Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБЛЕННЯ ВЕБПЛАТФОРМИ ДЛЯ АВТОМАТИЗОВАНОГО СТВОРЕННЯ КОРОТКИХ ВІДЕОКЛІПІВ ІЗ ВИКОРИСТАННЯМ ЗАСОБІВ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ОБРОБЛЕННЯ ВІДЕО ТА ТРАНСКРИПЦІЇ МОВЛЕННЯ
(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-22-1
Ольховський Б.Д.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник ст.викл. Сінельнікова Т.Ф.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О. А.
(підпис) (прізвище, ініціали)
2026 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Ольховському Богдану Дмитровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення вебплатформи для автоматизованого створення коротких відеокліпів із використанням засобів штучного інтелекту для оброблення відео та транскрипції мовлення
затверджена наказом університету від 26 травня 2026 року № 435Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 23 червня 2026 р.
3. Вихідні дані до роботи та науково-технічна література, матеріали конференцій, дані інтернет-мережі, документація FastAPI, FFmpeg, faster-whisper, yt-dlp, YouTube Data API v3, TikTok ContentPosting API, Instagram Graph API, Stripe API, а також вихідний код розробленої вебплатформи clipperok.

4. Перелік питань, що потрібно опрацювати в роботі
1. Огляд методів автоматичного розпізнавання мовлення, визначення ключових моментів у відео та програмного оброблення медіаданих.
2. Проєктування архітектури вебплатформи для автоматизованого створення коротких вертикальних відеокліпів.
3. Розроблення медіапайплайну оброблення відео з використанням транскрипції мовлення, ШІ-аналізу та рендерингу кліпів.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність автоматизованого створення коротких відеокліпів, постановка задачі, архітектура вебплатформи, схема бази даних, IDEF0-діаграма процесу оброблення відео, діаграми декомпозиції процесу, діаграма варіантів використання, схема медіапайплайну, скріншоти інтерфейсу користувача.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	13.04.2026	
2	Аналіз завдання, підбір літератури	15.04.26-16.04.26	
3	Аналіз літератури з проблеми, що вирішується	21.04.26-23.04.26	
4	Аналіз технічних засобів	25.04.26-26.04.26	
5	Формування кроків вирішення проблеми	26.04.26-02.05.26	
6	Програмна реалізація	02.04.26-01.05.26	
7	Аналіз отриманих результатів	01.05.26-04.06.26	
8	Оформлення пояснювальної записки	06.06.26-08.06.26	
9	Перевірка на нормоконтроль	10.06.26-19.06.26	
10	Перевірка на плагіат	11.06.26-30.06.26	
11	Рецензування	20.06.26-30.06.26	
12	Підготовка презентації та доповіді	20.06.26-30.06.26	
13	Занесення роботи в електронний архів	30.06.26-09.07.26	
14	Попередній захист кваліфікаційної роботи	30.06.26-09.07.26	

Дата видачі завдання 13 квітня 2026 р.

Здобувач _____
(підпис)

Керівник роботи _____ _____ст. викл Сінельнікова Т.Ф
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 91 с., 11 табл., 16 рис., 1 дод., 37 джерел.

АВТОМАТИЗАЦІЯ, ВЕБПЛАТФОРМА, ВІДЕОКЛІПИ, ШТУЧНИЙ ІНТЕЛЕКТ, DEEPSEEK, FASTAPI, FFMPEG, PYTHON, SAAS, WHISPER.

Об’єктом роботи є процес автоматизованого оброблення відеоданих та створення коротких вертикальних відеокліпів.

Метою роботи є розроблення вебплатформи для автоматичної транскрипції мовлення, аналізу ключових фрагментів та публікації вертикальних кліпів у соціальних мережах.

Під час роботи використано: методи штучного інтелекту (автоматична транскрипція мовлення, семантичний аналіз, обробка відео), методи веброзробки створення серверної та клієнтської частин, інтеграція API соціальних мереж.

Практична цінність роботи полягає у скороченні часу створення відеоконтенту з кількох годин монтажу до кількох хвилин автоматизованої оброблення.

Розроблено вебплатформу, що забезпечує імпорт відео, транскрипцію, визначення кліпів, рендеринг із субтитрами та публікацію

Область застосування: digital-маркетинг, блогінг, медіавиробництво, автоматизація відеовиробничих процесів.

AUTOMATION, WEB PLATFORM, VIDEO CLIPS, ARTIFICIAL INTELLIGENCE, DEEPSEEK, FASTAPI, FFMPEG, PYTHON, SAAS, WHISPER.

The objective of this work is the process of automated video data processing and creation of short vertical video clips.

The goal of this work is to develop a web platform for automatic speech transcription, key fragment detection, and publishing vertical clips to social media.

The following were used: artificial intelligence methods (automatic speech recognition, semantic analysis, video processing), and web development methods (backend, frontend, social media API integration).

The practical value of the work lies in reducing video content creation time from several hours of manual editing to a few minutes of automated processing.

A web platform has been developed providing video import, transcription, clip detection, subtitle rendering, and publishing.

Applications: digital marketing, blogging, media production, video workflow automation.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

.ass – розширення файлу субтитрів формату Advanced SubStation Alpha

9:16 – співвідношення сторін вертикального відео, стандарт платформ TikTok, Instagram Reels та YouTube Shorts

1080×1920 – стандартна роздільна здатність вертикального відео у пікселях (ширина × висота)

ШІ – штучний інтелект

AAC – Advanced Audio Coding (удосконалене аудіокодування; стандарт стиснення аудіо, що застосовується у відеофайлах MP4)

AES-128-CBC – Advanced Encryption Standard, 128-bit key, Cipher Block Chaining (стандарт симетричного блочного шифрування, що застосовується в алгоритмі Fernet)

Alembic – інструмент керування міграціями схеми бази даних для SQLAlchemy

API – Application Programming Interface (інтерфейс програмного застосунку)

ASR – Automatic Speech Recognition (автоматичне розпізнавання мовлення)

ASS – Advanced SubStation Alpha (формат файлів субтитрів із підтримкою розширеної стилізації тексту та анімації)

Attention – механізм уваги в нейронних мережах типу Transformer

B-roll – додатковий відеоряд, який використовується для візуального доповнення основного фрагмента

backend – серверна частина вебзастосунку, що відповідає за бізнес-логіку, роботу з базою даних, API та фонові задачі

bcrypt – адаптивний алгоритм хешування паролів із вбудованою сіллю та налаштовуваною складністю.

CDN – Content Delivery Network (мережа доставки контенту), інфраструктура для швидкого надання файлів користувачам через географічно розподілені сервери

CI/CD – Continuous Integration / Continuous Deployment (безперервна інтеграція та розгортання)

CNN – Convolutional Neural Network (згорткова нейронна мережа), клас нейронних мереж для аналізу зображень і відеокадрів

CPU – Central Processing Unit (центральний процесор)

CRF – Constant Rate Factor (константний коефіцієнт якості), параметр кодувальника H.264, що визначає якість зображення; значення 23 є стандартним балансом якість/розмір

CSRF – Cross-Site Request Forgery (міжсайтова підробка запиту), вид атаки на вебзастосунки

CUDA – Compute Unified Device Architecture, технологія NVIDIA для виконання паралельних обчислень на GPU

DBML – Database Markup Language, текстовий формат опису структури бази даних і зв’язків між таблицями

DDL – Data Definition Language (мова визначення даних), частина SQL для опису структури таблиць, ключів та обмежень

DeepSeek – велика мовна модель з відкритими вагами від компанії DeepSeek AI, сумісна з OpenAI API

Deno – сучасний рушій виконання JavaScript та TypeScript, що використовується як залежність yt-dlp

Docker – платформа контейнеризації застосунків; забезпечує ізоляцію середовища виконання та відтворюване розгортання

ERD – Entity Relationship Diagram (діаграма «сутність-зв’язок»), схема таблиць бази даних та зв’язків між ними

FastAPI – вебфреймворк для Python з автоматичною генерацією OpenAPI-документації та підтримкою асинхронних обробників.

faster-whisper – оптимізована реалізація моделі Whisper з квантизацією ваг до формату int8

Fernet – схема симетричного шифрування з автентифікацією повідомлень; використовується для захищеного зберігання OAuth-токенів.

FFmpeg – відкрита кросплатформна бібліотека та утиліта для оброблення відео та аудіо

frontend – клієнтська частина вебзастосунку, що відповідає за інтерфейс користувача та взаємодію з серверною частиною через API

GitHub Actions – хмарна система CI/CD від GitHub для автоматизації збірки, тестування та розгортання

GMM – Gaussian Mixture Model (гаусівська модель мішанини), статистична модель, що застосовувалась в ASR-системах першого покоління

GPU – Graphics Processing Unit (графічний процесор), пристрій для паралельних обчислень, який може використовуватися для прискорення моделей ШІ

H.264 – стандарт відеокодування Advanced Video Coding, найпоширеніший формат стиснення відео

HMM – Hidden Markov Model (прихована марківська модель), статистична модель послідовностей, що застосовувалась в ASR першого покоління

HMAC-SHA256 – Hash-based Message Authentication Code із хеш-функцією SHA-256, алгоритм перевірки цілісності та автентифікації повідомлень

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту)

HSTS – HTTP Strict Transport Security, заголовок безпеки, що забороняє браузеру звертатися до сайту через незахищений HTTP

HSTS - HTTP Strict Transport Security, заголовок безпеки, що забороняє браузеру звертатися до сайту через незахищений HTTP

IDEF0 – метод функціонального моделювання процесів, у якому відображаються входи, виходи, керування та механізми виконання функції

Instagram Graph API – офіційний API Meta для публікації контенту в Instagram, включаючи Reels

int8 – формат 8-бітного цілого числа; у контексті нейронних мереж є формат квантизації вагових коефіцієнтів

JSON – JavaScript Object Notation, текстовий формат обміну даними на основі підмножини JavaScript

JWT – JSON Web Token (відкритий стандарт RFC 7519 для передачі тверджень між сторонами у вигляді підписаного JSON-об’єкта)

LLM – Large Language Model (велика мовна модель) клас нейронних мереж, навчених на великих текстових масивах

LSTM – Long Short-Term Memory (довга короткострокова пам’ять) різновид рекурентної нейронної мережі

MediaFileUpload – клас бібліотеки google-api-python-client для завантаження локальних файлів через YouTube Data API

Mermaid – текстовий формат опису діаграм, який дозволяє створювати блок-схеми, ERD та інші схеми з коду

MIME – Multipurpose Internet Mail Extensions, стандарт опису типу вмісту файлу або HTTP-відповіді

MP4 – контейнерний формат мультимедійних файлів, що використовується для зберігання відео, аудіо та субтитрів

n8n – відкрита платформа автоматизації робочих процесів типу no-code/low-code

OAuth 2.0 – Open Authorization 2.0 (відкритий протокол авторизації, що дозволяє стороннім застосункам отримувати обмежений доступ до ресурсів користувача)

OpenAPI – специфікація для опису RESTful API у форматі JSON або YAML

ORM – Object-Relational Mapping (об’єктно-реляційне відображення), технологія взаємодії з базою даних через об’єкти мови програмування

PBKDF2 – Password-Based Key Derivation Function 2, функція деривації ключа шифрування з пароля із використанням salt та множинних ітерацій

Pillow – бібліотека оброблення зображень для Python; використовується для генерації title-оверлеїв на кліпах

PNG – Portable Network Graphics (формат растрових зображень із стисненням без втрат)

PostgreSQL – відкрита об’єктно-реляційна система керування базами даних

Pydantic – бібліотека валідації та серіалізації даних для Python на основі анотацій типів

Python – інтерпретована мова програмування загального призначення, що є основною мовою розробки серверной частини платформи

React – JavaScript-бібліотека від Meta для побудови компонентних користувацьких інтерфейсів

Redis – база даних типу «ключ-значення» в оперативній пам’яті (використовується як опціональний кеш)

REST – Representational State Transfer (архітектурний стиль взаємодії розподілених систем через HTTP)

RNN – Recurrent Neural Network (рекурентна нейронна мережа), клас нейронних мереж із циклічними зв’язками

SaaS – Software as a Service (модель надання програмного забезпечення як послуги через інтернет за підпискою)

SHA-256 – Secure Hash Algorithm 256-bit, криптографічна хеш-функція; використовується для зберігання API-ключів у БД

SOP – Same-Origin Policy (політика однакового походження), механізм браузерної безпеки, що обмежує міжсайтові запити

SPA – Single Page Application (односторінковий застосунок), вебзастосунок, що динамічно оновлює вміст без перезавантаження сторінки

SQLAlchemy – ORM-бібліотека для Python з підтримкою PostgreSQL, SQLite та інших СКБД

SSRF – Server-Side Request Forgery (підробка серверного запиту), атака, за якої зловмисник змушує сервер виконати запит до небажаної адреси

Stripe – хмарний платіжний сервіс; використовується для оброблення підписок та покупки токенів

TF-IDF – Term Frequency-Inverse Document Frequency, статистичний показник важливості слова в документі відносно набору документів

ThreadPoolExecutor – клас Python для паралельного виконання задач у пулі потоків; використовується для фонової оброблення відео

TikTok Content Posting API – офіційний API TikTok для програмної публікації відеоконтенту через Direct Post

Transformer – архітектура нейронної мережі, заснована на механізмі уваги (Attention), що є основою сучасних LLM та ASR-моделей

TypeScript – типізована надмножина JavaScript, що компілюється у звичайний JavaScript; використовується у клієнтській частині платформи

URI – Uniform Resource Identifier (уніфікований ідентифікатор ресурсу), рядок для ідентифікації ресурсу

URL – Uniform Resource Locator (уніфікований покажчик ресурсу), адреса ресурсу в мережі

UUID – Universally Unique Identifier (універсальний унікальний ідентифікатор), стандарт генерації унікальних ідентифікаторів; використовується як task_id

Uvicorn – ASGI-сервер для Python на базі uvloop та httptools; використовується для запуску FastAPI у у виробничому середовищі

WebSocket – протокол двостороннього повнодуплексного зв’язку поверх HTTP; використовується для відображення прогресу оброблення у реальному часі

WER – Word Error Rate (частота помилок у словах), метрика якості автоматичного розпізнавання мовлення

Whisper – модель автоматичного розпізнавання мовлення від OpenAI, навчена на 680 000 годинах різномовного аудіо

XSS – Cross-Site Scripting (міжсайтовий скриптинг), атака, за якої у вебсторінку впроваджується шкідливий JavaScript-код

yt-dlp – відкритий інструмент командного рядка для завантаження відео з YouTube та інших платформ

YouTube Data API v3 – офіційний API Google для взаємодії з YouTube, включаючи завантаження та публікацію відео

Zapier – хмарна платформа автоматизації, що може інтегруватись з платформою через API-ключі

ВСТУП

Сучасний розвиток цифрових технологій супроводжується стрімким зростанням обсягів мультимедійного контенту. Особливої популярності набув формат коротких вертикальних відео, представлений платформами TikTok, Instagram Reels та YouTube Shorts. Станом на 2026 рік сукупна аудиторія цих платформ перевищує 2 мільярди активних користувачів на місяць, а частка короткого формату відео у загальному інтернет-трафіку продовжує зростати.

Водночас процес створення якісного короткого відеоконтенту залишається трудомістким. Він охоплює перегляд і аналіз довгого відеоматеріалу, пошук найбільш значущих моментів, монтаж, додавання субтитрів та адаптацію під вимоги конкретної платформи. У більшості випадків ці операції виконуються вручну досвідченим відеоредактором, що потребує значних витрат часу та відповідних фахових навичок.

Розвиток методів штучного інтелекту, зокрема автоматичного розпізнавання мовлення (ASR) та великих мовних моделей (LLM), відкриває можливості для автоматизації цих процесів. Завдяки мультимодальним підходам відеоконтент можна аналізувати одночасно на рівні аудіо та тексту, що підвищує точність виявлення ключових фрагментів порівняно з суто сигнальними методами.

Актуальність роботи полягає у розробленні SaaS-вебплатформи для автоматизованого перетворення довгих відео на короткі вертикальні кліпи з використанням сучасних методів штучного інтелекту та їх подальшою публікацією у соціальних мережах.

1 ОГЛЯД ОСНОВНИХ МЕТОДІВ ОБРОБЛЕННЯ ВІДЕО ТА СТВОРЕННЯ КЛІПІВ

1.1 Методи автоматичної транскрипції мовлення

Автоматичне розпізнавання мовлення (ASR – Automatic Speech Recognition) є технологією перетворення аудіопотоку у текстове представлення. Оскільки визначення смислових фрагментів відео безпосередньо залежить від якості транскрипту, цей етап передує всьому подальшому аналізу відеоконтенту.

В еволюції ASR-систем прийнято виділяти три покоління. Перше ґрунтувалося на прихованих марківських моделях (HMM – Hidden Markov Model) у поєднанні з гаусівськими моделями мішанини (GMM – Gaussian Mixture Model). У контрольованих умовах ці підходи забезпечували прийнятну точність розпізнавання, однак при роботі з реальним мовленням – що містить фоновий шум, акценти та варіативні стилі – якість результатів суттєво погіршувалася [1].

Основу другого покоління склали рекурентні нейронні мережі (RNN – Recurrent Neural Network) та архітектура з довгою короткостроковою пам’яттю (LSTM – Long Short-Term Memory). Здатність моделі враховувати контекст попередніх токенів забезпечила помітний приріст точності порівняно з HMM/GMM. Разом із тим послідовна природа обчислень обмежувала швидкість навчання таких архітектур [2].

Третє покоління сформувалося на основі архітектури Transformer із механізмом уваги (Attention), яка на відміну від RNN обробляє всю послідовність паралельно та ефективніше утримує довгостроковий контекст [3]. Одним із найбільш показових представників цього класу є модель Whisper від OpenAI, навчена на понад 680 000 годин різномовного аудіо. Завдяки

цьому вона демонструє стабільні результати розпізнавання для широкого кола акцентів і мов [4].

У розроблюваній платформі застосовується faster-whisper – оптимізована реалізація Whisper із підтримкою квантизації int8, яка зменшує обчислювальні витрати порівняно з оригінальною бібліотекою. Окрім продуктивності, бібліотека надає часові мітки для кожного сегмента мовлення, що використовуються для визначення меж кліпів на наступних етапах обробки. Порівняння підходів до автоматичного розпізнавання мовлення наведено в таблиці та на рисунку (табл. 1.1) (рис. 1.1).

Таблиця 1.1 – Порівняння підходів до автоматичного розпізнавання мовлення

Підхід	**Точність**	**Стійкість до шуму**	**Часові мітки**	**Застосування**
HMM/GMM	Середня	Низька	Обмежені	Застаріле
RNN/LSTM	Висока	Середня	Наявні	Спеціалізоване
Transformer(Whisper)	Дуже висока	Висока	Точні	Загального призначення

З таблиці 1.1 видно, що підхід на базі Transformer перевершує попередні покоління за точністю, стійкістю до шуму та якістю часових міток. Це обґрунтовує його вибір для реалізації модуля транскрипції у розроблюваній платформі.

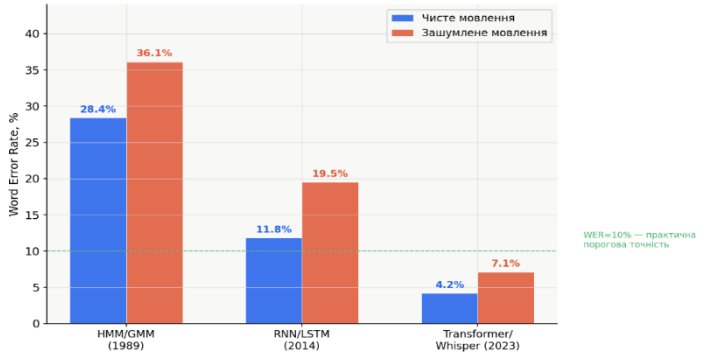

Рисунок 1.1 – Порівняння точності підходів ASR за показником WER

Підтримка часових міток на рівні слів (word timestamps) є однією з ключових характеристик faster-whisper для цього проєкту. Рендеринг кліпів залежить від точності початкових і кінцевих міток сегментів, тому коректна синхронізація транскрипту безпосередньо впливає на якість кінцевого результату [4, 5].

Квантизація вагів до формату int8 знижує вимоги до оперативної пам’яті та прискорює виконання на CPU, оскільки замість стандартного float32 частина обчислень виконується у 8-бітному цілочисельному представленні.Разом із тим ступінь впливу на якість розпізнавання залежить від моделі, мови, якості аудіо та апаратної платформи. З практичної точки зору int8 є розумним компромісом між швидкістю й точністю, але не універсальним рішенням [4, 5].

Модель Whisper навчена одночасно вирішувати чотири задачі, серед яких розпізнавання мовлення, визначення мови, переклад на англійську та ідентифікацію тиші. Навчальний корпус охоплює 680 000 годин різномовного аудіо з інтернету – широке покриття акцентів, діалектів і стилів мовлення. Завдяки цьому модель зберігає прийнятну точність навіть на записах низької якості та в зашумленому середовищі [4].

1.2 Методи автоматизованого виявлення ключових моментів у відео

Після отримання текстового представлення відео виникає задача визначення найбільш цінних фрагментів. Ця задача виявлення ключових моментів (highlight detection) і є ключовою для якості кінцевих кліпів. Від точності її вирішення залежить, наскільки цікавим і повноцінним буде сформований короткий кліп.

У науковій літературі виділяють три основні підходи до вирішення цієї задачі [6–8]:

– сигнальні підходи, що аналізують акустичні та візуальні характеристики відео (гучність, темп мовлення, зміни кадрів) без урахування змісту;

– семантичні підходи, що ґрунтуються на аналізі тексту транскрипту та виявляють ключові слова і теми;

– мультимодальні підходи, що поєднують аудіо, відеоряд і текст для найточнішої оцінки цінності фрагмента.

Порівняння підходів до виявлення ключових моментів у відео наведено у таблиці 1.2 та на рисунку 1.2.

Таблиця 1.2 – Порівняльний аналіз підходів до виявлення ключових моментів

Підхід	**Переваги**	**Недоліки**
Сигнальний	Швидкість, не потребує транскрипту	Ігнорує зміст
Семантичний	Аналізує зміст тексту	Залежить від якості ASR
Мультимодальний	Найвища точність, враховує контекст	Потребує LLM API

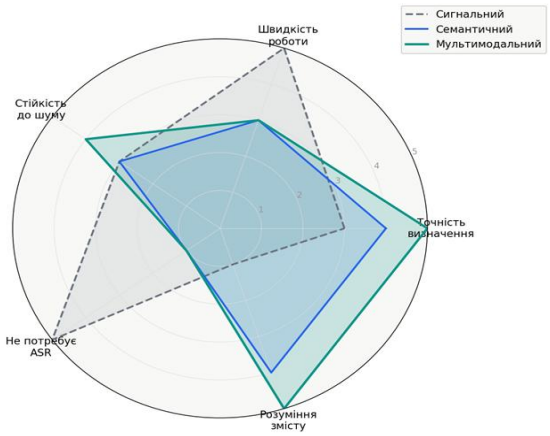

Рисунок 1.2 – Порівняльний аналіз підходів до виявлення ключових моментів

Семантичні методи працюють із текстовим представленням відео, отриманим через ASR. Вони застосовують техніки оброблення природної мови, виявлення ключових слів через TF-IDF, тематичне моделювання та визначення тональності висловлювань. Такі методи добре розуміють зміст, але повністю залежать від якості транскрипції і не враховують візуальну складову відео [6].

Застосування великих мовних моделей (LLM) для аналізу транскриптів є доцільним у задачах відбору ключових фрагментів, оскільки такі моделі враховують не лише окремі ключові слова, а й логічну структуру висловлювання, завершеність думки та зв’язок фрагмента з попереднім контекстом. На відміну від суто сигнальних методів, які спираються на гучність, темп мовлення або зміну кадрів, LLM-підходи дають змогу оцінювати семантичну цінність фрагмента [9].

Окремим напрямом є використання згорткових нейронних мереж (CNN) для аналізу відеокадрів і визначення їх інформативності. Такі підходи дають змогу виявляти візуально значущі моменти, зокрема появу певних об’єктів, зміну сцени або присутність людського обличчя крупним планом. У поєднанні з аналізом транскрипту це формує мультимодальний підхід, у якому візуальні ознаки доповнюються змістовим аналізом мовлення [8].

У межах розроблюваної платформи реалізовано семантичний підхід на основі аналізу транскрипту. Для першої версії системи це рішення виявилося практичнішим за мультимодальний аналіз, оскільки окрема модель комп’ютерного зору не потрібна, тоді як логіка висловлювання, завершеність думки та цінність фрагмента для короткого відео враховуються безпосередньо через текст [9].

Текст транскрипту передається до DeepSeek API – великої мовної моделі з інтерфейсом, сумісним із форматом OpenAI API. Модель визначає фрагменти з найбільшою інформаційною або емоційною цінністю, спираючись на логіку розповіді, наявність завершених думок і кульмінаційних моментів [10].

У результаті роботи модуля аналізу формується структурований набір пропозицій для кліпів:

– часові межі фрагмента (початок і кінець у секундах);

– заголовок кліпу, згенерований моделлю;

– пояснення вибору фрагмента;

– оцінка релевантності (score).

У наслідок цього відбір фрагментів, який раніше виконувався вручну досвідченим відеоредактором, переноситься в автоматизований режим.

Методи штучного інтелекту, комп’ютерного зору та машинного навчання активно досліджуються науковою спільнотою, зокрема представниками кафедри інформатики ХНУРЕ. Значний внесок у розвиток методів класифікації зображень, кластерного аналізу та використання нейронних мереж для опрацювання структурних описів зроблено у працях [8, 11–17].

Перспективними є також дослідження у сфері розпізнавання складних візуальних об’єктів та рукописних символів за допомогою глибинного навчання [18–21], а також застосування великих мовних моделей і методів аналізу даних для інтелектуальних систем [22–24].

Ці фундаментальні підходи формують теоретичну базу для розроблення прикладних мультимодальних систем оброблення медіаконтенту, до яких належить розроблювана вебплатформа.

1.3 Технології оброблення та нарізки відеоматеріалів

1.3.1 Інструменти програмного оброблення відео

Після визначення ключових фрагментів необхідно фізично сформувати відеокліпи. Для цього існують різні інструменти. MoviePy надає зручний Python-інтерфейс, однак суттєво поступається нативним рішенням за швидкістю; OpenCV орієнтований переважно на аналіз зображень і не

призначений для потокового перекодування; FFmpeg є стандартом галузі для командного рядка й забезпечує високу продуктивність завдяки нативній реалізації мовою C [25].

Основним інструментом у розроблюваній платформі є FFmpeg – відкрита кросплатформна бібліотека та утиліта для роботи з відео та аудіо, яка підтримує широкий спектр форматів і кодеків.

FFmpeg дозволяє виконувати:

– нарізку відео за часовими мітками без повного перекодування;

– масштабування та зміну роздільної здатності через граф фільтрів;

– перекодування у стандарти H.264/AAC, що вимагаються платформами соціальних мереж;

– накладання субтитрів форматів ASS та графічних оверлеїв.

Для завантаження відеоматеріалів із зовнішніх платформ у розроблюваній платформі застосовується yt-dlp [26] – активно підтримуваний форк youtube-dl. Інструмент отримує відео у доступній якості разом із метаданими, а саме назвою, тривалістю та описом. Оскільки YouTube використовує механізм підпису URL (signature cipher), yt-dlp залежить від актуальності механізмів розшифрування цих підписів. Для стабільної роботи із сучасними версіями YouTube у проєкті додатково використовується Deno середовище виконання окремих JavaScript-обробників. Крім YouTube, інструмент підтримує велику кількість інших відеоплатформ, що створює потенціал для подальшого розширення джерел відеоконтенту.

1.3.2 Методи формування та оформлення кліпів

При створенні кліпів необхідно враховувати специфічні вимоги платформ короткого відео щодо формату контенту. Стандартом для TikTok, Instagram Reels та YouTube Shorts є вертикальний формат із співвідношенням сторін 9:16 та роздільною здатністю 1080×1920 пікселів [27]. Оскільки

більшість відео записані у горизонтальному форматі (16:9), необхідне перетворення формату.

У розроблюваній платформі реалізовано два методи адаптації горизонтального відео до вертикального формату, які обирає користувач:

– метод обрізки (crop), що вирізає центральну частину кадру з шириною, яка відповідає співвідношенню 9:16; підходить для відео, де об’єкт розташований по центру кадру;

– метод розмитого фону (blur background), за якого горизонтальне відео розміщується по центру вертикального полотна, а порожні бічні смуги заповнюються розмитою копією того самого кадру; що дає змогу адаптувати субтитри до різних стилів короткого відео.

Субтитри у форматі ASS (Advanced SubStation Alpha) використовуються для синхронізованого відображення тексту мовлення поверх відеоряду. Формат підтримує налаштування шрифту, розміру, кольору, обводки, тіні та позиціонування, що дає змогу адаптувати субтитри до різних стилів короткого відео. У платформі файл субтитрів генерується автоматично на основі сегментів транскрипту з часовими мітками, а користувач може обрати один із п’яти стилів оформлення: classic, tiktok, neon, boxed або minimal.

Для додаткового виділення змісту кліпу формується графічний заголовок у вигляді PNG-оверлею за допомогою бібліотеки Pillow. Розміщується у верхній частині кадру та накладається на напівпрозору підкладку для кращої читабельності. Масштабування, обрізка, субтитри та графічні елементи об’єднуються в одному графі фільтрів FFmpeg, що спрощує рендеринг кліпу.

1.4 Архітектурні підходи до побудови вебплатформ

Системи автоматизованого оброблення відео одночасно вирішують обслуговування HTTP-запитів користувачів і виконання ресурсоємних

фонових операцій, зокрема транскрипції, ШІ-аналізу та рендерингу кліпів. Вибір архітектурного підходу до організації цих процесів безпосередньо впливає на продуктивність, надійність і складність подальшої підтримки системи.

Існують два основні підходи до побудови таких систем [28, 29]:

– мікросервісна архітектура, за якої кожен функціональний модуль є незалежним сервісом із власною базою даних; забезпечує горизонтальне масштабування, проте суттєво ускладнює розробку, налагодження та операційну підтримку;

– монолітна архітектура з фоновими задачами, за якої уся бізнес-логіка зосереджена в одному застосунку, а важкі операції виконуються в окремих потоках; простіша у розробленні та розгортанні для проєктів початкового і середнього масштабу.

Для розроблюваної платформи обрано монолітну архітектуру на базі FastAPI [28] із підтримкою фонових процесів через ThreadPoolExecutor. FastAPI забезпечує автоматичну генерацію OpenAPI-документації, вбудовану підтримку асинхронних обробників та розвинену систему ін’єкції залежностей. Основні операції (транскрипція, ШІ-аналіз, нарізка) виконуються асинхронно, тому основний потік не блокується, а інтерфейс зберігає сталий час відгуку.

Особливостями обраної реалізації є:

– виконання обчислювально складних задач у ThreadPoolExecutor з обмеженою кількістю потоків;

– трирівневе збереження стану задач, за якого in-memory кеш використовується для швидкого доступу, Redis як опціональний проміжний рівень, а PostgreSQL залишається джерелом правди;

– формалізований життєвий цикл задачі зі станами: pending → transcribing → processing → completed | failed;

– планувальник відкладених публікацій на базі DB-черги з механізмом SELECT FOR UPDATE SKIP LOCKED для безпечної роботи кількох воркерів.

Система моніторингу стану сервісу реалізована через ендпоінт GET /health, який перевіряє підключення до бази даних, наявність вільного місця на диску та стан фонових воркерів. При розгортанні у Docker цей ендпоінт використовується як HEALTHCHECK команда, і використовується оркестратором для автоматичного перезапуску нездорових інстансів.

Для забезпечення ізоляції середовища виконання платформа розгортається у Docker-контейнері. Образ побудований на базі python:3.11-slim та включає системні залежності: FFmpeg для оброблення відео, Deno 2.1 для підтримки yt-dlp, шрифти Montserrat для генерації оверлеїв. Сервіс запускається від непривілейованого користувача appuser (UID 1000) для мінімізації поверхні атаки [29].

Автоматизацію збірки та тестування реалізовано за допомогою робочих процесів GitHub Actions. Основний робочий процес ci-fast.yml виконується при кожному push та pull request: запускає PostgreSQL container, встановлює залежності, виконує міграції Alembic, запускає pytest та збирає клієнтську частину через npm. Час виконання процесу залежить від обсягу тестів і стану кешу залежностей.

Безпека платформи забезпечується комплексом заходів: OAuth-токени зберігаються у зашифрованому вигляді з використанням алгоритму Fernet, автентифікація реалізована через JWT у httpOnly-cookie із CSRF-захистом за схемою double-submit cookie, а авторизація у соціальних мережах виконується через OAuth 2.0 [30].

Для публікації кліпів у соціальних мережах використовуються офіційні API платформ:

– TikTok Content Posting API для публікації відео у TikTok [31, 32];

– Instagram Graph API v21.0 для публікації Reels через Meta Business Suite [33];

– YouTube Data API v3 для завантаження відео у YouTube Shorts та звичайні відео [34].

Для інтеграції платформи із зовнішніми системами автоматизації реалізовано систему API-ключів формату sk_live_*. Захищені ендпоінти /api/v1/* дозволяють зовнішнім системам (n8n, Zapier) завантажувати відео, отримувати транскрипти та ініціювати створення кліпів без браузерного інтерфейсу [35].

1.5 Постановка задачі

Стрімкий розвиток платформ короткого відео породжує значний попит на інструменти автоматизованого створення контенту. Водночас процес перетворення довгого відеоматеріалу на серію якісних коротких кліпів залишається трудомістким: він включає транскрипцію, пошук ключових моментів, монтаж, додавання субтитрів і публікацію. Виконання цих операцій вручну потребує значних витрат часу, особливо для довгих відеоматеріалів.

Актуальність роботи зумовлена потребою в SaaS-вебплатформі для автоматизованого перетворення довгих відео на короткі вертикальні кліпи з використанням методів штучного інтелекту та подальшою публікацією у соціальних мережах.

Об’єктом роботи є процес автоматизованого оброблення відеоданих та створення коротких відеокліпів для платформ короткого відео.

Метою роботи є розроблення вебплатформи для автоматизованого створення коротких вертикальних відеокліпів. Платформа охоплює повний цикл оброблення: завантаження або імпорт відео з YouTube, автоматичну транскрипцію мовлення, ШІ-визначення ключових фрагментів, рендеринг вертикальних кліпів із субтитрами та їх публікацію у TikTok, Instagram і YouTube.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– проаналізувати сучасні методи автоматичної транскрипції мовлення та обґрунтувати вибір моделі faster-whisper;

– дослідити підходи до автоматизованого виявлення ключових моментів у відео та обґрунтувати застосування LLM-моделі DeepSeek;

– розглянути технології програмного оброблення відео та обрати інструментарій на базі FFmpeg і yt-dlp;

– спроєктувати архітектуру вебплатформи з підтримкою фонових задач, трирівневого кешування та планувальника публікацій;

– реалізувати OAuth 2.0 інтеграцію з TikTok, Instagram та YouTube для автоматичної публікації кліпів;

– розробити систему API-ключів для інтеграції платформи із зовнішніми системами автоматизації (n8n, Zapier);

– провести автоматизоване тестування модулів серверної частини і оцінити працездатність основних сценаріїв оброблення відео.

2 АРХІТЕКТУРА ТА ТЕХНІЧНІ РІШЕННЯ ПЛАТФОРМИ

2.1 Загальна архітектура системи

Платформа реалізована у вигляді монолітної SaaS-архітектури з розподілом на рівні представлення, бізнес-логіки та даних. Це рішення дозволяє зосередити всю бізнес-логіку в одному репозиторії та спростити операційну підтримку без втрати модульності коду.

Система складається з таких основних компонентів:

– серверної частини на FastAPI, яка обробляє HTTP-запити, керує фоновими задачами та передає оновлення стану в реальному часі;

– клієнтської частини на React і TypeScript, яка забезпечує вебінтерфейс для завантаження відео, перегляду транскрипту та керування кліпами;

– медіапайплайну, що відповідає за транскрипцію мовлення, ШІ-аналіз, нарізку відео, формування субтитрів і графічних оверлеїв;

– зберігання даних на PostgreSQL з використанням SQLAlchemy та опціонального Redis-кешу для швидкого доступу до стану задач;

– планувальник публікацій на основі черги в базі даних для відкладеного розміщення кліпів у соціальних мережах;

– системи безпеки, що охоплює автентифікацію, CSRF-захист, шифрування OAuth-токенів і API-ключі для зовнішніх інтеграцій.

Повний перелік компонентів із технологіями та призначенням наведено у таблиці (табл. 2.1).

Таблиця 2.1 – Компоненти архітектури платформи

Компонент	**Технологія**	**Призначення**
1	**2**	**3**
Backend API	FastAPI, Python 3.11	Обробка HTTP-запитів, бізнес-логіка, WebSocket

Продовження таблиці 2.1

1	2	3
База даних	PostgreSQL + SQLAlchemy 2.0	Зберігання бізнес-даних, міграції Alembic
Медіапайплайн	FFmpeg, faster-whisper, Pillow	Транскрипція, нарізка кліпів, субтитри
ШІ-аналіз	DeepSeek API (OpenAI-сумісний)	Семантичний аналіз транскрипту
Кеш стану задач	In-memory L1 + Redis L2 (опційно)	Швидкий доступ до статусу задач
Планувальник	DB-черга + scheduler loop	Відкладена публікація кліпів
Завантажувач	yt-dlp + Deno 2.1	Імпорт відео з YouTube
OAuth-інтеграції	TikTok, Instagram Graph, YouTube Data API	Публікація у соціальних мережах
Білінг	Stripe API + webhook	Підписки, токени, платежі
CI/CD	GitHub Actions + Docker	Автоматичне тестування та розгортання
Frontend SPA	React, TypeScript, Vite	Браузерний інтерфейс користувача

Розгортання платформи виконується у Docker-контейнері на базі образу python:3.11-slim. До образу встановлено системні залежності, зокрема FFmpeg для оброблення відео, Deno 2.1 для підтримки yt-dlp і шрифти Montserrat для генерації оверлеїв. У виробничому середовищі серверна частина запускається через Uvicorn, а Nginx виступає зворотним проксі-сервером.

2.1.1 Об’єктно-орієнтована модель системи

Бізнес-логіка платформи поділена на окремі функціональні напрями, серед яких оброблення медіаданих, автентифікація, білінг і публікація у соціальні мережі. Кожен напрям має власні моделі даних, сервіси та маршрути

API. Такий поділ зменшує зв’язність коду й полегшує тестування окремих частин системи.

Клас User є центральним об’єктом системи. Він зберігає дані користувача, налаштування, стан блокування запису та токени доступу до підключених платформ. Токени соціальних мереж зберігаються у зашифрованому вигляді – шифрування відбувається під час запису, розшифрування під час читання засобами SQLAlchemy. Завдяки цьому бізнес-логіка працює з токенами як зі звичайними полями, тоді як у базі даних вони ніколи не зберігаються у відкритому вигляді.

Клас Task зберігає стан задачі оброблення відео. У ньому фіксуються шлях до вихідного файлу, поточний статус, прогрес виконання, транскрибовані сегменти та сформовані кліпи. Дані зі змінною структурою зберігаються у JSON-полях – це дозволяє розширювати формат результатів без змін схеми бази даних.

Клас TaskStore керує станом задач через трирівневу схему зберігання, що поєднує in-memory кеш для активних задач, Redis як опціональний проміжний рівень при масштабуванні та PostgreSQL як основне джерело правди. У результаті прогрес швидко оновлюється в інтерфейсі, а стан задач відновлюється після перезапуску сервера (рис. 2.1).

Сервіс TokenService керує балансом токенів користувача. Він виконує атомарне списання перед запуском платних операцій, обробляє щомісячне поповнення та враховує додатково придбані токени. Винесення логіки в окремий сервіс виключає можливість ручного списання в обхід єдиного механізму та знижує ризик помилок у розрахунках.

Разом ці класи формують основу доменної моделі платформи. User відповідає за користувача та його доступи, Task описує одиницю оброблення відео, TaskStore забезпечує узгоджене збереження стану, а TokenService контролює використання платних ресурсів.Такий розподіл відповідальності спрощує супровід коду та дозволяє розширювати окремі компоненти без порушення зв’язків між ними.

Рисунок 2.1 – Тришарова стратегія кешування стану задач

2.1.2 Взаємодія компонентів та потоки даних

Для розуміння роботи платформи розглянемо послідовність передавання даних між її основними компонентами. Взаємодія між клієнтською частиною, сервером, базою даних, медіапайплайном і зовнішніми сервісами відбувається через визначені інтерфейси, які обмежують залежність між модулями та спрощують тестування окремих частин системи.

У сценарії завантаження файлу користувач передає відео через вебінтерфейс. Сервер перевіряє тип, розмір і сигнатуру файлу, зберігає його у сховище вихідних відео та створює запис задачі в базі даних. Стан задачі фіксується у сховищі задач, після чого фоновий обробник послідовно виконує транскрипцію мовлення, ШІ-аналіз транскрипту та рендеринг кліпів. Зміни стану передаються клієнтській частині в реальному часі, тому користувач бачить прогрес без перезавантаження сторінки.

У сценарії імпорту з YouTube перед основним пайплайном додається етап завантаження відео. Система перевіряє належність посилання до

дозволених доменів YouTube, завантажує відеофайл у заданій якості та передає його до того самого процесу транскрипції, аналізу й нарізки.

Публікація кліпу виконується після вибору платформи користувачем. Сервер перевіряє право доступу до кліпу, розшифровує OAuth-токен, атомарно списує токени за операцію та передає файл до відповідного модуля публікації. Після успішного завантаження зовнішня платформа повертає ідентифікатор публікації, який зберігається в базі даних разом зі станом операції.

Для підвищення надійності зовнішні запити до сервісів штучного інтелекту, соціальних мереж і платіжної системи обробляються з урахуванням можливих тимчасових збоїв. Недоступність сервісу аналізу транскрипту не зупиняє весь процес – користувач отримує транскрипт і може вручну скоригувати межі кліпів.

Оновлення стану задачі передається через постійне двостороннє з’єднання між клієнтом і сервером. Після підключення клієнт отримує поточний стан задачі та повідомлення про кожну подальшу зміну прогресу. Закриття вкладки браузера не перериває оброблення – задача продовжує виконуватися у фоновому режимі, а її фінальний стан зберігається в базі даних.

2.2 Медіапайплайн оброблення відео

Оброблення відео виконується у фоновому режимі, щоб тривалі операції транскрипції, аналізу та рендерингу не блокували роботу вебінтерфейсу. Сервер використовує пул потоків з обмеженою кількістю воркерів, а кожна задача проходить набір визначених станів. Це дозволяє відстежувати прогрес, фіксувати помилки та відновлювати стан задачі після збою.

Послідовність станів задачі при сценарії завантаження файлу показано на рисунку 2.2:

– pending – задача створена, вхідний файл перевірено за типом, розміром і сигнатурою та збережено у сховище вихідних відео;

– transcribing – виконується автоматична транскрипція мовлення, а проміжний прогрес передається користувачу через інтерфейс;

– processing – система аналізує транскрипт, визначає потенційні межі кліпів і запускає рендеринг відеофрагментів;

– completed | failed – задача переходить у фінальний стан: у разі успіху користувач отримує готові кліпи, у разі помилки система зберігає причину збою.

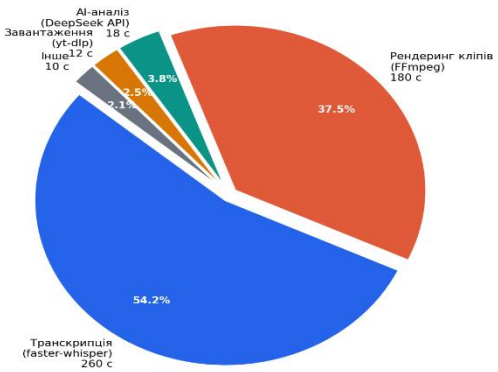

Рисунок 2.2 – Розподіл часу оброблення відео тривалістю 60 хв

Поділ пайплайну на послідовні етапи спрощує контроль помилок і збереження проміжних результатів. У разі збою система фіксує причину у стані задачі, тоді як інші задачі продовжують виконуватися незалежно. Користувач завдяки цьому бачить прогрес у реальному часі, тоді як серверна частина продовжує обробляти HTTP-запити без блокування.

2.2.1 Математична модель визначення меж кліпів

Алгоритм визначення меж кліпів працює з набором транскрибованих сегментів, кожен з яких містить час початку, час завершення та текстовий

фрагмент мовлення. Для заданих меж кліпу система відбирає сегменти, що перетинаються з вибраним часовим інтервалом.

Для заданих меж кліпу (start, end) з набору S відбираються всі сегменти, що задовольняють умову:

$$t^s_i < \text{end} \wedge t^e_i > \text{start}, \tag{2.1}$$

де t^s_i – час початку і-го сегмента;

t^e_i – час завершення і-го сегмента;

start – початкова межа кліпу;

end – кінцева межа кліпу.

Умова (2.1) означає, що до кліпу потрапляють усі сегменти, які повністю або частково перетинаються з вибраним часовим інтервалом. Якщо сегмент лише частково перетинає межі кліпу, його часові мітки обрізаються до інтервалу [start; end]. Після цього тексти відібраних сегментів об’єднуються у хронологічному порядку й використовуються для формування субтитрів та заголовка кліпу.

Тривалість кліпу визначається за формулою:

$$d = \text{end} - \text{start}, \tag{2.2}$$

де d – тривалість кліпу.

Отримане значення перевіряється валідатором. Для ручних кліпів мінімальна тривалість становить 0,5 с, а для ШІ-запропонованих фрагментів застосовується діапазон від 15 с до 180 с. Кліп із межами, що не відповідають обмеженням, відхиляється до потрапляння задачі в чергу рендерингу.

Модель DeepSeek повертає набір пропозицій кліпів у структурованому форматі, що містить початок і кінець фрагмента, заголовок, пояснення вибору та оцінку релевантності. Після валідації некоректні фрагменти відкидаються, решта сортується за оцінкою та передається до інтерфейсу користувача.

2.2.2 Модуль рендерингу кліпів

Рендеринг кліпів виконується на основі визначених часових меж. Система передає до FFmpeg початковий час, тривалість кліпу, режим адаптації кадру, параметри субтитрів і заголовок. Після чого на основі цих параметрів формується команда рендерингу, яка виконує обрізання відео, масштабування, накладання субтитрів і фінальне кодування.

Для вертикального формату 1080×1920 пікселів підтримуються два режими адаптації горизонтального відео. У режимі розмитого фону оригінальний кадр розміщується по центру вертикального полотна, а бічні області заповнюються розмитою копією того самого відео. Цей режим зберігає повну композицію кадру та підходить для матеріалів, де важливі об’єкти розташовані не лише в центрі. Альтернативним є режим обрізання, який відсікає бічні частини до пропорції 9:16. Він підходить для матеріалів, де головний об’єкт зосереджений у центрі кадру, проте крайні частини зображення до фінального кліпу не потрапляють.

Субтитри накладаються у форматі ASS, що підтримує налаштування шрифту, кольору, обводки, тіні та позиціонування. Заголовок кліпу формується як графічний PNG-оверлей і додається поверх відео під час фінального рендерингу. Це дає змогу поєднати в одному етапі обрізання, масштабування, субтитри, заголовок і кодування у формат MP4 з відеокодеком H.264 та аудіокодеком AAC.

2.2.3 Схема бази даних

База даних платформи реалізована на PostgreSQL з використанням ORM SQLAlchemy. Вона зберігає облікові записи користувачів, задачі оброблення відео, результати транскрипції, сформовані кліпи, токенний баланс, підписки, API-ключі та записи відкладених публікацій. PostgreSQL є основним

сховищем усіх довготривалих станів системи – кеші застосовуються лише для пришвидшення доступу до активних задач.

Схема містить 16 таблиць. При старті серверна частина виконує перевірку актуальності схеми через механізм міграцій Alembic, якщо фактична ревізія бази даних не відповідає очікуваній, запуск сервісу зупиняється. Це виключає роботу застосунку з несумісною структурою таблиць.

Схема охоплює не лише задачі оброблення відео, а й авторизацію, білінг, публікації та зовнішні інтеграції, тому логічну структуру бази даних доцільно розглянути через основні сутності. Ключові таблиці наведено у таблиці 2.2.

Таблиця 2.2 – Ключові таблиці бази даних платформи

Таблиця	**Ключові поля**	**Призначення**
users	id, email (unique), password_hash, google_id, encrypted OAuth-токени	Облікові записи та OAuth-токени
tasks	task_id (UUID), user_id FK, status, clips JSON, segments JSON, video_path	Задачі оброблення відео зі станом
token_balances	user_id (unique), available, used, purchased, reset_date	Баланс токенів з атомарним списанням
subscriptions	user_id, plan_id, status, interval, trial_expires_at	Підписки з підтримкою trial та Stripe
scheduled_publishes	user_id, task_id, clip_index, platform, scheduled_at, status	Черга відкладених публікацій
api_keys	user_id FK, api_key_hash (unique), key_prefix, is_active	API-ключі для n8n та зовнішніх систем
clip_analytics	task_id FK, clip_index, event, platform, user_id	Аналітика завантажень і публікацій
webhook_delivery_logs	user_id, task_id, url, success, attempt, response_ms	Журнал доставок user webhook

Центральними сутностями схеми є users і tasks, де користувач володіє задачами оброблення відео, токенним балансом, підписками, API-ключами та записами публікацій. Решта таблиць деталізує окремі підсистеми – білінг, інтеграції, аналітику та доставку webhook-сповіщень.

OAuth-токени соціальних мереж зберігаються у зашифрованому вигляді. Шифрування виконується на рівні моделі користувача: під час запису токен перетворюється алгоритмом Fernet у зашифрований рядок, а під час читання автоматично розшифровується для бізнес-логіки. Загальну схему шифрування OAuth-токенів наведено на рисунку 2.3.

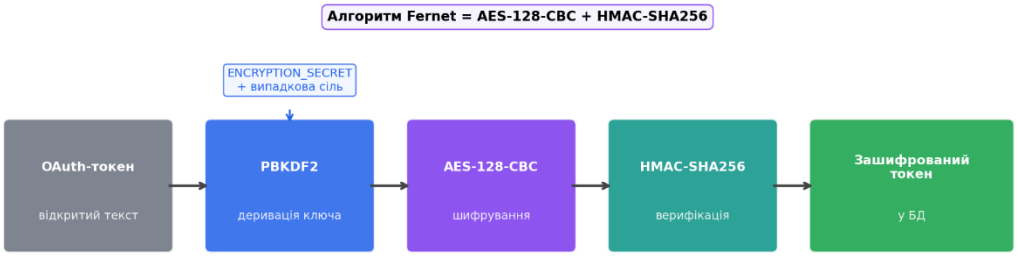

Рисунок 2.3 – Схема шифрування OAuth-токенів алгоритмом Fernet

У такій моделі токени не зберігаються у відкритому вигляді в таблиці users. Разом із тим ключ шифрування має залишатися стабільним – його зміна без попередньої міграції токенів унеможливить розшифрування вже підключених акаунтів.

2.2.4 Алгоритм атомарного списання токенів

Система токенів контролює використання ресурсоємних операцій платформи, зокрема транскрипції, ШІ-аналізу, рендерингу кліпів і автопублікації. Вона пов’язує доступний функціонал із тарифним планом користувача. Для користувачів без активної підписки передбачено безкоштовний місячний ліміт у 50 токенів.

Вартість операцій задається у сервісі token_service.py. Генерація кліпу коштує 5 токенів, ШІ-визначення ключових моментів – 10 токенів, генерація субтитрів – 2 токени, а автопублікація на одну платформу – 3.

Ключовою вимогою до токенної системи є захист від від'ємного балансу при паралельних запитах. Якщо перевіряти баланс окремим запитом, а потім окремо оновлювати, між цими операціями виникає race condition – наприклад, коли користувач одночасно запускає кілька операцій створення кліпів або публікацій. Тому списання реалізовано атомарно одним SQL-запитом, який одночасно перевіряє достатність балансу та виконує оновлення, як показано у лістингу 2.1.

Лістинг 2.1 SQL-запит атомарного списання токенів:

```
UPDATE token_balances
SET available = available - :cost,
    used = used + :cost
WHERE user_id = :user_id
  AND available >= :cost
```

Умова available >= :cost гарантує, що баланс змінюється лише за наявності достатньої кількості токенів. Якщо жоден рядок не було оновлено, сервіс повертає помилку HTTP 402, що означає недостатній баланс. Після успішного списання у таблиці token_transactions створюється запис із типом операції, кількістю списаних токенів і часом виконання.

Окремо система підтримує щомісячне оновлення балансу. Планові токени нараховуються заново, використані скидаються, а придбані токени зберігаються окремо і під час скидання не втрачаються. Завдяки цьому користувач не втрачає придбані пакети, а платформа зберігає прозорий журнал усіх змін балансу.

2.2.5 Механізм планувальника публікацій

Планувальник реалізовано як фоновий цикл у серверному сервісі. Він періодично перевіряє таблицю scheduled_publishes і вибирає задачі, час виконання яких настав. Для запобігання одночасній обробці однієї публікації кількома воркерами під час вибірки застосовується блокування рядків у базі даних. За один цикл планувальник бере обмежену кількість задач, і не перевантажує зовнішні API.

Алгоритм роботи планувальника показано на рисунку 2.4 і складається з таких кроків:

– вибір задач зі статусом pending, для яких scheduled_at менше або дорівнює поточному часу;

– блокування вибраних рядків у базі даних для захисту від подвійної публікації;

– переведення задачі у статус running;

– визначення цільової платформи та виклик відповідного модуля публікації;

– збереження результату публікації у полі result_id;

– переведення задачі у статус done або failed залежно від результату виконання.

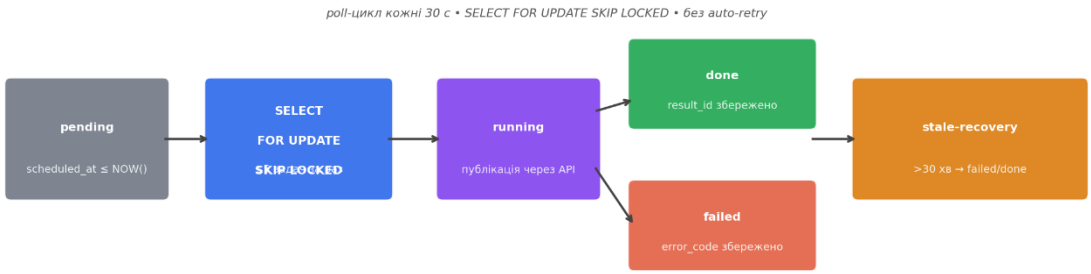

Рисунок 2.4 – Алгоритм роботи планувальника публікацій

Додатково реалізовано механізм відновлення завислих задач. Якщо задача тривалий час залишається у статусі running, планувальник перевіряє її результат і переводить у фінальний стан. Це виключає ситуацію, коли збій процесу або помилка зовнішнього API залишає публікацію у проміжному стані.

Стан кожної запланованої публікації зберігається у базі даних, тому планувальник не втрачає чергу після перезапуску сервісу. Невиконані задачі залишаються у статусі pending і обробляються після наступного запуску. На відміну від черги в оперативній пам’яті, де всі заплановані дії втрачаються після збою, такий підхід гарантує збереження стану незалежно від обставин зупинки сервісу.

Автоматичні повторні публікації після помилки не виконуються. Такий підхід обрано свідомо, оскільки повторний запит до соціальної мережі може призвести до дублювання контенту. У разі невдачі користувач отримує повідомлення про помилку та може створити нову заплановану публікацію вручну.

2.2.6 Підсистема безпеки

Підсистема безпеки платформи охоплює автентифікацію користувачів, захист сесій, контроль доступу до файлів, безпечне зберігання токенів і перевірку зовнішніх інтеграцій. Оскільки система працює з відеофайлами, OAuth-токенами соціальних мереж, API-ключами та платіжними подіями, захист реалізовано на кількох рівнях.

Автентифікація користувачів виконується через JWT-токен, підписаний алгоритмом HS256. Токен зберігається у cookie з прапорцем httpOnly, тому клієнтський JavaScript не має прямого доступу до його значення. Для запитів, що змінюють стан системи, додатково застосовується CSRF-захист за схемою

double-submit cookie [35]: сервер встановлює csrf_token, а клієнт передає це значення у заголовку X-CSRF-Token.

Паролі зберігаються у вигляді bcrypt-хешу з випадковою сіллю та не потрапляють до бази даних. Під час входу система порівнює введений пароль із наявним хешем. Для зменшення ризику перебору паролів передбачено облік невдалих спроб входу та тимчасове блокування облікового запису після перевищення допустимого порогу.

Окремий рівень захисту стосується роботи з файлами. Перед обробленням відео перевіряється не лише розширення файлу, а й вміст файлу за magic bytes. При видачі готових кліпів використовується функція resolve_clip_path(), яка перевіряє, що запитаний файл належить до директорії CLIPS_DIR. Це захищає систему від path traversal, коли користувач намагається отримати доступ до файлів поза дозволеним каталогом.

Для інтеграції із зовнішніми системами платформа використовує API-ключі. Сирий ключ показується користувачу лише один раз під час створення, а в базі даних зберігається його SHA-256 хеш і короткий префікс для відображення в інтерфейсі. Навіть при доступі до таблиці api_keys отримати готовий ключ авторизації з хешу неможливо.

Додатково серверна частина додає до відповідей security headers: X-Frame-Options, X-Content-Type-Options, Content-Security-Policy, Strict-Transport-Security, Referrer-Policy та Permissions-Policy. Вони обмежують ризик clickjacking, MIME-sniffing, небажаного завантаження зовнішніх ресурсів і доступу сторінки до чутливих можливостей браузера.

Інтеграції з TikTok, Instagram і YouTube захищаються через параметри стану. На початку авторизації система генерує випадкове значення стану, зберігає його у відповідній таблиці та звіряє під час запиту зворотного виклику. Це унеможливлює підміну авторизаційного потоку та CSRF-атаки під час підключення зовнішніх акаунтів [34, 35].

2.2.7 Система API-ключів та інтеграція з n8n

Для зовнішніх автоматизацій платформа надає окремий API під префіксом /api/v1 для сценаріїв, де оброблення відео запускається не з вебінтерфейсу, а із зовнішньої системи, наприклад n8n, Zapier, CRM або власного серверного сервісу. Автентифікація виконується через API-ключ у заголовку Authorization, і підтримує міжсервісні запити без браузерної сесії.

API-ключ створюється в інтерфейсі користувача і показується лише один раз. У базі даних зберігається SHA-256 хеш ключа, короткий префікс для відображення, назва, дата останнього використання, термін дії та ознака активності. Відкликання ключа не видаляє запис фізично, він переводиться у неактивний стан, що зберігає історію інтеграції та блокує подальші запити.

Для зовнішніх систем доступні такі основні ендпоінти:

– ендпоінт POST /api/v1/upload для завантаження відеофайлу й створення задачі оброблення;

– ендпоінт GET /api/v1/transcribe/{task_id} для отримання результатів транскрипції;

– ендпоінт POST /api/v1/clips/create для створення кліпів за заданими часовими межами.

Типовий сценарій автоматизації через n8n складається з кількох кроків. Вузол HTTP Request надсилає відеофайл до /api/v1/upload разом з API-ключем, після завершення транскрипції зовнішня система отримує результат через /api/v1/transcribe/{task_id}, а запит до /api/v1/clips/create передає часові межі кліпів і запускає рендеринг. Весь процес від завантаження відео до створення готових кліпів виконується без ручної роботи у вебінтерфейсі.

Webhook-сповіщення доповнюють API-інтеграцію. Замість постійного опитування статусу задачі зовнішня система отримує повідомлення після завершення оброблення. Це зменшує кількість зайвих запитів до серверної частини і спрощує побудову автоматичних сценаріїв, де наступна дія запускається одразу після завершення попередньої.

2.3 Інтеграція з соціальними мережами та система публікацій

Система публікацій є одним із ключових функціональних модулів платформи, що забезпечує передавання готових кліпів до зовнішніх соціальних мереж. У межах роботи реалізовано інтеграцію з TikTok, Instagram та YouTube – основними платформами поширення коротких вертикальних відео.

Архітектура підсистеми побудована за модульним принципом. Для кожної платформи виділено окремі компоненти авторизації та публікації: OAuth-модуль керує підключенням акаунта й токенами, publish-модуль виконує завантаження відео через API відповідної платформи. Такий поділ дозволяє змінювати авторизаційний механізм незалежно від логіки публікації.

Сценарій роботи для користувача єдиний незалежно від вибраної платформи і включає підключення акаунта на сторінці /social, вибір кліпу, задання параметрів публікації та запуск розміщення одразу або через планувальник. Відмінності між TikTok, Instagram і YouTube приховані всередині платформо-специфічних адаптерів.

2.3.1 OAuth 2.0 авторизація та управління токенами

Підключення соціальних мереж реалізовано через OAuth 2.0 [34]. Цей механізм надає платформі обмежений доступ до акаунта користувача без передавання логіна й пароля. Для системи публікацій це принципово, оскільки відео розміщується від імені конкретного користувача, а не від імені застосунку загалом.

Загальна послідовність авторизації однакова для TikTok, Instagram та YouTube. Користувач натискає кнопку підключення платформи на сторінці /social, після чого серверна частина формує посилання авторизації з ідентифікатором застосунку, адресою повернення, переліком дозволів і

параметром стану. Параметр стану генерується випадково і зберігається у базі даних для перевірки належності відповіді зворотного виклику до поточної авторизаційної сесії.

Після підтвердження дозволів соціальна мережа повертає користувача на ендпоінт зворотного виклику разом з кодом авторизації та значенням стану. Серверна частина перевіряє значення стану, обмінює код авторизації на токен доступу і за потреби зберігає токен оновлення або довготривалий токен доступу. Отримані токени записуються до профілю користувача у зашифрованому вигляді.

Кожна платформа має власні вимоги до дозволів і життєвого циклу токенів. TikTok використовує окремі області дозволів для роботи з профілем і публікації відео, Instagram працює через Meta Graph API з довготривалим токеном доступу, а YouTube використовує Google OAuth із доступом до YouTube Data API. Порівняльну характеристику OAuth-параметрів наведено у таблиці 2.3.

Таблиця 2.3 – Порівняльна характеристика OAuth-параметрів платформ

Параметр	**TikTok**	**Instagram**	**YouTube**
1	**2**	**3**	**4**
Протокол	OAuth 2.0 Authorization Code Flow	OAuth 2.0 Authorization Code Flow	OAuth 2.0 Authorization Code Flow
Основне призначення	Публікація відео у TikTok	Публікація Reels через Instagram Graph API	Завантаження відео через YouTube Data API
Основні дозволи	user.info.basic, user.info.profile, video.list, video.publish	instagram_business_basic, instagram_content_publish	youtube.upload, userinfo.email, userinfo.profile
Механізм захисту callback	state у таблиці tiktok_oauth_states	state у таблиці instagram_oauth_states	state для Google OAuth-сесії

Продовження таблиці 2.3

1	2	3	4
Тип токена	access token + refresh token	long-lived access token	access token + refresh token
Оновлення доступу	через refresh token	через refresh long-lived token	через Google refresh token
Бібліотека / клієнт	httpx	httpx	google-api-python-client

Термін дії токенів перевіряється перед кожною операцією, що потребує доступу до зовнішньої платформи. Чинний токен доступу використовується без повторної авторизації. Якщо термін дії наближається до завершення, серверна частина оновлює токен через відповідний механізм платформи. У разі неможливості оновлення користувач має повторно підключити акаунт на сторінці /social.

Такий підхід розділяє відповідальність між модулями. Авторизаційний модуль готує чинний токен доступу, а модуль публікації отримує його вже готовим і не відповідає за повний сценарій авторизації. Зміни в правилах авторизації конкретної платформи вносяться в один модуль без втручання в логіку завантаження відео.

2.3.2 Реалізація публікації у TikTok

Публікація відео у TikTok виконується через TikTok Content Posting API за підходом Direct Post. Серверна частина ініціалізує завантаження, отримує адресу завантаження, передає відеофайл до TikTok і зберігає ідентифікатор публікації для подальшої перевірки статусу.

Перед початком публікації система отримує чинний токен доступу користувача. За його відсутності або неможливості оновлення публікація не запускається, користувач має повторно підключити акаунт TikTok на сторінці

/social. Це запобігає ситуації, коли відео потрапляє у чергу публікації без дійсного доступу до зовнішньої платформи.

Процес публікації у TikTok складається з таких етапів:

– перевірка наявності відеофайлу та визначення його розміру;

– формування параметрів публікації: назва, опис, рівень приватності, обмеження коментарів, duet і stitch;

– ініціалізація завантаження через TikTok API;

– отримання URL-адреса завантаження та publish_id;

– передавання відеофайлу на адресу URL завантаження частинами;

– збереження publish_id для подальшої перевірки результату.

Для великих файлів передбачено завантаження частинами. Серверна частина обчислює розмір фрагмента, кількість частин і для кожного запиту передає заголовки Content-Length та Content-Range. Завдяки цьому система працює не лише з короткими файлами, а й з більшими відео, не завантажуючи весь файл одним запитом.

Після успішного передавання файлу TikTok не гарантує миттєву появу публікації, відео переходить у стан оброблення на стороні платформи, а серверна частина отримує publish_id. Поточний стан перевіряється окремим запитом до відповідного endpoint, який повертає поточний стан публікації: оброблення, успішне завершення або помилку. Технічні параметри публікації у TikTok наведено у таблиці 2.4 та на рисунку 2.5.

Таблиця 2.4 – Технічні вимоги TikTok Content Posting API

Параметр	**Значення**	**Примітка**
1	**2**	**3**
Максимальний розмір файлу	4 ГБ	Для кліпів 15–180 с не є обмеженням
Максимальна тривалість	10 хвилин	Кліпи платформи: 15–180 с
Підтримувані формати	MP4, WebM	Рекомендовано MP4 H.264/AAC

Продовження таблиці 2.4

1	2	3
Метод завантаження	FILE_UPLOAD(PUT на upload_url)	Ініціалізація через POST /init
Час очікування статусу	до 60 секунд	Опитування кожні 5 секунд
Ендпоінт ініціалізації	/v2/post/publish/video/init/	Direct Post method
Ендпоінт перевірки статусу	/v2/post/publish/status/fetch/	Повертає PROCESSING / COMPLETE / FAILED

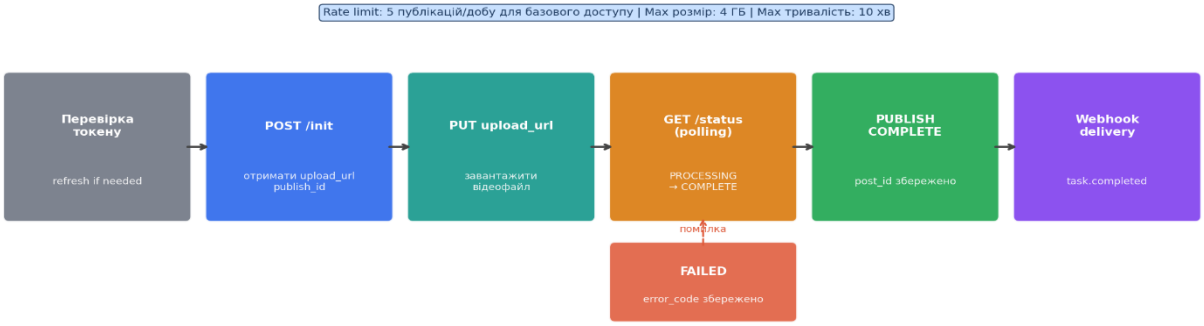

Рисунок 2.5 – Послідовність публікації відео у TikTok

Інтеграція з TikTok побудована так, що передавання відеофайлу та перевірка результату публікації виконуються як два окремі етапи. Серверна частина відповідає за підготовку файлу, авторизацію користувача та передавання відео через адресу завантаження, тоді як остаточне оброблення й публікація виконуються на стороні TikTok. Отриманий publish_id використовується для подальшого відстеження стану публікації та збереження результату в системі.

2.3.3 Реалізація публікації у Instagram та YouTube

Публікація в Instagram реалізована через Instagram Graph API. На відміну від TikTok, який приймає файл через URL-адресу завантаження, Instagram

вимагає передати публічно доступне посилання на відео. Серверна частина формує URL готового кліпу через публічний маршрут платформи та передає його до Instagram Graph API.

Процес публікації Reels складається з двох етапів. На першому серверна частина створює медіаконтейнер із параметрами media_type=REELS, video_url і caption, у відповідь Instagram повертає ідентифікатор підготовленого контейнера. На другому етапі серверна частина викликає media_publish із цим ідентифікатором для фінального розміщення відео в акаунті користувача.

Перед виконанням публікації система отримує чинний токен доступу Instagram. За його відсутності або неможливості оновлення користувач має повторно підключити акаунт на сторінці /social. Публікація без дійсного доступу до зовнішньої платформи в такому разі не запускається.

Публікація у YouTube реалізована через YouTube Data API v3 [32]. Для завантаження відео використовується бібліотека google-api-python-client, яка формує об’єкт MediaFileUpload для локального MP4-файлу. Разом із файлом передаються метадані: назва, опис, теги, категорія та параметр приватності.

Перед завантаженням серверна частина отримує облікові дані користувача. Якщо токен доступу прострочений, система оновлює його через токен оновлення Google та зберігає нове значення у профілі користувача. Після успішного завантаження YouTube повертає video_id, на основі якого формується URL опублікованого відео.

Отже, Instagram і YouTube використовують різні моделі передавання відео, Instagram працює з публічним посиланням на готовий кліп, а YouTube приймає локальний MP4-файл через API-клієнт. Через це в серверній частині реалізовано окремі модулі публікації. Порівняльну характеристику API публікацій наведено у таблиці 2.5. та на рисунку 2.6.

Таблиця 2.5 – Порівняльна характеристика API публікацій

Критерій	**TikTok**	**Instagram**	**YouTube**
Метод передавання	Ініціалізація URL-адреси завантаження і PUT-завантаження файлу	Передавання публічного URL кліпу до медіаконтейнера	Завантаження локального MP4-файлу через API-клієнт
Основні етапи	init → upload file → status check	медіаконтейнер → media_publish	metadata → MediaFileUpload → videos().insert()
OAuth-доступ	TikTok access token	Instagram access token	Google OAuth credentials
Оновлення токена	Через refresh token	Через long-lived token refresh	Через Google refresh token
Результат	publish_id, status	media_id, status	video_id, URL, status
Клієнтська бібліотека	httpx	httpx	google-api-python-client
API	TikTok Content Posting API	Instagram Graph API	YouTube Data API v3

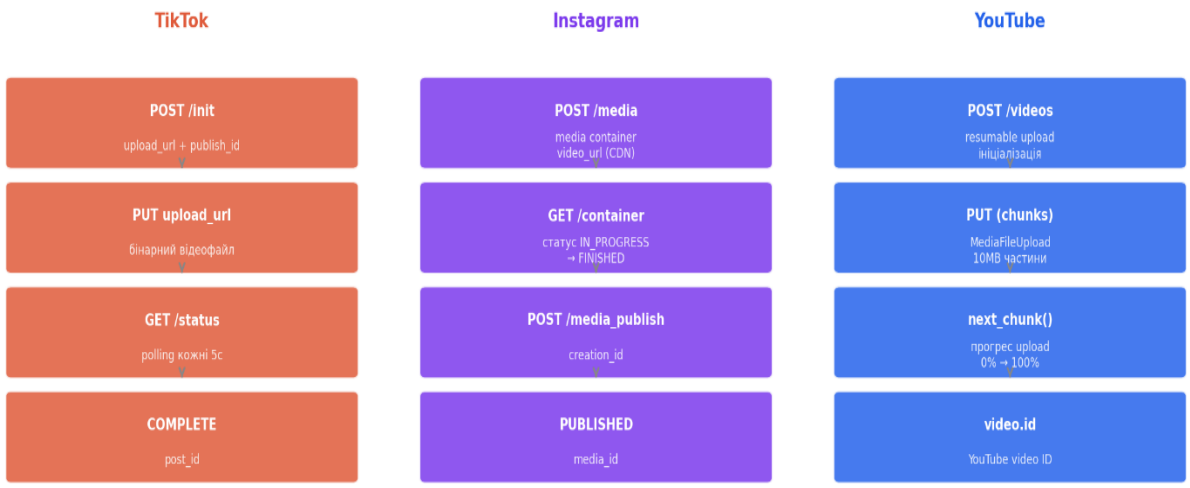

Рисунок 2.6 – Порівняння процесів публікації: TikTok, Instagram, YouTube

2.3.4 Система webhook-сповіщень

Webhook-сповіщення використовуються для інтеграції платформи із зовнішніми системами автоматизації. Якщо користувач передає webhook-адресу під час створення задачі, серверна частина надсилає POST-запит після завершення оброблення відео. Це дозволяє n8n, Zapier або власному серверному сервісі отримати результат без постійного опитування статусу.

На відміну від підходу з опитуванням, webhook працює за подієвою моделлю. Зовнішня система не перевіряє стан задачі кожні кілька секунд, а очікує повідомлення від вебплатформи. Подієва модель скорочує кількість зайвих HTTP-запитів і спрощує побудову автоматичних сценаріїв, оскільки після отримання webhook можна одразу запускати наступний крок, наприклад збереження кліпів, надсилання повідомлення або створення публікації.

Перед збереженням webhook-адреси система виконує її валідацію. Дозволені лише HTTPS-адреси, тоді як адреси localhost, внутрішніх доменів, приватних IP-діапазонів, link-local адрес і хмарних ендпоінтів метаданих блокуються. Така перевірка зменшує ризик SSRF-атаки, за якої користувач міг би змусити серверну частину надсилати запити до внутрішніх ресурсів інфраструктури.

Доставка webhook виконується у фоновому потоці, щоб зовнішній сервіс не блокував основний пайплайн оброблення відео. Для кожного запиту формується JSON-payload із подією, ідентифікатором задачі, статусом, списком готових кліпів та повідомленням про помилку.

Для підвищення надійності передбачено повторні спроби доставки з експоненційною затримкою. Якщо зовнішній сервіс тимчасово недоступний або повертає помилку, серверна частина повторює надсилання кілька разів. Результат кожної спроби фіксується у таблиці webhook_delivery_logs. Структуру задачі наведено у таблиці 2.6 та на рисунку 2.7.

Таблиця 2.6 – Структура webhook-payload для події task.completed

Поле	Тип	Опис
task_id	string	Ідентифікатор задачі оброблення відео
status	string	Фінальний стан задачі: completed або failed
clips	array	Список URL готових кліпів; при помилці може бути порожнім
error	string|null	Повідомлення про помилку; null при успішному завершенні
event	string	Назва події; для завершення задачі використовується task_completed

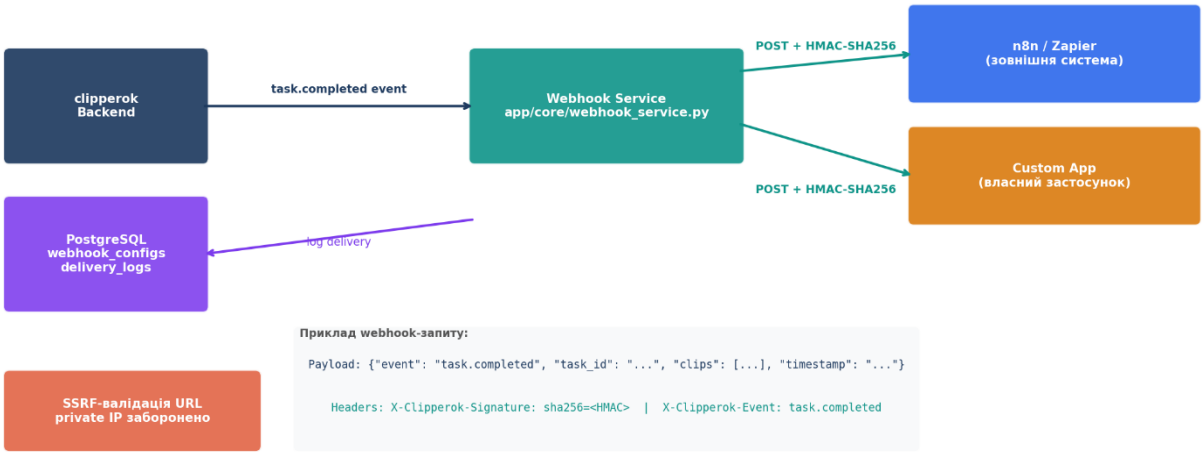

Рисунок 2.7 – Схема роботи webhook-системи платформи

Webhook-система подає завершення оброблення відео як подію, доступну для зовнішніх автоматизацій. Користувач запускає задачу один раз, а подальші дії після її завершення можуть виконуватися автоматично в іншій системі.

2.3.5 Оброблення помилок під час публікації

Публікація у соціальні мережі залежить від зовнішніх API, тому не є повністю контрольованою операцією. Помилки можуть виникати через

недійсний токен доступу, недостатні дозволи, перевищення лімітів платформи, недоступність зовнішнього сервісу або некоректні параметри відео.

Помилки поділяються на тимчасові та постійні. Тимчасові помилки пов’язані з мережею, timeout або короткочасною недоступністю API; для таких випадків доцільні повторні спроби. Постійні помилки виникають тоді, коли повтор без зміни умов не допоможе. Наприклад, якщо користувач відкликав доступ, прострочений токен без можливості оновлення або відео, що не відповідає вимогам платформи.

Для миттєвих публікацій серверна частина повертає повідомлення про помилку, щоб користувач міг повторно підключити акаунт, змінити параметри відео або запустити публікацію знову. Для відкладених публікацій результат фіксується у таблиці scheduled_publishes: успішна задача переходить у статус done, а невдала – у статус failed із текстом помилки.

Автоматичне повторне розміщення після переходу у статус failed не виконується. Це зроблено для запобігання дублюванню контенту, якщо зовнішня платформа прийняла відео, але відповідь не була коректно отримана серверною частиною, повторна публікація може створити два однакові дописи. Тому рішення про повторну публікацію залишається за користувачем. Помилка не приховується, задача не залишається у проміжному стані, а користувач отримує зрозумілий результат і може самостійно прийняти рішення щодо подальших дій.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ПЛАТФОРМИ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Середовище розроблення та конфігурація застосунку

У цьому розділі розглянуто програмну реалізацію вебплатформи clipperok та результати її перевірки. Основну увагу приділено практичному налаштуванню середовища виконання, реалізації медіапайплайну, підсистеми публікації, автентифікації, токенної моделі, а також аналізу результатів тестування і продуктивності платформи.

Конфігурація серверної частини відокремлює код застосунку від параметрів розгортання: адреси бази даних, секретів, ключів зовнішніх API, параметрів медіаоброблення та лімітів фонових задач. Основні параметри конфігурації наведено у лістингу 3.1.

Лістинг 3.1 Основні параметри конфігурації серверної частини:

```
# База даних і публічна адреса
DATABASE_URL=postgresql://user:pass@localhost:5432/clipperok
PUBLIC_BASE_URL=https://example.com

# Безпека
JWT_SECRET=<мінімум 32 символи>
ENCRYPTION_SECRET=<ключ шифрування OAuth-токенів>

# ШІ-аналіз
DEEPSEEK_API_KEY=sk-...
AUTO_CLIP_ANALYSIS_ENABLED=true
AUTO_CLIP_MAX_CLIPS=5

# Соціальні мережі
TIKTOK_CLIENT_KEY=...
TIKTOK_CLIENT_SECRET=...
INSTAGRAM_APP_ID=...
INSTAGRAM_APP_SECRET=...
GOOGLE_CLIENT_ID=...
```

```
GOOGLE_CLIENT_SECRET=...
# Stripe
STRIPE_SECRET_KEY=sk_live_...
STRIPE_WEBHOOK_SECRET=whsec_...

# Медіаоброблення та фонові задачі
YTDLP_MAX_HEIGHT=720
BG_WORKERS=16
MAX_CONCURRENT_TASKS=2
FFMPEG_WORKERS=2
FFMPEG_TIMEOUT_SECONDS=900
AUTO_CLIP_TARGET_DURATION=60
```

У виробничому середовищі обов’язковим параметром є PUBLIC_BASE_URL. Він використовується для формування OAuth redirect URI, публічних посилань на кліпи та webhook-payload. За відсутності цього параметра поза режимом розробки застосунок зупиняє запуск, щоб уникнути некоректних адрес зворотного виклику та небезпечних резервних значень.

Параметр BG_WORKERS задає верхню межу кількості фонових задач у ThreadPoolExecutor. Він не визначає кількість процесів Uvicorn, а обмежує паралельне виконання ресурсоємних операцій усередині серверної частини. Параметр FFMPEG_WORKERS обмежує кількість одночасних рендерингів кліпів для контролю навантаження на процесор і пам’ять під час оброблення відео.

Для виробничого розгортання застосунок контейнеризовано через Docker. Образ серверної частини базується на python:3.11-slim, містить статичну збірку FFmpeg, Deno 2.1.4, шрифти Montserrat і запускається від непривілейованого користувача appuser. Робочі дані не вбудовуються в образ, а зберігаються у змонтованій директорії storage з вихідними відео, готовими кліпами, субтитрами, тимчасовими файлами та кешами.

Docker Compose описує кілька сервісів: серверну частину, окремий сервіс транскрипції на базі Whisper, клієнтську частину на базі React і Nginx, Postgres для локального профілю розробки та допоміжні сервіси для роботи з

YouTube. Серверна частина має healthcheck на /health, що дозволяє перевіряти доступність застосунку після запуску контейнера.

Автоматизацію перевірок реалізовано через GitHub Actions. У процесі ci-fast.yml запускаються перевірки серверної частини, міграції Alembic, тести pytest і збірка клієнтської частини. Окремий процес ci-docker.yml перевіряє збірку образу та базову працездатність контейнера. Загальну схему CI/CD-процесу наведено на рисунку 3.1.

Рисунок 3.1 – Схема CI/CD пайплайну платформи

3.2 Моделювання структури та процесів вебплатформи

Для формалізації логіки роботи вебплатформи застосовано функціональне моделювання процесів. Такий підхід описує систему не як набір програмних модулів, а як послідовність перетворення вхідних даних у готовий результат, від відеофайлу або посилання YouTube до коротких кліпів, субтитрів, статусу задачі та публікації у соціальних мережах.

На контекстному рівні платформа розглядається як єдиний процес A0 – автоматизоване створення та публікація коротких відеокліпів. Входами є відеофайл користувача, посилання YouTube, параметри створення кліпів і дані авторизації. Керування задають вимоги платформ TikTok, Instagram Reels та YouTube Shorts, обмеження тарифного плану, правила безпеки й обмеження тривалості кліпу. Механізмами виконання є серверна частина на FastAPI, faster-whisper, DeepSeek API, FFmpeg, PostgreSQL та API соціальних

платформ. Виходами процесу є готові MP4-кліпи, транскрипт, субтитри, статус задачі, опублікований пост і webhook-сповіщення [35].

Контекстну діаграму процесу автоматизованого створення та публікації коротких відеокліпів наведено на рисунку 3.2.

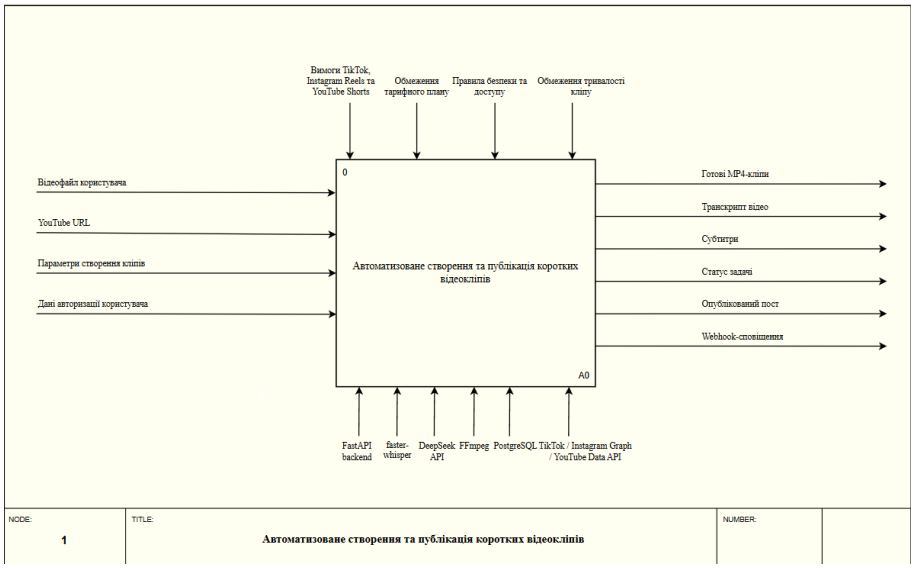

Рисунок 3.2 – Контекстна діаграма процесу автоматизованого створення та публікації коротких відеокліпів

Процес A0 декомпозовано на основні функціональні етапи, зокрема приймання відео, транскрипцію мовлення, визначення меж кліпів, рендеринг готових фрагментів, публікацію у соціальних мережах і надсилання сповіщень зовнішнім системам. Декомпозиція відображає реальний порядок роботи платформи та відповідає основним підсистемам серверної частини. Діаграму декомпозиції першого рівня наведено на рисунку 3.3. На другому рівні деталізовано процес визначення меж кліпів. Система отримує сегменти транскрипції, формує запит до моделі штучного інтелекту, приймає набір запропонованих фрагментів, перевіряє їхню тривалість і передає коректні результати до інтерфейсу користувача. Діаграму декомпозиції другого рівня наведено на рисунку 3.4.

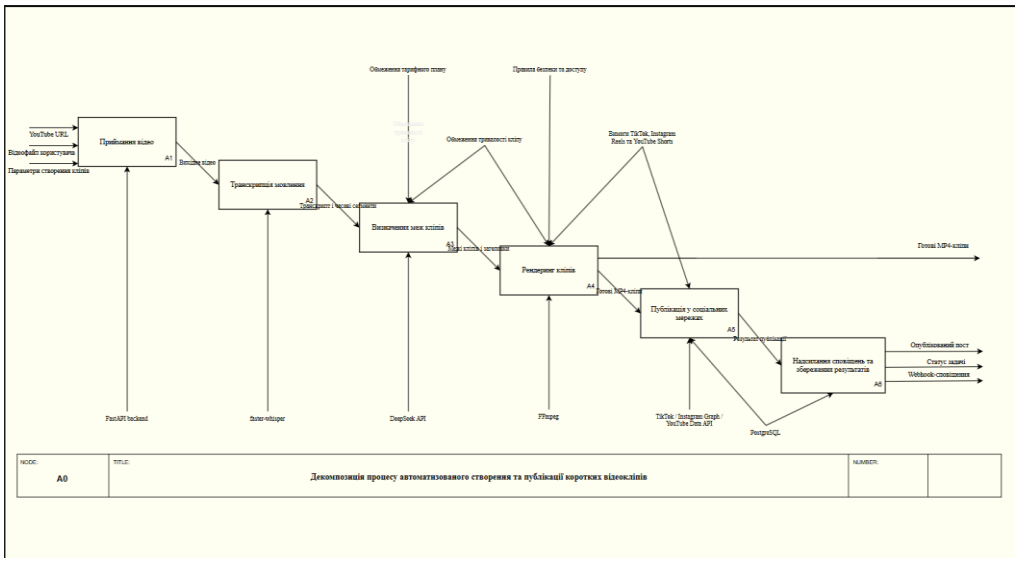

Рисунок 3.3 – Діаграма декомпозиції процесу автоматизованого створення та публікації коротких відеокліпів

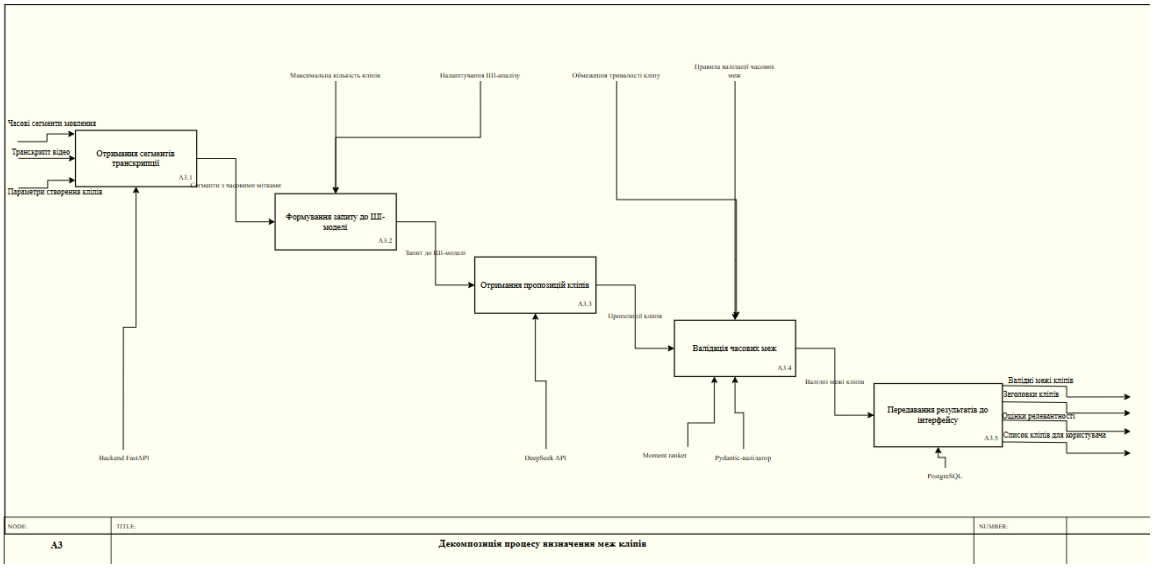

Рисунок 3.4 – Діаграма декомпозиції другого рівня процесу визначення меж кліпів

Діаграма варіантів використання відображає взаємодію користувача, адміністратора та зовнішніх систем із платформою. Основним актором є користувач, який завантажує відео, переглядає транскрипт, створює кліпи, публікує їх у соціальні мережі та керує підпискою. Зовнішніми акторами виступають API соціальних платформ, Stripe і webhook-отримувачі. Адміністратор контролює роботу системи, облікові записи користувачів і

налаштування сервісу. Діаграму варіантів використання вебплатформи наведено на рисунку 3.5.

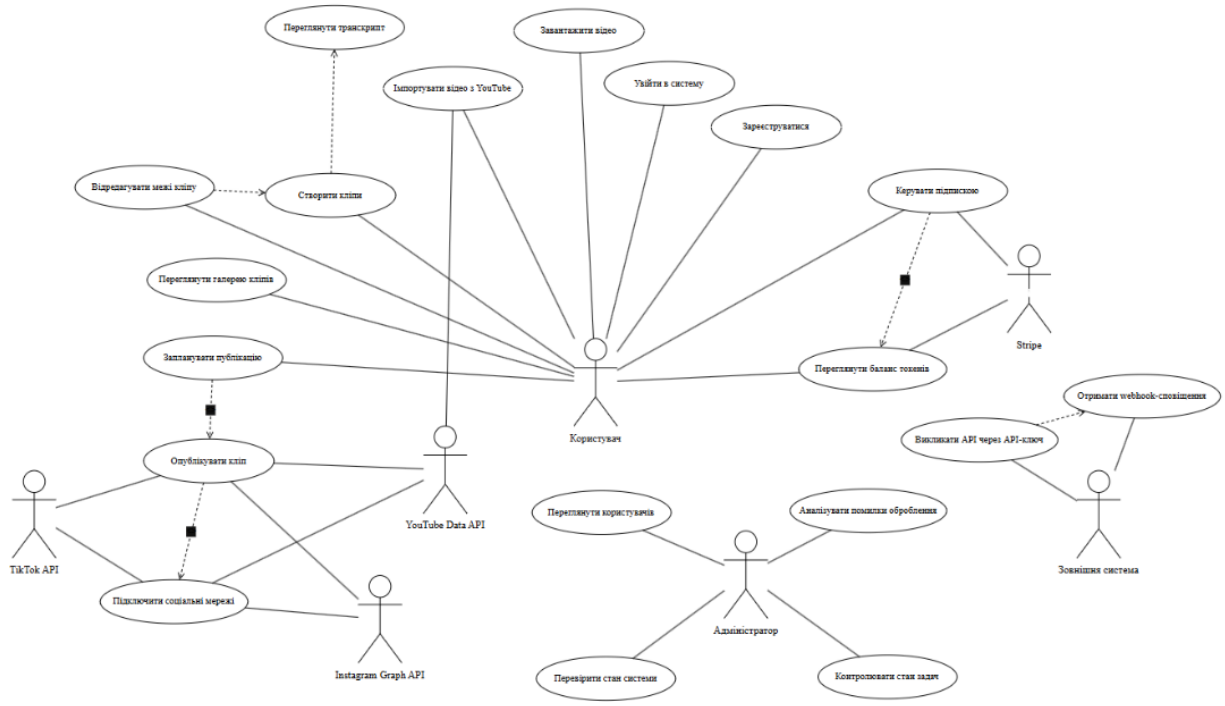

Рисунок 3.5 – Головна діаграма варіантів використання вебплатформи

Побудовані діаграми узгоджують функціональну модель платформи з її програмною реалізацією. Контекстна діаграма визначає межі системи та зовнішні взаємодії, діаграми декомпозиції деталізують послідовність оброблення відео, діаграма варіантів використання описує основні сценарії роботи користувача. На основі цієї моделі в наступних підрозділах розглянуто реалізацію медіапайплайну, підсистеми публікації, автентифікації, білінгу та тестування вебплатформи.

3.3 Реалізація медіапайплайну оброблення відео

3.3.1 Транскрипція відео та визначення меж кліпів

Практична реалізація медіапайплайну конкретизує архітектурну схему оброблення відео, описану в розділі 2. У цьому підрозділі основну увагу приділено параметрам запуску faster-whisper, отриманню часових міток, валідації меж кліпів і налаштуванням FFmpeg для формування готових MP4-файлів.

Для транскрипції використовується faster-whisper. У поточній конфігурації модель завантажується один раз і повторно використовується для наступних задач. Застосовується модель base на CPU з типом обчислень int8, що зменшує вимоги до пам’яті та дозволяє виконувати транскрипцію без окремого GPU-сервера [36]. Основну конфігурацію запуску транскрипції наведено у лістингу 3.2.

Лістинг 3.2 Ініціалізація faster-whisper та транскрипція аудіо:

```
model = WhisperModel(
    "base",
    device="cpu",
    compute_type="int8",
    cpu_threads=2,
    num_workers=1,
)
segments, info = model.transcribe(
    video_path,
    word_timestamps=True,
    beam_size=1,
)
```

У лістингу 3.2 показано базові параметри транскрипції. Параметр word_timestamps вмикає формування часових міток для окремих слів, а beam_size=1 зменшує обчислювальні витрати під час розпізнавання. Така конфігурація є практичною для фонової обробки користувацьких відео, де важлива не лише якість розпізнавання, а й стабільний час виконання задачі.

Результатом транскрипції є список сегментів. Кожен сегмент містить час початку, час завершення, текстовий фрагмент мовлення і, за наявності, часові мітки окремих слів. Ці дані зберігаються у задачі та використовуються для формування субтитрів, визначення меж кліпів і відображення результатів користувачу.

Якщо модель не повертає часових міток окремих слів, система оцінює їх приблизно на основі тривалості сегмента та довжини слів у тексті. Такий резервний механізм не замінює повноцінну транскрипцію, але дозволяє зберегти працездатність пайплайну і сформувати субтитри навіть за неповного результату розпізнавання.

Для запобігання зависанню задачі транскрипція має обмеження за часом виконання. Максимальна тривалість операції становить 1800 с. Якщо оброблення перевищує цей ліміт або завершується помилкою, задача переводиться у стан failed, а повідомлення про помилку зберігається для відображення користувачу.

Після отримання транскрипту система переходить до визначення фрагментів. За увімкненого ШІ-аналізу текстові сегменти передаються до DeepSeek API. Модель аналізує зміст відео і повертає набір пропозицій із початком і кінцем фрагмента, заголовком, поясненням вибору та оцінкою релевантності.

Отримані пропозиції не передаються одразу на рендеринг. Система перевіряє коректність часових меж, тривалість кліпу та наявність текстового змісту. Некоректні фрагменти відкидаються, валідні сортуються за оцінкою релевантності й передаються до інтерфейсу користувача для перегляду або редагування.

3.3.2 Рендеринг кліпів та збереження результатів

Після визначення меж кліпів задача переходить до етапу рендерингу. Перед обрізанням відео система отримує технічні параметри файлу через ffprobe: тривалість, роздільну здатність і наявність аудіодоріжки. Ці дані використовуються для перевірки коректності часових меж і вибору способу масштабування.

Рендеринг виконується через FFmpeg. Готові кліпи формуються у форматі MP4 з відеокодеком H.264 та аудіокодеком AAC. Для горизонтальних відео застосовується вертикальний формат 1080×1920 із розмитим фоном, що адаптує контент до вимог TikTok, Instagram Reels та YouTube Shorts. Спрощений фрагмент формування параметрів кодування наведено у лістингу 3.3.

Лістинг 3.3 Параметри FFmpeg для кодування готового кліпу:

```
def _encode_args() -> list[str]:
    return [
        "-c:v", "libx264",
        "-crf", FFMPEG_CRF,
        "-preset", FFMPEG_PRESET,
        "-threads", FFMPEG_THREADS,
        "-c:a", "aac",
        "-b:a", FFMPEG_AUDIO_BITRATE,
    ]
```

У лістингу 3.3 показано спільні параметри кодування вихідних MP4-файлів. Використання libx264 забезпечує сумісність із більшістю соціальних платформ, параметр CRF керує співвідношенням якості та розміру файлу, а AAC використовується як типовий аудіокодек для вебвідео.

Вертикальне компонування кліпу формується через набір FFmpeg-фільтрів. Для відео з горизонтальним співвідношенням сторін система створює розмитий фон, масштабує основний кадр до ширини вертикального відео і розміщує його по центру. Основна область кадру зберігається, а вертикальний простір заповнюється без чорних полів.

Субтитри накладаються у форматі ASS на основі сегментів транскрипції та вбудовуються у відео під час рендерингу. Готовий кліп не потребує окремого файла субтитрів під час публікації на зовнішні платформи.

За наявності заголовка система створює окремий PNG-оверлей і накладає його у верхній частині кадру. Заголовок відображається протягом перших секунд кліпу і поступово зникає, що дозволяє передати основну ідею фрагмента без ручного монтажу.

Рендеринг кількох кліпів може виконуватися паралельно через ThreadPoolExecutor. Кількість одночасних операцій обмежується параметром FFMPEG_WORKERS, щоб не перевантажувати процесор і пам’ять під час оброблення великих відеофайлів. Якщо один із кліпів не вдається сформувати, система зберігає повідомлення про помилку і не приховує збій від користувача.

Готові MP4-файли зберігаються у директорії кліпів, інформація про них записується у стан задачі. Після успішного завершення всіх операцій задача переводиться у стан completed. Якщо на будь-якому етапі виникає помилка FFmpeg або перевищено ліміт часу рендерингу, задача переводиться у стан failed із діагностичним повідомленням.

3.4 Реалізація підсистеми публікації та зовнішніх інтеграцій

3.4.1 Публікація кліпів у соціальні мережі

У розділі 2 підсистему публікації розглянуто на рівні архітектури та відмінностей між API TikTok, Instagram і YouTube. У цьому підрозділі

показано програмну реалізацію єдиного механізму вибору publish-модуля, який залежно від платформи передає готовий MP4-кліп до відповідного зовнішнього сервісу.

Для кожної платформи у коді виділено окремий модуль авторизації та публікації. Архітектурні відмінності між API TikTok, Instagram і YouTube розглянуто в розділі 2, а в цьому підрозділі ці відмінності відображено на рівні маршрутизації publish-запиту. Узагальнений фрагмент вибору publish-модуля наведено у лістингу 3.4.

Лістинг 3.4 Вибір publish-модуля залежно від платформи:

```
if platform == "tiktok":
    result = await upload_video_to_tiktok(
        video_path=local_path,
        title=title,
        user_id=user_id,
        db=db,
        description=description,
    )

  elif platform == "instagram":
      public_video_url = f"{PUBLIC_BASE_URL}/clips/{video_filename}"
      result = await publish_reel_to_instagram(
          video_url=public_video_url,
          caption=caption,
          user_id=user_id,
          db=db,
      )

elif platform == "youtube":
    result = await upload_video_to_youtube(
```

```
        video_path=local_path,
        title=title,
        user_id=user_id,
        db=db,
        description=description,
        privacy_status=privacy_status,
    )
```

Лістинг 3.4 демонструє принцип розділення логіки публікації за платформами. Серверна частина не зводить усі зовнішні API до одного технічного сценарію, а реалізує єдиний користувацький інтерфейс поверх різних механізмів передавання відео. Це спрощує подальше розширення системи, оскільки підтримку нової платформи можна додати окремим модулем без зміни медіапайплайну.

Публікація у TikTok виконується через TikTok Content Posting API. Серверна частина отримує чинний токен доступу користувача і ініціалізує завантаження відео. У відповідь TikTok повертає URL-завантаження і publish_id. Далі файл передається через PUT-запити з урахуванням розміру відео та кількості частин. Після завантаження TikTok повертає статус обробки, оскільки фінальна обробка відео виконується на стороні платформи.

Публікація в Instagram виконується через Instagram Graph API. Особливість цього сценарію полягає в тому, що Instagram не приймає локальний файл напряму, а потребує публічно доступного URL відео. Серверна частина формує посилання на готовий кліп через PUBLIC_BASE_URL і передає його як параметр video_url. Процес складається з двох етапів: створення медіаконтейнера і фінальна публікація через media_publish.

Публікація у YouTube виконується через YouTube Data API v3 [32]. Серверна частина отримує облікові дані користувача, за потреби оновлює токен доступу через токен оновлення і формує об’єкт MediaFileUpload для

локального MP4-файлу. Разом із відео передаються метадані: назва, опис, теги, категорія та параметр приватності. Після успішного завантаження YouTube повертає video_id, на основі якого формується URL опублікованого відео.

Перед публікацією система перевіряє доступність функції для поточного тарифного плану користувача. Якщо тарифний план не підтримує автопублікацію в обрану платформу, серверна частина повертає помилку доступу і не запускає зовнішній API-запит. Також перед виконанням publish-операції списується відповідна кількість токенів, що дозволяє пов’язати використання зовнішніх інтеграцій із білінговою моделлю платформи.

Після успішної публікації серверна частина зберігає ідентифікатор результату від зовнішньої платформи: ідентифікатор публікації для TikTok, ідентифікатор медіа для Instagram або ідентифікатор відео для YouTube. Додатково фіксується аналітична подія публікації для відстеження використання кліпів і побудови статистики активності користувача.

3.4.2 Планування публікацій

У розділі 2 механізм відкладених публікацій описано як DB-чергу запланованих задач. У програмній реалізації основним елементом цього механізму є вибір найближчої публікації зі статусом pending із блокуванням відповідного рядка в базі даних.

Під час створення відкладеної публікації серверна частина перевіряє належність кліпу поточному користувачу, коректність часу публікації та наявність відповідної функції у тарифному плані. Час scheduled_at нормалізується до UTC, а значення у минулому відхиляються. Для уникнення дублювання система перевіряє, чи вже існує запланована публікація з таким самим користувачем, кліпом, платформою і часом виконання.

Виконання відкладених публікацій реалізовано як фоновий цикл планувальника. Він періодично перевіряє таблицю scheduled_publishes і

вибирає задачі зі статусом pending, час яких уже настав. Для запобігання одночасному виконанню однієї задачі кількома процесами використовується блокування рядків на рівні бази даних. Фрагмент вибірки запланованих задач наведено у лістингу 3.5.

Лістинг 3.5 Вибірка запланованих публікацій із блокуванням рядків:

```
job = (
    db.query(ScheduledPublish)
    .filter(
        ScheduledPublish.status == "pending",
        ScheduledPublish.scheduled_at <= now,
    )
    .order_by(
        ScheduledPublish.scheduled_at.asc(),
        ScheduledPublish.id.asc(),
    )
    .with_for_update(skip_locked=True)
    .first()
)
```

У лістингу 3.5 показано вибірку найближчої запланованої задачі з використанням SELECT FOR UPDATE SKIP LOCKED. Якщо один процес уже заблокував рядок, інший пропускає його і переходить до наступної доступної задачі – виключає подвійну публікацію при паралельній роботі кількох воркерів.

Після вибірки задача одразу переводиться у статус running. Планувальник викликає відповідний модуль публікації, передає йому шлях до локального MP4-файлу та параметри публікації. За успішного результату задача отримує статус done, а в полі result_id зберігається ідентифікатор від

зовнішньої платформи. У разі помилки задача переводиться у статус failed із текстом помилки для відображення користувачу.

Додатково реалізовано механізм відновлення завислих задач. Якщо задача тривалий час залишається у статусі running, планувальник перевіряє її результат. За наявності result_id задача позначається як done, за його відсутності переводиться у failed із кодом STALE_TIMEOUT. Це захищає систему від ситуацій, коли процес було перервано під час виконання публікації.

Автоматичні повторні публікації після помилки не виконуються, повторний запит до соціальної мережі може призвести до дублювання контенту. Якщо публікація завершилася невдало, користувач бачить повідомлення про помилку і може створити нову заплановану публікацію вручну.

У поточній реалізації відкладені публікації застосовуються до платформ, підтримуваних серверним планувальником. Миттєва публікація в Instagram реалізована окремим маршрутом через Instagram Graph API, однак її виконання через планувальник потребує окремого узгодження з вимогами Meta до доступності публічного URL відео.

3.4.3 Webhook-сповіщення та API-ключі

Архітектурну роль webhook-сповіщень та API-ключів описано в розділі 2. У цьому підрозділі розглянуто їх програмну реалізацію: формування повідомлення про завершення задачі, передавання його на задану webhook-адресу та використання API-ключів для доступу до окремих маршрутів серверної частини.

Після завершення оброблення відео серверна частина формує JSON-повідомлення зі статусом задачі, списком готових кліпів і можливим текстом помилки. Якщо користувач передав webhook_url під час створення задачі, це

повідомлення надсилається на відповідну адресу. Спрощений приклад формування такого повідомлення наведено у лістингу 3.6.

Лістинг 3.6 Формування webhook-повідомлення про завершення задачі:

```
payload = {
    "event": "task_completed",
    "task_id": task_id,
    "status": status,
    "clips": clips,
    "error": error,
}

send_webhook(
    webhook_url,
    payload,
    user_id=user_id,
    task_id=task_id,
)
```

У лістингу 3.6 показано структуру повідомлення для зовнішньої системи. Поле event визначає тип події, task_id дозволяє зіставити повідомлення з конкретною задачею, status показує результат оброблення, clips містить список сформованих кліпів. За наявності помилки її опис передається у полі error.

Перед надсиланням webhook система перевіряє адресу webhook для зниження ризику SSRF-атак. Дозволяється лише HTTPS-адреси, забороняються localhost, внутрішні домени, приватні IP-адреси та адреси служб метаданих хмарних провайдерів. Це не дозволяє використати webhook як спосіб доступу до внутрішньої інфраструктури сервера.

Другим механізмом зовнішньої інтеграції є API-ключі. Користувач створює ключ у налаштуваннях і отримує його значення лише один раз. У базі даних зберігається SHA-256 хеш і короткий префікс для відображення в інтерфейсі. Під час звернення до API користувач передає ключ у заголовку Authorization, серверна частина хешує отримане значення і порівнює його із записом у таблиці api_keys.

API-ключі мають ознаку активності, дату створення, час останнього використання та необов’язковий термін дії. Це дозволяє відкликати ключ без видалення історії, відстежувати активність інтеграцій і обмежувати час доступу зовнішніх систем. За відсутності, простроченості або деактивації ключа серверна частина повертає помилку авторизації і не виконує запит.

Поєднання webhook-сповіщень та API-ключів робить платформу частиною ширшого автоматизованого процесу. Зовнішня система може створити задачу через API, отримати webhook після завершення оброблення і далі використати готові кліпи у власному сценарії публікації або аналітики.

3.5 Реалізація системи автентифікації та безпеки

Архітектурні засади безпеки вебплатформи описано в розділі 2. У програмній реалізації ці засади втілено через JWT-автентифікацію, CSRF-перевірку запитів, контроль доступу до файлів, валідацію зовнішніх адрес і прозоре шифрування OAuth-токенів на рівні ORM-моделі.

Автентифікація користувачів виконується через JWT-токен, який містить ідентифікатор користувача, email і час завершення дії. У вебінтерфейсі токен зберігається у cookie з прапорцем httpOnly, що зменшує ризик викрадення токена через XSS-уразливості.

Для запитів, що змінюють стан системи, додатково застосовується захист від міжсайтової підробки запитів за схемою подвійної передачі cookie [35]. Сервер встановлює значення csrf_token, а клієнт передає це саме значення

у заголовку X-CSRF-Token. За відсутності або розбіжності значень серверна частина відхиляє запит. Цей механізм охоплює публікації, зміну налаштувань, роботу з кліпами, підписками та API-ключами.

Токени доступу та оновлення для Google, TikTok та Instagram не записуються у базу даних у відкритому вигляді. У моделі користувача для цього використано зашифровані поля та гібридні властивості SQLAlchemy, під час запису значення автоматично шифрується, а під час читання – розшифровується для бізнес-логіки. Спрощений фрагмент реалізації наведено у лістингу 3.7.

Лістинг 3.7 Шифрування OAuth-токенів користувача:

```
def _make_encrypted_property(private_attr: str):
    @hybrid_property
    def prop(self):
        raw = getattr(self, private_attr)
        if raw:
            return decrypt_token(raw)
        return None

    @prop.setter
    def prop(self, value):
        if value:
            setattr(self, private_attr, encrypt_token(value))
        else:
            setattr(self, private_attr, None)
    return prop

google_access_token = _make_encrypted_property("_google_access_token")
tiktok_access_token = _make_encrypted_property("_tiktok_access_token")
```

instagram_access_token = _make_encrypted_property("_instagram_access_token")

У лістингу 3.7 показано принцип прозорого шифрування токенів на рівні ORM-моделі. Бізнес-логіка працює з властивістю як зі звичайним полем, але в базі даних зберігається зашифроване значення. За витоку бази даних токени залишаються непридатними для використання без ключа шифрування.

Перед видачею або публікацією файла серверна частина викликає resolve_clip_path(), яка нормалізує шлях і перевіряє належність файла до дозволеної директорії кліпів. Це унеможливлює атаки на обхід шляху, коли користувач намагається отримати доступ до файлів поза межами сховища застосунку.

Під час завантаження відео перевіряється не лише розширення файла, а й його вміст за сигнатурними байтами. Це знижує ризик підміни типу файла, коли небезпечний або непідтримуваний файл маскується під відео через назву або MIME-тип. Після перевірки файл зберігається у службовій директорії та обробляється фоновим пайплайном.

Захист платформи поєднує автентифікацію, перевірку запитів, шифрування чутливих даних, контроль доступу до файлів і базові механізми безпеки браузера. Це дозволяє обмежити ризики, пов’язані з обробленням користувацького контенту, OAuth-інтеграціями та зовнішніми API.

3.6 Реалізація системи токенів та білінгу

Архітектурну роль токенної моделі та принцип атомарного списання описано в розділі 2. У цьому підрозділі наведено програмну реалізацію білінгової логіки: вартість основних операцій, перевірку балансу перед запуском ресурсоємних дій і фрагмент атомарного оновлення token_balances.

На рівні бази даних білінгова логіка спирається на таблиці token_balances, token_transactions і subscriptions. Перша зберігає поточний баланс користувача, друга фіксує історію списань і нарахувань, а третя пов’язує користувача з активним тарифним планом.

Основні операції платформи мають фіксовану вартість у токенах, що дає користувачу змогу прогнозувати витрати до запуску оброблення або публікації. Вартість основних дій наведено у таблиці 3.1.

Таблиця 3.1 – Вартість основних операцій платформи у токенах

Операція	**Вартість, токенів**	**Призначення**
ШІ-визначення кліпів	10	Аналіз транскрипту та пошук потенційно цікавих фрагментів
Генерація кліпу	5	Формування готового MP4-файлу через FFmpeg
Генерація субтитрів	2	Створення субтитрів на основі транскрипції
Публікація у TikTok	3	Автоматичне розміщення кліпу через TikTok API
Публікація в Instagram	3	Автоматичне розміщення Reels через Instagram Graph API
Публікація у YouTube	3	Завантаження кліпу через YouTube Data API v3

Списання токенів виконується до запуску ресурсомісткої операції. Це важливо для запобігання ситуації, коли транскрипція, рендеринг або публікація запускаються без достатнього балансу. За нестачі токенів серверна частина повертає помилку з кодом 402, а задача не потрапляє у фонову чергу.

Для уникнення race condition під час паралельних запитів списання реалізовано атомарно на рівні SQL-запиту. Система не читає баланс окремо з подальшим оновленням, а виконує UPDATE з умовою available >= cost. Якщо

жоден рядок не оновлено, це означає, що на балансі недостатньо токенів. Спрощений фрагмент атомарного списання наведено у лістингу 3.8.

Лістинг 3.8 Атомарне списання токенів користувача:

```
updated = (
    db.query(TokenBalance)
    .filter(
        TokenBalance.user_id == user_id,
        TokenBalance.available >= cost,
    )
    .update(
        {
            TokenBalance.available: TokenBalance.available - cost,
            TokenBalance.used: TokenBalance.used + cost,
        },
        synchronize_session=False,
    )
)
if updated == 0:
    raise HTTPException(
        status_code=402,
        detail="Insufficient token balance",
    )
```

У лістингу 3.8 показано принцип атомарного списання токенів. Умова available ≥ cost перевіряється під час оновлення рядка, тому два паралельні запити не можуть одночасно витратити один і той самий залишок. Це особливо важливо для операцій рендерингу та публікації, які можуть запускатися кілька разів поспіль.

Підписки зберігаються у таблиці subscriptions і пов’язані з планами платформи. Для інтеграції з платіжною системою використовується Stripe. Події Stripe обробляються через webhook [36], а їхні ідентифікатори зберігаються у таблиці stripe_webhook_events – це забезпечує ідемпотентність, якщо Stripe повторно надішле ту саму подію, система не застосує її вдруге.

Токенна модель також слугує механізмом контролю доступу до функцій. Перед виконанням платної операції серверна частина перевіряє не лише баланс, а й доступність відповідної можливості в межах його тарифного плану. Автопублікація та ШІ-визначення кліпів можуть бути обмежені для окремих планів.

Білінгова підсистема поєднує тарифні плани, токенний баланс, журнал транзакцій і оброблення платіжних подій Stripe. Це дозволяє прозоро контролювати витрати користувача, запобігати запуску неоплачених операцій і зберігати історію змін балансу.

3.7 Тестування платформи

3.7.1 Організація тестів та їх результати

Тестування вебплатформи виконувалося на рівні API, бізнес-логіки та допоміжних модулів. Основна мета тестування полягала у перевірці коректності автентифікації, токенної моделі, створення задач, оброблення кліпів, публікації, webhook-сповіщень і роботи з базою даних.

Тестова база поділена на два основні каталоги tests/api та tests/unit. У tests/api розміщено інтеграційні тести, які перевіряють поведінку HTTP-ендпоінтів через FastAPI TestClient. У tests/unit розміщено модульні тести для окремих функцій, зокрема валідаторів, сервісів токенів, допоміжних функцій, логіки планувальника та оброблення помилок.Під час тестування зовнішні сервіси замінювалися заглушками. Це стосується DeepSeek API, TikTok API, Instagram Graph API, YouTube Data API, Stripe, FFmpeg, faster-whisper, yt-dlp і

webhook-доставки. База даних, автентифікація та токенний сервіс не замінювалися, оскільки саме їхня взаємодія є критичною для перевірки коректності роботи платформи. Команди запуску тестів наведено у лістингу 3.9.

Лістинг 3.9 Команди запуску тестів:

```
python -m pytest -q
python -m pytest --cov=app --cov-report=term-missing
```

У лістингу 3.9 наведено дві основні команди. Перша команда запускає весь набір тестів, друга додатково формує звіт про покриття коду. Результати контрольного запуску наведено у таблиці 3.2.

Контрольний запуск виконувався після внесення основних змін у серверну частину платформи. Оцінювання проводилося за двома показниками: успішністю проходження тестів і рівнем покриття коду. Перший підтверджує відсутність регресій у перевірених сценаріях, другий показує, яка частина програмного коду була виконана під час тестування.

Таблиця 3.2 – Результати автоматизованого тестування

Показник	**Значення**
Кількість тестових файлів	65
Кількість тест-функцій	795
Кількість виконаних pytest-кейсів	809
Результат запуску	809 passed
Покриття коду	68 %
Основні каталоги тестів	tests/api, tests/unit

Різниця між кількістю тест-функцій і кількістю виконаних перевірок пояснюється параметризацією: одна тестова функція може запускатися кілька

разів з різними наборами вхідних даних. Фактична кількість виконаних перевірок становить 809.

3.7.2 Перевірка критичних сценаріїв платформи

Інтеграційні тести перевіряють ключові сценарії роботи серверної частини: реєстрацію та вхід користувача, захищені маршрути, CSRF-захист, створення задач, списання токенів, роботу з API-ключами, журналами webhook-сповіщень, подіями Stripe та публікацією у соціальні мережі. Для цих тестів використовується окрема тестова база даних, що дає змогу перевіряти реальні зміни стану без звернення до виробничого середовища.

Модульні тести охоплюють ізольовану бізнес-логіку: розрахунок вартості операцій, атомарне списання токенів, перевірку меж кліпів, оброблення помилок, валідацію шляхів до файлів і поведінку допоміжних сервісів. Завдяки такому поділу помилки можна локалізувати як на рівні окремої функції, так і на рівні взаємодії кількох компонентів.

Окрему увагу приділено перевірці сценаріїв відмови: недостатній токенний баланс, некоректні запити, помилки зовнішніх API, недоступність адреси webhook-сповіщення, невдала публікація та спроби доступу до чужих або неіснуючих ресурсів. Це важливо для платформи, де значна частина роботи виконується асинхронно і залежить від зовнішніх сервісів.

Отримані результати контрольного запуску підтверджують працездатність основних сценаріїв платформи. Покриття коду становить 68 %, що є достатнім для перевірки критичних механізмів серверної частини, але залишає простір для подальшого розширення тестової бази, зокрема для рідкісних граничних випадків медіаоброблення та окремих помилок зовнішніх інтеграцій.

3.8 Аналіз продуктивності

Продуктивність вебплатформи clipperok визначається трьома основними групами операцій, а саме транскрипцією мовлення, ШІ-аналізом транскрипту та рендерингом готових кліпів. Ці етапи мають різну природу навантаження: транскрипція залежить від тривалості аудіодоріжки та параметрів моделі faster-whisper, ШІ-аналіз – від довжини транскрипту й затримки зовнішнього API, а рендеринг – від роздільної здатності відео, кількості кліпів і параметрів FFmpeg.

Для запобігання перевантаженню серверної частини важкі операції не виконуються безпосередньо в HTTP-запиті. Після створення задачі оброблення передається у фоновий ThreadPoolExecutor. Завдяки цьому користувач швидко отримує task_id, а подальший прогрес відстежується через двостороннє з’єднання. Такий підхід скорочує час очікування відповіді API та дозволяє виконувати тривалі операції незалежно від життєвого циклу HTTP-з’єднання. Основні параметри, що впливають на продуктивність платформи, наведено у таблиці 3.3.

Таблиця 3.3 – Чинники, що впливають на продуктивність платформи

Чинник	**Вплив на продуктивність**	**Коментар**
1	**2**	**3**
Тривалість відео	Високий	Збільшує час транскрипції та рендерингу
Роздільна здатність відео	Високий	Впливає на швидкість FFmpeg-кодування
Кількість сформованих кліпів	Високий	Кожен кліп потребує окремої операції рендерингу
Довжина транскрипту	Середній	Впливає на розмір запиту до ШІ-моделі
Кількість одночасних задач	Високий	Обмежується кількістю фонових воркерів

Продовження таблиці 3.3

1	2	3
Затримка зовнішніх API	Середній	Впливає на ШІ-аналіз і публікацію
Швидкість дискової підсистеми	Середній	Впливає на читання та запис великих MP4-файлів

Найбільш ресурсомістким етапом є рендеринг кліпів, оскільки FFmpeg виконує обрізання відео, масштабування до формату 1080×1920, накладання субтитрів і заголовка. Кількість одночасних операцій рендерингу обмежується параметром FFMPEG_WORKERS, що дозволяє контролювати навантаження на процесор і пам’ять під час оброблення кількох кліпів одного відео.

Транскрипція також належить до тривалих операцій, але її виконання оптимізовано за рахунок faster-whisper з типом обчислень int8 [36]. Модель завантажується один раз і повторно використовується для наступних задач, що знижує накладні витрати при послідовному обробленні відео. Для транскрипції встановлено ліміт часу 1800 с, після його перевищення задача переводиться у стан failed.

ШІ-аналіз транскрипту виконується через зовнішній DeepSeek API, тому його швидкість залежить від доступності сервісу, довжини транскрипту та мережевої затримки. У разі помилки ШІ-аналізу результати транскрипції зберігаються, користувач може продовжити роботу з текстом і сформувати межі кліпів вручну.

Додатковим чинником продуктивності є робота з великими файлами. Вихідні відео і готові кліпи зберігаються у службових директоріях. Для запобігання накопиченню зайвих файлів реалізовано механізм очищення, який видаляє застарілі вихідні файли після завершення їхнього життєвого циклу.

Стабільна продуктивність платформи формується завдяки фоновому виконанню важких задач, обмеженню кількості воркерів, повторному використанню моделі транскрипції, контролю лімітів часу та очищенню

тимчасових файлів.Це дозволяє зберігати стабільність серверної частини навіть під час оброблення великих відеофайлів.

3.9 Інструкція користувача

Робота з вебплатформою виконується послідовно. Користувач входить до акаунта, створює задачу оброблення відео, переглядає результати, за потреби редагує параметри кліпів і публікує готові фрагменти у соціальні мережі.

Етап 1. Реєстрація або вхід до акаунта.

Користувач відкриває вебінтерфейс платформи та створює новий обліковий запис або входить до вже наявного акаунта. Після успішної автентифікації відкривається основна сторінка застосунку, де доступні завантаження відео, перегляд задач, робота з кліпами, підключення соціальних мереж і контроль токенного балансу.

Етап 2. Створення задачі оброблення відео.

Користувач може створити задачу двома способами: завантажити локальний відеофайл або ввести посилання на YouTube-відео. При завантаженні локального файла система перевіряє розширення, розмір і фактичний вміст за magic bytes. При використанні YouTube-посилання серверна частина перевіряє домен і завантажує відео через yt-dlp. Після цього створюється задача оброблення.

Етап 3. Очікування транскрипції та аналізу відео.

Після створення задачі система запускає фонове оброблення. Користувач бачить зміну статусів у реальному часі: transcribing, processing, completed або failed. Прогрес передається через WebSocket, тому сторінка може відображати актуальний стан задачі без ручного оновлення. Якщо під час оброблення виникає помилка, користувач отримує повідомлення з її причиною.

Етап 4. Перегляд транскрипту та запропонованих кліпів.

Після завершення транскрипції користувач може переглянути текст відео, часові сегменти та запропоновані межі кліпів. Якщо увімкнено ШІ-аналіз, система додатково формує пропозиції фрагментів із заголовками та поясненням вибору. Користувач може прийняти запропоновані межі або змінити початок і кінець кліпу вручну.

Етап 5. Рендеринг готових кліпів.

Після підтвердження меж кліпу користувач запускає рендеринг. Серверна частина формує MP4-файл у вертикальному форматі 1080×1920, додає субтитри та, за потреби, заголовок кліпу. Після завершення рендерингу готовий файл з’являється у галереї результатів.

Етап 6. Завантаження або перегляд кліпу.

У галереї користувач може переглянути готовий кліп, перевірити якість субтитрів і завантажити MP4-файл на локальний пристрій. Якщо результат потребує уточнення, користувач може повернутися до редагування меж кліпу та повторно сформувати файл.

Етап 7. Підключення соціальних мереж.

Для автоматичної публікації користувач переходить на сторінку соціальних інтеграцій і підключає потрібну платформу: TikTok, Instagram або YouTube. Авторизація виконується через OAuth [34]. Після підтвердження доступу серверна частина отримує токен платформи та зберігає його у зашифрованому вигляді.

Етап 8. Публікація кліпу.

Після підключення соціальної мережі користувач може опублікувати кліп безпосередньо з галереї. Для цього обирається платформа, заповнюються назва, опис і параметри приватності. Серверна частина перевіряє токенний баланс, списує вартість операції та передає кліп до відповідного publish-модуля.

Етап 9. Планування публікації.

Якщо кліп потрібно опублікувати пізніше, користувач може створити заплановану публікацію. У цьому випадку система зберігає запис у таблиці scheduled_publishes із часом публікації та статусом pending. Планувальник перевіряє такі записи у фоновому режимі й виконує публікацію у зазначений час.

Етап 10. Контроль токенного балансу та webhook-сповіщень.

Користувач може переглядати залишок токенів і контролювати витрати на ШІ-аналіз, рендеринг, субтитри та автопублікацію. За потреби також можна вказати webhook-адресу, на яку система надсилатиме сповіщення про завершення задачі. Це дозволяє інтегрувати clipperok із зовнішніми сценаріями автоматизації.

3.10 Перспективи подальшого розвитку

На основі аналізу вебплатформи clipperok та особливостей її реалізації визначено такі напрями подальшого розвитку:

– підтримка GPU-прискорення для транскрипції через CUDA, що дозволить скоротити час розпізнавання мовлення та підвищити пропускну здатність системи;

– масштабування фонового оброблення через винесення транскрипції, ШІ-аналізу та рендерингу в окремі worker-сервіси для гнучкого розподілу навантаження між серверами;

– розширення підтримки платформ публікації зі збереженням підходу platform-specific publish-модулів, щоб нові інтеграції не змінювали основну бізнес-логіку;

– розвиток мультимодального ШІ-аналізу відео з урахуванням транскрипту, зміни сцен, появи облич, руху в кадрі, емоційної виразності та динаміки аудіо;

– автоматизація створення додаткових матеріалів до кліпу: прев’ю-зображення, варіантів заголовків, описів, хештегів і перекладів субтитрів кількома мовами;

– підтримка пакетного оброблення відео, за якої користувач зможе завантажити кілька файлів або ZIP-архів і створити чергу задач;

– удосконалення аналітики після публікації: збір статистики переглядів, лайків, коментарів і переходів для порівняння ефективності кліпів.

ВИСНОВКИ

У рамках кваліфікаційної роботи розроблено і реалізовано SaaS-вебплатформу clipperok для автоматизованого перетворення довгих відео на короткі вертикальні кліпи з подальшою публікацією у соціальних мережах. Робота охоплює повний цикл створення системи: від аналізу методів транскрипції, визначення ключових моментів і програмного оброблення відео до реалізації серверної частини, клієнтської частини, медіапайплайну, білінгу, інтеграцій із соціальними мережами та тестування готової платформи. У процесі виконання роботи вирішено такі завдання:

– проаналізовано сучасні методи автоматичного розпізнавання мовлення, зокрема HMM/GMM, RNN/LSTM та Transformer-підходи, що дозволило обґрунтувати вибір faster-whisper на базі Whisper як практичного рішення для CPU-середовища з підтримкою int8-квантизації;

– досліджено підходи до автоматизованого виявлення ключових моментів у відео: сигнальний, семантичний і мультимодальний. На основі аналізу обґрунтовано застосування LLM-моделі DeepSeek для семантичного аналізу транскриптів і формування пропозицій кліпів;

– розглянуто технології програмного оброблення відео та обрано інструментарій на базі FFmpeg і yt-dlp, що забезпечує імпорт відео з YouTube, нарізку фрагментів, масштабування до вертикального формату, накладання субтитрів ASS і формування готових MP4-кліпів;

– спроєктовано архітектуру вебплатформи з підтримкою фонових задач через ThreadPoolExecutor, збереженням стану задач у PostgreSQL, використанням in-memory-кешу для активних задач і планувальником публікацій на базі DB-черги з механізмом SELECT FOR UPDATE SKIP LOCKED;

– реалізовано OAuth 2.0 інтеграції з TikTok Content Posting API, Instagram Graph API та YouTube Data API v3, що дозволяє користувачу

підключати соціальні акаунти та публікувати готові кліпи безпосередньо з інтерфейсу платформи;

– розроблено систему API-ключів і webhook-сповіщень для інтеграції clipperok із зовнішніми системами автоматизації, зокрема n8n, Zapier або власними серверними сервісами;

– проведено автоматизоване тестування платформи: підготовлено 65 тестових файлів і 795 тест-функцій, під час контрольного запуску виконано 809 pytest-кейсів, усі вони завершилися успішно; покриття коду становить 68 відсотків.

Розглянуто існуючі методи та підходи у сферах автоматичного розпізнавання мовлення, виявлення ключових фрагментів у відео та програмного оброблення медіаданих. Проаналізовано архітектурні підходи до побудови SaaS-платформ для роботи з відеоконтентом і обґрунтовано вибір монолітної архітектури серверної частини з фоновими задачами як доцільного рішення для платформи поточного масштабу.

Спроєктовано структуру системи, що включає серверну частину API, клієнтську SPA, медіапайплайн, підсистему автентифікації, модулі публікації, токенну модель, білінг, планувальник і webhook-інтеграції. Схема бази даних реалізована на PostgreSQL і містить 16 таблиць; для керування змінами схеми використано 43 міграції Alembic. Окрему увагу приділено моделі користувача, задачам оброблення відео, токенному балансу, запланованим публікаціям і журналу webhook-доставок.

Реалізовано медіапайплайн оброблення відео зі станами pending → transcribing → processing → completed | failed. Транскрипція виконується через faster-whisper із підтримкою часових міток, результати зберігаються у вигляді сегментів і використовуються для формування субтитрів та визначення меж кліпів. ШІ-аналіз транскрипту через DeepSeek API повертає структуровані пропозиції фрагментів із часовими межами, заголовком, поясненням вибору та оцінкою релевантності. Рендеринг вертикальних кліпів виконується через

FFmpeg із підтримкою масштабування, розмитого фону, обрізання, субтитрів і заголовків.

Проведено автоматизоване тестування основних сценаріїв платформи. Інтеграційні тести перевіряють автентифікацію, захищені маршрути, CSRF-захист, створення задач, списання токенів, API-ключі, webhook-логи, Stripe-події та публікацію у соціальні мережі. Модульні тести охоплюють ізольовану бізнес-логіку, зокрема перевірку меж кліпів, токенну модель, оброблення помилок і допоміжні сервіси. Результат запуску – 809 passed – підтверджує працездатність основних сценаріїв серверної частини.

Розглянуто перспективи подальшого розвитку платформи: підтримку GPU-прискорення транскрипції через CUDA, масштабування фонових задач в окремі worker-сервіси, розширення переліку платформ публікації, розвиток мультимодального аналізу відео, автоматичну генерацію прев'ю-зображень, заголовків, хештегів і перекладів субтитрів, а також удосконалення аналітики після публікації.

Результати кваліфікаційної роботи можуть бути корисними у сферах digital-маркетингу, блогінгу, медіавиробництва та автоматизації відеовиробничих процесів. Реалізована система API-ключів і webhook-сповіщень дозволяє інтегрувати платформу у no-code та low-code сценарії автоматизації без необхідності зміни вихідного коду.

Результати роботи апробовано у вигляді тез доповіді під час Міжнародного молодіжного форуму «Радіоелектроніка і молодь у XXI столітті» [37].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. Proceedings of the IEEE, 77(2), 257–286.

2. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

3. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In Advances in Neural Information Processing Systems 30 (pp. 6000–6010). Curran Associates.

4. Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., & Sutskever, I. (2023). Robust speech recognition via large-scale weak supervision. In Proceedings of the 40th International Conference on Machine Learning (Vol. 202, pp. 28492–28518). PMLR.

5. SYSTRAN. (2025). faster-whisper: Faster Whisper transcription with CTranslate2 [Software]. GitHub. URL: https://github.com/SYSTRAN/faster-whisper (дата звернення 24.05.2026).

6. Otani, M., Nakashima, Y., Rahtu, E., & Heikkilä, J. (2019). Rethinking the evaluation of video summaries. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 7596–7604).

7. Zhang, K., Chao, W.-L., Sha, F., & Grauman, K. (2016). Video summarization with long short-term memory. In European Conference on Computer Vision (Vol. 9907, pp. 766–782). Springer.

8. Ramírez, S. (2024). FastAPI documentation (Version 0.115). URL: https://fastapi.tiangolo.com (дата звернення 21.03.2026).

9. Tvoroshenko, I., Pomazan, V., Gorokhovatskyi, V., & Kobylin, O. (2023). Application of video data classification models using convolutional neural networks. International Journal of Academic and Applied Research, 7(11), 134–145.

10. DeepSeek-AI. (2024). Scaling open-source language models with longtermism. arXiv. URL: https://arxiv.org/abs/2401.02954 (дата звернення 02.05.2026).

11. Daradkeh, Y. I., Gorokhovatskyi, V., Tvoroshenko, I., Gadetska, S., & Al-Dhaifallah, M. (2023). Statistical data analysis models for determining the relevance of structural image descriptions. IEEE Access, 11, 126938–126949.

12. Daradkeh, Y. I., & Tvoroshenko, I. (2020). Application of an improved formal model of the hybrid development of ontologies in complex information systems. Applied Sciences, 10(19), Article 6777.

13. Daradkeh, Y. I., Tvoroshenko, I., Gorokhovatskyi, V., Latiff, L. A., & Ahmad, N. (2021). Development of effective methods for structural image recognition using the principles of data granulation and apparatus of fuzzy logic. IEEE Access, 9, 13417–13428.

14. Daradkeh, Y. I., Gorokhovatskyi, V., Tvoroshenko, I., Gadetska, S., & Al-Dhaifallah, M. (2021). Methods of classification of images on the basis of the values of statistical distributions for the composition of structural description components. IEEE Access, 9, 92964–92973.

15. Daradkeh, Y. I., Gorokhovatskyi, V., Tvoroshenko, I., & Zeghid, M. (2022). Tools for fast metric data search in structural methods for image classification. IEEE Access, 10, 124738–124746.

16. Gorokhovatskyi, V., Peredrii, O., Tvoroshenko, I., & Markov, T. (2023). Distance matrix for a set of structural description components as a tool for image classifier creating. Advanced Information Systems, 7(1), 5–13.

17. Gorokhovatskyi, V., Tvoroshenko, I., Kobylin, O., & Vlasenko, N. (2023). Search for visual objects by request in the form of a cluster representation for the structural image description. Advances in Electrical and Electronic Engineering, 21(1), 19–27.

18. Tvoroshenko, I., Gorokhovatskyi, V., Kobylin, O., & Tvoroshenko, A. (2023). Application of deep learning methods for recognizing and classifying

culinary dishes in images. International Journal of Academic and Applied Research, 7(9), 57–70.

19. Pomazan, V., Tvoroshenko, I., & Gorokhovatskyi, V. (2023). Handwritten character recognition models based on convolutional neural networks. International Journal of Academic Engineering Research, 7(9), 64–72.

20. Daradkeh, Y. I., Gorokhovatskyi, V., Tvoroshenko, I., & Zeghid, M. (2024). Improving the effectiveness of image classification structural methods by compressing the description according to the information content criterion. Computers, Materials & Continua, 80(2), 2519–2535.

21. Ahmad, A., Gorokhovatskyi, V., Tvoroshenko, I., Vlasenko, N., & Mustafa, S. K. (2021). The research of image classification methods based on the introducing cluster representation parameters for the structural description. International Journal of Engineering Trends and Technology, 69(10), 186–192.

22. Suprun, A., Tvoroshenko, I., Gorokhovatskyi, V., & Yakovleva, O. (2025). Development and research of a method for the combined use of large language models for text generation. International Journal of Academic and Applied Research, 9(10), 249–263.

23. Daradkeh, Y. I., Gorokhovatskyi, V., Tvoroshenko, I., & Al-Dhaifallah, M. (2022). Cluster representation of the structural description of images for effective classification. Computers, Materials & Continua, 73(3), 6069–6084.

24. Gorokhovatskyi, V., Tvoroshenko, I., & Chmutov, Y. (2022). Application of systems of orthogonal functions for formation of sign space in image classification methods. Advanced Information Systems, 6(3), 5–12.

25. Bohdan, N., Tvoroshenko, I., Gorokhovatskyi, V., & Kobylin, O. (2025). Development of a hybrid method to enhance context memory for a chatbot application based on large language models. International Journal of Academic Information Systems Research, 9(10), 7–18.

26. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter,

C., … Amodei, D. (2020). Language models are few-shot learners. In Advances in Neural Information Processing Systems 33 (pp. 1877–1901). Curran Associates.

27. FFmpeg Developers. (2024). FFmpeg documentation (Version 7.1). URL: https://ffmpeg.org/documentation.html (дата звернення 01.04.2026).

28. yt-dlp contributors. (2024). yt-dlp: A feature-rich command-line audio/video downloader [Software]. GitHub. URL: https://github.com/yt-dlp/yt-dlp (дата звернення 28.04.2026).

29. Newman, S. (2021). Building microservices: Designing fine-grained systems (2nd ed.). O'Reilly Media.

30. Hardt, D. (2012). The OAuth 2.0 authorization framework. RFC 6749. Internet Engineering Task Force. URL: https://www.rfc-editor.org/rfc/rfc6749 (дата звернення 26.05.2026).

31. TikTok. (2024). Video specs and guidelines. TikTok for Business Help Center. URL: https://ads.tiktok.com/help/article/video-ad-specs (дата звернення 22.05.2026).

32. TikTok. (2024). Content posting API – developer documentation. URL: https://developers.tiktok.com/doc/content-posting-api-get-started (дата звернення 21.03.2026).

33. Meta Platforms. (2024). Instagram Graph API reference (v21.0). Meta for Developers. URL: https://developers.facebook.com/docs/instagram-api (дата звернення 16.05.2026).

34. Google. (2024). YouTube Data API v3 documentation. Google for Developers. URL: https://developers.google.com/youtube/v3 (дата звернення 11.04.2026).

35. Stripe. (2024). Stripe API reference. Stripe Documentation. https://docs.stripe.com/api (дата звернення 07.05.2026).

36. Hardt, D. (2012). The OAuth 2.0 authorization framework. RFC 6749. Internet Engineering Task Force. URL: https://www.rfc-editor.org/rfc/rfc6749 (дата звернення 26.05.2026).

37. Ольховський Б. Д., Сінельнікова Т. Ф. (2026). Розробка вебплатформи для автоматизованого створення коротких відеокліпів із використанням ШІ-оброблення відео та транскрипції мовлення. Матеріали Міжнародного молодіжного форуму «Радіоелектроніка і молодь у XXI столітті» (Харків, квітень 2026 р.). Харків: ХНУРЕ.