Opis wersji aplikacji

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

Data: 2025-06-25

Spis treści

[1. Wstęp 3](#_Toc130745239)

[2. Cel projektu 3](#_Toc130745240)

[3. Organizacja projektu 3](#_Toc130745241)

[4. Struktura organizacyjna 3](#_Toc130745242)

[5. Role i odpowiedzialność 3](#_Toc130745243)

[6. Harmonogram projektu 3](#_Toc130745244)

[7. Kosztorys projektu 3](#_Toc130745245)

[8. Zasoby projektu 3](#_Toc130745246)

[9. Standardy i narzędzia w projekcie 3](#_Toc130745247)

[10. Procesy zarządzania 3](#_Toc130745248)

[10.1 Plan zarządzania konfiguracją 3](#_Toc130745249)

[10.2 Plan zarządzania ryzykiem 3](#_Toc130745250)

[10.3 Plan zarządzania testami 3](#_Toc130745251)

# 1. Wstęp

# Niniejszy dokument opisuje plan projektu stworzenia aplikacji desktopowej wspomagającej pracę wypożyczalni samochodów. Projekt został zrealizowany w technologii WPF w języku C# z wykorzystaniem wzorca MVVM oraz komunikacją z API opartym o platformę .NET i Entity Framework Core. System pozwala na zarządzanie flotą pojazdów, wypożyczeniami oraz podstawowe operacje CRUD.

# 2. Cel projektu

Celem projektu jest stworzenie intuicyjnej i funkcjonalnej aplikacji dla pracowników wypożyczalni samochodów, która umożliwi:

* dodawanie, modyfikowanie i usuwanie pojazdów,
* zarządzanie wypożyczeniami,
* przeglądanie dostępnych aut,
* komunikację z bazą danych poprzez API RESTful.

System ma za zadanie uprościć codzienną obsługę klienta i zwiększyć efektywność procesów w firmie.

# 3. Organizacja projektu

Projekt będzie organizowany i zarządzany przez zespół specjalistów obejmujących wszystkie kluczowe obszary procesu wytwarzania oprogramowania. Zespół składa się z ról takich jak Business Analyst, Change Manager, Configuration Manager, Database Developer, Developer, Instalation Engineer, Project Manager, QA Manager, Risk Analyst, Risk Manager, System Analyst, System Architect, Test Designer, Test Engineer oraz Test Manager. Tak rozbudowana struktura umożliwia precyzyjne przypisanie zadań i odpowiedzialności, co przekłada się na wysoką jakość realizacji projektu oraz jego skuteczne zarządzanie.

# 4. Struktura organizacyjna

* Business Analyst – analizuje wymagania jakości i opracowuje specyfikacje funkcjonalne i niefunkcjonalne.
* Change Manager – zarządza zmianami projektowymi i kontroluje ich wpływ na harmonogram.
* Configuration Manager – odpowiada za kontrolę wersji i utrzymanie spójności środowiska projektowego.
* Database Developer / Developer – implementują logikę systemu oraz zarządzają strukturą bazy danych.
* Instalation Engineer – przygotowuje i konfiguruje środowiska uruchomieniowe aplikacji.
* Project Manager – nadzoruje przebieg projektu, harmonogram, budżet i współpracę zespołu.
* QA Manager – odpowiada za jakość produktu i wdraża standardy testowania.
* Risk Analyst – identyfikuje potencjalne zagrożenia techniczne i organizacyjne.
* Risk Manager – tworzy i wdraża strategie zarządzania ryzykiem.
* System Analyst – analizuje wymagania techniczne i procesy w systemie.
* System Architect – projektuje architekturę techniczną aplikacji.
* Test Designer – projektuje przypadki testowe.
* Test Engineer – realizuje testy aplikacji i raportuje wyniki.
* Test Manager – planuje, organizuje i kontroluje cały proces testowania.

# 5. Role i odpowiedzialność

# Business Analyst

# Obowiązki: przygotowanie dokumentacji wymagań, umów, akceptacja specyfikacji.

# Kompetencje: analiza biznesowa, komunikacja, modelowanie wymagań.

# Change Manager

# Obowiązki: zarządzanie żądaniami zmian i ich wpływem na projekt.

# Kompetencje: analiza ryzyka, decyzyjność, organizacja.

# Configuration Manager

# Obowiązki: opracowanie planu konfiguracji, kontrola wersji, definiowanie baseline.

# Kompetencje: znajomość Git, zarządzanie konfiguracją.

# Developer / Database Developer

# Obowiązki: kodowanie, dokumentacja techniczna, testy jednostkowe.

# Kompetencje: C#, SQL, WPF, debugowanie.

# Instalation Engineer

# Obowiązki: przygotowanie środowiska produkcyjnego i wdrażanie aplikacji.

# Kompetencje: znajomość systemów Windows, konfiguracja środowisk.

# Project Manager

# Obowiązki: planowanie, harmonogramowanie, kontrola projektu.

# Kompetencje: zarządzanie zespołem, organizacja pracy, nadzór.

# QA Manager

# Obowiązki: tworzenie strategii jakości, dokumentacja jakościowa.

# Kompetencje: znajomość procesów jakości, nadzór nad testami.

# Risk Analyst

# Obowiązki: analiza ryzyk i opracowanie raportu.

# Kompetencje: identyfikacja zagrożeń, ocena wpływu.

# Risk Manager

# Obowiązki: tworzenie i nadzorowanie realizacji planu zarządzania ryzykiem.

# Kompetencje: proaktywność, planowanie awaryjne.

# System Analyst

# Obowiązki: analiza wymagań, tworzenie modeli przypadków użycia.

# Kompetencje: modelowanie UML, analityka systemowa.

# System Architect

# Obowiązki: projektowanie architektury systemu, interfejsy techniczne.

# Kompetencje: wzorce projektowe, architektura systemowa.

# Test Designer

# Obowiązki: projektowanie przypadków testowych.

# Kompetencje: testowanie manualne i automatyczne.

# Test Engineer

# Obowiązki: przygotowanie i wykonanie testów funkcjonalnych.

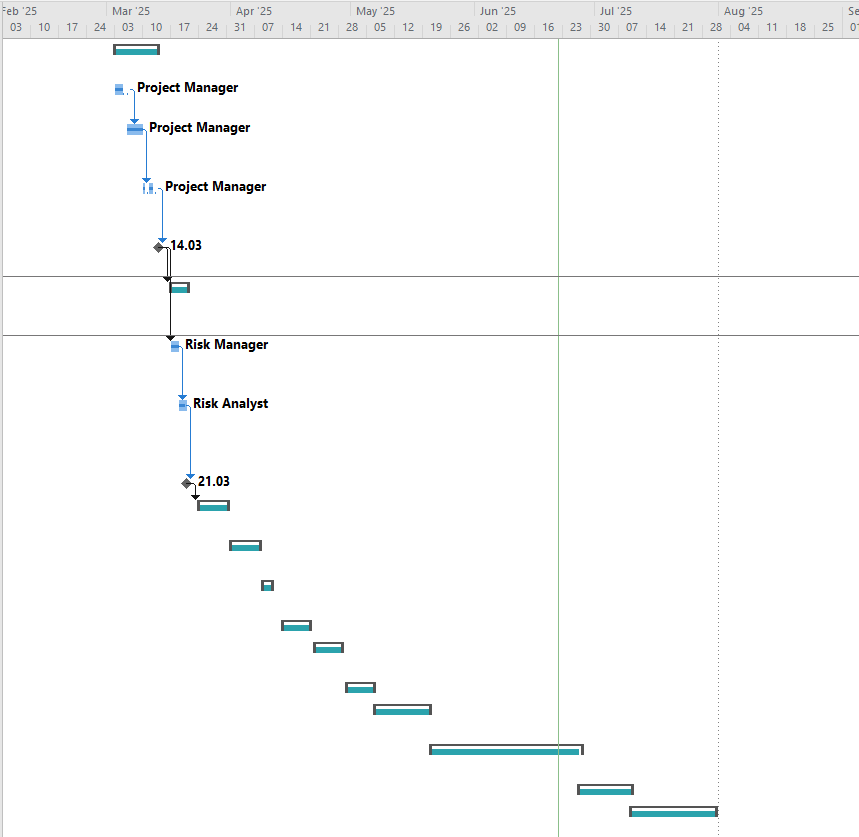
# Kompetencje: znajomość narzędzi testujących, analiza wyników.

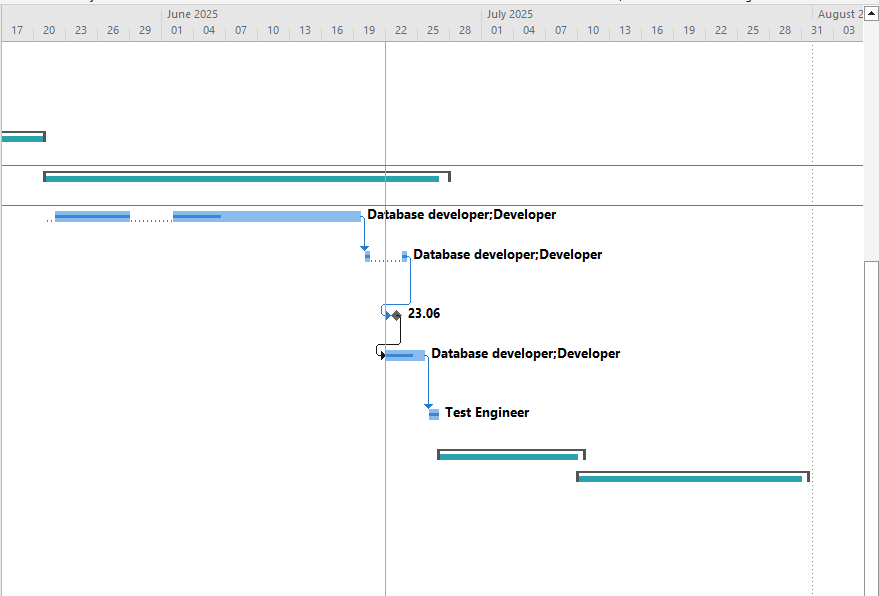
# Test Manager

# Obowiązki: planowanie testów, nadzór nad procesem testowym, baza błędów.

# Kompetencje: zarządzanie testami, planowanie jakości.

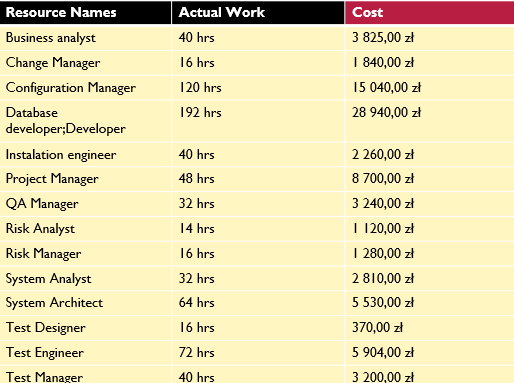
# 6. Harmonogram projektu



**

Wybrane fragmenty harmonogramu, ponieważ wykres jest zbyt duży, aby załączyć. Całość widoczna w pliku „Harmonogram\_Projektu\_v\_23\_04\_25.mpp”.

# 7. Kosztorys projektu



# 8. Zasoby projektu

* Dwa komputery z systemem operacyjnym Windows 11.
* Urządzenia wejścia:
  + Mysz
  + Klawiatura
* Urządzenia wyjścia:
  + Monitor
* Program SQL Server Management Studio 2021, który służy do modyfikacji bazy danych oraz zarządzania nią.
* Środowisko programistyczne: Visual Studio Community 2022 - 17.9.5
* Wybrane program z pakietu Microsoft Office 2022:
  + Word
  + Excel
  + MsProject
* Narzędzia do kontroli wersji: Git (Wersja 2.34.1)

# 9. Standardy i narzędzia w projekcie

* Język programowania: C#
* Framework: .NET 6.0 + WPF
* Środowisko programistyczne: Visual Studio Community 2022 – 17.9.5
* Baza danych: Azure SQL Server Gen5,
* Metodologia zarządzania projektem: Model kaskadowy
* Narzędzia do kontroli wersji: Git (Wersja 2.34.1)

# 10. Procesy zarządzania

## 10.1 Plan zarządzania konfiguracją

* **Zarządzanie wersjami:** Do kontroli wersji wykorzystywany jest system Git. Każda zmiana w kodzie źródłowym jest rejestrowana za pomocą commitów, co umożliwia śledzenie historii oraz współpracę w zespole. Wersje aplikacji są nadawane w formacie 1.0, 1.1, 2.0, w zależności od znaczenia zmian. Dokumenty projektowe są wersjonowane zgodnie z datą i numeracją (np. v\_15\_03\_25\_1).
* **Hierarchia uprawnień:** Programiści posiadają pełny dostęp do repozytorium, natomiast testerzy mają dostęp odczytowy.
* **Weryfikacja tożsamości:** Każdy członek zespołu posiada indywidualny login i hasło, co umożliwia identyfikację zmian.
* **Monitorowanie dostępu:** Zmiany w kodzie są regularnie przeglądane, a repozytorium podlega audytowi aktywności.
* **Konfiguracja oprogramowania:** Wszystkie zmiany i aktualizacje są dokumentowane w changelogu oraz odpowiednio oznaczone wersją.
* **Komunikacja:** W przypadku istotnych zmian w konfiguracji przeprowadzane są spotkania informacyjne, a dokumentacja aktualizowana jest w repozytorium.

**10.2 Plan zarządzania ryzykiem**

* **Identyfikacja ryzyk:** Przeprowadzane są burze mózgów i konsultacje zespołowe w celu zidentyfikowania potencjalnych zagrożeń technicznych i organizacyjnych.
* Ocena ryzyk: Każde ryzyko oceniane jest pod względem prawdopodobieństwa (P) i skutku (S), według skali:
* Prawdopodobieństwo:
  + 1 – bardzo mało prawdopodobne (25%)
  + 2 – mało prawdopodobne (50%)
  + 3 – prawdopodobne (75%)
  + 4 – bardzo prawdopodobne (99%)
* Skutek:
  + 1 – bardzo mała szkoda
  + 2 – mała szkoda
  + 3 – zauważalna szkoda, nie wpływa na kluczowe funkcje
  + 4 – szkoda ograniczająca funkcjonalność modułu
  + 5 – poważna szkoda, wpływająca na działanie systemu
  + 6 – krytyczna szkoda lokalna (np. u jednego klienta)
  + 7 – bardzo poważna szkoda systemowa
  + 8 – katastrofalna szkoda, zagrożenie dla realizacji projektu
* Działania minimalizujące: Dla każdego ryzyka opracowywane są działania prewencyjne wraz z szacowanymi kosztami ich wdrożenia.
* Dokumentacja ryzyk: Lista ryzyk oraz plan działań zaradczych znajduje się w pliku: „Ryzyka projektu v\_04\_04\_23\_1.xlsx”

## 10.3 Plan zarządzania testami

* Strategia testowania:
  + Projekt zakłada testy jednostkowe (dla logiki biznesowej i usług), testy UI (interfejsu użytkownika) oraz testy integracyjne API.
* Planowanie:
  + Faza testowania obejmuje testy WPF oraz API.
* Środowiska testowe:
  + Aplikacja WPF (UI / desktop):
    - Nazwa: TestStack.White
    - Opis: Framework do automatyzacji testów GUI dla aplikacji WPF/.NET
    - Język: C#
    - Licencja: Open Source
    - Link: https://github.com/TestStack/White
  + Backend API:
  + Nazwa: Postman
  + Opis: Narzędzie do testowania i dokumentowania REST API
  + Zastosowanie: testy metod POST/GET/PUT/DELETE, walidacja odpowiedzi JSON
  + Link: https://www.postman.com
* Szkolenia:
  + Zespół testowy został przeszkolony z użycia Postmana oraz tworzenia prostych skryptów testowych w TestStack.White.
* Raportowanie:
  + Testy API dokumentowane w formacie JSON/CSV z eksportem z Postmana
  + Testy GUI rejestrowane automatycznie i ręcznie w dzienniku testów
  + Raporty przesyłane do przełożonego według szablonu „Test case – szablon