

Dokumentacja aplikacji InstaCar

Sebastian Górski
Jakub Grzymisławski
Łukasz Szenkiel



Spis treści

Opis	4
1 Architektura systemu	5
1.1 Diagram czynności	5
1.2 Diagram kluczowych klas	6
1.3 Diagram encji	7
1.4 Diagram przypadków użycia	8
1.5 Implementacja kodu	9
1.5.1 Stos technologiczny	9
1.5.2 Widoki i formularze	9
1.5.3 Obsługa błędów	9
1.5.4 Przykładowe listingu kodu w języku programowania Java	9
2 Interfejs użytkownika	11
2.1 Ogólna charakterystyka	11
2.2 Widok użytkownika końcowego	12
2.3 Widok administratora	14
2.4 Styl wizualny i doświadczenie użytkownika (UX)	15
3 Dokumentacja technologiczna	16
3.1 Struktura bazy danych	16
3.1.1 flyway_schema_history	16
3.1.2 roles	16
3.1.3 app_users	16
3.1.4 email_tokens i password_tokens	16
3.1.5 car_models	16
3.1.6 vehicles	16
3.1.7 city_cars i sport_cars	17
3.1.8 user_details	17
3.1.9 cities	17
3.1.10 rents	17
3.2 Opis środowiska	17
3.2.1 Baza danych - PostgreSQL 17.2	17
3.2.2 Migracje schematu – Flyway 10.20.1	17
3.2.3 Cache – Redis 7.4.2-alpine	17
4 Zagadnienia kwalifikacyjne	18
4.1 Framework MVC	18
4.2 Framework CSS	18
4.3 Baza danych	18
4.4 Cache	18
4.5 Dependency manager	18
4.6 HTML	19
4.7 CSS	20
4.8 JavaScript	21
4.9 Routing	21

4.10 ORM	22
4.11 Uwierzytelnienie	22
4.12 Lokalizacja	22
4.13 Mailing	22
4.14 Formularze	22
4.15 Asynchroniczne interakcje	23
4.16 Konsumpcja API	23
4.17 Publikacja API	23
4.18 RWD	24
4.19 Logger	24
4.20 Deployment	24
Spis rysunków	25
Spis listingów	25

Opis

Dokumentacja dotyczy aplikacji *InstaCar*, nowoczesnego systemu służącego do rezerwacji samochodów na wynajem. Aplikacja umożliwia użytkownikom przeglądanie dostępnych pojazdów, filtrowanie wyników według, daty, ceny i rodzaju pojazdu, a następnie dokonywanie rezerwacji w wybranym terminie. Zarejestrowani użytkownicy mają również możliwość zarządzania swoimi rezerwacjami.

InstaCar została zaprojektowana z myślą o wygodzie użytkownika i łatwej skalowalności. Interfejs użytkownika jest responsywny i intuicyjny, dostosowany zarówno do urządzeń mobilnych, jak i komputerów stacjonarnych. Aplikacja wykorzystuje nowoczesne technologie frontendowe oraz bezpieczne mechanizmy logowania i autoryzacji. System wspiera również funkcjonalności takie jak powiadomienia e-mail o dokonaniu rezerwacji.

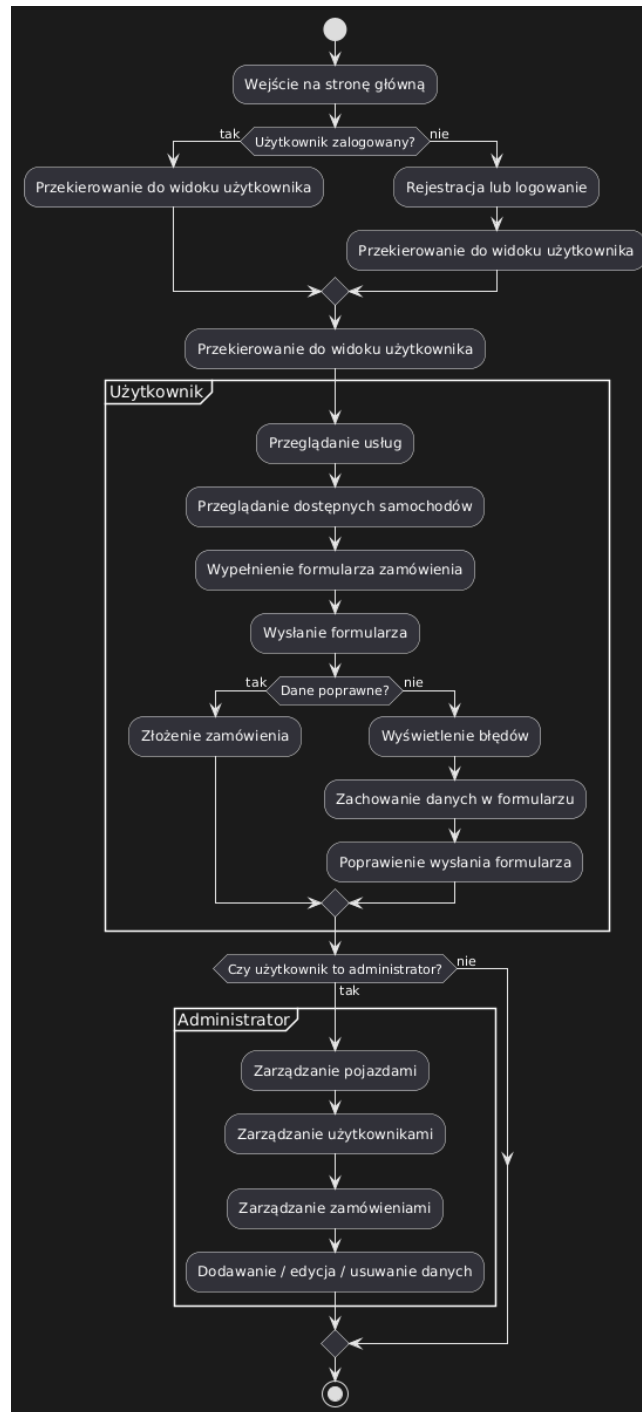
Dokumentacja zawiera szczegółowy opis funkcjonalności systemu, architektury aplikacji, instrukcję użytkowania, wymagania techniczne oraz informacje przeznaczone dla administratorów i deweloperów. Uwzględniono również aspekty związane z bezpieczeństwem, testowaniem oraz możliwością dalszego rozwoju aplikacji.

1 Architektura systemu

W tej sekcji przedstawiono opis architektury systemu *Instacar*.

1.1 Diagram czynności

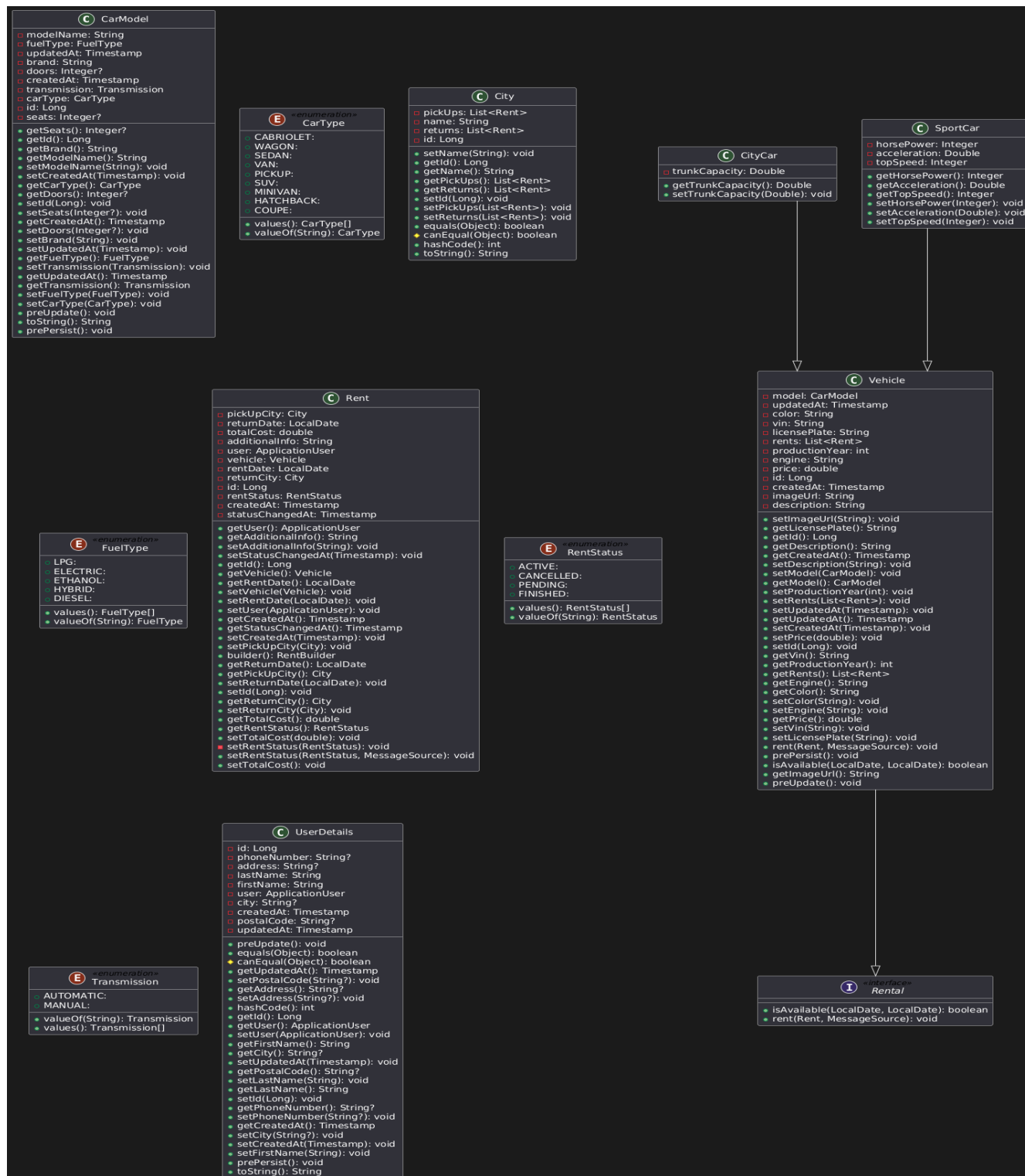
Diagram czynności ilustruje przebieg głównych procesów w aplikacji, takich jak rezerwacja pojazdu czy logowanie użytkownika.



Rysunek 1: Diagram czynności

1.2 Diagram kluczowych klas

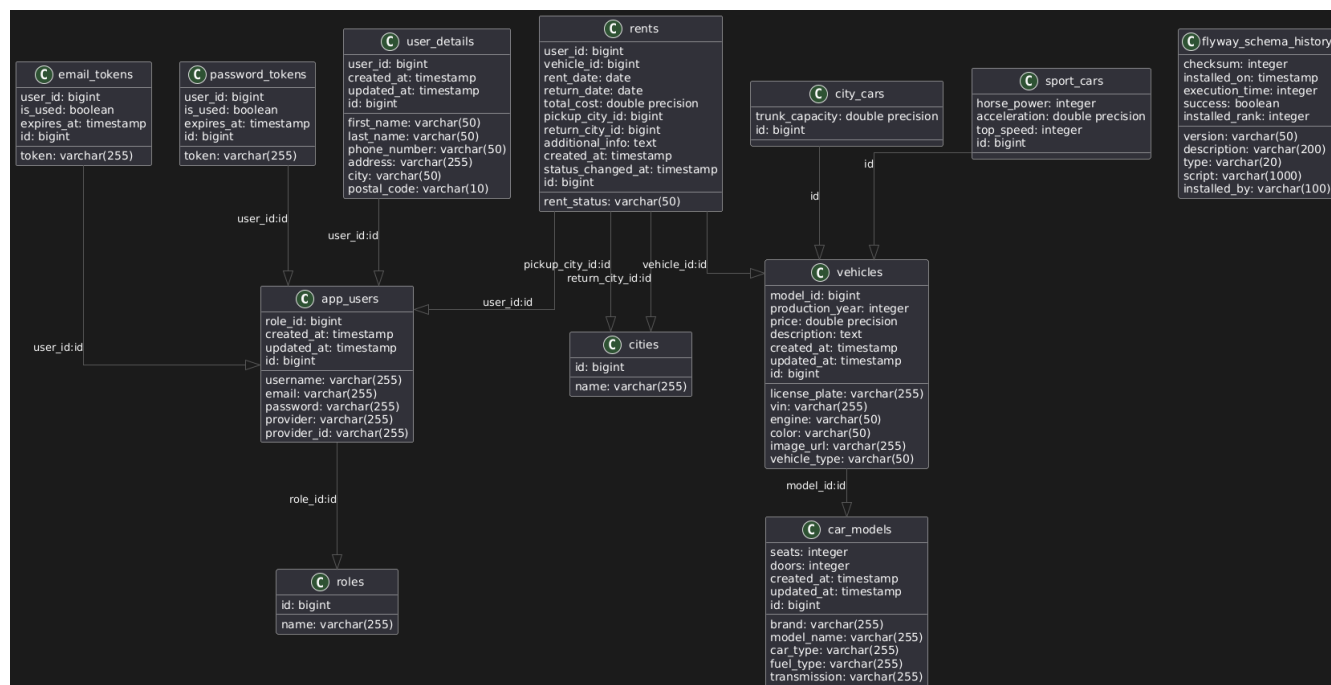
Diagram klas przedstawia strukturę aplikacji *InstaCar* w ujęciu obiekowym, wraz z zależnościami między klasami.



Rysunek 2: Diagram klas

1.3 Diagram encji

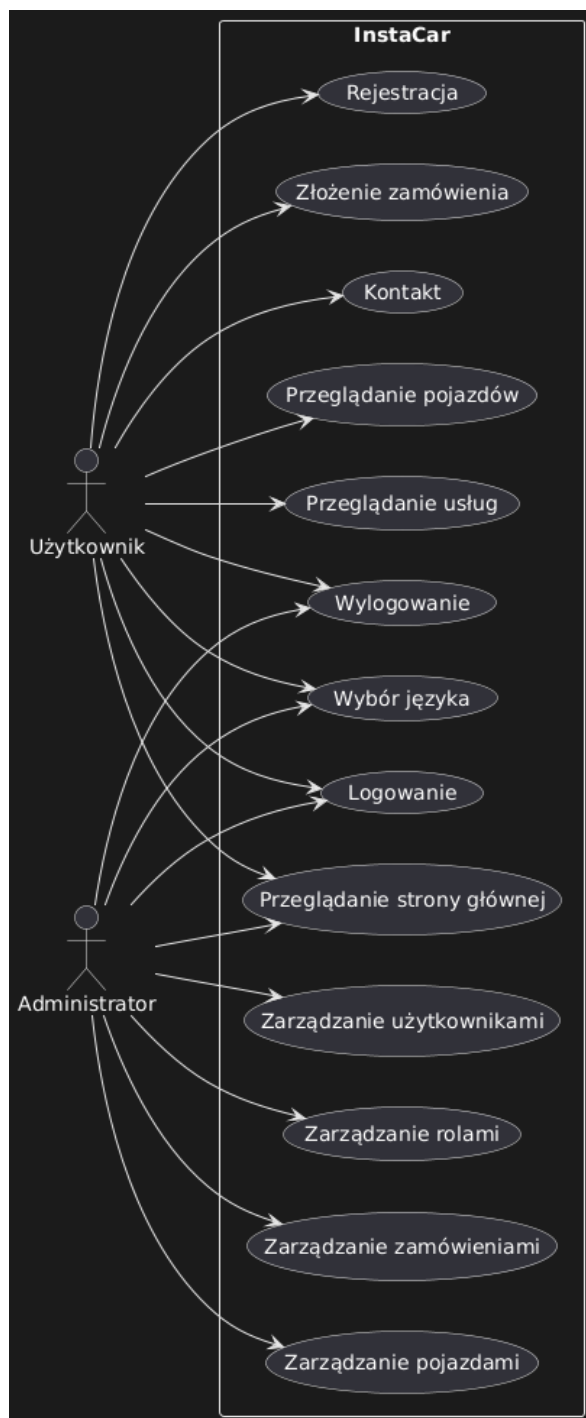
Diagram encji przedstawia strukturę bazy danych systemu wypożyczalni pojazdów, uwzględniając tabele użytkowników, pojazdów, wynajmów oraz relacje między nimi.



Rysunek 3: Diagram encji bazy danych

1.4 Diagram przypadków użycia

Diagram przypadków użycia ukazuje interakcje użytkowników z systemem, identyfikując główne funkcje dostępne w aplikacji.



Rysunek 4: Diagram przypadków użycia

1.5 Implementacja kodu

Aplikacja *InstaCar* została zaimplementowana z wykorzystaniem frameworka Spring Boot, który ułatwia tworzenie aplikacji webowych w języku Java. Projekt został zorganizowany zgodnie z architekturą MVC (Model-View-Controller), co zapewnia czytelność i separację odpowiedzialności w kodzie.

1.5.1 Stos technologiczny

- **Backend:** Java 21, Spring Boot, Spring Data JPA, Spring Security
- **Frontend:** HTML, CSS, Thymeleaf
- **Baza danych:** PostgreSQL, Redis 7.4.2-alpine
- **Zarządzanie projektem:** Maven

1.5.2 Widoki i formularze

Do tworzenia warstwy prezentacji wykorzystano szablony Thymeleaf, które integrują się bezpośrednio ze Spring Boot. Formularze użytkownika przesyłają dane do kontrolerów, gdzie są one poddawane walidacji. W przypadku błędów formularz zostaje ponownie załadowany z zachowaniem wcześniej wpisanych danych oraz komunikatami o nieprawidłowościach.

1.5.3 Obsługa błędów

System posiada mechanizmy obsługi wyjątków oraz walidacji danych wejściowych przy użyciu adnotacji takich jak `@Valid`, `@NotBlank`, czy `@Email`. Komunikaty o błędach wyświetlane są bezpośrednio w interfejsie użytkownika. Dodatkowo, zostały zaimplementowane customowe walidatory do loginu i hasła, umożliwiające szczegółową kontrolę nad poprawnością tych danych zgodnie z przyjętymi regułami biznesowymi.

1.5.4 Przykładowe listingu kodu w języku programowania Java

```
1 @Component
2 @Profile({"dev", "prod"})
3 public class DataSeeder implements CommandLineRunner {
4
5     private final CarModelService carModelService;
6     private final Faker faker = new Faker();
7
8     private static final String[] CAR_BRANDS = {
9         "Toyota", "BMW", "Mercedes", "Ford", "Audi", "Volkswagen", "Honda",
10        "Hyundai"
11    };
12
13     private static final String[] CAR_MODELS = {
14         "Mondeo", "Corolla", "Stilo", "911", "Passat", "Ceed",
15         "Sander", "Ibiza", "i30", "E47", "A4", "3008"
16    };
17 }
```

```

16
17 @Override
18 public void run(String... args) throws Exception {
19     List<CarModel> models = generateCarModels(15);
20     // ...
21 }
22
23 private List<CarModel> generateCarModels(int quantity) {
24     List<CarModel> models = carModelService.getAllCarModels();
25     if (models.size() < 3) {
26         for (int i = 0; i < quantity; i++) {
27             CarModel carModel = new CarModel();
28             carModel.setModelName(faker.options().option(CAR_MODELS));
29             carModel.setBrand(faker.options().option(CAR_BRANDS));
30             carModel.setCarType(faker.options().option(CarType.values()));
31             carModel.setFuelType(faker.options().option(FuelType.values()));
32             carModel.setTransmission(faker.options().option(Transmission.values()));
33             carModel.setDoors(faker.number().numberBetween(2, 6));
34             carModel.setSeats(faker.number().numberBetween(2, 9));
35
36             models.add(carModel);
37         }
38         carModelService.saveAll(models);
39     }
40     return models;
41 }
42 }

```

Listing 1: Fragment klasy DataSeeder – generowanie modeli samochodów

```

1 public class PaginationUtils {
2     public static int[] getPageNumbers(int currentPage, int visiblePages,
3         int totalPages) {
4         currentPage = Math.max(1, currentPage);
5         totalPages = Math.max(1, totalPages);
6         visiblePages = Math.max(1, visiblePages);
7
8         int radius = visiblePages / 2;
9
10        int startPage = Math.max(1, currentPage - radius);
11        int endPage = Math.min(totalPages, startPage + visiblePages - 1);
12
13        if (endPage - startPage + 1 < visiblePages) {
14            startPage = Math.max(1, endPage - visiblePages + 1);
15        }
16
17        int[] pages = new int[Math.min(visiblePages, totalPages)];
18        for (int i = 0; i < pages.length; i++) {
19            pages[i] = startPage + i;
20        }
21    }
22 }

```

```

20
21 return pages;
22 }
23 }

```

Listing 2: Klasa PaginationUtils - oblicza numery stron do wyświetlenia w paginacji

```

1
2 public class SportCar extends Vehicle {
3
4     @Min(value = 50, message = "{Min.horsePower}")
5     @Max(value = 1000, message = "{Max.horsePower}")
6     private Integer horsePower;
7
8     @Positive(message = "{Positive.acceleration}")
9     private Double acceleration; // 0-100 km/h in seconds
10
11     @Min(value = 100, message = "{Min.topSpeed}")
12     @Max(value = 400, message = "{Max.topSpeed}")
13     private Integer topSpeed;
14 }
15

```

Listing 3: Klasa dziedzicząca SportCar

2 Interfejs użytkownika

2.1 Ogólna charakterystyka

Interfejs użytkownika stanowi warstwę wizualną aplikacji, umożliwiającą użytkownikom końcowym oraz administratorom wygodną i intuicyjną interakcję z systemem. Głównym celem projektowym było zapewnienie prostoty obsługi, przejrzystości układu oraz dostępności funkcji przy minimalnym nakładzie poznawczym użytkownika.

Do budowy interfejsu wykorzystano silnik szablonów **Thymeleaf**, który ściśle integruje się z technologią **Spring Boot**, umożliwiając dynamiczne generowanie widoków w oparciu o dane przekazywane z kontrolerów. Komponenty interfejsu zostały opracowane z użyciem **HTML5**, **CSS3** oraz frameworka **Bootstrap**, co gwarantuje ich poprawne wyświetlanie na różnych urządzeniach i rozdzielczościach ekranu (tzw. responsywność).

System rozróżnia dwa podstawowe poziomy interfejsu:

- **Widok ogólnodostępny** — dostępny dla wszystkich użytkowników, obejmujący m.in. stronę główną, formularze rejestracji, logowania oraz dostęp do podstawowych zasobów systemu.
- **Panel administracyjny** — dostępny wyłącznie po autoryzacji, umożliwiający zarządzanie użytkownikami, zasobami oraz wgląd w dane systemowe.

Interfejs został zaprojektowany zgodnie z podstawowymi zasadami **User Experience (UX)**, takimi jak spójność nawigacji, czytelność, minimalizm oraz szybka reakcja systemu na akcje użytkownika. Wszystkie formularze w systemie są walidowane zarówno po stronie klienta, jak i serwera,

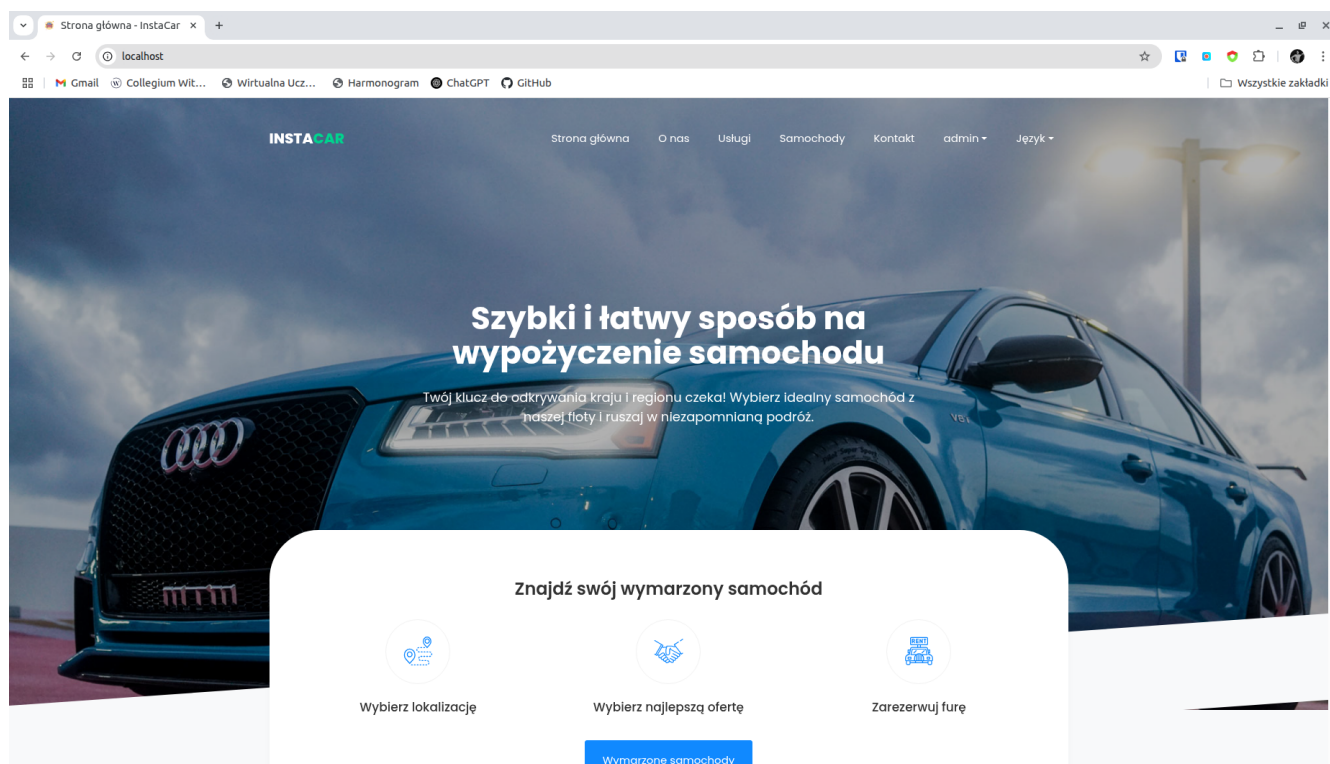
a w przypadku błędnych danych użytkownik otrzymuje czytelne komunikaty o błędach bez utraty dotychczas wprowadzonych informacji.

2.2 Widok użytkownika końcowego

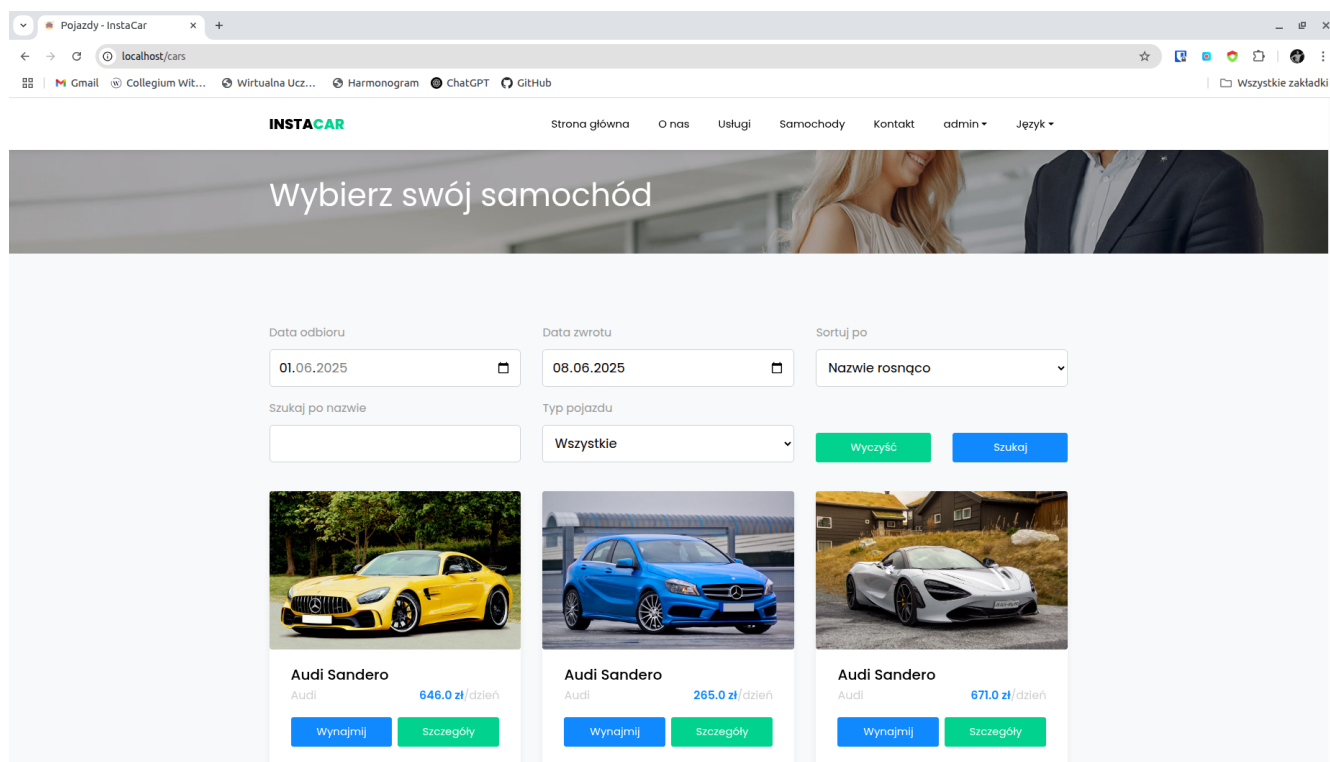
Widok użytkownika końcowego został zaprojektowany z myślą o łatwej i intuicyjnej nawigacji. Na górze każdej strony znajduje się pasek nawigacyjny, który umożliwia szybki dostęp do głównych sekcji serwisu. Użytkownik ma do wyboru następujące opcje:

- **Strona główna** — zawiera ogólne informacje o serwisie oraz skróty do najważniejszych funkcji.
- **O nas** — sekcja prezentująca informacje o firmie, jej historii, misji i wartościach.
- **Usługi** — przegląd dostępnych usług oferowanych przez firmę, takich jak wypożyczanie samochodów czy rezerwacje online.
- **Samochody** — katalog dostępnych pojazdów wraz ze szczegółowymi informacjami i możliwością rezerwacji.
- **Kontakt** — formularz kontaktowy oraz dane teleadresowe firmy.
- **Admin** — odnośnik do panelu logowania administratora; widoczny jest również dla użytkownika końcowego, jednak dostęp do funkcji administracyjnych wymaga autoryzacji.
- **Język** — możliwość zmiany wersji językowej interfejsu.

Interfejs użytkownika końcowego jest w pełni responsywny i przystosowany do obsługi zarówno na komputerach stacjonarnych, jak i na urządzeniach mobilnych. Dzięki zastosowaniu szablonów **Thymeleaf** możliwe jest dynamiczne wczytywanie treści oraz integracja danych pochodzących z kontrolerów aplikacji Spring Boot. Nawigacja między podstronami jest płynna, a układ graficzny spójny w całej aplikacji.



Rysunek 5: Widok strony głównej



Rysunek 6: Widok wyboru samochodu

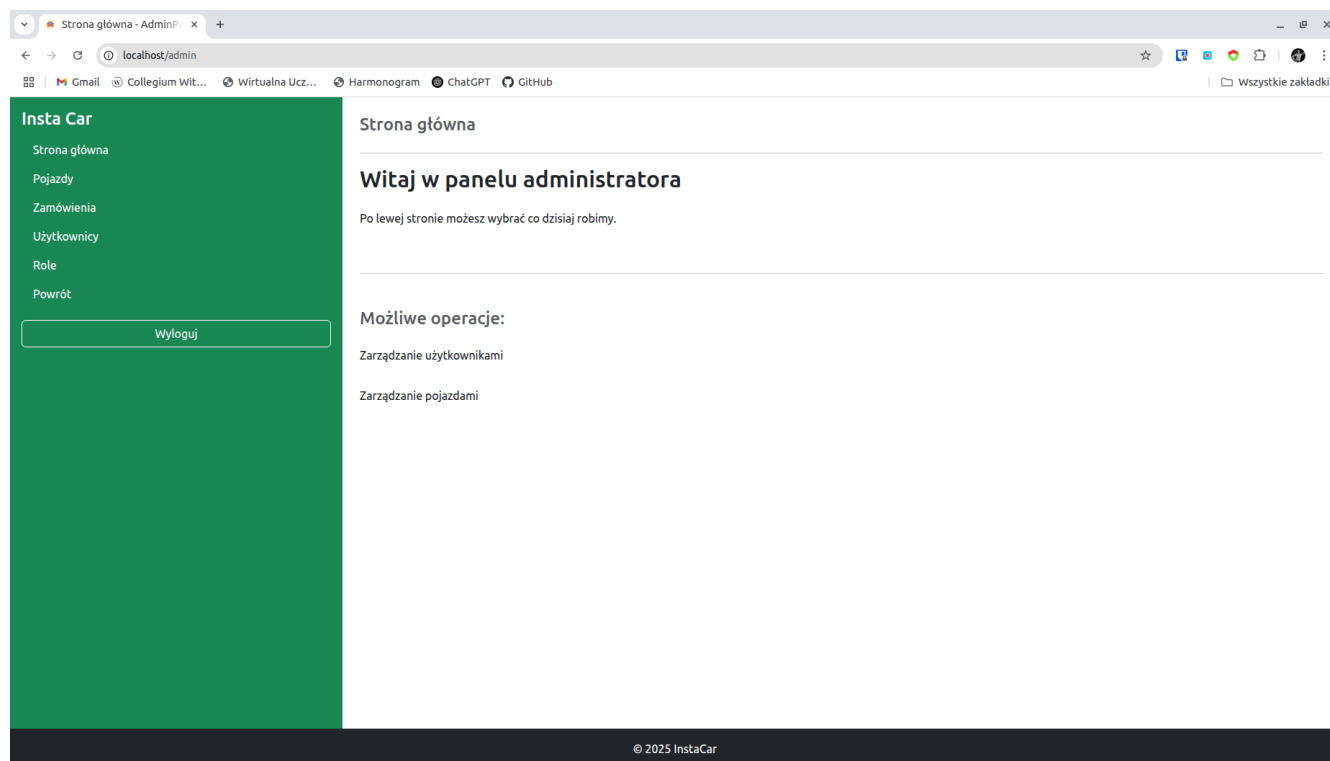
2.3 Widok administratora

Interfejs administratora dostępny jest po zalogowaniu się użytkownika z odpowiednimi uprawnieniami. Układ strony został podzielony na dwie główne części: panel nawigacyjny znajdujący się po lewej stronie oraz obszar roboczy po prawej.

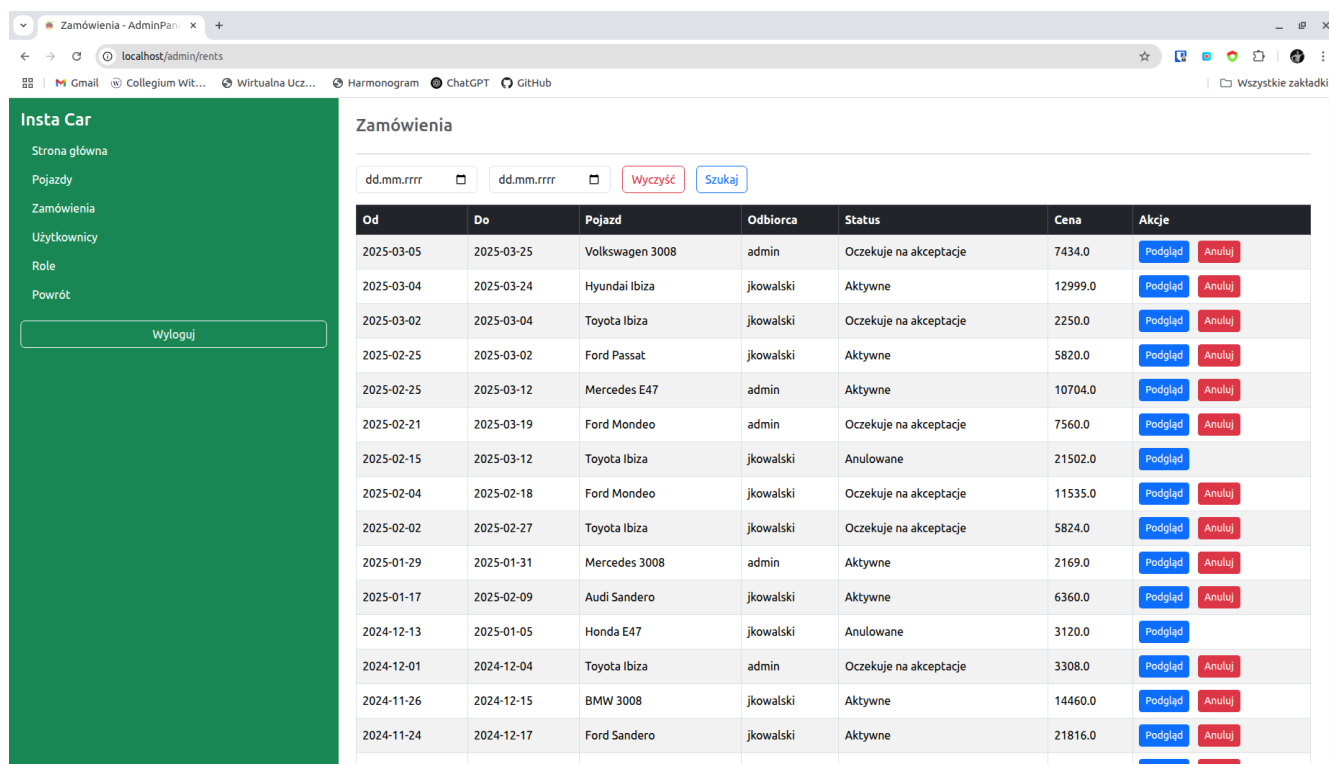
Panel boczny umożliwia szybki dostęp do kluczowych funkcjonalności systemu:

- **Strona główna** – ekran powitalny z podstawowymi informacjami,
- **Pojazdy** – zarządzanie bazą samochodów,
- **Zamówienia** – przegląd i edycja rezerwacji,
- **Użytkownicy** – zarządzanie kontami użytkowników,
- **Role** – wyświetlanie dostępnych ról i liczby przypisanych do nich użytkowników,
- **Powrót** – przejście do widoku użytkownika końcowego,
- **Wyloguj** – zakończenie sesji i wylogowanie z systemu.

W prawej części ekranu wyświetlana jest zawartość wybranej sekcji, umożliwiającą administratorowi wykonywanie odpowiednich operacji zarządczych.



Rysunek 7: Panel administracyjny – widok główny



Rysunek 8: Panel administracyjny – widok zamówień

2.4 Styl wizualny i doświadczenie użytkownika (UX)

Projekt interfejsu użytkownika aplikacji został przygotowany z myślą o prostocie, przejrzystości oraz łatwości nawigacji. Główne założenia dotyczące stylu wizualnego oraz UX obejmują:

- **Minimalistyczny design** – zredukowana ilość elementów dekoracyjnych na rzecz funkcjonalności i czytelności treści.
- **Konsekwentna kolorystyka** – dominującym kolorem jest zieleń, stosowana w panelach bocznych, przyciskach i nagłówkach. Kolor ten jest używany także do wyróżniania aktywnych elementów nawigacyjnych.
- **Responsywny układ** – interfejs został zaprojektowany w sposób umożliwiający wygodne korzystanie zarówno na komputerach, jak i na urządzeniach mobilnych.
- **Intuicyjna nawigacja** – wszystkie kluczowe sekcje aplikacji są dostępne z poziomu głównego menu, dzięki czemu użytkownik zawsze wie, gdzie się znajduje i jak wrócić do wcześniejszych ekranów.
- **Powiadomienia i walidacja** – użytkownik jest na bieżąco informowany o wynikach swoich akcji (np. wysyłanie formularzy, błędy walidacyjne), co zwiększa poczucie kontroli nad systemem.
- **Zachowanie danych w formularzach** – w przypadku błędów formularzowych dane wprowadzone przez użytkownika nie są tracone, co znacząco poprawia komfort korzystania z aplikacji.

Dzięki powyższym rozwiązaniom interfejs jest przyjazny dla użytkownika końcowego, a zarazem funkcjonalny i efektywny z punktu widzenia administratora systemu.

3 Dokumentacja technologiczna

3.1 Struktura bazy danych

Baza danych aplikacji do wynajmu samochodów została zaprojektowana tak, aby efektywnie zarządzać wszystkimi kluczowymi aspektami systemu – od danych użytkowników i pojazdów, po szczegóły transakcji wynajmu. Poniżej przedstawiono przegląd głównych tabel i ich wzajemnych relacji:

3.1.1 flyway_schema_history

Ta tabela służy do śledzenia i zarządzania **wersjami schematu bazy danych** za pomocą narzędzia Flyway. Zawiera metadane o każdej zastosowanej migracji, takie jak wersja, opis, skrypt i status wykonania.

3.1.2 roles

Definiuje **role użytkowników** w systemie (np. ADMIN, USER), co umożliwia zarządzanie uprawnieniami i dostępem.

3.1.3 app_users

Centralna tabela przechowująca **informacje o kontach użytkowników**, w tym dane uwierzytelniające (nazwa użytkownika, e-mail, hasło) oraz odniesienie do ich roli w systemie. Obsługuje również różne metody uwierzytelniania, takie jak lokalne i przez zewnętrznych dostawców (np. Google).

3.1.4 email_tokens i password_tokens

Te tabele są używane do zarządzania **tokenami weryfikacyjnymi** (dla adresów e-mail) oraz **tokenami do resetowania hasła**. Przechowują tokeny, ich status użycia i daty wygaśnięcia, zapewniając bezpieczeństwo procesów związanych z kontem.

3.1.5 car_models

Zawiera ogólne **dane techniczne modeli samochodów**, takie jak marka, nazwa modelu, typ nadwozia, liczba miejsc, drzwi, rodzaj paliwa i skrzynia biegów. Jest to słownik modeli, na podstawie których później tworzone są konkretne pojazdy.

3.1.6 vehicles

Ta tabela przechowuje **szczegółowe informacje o poszczególnych pojazdach** dostępnych do wynajęcia. Każdy wpis odwołuje się do **car_models** i zawiera unikalne dane, takie jak numer rejestracyjny, VIN, rok produkcji, kolor, cena wynajmu oraz adres URL do zdjęcia. Jest to tabela nadrzędna dla bardziej specyficznych typów pojazdów.

3.1.7 city_cars i sport_cars

Są to tabele dziedziczące po **vehicles**, przechowujące **dodatkowe atrybuty specyficzne dla danego typu pojazdu**. Na przykład **city_cars** zawiera pojemność bagażnika, a **sport_cars** moc silnika, przyspieszenie i prędkość maksymalną.

3.1.8 user_details

Rozszerza informacje o użytkownikach, przechowując **szczegóły osobiste i kontaktowe**, takie jak imię, nazwisko, numer telefonu i adres.

3.1.9 cities

Lista **miast**, w których możliwy jest odbiór lub zwrot pojazdów.

3.1.10 rents

Kluczowa tabela do zarządzania **transakcjami wynajmu**. Rejestruje, kto wynajął pojazd, który pojazd, daty wynajmu, całkowity koszt, miejsca odbioru i zwrotu, a także status wynajmu.

3.2 Opis środowiska

3.2.1 Baza danych - PostgreSQL 17.2

System bazodanowy wykorzystywany w projekcie to PostgreSQL w wersji 17.2. PostgreSQL to wydajny, stabilny i bezpieczny system zarządzania relacyjną bazą danych, szeroko stosowany w środowiskach produkcyjnych.

Konfiguracja dostępu do bazy danych znajduje się w pliku `docker-compose.yml`.

3.2.2 Migracje schematu – Flyway 10.20.1

Do zarządzania migracjami schematu bazy danych wykorzystywane jest narzędzie Flyway w wersji 10.20.1. Flyway umożliwia wersjonowanie zmian w strukturze bazy danych i ich automatyczne stosowanie w środowiskach deweloperskich, testowych oraz produkcyjnych.

Cechy użycia Flyway w projekcie:

- migracje definiowane w postaci skryptów SQL w katalogu `db/migration`,
- automatyczne wykrywanie i stosowanie nowych migracji przy starcie aplikacji,
- wsparcie dla rollbacków i walidacji historii migracji.

3.2.3 Cache – Redis 7.4.2-alpine

Do przechowywania danych tymczasowych oraz buforowania odpowiedzi system korzysta z Redis w wersji 7.4.2-alpine. Redis działa jako zewnętrzny, szybki magazyn danych typu key-value, wykorzystywany m.in. do cache'owania wyników zapytań i danych sesyjnych.

Wersja alpine została wybrana ze względu na minimalny rozmiar obrazu oraz szybki czas uruchamiania kontenera.

4 Zagadnienia kwalifikacyjne

4.1 Framework MVC

Projekt wykorzystuje framework Spring Boot, który implementuje wzorzec architektoniczny MVC (Model-View-Controller). Dzięki temu możliwe jest logiczne rozdzielenie warstwy danych (Model), interfejsu użytkownika (View) oraz logiki sterującej (Controller).

W projekcie zastosowano m.in. startery Spring Boot:

- spring-boot-starter-web – do tworzenia warstwy webowej, obsługującej żądania HTTP
- spring-boot-starter-thymeleaf – jako silnik szablonów do generowania widoków HTML
- spring-boot-starter-security – do zarządzania uwierzytelnianiem i autoryzacją
- spring-boot-starter-data-jpa – do integracji warstwy modelu z bazą danych poprzez JPA (ORM).

Struktura aplikacji jest zgodna z koncepcją MVC, gdzie kontrolery odbierają zapytania od użytkownika, komunikują się z modelem (np. warstwą usług i repozytoriów bazodanowych) i zwracają odpowiednie widoki generowane przez Thymeleaf. Takie podejście ułatwia utrzymanie, testowanie i rozwój aplikacji.

4.2 Framework CSS

W projekcie zastosowano framework CSS **Bootstrap** w wersji 4.5.0, co pozwoliło na szybkie i efektywne tworzenie responsywnego i estetycznego interfejsu użytkownika. Bootstrap dostarcza gotowe komponenty, siatkę (grid system) oraz zestaw klas ułatwiających stylizację elementów strony bez konieczności pisania dużej ilości własnego kodu CSS. Wykorzystanie frameworka znacząco przyspieszyło proces implementacji oraz zapewniło spójny wygląd aplikacji na różnych urządzeniach i rozdzielczościach.

4.3 Baza danych

Zagadnienie **baza danych** jest w pełni spełnione. Szczegółowy opis struktury bazy danych znajduje się w rozdziale [3 – Dokumentacja technologiczna](#).

4.4 Cache

Zagadnienie **cache** zostało w pełni zaimplementowane. W projekcie wykorzystano system Redis do buforowania danych i przyspieszania działania aplikacji. Szczegółowy opis konfiguracji oraz zastosowań pamięci podręcznej znajduje się w podrozdziale [3.2.3 - Cache – Redis 7.4.2-alpine](#).

4.5 Dependency manager

W projekcie zarządzanie zależnościami odbywa się za pomocą narzędzia Maven. Plik konfiguracyjny `pom.xml` zawiera wszystkie niezbędne zależności, takie jak Spring Boot, bazy danych (PostgreSQL, H2), biblioteki do bezpieczeństwa, szablonów Thymeleaf, obsługi maili, Redis oraz testów. Maven automatycznie pobiera i aktualizuje wymagane biblioteki, co ułatwia utrzymanie spójności

wersji oraz integrację z innymi narzędziami. Dzięki temu proces budowania aplikacji jest zautomatyzowany i powtarzalny, a także umożliwia łatwe zarządzanie wersjami i rozszerzeniami projektu.

4.6 HTML

Struktura dokumentów HTML w aplikacji została przygotowana z myślą o czytelności i łatwej nawigacji. W celu zapewnienia responsywności oraz estetycznego wyglądu, szablony HTML wykorzystują klasy i komponenty frameworka Bootstrap w wersji 4.5, co umożliwia szybkie i spójne tworzenie układów strony, formularzy, przycisków oraz innych elementów interfejsu użytkownika.

```
1 <html lang="pl">
2 <th:block th:replace="~{/fragments/headerFragments :: head("#{home.title}})"><
  /th:block>
3 <body>
4 <th:block th:replace="~{/fragments/bodyFragments :: navbar}"></th:block>
5
6 <div class="hero-wrap ftco-degree-bg" style="background-image: url('/images/
  bg_1.jpg');"
7 data-stellar-background-ratio="0.5">
8 <div class="overlay"></div>
9 <div class="container">
10 <div class="row no-gutters slider-text justify-content-start align-items-
  center justify-content-center">
11 <div class="col-lg-8 ftco-animate">
12 <div class="text w-100 text-center mb-md-5 pb-md-5">
13 <h1 class="mb-4" th:text="#{home.header}"></h1>
14 <p style="font-size: 18px;" th:text="#{home.subheader}"></p>
15 </div>
16 </div>
17 </div>
18 </div>
19 </div>
20
21 <section class="ftco-section ftco-no-pt bg-light">
22 <div class="container">
23 <div class="row no-gutters">
24 <div class="col-md-12 featured-top">
25 <div class="row no-gutters">
26 <div class="col-md-12 d-flex align-items-center">
27 <div class="services-wrap rounded-right rounded-left w-100 text-center">
28 <h3 class="heading-section mb-4 text-center" th:text="#{home.text.main}"></h3>
  >
29 <div class="row d-flex mb-4">
30 <div class="col-md-4 d-flex align-self-stretch ftco-animate">
31 <div class="services w-100 text-center">
32 <div class="icon d-flex align-items-center justify-content-center"><span
  class="flaticon-route"></span></div>
33 <div class="text w-100">
34 <h3 class="heading mb-2" th:text="#{home.text.location}"></h3>
35 </div>
36 </div>
37 </div>
38 <div class="col-md-4 d-flex align-self-stretch ftco-animate">
39 <div class="services w-100 text-center">
40 <div class="icon d-flex align-items-center justify-content-center"><span
```

```

41     class="flaticon-handshake"></span></div>
42 <div class="text w-100">
43 <h3 class="heading mb-2" th:text="#{home.text.offer}"></h3>
44 </div>
45 </div>
46 <div class="col-md-4 d-flex align-self-stretch ftco-animate">
47 <div class="services w-100 text-center">
48 <div class="icon d-flex align-items-center justify-content-center"><span
49   class="flaticon-rent"></span></div>
50 <div class="text w-100">
51 <h3 class="heading mb-2" th:text="#{home.text.rent}"></h3>
52 </div>
53 </div>
54 </div>
55 <p><a th:href="@{/cars}" class="btn btn-primary py-3 px-4" th:text="#{home.
56   submit}"></a></p>
57 </div>
58 </div>
59 </div>
60 </div>
61 </div>
62 </section>
63 <th:block th:replace="~/fragments/bodyFragments :: services"></th:block>
64 <th:block th:replace="~/fragments/bodyFragments :: footer"></th:block>
65 <th:block th:replace="~/fragments/bodyFragments :: loader"></th:block>
66 <th:block th:replace="~/fragments/bodyFragments :: scripts"></th:block>
67 </body>
68 </html>
69

```

Listing 4: Kod HTML strony głównej

4.7 CSS

Poza wykorzystaniem Bootstrapa, projekt zawiera również własne style CSS, które uzupełniają i modyfikują wygląd interfejsu zgodnie z wymaganiami funkcjonalnymi i wizualnymi aplikacji. Style te obejmują dostosowanie kolorów, marginesów, fontów oraz drobne korekty elementów, których nie obsługuje Bootstrap domyślnie. Własny kod CSS jest dobrze zorganizowany i umieszczony w osobnych plikach, co ułatwia jego utrzymanie i rozwój projektu.

```

1  html {
2    font-family: sans-serif;
3    line-height: 1.15;
4    -webkit-text-size-adjust: 100%;
5    -webkit-tap-highlight-color: rgba(0, 0, 0, 0); }
6
7  article, aside, figcaption, figure, footer, header, hgroup, main, nav,
8    section {
9    display: block; }
10
11  body {
12    margin: 0;

```

```

12 font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, "
    Helvetica Neue", Arial, "Noto Sans", sans-serif, "Apple Color Emoji", "Segoe
    UI Emoji", "Segoe UI Symbol", "Noto Color Emoji";
13 font-size: 1rem;
14 font-weight: 400;
15 line-height: 1.5;
16 color: #212529;
17 text-align: left;
18 background-color: #fff; }
19

```

Listing 5: Fragment kodu CSS

4.8 JavaScript

W projekcie JavaScript został wykorzystany głównie do obsługi interakcji na stronie frontendowej oraz działania komponentów z frameworka Bootstrap. Nie zastosowano dedykowanych frameworków JavaScript ani rozbudowanych skryptów. Implementacja opiera się na standardowych rozwiązaniach, które wspierają responsywność i podstawową dynamikę aplikacji.

```

1  var isMobile = {
2    Android: function() {
3      return navigator.userAgent.match(/Android/i);
4    },
5    BlackBerry: function() {
6      return navigator.userAgent.match(/BlackBerry/i);
7    },
8    iOS: function() {
9      return navigator.userAgent.match(/iPhone|iPad|iPod/i);
10   },
11   Opera: function() {
12     return navigator.userAgent.match(/Opera Mini/i);
13   },
14   Windows: function() {
15     return navigator.userAgent.match(/IEMobile/i);
16   },
17   any: function() {
18     return (
19       isMobile.Android() ||
20       isMobile.BlackBerry() ||
21       isMobile.iOS() ||
22       isMobile.Opera() ||
23       isMobile.Windows()
24     );
25   }
26 };
27

```

Listing 6: Fragment kodu JavaScript

4.9 Routing

Routing w projekcie realizowany jest za pomocą frameworka Spring Boot. Żądania HTTP są mapowane na odpowiednie metody kontrolerów przy użyciu adnotacji takich jak `@GetMapping` czy

@PostMapping. Dzięki temu możliwe jest czytelne i efektywne zarządzanie ścieżkami URL oraz obsługa różnych metod HTTP.

4.10 ORM

Zagadnienie **ORM** zostało zrealizowane przy użyciu warstwy mapowania obiektowo-relacyjnego, która pozwala na bezpośrednie odwzorowanie klas w aplikacji na tabele w bazie danych. Dzięki temu możliwa jest wygodna i bezpieczna manipulacja danymi bez konieczności pisania zapytań SQL. ORM obsługuje relacje między encjami, dziedziczenie oraz walidację danych.

4.11 Uwierzytelnienie

Zagadnienie **uwierzytelnienie** zostało w pełni zaimplementowane. System umożliwia logowanie lokalne oraz za pomocą zewnętrznych dostawców (np. Google). Dane uwierzytelniające są przechowywane w tabeli `app_users`, a dodatkowe mechanizmy bezpieczeństwa, takie jak tokeny resetowania hasła i potwierdzania adresu e-mail, zostały zaimplementowane z użyciem tabel `email_tokens` i `password_tokens`.

4.12 Lokalizacja

W projekcie zaimplementowano mechanizm lokalizacji umożliwiający wybór języka polskiego lub angielskiego. Teksty w aplikacji są przechowywane w plikach `messages_en.properties` oraz `messages_pl.properties`, które odpowiadają poszczególnym wersjom językowym. Dzięki temu aplikacja dynamicznie dostosowuje się do preferencji użytkownika, wyświetlając treści w wybranym języku.

4.13 Mailing

Zagadnienie **mailing** zostało zrealizowane poprzez wysyłanie wiadomości e-mail służących m.in. do weryfikacji konta i resetowania hasła. System generuje unikalne tokeny, które są przysyłane użytkownikom w wiadomościach zawierających odpowiednie linki aktywacyjne lub resetujące.

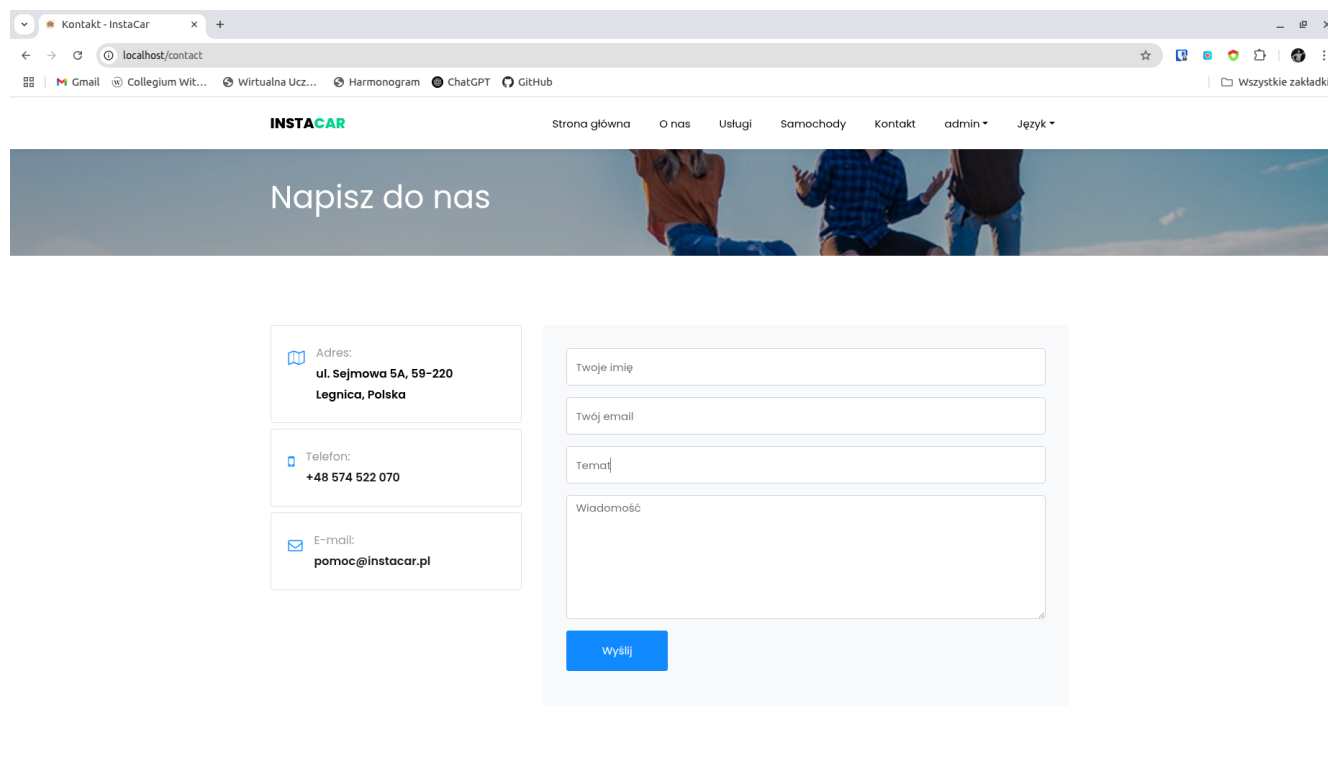
4.14 Formularze

W aplikacji formularze występują w różnych miejscach – zarówno po stronie użytkownika końcowego, jak i w panelu administracyjnym. Służą one do obsługi różnorodnych operacji, takich jak rejestracja, logowanie, składanie zamówień, dodawanie produktów, edycja danych i wiele innych funkcjonalności systemu.

Do tworzenia formularzy wykorzystano silnik szablonów Thymeleaf, który umożliwia łatwe powiązanie elementów formularza z danymi modelu oraz wspiera integrację z mechanizmem walidacji Spring Boota. Po wysłaniu danych formularzowych, są one przekazywane do odpowiedniego kontrolera w warstwie backendowej, gdzie poddawane są szczegółowej walidacji (zarówno po stronie klienta, jak i serwera).

W przypadku wykrycia błędów walidacyjnych (np. brak wymaganych pól, nieprawidłowy format danych), użytkownikowi wyświetlane są odpowiednie komunikaty o błędach, bez konieczności ponownego wypełniania całego formularza – dane wprowadzone przez użytkownika są zachowywane

i ponownie wyświetlane w formularzu. Dzięki temu rozwiązaniu użytkownik może szybko poprawić błędne dane, co znacząco wpływa na wygodę obsługi interfejsu i użyteczność aplikacji.

The image shows a web browser window with the address bar displaying 'localhost/contact'. The browser's tab is labeled 'Kontakt - InstaCar'. The website's header features the 'INSTACAR' logo on the left and a navigation menu with links: 'Strona główna', 'O nas', 'Usługi', 'Samochody', 'Kontakt', 'admin', and 'Język'. Below the header is a large banner with the text 'Napisz do nas' and a background image of people. The main content area contains a contact form. On the left, there are three boxes with contact information: 'Adres: ul. Sejmwowa 5A, 59-220 Legnica, Polska', 'Telefon: +48 574 522 070', and 'E-mail: pomoc@instacar.pl'. On the right, there is a larger form with fields for 'Twoje imię', 'Twój email', 'Temat', and a text area for 'Wiadomość'. A blue 'Wyślij' button is at the bottom of the right-hand form.

Rysunek 9: Formularz kontaktowy

4.15 Asynchroniczne interakcje

Zagadnienie **asynchronicznych interakcji** nie zostało zaimplementowane w obecnej wersji aplikacji. Wszystkie operacje realizowane są w sposób synchroniczny.

4.16 Konsumpcja API

Projekt wykorzystuje mechanizmy komunikacji z zewnętrznymi API, co umożliwia pobieranie i przetwarzanie danych spoza aplikacji. Dzięki temu system może integrować się z innymi serwisami i rozszerzać swoją funkcjonalność. W ramach konsumpcji API, aplikacja pobiera aktualne kursy walut bezpośrednio z Narodowego Banku Polskiego (NBP), co zapewnia dostęp do wiarygodnych i zawsze aktualnych danych.

4.17 Publikacja API

Aplikacja udostępnia własne API oparte na architekturze REST, które pozwala na zdalny dostęp do danych i funkcji systemu. Implementacja opiera się na frameworku Spring Boot, który zapewnia wygodne narzędzia do tworzenia i zabezpieczania endpointów. W ramach tego API, publikujemy dane dotyczące pojazdów oraz modeli samochodów, umożliwiając innym systemom łatwy dostęp i integrację z naszą bazą danych.

4.18 RWD

Interfejs aplikacji jest responsywny i dostosowuje się do różnych rozmiarów ekranów, zapewniając poprawne wyświetlanie i użyteczność na urządzeniach mobilnych, tabletach oraz komputerach stacjonarnych. W projekcie wykorzystano framework Bootstrap, który znacząco ułatwia tworzenie responsywnych layoutów.

4.19 Logger

W projekcie zastosowano mechanizmy logowania zdarzeń i błędów, które pomagają w monitorowaniu działania aplikacji oraz diagnozowaniu problemów. Logi są konfigurowane i zapisywane zgodnie z najlepszymi praktykami Spring Boot, co umożliwia łatwą analizę działania systemu.

4.20 Deployment

Zagadnienie **deployment** zostało zrealizowane z wykorzystaniem kontenerów Docker, co umożliwia łatwe wdrożenie aplikacji w różnych środowiskach. Cała konfiguracja usług (baza danych PostgreSQL, Redis, backend, frontend) znajduje się w pliku `docker-compose.yml`, co pozwala na szybkie uruchomienie całego systemu.

Spis rysunków

1	Diagram czynności	5
2	Diagram klas	6
3	Diagram encji bazy danych	7
4	Diagram przypadków użycia	8
5	Widok strony głównej	13
6	Widok wyboru samochodu	13
7	Panel administracyjny – widok główny	14
8	Panel administracyjny – widok zamówień	15
9	Formularz kontaktowy	23

Spis listingów

1	Fragment klasy DataSeeder – generowanie modeli samochodów	9
2	Klasa PaginationUtils - oblicza numery stron do wyświetlenia w paginacji	10
3	Klasa dziedzicząca SportCar	11
4	Kod HTML strony głównej	19
5	Fragment kodu CSS	20
6	Fragment kodu JavaScript	21