TaskAPP

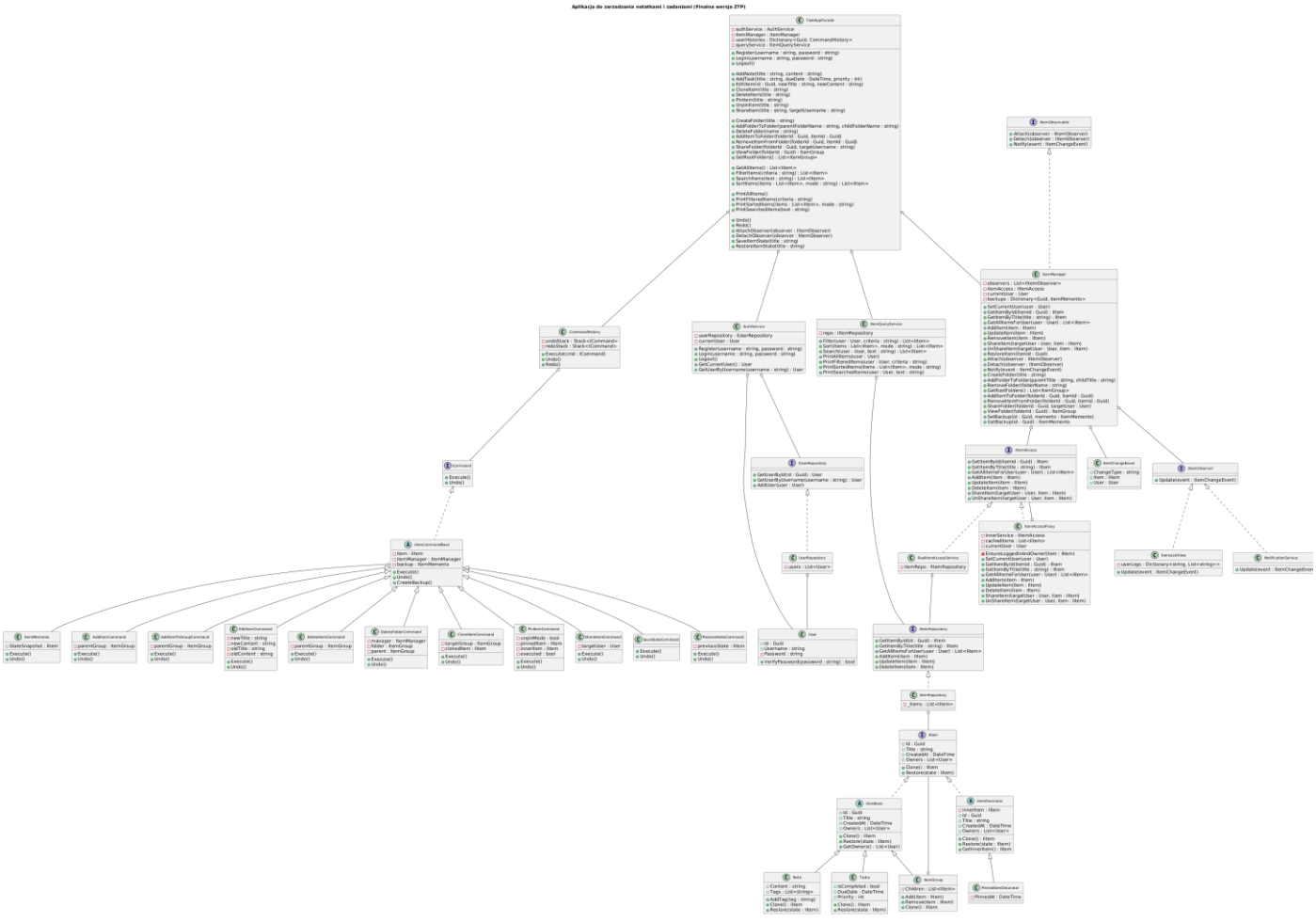
Aplikacja do zarządzania notatkami oraz zadaniami

Zespół:

- Miłosz Szymczuk
- Miłosz Pawlaczyk
- Jakub Komorowski
- Adrian Lachowicz
- Jan Kozłowski
- Paweł Gronostajski

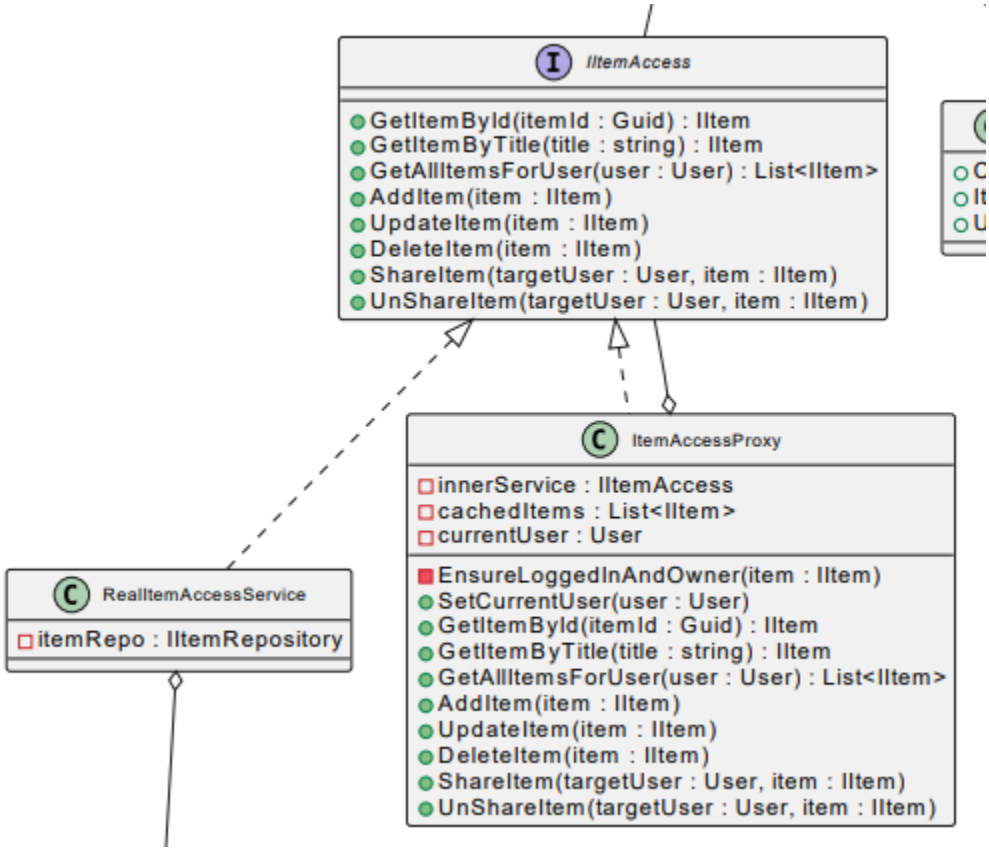
Diagram UML

Link: https://tinyurl.com/yjmf7ykb



Opis zastosowanych wzorców projektowych

Proxy (Protection Proxy + Cache)



Cel użycia

Celem zastosowania wzorca Proxy było wprowadzenie kontroli dostępu do obiektów Item oraz centralne sprawdzanie uprawnień użytkownika, bez modyfikowania logiki właściwych serwisów. Dodatkowo proxy pełni rolę prostego mechanizmu cache.

Role wzorca i klasy

- Subject: IItemAccess
- RealSubject: implementacja IItemAccess (serwis właściwy)
- Proxy: ItemAccessProxy

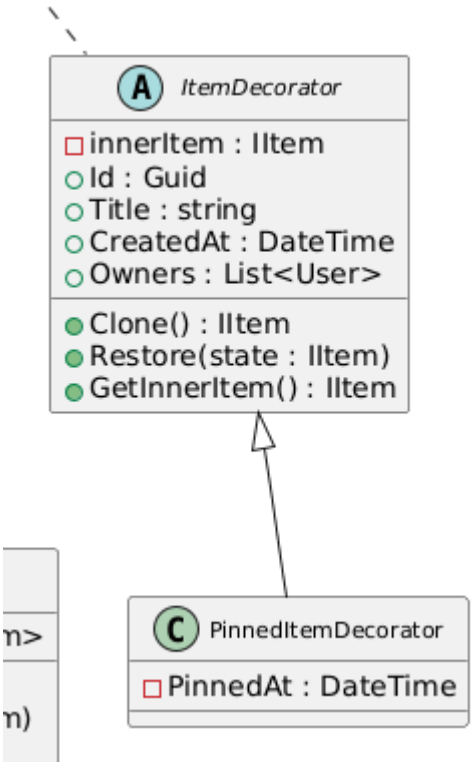
Lokalizacja w kodzie

- Definicja:
  TaskApp/Access/ItemAccessProxy.cs
- Użycie:
  Tworzenie obiektu proxy w ItemManager, wszystkie operacje na itemach przechodzą przez proxy

Wektor zmian

- możliwość dodania ról użytkowników (np. admin, read-only)
- rozbudowa cache o strategię wygaszania lub synchronizacji
- logowanie dostępu do zasobów

Decorator (rozszerzanie zachowania IItem bez modyfikacji klas bazowych)



Cel użycia

- Wzorzec Decorator służy do dodania dodatkowych cech i zachowań do obiektów implementujących IItem bez zmiany ich klas (Note, Tasky, ItemGroup).
- W projekcie wykorzystano go do oznaczania „przypiętych” elementów (pin), przechowywania czasu przypięcia i transparentnego operowania dalej na tym samym interfejsie IItem.
- Dzięki dekoratorowi UI/logika wyświetlania i sortowania mogą reagować na „pin” bez ingerencji w logikę elementów bazowych.

Role wzorca i klasy

- Component: IItem
- ConcreteComponent: Note, Tasky, ItemGroup
- Decorator: ItemDecorator
- ConcreteDecorator: PinnedItemDecorator

Lokalizacja w kodzie

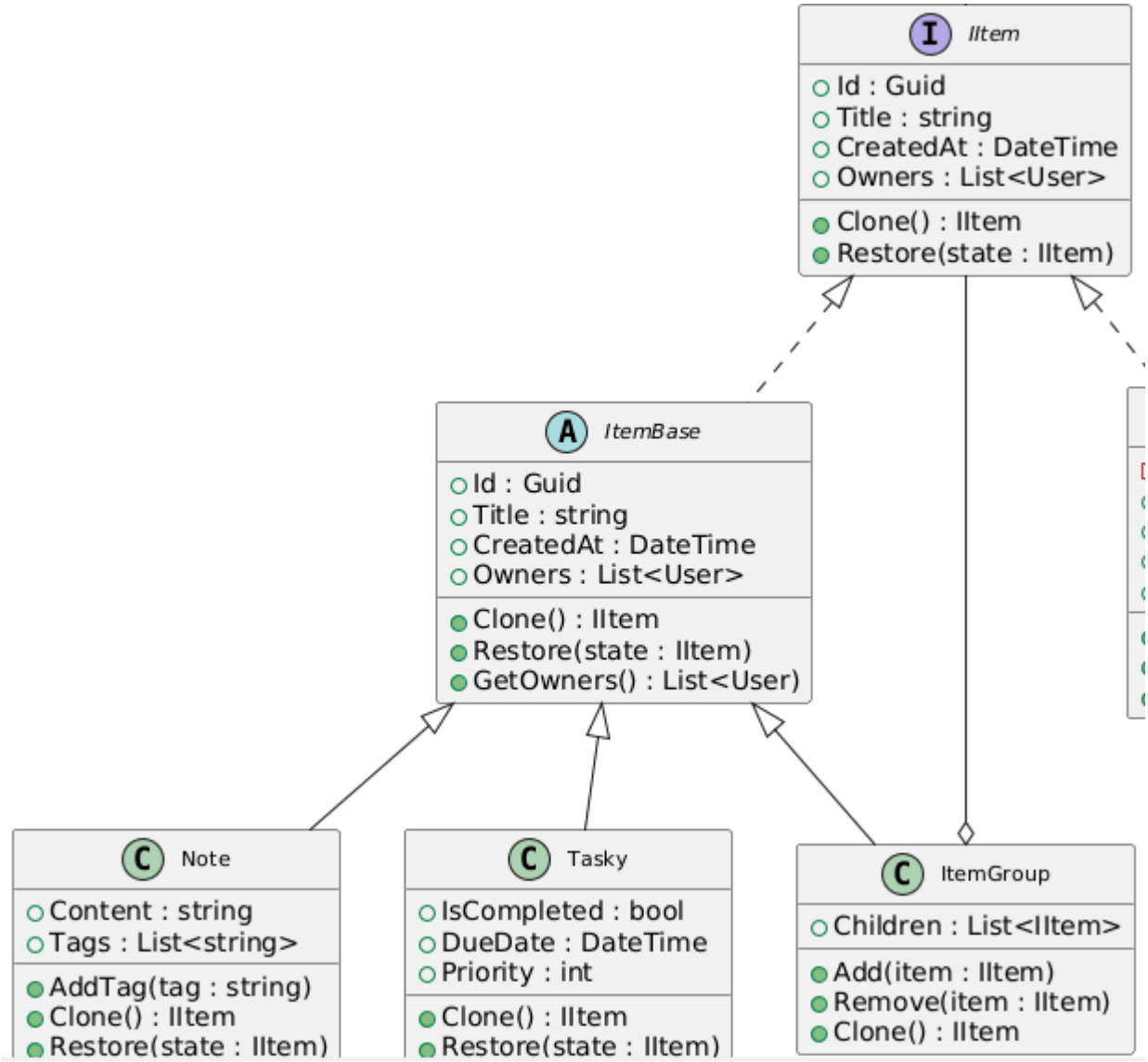
- Definicja:
  - ItemDecorator.cs
  - PinnedItemDecorator.cs
- Użycie:

- Pinowanie/odpinanie przez komendę: PinItemCommand.cs — opakowuje IItem w PinnedItemDecorator i potrafi cofnąć operację.
- Formatowanie w UI: metoda `FormatItemForDisplay` w Program.cs rozpoznaje PinnedItemDecorator i dodaje prefix `[PIN]`.
- Logika listowania/sortowania: ItemQueryService.cs potrafi priorytetyzować przypięte notatki przy wypisywaniu.
- Operacje fasady: metody `PinItem`/`UnpinItem` w TaskAppFacade.cs delegują do komend pin/unpin.

Wektor zmian

- Dodanie kolejnych dekoratorów (np. ArchivedItemDecorator, EncryptedItemDecorator, TaggedItemDecorator) — kaskadowe nakładanie wielu dekoratorów.
- Ujednolicenie obsługi „zdjęcia" dekoratorów w warstwie prezentacji i zapytań (rozpoznawanie łańcucha dekoratorów).
- Rozszerzenie Clone/Restore o pełne wsparcie dla złożonych dekoratorów (zachowanie metadanych przy klonowaniu).
- Integracja z repozytorium i cache: traktowanie dekoratorów jako przezroczystych opakowań w filtrach/sortowaniu oraz przy współdzieleniu (share).
- Rozszerzenie sortowania/drukowania o reguły dla różnych typów dekoratorów (np. „pinned first", „archived last").

Composite (foldery jako złożone obiekty IItem)



Cel użycia:

Wzorzec Composite umożliwia jednolite traktowanie pojedynczych elementów (Note, Tasky) oraz ich grup (ItemGroup) poprzez wspólny interfejs IItem.

W projekcie wzorzec został użyty do budowy hierarchii folderów, które mogą zawierać zarówno elementy proste, jak i inne foldery. Dzięki temu operacje takie jak dodawanie, usuwanie, udostępnianie, wyświetlanie czy klonowanie mogą być wykonywane w jednakowy sposób, niezależnie od tego, czy dotyczą pojedynczego elementu czy całego drzewa.

Role wzorca i klasy

- Interface: IItem
- Leaf: Note, Tasky
- Composite:
  ItemGroup
  – przechowuje Children: List<IItem>
  – udostępnia operacje Add, Remove
  – wspiera klonowanie (Clone)
  – reprezentuje folder w strukturze drzewa

Lokalizacja w kodzie

- Definicja:

  ItemGroup.cs, IItem.cs, ItemBase.cs, Note.cs, Tasky.cs

- Użycie:

Operacje na folderach w fasadzie:

- TaskAppFacade.cs
  - CreateFolder
  - AddFolderToFolder
  - GetRootFolders
  - AddItemToFolder
  - RemoveItemFromFolder
  - ViewFolder
  - ShareFolder
  - DeleteFolder

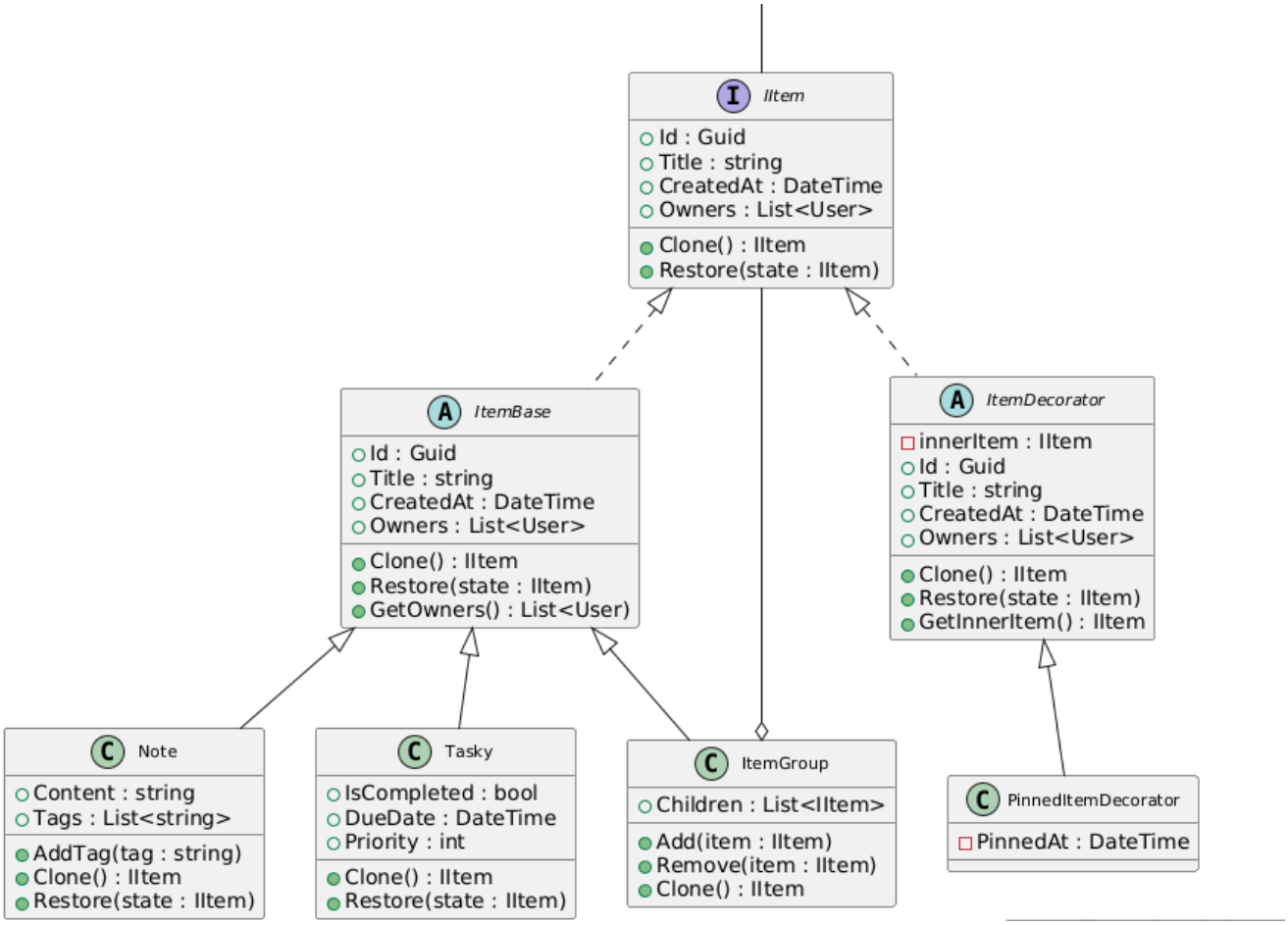
Interakcje UI (menu folderów):

- Program.cs
  - CreateFolder
  - AddItemToFolder
  - RemoveItemFromFolder
  - ShareFolder
  - ViewFolder
  - DeleteFolder
  - AddFolderToFolderConsole

Aktualizacje struktury kompozytu:

- wywołania itemManager.UpdateItem(folder) po modyfikacjach dzieci (TaskAppFacade.cs)

Prototype (Klonowanie obiektów)



Cel użycia:

Wzorzec prototype umożliwia tworzenie wiernych kopii obiektów (notatki, zadania i foldery) bez uzależniania kodu klienta od ich konkretnych klas. Ponadto wykorzystywany jest do wykonywania snapshotów stanu obiektu przed zmianą, co umożliwia działanie Undo/Redo.

Role wzorca i klasy

- Interface: IItem
- Definiuje metodę Clone()
- Prototype:
    - Tasky, Note – implementują klonowanie własnych pól.
    - ItemGroup – implementuje klonowanie rekurencyjnej dla całej struktury drzewiastej.
    - Decorator – przekazują wywołanie klonowania do obiektu dekorowanego.

Lokalizacja w kodzie

- Definicja:

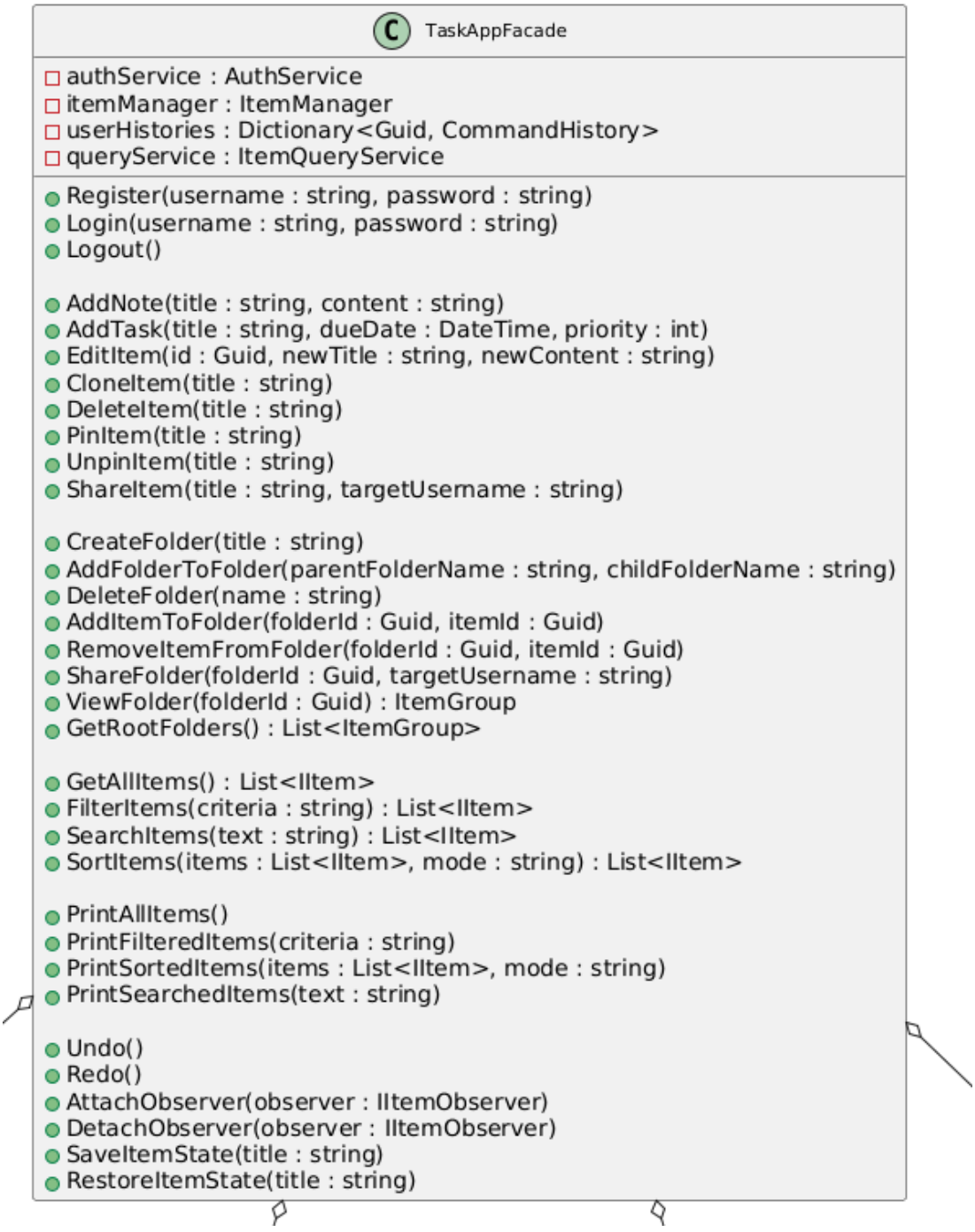
  IItem.cs, Tasky.cs, Note.cs, ItemGroup.cs, ItemDecorator.cs

- Użycie:
    - CloneItemCommand.cs – tworzy kopię elementu i dodaje dopisek (Copy) po czym umieszcza go w repozytorium.
    - SaveStateCommand.cs – używa Clone() do stworzenia obiektu ItemMemento.
    - RestoreStateCommand.cs – używa Clone() do zabezpieczenia obecnego stanu przed przywróceniem kopii zapasowej.

Wektor zmian

1. Klonowanie z możliwością modyfikacji (np. jak klonujemy zadanie to byłaby możliwość modyfikacji pola dueDate).
2. Rozróżnienie intencji klonowania.

Facade (interfejs)



Cel użycia:

Wzorzec Facade został zastosowany w celu dostarczenia uproszczonego i jednolitego interfejsu dla UI użytkownika. Jego zadaniem jest ukrycie złożoności interkacji między wieloma komponentami aplikacji.

Rola wzorca i klasy

- Facade: TaskAppFacade – główny punkt dostępu dla aplikacji
- Subsystem Classes:
  - AuthService – zarządzanie sesją i logowaniem.
  - ItemManager – operacje na itemach i powiadomienia.
  - CommandHistory – zarządzanie stosem poleceń.
  - ItemQueryService – filtrowanie i wyszukiwanie.
- Klient: Program – UI, korzystające wyłącznie z fasady.

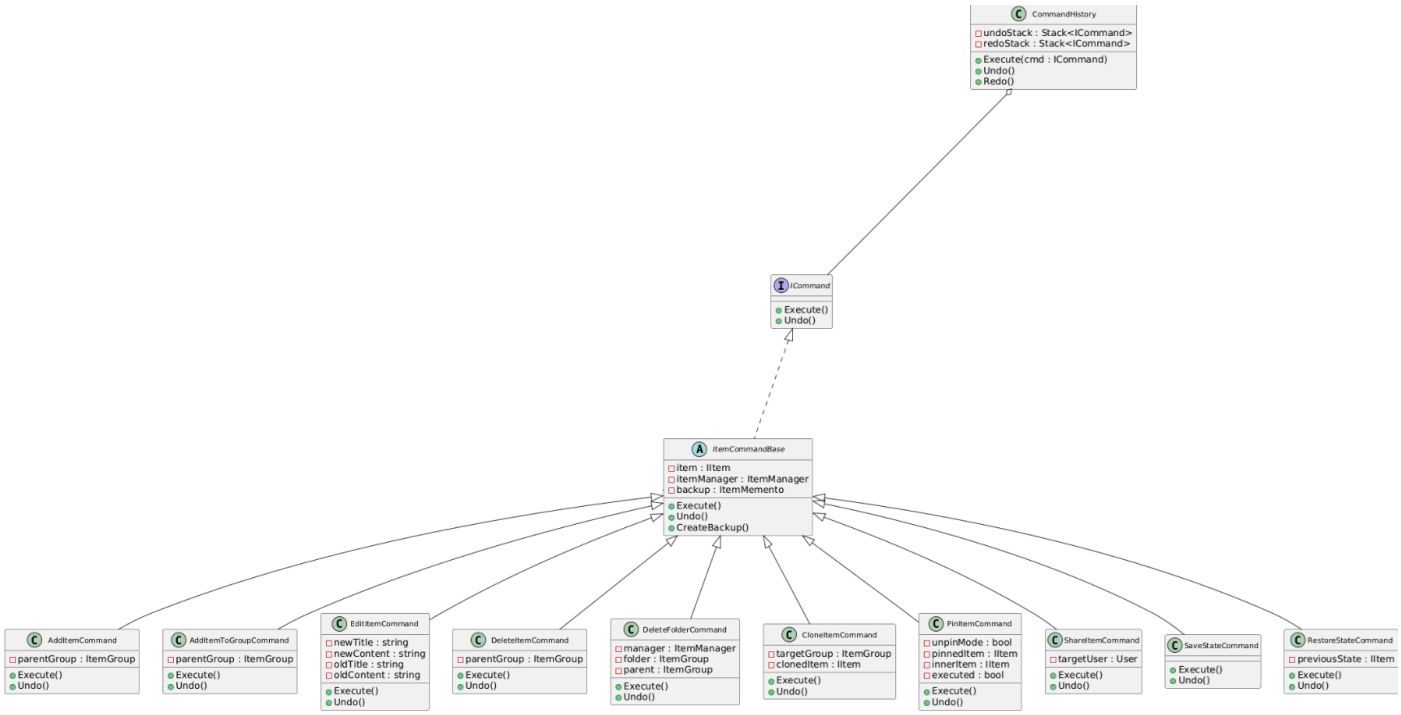
Lokalizacja w kodzie

- Definicja:
  TaskAppFacade.cs
- Użycie:
  - Program.cs – w mainie aplikacji tworzona jest instancja TaskAppFacade, potem przekazywana jest ona do wszystkich metod obsługujących menu.

Wektor zmian

1. Podział fasady (aby ta nie stała się ogromna dobrym pomysłem jest podzielenie jej na mniejsze fasady)
2. Przeniesienie obsługi wyjątków z maina do fasady.

Command



Cel użycia:

Wzorzec Command został zastosowany w celu hermetyzacji wszystkich operacji zmieniających stan aplikacji (dodawanie, edycja, usuwanie, pinowanie, klonowanie) w postaci obiektów. Głównym celem jest umożliwienie implementacji mechanizmu Undo/Redo (cofania i ponawiania zmian) oraz oddzielenie obiektu inicjującego operację (TaskAppFacade) od obiektu, który faktycznie wykonuje logikę biznesową (ItemManager).

Rola wzorca i klasy:

Interface: ICommand – definiuje wspólny interfejs dla wszystkich poleceń.

ItemCommandBase – abstrakcyjna klasa, która standaryzuje implementację komend operujących na elementach.

Concrete Commands:

AddItemCommand – dodawanie itemów,

DeleteItemCommand – usuwanie itemów,

EditItemCommand – edycja itemów,

PinItemCommand – przypinanie itemów,

ShareItemCommand - udostępnianie itemów innym użytkownikom,

CloneItemCommand – klonowanie itemów,

DeleteFolderCommand – usuwanie folderów,

AddItemToGroupCommand – dodawanie itemów do grup,

SaveStateCommand – zapisywanie stanu itemów(tworzennie backupu),

RestoreStateCommand – przywracanie itemów(przywrócenie wersji itemu z backupu itemu)

Odbiorca: ItemManager (oraz ItemGroup dla operacji wewnątrz folderów) – zawiera faktyczną logikę biznesową operacji. Komendy delegują pracę do tych klas.

Wywołujący: CommandHistory – przechowuje historię wykonanych poleceń (stosy undoStack i redoStack) i zarządza ich wywoływaniem.

Klient: TaskAppFacade – tworzy i konfiguruje konkretne obiekty poleceń, a następnie przekazuje je do CommandHistory w celu wykonania.

Lokalizacja w kodzie

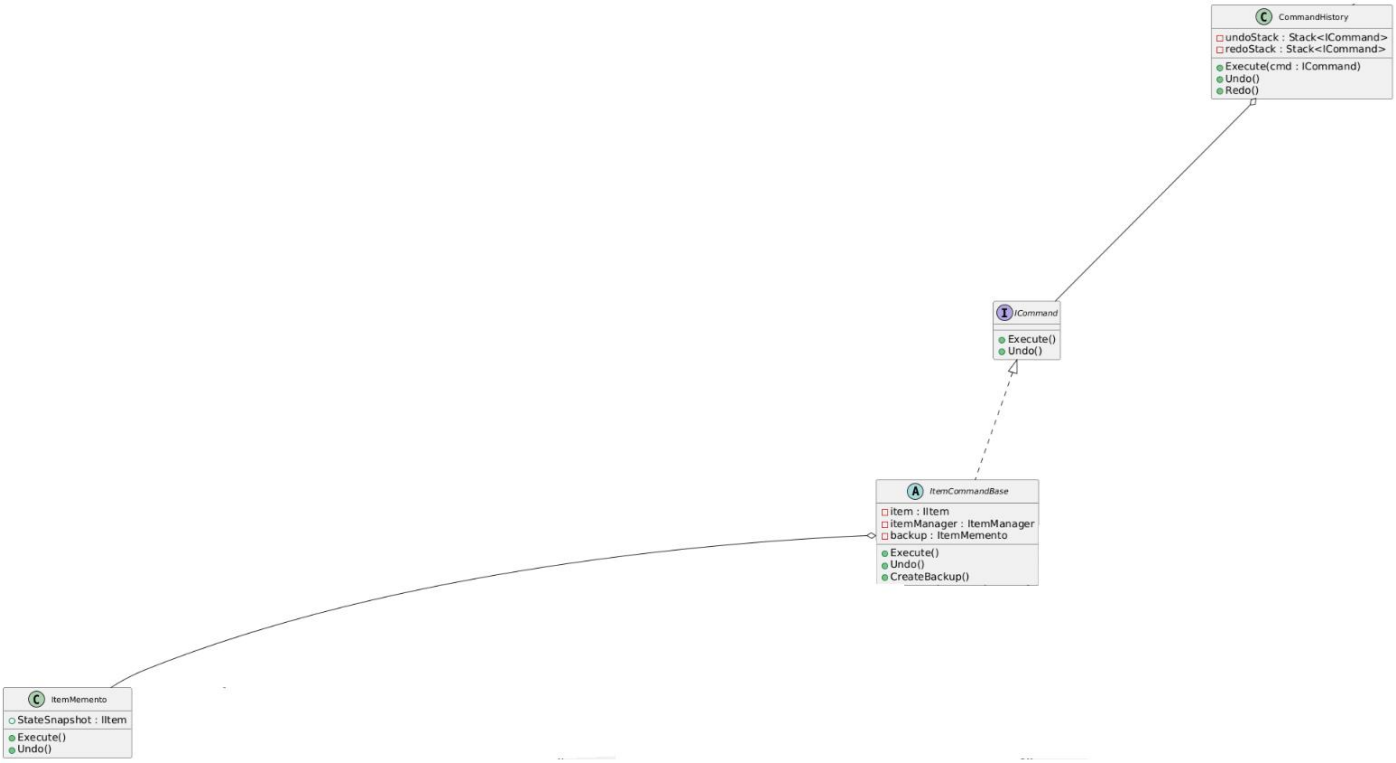
Definicja: ICommand.cs, ItemCommandBase.cs, CommandHistory.cs,

Użycie: TaskAppFacade.cs – metody modyfikujące dane (np. AddNote, EditItem) tworzą instancję odpowiedniej komendy i wywołują ją poprzez metodę GetHistoryForCurrentUser().Execute(command).

Wektor zmian:

- Możliwość wyboru, którą zmianę chcemy cofnąć(obecnie cofana jest ostatnia zmiana)

Memento



Cel użycia: Przechwytywanie i przechowywanie wewnętrznego stanu obiektu bez naruszania jego hermetyzacji. Umożliwia przywrócenie obiektu do stanu poprzedniego w ramach mechanizmu Undo/Redo oraz ręczne tworzenie punktów przywracania (backup).

Role wzorca i klasy:

Memento: ItemMemento – kontener przechowujący migawkę stanu (StateSnapshot).

Originator: IItem (oraz Tasky, Note) – tworzy swoją kopię i potrafi odtworzyć swój stan metodą Restore.

Caretaker:

ItemCommandBase – zarządza pamiątką na potrzeby operacji Undo.

ItemManager – zarządza pamiątkami na potrzeby trwałego backupu (backups).

Lokalizacja w kodzie

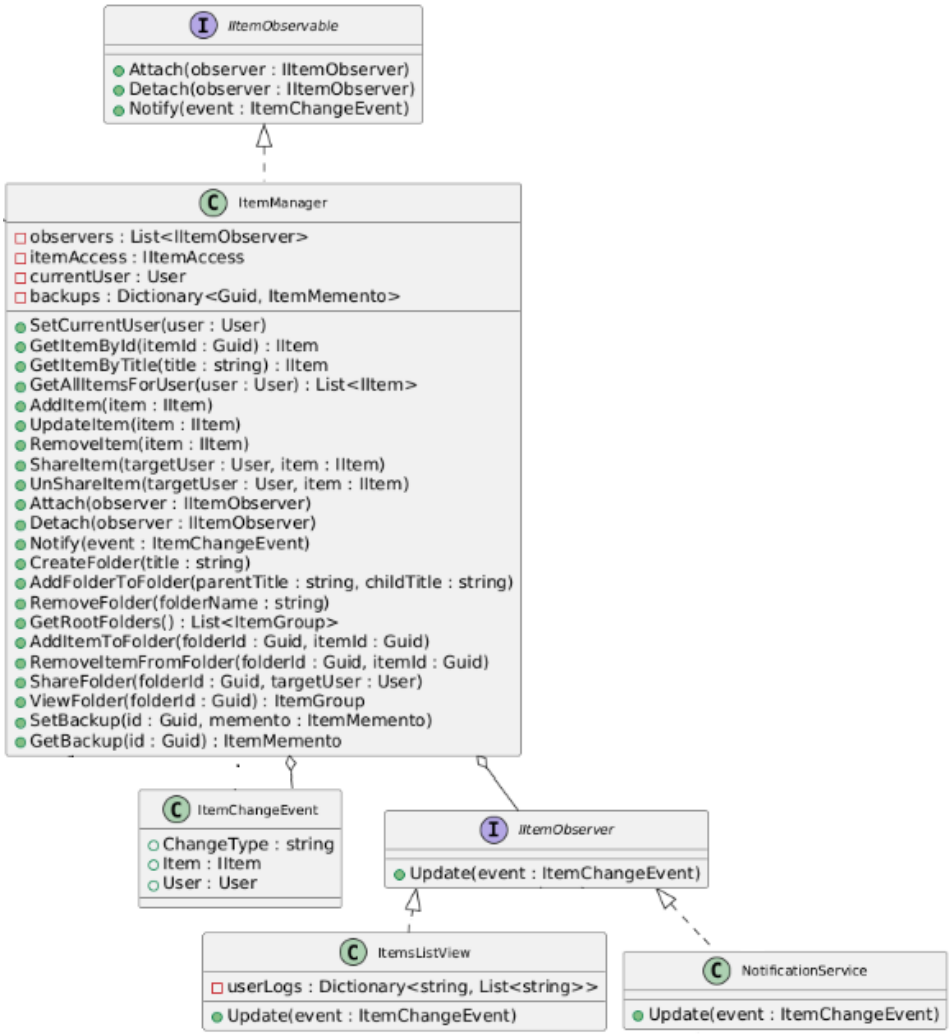
Definicja: ItemMemento.cs, IItem.cs (metoda Restore), ItemBase.cs.

Użycie: SaveStateCommand.cs – zapis stanu do Managera. RestoreStateCommand.cs – odczyt stanu i przywrócenie.

Wektor zmian:

- Przechowywanie tylko różnic zamiast pełnych kopii obiektów.

Wzorzec Obserwator (Observer)



Wzorzec został zastosowany w celu odseparowania logiki (ItemManager) od warstwy prezentacji (widok listy) oraz systemu powiadomień. Dzięki temu dodawanie nowych reakcji na zmiany w systemie nie wymaga modyfikacji kodu zarządzającego przedmiotami.

INTERFACE IItemObserver

- Update(evt: ItemChangeEvent): void – metoda wywoływana w momencie wystąpienia zmiany w obserwowanym obiekcie.

INTERFACE IItemObservable

- Attach(observer: IItemObserver): void – dodaje nowego obserwatora do listy subskrybentów.
- Detach(observer: IItemObserver): void – usuwa obserwatora z listy.
- Notify(evt: ItemChangeEvent): void – powiadamia wszystkich zarejestrowanych obserwatorów o zdarzeniu.

ItemManager (Subject) Implementuje IItemObservable. Zarządza stanem zadań i notatek, powiadamiając subskrybentów o każdej operacji modyfikującej dane.

- observers: List<IItemObserver> – lista zarejestrowanych słuchaczy.
- AddItem(item: IItem): void – po dodaniu elementu wywołuje Notify ze statusem "DODANO".
- UpdateItem(item: IItem): void – po edycji wywołuje Notify ze statusem "ZAKTUALIZOWANO".

- RemoveItem(item: IItem): void – po usunięciu wywołuje Notify ze statusem "USUNIĘTO".
- ShareItem(targetUser: User, item: IItem): void – po nadaniu uprawnień wywołuje Notify z dynamicznym statusem zawierającym nazwę odbiorcy ("UDOSTĘPNIONO DLA: {user}").
- UnShareItem(targetUser: User, item: IItem): void – po odebraniu uprawnień wywołuje Notify z informacją o zakończeniu współdzielenia ("PRZESTANO UDOSTĘPNIAĆ DLA: {user}").

NotificationService (Concrete Observer) Implementuje IItemObserver. Odpowiada za symulację systemu powiadomień.

- Update(evt: ItemChangeEvent): void – wyświetla komunikat wraz z informacją, który użytkownik dokonał zmiany.

ItemsListView (Concrete Observer) Implementuje IItemObserver. Symuluje widok listy zadań, który musi zostać odświeżony po zmianie danych.

- userLogs: Dictionary<string, List<string>> – struktura przechowująca pełną historię operacji dla każdego użytkownika w sesji.
- Update(evt: ItemChangeEvent): void – Rejestruje nowe zdarzenie w słowniku userLogs. Pobiera i wyświetla 3 ostatnie akcje bieżącego użytkownika (Live Feed), tworząc dynamiczny podgląd aktywności nad listą zadań.

ItemChangeEvent Klasa transportowa przekazywana w metodzie Update.

- ChangeType: string – typ operacji (np. "DODANO", "USUNIĘTO").
- Item: IItem – przedmiot, którego dotyczy zmiana.
- User: User – użytkownik inicjujący zmianę.

Integracja z Fasadą (TaskAppFacade): Fasada udostępnia metody AttachObserver i DetachObserver, delegując je do instancji ItemManager. Dzięki temu konfiguracja obserwatorów odbywa się w warstwie inicjalizacji aplikacji (Program.cs), a ItemManager pozostaje niezależny od konkretnych implementacji widoków.

Opis użytych rozwiązań

- List<T> - Generyczna lista — dynamiczna kolekcja uporządkowana (sekwencja) zapewniająca indeksowany dostęp do elementów, możliwość dodawania/usuwania, iteracji i sortowania. Typ generyczny T gwarantuje bezpieczeństwo typów w czasie kompilacji.
- Dictionary<TKey, TValue> - Generyjny słownik (mapa) przechowujący pary klucz–wartość. Umożliwia szybkie (średnio O(1)) wstawianie i wyszukiwanie po kluczu. Klucze muszą być unikalne; wykorzystuje równość i haszowanie typu klucza.
- HashSet<T> - Zbiór bez duplikatów, optymalny do sprawdzania przynależności elementu i eliminacji powtórzeń. Operacje dodawania i sprawdzania zawartości są średnio O(1). Nie przechowuje kolejności.
- Nullable reference types - Funkcja C# 8+ zmieniająca domyślne traktowanie typów referencyjnych jako nienullowalnych. Dodanie `?` (np. `string?`) oznacza, że referencja może

być `null`, a kompilator ostrzega o potencjalnych problemach. Często stosowane wraz z operatorami `??` (null-coalescing) i `?.` (null-conditional) do bezpiecznej pracy z `null`.

- LINQ (Language Integrated Query) - Zestaw metod rozszerzających dla `IEnumerable<T>` i zapytaniowy „język” w C#, umożliwiający deklaratywne operacje na kolekcjach: filtrowanie (`Where`), projekcję (`Select`), agregacje, sortowanie (`OrderBy`), wyszukiwanie (`Any`, `FirstOrDefault`) itd. LINQ oferuje leniwą (odroczoną) ewaluację do momentu materializacji (np. `ToList()`), co pozwala wydajnie łączyć operacje na strumieniach danych.

Podział pracy

Miłosz Szymczuk: Implementacja wzorca Proxy, utworzenie szkieletu aplikacji, zaimplementowanie mechanizmu uwierzytelniania użytkowników, przygotowanie code review.

Jakub Komorowski: Implementacja wzorca Prototype, implementacja funkcji shareowania (tylko dla plików), utworzenie menu (UI konsolowego).

Miłosz Pawlaczyk: Implementacja wzorca Observer, implementacja obsługi wyjątków.

Paweł Gronostajski: Implementacja wzorca Decorator, implementacja ItemQueryService (filtrowanie, szukanie, sortowanie, wypisywanie wszystkich itemów.)

Jan Kozłowski: Implementacja wzorca Command, implementacja wzorca Memento.

Adrian Lachowicz: Implementacja wzorca Composite, Implementacja mechanizmu folderów

Instrukcja użytkownika

Instrukcja użytkowania programu (perspektywa użytkownika)

Ekran startowy Po uruchomieniu zobaczysz menu główne:

1. Register
2. Login
3. Exit

Rejestracja

1. Wybierz 1 (Register).
2. Podaj Username i Password.
3. Jeśli nazwa użytkownika jest wolna, konto zostanie utworzone. W przeciwnym razie pojawi się komunikat o błędzie.

Logowanie i wylogowanie

- Logowanie:
    - Wybierz 2 (Login).
    - Podaj Username i Password.
    - Po sukcesie zobaczysz menu użytkownika.
- Wylogowanie:
    - Z menu użytkownika wybierz 0 (Logout).

Menu użytkownika po zalogowaniu zobaczysz:

1. Add Note/Task
2. Show my items
3. Edit item
4. Share item
5. Pin item
6. Clone item
7. Folder management
8. Delete item
9. Backup Item
10. Restore Item
0. Logout

Dodawanie notatki

1. Wybierz 1 (Add Note/Task), potem 1 (Add Note).
2. Podaj Title oraz Content.
3. Po sukcesie otrzymasz potwierdzenie „Note added successfully”.

Dodawanie zadania

1. Wybierz 1 (Add Note/Task), potem 2 (Add Task).
2. Podaj Title.
3. Podaj Due date w formacie YYYY-MM-DD (np. 2026-02-15). Jeśli wpiszesz nieprawidłową datę, aplikacja ustawi termin na jutro.
4. Podaj Priority jako liczbę całkowitą (jeśli wpis nie jest liczbą, ustawi się 1).
5. Zadanie zostanie dodane.

Przeglądanie elementów

- Wybierz 2 (Show my items).
- Zobaczysz listę wszystkich swoich elementów. Format:
    - [PIN] [Completed]/[Not completed] Tytuł (Due: RRRR-MM-DD, Priority: X) — dla zadań
    - [PIN] [Note] Tytuł: Treść — dla notatek
    - [PIN] [Folder] Tytuł (Items: N) — dla folderów
- Naciśnij dowolny klawisz, aby wrócić.

Edycja elementu

1. Wybierz 3 (Edit item).
2. Podaj Title elementu do edycji.
3. Jeśli jest wiele elementów o tej nazwie, wybierz właściwy z listy.
4. Podaj New Title (nowy tytuł).
5. Podaj New Content — dotyczy tylko notatek, dla zadań jest ignorowane.
6. Otrzymasz komunikat „Item updated successfully”.

Klonowanie elementu

1. Wybierz 6 (Clone item).
2. Podaj Title elementu do sklonowania.
3. Jeśli istnieje wiele pasujących, wybierz z listy.
4. Po sukcesie: „Item cloned successfully.”

Przypinanie elementu (Pin)

1. Wybierz 5 (Pin item).
2. Podaj Title elementu do przypięcia.
3. Po sukcesie element w listach będzie miał prefix [PIN]. Uwaga: W menu nie ma opcji „Unpin”. Odpinanie nie jest dostępne z poziomu interfejsu użytkownika.

Udostępnianie elementu innemu użytkownikowi

1. Wybierz 4 (Share item).
2. Podaj Title elementu do udostępnienia.
3. Podaj username użytkownika, któremu chcesz udostępnić. Wymagania i uwagi:
    - Konto docelowe musi istnieć.
    - Nie można udostępnić elementu samemu sobie.
    - Jeśli element jest już udostępniony temu użytkownikowi, pojawi się błąd.

- Gdy masz wiele elementów o tej samej nazwie, aplikacja wybierze pierwszy znaleziony — warto unikać duplikatów tytułów.

Zarządzanie folderami (Folder management — opcja 7) Po wejściu do tego menu dostępne są:

1. Create folder
2. Add item to folder
3. Remove item from folder
4. Share folder
5. View folder
6. Delete folder
7. Add folder to folder
8. Back

1. Create folder
   - Podaj tytuł folderu (nie może być pusty ani powielony wśród Twoich folderów).
   - Folder zostanie utworzony.
2. Add item to folder
   - Zobaczysz listę dostępnych folderów.
   - Wpisz nazwę folderu, do którego chcesz dodać plik.
   - Otrzymasz listę „Files available to add” (wszystkie elementy niebędące folderami i niebędące już w tym folderze).
   - Wpisz nazwę pliku do dodania.
   - Po sukcesie: „File 'X' added to folder 'Y'.”
3. Remove item from folder
   - Wybierz folder z listy, potem nazwę elementu wewnątrz.
   - Po sukcesie element zostanie usunięty z danego folderu.
4. Share folder
   - Wybierz z listy folder, wpisz jego nazwę.
   - Podaj username odbiorcy.
   - Folder zostanie udostępniony (udział niekoniecznie propaguje się na wszystkie dzieci — zależy od logiki kontroli dostępu; pamiętaj, że dostęp jest weryfikowany przy odczycie).
5. View folder
   - Wpisz nazwę jednego z „root folders” (folderów niebędących dziećmi innych folderów).
   - Zobaczysz listę elementów w tym folderze.
6. Delete folder
   - Wpisz nazwę folderu do usunięcia (działa dla folderów głównych i jednego poziomu dzieci).
   - Folder zostanie usunięty.
7. Add folder to folder
   - Wybierz folder nadrzędny z listy „Available parent folders:” (root folders).
   - Podaj nazwę nowego folderu potomnego.
   - Nowy folder zostanie dodany jako dziecko.

Formaty danych i wskazówki

- Data: wpisuj jako yyyy-MM-dd (np. 2026-01-16).
- Priorytet: liczba całkowita; domyślnie 1, jeśli podasz wartość nieprawidłową.
- Unikaj duplikatów tytułów — ułatwia to edycję, udostępnianie i przypinanie.
- Widoczne prefiksy:
    - [PIN] — element przypięty.
    - [Completed]/[Not completed] — status zadania (w aplikacji brak opcji zmiany statusu z menu).
    - [Note]/[Folder] — typ elementu.

Usuwanie notatki/zadania:

- Wybierz 8 (Delete item).
- Wprowadź tytuł notatki/zadania do usunięcia.
- Jeśli istnieje wiele pasujących to wybierz odpowiadającą notatkę/zadanie z listy.
- Powinien wyświetlić się komunikat "Item deleted successfully".

Tworzenie kopii notatki/zadania:

- Podać nazwę notatki
- Wyświetla się komunikat Backup “saved in storage/State of 'x' has been saved.”

Przywracanie kopii notatki/zadania:

- Podać nazwę notatki
- Item 'x' restored from backup.

Wyjście z programu

- W menu głównym wybierz 3 (Exit). W menu użytkownika wybierz 0 (Logout), a następnie 3 (Exit) w ekranie startowym.

Instrukcja instalacji

Instrukcja uruchomienia (Windows):

1. Pobierz plik TaskApp.exe.
2. Uruchom go dwukrotnym kliknięciem.
3. Aplikacja uruchomi się w oknie terminala.