UNIWERSYTET
KAZIMIERZA WIELKIEGO
W BYDGOSZCZY

WYDZIAŁ INFORMATYKI

Podstawy Inżynierii Oprogramowania

Pracę wykonał: Szymon Dybowski

Nr.albumu: 104550

Grupa: nr 2

Data: 28.01.2026

Spis treści

1. Wstęp

Laboratoria koncentrowały się na praktycznych aspektach procesu wytwarzania oprogramowania. Kluczowym zadaniem było przejście przez wszystkie etapy budowy systemu „TaskManagerV2" – aplikacji internetowej służącej do zarządzania listą zadań. Dzięki temu projektowi możliwe było opanowanie technologii C#, obsługi Visual Studio oraz zasad architektury warstwowej i programowania obiektowego.

Zgodnie z wytycznymi, końcowa faza prac (testy i dokumentacja) została przeprowadzona na finalnej wersji aplikacji udostępnionej w materiałach dydaktycznych, co zapewniło spójność środowiska testowego dla wszystkich studentów.

2. Cykl życia oprogramowania (SDLC)

Proces wytwórczy aplikacji oparto na klasycznym modelu cyklu życia oprogramowania (SDLC). Poszczególne fazy, których teoretyczne podstawy omówiono na początku semestru, wdrożono w następujący sposób:

1. **Analiza i planowanie:** Zrozumienie celu systemu, określenie grupy docelowej oraz zdefiniowanie wymagań funkcjonalnych (MVP).
2. **Projektowanie:** Zaplanowanie architektury aplikacji zgodnej z wzorcem Domain Driven Design (DDD) oraz podział na warstwy (Core, Infrastructure, Web, Tests).
3. **Implementacja:** Pisanie kodu w języku C# w iteracjach, tworzenie modeli domenowych oraz logiki biznesowej.

4. **Testowanie:** Weryfikacja działania programu poprzez testy jednostkowe (logika) oraz funkcjonalne (interfejs użytkownika), wykonana na finalnej wersji aplikacji.
5. **Wdrożenie i utrzymanie:** Konfiguracja bazy danych, przeprowadzenie migracji oraz przygotowanie aplikacji do działania.

3. Zarządzanie wymaganiami

Aby zminimalizować ryzyko błędów interpretacyjnych na etapie tworzenia kodu, prace zainaugurowano procesem identyfikacji i analizy wymagań. Rezultatem tych działań było sporządzenie specyfikacji opartej na przypadkach użycia oraz liście priorytetowych funkcji dla użytkownika.

Główne wymagania funkcjonalne aplikacji TaskManagerV2:

- **Dodawanie zadań:** Użytkownik może utworzyć nowe zadanie, definiując jego tytuł (wymagane) oraz opcjonalny opis.
- **Wyświetlanie listy:** Prezentacja wszystkich zadań wraz z ich aktualnym statusem.
- **Edycja zadań:** Możliwość modyfikacji treści i tytułu istniejącego zadania.
- **Usuwanie zadań:** Możliwość trwałego usunięcia zadania z systemu.
- **Zmiana statusu (Wykonane/Otwarte):** Funkcja pozwalająca oznaczyć zadanie jako "zrobione" oraz możliwość cofnięcia tej operacji.
- **Historia i daty:** System musi przechowywać i wyświetlać datę utworzenia zadania oraz datę jego wykonania.

4. Opis aplikacji TaskManagerV2

Aplikacja internetowa TaskManagerV2 służy do elektronicznej obsługi i organizacji zadań typu „to-do”. Jej wdrożenie pozwala wyeliminować konieczność prowadzenia papierowych wykazów, oferując w zamian stabilne i trwałe przechowywanie wprowadzonych informacji.

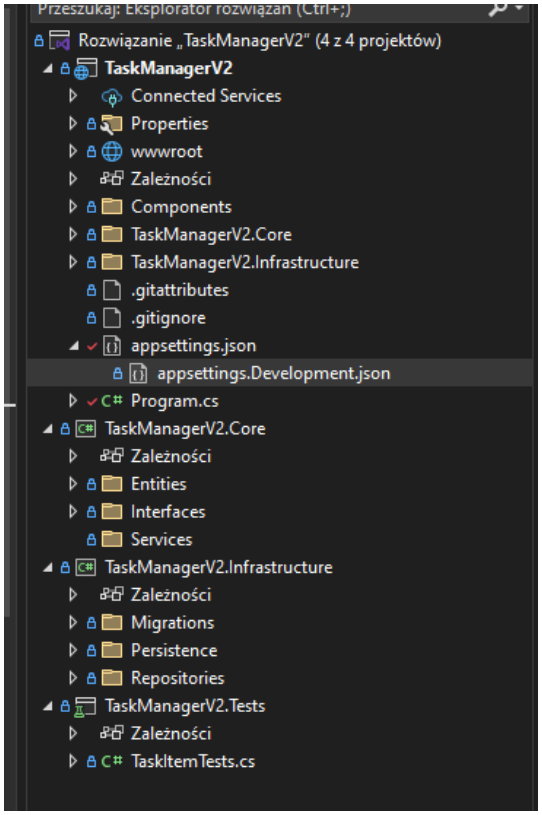
Odniesienie do wersji finalnej: Sprawozdanie opiera się na **finalnej wersji aplikacji udostępnionej w materiałach z zajęć**. Wersja ta posiada zaimplementowany pełny podział na warstwy, działającą integrację z bazą danych oraz kompletny interfejs użytkownika, który umożliwia wyświetlanie listy zadań, ich statusów oraz dat. System pozwala na interakcję w czasie rzeczywistym (CRUD), co zostało zweryfikowane w dalszej części sprawozdania.

5. Struktura projektu i architektura

W realizacji projektu zastosowano podejście architektoniczne Domain Driven Design (DDD). Wymusiło to organizację kodu w formie odrębnych projektów wewnątrz solucji, zapewniając tym samym ścisłe rozdzielenie logiki domeny od implementacji technicznej.

Omówienie warstw:

1. **TaskManagerV2.Core:** "Serce" aplikacji. Zawiera klasy modeli domenowych (np. TaskItem) oraz interfejsy repozytoriów (np. ITaskRepository). Warstwa ta jest niezależna od bazy danych i interfejsu.
2. **TaskManagerV2.Infrastructure:** Warstwa odpowiedzialna za komunikację ze światem zewnętrznym. Tutaj zaimplementowano AppDbContext oraz konkretne repozytoria, które realizują metody zdefiniowane w Core.
3. **TaskManagerV2.Tests:** Projekt zawierający testy jednostkowe weryfikujące logikę aplikacji w izolacji.



6. Baza danych

Przechowywanie informacji o zadaniach odbywa się w relacyjnej bazie danych. Projekt bazuje na podejściu Code-First, wykorzystując framework ORM Entity Framework Core do mapowania obiektowo-relacyjnego.

Proces inicjalizacji: Struktura tabel nie była tworzona ręcznie w SQL, lecz wygenerowana na podstawie klas C# (TaskItem). Proces ten odbył się poprzez mechanizm **Migracji**:

1. Utworzenie migracji komendą: Add-Migration InitialCreate.
2. Aktualizacja bazy komendą: Update-Database, co fizycznie utworzyło tabele w bazie danych.

Dzięki temu zapewniono trwałość danych (persistence) pomiędzy sesjami aplikacji.

7. Testy jednostkowe

Testy jednostkowe zostały umieszczone w projekcie TaskManagerV2.Tests. Ich celem jest weryfikacja poprawności działania pojedynczych metod domeny biznesowej bez angażowania bazy danych czy interfejsu graficznego.

Przykłady testów zawartych w projekcie:

- **Test oznaczania jako wykonane:** Weryfikuje, czy wywołanie metody MarkDone() na obiekcie zadania poprawnie zmienia flagę IsDone na true oraz czy ustawia datę zakończenia.
- **Test wznawiania zadania:** Sprawdza, czy metoda Reopen() przywraca status zadania na "Otwarte" i czyści datę wykonania.
- **Test walidacji:** Sprawdza, czy nie można utworzyć zadania bez podania tytułu (zgodnie z regułami walidacji).

Testy te dają pewność, że kluczowa logika biznesowa działa poprawnie przed integracją z resztą systemu.

8. Testy funkcjonalne

Test 1: Oznaczenie zadania jako wykonane

Nazwa testu: TF-03 - Zmiana statusu zadania na wykonane.

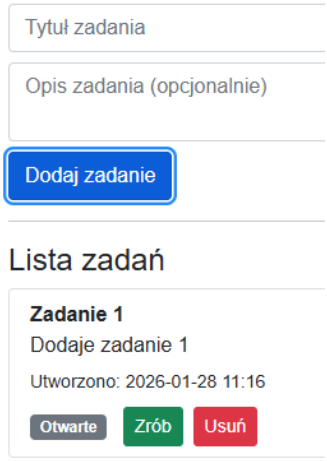
Cel testu: Sprawdzenie, czy użytkownik może oznaczyć istniejące zadanie jako zakończone oraz czy system poprawnie aktualizuje datę wykonania.

Kroki wykonania:

- Użytkownik uruchamia aplikację TaskManagerV2.
- Na liście zadań odnajduje zadanie o statusie „Otwarte”.
- Klika przycisk akcji „Zrób” (lub checkbox) przy wybranym zadaniu.

Oczekiwany rezultat:

- Status zadania zmienia się na „Wykonane” (zmiana koloru/etykiety).
- Przy zadaniu pojawia się data i godzina wykonania (CompletedAt).
- Opcja zmiany statusu zmienia się na „Cofnij” (umożliwiając powrót do statusu otwartego).



Test 2: Usunięcie zadania

Nazwa testu: TF-02 - Trwałe usunięcie zadania.

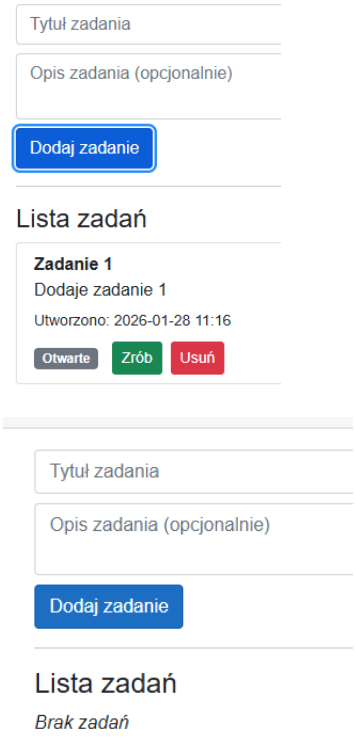
Cel testu: Weryfikacja, czy system pozwala użytkownikowi na skuteczne usunięcie zadania z listy.

Kroki wykonania:

- Użytkownik wybiera z listy zadanie, które chce usunąć.
- Klika czerwony przycisk „Usuń” przy wybranym wierszu.
- System odświeża widok listy zadań.

Oczekiwany rezultat:

- Wybrane zadanie znika z listy widocznej na ekranie.
- Liczba zadań w tabeli zmniejsza się.
- Zadanie nie pojawia się ponownie po ręcznym odświeżeniu strony (trwałe usunięcie z bazy).

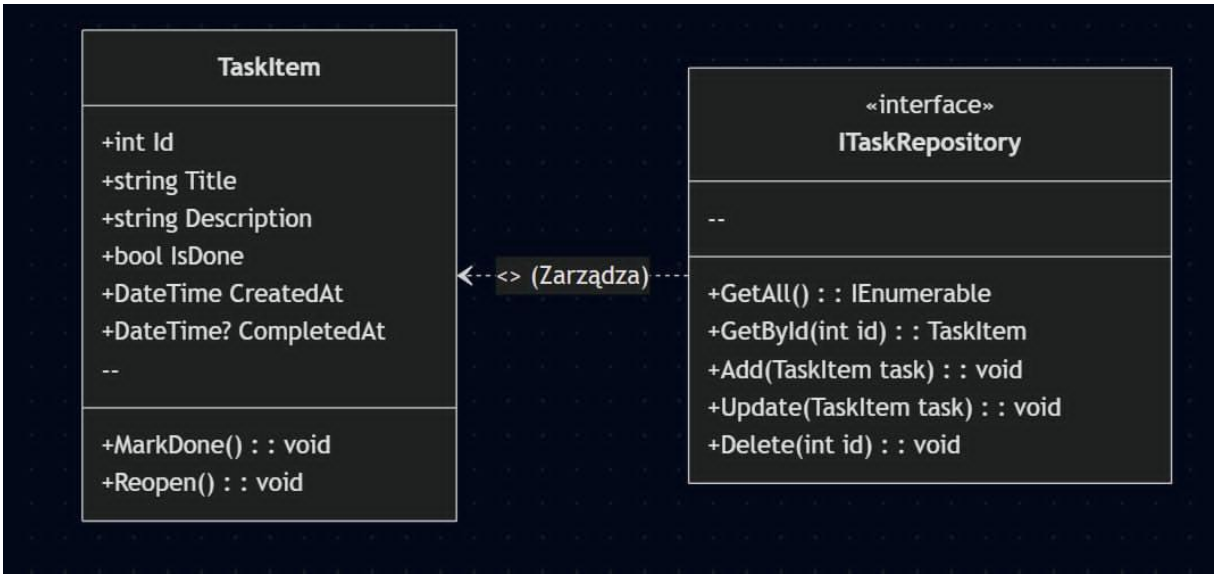


Analiza pracochłonności i jakości testów

- **Koszt czasowy:** Najbardziej kosztownym czasowo testem funkcjonalnym jest scenariusz TF-03 (Zmiana statusu). Wynika to z faktu, że wymaga on spełnienia szerszych warunków wstępnych – aby go przeprowadzić, tester musi najpierw przejść przez proces tworzenia nowego zadania, co wydłuża czas przygotowania środowiska.

- **Wpływ na jakość:** Testy funkcjonalne pozwoliły wykryć aspekty niewidoczne w kodzie, takie jak czytelność interfejsu (UX) oraz poprawne odświeżanie listy zadań po wykonaniu akcji (np. zniknięcie zadania po usunięciu). Potwierdzają one użyteczność aplikacji dla końcowego użytkownika.

9. Diagram UML



Powyższy Diagram Klas (Class Diagram) obrazuje model domenowy aplikacji w warstwie Core. Diagram przedstawia dwa kluczowe elementy systemu oraz relację między nimi:

1. **Klasa TaskItem (Encja domenowa):** Jest to główny obiekt biznesowy aplikacji, reprezentujący pojedyncze zadanie.

- **Atrybuty:** Przechowuje stan zadania, m.in. unikalny identyfikator (Id), tytuł (Title), opis (Description) oraz flagę statusu (IsDone). Przechowywane są również znaczniki czasowe: moment utworzenia (CreatedAt) oraz moment wykonania (CompletedAt).

- **Metody:** Klasa zawiera nie tylko dane, ale również logikę biznesową w postaci metod MarkDone() (oznacz jako wykonane) oraz Reopen() (wznów zadanie), co jest zgodne z podejściem programowania obiektowego.

2. **Interfejs ITaskRepository (Abstrakcja danych):** Interfejs ten definiuje kontrakt dla warstwy dostępu do danych (Data Access Layer). Określa on, jakie operacje można wykonać na bazie danych, nie zdradzając jednak szczegółów technicznych (jak SQL czy Entity Framework).

- **Zawiera metody typu CRUD:** Add (dodawanie), GetAll (pobieranie listy), Update (aktualizacja) oraz Delete (usuwanie).

3. **Relacja (Zależność):** Strzałka z opisem "Zarządza" wskazuje, że repozytorium jest odpowiedzialne za cykl życia obiektów TaskItem. To repozytorium "używa" klasy TaskItem do przesyłania danych między bazą a aplikacją. Dzięki zastosowaniu interfejsu zamiast konkretnej klasy, logika biznesowa (Core) jest odseparowana od szczegółów bazy danych (Infrastructure).

10. Wnioski

Realizacja projektu „TaskManagerV2" pozwoliła na praktyczne zastosowanie wiedzy z zakresu inżynierii oprogramowania i przejście przez pełen cykl wytwarzania aplikacji (SDLC). Do najważniejszych wniosków i spostrzeżeń należą:

1. **Zrozumienie Architektury DDD:** Projekt pozwolił mi zrozumieć zalety podziału aplikacji na warstwy (Core, Infrastructure, Web). Dzięki temu logika biznesowa jest niezależna od szczegółów technicznych, co ułatwia zarządzanie kodem i jego testowanie.
2. **Praca z Bazą Danych i ORM:** Nauczyłem się wykorzystywać Entity Framework Core w podejściu *Code-First*. Zrozumiałem, jak mapować obiekty C# na tabele w bazie MySQL oraz jak ważna jest poprawna konfiguracja plików appsettings.json w środowisku deweloperskim.
3. **Rozwiązywanie problemów konfiguracyjnych:** Podczas pracy napotkałem istotne problemy z połączeniem do bazy danych (błąd Access denied oraz brak tabel). Rozwiązanie tych problemów wymagało ode mnie analizy konfiguracji połączenia (Connection String) oraz zaimplementowania mechanizmu automatycznego tworzenia bazy (db.Database.EnsureCreated()) w kodzie startowym aplikacji. Była to cenna lekcja debugowania środowiska uruchomieniowego .
4. **Rola Testów Funkcjonalnych:** Przeprowadzenie manualnych testów funkcjonalnych (oznaczanie zadań, usuwanie) pokazało mi, że poprawność kodu to nie wszystko – równie ważna jest perspektywa użytkownika końcowego i użyteczność interfejsu (UX) .

5. **Dokumentacja UML:** Przygotowanie diagramu klas pozwoliło mi lepiej zwizualizować strukturę danych projektu i relacje między klasą domenową (TaskItem) a warstwą dostępu do danych (ITaskRepository).

Podsumowując, projekt był kompleksowym ćwiczeniem łączącym programowanie, konfigurację serwerową oraz dbałość o jakość oprogramowania poprzez testy i dokumentację.

11. Git i zarządzanie wersjami

Projekt, zgodnie z wymaganiami, został umieszczony w systemie kontroli wersji.

Link do repozytorium GitHub: [TUTAJ WKLEJ LINK DO SWOJEGO REPOZYTORIUM]