Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Комп'ютерного моделювання та інтелектуальних технологій
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)
Дослідження та розробка системи автоматичного розпізнавання мовлення та генерації субтитрів для веб-платформ
(тема)

Виконав:
студент 2 курсу, групи СПРм-24-1
Торосян А.А.
(прізвище, ініціали)

Спеціальність F3 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне проектування
(повна назва освітньої програми)

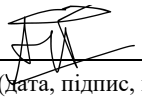
Керівник ст. викл. Яцик М.В.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри _____ Гребеннік І.В
(підпис) (прізвище, ініціали)

2026 р.

Я як студент ХНУРЕ розумію і підтримую політику закладу із академічної доброчесності. Я не надавав і не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

08.05.2026  Торосян Арзуман Арменович

(дата, підпис, прізвище студента/-ки)

Кваліфікаційна робота не містить відомостей заборонених до відкритого опублікування.

Кваліфікаційна робота виконана у відповідності до стандартів, що діють в Україні.

Попередній захист проведено 09 травня 2026 р.

Керівник кваліфікаційної роботи Яцик М.В. 

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Комп'ютерного моделювання та інтелектуальних технологій

Рівень вищої освіти другий (магістерський)

Спеціальність F3 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне проектування
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)
«__» _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Торосяну Арзуману Арменовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Дослідження та розробка системи автоматичного розпізнавання мовлення та генерації субтитрів для веб-платформ»

затверджена наказом університету від «06» квітня 2026 р. № 268Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії «13» травня 2026 р.

3. Вихідні дані до роботи Функція: дослідження та розробка веб-системи автоматичного розпізнавання мовлення та генерації субтитрів для відеоконтенту. Предметна область: обробка мультимедійного контенту для платформ коротких відео. Архітектурний патерн: Web-Queue-Worker з асинхронною комунікацією через чергу повідомлень. Організація даних: реляційна база даних PostgreSQL. Технологічний стек: TypeScript, Next.js, Fastify, TypeORM, Turborepo, FFmpeg, OpenAI Whisper API. Хмарна платформа: Google Cloud Platform. Управління інфраструктурою: Terraform. CI/CD: GitHub Actions. Технічне забезпечення: комп'ютер з доступом до мережі Інтернет, обліковий запис Google Cloud Platform, обліковий запис OpenAI API, обліковий запис Neon.

4. Перелік питань, що потрібно опрацювати в роботі Вступ. Аналіз предметної області і постановка задачі: огляд технологій автоматичного розпізнавання мовлення; роль субтитрів у сучасному цифровому контенті; порівняльний аналіз існуючих комерційних та відкритих рішень; обґрунтування актуальності та постановка завдань дослідження. Дослідження технологій та методів розв'язання задач: порівняльний аналіз моделей розпізнавання мовлення з підтримкою української мови; технології обробки відеофайлів та формати субтитрів; архітектурні підходи до систем обробки мультимедіа; аналіз хмарних технологій та інструментів інфраструктури як коду. Проєктування системи: загальна архітектура та взаємодія компонентів; проєктування моделі даних; проєктування конвеєра асинхронної обробки медіа; механізми забезпечення надійності; проєктування CI/CD конвеєра. Розробка та тестування системи. Аналіз результатів. Висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій Плакати на аркушах формату А4: актуальність теми дослідження та прогноз обсягу світового ринку систем автоматичної генерації субтитрів на основі штучного інтелекту, аналіз предметної області (комерційні рішення та відкриті компоненти), об'єкт, предмет та мета дослідження, задачі та методи дослідження, вибір моделі автоматичного розпізнавання мовлення, технологічний стек системи, загальна архітектура системи, модель даних, діаграма послідовності процесу транскрибування мовлення, діаграма послідовності процесу генерації відео з вбудованими субтитрами, налаштування CI/CD конвеєра та тестування функціоналу, аналіз якості розпізнавання мовлення та аналіз продуктивності системи, висновки.

6. Консультанти розділів роботи

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / термін виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	06.04.2026	Виконано
2	Аналіз предметної області, огляд літератури та існуючих рішень	10.04.2026	Виконано
3	Дослідження та обґрунтування вибору технологій і архітектурних підходів	12.04.2026	Виконано
4	Проєктування архітектури системи та моделі даних	15.04.2026	Виконано
5	Розробка програмного забезпечення	23.04.2026	Виконано
6	Розгортання хмарної інфраструктури	25.04.2026	Виконано
7	Налаштування CI/CD конвеєра	26.04.2026	Виконано
8	Тестування роботи системи та аналіз результатів	27.04.2026	Виконано
9	Оформлення пояснювальної записки кваліфікаційної роботи	06.05.2026	Виконано
10	Оформлення графічного матеріалу та презентації	07.05.2026	Виконано
11	Представлення на рецензування	08.05.2026	Виконано

Дата видачі завдання «06» квітня 2026 р.

Здобувач _____
(підпис)

Керівник роботи _____ ст. викл. Яцик М.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Кваліфікаційна робота магістра містить: 167 сторінок, 38 рисунків, 2 таблиці, 56 джерел, 3 додатки. Графічна частина кваліфікаційної роботи містить 20 плакатів.

АВТОМАТИЧНЕ РОЗПІЗНАВАННЯ МОВЛЕННЯ, ГЕНЕРАЦІЯ СУБТИТРІВ, ВЕБ-ПЛАТФОРМА, АСИНХРОННА ОБРОБКА, ХМАРНІ ТЕХНОЛОГІЇ, МІКРОСЕРВІСНА АРХІТЕКТУРА, OPENAI WHISPER.

Об'єкт дослідження — процес автоматичного розпізнавання мовлення та генерації субтитрів для відеоконтенту у веб-середовищі.

Предмет дослідження — методи та засоби проєктування і розробки розподіленої веб-системи обробки відеоконтенту з використанням моделей автоматичного розпізнавання мовлення, асинхронних архітектурних патернів та декларативного управління хмарною інфраструктурою.

Мета роботи — аналіз та обґрунтування оптимальних підходів до побудови відтворюваних веб-систем автоматичного розпізнавання мовлення та генерації субтитрів, а також проектування, розробка та верифікація відповідної веб-системи.

Методи дослідження — системний аналіз, порівняльний аналіз, методи моделювання програмних систем, методи верифікації програмного забезпечення.

У роботі проведено аналіз технологій автоматичного розпізнавання мовлення та обґрунтовано вибір моделі OpenAI Whisper для підтримки української мови. Розроблено документоване архітектурне рішення та реалізовано діючий прототип веб-платформи автоматичного субтитрування з верифікацією його роботи.

Галузь застосування — розробка програмного забезпечення, контент-маркетинг.

ABSTRACT

Master's Thesis contains: 167 pages, 38 figures, 2 tables, 56 references, 3 appendixes. The graphic part of the master's thesis contains 20 posters.

AUTOMATIC SPEECH RECOGNITION, SUBTITLE GENERATION, WEB PLATFORM, ASYNCHRONOUS PROCESSING, CLOUD TECHNOLOGIES, MICROSERVICE ARCHITECTURE, OPENAI WHISPER.

The object of research — the process of automatic speech recognition and subtitle generation for video content in a web environment.

The subject of research — methods and tools for designing and developing a distributed web system for video content processing, utilizing automatic speech recognition models, asynchronous architectural patterns, and declarative cloud infrastructure management.

The purpose of the research — analysis and justification of optimal approaches to building reproducible web systems for automatic speech recognition and subtitle generation, as well as the design, development, and verification of a corresponding web system.

Research methods — system analysis, comparative analysis, software system modelling methods, software verification methods.

The work includes an analysis of automatic speech recognition technologies and justification for selecting the OpenAI Whisper model for Ukrainian language support. A documented architectural solution was developed and a working prototype of an automatic subtitling web platform was implemented and verified.

The field of application — software development, content marketing.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API — Application Programming Interface (прикладний програмний інтерфейс).

ASS — Advanced Sub Station Alpha (формат файлів субтитрів з підтримкою розширеного форматування та анімацій).

CI/CD — Continuous Integration / Continuous Deployment (безперервна інтеграція та безперервне розгортання).

GPU — Graphics Processing Unit (графічний процесор).

HTTP — HyperText Transfer Protocol (протокол передачі гіпертексту).

IaC — Infrastructure as Code (інфраструктура як код).

SRT — SubRip Text (формат файлів субтитрів).

URL — Uniform Resource Locator (уніфікований локатор ресурсу).

UUID — Universally Unique Identifier (універсальний унікальний ідентифікатор).

VTT — Web Video Text Tracks (формат файлів субтитрів для веб-середовища).

WER — Word Error Rate (рівень помилок розпізнавання мовлення на рівні слів).

ВСТУП

Сучасний етап розвитку інформаційних технологій характеризується стрімким зростанням обсягів мультимедійного контенту, що створюється та поширюється через мережу Інтернет. За останні кілька років короткі відеоформати стали домінуючим засобом комунікації на таких платформах, як YouTube Shorts, Instagram Reels та TikTok, що суттєво вплинуло на підходи до маркетингу та просування контенту в соціальних мережах. У цьому контексті субтитри перетворилися з допоміжного елемента доступності на обов'язковий інструмент утримання уваги глядача, оскільки значна частина користувачів переглядає відео без звуку, перегортаючи стрічку новин у громадських місцях або в умовах, де відтворення аудіо є незручним.

Технології автоматичного розпізнавання мовлення пройшли значний шлях еволюції від ранніх статистичних моделей до сучасних нейромережевих підходів, заснованих на архітектурі трансформерів. Зокрема, поява моделей, здатних з високою точністю перетворювати мовленнєвий сигнал у текст для десятків мов, створила передумови для автоматизації процесу генерації субтитрів без залучення людини-транскрибера [1]. Водночас інтеграція таких моделей у повноцінні веб-орієнтовані програмні системи залишається нетривіальною інженерною задачею, яка вимагає вирішення низки архітектурних, інфраструктурних та проєктних питань.

Наразі на ринку існує ряд комерційних рішень для автоматичної генерації субтитрів, однак переважна більшість із них побудована за принципом закритих монолітних конвеєрів обробки, які не розкривають свою внутрішню архітектуру та не надають можливостей для гнучкої інтеграції у зовнішні робочі процеси. Крім того, для спеціалістів із маркетингу в соціальних мережах, яким необхідно оперативно обробляти велику кількість коротких відеороликів, існуючі інструменти часто виявляються або надмірно складними у використанні, або недостатньо гнучкими з точки зору налаштування та підтримки різних форматів вихідного контенту.

Актуальність даного дослідження зумовлена, з одного боку, зростаючим попитом на автоматизацію створення субтитрів з боку професійних учасників ринку контент-маркетингу, а з іншого — необхідністю розроблення відкритих архітектурних рішень, що дозволяють побудувати надійний та масштабований конвеєр обробки мультимедійних даних у хмарному середовищі. Особливого значення набуває проєктування та реалізація таких систем із застосуванням сучасних підходів до розподілених обчислень, зокрема патерну асинхронної обробки з використанням черг повідомлень, подійно-орієнтованої архітектури та декларативного управління інфраструктурою за принципами інфраструктури як коду (англ. Infrastructure as Code, IaC) [2].

Метою кваліфікаційної роботи є аналіз та обґрунтування оптимальних підходів до побудови відтворюваних веб-систем автоматичного розпізнавання мовлення та генерації субтитрів, а також проектування, розробка та верифікація відповідної веб-системи.

Для досягнення поставленої мети в рамках кваліфікаційної роботи визначено такі завдання:

– проаналізувати сучасний стан технологій автоматичного розпізнавання мовлення та визначити найбільш перспективні підходи для застосування у веб-системах;

– провести порівняльний аналіз існуючих комерційних рішень для генерації субтитрів та виявити їхні обмеження;

– обґрунтувати вибір архітектурних патернів та технологічного стеку для побудови системи;

– спроєктувати загальну архітектуру системи, модель бази даних та схему конвеєра асинхронної обробки медіаданих;

– реалізувати програмний прототип системи та провести верифікацію його роботи.

Об'єктом дослідження є процес автоматичного розпізнавання мовлення та генерації субтитрів для відеоконтенту у веб-середовищі.

Предметом дослідження є методи та засоби проєктування і розробки розподіленої веб-системи обробки відеоконтенту з використанням моделей автоматичного розпізнавання мовлення, асинхронних архітектурних патернів та декларативного управління хмарною інфраструктурою.

Методами дослідження є системний аналіз для декомпозиції задачі автоматичного субтитрування на окремі етапи та визначення взаємозв'язків між ними; порівняльний аналіз для обґрунтування вибору конкретних технологій та архітектурних рішень на кожному етапі конвеєра; методи моделювання програмних систем для проєктування архітектури, потоків даних та структури бази даних; а також методи верифікації програмного забезпечення для підтвердження коректності реалізованих функціональних вимог та механізмів надійності.

Наукова новизна одержаних результатів полягає у тому, що запропоновано документоване архітектурне рішення для побудови веб-системи автоматичного розпізнавання мовлення та генерації субтитрів, яке, на відміну від існуючих комерційних платформ із закритою архітектурою та розрізнених відкритих інструментів, що вирішують лише окремі етапи задачі, об'єднує відкриті компоненти у цілісний конвеєр обробки медіаконтенту з модульною структурою, незалежним масштабуванням компонентів та відтворюваною хмарною інфраструктурою.

Практичне значення одержаних результатів полягає у тому, що розроблений програмний прототип та задокументоване архітектурне рішення можуть бути використані як відтворювана основа для побудови аналогічних систем обробки медіаконтенту розробниками, контент-мейкерами та освітніми платформами, що потребують автоматизації субтитрування україномовного відеоконтенту.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАДАЧІ

1.1 Огляд технологій автоматичного розпізнавання мовлення

Автоматичне розпізнавання мовлення (Automatic Speech Recognition, ASR) являє собою міждисциплінарну галузь, що перебуває на перетині комп'ютерних наук, штучного інтелекту та обчислювальної лінгвістики. Її головне завдання полягає у перетворенні людського мовлення, представленого в аудіоформаті, на машинозчитуваний текст. По суті, система автоматичного розпізнавання мовлення є статистичною або нейромережевою моделлю, яка на основі вхідної послідовності акустичних ознак прагне з максимальною точністю відтворити відповідну текстову послідовність — слова або символи [3]. У класичному математичному формулюванні ця задача зводиться до пошуку такої послідовності слів, що максимізує апостеріорну ймовірність за теоремою Баєса, тобто модель має обрати найбільш вірогідну текстову гіпотезу серед усіх можливих, виходячи з наявного акустичного сигналу.

Сьогодні системи автоматичного розпізнавання мовлення відіграють критичну роль у побудові інтерфейсів взаємодії між людиною та комп'ютером. Вони лежать в основі роботи віртуальних асистентів, таких як голосові помічники у смартфонах та розумних пристроях, забезпечують функціонування систем автоматичного субтитрування відеоконтенту, а також є ключовим інструментом інклюзивного доступу до інформації для людей з порушеннями слуху [3]. Саме стрімкий розвиток цих технологій створив передумови для появи цілої індустрії автоматизованої генерації субтитрів, що є безпосереднім предметом дослідження даної роботи.

Історія розвитку автоматичного розпізнавання мовлення характеризується кількома парадигмальними зсувами, кожен із яких суттєво змінював як архітектуру систем, так і якість розпізнавання [4]. Перші експерименти у цій галузі відносяться до 1950–1960-х років, коли компанія IBM продемонструвала пристрій «Shoebox», здатний розпізнавати та реагувати на шістнадцять

вимовлених слів, включаючи цифри від нуля до дев'яти. На той час це було новаторське досягнення, яке, втім, мало суто демонстраційний характер і не могло бути застосоване у реальних умовах. У 1960-х роках у лондонському University College була створена система розпізнавання на основі ймовірнісного підходу, в якій уперше використовувалися ймовірності переходів між фонемами, що заклало теоретичне підґрунтя для подальших статистичних методів.

Значний крок уперед було зроблено у 1970-х роках, коли Агентство перспективних дослідницьких програм Міністерства оборони США профінансувало кілька конкуруючих проєктів із розробки систем розпізнавання мовлення. Переможцем стала система HARPY, розроблена в університеті Карнегі-Меллон, яка була здатна розпізнавати словник із тисячі слів із точністю дев'яносто відсотків. Саме в цей період для ймовірнісного моделювання слів мовлення вперше були застосовані приховані марковські моделі, які на десятиліття стали домінуючим підходом у галузі.

Аж до 2010-х років основою більшості промислових систем розпізнавання мовлення залишалася гібридна архітектура на основі прихованих марковських моделей та моделей гауссівських сумішей [3]. У такій архітектурі процес розпізнавання був строго розділений на окремі компоненти: акустичну модель на базі зв'язки «моделі гауссівських сумішей — приховані марковські моделі», фонетичний словник та статистичну N-грамну мовну модель. Для виділення ознак із аудіосигналу традиційно використовувалися мел-кепстральні коефіцієнти, які дозволяли компактно представити спектральні характеристики мовного сигналу. Попри свою ефективність та широке розповсюдження, цей підхід мав принциповий недолік — марковське припущення про умовну незалежність станів, яке не дозволяло моделі враховувати довгостроковий контекст у мовленні. Іншими словами, система аналізувала кожен фрагмент сигналу переважно ізольовано, що обмежувало її здатність коректно розпізнавати складні лінгвістичні конструкції.

Революція глибокого навчання, що відбулася з розвитком апаратного забезпечення та накопиченням великих обсягів даних, докорінно змінила

ландшафт технологій розпізнавання мовлення. На першому етапі моделі гауссівських сумішей в акустичних моделях були замінені глибокими нейронними мережами, що дозволило зменшити рівень помилок розпізнавання більш ніж удвічі. Згодом архітектури еволюціонували до використання згорткових нейронних мереж, які ефективно виділяли локальні патерни у спектрограмах мовного сигналу, та рекурентних нейронних мереж, зокрема мереж із довгою короткочасною пам'яттю, які значно краще моделювали часову динаміку мовлення та враховували контекст попередніх фрагментів.

Справжній прорив відбувся з появою наскрізних систем розпізнавання мовлення [5]. На відміну від класичного підходу, де акустична модель, фонетичний словник та мовна модель тренувалися окремо, наскрізні системи оптимізують єдину нейронну мережу, яка безпосередньо перетворює аудіосигнал на текст. Це принципово спростило архітектуру систем та усунуло необхідність ручного проєктування проміжних компонентів. Ключовими архітектурами наскрізного розпізнавання стали три підходи. По-перше, метод часової класифікації зв'язностей (англ. Connectionist Temporal Classification, CTC), який дозволяє тренувати мережу навіть за відсутності точного вирівнювання між акустичним сигналом та текстовою транскрипцією. По-друге, рекурентний нейромережевий перетворювач (англ. Recurrent Neural Network Transducer, RNN-T), який поєднує переваги рекурентних мереж із можливістю потокового розпізнавання в реальному часі. По-третє, моделі на основі механізму уваги (англ. Attention-based Encoder-Decoder), де кодувальник формує проміжне представлення аудіосигналу, а декодер авторегресивно генерує текстову послідовність, використовуючи механізм уваги для динамічного фокусування на релевантних частинах вхідного сигналу.

Сучасний етап розвитку автоматичного розпізнавання мовлення, що розпочався приблизно з 2017 року, визначається безумовним домінуванням архітектури Transformer (далі – трансформер) [6]. Завдяки механізму внутрішньої уваги трансформери здатні захоплювати глобальні контекстуальні залежності в межах усієї аудіопослідовності, при цьому уникаючи проблем зі

згасанням градієнта, що були притаманні рекурентним нейронним мережам. Це означає, що модель може ефективно «бачити» весь контекст висловлювання одночасно, а не покладатися на послідовну обробку фрагментів сигналу. Подальша еволюція трансформерних архітектур привела до появи моделей типу Conformer, які комбінують згорткові шари з механізмом уваги. Такий гібридний підхід дозволяє одночасно ефективно виділяти як локальні акустичні ознаки за допомогою згорткових операцій, так і глобальні контекстуальні залежності за допомогою механізму уваги, що забезпечує суттєве покращення якості розпізнавання.

Для об'єктивної оцінки якості систем автоматичного розпізнавання мовлення в науковому та інженерному середовищі використовується метрика Word Error Rate (WER), яка визначається як відношення суми трьох типів помилок — замін, видалень та вставок слів — до загальної кількості слів в еталонному тексті [3]. Вона обчислюється за формулою (1.1):

$$WER = \frac{S + D + I}{N}, \tag{1.1}$$

де S – кількість замін; D – кількість видалень; I – кількість вставок; N – загальна кількість слів в еталонному тексті.

Чим нижчим є значення цього показника, тим точніше працює система. Для англійської мови найкращі сучасні моделі досягають рівня помилок у чотири-шість відсотків [5], що наближається до якості людської транскрипції. Водночас для мов із багатою морфологією, зокрема для української мови, ситуація є значно складнішою. Українська мова характеризується розвиненою флективною системою — наявністю відмінків, родів, великою кількістю суфіксів, що породжує величезну різноманітність словоформ. За результатами досліджень 2024–2025 років, одна з найкращих сучасних моделей демонструє рівень помилок для української мови приблизно на рівні шістнадцяти-сімнадцяти відсотків на наборі даних Mozilla Common Voice, що є суттєво вищим

показником порівняно з англійською мовою та свідчить про наявність значного простору для подальших досліджень та вдосконалень [7].

Таким чином, еволюція технологій автоматичного розпізнавання мовлення пройшла шлях від найпростіших детекторів окремих слів через статистичні моделі на основі прихованих марковських ланцюгів до сучасних наскрізних нейромережевих архітектур на базі трансформерів. Кожен парадигмальний зсув супроводжувався значним зниженням рівня помилок розпізнавання та розширенням сфер застосування цих технологій. Саме досягнення сучасного етапу розвитку, зокрема поява потужних багатомовних моделей із підтримкою десятків мов, зробили можливим створення повністю автоматизованих систем генерації субтитрів для веб-платформ.

1.2 Роль субтитрів у сучасному цифровому контенті

Споживання медіаконтенту за останні роки зазнало кардинальних трансформацій, зумовлених стрімкою експансією платформ коротких відеороликів. Такі сервіси, як TikTok, Instagram Reels та YouTube Shorts, сформували принципово новий формат взаємодії з аудиторією, де відеоролик тривалістю від п'ятнадцяти секунд до трьох хвилин є основною одиницею контенту. У цьому середовищі субтитри пережили фундаментальну трансформацію своєї ролі: з допоміжного інструменту забезпечення доступності для людей із порушеннями слуху вони перетворилися на один із ключових елементів залучення та утримання уваги глядача.

Ця трансформація пояснюється зміною моделі споживання відео. Значна частина користувачів переглядає контент у ситуаціях, коли увімкнення звуку є незручним або неможливим — у громадському транспорті, на робочому місці, в черзі чи будь-якому іншому публічному просторі. За результатами досліджень, приблизно вісімдесят п'ять відсотків усіх відео у соціальній мережі Facebook переглядаються без звуку, а вісімдесят відсотків глядачів негативно реагують на відеорекламу, яка автоматично відтворюється з увімкненим гучним звуком [8].

Ці цифри наочно демонструють, що для переважної більшості аудиторії саме текстова складова відеоролика стає першим і часто єдиним каналом сприйняття інформації під час швидкого скролінгу стрічки новин.

Вплив субтитрів на ключові метрики залучення аудиторії є статистично значущим. Дослідження показують, що наявність субтитрів здатна підвищити кількість переглядів відео приблизно на сорок відсотків та збільшити рівень додивляння на вісімдесят відсотків [9]. Ці показники мають пряме економічне значення для контент-мейкерів, маркетологів та бізнесів, оскільки алгоритми рекомендацій більшості платформ враховують саме рівень утримання глядача як один із головних факторів ранжування контенту. Крім того, субтитри забезпечують суттєві переваги для пошукової оптимізації відеоконтенту, оскільки пошукові системи здатні індексувати текстові дані, роблячи відеоролики доступними для текстових пошукових запитів [10]. Таким чином, відео із субтитрами має вищу вірогідність бути знайденим органічно, без додаткових витрат на просування.

Окремим важливим аспектом є роль субтитрів у забезпеченні інклюзивності та доступності цифрового контенту. Міжнародні нормативні вимоги, зокрема стандарт веб-доступності Web Content Accessibility Guidelines (WCAG) та американський закон Americans with Disabilities Act (ADA), дедалі активніше регулюють питання обов'язкового субтитрування відеоматеріалів. Ці вимоги поширюються не лише на державні установи та освітні платформи, а й на комерційні вебсайти та медіаресурси. Субтитри є необхідними не лише для людей із порушеннями слуху, а й для тих, хто вивчає мову відео як іноземну, для літніх людей із віковим зниженням слуху, а також для будь-якого користувача в шумному середовищі. Забезпечення доступності контенту, таким чином, є не лише етичним обов'язком, а й прагматичним рішенням, що розширює потенційну аудиторію.

Важливим трендом останніх років стало поширення так званих гіпердинамічних субтитрів, в яких кожне слово підсвічується синхронно з його вимовою, створюючи візуальний ефект, подібний до караоке. Цей формат став

фактичним індустріальним стандартом у сфері маркетингу в соціальних мережах, оскільки він знижує когнітивне навантаження на глядача, утримує його увагу під час швидкого скролінгу стрічки та значно зменшує ймовірність пропуску відео. Динамічне виділення слів допомагає глядачу синхронізувати візуальне сприйняття тексту з аудіорядом, що покращує засвоєння інформації та створює відчуття залученості до контенту. Технічно реалізація таких субтитрів вимагає точної синхронізації на рівні окремих слів, що є значно складнішим завданням порівняно з традиційним посегментним субтитруванням і потребує використання відповідних форматів, здатних підтримувати складні анімації.

Описані тенденції безпосередньо впливають на формування та розвиток ринку автоматичної генерації субтитрів на основі штучного інтелекту, який наразі перебуває у фазі активного зростання. За оцінками аналітичних агентств, обсяг глобального ринку рішень для генерації субтитрів на базі штучного інтелекту у 2024–2025 роках становив від 817 мільйонів до 1,12 мільярда доларів США. Прогнози на наступне десятиліття свідчать про стрімке зростання: очікується, що до 2033–2035 років ринок досягне обсягу від 8,13 до 18,22 мільярдів доларів США. Сукупний середньорічний темп зростання при цьому прогнозується на надзвичайно високому рівні — від двадцяти двох до тридцяти шести відсотків, що є одним із найвищих показників серед суміжних технологічних сегментів [11].

Структура цього ринку має кілька характерних особливостей. Домінуючою технологією, що лежить в основі більшості рішень, є саме автоматичне розпізнавання мовлення, яке займає приблизно шістдесят дев'ять відсотків ринку. Переважна більшість рішень, а саме близько вісімдесяти дев'яти відсотків, постачається у формі хмарних сервісів за моделлю підписки, що зумовлено високою обчислювальною вимогливістю процесу розпізнавання мовлення та рендерингу відео. Основними споживачами є сегменти медіа та розваг, маркетинг у соціальних мережах, а також корпоративне навчання та електронна освіта [11].

Зростання ринку стимулюється одночасно кількома потужними факторами. Перш за все, це експоненційне збільшення обсягів створення відеоконтенту — дедалі більше малих бізнесів, індивідуальних контент-мейкерів та освітніх установ починають регулярно створювати відеоматеріали для своїх аудиторій. По-друге, посилення нормативних вимог щодо доступності медіа змушує організації забезпечувати субтитрування свого контенту. По-третє, потреба у багатомовній локалізації контенту для виходу на глобальні ринки зростає паралельно з глобалізацією цифрової економіки. Нарешті, суттєвим каталізатором є економічна ефективність автоматизованих рішень: використання штучного інтелекту для субтитрування дозволяє значно знизити витрати на транскрипцію, що робить субтитрування доступним навіть для невеликих авторів із обмеженим бюджетом.

Описана ринкова динаміка та об'єктивна потреба у якісних, доступних та гнучких інструментах автоматичного субтитрування створюють сприятливе підґрунтя для дослідження та розробки нових систем у цій галузі. Особливо актуальною залишається проблема підтримки мов із обмеженими ресурсами, зокрема української, для якої більшість комерційних платформ не забезпечують достатньої якості розпізнавання.

1.3 Порівняльний аналіз існуючих рішень

Задача автоматичного розпізнавання мовлення та генерації субтитрів може бути вирішена із використанням двох принципово різних категорій засобів. Першу категорію становлять комерційні платформи, що надають готовий продукт для кінцевого користувача. Другу — інструменти з відкритим вихідним кодом, кожен із яких вирішує окрему складову загального конвеєра обробки. Для обґрунтування доцільності розробки власної системи необхідно проаналізувати обидва сегменти — не лише з погляду функціональних можливостей, а передусім з погляду архітектурної прозорості та можливості відтворення запропонованих рішень.

У комерційному сегменті провідними платформами генерації субтитрів для коротких відеороликів є VEED.io, Kapwing, Submagic та Captions.ai. Кожна з них пропонує веб-інтерфейс або мобільний застосунок, через який користувач завантажує відео, отримує автоматично згенеровані субтитри та може експортувати результат. Платформи розрізняються за цільовою аудиторією та набором додаткових функцій. VEED.io та Kapwing позиціонують себе як комплексні хмарні відеоредактори з розвиненими можливостями командної роботи, включаючи спільне редагування проєктів у реальному часі, управління бренд-кітами та автоматичне видалення пауз і тиші у відео. Submagic та Captions.ai є вузькоспеціалізованими інструментами, сфокусованими безпосередньо на генерації анімованих субтитрів з ефектами підсвічування, автоматичним додаванням тематичних емодзі та підбором супровідних відеофрагментів [12]. Ці платформи декларують підтримку від сорока восьми до понад ста двадцяти п'яти мов розпізнавання та забезпечують повний цикл обробки — від завантаження відео до експорту результату із вшитими субтитрами.

Незважаючи на відмінності у функціоналі для кінцевого користувача, з точки зору інженерії програмного забезпечення усі чотири платформи мають спільну фундаментальну рису — жодна з них не розкриває деталей своєї внутрішньої реалізації. Неможливо з'ясувати, як саме організовано конвеєр обробки від моменту завантаження відео до отримання готового результату: чи є обробка синхронною або асинхронною, як розподілено відповідальність між компонентами системи, яким чином забезпечується масштабування при зростанні кількості одночасних запитів і які механізми стійкості до відмов застосовуються. Це не є критикою бізнес-моделі цих продуктів — комерційні платформи за своєю природою не зобов'язані розкривати внутрішню реалізацію. Проте з позиції розробника чи дослідника, який прагне побудувати подібну систему, це означає, що досвід цих платформ не може бути використаний як основа для проєктування. Окремо слід зазначити, що якість підтримки української мови у розглянутих платформах залишається недостатньою: на

практиці спостерігаються помилки розстановки розділових знаків, некоректна сегментація тексту та загальне зниження точності транскрипції порівняно з англомовним контентом.

Другу категорію засобів становлять інструменти з відкритим вихідним кодом, які вирішують окремі складові загальної задачі субтитрування. Найбільш значущим відкритим інструментом у галузі розпізнавання мовлення є модель Whisper від OpenAI, вихідний код та ваги якої опубліковані у відкритому доступі, що дозволяє використовувати її як локально, так і через комерційний прикладні програмні інтерфейси (англ. Application Programming Interface, API) [13]. Навколо базової моделі сформувалася розвинена екосистема похідних проєктів: faster-whisper, whisper.cpp, WhisperX тощо. Для обробки відео та накладання субтитрів індустріальним стандартом є FFmpeg — мультимедійний фреймворк із відкритим вихідним кодом, який підтримує вилучення аудіодоріжки з відеофайлу, конвертацію між форматами та вшивання субтитрів безпосередньо у пікселі відеокадру через систему відеофільтрів [14]. Для програмної роботи з форматами субтитрів існують спеціалізовані бібліотеки, зокрема pysubs2 для мови Python, яка дозволяє генерувати та маніпулювати файлами форматів SubRip (SRT), WebVTT (VTT) та Advanced Sub Station Alpha (ASS) [15].

Кожен із перелічених інструментів є технологічно зрілим, добре документованим та активно підтримуваним спільнотою розробників. Проте всі вони вирішують лише свою ізольовану частину загальної задачі. Whisper перетворює аудіо на текст із мітками часу, FFmpeg вшиває субтитри у відео, pysubs2 генерує файли субтитрів у потрібному форматі. Між ними відсутній сполучний шар — архітектурне рішення, яке б об'єднувало ці компоненти у цілісний конвеєр обробки, придатний для функціонування у веб-середовищі. Аналіз відкритих репозиторіїв свідчить, що більшість існуючих проєктів обмежуються скриптами командного рядка, які послідовно викликають Whisper та FFmpeg на локальній машині розробника. Жоден із відомих відкритих проєктів не пропонує завершеної системи, яка б одночасно забезпечувала веб-інтерфейс для завантаження відео, асинхронну обробку через чергу повідомлень,

автоматичне розпізнавання мовлення, генерацію субтитрів, серверний рендеринг відео з вшитими субтитрами, управління станом обробки через базу даних та декларативне розгортання всієї хмарної інфраструктури.

Узагальнюючи результати аналізу обох сегментів, можна констатувати наявність чіткої прогалини. У комерційному сегменті задача субтитрування вирішена на рівні кінцевого продукту, однак архітектурний досвід цих рішень залишається недоступним для вивчення та відтворення. У сегменті відкритого програмного забезпечення наявні всі необхідні компоненти для побудови подібної системи, проте вони існують як розрізнені інструменти без інтеграційного архітектурного рішення рівня веб-платформи. Додатковою проблемою залишається недостатня якість підтримки української мови у більшості комерційних рішень. Ці обставини безпосередньо визначають напрям даного дослідження — розробку документованого архітектурного рішення, яке поєднує зрілі відкриті інструменти у єдиний конвеєр обробки з прозорою модульною структурою, обґрунтуванням кожного проєктного вибору та можливістю повного відтворення системи з нуля.

1.4 Обґрунтування напряму дослідження

Аналіз, проведений у попередніх підрозділах, дозволяє окреслити напрям дослідження на кількох взаємопов'язаних рівнях.

На рівні технологій встановлено, що автоматичне розпізнавання мовлення за останнє десятиліття зазнало кількох парадигмальних зсувів — від класичних статистичних моделей до наскрізних нейромережевих архітектур на базі трансформерів — і досягло рівня якості, достатнього для практичного застосування у задачах автоматичного субтитрування. Сучасні моделі, зокрема Whisper від OpenAI, підтримують десятки мов, включаючи українську, та демонструють стійкість до фонового шуму і варіативності акустичних умов.

На рівні ринку та соціального попиту продемонстровано, що потреба в автоматизованих інструментах генерації субтитрів зростає надзвичайно

високими темпами. Зміна моделі споживання відеоконтенту, де переважна більшість відео переглядається без звуку, перетворила субтитри з допоміжного інструменту доступності на фундаментальний елемент залучення аудиторії. Посилення нормативних вимог щодо доступності медіа та потреба у багатомовній локалізації контенту додатково стимулюють розвиток цього сегменту.

На рівні існуючих рішень виявлено ключову прогалину. У комерційному сегменті задача субтитрування вирішена на рівні кінцевого продукту такими платформами, як VEED.io, Kapwing, Submagic та Captions.ai, проте їхні архітектурні рішення залишаються повністю закритими та недоступними для вивчення чи відтворення. У сегменті відкритого програмного забезпечення наявні всі необхідні компоненти для побудови подібної системи — модель розпізнавання мовлення Whisper, мультимедійний фреймворк FFmpeg, бібліотеки генерації субтитрів — проте вони існують як розрізнені інструменти без інтеграційного архітектурного рішення рівня веб-платформи. Додатковою проблемою залишається недостатня якість підтримки української мови у більшості наявних комерційних рішень.

Зазначені обставини у сукупності визначають напрям даного дослідження — розробку документованого архітектурного рішення та його практичну реалізацію у вигляді діючої веб-платформи, яка поєднує відкриті інструменти у єдиний конвеєр обробки медіаконтенту з прозорою модульною структурою, обґрунтуванням кожного проєктного вибору та можливістю повного відтворення системи з нуля.

2 ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ТА МЕТОДІВ РОЗВ'ЯЗАННЯ ЗАДАЧ

2.1 Порівняльний аналіз моделей розпізнавання мовлення

Вибір моделі автоматичного розпізнавання мовлення є одним із ключових архітектурних рішень для системи генерації субтитрів, оскільки саме від якості транскрипції безпосередньо залежить кінцева цінність продукту для користувача. У межах даного дослідження було проведено порівняльний аналіз п'яти провідних систем, які наразі підтримують українську мову: OpenAI Whisper, Google Cloud Speech-to-Text (моделі Chirp), AWS Transcribe, AssemblyAI та Deepgram. Аналіз проводився за критеріями, що є найбільш релевантними для задачі побудови веб-платформи субтитрування з підтримкою української мови, а саме: архітектура та кількість параметрів моделі, якість розпізнавання для української мови, наявність відкритих бенчмарків, вартість використання через API, а також специфічні обмеження кожної системи.

З точки зору архітектурних підходів, розглянуті системи представляють різні парадигми побудови моделей розпізнавання мовлення. Whisper побудований на архітектурі трансформер у конфігурації «кодер-декодер» і навчений на масиві даних обсягом близько п'яти мільйонів годин аудіо у мультизадачному режимі, що дозволяє одній моделі виконувати транскрипцію, переклад, ідентифікацію мови та передбачення часових міток [1]. Google Chirp базується на архітектурі Conformer із двома мільярдами параметрів і триетапним процесом навчання, що включає самонавчання на дванадцяти мільйонах годин немаркованого аудіо [16]. AssemblyAI використовує зв'язку Conformer-кодера з декодером на основі рекурентного нейромережевого перетворювача, що, за заявою розробників, забезпечує значно меншу схильність до галюцинацій порівняно з авторегресивним декодером Whisper [17]. AWS Transcribe та Deepgram використовують пропрієтарні архітектури, деталі яких не розкриваються у публічному доступі.

Критичним фактором для даного дослідження є якість розпізнавання саме української мови. Тут виявляється принципова відмінність між системами: жоден із комерційних провайдерів не публікує верифікованих показників якості розпізнавання для української мови. Єдиною системою, для якої існують прозорі та відтворювані бенчмарки на українських даних, є Whisper [18]. За результатами спільнотних бенчмарків на датасеті Common Voice 10, модель Whisper large-v2 досягає показника Word Error Rate на рівні 13,72 відсотка для української мови, що є найкращим результатом серед усіх протестованих конфігурацій без додаткового дотренування. Цікавим є те, що новіша версія large-v3 продемонструвала регресію якості на українській мові до 20,53 відсотка, що, ймовірно, пов'язано зі зміною розподілу навчальних даних у розширеному датасеті. Водночас дотреновані варіанти Whisper досягають ще кращих результатів, зокрема окремі моделі спільноти демонструють показник нижче 11 відсотків на Common Voice 11.

Щодо підтримки української мови комерційними сервісами варто зазначити, що вона є обмеженою. Google Chirp 2 не підтримує потокове розпізнавання для української, пропонуючи лише пакетний режим; ця проблема вирішена у Chirp 3 [19]. AWS Transcribe додав підтримку потокового режиму для української лише у жовтні 2024 року, проте такі функції як редагування персональних даних та користувацькі мовні моделі для української досі недоступні [20]. AssemblyAI підтримує українську лише через модель Universal-2, тоді як новіша Universal-3 Pro обмежена шістьма мовами [21].

Оскільки в межах розроблюваної системи передбачається використання моделі розпізнавання мовлення через API-інтерфейс, а не у режимі локального розгортання, суттєвим критерієм є вартість такого використання. У таблиці 2.1 наведено зведене порівняння усіх п'яти систем за ключовими характеристиками, включаючи вартість обробки через API.

Таблиця 2.1 - Порівняльна характеристика ASR моделей

Назва	Архітектура	Відкриті ваги моделі	WER для української мови, %	Вартість обробки, доларів/хв
OpenAI Whisper	Transformer	Так	13,72	0,006
Google Cloud STT	Conformer	Ні	Не опубліковано	0,016
AWS Transcribe	Пропрієтарна	Ні	Не опубліковано	0,024
AssemblyAI	Conformer + RNN-T	Ні	Не опубліковано	0,0025
Deepgram	Transformer	Ні	Не опубліковано	0,0066

На підставі проведеного аналізу, для розроблюваної системи автоматичного розпізнавання мовлення та генерації субтитрів було обрано модель OpenAI Whisper, яка використовуватиметься через офіційний API OpenAI. Це рішення обґрунтовується кількома ключовими факторами. По-перше, Whisper є єдиною системою з прозорими та відтворюваними бенчмарками для української мови, що є критично важливим для дослідження. По-друге, вартість використання через API складає 0,006 долара за хвилину, що є помірною ціною, значно нижчою за Google Cloud STT та AWS Transcribe, і при цьому забезпечує найкращу задокументовану якість розпізнавання української мови. По-третє, наявність відкритих ваг моделі створює перспективу подальшої оптимізації та дотренування на українських даних, а також можливість переходу до самостійного розгортання у разі масштабування системи. По-четверте, обрання Whisper дозволяє валідувати результати роботи системи, порівнюючи їх із існуючими спільнотними бенчмарками на Common Voice, що забезпечує відтворюваність дослідження.

Варто також відзначити, що відома проблема галюцинацій Whisper, коли модель генерує фіктивний текст на ділянках тиші або неякісного аудіо, не є критичною для задачі субтитрування відеоконтенту, оскільки у типовому сценарії використання аудіодоріжка містить безперервне мовлення. Тим не менш, у процесі проєктування системи потенційно може бути реалізовано механізми постобробки, що мінімізують вплив цього артефакту на якість кінцевих субтитрів.

2.2 Технології обробки відеофайлів та формати субтитрів

Система автоматичної генерації субтитрів для веб-платформ передбачає не лише розпізнавання мовлення, а й комплекс операцій із медіафайлами, зокрема вилучення аудіодоріжки з відео, формування файлів субтитрів у відповідному форматі та, за потребою, накладання тексту безпосередньо на відео. Центральним інструментом для виконання цих операцій є FFmpeg — мультимедійний фреймворк з відкритим вихідним кодом, що є визнаним індустріальним стандартом для декодування, кодування, транскодування, мультиплексування та фільтрації аудіоконтенту та відеоконтенту [14].

У контексті розроблюваної системи FFmpeg виконує дві ключові функції. Перша — це вилучення аудіо з вхідного відеофайлу для подальшої передачі моделі розпізнавання мовлення. На цьому етапі аудіодоріжка конвертується у формат, оптимальний для обробки моделлю Whisper. Друга функція — це операція вбудовування субтитрів безпосередньо у відеопотік. Результатом цієї операції є відеофайл, у якому текст субтитрів є невід'ємною частиною зображення і не може бути вимкнений засобами програвача. Така технологія є необхідною для публікації контенту у соціальних мережах, зокрема TikTok та Instagram Reels, де програвачі не підтримують зовнішні файли субтитрів.

Вибір формату субтитрів суттєво впливає на можливості стилізації та візуального оформлення тексту на відео. На сьогодні найбільш поширеними

форматами є SRT, VTT та ASS, кожен з яких має свої характеристики та сферу застосування.

Формат SRT є найпростішим і водночас найпоширенішим форматом субтитрів [22]. Його структура складається з порядкового номера сегмента, пари часових міток (початок і кінець відображення) та текстового вмісту. SRT не підтримує стилізацію, позиціонування тексту на екрані чи будь-які візуальні ефекти, що обмежує його застосування у випадках, де потрібне складне оформлення субтитрів.

Формат VTT був стандартизований як частина специфікації HTML5 і призначений для використання у веб-відеопрогравачах [23]. Він розширює можливості SRT базовим стилізуванням, зокрема підтримкою жирного шрифту, курсиву та вирівнювання тексту. Саме цей формат використовується у розроблюваній системі для програмного відображення субтитрів у веб-плеєрі, оскільки він є нативним форматом для елемента track у HTML5 і не потребує додаткових бібліотек для рендерингу у браузері.

Формат ASS є найбільш функціональним серед розглянутих форматів [24]. Він надає можливість точного позиціонування тексту за абсолютними координатами на екрані, використання довільних шрифтів, ефектів тіні та обведення, а також — що є найважливішим для платформ коротких відео — складних анімацій на рівні окремих слів. Зокрема, саме формат ASS дозволяє реалізувати так званий «караоке-ефект», при якому кожне слово підсвічується або змінює колір синхронно з моментом його вимови у аудіодоріжці. Такий тип гіпердинамічних субтитрів став фактичним стандартом у сфері контенту для соціальних мереж, оскільки значно підвищує рівень залучення та утримання уваги глядача.

Процес вбудовування субтитрів у FFmpeg реалізується за допомогою механізму графів відеофільтрів [14]. Для простих форматів SRT та VTT використовується фільтр subtitles, який завантажує файл субтитрів і виконує рендеринг тексту на кожному відповідному кадрі. Для формату ASS застосовується фільтр ass, який забезпечує повну підтримку розширеного

форматування та анімацій, закладених у файл субтитрів. Вибір між цими фільтрами визначається складністю візуального оформлення, яке потрібно відтворити.

Окремим аспектом є оптимізація процесу кодування відео після накладання субтитрів. За замовчуванням FFmpeg використовує програмний кодек libx264, який забезпечує високу якість стиснення, але є ресурсомістким з точки зору процесорного часу. Альтернативою є апаратно-прискорені кодеки, зокрема h264_nvenc для графічних прискорювачів NVIDIA, що дозволяє суттєво скоротити час кодування за рахунок делегування обчислень на Graphics Processing Unit (GPU). У контексті хмарного розгортання системи, де обробка відео виконується воркерами на базі контейнерів, вибір між програмним та апаратним кодуванням залежить від конфігурації обчислювальних ресурсів та економічної доцільності використання контейнерів з підтримкою GPU.

Таким чином, у розроблюваній системі для роботи із субтитрами обрано єдиний формат — VTT. Це рішення обумовлене кількома факторами. По-перше, VTT є стандартизованим форматом для веб-середовища, що нативно підтримується елементом у HTML5 і не потребує додаткових бібліотек чи плагінів для відображення субтитрів у браузерному відеоплеєрі. По-друге, OpenAI Whisper API підтримує генерацію результатів транскрипції безпосередньо у форматі VTT [25], що усуває потребу у додатковому етапі конвертації між форматами та спрощує конвеєр обробки. По-третє, FFmpeg повноцінно підтримує VTT як вхідний формат для операції вбудовування субтитрів через фільтр subtitles, що дозволяє використовувати один і той самий файл субтитрів як для відображення у веб-плеєрі, так і для вшивання у відеопотік при генерації фінального файлу для експорту. Використання єдиного формату на всіх етапах конвеєра обробки — від отримання результатів транскрипції до фінального рендерингу — зменшує складність системи, мінімізує ймовірність помилок при перетворенні між форматами та спрощує підтримку і розвиток програмного забезпечення.

2.3 Архітектурні підходи до систем обробки мультимедіа

Обробка мультимедійних даних є обчислювально вимогливим завданням, яке потребує ретельного вибору архітектурного підходу. Операції вилучення аудіо, виклику зовнішнього API розпізнавання мовлення та рендерингу відео із вбудованими субтитрами можуть тривати від десятків секунд до кількох хвилин залежно від тривалості та якості вхідного відеофайлу. Це означає, що архітектура системи повинна забезпечувати асинхронність обробки, стійкість до відмов та можливість незалежного масштабування обчислювальних компонентів. У межах даного дослідження було проаналізовано три основні архітектурні підходи — монолітний, мікросервісний та безсерверний — з точки зору їх придатності для побудови веб-платформи генерації субтитрів.

Монолітна архітектура, при якій інтерфейс користувача, бізнес-логіка та обробка відео зведені в єдиний виконуваний модуль, є прийнятним рішенням лише на етапі прототипування. При збільшенні навантаження моноліт стає вузьким місцем системи, оскільки ресурсомісткі процеси рендерингу відео навантажують процесор та оперативну пам'ять, що призводить до деградації продуктивності веб-додатку в цілому і погіршення досвіду всіх користувачів.

Мікросервісна архітектура вирішує цю проблему шляхом ізоляції компонентів у незалежні сервіси, кожен з яких може масштабуватися окремо. Веб-сервер, що обробляє HTTP-запити користувачів, і сервіси-воркери, що виконують важкі обчислення з медіафайлами, функціонують як окремі процеси або контейнери. Це означає, що збій одного воркера через пошкоджений відеофайл не впливає на стабільність усієї платформи, а кількість воркерів може динамічно збільшуватися у відповідь на зростання черги завдань.

Безсерверні архітектури, такі як AWS Lambda або Google Cloud Functions, пропонують привабливу економічну модель з оплатою лише за фактичний час виконання. Проте для задач медіа-обробки вони мають критичні обмеження. Передусім, це жорсткий ліміт на тривалість виконання функції, який зазвичай складає п'ятнадцять хвилин, що може бути недостатнім для обробки довгих

відеозаписів [26]. Крім того, безсерверні платформи, як правило, не надають доступу до графічних прискорювачі GPU, що унеможливлює апаратне прискорення кодування відео. Нарешті, явище холодного старту, коли ініціалізація нового контейнера з важкими бібліотеками на кшталт FFmpeg вносить додаткову затримку, негативно впливає на загальний час обробки [27].

Оптимальним компромісом для розроблюваної системи є гібридний підхід, що поєднує переваги мікросервісної декомпозиції з використанням безсерверних контейнерів. Зокрема, платформа Google Cloud Run дозволяє розгортати контейнеризовані сервіси, які автоматично масштабуються, при цьому не мають жорстких обмежень на час виконання, характерних для класичних безсерверних функцій [28].

Ключовим архітектурним патерном для побудови конвеєра медіа-обробки є патерн Web-Queue-Worker, також відомий як Asynchronous Request-Reply [29]. Його необхідність зумовлена тим, що рендеринг відео з субтитрами може тривати значно довше за тайм-аут стандартного HTTP-запиту, який зазвичай становить тридцять-шістдесят секунд. Синхронна обробка в таких умовах є неприпустимою, оскільки клієнтський запит буде розірвано через перевищення тайм-ауту задовго до завершення обробки.

Механізм роботи патерну Web-Queue-Worker у контексті розроблюваної системи полягає в наступному. Клієнтський додаток завантажує відеофайл безпосередньо у хмарне сховище та надсилає HTTP-запит до API-сервісу. API-сервіс зберігає метадані проєкту в базі даних, публікує повідомлення у брокер повідомлень і миттєво повертає клієнту відповідь з успішним статусом, що сигналізує про прийняття завдання в обробку. Далі окремий сервіс-воркер асинхронно отримує повідомлення з черги, завантажує відеофайл зі сховища, вилучає аудіодоріжку, надсилає її до OpenAI Whisper API для транскрипції, зберігає згенерований файл субтитрів у сховищі та оновлює статус проєкту в базі даних. Клієнтський додаток періодично опитує API щодо статусу обробки і відображає результат, коли той стає доступним.

Окрім базового патерну асинхронної обробки, для забезпечення надійності системи необхідно застосовувати додаткові інженерні патерни. Першим із них є принцип ідемпотентності обробки. Хмарні брокери повідомлень зазвичай гарантують доставку за принципом «принаймні один раз», що означає можливість повторної доставки одного й того самого повідомлення у разі мережевих затримок або збоїв [30]. Воркери повинні бути спроєктовані таким чином, щоб повторна обробка одного й того самого завдання не призводила до дублювання результатів, подвійного виклику зовнішніх API або некоректного оновлення стану в базі даних. На практиці це досягається перевіркою поточного статусу проєкту перед початком обробки та використанням унікальних ідентифікаторів.

Другим важливим патерном є Dead Letter Queue, що забезпечує стійкість системи до так званих «отруєних повідомлень» [31]. Якщо відеофайл пошкоджено або FFmpeg не може його обробити, воркер виконує кілька спроб обробки з експоненційною затримкою між ними. Якщо всі спроби вичерпано, повідомлення автоматично перенаправляється у спеціальну чергу «мертвих повідомлень», а не повертається назад у основну чергу. Це запобігає нескінченному циклічному перебиранню одного пошкодженого завдання, яке могло б заблокувати обробку інших відео, і водночас зберігає інформацію про помилку для подальшого аналізу адміністраторами.

Третім патерном, що застосовується у контексті передачі медіафайлів між компонентами системи, є Claim-Check [32]. Замість передачі самого відеофайлу через чергу повідомлень, що є неприпустимим з огляду на обмеження розміру повідомлень у більшості брокерів, через чергу передається лише посилання на файл у хмарному сховищі. Воркер, отримавши повідомлення, використовує це посилання для завантаження файлу безпосередньо зі сховища. Такий підхід розділяє площину передачі метаданих та площину передачі медіаданих, що забезпечує ефективне використання інфраструктури черг повідомлень та зменшує навантаження на брокер.

Таким чином, для розроблюваної системи автоматичного розпізнавання мовлення та генерації субтитрів обрано гібридну мікросервісну архітектуру з асинхронною комунікацією через чергу повідомлень. Система декомпозується на три основних компоненти: веб-додаток для інтерфейсу користувача, API-сервіс для обробки запитів та управління даними, і сервіс-воркер для виконання обчислювально вимогливих операцій з медіафайлами. Комунікація між API-сервісом та воркером відбувається за патерном Web-Queue-Worker із застосуванням принципів ідемпотентності, Dead Letter Queue та Claim-Check, що в сукупності забезпечує масштабованість, стійкість до відмов та ефективне використання ресурсів. Більш детальний огляд інтеграції архітектурного рішення у систему наданий у розділі 3 цієї роботи.

2.4 Аналіз хмарних технологій та інструментів інфраструктури як коду

Реалізація архітектури, обґрунтованої у попередніх підрозділах, потребує конкретного набору хмарних сервісів, що забезпечать зберігання медіафайлів, асинхронну комунікацію між компонентами, виконання контейнеризованих сервісів та зберігання реляційних даних. У межах даного дослідження було проведено порівняльний аналіз трьох провідних хмарних платформ — Google Cloud Platform, Amazon Web Services та Microsoft Azure — з акцентом на вартість, поріг входження, зручність розгортання та можливості масштабування обчислювальних сервісів.

Усі три провайдери пропонують функціонально еквівалентні базові сервіси для побудови системи медіа-обробки: об'єктне сховище з підтримкою прямого завантаження через підписані посилання, брокер повідомлень із гарантованою доставкою та підтримкою Dead Letter Queue, а також середовище для виконання контейнерів. Проте суттєві відмінності виявляються саме на рівні вартості, складності конфігурації та зручності інтеграції компонентів між собою. З точки зору порогу входження та початкових витрат, Google Cloud Platform пропонує кредит у розмірі трьохсот доларів на дев'яносто днів для нових

користувачів, що є найбільш дешевою пропозицією серед трьох платформ [33]. Amazon Web Services надає дванадцятимісячний пробний період із лімітованим доступом до окремих сервісів, проте безкоштовні ліміти для обчислювальних контейнерних сервісів є мінімальними [34]. Microsoft Azure пропонує кредит у двісті доларів, але лише на тридцять днів, що є найкоротшим пробним періодом [35].

Ключовою відмінністю між платформами для розроблюваної системи є модель розгортання та масштабування контейнерних сервісів. Google Cloud Run дозволяє розгортати контейнери з автоматичним масштабуванням від нуля екземплярів, що означає повну відсутність витрат у періоди, коли система не обробляє запити [28]. Розгортання нового сервісу потребує лише Docker-образу та однієї команди, без необхідності попередньо конфігурувати кластери, мережеві балансувальники або групи автомасштабування. Аналогічний сервіс в екосистемі Amazon Web Services (AWS) — Fargate або App Runner — потребує значно більшої кількості конфігураційних кроків: визначення Task Definition, створення кластеру ECS, налаштування Application Load Balancer та цільових груп [36]. Azure Container Apps функціонально наближається до Cloud Run за простотою, проте екосистема Azure в цілому є більш орієнтованою на корпоративний сегмент із відповідною складністю управління ресурсами та політиками доступу [37].

Окремим важливим аспектом є інтеграція брокера повідомлень із обчислювальними сервісами. Google Cloud Pub/Sub нативно інтегрується з Cloud Run через механізм push-підписки: повідомлення автоматично доставляються воркеру у вигляді HTTP-запиту без необхідності писати код для опитування черги [38]. Це суттєво спрощує реалізацію патерну Web-Queue-Worker. Крім того, Pub/Sub об'єднує функціональність публікації та підписки в єдиному сервісі, тоді як в AWS аналогічна архітектура потребує конфігурації двох окремих сервісів — SNS для публікації та SQS для черг — з додатковим налаштуванням зв'язку між ними [39]. Azure Service Bus пропонує розширені

транзакційні можливості, які є перевагою для фінансових систем, але є надлишковими для задач медіа-обробки [40].

У таблиці 2.2 наведено зведене порівняння хмарних сервісів трьох провайдерів за характеристиками, що є ключовими для розроблюваної системи.

Таблиця 2.2 - Порівняння хмарних платформ для системи медіа-обробки

Характеристика	Google Cloud	AWS	Microsoft Azure
Об'єктне сховище	Cloud Storage	S3	Blob Storage
Брокер повідомлень	Pub/Sub	SNS + SQS	Service Bus
Push-доставка у контейнери	Нативна з Cloud Run	Потребує додаткової конфігурації	Потребує додаткової конфігурації
Контейнерний сервіс	Cloud Run	Fargate / App Runner	Container Apps
Масштабування від нуля	Так	Часткове (App Runner)	Так
Складність початкового розгортання	Низька	Висока	Середня
Безкоштовний кредит	$300 / 90 днів	Пробний період на 12 місяців з обмеженим використанням сервісів	$200 / 30 днів

На підставі проведеного аналізу, для розроблюваної системи обрано Google Cloud Platform. Це рішення обумовлене поєднанням кількох факторів: найнижчий поріг входження; великий обсяг безкоштовного кредиту; найменша

кількість конфігураційних кроків для запуску контейнеризованого сервісу з автоматичним масштабуванням від нуля; нативна інтеграція між Pub/Sub та Cloud Run, що усуває потребу в додатковому коді для зв'язування брокера повідомлень з обчислювальними сервісами; а також модель оплати за фактичне використання на рівні Cloud Run, що мінімізує витрати в періоди низького навантаження, характерні для етапу дослідження та початкового розгортання.

Окремо слід обґрунтувати вибір сервісу бази даних. Розроблювана система потребує реляційної бази даних PostgreSQL для зберігання облікових записів користувачів, метаданих проєктів та статусів обробки. Кожен із розглянутих хмарних провайдерів пропонує власний керований сервіс PostgreSQL: Google Cloud SQL, Amazon RDS та Azure Database for PostgreSQL. Проте мінімальна вартість цих сервісів є відносно високою навіть для найменших конфігурацій, оскільки вони передбачають постійне виділення обчислювальних ресурсів незалежно від фактичного навантаження. Зокрема, найдешеві конфігурації Google Cloud SQL або Amazon RDS для PostgreSQL коштують приблизно сім доларів на місяць лише за обчислювальні ресурси без урахування сховища та трафіку [41, 42].

Для розроблюваної системи обрано Neon — безсерверний PostgreSQL-сервіс, що реалізує модель оплати за фактичне використання обчислювальних ресурсів. Ключовою перевагою Neon є здатність автоматично масштабувати обчислювальні ресурси до нуля у періоди відсутності запитів, що означає повну відсутність витрат за період без навантаження [43]. Neon надає безкоштовний рівень використання, який включає достатній обсяг сховища та обчислювальних годин для етапу розробки і тестування. Це суттєво знижує фінансовий бар'єр для науково-дослідної роботи порівняно з традиційними керованими сервісами, де мінімальна щомісячна вартість є фіксованою незалежно від навантаження. Водночас Neon є повністю сумісним зі стандартним PostgreSQL, що забезпечує можливість легкої міграції на будь-який інший керований PostgreSQL-сервіс у разі масштабування системи в майбутньому. Таке рішення узгоджується із

загальним принципом, закладеним в архітектуру системи: оплата лише за фактично спожиті ресурси як на рівні обчислень, так і на рівні зберігання даних.

Окремим і надзвичайно важливим аспектом побудови хмарної інфраструктури є підхід IAC. Ручне створення та конфігурація хмарних ресурсів через веб-консоль провайдера є прийнятним лише на етапі прототипування, оскільки такий підхід не забезпечує відтворюваності, не підлягає версіонуванню і є схильним до людських помилок. Для розроблюваної системи обрано Terraform — інструмент IAC від компанії HashiCorp, який дозволяє описувати хмарну інфраструктуру у декларативних конфігураційних файлах мовою HashiCorp Configuration Language (HCL) [44].

Terraform вирішує кілька ключових задач у контексті даного дослідження. Замість ручного створення бакетів Cloud Storage, топіків та підписок Pub/Sub, сервісів Cloud Run та інших ресурсів через веб-інтерфейс, усі вони описуються у конфігураційних файлах, що зберігаються у репозиторії разом із кодом додатку. Це забезпечує версіонування інфраструктури на рівні з програмним кодом та повну відтворюваність середовищ розгортання. Розробник може розгорнути ідентичну інфраструктуру для середовищ розробки, тестування та виробництва за допомогою одного набору конфігураційних файлів із різними значеннями параметрів.

Таким чином, технологічний стек хмарної інфраструктури розроблюваної системи складається з Google Cloud Storage для зберігання медіафайлів, Cloud Pub/Sub для асинхронної комунікації, Cloud Run для розгортання контейнеризованих сервісів, Neon Serverless PostgreSQL для зберігання реляційних даних та Terraform для декларативного управління ресурсами. Обраний стек забезпечує мінімальні витрати на етапі дослідження, низький поріг входження, автоматичне масштабування та повну відтворюваність інфраструктури.

3 ПРОЄКТУВАННЯ СИСТЕМИ

На основі аналізу предметної області, проведеного у першому розділі, та дослідження технологій і архітектурних підходів, викладеного у другому розділі, у даному розділі представлено проєктні рішення щодо побудови системи автоматичного розпізнавання мовлення та генерації субтитрів для веб-платформ. Центральним результатом проєктування є високорівнева архітектура системи, що визначає склад компонентів, їх відповідальності та потоки даних між ними.

3.1 Загальна архітектура системи та взаємодія компонентів

Архітектура розроблюваної системи ґрунтується на патерні Web-Queue-Worker, обґрунтування вибору якого було надано у підрозділі 2.3. Відповідно до цього патерну, система декомпозується на три окремі застосунки: веб-застосунок (Web App), серверний API-сервіс (Web Service) та сервіс-воркер (Worker Service). Кожен із цих застосунків виконує чітко визначену роль у загальному конвеєрі обробки медіаконтенту, і саме такий поділ дозволяє масштабувати кожен компонент незалежно від інших.

Веб-застосунок є клієнтською частиною системи та єдиною точкою взаємодії користувача з платформою. Він реалізований на базі фреймворку Next.js і виконує кілька ключових функцій. По-перше, він забезпечує автентифікацію та авторизацію користувача через механізм електронної пошти та пароля. По-друге, після входу в систему користувач потрапляє на інформаційну панель, де відображається перелік усіх раніше створених проєктів із їх поточними статусами обробки. По-третє, веб-застосунок реалізує сценарій створення нового проєкту: користувач обирає відеофайл на своєму пристрої, після чого браузер отримує від серверного API тимчасове підписане посилання (Signed URL) і завантажує відео безпосередньо у хмарне об'єктне сховище Google Cloud Storage, минаючи при цьому сам серверний сервіс. Такий підхід розвантажує серверну частину від передачі великих бінарних файлів і дозволяє

використовувати пропускну здатність хмарного сховища замість серверних ресурсів API. Нарешті, веб-застосунок надає інтерфейс перегляду результатів: вбудований відеоплеєр динамічно накладає згенерований файл субтитрів у форматі VTT поверх відеопотоку, а також дозволяє ініціювати запит на генерацію фінального відео зі вбудованними субтитрами для подальшого завантаження та публікації у соціальних мережах.

Серверний API-сервіс побудований на базі фреймворку Fastify та виконує роль центрального координатора системи. Він приймає HTTP-запити від веб-застосунку, здійснює валідацію вхідних даних, взаємодіє з базою даних для збереження та читання метаданих проєктів і користувачів, а також генерує підписані посилання для прямого завантаження файлів у хмарне сховище. Ключовою функцією API-сервісу є ініціювання асинхронної обробки: після того як відео успішно завантажено у сховище і метадані збережено в базі, сервіс публікує повідомлення у чергу повідомлень Google Cloud Pub/Sub і миттєво повертає клієнту відповідь, яка підтверджує прийняття завдання в обробку. Таким чином, API-сервіс не виконує жодних обчислювально вимогливих операцій, що дозволяє йому обслуговувати велику кількість одночасних клієнтських запитів із мінімальними затримками.

Сервіс-воркер є обчислювальним ядром системи, відповідальним за всі ресурсомісткі операції з медіафайлами. Він отримує повідомлення з черги Cloud Pub/Sub, і кожне таке повідомлення містить не сам медіафайл, а лише посилання на нього у хмарному сховищі — відповідно до патерну Claim-Check, описаного у підрозділі 2.3. Отримавши завдання, воркер завантажує відеофайл зі сховища, вилучає з нього аудіодоріжку за допомогою FFmpeg, надсилає отриманий аудіофайл до OpenAI Whisper API для розпізнавання мовлення, формує файл субтитрів у форматі VTT на основі отриманої транскрипції з часовими мітками, зберігає його у сховище та оновлює статус відповідного проєкту в базі даних. Окрім транскрибування, воркер також обробляє запити на генерацію відео зі вбудованими субтитрами, використовуючи граф фільтрів FFmpeg для накладання субтитрів безпосередньо на відеокадри. Результат рендерингу так

само зберігається у хмарному сховищі, а статус завантаження проєкту оновлюється в базі даних.

Взаємодія між зазначеними компонентами відбувається через кілька каналів передачі даних, кожен з яких оптимізований для свого типу навантаження. Синхронна комунікація між веб-застосунком та API-сервісом здійснюється через HTTP-запити, що є ефективним механізмом для операцій із низькою затримкою, таких як автентифікація, отримання списку проєктів або генерація підписаних посилань. Асинхронна комунікація між API-сервісом та сервісом-воркером реалізована через чергу повідомлень Cloud Pub/Sub, що повністю декомпозує ці компоненти та дозволяє їм працювати незалежно один від одного. Передача медіафайлів між компонентами відбувається не безпосередньо, а через хмарне об'єктне сховище Google Cloud Storage, яке виступає спільною площиною даних. Браузер завантажує оригінальне відео у сховище через підписане посилання, воркер читає його звідти для обробки та записує результати назад. Нарешті, як API-сервіс, так і сервіс-воркер взаємодіють з єдиною базою даних PostgreSQL для читання та запису метаданих та статусів проєктів. Високорівнева архітектурна діаграма системи, що ілюструє описані компоненти та потоки даних між ними, представлена на рисунку 3.1.

Важливим організаційним рішенням є розміщення вихідного коду всіх трьох застосунків у єдиному монорепозиторії, керованому інструментом Turborepo [45]. Монорепозиторій об'єднує Web App, Web Service та Worker Service як окремі пакети в межах одного сховища коду разом зі спільними пакетами, що містять типи даних, утиліти та конфігурації, використовувані кількома застосунками одночасно. Такий підхід забезпечує атомарність змін: модифікація спільного контракту між API-сервісом та воркером, наприклад, структури повідомлення у черзі, відображається в обох застосунках в межах одного коміту, що унеможливлює неузгодженість інтерфейсів між компонентами. Turborepo при цьому забезпечує оптимізацію процесу збірки за рахунок кешування та паралельного виконання задач, що зменшує час розробки та безперервної інтеграції.

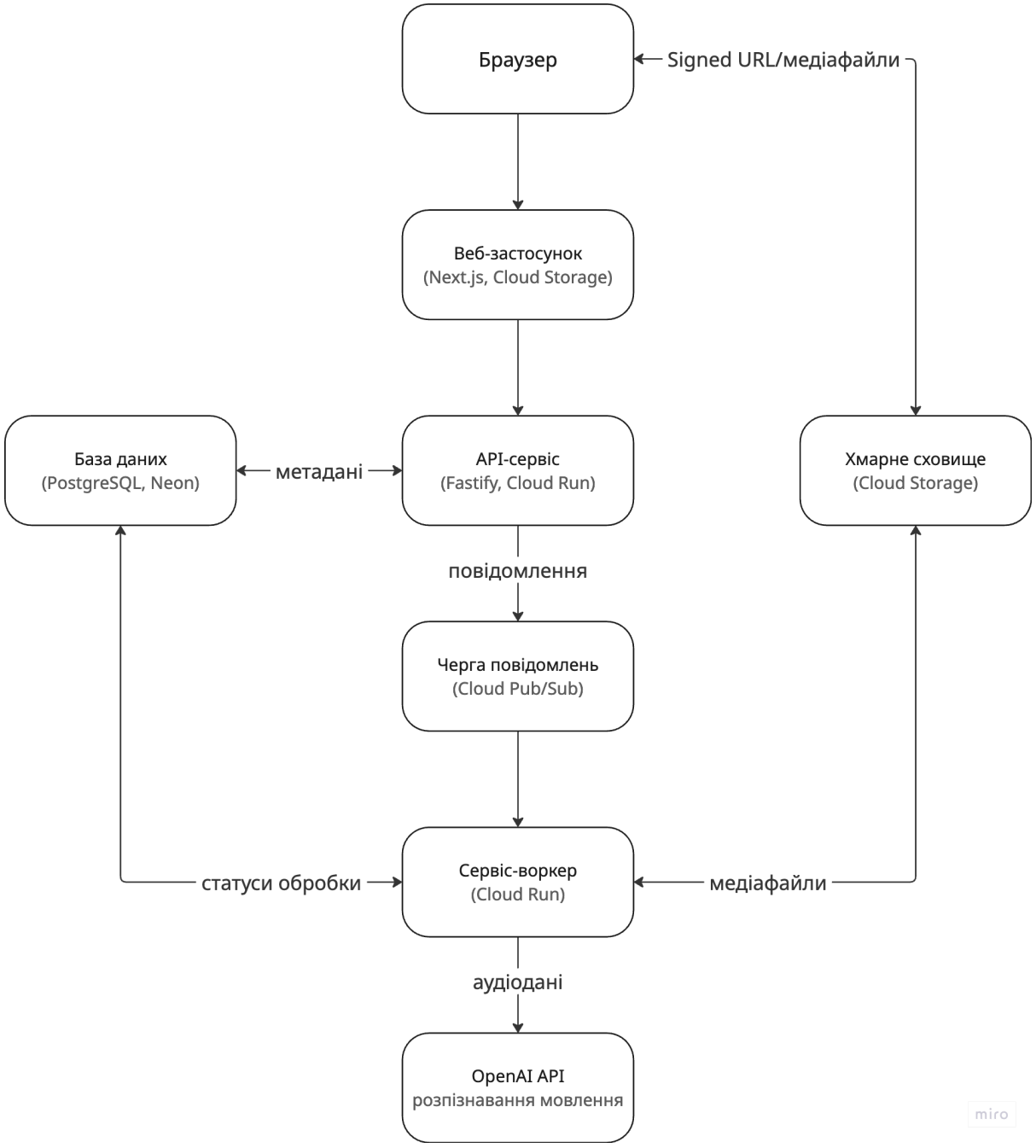


Рисунок 3.1 — Архітектурна діаграма системи автоматичного розпізнавання мовлення та генерації субтитрів

Обґрунтування вибору загального технологічного стеку системи ґрунтується на кількох принципах. Мовою програмування для всіх трьох застосунків обрано TypeScript, що забезпечує єдину мовну екосистему та статичну типізацію на всіх рівнях системи — від клієнтського інтерфейсу до серверної логіки та обробки медіа [46]. Використання однієї мови для всього

стеку суттєво спрощує розробку, оскільки типи даних, валідаційні схеми та утиліти можуть бути винесені у спільні пакети монорепозиторію і перевикористані без дублювання. Next.js обрано як фреймворк для веб-застосунку завдяки його вбудованій підтримці серверного рендерингу, маршрутизації на основі файлової системи та оптимізації продуктивності [47]. Fastify обрано для побудови API-сервісу як високопродуктивний серверний фреймворк для Node.js, що забезпечує низьку латентність обробки HTTP-запитів та вбудовану підтримку валідації через JSON Schema [48]. PostgreSQL обрано як реляційну базу даних завдяки зрілості платформи, надійності та цілісності даних [49], а також наявності клієнтських бібліотек для середовища Node.js, зокрема TypeORM [50], що використовується у даній системі. Розгортання на платформі Neon Serverless забезпечує автоматичне масштабування та оплату лише за фактично спожиті ресурси, що узгоджується із загальною безсерверною філософією хмарної інфраструктури системи.

Таким чином, спроєктована архітектура забезпечує чіткий поділ відповідальностей між компонентами, асинхронну комунікацію для ресурсомістких операцій, незалежне масштабування обчислювальних вузлів та ефективне використання хмарних ресурсів, що в сукупності створює основу для подальшого детального проєктування бази даних, конвеєра обробки медіа та інфраструктури розгортання, які розглядаються у наступних підрозділах.

3.2 Проєктування моделі даних

Проєктування моделі даних є невід'ємною складовою побудови будь-якої веб-системи, оскільки саме структура бази даних визначає, яким чином зберігається, оновлюється та зчитується інформація про стан системи та її об'єкти. У контексті розроблюваної системи автоматичного розпізнавання мовлення та генерації субтитрів база даних відіграє роль єдиного джерела істини щодо стану кожного проєкту обробки відео: від моменту завантаження оригінального файлу до отримання готового результату із вбудованими

субтитрами. Як було зазначено у підрозділі 3.1, базу даних PostgreSQL розгорнуто на платформі Neon Serverless, а взаємодія з нею відбувається через бібліотеку TypeORM як з боку API-сервісу, так і з боку сервісу-воркера.

Модель даних системи побудована за принципом мінімальної достатності: вона містить лише ті сутності та атрибути, які безпосередньо необхідні для функціонування конвеєра обробки медіаконтенту. На поточному етапі проєктування модель складається з двох основних таблиць — users та projects — пов'язаних відношенням «один до багатьох», а також двох типів-перелічень, що формалізують життєвий цикл обробки проєкту. Схема бази даних (діаграма «сутність-зв’язок»), що ілюструє структуру таблиць, їх атрибути, типи даних та зв'язки, представлена на рисунку 3.2.

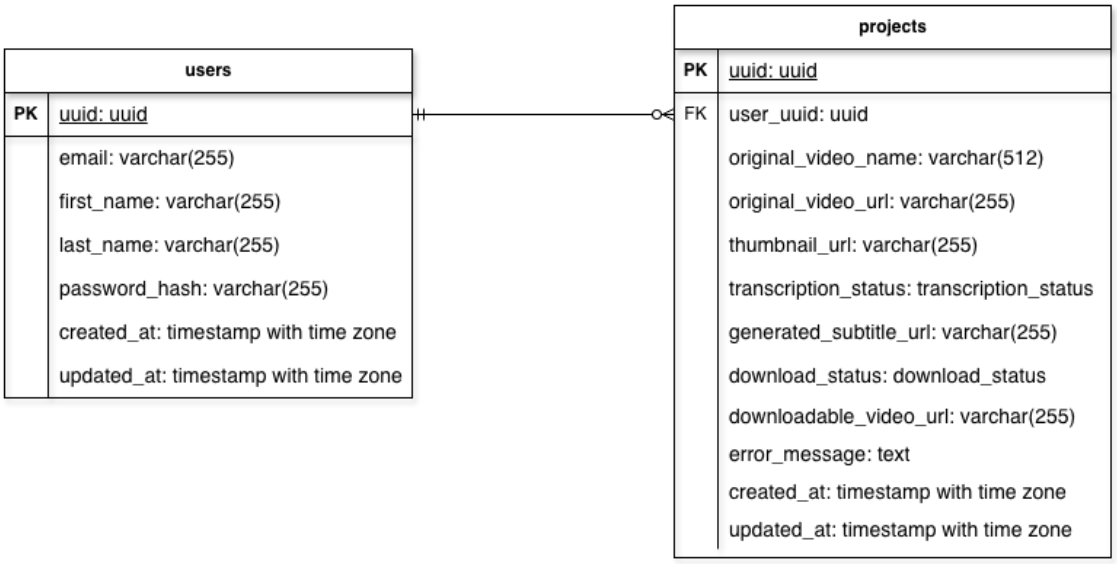


Рисунок 3.2 — Схема бази даних системи автоматичного розпізнавання мовлення та генерації субтитрів

Таблиця users зберігає інформацію про зареєстрованих користувачів системи. Первинним ключем таблиці є поле uuid, яке автоматично генерується за допомогою функції uuid_generate_v4() і є універсально унікальним ідентифікатором четвертої версії. Вибір Universally Unique Identifier (UUID) замість послідовного цілочислового ідентифікатора зумовлений кількома міркуваннями. По-перше, UUID не розкриває інформацію про кількість записів

у таблиці, що є важливим з точки зору безпеки, оскільки послідовні ідентифікатори дозволяють зловмиснику здійснити перебір сусідніх значень. По-друге, UUID може бути згенерований на будь-якому рівні системи — як на сервері, так і на клієнті — без необхідності звернення до бази даних для отримання наступного значення послідовності. Таблиця містить поля для зберігання електронної адреси користувача, імені, прізвища та хешу пароля. Окрім основних полів, таблиця містить часові мітки created_at та updated_at, що фіксують момент створення та останнього оновлення запису відповідно, із використанням типу TIMESTAMPTZ для коректного зберігання часу з урахуванням часового поясу.

Таблиця projects є центральною таблицею системи, оскільки саме вона відображає стан кожного проєкту обробки відео протягом усього його життєвого циклу. Кожен запис у таблиці відповідає одному завантаженому відеофайлу та всім пов'язаним із ним результатам обробки. Зв'язок із таблицею users реалізовано через зовнішній ключ user_uuid, який посилається на первинний ключ таблиці users із каскадним видаленням — при видаленні облікового запису користувача автоматично видаляються всі його проєкти.

Таблиця projects містить кілька груп атрибутів, кожна з яких відповідає певному аспекту обробки. Перша група стосується оригінального відео: поле original_video_name зберігає оригінальну назву файлу, яку надав користувач при завантаженні, а поле original_video_url — ключ об'єкта у хмарному сховищі Google Cloud Storage, за яким файл може бути завантажений для обробки. Поле thumbnail_url зберігає посилання на зображення-прев’ю відео, яке використовується для відображення на інформаційній панелі. Друга група атрибутів стосується процесу транскрибування: поле transcription_status відображає поточний стан розпізнавання мовлення, а поле generated_subtitle_url — ключ об'єкта згенерованого файлу субтитрів у форматі VTT у хмарному сховищі. Третя група стосується процесу генерації відео із вбудованими субтитрами: поле download_status відображає стан цього процесу, а поле downloadable_video_url — ключ об'єкта готового відеофайлу. Нарешті, поле

error_message призначене для збереження детальної інформації про причину помилки у випадку, коли будь-який із процесів завершився зі статусом невдачі, що є важливим для діагностики та подальшого аналізу проблем.

Особливу роль у моделі даних відіграють два типи-перелічення, які формалізують скінченні автомати станів для двох паралельних процесів обробки в межах одного проєкту. Тип transcription_status визначає чотири можливі стани процесу розпізнавання мовлення:

– стан PENDING є початковим і встановлюється при створенні проєкту, коли завантаження відеофайлу ще триває або щойно завершилось;

– стан PROCESSING означає, що завдання на транскрибування було відправлено у чергу повідомлень і знаходиться в обробці сервісом-воркером

– стан COMPLETED сигналізує про успішне завершення розпізнавання мовлення та генерації файлу субтитрів;

– стан FAILED фіксує невдачу процесу транскрибування, при цьому причина помилки записується у поле error_message.

Тип download_status визначає п'ять можливих станів процесу генерації відео із вбудованими субтитрами:

– стан IDLE є початковим і означає, що користувач ще не ініціював запит на створення такого відео — цей стан принципово відрізняється від PENDING у transcription_status, оскільки процес впаювання субтитрів не починається автоматично після завантаження, а потребує явної дії користувача;

– стан PENDING фіксує момент, коли запит на впаювання субтитрів прийнято і відповідне повідомлення опубліковано у чергу;

– стан PROCESSING означає, що сервіс-воркер виконує рендеринг відео з накладеними субтитрами за допомогою FFmpeg;

– стан COMPLETED сигналізує про доступність готового файлу для завантаження;

– стан FAILED фіксує помилку при процесі рендерингу.

Використання типів-перелічень замість довільних рядкових значень для зберігання статусів є свідомим архітектурним рішенням, яке забезпечує

цілісність даних на рівні бази: PostgreSQL не дозволить записати у відповідне поле значення, яке не входить до визначеного переліку, що унеможливлює появу некоректних або неочікуваних станів внаслідок помилки у програмному коді. Крім того, перелічувані типи дозволяють явно документувати всі допустимі стани системи безпосередньо на рівні схеми бази даних, що спрощує розуміння бізнес-логіки як для розробників, так і для рецензентів.

Важливо зазначити, що самі медіафайли — оригінальне відео, файл субтитрів та відео із вбудованими субтитрами — не зберігаються у базі даних. У таблиці projects зберігаються лише посилання на відповідні об'єкти у хмарному сховищі Google Cloud Storage. Такий підхід повністю узгоджується з патерном Claim-Check, описаним у підрозділі 2.3, і забезпечує розділення площини метаданих та площини медіаданих. База даних залишається швидким сховищем, оскільки вона оперує лише текстовими метаданими та статусами, тоді як великі бінарні файли зберігаються у спеціалізованому об'єктному сховищі, оптимізованому для такого типу навантаження.

Стратегія індексування у спроєктованій базі даних є мінімальною. Окрім первинних ключів, які автоматично індексуються PostgreSQL, створено два додаткові індекси. Унікальний індекс idx_users_email на полі email у таблиці users забезпечує швидкий пошук під час автентифікації — операції, що виконується при кожному вході користувача у систему. Індекс idx_projects_user_uuid на полі user_uuid у таблиці projects оптимізує вибірку всіх проєктів конкретного користувача, яка є основною операцією читання при завантаженні інформаційної панелі. Додаткових індексів на полях статусів наразі не передбачено, оскільки фільтрація проєктів за статусом відбувається в межах вже відфільтрованої за user_uuid вибірки, обсяг якої для типового користувача є достатньо малим для послідовного сканування.

3.3 Проєктування конвеєра асинхронної обробки медіа

У попередніх підрозділах було визначено загальну архітектуру системи та модель даних. Даний підрозділ описує наскрізний сценарій обробки медіаконтенту — послідовність кроків від моменту, коли користувач обирає відеофайл на своєму пристрої, до моменту, коли він отримує готовий результат із субтитрами.

Конвеєр обробки складається з двох незалежних процесів: процесу генерації субтитрів, який запускається автоматично після завантаження відео, та процесу завантаження відео із вбудованими субтитрами, який ініціюється явною дією користувача. Обидва процеси побудовані за однаковим принципом асинхронної комунікації через чергу повідомлень, проте виконують різні операції з медіафайлами. Діаграми послідовностей, що ілюструють ці процеси надані на рисунках А.1 та А.2.

3.3.1 Процес генерації субтитрів

Процес генерації субтитрів розпочинається з дії користувача у веб-застосунку. Коли користувач обирає відеофайл для завантаження, веб-застосунок надсилає HTTP-запит до API-сервісу із запитом на створення нового проєкту. API-сервіс виконує кілька операцій у відповідь на цей запит. Спочатку він створює новий запис у таблиці projects бази даних із початковим статусом транскрибування PENDING. Далі API-сервіс генерує тимчасове підписане посилання для завантаження файлу безпосередньо у хмарне сховище Google Cloud Storage. Це посилання є обмеженим у часі та дозволяє виконати лише одну операцію запису за конкретним ключем об'єкта, що забезпечує безпеку без необхідності надавати клієнту повні облікові дані доступу до сховища. API-сервіс повертає веб-застосунку ідентифікатор створеного проєкту та підписане посилання, після чого веб-застосунок відповідає на цей запит миттєво — без очікування завершення завантаження файлу.

Механізм прямого завантаження через підписане посилання є принциповим архітектурним рішенням, яке має суттєві переваги порівняно з

альтернативним підходом, при якому відеофайл завантажується спочатку на API-сервіс, а потім передається у сховище. При прямому завантаженні бінарні дані відеофайлу передаються безпосередньо від браузера користувача до Google Cloud Storage, повністю минаючи API-сервіс. Це означає, що API-сервіс не витрачає оперативну пам'ять на буферизацію великих файлів, не витрачає мережеву пропускну здатність на їх ретрансляцію і не блокує з'єднання на тривалий час завантаження. У результаті API-сервіс може обслуговувати значно більшу кількість одночасних запитів, оскільки кожен запит на створення проєкту є легковагим і завершується за мілісекунди.

Після успішного завантаження відеофайлу у хмарне сховище веб-застосунок повідомляє API-сервіс про завершення завантаження. API-сервіс у відповідь виконує дві ключові дії: оновлює статус транскрибування проєкту з PENDING на PROCESSING та публікує повідомлення у топік Cloud Pub/Sub. Повідомлення містить не сам відеофайл, а лише ідентифікатор проєкту та ключ об'єкта у хмарному сховищі, відповідно до патерну Claim-Check. Після публікації повідомлення API-сервіс миттєво повертає клієнту підтвердження прийняття завдання в обробку. З цього моменту клієнтський веб-застосунок починає періодично опитувати API-сервіс щодо поточного статусу проєкту, відображаючи користувачу індикатор обробки.

Сервіс-воркер, який є підписником відповідного топіку Cloud Pub/Sub, отримує повідомлення та розпочинає послідовність обчислювально вимогливих операцій. Першим кроком воркер завантажує оригінальний відеофайл із хмарного сховища у локальну тимчасову файлову систему контейнера. Далі воркер вилучає аудіодоріжку з відеофайлу за допомогою FFmpeg. Ця операція конвертує аудіо у формат, оптимальний для розпізнавання мовлення, що відповідає вхідним вимогам моделі Whisper та мінімізує обсяг даних, які передаються до зовнішнього API.

Наступним кроком воркер надсилає вилучений аудіофайл до OpenAI API для розпізнавання мовлення. API повертає результат транскрипції у вигляді структурованих даних із часовими мітками для кожного сегмента тексту. На

основі цих даних воркер формує файл субтитрів у форматі WebVTT. Кожен запис у файлі VTT містить часову мітку початку та кінця відображення, а також відповідний текстовий фрагмент транскрипції. Згенерований файл субтитрів завантажується у хмарне сховище, а його ключ записується у поле generated_subtitle_url відповідного запису в таблиці projects. Статус транскрибування оновлюється на COMPLETED, після чого наступний запит від клієнтського веб-застосунку виявить зміну статусу і відобразить користувачу результат — відеоплеєр із динамічно накладеними субтитрами.

У випадку виникнення помилки на будь-якому етапі обробки — чи то через пошкодження відеофайлу, збій FFmpeg, помилку зовнішнього API або іншу причину — воркер фіксує детальну інформацію про помилку у полі error_message відповідного запису проєкту та встановлює статус транскрибування у FAILED. Це дозволяє як користувачу побачити, що обробка не вдалася, так і адміністраторам системи проаналізувати причину збою.

3.3.2 Процес завантаження відео із вбудованими субтитрами

Процес завантаження відео із вбудованими субтитрами є другим незалежним процесом конвеєра і запускається лише за явним запитом користувача. Після перегляду згенерованих субтитрів користувач може ініціювати створення фінального відеофайлу. Такий файл необхідний для публікації у соціальних мережах, таких як TikTok, Instagram Reels та YouTube Shorts, де платформи не підтримують зовнішні файли субтитрів і єдиним способом відображення тексту є його вбудовування безпосередньо у відеопотік.

Процес завантаження за структурою аналогічний процесу генерації субтитрів. Веб-застосунок надсилає запит до API-сервісу, який оновлює download_status проєкту з IDLE на PENDING, публікує відповідне повідомлення у чергу Cloud Pub/Sub і миттєво повертає відповідь клієнту. Сервіс-воркер отримує повідомлення, завантажує оригінальний відеофайл та файл субтитрів із хмарного сховища, після чого виконує рендеринг відео з накладеними субтитрами за допомогою графа відеофільтрів FFmpeg. Ця операція є найбільш обчислювально вимогливою в усьому конвеєрі, оскільки вона передбачає повне

декодування кожного кадру відео, растеризацію тексту субтитрів поверх відеокадру та повторне кодування результату. Готовий відеофайл завантажується у хмарне сховище, його ключ записується у поле downloadable_video_url, а download_status оновлюється на COMPLETED.

Важливою характеристикою спроєктованого конвеєра є повне розділення синхронної та асинхронної площин обробки. Усі взаємодії між браузером та API-сервісом є синхронними, короткотривалими HTTP-запитами, що завершуються за мілісекунди. Усі обчислювально вимогливі операції — вилучення аудіо, виклик зовнішнього API розпізнавання мовлення, генерація субтитрів та рендеринг відео — виконуються асинхронно сервісом-воркером, повністю декомпозованим від HTTP-запитів користувача. Черга повідомлень Cloud Pub/Sub виступає буфером між цими двома площинами: вона приймає повідомлення від API-сервісу незалежно від того, чи є вільні воркери для їх негайної обробки, і доставляє їх воркерам у міру доступності обчислювальних ресурсів. Це забезпечує згладжування пікових навантажень — якщо кілька користувачів одночасно завантажать відео, завдання поступово оброблятимуться без перевантаження системи, замість того щоб всі одночасно конкурували за обмежені ресурси.

Відстеження стану обробки на клієнтському боці реалізовано через механізм періодичного опитування (англ. polling). Після ініціювання асинхронної обробки веб-застосунок з фіксованим інтервалом надсилає HTTP-запити до API-сервісу для перевірки поточного значення полів transcription_status або download_status відповідного проєкту. Коли статус змінюється на COMPLETED або FAILED, веб-застосунок припиняє опитування та оновлює інтерфейс відповідно: або відображає результат із субтитрами, або повідомляє користувача про помилку. Такий підхід є простішим у реалізації порівняно з альтернативами на кшталт WebSocket або Server-Sent Events і є достатнім для даного сценарію використання, де час обробки становить від десятків секунд до кількох хвилин і користувач очікує результат не в режимі реального часу.

Таким чином, спроєктований конвеєр асинхронної обробки медіа забезпечує наскрізний цикл від завантаження відео до отримання готового результату, реалізуючи при цьому ключові архітектурні принципи: розділення синхронної та асинхронної площин обробки, пряме завантаження медіафайлів через підписані посилання, передачу лише посилань на файли через чергу повідомлень та поетапне оновлення статусів у базі даних для забезпечення прозорості процесу обробки для кінцевого користувача.

3.4 Проектування механізмів забезпечення надійності системи

Асинхронний конвеєр обробки медіаконтенту, описаний у попередньому підрозділі, за своєю природою є розподіленою системою, в якій компоненти взаємодіють через мережу та хмарну інфраструктуру. У таких системах збої є не винятковою ситуацією, а очікуваною частиною нормального функціонування: мережеві з'єднання можуть переривatися, контейнери можуть перезапускатися, зовнішні API можуть тимчасово бути недоступними, а вхідні дані від користувачів можуть виявитися пошкодженими або невідповідними очікуваному формату. Тому проєктування механізмів надійності є невід'ємною частиною архітектурного рішення, а не додатковою оптимізацією. У підрозділі 2.3 було теоретично обґрунтовано доцільність застосування трьох інженерних патернів надійності — ідемпотентності обробки, Dead Letter Queue та Claim-Check. У даному підрозділі описано, яким чином ці патерни конкретно реалізуються та взаємодіють між собою у спроєктованому конвеєрі.

Першим і, мабуть, найважливішим механізмом надійності є забезпечення ідемпотентності обробки повідомлень сервісом-воркером. Необхідність цього механізму зумовлена гарантією доставки, яку надає брокер повідомлень Cloud Pub/Sub. Pub/Sub працює за принципом «принаймні одна доставка», що означає: в нормальних умовах кожне повідомлення доставляється рівно один раз, але у випадку мережевих затримок, перезапуску контейнера воркера або тайм-ауту підтвердження обробки одне й те саме повідомлення може бути доставлене

повторно [30]. Без механізму ідемпотентності таке повторне доставлення призвело б до критичних наслідків: подвійного виклику OpenAI API з відповідним подвійним нарахуванням вартості, перезапису вже готового файлу субтитрів проміжним результатом, або некоректного оновлення статусу проєкту в базі даних.

У спроєктованій системі ідемпотентність досягається через перевірку поточного стану проєкту перед початком будь-якої обробки. Коли сервіс-воркер отримує повідомлення з черги, першою його дією є зчитування запису відповідного проєкту з бази даних та перевірка значення поля transcription_status або download_status, залежно від типу завдання. Якщо статус вже має значення COMPLETED або PROCESSING, воркер негайно підтверджує отримання повідомлення брокеру без виконання будь-яких операцій з медіафайлами. Повторна обробка виконується лише у тому випадку, коли статус відповідає початковому стану, що свідчить про те, що попередня спроба або не розпочиналася, або завершилася невдачею. Такий підхід гарантує, що незалежно від кількості доставок одного повідомлення, результат обробки буде застосований лише один раз, виклик зовнішнього API буде здійснений лише один раз, і стан у базі даних залишатиметься консистентним.

Другим механізмом надійності є Dead Letter Queue, що забезпечує ізоляцію так званих «отруєних повідомлень» — завдань, які не можуть бути успішно оброблені навіть після кількох спроб. У контексті розроблюваної системи такі ситуації виникають, коли користувач завантажує пошкоджений відеофайл, з якого FFmpeg не може вилучити аудіодоріжку, або коли файл має непідтримуваний кодек чи формат контейнера. Без механізму Dead Letter Queue таке повідомлення поверталось би у основну чергу після кожної невдалої спроби обробки, створюючи нескінченний цикл: воркер отримує повідомлення, намагається обробити, зазнає невдачі, не підтверджує обробку, Pub/Sub повторно доставляє повідомлення, і цикл повторюється. Такий цикл не лише марнує обчислювальні ресурси, але й може заблокувати обробку інших, цілком коректних завдань, якщо кількість одночасних екземплярів воркера обмежена.

Механізм Dead Letter Queue у спроєктованій системі реалізується засобами Cloud Pub/Sub. Для основної підписки на топік повідомлень налаштовується максимальна кількість спроб доставки та окремий топік для «мертвих повідомлень». Коли воркер не може обробити повідомлення, він не підтверджує його отримання, і Pub/Sub виконує повторну доставку з експоненційною затримкою між спробами. Якщо після визначеної кількості спроб обробка так і не вдалася, Pub/Sub автоматично перенаправляє повідомлення у топік Dead Letter Queue замість повернення у основну чергу. Паралельно воркер при кожній невдалій спробі записує детальну інформацію про помилку у поле error_message відповідного проєкту в базі даних та оновлює статус на FAILED. У результаті користувач бачить повідомлення про помилку в інтерфейсі, а адміністратор системи може проаналізувати повідомлення у Dead Letter Queue для виявлення системних проблем або повторних закономірностей у типах помилок.

Третім механізмом, який забезпечує ефективність та надійність передачі даних між компонентами, є патерн Claim-Check. Як було зазначено у підрозділах 3.1 та 3.3, медіафайли не передаються через чергу повідомлень напряму — замість цього через Pub/Sub передається лише ідентифікатор проєкту та ключ об'єкта у хмарному сховищі. Це рішення зумовлене не лише технічними обмеженнями брокера повідомлень на розмір повідомлення, який у Cloud Pub/Sub складає десять мегабайт, тоді як відеофайли можуть сягати сотень мегабайт, але й архітектурними міркуваннями. Розділення площини передачі метаданих та площини передачі медіаданих дозволяє кожному каналу працювати в оптимальному режимі: черга повідомлень обробляє компактні текстові повідомлення з низькою латентністю, а хмарне об'єктне сховище забезпечує високопропускну передачу великих бінарних файлів. Крім того, при такому підході повторна доставка повідомлення через Pub/Sub не спричиняє повторної передачі самого відеофайлу, що суттєво зменшує навантаження на мережу та сховище у сценаріях відновлення після збоїв.

Окрім трьох основних патернів, у спроєктованій системі передбачено додаткові механізми, що підвищують загальну надійність. Каскадне видалення

записів проєктів при видаленні облікового запису користувача, реалізоване через зовнішній ключ з обмеженням ON DELETE CASCADE у моделі даних, запобігає появі «осиротілих» записів у базі даних. Використання типів-перелічень для полів статусів, описане у підрозділі 3.2, унеможливлює перехід проєкту у невизначений стан через програмну помилку. Механізм підписаних посилань із обмеженим терміном дії для завантаження та вивантаження файлів забезпечує безпеку доступу до хмарного сховища без необхідності зберігання чи передачі повних облікових даних до клієнтського браузера.

Таким чином, спроєктовані механізми забезпечення надійності у сукупності гарантують, що конвеєр асинхронної обробки медіаконтенту здатний коректно функціонувати в умовах, типових для хмарних розподілених систем: при повторних доставках повідомлень, тимчасових збоях зовнішніх сервісів, пошкоджених вхідних даних та непередбачуваних перезапусках контейнерів. Ідемпотентність обробки, Dead Letter Queue та Claim-Check доповнюють один одного і разом із базовими механізмами цілісності даних на рівні моделі бази даних формують цілісну стратегію надійності, яка охоплює всі етапи конвеєра — від прийняття завдання в обробку до збереження кінцевого результату.

3.5 Проектування CI/CD конвеєра

Попередні підрозділи описали архітектуру системи, модель даних, конвеєр обробки та механізми надійності. Проте навіть бездоганно спроєктована система не має практичної цінності без надійного та відтворюваного процесу її розгортання у хмарному середовищі. Даний підрозділ описує проєктування конвеєра безперервної інтеграції та безперервного розгортання (CI/CD) на базі GitHub Actions, а також декларативне управління хмарною інфраструктурою за допомогою Terraform.

Конвеєр CI/CD спроєктований як єдиний робочий процес GitHub Actions [51], який автоматично запускається при кожному об'єднанні змін у головну гілку репозиторію. Оскільки система побудована як монорепозиторій із трьома

застосунками, ключовою архітектурною проблемою є уникнення надлишкових збірок: зміна одного рядка у коді веб-застосунку не повинна спричиняти повторну збірку та розгортання API-сервісу і сервісу-воркера. Для вирішення цієї проблеми першим етапом конвеєра є аналіз змін за допомогою утиліти turbo-ignore, яка є частиною екосистеми Turborepo. Ця утиліта порівнює поточний коміт із попереднім і визначає, які саме застосунки та їх залежності були змінені [52]. На основі цього аналізу конвеєр приймає рішення, які з наступних етапів необхідно виконати, а які можна пропустити.

Процес збірки та розгортання відрізняється для веб-застосунку та серверних сервісів, оскільки вони мають принципово різну природу. Веб-застосунок, побудований на Next.js, експортується як набір статичних файлів (HTML, CSS, JavaScript), які не потребують серверного середовища виконання. Конвеєр збірки веб-застосунку виконує перевірку якості коду за допомогою лінтера та форматувальника, після чого генерує статичний експорт і завантажує його у бакет Google Cloud Storage, який налаштований для обслуговування веб-контенту [53]. Такий підхід забезпечує мінімальну вартість хостингу та максимальну швидкість завантаження для кінцевих користувачів, оскільки статичні файли подаються безпосередньо з хмарного сховища без проміжного серверу додатків.

API-сервіс та сервіс-воркер, на відміну від веб-застосунку, є серверними додатками, що потребують середовища виконання Node.js. Для їх розгортання використовується контейнеризація: конвеєр збирає Docker-образ кожного сервісу, присвоює йому тег на основі хешу коміту, що забезпечує унікальну та однозначну ідентифікацію кожної версії, і завантажує образ у Google Artifact Registry — керований реєстр контейнерних образів [54]. Використання хешу коміту як тегу образу гарантує повну відстежуваність: для будь-якого розгорнутого сервісу можна точно визначити, з якого стану вихідного коду він був зібраний.

Між етапами збірки контейнерних образів та розгортання інфраструктури виконується окремий крок — застосування міграцій бази даних. Міграції

описують зміни у схемі бази даних та виконуються за допомогою TypeORM. Цей крок розташований після збірки, але перед розгортанням нових версій сервісів, що гарантує: на момент, коли новий код починає обробляти запити, структура бази даних вже відповідає його очікуванням. Такий порядок виконання є стандартною практикою для систем із реляційними базами даних і запобігає ситуаціям, коли новий код намагається звернутися до ще не створених таблиць або полів.

Фінальним етапом конвеєра CI/CD є застосування Terraform-конфігурації, яке об'єднує результати всіх попередніх етапів. Terraform отримує теги щойно зібраних Docker-образів від етапу збірки та використовує їх для оновлення конфігурації сервісів Cloud Run. Якщо для певного сервісу нових змін не було і, відповідно, новий образ не був зібраний, Terraform автоматично зчитує тег поточного розгорнутого образу через Google Cloud SDK і використовує його, залишаючи сервіс без змін. Таким чином, команда terraform apply виконується при кожному запуску конвеєра, але фактичні зміни в інфраструктурі відбуваються лише тоді, коли стан, описаний у конфігурації, відрізняється від поточного стану розгорнутих ресурсів. Це є проявом декларативного підходу Terraform: розробник описує бажаний стан інфраструктури, а Terraform самостійно обчислює, які дії необхідно виконати для досягнення цього стану.

Конфігурація хмарної інфраструктури у Terraform організована за модульним принципом. Для сервісів Cloud Run створено повторно використовуваний модуль, який інкапсулює всі необхідні ресурси для розгортання одного контейнеризованого сервісу: сам сервіс Cloud Run із визначеними лімітами обчислювальних ресурсів, сервісний обліковий запис із мінімально необхідними правами доступу, налаштування автоматичного масштабування та політики доступу. Цей модуль приймає параметри — назву сервісу, тег Docker-образу, обсяг процесора та оперативної пам'яті, змінні середовища та інші налаштування — і використовується двічі: один раз для API-сервісу і один раз для сервісу-воркера, з різними значеннями параметрів. Такий підхід усуває дублювання конфігурації та гарантує, що обидва сервіси

розгортаються за єдиним шаблоном, відрізняючись лише у параметрах, специфічних для їхніх ролей.

Окрім модулів Cloud Run, Terraform-конфігурація описує всі інші хмарні ресурси системи: бакети Cloud Storage для зберігання статичних файлів веб-застосунку та медіафайлів користувачів із відповідними політиками доступу, топіки та підписки Cloud Pub/Sub із налаштуванням Dead Letter Queue, секрети у Google Secret Manager для зберігання конфіденційних параметрів (ключів API, рядків підключення до бази даних) та правила доступу між сервісами. Усі ці ресурси описані у конфігураційних файлах, що зберігаються у репозиторії разом із кодом застосунків, що забезпечує версіонування інфраструктури на рівні з програмним кодом.

Стратегія розгортання на Google Cloud Run є безпосереднім продовженням архітектурних рішень, описаних у підрозділі 3.1. API-сервіс розгортається як публічно доступний сервіс, що приймає HTTP-запити від веб-застосунку, з автоматичним масштабуванням від нуля до визначеної максимальної кількості екземплярів. Масштабування від нуля означає, що у періоди відсутності запитів жоден контейнер не працює і жодних витрат не нараховується, а при надходженні першого запиту Cloud Run автоматично запускає новий екземпляр контейнера. Сервіс-воркер розгортається як внутрішній сервіс, недоступний ззовні, який отримує завдання виключно через push-підписку Cloud Pub/Sub. Для сервісу-воркера налаштовано збільшені ліміти обчислювальних ресурсів та часу виконання запиту порівняно з API-сервісом, що зумовлено обчислювально вимогливою природою операцій з медіафайлами. Кожен сервіс має власний сервісний обліковий запис із мінімально необхідними правами доступу, що реалізує принцип найменших привілеїв [55]: API-сервіс має права на читання та запис у базу даних та публікацію повідомлень у Pub/Sub, але не має доступу до Dead Letter Queue; сервіс-воркер має права на читання та запис у хмарне сховище та оновлення статусів у базі даних, але не може публікувати повідомлення у основний топік.

Таким чином, спроєктований конвеєр CI/CD у поєднанні з декларативним управлінням інфраструктурою через Terraform забезпечує повний цикл від об'єднання змін у репозиторії до їх розгортання у хмарному середовищі. Вибірковість збірок на основі аналізу змін зменшує час та вартість кожного розгортання, модульна організація Terraform-конфігурації усуває дублювання і забезпечує однорідність розгортання сервісів, а зберігання всієї конфігурації інфраструктури у репозиторії поруч із кодом додатків гарантує відтворюваність середовища розгортання з будь-якого стану вихідного коду.

4 РОЗРОБКА СИСТЕМИ

4.1 Розробка серверного API-сервісу

Бізнес-логіка API-сервісу зосереджена у двох сервісних класах: сервісі автентифікації та сервісі управління проєктами (див. додаток Б).

Сервіс автентифікації реалізує два ендпоінти:

– реєстрація — сервіс перевіряє унікальність наданої електронної адреси у базі даних. У разі конфлікту повертається відповідна помилка. Якщо адреса вільна, пароль хешується з використанням алгоритму bcrypt [56] та додаткової серверної солі, після чого новий запис користувача зберігається у базі. На завершення сервіс формує підписаний токен доступу, що містить ідентифікатор, ім'я та прапорець тестувальника, і повертає його клієнту.

– вхід у систему — сервіс знаходить запис користувача за електронною адресою та порівнює наданий пароль зі збереженим хешем. В обох випадках невідповідності (користувача не знайдено або пароль невірний) повертається однакова помилка — без розкриття того, яка саме перевірка провалилась, що унеможливлює атаки на перебір облікових даних. За успішної верифікації формується та повертається токен доступу аналогічно до реєстрації. Термін дії токена обмежений однією годиною.

Сервіс управління проєктами реалізує п'ять ендпоінтів. Усі вони захищені перевіркою токена доступу, а також додатковим прапорцем тестувальника, що дозволяє обмежувати доступ до системи на етапі бета-тестування:

– отримання списку проєктів — сервіс вибирає з бази даних усі проєкти поточного користувача, відсортовані за датою створення у зворотному порядку, з підтримкою посторінкової навігації через параметри зміщення та ліміту. Для кожного проєкту у списку динамічно формуються тимчасові підписані посилання для читання обкладинки відео із хмарного сховища — безпосередні URL файлів у відповідь не повертаються.

– отримання деталей проєкту — сервіс знаходить конкретний проєкт за його ідентифікатором, перевіряє належність проєкту поточному користувачу та повертає повний набір метаданих. Разом із базовими полями клієнту повертаються тимчасові підписані посилання для читання оригінального відео, обкладинки та файлу субтитрів — якщо вони вже згенеровані. Саме цей ендпоінт використовується веб-застосунком для періодичного опитування стану обробки.

– створення проєкту — перед збереженням сервіс виконує дві бізнес-перевірки: загальна кількість проєктів користувача не перевищує встановленого ліміту, а також відсутній інший проєкт, що перебуває у стані активного транскрибування. Такі обмеження захищають систему від надмірного навантаження з боку одного користувача. За умови проходження перевірок запис проєкту зберігається у базі зі статусом очікування, після чого для нього формується тимчасове підписане посилання для прямого завантаження відео у хмарне сховище і повертається клієнту разом із ідентифікатором проєкту.

– ініціювання експорту відео — сервіс перевіряє, що транскрибування проєкту вже завершено, та що рендеринг не було розпочато раніше. Остання перевірка забезпечує ідемпотентність: повторний виклик ендпоінту для проєкту, що вже перебуває в черзі або опрацьовується, ігнорується без помилки. За виконання умов сервіс публікує повідомлення у окрему чергу для рендерингу та оновлює статус завантаження проєкту.

– видалення проєкту — сервіс блокує видалення, якщо проєкт наразі перебуває у стані активного транскрибування або рендерингу. За відсутності блокувань послідовно видаляються всі пов'язані файли з хмарного сховища — оригінальне відео, обкладинки, файл субтитрів та фінальне відео із вбудованими субтитрами, — після чого запис проєкту видаляється з бази даних. Помилки видалення окремих файлів зі сховища логуються, але не перешкоджають завершенню операції.

– отримання посилання для завантаження результату — ендпоінт перевіряє, що рендеринг відео завершено, після чого формує тимчасове підписане посилання для читання відео із вбудованими субтитрами. Для

запобігання зловживанням ендпоінт захищено обмеженням частоти звернень на рівні окремого користувача.

4.2 Розробка сервісу-воркера

Сервіс-воркер є обчислювальним ядром системи та реалізований як окремий Fastify-застосунок із двома незалежними обробниками вхідних повідомлень — для транскрибування та для рендерингу (див. додаток Б). Обидва обробники побудовані за однаковим принципом: вони приймають push-повідомлення від черги, виконують обчислювально вимогливий конвеєр і повертають код успіху або помилки, що безпосередньо визначає поведінку брокера щодо повторної доставки.

4.2.1 Розробка конвеєру транскрибування

Конвеєр транскрибування запускається при надходженні повідомлення про завантаження відеофайлу у хмарне сховище. Першою дією сервіс зчитує поточний стан проєкту з бази даних і перевіряє його статус — якщо він вже відповідає стану завершення або активної обробки, повідомлення підтверджується без виконання жодних операцій. Це є ключовим механізмом захисту від повторної обробки, оскільки черга гарантує доставку «принаймні один раз». Реалізація логіки ідемпотентності надана у лістингу 4.1.

Лістинг 4.1 – Програмний код логіки ідемпотентності

```
const existingProject = await projectRepo.findOne({
  where: { originalVideoUrl: videoPath },
})
if (!existingProject) {
  fastify.log.warn(
    `Project with video path ${videoPath} not found. Skip processing.`
  )
  return
}
```

Продовження 1 Лістингу 4.1

```
if (
    existingProject.transcriptionStatus ===
      ProjectTranscriptionStatus.PROCESSING ||
    existingProject.transcriptionStatus ===
ProjectTranscriptionStatus.COMPLETED
  ) {
    fastify.log.warn(
      `Project with video path ${videoPath} is
${existingProject.transcriptionStatus}. Skip processing.`
    )
    return
  }
```

За умови коректного стану воркер завантажує відеофайл із хмарного сховіща у тимчасову файлову систему контейнера та паралельно виконує наступні операції засобами FFmpeg: вилучення аудіодоріжки, генерацію обкладинки відео та вилучення метаданих відеофайлу.

При вилученні аудіо файл конвертується у моноканальний формат із частотою дискретизації 16 кГц та бітрейтом 64 кбіт/с — таке поєднання параметрів є достатнім для якісного розпізнавання мовлення і водночас обмежує розмір аудіофайлу в межах ліміту OpenAI API. Функція вилучення аудіодоріжки надана у лістингу 4.2.

Лістинг 4.2 – Програмний код функції вилучення аудіодоріжки

```
export async function extractAudio(videoPath: string): Promise<string> {
  const outputPath = path.join(os.tmpdir(), `audio-
${crypto.randomUUID()}.mp3`)

  await execFileAsync('ffmpeg', [
    '-i',
    videoPath,
    '-vn', // no video stream
```

Продовження 1 лістингу 4.2

```
'-ac',
    '1', // mono
    '-ar',
    '16000', // 16kHz sample rate (sufficient for speech)
    '-b:a',
    '64k', // 64kbps → handles up to ~54 min before hitting Whisper's
25MB limit
    '-f',
    'mp3',
    '-y', // overwrite output without asking
    outputPath,
  ])

  return outputPath
}
```

Обкладинка формується шляхом захоплення одного кадру на першій секунді відео та масштабування його до ширини 640 пікселів зі збереженням пропорцій. Функція вилучення обкладинки надана у лістингу 4.3.

Лістинг 4.3 – Програмний код функції вилучення обкладинки відео

```
export async function extractThumbnail(videoPath: string):
Promise<string> {
  const outputPath = path.join(os.tmpdir(), `thumb-
${crypto.randomUUID()}.jpg`)

  await execFileAsync('ffmpeg', [
    '-i',
    videoPath,
    '-ss',
    '00:00:01', // seek to 1 second
    '-vframes',
    '1', // extract a single frame
```

Продовження 1 лістингу 4.3

```
'-vf',
    'scale=640:-1', // 640px wide, maintain aspect ratio
    '-y', // overwrite output without asking
    outputPath,
  ])

  return outputPath
}
```

Окрім того, засобами аналізатора відеопотоку зчитуються технічні метадані файлу — тривалість, розміри кадру та розмір файлу. Функція вилучення метаданих надана у лістингу 4.4.

Лістинг 4.4 – Програмний код функції вилучення метаданих відео

```
export async function extractVideoMetadata(
  videoPath: string
): Promise<ProjectMetadata> {
  const fileSize = fs.statSync(videoPath).size

  const { stdout } = await execFileAsync('ffprobe', [
    '-v',
    'quiet',
    '-print_format',
    'json',
    '-show_streams',
    '-show_format',
    videoPath,
  ])

  const probe = JSON.parse(stdout)
  const duration = probe.format?.duration
    ? parseFloat(probe.format.duration)
```

Продовження 1 лістингу 4.4

```
: null
  const videoStream = probe.streams?.find(
    (s: { codec_type: string }) => s.codec_type === 'video'
  )
  const width: number | null = videoStream?.width ?? null
  const height: number | null = videoStream?.height ?? null

  return { duration, fileSize, width, height }
}
```

Вилучений аудіофайл надсилається до OpenAI API для розпізнавання мовлення. Запит явно вказує українську мову та формат відповіді у вигляді готового файлу субтитрів, а також містить підказку для моделі щодо очікуваних лінгвістичних характеристик — стандартна пунктуація, великі літери, правильні відмінкові закінчення. Отриманий вміст зберігається у тимчасовий файл і завантажується у хмарне сховище за визначеним шляхом. Функція генерації субтитрів через OpenAI API надана у лістингу 4.5.

Лістинг 4.5 – Програмний код функції генерації субтитрів

```
export async function transcribeAudio(audioPath: string): Promise<string>
{
  const openai = new OpenAI({ apiKey: env.openaiApiKey })

  const transcription = await openai.audio.transcriptions.create({
    file: fs.createReadStream(audioPath),
    model: 'whisper-1',
    language: 'uk',
    response_format: 'vtt',
    prompt:
      'Це запис українською мовою. Будь ласка, використовуй стандартну
пунктуацію, великі літери та правильні закінчення слів.',
  })
```

Продовження 1 лістингу 4.5

```
  // When response_format is 'vtt', the SDK returns the raw string
content
  return transcription as unknown as string
}
```

На завершення сервіс атомарно оновлює запис проєкту у базі даних: записує шляхи до збережених файлів субтитрів і файлу обкладники, зберігає технічні метадані відео та виставляє статус транскрибування як завершений. У разі виникнення будь-якої помилки на будь-якому кроці конвеєру сервіс записує її текст у відповідне поле бази даних, виставляє статус невдачі та повертає код помилки — що спонукає брокер повідомлень виконати повторну доставку або, після вичерпання спроб, перенаправити повідомлення у чергу «мертвих повідомлень». У кінці процесу, тимчасові файли завантажені у пам’ять контейнера видаляються, що гарантує звільнення дискового простору незалежно від результату обробки.

4.2.2 Розробка конвеєру рендерингу

Конвеєр рендерингу запускається при надходженні повідомлення про запит на генерацію відео із вбудованими субтитрами. Він побудований за аналогічною до транскрибування схемою з перевіркою ідемпотентності на початку. Після підтвердження готовності сервіс завантажує оригінальний відеофайл та файл субтитрів у тимчасову файлову систему і передає їх на вхід FFmpeg.

Операція вбудовування субтитрів реалізована через граф відеофільтрів FFmpeg із фільтром накладання тексту. Параметри стилізації визначають шрифт, розмір, колір тексту та контур — білий текст із чорним обведенням для максимальної читабельності на довільному відеофоні. Відео кодується за допомогою програмного кодека із фіксованим рівнем якості та швидким пресетом, тоді як аудіодоріжка копіюється без перекодування, що суттєво скорочує час обробки. Функція вбудовування субтитрів надана у лістингу 4.6.

Лістинг 4.6 – Програмний код функції вбудовування субтитрів

```
export async function burnSubtitles(
  videoPath: string,
  subtitlePath: string
): Promise<string> {
  const outputPath = path.join(
    os.tmpdir(),
    `rendered-${crypto.randomUUID()}.mp4`
  )

  await execFileAsync(
    'ffmpeg',
  [
    '-i',
    videoPath,
    '-vf',

`subtitles=filename=${subtitlePath}:force_style=FontName=Arial\\,FontSize
=16\\,PrimaryColour=&H00FFFFFF&\\,OutlineColour=&H00000000&\\,Outline=1`,
    '-c:v',
    'libx264',
    '-crf',
    '23',
    '-preset',
    'fast',
    '-c:a',
    'copy',
    '-y',
    outputPath,
  ]
)

  return outputPath
}
```

Після вбудовування субтитрів, готовий файл завантажується у хмарне сховище, після чого запис проєкту оновлюється шляхом збереження посилання на фінальний відеозапис у хмарному сховищі та встановлення статусу рендерингу як завершеного.

4.3 Розгортання хмарної інфраструктури та CI/CD

Уся хмарна інфраструктура системи описана декларативно засобами Terraform та організована за модульним принципом.

4.3.1 Налаштування обчислювальних контейнерів

Детально розглянемо саме конфігурацію Cloud Run контейнерів для кожного сервісу, так як вони є головними обчислювальними елементами у системі, що мають найбільший вплив на масштабованість та надійність. Два серверних сервіси системи мають принципово різні профілі навантаження, що зумовлює суттєво відмінні конфігурації їх обчислювальних ресурсів.

API-сервіс налаштовано на 2 процесорних ядра та 1 ГБ оперативної пам'яті з масштабуванням від 0 до 4 екземплярів контейнера та лімітом у 80 одночасних запитів на один екземпляр (див. лістинг 4.7).

Лістинг 4.7 – Програмний код налаштування контейнера API-сервісу

```
module "web_service_cloud_run" {
  source = "./modules/cloud-run-service"

  name                          = "web-service"
  location                      = var.region
  service_account_id            = "web-service-account"
  artifact_registry_repository =
google_artifact_registry_repository.subgen_service.name
  publicly_accessible           = true
  deployer_service_accounts     = [local.artifacts_deployer_sa]
```

Продовження 1 лістингу 4.7

```
 min_instance_count                = 0
  max_instance_count                = 4
  cpu                               = "2"
  memory                            = "1Gi"
  timeout                           = "120s"
  cpu_idle                          = false
  startup_cpu_boost                 = true
  max_instance_request_concurrency = 80
}
```

Збільшення кількості ядер до двох зумовлене потребою у паралельному виконанні операцій хешування паролів: алгоритм bcrypt є навмисно обчислювально затратним, і на одному ядрі з увімкненим дроселюванням він ефективно серіалізує всі одночасні запити автентифікації. Ліміт оперативної пам'яті в 1 ГБ забезпечує достатній запас для купи Node.js, гідратації сутностей TypeORM та буферизації до 80 одночасних об'єктів запитів в оперативній пам'яті екземпляра. Максимальна кількість у 4 екземпляри забезпечує пропускну здатність у 320 одночасних слотів — з трикратним запасом відносно цільового навантаження у 100 одночасних користувачів. Тайм-аут у 120 секунд покриває найгірший сценарій при отриманні списку проєктів, де паралельне формування підписаних посилань для обкладинок може суттєво зростати у часі при збільшенні кількості проєктів у користувача.

Сервіс-воркер налаштовано на 2 процесорних ядра та 2 ГБ оперативної пам'яті, з масштабуванням до 10 екземплярів та жорстким лімітом в 1 одночасний запит на екземпляр (див. лістинг 4.8).

Лістинг 4.8 - Програмний код налаштування контейнера сервісу-воркеру

```
module "worker_service_cloud_run" {
  source = "./modules/cloud-run-service"

  name                          = "worker-service"
```

Продовження 1 лістингу 4.8

```
  location                      = var.region
  service_account_id            = "worker-service-account"
  artifact_registry_repository =
google_artifact_registry_repository.subgen_service.name
  publicly_accessible           = false
  deployer_service_accounts     = [local.artifacts_deployer_sa]

  min_instance_count               = 0
  max_instance_count               = 10
  cpu                              = "2"
  memory                           = "2Gi"
  timeout                          = "600s" # aligned with Pub/Sub ack
deadline
  cpu_idle                         = false
  startup_cpu_boost                = true
  max_instance_request_concurrency = 1
}
```

Ключовою особливістю Cloud Run є те, що тимчасова файлова система /tmp — куди воркер завантажує відеофайли під час обробки — є не дисковою, а оперативною пам'яттю. Пікове споживання /tmp одним завданням рендерингу становить близько 250 МБ, при ліміті у 50 МБ для завантаження вихідного відеофайлу, а з урахуванням процесів Node.js та FFmpeg загальний пік на завдання досягає ~700 МБ. За відсутності обмеження на одночасність декілька завдань на одному екземплярі накопичували б тимчасові файли разом, що при десяти паралельних завданнях призвело б до вичерпання оперативної пам'яті та примусового перезапуску контейнера. Саме тому обмеження у 1 запит на екземпляр є принциповим архітектурним рішенням: кожне завдання отримує ізольований доступ до всіх ресурсів екземпляра, а горизонтальне масштабування до 10 екземплярів забезпечує підтримку 10 паралельних користувачів. Крім того, два ядра дозволяють FFmpeg задіяти багатопотокове кодування libx264, а

конкуренції між паралельними процесами FFmpeg за процесорний час при такому підході не виникає. Тайм-аут у 600 секунд узгоджений із дедлайном підтвердження Pub/Sub-повідомлення — щоб примусовий перезапуск контейнера не відбувся раніше, ніж брокер вирішить повторно доставити повідомлення.

Обидва сервіси мають дві спільні конфігурації, що стосуються управління процесорними ресурсами. Параметр cpu_idle = false вимикає дроселювання процесора між запитами на працюючому екземплярі. За замовчуванням Cloud Run знижує пріоритет процесора для екземпляра, який не обробляє активний запит, — це призводить до затримок на початку кожного нового запиту, доки процесор «прогрівається». Для API-сервісу це критично через bcrypt, для воркера — через FFmpeg, що розпочинає кодування одразу після отримання завдання. Оскільки мінімальна кількість екземплярів обох сервісів дорівнює нулю, цей параметр не спричиняє цілодобових витрат: процесор залишається активним лише тоді, коли екземпляр реально обслуговує трафік. Параметр startup_cpu_boost = true надає тимчасово підвищену кількість процесорних ресурсів під час запуску нового екземпляра, скорочуючи тривалість «холодного старту» — що особливо важливо для воркера, контейнер якого є важчим через вбудований FFmpeg.

4.3.2 Налаштування черг повідомлень

Для кожної з двох черг повідомлень — черги транскрибування та черги рендеринг — налаштовано окрему чергу для «мертвих повідомлень» та відповідну підписку (див. лістинг 4.9).

Лістинг 4.9 – Програмний код налаштування черг повідомлень

```
resource "google_pubsub_topic" "video_uploads" {
  name = "video-uploads"
}
resource "google_pubsub_topic" "render_requests" {
  name = "render-requests"
}
```

Продовження 1 лістингу 4.9

```
resource "google_pubsub_topic" "video_uploads_dlq" {
  name = "video-uploads-dlq"
}
resource "google_pubsub_topic" "render_requests_dlq" {
  name = "render-requests-dlq"
}
locals {
  worker_subscriptions = {
    "video-uploads" = {
      topic_name     = google_pubsub_topic.video_uploads.name
      dlq_topic_id   = google_pubsub_topic.video_uploads_dlq.id
      dlq_topic_name = google_pubsub_topic.video_uploads_dlq.name
      push_path      = "/transcription/on-video-upload"
    }
    "render-requests" = {
      topic_name     = google_pubsub_topic.render_requests.name
      dlq_topic_id   = google_pubsub_topic.render_requests_dlq.id
      dlq_topic_name = google_pubsub_topic.render_requests_dlq.name
      push_path      = "/render/on-render-request"
    }
  }
}
resource "google_pubsub_subscription" "worker_push" {
  for_each = local.worker_subscriptions

  name  = "${each.key}-worker-push"
  topic = each.value.topic_name

  push_config {
    push_endpoint =
"${module.worker_service_cloud_run.service_uri}${each.value.push_path}"

    oidc_token {
```

Продовження 2 лістингу 4.9

```
      service_account_email = google_service_account.pubsub_invoker.email
    }
  }

  ack_deadline_seconds       = 600
  message_retention_duration = "86400s"

  retry_policy {
    minimum_backoff = "10s"
    maximum_backoff = "600s"
  }

  dead_letter_policy {
    dead_letter_topic     = each.value.dlq_topic_id
    max_delivery_attempts = 7
  }
}

resource "google_pubsub_subscription" "worker_dlq_pull" {
  for_each = local.worker_subscriptions

  name  = "${each.key}-dlq-pull"
  topic = each.value.dlq_topic_name

  message_retention_duration = "604800s"
  ack_deadline_seconds       = 600
}
```

Механізм повторних спроб реалізований через поведінку самого воркера: у разі невдачі обробки він повертає код HTTP 500, що є сигналом для Pub/Sub виконати повторну доставку повідомлення з експоненціальною затримкою. Після вичерпання встановленої кількості спроб Pub/Sub автоматично

перенаправляє повідомлення у чергу «мертвих повідомлень», виводячи його з основної черги та зберігаючи для подальшого аналізу.

4.3.3 Налаштування CI/CD

CI/CD конвеєр реалізований як єдиний робочий процес GitHub Actions, що запускається при кожному злитті змін у головну гілку. Першим кроком конвеєр визначає, які з трьох застосунків зазнали змін, порівнюючи поточний коміт із попереднім засобами утиліти turbo-ignore. Це унеможливлює надлишкові збірки: зміна у коді воркера не тригерить перезбірку API-сервісу.

Після аналізу змін конвеєр виконує три незалежні гілки: застосовує Terraform-конфігурацію до хмарної інфраструктури, застосовує міграції схеми бази даних, і лише після успішного завершення обох — умовно розгортає змінені застосунки. Серверні сервіси збираються у Docker-образи з тегом на основі хешу коміту та завантажуються у реєстр контейнерних образів, веб-застосунок генерується як статичний експорт і синхронізується у хмарне сховище. Порядок виконання — спочатку міграції, потім розгортання — гарантує, що новий код ніколи не потрапляє у робоче середовище раніше, ніж схема бази даних приведена у відповідність до його очікувань.

Схема розгортання системи надана на рисунку 4.1.

Рисунок 4.1 – Схема розгортання системи

5 ТЕСТУВАННЯ СИСТЕМИ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

Тестування розробленої системи проводилося у чотири етапи: функціональне тестування користувацьких сценаріїв використання системи, тестування механізмів надійності, тестування якості розпізнавання мовлення та тестування продуктивності конвеєра обробки.

5.1 Тестування користувацьких сценаріїв

Функціональне тестування користувацьких сценаріїв використання системи охоплювало повний шлях користувача у системі відповідно до сценаріїв використання, визначених у підрозділі 3.1. Тестування проводилось із використанням веб-браузера Google Chrome.

На першому кроці перевірялась коректність роботи автентифікації: реєстрація нового облікового запису із заповненням усіх обов'язкових полів — електронної адреси, імені, прізвища та пароля — завершувалась успішним створенням запису користувача у базі даних та автоматичним перенаправленням на інформаційну панель (див. рисунки 5.1 – 5.2).

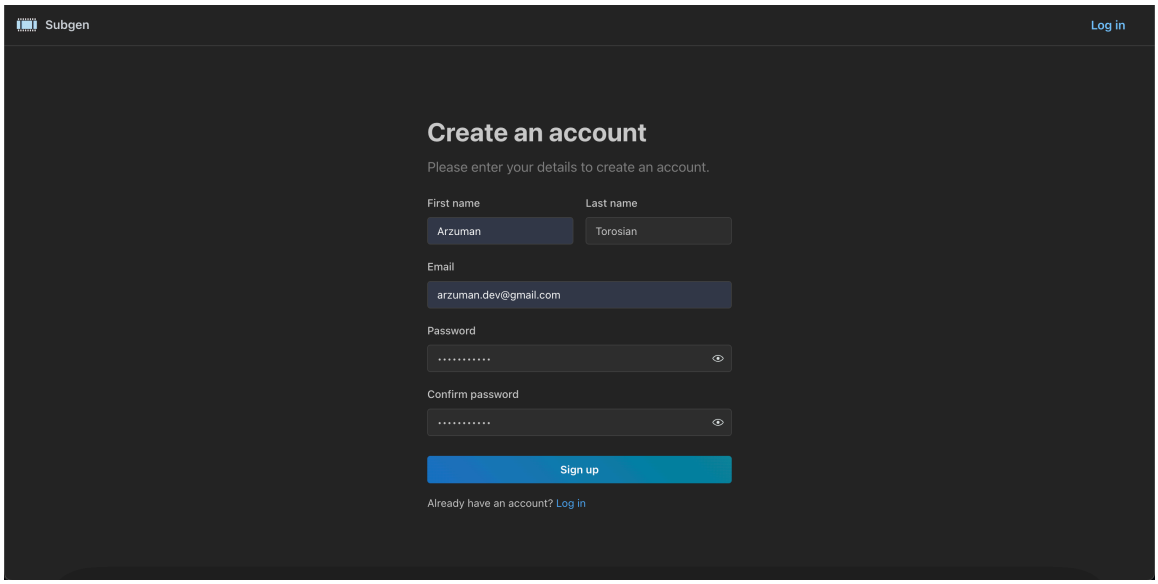


Рисунок 5.1 – Скріншот сторінки реєстрації

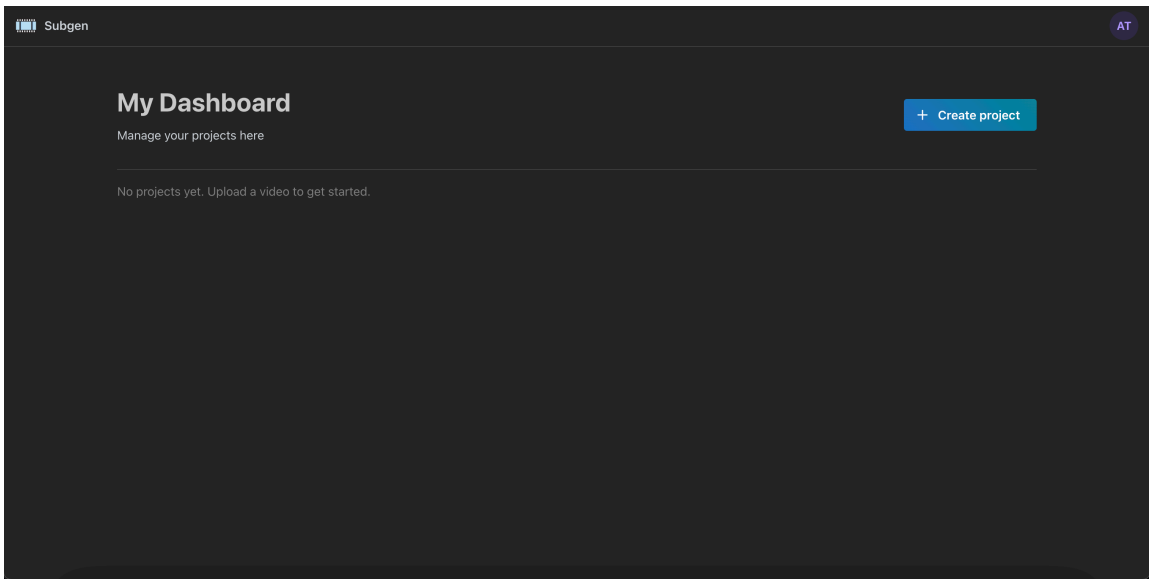


Рисунок 5.2 – Скріншот головної сторінки після входу в систему

На другому кроці тестувалось створення проєкту та завантаження відео. Після вибору відеофайлу веб-застосунок отримував від API підписане посилання та завантажував файл безпосередньо у хмарне сховище. Впродовж завантаження та подальшої асинхронної обробки інтерфейс відображав індикатор стану (див. рисунок 5.3).

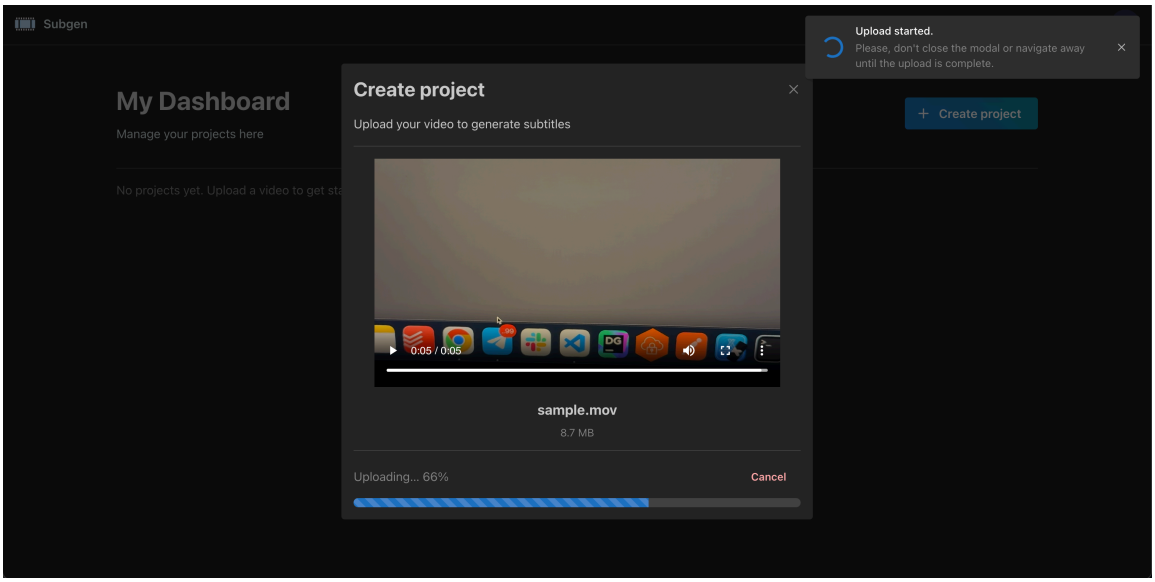


Рисунок 5.3 – Скріншот завантаження файлу до системи

По завершенні транскрибування статус транскрибування змінювався на завершений і в плеєрі з'являлись накладені субтитри (див. рис. 5.4 – 5.5).

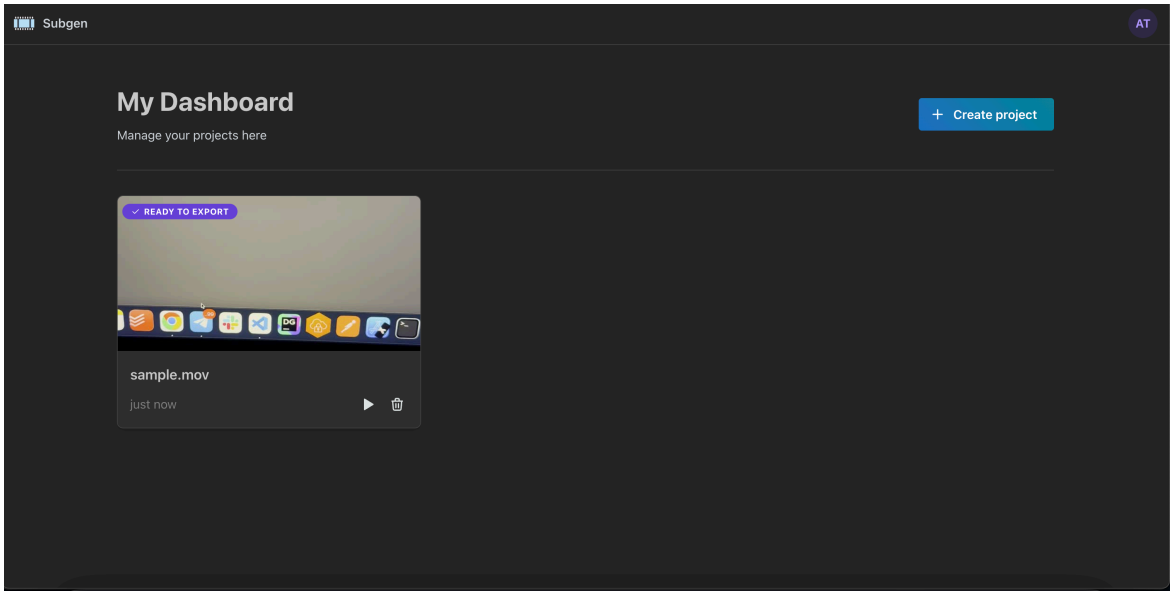


Рисунок 5.4 – Скріншот головної сторінки після завершення транскрибування відео

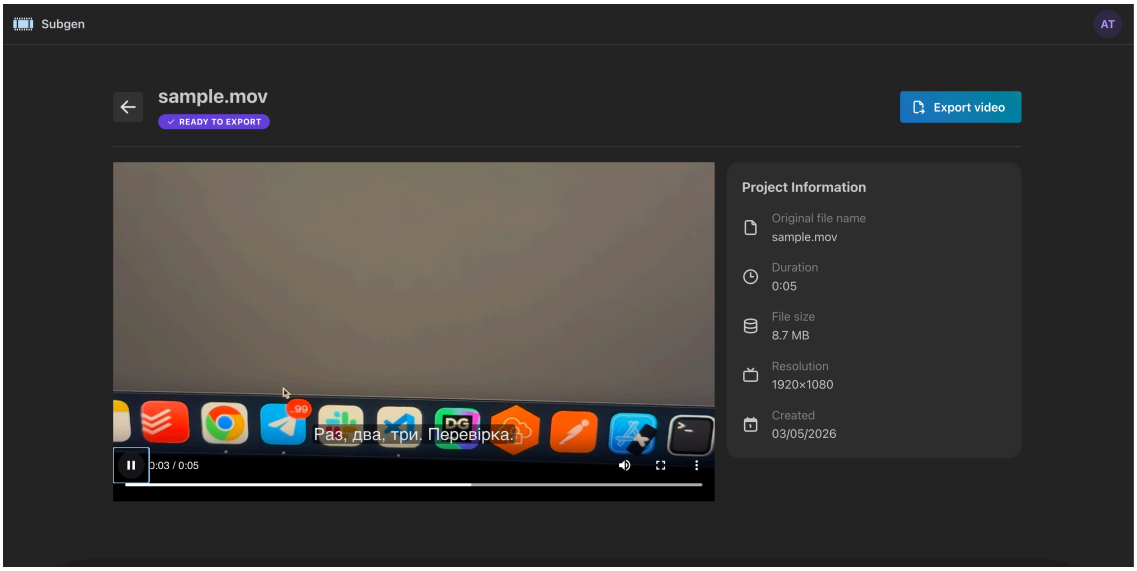


Рисунок 5.5 – Скріншот сторінки відеоплеєру для відео зі згенерованими субтитрами

На третьому кроці тестувалось ініціювання процесу вбудовування субтитрів та завантаження готового відео. Після натискання кнопки експорту система переводила проєкт у стан обробки (див. рисунок 5.6)

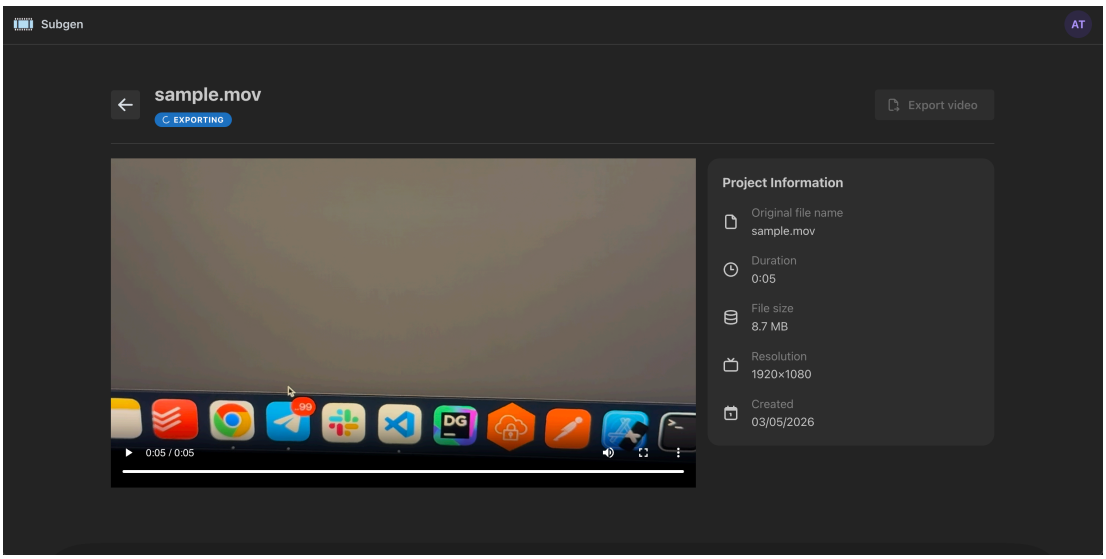


Рисунок 5.6 – Скріншот сторінки відеоплеєру після ініціювання процесу

Після завершення процесу вбудовування субтитрів система надавала посилання для завантаження відеофайлу із вбудованими субтитрами (див. рисунок 5.7).

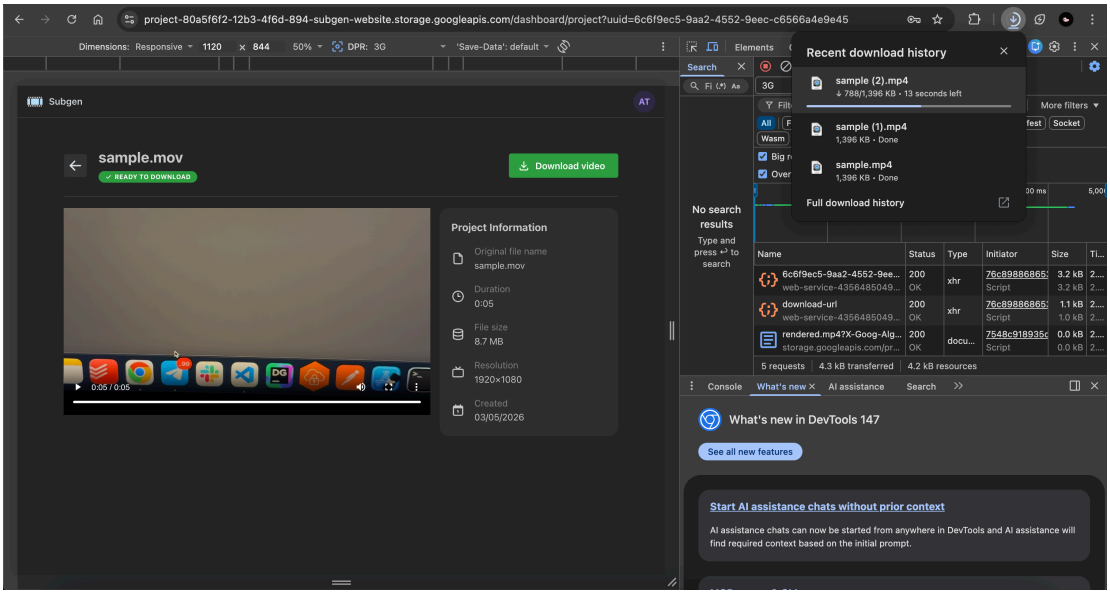


Рисунок 5.7 – Скріншот сторінки після ініціювання завантаження відео із вбудованими субтитрами

Завантажене відео було переглянуте у стандартному програвачі операційної системи macOS, щоб підтвердити коректність субтитрів (див. рис. 5.8).

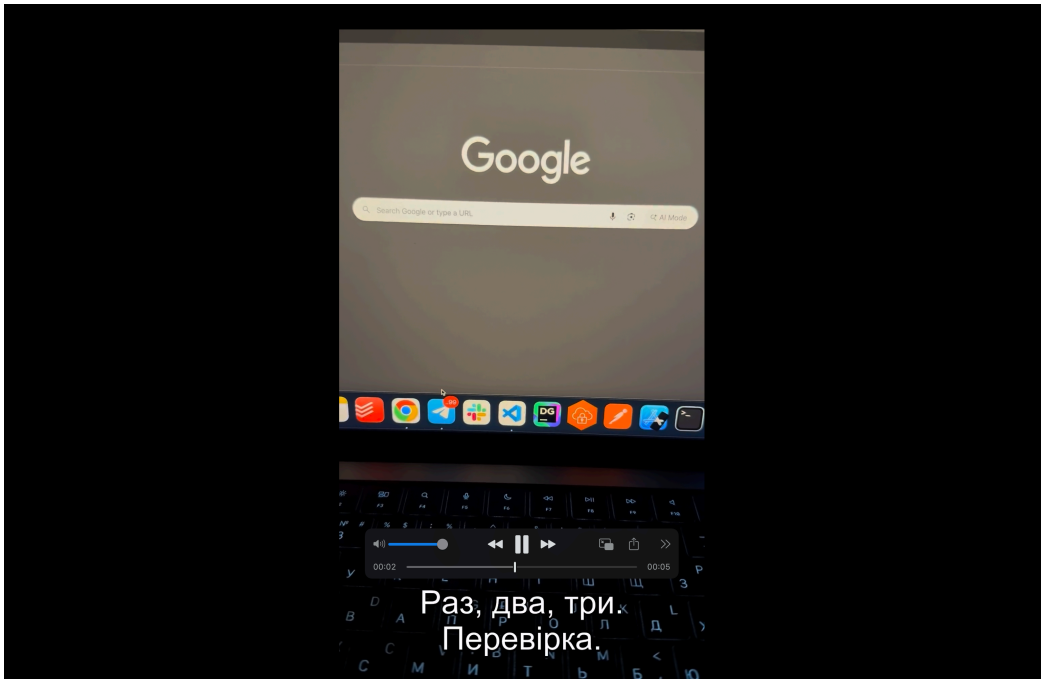


Рисунок 5.8 – Скріншот перегляду готового відео із вбудованими субтитрами

Таким чином, тестування підтвердило коректність реалізації всіх сценаріїв використання системи: від реєстрації до отримання готового відео із вбудованими субтитрами.

5.2 Тестування механізмів надійності

Верифікація механізмів надійності проводилась відповідно до архітектурних рішень, описаних у підрозділі 3.4. Перевірялись два ключових механізми: ідемпотентність обробки повідомлень та поведінка черги «мертвих повідомлень».

Для верифікації ідемпотентності було проведено наступний тест: після успішного завершення транскрибування проєкту повідомлення з тим самим ідентифікатором було вручну опубліковано у топік Pub/Sub через консоль Google Cloud. Аналіз логів сервісу-воркера у Cloud Run підтвердив, що при отриманні повторного повідомлення сервіс зчитав поточний статус проєкту з бази даних, виявив значення статусу «завершено» та негайно підтвердив отримання

повідомлення — без завантаження відеофайлу, без виклику утиліти FFmpeg та без звернення до зовнішнього API розпізнавання мовлення (див. рисунок 5.9).

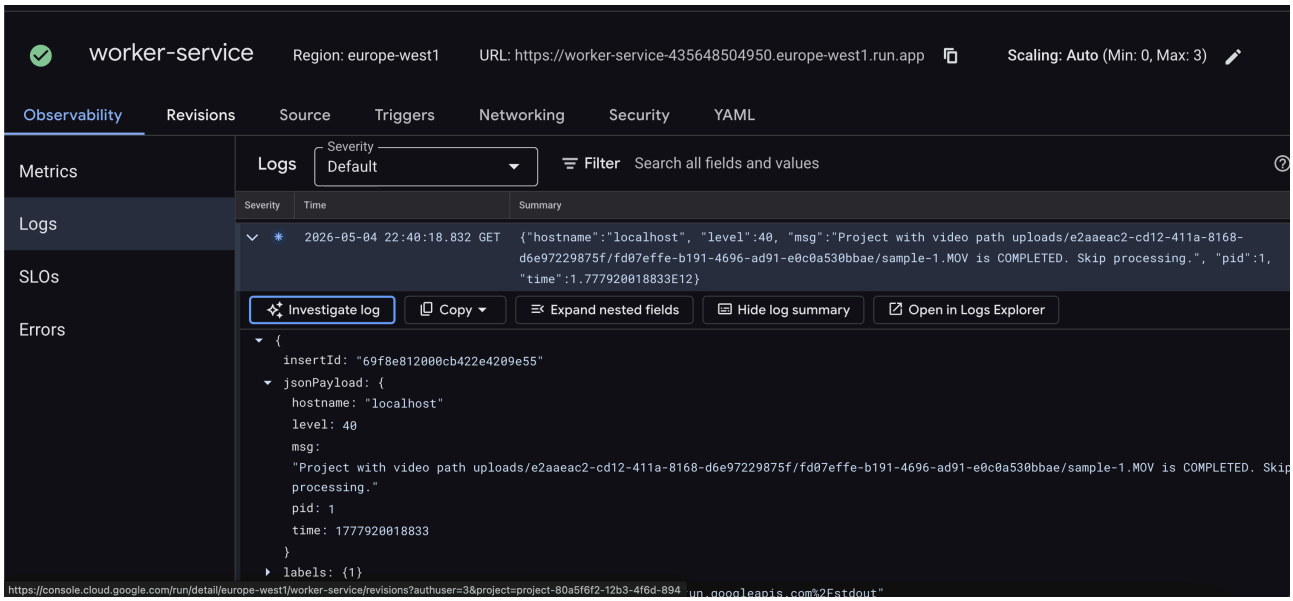


Рисунок 5.9 – Скріншот логів воркер-сервісу при обробці дубльованого запису

Це підтвердило, що перевірка стану на початку конвеєра, описана у підрозділі 4.2, коректно запобігає повторній обробці та подвійному нарахуванню витрат за виклик зовнішнього API.

Для верифікації механізму «мертвих повідомлень» у систему було завантажено навмисно пошкоджений файл — текстовий файл із розширенням .mp4. При першій спробі обробки сервіс повернув зафіксував помилку, встановив статус FAILED у базі даних та повернув код HTTP 500, що сигналізувало брокеру про необхідність повторної доставки. Pub/Sub виконав повторну доставку повідомлення відповідно до налаштованої політики — кожна наступна спроба завершувалась ідентичним результатом, оскільки пошкоджений файл залишався незмінним. Після вичерпання ліміту у 7 спроб повідомлення було автоматично перенаправлено у чергу «мертвих повідомлень» (див. рисунок 5.10). Основна черга при цьому залишалась розблокованою — наступне коректне повідомлення, опубліковане після пошкодженого, було оброблено успішно без затримок

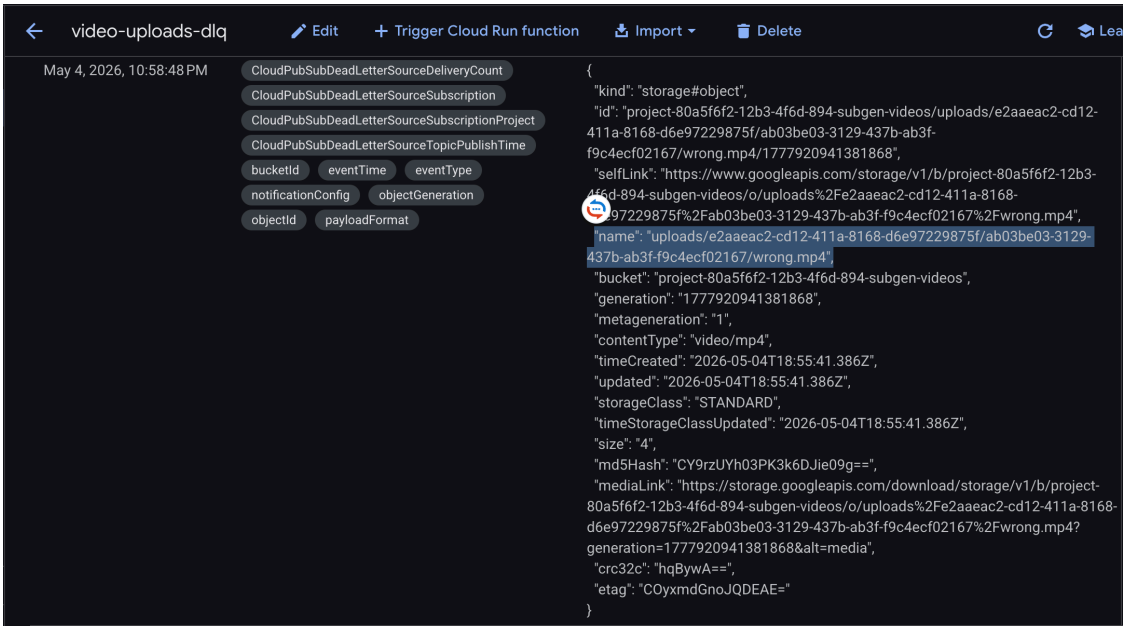


Рисунок 5.10 – Скріншот повідомлення з черги «мертвих повідомлень»

В інтерфейсі користувача картка проєкту з пошкодженим файлом відобразила статус помилки, що зображено на рисунку 5.11.

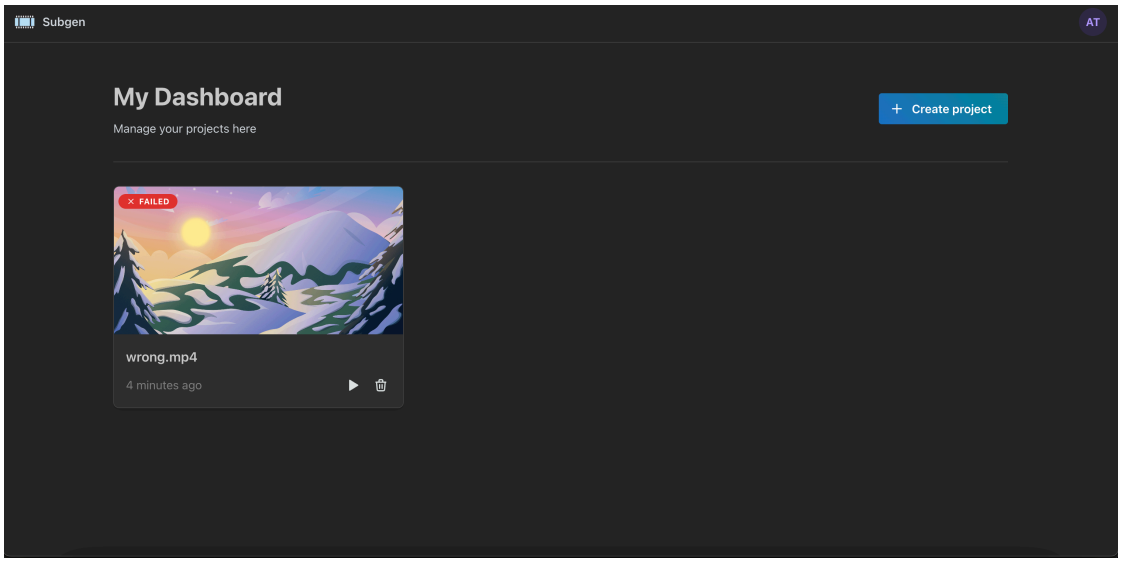


Рисунок 5.11 – Скріншот головної сторінки застосунку з пошкодженим відеопроектом

Таким чином, тестування підтвердило коректність обох механізмів надійності: ідемпотентності та черги «мертвих повідомлень».

5.3 Тестування якості розпізнавання мовлення

Для кількісної оцінки якості розпізнавання мовлення на україномовному контенті застосовано метрику WER, що обчислюється за формулою (1.1).

Для проведення дослідження підготовлено три тестові зразки із різними характеристиками аудіосигналу, що представляють типові сценарії використання системи. Для кожного зразка вручну підготовлено еталонний транскрипт. Кожен зразок було оброблено системою двічі: із системною підказкою, що передається моделі розпізнавання у параметрі запиту та визначає вимоги до пунктуації, великих літер та відмінкових закінчень, та без неї — для кількісної оцінки ефекту від застосованої промпт-інженерії. Обчислення WER виконано за допомогою бібліотеки jiwer для Python. Результати наведено у на рисунку 5.12.

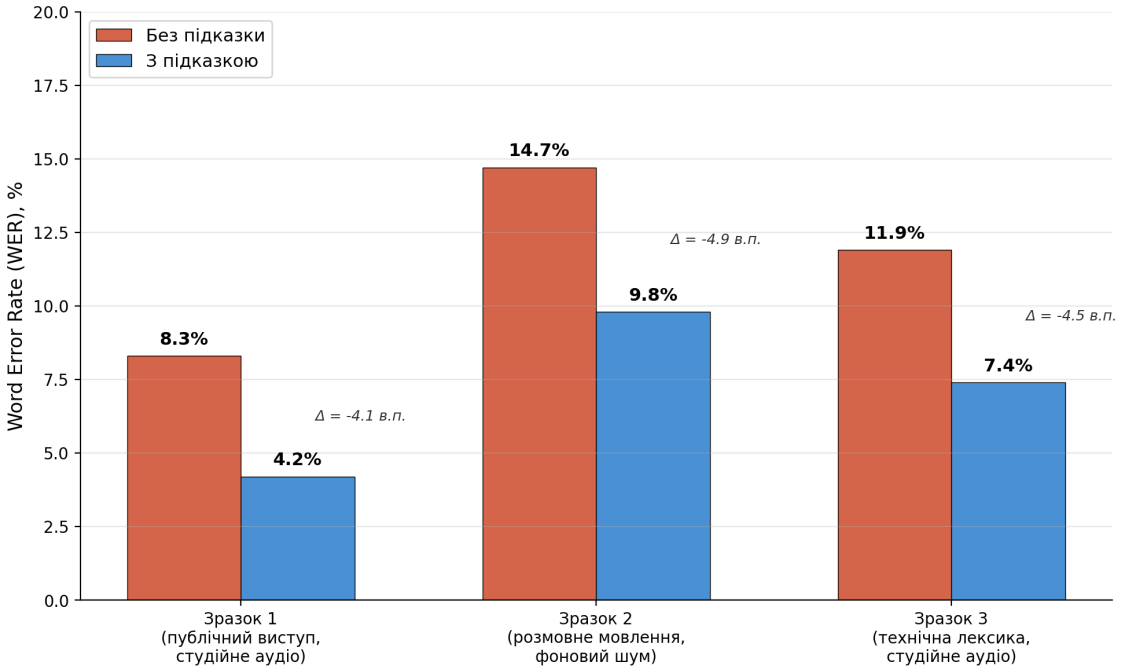


Рисунок 5.12 – Стовпчаста діаграма для WER тестових зразків

Аналіз результатів дозволяє сформулювати кілька висновків. По-перше, на матеріалі зі студійною якістю аудіо та загальновживаною лексикою (зразок 1) модель демонструє WER на рівні 4,2 %, що свідчить про високу точність

розпізнавання української мови та є прийнятним показником для автоматичної генерації субтитрів без ручного редагування. По-друге, наявність фонового шуму (зразок 2) є найбільш впливовим фактором погіршення якості — WER зростає до 9,8 % навіть із підказкою, причому помилки зосереджені переважно на словах із нечіткою артикуляцією та на межах речень, де шум ускладнює розмежування пауз. По-третє, технічна та іншомовна лексика (зразок 3) призводить до помірного зростання WER до 7,4 % за рахунок англомовних термінів, які модель транслітерує кирилицею або замінює фонетично схожими українськими словами.

Найбільш практично значущим результатом є стабільне зниження WER на 4,1–4,9 відсоткових пункти (в.п.) при використанні системної підказки на всіх трьох зразках. Ручний аналіз помилок показав, що підказка впливає переважно на два типи помилок: некоректну розстановку розділових знаків та неправильні відмінкові закінчення — обидва технічно враховуються у WER як заміни, хоча не спотворюють змістову точність транскрипції. Це підтверджує доцільність застосованого у підрозділі 4.2 підходу до промпт-інжинірингу для підвищення читабельності субтитрів.

5.4 Тестування продуктивності конвеєра обробки

Для оцінки часових характеристик конвеєра транскрибування проведено серію вимірювань на відеофайлах різної тривалості. Тестові відео мали однакову роздільну здатність та були записані із однаковою якістю аудіо без фонового шуму, щоб виключити вплив параметрів кодування та характеристик аудіосигналу на час обробки і забезпечити порівнянність результатів між зразками. Для кожного відеофайлу зафіксовано час виконання окремих етапів конвеєра — завантаження файлу з хмарного сховища, обробка засобами FFmpeg, виклик зовнішнього API розпізнавання мовлення та завантаження результатів назад у сховище — за мітками у логах Cloud Run з точністю до секунди. Результати наведено на рисунку 5.13.

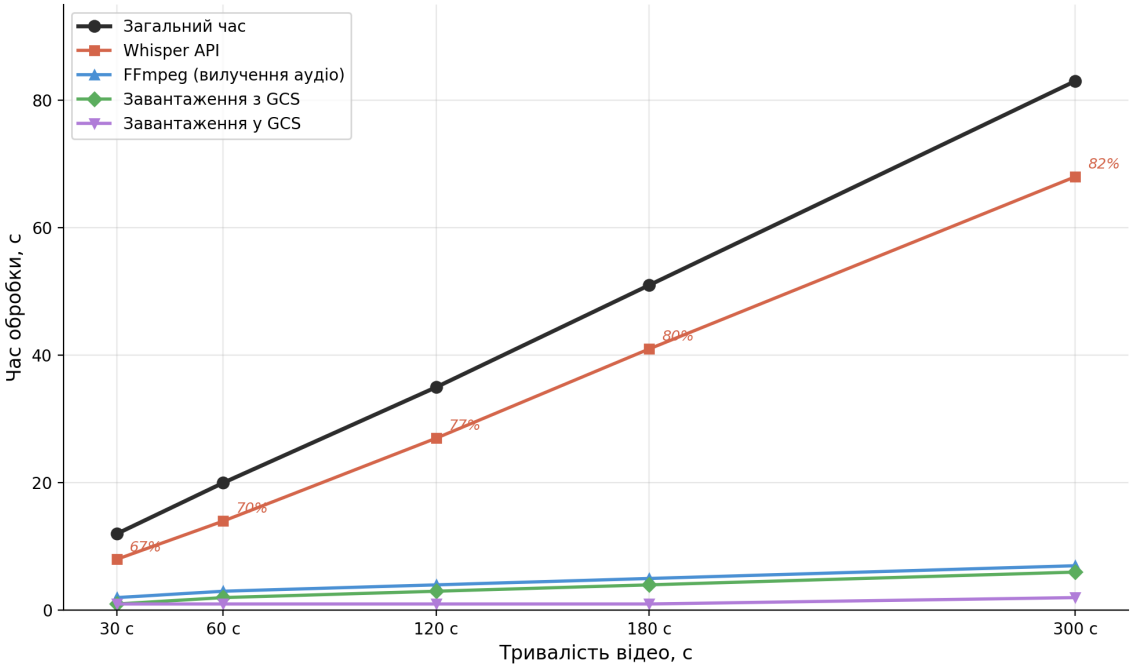


Рисунок 5.13 – Графік залежності часу виконання етапів конвеєра транскрибування від тривалості вхідного відео

Аналіз результатів, наведених на рисунку 5.13, демонструє близьку до лінійної залежність між тривалістю вхідного відео та загальним часом обробки.

Домінуючою складовою на всіх зразках є виклик зовнішнього API розпізнавання мовлення, частка якого стабільно перевищує дві третини загального часу і зростає зі збільшенням тривалості відео. Це пояснюється тим, що інші складові конвеєра зростають значно повільніше: вилучення аудіо засобами FFmpeg та взаємодія з хмарним сховищем разом не перевищують третини загального часу навіть на найкоротших зразках, а на довших — їх сукупна частка знижується ще більше. Таким чином, зі збільшенням тривалості відео структура витрат часу дедалі більше зміщується на користь зовнішнього API.

Для типового цільового контенту системи — коротких відеороликів тривалістю до двох хвилин, призначених для публікації у соціальних мережах, — загальний час обробки не перевищує 35 секунд, що є прийнятним для асинхронного сценарію використання. Водночас результати вказують на те, що оптимізація параметрів команди FFmpeg для вилучення аудіо не дасть

відчутного скорочення загального часу, а подальші зусилля мають бути зосереджені саме на етапі розпізнавання мовлення. Рекомендованими напрямами оптимізації є: самостійне розгортання моделі Whisper на інфраструктурі з GPU, що дозволить уникнути мережевих затримок, усунути залежність від зовнішнього сервісу та забезпечити повний контроль над параметрами моделі; сегментація довгих аудіофайлів із паралельним надсиланням фрагментів на обробку, що дозволить скоротити час очікування за рахунок конкурентного виконання; а також попередня фільтрація тиші та нерелевантних аудіосегментів засобами FFmpeg перед надсиланням на розпізнавання, що зменшить обсяг даних для обробки моделлю без втрати змістовного мовлення.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи проведено комплексне дослідження предметної області автоматичного розпізнавання мовлення та генерації субтитрів для веб-платформ, виконано проектування відповідної системи, з подальшою її розробкою та тестуванням.

Встановлено, що наскрізні нейромережеві архітектури на базі трансформерів досягли рівня якості розпізнавання, достатнього для повної автоматизації генерації субтитрів. Виявлено ключову прогалину в існуючих рішеннях: комерційні платформи функціонують як системи із закритою архітектурою, тоді як відкриті компоненти існують виключно як розрізнені інструменти без інтеграційного рішення рівня веб-платформи. Зазначена прогалина визначає наукову новизну роботи — запропоновано документоване архітектурне рішення, що об'єднує відкриті компоненти у цілісний відтворюваний конвеєр обробки медіаконтенту.

Розроблено архітектурне рішення з декомпозицією на три незалежні компоненти — веб-застосунок, API-сервіс та сервіс-воркер, — з'єднані через чергу повідомлень та спільне хмарне сховище. На його основі реалізовано діючий програмний прототип, що включає серверний API-сервіс, сервіс-воркер із конвеєрами транскрибування та рендерингу на базі FFmpeg і OpenAI API, хмарну інфраструктуру GCP під управлінням Terraform та CI/CD конвеєр на базі GitHub Actions.

Проведено комплексне тестування за чотирма напрямами. Функціональне тестування підтвердило коректність реалізації усіх визначених користувацьких сценаріїв. Тестування механізмів надійності підтвердило коректність роботи ідемпотентності обробки та черги «мертвих повідомлень», що забезпечує коректне функціонування системи під навантаженням. Оцінювання якості розпізнавання україномовного контенту продемонструвало показник WER на рівні 4,2 % для студійного аудіо та 7,4–9,8 % для ускладнених умов запису; застосування системної підказки стабільно знижує WER на 4,1–4,9 в.п.

Тестування продуктивності системи показало сприятливі результати: для контенту тривалістю до двох хвилин загальний час обробки не перевищує 35 секунд.

Практичне значення одержаних результатів полягає у можливості використання розробленого рішення розробниками програмного забезпечення, як документованого шаблону для побудови систем обробки медіаконтенту. Також система може бути використана SMM-спеціалістами, контент-мейкерами та маркетинговими агентствами для автоматизації субтитрування україномовного відеоконтенту з метою подальшої публікації на платформи TikTok, Instagram Reels та YouTube Shorts.

Перспективними напрямами подальшого розвитку є самостійне розгортання моделі Whisper на інфраструктурі з підтримкою GPU для збільшення продуктивності та можливостей масштабування системи, підтримка формату субтитрів ASS для розширення можливостей стилізації, а також застосування великих мовних моделей у пост-обробці результатів для підвищення якості кінцевих субтитрів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Robust Speech Recognition via Large-Scale Weak Supervision / Radford A. et al. Proceedings of the 40th International Conference on Machine Learning (ICML). 2023. Vol. 202. P. 28492–28518.

2. Morris K. Infrastructure as Code: Dynamic Systems for the Cloud Age. 2nd ed. Sebastopol : O'Reilly Media, 2020. 432 p.

3. Evolutionary trends in automatic speech recognition with artificial intelligence: a systematic literature review / Gabriel S. et al. ResearchGate. 2026. URL: https://www.researchgate.net/publication/401017444_Evolutionary_trends_in_automatic_speech_recognition_with_artificial_intelligence_a_systematic_literature_review (дата звернення: 05.03.2026).

4. Parween S. Tracing the evolution of speech recognition through machine learning. Medium. 2024. URL: https://medium.com/@shaziaparween333/tracing-the-evolution-of-speech-recognition-through-machine-learning-8632e92b4f66 (дата звернення: 05.03.2026).

5. End-to-End Speech Recognition: A Survey / Prabhavalkar R. et al. arXiv. 2023. URL: https://arxiv.org/pdf/2303.03329 (дата звернення: 05.03.2026).

6. Evolution of Transformers in Speech Recognition / Adamjee S. et al. ResearchGate. 2025. URL: https://www.researchgate.net/publication/398763356_Evolution_of_Transformers_in_Speech_Recognition (дата звернення: 05.03.2026).

7. Наход О. Автоматичне розпізнавання українського мовлення на основі глибокого навчання. ResearchGate. 2025. URL: https://www.researchgate.net/publication/389218685_AVTOMATICNE_ROZPIZNAVANNA_UKRAINSKOGO_MOVLENNA_NA_OSNOVI_GLIBOKOGO_NAVCANNA (дата звернення: 08.03.2026).

8. Julia E. 80% of People Prefer Video Subtitles. Here's How they Affect Engagement. Kapwing. 2024. URL: https://www.kapwing.com/resources/subtitle-statistics/ (дата звернення: 08.03.2026).

9. 11 Subtitle Generation Trends: Key Statistics Every Content Creator Should Know in 2026. Sonix. 2026. URL: https://sonix.ai/resources/subtitle-generation-trends (дата звернення: 08.03.2026).

10. Why Video Captions Are Essential for SEO and Engagement in 2025. CaptionCut. 2025. URL: https://www.captioncut.com/blog/video-captions-seo-engagement-2025 (дата звернення: 08.03.2026).

11. AI-Powered Subtitle Generator Market Size, Share, Growth, and Industry Analysis. market.us. 2026. URL: https://market.us/report/ai-powered-subtitle-generator-market (дата звернення: 09.03.2026).

12. Compare Kapwing vs. Submagic vs. VEED in 2026. Slashdot. URL: https://slashdot.org/software/comparison/Kapwing-vs-SubMagic-vs-VEED (дата звернення: 09.03.2026).

13. Introducing Whisper. OpenAI. 2022. URL: https://openai.com/index/whisper/ (дата звернення: 09.03.2026).

14. FFmpeg Documentation. FFmpeg. URL: https://ffmpeg.org/documentation.html (дата звернення: 13.03.2026).

15. pysubs2 Documentation. pysubs2. URL: https://pysubs2.readthedocs.io/en/latest/ (дата звернення: 13.03.2026).

16. Google USM: Scaling Automatic Speech Recognition Beyond 100 Languages / Zhang Y. et al. arXiv. 2023. URL: https://arxiv.org/pdf/2303.01037 (дата звернення: 13.03.2026).

17. Universal-2-TF. AssemblyAI. URL: https://www.assemblyai.com/research/universal-2 (дата звернення: 13.03.2026).

18. Speech Recognition & Synthesis for Ukrainian. GitHub. URL: https://github.com/egorsmkv/speech-recognition-uk (дата звернення: 13.03.2026).

19. Cloud Speech-to-Text V2 supported languages. Google Cloud. URL: https://cloud.google.com/speech-to-text/docs/speech-to-text-supported-languages (дата звернення: 13.03.2026).

20. Amazon Transcribe now supports streaming transcription in 30 additional languages. Amazon Web Services. 2024. URL: https://aws.amazon.com/about-

aws/whats-new/2024/10/amazon-transcribe-streaming-transcription-additional-languages (дата звернення: 14.03.2026).

21. Supported Languages & Features. AssemblyAI. URL: https://www.assemblyai.com/docs/getting-started/supported-languages (дата звернення: 14.03.2026).

22. Andress M. Subtitle Formats Explained: SRT vs. VTT vs. TXT vs. ASS. subvideo.ai blog. 2025. URL: https://subvideo.ai/blog/subtitle-formats-srt-vtt-txt-ass/ (дата звернення: 16.03.2026).

23. WebVTT: The Web Video Text Tracks Format : W3C Candidate Recommendation, 4 April 2019. W3C. URL: https://www.w3.org/TR/webvtt1/ (дата звернення: 16.03.2026).

24. Automated Subtitle Embedding Using FFmpeg. Cincopa. URL: https://www.cincopa.com/learn/automated-subtitle-embedding-using-ffmpeg (дата звернення: 16.03.2026).

25. Create transcription. OpenAI Developers. URL: https://platform.openai.com/docs/api-reference/audio/createTranscription (дата звернення: 18.03.2026).

26. Lambda quotas. Amazon Web Services. URL: https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html (дата звернення: 20.03.2026).

27. Monoliths vs Microservices vs Serverless. Harness. 2024. URL: https://www.harness.io/blog/monoliths-vs-microservices-vs-serverless (дата звернення: 20.03.2026).

28. What is Cloud Run. Google Cloud. URL: https://docs.cloud.google.com/run/docs/overview/what-is-cloud-run (дата звернення: 20.03.2026).

29. Asynchronous Request-Reply pattern. Microsoft Learn. URL: https://learn.microsoft.com/en-us/azure/architecture/patterns/async-request-reply (дата звернення: 20.03.2026).

30. What is Pub/Sub? Google Cloud. URL: https://cloud.google.com/pubsub/docs/overview (дата звернення: 20.03.2026).

31. Dead-letter topics. Google Cloud. URL: https://cloud.google.com/pubsub/docs/handling-failures (дата звернення: 20.03.2026).

32. Claim-Check pattern. Microsoft Learn. URL: https://learn.microsoft.com/en-us/azure/architecture/patterns/claim-check (дата звернення: 20.03.2026).

33. Free Tier products. Google Cloud. URL: https://cloud.google.com/free (дата звернення: 22.03.2026).

34. AWS Free Tier. Amazon Web Services. URL: https://aws.amazon.com/free/ (дата звернення: 22.03.2026).

35. Explore free Azure services. Microsoft Azure. URL: https://azure.microsoft.com/en-us/pricing/free-services (дата звернення: 22.03.2026).

36. Learn how to create and use Amazon ECS resources. Amazon Web Services. URL: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/getting-started.html (дата звернення: 22.03.2026).

37. Azure Container Apps overview. Microsoft Learn. URL: https://learn.microsoft.com/en-us/azure/container-apps/overview (дата звернення: 22.03.2026).

38. Use Pub/Sub with Cloud Run tutorial. Google Cloud. URL: https://cloud.google.com/run/docs/tutorials/pubsub (дата звернення: 22.03.2026).

39. What is Amazon SNS? Amazon Web Services. URL: https://docs.aws.amazon.com/sns/latest/dg/sns-common-scenarios.html (дата звернення: 22.03.2026).

40. What is Azure Service Bus? Microsoft Learn. URL: https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-messaging-overview (дата звернення: 22.03.2026).

41. Cloud SQL pricing. Google Cloud. URL: https://cloud.google.com/sql/pricing#postgresql (дата звернення: 22.03.2026).

42. Amazon RDS for PostgreSQL pricing. Amazon Web Services. URL: https://aws.amazon.com/rds/postgresql/pricing/ (дата звернення: 22.03.2026).

43. Scale to Zero. Neon. URL: https://neon.tech/docs/introduction/scale-to-zero (дата звернення: 22.03.2026).

44. What is Infrastructure as Code with Terraform? HashiCorp. URL: https://developer.hashicorp.com/terraform/tutorials/aws-get-started/infrastructure-as-code (дата звернення: 24.03.2026).

45. Introduction. Turborepo. URL: https://turborepo.dev/docs (дата звернення: 24.03.2026).

46. The TypeScript Handbook. TypeScript. URL: https://www.typescriptlang.org/docs/handbook/intro.html (дата звернення: 24.03.2026).

47. Layouts and Pages. Next.js. URL: https://nextjs.org/docs/app/building-your-application/routing (дата звернення: 24.03.2026).

48. Validation and Serialization. Fastify. URL: https://fastify.dev/docs/latest/Reference/Validation-and-Serialization/ (дата звернення: 24.03.2026).

49. About. PostgreSQL. URL: https://www.postgresql.org/about/ (дата звернення: 24.03.2026).

50. Getting Started. TypeORM. URL: https://typeorm.io/docs/getting-started (дата звернення: 24.03.2026).

51. GitHub Actions documentation. GitHub. URL: https://docs.github.com/en/actions (дата звернення: 25.03.2026).

52. turbo-ignore. Turborepo. URL: https://turborepo.dev/docs/reference/turbo-ignore (дата звернення: 25.03.2026).

53. Hosting a static website using Cloud Storage. Google Cloud. URL: https://cloud.google.com/storage/docs/hosting-static-website (дата звернення: 25.03.2026).

54. Artifact Registry overview. Google Cloud. URL: https://docs.cloud.google.com/artifact-registry/docs/overview (дата звернення: 25.03.2026).

55. Ensure Safe Deployments by Granting Only Necessary Access to Cloud Run Resources. Google Cloud Blog. URL: https://cloud.google.com/blog/products/identity-security/securing-cloud-run-deployments-with-least-privilege-access (дата звернення: 25.03.2026).

56. Arias D. Hashing in Action: Understanding bcrypt. auth0. 2021. URL: https://auth0.com/blog/hashing-in-action-understanding-bcrypt/ (дата звернення: 10.04.2026).