Analiza specyfikacji wymagań

Nazwa projektu: System Zarządzania Wypożyczalnią Filmów

Wersja: 1.0

**1. Wstęp**

System ma umożliwić zarządzanie wypożyczeniami filmów w formie aplikacji desktopowej (WPF) z integracją z ASP.NET WebAPI i bazą danych. Kluczowymi funkcjonalnościami są dodawanie, edycja, usuwanie i raportowanie wypożyczeń.

**2. Cele analizy specyfikacji**

- Określenie wymagań funkcjonalnych i niefunkcjonalnych.  
- Stworzenie modelu aktorów i przypadków użycia.  
- Opracowanie podstaw projektowych bazy danych.  
- Zapewnienie podstaw do rozwoju, testowania i utrzymania systemu.

**3. Zakres specyfikacji**

System obsługuje operacje związane z zarządzaniem wypożyczeniami filmów (dodawanie, edycja, usuwanie, raporty), przechowywaniem danych w SQL Server oraz interakcją z użytkownikiem poprzez aplikację WPF.

**4. Analiza wymagań funkcjonalnych – model use case’ów**

**4.1 Specyfikacja aktorów**

- Pracownik wypożyczalni  
- System zewnętrzny (API)

**4.2 Lista use case’ów**

- Dodanie nowego wypożyczenia  
- Edycja wypożyczenia  
- Usunięcie wypożyczenia  
- Załaduj wypożyczenia z API  
- Eksport danych (PDF, CSV)  
- Wyszukiwanie wypożyczeń  
- Przeglądanie raportów  
- Dodanie daty zwrotu

**4.3 Diagram use case’ów UML**

[Do uzupełnienia graficznie, sugerowana forma: aktor — use case — system.]

**4.4 Specyfikacja use case’ów**

\*\*Dodanie wypożyczenia\*\*  
Aktor: Pracownik  
Opis: Wprowadza dane klienta i filmu, zatwierdza.  
Scenariusz alternatywny: Anulowanie okna.

\*\*Eksport do PDF\*\*  
Aktor: Pracownik  
Opis: Eksportuje statystyki i wykresy do pliku PDF.

**4.5 Diagramy aktywności UML**

[Do uzupełnienia graficznie — np. dodanie wypożyczenia: kliknięcie przycisku → otwarcie formularza → walidacja → wysyłka POST do API]

**5. Analiza wymagań niefunkcjonalnych**

**5.1 Interfejsy użytkownika**

- Aplikacja desktopowa WPF  
- Formularze okienne: dodawanie, edycja, raporty

**5.2 Interfejsy sprzętowe**

- Komputer klasy PC z systemem Windows  
- Dostęp do internetu (komunikacja z API)

**5.3 Interfejsy komunikacyjne**

- REST API (<https://localhost:7214/api/rentals>)  
- JSON jako format wymiany danych

**5.4 Interfejsy programowe**

- WPF (MVVM)  
- ASP.NET Core Web API  
- Entity Framework Core  
- OxyPlot do wykresów  
- iTextSharp do eksportu PDF

**6. Analiza wymagań dotyczących jakości**

- Użyteczność: Intuicyjny interfejs okienkowy  
- Niezawodność: Obsługa wyjątków, walidacja danych  
- Rozszerzalność: Możliwość dodania np. logowania, historii wypożyczeń  
- Przenośność: Zależna od .NET Core i Windows

**7. Analiza warunków serwisowania**

System oparty o znane technologie (WPF, .NET Core). Kod czytelnie zorganizowany wg wzorca MVVM. Łatwe debugowanie i rozwijanie dzięki modularności.

**8. Analiza ograniczeń architektury systemu**

- Brak wielodostępności — aplikacja jednostanowiskowa  
- Wymaga stałego dostępu do API w lokalnym środowisku  
- Brak funkcji logowania i autoryzacji  
- Brak rejestracji użytkownika końcowego — tylko operatorzy lokalni

**9. Model bazy danych**

**9.1 Specyfikacja atrybutów bazy danych**

Tabela Rentals:  
- Id (int, klucz główny)  
- MovieTitle (string)  
- CustomerName, CustomerSurname (string)  
- RentalDate (DateTime)  
- ReturnDate (DateTime?, opcjonalne)

**9.2 Specyfikacja operacji na danych**

- Dodawanie (POST /api/rentals)  
- Pobieranie listy (GET /api/rentals)  
- Edycja (PUT /api/rentals/{id})  
- Usuwanie (DELETE /api/rentals/{id})  
- Aktualizacja daty zwrotu (PUT /api/rentals/return/{id})

**9.3 Specyfikacja reguł poprawności i zgodności typów**

- CustomerName, MovieTitle — wymagane ([Required])  
- ReturnDate — może być null  
- Spójność zapewniona przez EF Core i walidację formularzy w WPF