (Uzytkownicy) ma jeden adres (Miasta) - relacja jeden do wielu
2. Samochód (Auta) może mieć wiele zdjęć (AutaZdj) - relacja jeden do wielu
3. Samochód (Auta) może być w wielu wypożyczeniach (Wypozyczenie) - relacja jeden do wielu
4. Użytkownik (Uzytkownicy) może mieć wiele wypożyczeń (Wypozyczenie) - relacja jeden do wielu
5. Użytkownik (Uzytkownicy) może być na czarnej liście (CzarnaLista) - relacja jeden do wielu
6. Administrator (Admin) może dodać wielu użytkowników do czarnej listy (CzarnaLista) - relacja jeden do wielu

3.2 Opis poszczególnych encji

3.2.1 Auta

Tabela przechowująca informacje o samochodach dostępnych w wypożyczalni.

Pole	Typ	Opis
id_auta	AutoField (PK)	Unikalny identyfikator samochodu
marka	CharField(100)	Marka samochodu
model	CharField(100)	Model samochodu
rocznik	IntegerField	Rok produkcji
opis	CharField(1000)	Opis samochodu
osiagi	CharField(1000)	Osiągi samochodu

Tabela 1: Struktura tabeli Auta

3.2.2 AutaZdj

Tabela przechowująca zdjęcia samochodów.

Pole	Typ	Opis
id_zdj	AutoField (PK)	Unikalny identyfikator zdjęcia
id_auta	ForeignKey	Powiązanie z samochodem
zdj	CharField(255)	Ścieżka do pliku zdjęcia
kolejnosc	IntegerField	Kolejność wyświetlania zdjęcia
created_at	DateTimeField	Data dodania zdjęcia

Tabela 2: Struktura tabeli AutaZdj

3.2.3 Miasta

Tabela przechowująca adresy użytkowników.

Pole	Typ	Opis
id_zamieszkania	AutoField (PK)	Unikalny identyfikator adresu
miasto	CharField(100)	Nazwa miasta
ulica	CharField(100)	Nazwa ulicy
nr_ulicy	CharField(5)	Numer budynku
kod_pocztowy	CharField(6)	Kod pocztowy (format: XX-XXX)

Tabela 3: Struktura tabeli Miasta

3.2.4 Użytkownicy

Tabela przechowująca dane użytkowników systemu.

Pole	**Typ**	**Opis**
id_user	AutoField (PK)	Unikalny identyfikator użytkownika
imie	CharField(100)	Imię użytkownika
nazwisko	CharField(100)	Nazwisko użytkownika
pesel	CharField(11)	Numer PESEL
email	CharField(50)	Adres email (używany do logowania)
haslo	CharField(100)	Zahaszowane hasło użytkownika
id_zamieszkania	ForeignKey	Powiązanie z adresem zamieszkania

Tabela 4: Struktura tabeli Użytkownicy

3.2.5 Admin

Tabela przechowująca dane administratorów systemu.

Pole	**Typ**	**Opis**
id_admin	AutoField (PK)	Unikalny identyfikator administratora
imie	CharField(50)	Imię administratora
nazwisko	CharField(50)	Nazwisko administratora
email	CharField(50)	Adres email (używany do logowania)
haslo	CharField(255)	Zahaszowane hasło administratora

Tabela 5: Struktura tabeli Admin

3.2.6 Wypozyczenie

Tabela przechowująca informacje o wypożyczeniach.

Pole	**Typ**	**Opis**
id_wypozyczenia	AutoField (PK)	Unikalny identyfikator wypożyczenia
data_poczatkowa	DateField	Data rozpoczęcia wypożyczenia
data_koncowa	DateField	Data zakończenia wypożyczenia
id_auta	ForeignKey	Powiązanie z wypożyczanym samochodem
id_user	ForeignKey	Powiązanie z użytkownikiem wypożyczającym

Tabela 6: Struktura tabeli Wypozyczenie

3.2.7 CzarnaLista

Tabela przechowująca informacje o użytkownikach na czarnej liście.

Pole	**Typ**	**Opis**
id_bl	AutoField (PK)	Unikalny identyfikator wpisu
id_user	ForeignKey	Powiązanie z użytkownikiem
powod	CharField(50)	Powód dodania do czarnej listy
data_poczatkowa	DateField	Data rozpoczęcia blokady
data_koncowa	DateField	Data zakończenia blokady
id_admin	ForeignKey	Administrator, który dodał wpis

Tabela 7: Struktura tabeli CzarnaLista

3.2.8 HistoriaZmian

Tabela przechowująca historię operacji wykonywanych na danych w systemie.

Pole	**Typ**	**Opis**
id_historii	AutoField (PK)	Unikalny identyfikator wpisu historii
tabela_zrodlowa	CharField(50)	Nazwa tabeli, której dotyczy operacja
id_rekordu	IntegerField	ID rekordu, którego dotyczy operacja
operacja	CharField(10)	Rodzaj operacji (INSERT, UPDATE, DELETE)
data_operacji	DateTimeField	Data i czas wykonania operacji
miasto	CharField(100)	Miasto (jeśli dotyczy)
ulica	CharField(100)	Ulica (jeśli dotyczy)
nr_ulicy	CharField(5)	Numer ulicy (jeśli dotyczy)
kod_pocztowy	CharField(6)	Kod pocztowy (jeśli dotyczy)
id_user	IntegerField	ID użytkownika (jeśli dotyczy)
imie	CharField(100)	Imię (jeśli dotyczy)
nazwisko	CharField(100)	Nazwisko (jeśli dotyczy)
pesel	CharField(11)	PESEL (jeśli dotyczy)
email	CharField(50)	Email (jeśli dotyczy)
id_zamieszkania	IntegerField	ID zamieszkania (jeśli dotyczy)

Tabela 8: Struktura tabeli HistoriaZmian

3.3 Model MVC

Aplikacja została zaimplementowana zgodnie z architekturą Model-View-Controller (MVC), która w Django jest nazywana często Model-Template-View (MTV). Poniżej przedstawiono podział komponentów aplikacji według tego wzorca:

3.3.1 Model (Models)

Modele reprezentują strukturę danych i logikę biznesową aplikacji:

- **models.py** - zawiera definicje wszystkich klas modeli (Auta, AutaZdj, Miasta, Uzytkownicy, Admin, Wypozyczenie, CzarnaLista), które odpowiadają tabelom w bazie danych.
- **repositories/** - katalog zawierający klasy repozytoriów, które zapewniają dostęp do danych:
  - **admin_repository.py** - operacje na danych administratorów
  - **blacklist_repository.py** - operacje na danych czarnej listy
  - **car_repository.py** - operacje na danych samochodów i ich zdjęć
  - **rental_repository.py** - operacje na danych wypożyczeń
  - **user_repository.py** - operacje na danych użytkowników i adresów
  - **change_history_repository.py** - operacje na danych historii zmian

3.3.2 Widok (Template)

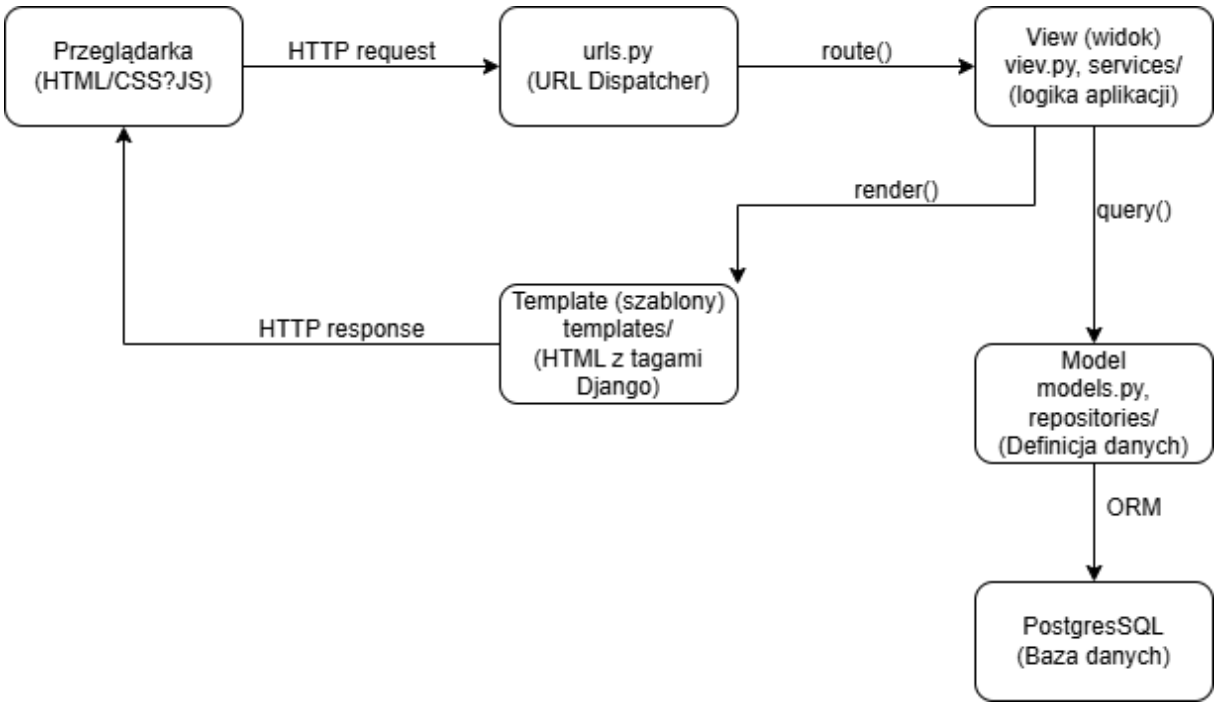
Szablony definiują sposób prezentacji danych użytkownikowi:

- **templates/** - katalog zawierający wszystkie szablony HTML aplikacji:
  - Strony główne: index.html, login.html, register.html
  - Panel użytkownika: user_dashboard.html, car_detail.html, rent_car.html, user_ban_info.html
  - Panel administracyjny: admin_dashboard.html, admin_car_view.html, admin_car_photos.html admin_address_view.html, admin_rent_view.html, admin_admin_view.html, admin_user_view.html, admin_blacklist_view.html
  - Komponenty wspólne: includes/navbar.html, includes/navbar1.html, includes/navbar2.html, includes/navbar3.html, includes/admin_sidebar.html, includes/footer.html, includes/head.html
  - Strony błędów: 400.html, 404.html, 500.html, error.html

3.3.3 Kontroler (View / Controller)

Kontrolery obsługują interakcje użytkownika i zarządzają przepływem danych:

- **views.py** - zawiera funkcje widoków, które obsługują żądania HTTP i zwracają odpowiedzi
- **services/** - katalog zawierający klasy serwisów, które implementują logikę biznesową:
  - **admin_service.py** - logika biznesowa związana z administratorami
  - **blacklist_service.py** - logika biznesowa związana z czarną listą
  - **car_service.py** - logika biznesowa związana z samochodami
  - **rental_service.py** - logika biznesowa związana z wypożyczeniami
  - **user_service.py** - logika biznesowa związana z użytkownikami
  - **change_history_service.py** - logika biznesowa związana z historią zmian
- **forms.py** - zawiera definicje formularzy wykorzystywanych w aplikacji
- **urls.py** - definiuje mapowanie URL do funkcji widoków



Rysunek 4: Architektura Model-Template-View (MTV) aplikacji Django

4 Interfejs Użytkownika

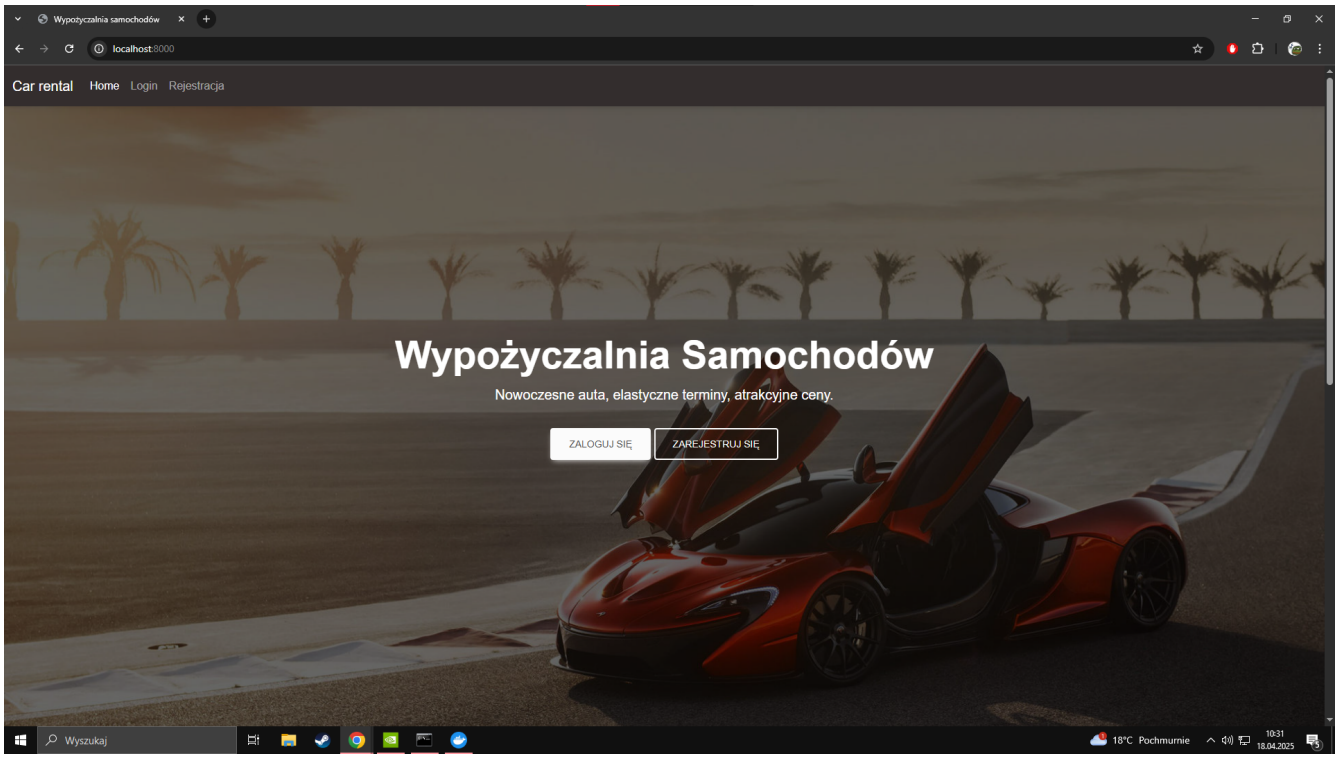
4.1 Opis GUI

System wykorzystuje interfejs webowy oparty na frameworku Bootstrap, co zapewnia responsywność i spójny wygląd na różnych urządzeniach. Interfejs podzielony jest na dwie główne części: panel użytkownika i panel administracyjny.

4.1.1 Panel publiczny

Dostępny dla niezalogowanych użytkowników:

1. **Strona główna (index.html)**
   - Prezentuje podstawowe informacje o wypożyczalni
   - Zawiera przyciski do logowania i rejestracji
   - Menu nawigacyjne z opcjami: Home, Login, Rejestracja
2. **Strona logowania (login.html)**
   - Formularz logowania z polami: email, hasło
   - Przekierowanie do odpowiedniego panelu (administratora lub użytkownika) po zalogowaniu
   - Obsługa komunikatów błędów (np. nieprawidłowe dane logowania)
3. **Strona rejestracji (register.html)**
   - Formularz rejestracji z sekcjami:
     - Dane osobowe (imię, nazwisko, PESEL, email)
     - Adres zamieszkania (miasto, ulica, numer, kod pocztowy)
     - Hasło (hasło, potwierdzenie hasła)
   - Walidacja wprowadzanych danych
   - Komunikaty o błędach formularza



Rysunek 5: Widok strony głównej

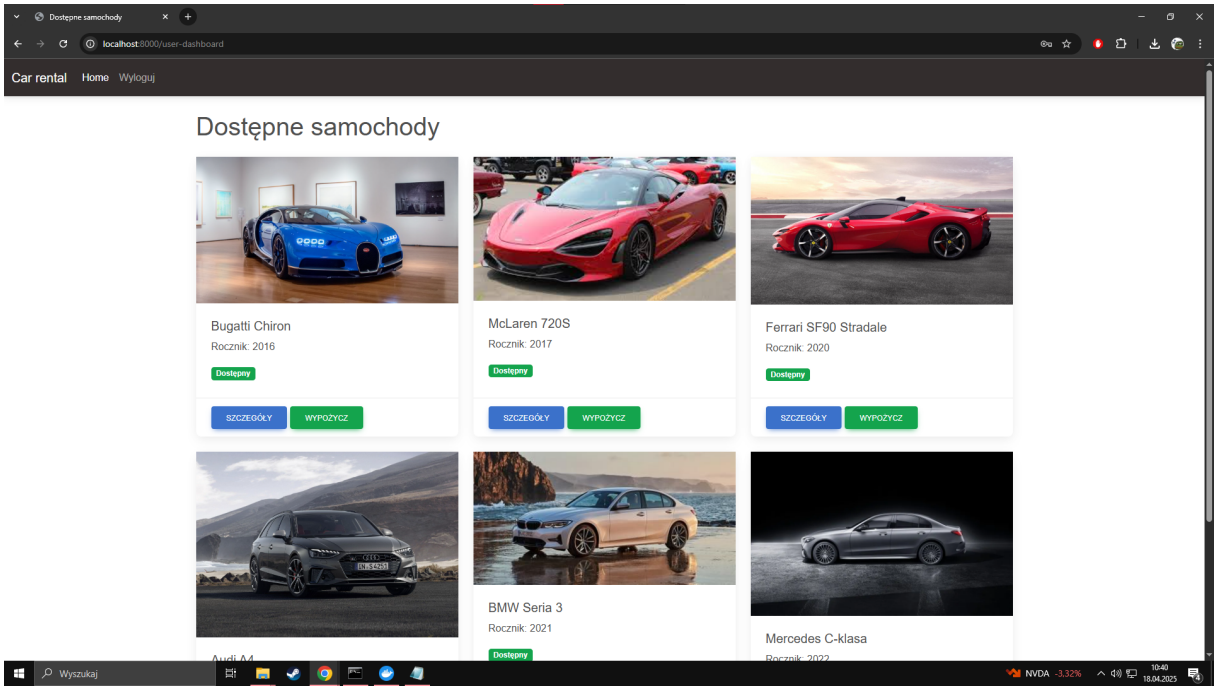
4.1.2 Panel użytkownika

Dostępny dla zalogowanych klientów:

1. **Strona główna panelu użytkownika (user_dashboard.html)**
   - Wyświetla listę aktualnie dostępnych samochodów wraz z podstawowymi informacjami
   - Zawiera filtry pozwalające zawęzić wybór samochodów (np. marka, model, rocznik)
   - Menu nawigacyjne z opcjami: Strona główna, Moje wypożyczenia, Mój profil, Wyloguj
2. **Strona szczegółów samochodu (car_detail.html)**
   - Prezentuje szczegółowe informacje o wybranym samochodzie
   - Wyświetla galerię zdjęć z możliwością powiększenia
   - Zawiera sekcje: specyfikacja, osiągi, opis
   - Pokazuje dostępność samochodu w kalendarzu
   - Zawiera przycisk "Wypożycz"prowadzący do formularza wypożyczenia
3. **Formularz wypożyczenia samochodu (rent_car.html)**
   - Pozwala na wybór daty rozpoczęcia i zakończenia wypożyczenia
   - Wyświetla koszt wypożyczenia

- Zawiera przycisk potwierdzenia wypożyczenia
- Waliduje dostępność samochodu w wybranym terminie
- Sprawdza, czy użytkownik nie jest na czarnej liście

4. **Strona moich wypożyczeń**
   - Wyświetla listę aktualnych i historycznych wypożyczeń użytkownika
   - Pokazuje status każdego wypożyczenia (aktywne, zakończone)
   - Umożliwia podgląd szczegółów każdego wypożyczenia
5. **Strona mojego profilu**
   - Wyświetla dane osobowe użytkownika
   - Umożliwia edycję danych kontaktowych i adresowych
   - Pozwala na zmianę hasła
6. **Strona informacji o blokadzie (user_ban_info.html)**
   - Wyświetlana, gdy użytkownik jest na czarnej liście
   - Informuje o powodzie blokady
   - Pokazuje datę rozpoczęcia i zakończenia blokady
   - Zawiera informacje kontaktowe do administracji



Rysunek 6: Przykładowy widok panelu użytkownika

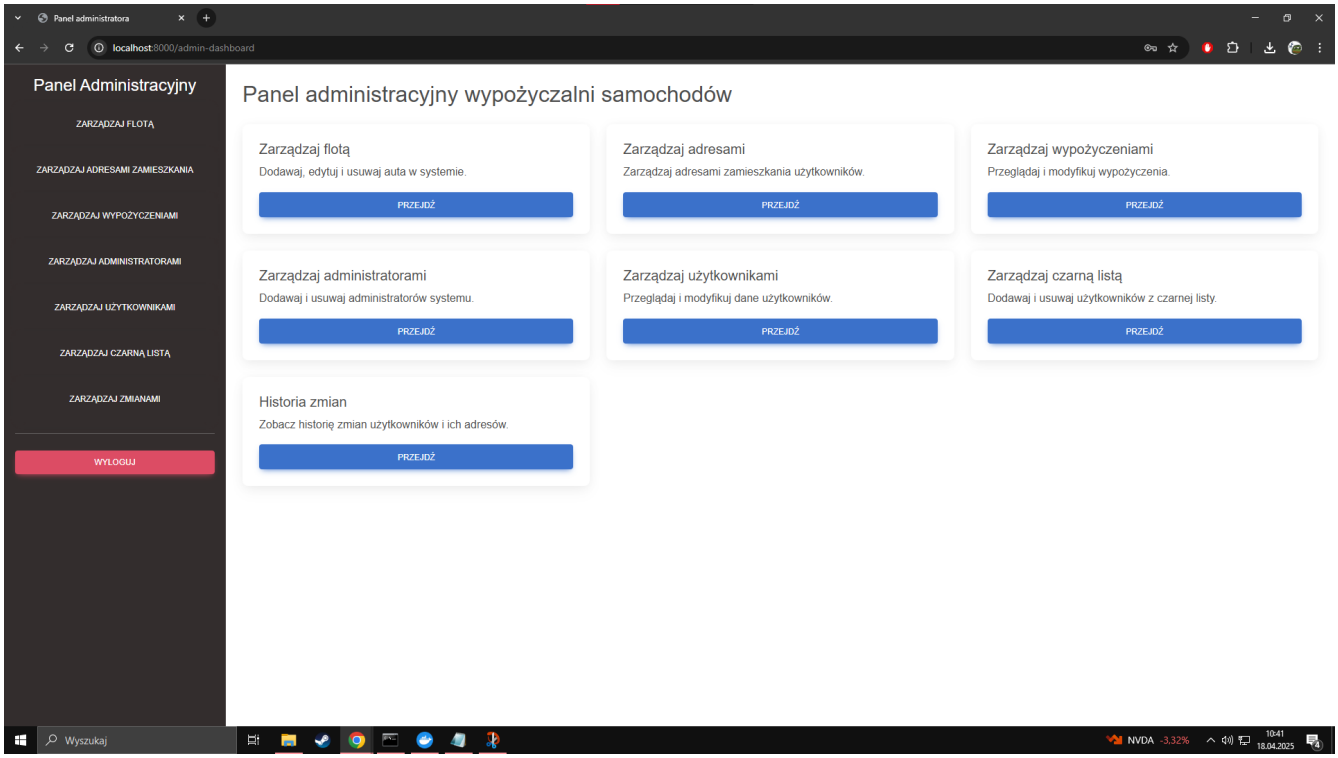
4.1.3 Panel administracyjny

Dostępny dla zalogowanych administratorów:

1. **Strona główna panelu administratora (admin_dashboard.html)**
   - Wyświetla podsumowanie systemu (liczba samochodów, użytkowników, aktywnych wypożyczeń)
   - Zawiera skróty do najczęściej używanych funkcji
   - Menu boczne (sidebar) z dostępem do wszystkich sekcji administracyjnych
   - Wyświetla alerty o terminach zwrotu samochodów i wygasających blokadach
2. **Zarządzanie samochodami (admin_car_view.html)**
   - Lista wszystkich samochodów w systemie z możliwością filtrowania i sortowania
   - Formularz dodawania nowego samochodu
   - Opcje edycji i usuwania istniejących samochodów
   - Przycisk przekierowujący do zarządzania zdjęciami dla wybranego samochodu
3. **Zarządzanie zdjęciami samochodów (admin_car_photos.html)**
   - Możliwość dodawania, usuwania i zmiany kolejności zdjęć dla wybranego samochodu
   - Podgląd wszystkich zdjęć samochodu
   - Formularz uploadu nowych zdjęć
   - Drag&drop do zmiany kolejności zdjęć
4. **Zarządzanie wypożyczeniami (admin_rent_view.html)**
   - Lista wszystkich wypożyczeń z możliwością filtrowania (aktywne, zakończone, według dat)
   - Szczegóły każdego wypożyczenia
   - Możliwość dodawania, edycji i usuwania wypożyczeń
   - Oznaczanie wypożyczeń jako zakończone
5. **Zarządzanie użytkownikami (admin_user_view.html)**
   - Lista wszystkich użytkowników systemu
   - Szczegóły każdego użytkownika wraz z historią wypożyczeń
   - Możliwość dodawania, edycji i usuwania użytkowników
   - Przycisk do dodawania użytkownika na czarną listę
6. **Zarządzanie czarną listą (admin_blacklist_view.html)**
   - Lista użytkowników znajdujących się na czarnej liście
   - Formularze dodawania użytkowników do czarnej listy
   - Możliwość edycji powodu i okresu blokady

- Opcja usunięcia użytkownika z czarnej listy

7. **Zarządzanie adresami (admin_address_view.html)**
   - Lista wszystkich adresów w systemie
   - Możliwość dodawania, edycji i usuwania adresów
   - Powiązanie adresów z użytkownikami
8. **Zarządzanie administratorami (admin_admin_view.html)**
   - Lista wszystkich administratorów systemu
   - Możliwość dodawania nowych administratorów
   - Opcje edycji danych i usuwania administratorów
   - Zmiana hasła dla administratorów



Rysunek 7: Przykładowy widok panelu administratora

4.2 Implementacja

W tej sekcji przedstawiono wybrane fragmenty kodu źródłowego aplikacji, które ilustrują kluczowe aspekty implementacji systemu wypożyczalni samochodów.

4.2.1 Definicje modeli

Poniżej przedstawiono przykładową definicję modelu odpowiadającemu tabeli w bazie danych.

```
class Auto(models.Model):
    id_auta = models.AutoField(primary_key=True)
    marka = models.CharField(max_length=100)
    model = models.CharField(max_length=100)
    rocznik = models.IntegerField()
    opis = models.CharField(max_length=1000)
    osiagi = models.CharField(max_length=1000)

    def __str__(self):
        return f"{self.marka} {self.model} ({self.rocznik})"

    class Meta:
        db_table = 'Auta'
```

Listing 1: Definicja modelu Auto

4.2.2 Repozytoria danych

Repozytoria zapewniają warstwę abstrakcji dla operacji na bazie danych, ukrywając bezpośrednie zapytania. Poniżej przedstawiono przykładpwe repozytorium.

```
class CarRepository:
    @staticmethod
    def get_all_cars():
        return Auto.objects.all()

    @staticmethod
    def get_car_by_id(car_id):
        try:
            return Auto.objects.get(id_auta=car_id)
        except Auto.DoesNotExist:
            return None

    @staticmethod
    def get_car_photos(car_id):
        return AutaZdj.objects.filter(id_auta=car_id).order_by('kolejnosc')
```

Listing 2: Fragment klasy CarRepository

4.2.3 Serwisy biznesowe

Serwisy implementują logikę biznesową aplikacji, niezależną od warstwy prezentacji.

```
class RentalService:
    def __init__(self, rental_repository, car_repository, blacklist_repository):
        self.rental_repository = rental_repository
        self.car_repository = car_repository
        self.blacklist_repository = blacklist_repository

    def rent_car(self, user_id, car_id, start_date, end_date):
        if self.blacklist_repository.is_user_blacklisted(user_id):
            return False, "Nie mo esz wypo yczy  samochodu, poniewa  jeste  na
    czarnej li cie."

        car = self.car_repository.get_car_by_id(car_id)
        if not car:
            return False, "Wybrany samoch d nie istnieje."

        if not self.is_car_available(car_id, start_date, end_date):
            return False, "Samoch d jest niedost pny w wybranym terminie."

        try:
            self.rental_repository.create_rental(user_id, car_id, start_date, end_date)
            return True, "Samoch d zosta  pomy lnie wypo yczony."
        except Exception as e:
            return False, f"Wyst pi  b  d podczas wypo yczania: {str(e)}"

    def is_car_available(self, car_id, start_date, end_date):
        rentals = self.rental_repository.get_car_rentals_in_period(car_id, start_date,
    end_date)
        return len(rentals) == 0
```

Listing 3: Fragment klasy RentalService

4.2.4 Widoki (Controllers)

Widoki obsługują żądania HTTP i delegują operacje do odpowiednich serwisów.

```
@login_required
def rent_car_view(request, car_id):
    car_repository = CarRepository()
    rental_repository = RentalRepository()
    blacklist_repository = BlacklistRepository()
    rental_service = RentalService(rental_repository, car_repository,
    blacklist_repository)

    car = car_repository.get_car_by_id(car_id)
    if not car:
        messages.error(request, "Wybrany samoch d nie istnieje.")
        return redirect('user_dashboard')

    if request.method == 'POST':
        form = RentCarForm(request.POST)
        if form.is_valid():
            start_date = form.cleaned_data['start_date']
            end_date = form.cleaned_data['end_date']

            if start_date < date.today():
                messages.error(request, "Data rozpocz cia nie mo e by  w przesz o ci
    .")
                return render(request, 'rent_car.html', {'form': form, 'car': car})

            if end_date < start_date:
                messages.error(request, "Data zako czenia nie mo e by  wcze niejsza
    ni  data rozpocz cia.")
                return render(request, 'rent_car.html', {'form': form, 'car': car})

            success, message = rental_service.rent_car(
                request.user.id_user,
                car_id,
                start_date,
```

```
                end_date
            )

            if success:
                messages.success(request, message)
                return redirect('user_rentals')
            else:
                messages.error(request, message)

        else:
            for field, errors in form.errors.items():
                for error in errors:
                    messages.error(request, f"{field}: {error}")
    else:
        form = RentCarForm()

    return render(request, 'rent_car.html', {
        'form': form,
        'car': car
    })
```

Listing 4: Fragment widoku wypożyczania samochodu

4.2.5 Konfiguracja URL

Plik urls.py definiuje mapowanie adresów URL na funkcje widoków.

```
urlpatterns = [
    path('', views.index, name='index'),
    path('login/', views.login_view, name='login'),
    path('register/', views.register_view, name='register'),
    path('logout/', views.logout_view, name='logout'),

    path('dashboard/', login_required(views.user_dashboard), name='user_dashboard'),
    path('car/<int:car_id>/', login_required(views.car_detail), name='car_detail'),
    path('rent/<int:car_id>/', login_required(views.rent_car_view), name='rent_car'),
    path('my-rentals/', login_required(views.user_rentals), name='user_rentals'),
    path('profile/', login_required(views.user_profile), name='user_profile'),

    path('admin/dashboard/', views.admin_required(views.admin_dashboard), name='
    admin_dashboard'),
    path('admin/cars/', views.admin_required(views.admin_car_view), name='admin_car_view'
    ),
    path('admin/car-photos/<int:car_id>/', views.admin_required(views.admin_car_photos),
    name='admin_car_photos'),
    path('admin/rentals/', views.admin_required(views.admin_rent_view), name='
    admin_rent_view'),
    path('admin/users/', views.admin_required(views.admin_user_view), name='
    admin_user_view'),
    path('admin/blacklist/', views.admin_required(views.admin_blacklist_view), name='
    admin_blacklist_view'),
    path('admin/addresses/', views.admin_required(views.admin_address_view), name='
    admin_address_view'),
    path('admin/admins/', views.admin_required(views.admin_admin_view), name='
    admin_admin_view'),
    path('admin/history-change/', views.admin_required(views.admin_histroy_change_list),
    name='admin_history_change_list'),
]
```

Listing 5: Fragment konfiguracji URL

4.2.6 Walidacja i bezpieczeństwo

System zawiera mechanizmy walidacji danych oraz zabezpieczenia przed nieautoryzowanym dostępem.

```
def admin_required(view_func):
    @wraps(view_func)
    def _wrapped_view(request, *args, **kwargs):
        if not request.user.is_authenticated:
            return redirect('login')

        try:
            admin = Admin.objects.get(email=request.user.email)
            request.user.is_admin = True
            return view_func(request, *args, **kwargs)
        except Admin.DoesNotExist:
            messages.error(request, "Nie masz uprawnie  do dost pu do panelu
    administracyjnego.")
            return redirect('index')

    return _wrapped_view
```

Listing 6: Dekorator weryfikujący uprawnienia administratora

```
class RentCarForm(forms.Form):
    start_date = forms.DateField(
        label="Data rozpocz cia",
        widget=forms.DateInput(attrs={'type': 'date', 'class': 'form-control'}),
        validators=[MinValueValidator(date.today())]
    )
    end_date = forms.DateField(
        label="Data zako czenia",
        widget=forms.DateInput(attrs={'type': 'date', 'class': 'form-control'})
    )

    def clean(self):
        cleaned_data = super().clean()
        start_date = cleaned_data.get('start_date')
        end_date = cleaned_data.get('end_date')

        if start_date and end_date:
            if end_date < start_date:
                raise ValidationError(
                    "Data zako czenia nie mo e by  wcze niejsza ni  data
    rozpocz cia."
                )

            if (end_date - start_date).days > 30:
                raise ValidationError(
                    "Maksymalny okres wypo yczenia to 30 dni."
                )

        return cleaned_data
```

Listing 7: Walidacja danych formularza

5 Słownik pojęć

F	Wymaganie Funkcjonalne (Functional Requirement) - określa funkcję, którą system powinien wykonywać. Opisuje, co system ma robić.
NF	Wymaganie Niefunkcjonalne (Non-Functional Requirement) - określa ograniczenia i cechy jakościowe systemu, takie jak wydajność, bezpieczeństwo, niezawodność czy użyteczność.
MTV	Model-Template-View - implementacja wzorca MVC w Django, gdzie Model odpowiada za dane, Template (szablon) za prezentację, a View (widok) za logikę biznesową.
Django	Framework webowy napisany w języku Python, wspierający szybkie tworzenie bezpiecznych i skalowalnych aplikacji webowych.
PostgreSQL	Obiektowo-relacyjny system zarządzania bazą danych, wykorzystywany w projekcie jako główne repozytorium danych.
CRUD	Create, Read, Update, Delete - cztery podstawowe operacje trwałego przechowywania danych, odpowiadające za tworzenie, odczytywanie, aktualizowanie i usuwanie rekordów.
Docker	Platforma do konteneryzacji aplikacji, umożliwiająca pakowanie aplikacji wraz z jej zależnościami w standaryzowany sposób.