Projekt zaliczeniowy

Programowanie obiektowe
Rok akademicki 2025/2026

Autorzy:

Konrad Biały

TaskBoard

Projekt TaskBoard to aplikacja do zarządzania zadaniami, napisana w języku C# z wykorzystaniem programowania obiektowego. W projekcie zaimplementowano hierarchię klas reprezentujących zadania, w tym klasę bazową TaskItem oraz klasy pochodne (np. BugTask), co pozwoliło na wykorzystanie dziedziczenia i polimorfizmu. Zastosowano klasy abstrakcyjne oraz metody wirtualne i nadpisywane.

Dane przechowywane są w kolekcjach, a dostęp do pól realizowany jest poprzez właściwości z kontrolą poprawności danych. W projekcie zaimplementowano odczyt i zapis danych do pliku w formacie JSON, co umożliwia prostą persistencję stanu aplikacji. Kod został podzielony na logiczne warstwy, obejmujące m.in. logikę domenową oraz obsługę danych.

Podział ról

Konrad Biały: Wszystko

Spis treści

Klasy w projekcie:

TaskItem:

Opis:

TaskItem jest klasą abstrakcyjną stanowiącą bazę dla wszystkich typów zadań w systemie. Zawiera wspólne dane i zachowania, takie jak identyfikator, tytuł, opis, status, priorytet, data utworzenia, termin wykonania oraz lista przypisanych użytkowników. Dzięki temu unika się duplikacji kodu i można obsługiwać różne typy zadań polimorficznie (np. przez wywołanie GetSummary() niezależnie od konkretnego typu zadania).

Rola w projekcie / uzasadnienie istnienia:

Klasa jest potrzebna, ponieważ w projekcie występują różne rodzaje zadań (BugTask, FeatureTask), które mają wspólne cechy (np. Title, DueDate, Priority) oraz wspólne operacje (np. zmiana terminu, przypisanie użytkowników). TaskItem stanowi centralny element domeny i umożliwia rozbudowę o kolejne typy zadań w przyszłości.

Uzasadnienie modyfikatorów dostępu:

Pola są private (np. _title, _createdAt, _assignedTo), aby wymusić kontrolę zmian przez metody/właściwości i utrzymać poprawność stanu obiektu (enkapsulacja).

CreatedAt ma tylko getter, ponieważ jest ustalane w konstruktorze i nie powinno być modyfikowane z zewnątrz.

DueDate również jest tylko do odczytu, a zmiana terminu jest realizowana przez metodę ChangeDueDate, która waliduje poprawność daty.

Metody SetTitle i SetDueDate są private, bo są wewnętrznymi mechanizmami walidacji i ustawiania pól; mają być używane wyłącznie przez konstruktor i metody publiczne (Rename, ChangeDueDate). GetSummary() jest virtual, aby klasy pochodne mogły rozszerzyć opis o dane specyficzne dla danego typu zadania (np. moduł w FeatureTask, severity w BugTask).

BugTask:

Opis:

BugTask to specjalizacja TaskItem, reprezentująca zadanie typu „bug”. Rozszerza zadanie o pola specyficzne dla błędów: severity, kroki reprodukcji oraz oczekiwany i rzeczywisty rezultat. Nadpisuje GetSummary(), aby prezentować dodatkowe informacje (severity).

Rola w projekcie / uzasadnienie istnienia:

Klasa jest potrzebna, aby rozróżniać różne typy zadań i przechowywać dane charakterystyczne wyłącznie dla błędów, bez „zaśmiecania” klasy bazowej TaskItem polami niepasującymi do innych zadań.

Uzasadnienie modyfikatorów dostępu:

Pola specyficzne (_severity, _stepsToReproduce, itd.) są private, a dostęp do nich odbywa się przez właściwości (Severity, StepsToReproduce…), co pozwala w przyszłości dodać walidację bez zmiany interfejsu klasy.

Nadpisanie GetSummary() (override) wynika z potrzeby dopisania informacji domenowych typowych dla błędu.

FeatureTask:

Opis:

FeatureTask to typ zadania reprezentujący funkcjonalność/feature. Dziedziczy po TaskItem i rozszerza model o ModuleName, czyli informację o module/systemie, którego dotyczy zadanie. GetSummary() jest nadpisane, aby uwzględnić nazwę modułu.

Rola w projekcie / uzasadnienie istnienia:

Klasa umożliwia rozróżnienie zadań typu „feature" od innych typów zadań oraz przechowywanie danych specyficznych dla rozwoju funkcjonalności. Dzięki temu logika wspólna pozostaje w TaskItem, a dane charakterystyczne są w klasach pochodnych.

Uzasadnienie modyfikatorów dostępu:

_moduleName jest private, a ustawianie odbywa się przez właściwość ModuleName, która waliduje wartość (nie pozwala na null/whitespace i przycina spacje). To chroni spójność danych w obiekcie.

Board:

Opis:

Board reprezentuje tablicę z kolumnami. Przechowuje Id, Name oraz listę Columns i udostępnia operacje domenowe: dodawanie/usuwanie kolumn, pobieranie wszystkich zadań oraz przenoszenie zadania między kolumnami (MoveTask).

Rola w projekcie / uzasadnienie istnienia:

Klasa spina strukturę aplikacji: to ona agreguje kolumny i umożliwia operacje na poziomie całej tablicy (np. przeniesienie zadania między kolumnami), co jest naturalną odpowiedzialnością obiektu domenowego „Board".

Uzasadnienie modyfikatorów dostępu:

Id ma private set, ponieważ identyfikator jest nadawany w konstruktorze i nie powinien być dowolnie zmieniany z zewnątrz.

Columns ma private set, aby lista nie została podmieniona z zewnątrz. Zmiany w strukturze tablicy są wykonywane przez metody domenowe (AddColumn, RemoveColumn).

Pole _name jest private, a ustawianie przez właściwość Name wymusza walidację (brak pustych nazw).

Konstruktor bezparametrowy jest private, co ogranicza tworzenie obiektu bez ustawionych danych (np. bez nazwy) i wymusza użycie konstruktora Board(string name).

Column:

Opis:

Column reprezentuje pojedynczą kolumnę na tablicy. Zawiera własne Id, nazwę oraz listę zadań (TaskItem). Udostępnia metody dodawania i usuwania zadań.

Rola w projekcie / uzasadnienie istnienia:

Klasa jest potrzebna do modelowania struktury tablicy Kanban: zadania są grupowane w kolumnach, a kolumna stanowi granicę odpowiedzialności dla operacji takich jak dodanie/usunięcie zadania.

Uzasadnienie modyfikatorów dostępu:

Lista _tasks jest private, natomiast na zewnątrz wystawiany jest IReadOnlyList<TaskItem> Tasks, co chroni przed modyfikacją listy bez kontroli (np. bez walidacji w AddTask).

Pole _name jest ustawiane przez właściwość Name, która waliduje dane wejściowe i zapobiega tworzeniu kolumn bez nazwy.

Person:

Opis:

Person jest klasą opisującą podstawowe dane osoby: FirstName, LastName oraz Id. Zawiera metodę GetDisplayName() zwracającą nazwę w formie „Imię Nazwisko”.

Rola w projekcie / uzasadnienie istnienia:

Klasa pełni rolę bazową dla użytkownika (User). Dzięki temu dane osobowe są wydzielone i mogą być wielokrotnie wykorzystywane, a klasa User rozszerza je o informacje typowo „systemowe” (email, rola).

Uzasadnienie modyfikatorów dostępu:

Pola są private, a właściwości FirstName i LastName walidują wartości (nie dopuszczają null/whitespace), co chroni obiekt przed niepoprawnym stanem.

GetDisplayName() jest publiczne, bo jest bezpieczną operacją prezentacyjną i może być używane przez inne elementy aplikacji.

User:

Opis:

User dziedziczy po Person i reprezentuje użytkownika systemu. Rozszerza osobę o Email, Role oraz listę przypisanych zadań (AssignedTasks). Udostępnia metody przypisywania i odpinania zadań oraz zmianę roli. Dodatkowo implementuje IEquatable<User> i przeciąża operatory == i !=, aby porównywać użytkowników po Id.

Rola w projekcie / uzasadnienie istnienia:

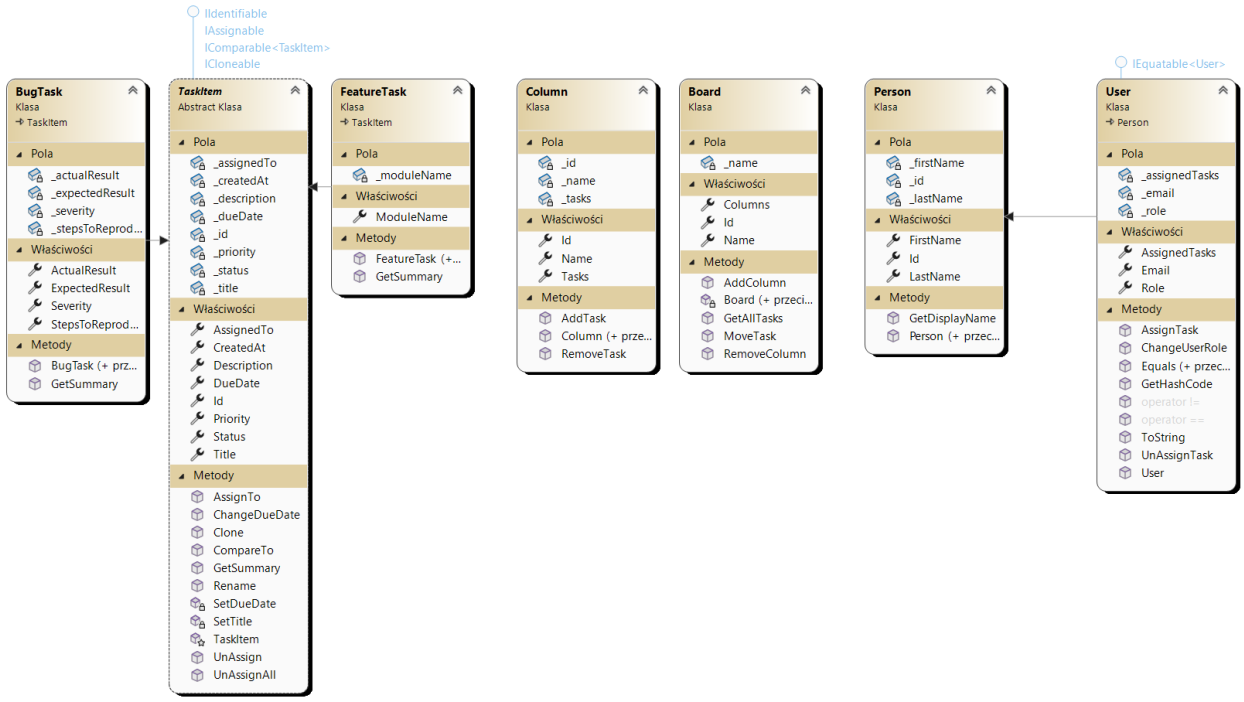
Klasa jest potrzebna do modelowania użytkowników, którzy mogą mieć role (np. admin/manager) i mogą być przypisywani do zadań. Lista przypisanych zadań pozwala śledzić, nad czym pracuje dany użytkownik.

Uzasadnienie modyfikatorów dostępu:

_email jest private, a walidacja odbywa się we właściwości Email (ochrona poprawności danych).

Lista _assignedTasks jest private, a na zewnątrz jest wystawiona jako właściwość tylko do odczytu (AssignedTasks), dzięki czemu modyfikacje powinny przechodzić przez metody AssignTask/UnAssignTask (kontrola logiki).

Implementacja Equals/GetHashCode i operatorów jest uzasadniona, ponieważ użytkownicy są identyfikowani przez Id, co jest spójne z działaniem kolekcji (np. Contains).



Rysunek 1 - Diagram klas w projekcie

Interfejsy:

IAssignable:

Opis oraz rola w projekcie:

Interfejs IAssignable definiuje kontrakt dla obiektów, do których mogą być przypisywani użytkownicy. Udostępnia operacje przypisania (AssignTo), odpięcia pojedynczego użytkownika (UnAssign) oraz usunięcia wszystkich przypisań (UnAssignAll). W projekcie jest implementowany

przez klasę TaskItem, co oznacza, że każde zadanie może być przypisane do jednego lub wielu użytkowników.

Uzasadnienie modyfikatorów dostępu:

Interfejs posiada modyfikator internal, co ogranicza jego użycie wyłącznie do wnętrza warstwy domenowej. Dzięki temu mechanizm przypisywania użytkowników nie jest wystawiony jako publiczne API biblioteki i pozostaje szczegółem implementacyjnym domeny.

IIdentifiable:

Opis:

IIdentifiable definiuje, że obiekt posiada Id. Jest używany w TaskItem, co ułatwia traktowanie encji domenowych w jednolity sposób (np. identyfikowanie obiektów w kolekcjach i operacjach domenowych).

Uzasadnienie modyfikatorów dostępu:

internal ogranicza użycie interfejsu do wnętrza domeny. W sprawozdaniu możesz to opisać jako ochronę przed „wyciekiem" szczegółów implementacyjnych na zewnątrz projektu.

IBoardRepository:

Opis oraz rola w projekcie:

IBoardRepository jest abstrakcją dostępu do danych dla obiektów typu Board. Udostępnia podstawowe operacje CRUD (GetAll, GetById, Add, Update, Remove). Dzięki temu logika domenowa nie zależy od konkretnego sposobu przechowywania danych (np. plik, baza danych).

Uzasadnienie modyfikatorów dostępu:

Interfejs jest public, ponieważ ma być używany przez inne warstwy (np. UI / aplikację), które potrzebują pobierać i zapisywać tablice bez znajomości implementacji repozytorium.

IUnitOfWork:

Opis oraz rola w projekcie:

IUnitOfWork agreguje repozytoria (tu: Boards) i udostępnia metody utrwalenia oraz odświeżenia danych (SaveChanges, Reload). Zapewnia spójny sposób pracy z zapisem zmian jako jedną operacją.

Uzasadnienie modyfikatorów dostępu:

Jest public, bo stanowi „punkt wejścia" do warstwy danych dla reszty aplikacji (np. UI). Dzięki temu UI korzysta z jednego obiektu do pracy z danymi, bez zależności od szczegółów implementacji.

IBoardDataStore:

Opis oraz rola w projekcie:

IBoardDataStore definiuje sposób zapisu i odczytu danych tablic (Board) do oraz z pliku. Interfejs abstrahuje mechanizm persistencji danych, umożliwiając zapis i wczytywanie stanu aplikacji bez uzależniania logiki domenowej od konkretnego formatu przechowywania danych (np. JSON).

Uzasadnienie modyfikatorów dostępu:

Interfejs jest oznaczony jako public, ponieważ może być wykorzystywany przez inne warstwy aplikacji (np. warstwę aplikacyjną lub UI), które korzystają z zapisu i odczytu danych, nie znając szczegółów implementacji.

Opis funkcjonalności:

Projekt TaskBoard umożliwia zarządzanie zadaniami w postaci tablicy składającej się z kolumn oraz zadań różnego typu. Poniżej opisano podstawowe funkcjonalności systemu wraz z klasami odpowiedzialnymi za ich realizację.

Tworzenie i zarządzanie tablicami:

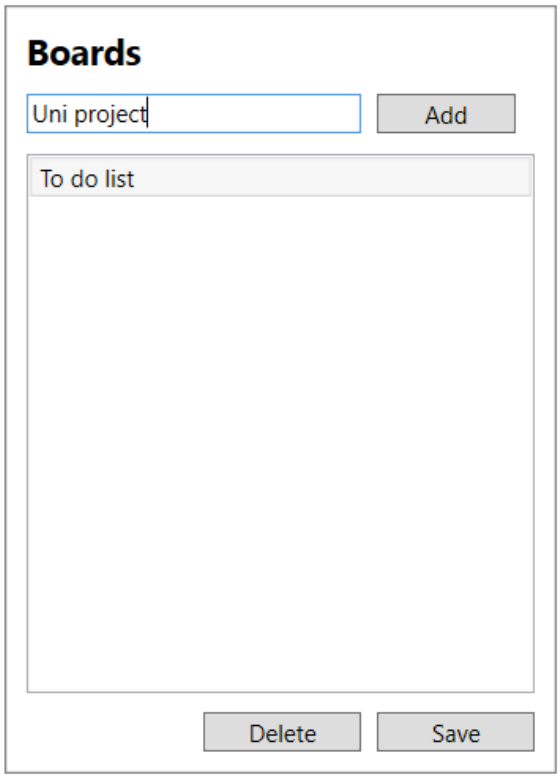
Użytkownik może tworzyć tablice robocze, które stanowią główną strukturę organizacyjną projektu.

Funkcjonalności:

1. utworzenie tablicy o zadanej nazwie,
2. przechowywanie listy kolumn w obrębie tablicy,
3. pobieranie wszystkich zadań znajdujących się na tablicy.

Klasy odpowiedzialne:

- Board – reprezentuje tablicę i zarządza kolekcją kolumn,
- Column – przechowuje zadania należące do danej kolumny.



Rysunek 2 - Board menu

Zarządzanie kolumnami

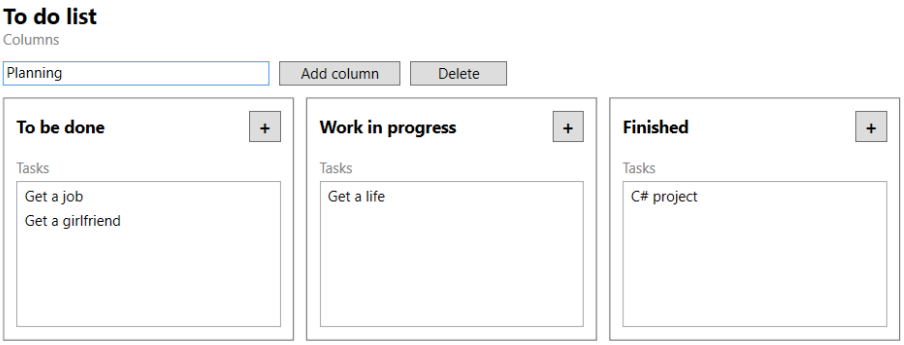
Tablica może składać się z dowolnej liczby kolumn, które grupują zadania według ich statusu lub etapu realizacji.

Funkcjonalności:

1. dodawanie kolumn do tablicy,
2. usuwanie kolumn,
3. przechowywanie listy zadań w danej kolumnie.

Klasy odpowiedzialne:

- Board – dodawanie i usuwanie kolumn,
- Column – przechowywanie i zarządzanie zadaniami w kolumnie.



Rysunek 3 - Menu kolumn

Tworzenie i obsługa zadań

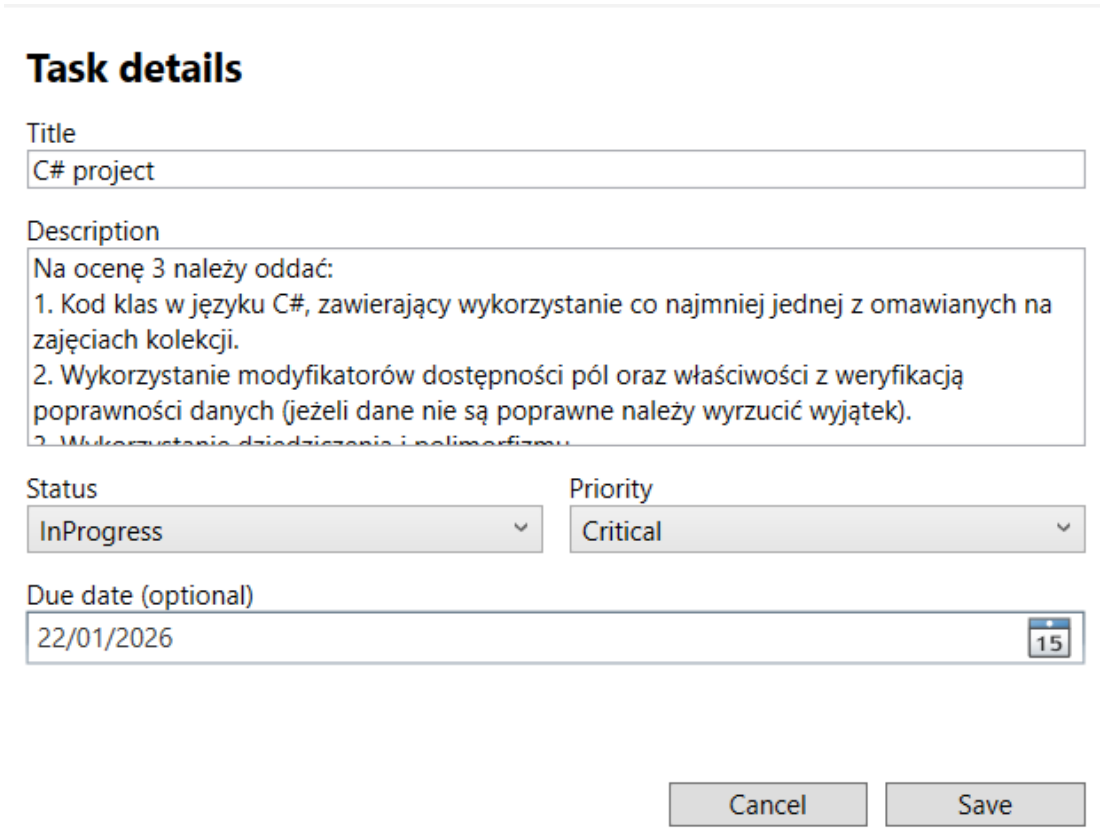
System umożliwia tworzenie i obsługę zadań o różnych typach, posiadających wspólne cechy, ale także dane specyficzne dla danego rodzaju zadania.

Funkcjonalności:

1. tworzenie zadań,
2. zmiana tytułu i terminu wykonania zadania,
3. ustawianie priorytetu i statusu,
4. wyświetlanie podsumowania zadania.

Klasy odpowiedzialne:

- TaskItem – klasa abstrakcyjna zawierająca wspólną logikę wszystkich zadań,
- BugTask – zadanie typu błąd, rozszerzone o informacje takie jak severity,
- FeatureTask – zadanie typu funkcjonalność, zawierające nazwę modułu,
- TaskPriority, TaskStatus – enumy określające priorytet i status zadania.



Rysunek 4 - Menu zadań

Przenoszenie zadań pomiędzy kolumnami

Zadania mogą być przenoszone pomiędzy kolumnami, co odzwierciedla postęp prac.

Funkcjonalności:

1. wyszukiwanie zadania w kolumnach,
2. przeniesienie zadania do innej kolumny tablicy.

Klasy odpowiedzialne:

- Board – metoda MoveTask, która koordynuje przeniesienie zadania,
- Column – dodawanie i usuwanie zadań z listy kolumny.

Zapis i odczyt danych

Projekt umożliwia zapis i odczyt stanu tablic do pliku, co pozwala na zachowanie danych pomiędzy uruchomieniami aplikacji.

Funkcjonalności:

1. zapis tablic do pliku,
2. odczyt tablic z pliku.

Klasy odpowiedzialne:

- IBoardDataStore – interfejs definiujący operacje zapisu i odczytu danych,
- klasy domenowe (Board, Column, TaskItem) – obiekty podlegające serializacji.

Insktrukcja uruchomienia i użytkowania projektu:

Uruchomienie projektu:

1. Otworzyć rozwiązanie projektu w środowisku Visual Studio.
2. Upewnić się, że poprawnie ustawiony jest projekt startowy.
3. Uruchomić aplikację

Podstawowe użytkowanie aplikacji:

1. Po uruchomieniu aplikacji możliwe jest utworzenie tablicy roboczej.
2. Do tablicy można dodawać kolumny reprezentujące etapy pracy.
3. W obrębie kolumn można tworzyć zadania różnego typu (np. zadania typu *bug* lub *feature*).
4. Zadaniom można przypisywać priorytet, status oraz użytkowników.
5. Zadania mogą być przenoszone pomiędzy kolumnami w celu odzwierciedlenia postępu prac.
6. Dane projektu mogą zostać zapisane do pliku oraz ponownie wczytane przy kolejnym uruchomieniu aplikacji.

Zakończenie pracy:

Po zakończeniu pracy z aplikacją użytkownik może zapisać aktualny stan tablic do pliku. Zapisane dane umożliwiają odtworzenie stanu projektu przy ponownym uruchomieniu aplikacji.