**Проектування та моделювання програми для автоматизації виконання команд в терміналі**

**Зміст**

1. Проектування програми
   1. Опис процесу діяльності
   2. Архітектура програмного забезпечення
2. Моделювання та конструювання програмного забезпечення
   1. Детальний опис класів програмита їх методів

**1. Проектування програми**

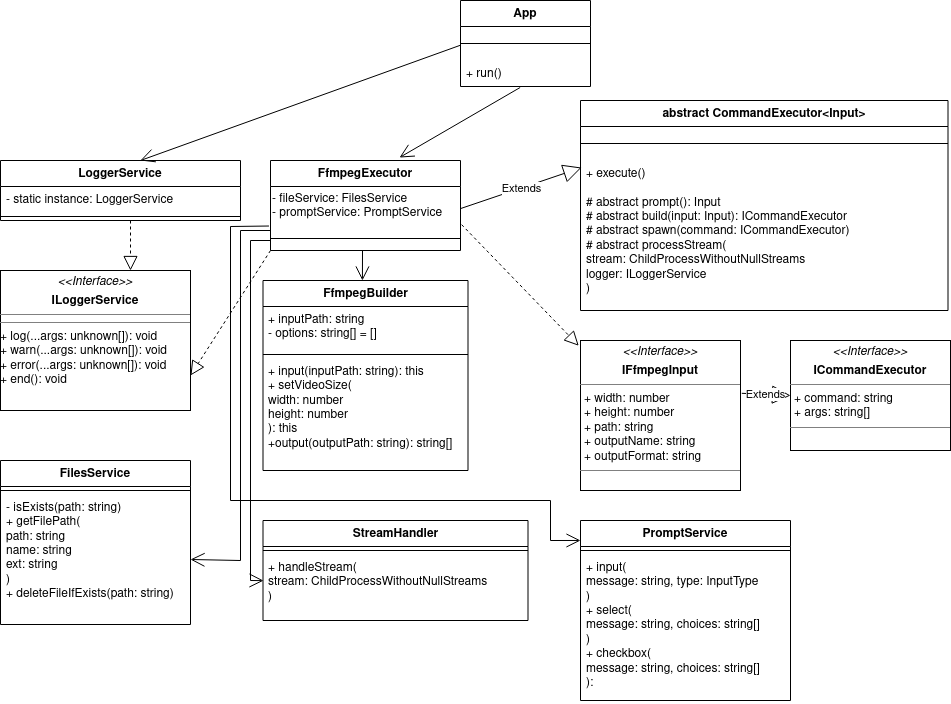
**1.1 Опис процесу діяльності**

У випадку якщо програма буде опублікована в NPM (Node package manager), для запуску можна використати: **npx <ім’я пакета>** (необхідно, щоб була встановлена Node.js на комп’ютері).

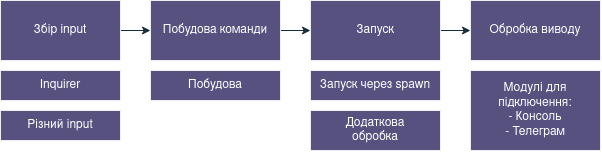
Після запуску з’являється меню, вибираєте необхідну команду для запуску в терміналі. Після вибору команди з’являється відповідне меню, де ви заповнюєте поля.

Далі команду буде виконано враховуючи усі опції які ви ввели у меню.

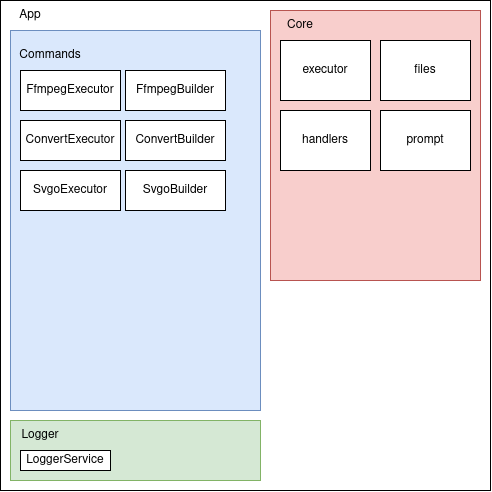
**1.2 Архітектура програмного забезпечення**



*Рис. 1 Діаграма класів*



*Рис.2 Template method pattern процеси*



*Рис.3 Структурна схема класів*

*Таблиця 1 Загальний опис класів та їх призначення*

|  |  |
| --- | --- |
| **Клас** | **Опис та призначення класу** |
| *App* | Корневий клас, що запускає програму |
| *LoggerService* | Сервіс для логування |
| *PromptService* | Власна обгортка над бібліотекою inquirer (ввід в терміналі) |
| *StreamHandler* | Обробка стріма |
| *FilesService* | Робота з шляхами та файлами в ОС |
| *CommandExecutor* | Абстрактний клас, що реалізує template method pattern |
| *\*Builder* | *Білдер класи, це класи які реалізують патерн Builder для збору аргументів з меню* |
| *\*Executor* | *Executor класи, це класи, що місять основну логіку виводу меню, запуску дочірнього процесу та найгловніше – вони екстендять CommandExecutor.* |

**2. Моделювання та конструювання програмного забезпечення**

**2.1 Детальний опис класів програми та їх методів**

**App -** корневий клас, що запускає програму**.**

**Методи:**

* **run() –** при створенні інстансу класу **App** та виклику методу **run()**, створюється інстанс класу **PromptService**, у нього викликається метод **select()**. У термінал виводиться питання: “Яку команду ви хочете обрати”, і нижче, переміщаючись стрілочками на клавіатурі можемо обрати відповідну команду.

Нижче в коді знаходиться **switch**, де в залежності від вибраної команди запускається відповідний клас executor.

**LoggerService –** сервіс логування який імплементує інтерфейс **ILoggerService**

**Методи:**

* **log() –** логування звичайного повідомлення
* **warn()** – попередження
* **error() –** логування при помилці
* **end() –** повідомлення логування при завершені виконання програми
* **getInstance() –** в цьому класі також реалізований патерн **синглтон** для того, щоб постійно не створювати новий інстанс класу, а діставати перший створений.

**CommandExecutor -** абстрактний клас, що реалізує **template method pattern**.

У конструктор приймає логер.

Процеси запуску кожної команди повторюються, лиш кожна команда має свою логіку.

**Методи:**

* **prompt() –** абстрактний метод, реалізацією якого буде логіка вводу з терміналу
* **build() –** абстрактний метод, реалізацією якого буде логіка побудови всієї команди з аргументами
* **spawn() –** абстрактний метод, реалізацією якого буде запуск дочірнього shell процесу для команди
* **processStream() –** абстрактний метод, реалізацією якого буде обробка стріма (нашого дочірнього процеса) та вивід результату.
* **execute()** – публічний метод класу, який буде по черзі запускати чотири попередні функції:

**const** input = **await** this.**prompt**()

**const** command = this.**build**(input)

**const** stream = this.**spawn**(command)

this.**processStream**(stream, this.logger)

**FilesService –** робота з шляхами та файлами в ОС

**Методи:**

* **isExists(path: string) –** перевіряє чи існує така папка або файл в ОС
* **getFilePath(path: string, name: string, ext: string) –** створює коректну строку шляху до файлу в залежності яка ОС.
* **deleteFileIfExists(path: string) –** відповідно видаляє файл, якщо він вже існує. Ця функція використовує **isExists(path: string)**

**StreamHandler –** обробка стріма (результату запуску **child** процесу).

В конструктор приймає логер

**Методи:**

* **handleStream(stream: ChildProcessWithoutNullStreams) –** виводить данірезультату від дочірнього процесу. Якщо є помилка то виводить і її. При завершенні - виводить, що кінець виконання.

**PromptService -** власна обгортка над бібліотекою inquirer (ввід в терміналі). Було так зроблено задля більш зручної типізації.

**Методи:**

* **input(message: string, type: ‘input’ | ‘number’ | ‘password’) –** створює поле в терміналі, де можна буде вводити текст, число або навіть пароль. Символи паролю не будуть показуватись.
* **select(message: string, choices: string[]) –** створює можливість переміщатись стрілочками через меню і вибрати необхідний пункт.
* **select(message: string, choices: string[]) –** створює можливість позначити певні опції як вибрані або не вибрані.

**FfmpegBuilder** – клас що реалізує патерн Builder для збору аргументів з меню. **ConvertBuilder** та **SvgoBuilder** відповідно теж для цієї задачі, просто з іншою логікою та параметрами.

Змінні:

* **inputPath: string**
* **options: Map<string, string> = new Map()**

Методи для **FfmpegBuilder**:

* **input(inputPath: string)** – записує у змінну **inputPath** шлях до відео, який був вказаний користувачем при заповенні інпутів з меню в терміналі.
* **setVideoSize(width: number, height: number)** – в опції записує розмір та висоту картинки (в подальшому з цих опцій формуються аргументи)
* **output()** –формує остаточні аргументи

**FfmpegExecutor -** містить основну логіку виводу меню, запуску дочірнього процесу та найгловніше – вони екстендять CommandExecutor.

Методи:

Кожен executor імплментує абстрактний клас **CommandExecutor**. В кожній імплементації буде відрізняись лише логіка.