ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА
(підпис)

«_____»___202_ р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТЕЛЕГРАМ- БОТА ДЛЯ ЗАВАНТАЖЕННЯ ТА ОБРОБКИ ВІДЕО З YOUTUBE

зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня бакалавр

Виконавець роботи Ганзя Іван Олександрович
_____«____» _____202_ р.
(підпис)

Науковий керівник доцент, к.ф.-м.н. Парфьонова Т. О.
_____«____»_____202_ р.
(підпис)

Рецензент

ПОЛТАВА 2025

РЕФЕРАТ

Записка: 54 с., 6 рис., 1 таблиця, 1 додаток, 10 джерел.

TELEGRAM-БОТ, PYTHON, YT-DLP, AIROGRAM, ЗАВАНТАЖЕННЯ ВІДЕО, ОБРОБКА АУДІО

Об'єктом розробки є програмне забезпечення у вигляді Telegram-бота для завантаження відео та аудіо з популярних медіаплатформ. Метою роботи є створення універсального Telegram-бота, який дозволяє завантажувати відео та аудіо з таких платформ, як YouTube, TikTok, Instagram, Facebook, а також реалізує функції пошуку музики за назвою, завантаження плейлистів і обрізання аудіотреків.

У процесі розробки було реалізовано такі можливості:

- завантаження відео з підтримкою форматів до 720p/1080p;
- збереження та надсилання аудіофайлів у форматі .mp3;
- автоматичний пошук музики на YouTube за запитом користувача;
- підтримка посилань на повноцінні плейлисти;
- обрізання аудіофайлів за заданими тайм-кодами без втрати якості;
- інтеграція з Telegram API через бібліотеку aiogram;
- використання yt-dlp для стійкого та якісного парсингу відеопотоків з підтримкою десятків платформ.

Архітектура проєкту модульна: окремі компоненти відповідають за завантаження відео, обробку аудіо, пошук музики, парсинг плейлистів, обробку команд користувача тощо. Для реалізації логіки використано мову програмування Python 3.10.

Результатом роботи стало стабільне програмне рішення з інтуїтивним інтерфейсом, яке дозволяє звичайному користувачеві Telegram без технічних знань швидко отримати медіаконтент у зручному форматі.

Практична цінність роботи полягає у створенні інструменту, що може бути

використаний у повсякденному житті для швидкого доступу до відео- та аудіоконтенту, а також у медіаосвіті - як частина сервісу для створення та обробки навчальних матеріалів.

Робота підтверджена тестуванням, має високу стабільність виконання, розширювану структуру та повністю задовольняє вимоги до сучасного Telegram-бота з медіафункціоналом.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ, ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, скорочень, символів
aiogram	Python-фреймворк для створення Telegram-ботів
API	Інтерфейс прикладного програмування (Application Programming Interface)
Command	Телеграм-команда, яку вводить користувач (наприклад, /video)
Dispatcher	Компонент aiogram, що обробляє події Telegram
FFmpeg	Програма для обробки відео- та аудіофайлів
MP3	Формат стисненого аудіо файлу
MP4	Формат відеофайлу з підтримкою аудіо і відео
Playlist	Список відео або аудіо, що відтворюються послідовно
Python	Мова програмування високого рівня
Telegram Bot	Програмний агент, що працює в екосистемі Telegram і реагує на команди
TikTok	Соціальна платформа коротких відео
YouTube	Відеохостингова платформа, з якої здійснюється завантаження контенту

yt-dlp	Інструмент для завантаження відео та аудіо з онлайн-платформ
URL	Уніфікований локатор ресурсу (посилання на сайт чи відео)
.env	Файл середовищних змінних для конфігурації проєкту

ВСТУП

У сучасну цифрову епоху користувачі все активніше споживають мультимедійний контент, зокрема відео з таких платформ, як YouTube. Зростання обсягів відеоінформації породжує потребу у зручних інструментах для її обробки, збереження та подальшого використання. Особливо актуальним є створення програмних засобів, які дозволяють автоматизувати завантаження відео та конвертацію їх у різні формати для подальшої роботи.

Одним із популярних рішень у сфері автоматизації є застосування чат-ботів, які дозволяють здійснювати взаємодію з користувачами у зручному для них середовищі - месенджерах. Telegram є однією з таких платформ, що забезпечує гнучкий API для розробки ботів, які можуть виконувати складні обчислювальні та мережеві операції.

Створення Telegram-бота, що дозволяє завантажувати відео або аудіо з YouTube, обробляти його, зберігати локально або пересилати користувачу, є прикладом практичного застосування сучасних технологій Python-програмування, мережевих запитів, роботи з медіафайлами та API зовнішніх сервісів.

Актуальність теми обумовлена необхідністю автоматизованих засобів для роботи з відеоконтентом у межах навчального процесу, розваг, створення контенту та інших цифрових задач.

Мета роботи - розробити функціонального Telegram-бота для завантаження, конвертації та передачі відео- та аудіофайлів з YouTube і підтримки інших популярних платформ.

Завдання роботи:

- Ознайомитися з принципами роботи Telegram-ботів та API Telegram.
- Дослідити можливості бібліотеки yt-dlp для роботи з відео.
- Реалізувати підтримку завантаження відео та аудіо з YouTube.
- Додати функції пошуку музики, обрізання треків, підтримки TikTok, Instagram, Facebook.
- Забезпечити підтримку завантаження цілих плейлистів.

- Налаштувати Telegram-бота для стабільної роботи на сервері.
- Перевірити стабільність роботи бота та його коректну взаємодію з користувачами.

Об’єкт дослідження - процес обробки відео- та аудіоконтенту із відкритих онлайн-ресурсів за допомогою програмного забезпечення.

Предмет дослідження - створення Telegram-бота з використанням мови Python, бібліотек aiogram, yt-dlp, ffmpeg та інших інструментів для роботи з відео й аудіо.

Практичне значення роботи полягає у створенні програмного продукту, який дозволяє будь-якому користувачу без спеціальних навичок легко завантажувати та обробляти відео з YouTube через звичний інтерфейс месенджера Telegram. Цей бот може бути застосований як у повсякденному використанні, так і в освітньому чи комерційному середовищі.

Методи дослідження, застосовані у роботі, включають аналіз наукових джерел, порівняння сучасних бібліотек і фреймворків для Python, а також експериментальне програмування з подальшим тестуванням функціональності бота в реальному середовищі. Особливу увагу приділено дослідженню архітектурних рішень для побудови Telegram-ботів та інтеграції з зовнішніми сервісами, зокрема YouTube. У процесі реалізації було використано низку популярних інструментів: aiogram — для роботи з Telegram API, yt-dlp — для завантаження відео й аудіо з онлайн-ресурсів, ffmpeg — для обробки мультимедійного контенту, python-dotenv — для керування конфігураційними змінними.

Дипломна робота має чітко структурований вигляд: спочатку розкривається загальна характеристика проблеми та описується актуальність теми, далі подано огляд сучасних технологій і підходів до реалізації ботів, розглянуто технічні аспекти роботи з відео- та аудіофайлами, описано процес створення архітектури проєкту та реалізації ключових функціональних модулів. У завершальній частині роботи наведено висновки щодо результатів дослідження, а також можливості подальшого розвитку програмного продукту.

Загальний обсяг дипломної роботи становить 54 сторінки. Робота містить 6 ілюстрацій, 1 таблицю, список із 10 використаних джерел, а також один додаток з прикладами коду та інструкцією користувача. Структура роботи дозволяє всебічно охопити як теоретичні, так і практичні аспекти створення Telegram-бота для завантаження та обробки відеоконтенту, демонструючи актуальність і ефективність розробленого рішення.

1.ПОСТАНОВКА ЗАДАЧІ

Основною метою роботи є створення зручного телеграм-бота для завантаження та обробки відео- і аудіоконтенту з платформ YouTube, TikTok, Instagram і Facebook. Бот має забезпечити користувачам можливість швидко отримувати відеофайли, витягати з них аудіо, обрізати аудіофрагменти та завантажувати плейлисти безпосередньо в месенджері Telegram. Такий інструмент дозволяє автоматизувати рутинні дії, пов’язані з обробкою контенту, і робить їх доступними без використання сторонніх сайтів чи програм.

Основні завдання:

1. Реалізація підтримки основних платформ:
    - Автоматичне розпізнавання посилань з YouTube, TikTok, Instagram, Facebook.
    - Підтримка коротких та повних форматів URL.
2. Завантаження відео та аудіо:
    - Завантаження відео з YouTube та інших платформ у форматі .mp4.
    - Витягування аудіо з відео та конвертація у формат .mp3.
    - Формування імені файлу на основі назви відео.
3. Пошук музики по назві:
    - Реалізація пошуку музики через YouTube з автоматичним завантаженням першого релевантного результату.
4. Обрізка аудіо:
    - Додавання функції обрізання аудіо за вказаним діапазоном часу (наприклад, 00:30-01:45).
    - Збереження обрізаного фрагменту як окремий файл з відповідною назвою.
5. Підтримка плейлистів:
    - Завантаження всіх відео з YouTube-плейлиста в окрему директорію.
    - Формування структури папок з назвами плейлистів і нумерацією треків.

6. Інтеграція з Telegram Bot API:
    - Реалізація команд /start, /video, /audio, /music, /cut_audio, /playlist.
    - Обробка повідомлень користувача та автоматичне надсилання результату.
7. Розробка структури проєкту:
    - Модульний поділ на логіку обробки відео, аудіо, пошуку та API Telegram.
    - Збереження конфігурації у файлі .env.
8. Реалізація логування та обробки помилок:
    - Запис помилок при завантаженні або відправці файлів у лог-файл.
    - Надсилання повідомлень користувачу у разі збоїв з поясненням причин.
9. Оптимізація тимчасових файлів:
    - Створення тимчасової директорії для збереження результатів.
    - Автоматичне видалення файлів після відправлення користувачу.
10. Оформлення супровідної документації:
    - Опис команд для користувачів.
    - Інструкція з розгортання бота локально або на хостингу.

Вимоги до програмного забезпечення:

- Зручність використання – бот має працювати у кілька кліків і не вимагати складних дій від користувача.
- Стабільність – обробка відео та аудіо має виконуватися надійно навіть при великих розмірах файлів.
- Швидкодія – мінімальні затримки під час завантаження та відправлення результатів.
- Масштабованість – можливість розширення функціоналу у майбутньому (наприклад, підтримка субтитрів або GIF).

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Загальний огляд ботів у месенджерах

Чат-боти - це програми, які імітують реальну бесіду з людьми за допомогою штучного інтелекту та обробки природної мови. Вони використовуються у багатьох сферах, включаючи обслуговування клієнтів, розваги, освіту та автоматизацію рутинних завдань. Боти можуть бути інтегровані з різними месенджерами, такими як Telegram, WhatsApp, Facebook Messenger та інші платформи, що значно розширює їх доступність і функціонал.

Telegram, створений Павлом Дуровим, є одним з найпопулярніших месенджерів для використання ботів, завдяки своїй високій швидкості, сильному шифруванню та зручному API для розробників. Telegram-боти можуть виконувати широкий спектр дій, таких як відправка повідомлень, зображень, аудіофайлів, відео та документів, а також реагування на вхідні команди користувачів. [1]

Боти у месенджерах набули великої популярності завдяки можливості швидкого доступу до необхідної інформації без необхідності встановлення додаткових додатків. Вони надають можливість здійснювати покупки, бронювати квитки, вчитися, отримувати новини та багато іншого, безпосередньо з інтерфейсу чату. Такий підхід економить час користувачів та поліпшує їхній досвід використання сервісів.

Зараз на ринку існує безліч платформ та інструментів для створення чат-ботів, кожен з яких має свої унікальні особливості. Проте, незмінним залишається основний принцип роботи бота - забезпечення швидкого та зручного способу взаємодії з сервісами та інформацією через текстові команди або інтерфейс на основі кнопок.

В контексті даної роботи особлива увага приділяється розробці ботів, які використовують API відеоплатформ, таких як YouTube, для завантаження та обробки медіаконтенту. Такі боти вимагають розуміння як принципів роботи відеоплатформ, так і месенджерів, де вони використовуються. Завдяки поєднанню

функціоналу YouTube та Telegram боти можуть надавати користувачам зручний інструмент для перегляду, пошуку та завантаження відеоконтенту.

2.2. Особливості ботів у Telegram

Telegram боти - це високофункціональні віртуальні асистенти, які вбудовані безпосередньо у саму структуру месенджера, забезпечуючи користувачам широкий спектр можливостей. Вони здатні автоматизувати завдання, виконувати запити в реальному часі та надавати інтерактивний інтерфейс для спілкування з системою чи сервісами. Ці боти стали невід'ємною частиною досвіду використання Telegram через свою багатофункціональність та зручність використання.

Розробка ботів у Telegram здійснюється за допомогою Telegram Bot API — могутнього інструменту, який дає розробникам волю до створення найрізноманітніших ботів. Від простих ботів, що відправляють щоденні погодні прогнози або новини, до складних систем, котрі здатні інтегруватися з зовнішніми базами даних, API та іншими сервісами, Telegram боти пропонують безмежні можливості для взаємодії.

Особливою властивістю ботів у Telegram є їх здатність впроваджувати індивідуальний підхід до кожного користувача. За допомогою штучного інтелекту та машинного навчання боти можуть аналізувати запити користувачів, запам'ятовувати їхні переваги та надавати персоналізований контент. Також боти можуть використовувати інтерактивні кнопки, які дозволяють користувачам швидко обирати з певних опцій, що значно спрощує процес навігації та вибору. [2]

Ще одна важлива риса Telegram ботів полягає в їх здатності працювати у групах та каналах. Вони можуть виступати в ролі модераторів, відстежувати повідомлення за певними ключовими словами, відповідати на запитання або навіть проводити опитування. Ця функція робить ботів вкрай корисними для спільнот та бізнесу, оскільки вони допомагають автоматизувати взаємодію та забезпечують збір цінної інформації від користувачів.

Робота має на меті детально розглянути всі ці аспекти. Вона охоплює процес

розробки від ідеї до реалізації бота, який не просто виконує задані функції, а є справжнім помічником користувачам, який може відкрити нові можливості для взаємодії з контентом YouTube в рамках звичного месенджера. [3]

2.3. Приклади використання ботів

Боти у Telegram пропонують незліченні способи використання, від особистих помічників до інструментів для бізнесу. Наприклад, особисті боти можуть допомагати користувачам управляти їхніми щоденними завданнями, нагадувати про важливі події чи навіть відстежувати особисті витрати. З іншого боку, боти для бізнесу можуть оптимізувати процеси обслуговування клієнтів, автоматизувати відповіді на часті запитання, обробляти замовлення чи навіть управляти проектами.

Один з прикладів – боти для бронювання. Вони можуть використовуватися в ресторанах, готелях та інших сервісах для зручного бронювання місць або замовлення послуг. Користувач просто вводить запит, а бот автоматично обробляє його та підтверджує бронь, значно скорочуючи час і зусилля, необхідні для виконання цієї задачі.

Інший приклад – навчальні боти. Вони можуть бути запрограмовані для надання освітніх матеріалів, проведення тестів та вікторин, а також для перевірки знань у формі інтерактивного діалогу. Це забезпечує студентам більш гнучкі та доступні способи навчання.

Для інформаційного контенту боти можуть слугувати каналами доставки новин, погодних прогнозів, фінансових звітів тощо. Користувачі можуть підписатися на оновлення за своїми інтересами та отримувати актуальну інформацію безпосередньо у свій чат.

Також важливе місце посідають боти для соціальних опитувань та голосувань, що дозволяють збирати думки та переваги користувачів з різних питань. Вони забезпечують простий спосіб ангажувати велику аудиторію та швидко збирати відгуки. [4]

Боти, які забезпечують завантаження та обробку медіаконтенту, подібні до

нашого проекту, відкривають нові можливості для взаємодії з відеоконтентом. Користувачі можуть використовувати ці боти для швидкого доступу до відео з YouTube, завантаження їх для офлайн перегляду, а також для конвертації відео у різні формати або вилучення аудіо з них. Це робить доступ до медіа більш гнучким і зручним, адаптуючи його до індивідуальних потреб користувача.

Таким чином, використання ботів у Telegram відкриває широкий спектр можливостей, які можуть бути індивідуально адаптовані до потреб кожного користувача чи організації.

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Огляд мови програмування Python

Python - це високорівнева, інтерпретована мова програмування з динамічною типізацією, яка заслужила визнання завдяки своїй зрозумілості та читабельності коду. Створена в кінці 1980-х років Гвідо ван Россумом, Python забезпечує ефективний процес розробки завдяки своїй чіткій синтаксичній структурі та великій стандартній бібліотеці.

Приклад простого Python-коду, що виводить "Привіт, світ!":

```
print("Привіт, світ!")
```

Приклад роботи зі списками:

```
# Створення списку
my_list = [1, 2, 3, 4, 5]
print(my_list)
# Додавання елементу до списку
my_list.append(6)
print(my_list)
# Видалення елементу зі списку
my_list.remove(3)
print(my_list)
```

Приклад використання циклу та умовних операторів:

```
for i in my_list:
    if i % 2 == 0:
        print(f"{i} - парне число")
    else:
        print(f"{i} - непарне число")
```

Основною ідеологією Python є читабельність та простота коду, що дозволяє програмістам висловлювати концепції у меншій кількості рядків коду, порівняно з C++ або Java. Python підтримує кілька парадигм програмування, включаючи

об'єктно-орієнтоване, імперативне, функціональне та процедурне програмування, що робить його надзвичайно гнучким.

Завдяки своїй універсальності, Python використовується в широкому спектрі застосувань, від веб-розробки (за допомогою фреймворків як Django та Flask) до наукових та чисельних обчислень (з бібліотеками як NumPy, SciPy, Matplotlib), розробки програмного забезпечення, автоматизації задач, машинного навчання та штучного інтелекту.

Фреймворки в Python - це набори готових модулів та утиліт, які спрощують розробку програмного забезпечення, надаючи структуру та набір інструментів для побудови додатків. Вони визначають архітектуру додатку, пропонуючи шаблони та рішення для звичайних задач, таких як управління даними, взаємодія з базою даних, обробка запитів та відповідей, і маршрутизація в веб-додатках.

Веб-фреймворки допомагають розробникам створювати веб-додатки, забезпечуючи інструменти для простішого управління HTTP запитами та відповідями, маршрутизацією, сесіями, безпекою та іншими аспектами веб-розробки.

- Django - один з найпопулярніших високорівневих фреймворків, який слідує принципу "батарейки включені". Він має вбудовану підтримку адміністративної панелі, форм, аутентифікації користувачів, і багато іншого, що дозволяє швидко розгортати повнофункціональні веб-додатки.
- Flask - "мікро" фреймворк, який надає більш гнучкий, але мінімалістичний підхід до веб-розробки. Flask пропонує основні інструменти для маршрутизації та шаблонів, залишаючи більшість архітектурних рішень на вибір розробника.

Для роботи з науковими та чисельними обчисленнями, Python пропонує ряд потужних фреймворків, що дозволяють ефективно обробляти дані, виконувати математичні обчислення та аналізувати великі набори даних.

- NumPy - фундаментальна бібліотека для наукових обчислень в Python, яка надає підтримку великих багатовимірних масивів та матриць, разом з великим набором високорівневих математичних функцій для роботи з цими масивами.

- Pandas - бібліотека, яка надає зручні структури даних та інструменти для аналізу та маніпуляції даними. Особливо ефективна для роботи з табличними даними (наприклад, з даними, імпортованими з Excel).

У сфері машинного навчання Python також має величезний набір фреймворків, які спрощують розробку та тренування моделей.

- TensorFlow та Keras - потужні бібліотеки для глибокого навчання, які дозволяють легко створювати та тренувати нейронні мережі з використанням високорівневих API.
- Scikit-learn - бібліотека для машинного навчання, яка надає простий і ефективний інструментарій для аналізу даних та моделювання, включаючи класифікацію, регресію, кластеризацію та зниження розмірності.

Використання цих фреймворків в нашому Telegram боті дозволяє розширити його можливості, від простого завантаження та обробки відео до виконання складних обчислень та аналізу даних.

Модульність в Python дозволяє розробникам організовувати програмний код у вигляді окремих модулів, які можна використовувати повторно в різних частинах програми або навіть у різних проектах. Модулі - це просто файли Python, що містять визначення функцій, класів та змінних, а також виконуваний код.

Пакети в Python - це спосіб структурування простору імен модулів шляхом використання "точкової" нотації. Наприклад, модуль moduleA в пакеті package імпортується як package.moduleA. Python має сотні тисяч пакетів, що охоплюють практично всі області розробки програмного забезпечення, від веб-розробки до наукових досліджень.

Python відомий своєю гнучкістю та інтеграцією з іншими мовами програмування та системами. Це означає, що Python може використовуватися як міст між різними компонентами програмного забезпечення, створеними на різних мовах. Наприклад, можлива інтеграція Python з C/C++ для оптимізації продуктивності критичних за швидкістю ділянок коду, або з Java для використання в Android-розробці через Jython. [5]

Python є ідеальним вибором для проектів, що вимагають швидкої розробки та

високої модульності завдяки своїй простоті, гнучкості та потужній екосистемі. Його спроможність легко інтегруватися з іншими технологіями робить його незамінним інструментом у сучасному світі програмування.

3.2. Аналіз бібліотек для створення телеграм-ботів

Для створення телеграм-ботів розробники мають у своєму розпорядженні кілька потужних бібліотек Python, кожна з яких має свої особливості, переваги та способи використання. Двома популярними бібліотеками є python-telegram-bot та Telebot. Ці бібліотеки надають інструменти для спілкування з Telegram API, дозволяючи легко створювати ботів, які можуть взаємодіяти з користувачами через повідомлення, команди та різні типи вмісту.

python-telegram-bot - це бібліотека, що дозволяє розробникам працювати з Telegram Bot API безпосередньо в Python. Вона відрізняється великою кількістю можливостей і гнучкістю, пропонуючи як прості, так і розширені інструменти для створення різноманітних ботів.

Приклад простого exo-бота:

```
from telegram.ext import Updater, CommandHandler, MessageHandler, Filters

def start(update, context):
    update.message.reply_text('Привіт! Я exo-бот. Надішліть мені повідомлення, і я повторю його.')

def echo(update, context):
    update.message.reply_text(update.message.text)

updater = Updater("ТВІЙ_ТОКЕН", use_context=True)
dp = updater.dispatcher
dp.add_handler(CommandHandler("start", start))
dp.add_handler(MessageHandler(Filters.text & ~Filters.command, echo))
```

```
updater.start_polling()
updater.idle()
```

Telebot (або pyTelegramBotAPI) - це ще одна популярна бібліотека, яка спрощує роботу з Telegram API. Вона менш гнучка порівняно з python-telegram-bot, але її простота робить її ідеальною для швидкого старту та реалізації простих ботів.

Приклад простого ехо-бота з використанням Telebot:

```
import telebot

bot = telebot.TeleBot("ТВІЙ_ТОКЕН")

@bot.message_handler(commands=['start'])
def send_welcome(message):
    bot.reply_to(message, "Привіт! Я ехо-бот. Надішліть мені повідомлення, і я повторю його.")

@bot.message_handler(func=lambda message: True)
def echo_all(message):
    bot.reply_to(message, message.text)

bot.polling()
```

Обидва приклади демонструють створення простого ехо-бота, який відповідає на повідомлення користувача тим же текстом. Вибір між python-telegram-bot та Telebot залежить від особистих переваг розробника та вимог до функціональності бота. В обох випадках, завдяки великій спільноті та багатій документації, розробники можуть легко знайти відповіді на свої питання та розширити можливості своїх ботів

3.3. Інтеграція з YouTube API

Інтеграція телеграм-бота з YouTube API відкриває широкі можливості для розробників, дозволяючи виконувати пошук відео, отримувати інформацію про

канали, відео, плейлисти, а також управляти ними. Для роботи з YouTube API необхідно створити проект у Google Cloud Platform та отримати API ключ. Нижче наведено кроки, які допоможуть у цьому процесі:

Крок 1: Створення проекту в Google Cloud Platform

1. Відкрийте Google Cloud Console та увійдіть у свій акаунт Google.
2. Натисніть на "Select a project" у верхній панелі навігації, а потім "NEW PROJECT".
3. Введіть назву проекту та виберіть організацію та локацію. Натисніть "CREATE".

Крок 2: Включення YouTube Data API v3

1. У вашому проекті на Google Cloud Platform перейдіть до "API & Services > Dashboard".
2. Натисніть "ENABLE APIS AND SERVICES" для пошуку доступних API.
3. Знайдіть "YouTube Data API v3" та натисніть "ENABLE" для активації API у вашому проекті.

Крок 3: Створення API ключа

1. Після активації YouTube Data API перейдіть до "Credentials" у меню "API & Services".
2. Натисніть "CREATE CREDENTIALS" та оберіть "API key".
3. Система створить API ключ, який ви зможете використовувати у вашому боті. Збережіть цей ключ.

Використання YouTube API у Python

Для використання YouTube API у вашому телеграм-боті на Python, ви можете використовувати бібліотеку google-api-python-client. Нижче наведено приклад, як виконати пошук відео на YouTube:

```
from googleapiclient.discovery import build

youtube_api_key = 'ВАШ_API_КЛЮЧ'
youtube = build('youtube', 'v3', developerKey=youtube_api_key)
```

```
request = youtube.search().list(
    q='Python programming', # запит для пошуку
    part='snippet',
    type='video',
    maxResults=5
)
response = request.execute()

for item in response['items']:
    print(f"Назва: {item['snippet']['title']}\nURL: https://www.youtube.com/watch?v={item['id']['videoId']}\n")
```

Цей код створює клієнт YouTube API, виконує пошук відео за ключовими словами "Python programming" та виводить назви знайдених відео разом з посиланнями на них.

Інтеграція з YouTube API дозволяє телеграм-боту надавати користувачам багатий функціонал, пов'язаний із пошуком та обробкою відеоконтенту, розширюючи його можливості та забезпечуючи більш залучений досвід користування.

3.4. Принципи роботи з відео та аудіо потоками

Робота з відео та аудіо потоками є ключовою частиною функціональності багатьох сучасних застосунків, включаючи телеграм-ботів, які забезпечують завантаження, обробку та розподіл медіаконтенту з YouTube. Цей розділ надає огляд основних принципів роботи з відео та аудіо потоками, використовуючи наш телеграм-бот як приклад.

Принципи роботи з відео та аудіо потоками

1. Отримання потоків даних: відео на YouTube можуть містити кілька потоків (stream), які включають відео та аудіо інформацію у різних роздільних

здатностях та форматах. Завданням бота є ідентифікація доступних потоків для конкретного відео.

2. Вибір потоків: користувачі можуть вибрати певну якість відео або аудіо, яке вони хочуть завантажити. Бот аналізує доступні потоки та надає користувачам опції для вибору.
3. Завантаження потоків: після вибору потоку, бот ініціює процес завантаження, зберігаючи відео або аудіо локально або на хмарному сховищі.
4. Обробка потоків: у випадку необхідності виконання додаткової обробки (наприклад, конвертація формату або обрізка відео), бот застосовує відповідні інструменти та бібліотеки для обробки медіаданих.
5. Надсилання потоків: після завантаження та обробки, бот надсилає відео або аудіо потік користувачу у телеграм-чаті або генерує посилання для завантаження.

Інструменти та технології

Для роботи з відео та аудіо потоками бот використовує різноманітні інструменти та технології:

- Pytube: ця бібліотека Python дозволяє легко отримувати інформацію про відео з YouTube та завантажувати відео та аудіо потоки. [5]
- FFmpeg: могутній інструмент для обробки мультимедійних файлів, який використовується для конвертації медіафайлів у різні формати, обрізки, злиття відеокліпів тощо.

Приклад коду

Отримання та завантаження відео з YouTube:

```
from pytube import YouTube

def download_video(video_url, resolution):
    yt = YouTube(video_url)
    video = yt.streams.filter(res=resolution).first()
    video.download(output_path='videos/', filename='video.mp4')
```

video_url = 'https://www.youtube.com/watch?v=dQw4w9WgXcQ'

download_video(video_url, '720p')

Цей приклад демонструє, як за допомогою бібліотеки Pytube, бот може завантажувати відео вибраної якості з YouTube. Обробка та надсилання відео потім може бути здійснена з використанням додаткових інструментів, таких як FFmpeg для конвертації форматів або обрізки відео.

Ці принципи та інструменти роботи з відео та аудіо потоками є основою для створення функціональних та зручних у використанні телеграм-ботів, здатних обробляти медіаконтент. [6]

4. ПРАКТИЧНА ЧАСТИНА

4.1. Постановка задачі та вимоги до бота

Практичною метою дипломної роботи є розробка телеграм-бота, призначеного для завантаження, обробки та розповсюдження мультимедійного контенту з таких популярних платформ, як YouTube, TikTok, Instagram і Facebook. Створення цього бота покликане вирішити проблему складності отримання і подальшого використання відео- та аудіоматеріалів користувачами без спеціалізованих знань у сфері мультимедіа або програмування.

Телеграм-бот повинен забезпечити простий, швидкий та зручний доступ користувачів до мультимедійних ресурсів, що дозволить значно спростити завантаження контенту з вищезгаданих платформ без потреби використовувати сторонні ресурси та сервіси.

Для реалізації поставленої задачі було сформовано перелік основних вимог до програмного забезпечення.

Основні функціональні вимоги:

1. Підтримка завантаження відео та аудіо з різних платформ:
    - Автоматичне визначення платформи за наданим посиланням (YouTube, TikTok, Instagram, Facebook).
    - Завантаження відеофайлів у форматі .mp4.
    - Виділення аудіодоріжки та конвертація у формат .mp3.
2. Пошук та завантаження музики за назвою:
    - Автоматичний пошук першого відповідного результату на платформі YouTube за запитом користувача.
    - Завантаження знайденого аудіотреку та відправка користувачу.
3. Автоматичне обрізання аудіофрагментів:
    - Можливість задавати часовий інтервал (наприклад, 00:30-01:45) для створення коротких аудіофрагментів.
    - Автоматичне створення і відправлення обрізаних аудіофрагментів.

4. Підтримка завантаження повних плейлистів з YouTube:
    - Завантаження усіх відео з вибраного плейлиста.
    - Збереження плейлиста в окремій папці із структурою, яка відображає порядок і назви відео.
5. Інтуїтивний та зрозумілий інтерфейс взаємодії з користувачем:
    - Простий набір команд (/start, /video, /audio, /music, /cut_audio, /playlist) з чіткими інструкціями щодо їх використання.
    - Повідомлення про хід виконання задач, помилки або успішне завершення процесів.

Нефункціональні вимоги:

- Швидкодія: мінімізація часу очікування під час завантаження та відправки медіафайлів.
- Стабільність: забезпечення стабільної роботи при одночасній обробці декількох запитів.
- Безпека: дотримання стандартів безпеки, відсутність зберігання персональних даних користувачів та конфіденційних матеріалів.
- Простота розгортання: використання стандартних технологій (Python, бібліотеки aiogram та yt-dlp), що дозволяють легко розгорнути бот на різних платформах.

Основні технології, що використовуються для реалізації задачі:

- Мова програмування: Python (версія 3.10).
- Фреймворк для Telegram-бота: aiogram.
- Бібліотека для роботи з мультимедіа-контентом: yt-dlp.
- Інструмент для роботи з аудіо та відео: ffmpeg.
- Управління залежностями: pip, virtualenv.
- Конфігурація та змінні оточення: python-dotenv.

Очікуваний результат:

У результаті реалізації проєкту має бути отримано стабільний, надійний та інтуїтивно зрозумілий телеграм-бот, який повністю відповідає заявленим вимогам і здатний ефективно обслуговувати запити користувачів на завантаження та обробку

мультимедійного контенту з YouTube та інших популярних соціальних платформ. Бот має бути зручним як для пересічних користувачів, так і для адміністраторів, забезпечуючи швидке розгортання і легку підтримку.

4.2. Структура та архітектура проєкту

Для успішної реалізації телеграм-бота з підтримкою завантаження та обробки мультимедійних файлів з платформ YouTube, TikTok, Instagram та Facebook була обрана модульна структура та багаторівнева архітектура. Це дозволяє ефективно підтримувати, масштабувати та розширювати функціонал проєкту у майбутньому.

Архітектура проєкту побудована за принципом поділу відповідальності між окремими компонентами:

1. Рівень взаємодії з користувачем (інтерфейсний шар)
   Включає в себе обробку команд та повідомлень від користувачів за допомогою бібліотеки aiogram. Відповідає за отримання та обробку запитів, а також надання зрозумілих відповідей користувачам.
2. Рівень бізнес-логіки
   Реалізує алгоритми завантаження та обробки мультимедійного контенту. Використовує бібліотеку yt-dlp для отримання контенту з платформ та ffmpeg для конвертації і обробки аудіо/відео-файлів.
3. Рівень роботи з даними (шар зберігання)
   Забезпечує тимчасове зберігання завантажених файлів у спеціально створеному каталозі, звідки вони надалі пересилаються користувачам та видаляються після завершення операцій.

Структура файлів була ретельно продумана для зручності підтримки, розширення функціоналу і забезпечення зрозумілості проєкту:

youload/
├── main.py # Основний файл запуску бота
├── config.py # Конфігураційний файл (TOKEN, шляхи до файлів)
├── requirements.txt # Перелік залежностей проєкту

```
├── .env                    # Файл із конфіденційними змінними (TOKEN)
├── temp/                   # Тимчасове сховище завантажених файлів
│   └── playlists/          # Каталог для збереження завантажених плейлистів
└── modules/                # Модулі бізнес-логіки
    ├── video_downloader.py     # Завантаження відео (YouTube, TikTok, Instagram, Facebook)
    ├── music_downloader.py     # Завантаження і пошук музики (YouTube)
    └── audio_processor.py      # Модуль обробки та обрізки аудіо
```

Опис основних модулів:

1. Модуль взаємодії з користувачем (main.py)

Цей модуль забезпечує взаємодію з користувачами через Telegram, обробку команд та формування відповідей. Включає в себе наступні команди:

- /start – стартове повідомлення з описом доступних функцій.
- /video – завантаження відео з підтримуваних платформ.
- /audio – завантаження аудіо із відео.
- /music – пошук та завантаження музики за запитом.
- /playlist – завантаження відео з повного плейлиста YouTube.
- /cut_audio – автоматична обрізка аудіофрагментів за тайм-кодами.

2. Модуль завантаження відео (video_downloader.py)

Використовує бібліотеку yt-dlp для завантаження медіаконтенту. Основні функції модуля:

- Визначення платформи за посиланням.
- Завантаження відео у формат .mp4.
- Завантаження плейлистів YouTube з відповідною структурою.

3. Модуль завантаження музики (music_downloader.py)

Цей модуль відповідає за пошук та завантаження музики із YouTube за запитом:

- Пошук відео на YouTube.
- Завантаження аудіо доріжки у форматі .mp3.

4. Модуль обробки аудіо (audio_processor.py)

Використовує інструмент ffmpeg для обрізання аудіофрагментів за вказаними часовими проміжками:

- Обрізка аудіо за допомогою точних тайм-кодів.
- Створення коротких аудіофрагментів за запитом користувача.

Використані технології та бібліотеки:

- Python 3.10 – основна мова програмування.
- aiogram – бібліотека для створення Telegram-ботів.
- yt-dlp – універсальний інструмент для завантаження мультимедіа контенту.
- ffmpeg – обробка, конвертація та обрізка аудіо- та відеофайлів.
- python-dotenv – керування конфіденційними змінними.

Переваги обраної структури та архітектури:

- Модульність: кожен модуль відповідає за окрему функцію, що спрощує тестування та підтримку.
- Масштабованість: можливість легко додавати нові функції та сервіси.
- Простота супроводу: чітке розмежування обов'язків спрощує діагностику та виправлення помилок.

Таким чином, обрана архітектура та структура проєкту забезпечують високий рівень надійності, гнучкості, зрозумілості та зручності як для користувачів, так і для розробників.

4.3. Реалізація основних функцій

Для забезпечення повної функціональності розробленого Telegram-бота були реалізовані основні функції, описані в постановці задачі. Нижче наведено опис реалізації цих функцій та ключові фрагменти програмного коду.

Основні функції, реалізовані в боті:

1. Завантаження відео з платформ YouTube, TikTok, Instagram, Facebook
2. Завантаження аудіо з відео
3. Пошук та завантаження музики за назвою
4. Підтримка завантаження YouTube-плейлистів

5. Обрізання аудіо за заданими часовими мітками

1. Завантаження відео з платформ (YouTube, TikTok, Instagram, Facebook)

Для завантаження мультимедійних файлів було використано бібліотеку yt-dlp, яка автоматично визначає джерело за посиланням та завантажує контент у форматі MP4.

Код реалізації функції:

```
@staticmethod
def download_video(url: str):
    unique_id = secrets.token_hex(6)
    temp_video_path = os.path.join(STORAGE_FOLDER, f"{unique_id}.mp4")

    ydl_opts = {
        "format": "bv*+ba/best",
        "outtmpl": temp_video_path,
        "merge_output_format": "mp4",
        "noplaylist": True,
    }

    try:
        with yt_dlp.YoutubeDL(ydl_opts) as ydl:
            info = ydl.extract_info(url, download=True)
            title = info.get("title", "video").replace("/", "_").replace("\\", "_")

        final_video_path = os.path.join(STORAGE_FOLDER, f"{title}.mp4")
        os.rename(temp_video_path, final_video_path)

        return final_video_path, title
    except Exception as e:
        logging.error(f"Помилка при завантаженні відео: {e}")
        return None, None
```

2. Завантаження аудіо з відео

Для отримання аудіо з відео була реалізована функція, яка завантажує аудіодоріжку та конвертує її в формат MP3:

Код реалізації функції:

```
@staticmethod
def download_audio(url: str):
```

```
    unique_id = secrets.token_hex(6)
    temp_audio_path = os.path.join(STORAGE_FOLDER, f"{unique_id}.mp3")

    ydl_opts = {
        "format": "bestaudio/best",
        "outtmpl": temp_audio_path,
        "postprocessors": [{
            "key": "FFmpegExtractAudio",
            "preferredcodec": "mp3",
            "preferredquality": "256",
        }],
    }

    try:
        with yt_dlp.YoutubeDL(ydl_opts) as ydl:
            info = ydl.extract_info(url, download=True)
            title = info.get("title", "audio").replace("/", "_").replace("\\", "_")

        final_audio_path = os.path.join(STORAGE_FOLDER, f"{title}.mp3")

        if os.path.exists(temp_audio_path + ".mp3"):
            os.rename(temp_audio_path + ".mp3", final_audio_path)
        elif os.path.exists(temp_audio_path):
            os.rename(temp_audio_path, final_audio_path)

        return final_audio_path, title
    except Exception as e:
        logging.error(f"Помилка при завантаженні аудіо: {e}")
        return None, None
```

3. Пошук та завантаження музики за назвою

Для реалізації пошуку музики використовується вбудований функціонал yt-dlp для пошуку відео за ключовими словами:

Код реалізації функції пошуку:

```
@staticmethod
def search_music(query: str):
    ydl_opts = {
        "quiet": True,
```

```
    "default_search": "ytsearch1",
    "skip_download": True,
    "force_generic_extractor": True
}

with yt_dlp.YoutubeDL(ydl_opts) as ydl:
    try:
        info = ydl.extract_info(query, download=False)
        if "entries" in info and len(info["entries"]) > 0:
            return {
                "title": info["entries"][0]["title"],
                "link": info["entries"][0]["webpage_url"]
            }
        else:
            return None
    except Exception as e:
        logging.error(f"Помилка при пошуку відео: {e}")
        return None
```

Після пошуку, музика завантажується як аудіофайл за допомогою функції завантаження аудіо (описано вище).

4. Підтримка завантаження YouTube-плейлистів

Для завантаження плейлистів з YouTube реалізована окрема функція, що завантажує всі відео плейлиста у відповідну папку:

Код реалізації завантаження плейлистів:

```
@staticmethod
def download_playlist(url: str):
    playlist_folder = os.path.join(STORAGE_FOLDER, "playlists")
    os.makedirs(playlist_folder, exist_ok=True)

    ydl_opts = {
        "format": "bv*+ba/best",
        "outtmpl": os.path.join(playlist_folder, "%(playlist)s/%(playlist_index)s - %(title)s.%(ext)s"),
        "merge_output_format": "mp4",
        "yes_playlist": True,
    }
```

```
    try:
        with yt_dlp.YoutubeDL(ydl_opts) as ydl:
            info = ydl.extract_info(url, download=True)
            playlist_title = info.get("title", "playlist").replace("/", "_")

        return os.path.join(playlist_folder, playlist_title)
    except Exception as e:
        logging.error(f"Помилка при завантаженні плейлиста: {e}")
        return None
```

5. Обрізання аудіо за часовими мітками

Для автоматичного обрізання аудіофайлів використано утиліту ffmpeg. Реалізовано наступний функціонал:

Код реалізації функції обрізання:

```
@staticmethod
def cut_audio(input_path: str, start_time: str, end_time: str, title: str):
    output_path = input_path.replace(".mp3", f"_{start_time.replace(':', '')}-{end_time.replace(':', '')}.mp3")

    command = [
        "ffmpeg", "-i", input_path, "-ss", start_time, "-to", end_time,
        "-c", "copy", output_path, "-y"
    ]

    try:
        subprocess.run(command, check=True)
        return output_path if os.path.exists(output_path) else None
    except Exception as e:
        logging.error(f"Помилка при обрізці аудіо: {e}")
        return None
```

Обробка помилок і логування

У всіх описаних функціях реалізовано обробку винятків і ведення журналу помилок через модуль logging. Це дозволяє вчасно відслідковувати можливі проблеми та ефективно їх вирішувати, забезпечуючи стабільність роботи бота.

Завдяки реалізованим функціям забезпечується повноцінний користувацький досвід взаємодії з Telegram-ботом, що дозволяє користувачам ефективно

завантажувати і обробляти контент з популярних соціальних платформ та відеохостингів. Використання модульної архітектури сприяє простоті підтримки та розширення проєкту в майбутньому.

4.4. Інструкція для користувача

1. Початок роботи з ботом (див. рис. 3.1)

- Відкрийте чат з ботом у Telegram.
- Введіть команду /start для отримання стартового повідомлення, яке містить опис доступних функцій:
    - /video — Завантажити відео з YouTube, TikTok, Instagram або Facebook.
    - /audio — Завантажити аудіо з відео.
    - /music назва пісні — Завантажити музику за назвою.
    - /playlist — Завантажити весь плейлист.
    - /cut_audio — Обрізати аудіо з відео за вказаним тайм-кодом.

Рисунок 3.1 – Початкове повідомлення з командами бота

2. Завантаження відео (див. рис. 3.2)

- Введіть команду /video та вставте посилання на потрібне відео.
- Після введення посилання бот починає процес завантаження, що позначається повідомленням «⏳ Завантажуємо відео, зачекайте...».
- Після завершення завантаження бот надсилає відео у чат з назвою контенту.

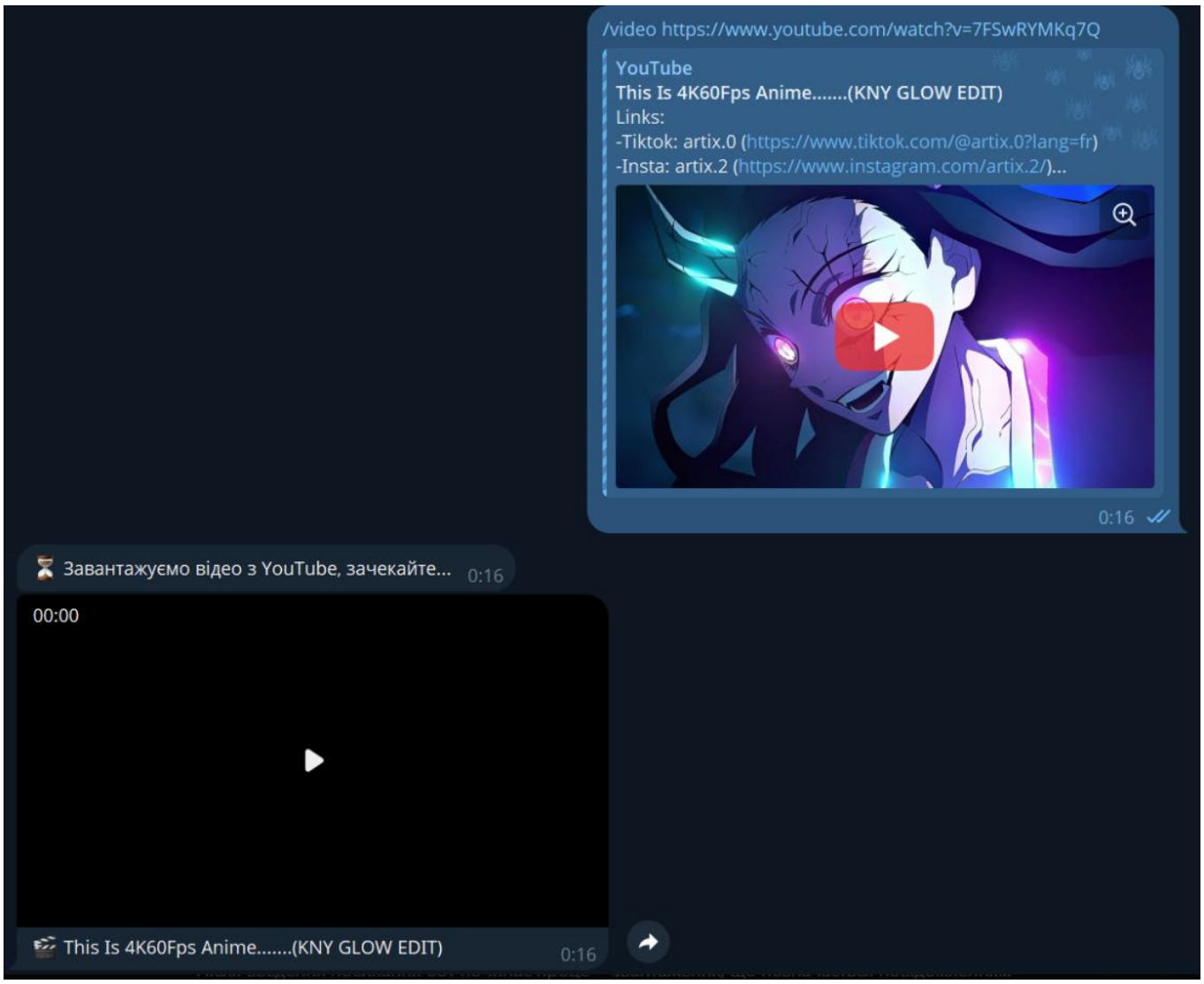

Рисунок 3.2 – Процес завантаження відео

3. Завантаження аудіо (див. рис. 3.3)

- Введіть команду /audio і посилання на потрібне відео.
- Бот повідомить вас про початок завантаження аудіофайлу повідомленням «⌛ Завантажуємо аудіо, зачекайте...».
- Після завершення процесу ви отримаєте аудіофайл із назвою відео.

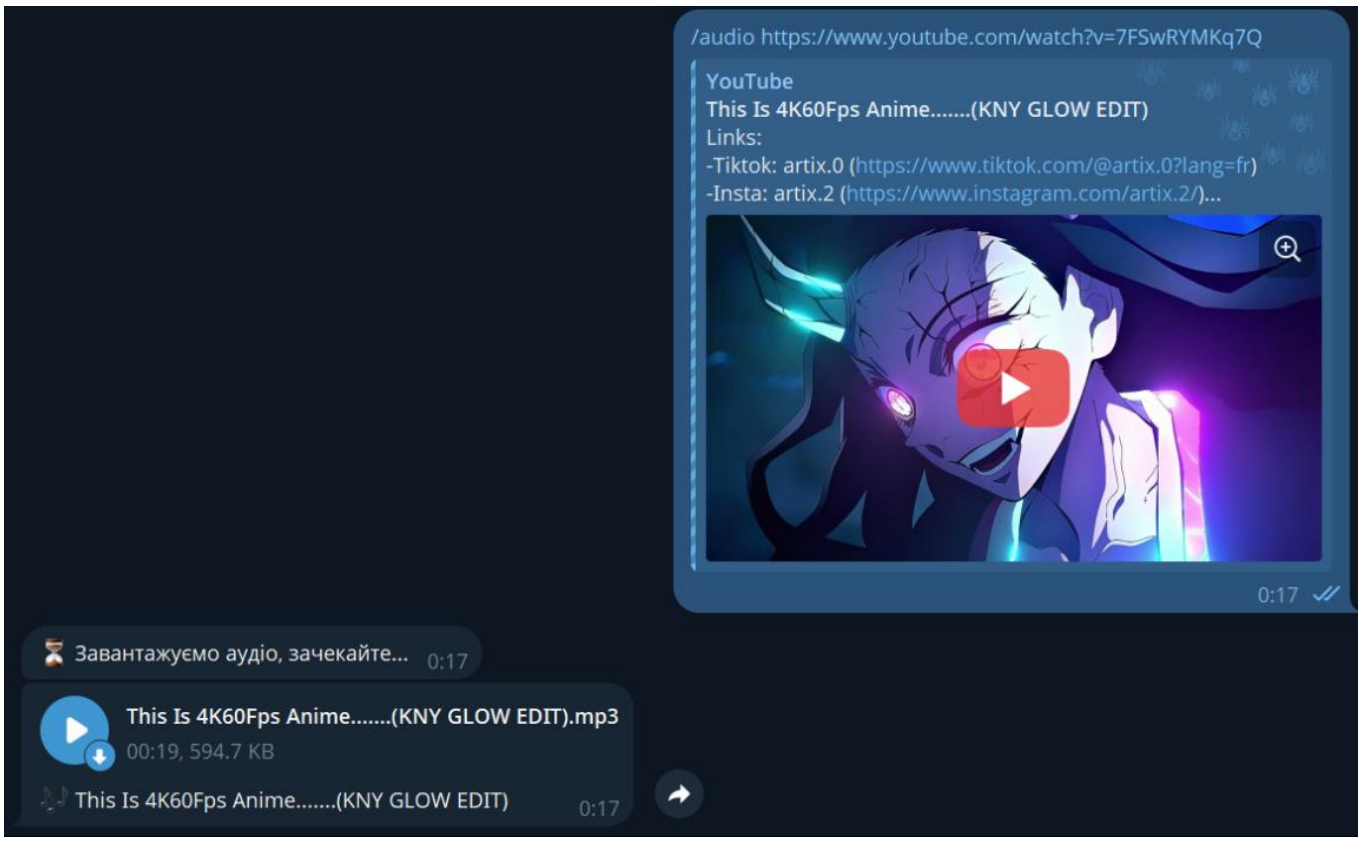

Рисунок 3.3 – Процес завантаження аудіо з відео

4. Завантаження музики за назвою (див. рис. 3.4)

- Введіть команду /music з назвою пісні (наприклад: /music tito after dark).
- Бот почне пошук музики, повідомивши «🔍 Пошук музики: [назва]».
- Після успішного пошуку ви отримаєте аудіофайл з назвою знайденої композиції.

Рисунок 3.4 – Пошук та завантаження музики за назвою

5. Завантаження плейлистів (див. рис. 3.5)

- Введіть команду /playlist і вставте посилання на YouTube-плейлист.
- Бот повідомить про початок завантаження плейлиста.
- Після завершення ви отримаєте повідомлення із зазначенням шляху до папки, де розташовані файли плейлиста.

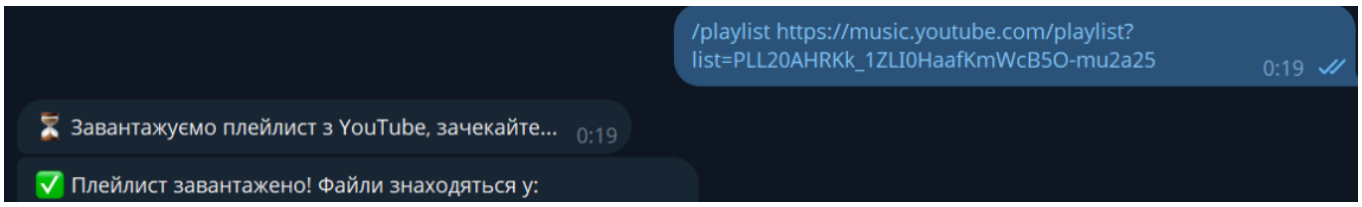

Рисунок 3.5 – Завантаження плейлистів

6. Обрізка аудіо з відео (див. рис. 3.6)

- Введіть команду /cut_audio, посилання на відео і часовий проміжок, що

потрібно вирізати (наприклад: /cut_audio [посилання] 00:30-00:55).

- Бот завантажить аудіо, після чого обрізає заданий фрагмент.
- Ви отримаєте обрізаний аудіофайл у чаті.

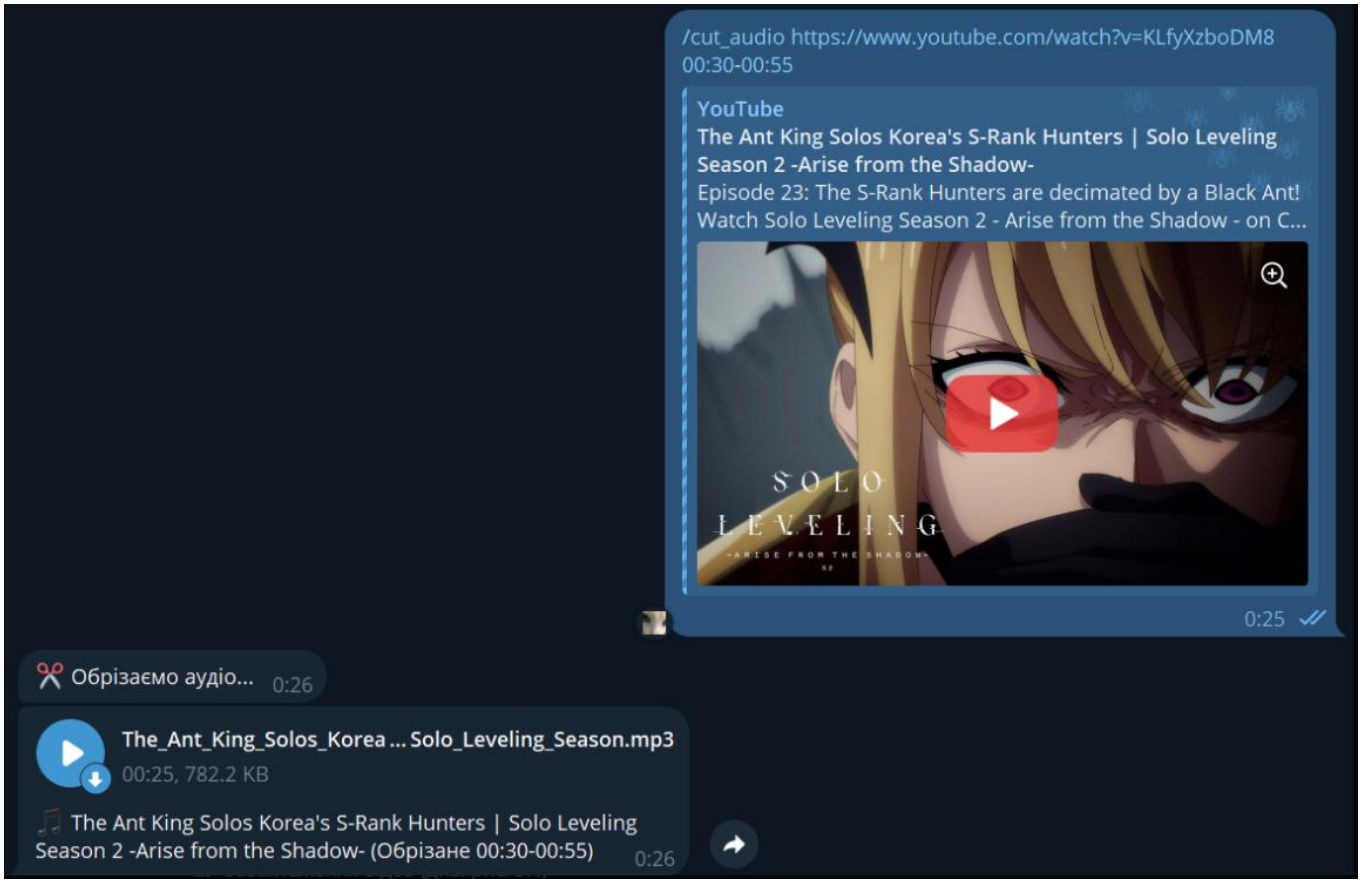

Рисунок 3.6 – Обрізка аудіо за тайм-кодом

Таким чином, розроблений Telegram-бот забезпечує зручний інтерфейс для завантаження та обробки медіаконтенту з популярних сервісів, спрощуючи доступ до відео- та аудіоматеріалів.

ВИСНОВКИ

У дипломній роботі розглянуто процес розробки програмного забезпечення Telegram-бота, призначеного для завантаження та обробки медіаконтенту з популярних платформ, таких як YouTube, TikTok, Instagram та Facebook. Реалізований бот значно спрощує доступ до мультимедійних матеріалів, дозволяючи користувачам швидко завантажувати відео, аудіо, музичні композиції за назвою, цілі плейлисти, а також виконувати обрізку аудіо за заданими тайм-кодами.

У рамках виконання роботи було успішно вирішено поставлені завдання:

- Проаналізовано сучасні технології створення ботів у месенджерах, зокрема Telegram, та визначено найбільш підходящі бібліотеки та інструменти для реалізації проєкту.
- Вибрано мову програмування Python та використано бібліотеки aiogram та yt-dlp, що дозволили забезпечити ефективну взаємодію бота з медіаплатформами.
- Реалізовано модульну архітектуру програмного забезпечення, яка включає окремі компоненти для завантаження відео та аудіо, пошуку музики, роботи з плейлистами і обрізки аудіотреків.
- Забезпечено підтримку популярних медіаплатформ (YouTube, TikTok, Instagram, Facebook), що розширило сферу застосування розробленого рішення.
- Створено детальну інструкцію для користувачів, що дозволяє легко орієнтуватись у функціях та можливостях програмного забезпечення.

Практичне значення роботи полягає у створенні зручного та інтуїтивно зрозумілого інструменту для роботи з мультимедійним контентом, що може бути використаний як для особистих цілей, так і для освітнього процесу та медіапроектів. Розроблений бот відповідає сучасним вимогам до якості програмного забезпечення, характеризується високою продуктивністю,

стабільністю роботи та простотою в експлуатації.

Подальші перспективи розвитку проєкту передбачають інтеграцію нових платформ, покращення інтерфейсу та розширення функціональних можливостей для ще більш ефективної роботи з мультимедійним контентом.

СПИСОК ЛІТЕРАТУРИ

1. Лутц М. Вивчаємо Python, том 1, 5-е видання. - Пер. с англ. - Науковий Світ, 2023. - 832 с.
2. Гвідо ван Россум. Python. Повний опис мови програмування. - ДМК Прес, 2018. - 560 с.
3. Мартін Скіннер. Професійна розробка на Python: Вступ до розробки з використанням Python, Git, Vim, PostgreSQL і Flask. - БВІ, 2019. - 400 с.
4. Ерік Метезен. Python для дітей. - Видавництво Старого Лева, 2020. - 352 с.
5. Джейк Вандерплас. Python для науковців і інженерів. - ДМК Прес, 2018. - 592
6. Аллен Б. Дауни. Навчальний курс "Python 3". - Видавництво БХВ, 2019. - 496
7. Google Cloud Conlose [Електронний ресурс] – Режим доступу: https://cloud.google.com/
8. Python.org. Офіційна документація Python. [Електронний ресурс] – Режим доступу: https://docs.python.org/3/
9. Telegram Bot API. Офіційна документація. [Електронний ресурс] – Режим доступу: https://core.telegram.org/bots/api
10. Ольховська О. В. Методичні рекомендації до виконання кваліфікаційної роботи для студентів спеціальності 122 Комп’ютерні науки освітня програма «Комп’ютерні науки» ступеня бакалавра / О. В. Ольховська, О. О. Черненко. – Полтава : ПУЕТ, 2022. – 67 с. – 1 електрон. опт. диск (CVD-ROM).

ДОДАТОК А.

Main.py

```
import asyncio
import os
import logging
from aiogram import Bot, Dispatcher, types
from aiogram.filters import Command
from aiogram.types import FSInputFile
from aiogram.client.default import DefaultBotProperties
from config import TOKEN
from music_downloader import MusicDownloader
from video_downloader import VideoDownloader

bot = Bot(token=TOKEN, default=DefaultBotProperties(parse_mode="HTML"))
dp = Dispatcher()

@dp.message(Command("start"))
async def start_command(message: types.Message):
    """ Стартове повідомлення """
    text = (
        " 🖐 Вітаю! Надішліть мені посилання на відео YouTube, TikTok, Instagram або Facebook.\n"
        "\n<b>Команди:</b>\n"
        "/video - Завантажити відео 📹\n"
        "/audio - Завантажити аудіо 🎵\n"
        "/music <code>назва пісні</code> - Завантажити музику 🎶\n"
        "/playlist - Завантажити плейлист 🎵\n"
        "/cut_audio - Обрізати аудіо 00:30-01:45 🎵"
    )

    await message.answer(text)

@dp.message(Command("music"))
async def handle_music_request(message: types.Message):
    """ Обробка запиту на завантаження музики """
    query = message.text.replace("/music", "").strip()
```

```
    if not query:
        await message.answer("❌ Будь ласка, введіть назву пісні.")
        return

    await message.answer(f"🔍 Пошук музики: {query}...")

    music_info = MusicDownloader.search_music(query)
    if not music_info:
        await message.answer("❌ Не вдалося знайти пісню.")
        return

    title = music_info["title"].replace(" ", "_")
    url = music_info["link"]

    await message.answer(f"🎵 Завантажуємо: {title}\n🔗{url}")

    audio_path = MusicDownloader.download_audio(url, title)

    if audio_path:
        try:
            await message.answer_audio(FSInputFile(audio_path), caption=f"🎶{title}")
        except Exception as e:
            await message.answer("❌ Помилка при відправці аудіо.")
            logging.error(f"Помилка при відправці аудіо: {e}")
        finally:
            os.remove(audio_path)
    else:
        await message.answer("❌ Не вдалося завантажити аудіо.")

@dp.message(Command("video"))
async def handle_video_request(message: types.Message):
    """ Обробка запиту на завантаження відео """
    url = message.text.replace("/video", "").strip()

    if not url:
        await message.answer("❌ Будь ласка, введіть посилання на відео.")
```

```
            return

        platform = VideoDownloader.get_platform(url)
        if not platform:
            await message.answer("⚠ Непідтримуване посилання. Надішліть правильне посилання з YouTube, TikTok, Instagram або Facebook.")
            return

        await message.answer(f"⏳ Завантажуємо відео з {platform}, зачекайте...")

        video_path, title = VideoDownloader.download_video(url)

        if video_path:
            try:
                await message.answer_video(FSInputFile(video_path), caption=f"🎬 {title}")
            except Exception as e:
                await message.answer("❌ Помилка при відправці відео.")
                logging.error(f"Помилка при відправці відео: {e}")
            finally:
                os.remove(video_path)
        else:
            await message.answer("❌ Не вдалося завантажити відео.")

    @dp.message(Command("audio"))
    async def handle_audio_request(message: types.Message):
        """ Обробка запиту на завантаження аудіо """
        url = message.text.replace("/audio", "").strip()

        if not url:
            await message.answer("❌ Будь ласка, введіть посилання на відео.")
            return

        await message.answer("⏳ Завантажуємо аудіо, зачекайте...")

        audio_path, title = VideoDownloader.download_audio(url)

        if audio_path:
```

```
            try:
                await message.answer_audio(FSInputFile(audio_path), caption=f"🎶{title}")
            except Exception as e:
                await message.answer("❌ Помилка при відправці аудіо.")
                logging.error(f"Помилка при відправці аудіо: {e}")
            finally:
                os.remove(audio_path)
        else:
            await message.answer("❌ Не вдалося завантажити аудіо.")

    @dp.message(Command("playlist"))
    async def handle_playlist_request(message: types.Message):
        """ Обробка запиту на завантаження плейлиста """
        url = message.text.replace("/playlist", "").strip()

        if not url:
            await message.answer("❌ Будь ласка, введіть посилання на плейлист.")
            return

        platform = VideoDownloader.get_platform(url)
        if not platform:
            await message.answer("⚠️ Непідтримуване посилання. Надішліть правильне посилання на плейлист.")
            return

        await message.answer(f"⏳ Завантажуємо плейлист з {platform}, зачекайте...")

        playlist_path = VideoDownloader.download_playlist(url)

        if playlist_path:
            try:
                await message.answer(f"✅ Плейлист завантажено! Файли знаходяться у: `{playlist_path}`", parse_mode="Markdown")
            except Exception as e:
                await message.answer("❌ Помилка при відправці плейлиста.")
                logging.error(f"Помилка при відправці плейлиста: {e}")
        else:
```

```
            await message.answer("❌ Не вдалося завантажити плейлист.")

        @dp.message(Command("cut_audio"))
        async def handle_cut_audio_request(message: types.Message):
            """ Обробка запиту на обрізку аудіо """
            args = message.text.replace("/cut_audio", "").strip().split()

            if len(args) < 2:
                    await message.answer("❌ Формат: <code>/cut_audio [посилання] [час_початку-час_кінця]</code>\n\nНаприклад:\n<code>/cut_audio https://youtu.be/xyz123 00:30-01:45</code>")
                return

            url, time_range = args[0], args[1]

            if "-" not in time_range:
                await message.answer("❌ Неправильний формат часу. Використовуйте формат **00:30-01:45**.")
                return

            start_time, end_time = time_range.split("-")

            await message.answer(f"⏳ Завантажуємо аудіо з {url}...")

            audio_path, title = VideoDownloader.download_audio(url)

            if not audio_path:
                await message.answer("❌ Не вдалося завантажити аудіо.")
                return

            await message.answer("✂️ Обрізаємо аудіо...")

            cut_audio_path = MusicDownloader.cut_audio(audio_path, start_time, end_time, title)

            if cut_audio_path:
                try:
                    await message.answer_audio(FSInputFile(cut_audio_path), caption=f"🎵 {title} (Обрізане {start_time}-{end_time})")
```

```
        except Exception as e:
            await message.answer("✗ Помилка при відправці аудіо.")
            logging.error(f"Помилка при відправці аудіо: {e}")
        finally:
            os.remove(audio_path)
            os.remove(cut_audio_path)
    else:
        await message.answer("✗ Не вдалося обрізати аудіо.")

async def main():
    """ Запуск бота """
    await dp.start_polling(bot)

if __name__ == "__main__":
    asyncio.run(main())
```

music_downloader.py

```
import os
import secrets
import logging
import yt_dlp
import re
from config import STORAGE_FOLDER
import subprocess

class MusicDownloader:
    """ Завантажує музику з YouTube """

    @staticmethod
    def search_music(query: str):
        """ Використовує yt-dlp для пошуку музики на YouTube """
        ydl_opts = {
            "quiet": True,
            "default_search": "ytsearch1",  # Пошук тільки одного відео
            "skip_download": True,
            "force_generic_extractor": True
        }
```

```
    with yt_dlp.YoutubeDL(ydl_opts) as ydl:
        try:
            info = ydl.extract_info(query, download=False)
            if "entries" in info and len(info["entries"]) > 0:
                return {
                    "title": info["entries"][0]["title"],
                    "link": info["entries"][0]["webpage_url"]
                }
            else:
                return None
        except Exception as e:
            logging.error(f"Помилка при пошуку відео: {e}")
            return None

@staticmethod
def sanitize_filename(filename: str):
    """ Очищає назву файлу від заборонених символів """
    return re.sub(r'[\\/*?:"<>|]', "", filename)

@staticmethod
def download_audio(url: str, title: str):
    """ Завантажує аудіофайл за посиланням """
    unique_id = secrets.token_hex(6)
    temp_audio_path = os.path.join(STORAGE_FOLDER, f"{unique_id}.mp3")

    ydl_opts = {
        "format": "bestaudio/best",
        "outttmpl": temp_audio_path,
        "postprocessors": [{
            "key": "FFmpegExtractAudio",
            "preferredcodec": "mp3",
            "preferredquality": "256",
        }],
    }

    try:
        with yt_dlp.YoutubeDL(ydl_opts) as ydl:
```

```
                ydl.download([url])

            final_audio_path = os.path.join(STORAGE_FOLDER, f"{title}.mp3")

            if os.path.exists(temp_audio_path + ".mp3"):
                os.rename(temp_audio_path + ".mp3", final_audio_path)
            else:
                os.rename(temp_audio_path, final_audio_path)

            return final_audio_path
        except Exception as e:
            logging.error(f"Помилка при завантаженні аудіо: {e}")
            return None

    @staticmethod
    def cut_audio(input_path: str, start_time: str, end_time: str, title: str):
        """ Обрізає аудіофайл за тайм-кодами """
            output_path = input_path.replace(".mp3", f"_{start_time.replace(':', '')}-{end_time.replace(':',
'')}.mp3")

            command = [
                "ffmpeg", "-i", input_path, "-ss", start_time, "-to", end_time,
                "-c", "copy", output_path, "-y"
            ]

            try:
                subprocess.run(command, check=True)
                return output_path if os.path.exists(output_path) else None
            except Exception as e:
                logging.error(f"Помилка при обрізці аудіо: {e}")
                return None
```

Video_downloader.py

```
import os
import secrets
import logging
import yt_dlp
from config import STORAGE_FOLDER
```

```
class VideoDownloader:
    """ Завантажує відео або аудіо з YouTube, TikTok, Instagram, Facebook """

    @staticmethod
    def get_platform(url: str):
        """ Визначає платформу за посиланням """
        if "youtube.com" in url or "youtu.be" in url:
            return "YouTube"
        elif "tiktok.com" in url:
            return "TikTok"
        elif "instagram.com" in url:
            return "Instagram"
        elif "facebook.com" in url or "fb.watch" in url:
            return "Facebook"
        else:
            return None

    @staticmethod
    def download_video(url: str):
        """ Завантажує відео з YouTube, TikTok, Instagram, Facebook """
        unique_id = secrets.token_hex(6)
        temp_video_path = os.path.join(STORAGE_FOLDER, f"{unique_id}.mp4")

        ydl_opts = {
            "format": "bv*+ba/best",
            "outttmpl": temp_video_path,
            "merge_output_format": "mp4",
            "noplaylist": True,
        }

        try:
            with yt_dlp.YoutubeDL(ydl_opts) as ydl:
                info = ydl.extract_info(url, download=True)
                title = info.get("title", "video").replace("/", "_").replace("\\", "_")

            final_video_path = os.path.join(STORAGE_FOLDER, f"{title}.mp4")
```

```
            os.rename(temp_video_path, final_video_path)

            return final_video_path, title
        except Exception as e:
            logging.error(f"Помилка при завантаженні відео: {e}")
            return None, None

    @staticmethod
    def download_playlist(url: str):
        """ Завантажує всі відео з плейлиста """
        playlist_folder = os.path.join(STORAGE_FOLDER, "playlists")
        os.makedirs(playlist_folder, exist_ok=True)

        ydl_opts = {
            "format": "bv*+ba/best",
            "outtmpl": os.path.join(playlist_folder, "%(playlist)s/%(playlist_index)s - %(title)s.%(ext)s"),
            "merge_output_format": "mp4",
            "yes_playlist": True,  # Дозволяємо завантаження плейлистів
        }

        try:
            with yt_dlp.YoutubeDL(ydl_opts) as ydl:
                info = ydl.extract_info(url, download=True)
                playlist_title = info.get("title", "playlist").replace("/", "_").replace("\\", "_")

            return os.path.join(playlist_folder, playlist_title)
        except Exception as e:
            logging.error(f"Помилка при завантаженні плейлиста: {e}")
            return None

    @staticmethod
    def download_audio(url: str):
        """ Завантажує аудіо """
        unique_id = secrets.token_hex(6)
        temp_audio_path = os.path.join(STORAGE_FOLDER, f"{unique_id}.mp3")

        ydl_opts = {
            "format": "bestaudio/best",
```

```
            "outttmpl": temp_audio_path,
            "postprocessors": [{
                "key": "FFmpegExtractAudio",
                "preferredcodec": "mp3",
                "preferredquality": "256",
            }],
        }

        try:
            with yt_dlp.YoutubeDL(ydl_opts) as ydl:
                info = ydl.extract_info(url, download=True)
                title = info.get("title", "audio").replace("/", "_").replace("\\", "_")  # Уникнення некоректних символів

                final_audio_path = os.path.join(STORAGE_FOLDER, f"{title}.mp3")

                # Перевіряємо, який файл існує
                if os.path.exists(temp_audio_path + ".mp3"):
                    os.rename(temp_audio_path + ".mp3", final_audio_path)
                elif os.path.exists(temp_audio_path):
                    os.rename(temp_audio_path, final_audio_path)
                else:
                    logging.error(f"Файл не знайдено: {temp_audio_path} або {temp_audio_path}.mp3")
                    return None, None

                return final_audio_path, title
        except Exception as e:
            logging.error(f"Помилка при завантаженні аудіо: {e}")
            return None, None
```

Config.py

```
import os
from dotenv import load_dotenv

load_dotenv()  # Завантажує змінні з .env файлу

TOKEN = os.getenv("TOKEN")

STORAGE_FOLDER = "./temp/"
```

os.makedirs(STORAGE_FOLDER, exist_ok=True)