Міністерство освіти і науки україни

Національний університет «києво-могилянська академія»

Кафедра інформатики факультету інформатики

Курсова робота

З дисципліни: інженерія програмного забезпечення

На тему:

«Додаток з використанням технології Nest.js та імплементацією функцій штучного інтелекту»

Керівник курсової роботи
___Борозенний с. О.___
(прізвище та ініціали)
_____
(підпис)
«_____» _____
2025 р.

Виконав студент _____
__Харишин І. М._____

(прізвище та ініціали)

«_____» _____ 2025 р.

Київ – 2025

Міністерство освіти і науки України
Національний університет «Києво-Могилянська академія»
Кафедра мультимедійних систем факультету інформатики

Затверджую
Зав. Кафедри інформатики
Доцент, к. Ф-м. Н.
Гороховський с.с.
«_____» _____
2024 р.

Індивідуальне завдання
На курсову роботу

Студенту Харишину Ігорю Михайловичу факультету інформатики 3 курсу
Тема: додаток з використанням технології nest.js та імплементацією функцій штучного інтелекту
Зміст тч до курсової роботи:

індивідуальне завдання
календарний план
анотація
вступ
огляд можливих рішень
структурна розробка веб-застосунку
розробка власного рішення
висновки
використані джерела

Дата видачі
«____» _____ 2024 р.
Керівник
_____
(прізвище та ініціали)
_____
(підпис)
Завдання отримав
_____
(прізвище та ініціали)
_____
(підпис)

Тема: «Додаток з використанням технології Nest.js та імплементацією функцій штучного інтелекту»

Календарний план виконання роботи:

№	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання теми курсової роботи	14.12.2024	
2.	Огляд літератури за темою роботи	21.12.2024	
3.	Вивчення предметної області	20.02.2025	
4.	Визначення інструментів необхідних для імплементації	26.02.2025	
5.	Аналіз проблеми та пошук рішення	10.03.2025	
6.	Виконання технічного завдання	24.04.2025	
7.	Написання теоретичної частини курсової роботи	17.05.2025	
8.	Захист роботи		

Студент _____

Керівник _____

«_____» _____ 20__ р.

Зміст

Анотація

У курсовій роботі розглянуто процес створення веб-застосунку з використанням фреймворку nestjs, який реалізує функціонал автоматичного створення субтитрів на основі аудіо- або відеофайлів. Основний акцент зроблено на інтеграції сучасних технологій штучного інтелекту, зокрема мовних моделей від google (gemini) та сервісу розпізнавання мовлення google speech-to-text.

Запропоноване рішення дозволяє користувачеві завантажити мультимедійний файл, згенерувати транскрипцію, редагувати текст, визначати ключові моменти, знайти найкращі уривки за ключовими словами, знайти найвідповідніші введеним фразам уривки та автоматично створювати субтитри у форматі .vtt або .srt. У роботі наведено аналіз існуючих рішень, обґрунтовано вибір технологій, детально описано структуру системи та реалізовано приклад інтеграції нейромереж у сучасний веб-застосунок.

Результати роботи демонструють потенціал поєднання інструментів веб-розробки та штучного інтелекту для створення інноваційних рішень у сфері обробки мультимедійних даних.

Вступ

Останніми роками нейронні мережі та моделі штучного інтелекту (ші) стали однією з найдинамічніших і найвпливовіших технологій, які активно впроваджуються в різні сфери людської діяльності. Від автоматичного перекладу та медичної діагностики до обробки зображень і створення музики — штучний інтелект демонструє здатність істотно покращувати ефективність, точність і зручність повсякденних процесів. Однією з особливо важливих властивостей таких систем є їхня здатність до аналізу великих обсягів неструктурованих даних — текстів, аудіо, відео — що відкриває нові можливості для бізнесу, науки, освіти та комунікації.

У сучасному цифровому середовищі мультимедійні дані, такі як відео- та аудіофайли, відіграють дедалі важливішу роль. Зокрема, відеоконтент став основною формою комунікації в інтернеті — від онлайн-лекцій і подкастів до презентацій, соціальних мереж і технічної документації. Проте обробка такого контенту часто ускладнена через його неструктуровану природу: для пошуку, архівації або перекладу потрібен текстовий еквівалент. Саме тут на перший план виходять технології автоматичного розпізнавання мовлення, транскрипції, а також генерації субтитрів і ключових моментів із використанням ші.

Особливої актуальності набуває інтеграція нейронних мереж у веб-застосунки. Поєднання класичних серверних архітектур із сучасними моделями, такими як великі мовні моделі (llm) чи сервіси на основі глибокого навчання, дозволяє створювати «розумні» системи, що забезпечують високий рівень автоматизації, зручності для користувача та функціональної гнучкості. Такі застосунки не лише трансформують досвід взаємодії з інформацією, а й створюють нові стандарти в розробці пз.

У цій курсовій роботі розглядається створення веб-застосунку на основі фреймворку nestjs, який дозволяє користувачеві завантажити аудіо- або відеофайл, автоматично отримати з нього текстову транскрипцію за допомогою сервісу google cloud speech-to-text, відредагувати цей текст у зручному інтерфейсі, згенерувати субтитри та визначити найцікавіші або ключові моменти за допомогою мовної моделі gemini. Такий підхід поєднує сучасні розробницькі практики з потужністю штучного інтелекту, демонструючи приклад реального використання нейромереж для обробки мультимедійного контенту.

Метою цієї роботи є не лише реалізація технічно функціонального застосунку, а й дослідження можливостей інтеграції штучного інтелекту в повсякденні інструменти веб-розробки. У звіті проєкту подано огляд сучасних рішень на ринку, проаналізовано обрані технології, детально описано архітектуру застосунку та обґрунтовано вибір методів обробки даних. Результати демонструють практичне втілення актуальних тенденцій у галузі ші та веб-інженерії.

Розділ 1. Огляд можливих рішень

1.1 проблематика роботи з мультимедійним контентом

У сучасному цифровому середовищі аудіо- та відеоматеріали набувають усе більшої популярності, проте вони залишаються складними для автоматичної обробки. Основна проблема полягає в тому, що такий контент є неструктурованим, а його зміст прихований у звукових доріжках. Це унеможливлює ефективний пошук, аналіз, класифікацію та повторне використання інформації без попереднього перетворення мовлення в текст.

Автоматичне розпізнавання мовлення (asr — automatic speech recognition) стало ключовим етапом в обробці мультимедійного контенту. Отриманий транскрибований текст дозволяє застосовувати методи обробки природної мови (nlp), виконувати семантичний аналіз, створювати субтитри, формувати короткий зміст, здійснювати навігацію по контенту та забезпечувати доступність для користувачів з порушеннями слуху.

1.2 сервіси для розпізнавання мови

Найпопулярніші платформи для розпізнавання мови включають:

Google cloud speech-to-text — потужний сервіс з підтримкою численних мов і режимів реального часу;

Whisper (від openai) — open-source модель, що дозволяє обробляти аудіо локально;

Ibm watson speech to text, azure speech service, amazon transcribe — комерційні рішення з api-доступом.

Сервіс	Тип доступу	Підтримка мов	Переваги	Недоліки
Google cloud speech-to-text	Хмарний (api)	>100	Висока точність, реальний час, таймкоди	Платний, хмарна залежність
Whisper (openai)	Локальний / open-source	>90	Безкоштовний, локальне використання	Високі вимоги до ресурсів
Amazon transcribe	Хмарний (api)	>100	Інтеграція з aws, точність	Вартість, залежність від aws
Azure speech service	Хмарний (api)	>90	Інтеграція з microsoft сервісами	Потрібна складна конфігурація
Ibm watson stt	Хмарний (api)	~50	Гнучка настройка	Обмежена підтримка мов

У межах даного проєкту було обрано google speech-to-text api як оптимальне рішення за співвідношенням ціни(3 місяці безкоштовного доступу), точності, швидкості, інтеграції з іншими api, що були використані в проєкті та підтримки української мови.

1.3 сервіси та моделі для аналізу тексту

Після отримання транскрипції наступним кроком є її аналіз. Для цього використовуються моделі обробки природної мови, які здатні розпізнавати ключові теми, створювати резюме, класифікувати зміст за категоріями тощо. Ці задачі є частиною семантичного аналізу.

Модель	Розробник	Можливості	Формат доступу	Особливості
Gemini 2.0 flash	Google deepmind	Семантичний аналіз, стислий зміст	Api (хмара)	Висока швидкість, мультимодальність
Chatgpt (gpt-4)	Openai	Узагальнення, q&a, генерація	Api / локально	Підтримка розширених інструкцій
Claude	Anthropic	Когерентність, контекстна відповідність	Api	Фокус на безпеці відповідей
Bert	Google	Класифікація, витяг фактів	Локально / huggingface	Найбільш досліджувана модель

До основних проблем роботи з подібними моделями належать: обмеження обсягу контексту, потреба в обчислювальних ресурсах, ризики генерації некоректного змісту, складність адаптації до специфічної термінології.

Для проєкту я обрав gemini 2.0 flash як основну модель для семантичного аналізу транскриптів. Цей вибір зумовлений її високою продуктивністю, підтримкою швидкого інференсу в хмарі та можливістю ефективно обробляти великі обсяги тексту з дотриманням контексту.

1.4 створення субтитрів

Субтитри є важливим елементом для підвищення доступності та зручності перегляду відеоконтенту. Процес створення субтитрів включає кілька етапів:

Отримання транскрипції мовлення з таймкодами.

Формування блоків субтитрів відповідно до часових меж і розміру тексту.

Експорт у формати srt або webvtt — популярні стандарти, які підтримуються відеоплеєрами та платформами.

Інтеграція субтитрів у відео або окремий файл.

Для реалізації цих етапів використовуються такі інструменти:

Ffmpeg — утиліта для обробки аудіо- та відеофайлів, включаючи вбудовування субтитрів.

Subtitle.js, autosub, aeneas — бібліотеки для автоматизації процесу створення та синхронізації субтитрів.

Webvtt, srt, ass — формати з підтримкою часових маркерів і текстових стилів.

1.5 виклики при інтеграції моделей у реальні системи

Попри високу ефективність сучасних мовних моделей і asr-сервісів, їх впровадження в реальні веб-застосунки супроводжується низкою викликів:

- Безпека та конфіденційність — передача даних у хмару вимагає захищеного каналу та гарантій приватності.
- Швидкодія — обробка великих медіафайлів у режимі реального часу потребує оптимізації, черг обробки та паралельного виконання.
- Вартість — використання комерційних api при масштабуванні може стати фінансово затратним.
- Якість результатів — моделі не завжди дають точний або релевантний зміст, особливо в умовах шумного аудіо або спеціалізованої лексики.

- Технічна стабільність — оновлення api, зміни версій моделей, технічні обмеження можуть потребувати частих адаптацій у коді застосунку.

Розділ 2. Структурна розробка веб-застосунку

2.1 загальна архітектура

Застосунок побудовано за архітектурою клієнт-сервер з використанням серверного фреймворку nestjs, який базується на node.js та дозволяє організувати код за принципами модульності та залежностей (dependency injection).

Чому обрано nestjs:

- Модульна архітектура. Nestjs побудований за принципом модульності, що дозволяє чітко структурувати великі проєкти. Це спрощує підтримку, масштабування і повторне використання коду.
- Підтримка typescript "із коробки". Nestjs повністю написаний на typescript, що дає сильну типізацію, знижує кількість помилок і полегшує автодоповнення в ide.
- Інтеграція з сучасними інструментами. Має вбудовану підтримку для популярних бібліотек — наприклад, express.
- Вбудований механізм di дозволяє зручно керувати залежностями, підвищуючи тестованість і розширюваність.

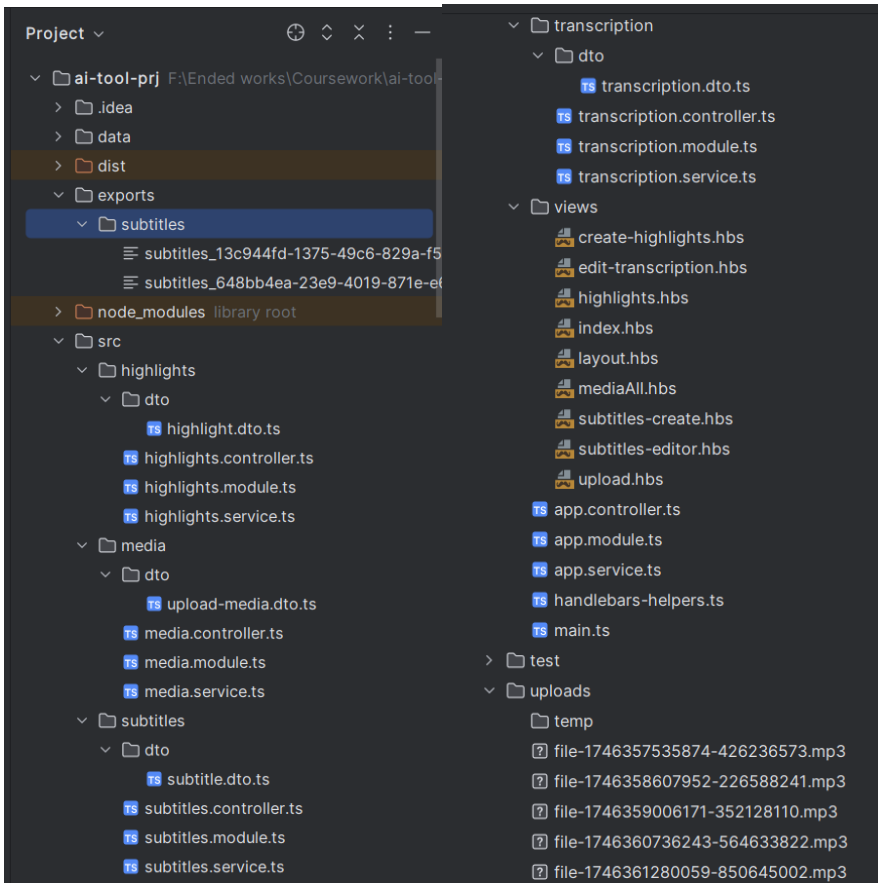
Основні компоненти:

- Frontend (hbs + html/css/js) — відповідає за інтерфейс, завантаження файлів, редагування тексту.
- Backend (nestjs) — обробляє запити, взаємодіє з хмарними сервісами, формує субтитри.
- Google cloud api — розпізнавання мовлення.
- Gemini api — аналіз тексту і виділення ключових моментів.
- Система збереження файлів — тимчасове збереження медіа та результатів (локальна або хмарна, наприклад, google cloud storage чи просто файловий диск).

2.2 використані технології

Компонент	**Технологія**
Сервер	Nestjs
Мова програмування	Typescript
Шаблонізатор	Handlebars (hbs)
Субтитри	Srt формат
Хмарний api	Google speech-to-text, gemini, google cloud
Медіаобробка	Ffmpeg
Стилізація	Bootstrap / кастомна css

2.3 структура проєкту



2.4 handlebars (hbs)

1. Простота інтеграції з nestjs

Handlebars (hbs) є одним з офіційно підтримуваних шаблонізаторів у фреймворку nestjs. Його легко інтегрувати за допомогою @nestjs/platform-express, що значно спрощує налаштування серверного рендерингу сторінок.

2. Зрозумілий синтаксис шаблонів

Hbs має просту, читабельну і декларативну структуру, яка дозволяє швидко створювати html-сторінки. Завдяки цій перевазі розробники можуть легко підтримувати та розширювати шаблони без необхідності вивчення складного синтаксису.

3. Підтримка умов, циклів та вкладених елементів

Handlebars дозволяє працювати з масивами даних, створювати умовні блоки, вкладення та повторювані елементи. Це дозволяє створити динамічні сторінки, як-от медіа-галерею, де відображається список файлів з інформацією про них.

4. Придатність для серверного рендерингу (ssr)

Проєкт не є spa-додатком і не потребує клієнтського javascript-фреймворку. Враховуючи цілі проєкту (завантаження, відображення списку медіафайлів, форма завантаження), hbs є доцільним вибором для швидкого серверного рендерингу.

5. Безпечне відображення даних

Handlebars автоматично екранує вивід, що мінімізує ризик атак типу xss. Це особливо важливо при виведенні імен файлів або інших даних, які можуть містити шкідливий вміст.

6. Зручність у розробці та налагодженні

Hbs дозволяє швидко створювати шаблони, які легко тестувати і змінювати. У разі помилок шаблонізатор надає зрозумілі повідомлення, що полегшує відлагодження.

Основні альтернативи такі як react та vue (з використанням ssr або ssg)

Переваги:

- Потужні javascript-фреймворки з великими екосистемами.
- Добре підходять для складних spa (single page application).
- Можливість використання компонентного підходу.

Чому не обрано:

- Надлишкова складність для завдань ssr: проєкт має прості потреби — відображення списку медіафайлів, форма завантаження — і не потребує spa-архітектури або складного клієнтського інтерфейсу.
- Потреба у додаткових інструментах: для використання ssr з react/vue потрібно налаштовувати webpack, babel, або використовувати фреймворки типу next.js/nuxt.js. Це значно ускладнює структуру проєкту.
- Вища вартість розробки та супроводу: потрібно більше часу на налаштування та більше залежностей, що необґрунтовано для малого або середнього серверного додатку.

2.5 опис модуля генерації субтитрів

Модуль субтитрів відповідає за створення, редагування, імпорт, експорт та видалення субтитрів, що прив’язані до медіафайлів. Він надає api та веб-інтерфейс для роботи із субтитрами у форматах .srt та .vtt.

Основні можливості модуля:

1. Генерація субтитрів з транскрипції
   автоматично створює субтитри на основі вже наявної транскрипції аудіо чи відео. Визначає часові рамки для кожної репліки.

2. Імпорт субтитрів із файлів (srt/vtt)
   дозволяє завантажити субтитри з локального файлу та перетворити їх у внутрішній формат, придатний для редагування.
3. Експорт у формати .srt та .vtt
   можливість завантажити субтитри у стандартному форматі для подальшого використання з відео.
4. Редагування субтитрів у браузері
   через інтегрований редактор можна змінювати текст та часові мітки кожного елементу субтитрів.
5. Збереження даних у файловій системі
   всі субтитри зберігаються у вигляді json-файлів у директорії data/subtitles.

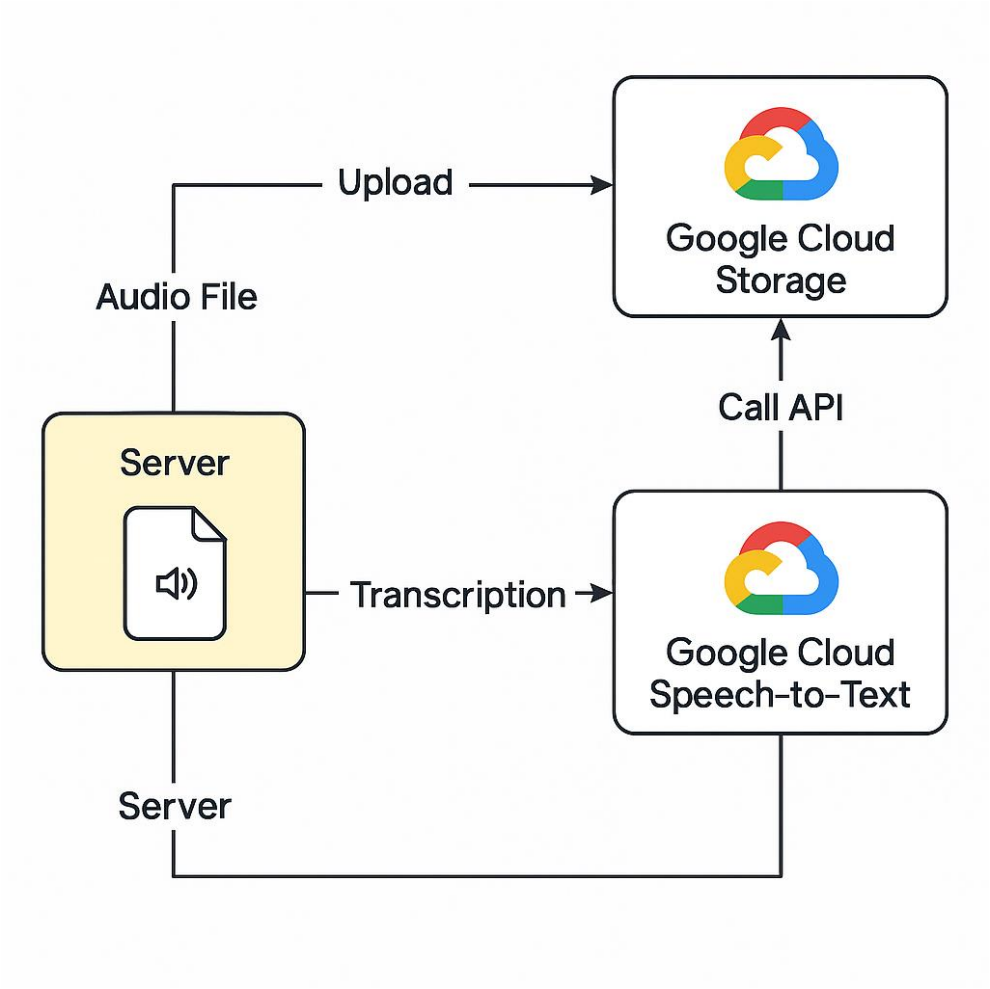


Рисунок 1. Робота модуля генерації субтитрів

Архітектура модуля

- Subtitlescontroller — rest-контролер, який обробляє http-запити для створення, редагування, імпорту/експорту субтитрів.
- Subtitlesservice — основна логіка роботи з субтитрами: зчитування, запис, форматування, конвертація.
- Dtos — використовуються для валідації та передачі даних (наприклад, createsubtitledto, updatesubtitledto).

Взаємодія з іншими модулями:

- Transcriptionservice — використовується для створення субтитрів на основі існуючої транскрипції.
- Mediaservice — забезпечує зв'язок субтитрів із відповідним медіа.

Формати, які підтримуються

- .srt — класичний формат субтитрів з часовими мітками у форматі hh:mm:ss,mmm.
- .vtt — формат для html5 video плеєрів, підтримує hh:mm:ss.mmm.
- (частково передбачена підтримка .ass, але вона поки не реалізована у subtitlesservice)

Алгоритми та реалізація

- Парсинг srt/vtt:
    - Визначення блоків субтитрів за порожнім рядком.
    - Розбір часових міток через регулярні вирази.
    - Обчислення мілісекунд для точного позиціонування.
- Генерація файлів:
    - Метод generatesrt() формує рядки для .srt.
    - Метод generatevtt() аналогічно формує текст для .vtt.
    - Файл зберігається у директорії exports/subtitles.
- Збереження:

  Json-структура зберігає id, мову, час створення, список entries.

  ```
  {
   "id": "d2314dfe-abc1-44f8-9e1f-xxxxx",
   "mediaid": "media123",
   "language": "uk",
   "entries": [
    {
     "starttime": 0,
     "endtime": 2000,
     "text": "це приклад."
  ```

```
    }
  ],
  "createdat": "2025-05-05t12:00:00.000z",
  "updatedat": "2025-05-05t12:00:00.000z"
}
```


2.6 інтерфейс користувача: шаблон сторінки генерації субтитрів

- /subtitles/create — форма для створення субтитрів.
- /subtitles/editor/:mediaid — редактор субтитрів з таймлайном.

Імпорт — можливість завантажити .srt або .vtt файл та автоматично обробити його.

Експорт — кнопка для завантаження субтитрів у потрібному форматі.

2.7 завантаження файлів (file-upload)

Призначення модуля

Модуль media призначений для обробки користувацьких медіафайлів — аудіо та відео.

Основні функції включають:

- Завантаження медіа;
- Збереження інформації про файли (метадані);
- Відображення списку медіафайлів;
- Перегляд окремих медіа за id;
- Видалення файлів з файлової системи та пам'яті.

Основні компоненти

1. Mediacontroller

- Керує http-запитами, пов'язаними з медіа:
- Get /media/:id — повертає метадані конкретного медіа;

- Get /media — повертає або список медіа, або відображає сторінку галереї;
- Get /media/upload — сторінка завантаження;
- Post /media/upload — завантаження файлу з перевіркою типу (лише audio або video).

2. Mediaservice

- Здійснює логіку роботи з файлами:
- Генерує uuid для медіа;
- Обчислює тривалість (за допомогою get-audio-duration);
- Зберігає метадані у пам'яті (map);
- Видаляє файли з файлової системи.

Формат збереження

Медіафайли зберігаються у локальній папці ./uploads, а метадані — в оперативній пам’яті (тобто при перезавантаженні серверу втрачаються).

Збережена інформація:

Id: унікальний ідентифікатор (uuid);

Originalname: ім’я файлу користувача;

Filename: згенероване ім’я для зберігання;

Path: шлях до файлу;

Size: розмір;

Mimetype: mime-тип;

Duration: тривалість (для аудіо);

Createdat: дата створення.

Технічні деталі

Модуль використовує multer для обробки файлів.

Типи файлів обмежені до audio/* або video/*.

Назви файлів генеруються динамічно для унікальності.

Тривалість аудіо визначається тільки для аудіофайлів (якщо можливо).

У випадку помилки при зчитуванні тривалості, тривалість буде null.

2.8 опис модуля transcription

Опис модуля transcription

Модуль transcription реалізує повний цикл обробки аудіофайлів для перетворення мови у текст. Він працює з локальними файлами або медіа, які завантажуються раніше, і використовує сервіс google cloud speech-to-text.

Основні можливості:

Підтримка різних форматів аудіо:

Підтримуються .wav, .mp3, .flac, .ogg, .webm, .mp4. Якщо аудіо вкладено у відео .mp4, перед транскрипцією здійснюється його конвертація у .wav (через ffmpeg).

Визначення тривалості файлу:

Для файлів тривалістю до 60 секунд використовується синхронне розпізнавання (api recognize()), для довших — асинхронне (api longrunningrecognize() з попереднім завантаженням у google cloud storage).

Обробка результату:

Відповідь обробляється для створення сегментів (із таймкодами та текстом), а також повного зведеного тексту.

Підтримка декількох мов:

Мова розпізнавання задається через dto. За замовчуванням використовується "uk-ua".

Збереження результатів:

Транскрипції зберігаються у пам'яті (map), їх можна отримати за id або mediaid, редагувати, або видалити.

Основні компоненти:

1. Transcriptionservice

Основна бізнес-логіка: обробка файлів, інтеграція з google api, парсинг результатів, керування локальним сховищем транскрипцій.

Додатково містить:

Завантаження у gcs (uploadtogcs)

Видалення з gcs (deletefromgcs)

Конвертація з mp4 у wav (extractaudio)

Визначення типу кодування (detectencoding)

2. Transcriptioncontroller

Rest api для керування транскрипціями:

Post /transcription — створення нової транскрипції

Get /transcription — список транскрипцій (можна фільтрувати за mediaid)

Get /transcription/:id — отримання транскрипції за id

Put /transcription/:id — редагування тексту або сегментів

Delete /transcription/:id — видалення транскрипції

Зовнішні залежності:

Google cloud speech-to-text — розпізнавання аудіо

Google cloud storage — хостинг аудіофайлів для довгих транскрипцій

Ffmpeg-static + fluent-ffmpeg — конвертація відео у аудіо

Ffprobe-static — визначення тривалості аудіо

Переваги реалізації:

Гнучкість — підтримка коротких і довгих записів

Безпечне розмежування логіки (dto, сервіс, контролер)

Автоматичне видалення тимчасових файлів з google cloud storage

Мінімальні вимоги до сторонньої конфігурації

2.9 опис модуля highlights

Модуль виділення найцікавіших моментів - це сервіс nestjs, розроблений для обробки транскрипцій та виокремлення значущих моментів за допомогою google vertex ai з моделями gemini. Модуль пропонує три основні функціональності:

- Генерація загальних найцікавіших моментів з транскрипцій
- Генерація моментів на основі конкретних ключових слів
- Виконання семантичного пошуку в межах транскрипцій

Технічна архітектура

Основні компоненти

- Highlightsservice: основний сервіс, що взаємодіє з google vertex ai
- Highlightscontroller: rest api кінцеві точки, що надають доступ до функціональності сервісу
- Інтеграція: працює з transcriptionservice для доступу до збережених даних транскрипцій

Інтеграція зі штучним інтелектом

- Основна модель ші: gemini-2.0-flash-001
- Резервна модель ші: gemini-1.0-pro
- Використовує платформу google cloud vertex ai

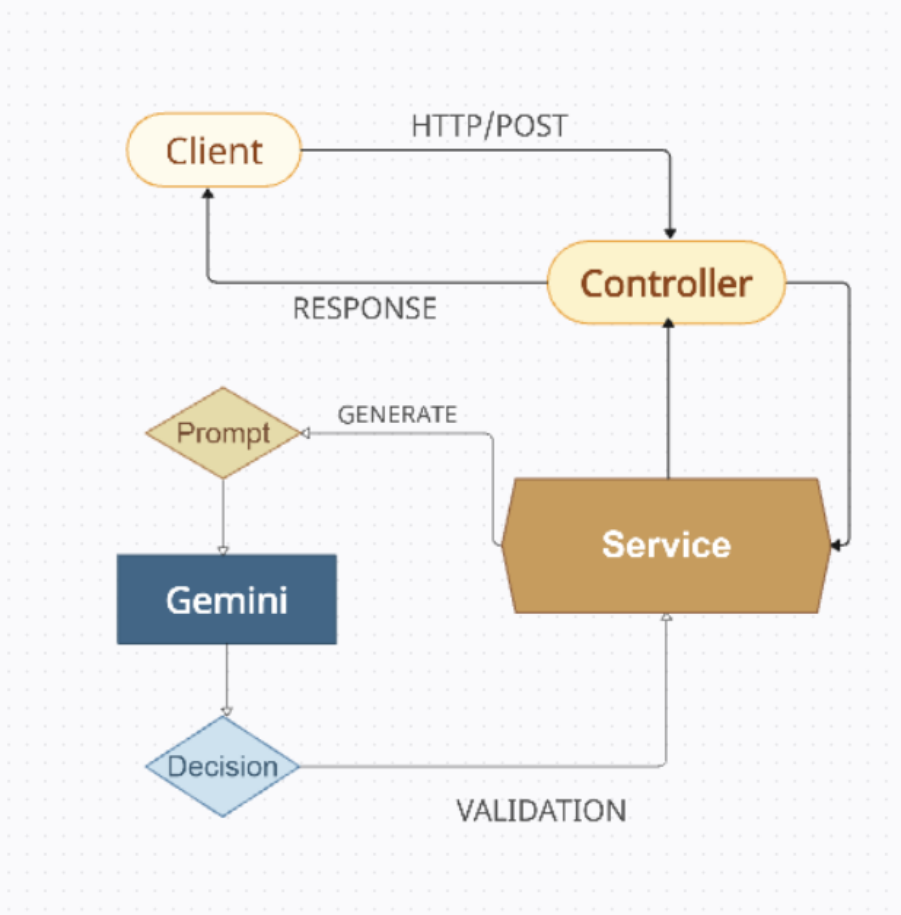


Рисунок 2. Архітектура сервісу highlights

Ключові функції

1. Генерація найцікавіших моментів

Виокремлює найцікавіші або найінформативніші сегменти з транскрипцій

Налаштовувані параметри:

Кількість моментів для генерації

Мінімальна тривалість (за замовчуванням: 10 секунд)

Максимальна тривалість (за замовчуванням: 60 секунд)

Повертає структуровані дані, включаючи заголовок, опис та часові мітки

2. Моменти на основі ключових слів

Знаходить сегменти, найбільш релевантні до конкретного ключового слова або фрази

Використовує ті самі параметри налаштування, що й загальні моменти

Оптимізовано для пошуку сильних семантичних відповідностей до наданого ключового слова

3. Семантичний пошук

Забезпечує запити природною мовою до транскрипцій

Розширені параметри:

Поріг схожості для збігів (за замовчуванням: 0.7)

Максимальна кількість результатів (за замовчуванням: 5)

Розмір контексту в секундах (за замовчуванням: 30)

Повертає результати, відсортовані за релевантністю

Технічна реалізація

Стратегія промптів

Використовує детальні, структуровані запити для керування моделлю gemini

Включає як повний текст транскрипції, так і сегментовані дані часової шкали

Забезпечує формат виводу json для стабільного парсингу

Обробка помилок і стійкість

Реалізує стратегію резервної моделі при збої основної моделі

Розширене логування помилок та контекстні повідомлення про помилки

Валідація вхідних даних на рівні контролера

Обробка результатів

Постобробка результатів, згенерованих ші, для забезпечення часового вирівнювання

Використовує декілька стратегій пошуку відповідностей:

Точне співпадіння фраз

Часткове співпадіння фраз

Пошук на основі ключових слів

Фільтрує результати на основі порогів якості та обмежень тривалості

Міркування щодо продуктивності

Використовує низьку температуру (0.2) для більш детермінованих результатів

Форматування відповідей оптимізовано для подальшої обробки

Управління контекстом для уникнення обмежень токенів

Арі-ендпоінти

- Get /highlights/create - відображає сторінку створення найцікавіших моментів
- Post /highlights/generate - генерує загальні найцікавіші моменти
- Post /highlights/generate-by-keyword - генерує моменти на основі ключових слів
- Post /highlights/semantic-search - виконує семантичний пошук

Формат введення/виведення

Вхідні дані: дані транскрипції, що складаються з текстових сегментів із часовими мітками

Вихідні дані: структуровані найцікавіші моменти з заголовками, описами та точними часовими посиланнями

Розділ 3. Розробка власного рішення

3.1 мета реалізації

Основна мета створеного застосунку — надати користувачу інструмент для автоматичної транскрипції відео чи аудіо, редагування тексту, виділення ключових моментів за допомогою нейромереж та створення субтитрів. Реалізація акцентована на використанні сучасних ii-рішень у повсякденних завданнях, таких як обробка відеоконтенту для youtube, подкастів чи онлайн-курсів.

3.2 інтеграція google speech-to-text для транскрипції

Першим етапом обробки є транскрипція — автоматичне перетворення мовлення у текст. Для цього було реалізовано обгортку над google cloud speech-to-text api, що дозволяє:

Обробляти файли до 1 години;

Працювати з українською мовою;

Отримувати таймінги для побудови субтитрів.

Після завантаження медіафайлу він зберігається тимчасово на сервері. Потім обробляється за допомогою сервісу:

```
Async generatesubtitles(file: express.multer.file): promise<string> {
  const transcript = await this.googleservice.transcribeaudio(file.path);
  return this.formattosrt(transcript);
}
```

Форматування субтитрів виконується через обчислення часу за ключовими фрагментами тексту, що надає api.

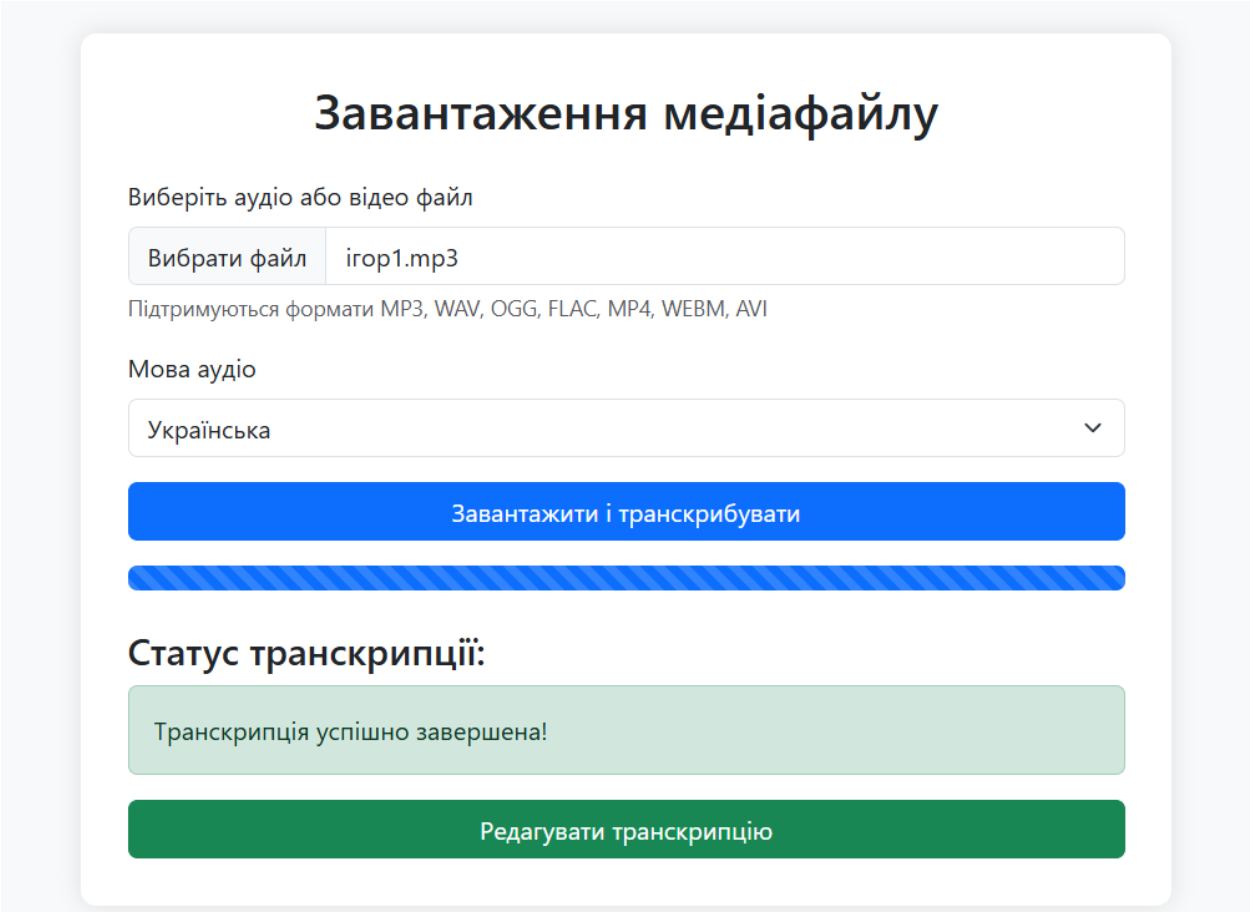


Рисунок 3. Завантаження файлу на сервер

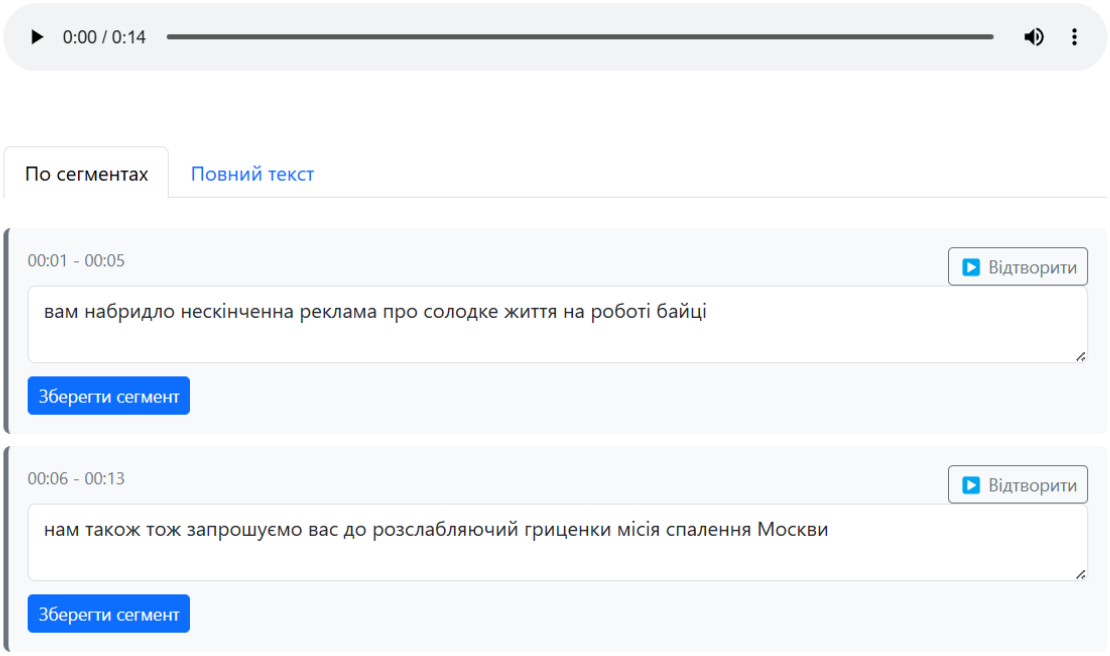


Рисунок 4. Редагування транскрипції

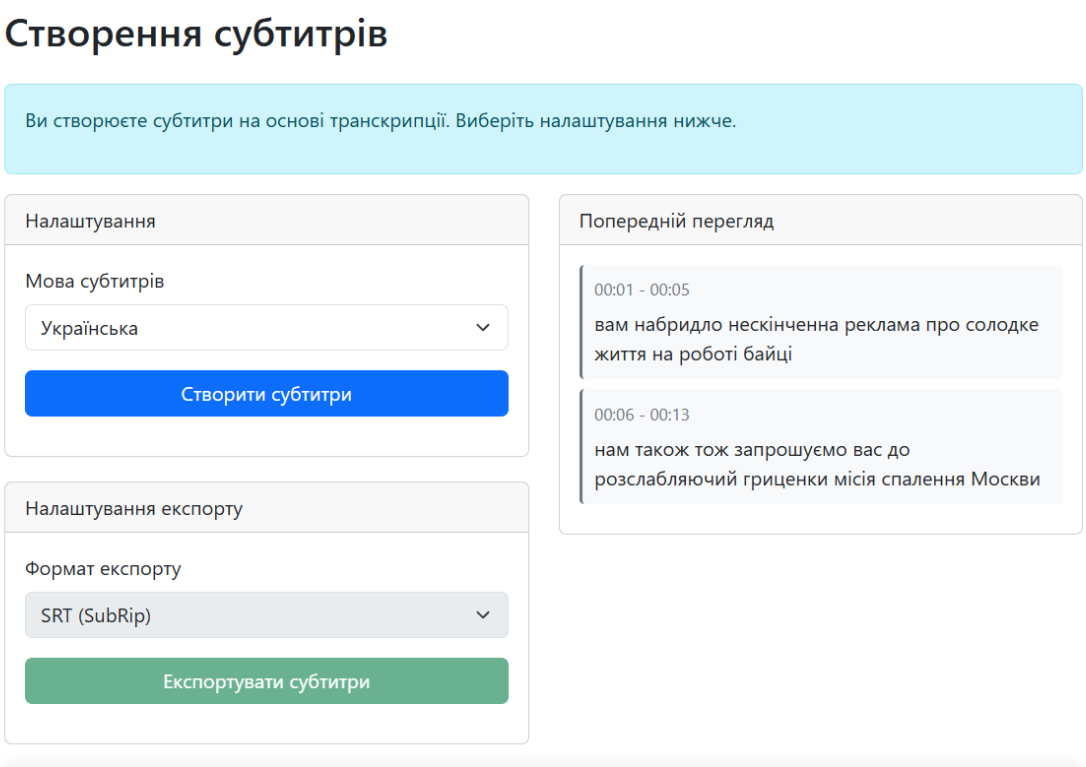


Рисунок 5. Створення субтитрів

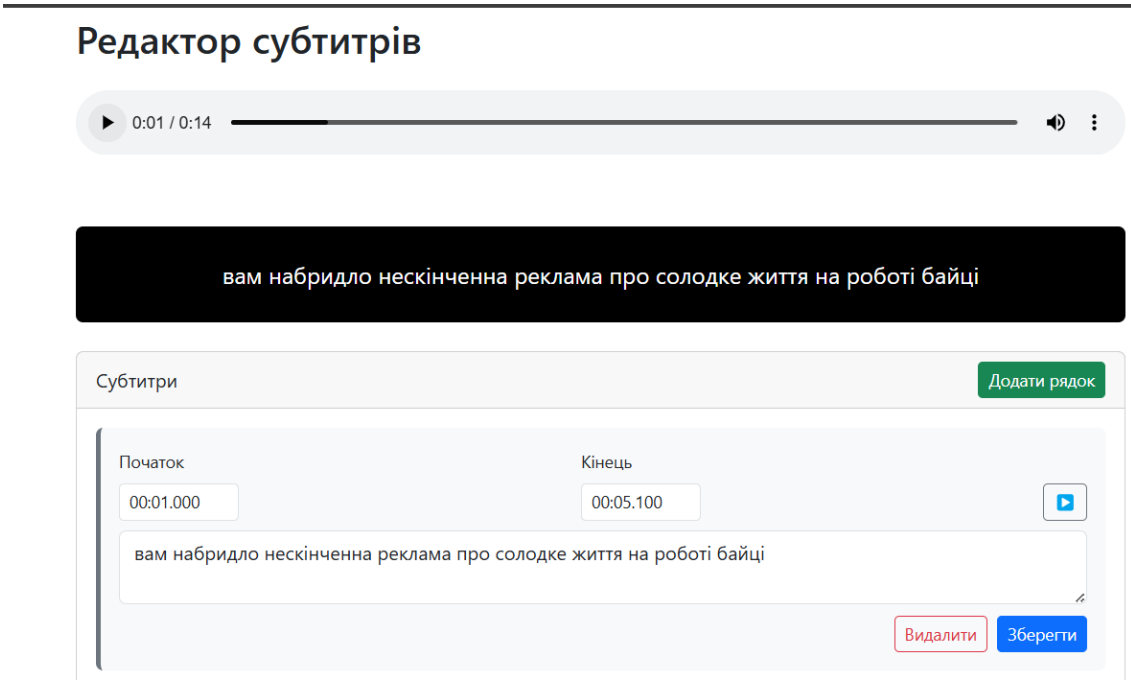


Рисунок 6. Редактор субтитрів

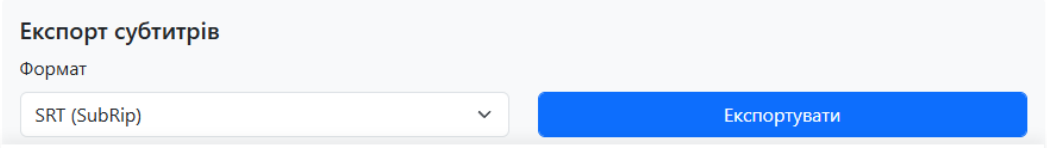


Рисунок 7. Експорт субтитрів

3.3 інтеграція моделі gemini: виділення ключових моментів

Наступний крок — аналіз отриманого тексту. Тут застосовується модель gemini — сучасна мовна модель від google, яка спеціалізується на контекстному розумінні текстів. Вона дозволяє:

Автоматично виділяти найцікавіші моменти (ключові фрази, емоційні кульмінації, питання тощо);

Формувати текстові резюме чи тематичні маркери;

Готувати текст до нарізки відео на фрагменти.

3.4 проблеми при розробці

Складнощі з узгодженням часових міток

1. Невідповідність часових міток від gemini:

- Штучний інтелект часто повертав часові мітки, які не відповідали реальним сегментам транскрипції
- Було необхідно розробити складні алгоритми узгодження для корекції цих невідповідностей

2. Пошук відповідних сегментів:

- Проблема пошуку точної відповідності між текстом, згенерованим ai, та оригінальними сегментами
- Необхідність створення багаторівневого алгоритму пошуку з поступовим зниженням точності

Проблеми з api vertex ai

1. Нестабільність основної моделі:

- Модель gemini-2.0-flash-001 іноді видавала помилки або не відповідала
- Довелося реалізувати механізм переходу на резервну модель gemini-1.0-pro

2. Обмеження формату відповіді:

- Труднощі з отриманням чистого json від моделі
- Потрібні були додаткові методи парсингу та обробки відповіді

Виклики семантичного пошуку

1. Ефективність пошуку:

- Складнощі з пошуком релевантних фрагментів у великих транскрипціях
- Потреба у створенні алгоритму для оцінки семантичної схожості

2. Визначення контексту:

- Проблема з визначенням правильного розміру контексту навколо знайдених фрагментів
- Збалансування між інформативністю та стислістю результатів

Якість розпізнавання релевантності

1. Низькоякісні збіги:

- Проблема з "фантомними" збігами, коли модель вважала сегменти релевантними, хоча вони такими не були
- Довелося впровадити додаткові фільтри та перевірки якості

2. Мовні нюанси:

- Складнощі з обробкою української мови та її специфіки
- Необхідність адаптації промптів для кращого розуміння контексту моделлю

Технічні труднощі

1. Управління токенами:

- Ризик перевищення ліміту токенів при великих транскрипціях
- Необхідність обрізання та стиснення запитів для моделі

2. Обробка помилок:

- Складність передбачення всіх можливих сценаріїв помилок api
- Потреба в розробці надійної системи обробки помилок та повторних спроб

Проблеми з продуктивністю

1. Час відповіді:

- Відносно тривалий час генерації найцікавіших моментів, особливо для довгих транскрипцій

2. Ресурсомісткість:

- Обробка великих транскрипцій вимагала значних обчислювальних ресурсів
- Необхідність оптимізації алгоритмів для зменшення навантаження

Загальні проблеми розробки

1. Складність тестування:

- Труднощі з написанням тестів для функцій, що залежать від відповідей ai
- Варіативність результатів та необхідність мокування ai-відповідей

2. Документація vertex ai:

- Неповна документація по роботі з gemini api в рамках vertex ai
- Довелося експериментувати з різними підходами для досягнення бажаних результатів

3.5 вирішення цих проблем

Вирішення проблем з узгодженням часових міток

1. Невідповідність часових міток від gemini:

- Впровадження методу alignhighlightswithsegments для синхронізації міток ai з реальними сегментами

- Створення чітких інструкцій у промптах для ai з наголосом на важливості точних часових міток
- Додавання спеціальних поміток у промпті: "важливо: для кожного знайденого моменту вкажи конкретні сегменти"

2. Пошук відповідних сегментів:

- Розробка багаторівневого алгоритму пошуку у методі findbestmatchingsegments
- Впровадження каскадного підходу: від точного пошуку до пошуку по ключовим фразам та словам
- Використання оцінки подібності текстів для визначення найкращих відповідностей

Рішення проблем з api vertex ai

1. Нестабільність основної моделі:

- Реалізація функції usebackupmodel для автоматичного переключення на резервну модель
- Додавання обробки помилок та логування проблем з api
- Створення системи повторних спроб з використанням альтернативних моделей

2. Обмеження формату відповіді:

- Впровадження регулярних виразів (jsonmatch = textresponse.match(/\[[\s\s]*\]/);) для точного вилучення json
- Додаткова перевірка та обробка відповідей моделі
- Чіткі вказівки у промптах щодо необхідного формату: "поверни результат виключно у форматі json масиву"

Вирішення викликів семантичного пошуку

1. Ефективність пошуку:

- Створення спеціальної функції calculatesimilarity для оцінки релевантності на основі перетину множин слів
- Впровадження порогових значень для відсікання низькоякісних результатів
- Оптимізація параметрів запитів до аі для покращення семантичного пошуку

2. Визначення контексту:

- Реалізація динамічного контексту за допомогою параметра contextsize
- Алгоритм розширення контексту, що додає сусідні сегменти до знайдених фрагментів
- Балансування контексту з урахуванням мінімальної та максимальної тривалості

Покращення якості розпізнавання релевантності

1. Низькоякісні збіги:

- Впровадження метаданих для відстеження якості збігів (_matchquality, _lowqualitymatch)
- Фільтрація результатів з низькою якістю збігу: highlight.similarity >= similaritythreshold
- Додаткова перевірка тривалості фрагментів для уникнення занадто коротких чи довгих результатів

2. Мовні нюанси:

- Адаптація промптів спеціально для української мови
- Очищення тексту від надлишкових пробілів та приведення до нижнього регістру для кращого порівняння
- Використання кількох ключових слів для пошуку відповідностей у випадку складних мовних конструкцій

Технічні оптимізації

1. Управління токенами:

- Налаштування моделі на низьку температуру (0.2) для більш детермінованих результатів
- Оптимізація промптів для зменшення кількості токенів
- Використання параметрів temperature, topk, topp та maxoutputtokens для контролю генерації

2. Обробка помилок:

- Розробка комплексної системи перехоплення помилок з інформативними повідомленнями
- Реалізація детального логування для відстеження проблем
- Створення конкретних типів помилок http на рівні контролера для зручного дебагу

Вирішення проблем продуктивності

1. Час відповіді:

- Оптимізація налаштувань генеративних моделей для швидшої відповіді
- Раціональне використання контексту для зменшення розміру запитів

2. Ресурсомісткість:

- Оптимізація алгоритмів пошуку для зменшення обчислювального навантаження
- Впровадження ранньої фільтрації неперспективних сегментів
- Інтелектуальний відбір ключових слів для пошуку, замість повного перебору

Загальні підходи до розробки

1. Тестування:

- Створення проміжних точок перевірки для валідації результатів
- Додавання метаданих для відстеження процесу обробки

- Розділення логіки на менші функції для полегшення тестування

2. Документація та api:

- Розробка чіткого та детального api для контролера
- Впровадження структурованого вводу та виводу з використанням інтерфейсів typescript
- Забезпечення зрозумілих повідомлень про помилки для кінцевих користувачів

Успішне вирішення цих проблем дозволило створити робочий сервіс виділення найцікавіших моментів, який може ефективно працювати з різними типами транскрипцій та надавати користувачам високоякісні результати навіть при неідеальних вхідних даних або складних запитах.

3.8 інтерфейс користувача (частина з gemini)

Інтерфейс взаємодії з gemini було розроблено з фокусом на надійність та стабільність. Основна увага приділялася чітким промптам з детальними інструкціями, що примушували модель дотримуватися заданої структури json відповідей. Це дозволило створити передбачуваний та стабільний api, який успішно опрацьовує різноманітні транскрипції та повертає точні часові мітки навіть при складних запитах. Запасна стратегія з використанням альтернативної моделі забезпечила безперервність роботи сервісу навіть при виникненні проблем з основною моделлю.

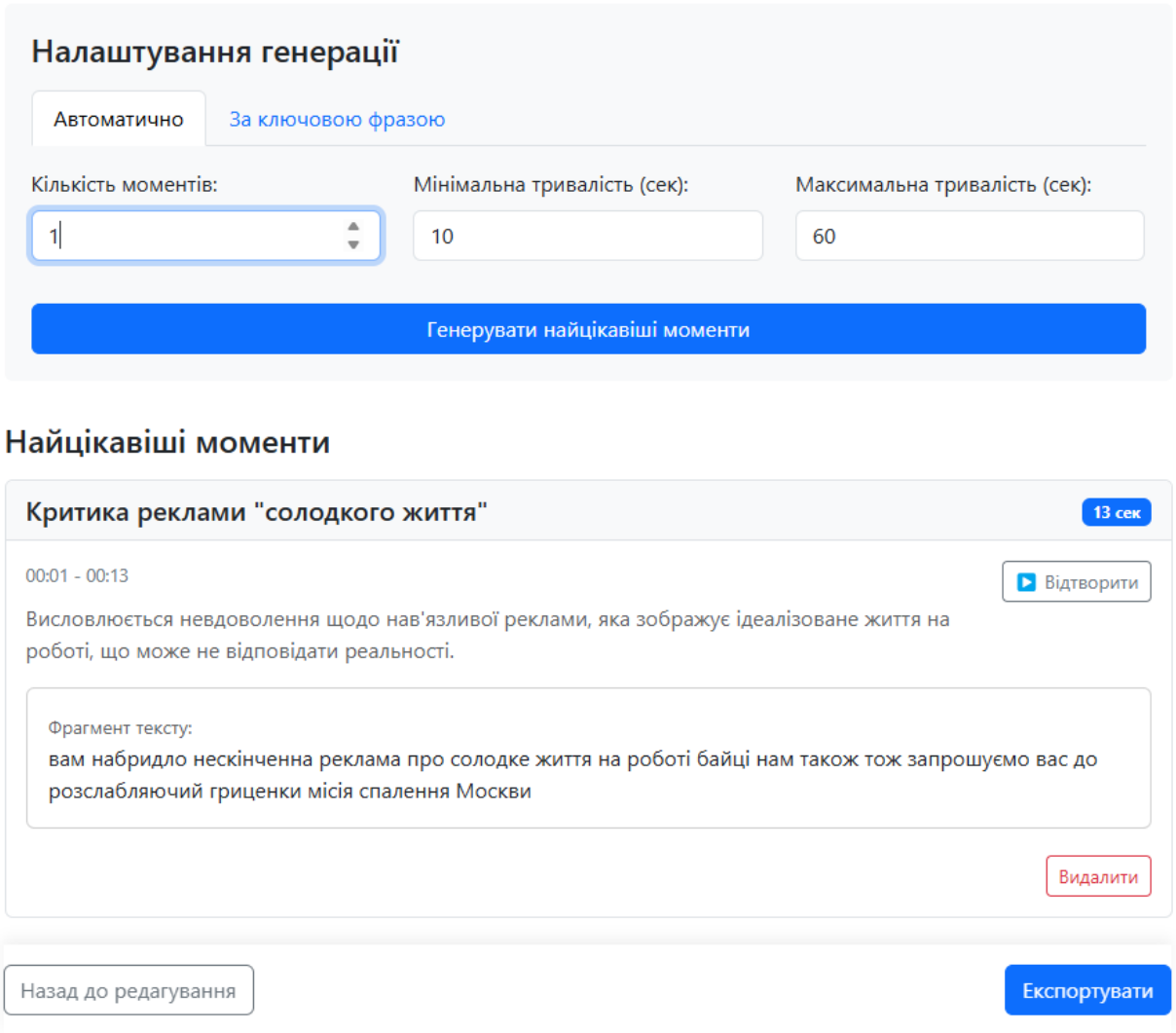


Рисунок 7. Генерація найцікавіших моментів

Висновок

У результаті виконання курсової роботи було реалізовано веб-застосунок, який автоматизує процес створення субтитрів до аудіо- та відеофайлів із використанням сучасних технологій штучного інтелекту. Основними досягненнями проєкту є:

1. Інтеграція з сервісами Google Cloud — успішно реалізовано взаємодію з Google Speech-to-Text для отримання якісної транскрипції з мультимедійних файлів, що суттєво підвищує точність та ефективність обробки звукових даних.
2. Використання мовної моделі Gemini — запровадження LLM для аналізу транскрипту, пошуку ключових моментів та формування найрелевантніших уривків за заданими критеріями продемонструвало потенціал глибокого навчання у практичних задачах.
3. Реалізація зручного інтерфейсу користувача — забезпечено можливість редагування тексту, перегляду фрагментів відео, генерації субтитрів у форматах .srt та .vtt, що робить застосунок практичним і доступним для широкого кола користувачів.
4. Сучасна архітектура на базі NestJS — обраний фреймворк дозволив побудувати масштабовану, модульну систему, яка легко підтримується та розширюється, відкриваючи перспективи для подальшого розвитку.

Список використаних джерел

1.The ultimate guide to finding bugs with nuclei – ProjectDiscovery Blog [Електронний ресурс] // ProjectDiscovery Blog. – Режим доступу: https://projectdiscovery.io/blog/ultimate-nuclei-guide (дата звернення: 05.05.2025) ProjectDiscovery

2.Medium [Електронний ресурс] // Medium. – Режим доступу: https://medium.com/ (дата звернення: 05.05.2025)

3.Web application security vulnerabilities detection approaches: a systematic mapping study / Sajjad Rafique [та ін.] // 2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD). – 2015. – С. 1–6

4.Hoffman A. Web application security: exploitation and countermeasures for modern web applications / Andrew Hoffman. – 2-ге вид. – [Б. м.] : O'Reilly Media, 2024. – 441 с.

5.Using Nuclei Templates for Vulnerability Scanning [Електронний ресурс] // Orca Security Blog. – Режим доступу: https://orca.security/resources/blog/using-nuclei-templates-for-vulnerability-scanning/ (дата звернення: 05.05.2025)

6.GitHub – projectdiscovery/nuclei-templates: community curated list of templates for the nuclei engine to find security vulnerabilities [Електронний ресурс] // GitHub. – Режим доступу: https://github.com/projectdiscovery/nuclei-templates (дата звернення: 05.05.2025)

7.Saad E., Mitchell R. Web security testing guide [Електронний ресурс] / Elie Saad, Rick Mitchell. – [Б. м.] : OWASP, 2020. – 465 с. – Режим доступу: https://github.com/OWASP/wstg/releases/download/v4.2/wstg-v4.2.pdf (дата звернення: 05.05.2025)