Projekt aplikacji

Projekt: Aplikacja do wypożyczania samochodów (C#/XAML)

Data: 2025-05-27

Spis treści

[1. Wstęp 3](#_Toc201857268)

[2. Cel i zakres dokumentu 3](#_Toc201857269)

[3. Definicja architektury aplikacji 3](#_Toc201857270)

[4. Cele i ograniczenia architektury 3](#_Toc201857271)

[4.1. Cele architektury 3](#_Toc201857272)

[4.2. Ograniczenia architektury 3](#_Toc201857273)

[5. Obraz logiczny aplikacji 4](#_Toc201857274)

[5.1. Charakterystyka pakietów 4](#_Toc201857275)

[5.2. Diagram klas aplikacji 5](#_Toc201857276)

[5.3. Specyfikacja funkcji i metod aplikacji 5](#_Toc201857277)

[6. Dynamiczny obraz modelowanej aplikacji 6](#_Toc201857278)

[6.1. Diagram sekwencji UML dla obiektów 6](#_Toc201857279)

[6.2. Diagram aktywności UML dla obiektów 6](#_Toc201857280)

[7. Statyczny obraz modelowanej aplikacji 7](#_Toc201857282)

[7.1. Diagram komponentów UML 7](#_Toc201857283)

[7.2. Diagram instalacji UML 7](#_Toc201857284)

[8. Projekt bazy danych 7](#_Toc201857285)

[9. Operacje na danych: 8](#_Toc201857286)

# Wstęp

Niniejszy dokument przedstawia projekt systemu dla aplikacji do zarządzania wypożyczalnią samochodów. Celem jest umożliwienie pracownikom łatwego przeglądania, dodawania, edytowania i usuwania samochodów, jak również rejestrowania i zarządzania rezerwacjami pojazdów. Dokument opisuje architekturę, funkcjonalność, strukturę danych oraz dynamiczne i statyczne aspekty aplikacji.

# Cel i zakres dokumentu

Celem dokumentu jest dostarczenie szczegółowego opisu architektury, funkcjonalności i ograniczeń systemu „Wypożyczalnia samochodów”. Zakres obejmuje strukturę logiczną, charakterystykę modułów, diagramy UML, opis bazy danych oraz kluczowe operacje biznesowe systemu.

# Definicja architektury aplikacji

Architektura aplikacji została zaprojektowana jako wielowarstwowa aplikacja desktopowa oparta o język C# oraz WPF/XAML. System składa się z następujących warstw:  
- Warstwa interfejsu użytkownika (UI): odpowiada za prezentację danych i interakcję z użytkownikiem,  
- Warstwa logiki biznesowej: odpowiada za przetwarzanie danych oraz obsługę operacji na samochodach i rezerwacjach,  
- Warstwa dostępu do danych: realizuje komunikację z lokalną bazą danych lub API.  
  
Komunikacja pomiędzy warstwami odbywa się za pomocą wywołań metod i przekazywania danych w formie obiektów.

# Cele i ograniczenia architektury

## Cele architektury

- Łatwość obsługi: Interfejs intuicyjny, przejrzysty, przyjazny użytkownikowi.  
- Stabilność i wydajność: Szybkie działanie nawet przy dużej liczbie rekordów.  
- Bezpieczeństwo danych: Ograniczenie błędów i zapobieganie utracie danych (backup, komunikaty o błędach).  
- Rozszerzalność: Możliwość łatwej rozbudowy np. o integrację płatności czy inne funkcje.

## Ograniczenia architektury

- Technologiczne: System działa wyłącznie na Windows z .NET Framework.  
- Brak logowania: Każdy użytkownik ma te same uprawnienia.  
- Lokalność: Brak obsługi pracy przez sieć poza obsługą API/bazy.

# Obraz logiczny aplikacji

## Charakterystyka pakietów

-Models: Klasy opisujące główne obiekty systemu (Car, Rental).  
- Commands: Komendy do komunikacji z bazą/API (Database.cs).  
- Views: Okna i widoki aplikacji (MainWindow, AddCarView, RentalsView, itp.).  
- System/Configuration: Pliki konfiguracyjne i ustawienia środowiskowe.

A screenshot of a computer program

Description automatically generated

## Diagram klas aplikacji

A screenshot of a computer

Description automatically generatedKlasy:  
- Car: marka, model, VIN, ID  
- Rental: ID, CarId, CustomerName, RentalStartDate, RentalEndDate  
- Database: metody do obsługi CRUD (Create/Read/Update/Delete) dla samochodów i rezerwacji

## Specyfikacja funkcji i metod aplikacji

Car.cs  
 - Get/Set: Id, Brand, Model, VinNumber  
- Rental.cs  
 - Get/Set: Id, CarId, CustomerName, RentalStartDate, RentalEndDate, Car  
- Database.cs  
 - ObservableCollection<Car> GetData()  
 - void AddCarToDatabase(Car car)  
 - void EditCar(Car car)  
 - void DeleteCar(Car car)  
 - ObservableCollection<Car> GetAvailableCars()  
 - bool AddRentalToDatabase(Rental rental)  
 - ObservableCollection<Rental> GetRentals()  
 - void DeleteRental(Rental rental)  
 - bool UpdateRentalEndDate(int rentalId, DateTime newEndDate)

# Dynamiczny obraz modelowanej aplikacji

## Diagram sekwencji UML dla obiektów

A black screen with white text

Description automatically generated

## Diagram aktywności UML dla obiektów

# A group of white rectangular shapes Description automatically generated

# Statyczny obraz modelowanej aplikacji

## A screenshot of a computer screen Description automatically generatedDiagram komponentów UML

## Diagram instalacji UML

A black background with a black square

Description automatically generated with medium confidence

# Projekt bazy danych

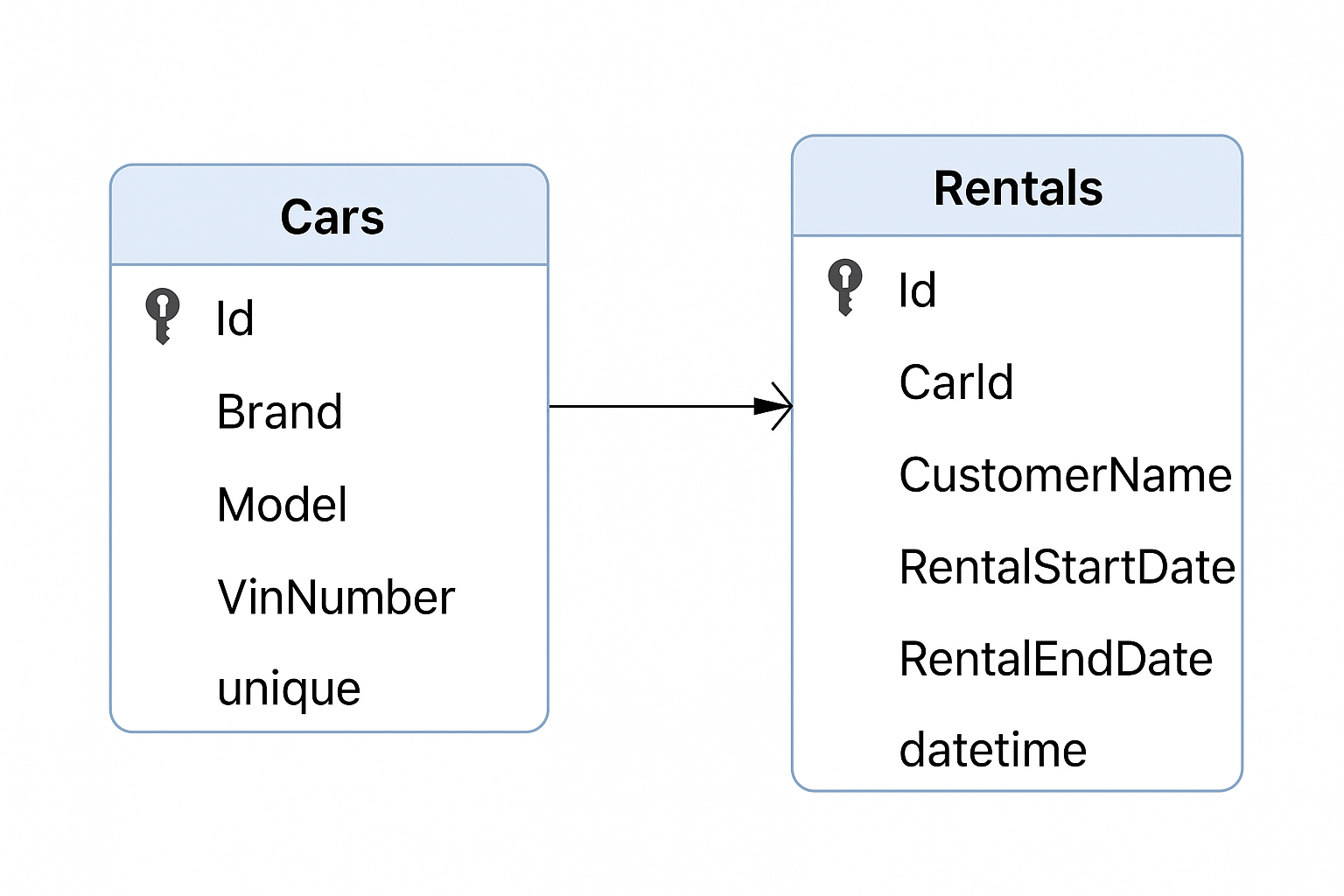


Tabela: Cars  
- Id (int, klucz główny)  
- Brand (string)  
- Model (string)  
- VinNumber (string, unikalny)  
  
Tabela: Rentals  
- Id (int, klucz główny)  
- CarId (int, klucz obcy do Cars)  
- CustomerName (string)  
- RentalStartDate (datetime)  
- RentalEndDate (datetime)

# Operacje na danych:

- Dodawanie, edycja, usuwanie samochodów  
- Dodawanie, edycja, usuwanie rezerwacji  
- Wyszukiwanie dostępnych aut  
- Walidacja unikalności VIN  
- Sprawdzenie dostępności samochodu w wybranym terminie