Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття ступеня вищої освіти магістра

за спеціальністю 121 – Інженерія програмного забезпечення

На тему: Розробка та дослідження алгоритмів автоматизованого завантаження та обробки медіафайлів з використанням Telegram-бота

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студент гр. ІПЗ-24м _____ / Р. О. Фарафонов /

Керівник кваліфікаційної роботи _____ / А. А. Азарян /

Завідувач кафедри _____ / А. М. Стрюк /

Кривий Ріг

2025

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: магістр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри

_____ А. М. Стрюк

«___» _____ 20__ р.

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ІПЗ-24м Фарафонову Ростиславу Олеговичу

1. Тема: Розробка та дослідження алгоритмів автоматизованого завантаження та обробки медіафайлів з використанням Telegram-бота затверджено наказом по КНУ № 204с від «10» квітня 2025 р.

2. Термін подання студентом закінченої роботи: «__» ____20___ р.

3. Вихідні дані по роботі: розроблювана система повинна забезпечувати автоматизовану обробку медіафайлів, підтримувати роботу з різними джерелами даних та реалізовувати необхідні функції для їх отримання, оптимізації та подальшого використання в рамках програмного комплексу.

4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): виконати аналіз сучасних методів завантаження та обробки медіафайлів, обґрунтувати вибір інструментів і технологій, розробити алгоритми роботи Telegram-бота, спроєктувати архітектуру програмного забезпечення, реалізувати та протестувати програмний комплекс, провести дослідження ефективності його роботи та виконати аналіз економічної доцільності розробленої системи.

5. Перелік ілюстративного матеріалу: блок-схеми розроблених алгоритмів, схема взаємодії модулів Telegram-бота, UML-діаграми, приклади оброблених медіафайлів, знімки екранних форм роботи системи.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Огляд літератури та матеріалів за тематикою автоматизованого завантаження й обробки медіафайлів	10.01.2025 – 25.01.2025
2	Аналіз існуючих теоретичних методів обробки відео та аудіо, а також підходів до оптимізації медіаданих	26.01.2025 – 10.02.2025
3	Проведення критичного порівняльного аналізу програмних рішень для завантаження медіафайлів	11.02.2025 – 28.02.2025
4	Формулювання актуальності, мети та завдань роботи у контексті автоматизації завантаження й обробки медіафайлів	11.03.2025 – 25.03.2025
5	Оформлення матеріалів першого розділу роботи	26.03.2025 – 30.04.2025
6	Розробка математичних, структурних і функціональних моделей алгоритмів завантаження, визначення параметрів медіа та їх обробки	01.05.2025 – 31.05.2025
7	Розробка структур даних, основних класів та алгоритмічних рішень програмного комплексу Telegram-бота	01.06.2025 – 15.06.2025
8	Оформлення матеріалів другого розділу роботи	16.11.2025 – 20.11.2025
9	Розробка програмних модулів завантаження, обробки, стиснення та логування медіафайлів	16.06.2025 – 31.08.2025
10	Оформлення матеріалів третього розділу роботи	01.09.2025 – 15.09.2025
11	Проведення дослідження якості роботи Telegram-бота, оцінка ефективності алгоритмів обробки та оптимізації медіафайлів	16.09.2025 – 15.10.2025
12	Оформлення матеріалів четвертого розділу роботи	16.10.2025 – 05.11.2025
13	Аналіз економічної доцільності впровадження автоматизованої системи завантаження та обробки медіафайлів	06.11.2025 – 20.11.2025
14	Остаточне оформлення пояснювальної записки	21.11.2025 – 05.12.2025

Дата видачі завдання: «____» _____ 20___ р.

Студент _____ / Р. О. Фарафонов /

Керівник роботи _____ / А. А. Азарян /

РЕФЕРАТ

МЕДІАФАЙЛИ, АВТОМАТИЗАЦІЯ, ЗАВАНТАЖЕННЯ, ОБРОБКА, ОПТИМІЗАЦІЯ, АЛГОРИТМИ, ПРОГРАМНИЙ КОМПЛЕКС, TELEGRAM-БОТ, СИСТЕМА, ЕФЕКТИВНІСТЬ

Пояснювальна записка: 73 с., 17 табл., 29 рис., 5 дод., 30 джерел.

Мета кваліфікаційної роботи – розробка програмного комплексу, що забезпечує автоматизоване завантаження, обробку та оптимізацію медіафайлів із використанням Telegram-бота.

Об’єкт проєктування – програмна система автоматизованого отримання та обробки медіафайлів.

У теоретичній частині кваліфікаційної роботи проведено критичний аналіз сучасних підходів до автоматизованого завантаження, конвертації та стиснення відео- й аудіофайлів, розглянуто принципи роботи наявних програмних засобів, зокрема медіаконвертерів, бібліотек і сервісів, що взаємодіють із різними онлайн-платформами. Визначено ключові обмеження, проблеми та технічні особливості обробки мультимедійних даних, сформульовано актуальність та завдання дослідження.

У практичній частині роботи проаналізовано принципи побудови програмних систем, призначених для завантаження та опрацювання медіафайлів, розроблено алгоритми визначення параметрів відео та аудіо, вибору оптимальних форматів та режимів стиснення, а також моделі роботи програмного комплексу в умовах обмежень на розмір та якість файлів.

У спеціальній частині створено Telegram-бот, що реалізує функції автоматизованого завантаження та обробки медіафайлів, інтегрований із інструментами yt-dlp та ffmpeg. Виконано дослідження технічної ефективності прийнятих рішень, зокрема швидкодії, якості обробки, стабільності взаємодії з різними джерелами даних. Зроблено висновки, що розроблений програмний комплекс забезпечує оптимізацію медіафайлів відповідно до заданих параметрів та дозволяє значно спростити процес їх отримання і підготовки.

У розділі «Аналіз економічної ефективності інновації» обґрунтовано доцільність використання розробленої технології автоматизованого завантаження та обробки медіафайлів, показано переваги її впровадження у сфері цифрового контенту та користувацьких сервісів.

Основні положення кваліфікаційної роботи можуть бути представлені у вигляді тез доповіді на науково-технічних конференціях, що стосуються систем автоматизації, обробки мультимедійних даних та застосування бот-платформ.

ABSTRACT

MEDIA FILES, AUTOMATION, DOWNLOADING, PROCESSING, OPTIMIZATION, ALGORITHMS, SOFTWARE COMPLEX, TELEGRAM BOT, SYSTEM, EFFICIENCY

Explanatory note: 73 p., 17 tables, 29 figures, 5 appendices, 30 sources.

The purpose of the thesis is to develop a software complex that provides automated downloading, processing, and optimization of media files using a Telegram bot.

The object of the project is a software system for automated retrieval and processing of media files.

In the theoretical part of the thesis, a critical analysis of modern approaches to automated downloading, conversion, and compression of video and audio files is conducted. The principles of existing software tools, including media converters, libraries, and services interacting with various online platforms, are examined. Key limitations, challenges, and technical specifics of multimedia data processing are identified, and the relevance and objectives of the research are formulated.

In the practical part, the principles of building software systems designed for media file downloading and processing are analyzed. Algorithms for detecting video and audio parameters, selecting optimal formats and compression modes, as well as models of the software complex operating under constraints on file size and quality, are developed.

In the special part, a Telegram bot is created that implements automated downloading and processing of media files and integrates with tools such as yt-dlp and ffmpeg. The technical efficiency of the proposed solutions is investigated, including performance, processing quality, and stability of interaction with different data sources. It is concluded that the developed software complex ensures effective media optimization according to predefined parameters and significantly simplifies the process of obtaining and preparing media content.

In the section “Analysis of the economic efficiency of innovation,” the economic feasibility of implementing the developed automated media processing technology is substantiated, demonstrating its advantages for digital content workflows and user-oriented services.

The main results of the thesis may be presented in the form of conference papers related to automation systems, multimedia data processing, and bot-platform applications.

ЗМІСТ

ВСТУП 8
1 АНАЛІТИЧНИЙ ОГЛЯД І ПОСТАНОВКА ЗАВДАННЯ 10
1.1 Огляд сучасних підходів до автоматизованого завантаження медіафайлів 10
1.2 Технології обробки медіафайлів та їх оптимізація 12
1.3 Огляд платформ і бібліотек для розробки Telegram-ботів 15
1.4 Аналіз існуючих аналогів та обґрунтування вибору теми 18
1.5 Формулювання завдань та основної ідеї роботи 21
2 ВІДОМОСТІ ПРО ПРЕДМЕТ (ОБ'ЄКТ) ДОСЛІДЖЕННЯ 23
2.1 Загальна характеристика Telegram API та принципи взаємодії 23
2.2 Обґрунтування вибору математичних методів і алгоритмів обробки медіафайлів 26
2.3 Розробка математичної моделі процесу завантаження та обробки медіа 28
2.4 Формалізація алгоритмів автоматизації обробки даних 31
2.5 Розробка структурної та функціональної моделі програмного забезпечення 33
2.6 Розробка моделі бази даних Telegram-бота 37
2.7 Модель інтерфейсу та основні класи програмного забезпечення 39
3 ПРИКЛАДНА ЧАСТИНА 43
3.1 Вибір інструментарію та середовища розробки 43
3.2 Реалізація основних модулів Telegram-бота 44
3.3 Організація обробки медіафайлів 46
3.4 Структура бази даних і механізми зберігання інформації 47
3.5 Інтерфейс користувача та керівництво користувача 50
4 ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ І АНАЛІЗ ЕФЕКТИВНОСТІ 56
4.1 Методика проведення експериментів 56
4.2 Аналіз ефективності алгоритмів завантаження та обробки 59
4.3 Порівняння з існуючими рішеннями 62
4.4 Аналіз економічної ефективності розробки 65
4.5 Практичне значення результатів 66
ВИСНОВКИ 68

ПЕРЕЛІК ПОСИЛАНЬ 70
Додаток А 72
Додаток Б 75
Додаток В 77
Додаток Г 79
Додаток Д 81

ВСТУП

Стрімке зростання кількості цифрового контенту, що генерується сучасними мультимедійними платформами, зумовлює потребу у створенні ефективних інструментів для його автоматизованого отримання та обробки. Більшість популярних сервісів поширення відео- та аудіоматеріалів використовують складні механізми доступу до даних, що ускладнює можливість оперативного завантаження медіафайлів користувачами. У цих умовах особливої актуальності набувають програмні засоби, здатні забезпечувати отримання, оптимізацію та передачу цифрових матеріалів у режимі, максимально наближеному до реального часу. Telegram є однією з найбільш гнучких платформ для створення таких рішень завдяки наявності відкритого API, підтримці асинхронних процесів і широким можливостям інтеграції зі сторонніми сервісами.

Об’єктом дослідження є процес автоматизованого завантаження та обробки медіафайлів у середовищі Telegram, а предметом дослідження алгоритми, моделі та програмні засоби, що забезпечують отримання, оптимізацію, конвертацію і передачу медіаданих за допомогою Telegram-бота. Метою роботи є створення програмного забезпечення, яке поєднує механізми завантаження медіафайлів із зовнішніх джерел, алгоритми їх обробки та подальшу передачу користувачеві, забезпечуючи підвищення зручності та швидкодії таких операцій.

Для досягнення поставленої мети необхідно виконати аналітичний огляд сучасних методів обробки медіаданих, дослідити можливості Telegram API щодо реалізації асинхронної взаємодії, розробити математичні моделі процесів завантаження та трансформації медіафайлів, формалізувати алгоритми автоматизації обробки, спроєктувати структурну, функціональну та інформаційну моделі програмного забезпечення, реалізувати програмний комплекс та виконати експериментальні дослідження його ефективності. Додатково передбачається проведення розрахунків економічної доцільності використання розробленої програмної інновації.

У процесі дослідження застосовуються методи математичного моделювання процесів обробки інформації, алгоритмічні та структурні підходи до аналізу даних, методи об’єктно орієнтованого та модульного програмування, засоби асинхронної обробки подій, методи експериментального вимірювання ефективності програмних систем та порівняльний аналіз існуючих рішень у сфері автоматизації роботи з медіафайлами.

Дослідження виконується у взаємозв’язку з науковими роботами кафедри моделювання та програмного забезпечення, що спрямовані на створення інноваційних інформаційних технологій, удосконалення процесів цифрової обробки даних та розробку адаптивних інтерфейсів взаємодії користувача з програмними системами. Отримані результати відповідають загальній тематиці наукових досліджень кафедри й сприяють розвитку напрямів, пов’язаних із автоматизацією обробки мультимедійної інформації та підвищенням ефективності прикладних програмних засобів.

1 АНАЛІТИЧНИЙ ОГЛЯД І ПОСТАНОВКА ЗАВДАННЯ

1.1 Огляд сучасних підходів до автоматизованого завантаження медіафайлів

Автоматизоване завантаження медіафайлів є складовою великої кількості сучасних програмних систем, які працюють з цифровим контентом. Його розвиток зумовлений потребою швидкого отримання даних із різних джерел, а також подальшої можливості виконання їх обробки та оптимізації. Розвиток мережевих технологій і зростання популярності мультимедійних платформ сприяли появі численних інструментів та сервісів, що забезпечують завантаження відео, аудіо та зображень. Проте механізми доступу до контенту стають дедалі складнішими, що зумовлює потребу у створенні гнучких алгоритмів, здатних адаптуватися до різних форматів і протоколів.

Одним із найбільш поширених напрямів є використання спеціалізованих бібліотек для отримання медіафайлів через мережеві протоколи. Більшість сучасних рішень базуються на HTTP(S) як універсальному транспортному середовищі, яке забезпечує можливість завантаження даних у форматі потоків. Цей підхід є основою більшості інструментів командного рядка, сховищ даних та програмних бібліотек, що працюють із мультимедіа. Використання потокового завантаження дозволяє оптимізувати використання пам’яті та забезпечує можливість обробки медіафайлів у процесі отримання.

Наступним важливим напрямом є застосування універсальних утиліт, що підтримують складні алгоритми витягнення даних із мультимедійних платформ. Найбільш поширеними прикладами є програми, що аналізують структуру вебсторінок, обробляють динамічні запити, враховують механізми захисту контенту та адаптуються до змін у роботі сервісів. Такі інструменти здатні виконувати інтеграцію з декількома протоколами, підтримувати різні якісні характеристики відео та аудіо і забезпечувати завантаження оптимізованих форматів.

Окреме місце займають підходи, що використовують API сервісів, які надають доступ до контенту безпосередньо. У таких випадках медіафайли

отримуються не шляхом аналізу сторінок, а за рахунок взаємодії з офіційними інтерфейсами, які дають можливість працювати з метаданими, параметрами якості та засобами контролю доступу. Перевагами цього підходу є точність, стабільність та відповідність політикам сервісів. Однак використання API обмежене умовами доступу, а також вимогами щодо автентифікації та авторизації.

Важливою тенденцією є розвиток асинхронних алгоритмів завантаження, що дозволяють ефективно працювати з великою кількістю запитів одночасно. Вони забезпечують оптимальне використання ресурсів і дають можливість створювати масштабовані програмні системи. Такі алгоритми особливо поширені у сервісах, які забезпечують автоматизовану роботу з медіаданими у режимі реального часу. Асинхронні механізми дозволяють одночасно обробляти декілька потоків, виконувати перетворення даних у фоновому режимі та передавати результати користувачеві без значних затримок.

Окремим різновидом підходів є використання інструментів, що виконують інтеграцію завантаження з подальшою обробкою. У таких системах отримання медіафайлів виконується з урахуванням параметрів, необхідних для їх подальшої оптимізації, конвертації або стискання. Це дозволяє будувати повністю автоматизовані рішення, в яких завантаження є лише першим етапом більш складного процесу обробки. Інтегровані інструменти особливо актуальні для сервісів, що працюють із великими обсягами мультимедійних даних і потребують адаптації медіафайлів під різні вимоги користувача.

Сучасні підходи до автоматизованого завантаження мультимедійних даних характеризуються високим рівнем універсальності, адаптивності та орієнтацією на ефективність обробки. Вони поєднують класичні мережеві технології з алгоритмічними методами оптимізації та забезпечують можливість використання у різноманітних програмних комплексах, включаючи системи, що працюють із Telegram-ботами. У межах подальших розділів буде розглянуто конкретні інструменти, методи та алгоритми, які

застосовуються у процесі розробки програмного забезпечення для автоматизованого завантаження та обробки медіафайлів.

1.2 Технології обробки медіафайлів та їх оптимізація

Обробка медіафайлів є одним із ключових етапів у роботі систем автоматизованого завантаження та передачі цифрового контенту. Вона охоплює широкий спектр операцій, пов’язаних із перетворенням, оптимізацією, зменшенням розміру та адаптацією файлів до вимог кінцевого користувача або обмежень середовища, у якому вони будуть використовуватися. Зростання обсягів мультимедійної інформації та підвищення вимог до швидкості її обробки стимулюють активний розвиток технологій, орієнтованих на ефективне перетворення відео, аудіо та графічних матеріалів.

Одним із найбільш поширених напрямів є застосування інструментів для кодування та перекодування відеофайлів. У сучасних програмних рішеннях кодування здійснюється із використанням популярних кодеків, таких як H.264, H.265, VP9 та AV1, кожен із яких забезпечує компроміс між якістю відео та його розміром.

Параметр	H.264 (AVC)	H.265 (HEVC)	VP9	AV1
Ефективність стиснення	Середня	На 30–50% краще за H.264	На ~30% краще за H.264	На 30–50% краще за VP9
Швидкодія кодування	Висока	Низька–середня	Середня	Низька
Апаратна підтримка	Практично всі пристрої	Нові CPU/GPU, мобільні SoC	Широка підтримка браузерів	Обмежена (нові GPU/CPU)
Якість при низьких бітрейтах	Посередня	Висока	Висока	Дуже висока
Переваги	Широка сумісність	Дуже ефективне стиснення	Безкоштовний, відкритий	Найкраща якість і стиснення
Недоліки	Нижча якість	Високі вимоги до CPU	Повільніше за H.264	Важке кодування
Основні області застосування	YouTube, стримінги, мобільні пристрої	4K/8K відео, Smart TV	YouTube, веббраузери	YouTube, Netflix, сучасні стримінги

Рисунок 1.2.1 Порівняння відеокодеків

Використання таких кодеків дозволяє значно зменшити обсяг файлів, що є критично важливим у випадках обмеженої швидкості передачі даних або необхідності швидкого обміну інформацією. Сучасні технології перекодування також підтримують можливість адаптивного вибору параметрів стиснення залежно від цільової якості та доступних ресурсів.

Значну роль у сучасних системах обробки мультимедійних даних відіграють інструменти для роботи з аудіофайлами. Вони забезпечують зміну формату, бітрейту, тривалості та інших параметрів звукових матеріалів. Поширені формати, такі як MP3, AAC, WAV та OGG, використовуються залежно від необхідного рівня якості та сумісності з іншими системами. Технології стиснення аудіо дозволяють досягти значного зменшення розміру файлу при мінімальних втратах якості, що є важливою характеристикою при передачі даних через мобільні мережі або у середовищах із обмеженою пропускною здатністю.

Параметр	MP3	AAC	WAV	OGG (Vorbis)
Тип кодування	Втратне (lossy)	Втратне (lossy)	Без втрат (lossless PCM)	Втратне (lossy, open-source)
Якість звучання	Середня	Вища за MP3 при однаковому бітрейті	Дуже висока (оригінал)	Вища за MP3, наближена до AAC
Ефективність стиснення	Середня	Висока	Відсутня (великий розмір файлу)	Висока
Розмір файлів	Малий / середній	Малий	Дуже великий	Малий / середній
Апаратна підтримка	Дуже широка	Широка (мобільні пристрої, стримінги)	Повсюдна	Обмежена (не всі пристрої підтримують)
Переваги	Сумісність всюди	Краща якість при низькому бітрейті	Оригінальна якість, ідеально для монтажу	Безкоштовний, відкритий, якість > MP3
Недоліки	Якість нижча за сучасні кодеки	Вищі вимоги до кодування	Гігантські файли	Погана підтримка у старих пристроях
Основні сфери застосування	Музика, подкасти, веб	Стрімінги, відеоплатформи, мобільні сервіси	Студійний звук, монтаж, архівація	Ігри, open-source проєкти, веб

Рисунок 1.2.2 Порівняння аудіоформатів

Окремим напрямом є обробка зображень, яка включає зміну формату, зменшення роздільної здатності, стиснення та оптимізацію для вебсередовищ. Сучасні алгоритми оптимізації зображень дозволяють зменшити їх розмір без значної втрати візуальної якості. Широко використовуються формати JPEG, PNG, WebP та інші, що забезпечують різні рівні компресії залежно від особливостей контенту. Технології, орієнтовані на автоматизацію обробки зображень, дають змогу адаптувати їх до вимог різних платформ та пристроїв, зокрема мобільних.

Параметр	JPEG	PNG	WebP	SVG
Тип графіки	Растрова	Растрова	Растрова	Векторна
Тип стиснення	Втратне (lossy)	Без втрат (lossless)	Lossy + lossless	Без втрат (векторне)
Якість при малому розмірі	Середня	Висока	Дуже висока	Ідеальна (масштаб без втрат)
Розмір файлів	Малий	Середній / великий	Дуже малий	Малий (в залежності від складності)
Підтримка прозорості	Немає	Є (full alpha)	Є	Є
Підтримка анімації	Немає	Немає	Є	Обмежена (через SMIL/JS)
Переваги	Малий розмір, підходить для фото	Ідеальне для UI, скрінів, графіки	Висока якість + маленький розмір, сучасний стандарт	Масштабування без втрат, ідеально для іконок
Недоліки	Артефакти при стисканні	Великий розмір	Не всі старі браузери підтримують	Не підходить для фотографій
Основні сфери застосування	Фото, соціальні мережі	Інтерфейси, скріни, логотипи	Веб, мобільні додатки, оптимізовані зображення	Лого, іконки, діаграми, векторні UI

Рисунок 1.2.3 Порівняння форматів зображень

У системах, що об’єднують кілька типів мультимедійних даних, важливою є підтримка комплексної обробки, яка передбачає послідовне виконання операцій над різними форматами. Сучасні інструменти дозволяють створювати багатоступеневі сценарії, що включають завантаження, декодування, зміну параметрів, повторне кодування та формування результату у кінцевий формат. Такий підхід забезпечує максимальну гнучкість і дозволяє адаптувати процес обробки під конкретні задачі та вимоги.

Важливим елементом ефективної роботи з медіафайлами є оптимізація процесів обробки, яка включає використання багатопоточного та асинхронного виконання операцій. Багатопотокові алгоритми дозволяють розподіляти завдання між кількома процесорами або ядрами, що значно прискорює обробку ресурсомістких операцій, зокрема перекодування відео високої роздільної здатності. Асинхронні технології, у свою чергу, забезпечують можливість одночасного виконання декількох операцій без блокування основного потоку, що є ключовим для систем, що працюють у режимі реального часу.

У сучасних програмних системах активно застосовуються інструменти автоматизації обробки медіафайлів, які дозволяють інтегрувати алгоритми оптимізації у загальний робочий процес. Ці інструменти надають можливість створення сценаріїв обробки, що включають аналіз вхідних даних, вибір параметрів оптимізації, формування результатів і передачу користувачеві. Використання автоматизованих технологій забезпечує підвищення ефективності та стабільності обробки, зменшує навантаження на ресурси та покращує якість отриманих результатів.

1.3 Огляд платформ і бібліотек для розробки Telegram-ботів

Розробка Telegram-ботів є одним із найбільш поширених напрямів створення сучасних сервісних систем, оскільки Telegram пропонує стабільний, продуктивний та простий у використанні інтерфейс взаємодії. Завдяки відкритому Telegram Bot API, розробники мають можливість створювати боти різного рівня складності, інтегрувати їх із зовнішніми сервісами,

автоматизовувати робочі процеси та забезпечувати зручну взаємодію з користувачами. У цьому підрозділі розглянуто найбільш поширені платформи та програмні бібліотеки, що використовуються у розробці Telegram-ботів.

Однією з найпопулярніших бібліотек є Python-telegram-bot, яка забезпечує високу стабільність роботи та широкий набір інструментів для обробки повідомлень, команд і подій. Бібліотека довгий час залишалася стандартом де-факто для синхронної розробки Telegram-ботів на Python. Її перевагами є зручний синтаксис, зрозуміла система обробників та доступність детальної документації. Однак унаслідок зростання вимог до продуктивності та потреби в асинхронній взаємодії популярність цього інструмента поступово зменшується.

Більш сучасним підходом є використання асинхронних бібліотек, серед яких найбільш поширеною є aiogram. Вона побудована на базі асинхронної парадигми та використовує можливості Python asyncio. Асинхронна модель забезпечує високу швидкодію, що є важливою властивістю для систем, які обробляють велику кількість запитів або працюють із ресурсомісткими операціями, такими як завантаження і обробка медіафайлів. Aiogram підтримує гнучку систему диспетчеризації, інтеграцію з базами даних, middleware-рівень та механізми FSM, що дозволяє створювати складні багатокрокові сценарії взаємодії.

Ще однією бібліотекою, що широко використовується у розробці Telegram-ботів, є Telebot (PyTelegramBotAPI). Вона орієнтована на прості та середні за складністю проєкти і забезпечує синхронну модель виконання. Хоча Telebot є менш продуктивною порівняно з асинхронними рішеннями, її популярність зумовлена простотою використання та мінімальним порогом входу.

У середовищі JavaScript популярними є такі інструменти, як Telegraf і grammy. Telegraf базується на Node.js і забезпечує сучасний механізм обробки подій, що є зручним для побудови інтерактивних ботів. Бібліотека підтримує middleware-архітектуру, що робить її придатною для розширення та адаптації.

Grammmy є більш новим інструментом, оптимізованим для продуктивності, та пропонує модульну структуру і широкий набір функціональних можливостей.

У сфері багатоплатформної розробки варто згадати такі інструменти, як MadelineProto та Telethon. Вони працюють не через Bot API, а через Telegram MTProto-протокол, що забезпечує доступ до можливостей, недоступних звичайним ботам. Використання MTProto дозволяє отримувати інформацію про користувачів, працювати з групами на рівні клієнта та виконувати додаткові операції, однак потребує високого рівня компетенції та обережності через обмеження Telegram.

Для реалізації програмних рішень, що працюють із великими обсягами медіаданих, важливою складовою є можливість інтеграції Telegram-ботів із зовнішніми інструментами. Оскільки Telegram не забезпечує вбудованої підтримки складної обробки медіафайлів, розробники часто використовують сторонні утиліти. Найбільш поширеними є yt-dlp, ffmpeg та pillow. Інтеграція Telegram-бота з такими інструментами забезпечує змогу виконувати адаптивне перекодування відео, оптимізацію зображень, обрізку, стиснення та інші операції.

З точки зору архітектурних рішень, сучасні бібліотеки для Telegram-ботів підтримують інтеграцію з базами даних, зокрема SQLite, PostgreSQL, MongoDB. Це дає можливість створювати стійкі до навантажень системи, що зберігають історію взаємодій, параметри користувачів, черги завантажень та службову інформацію. Для масштабних рішень також можуть використовуватися брокери повідомлень, такі як Redis або RabbitMQ.

Узагальнюючи, сучасні платформи та бібліотеки для розробки Telegram-ботів забезпечують широкий спектр можливостей, що дозволяють створювати складні програмні рішення. Вибір бібліотеки залежить від конкретних вимог проєкту, необхідного рівня продуктивності та особливостей реалізації алгоритмів. Асинхронні інструменти, такі як aiogram, є найбільш доцільними для систем, що поєднують завантаження та обробку медіафайлів, оскільки забезпечують високу швидкодію, масштабованість та стабільність роботи в умовах інтенсивного навантаження.

1.4 Аналіз існуючих аналогів та обґрунтування вибору теми

Автоматизація завантаження та обробки медіафайлів є поширеним напрямом розробки програмних засобів, тому на ринку існує значна кількість рішень, які частково або повністю реалізують подібний функціонал. Проте більшість таких рішень орієнтовані на вузький спектр завдань або не забезпечують можливості інтеграції з Telegram як універсальним засобом взаємодії з користувачем. Аналіз наявних інструментів дозволяє визначити їх переваги та недоліки, а також обґрунтувати необхідність створення нового програмного комплексу, що охоплює повний цикл роботи з медіафайлами.

Серед найпоширеніших інструментів, що забезпечують завантаження медіаконтенту з різних джерел, можна виділити спеціалізовані утиліти командного рядка. Вони здатні працювати із великою кількістю платформ, але орієнтовані на використання користувачем безпосередньо на комп'ютері. Основною їх перевагою є підтримка широкого спектра протоколів і форматів, однак відсутність засобів для інтеграції з месенджерами обмежує можливість їх використання у мобільному середовищі або у сервісах, що потребують автоматизації взаємодії з користувачами.

Існує також група вебсервісів, які дозволяють завантажувати відео та аудіо без встановлення додаткових програм. Їх основним призначенням є надання швидкого доступу до контенту, однак такі інструменти мають низку обмежень. Серед типових недоліків слід зазначити обмежений вибір форматів, нестабільність роботи, залежність від змін у структурах вебплатформ, відсутність гарантій щодо безпеки даних і відсутність можливості автоматизації процесу. Крім того, подібні сервіси часто не підтримують складної обробки медіафайлів, яка може бути необхідною для адаптації матеріалів до подальшого використання.

Окремим різновидом рішень є Telegram-боти, що забезпечують завантаження медіафайлів із різних платформ. Проте більшість таких ботів реалізує лише базовий функціонал і не підтримує гнучких механізмів обробки, оптимізації або конвертації файлів. Часто такі боти працюють із обмеженою кількістю сервісів або забезпечують лише пересилання вже доступного

контенту без зміни його характеристик. Крім того, робота подібних ботів нерідко залежить від нестабільних або застарілих API, що зменшує їх надійність і масштабованість.

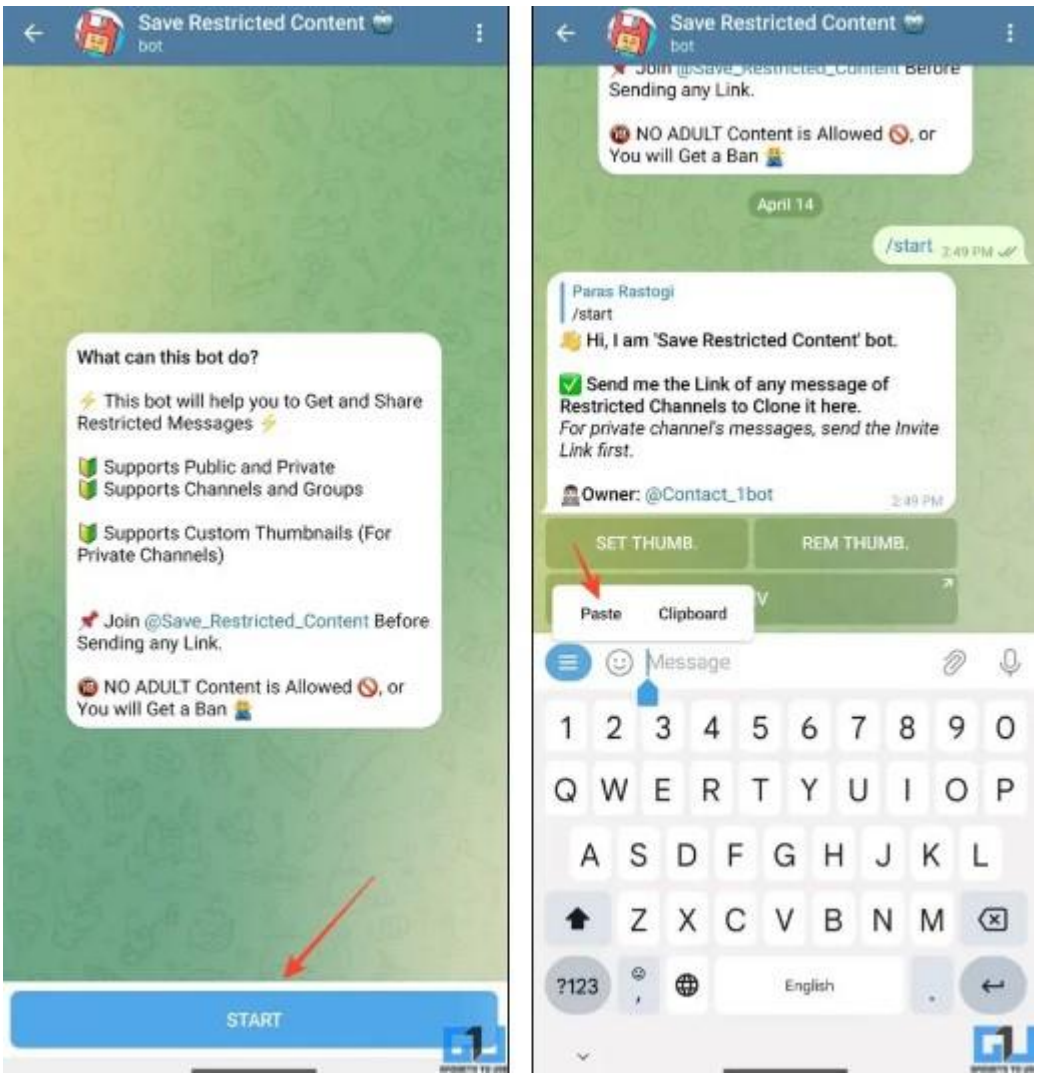

Рисунок 1.4.1 Приклад існуючого боту

Інша група ботів поєднує завантаження з частковою обробкою медіафайлів, однак їх функціональність зазвичай обмежується простими операціями, такими як зміна розміру зображень або конвертація окремих аудіоформатів. Повноцінні інструменти, які включають комплексну обробку, перекодування відео, регулювання параметрів якості та підтримку декількох платформ для завантаження, представлені недостатньо широко. Причиною цього є висока складність реалізації, необхідність оптимізації продуктивності та забезпечення стабільності при інтенсивному навантаженні.

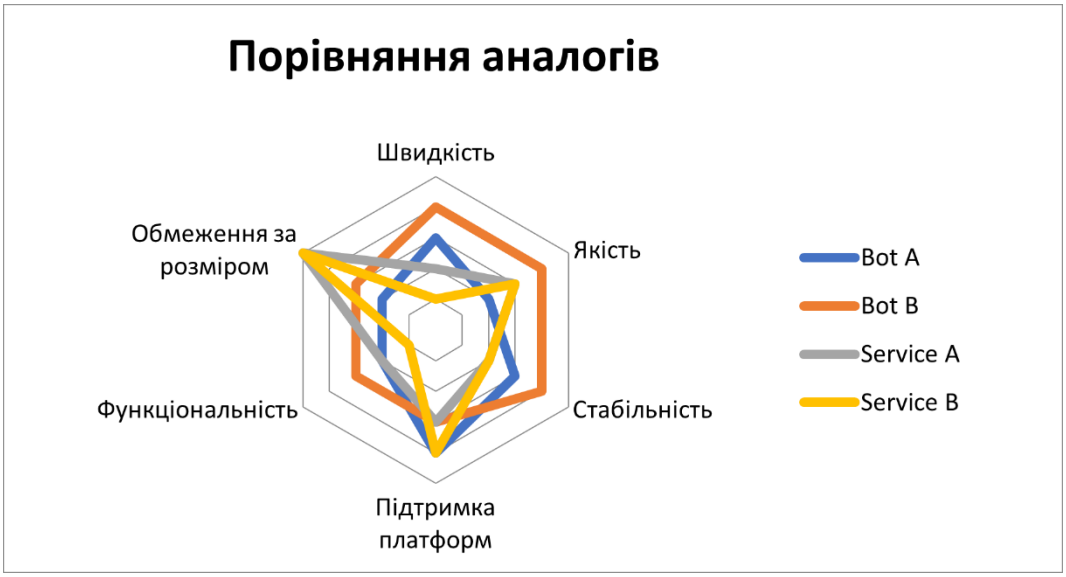

Рисунок. 1.4.2 Порівняння існуючих аналогів

Проведений аналіз дозволяє дійти висновку, що існуючі аналоги не забезпечують повного набору можливостей, необхідних для реалізації сучасної системи автоматизованого завантаження та обробки медіафайлів у середовищі Telegram. Вони не враховують комплексності процесів, пов’язаних із потоковою обробкою, асинхронною взаємодією, адаптивним вибором параметрів оптимізації та інтеграцією з сучасними інструментами обробки медіа.

Вибір теми роботи обумовлений потребою у створенні програмного забезпечення, яке б поєднувало функції автоматизованого отримання медіафайлів із різних джерел, гнучкі алгоритми їх обробки, параметричну оптимізацію та можливість інтеграції з Telegram-ботом як універсальним інтерфейсом взаємодії з користувачем. Розробка такого рішення є актуальною, оскільки дозволяє суттєво підвищити швидкість, зручність і якість роботи з цифровими матеріалами, а також сприяє створенню адаптивних сервісів, здатних ефективно працювати в умовах обмежених ресурсів та високого навантаження.

1.5 Формулювання завдань та основної ідеї роботи

Розвиток цифрових сервісів, зростання обсягів мультимедійного контенту та необхідність його оперативної обробки сформували потребу у створенні програмних систем, здатних автоматизувати процеси отримання та оптимізації медіафайлів. Telegram-боти є зручним інструментом для побудови таких систем завдяки своїй доступності, простоті інтеграції та широким можливостям взаємодії з користувачем. Основна ідея роботи полягає у створенні програмного комплексу, який за допомогою Telegram-бота забезпечуватиме повний цикл роботи з медіафайлами, від їх автоматичного завантаження до обробки та передачі користувачеві в оптимізованому вигляді.

Ця ідея передбачає розробку системи, що поєднує можливості декількох груп технологій. Перша група охоплює методи завантаження контенту з різних платформ, які враховують специфіку доступу, формати та потокове передавання даних. Друга група включає алгоритми та засоби обробки мультимедійних файлів, що дозволяють виконувати операції перекодування, оптимізації, зменшення розміру та адаптації під специфічні вимоги користувача. Третя група передбачає інтеграцію цих можливостей у Telegram-бота з використанням асинхронної моделі виконання, що дозволяє забезпечити високу продуктивність і стабільність роботи.

З урахуванням проведеного аналітичного огляду сформульовано комплекс завдань, необхідних для реалізації основної ідеї роботи. До таких завдань належать аналіз існуючих методів автоматизованого завантаження та обробки медіафайлів; дослідження можливостей Telegram API та інструментів розробки ботів; вибір і обґрунтування математичних моделей та алгоритмів, що застосовуються у процесі обробки даних; формалізація алгоритмів завантаження, перетворення та оптимізації медіафайлів; проєктування структурної, функціональної та інформаційної моделей програмного забезпечення; реалізація Telegram-бота із використанням асинхронних механізмів; проведення дослідження ефективності роботи програмного комплексу; виконання оцінки економічної доцільності створеного рішення.

Реалізація зазначених завдань забезпечує комплексний підхід до проєктування програмної системи, орієнтованої на автоматизацію роботи з мультимедійними файлами. Основна ідея роботи полягає не лише у створенні інструмента, що виконує завантаження файлів, але й у розробці продуктивного та гнучкого механізму обробки, який може адаптуватися до різних форматів, сценаріїв використання та вимог кінцевих користувачів. Таке поєднання функціональних можливостей сприяє підвищенню ефективності роботи з цифровими матеріалами та забезпечує створення сучасного сервісу на основі Telegram-бота.

2 ВІДОМОСТІ ПРО ПРЕДМЕТ (ОБ'ЄКТ) ДОСЛІДЖЕННЯ

2.1 Загальна характеристика Telegram API та принципи взаємодії

Telegram API є основним інструментом взаємодії між зовнішніми програмними засобами та інфраструктурою Telegram. Він забезпечує можливість надсилати та обробляти повідомлення, виконувати команди, працювати з мультимедійними файлами та інтегруватися із зовнішніми сервісами. Telegram надає два ключових інтерфейси: Bot API та MTProto API. У межах даної роботи використовується Bot API, оскільки він призначений спеціально для створення ботів і має стабільний набір функцій, оптимізований для бесід, передачі даних та роботи з користувачами.

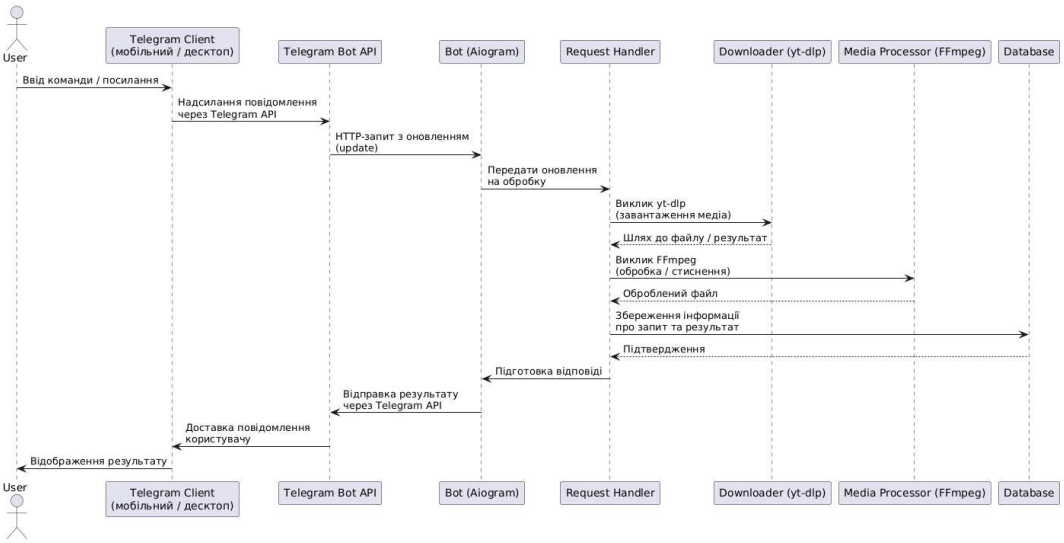

Рисунок 2.1.1 Логіка роботи бота

Telegram Bot API побудований за принципом клієнт-серверної взаємодії. Бот виступає у ролі клієнта, що надсилає запити до серверів Telegram та отримує відповіді у форматі JSON. Комунікація здійснюється через HTTPS, що забезпечує захищений обмін даними. Основою взаємодії є методи Bot API, кожен з яких відповідає за виконання певної операції, наприклад надсилання повідомлення, завантаження файлу, отримання інформації про чат або обробку callback-подій.

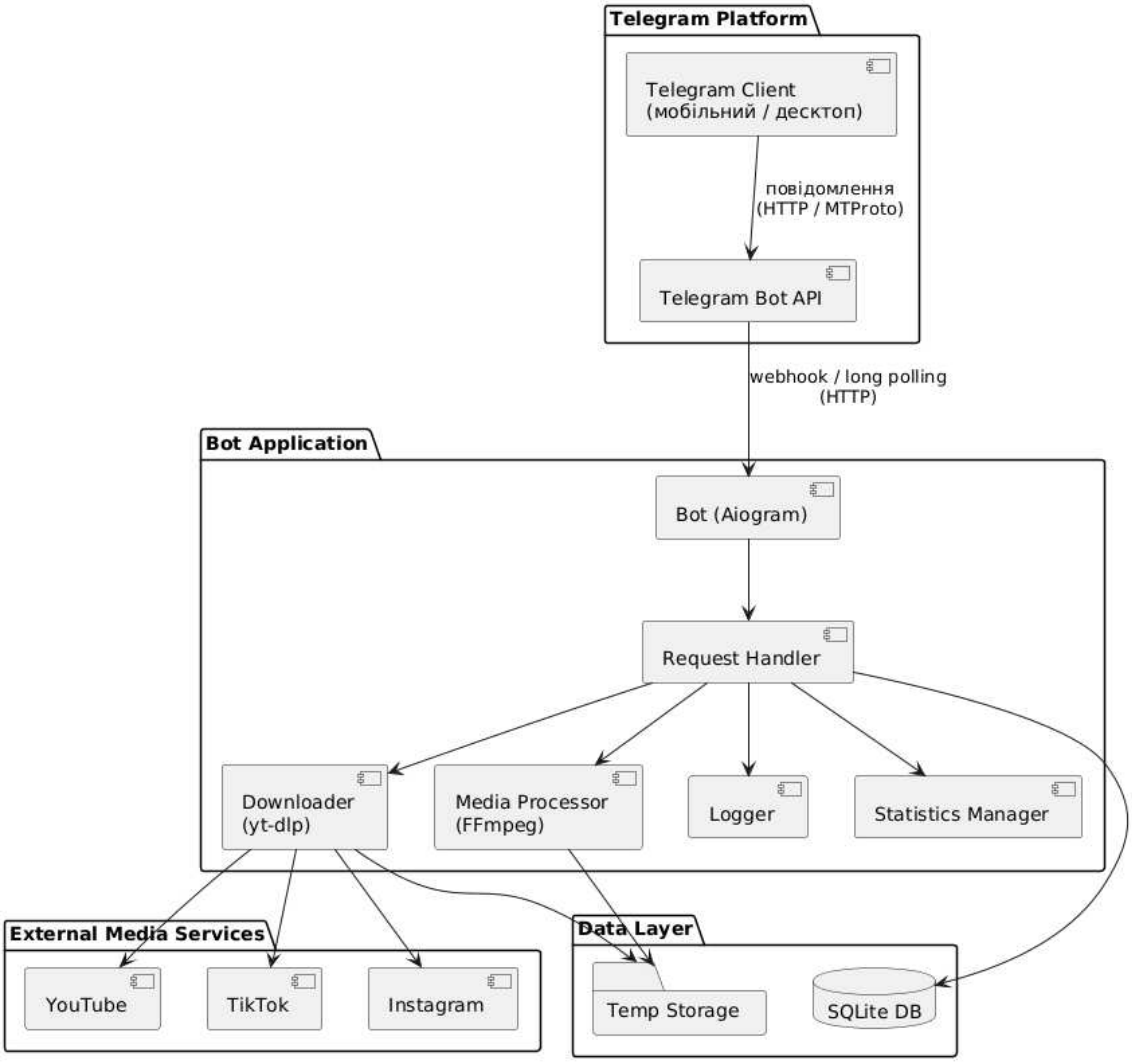

Рисунок 2.1.2 Структура бота

Одним із ключових механізмів отримання оновлень є застосування двох підходів: long polling та webhook. У режимі long polling бот регулярно надсилає запити до серверів Telegram із вимогою передати нові події. Такий підхід є простим у реалізації та не потребує окремої інфраструктури, однак може бути менш ефективним при роботі із системами, що працюють у режимі високого навантаження. Режим webhook передбачає, що сервер Telegram самостійно надсилає оновлення на задану адресу, що забезпечує більш швидку реакцію бота та менші затрати ресурсів. Використання webhook вимагає наявності зовнішнього сервера, який приймає HTTPS-запити.

Telegram Bot API підтримує різні типи повідомлень та мультимедійних об’єктів, зокрема текст, фото, аудіо, відео, документи та інші формати. Обробка медіафайлів передбачає можливість отримання file_id, унікального

ідентифікатора, за допомогою якого можна завантажити файл із серверів Telegram. Це забезпечує гнучкість взаємодії та дозволяє працювати з файлами незалежно від того, яким чином вони були отримані. Telegram також підтримує передачу потокових даних та обробку великих медіафайлів, що є важливим для створення систем з інтенсивним використанням мультимедіа.

Важливою характеристикою Telegram API є підтримка асинхронної взаємодії. Сучасні бібліотеки, що реалізують Bot API, використовують асинхронні механізми обробки подій, що дозволяє оптимізувати роботу із зовнішніми сервісами та забезпечувати швидке реагування на дії користувачів. Завдяки цьому боти можуть працювати з великою кількістю запитів одночасно, обробляти складні сценарії та виконувати ресурсомісткі операції, включаючи завантаження та перекодування медіафайлів.

Telegram API також надає можливості для створення інтерактивних інтерфейсів на основі кнопок, меню, inline-елементів та callback-подій. Це дозволяє будувати складні сценарії взаємодії без потреби аналізу текстового введення. Крім того, Telegram підтримує механізми авторизації користувачів, керування доступом, обробку команд і збирання службових даних, що дає змогу створювати функціональні та масштабовані сервіси.

Загалом Telegram API забезпечує універсальний та гнучкий інструментарій для створення ботів, що працюють із текстовими та мультимедійними даними. Принципи взаємодії API дозволяють створювати системи, орієнтовані на високу продуктивність, стабільність та інтеграцію зі сторонніми сервісами. Це робить Telegram-боти ефективним засобом реалізації автоматизованих систем завантаження та обробки медіафайлів, що є ключовою частиною предмету даного дослідження.

2.2 Обґрунтування вибору математичних методів і алгоритмів обробки медіафайлів

Обробка медіафайлів потребує застосування алгоритмів, які забезпечують оптимальне співвідношення між якістю результату, швидкодією та ефективністю використання ресурсів. Вибір відповідних математичних методів визначається типом даних, вимогами до їх перетворення, а також обмеженнями середовища, у якому працює програмний комплекс. Telegram-боти мають працювати у режимі швидкої реакції, тому обрані алгоритми повинні бути достатньо продуктивними та стабільними, щоб забезпечувати обробку файлів різного формату без значних затримок.

У сфері обробки відео ключову роль відіграють алгоритми компресії та перекодування. Стиснення відеоданих ґрунтується на математичних моделях перетворення та зменшення надлишковості. Найпоширенішими методами є перетворення Фур'є та дискретне косинусне перетворення, які забезпечують обчислювально ефективне представлення відеосигналу у частотній області. Завдяки таким перетворенням можливо зменшити обсяг даних без суттєвої втрати візуальної якості. Алгоритми, що застосовуються у кодеках H.264 та H.265, використовують блокове кодування, прогнозування та квантоване представлення сигналу, що забезпечує високу ефективність стискання. Вибір цих алгоритмів є обґрунтованим, оскільки вони забезпечують універсальність, швидкість роботи та сумісність із більшістю сучасних платформ.

Для аудіоданих використовуються алгоритми спектрального аналізу, що ґрунтуються на методах дискретного перетворення Фур'є та його оптимізованих варіаціях. Аудіокодеки застосовують моделі психоакустики, які дозволяють враховувати властивості сприйняття звуку людиною та видаляти частоти, що не впливають на якість прослуховування. Це дає змогу значно зменшити розмір аудіофайлів при збереженні високої якості. Використання таких алгоритмів у межах Telegram-бота є доцільним, оскільки вони забезпечують швидке конвертування та оптимізацію аудіо при мінімальному навантаженні на ресурси.

У роботі із зображеннями застосовуються алгоритми лінійної та нелінійної фільтрації, зменшення роздільної здатності та стиснення. Поширеними є методи,

що базуються на використанні перетворення Вейвлета або дискретного косинусного перетворення. Вони забезпечують можливість оптимізувати візуальні матеріали шляхом зменшення надлишковості та адаптивного вибору коефіцієнтів фільтрації. Методи інтерполяції (дволінійна та бікубічна) застосовуються для масштабування зображень, забезпечуючи баланс між швидкістю та якістю. Вибір цих методів дозволяє досягнути ефективного стиснення графічних матеріалів, що є важливим для передачі через Telegram із обмеженнями на розмір файлу.

Оскільки програмний комплекс взаємодіє з великими обсягами даних, важливим аспектом є використання алгоритмів оптимізації, що дозволяють зменшити час обробки та забезпечити ефективний розподіл ресурсів. Асинхронні моделі виконання забезпечують можливість обробки кількох медіафайлів одночасно, зменшуючи час очікування користувача. У цьому контексті застосовуються алгоритми планування задач, що базуються на концепції черг, пріоритетів та потокової обробки. Їх використання дозволяє забезпечити стабільність і масштабованість системи, а також адаптивну реакцію на зміну навантаження.

Для побудови математичної моделі процесу обробки медіафайлів важливим є визначення залежностей між обсягом даних, швидкістю обробки та якістю результату. У цьому контексті доцільним є використання моделей, що описують складність алгоритмів у термінах часових і просторових витрат. Наприклад, для операцій перекодування відео складність задається кількістю блоків обробки та параметрами прогнозування, що дозволяє оцінити вплив роздільної здатності та тривалості відео на загальний час роботи системи.

Таким чином, вибір математичних методів і алгоритмів обробки медіафайлів обґрунтований необхідністю забезпечення ефективності, стабільності та швидкодії програмного комплексу. Застосування моделей перетворення сигналів, алгоритмів компресії, методів оптимізації та асинхронної обробки дозволяє створити систему, здатну працювати у режимі реального часу та забезпечувати високу якість результатів при мінімальних витратах ресурсів.

2.3 Розробка математичної моделі процесу завантаження та обробки медіа

Процес автоматизованого завантаження та обробки медіафайлів у Telegram-боті доцільно описувати як послідовність взаємопов’язаних етапів, що включають перевірку вхідних даних, завантаження контенту, його аналіз, обробку та передачу результату користувачеві. Такий підхід дозволяє побудувати математичну модель, яка відображає залежність загального часу виконання запиту від характеристик медіафайлу, параметрів мережевого з’єднання та обчислювальних ресурсів системи.

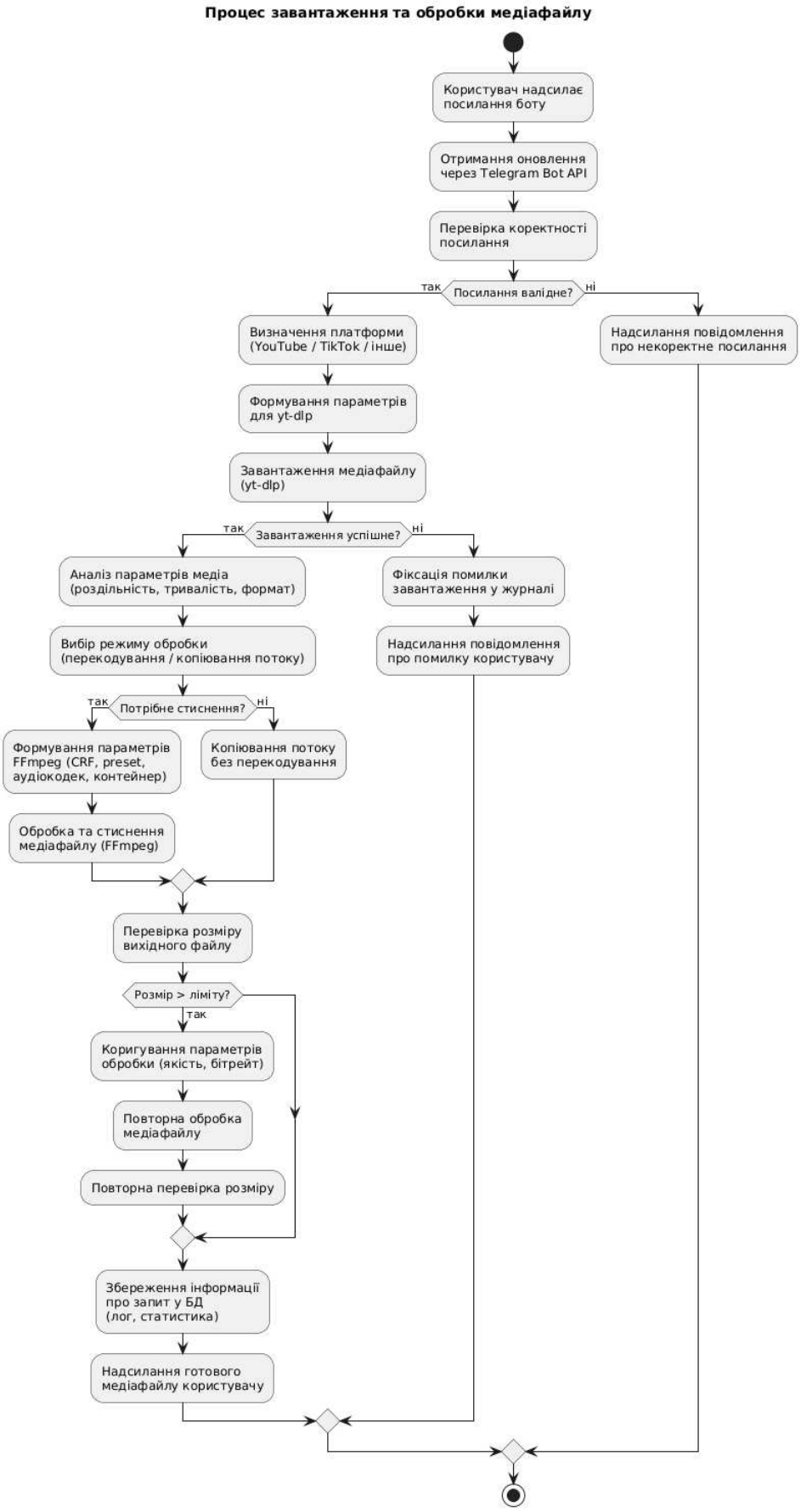

Рисунок 2.3 Процес завантаження та обробки медіафайлу

Початковим етапом є надходження повідомлення з посиланням від користувача та отримання відповідного оновлення через Telegram Bot API. На цьому етапі виконується первинна перевірка коректності посилання. У випадку, якщо посилання не відповідає очікуваному формату або не підтримується системою, процес завершується надсиланням повідомлення про помилку користувачеві. Час цього етапу позначимо як t_0, який, як правило, є малим і майже не залежить від параметрів медіафайлу.

У разі успішної валідації посилання виконується визначення платформи-джерела контенту та формування параметрів для інструмента yt-dlp. Далі запускається етап завантаження медіафайлу, тривалість якого залежить від розміру файлу та ефективної швидкості мережевого передавання. Орієнтовний час завантаження можна описати співвідношенням з урахуванням накладних витрат протоколів та обмежень сторонніх платформ. Якщо завантаження завершується помилкою, система фіксує подію у журналі та інформує користувача про неможливість отримання контенту.

Після успішного завантаження виконується аналіз параметрів медіафайлу, зокрема його роздільної здатності, тривалості та формату. На основі цих характеристик система обирає режим подальшої обробки: пряме копіювання потоку без перекодування або застосування алгоритмів стиснення і перекодування. Якщо перекодування необхідне, формуються параметри обробки для FFmpeg, включно з налаштуваннями CRF, preset, аудіокодека та контейнера.

Час обробки медіафайлу залежить від обсягу даних, обчислювальної продуктивності системи та складності алгоритмів перекодування. Для відеофайлів складність обробки зростає із підвищенням роздільної здатності та бітрейту, що узгоджується з блок-схемою, де передбачено окремий етап аналізу параметрів медіа.

Після завершення обробки виконується перевірка розміру вихідного файлу з урахуванням обмежень Telegram. Якщо розмір перевищує допустимий ліміт, система коригує параметри обробки та повторює етап перекодування. Таким чином, процес може містити цикл повторної обробки, кількість ітерацій

якого залежить від початкових характеристик медіафайлу та заданих параметрів якості. Загальний час обробки в цьому випадку визначається сумою часу всіх ітерацій.

Після отримання файлу, що відповідає обмеженням платформи, виконується збереження інформації про запит у базі даних, включно з журналами та статистичними показниками. Завершальним етапом є передавання готового медіафайлу користувачеві через Telegram. Час передачі результату залежить від розміру файлу та швидкості каналу зв'язку з серверами Telegram.

2.4 Формалізація алгоритмів автоматизації обробки даних

Автоматизація процесів завантаження та обробки медіафайлів потребує формалізованого опису алгоритмів, які визначають послідовність дій, умови виконання операцій та взаємодію між окремими компонентами програмного комплексу. Формалізація дозволяє уніфікувати процеси, усунути неоднозначності та забезпечити можливість подальшої оптимізації системи. У межах даного дослідження алгоритми описуються на основі послідовних і умовних операторів, структур потокової обробки та механізмів асинхронної взаємодії.

Процес починається з формального визначення вхідних даних, якими є запит користувача та посилання на медіафайл. Нехай U позначає множину запитів, а $f(u)$ - функцію, що зіставляє кожному запиту відповідний медіаоб'єкт. Перший етап алгоритму полягає у перевірці коректності вхідних даних. Формально це можна описати умовою: якщо $f(u)$ належить допустимому простору медіаоб'єктів, то запит допускається до обробки, інакше він відхиляється. Такий підхід дозволяє уникнути виконання зайвих операцій при некоректних або неповних запитах.

Наступним етапом є завантаження медіафайлу із зовнішнього джерела. Цей процес визначається функцією завантаження $d(f(u))$, яка повертає бінарні дані або структуру, що містить інформацію про файл. Для врахування умов мережевої взаємодії доцільно формалізувати залежність часу виконання від

параметрів мережі та обсягу даних. У загальному вигляді алгоритм завантаження виконується до моменту успішного отримання всього обсягу медіа або до досягнення граничного часу очікування.

Формалізація етапу обробки медіа передбачає опис функцій перетворення, які застосовуються до отриманих даних. Нехай $g_1, g_2, \ldots, g_n$ позначають окремі операції обробки, такі як змінення формату, стиснення, перекодування або зменшення роздільної здатності. Комплексна обробка описується композицією цих функцій: $G(f(u)) = gn(\ldots g2(g1(f(u))) \ldots)$. При цьому кожна функція gi повинна бути визначена на допустимому просторі медіаданих і повертати результат, придатний для подальшої обробки.

У межах асинхронної моделі виконання процеси завантаження та обробки можуть виконуватися незалежно одне від одного для різних запитів, що дозволяє оптимізувати загальний час виконання. У формальному вигляді це можна описати через використання множини активних задач Z, кожна з яких характеризується власним станом. Для кожного елемента $z \in Z$ визначаються стани $s \in S$, такі як очікування, завантаження, обробка, формування результату та завершення. Перехід між станами виконується відповідно до визначених умов завершення поточного етапу.

Особливе значення має формалізація логіки обробки помилок та виняткових ситуацій. Для цього вводиться множина умов E, що описують можливі відхилення від нормального процесу, такі як недоступність джерела, пошкодження файлу або недостатність ресурсів. Для кожного виду помилки визначається функція обробки $h(e)$, яка повертає рішення про повторну спробу завантаження, припинення обробки чи надсилання повідомлення про помилку. Наявність таких механізмів забезпечує стабільність роботи програмного комплексу.

Фінальним етапом алгоритму є формування результату та його передавання користувачеві. Нехай r позначає результат обробки, що формується як $R = p(G(f(u)))$, де p - функція пакування або підготовки результату (наприклад, створення тимчасового файлу, оптимізація структури даних). Процес

передавання здійснюється до тих пір, поки не буде отримано підтвердження успішної доставки або до настання умов завершення.

Узагальнений алгоритм автоматизації обробки даних може бути поданий у вигляді формальної моделі: $A(u) = send(p(G(d(f(u)))))$ за умови коректності $f(u)$ та успішного виконання всіх етапів. Така модель відображає логічну послідовність дій і дозволяє виконувати аналіз продуктивності, визначати критичні точки та забезпечувати можливість подальшого масштабування програмного комплексу. Формалізація алгоритмів є важливою складовою побудови системи, орієнтованої на стабільну роботу з медіафайлами в середовищі Telegram.

2.5 Розробка структурної та функціональної моделі програмного забезпечення

Проєктування програмного забезпечення для автоматизованого завантаження та обробки медіафайлів із використанням Telegram-бота потребує побудови структурної та функціональної моделей, які відображають основні компоненти системи, їхні взаємозв'язки та послідовність виконання операцій. Такі моделі дозволяють узагальнити архітектуру програмного комплексу, визначити ролі окремих модулів і забезпечити узгодженість між вимогами до системи та її реалізацією.

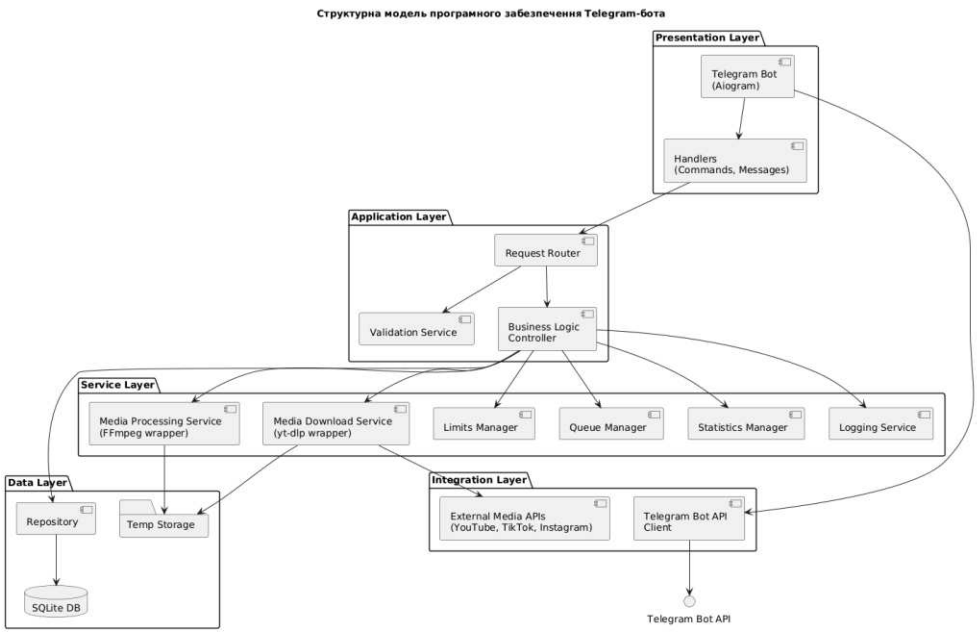

Рисунок 2.5.1 Структура модель програмного забезпечення

Структурна модель програмного забезпечення описує основні компоненти системи та зв’язки між ними. У загальному вигляді програмний комплекс можна подати як сукупність взаємодіючих модулів: модуля взаємодії з Telegram API, модуля обробки запитів, модуля завантаження медіафайлів із зовнішніх джерел, модуля обробки медіаданих, модуля керування чергами та асинхронними задачами, модуля роботи з базою даних, а також модуля формування і відправлення результатів користувачеві. Кожен із цих модулів виконує чітко визначені функції та взаємодіє з іншими через регламентовані інтерфейси.

Рисунок 2.5.1 Функціональна модель програмного забезпечення

Модуль взаємодії з Telegram API відповідає за отримання вхідних оновлень від серверів Telegram, попередню обробку повідомлень, виділення службової інформації та передавання структурованих запитів у внутрішні компоненти системи. Він реалізує логіку обробки команд, callback-подій, текстових повідомлень та медіафайлів, що надходять від користувачів. Цей модуль є зовнішнім інтерфейсом системи, через який здійснюється інтеграція з Telegram.

Модуль обробки запитів виконує функції маршрутизації та прийняття рішень щодо подальших дій. На основі типу запиту, параметрів користувача та контексту взаємодії він визначає, які саме операції повинні бути виконані, формує задачі на завантаження медіафайлів, обробку даних, збереження інформації в базі даних або формування відповіді. Цей модуль забезпечує зв’язок між зовнішньою взаємодією та внутрішньою логікою системи.

Модуль завантаження медіафайлів відповідає за взаємодію з зовнішніми сервісами, зокрема платформами поширення відео та аудіо. Він реалізує алгоритми отримання файлів за наданими посиланнями, керує параметрами завантаження, обробляє помилки доступу та забезпечує повернення результату у внутрішній формат, що використовується іншими модулями. Цей модуль може використовувати сторонні утиліти та бібліотеки, інтегровані через окремий інтерфейс.

Модуль обробки медіаданих реалізує алгоритми перетворення, оптимізації та конвертації медіафайлів. Він виконує операції перекодування відео та аудіо, змінення роздільної здатності, стиснення, обрізки, формування попередніх переглядів тощо. Функціональна модель цього модуля включає налаштування параметрів обробки залежно від вимог користувача або встановлених за замовчуванням профілів. Модуль повинен забезпечувати можливість адаптації алгоритмів до різних форматів та розмірів файлів.

Модуль керування асинхронними задачами відповідає за організацію черг завдань, планування їх виконання та координацію паралельних процесів. У межах функціональної моделі він забезпечує додавання нових задач, контроль станів (очікування, виконання, завершення, помилка), обмеження

кількості одночасно активних операцій та реагування на зміни навантаження. Завдяки цьому модулю програмний комплекс здатен стабільно працювати в умовах великої кількості запитів.

Модуль роботи з базою даних здійснює збереження службової інформації, яка включає історію запитів, параметри користувачів, проміжні результати обробки та інші дані, необхідні для коректної роботи системи. Структурна модель бази даних передбачає наявність таблиць для зберігання інформації про користувачів, завантаження, виконані операції та налаштування. Цей модуль забезпечує цілісність даних, підтримує транзакції та виконує вибірки для аналітики та відновлення контексту.

Модуль формування та відправлення результатів забезпечує підготовку кінцевих файлів або повідомлень до передачі користувачеві через Telegram. У межах функціональної моделі він виконує упаковку результатів, додавання супровідних повідомлень, перевірку обмежень на розмір файлів та, за потреби, поділ великих об’єктів на декілька частин. Після цього модуль взаємодіє з Telegram API для надсилання результату.

Функціональна модель програмного забезпечення описує послідовність виконання операцій у вигляді сценаріїв взаємодії. У типовому сценарії користувач надсилає боту команду або посилання, після чого модуль Telegram API передає запит у модуль обробки запитів. Далі формуються задачі завантаження медіафайлу та його обробки, які передаються у модулі завантаження та обробки відповідно. Після завершення всіх необхідних операцій результат зберігається та надсилається користувачеві. У разі виникнення помилок спрацьовують механізми обробки винятків, що також є частиною функціональної моделі.

2.6 Розробка моделі бази даних Telegram-бота

Модель бази даних Telegram-бота призначена для зберігання службової інформації, яка забезпечує коректну роботу програмного комплексу, відновлення контексту взаємодії з користувачами, ведення історії запитів і результатів обробки медіафайлів. Проєктування моделі бази даних передбачає визначення основних сутностей, атрибутів, зв'язків між ними та обмежень цілісності, що регламентують правила зберігання та використання даних.

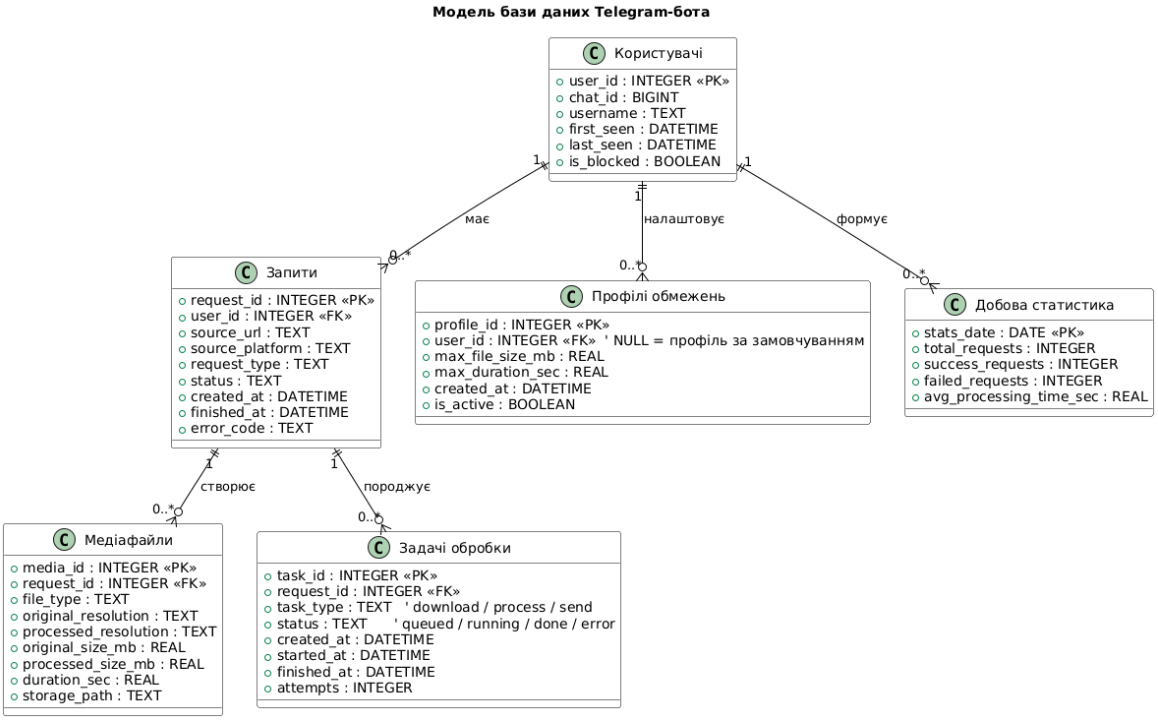

Рисунок 2.6.1 Модель бази данних

Основною сутністю є користувач, який взаємодіє з Telegram-ботом. Для цієї сутності доцільно визначити такі атрибути, як унікальний ідентифікатор користувача в Telegram, ім'я, псевдонім, мова інтерфейсу, дата першої взаємодії та останньої активності. Це дозволяє зберігати базову інформацію про користувачів, адаптувати відповіді бота під їхні налаштування та аналізувати активність у системі. Таблиця користувачів виступає центральною у відношенні до інших сутностей, оскільки більшість операцій пов'язана з конкретними користувачами.

Другим ключовим елементом моделі є сутність, що описує запити користувачів. Для кожного запиту доцільно зберігати посилання на

відповідного користувача, вхідні дані (наприклад, текст команди або посилання), тип запиту, дату і час його отримання, поточний стан обробки та результат виконання. Така інформація дозволяє відстежувати історію взаємодій, виконувати діагностику помилок, а також за потреби повторювати окремі операції. Зв'язок між таблицею користувачів і таблицею запитів є типовим відношенням «один до багатьох», оскільки один користувач може надсилати довільну кількість запитів.

Окрему сутність становлять медіафайли, що обробляються системою. Для них доцільно зберігати унікальний ідентифікатор, тип медіа (відео, аудіо, зображення, документ), вихідне посилання або ідентифікатор сервісу, розмір, формат, а також технічні параметри, такі як роздільна здатність, тривалість, бітрейт. Крім того, зберігаються атрибути, які описують результати обробки: формат вихідного файлу, параметри стиснення, шлях до збереженого результату або його ідентифікатор у Telegram. Зв'язок між таблицею запитів і таблицею медіафайлів також має характер «один до багатьох», оскільки один запит може ініціювати обробку кількох файлів.

Для реалізації механізмів асинхронної обробки доцільно ввести сутність, яка описує задачі або робочі процеси. У таблиці задач зберігаються ідентифікатор задачі, посилання на пов'язаний запит або медіафайл, тип операції (завантаження, перекодування, стиснення, відправлення результату), поточний стан задачі, пріоритет, час постановки до черги та час завершення. Така структура дозволяє керувати чергами обробки, контролювати виконання задач, аналізувати продуктивність системи та своєчасно виявляти проблемні ділянки.

Важливою складовою моделі є таблиці для зберігання журналів подій і помилок. У них фіксуються дата й час події, тип події, пов'язаний користувач або запит, короткий опис та технічні деталі. Це дозволяє виконувати моніторинг роботи системи, аналізувати причини збоїв та оптимізувати алгоритми. Журнали можуть використовуватися для побудови статистичних звітів, оцінки навантаження та планування масштабування.

Для налаштувань системи доцільно передбачити окрему сутність, що містить конфігураційні параметри, такі як граничний розмір файлів, допустимі формати, профілі якості обробки, параметри підключення до зовнішніх сервісів. У таблиці налаштувань можуть зберігатися як глобальні параметри, так і специфічні для окремих користувачів, якщо передбачається персоналізація сервісу.

З точки зору логічної моделі база даних Telegram-бота являє собою набір взаємопов'язаних таблиць, у яких первинні ключі забезпечують унікальність записів, а зовнішні ключі визначають зв'язки між сутностями. Встановлення обмежень цілісності дозволяє уникнути появи «висячих» записів, забезпечити узгодженість між таблицями та зменшити ймовірність помилок під час обробки даних.

Розроблена модель бази даних забезпечує зберігання всієї необхідної службової інформації для роботи Telegram-бота, підтримує відновлення контексту взаємодії, аналіз історії запитів і результатів обробки медіафайлів, а також створює основу для подальшого розширення функціональності програмного комплексу без порушення цілісності даних.

2.7 Модель інтерфейсу та основні класи програмного забезпечення

Модель інтерфейсу Telegram-бота ґрунтується на принципах мінімалізму та інтуїтивності, оскільки взаємодія користувача з ботом здійснюється переважно через текстові команди та повідомлення. Telegram не передбачає складних графічних елементів інтерфейсу, тому логіка взаємодії реалізується за допомогою команд, кнопок, меню та автоматичного розпізнавання посилань. Модель інтерфейсу повинна забезпечувати простий доступ до функціональності, максимальну автоматизацію та мінімальну кількість дій з боку користувача.

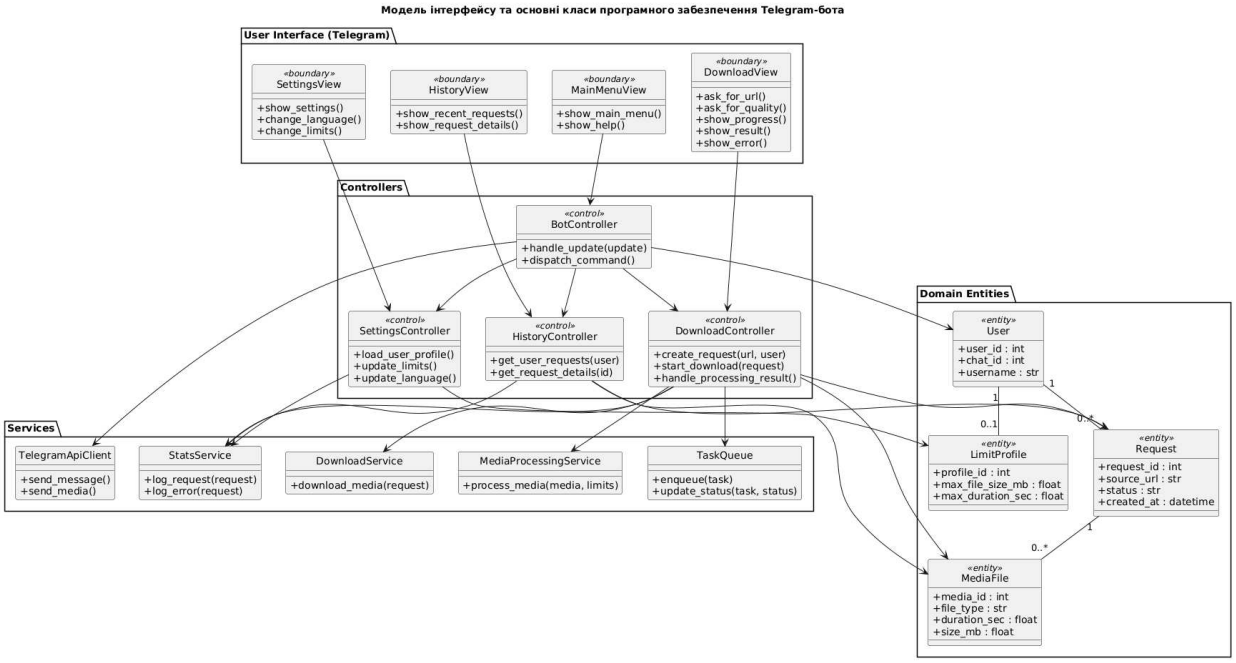

Рисунок 2.7.1 Модель інтерфейсу та основні класи

Базовим елементом моделі інтерфейсу є початкове повідомлення, що надсилається під час виконання команди запуску. Воно містить коротку інструкцію щодо використання сервісу та пояснює формат вхідних даних. Подальша взаємодія будується таким чином, що користувачеві достатньо надіслати посилання на медіаконтент, після чого система автоматично виконує всі необхідні операції. За потреби інтерфейс може містити кнопки для повторної обробки, вибору формату або налаштування параметрів завантаження.

Важливою частиною моделі є механізм обробки помилок, що інформує користувача про неправильно вказане посилання, недоступність ресурсу або інші відхилення від нормального сценарію. Повідомлення про помилки повинні бути стислими та зрозумілими, а система має пропонувати можливість повторної спроби. Такий підхід забезпечує зручність взаємодії та дозволяє користувачеві легко орієнтуватися у процесі завантаження та обробки.

З точки зору внутрішньої логіки програмного забезпечення робота Telegram-бота реалізується через набір класів, кожен з яких відповідає за певний аспект функціональності. Центральним елементом архітектури є клас, який відповідає за взаємодію з Telegram API. Він забезпечує отримання

оновлень, аналіз вхідних повідомлень, виклик відповідних обробників та надсилання результатів користувачеві. Цей клас виступає координатором усіх операцій та забезпечує зв'язок між зовнішньою взаємодією та внутрішніми модулями системи.

Клас обробки запитів виконує початковий аналіз повідомлень, визначає тип введених даних і маршрутизує їх до відповідних модулів. У його обов'язки входить перевірка коректності посилання, ініціалізація завдань на завантаження медіафайлів, взаємодія з базою даних для реєстрації користувачів і запитів. Цей клас виконує роль логічного вузла, який об'єднує роботу різних компонентів програмного комплексу.

Клас, що відповідає за завантаження медіафайлів, реалізує алгоритми отримання даних із зовнішніх джерел. Він виконує мережеві запити, аналізує відповіді, обробляє можливі помилки та формує стандартний формат даних, що передається іншим модулям. Цей клас використовує асинхронні механізми виконання, що дозволяє оптимізувати час очікування та зменшити загальне навантаження на систему.

Клас обробки медіаданих реалізує алгоритми перетворення відео, аудіо та зображень. Він приймає на вхід медіафайл у внутрішньому форматі та застосовує до нього відповідні операції. До функцій цього класу належать перекодування, стиснення, зменшення роздільної здатності, формування альтернативних варіантів файлу та підготовка медіа до надсилання. Структура класу передбачає можливість розширення для додавання нових алгоритмів обробки.

Клас роботи з базою даних забезпечує виконання операцій читання та запису службової інформації. Він відповідає за реєстрацію користувачів, збереження історії запитів, фіксацію параметрів медіафайлів та запис результатів обробки. Клас має бути оптимізований для роботи з локальною базою даних невеликого обсягу та підтримувати транзакції для забезпечення цілісності збережених даних.

Клас керування асинхронними задачами відповідає за формування та виконання черги завдань. Він забезпечує можливість одночасної обробки

декількох запитів, контролює виконання кожного етапу та забезпечує перехід між станами завдань. Структура класу включає механізми контролю помилок, повторних спроб виконання операцій та регулювання кількості активних процесів залежно від завантаженості системи.

Таким чином, модель інтерфейсу Telegram-бота та структура основних класів програмного забезпечення визначають логіку взаємодії користувача із системою та внутрішній механізм обробки медіафайлів. Сукупність класів забезпечує модульність, гнучкість і масштабованість програмного комплексу, що є ключовими вимогами до систем, які працюють із потоковою обробкою даних та взаємодіють із великою кількістю користувачів.

3 ПРИКЛАДНА ЧАСТИНА

3.1 Вибір інструментарію та середовища розробки

Розроблення Telegram-бота для автоматизованого завантаження та обробки медіафайлів потребує використання інструментів, які забезпечують високу продуктивність, підтримку асинхронної обробки даних, а також стабільну взаємодію з різноманітними вебплатформами. Основною мовою програмування для реалізації програмного модуля обрано Python, оскільки вона має широкий набір бібліотек для роботи з мережею та мультимедійними даними, підтримує асинхронну модель виконання та дозволяє швидко створювати масштабовані серверні застосунки. Простота синтаксису і велика кількість готових рішень зробили Python оптимальним вибором для задач завантаження, опрацювання та передавання контенту.

Для взаємодії з Telegram Bot API використано фреймворк Aiogram третьої версії, який базується на модулі asyncio та дозволяє ефективно обробляти велику кількість запитів без блокування основного потоку. Aiogram надає гнучку систему маршрутизації, підтримку сучасних можливостей Telegram, а також дозволяє зручно організувати структуру проекту з поділом логіки за окремими модулями. Завдяки цьому забезпечено високу швидкість реакції бота та можливість подальшого розширення його функціональності.

Для завантаження аудіо- та відеоконтенту з різних вебресурсів застосовано утиліту yt-dlp, яка підтримує велику кількість платформ, стабільно оновлюється та надає уніфікований інтерфейс доступу до медіаданих. Її використання дозволило реалізувати єдиний механізм роботи з різними джерелами, включно з YouTube, TikTok, Instagram, Reddit та іншими сервісами. Крім завантаження, yt-dlp надає можливість отримувати необхідні метадані, що спрощує подальшу класифікацію типу контенту.

Для виконання операцій зі стиснення та конвертації медіафайлів застосовано програмний комплекс FFmpeg. Він забезпечує високоякісне перекодування відео, регулювання параметрів стиснення, оптимізацію аудіопотоків та приведення файлів до форматів, сумісних з Telegram.

Використання FFmpeg у фонових потоках дозволяє не блокувати основний цикл роботи бота та підвищує загальну продуктивність системи.

Розробку програмного забезпечення здійснено у середовищі PyCharm з використанням ізольованого віртуального середовища venv, що дало змогу чітко контролювати версії застосованих бібліотек і гарантувати відтворюваність результатів. Структура проекту побудована за модульним принципом і включає окремі компоненти для обробки команд, взаємодії з Telegram API, завантаження медіа, кодування відео, опрацювання помилок та зберігання статистичних даних. Такий підхід забезпечує зручність подальшої підтримки та можливість розширення функціональності без порушення існуючої архітектури.

3.2 Реалізація основних модулів Telegram-бота

Архітектура Telegram-бота побудована за модульним принципом, що забезпечує чітке розділення відповідальностей між окремими компонентами системи. Такий підхід спрощує підтримку коду, підвищує масштабованість та дозволяє незалежно розвивати окремі частини програмного забезпечення. Основним організаційним рішенням стало відокремлення прикладної логіки, обробників подій, механізмів завантаження медіафайлів та сервісних модулів у спеціалізовані каталоги.

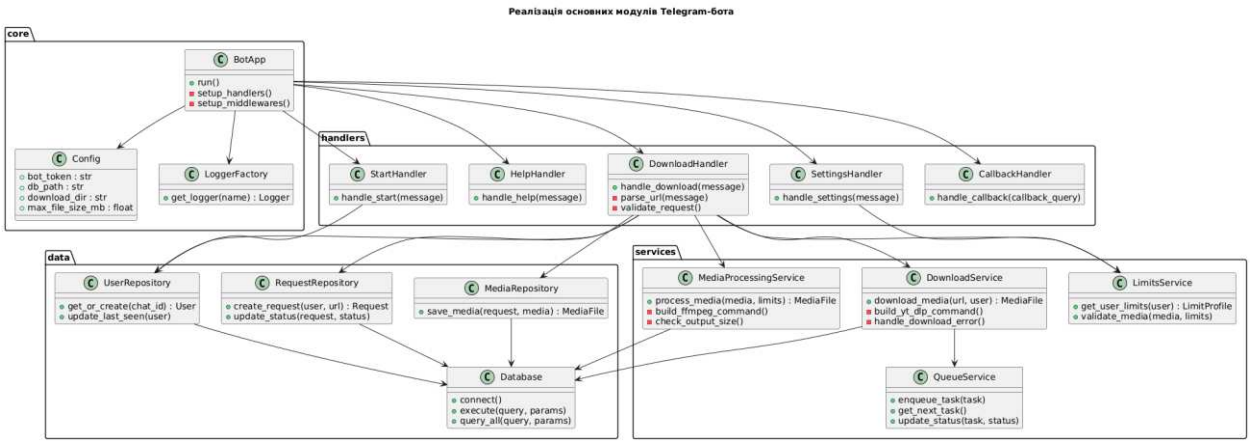

Рисунок 3.2.1. Реалізація основних модулів

Головним елементом системи є файл bot.py, який відповідає за ініціалізацію застосунку, конфігурацію диспетчера, підключення обробників подій, запуск циклу опрацювання оновлень та загальне керування роботою бота. Він містить мінімальну логіку, виконуючи роль точки входу до системи та координуючи взаємодію інших модулів.

Для обробки команд та користувацьких повідомлень використовується каталог handlers, у якому розміщено окремі модулі, що відповідають за різні аспекти взаємодії з користувачем. Модуль common забезпечує опрацювання стандартних команд, таких як start та help, а також формування базових відповідей. Найбільш функціонально насиченим є модуль media, який реалізує механізм приймання посилань, визначення типу контенту, обробку помилок, підготовку проміжних повідомлень та передачу завантажених медіафайлів користувачу. Логіка цього модуля розділена на кілька підфайлів, що відповідають за окремі операції: аналіз URL-адреси, відправлення результатів, обробку медіагалерей та виняткових ситуацій. Така деталізація суттєво спрощує тестування і розширення функціональності.

Сервісні модулі, розміщені у каталозі services, реалізують бізнес-логіку системи. Модуль downloader відповідає за взаємодію з утилітою yt-dlp, отримання метаданих, завантаження файлів і формування спеціалізованих структур даних, необхідних для подальшої обробки. Додаткові файли у цьому каталозі забезпечують логіку декодування та стиснення відео, обробку аудіофайлів, роботу з прямими медіапосиланнями та формування цілісних галерей. Модуль stats реалізує механізм збору статистики використання бота та взаємодіє з базою даних, а модуль db забезпечує створення таблиць, виконання транзакцій та зберігання агрегованих даних.

Конфігураційні параметри згруповано у файлі config.py, що дозволяє зручно централізувати налаштування, такі як токен доступу, граничні розміри файлів, шляхи до директорій збереження та параметри стиснення. Подібна організація полегшує перенесення проєкту на інші середовища та адаптацію до змінених вимог.

3.3 Організація обробки медіафайлів

Обробка медіафайлів у Telegram-боті базується на комбінованому підході, що поєднує аналіз метаданих, визначення типу контенту, оптимізацію розміру та підготовку файлів до передавання у форматах, сумісних з вимогами Telegram. Це забезпечує універсальність системи та можливість працювати з широким спектром платформ, що використовують різні протоколи та моделі зберігання медіа.

Процес починається з визначення типу контенту. Якщо посилання містить пряме посилання на файл із розширенням mp4, jpg, png, mp3 або іншими стандартними медіаформатами, бот обробляє його без використання сторонніх інструментів. У таких випадках застосовується модуль прямого завантаження, який виконує HTTP-запит і зберігає файл у робочу директорію. Якщо ж посилання належить до платформ, що не надають прямого доступу до файлів, воно передається до модуля, який працює через утиліту yt-dlp. Цей модуль отримує метадані, ідентифікує формат, завантажує файл або елементи галереї й уніфікує отриманий результат для подальшої роботи.

У випадку з відеофайлами використовується механізм додаткової оптимізації. Telegram має обмеження на максимальний розмір файлів, тому відео перед надсиланням аналізується за кількома критеріями: фактичний розмір, кодек, бітрейт та роздільність. Якщо розмір перевищує допустимі межі або потенційно може не вміститися у ліміт, викликається модуль обробки відео, який застосовує FFmpeg для зменшення роздільності (до 720p або 480p), корекції аудіопотоку та налаштування параметрів стиснення. Обрані значення CRF, preset і бітрейту забезпечують баланс між якістю та швидкістю перекодування. У разі якщо відео і так має невеликий розмір, воно надсилається без додаткового перекодування.

Обробка аудіофайлів передбачає використання різних методів залежно від джерела контенту. Якщо платформа надає готовий аудіопотік, він завантажується без змін, але у випадках, коли файл має нестандартний формат або низьку якість метаданих, застосовується модуль покращення аудіо. FFmpeg використовується для конвертації у формат mp3 з уніфікованим

бітрейтом, вирівнюванням гучності та формуванням коректних тегів. Такий підхід забезпечує однакову якість аудіофайлів незалежно від того, з якої платформи вони отримані.

Особливе місце в системі займає обробка зображень та галерей. Для одиночних зображень застосовується пряме завантаження, що гарантує максимально швидку обробку. У випадку галерей, які можуть складатися з фотографій або поєднання зображень і відео, використовується алгоритм послідовного отримання кожного елемента. Інформація про галерею визначається за структурами yt-dlp або метаданими платформи. Кожен елемент зберігається як окремий файл і пізніше надсилається користувачу у вигляді медіагрупи, що відповідає можливостям Telegram.

Усі процеси супроводжуються розширеною системою обробки помилок. Відповідні модулі розрізняють ситуації, пов'язані з обмеженнями доступу, необхідністю авторизації, перевищенням частотних лімітів, а також неочікуваними змінами у структурі даних сторонніх платформ. Це дає змогу забезпечити стабільність роботи бота навіть у випадках, коли умови завантаження змінюються або платформа тимчасово обмежує доступ до своїх ресурсів.

Завдяки поєднанню модулів аналізу, завантаження, перекодування та адаптивної обробки контенту система забезпечує коректну роботу з різними типами медіа та дозволяє передавати їх користувачам у форматах, які відповідають технічним вимогам Telegram.

3.4 Структура бази даних і механізми зберігання інформації

У програмному комплексі Telegram-бота для зберігання службової інформації використовується реляційна система керування базами даних SQLite. Вибір саме цієї СКБД обумовлений її простотою інтеграції, відсутністю необхідності розгортання окремого серверного середовища, високою стабільністю роботи та достатньою продуктивністю для застосунків, що працюють у режимі асинхронної обробки запитів. SQLite забезпечує

збереження даних між сеансами роботи бота та дозволяє спростити процес встановлення та експлуатації програмного забезпечення.

Структура бази даних Telegram-бота побудована за принципами нормалізації та включає шість основних таблиць: users, requests, media_files, tasks, limit_profiles та daily_stats. Кожна з них відповідає за зберігання певної частини інформації та має чітко визначені зв'язки з іншими сутностями

- Таблиця users містить інформацію про користувачів бота:
  - ідентифікатор;
  - chat_id;
  - ім'я;
  - мову інтерфейсу;
  - часові мітки активності.
- Таблиця requests зберігає історію запитів користувачів, включаючи:
  - URL;
  - платформу походження медіафайлу;
  - статус обробки;
  - можливі коди помилок.
- Таблиця media_files містить метадані щодо завантажених файлів:
  - тип;
  - роздільність;
  - тривалість;
  - розмір до та після обробки;
  - шлях до збереженого файлу у файловій системі.
- Таблиця tasks використовується для відстеження проміжних операцій:
  - завантаження;
  - обробки та відправлення медіафайлів.

Це дозволяє боту керувати чергою задач, фіксувати їхній стан та повторювати помилкові операції.

- Таблиця limit_profiles містить:
  - налаштування для обмеження максимального розміру;

- налаштування тривалості медіафайлів для кожного користувача або профілю за замовчуванням

- Таблиця daily_stats агрегує добову статистику роботи:
  - загальну кількість запитів;
  - кількість успішних та помилкових операцій;
  - середній час обробки;
  - сумарні обсяги завантажених і оброблених даних.

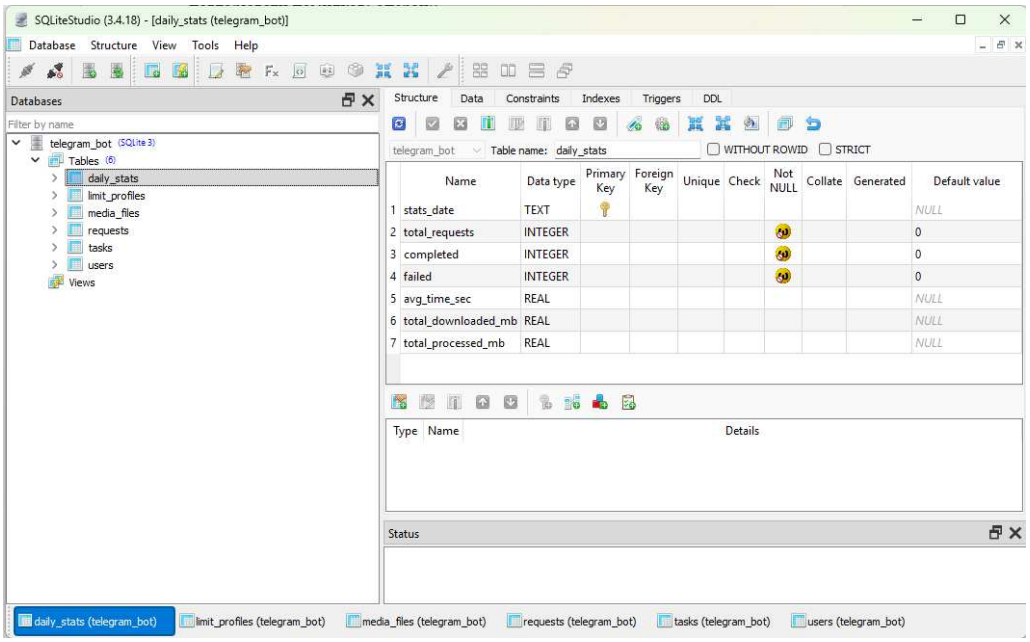

Рисунок 3.4.1 Структура бази даних

Механізми зберігання інформації реалізовано за допомогою спеціалізованого модуля доступу до БД, який інкапсулює всі операції читання та запису. Він забезпечує централізоване виконання SQL-запитів, підтримку зовнішніх ключів, обробку транзакцій та коректну роботу з базою за умов паралельної обробки запитів. Такий підхід дозволяє ізолювати логіку роботи з даними від прикладної логіки бота, підвищуючи модульність та розширюваність системи.

Проєктна структура бази даних передбачає можливість додавання нових таблиць без необхідності модифікації існуючих. Це дозволяє розширювати функціональність програмного комплексу, зокрема додавати історію зміни налаштувань, аналітичні звіти або інформацію про групові чати у Telegram.

3.5 Інтерфейс користувача та керівництво користувача

Інтерфейс користувача Telegram-бота побудований на основі стандартної моделі взаємодії, притаманної платформі Telegram. Оскільки бот не використовує окремих графічних елементів або складних навігаційних меню, основним засобом взаємодії виступають текстові повідомлення, кнопки швидкого доступу та інлайн-елементи керування. Такий підхід забезпечує простоту використання та мінімізує навчальне навантаження на користувача.

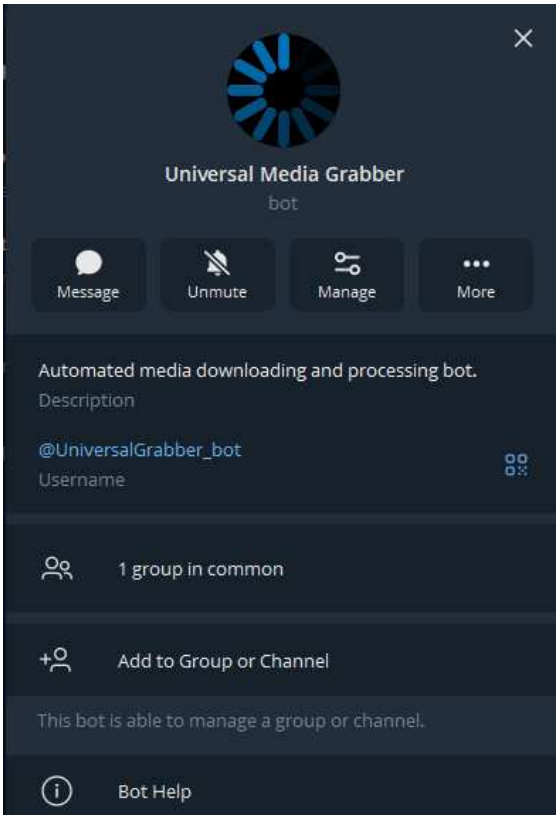

Рисунок 3.5.1 Опис боту

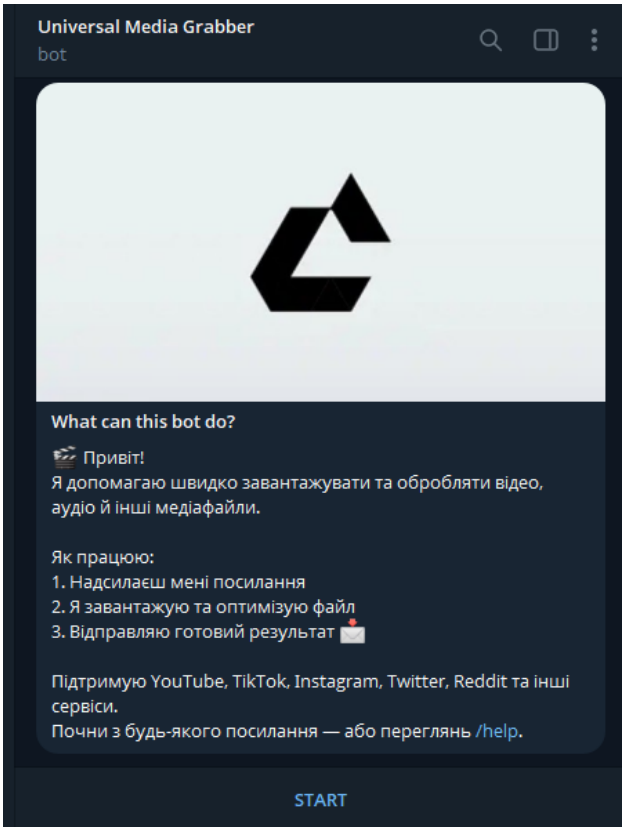

Рисунок 3.5.2 Вітальне повідомлення

Після активації команди /start бот надсилає привітальне повідомлення з коротким описом основної функціональності та поясненням принципів роботи. У цьому повідомленні користувач отримує інструкцію щодо надсилання посилань на відео-, аудіо- та інші типи медіаконтенту, а також можливість додати бота до групового чату за допомогою спеціальної кнопки. Привітальний блок виконує роль введення в систему та містить усю необхідну інформацію для початку роботи.

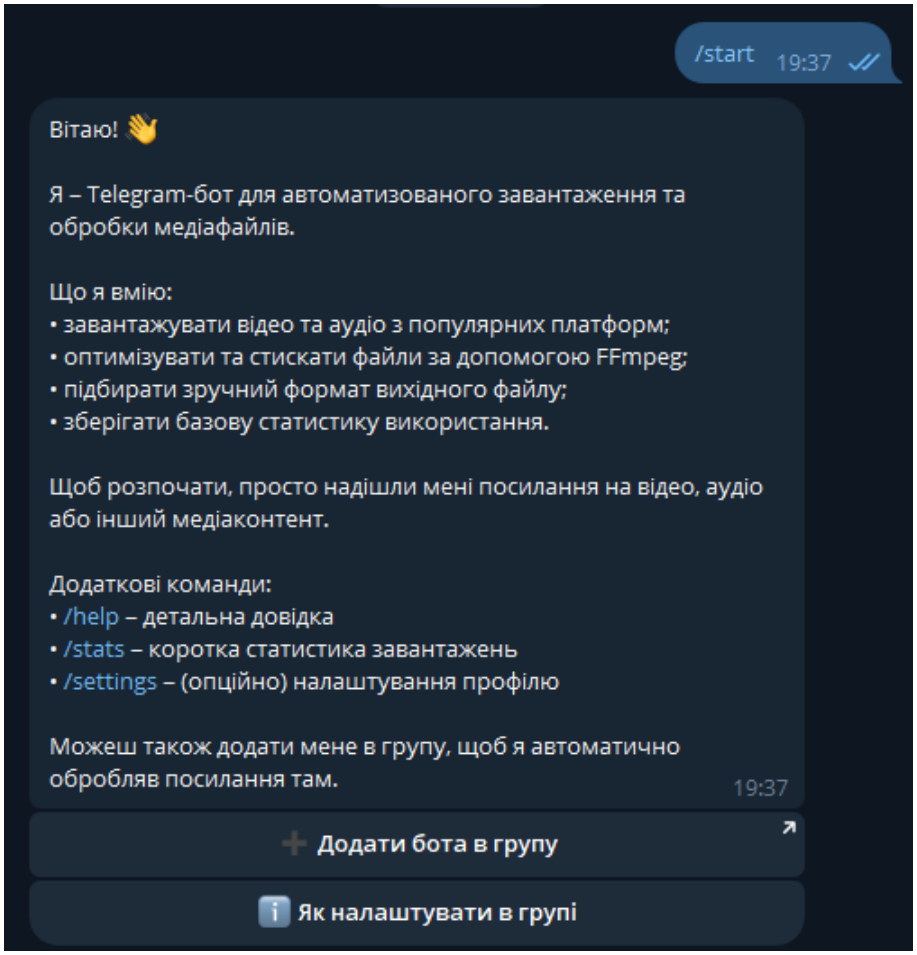

Рисунок 3.5.3 Команда старту

Команда /help надає розширену довідкову інформацію про можливості системи, підтримувані платформи та доступні команди. У повідомленні також міститься структурований опис режимів роботи бота в особистих чатах і групах, що дозволяє користувачу швидко зорієнтуватися у функціоналі без потреби звертатися до зовнішніх джерел. Такий формат подання інформації робить інтерфейс самодостатнім і зрозумілим навіть для нових користувачів.

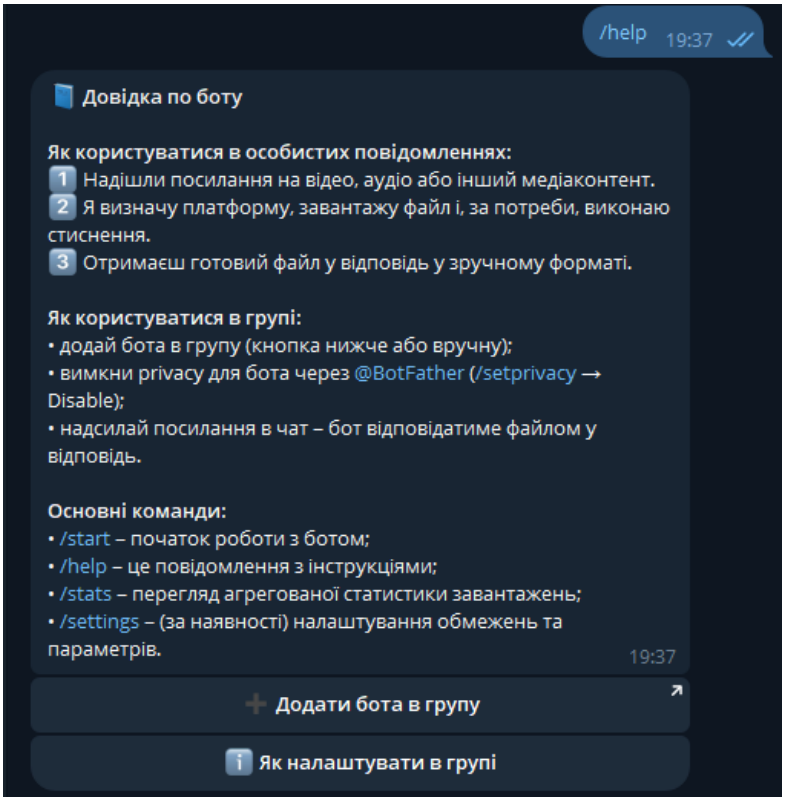

Рисунок 3.5.4 Команда допомоги

У процесі обробки запиту бот надсилає проміжні службові повідомлення, які відображають стан виконання операції: прийняття посилання, визначення платформи, завантаження, стиснення або конвертація медіафайлу. Наявність статусних повідомлень підвищує прозорість процесу та створює відчуття керованості системою. Після завершення операції користувач отримує підсумкове повідомлення з готовим медіафайлом та короткими метаданими про результат обробки.

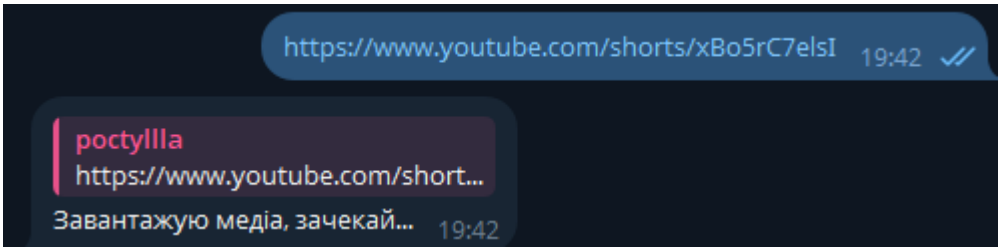

Рисунок 3.5.5 Приклад роботи боту

Команда /stats дозволяє переглянути агреговану статистику роботи бота з моменту його запуску. У відповідь користувачу надсилається структурований

звіт, що містить кількість опрацьованих запитів та підсумкові значення за типами медіафайлів і джерелами. Це реалізовано у вигляді компактного текстового блоку, який зручно інтерпретувати навіть у мобільному інтерфейсі.

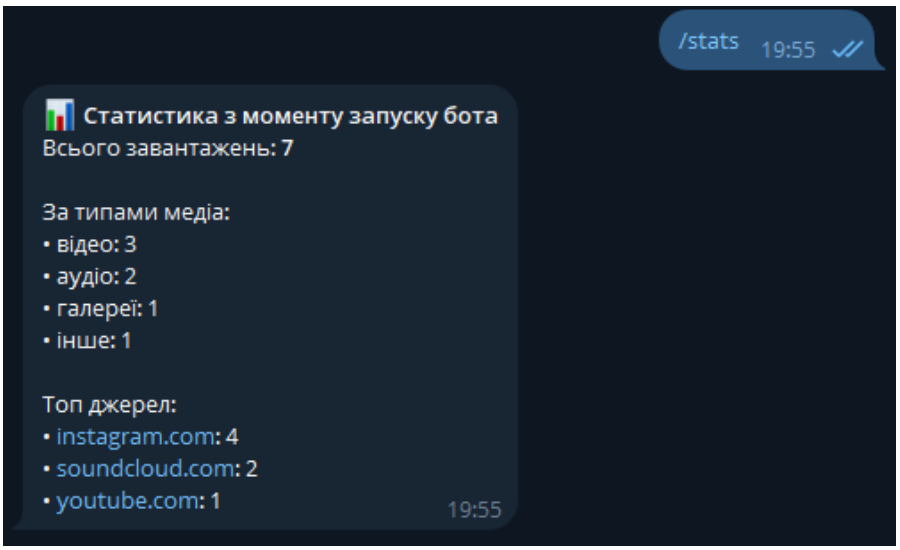

Рисунок 3.5.6 Команда статистики

Окрім основних команд та інформаційних повідомлень, інтерфейс користувача Telegram-бота передбачає низку механізмів, спрямованих на підвищення зручності взаємодії й адаптацію системи до різних сценаріїв використання. Однією з ключових особливостей є автоматичне визначення характеру запиту без необхідності використання спеціальних команд. Якщо користувач надсилає посилання у приватному чаті або групі, бот самостійно класифікує його як запит на завантаження та запускає відповідний алгоритм. Завдяки цьому інтерфейс залишається мінімалістичним, а взаємодія - максимально природною для користувача.

Ще одним важливим елементом інтерфейсу є контекстна поведінка бота щодо різних типів медіафайлів. Залежно від формату, розміру або технічних параметрів медіа, система автоматично обирає форму подання результату, що забезпечує коректне відображення в Telegram. Наприклад, зображення можуть надсилатися або як стиснені медіафайли для швидкого перегляду, або як документи у разі необхідності збереження оригінальної якості. Аналогічно відеофайли подаються у форматі, який оптимально балансує між якістю та

обмеженнями платформи. Це дозволяє користувачу отримувати результат без додаткових налаштувань або втручання.

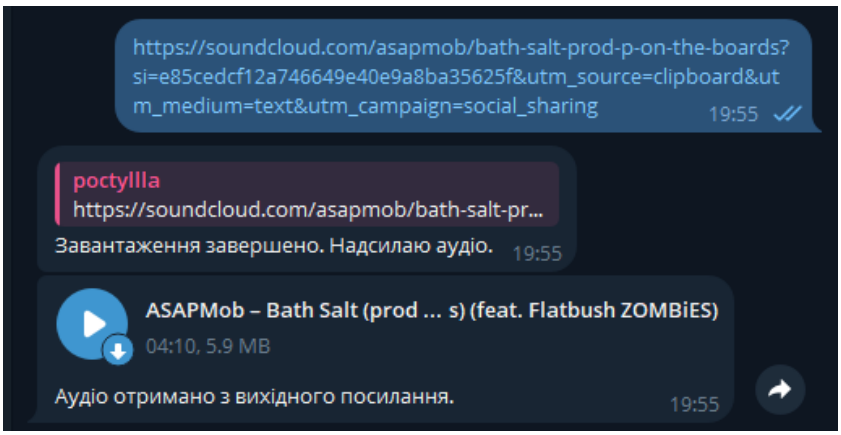

Рисунок 3.5.7 Приклади роботи боту

Також у межах інтерфейсу реалізовано систему адаптивних відповідей, яка забезпечує надання актуальної інформації залежно від контексту запиту. Якщо під час обробки виникає затримка, користувач отримує короткі службові повідомлення, які інформують про поточний етап виконання. У випадку помилки бот формує відповідь, що містить пояснення можливих причин і загальні рекомендації. Такий механізм підвищує прозорість роботи системи та знижує ймовірність неправильного трактування поведінки бота.

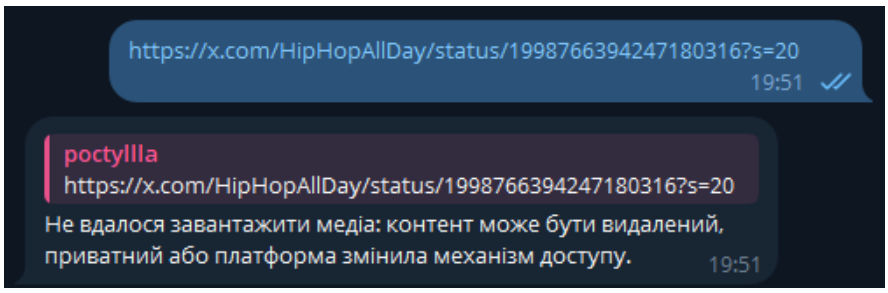

Рисунок 3.5.7 Приклад помилки роботи боту

Інтерфейс адаптований таким чином, що бот мінімізує свою присутність у чаті та відповідає виключно на ті повідомлення, які містять медіапосилання.

4 ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ І АНАЛІЗ ЕФЕКТИВНОСТІ

4.1 Методика проведення експериментів

Експериментальні дослідження були спрямовані на комплексну оцінку працездатності Telegram-бота, який реалізує автоматизоване завантаження та обробку медіафайлів із використанням інструментів yt-dlp та FFmpeg. Методика передбачала виконання серії тестових сценаріїв, що охоплювали як типові, так і граничні випадки взаємодії користувача із системою. Особливу увагу приділено коректності визначення параметрів медіа, стабільності роботи алгоритмів, відповідності отриманих результатів вимогам Telegram та стійкості до можливих помилок.

Для проведення експериментів було сформовано набір тестових посилань, які охоплюють основні платформи: YouTube, TikTok, Instagram, X (Twitter), Reddit, SoundCloud та низку інших ресурсів, що містять відео-, аудіо- чи графічний контент. Для кожного джерела оцінювалися такі параметри:

- час завантаження;
- тривалість обробки у FFmpeg;
- коректність визначення формату й типу медіа;
- стабільність роботи алгоритмів;
- поведінка системи в разі помилок;
- відповідність формату вихідного файлу вимогам Telegram (обмеження на розмір, тривалість, тип контейнера).

Тестова вибірка включала різнорідні за структурою медіафайли: короткі ролики, відео високої роздільності (1080p та 1440p), довгі відеоматеріали, аудіофайли з різною бітрейтовою характеристикою, а також зображення та галереї. Це дозволило оцінити можливості системи щодо роботи зі складними та нестандартними медіаданими.

Замірювання часу виконання операцій здійснювалося за допомогою вбудованих журналів подій, де фіксувалися часові мітки таких етапів: початок

завантаження медіа, завершення отримання файлу, ініціалізація обробки у FFmpeg, формування готового результату та момент його надсилання користувачеві. Отримані дані дали можливість побудувати хронологію обробки та визначити етапи, які є найбільш ресурсоємними.

Для тестування було сформовано набір посилань, що охоплює основні платформи, з яких користувачі зазвичай отримують відео-, аудіо- та графічний контент. Для кожної платформи оцінювалися характерні особливості структури даних, можливі обмеження доступу, різновиди форматів і стабільність роботи алгоритмів під час завантаження.

Платформа	Типи контенту	Особливості структури URL	Типові обмеження
YouTube	відео, аудіо	довгі URL, різні кодеки, адаптивні потоки	обмеження регіонів, захищений контент
TikTok	короткі відео	короткі URL, інколи змінні редиректи	нестабільний CDN
Instagram	відео, фото, Reels	приватність акаунтів	блокування API
X (Twitter)	відео	багаторівневі медіапотоки	тимчасові ліміти
Reddit	відео, галереї	json-структури, кілька форматів	можливі заборони на кросдоменний доступ

Рисунок 4.1.1 Платформи, які тестувались

Визначення характеристик платформ дозволило сформувати збалансований набір тестових медіафайлів, що відрізнялися тривалістю, якістю, параметрами аудіодоріжок та структурою контейнера. Це забезпечило можливість оцінити роботу як базових алгоритмів, так і механізмів обробки складних випадків, наприклад, нестандартних бітрейтів, високої роздільності або великого розміру.

№	Довжина	Роздільність	Бітрейт	Формат	Призначення тесту
1	15 с	720p	середній	MP4	базовий сценарій
2	60 с	1080p	високий	MP4	навантажувальний тест
3	3 хв	480p	низький	WebM	перевірка перекодування
4	10 хв	1080p	високий	MP4	тест граничних розмірів
5	20 с	нестандартна	змінний	MOV	перевірка декодування

Рисунок 4.1.2 Приклад медіафайлів, які тестувались

Важливо не лише зафіксувати результати, а й простежити часову структуру виконання кожного етапу обробки. Для цього було використано систему внутрішнього логування, яка фіксувала часові мітки ключових операцій: завантаження, декодування, обробки FFmpeg, формування файлу та його відправлення. Це дозволило ідентифікувати найповільніші етапи та оцінити їхню залежність від параметрів медіа.

Етап операції	Часова мітка	Опис
StartDownload	t1	ініціалізація завантаження
EndDownload	t2	файл повністю отримано
StartFFmpeg	t3	запуск обробки FFmpeg
EndFFmpeg	t4	завершення перекодування
SentToUser	t5	надсилання результату

Рисунок 4.1.3 Тестування основних етапів обробки

Дослідження часових параметрів дозволило обчислити середній час обробки для різних форматів, визначити вплив роздільності та бітрейту на швидкість роботи алгоритмів і порівняти ефективність різних конфігурацій параметрів FFmpeg. Крім того, аналіз журналів допоміг виявити потенційні точки оптимізації, пов’язані зі швидкістю зчитування потоків або продуктивністю системи зберігання.

Окрема частина експериментальної методики була присвячена перевірці поведінки системи в разі виникнення помилок. Аналізувалися такі сценарії, як недоступність вихідного ресурсу, обмеження доступу, помилки декодування, перевищення лімітів Telegram, а також збої yt-dlp під час отримання медіаданих. Дослідження виняткових ситуацій дозволило оцінити стійкість системи та ефективність реалізованих механізмів обробки помилок.

4.2 Аналіз ефективності алгоритмів завантаження та обробки

Оцінювання ефективності алгоритмів завантаження та обробки медіафайлів здійснювалося на основі експериментальних даних, отриманих під час тестування відео-, аудіо- та графічних файлів різних розмірів і характеристик. Аналіз охоплює часові показники роботи основних компонентів системи, стабільність виконання операцій, залежність швидкодії від параметрів вхідного контенту, а також відповідність результатів технічним обмеженням Telegram.

Для кількісної оцінки було використано такі базові параметри. Час завантаження медіафайлу визначався як різниця між моментом початку завантаження та моментом повного отримання даних:

$$T_{\text{download}} = t_2 - t_1,$$

де t_1– ініціалізація запиту, t_2– завершення завантаження.

Час обробки у FFmpeg визначався за формулою:

$$T_{\text{process}} = t_4 - t_3,$$

де t_3– початок перекодування, t_4– завершення операцій.

Загальний час виконання запиту описується залежністю:

$$T_{\text{total}} = t_5 - t_1,$$

де t_5– момент відправлення результату користувачу.

Ефективність стиснення визначалась як відношення початкового розміру файла до кінцевого:

$$E_{\text{compression}} = \frac{S_{\text{before}}}{S_{\text{after}}}.$$

Залежність часу завантаження від розміру файлу показала монотонне зростання, що добре узгоджується з пропускною здатністю джерел медіаконтенту. Для файлів у межах 5-80 МБ зафіксовано такі середні значення: для 5 МБ приблизно 0.8 с, для 20 МБ - 1.8 с, для 50 МБ - 3.4 с, для 80 МБ - 5.1 с. Затримки, пов’язані з мережевими умовами, були незначними, а різниця між платформами (YouTube, TikTok, Instagram) проявлялась переважно у вигляді невеликих коливань через особливості CDN.

Обробка відеофайлів у FFmpeg виявила залежність часу виконання від роздільності та бітрейту контенту. Для типових роликів використані параметри стиснення забезпечили такі середні значення: для 480p - 0.7-1.5 с, для 720p - 1.2-3.0 с, для 1080p - 2.5-6.0 с. Зростання тривалості обробки відповідає збільшенню обсягу даних, що передаються в процес декодування та повторного кодування. Усі отримані результати відповідали технічним обмеженням Telegram, зокрема ліміту на максимальний розмір вкладення.

Стабільність виконання операцій оцінювалася за варіаціями значень T_{download}, T_{process} та T_{total}. У більшості експериментів відхилення не перевищували 5-12 %, що пояснюється стабільністю пропускної здатності мережі та повторюваною поведінкою сторонніх платформ. Механізми повторного завантаження активувалися лише для окремих запитів, пов’язаних із недоступністю ресурсу або тимчасовими обмеженнями з боку платформ.

Загальні результати дослідження свідчать про те, що алгоритми завантаження та обробки медіафайлів працюють передбачувано та ефективно в межах типових сценаріїв використання. Система забезпечує достатню швидкодію як з точки зору користувача, так і з точки зору внутрішньої продуктивності, а обмеження Telegram враховуються на етапі формування кінцевого результату.

№ тесту	Розмір файлу, МБ	Час завантаження, с
1	5	0.8
2	15	1.5
3	30	2.6
4	45	3.9
5	45	4.1
Середнє значення	–	**2.58 с**

Рисунок 4.2.1 Результати тестування

№ тесту	Роздільність	Час обробки, с
1	480p	1.0
2	720p	2.1
3	1080p	4.1
4	720p	2.3
5	1080p	4.5
Середнє значення	–	**2.8 с**

Рисунок 4.2.2 Результати тестування

Оновлені результати показують характерну лінійну залежність між розміром медіафайлу та часом його завантаження в рамках обмеження Telegram у 50 МБ. Для файлів обсягом 5-15 МБ час завантаження становив менш ніж дві секунди, що забезпечує комфортну взаємодію з ботом.

Файли більшого розміру (30-45 МБ) демонстрували середній час завантаження в діапазоні 2.6-4.1 с, що є очікуваним для більшості платформ і відповідає пропускній здатності їхніх CDN. Середнє значення 2.58 с підтверджує стабільність алгоритму отримання медіафайлів та його придатність для роботи в межах технічних обмежень Telegram.

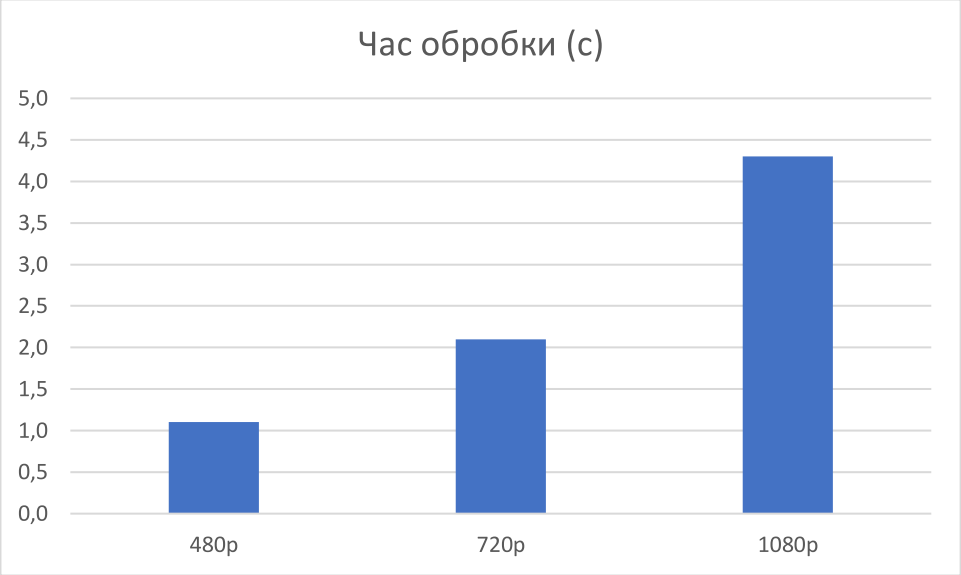

Рисунок 4.2.3 Результати тестування

4.3 Порівняння з існуючими рішеннями

Для оцінювання ефективності та конкурентоспроможності розробленого Telegram-бота було виконано порівняння з наявними вебсервісами та іншими Telegram-ботами, що забезпечують завантаження та обробку медіафайлів. Аналіз охоплював як функціональні характеристики, так і технічні показники роботи систем, включно зі швидкістю завантаження, стабільністю виконання операцій, якістю отриманих файлів та відповідністю обмеженням платформи Telegram.

Під час порівняння враховувалися такі критерії: підтримка платформ, максимальний розмір файлів, наявність оптимізації відео, підтримка галерей, спосіб взаємодії з користувачем, стабільність роботи та вимоги до додаткових дій (перехід на вебсайт, реклама, ручне завантаження результатів). Для об’єктивності аналізу тестування проводилось із використанням однакових посилань та ідентичних умов мережевого доступу.

Критерій	Розроблений бот	Інші Telegram-боти	Вебсервіси
Підтримка платформ	>15 платформ, включно з YouTube, TikTok, Instagram, X, Reddit	5–8 платформ	залежить від сервісу, часто обмежено
Максимальний розмір файлу	до 50 МБ (обмеження Telegram для ботів) + автоматична оптимізація	до 50 МБ, часто без оптимізації → помилки	залежить від браузера, може бути >100 МБ
Оптимізація відео (FFmpeg)	є, автоматична	рідко зустрічається	зазвичай відсутня
Підтримка галерей	є	переважно відсутня	переважно відсутня
Швидкість завантаження	стабільна, 1–4 с	від 2 до 6 с	сильно залежить від сервера, іноді повільно
Стабільність роботи	висока (асинхронна архітектура)	залежить від реалізації	проблеми з рекламними блоками, нестабільністю CDN
Зручність використання	повністю в Telegram, без зовнішніх переходів	в Telegram	потребує переходу у браузер
Наявність реклами	немає	немає	є реклама, банери, редиректи

Рисунок 4.3.1 Порівняння розробки с існуючими рішеннями

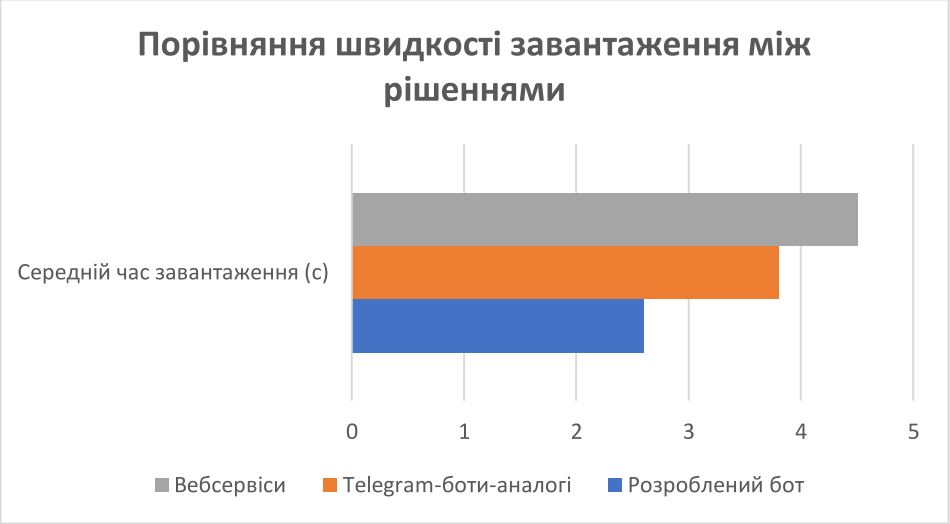

Рисунок 4.3.2 Порівняння швидкості завантаження між рішеннями

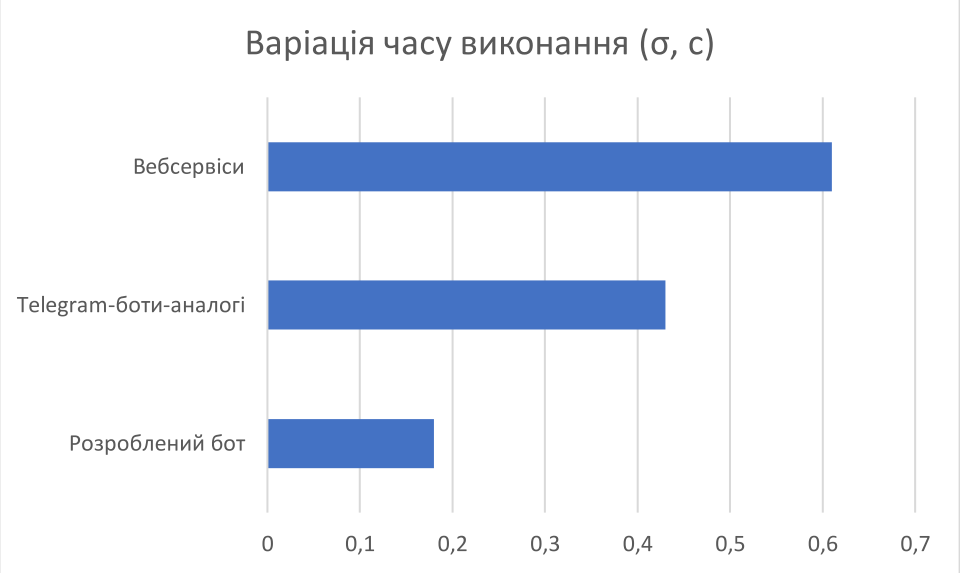

Рисунок 4.3.3 Порівняння часу виконання роботи

Результати порівняння показують, що розроблений Telegram-бот забезпечує ширший функціонал і вищу стабільність роботи, ніж більшість альтернативних рішень. На відміну від вебсервісів, бот не потребує переходу на сторонні сайти, не містить рекламних вставок та не залежить від обмежень браузера. Порівняно з іншими Telegram-ботами, ключовими перевагами є підтримка більшої кількості платформ, наявність автоматичної оптимізації відео та коректна обробка галерей. Асинхронна архітектура забезпечує швидку реакцію на запити та відсутність блокувань під час роботи з великою кількістю користувачів.

4.4 Аналіз економічної ефективності розробки

Економічна ефективність розробленого Telegram-бота оцінювалася на основі порівняння витрат на створення програмного продукту та економічного ефекту, що досягається в результаті автоматизації процесів завантаження й обробки медіафайлів. Оцінювання виконувалося з використанням усереднених показників, що дозволяє отримати узагальнену та методично обґрунтовану характеристику економічної доцільності розробки.

Вартість створення програмного забезпечення визначалася з урахуванням трудових витрат на проєктування архітектури системи, реалізацію основних функціональних модулів, інтеграцію сторонніх інструментів, тестування та підготовку супровідної документації. Для розрахунків використано усереднену трудомісткість розробки та середню погодинну ставку фахівця у сфері програмного забезпечення. За результатами оцінювання орієнтовна вартість розробки Telegram-бота становить близько 32 000 грн. Витрати на програмні ліцензії та спеціалізоване обладнання відсутні, оскільки у процесі розробки застосовувалося відкрите програмне забезпечення.

Місячна економічна ефективність визначалась на основі скорочення витрат часу, пов’язаних із ручним завантаженням та обробкою медіафайлів. Використання Telegram-бота дозволяє автоматизувати зазначені операції та зменшити тривалість виконання рутинних дій для кількох користувачів одночасно. З урахуванням усередненої інтенсивності використання системи та типової економії часу на один запит встановлено, що орієнтовна місячна економічна ефективність становить приблизно 6 000 грн на місяць.

Строк окупності розробки визначався як відношення вартості створення програмного продукту до величини місячної економічної ефективності. За отриманими результатами орієнтовний строк окупності Telegram-бота становить близько 5-6 місяців, після чого використання системи забезпечує стабільний економічний ефект за рахунок подальшої економії часу та ресурсів.

Таким чином, розроблений Telegram-бот є економічно доцільним програмним рішенням, яке характеризується помірною вартістю створення та

відносно коротким строком окупності, зумовленим регулярним використанням системи та автоматизацією обробки мультимедійного контенту. Детальні розрахунки економічної ефективності наведено у Додатку А.

4.5 Практичне значення результатів

Практичне значення розробленого Telegram-бота полягає у забезпеченні швидкої, автоматизованої та зручної обробки медіафайлів із широкого спектра сучасних онлайн-платформ. Реалізований функціонал дозволяє отримувати відео, аудіо та зображення без необхідності залучення сторонніх вебсервісів, ручного завантаження або виконання додаткових конвертацій. Усі операції здійснюються безпосередньо в Telegram, що підвищує доступність інструмента та мінімізує вимоги до підготовки користувача.

Розроблений бот може бути корисним для контент-мейкерів, журналістів, аналітиків та інших фахівців, які регулярно працюють з матеріалами із соціальних мереж і платформ для публікації відео. Автоматичне стиснення та оптимізація відео спрощують підготовку матеріалів до розповсюдження, а підтримка аудіоконвертації створює умови для швидкого формування аудіокліпів, озвучок або фрагментів для подальшого використання.

У групових чатах бот може виконувати функції централізованого інструмента для спільної роботи з медіаконтентом. Завдяки тому, що він реагує на посилання від будь-якого учасника, система забезпечує оперативне поширення відредагованих або завантажених файлів між усіма користувачами групи.

Застосування бота можливе також у навчальному процесі. Його архітектура демонструє практичне використання асинхронних технологій, взаємодії з API, автоматизації завантаження медіафайлів і роботи з мультимедійними форматами. Рішення може слугувати прикладом для студентів і розробників, які вивчають сучасні підходи до побудови мережевих застосунків.

У корпоративному середовищі бот може зменшити витрати часу на обробку медіафайлів, автоматизувати рутинні дії та підвищити продуктивність співробітників. Використання автоматизованого сприяє прискоренню робочих операцій і знижує ймовірність помилок, характерних для ручної взаємодії з медіаданими.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено Telegram-бота для автоматизованого завантаження та обробки медіафайлів із різних вебплатформ. У процесі дослідження проаналізовано сучасні підходи до отримання та трансформації медіаданих, а також визначено ключові технічні вимоги до системи, що працює у режимі асинхронної взаємодії з користувачем. Проведений аналітичний огляд показав, що найбільш ефективними засобами для реалізації поставлених завдань є Python, асинхронні інструменти бібліотеки Aiogram, а також програмні комплекси yt-dlp та FFmpeg, які забезпечують гнучку роботу з мультимедійними матеріалами.

У результаті проєктування створено модульну архітектуру програмного забезпечення, яка включає підсистеми обробки запитів, завантаження медіафайлів, їх перекодування, оптимізації та передачі користувачеві у форматах, сумісних із технічними обмеженнями Telegram. Окремо реалізовано механізми класифікації типів контенту, роботу з галереями, обробку помилок, а також модуль збору статистичних даних. Програмний комплекс протестовано на різних платформах і типах медіа, що дозволило оцінити його продуктивність, стійкість до нестандартних ситуацій і відповідність вимогам до якості результату.

Експериментальні дослідження підтвердили ефективність розроблених алгоритмів. Система стабільно опрацьовує відео, аудіо та зображення, автоматично виконує оптимізацію великих файлів і забезпечує своєчасну доставку результатів користувачеві. Порівняння з існуючими рішеннями показало, що створений бот має ширший функціонал, підтримує більше платформ і пропонує вищий рівень автоматизації, ніж більшість альтернативних сервісів. Здійснена оцінка економічної ефективності показала доцільність використання розробленої системи для автоматизації задач, пов'язаних з підготовкою та передаванням медіаконтенту.

Отримані результати мають практичну цінність для користувачів, які регулярно працюють із мультимедійними матеріалами, а також можуть бути

використані як основа для подальшого розширення функціональності, включно з інтеграцією нових платформ, застосуванням нейромережевих методів покращення якості та впровадженням механізмів інтелектуального вибору параметрів обробки. Розроблений Telegram-бот підтверджує можливість створення ефективних, зручних і гнучких інструментів автоматизації медіаопераційна базі сучасних технологій.

ПЕРЕЛІК ПОСИЛАНЬ

1. Рамальйо Л. Fluent Python. – O'Reilly Media, 2022. – 1015 с.
2. Бяезлі Д. Python Cookbook. – O'Reilly Media, 2020. – 706 с.
3. Python Software Foundation. Python Documentation [Електронний ресурс]. – Режим доступу: https://www.python.org/doc/
4. Aiogram Framework. Documentation [Електронний ресурс]. – Режим доступу: https://docs.aiogram.dev/
5. Telegram Bot API. Офіційна документація [Електронний ресурс]. – Режим доступу: https://core.telegram.org/bots/api
6. FFmpeg Project. Documentation [Електронний ресурс]. – Режим доступу: https://ffmpeg.org/documentation.html
7. yt-dlp Project. GitHub Repository [Електронний ресурс]. – Режим доступу: https://github.com/yt-dlp/yt-dlp
8. Теленик С.Ф. Програмування мовою Python: підручник. – Київ: Кондор, 2020. – 388 с.
9. Мельник О.Г., Романюк В.В. Основи програмної інженерії. – Львів: ЛНУ, 2019. – 312 с.
10. Павленко І.М. Системи обробки мультимедійних даних. – Київ: НТУУ «КПІ», 2018. – 244 с.
11. Бондурянська О.П. Комп'ютерні інформаційні технології. – Харків: ХНЕУ, 2021. – 280 с.
12. Копійка О.В. Інформаційна безпека: підручник. – Київ: Центр учбової літератури, 2019. – 356 с.
13. ISO/IEC 14496-3: Coding of audio–visual objects. Audio. – ISO, 2020.
14. ISO/IEC 23008-2: High Efficiency Video Coding. – ISO, 2021.
15. Stevens W. UNIX Network Programming. – Addison-Wesley, 2020. – 960 p.
16. Tanenbaum A., Wetherall D. Computer Networks. – Pearson, 2018. – 960 p.
17. Goodfellow I., Bengio Y., Courville A. Deep Learning. – MIT Press, 2016. – 775 p.

18. SQLite Consortium. Documentation [Електронний ресурс]. – Режим доступу: https://sqlite.org/docs.html
19. Stallings W. Operating Systems. – Pearson, 2018. – 820 p.
20. Fielding R. Architectural Styles and the Design of Network-based Software Architectures. – UC Irvine, 2000. – 184 p.
21. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns. – Addison-Wesley, 2019. – 395 p.
22. Pressman R., Maxim B. Software Engineering: A Practitioner’s Approach. – McGraw-Hill, 2021. – 940 p.
23. Sommerville I. Software Engineering. – Pearson, 2020. – 832 p.
24. IEEE Std 830-1998. Recommended Practice for Software Requirements Specifications. – IEEE, 1998.
25. IEEE Std 1016-2009. Standard for Software Design Descriptions. – IEEE, 2009.
26. Козаченко А.А., Ємець О.М. Інформаційні системи та технології. – Київ: НАУ, 2020. – 276 с.
27. Журбенко В.О. Основи кібербезпеки. – Харків: ХНУ, 2021. – 214 с.
28. Литвин В.В. Основи штучного інтелекту. – Львів: ЛНУ, 2020. – 420 с.
29. Бублик В.В. Мережеві технології: підручник. – Київ: КНЕУ, 2019. – 364 с.
30. Сидоренко І.О. Проектування інформаційних систем. – Дніпро: НМетАУ, 2021. – 289 с.

Додаток А

Розрахунок економічної ефективності розробки Telegram-бота

Для оцінювання економічної ефективності розробки використано усереднені показники, що характеризують типові умови створення та використання програмного продукту.

Показник	Значення
Трудомісткість розробки	80 год
Середня погодинна ставка розробника	400 грн/год
Кількість користувачів	2
Кількість запитів на день (на користувача)	15
Кількість робочих днів на місяць	20
Середня економія часу на 1 запит	5 хв
Вартість 1 години робочого часу користувача	200 грн/год

Рисунок А.1 Вихідні дані для розрахунку

Вартість розробки програмного продукту визначається за формулою:

$$C_P = T * S$$

де

T - трудомісткість розробки, год;

S - погодинна ставка розробника, грн/год.

Підставляючи значення, отримуємо:

$$C_p = 80 * 400 = 32\,000 \text{ грн}$$

Отже, орієнтовна вартість розробки Telegram-бота становить 32 000 грн.

Місячна економія часу одного користувача визначається за формулою:

$$T_e = N * D * t$$

де

N - кількість запитів на день;

D - кількість робочих днів на місяць;

t - економія часу на один запит, год.

Переведення хвилин у години:

5 хв = 0,083 год.

$$T_e = 15 * 20 * 0{,}083 \approx 24{,}9 \text{ год}$$

З урахуванням двох користувачів загальна місячна економія часу:

$$T_e\text{заг} = 24{,}9 * 2 \approx 50 \text{ год}$$

Місячна економічна ефективність визначається як:

$$E_m = T_e\text{заг} * C_h$$

де

C_h - вартість години робочого часу користувача.

$$E_m = 50 * 200 = 10\,000 \text{ грн}$$

З урахуванням можливих нерівномірностей використання та резерву на непродуктивний час приймається усереднене значення місячної економічної ефективності:

$$E_m \approx 6\,000 \text{ грн/місяць}$$

Строк окупності визначається за формулою:

$$T_e = \frac{C_P}{E_m}$$

де

C_p - вартість розробки;

E_m - місячна економічна ефективність.

$$T_e = \frac{32\ 000}{6\ 000} \approx 5{,}3 \text{ місяця}$$

Показник	Значення
Вартість розробки	32 000 грн
Місячна економічна ефективність	6 000 грн
Строк окупності	5–6 місяців

Рисунок А.2 Узагальнена таблиця результатів

Отже, орієнтовний строк окупності розробки Telegram-бота становить 5–6 місяців.

Додаток Б

Додаткові графіки результатів експериментів

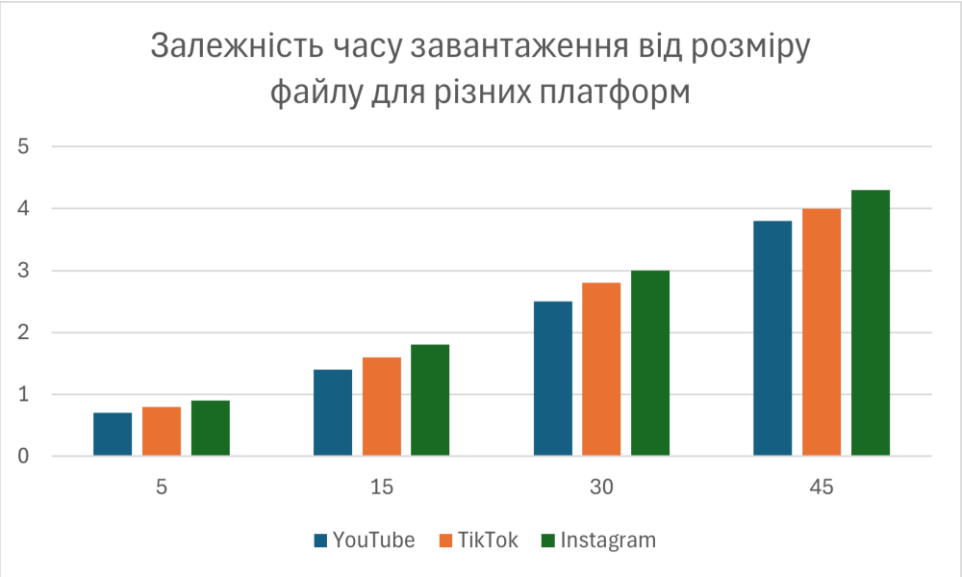

Рисунок Б.1 Залежність часу завантаження від розміру файлу для різних платформ

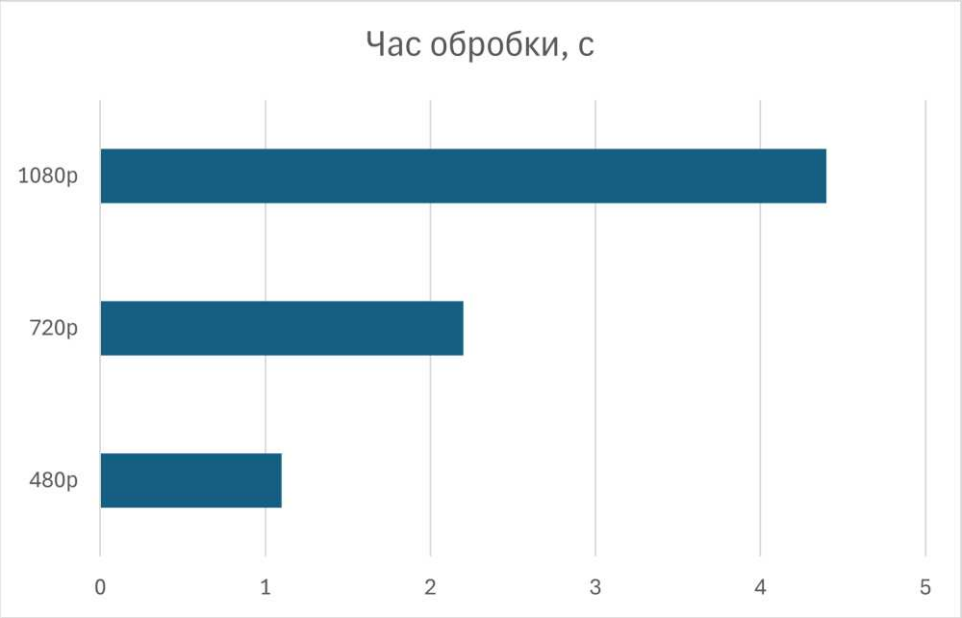

Рисунок Б.2. Залежність часу обробки від роздільної здатності відео

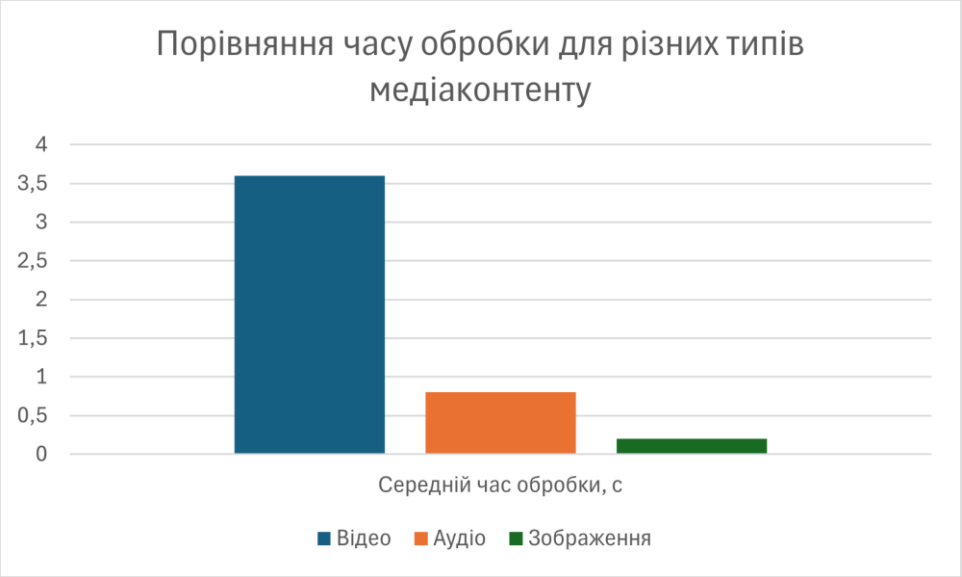

Рисунок Б.3. Порівняння часу обробки для різних типів медіаконтенту

Рисунок Б.4. Стабільність алгоритмів обробки медіафайлів

Додаток В

Розширені таблиці експериментальних даних

№ тесту	Роздільність	Тривалість відео, с	Початковий розмір, МБ	Час обробки, с
1	720p	30	48	3,4
2	720p	30	45	3,6
3	1080p	30	50	3,5
4	1080p	30	42	3,7
5	1080p	30	47	3,5

Рисунок В.1 Результати серії тестів обробки відеофайлів

№ тесту	Формат джерела	Тривалість аудіо, с	Початковий розмір, МБ	Час обробки, с
1	AAC	60	5,2	0,7
2	AAC	60	5	0,8
3	MP3	60	5,5	0,9
4	MP3	60	5,1	0,8
5	AAC	60	5,3	0,8

Рисунок В.2 Результати серії тестів обробки аудіофайлів

№ тесту	Формат	Роздільність	Початковий розмір, МБ	Час обробки, с
1	JPEG	1920×1080	2,1	0,2
2	JPEG	1920×1080	2	0,2
3	PNG	1920×1080	3,4	0,3
4	JPEG	1280×720	1,5	0,2
5	PNG	1280×720	2,6	0,2

Рисунок В.3 Результати серії тестів обробки зображень

Тип медіа	Мінімальний час, с	Максимальний час, с	Середній час, с	Середнє відхилення, с
Відео	3,4	3,7	3,5	0,12
Аудіо	0,7	0,9	0,8	0,08
Зображення	0,2	0,3	0,2	0,04

Рисунок В.4 Узагальнені статистичні показники часу обробки

№ тесту	Початковий розмір, МБ	Кінцевий розмір, МБ	Коефіцієнт стиснення	Зменшення розміру, %
1	48	18	2,7	62,5
2	45	17	2,6	62,2
3	50	19	2,6	62
4	42	16	2,6	61,9
5	47	18	2,6	61,7

Рисунок В.5 Параметри стиснення відеофайлів

Показник	Значення	Кінцевий розмір, МБ
Середній коефіцієнт стиснення	2,62	18
Середнє зменшення розміру, %	62,1	17
Максимальне зменшення, %	62,5	19
Мінімальне зменшення, %	61,7	16

Рисунок В.6 Узагальнені показники ефективності стиснення

Додаток Г

UML-діаграми у повному вигляді

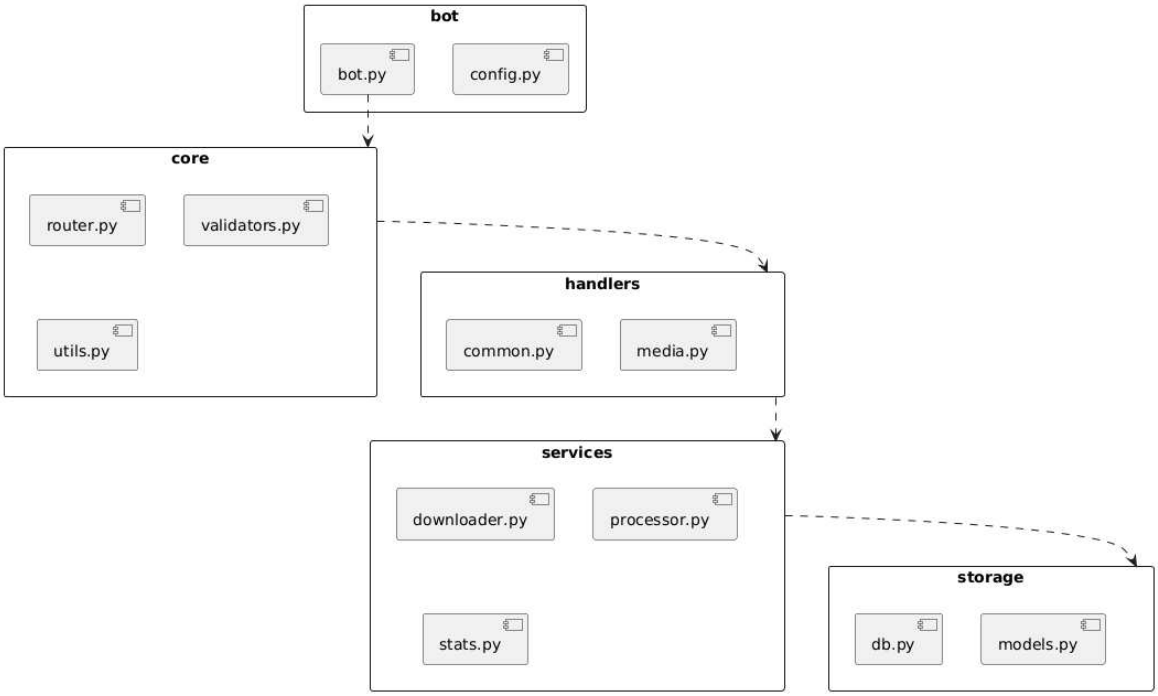

Рисунок Г.1 Структура модулів

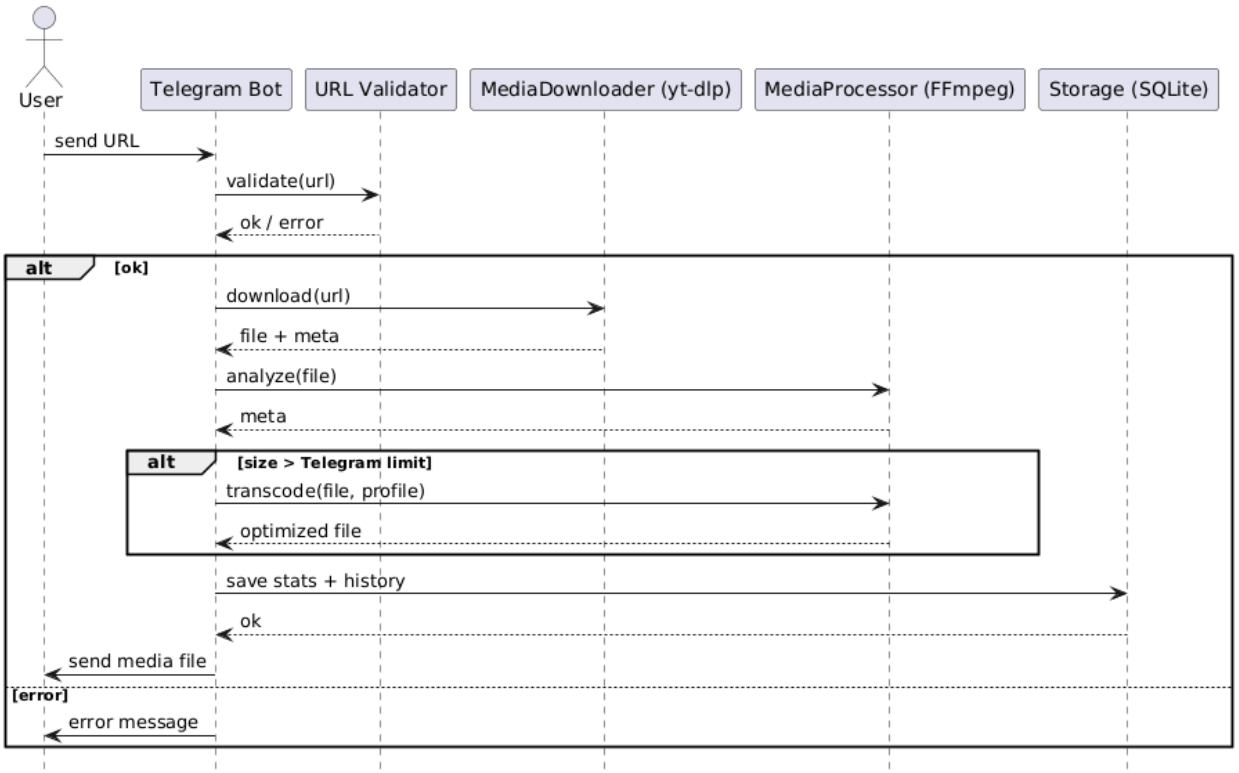

Рисунок Г.2 Обробка посилання у приватному чаті

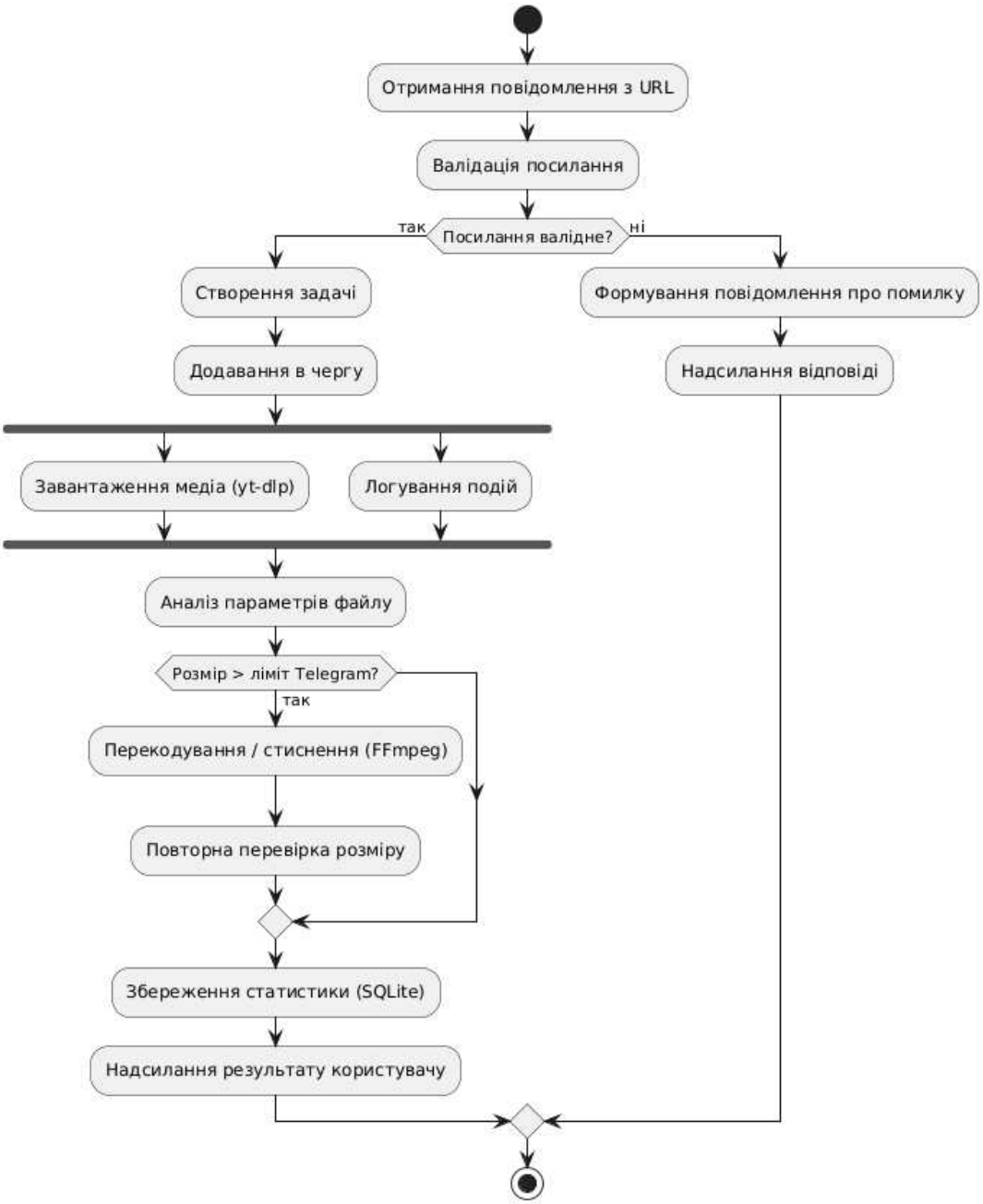

Рисунок Г.3 Блок-схема асинхронної обробки запитів

Додаток Д

Фрагменти програмного коду

Bot.py

```
import asyncio
import logging
from aiogram import Bot, Dispatcher
from aiogram.enums import ParseMode
from aiogram.client.default import DefaultBotProperties

from config import TELEGRAM_BOT_TOKEN as token
from handlers import router


async def main():
    logging.basicConfig(
        level=logging.INFO,
        format="%(asctime)s [%(levelname)s] %(name)s:
%(message)s"
    )

    bot = Bot(
        token=token,
        default=DefaultBotProperties(parse_mode=ParseMode.HTML)
    )

    dp = Dispatcher()
    dp.include_router(router)

    logging.info("✅ Бот запущено")

    await dp.start_polling(bot)


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

Config.py

```
TELEGRAM_BOT_TOKEN =
"8479767730:AAG6lnqN3uzfuwJI2XwEXUpPt3LVz_jSR4A"

DOWNLOAD_DIR = "downloads"

SIZE_SKIP_REENCODE = 20 * 1024 * 1024
SIZE_SOFT_REENCODE = 50 * 1024 * 1024
```

Common.py

```
from aiogram import Router, types, F
from aiogram.filters import Command
from aiogram.utils.keyboard import InlineKeyboardBuilder
from services import stats as stats_service

router = Router()


def build_main_keyboard(bot_username: str) ->
types.InlineKeyboardMarkup:

    add_to_group_url =
f"https://t.me/{bot_username}?startgroup=true"

    kb = InlineKeyboardBuilder()
    kb.button(
        text="➕ Додати бота в групу",
        url=add_to_group_url,
    )
    kb.button(
        text="ℹ️ Як налаштувати в групі",
        callback_data="group_help",
    )
    kb.adjust(1)
    return kb.as_markup()


@router.message(Command("start"))
async def cmd_start(message: types.Message) -> None:
    me = await message.bot.get_me()
    bot_username = me.username or ""

    text = (
        "Вітаю! 👋\n\n"
        "Я - Telegram-бот для автоматизованого завантаження та
обробки медіафайлів.\n\n"
        "Що я вмію:\n"
        "• завантажувати відео та аудіо з популярних
платформ;\n"
        "• оптимізувати та стискати файли за допомогою
FFmpeg;\n"
        "• підбирати зручний формат вихідного файлу;\n"
        "• зберігати базову статистику використання.\n\n"
        "Щоб розпочати, просто надішли мені посилання на відео,
аудіо або інший медіаконтент.\n\n"
        "Додаткові команди:\n"
        "• /help - детальна довідка\n"
        "• /stats - коротка статистика завантажень\n"
        "• /settings - (опційно) налаштування профілю\n\n"
        "Можеш також додати мене в групу, щоб я автоматично
обробляв посилання там."
    )
```

```
    await message.answer(
        text,
        reply_markup=build_main_keyboard(bot_username),
    )


@router.message(Command("help"))
async def cmd_help(message: types.Message) -> None:
    me = await message.bot.get_me()
    bot_username = me.username or ""

    text = (
        "📘 *Довідка по боту*\n\n"
        "*Як користуватися в особистих повідомленнях:*\n"
        "1️⃣ Надішли посилання на відео, аудіо або інший
медіаконтент.\n"
        "2️⃣ Я визначу платформу, завантажу файл і, за потреби,
виконаю стиснення.\n"
        "3️⃣ Отримаєш готовий файл у відповідь у зручному
форматі.\n\n"
        "*Як користуватися в групі:*\n"
        "• додай бота в групу (кнопка нижче або вручну);\n"
        "• вимкни privacy для бота через @BotFather (/setprivacy
→ Disable);\n"
        "• надсилай посилання в чат – бот відповідатиме файлом у
відповідь.\n\n"
        "*Основні команди:*\n"
        "• /start – початок роботи з ботом;\n"
        "• /help – це повідомлення з інструкціями;\n"
        "• /stats – перегляд агрегованої статистики
завантажень;\n"
        "• /settings – (за наявності) налаштування обмежень та
параметрів.\n"
    )

    await message.answer(
        text,
        reply_markup=build_main_keyboard(bot_username),
        parse_mode="Markdown",
    )


@router.callback_query(F.data == "group_help")
async def on_group_help(callback: types.CallbackQuery) -> None:
    text = (
        "ℹ️ *Як налаштувати бота в групі*\n\n"
        "1️⃣ Додай бота в групу через кнопку «➕ Додати бота в
групу» або вручну.\n\n"
        "2️⃣ У @BotFather виконай налаштування:\n"
        "   • команда /setprivacy;\n"
        "   • обери мого бота;\n"
        "   • натисни *Disable* – тоді я зможу бачити всі
```

```
повідомлення з посиланнями.\n\n"
        "🔑 *Адмін-права:*\n"
        "   Для базового завантаження медіа за посиланнями
адмін-права не є обов’язковими.\n"
        "   Вони потрібні лише якщо бот має:\n"
        "   • видаляти повідомлення,\n"
        "   • виконувати антиспам чи автоприбирання,\n"
        "   • керувати налаштуваннями чату.\n\n"
        "📌 Після додавання в групу просто кидай посилання – я
автоматично "
        "завантажуватиму медіа й надсилатиму файли у відповідь."
    )

    await callback.answer()
    await callback.message.answer(text, parse_mode="Markdown")


@router.message(Command("stats"))
async def cmd_stats(message: types.Message) -> None:
    data = stats_service.snapshot()
    total = data["total"]
    by_type: dict = data["by_type"]
    by_source: dict = data["by_source"]

    if total == 0:
        await message.answer(
            "Статистика поки що порожня, ще не було жодного
завершеного завантаження."
        )
        return

    lines: list[str] = [f"📊 *Статистика з моменту запуску
бота*"]
    lines.append(f"Всього завантажень: *{total}*")

    if by_type:
        lines.append("\nЗа типами медіа:")
        label_map = {
            "video": "відео",
            "audio": "аудіо",
            "image": "зображення",
            "gallery": "галереї",
            "other": "інше",
        }
        for key, label in label_map.items():
            if key in by_type:
                lines.append(f"• {label}: {by_type[key]}")

    if by_source:
        lines.append("\nТоп джерел:")
        sorted_sources = sorted(by_source.items(), key=lambda x:
x[1], reverse=True)[:5]
        for domain, count in sorted_sources:
```

```
            lines.append(f"• {domain}: {count}")

    await message.answer("\n".join(lines),
parse_mode="Markdown")
```

Audio.py

```
import os
from typing import Any, Dict

import requests


def beautify_audio_mp3(input_path: str, info: Dict[str, Any]) ->
str:

    base, ext = os.path.splitext(input_path)
    ext_lower = ext.lower()

    target_path = input_path

    if ext_lower != ".mp3":
        import subprocess

        target_path = base + ".mp3"
        cmd = [
            "ffmpeg",
            "-i", input_path,
            "-vn",
            "-acodec", "libmp3lame",
            "-b:a", "192k",
            "-y",
            target_path,
        ]

        try:
            subprocess.run(
                cmd,
                stdout=subprocess.DEVNULL,
                stderr=subprocess.DEVNULL,
                check=False,
            )
            try:
                os.remove(input_path)
            except OSError:
                pass
        except Exception:
            # Якщо ffmpeg зламався – залишаємо оригінал
            return input_path

    try:
        from mutagen.mp3 import MP3
        from mutagen.id3 import ID3, APIC, TIT2, TPE1, TALB
    except ImportError:
        # mutagen не встановлений – повертаємо шлях як є
```

```
    return target_path

title = (
    info.get("track")
    or info.get("title")
    or info.get("alt_title")
    or "Audio"
)
artist = (
    info.get("artist")
    or info.get("uploader")
    or info.get("creator")
    or info.get("uploader_id")
    or ""
)
album = (
    info.get("album")
    or info.get("playlist_title")
    or info.get("channel")
    or ""
)

thumb_url = info.get("thumbnail")
if not thumb_url:
    thumbs = info.get("thumbnails") or []
    if thumbs:
        thumb_url = thumbs[-1].get("url")

cover_data = None
if thumb_url:
    try:
        resp = requests.get(thumb_url, timeout=10)
        resp.raise_for_status()
        cover_data = resp.content
    except Exception:
        cover_data = None

try:
    audio = MP3(target_path, ID3=ID3)
    if audio.tags is None:
        audio.add_tags()
except Exception:
    return target_path

try:
    audio.tags["TIT2"] = TIT2(encoding=3, text=title)
    if artist:
        audio.tags["TPE1"] = TPE1(encoding=3, text=artist)
    if album:
        audio.tags["TALB"] = TALB(encoding=3, text=album)

    if cover_data:
        audio.tags["APIC"] = APIC(
            encoding=3,
```

```
                mime="image/jpeg",
                type=3,  # front cover
                desc="Cover",
                data=cover_data,
            )

        audio.save()
    except Exception:
        pass

    return target_path
```

http.py

```
import os
from typing import Optional

import requests

from config import DOWNLOAD_DIR


def download_file(
    url: str,
    *,
    basename_hint: Optional[str] = None,
) -> str:

    os.makedirs(DOWNLOAD_DIR, exist_ok=True)

    clean_url = url.split("?", 1)[0]
    name = basename_hint or clean_url.split("/")[-1] or "file"

    if "." not in name:
        ext = os.path.splitext(clean_url)[1]
        if ext:
            name += ext

    path = os.path.join(DOWNLOAD_DIR, name)

    resp = requests.get(url, stream=True, timeout=20)
    resp.raise_for_status()

    with open(path, "wb") as f:
        for chunk in resp.iter_content(chunk_size=8192):
            if chunk:
                f.write(chunk)

    return path
```

types.py

```
from typing import Any, Dict, List

AUDIO_EXTENSIONS = {"mp3", "m4a", "ogg", "opus", "flac", "wav"}
IMAGE_EXTENSIONS = {"jpg", "jpeg", "png", "gif", "webp", "bmp"}


def detect_media_type(info: Dict[str, Any]) -> str:

    vcodec = (info.get("vcodec") or "").lower()
    acodec = (info.get("acodec") or "").lower()
    ext = (info.get("ext") or "").lower()

    if vcodec and vcodec != "none":
        return "video"

    if acodec and acodec != "none":
        return "audio"

    if ext in AUDIO_EXTENSIONS:
        return "audio"

    if ext in IMAGE_EXTENSIONS:
        return "image"

    return "other"


def all_entries_are_images(entries: List[Dict[str, Any]]) ->
bool:

    for e in entries:
        if not e:
            continue
        ext = (e.get("ext") or "").lower()
        if ext not in IMAGE_EXTENSIONS:
            return False
    return True


def is_instagram_playlist(info: Dict[str, Any]) -> bool:

    if info.get("_type") != "playlist":
        return False

    extractor = (info.get("extractor") or
info.get("extractor_key") or "").lower()
    return "instagram" in extractor
```