**Specyfikacja aplikacji do wynajmu samochodów**

**1. Wymagania systemowe**

**Backend:**

* **Język programowania**: C# (ASP.NET Core)
* **Baza danych**: Microsoft SQL Server
* **Wymagania**:
  + .NET 8 SDK
  + Serwer SQL (np. Microsoft SQL Server lub alternatywne rozwiazanie wspierające T-SQL)

**Frontend:**

* **Framework**: Blazor Server
* **Wymagania**:
  + .NET 8 SDK
  + Przeglądarka wspierająca nowoczesne standardy HTML5

**2. Instalacja i konfiguracja**

**Backend**

1. **Klonowanie repozytorium**:

git clone https://github.com/mikolajkuzdzal/RentingApp.git

1. **Przygotowanie bazy danych**:
   * Utwórz bazę danych w swoim systemie SQL (np. Microsoft SQL Server).
   * Zaimportuj plik test\_database.sql z repozytorium do swojej bazy danych.
2. **Konfiguracja połączenia z bazą danych**:
   * Otwórz plik appsettings.json.
   * Zaktualizuj sekcję ConnectionStrings:
   * "ConnectionStrings": {
   * "DefaultConnection": "Server=localhost;Database=RentingAppDB;Integrated Security=True;TrustServerCertificate=True;"

}

1. **Uruchomienie aplikacji**:
   * Otwórz projekt w Visual Studio.
   * Przywróć zależności za pomocą NuGet.
   * Uruchom aplikację za pomocą opcji Run lub F5.

**Frontend**

1. **Klonowanie repozytorium**:

git clone https://github.com/mikolajkuzdzal/RentingAppFrontend.git

1. **Konfiguracja połączenia z backendem**:
   * W pliku Program.cs zaktualizuj adres API backendu:

client.BaseAddress = new Uri("http://localhost:5076/api/");

1. **Uruchomienie aplikacji**:
   * Otwórz projekt w Visual Studio.
   * Przywróć zależności.
   * Uruchom aplikację za pomocą opcji Run lub F5.

**3. Testowi użytkownicy**

|  |  |  |
| --- | --- | --- |
| **Użytkownik** | **Hasło** | **Rola** |
| admin | admin123 | Admin |
| user | user123 | User |

**4. Funkcjonalności aplikacji z perspektywy użytkownika**

**Publiczne strony**

* **Strona główna** (/):
  + Informacje o aplikacji lub powitanie użytkownika.
* **Lista samochodów** (/car-versions):
  + Przegląd dostępnych samochodów.
  + Opcja rezerwacji samochodu (wymaga zalogowania).

**Logowanie i rejestracja**

* **Logowanie** (/login):
  + Umożliwia logowanie do aplikacji.
  + Obsługuje błędne dane logowania (wyświetla komunikat o błędzie).

**Panel użytkownika**

* **Koszyk** (/basket):
  + Wyświetla listę samochodów dodanych do koszyka.
  + Opcje:
    - Usuwanie pozycji z koszyka.
    - Finalizacja zakupu (przekierowanie na stronę podziękowania).

**Funkcjonalności administratora**

* **Zarządzanie samochodami** (/car-management):
  + Wyświetla listę wszystkich samochodów.
  + Opcje:
    - Dodawanie nowego samochodu.
    - Edycja istniejącego samochodu.
    - Usuwanie samochodu.

**Strony dodatkowe**

* **Podziękowania** (/thanks):
  + Wyświetlane po finalizacji zakupu.
  + Zawiera przycisk powrotu na stronę główną.
* **Counter** (/counter):
  + Prosty licznik testowy do demonstracji funkcji Blazora.
* **Prognoza pogody** (/fetchdata):
  + Przykładowa strona prezentująca integrację z serwisem pogodowym.

**5. Wewnętrzne usługi aplikacji**

**ApiService**

Obsługuje komunikację z backendem.

* Zarządzanie samochodami:
  + Pobieranie listy samochodów.
  + Dodawanie, edycja i usuwanie samochodów.
* Zarządzanie rezerwacjami:
  + Tworzenie, pobieranie i usuwanie rezerwacji.

**AuthService**

Obsługuje uwierzytelnianie i zarządzanie sesją użytkownika:

* Logowanie i wylogowanie.
* Obsługa koszyka użytkownika.
* Role użytkowników: Admin i User.

**Specyfikacja aplikacji do wynajmu samochodów**

**1. Wymagania systemowe**

**Backend:**

* **Język programowania**: C# (ASP.NET Core)
* **Baza danych**: Microsoft SQL Server
* **Wymagania**:
  + .NET 8 SDK
  + Serwer SQL (np. Microsoft SQL Server lub alternatywne rozwiazanie wspierające T-SQL)

**Frontend:**

* **Framework**: Blazor Server
* **Wymagania**:
  + .NET 8 SDK
  + Przeglądarka wspierająca nowoczesne standardy HTML5

**2. Instalacja i konfiguracja**

**Backend**

1. **Klonowanie repozytorium**:
2. git clone https://github.com/mikolajkuzdzal/RentingApp.git
3. **Przygotowanie bazy danych**:
   * Utwórz bazę danych w swoim systemie SQL (np. Microsoft SQL Server).
   * Zaimportuj plik test\_database.sql z repozytorium do swojej bazy danych.
4. **Konfiguracja połączenia z bazą danych**:
   * Otwórz plik appsettings.json.
   * Zaktualizuj sekcję ConnectionStrings:
   * "ConnectionStrings": {
   * "DefaultConnection": "Server=localhost;Database=RentingAppDB;Integrated Security=True;TrustServerCertificate=True;"
   * }
5. **Uruchomienie aplikacji**:
   * Otwórz projekt w Visual Studio.
   * Przywróć zależności za pomocą NuGet.
   * Uruchom aplikację za pomocą opcji Run lub F5.

**Frontend**

1. **Klonowanie repozytorium**:
2. git clone https://github.com/mikolajkuzdzal/RentingAppFrontend.git
3. **Konfiguracja połączenia z backendem**:
   * W pliku Program.cs zaktualizuj adres API backendu:
   * client.BaseAddress = new Uri("http://localhost:5076/api/");
4. **Uruchomienie aplikacji**:
   * Otwórz projekt w Visual Studio.
   * Przywróć zależności.
   * Uruchom aplikację za pomocą opcji Run lub F5.

**3. Testowi użytkownicy**

| **Użytkownik** | **Hasło** | **Rola** |
| --- | --- | --- |
| admin | admin123 | Admin |
| user | user123 | User |

**4. Funkcjonalności aplikacji z perspektywy użytkownika**

**Publiczne strony**

* **Strona główna** (/):
  + Informacje o aplikacji lub powitanie użytkownika.
* **Lista samochodów** (/car-versions):
  + Przegląd dostępnych samochodów.
  + Opcja rezerwacji samochodu (wymaga zalogowania).

**Logowanie i rejestracja**

* **Logowanie** (/login):
  + Umożliwia logowanie do aplikacji.
  + Obsługuje błędne dane logowania (wyświetla komunikat o błędzie).

**Panel użytkownika**

* **Koszyk** (/basket):
  + Wyświetla listę samochodów dodanych do koszyka.
  + Opcje:
    - Usuwanie pozycji z koszyka.
    - Finalizacja zakupu (przekierowanie na stronę podziękowania).

**Funkcjonalności administratora**

* **Zarządzanie samochodami** (/car-management):
  + Wyświetla listę wszystkich samochodów.
  + Opcje:
    - Dodawanie nowego samochodu.
    - Edycja istniejącego samochodu.
    - Usuwanie samochodu.

**Strony dodatkowe**

* **Podziękowania** (/thanks):
  + Wyświetlane po finalizacji zakupu.
  + Zawiera przycisk powrotu na stronę główną.
* **Counter** (/counter):
  + Prosty licznik testowy do demonstracji funkcji Blazora.
* **Prognoza pogody** (/fetchdata):
  + Przykładowa strona prezentująca integrację z serwisem pogodowym.

**5. Struktura bazy danych**

**Tabela: CarVersions**

| **Kolumna** | **Typ danych** | **Uwagi** |
| --- | --- | --- |
| Id | int | Klucz główny, autoinkrementacja |
| Model | nvarchar(100) | Nazwa modelu samochodu |
| Brand | nvarchar(100) | Marka samochodu |
| PricePerDay | decimal(10, 2) | Cena wynajmu za dzień |

**Tabela: Reservations**

| **Kolumna** | **Typ danych** | **Uwagi** |
| --- | --- | --- |
| Id | int | Klucz główny, autoinkrementacja |
| UserId | int | Klucz obcy do tabeli Users |
| Type | nvarchar(50) | Typ rezerwacji |
| ItemId | int | Id przedmiotu (np. samochód) |
| StartDate | datetime | Data rozpoczecia rezerwacji |
| EndDate | datetime | Data zakończenia rezerwacji |

**Tabela: Users**

| **Kolumna** | **Typ danych** | **Uwagi** |
| --- | --- | --- |
| Id | int | Klucz główny, autoinkrementacja |
| Username | nvarchar(100) | Nazwa użytkownika |
| Email | nvarchar(100) | Adres email |
| Password | nvarchar(100) | Hasło |
| Role | nvarchar(50) | Rola użytkownika (Admin/User) |

**6. API Backend**

**Endpointy: CarVersionsController**

* **GET /api/carversions**: Pobiera listę wszystkich modeli samochodów.
* **GET /api/carversions/{id}**: Pobiera szczegóły danego modelu.
* **POST /api/carversions**: Tworzy nowy model samochodu.
* **PUT /api/carversions/{id}**: Aktualizuje dane istniejącego modelu.
* **DELETE /api/carversions/{id}**: Usuwa model samochodu.

**Endpointy: ReservationsController**

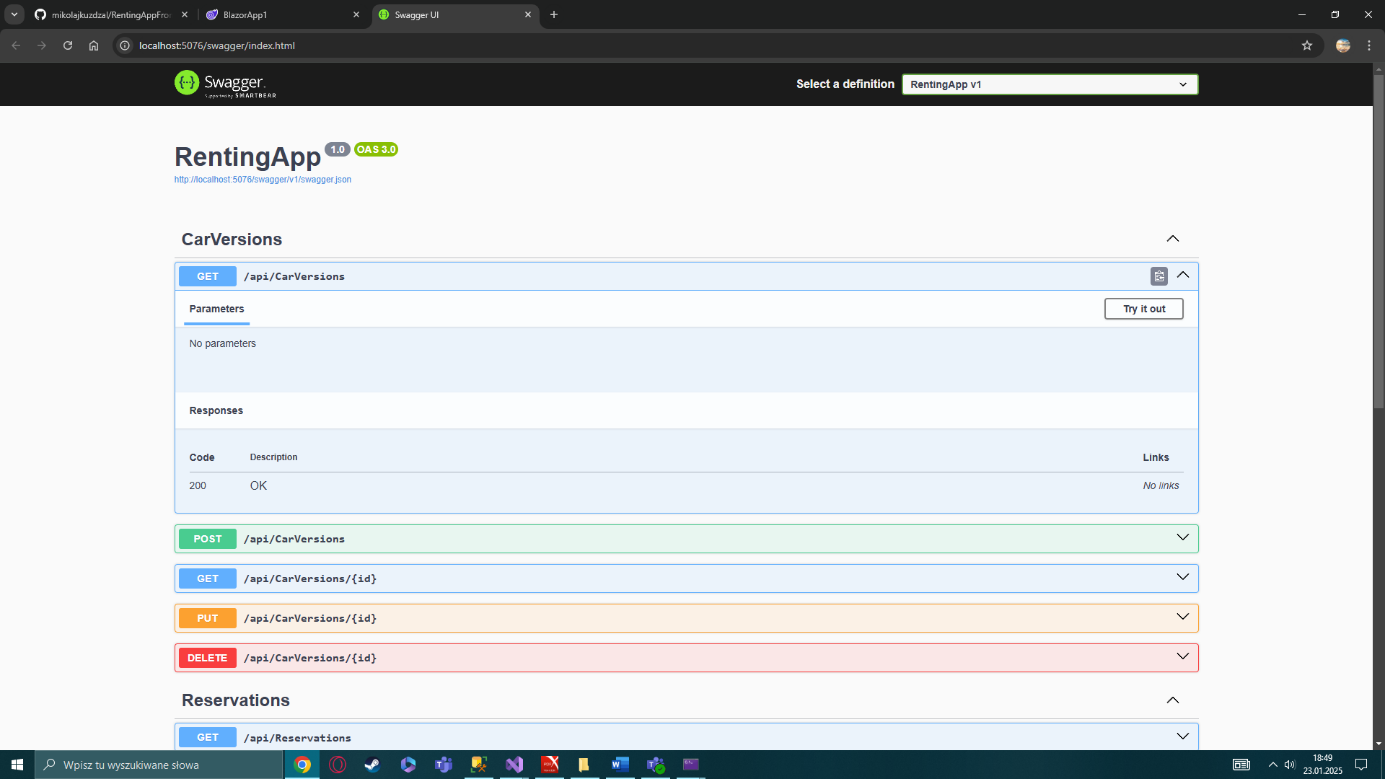
* **GET /api/reservations**: Pobiera listę wszystkich rezerwacji.
* **GET /api/reservations/{id}**: Pobiera szczegóły danej rezerwacji.
* **POST /api/reservations**: Tworzy nową rezerwację.
* **PUT /api/reservations/{id}**: Aktualizuje dane istniejącej rezerwacji.
* **DELETE /api/reservations/{id}**: Usuwa rezerwację.

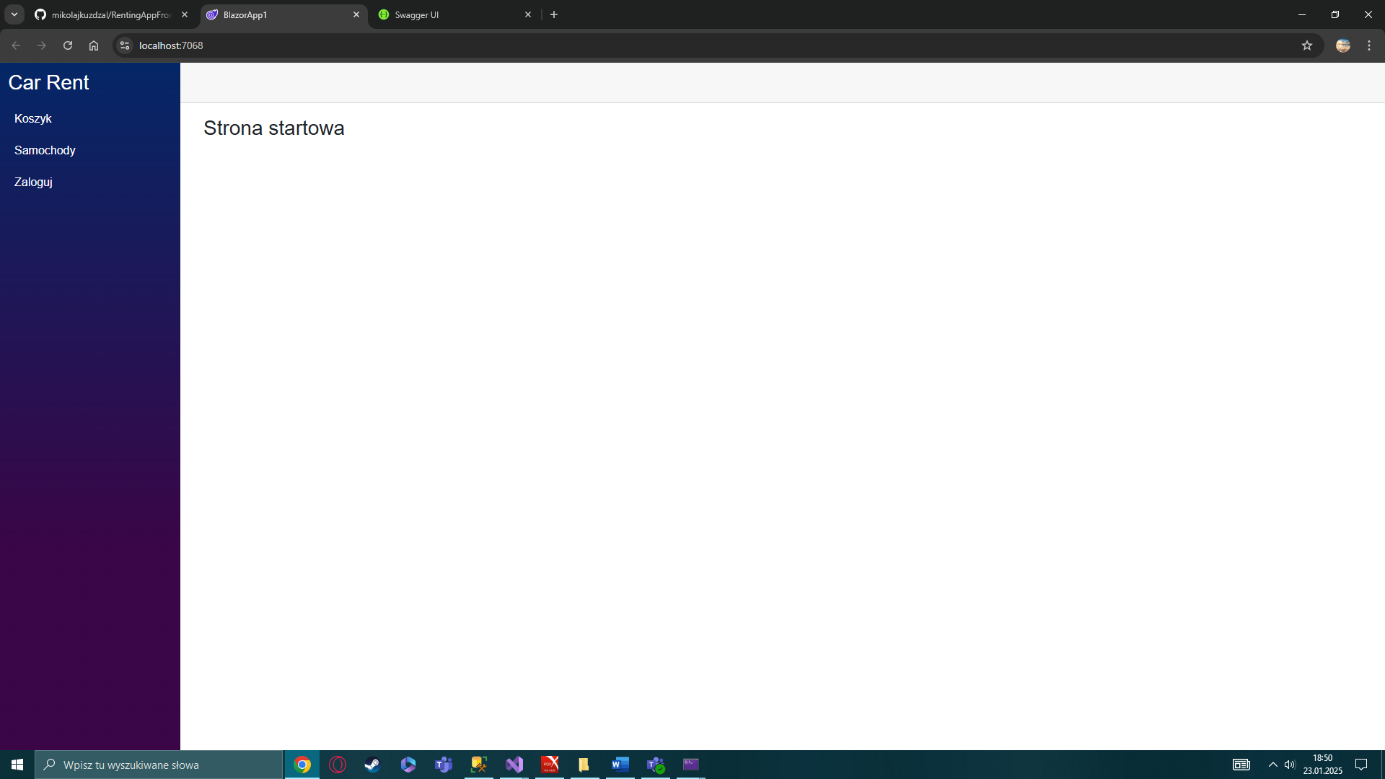
**Endpointy: UsersController**

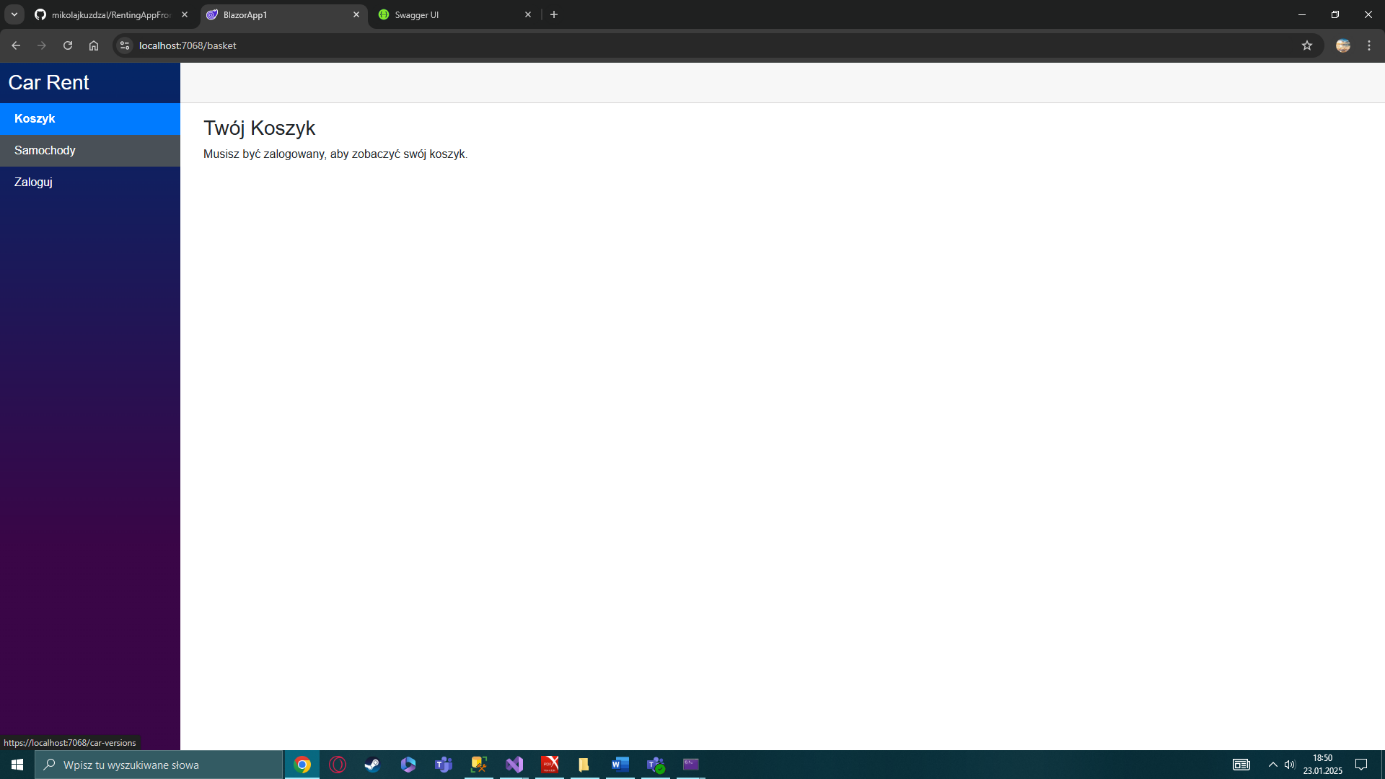
* **GET /api/users**: Pobiera listę użytkowników.
* **POST /api/users**: Tworzy nowego użytkownika.

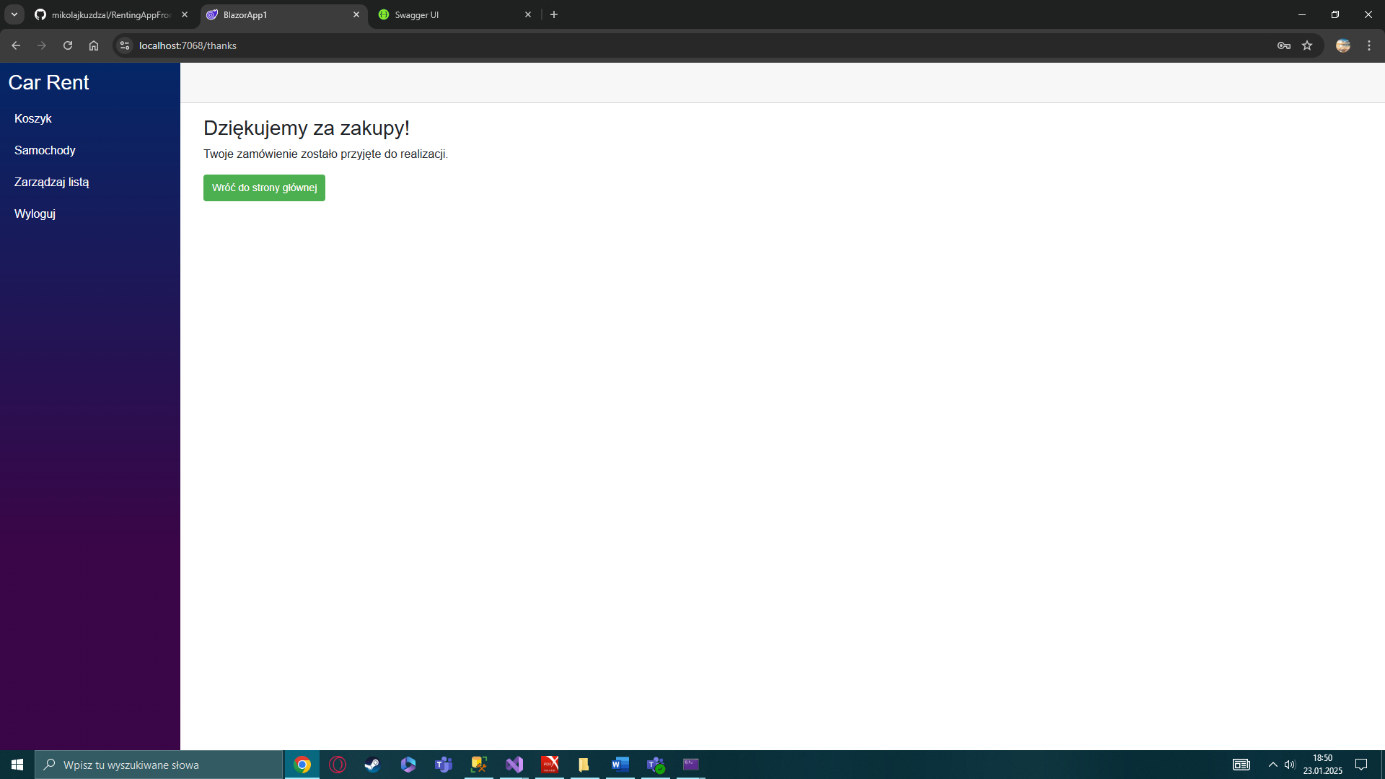
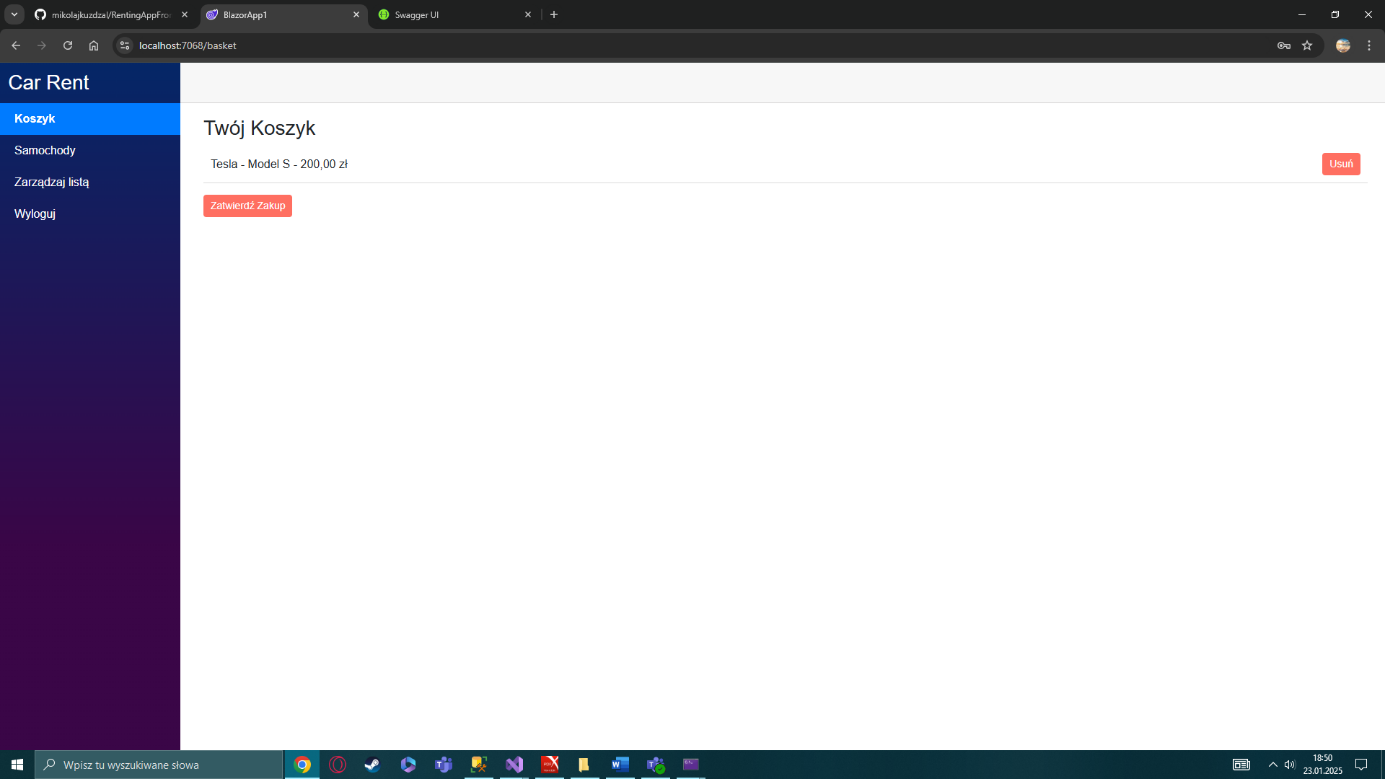
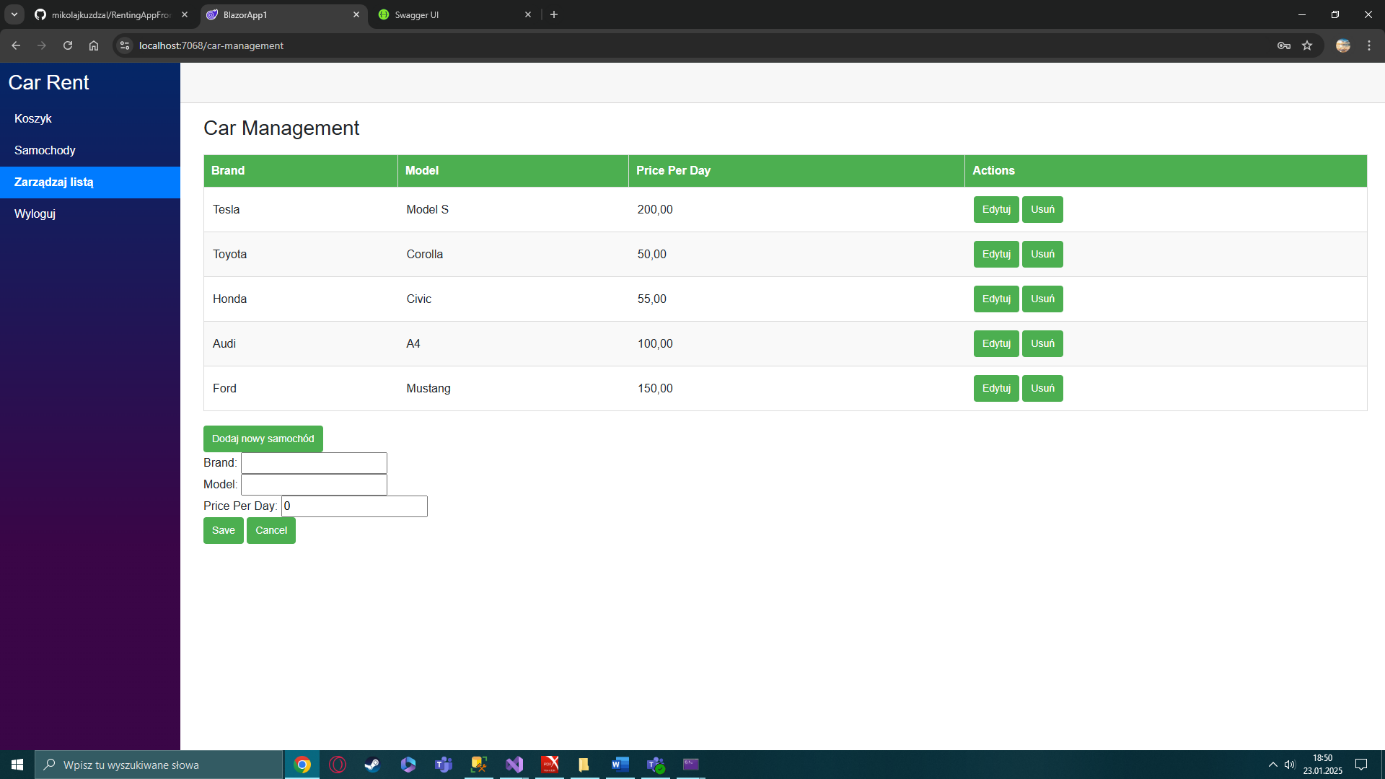
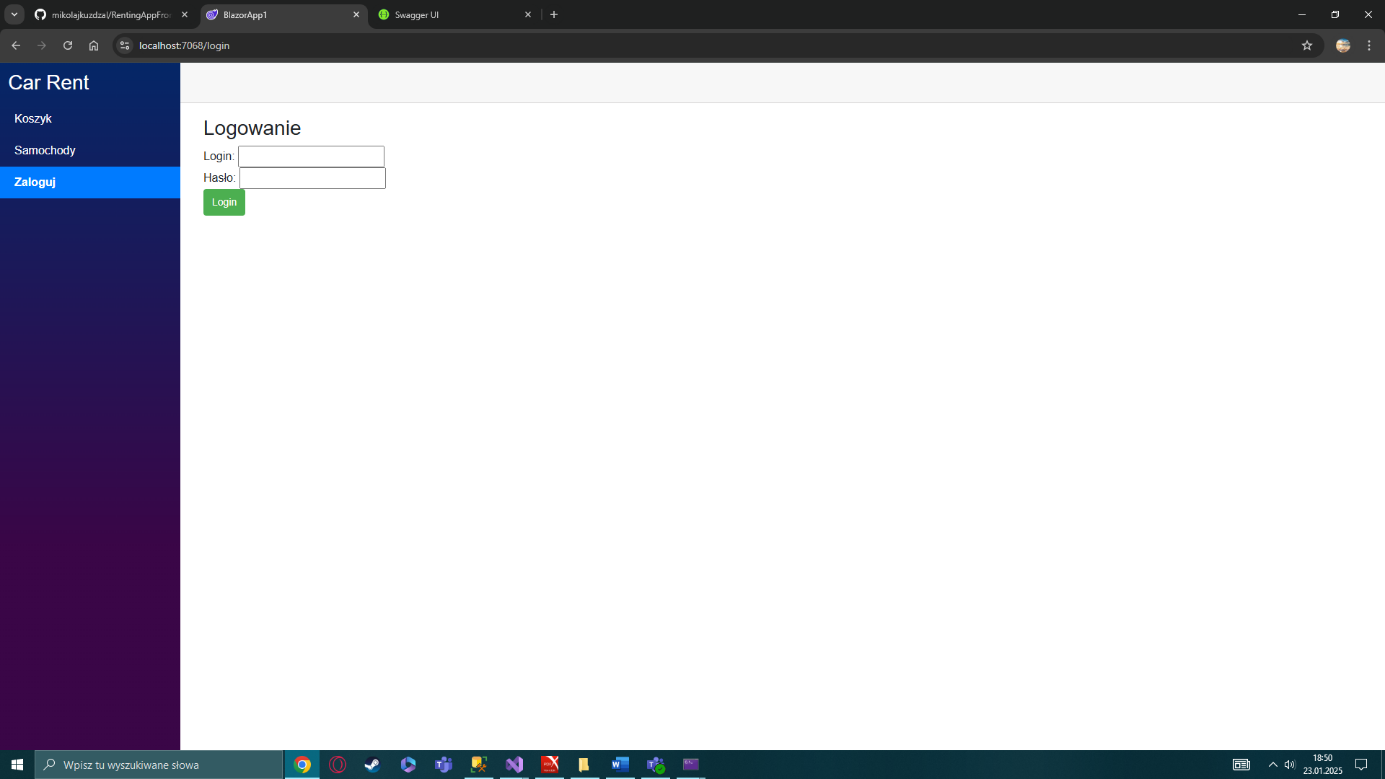
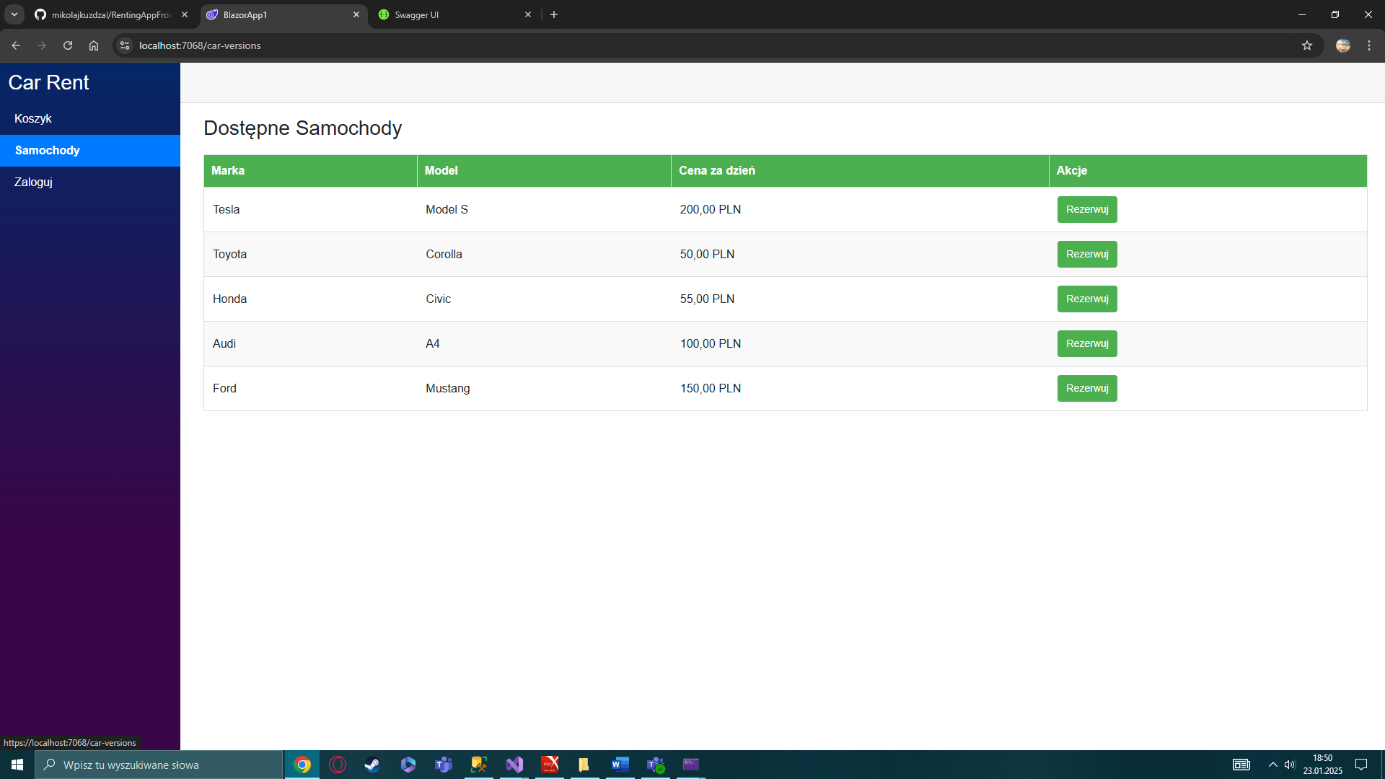
**7. Wewnętrzna logika backendu**

* **Walidacja danych**:
  + Sprawdzanie poprawności dat przy tworzeniu rezerwacji.
  + Weryfikacja istnienia powiązanych rekordów (np. użytkownika lub samochodu).
* **Powiadomienia i logowanie błędów**:
  + Rejestrowanie błędów za pomocą ILogger (np. w WeatherForecastController).
* **Zarządzanie rolami**:
  + Role są przypisywane użytkownikom i weryfikowane podczas logowania.









Dominik Kiepura 15469, Mikołaj Kużdżał 15162