# Dokumentacja techniczna

**Projekt: System zarządzania zadaniami (ToDo CLI)**

---

## 1. Zespół i podział pracy

- **Kuba**

  Odpowiedzialny za:

  - projekt i implementację klas modelu (Task, SpecialTask, TaskManager)
  - obsługę OOP: dziedziczenie, polimorfizm, enkapsulację, własne wyjątki, metody klasowe i statyczne
  - komentarze i testy jednostkowe w modelu

- **Kornel**

  Odpowiedzialny za:

  - projekt i implementację wzorców projektowych (Strategy, Command, Decorator)
  - wyraźne przykłady i integrację wzorców z resztą aplikacji
  - polskie komentarze do wzorców

- **Igor**

  Odpowiedzialny za:

  - interfejs użytkownika (menu tekstowe, CLI)
  - obsługę wejścia/wyjścia, czytelny przepływ danych, komunikaty
  - integrację całości (łączność modelu i wzorców)
  - obsługę wyjątków na poziomie UI

---

## 2. Architektura i pliki

- model.py – klasy OOP (Kuba):

  Task, SpecialTask, TaskManager, TaskNotFoundException

- patterns.py – wzorce projektowe (Kornel):

  SortStrategy (+ warianty), Command (+ warianty), TaskDecorator/TagDecorator

- main.py – interfejs użytkownika i integracja (Igor):

  Menu, pętla główna, obsługa akcji użytkownika, integracja wszystkich elementów

---

## 3. Spełnienie wymagań zadania (punkty z treści kursu):

**Zadanie minimum — Wszystkie punkty z listy spełnione:**

1. **Użycie klas** – Task, SpecialTask, TaskManager, wzorce projektowe
2. **Użycie dziedziczenia** – SpecialTask dziedziczy po Task, TaskDecorator po Task
3. **Użycie atrybutów w klasach; nadpisywanie atrybutów w klasach potomnych** – atrybuty prywatne i dodatkowe w klasach pochodnych
4. **Użycie metod w klasach; nadpisywanie metod w klasach potomnych** – np. display i **str** w SpecialTask, TagDecorator
5. **@classmethod lub @staticmethod** – metoda from_dict (classmethod) i static_example (staticmethod) w Task
6. **Klasa zawierająca więcej niż jeden konstruktor** – Task: konstruktor **init** oraz from_dict (classmethod)
7. **Użycie enkapsulacji (settery i gettery)** – prywatne atrybuty, publiczne metody dostępu we wszystkich klasach

8. **Polimorfizm** – wywołanie display na liście obiektów Task/SpecialTask/Decorator
9. **Użycie implementacji z klasy rodzica: super()** – wywołania super() w konstruktorach klas pochodnych
10. **Własny wyjątek (dziedziczenie z Exception)** – klasa TaskNotFoundException
11. **Trzy wzorce projektowe:**
    - **Strategy** (sortowanie zadań wg różnych kryteriów – Kornel)
    - **Command** (każda akcja użytkownika jako osobna klasa – Kornel)
    - **Decorator** (ozdabianie zadania tagiem – Kornel)

---

# 4. Obsługa programu

- **Uruchomienie:**

  python main.py

- **Główne opcje menu:**
  - Dodawanie, usuwanie, oznaczanie i edycja zadań
  - Wyświetlanie zadań (także posortowanych różnymi strategiami)
  - Dodawanie tagu do zadania (przykład dekoratora)
  - Obsługa wyjątków (np. brak zadania o podanym ID)
- **Każda funkcja programu wywołuje odpowiedni wzorzec, model OOP lub obsługuje błąd zgodnie z dobrymi praktykami.**

---

# 5. Dodatkowe uwagi

- Kod zawiera szczegółowe komentarze po polsku, z podziałem na autorów.
- Wzorce są widoczne i czytelne, każda klasa i metoda jasno opisana.
- Projekt jest łatwo rozwijalny, podzielony na moduły zgodnie z dobrymi praktykami.