

Task list manager

Вступ

Проект являє собою простий консольний додаток для керування календарними задачами, які описують та визначають певну заплановану подію в часі. Користувач має змогу керувати задачами, відслідковувати їхні терміни та додавати їхній короткий опис.

Команда розробників:

Батуркін Іван – студент групи ІП-03

Кирик Матвій – студент групи ІП-03

Цілі проекту

В рамках поставленої задачі ми бачимо сенс реалізувати консольний інтерфейс користувача, який надаватиме можливості:

- Додавати завдання. Консольний інтерфейс надає можливість задати заголовок задачі, короткий опис та дедлайн.
- Видаляти завдання. Користувач може видалити потрібне завдання.
- Редагувати завдання. При цьому наявна можливість змінити заголовок, опис завдання чи дедлайн.
- Позначити завдання виконаним. При додаванні завдання, за замовчування задача є невиконаною, проте коли задача вже стає виконаною користувач відмічатиме її такою.
- Відобразити список завдань. Програма виводи у консоль користувачеві список всіх завдань, що зберігаються у сховищі.

- Відобразити список протермінованих завдань. Програма перевірятиме чи задача є виконаною та чи не сплив її термін, і якщо ці умови справджуються, то ця задача буде відображена.
- Відобразити список завдань відсортованих по дедлайну. Задачі витягатимуться із сховища та відображатимуться у порядку зменшення їхніх термінів.

На відміну від існуючих аналогів цей проект не переслідує такі цілі:

- Інтерактивний режим роботи програми, коли користувач взаємодіє із програмою в реальному часі та надає їй вказівки до виконання. Наша ж програма працюватиме в неінтерактивному режимі, надаючи інтерфейс аргументів командного рядка.
- Середня статистика по виконаних завданнях, з визначення середніх показників тривалості виконання завдання, скільки задач протерміновано, скільки не виконано тощо.
- Підтримка декількох користувачів для взаємодії із програмою

Опис сховища даних

Під час проектування системи було прийнято рішення зберігати програмні сутності задач на файловій системі у форматі [JSON Lines](#) (JSONL).

JSON Lines – це зручний формат для зберігання структурованих даних, які можуть оброблятися по одному запису. Він добре працює з інструментами обробки тексту в стилі Unix і конвеєрами.

Переваги:

- Простий, оскільки базується на форматі JSON, який має значно простішу структуру, ніж альтернативи такі як XML
- Гнучкий для передачі даних між архітектурними рівнями застосунку

- Чудово підходить для моделювання предметної галузі задачі

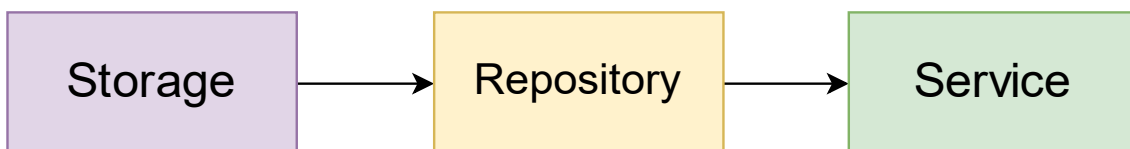
Недоліки:

- Складність представлення ієрархії типів
- Подекуди надмірна простота

Архітектурне рішення

Огляд системи

Архітектура програми розбита за 3 ключовими відповідальностями, які покладені на розроблювану програму:



Архітектурний рівень Storage відповідає за реалізацію програмної структури даних FileList, яка має функціональності файлового сховища даних у форматі JSON Lines, впроваджуючи базові дії над файлами.

Рівень репозиторіїв (Repository) інкапсулює взаємодію із функціями файлового сховища, які репрезентовані структурою даних FileList, та надає зручний інтерфейс для взаємодії із даними, не заглиблюючись у технічну реалізацію.

Service являє собою обгортку над Repository, використовуючи його функціональність, та виконує додаткові дії над результатами роботи Repository аби надавати кінцеві дані програмному інтерфейсу користувача відповідно до функціональних вимог.

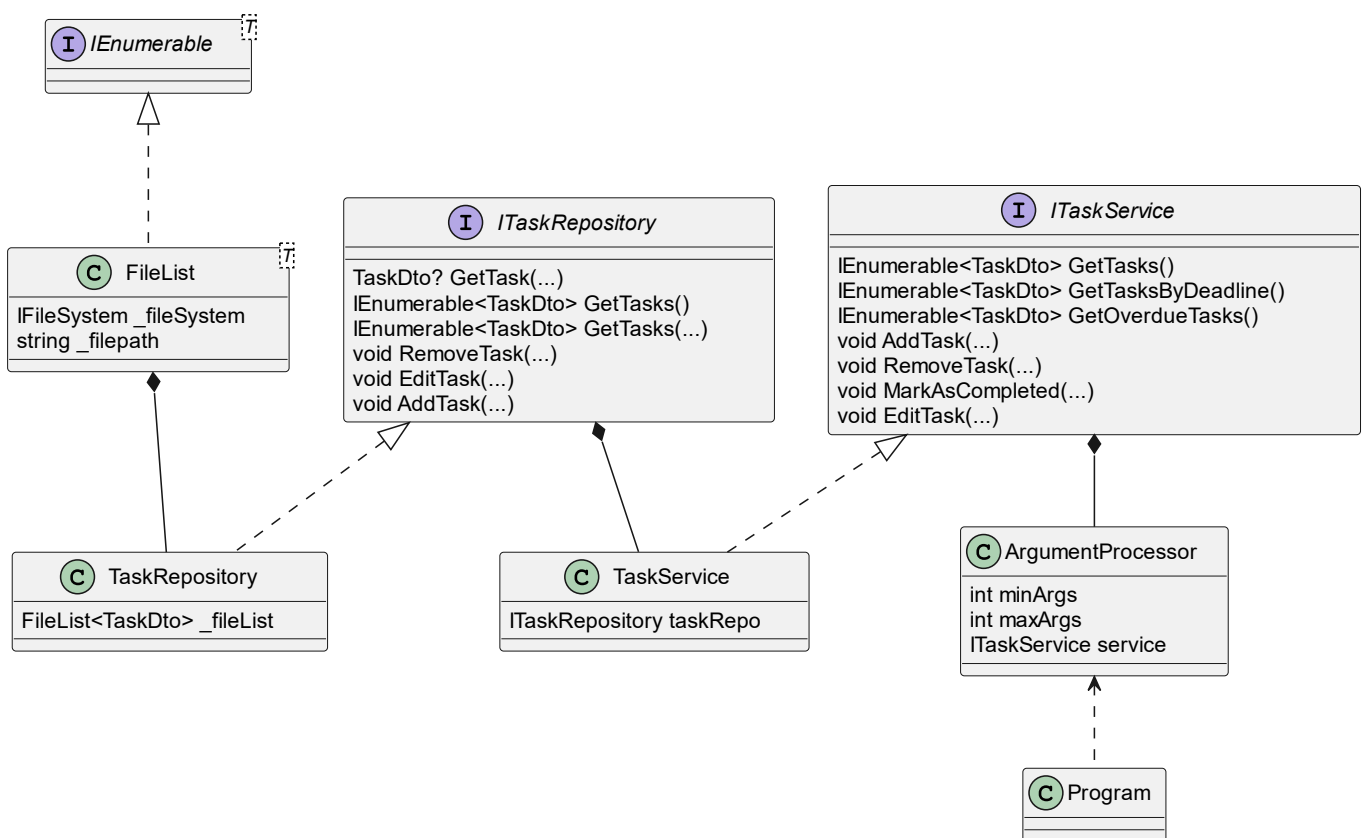
Підсистеми

Загальне рішення проекту розбите на такі підсистеми (проекти):

- ConsoleView (реалізує користувацький інтерфейс командного рядка та оброблення виключних ситуацій, що виникають в роботі програми)
- DataTransfer (містить програмну реалізацію сутностей, з якими взаємодіє система та визначає їхню візуальну репрезентацію)
- FileStorage (відповідає архітектурному рівню Storage)
- Repositories (відповідає архітектурному рівню Repository)
- Services (відповідає архітектурному рівню Service)

Програмні залежності

Відповідно до дизайну архітектури були спроектовані класи основних сутностей, які реалізують логіку роботи застосунку. Інтерфейси відокремлюють реалізацію класів від відповідальностей, які покладені на відповідні їм рівні архітектури.



Інтерфейс командного рядка

Всі можливості користувача щодо оперування над задачами реалізовані у вигляді аргументів командного рядка. Синтаксис відповідних аргументів командного рядка:

1. Для генерації додавання задачі у сховище:

```
dotnet run --add { Title } { Text } { Deadline }
```

2. Для редагування задачі:

```
dotnet run --edit { Id } { Title } { Text } { Deadline }
```

3. Для видалення задачі:

```
dotnet run --remove { Id }
```

4. Для відмічання задачі як виконаної:

```
dotnet run --mark { Id }
```

5. Для відображення списку всіх задач:

```
dotnet run --show
```

6. Для відображення списку задач, відсортованого по дедлайнах:

```
dotnet run --show-ordered
```

7. Для відображення списку протермінованих задач:

```
dotnet run --show-expired
```

Відповідні позначення аргументів командного рядка:

{ Id } – унікальний цілочисельний ідентифікатор задачі.

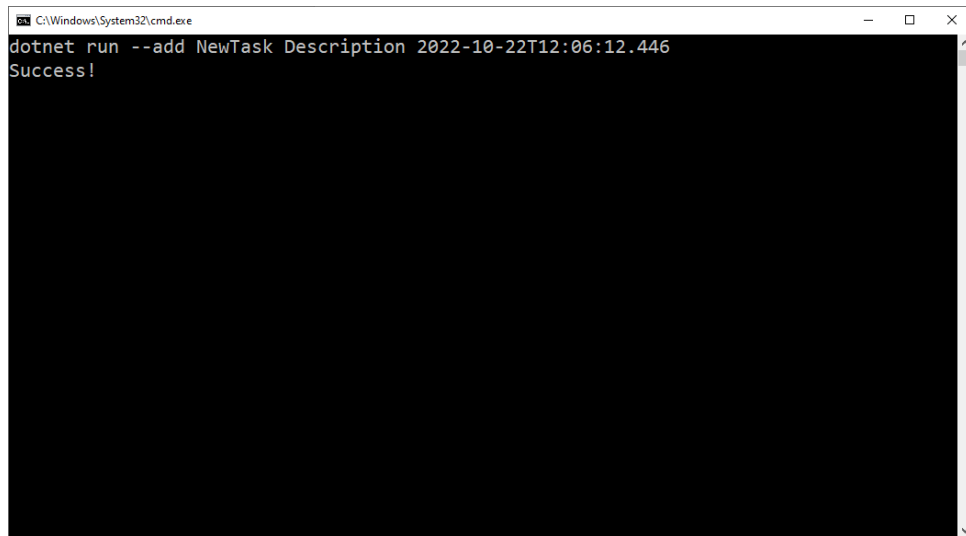
{ Title } – текстовий заголовок задачі.

{ Text } – текстовий опис задачі.

{ Deadline } – термін задачі. Може бути заданий у форматі ISO8601, RFC3339 або YYYY-MM-DD. Повний список підтримуваних форматів відповідає реалізації структури DateTime в .NET 6.0

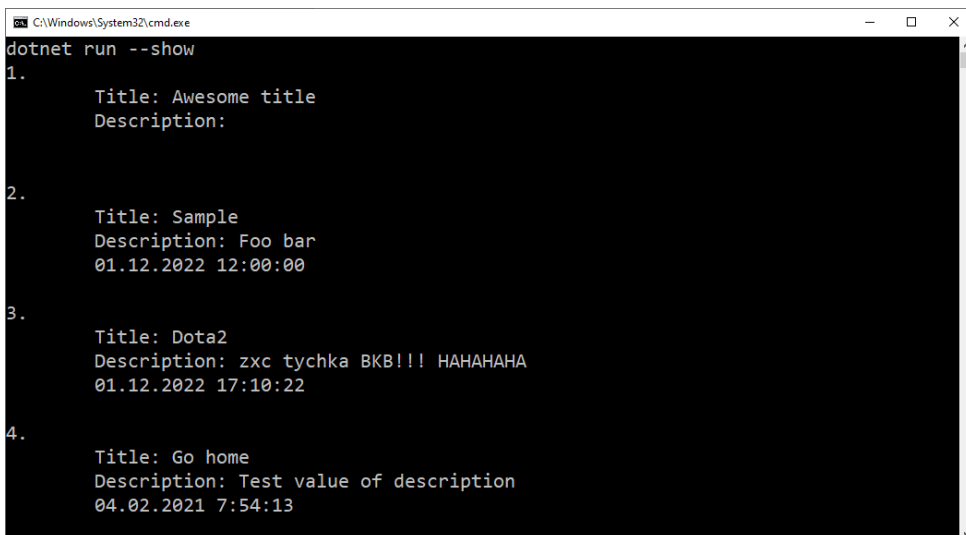
Приклади роботи програми

Приклад сповіщення програмою про додавання задачі:

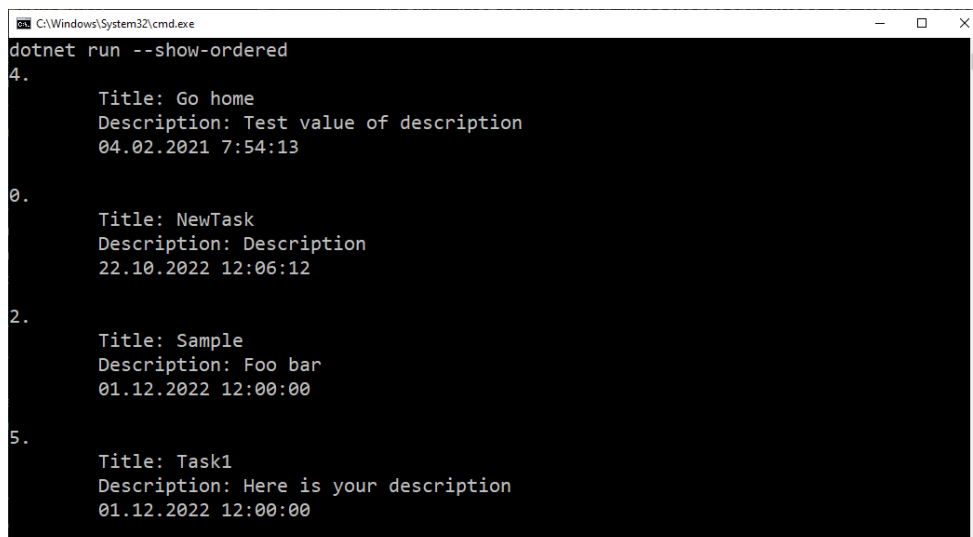


```
C:\Windows\System32\cmd.exe
dotnet run --add NewTask Description 2022-10-22T12:06:12.446
Success!
```

Приклади виведення списку всіх задач:

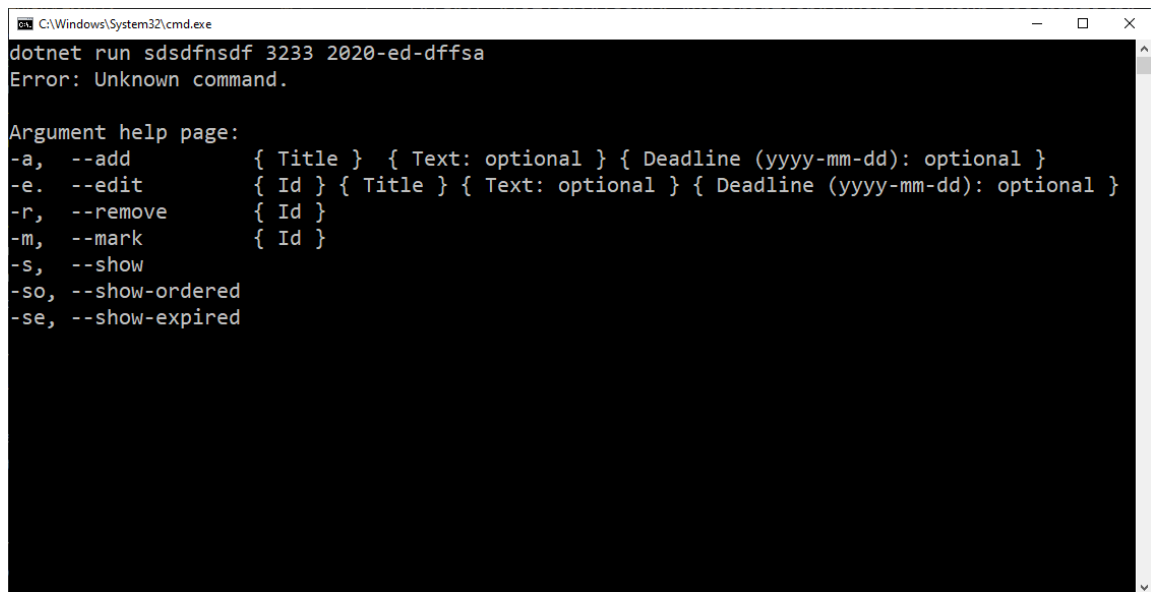


```
C:\Windows\System32\cmd.exe
dotnet run --show
1.
    Title: Awesome title
    Description:
2.
    Title: Sample
    Description: Foo bar
    01.12.2022 12:00:00
3.
    Title: Dota2
    Description: zxc tychka BKB!!! HAHAAHANA
    01.12.2022 17:10:22
4.
    Title: Go home
    Description: Test value of description
    04.02.2021 7:54:13
```



```
C:\Windows\System32\cmd.exe
dotnet run --show-ordered
4.
    Title: Go home
    Description: Test value of description
    04.02.2021 7:54:13
0.
    Title: NewTask
    Description: Description
    22.10.2022 12:06:12
2.
    Title: Sample
    Description: Foo bar
    01.12.2022 12:00:00
5.
    Title: Task1
    Description: Here is your description
    01.12.2022 12:00:00
```

Приклад відображення сторінки допомоги, як результат некоректного вводу вхідних даних користувачем:



```
dotnet run sdsdfnsdf 3233 2020-ed-dffsa
Error: Unknown command.

Argument help page:
-a, --add          { Title } { Text: optional } { Deadline (yyyy-mm-dd): optional }
-e, --edit         { Id } { Title } { Text: optional } { Deadline (yyyy-mm-dd): optional }
-r, --remove       { Id }
-m, --mark         { Id }
-s, --show
-so, --show-ordered
-se, --show-expired
```

Перспективи та висновки

Створений додаток полегшує роботу ведення записів про задачі. Використання JSON Lines є гнучким для передачі даних між різними рівнями архітектури та має простий формат для реалізації списку задач.

Розроблена система:

- дозволяє користувачеві маніпулювати задачами
- має інтерфейс командного рядка для взаємодії із користувачем
- дає можливість відслідковувати терміни поставлених задач, що може стати у нагоді в проблемі тайм-менеджменту

Весь вихідний код застосунку розміщений за [посиланням](#).

Додатки

Вихідний код UML діаграми класів, побудованої в безкоштовному додатку [PlantUML](#):

```
@startuml
interface IEnumerable<T>

interface ITaskService {
    IEnumerable<TaskDto> GetTasks()
    IEnumerable<TaskDto> GetTasksByDeadline()
    IEnumerable<TaskDto> GetOverdueTasks()
    void AddTask(...)
    void RemoveTask(...)
    void MarkAsCompleted(...)
    void EditTask(...)
}

interface ITaskRepository {
    TaskDto? GetTask(...)
    IEnumerable<TaskDto> GetTasks()
    IEnumerable<TaskDto> GetTasks(...)
    void RemoveTask(...)
    void EditTask(...)
    void AddTask(...)
}

class FileList<T> {
    IFileSystem _fileSystem
    string _filepath
}

class TaskRepository {
    FileList<TaskDto> _fileList
}

class TaskService {
    ITaskRepository taskRepo
}

class ArgumentProcessor {
    int minArgs
    int maxArgs
    ITaskService service
}

class Program

IEnumerable <|.. FileList
FileList *-- TaskRepository
ITaskRepository *-- TaskService
ITaskRepository <|.. TaskRepository
ITaskService <|.. TaskService
ITaskService *-- ArgumentProcessor
ArgumentProcessor <.. Program
@enduml
```