**Міністерство освіти і науки України**

**Національний університет «Львівська політехніка»**



**Пояснювальна записка до курсового проекту**

# на тему: **“Програма для автоматизації виконання команд в терміналі”**

Виконав ст.гр. КІ-39:

Гаргаш А.В.

Прийняв:

Олексів М.В.

**Львів 2023**

**Завдання на курсовий проект**

**“**Програма для автоматизації виконання команд в терміналі**”**

Програма може виконувати в терміналі різноманітні команди які доступні із меню. Кожна команда має певний набір аргументів для вибору або заповнення.

**Анотація**

Пояснювальна записка складається із 3 розділів.

У першому розділі наведені передумови необхідності, огляд існуючих рішень, функціональні вимоги, вимоги до запуску та інтеграції з сервісами сторонніх виробників.

У другому розділі наведено проектування системи, розробка архітектури програми, алгоритми її роботи.

У третьому розділі детально описано розроблені методи та класи.

Результатом є програма, в якій можна швидко виконувати команди в терміналі. А також масштабованість, для додавання любих інших команд.

**Зміст**

Вступ .................................................................................................................. 5

1. Призначення програми, її цілі та вимоги .................................................. 6
   1. Об’єкт розробки ................................................................................... 6
   2. Область застосування .......................................................................... 6
   3. Цілі розробки програми ....................................................................... 7
      1. Передумови необхідності ............................................................ 7
      2. Мета створення ............................................................................. 7
      3. Очікувані результати .................................................................... 7
   4. Аналітичний огляд існуючих рішень .................................................. 8
   5. Основні вимоги ..................................................................................... 9
      1. Загальні вимоги до запуску .......................................................... 9
      2. Функціональні вимоги .................................................................. 9
      3. Вимоги до інтеграції з сервісами сторонніх виробників .......... 10
2. Проектування програми .............................................................................. 11
   1. Опис процесу діяльності ...................................................................... 11
   2. Архітектура програмного забезпечення ............................................. 12
3. Моделювання та конструювання програмного забезпечення ................. 14
   1. Детальний опис класів програми та їм методів ................................. 14

Висновок ............................................................................................................ 17

Список використаних джерел .......................................................................... 18

Додаток А .......................................................................................................... 19

Додаток Б ........................................................................................................... 20

Додаток В ........................................................................................................... 32

Додаток С ........................................................................................................... 35

**Вступ**

Під час роботи ми часто запускаємо команди в терміналі, до того ж з різноманітними наборами аргументів. Проблема в тому, що ці аргументи забуваються і потрібно часто перечитувати документацію.

Рішенням цієї проблеми має стати програма яка запускається в терміналі. Завдяки ній можна буде вибрати команду з меню. Під кожну команду отримуємо більш зручний командний інтерфейс для введення аргументів або їх вибору.

**1. Призначення програми, її цілі та вимоги**

**1.1 Об’єкт розробки**

# Програма для автоматизації виконання команд в терміналі.

**1.2 Область застосування**

Програма призначена для розробників у яких є необхідність у частому запуску певних команд в терміналі з різним набором аргументів.

**1.3 Цілі розробки програми**

**1.3.1 Передумови необхідності**

Під час роботи часто виникають такі завдання, які повторюються. Наразі необхідна певна автоматизація. В даному випадку це запуск команд в терміналі з набором різноманітних агрументів.

**1.3.2 Мета створення**

Метою створення є перш за все зручність у виконанні рутинних команд.

**1.3.3 Очікувані результати**

Програма, в якій можна швидко виконувати команди в терміналі. А також масштабованість, для додавання любих інших команд.

**1.4 Огляд існуючих рішень**

Наразі існуючого рішення не було знайдено, адже потреба у такому функціоналі та ще й з даними командами для виклику у кожного може бути інша.

Тобто якщо у якоїсь компанії з’явиться така потреба, вона напише цей функціонал під свої потреби. Хоча, не є виключенням, що можна буде знайти базовий open-source варіант програми і вже дописати певні команди під себе.

Мій варіант проекту може існувати як базовий, на основі якого можна буде додавати команди, які вам необхідні .

**1.5 Основні вимоги**

**1.5.1 Загальні вимоги до запуску**

Необхідно, щоб програма була кросплатформенною (щоб запускалась на Windows, Linux, MacOS).

**1.5.2 Функціональні вимоги**

Нехай буде запуск програми (програма буде написана на node.js) в командному інтерфейсі: $ node index.js

Далі з'являються опції: "Яку команду ви хочете виконати"

* **ffmpeg**
* **svgo**
* **convert**

Далі в залежності від вибраної опції з’являється відповідне меню.

* Команда **ffmpeg** рендерить відео у різні розміра та необхідні розширення

При її виборі з’являється наступне меню:

* Ширина
* Висота
* Шлях до фалу
* Назва вихідного файлу
* Команда **svgo** оптимізує svg зображення

При її виборі з’являється наступне меню:

* Шлях до файлу
* Додаткові опціональні параметри
* Команда **convert** конвертує зображення у різноманітні розміра та розширення

При її виборі з’являється наступне меню:

* Ширина
* Висота
* Шлях до фалу
* Назва вихідного файлу
* Вихідний формат

**1.5.3 Вимоги до інтеграції з сервісами сторонніх виробників**

Необхідна чітка архітектура з можливістю майбутнього масштабування або швидкої можливості інтеграції з іншими сервісами, наприклад Telegram (чат-бот).

**2. Проектування програми**

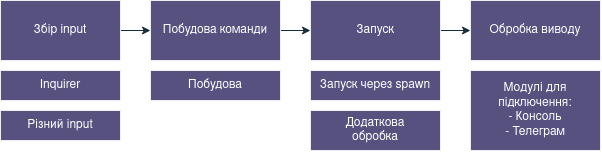
**2.1 Опис процесу діяльності**

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).

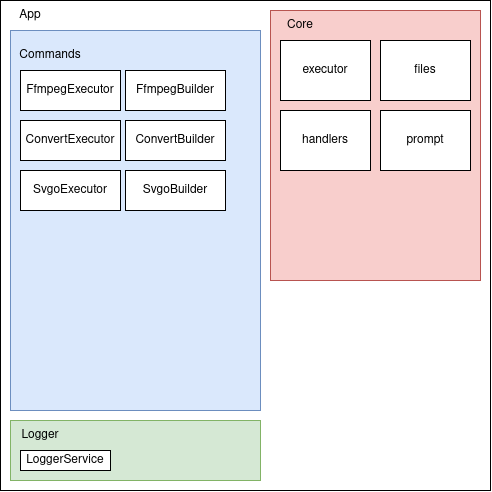
Після запуску з’являється меню, вибираєте необхідну команду для запуску в терміналі. Після вибору команди з’являється відповідне меню, де ви заповнюєте поля.

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

**2.2 Архітектура програмного забезпечення**



*Рис.1 Template method pattern процеси*



*Рис.2 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. |

**3. Моделювання та конструювання програмного забезпечення**

**3.1 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.

**Висновок:**

Програма для автоматизації виконання команд в терміналі, розроблена на Node.js, допоможе у полегшенні роботи з терміналом та прискоренні виконання команд. Цей проект вирішує проблему виконання складних команд, що вимагають багато різноманітних аргументів.

У результаті ретельного проектування та розробки, була створена програма з добре обміркованою архітектурою.

Вона дозволяє користувачам швидко та легко виконувати потрібні команди в терміналі, забезпечуючи зручність та ефективність у роботі.

Проект також враховує масштабованість, надаючи можливість додавання нових команд і розширення функціоналу з легкістю. Це робить його гнучким та адаптивним до змінних потреб користувачів.

Завдяки ретельно проробленій роботі над проектом, отримано досить цікавий інструмент, який полегшує роботу з терміналом.

**Список використаних джерел**

* <https://nodejs.org/uk>
* [https://stackoverflow.com/](./·%20https://stackoverflow.com/)

**Додаток А (Stories)**

1. **User story:** [As a user, i want to see all log details](https://github.com/users/TolyaDeveloper/projects/3?pane=issue&itemId=25106860)
   1. Task: [Create logger service](https://github.com/users/TolyaDeveloper/projects/3?pane=issue&itemId=25107283)
2. **User story:** [As a user, i want to select and execute a command with options](https://github.com/users/TolyaDeveloper/projects/3?pane=issue&itemId=25174089)
   1. Task: [Handle input stream](https://github.com/users/TolyaDeveloper/projects/3?pane=issue&itemId=25174124)
   2. Task: [Handle output stream](https://github.com/users/TolyaDeveloper/projects/3?pane=issue&itemId=25175079)
   3. Task: [Create Template pattern for executor](https://github.com/users/TolyaDeveloper/projects/3?pane=issue&itemId=25175644)
   4. Task: [Create one executor command. Use Builder pattern for that command](https://github.com/users/TolyaDeveloper/projects/3?pane=issue&itemId=25175691)
   5. Task: [Add command select in app](https://github.com/users/TolyaDeveloper/projects/3?pane=issue&itemId=25209710)
   6. Task: [Add possibility to choose video extension in ffmpeg executor](https://github.com/users/TolyaDeveloper/projects/3?pane=issue&itemId=26108158)
   7. Task: [Add other commands](https://github.com/users/TolyaDeveloper/projects/3?pane=issue&itemId=26420168)

[Github task board](https://github.com/users/TolyaDeveloper/projects/3)

**Додаток Б (Сорс-код продукту)**

*app.ts*

**import** { **FfmpegExecutor** } **from** './commands/ffmpeg/ffmpeg.executor'

**import** { **SvgoExecutor** } **from** './commands/svgo/svgo.executor'

**import** { **ConvertExecutor** } **from** './commands/convert/convert.executor'

**import** { **LoggerService** } **from** './logger/logger.service'

**import** { **PromptService** } **from** './core/prompt/prompt.service'

enum ECommands {

ffmpeg = 'ffmpeg',

svgo = 'svgo',

convert = 'convert'

}

**export** **class** **App** {

**public** **async** **run**() {

**const** answer = **await** **new** **PromptService**().select<keyof **typeof** ECommands>(

'What command do you want to execute?',

[ECommands.ffmpeg, ECommands.svgo, ECommands.convert]

)

**switch** (answer) {

**case** 'ffmpeg':

**return** **new** **FfmpegExecutor**(**LoggerService**.**getInstance**()).**execute**()

**case** 'svgo':

**return** **new** **SvgoExecutor**(**LoggerService**.**getInstance**()).**execute**()

**case** 'convert':

**return** **new** **ConvertExecutor**(**LoggerService**.**getInstance**()).**execute**()

default:

**return** **LoggerService**.**getInstance**().**error**('Such command does not exist!')

}

}

}

**new** **App**().**run**()

*logger.service.ts*

**import** { ILoggerService } **from** './logger.service.interface'

**export** **class** **LoggerService** **implements** ILoggerService {

**private** **static** instance: **LoggerService**

**private** **constructor**() {}

**public** **static** **getInstance**() {

**if** (!**LoggerService**.instance) {

**LoggerService**.instance = **new** **LoggerService**()

}

**return** **LoggerService**.instance

}

**public** **log**(...args: any[]): void {

console.**log**(args)

}

**public** **warn**(...args: any[]): void {

console.**log**(args)

}

**public** **error**(...args: any[]): void {

console.**log**(args)

}

**public** **end**(): void {

console.**log**('Done')

}

}

*logger.service.interface.ts*

**export** **interface** ILoggerService {

**log**(...args: any[]): void

**warn**(...args: any[]): void

**error**(...args: any[]): void

**end**(): void

}

*command.executor.ts*

**import** { **ChildProcessWithoutNullStreams** } **from** 'child\_process'

**import** { ILoggerService } **from** '../../logger/logger.service.interface'

**import** { ICommandExecutor } **from** './command.executor.interface'

**export** **abstract** **class** **CommandExecutor**<**Input**> {

**constructor**(**private** logger: ILoggerService) {}

**public** **async** **execute**() {

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

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

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

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

}

**protected** **abstract** **prompt**(): **Promise**<**Input**>

**protected** **abstract** **build**(input: **Input**): ICommandExecutor

**protected** **abstract** **spawn**(

command: ICommandExecutor

): **ChildProcessWithoutNullStreams**

**protected** **abstract** **processStream**(

stream: **ChildProcessWithoutNullStreams**,

logger: ILoggerService

): void

}

*command.executor.interface.ts*

**export** **interface** ICommandExecutor {

command: string

args: string[]

}

*files.service.ts*

**import** { dirname, isAbsolute, join } **from** 'path'

**import** { stat, unlink } **from** 'fs/promises'

**export** **class** **FilesService** {

**private** **async** **isExists**(path: string) {

**try** {

**await** **stat**(path)

**return** true

} **catch** {

**return** false

}

}

**public** **getFilePath**(path: string, name: string, ext: string): string {

**if** (!**isAbsolute**(path)) {

path = **join**(\_\_dirname, path)

}

**return** **join**(**dirname**(path), name + '.' + ext)

}

**public** **async** **deleteFileIfExists**(path: string) {

**if** (**await** this.**isExists**(path)) {

**await** **unlink**(path)

}

}

}

*stream.handler.ts*

**import** { **ChildProcessWithoutNullStreams** } **from** 'child\_process'

**import** { ILoggerService } **from** '../../logger/logger.service.interface'

**export** **class** **StreamHandler** {

**constructor**(**private** logger: ILoggerService) {}

**handleStream**(stream: ChildProcessWithoutNullStreams) {

stream.stdout.**on**('data', (data: any) => {

this.logger.**log**(data.**toString**())

})

stream.stderr.**on**('data', (data: any) => {

this.logger.**error**(data.**toString**())

})

stream.**on**('close', () => {

this.logger.**end**()

})

}

}

*prompt.service.ts*

**import** inquirer **from** 'inquirer'

**import** { **InputType** } **from** './prompt.service.interface'

**export** **class** **PromptService** {

**public** **async** input<T>(message: string, type: **InputType** = 'input') {

**const** { result } = **await** inquirer.prompt<{ result: T }>([

{ **type**, name: 'result', message }

])

**return** result

}

**public** **async** select<T>(message: string, choices: string[]) {

**const** { result } = **await** inquirer.prompt<{ result: T }>([

{ type: 'list', name: 'result', message, choices }

])

**return** result

}

**public** **async** checkbox<T>(message: string, choices: string[]) {

**const** { result } = **await** inquirer.prompt<{ result: T }>([

{ type: 'checkbox', name: 'result', message, choices }

])

**return** result

}

}

*prompt.service.interface.ts*

**export** **type** **InputType** = 'input' | 'number' | 'password'

*convert.builder.ts*

**export** **class** **ConvertBuilder** {

**private** inputPath: string

**private** options: string[] = []

**constructor**() {}

**public** **input**(inputPath: string): this {

this.inputPath = inputPath

**return** this

}

**setImageSize**(width: number, height: number): this {

this.options.**push**(`-size ${width}x${height}`)

**return** this

}

**public** **output**(outputPath: string): string[] {

**if** (!this.inputPath) {

**throw** **new** **Error**('Input path is not set!')

}

**const** args: string[] = [...this.options, this.inputPath, outputPath]

**return** args

}

}

*convert.executor.ts*

**import** { **ChildProcessWithoutNullStreams**, spawn } **from** 'child\_process'

**import** { **CommandExecutor** } **from** '../../core/executor/command.executor'

**import** { ICommandConvert, IConvertInput } **from** './convert.interface'

**import** { ILoggerService } **from** '../../logger/logger.service.interface'

**import** { **PromptService** } **from** '../../core/prompt/prompt.service'

**import** { **ConvertBuilder** } **from** './convert.builder'

**import** { **StreamHandler** } **from** '../../core/handlers/stream.handler'

**import** { **FilesService** } **from** '../../core/files/files.service'

**export** **class** **ConvertExecutor** **extends** **CommandExecutor**<IConvertInput> {

**private** fileService: **FilesService** = **new** **FilesService**()

**private** promptService: **PromptService** = **new** **PromptService**()

**constructor**(logger: ILoggerService) {

super(logger)

}

**protected** **async** **prompt**(): **Promise**<IConvertInput> {

**const** path = **await** this.promptService.input<string>('Path', 'input')

**const** width = **await** this.promptService.input<number>('Width', 'number')

**const** height = **await** this.promptService.input<number>('Height', 'number')

**const** outputName = **await** this.promptService.input<string>('Output name')

**const** outputFormat = **await** this.promptService.input<string>('Output format')

**return** { path, width, height, outputName, outputFormat }

}

**protected** **build**({

height,

outputFormat,

outputName,

path,

width

}: IConvertInput): ICommandConvert {

**const** output = this.fileService.**getFilePath**(path, outputName, outputFormat)

**const** args = **new** **ConvertBuilder**()

.**setImageSize**(width, height)

.**input**(path)

.**output**(output)

**return** { command: 'convert', args, output }

}

**protected** **spawn**({

args,

command,

output

}: ICommandConvert): **ChildProcessWithoutNullStreams** {

this.fileService.**deleteFileIfExists**(output)

**return** **spawn**(command, args, { shell: true })

}

**protected** **processStream**(

stream: **ChildProcessWithoutNullStreams**,

logger: ILoggerService

): void {

**const** handler = **new** **StreamHandler**(logger)

handler.**handleStream**(stream)

}

}

*convert.interface.ts*

**import** { ICommandExecutor } **from** '../../core/executor/command.executor.interface'

**export** **interface** IConvertInput {

path: string

width: number

height: number

outputName: string

outputFormat: string

}

**export** **interface** ICommandConvert **extends** ICommandExecutor {

output: string

}

*ffmpeg.builder.ts*

**export** **class** **FfmpegBuilder** {

**private** inputPath: string

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

**constructor**() {

this.options.**set**('-c:v', 'libx264')

}

**public** **input**(inputPath: string): this {

this.inputPath = inputPath

**return** this

}

**setVideoSize**(width: number, height: number): this {

this.options.**set**('-s', `${width}x${height}`)

**return** this

}

**public** **output**(outputPath: string): string[] {

**if** (!this.inputPath) {

**throw** **new** **Error**('Input path is not set!')

}

**const** args: string[] = ['-i', this.inputPath]

this.options.**forEach**((value, key) => {

args.**push**(key)

args.**push**(value)

})

args.**push**(outputPath)

**return** args

}

}

*ffmpeg.executor.ts*

**import** { **ChildProcessWithoutNullStreams**, spawn } **from** 'child\_process'

**import** { **CommandExecutor** } **from** '../../core/executor/command.executor'

**import** { ICommandFfmpeg, IFfmpegInput } **from** './ffmpeg.interface'

**import** { ILoggerService } **from** '../../logger/logger.service.interface'

**import** { **FilesService** } **from** '../../core/files/files.service'

**import** { **PromptService** } **from** '../../core/prompt/prompt.service'

**import** { **FfmpegBuilder** } **from** './ffmpeg.builder'

**import** { **StreamHandler** } **from** '../../core/handlers/stream.handler'

**export** **class** **FfmpegExecutor** **extends** **CommandExecutor**<IFfmpegInput> {

**private** fileService: **FilesService** = **new** **FilesService**()

**private** promptService: **PromptService** = **new** **PromptService**()

**constructor**(logger: ILoggerService) {

super(logger)

}

**protected** **async** **prompt**(): **Promise**<IFfmpegInput> {

**const** width = **await** this.promptService.input<number>('Width', 'number')

**const** height = **await** this.promptService.input<number>('Height', 'number')

**const** path = **await** this.promptService.input<string>('Path to file')

**const** outputName = **await** this.promptService.input<string>('Output name')

**const** outputFormat = **await** this.promptService.input<string>('Output format')

**return** { width, height, path, outputName, outputFormat }

}

**protected** **build**({

width,

height,

path,

outputName,

outputFormat

}: IFfmpegInput): ICommandFfmpeg {

**const** output = this.fileService.**getFilePath**(path, outputName, outputFormat)

**const** args = **new** **FfmpegBuilder**()

.**input**(path)

.**setVideoSize**(width, height)

.**output**(output)

**return** { command: 'ffmpeg', args, output }

}

**protected** **spawn**({

args,

command,

output

}: ICommandFfmpeg): **ChildProcessWithoutNullStreams** {

this.fileService.**deleteFileIfExists**(output)

**return** **spawn**(command, args)

}

**protected** **processStream**(

stream: **ChildProcessWithoutNullStreams**,

logger: ILoggerService

): void {

**const** handler = **new** **StreamHandler**(logger)

handler.**handleStream**(stream)

}

}

*ffmpeg.interface.ts*

**import** { ICommandExecutor } **from** '../../core/executor/command.executor.interface'

**export** **interface** IFfmpegInput {

width: number

height: number

path: string

outputName: string

outputFormat: string

}

**export** **interface** ICommandFfmpeg **extends** ICommandExecutor {

output: string

}

*svgo.builder.ts*

**export** **class** **SvgoBuilder** {

**private** inputPath: string

**private** options: string[] = []

**public** **input**(inputPath: string): this {

this.inputPath = inputPath

**return** this

}

**public** setArguments = (args: string[]) => {

args.**forEach**(arg => {

this.options.**push**(`--${arg}`)

})

**return** this

}

**public** **output**(): string[] {

**if** (!this.inputPath) {

**throw** **new** **Error**('Input path is not set!')

}

**const** args: string[] = [...this.options, this.inputPath]

**return** args

}

}

*svgo.executor.ts*

**import** { **ChildProcessWithoutNullStreams**, spawn } **from** 'child\_process'

**import** { **CommandExecutor** } **from** '../../core/executor/command.executor'

**import** { ICommandSvgo, ISvgoInput } **from** './svgo.interface'

**import** { ILoggerService } **from** '../../logger/logger.service.interface'

**import** { **PromptService** } **from** '../../core/prompt/prompt.service'

**import** { **SvgoBuilder** } **from** './svgo.builder'

**import** { **StreamHandler** } **from** '../../core/handlers/stream.handler'

**export** **class** **SvgoExecutor** **extends** **CommandExecutor**<ISvgoInput> {

**private** promptService: **PromptService** = **new** **PromptService**()

**constructor**(logger: ILoggerService) {

super(logger)

}

**protected** **async** **prompt**(): **Promise**<ISvgoInput> {

**const** path = **await** this.promptService.input<string>('Path to svg', 'input')

**const** additional = **await** this.promptService.checkbox<string[]>(

'Additional parameters',

['multipass']

)

**return** { additional, path }

}

**protected** **build**({ path, additional }: ISvgoInput): ICommandSvgo {

**const** args = **new** **SvgoBuilder**().**input**(path).**setArguments**(additional).**output**()

**return** { command: 'npx svgo', args }

}

**protected** **spawn**({

args,

command

}: ICommandSvgo): **ChildProcessWithoutNullStreams** {

**return** **spawn**(command, args, { shell: true })

}

**protected** **processStream**(

stream: **ChildProcessWithoutNullStreams**,

logger: ILoggerService

): void {

**const** handler = **new** **StreamHandler**(logger)

handler.**handleStream**(stream)

}

}

*svgo.interface.ts*

**import** { ICommandExecutor } **from** '../../core/executor/command.executor.interface'

**export** **interface** ISvgoInput {

path: string

additional: string[]

}

**export** **type** ICommandSvgo = ICommandExecutor

**Додаток В (конфігураційні файли проекту)**

*.eslintrc.js*

module.exports = {

parser: '@typescript-eslint/parser',

parserOptions: {

project: 'tsconfig.json',

tsconfigRootDir: \_\_dirname

},

plugins: ['@typescript-eslint/eslint-plugin'],

extends: [

'plugin:@typescript-eslint/recommended',

'plugin:prettier/recommended'

],

root: true,

ignorePatterns: ['.eslintrc.js'],

rules: {

'@typescript-eslint/interface-name-prefix': 'off',

'@typescript-eslint/explicit-function-return-type': 'off',

'@typescript-eslint/explicit-module-boundary-types': 'off',

'@typescript-eslint/no-explicit-any': 'off',

'@typescript-eslint/no-empty-function': 'off'

}

}

*.gitignore*

/build

/node\_modules

/dist

tsconfig.tsbuildinfo

\*.mp4

\*.mov

\*.svg

\*.jpg

\*.png

*.prettierrc*

{

"arrowParens": "avoid",

"trailingComma": "none",

"endOfLine": "auto",

"bracketSpacing": true,

"printWidth": 80,

"tabWidth": 2,

"semi": false,

"singleQuote": true

}

*package.json*

{

"name": "coursework",

"version": "1.0.0",

"description": "A program for automating the execution of commands in the terminal",

"main": "index.js",

"scripts": {

"start": "node dist/app.js",

"build": "tsc",

"format:fix": "prettier --write \"./\*\*/\*.{js,jsx,ts,tsx,css,md,json}\"",

"lint:fix": "eslint ./src --ext .jsx,.js,.ts,.tsx --quiet --fix --ignore-path ./.gitignore",

"lint": "npm run format:fix &amp;&amp; npm run lint:fix"

},

"dependencies": {

"inquirer": "^8.2.5"

},

"devDependencies": {

"@types/inquirer": "^9.0.3",

"@types/node": "^18.15.11",

"@typescript-eslint/eslint-plugin": "^5.57.0",

"@typescript-eslint/parser": "^5.57.0",

"eslint": "^8.37.0",

"eslint-config-prettier": "^8.8.0",

"eslint-plugin-prettier": "^4.2.1",

"pre-commit": "^1.2.2",

"prettier": "2.8.7",

"typescript": "^5.0.3"

},

"repository": {

"type": "git",

"url": "git+https://github.com/TolyaDeveloper/Harhash-KI-39-coursework.git"

},

"keywords": [

"shell",

"automation"

],

"author": "tolyaDeveloper",

"license": "ISC",

"bugs": {

"url": "https://github.com/TolyaDeveloper/Harhash-KI-39-coursework/issues"

},

"homepage": "https://github.com/TolyaDeveloper/Harhash-KI-39-coursework#readme",

"pre-commit": "lint"

}

*tsconfig.json*

{

"compilerOptions": {

*/\* Visit https://aka.ms/tsconfig to read more about this file \*/*

"incremental": true,

"target": "es2016",

"module": "commonjs",

"rootDir": "./src",

"outDir": "./dist",

"esModuleInterop": true,

"forceConsistentCasingInFileNames": true,

"strict": true,

"strictPropertyInitialization": false

}

}

*tsconfig.build.json*

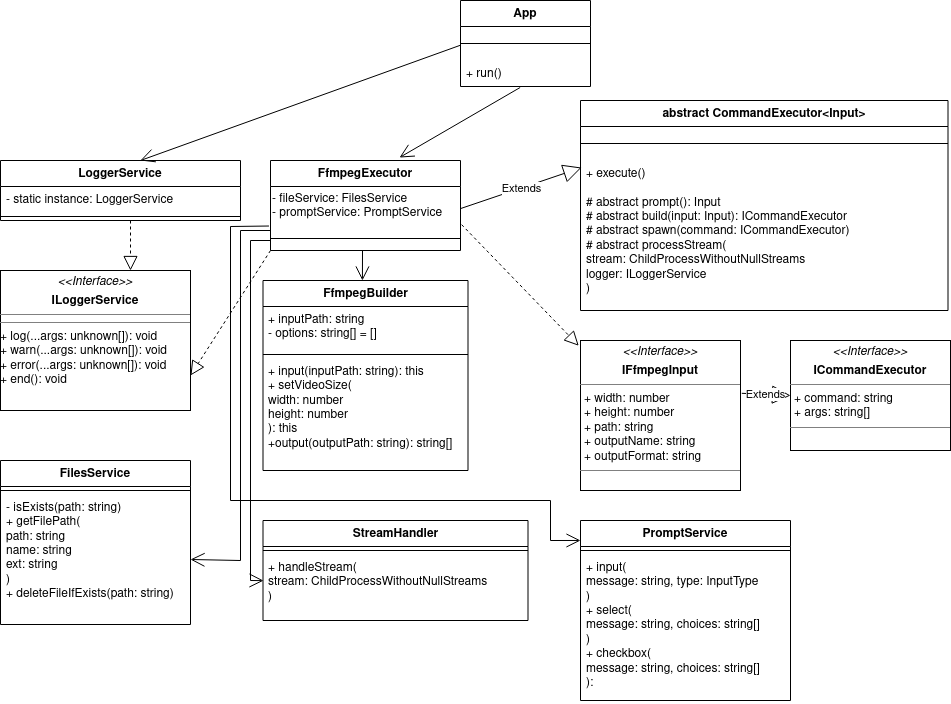
{

"extends": "./tsconfig.json",

"exclude": ["node\_modules", "test", "dist", "build", "\*\*/\*spec.ts"]

}

**Додаток С**



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Міністерство освіти і науки України | | | | | **КУРСОВИЙ ПРОЕКТ** | | | | |
|  |  |  |  |  | **Програма для автоматизації виконання команд в терміналі** | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  | **Модуль відображення даних**  Діаграма класів | Літера | Маса | | Масштаб |
| Зм | Арк. | No докум. | Підпис | Дата |  |  | |  |
| Виконав | | Гаргаш |  |  |
| Керівник | |  |  |  |
| Консульт. | |  |  |  | Аркуш 1 | | Аркушів 1 | |
| Консульт | |  |  |  |  | НУ ”ЛП”, ІКТА,  каф. СКС, гр. **КІ-39** | | | |
| Зав. каф. | |  |  |  |
| Реценз. | |  |  |  |