МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних
інформаційних систем
_____ Юрій КОНДРАТЕНКО
«____»_____2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ІНФОРМАЦІЙНА СИСТЕМА ТРАНСКРИБУВАННЯ ЗАПИСІВ ТЕЛЕФОННИХ ДЗВІНКІВ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувач _____ Валерія КОНОВАЛЬЧУК
«___»_____2025 р.

Керівник ст. викладач _____ Євген ДАРНАПУК
«___»_____2025 р.

Миколаїв – 2025

Чорноморський національний університет імені Петра Могили

Факультет	Комп'ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступень	Бакалавр
Спеціальність	122 Комп'ютерні науки
Освітня програма	Комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Юрій КОНДРАТЕНКО

«____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Коновальчук Валерії Сергіївни
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи «Інформаційна система транскрибування записів телефонних дзвінків».

Керівник роботи Дарнапук Євген Сергійович, ст. викладач.

Затверджена наказом ЧНУ ім. Петра Могили від «18» листопада 2024 р. № 321.

2. Строк представлення кваліфікаційної роботи студентом «18» червня 2025 р.

3. Вхідні (початкові) дані до роботи: аналіз існуючих рішень та технологій для автоматичного транскрибування аудіоінформації; наявні інструменти розпізнавання мовлення; вимоги до обробки аудіоданих у реальному часі; архітектурні підходи та технології для створення інформаційних систем мовного аналізу.

Очікуваний результат: інформацій система для транскрибування телефонних дзвінків з використанням сучасних інструментів розпізнавання мовлення та зберігання текстових даних, реалізована з урахуванням принципів масштабованості та надійності..

4. Перелік питань, що підлягають розробці (зміст пояснювальної записки):

– аналіз інформаційних систем розпізнавання мовлення та транскрибування аудіоданих;

– архітектурні підходи, патерни проєктування та технології, що використовуються для побудови інформаційних систем транскрибування;

– проєктування та розробка інформаційної системи транскрибування телефонних дзвінків;

– реалізація системи за допомогою сучасних фреймворків та API для розпізнавання мовлення;

– тестування системи на реальних аудіозаписах телефонних дзвінків.

5. Перелік графічного матеріалу: презентація.

Керівник роботи _____ Євген ДАРНАПУК

(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Валерія КОНОВАЛЬЧУК

(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «20» грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: Інтелектуальна система транскрибування записів телефонних дзвінків

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	18.12.2024	20.12.2024	Виконано
2	Аналіз області транскрибування та постановка задачі	21.12.2024	29.01.2025	Виконано
3	Складання календарного плану роботи на весь період виконання КР	30.01.2025	30.01.2025	Виконано
4	Огляд літературних джерел за темою КР щодо транскрибування аудіофайлів	31.01.2025	01.03.2025	Виконано
5	Аналіз існуючих систем транскрибування аудіофайлів, огляд існуючих технологій	02.03.2025	01.04.2025	Виконано
6	Розробка ПЗ за допомогою обраних технологій з аналізом отриманих результатів	02.04.2025	15.05.2025	Виконано
7	Перший попередній захист КР на засіданні комісії кафедри	16.05.2025	16.05.2025	Виконано
8	Корегування роботи за результатами попереднього предзахисту	17.05.2025	29.05.2025	Виконано
9	Другий попередній захист КР на засіданні комісії кафедри	30.05.2025	30.05.2025	Виконано
10	Доробка та остаточне оформлення КР	31.05.2025	10.06.2025	Виконано
11	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	11.06.2025	17.06.2025	Виконано
12	Захист КР перед екзаменаційною комісією (ЕК)	18.06.2025	18.06.2025	Виконано

Керівник роботи _____ Євген ДАРНАПУК
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Валерія КОНОВАЛЬЧУК
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«30» січня 2025 р.

АНОТАЦІЯ

до кваліфікаційної роботи
здобувачки групи 402 ЧНУ ім. Петра Могили

Коновальчук Валерії Сергіївни

на тему: "ІНФОРМАЦІЙНА СИСТЕМА ТРАНСКРИБУВАННЯ ЗАПИСІВ ТЕЛЕФОННИХ ДЗВІНКІВ"

Дана кваліфікаційна робота присвячена аналізу та розробці інформаційної системи для транскрибування телефонних дзвінків. Метою роботи є створення ефективної та надійної системи, що дозволить автоматично перетворювати мовлення з аудіозаписів телефонних розмов у текстову форму, забезпечуючи високу точність розпізнавання та зручне зберігання даних.

У роботі розглянуто основні вимоги до систем розпізнавання мовлення та транскрибування, проведено огляд сучасних інструментів та сервісів для перетворення мовлення в текст. На основі цього аналізу було розроблено власний проєкт системи, що використовує сучасні API для розпізнавання мовлення, мікросервісну архітектуру та засоби інтеграції з базами даних і зовнішніми сервісами.

Розроблена система складається з кількох ключових компонентів, серед яких:

– сервіс обробки аудіозаписів телефонних дзвінків;

– сервіс транскрибування з використанням інструментів розпізнавання мовлення;

– база даних для зберігання транскриптів;

– веб-інтерфейс для перегляду та пошуку транскрибованих дзвінків.

Особливу увагу приділено питанням якості розпізнавання мовлення, збереженню конфіденційності записів, а також забезпеченню зручного доступу до транскрибованих даних для різних категорій користувачів.

Запропонована інформаційна система дозволяє значно покращити процес документування розмов, підвищити ефективність обробки дзвінків у службах підтримки та контакт-центрах, а також зменшити людський фактор при створенні текстових звітів.

Об'єктом роботи є процес автоматизованого перетворення аудіоінформації у текстовий формат.

Предмет роботи – програмні засоби та алгоритми для транскрипції мовлення.

Мета роботи – підвищення ефективності обробки аудіофайлів у процесі транскрипції.

Загальний обсяг роботи – 177 сторінок. Кваліфікаційна робота містить 1 додаток, 115 рисунків, 11 таблиць і 31 джерел посилання.

Ключові слова: інформаційна система, транскрибування, розпізнавання мовлення, телефонні дзвінки, автоматизація, обробка аудіо, мікросервіси.

ABSTRACT

to the qualification work by the student of the group 402
of Petro Mohyla Black Sea National University

Konovalchuk Valeriia

« INFORMATION SYSTEM FOR TRANSCRIPTION OF TELEPHONE CALLS RECORDINGS»

This qualification work is devoted to the analysis and development of an information system for transcribing telephone calls. The aim of the work is to create an effective and reliable system that will allow to automatically convert speech from audio recordings of telephone conversations into text form, ensuring high recognition accuracy and convenient data storage.

The work considers the main requirements for speech recognition and transcription systems, and reviews modern tools and services for converting speech to text. Based on this analysis, an own system project was developed that uses modern APIs for speech recognition, microservice architecture, and means of integration with databases and external services.

The developed system consists of several key components, including:

– a service for processing audio recordings of telephone calls;

– a transcription service using speech recognition tools;

– a database for storing transcripts;

– a web interface for viewing and searching for transcribed calls.

Particular attention is paid to the issues of speech recognition quality, maintaining the confidentiality of recordings, as well as ensuring convenient access to transcribed data for different categories of users.

The proposed information system allows you to significantly improve the process of documenting conversations, increase the efficiency of call processing in support services and contact centers, and reduce the human factor when creating text reports.

The object of the work is the process of automated conversion of audio information into text format.

The subject of the work is software tools and algorithms for speech transcription.

The purpose of the work is to increase the efficiency of processing audio files in the transcription process.

The overall scope of the work is 177 pages. Thesis contains 1 application, 115 figures, 11 tables and 31 references in it.

Keywords: information system, transcription, speech recognition, telephone calls, automation, audio processing, microservices.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧКИ

БД – База даних

ПЗ – Програмне забезпечення

СКБД – Система керування базами даних

API – Application Programming Interface

CER – Character Error Rate

CORS – Cross-Origin Resource Sharing

JSON – JavaScript Object Notation

JWT – JSON Web Token

REST – Representational State Transfer

SDK – Software Development Kit

STT – Speech-to-Text

TS – TypeScript

TSX – TypeScript XML

UI – User Interface

UUID – Universally Unique Identifier

WAV – Waveform Audio File Format

WER – Word Error Rate

ВСТУП

У сучасному світі інформаційні технології стають невід’ємною частиною життя, неперервно інтегруючись у різноманітні сфери діяльності—від підприємництва й освіти до медіа та наукових досліджень. Одним із ключових напрямів розвитку цієї галузі є автоматизація обробки аудіоінформації, зокрема перетворення усного мовлення на текст. Такий процес, відомий як транскрибування, виконує критично важливі функції у створенні субтитрів, аналізі телефонних дзвінків, підготовці юридичних записів, розробці навчальних матеріалів та вирішенні багатьох інших завдань.

Актуальність цього питання визначається стрімким зростанням обсягів аудіовізуального контенту, такого як подкасти, відеолекції та інтерв'ю, що потребує ефективних методів швидкого й точного отримання текстової інформації. Оскільки ручне транскрибування є не лише трудомістким, але й витратним у часовому вимірі процесом, використання систем автоматичного розпізнавання мовлення, побудованих на базі штучного інтелекту, наприклад, OpenAI Whisper, набуває дедалі більшої популярності.

Метою даної роботи є підвищення ефективності обробки аудіофайлів у процесі транскрипції. Застосунок має забезпечувати можливість завантаження аудіоконтенту або запису мовлення за допомогою мікрофона, передавання цих даних на обробку за допомогою Whisper, а також отримання текстового результату у зручному форматі. Крім того, система повинна мати інтуїтивно зрозумілий інтерфейс, підтримувати чергу задач, інформувати користувачів про стан обробки та надавати можливість завантаження готових результатів у вигляді текстових файлів.

У рамках дослідження передбачено виконання наступних завдань:

- проведення аналізу наявних рішень для автоматичного транскрибування;
- визначення вимог до інформаційної системи;
- вибір відповідних технологій, архітектурних шаблонів і методів реалізації;

– розробка програмного застосунку із застосуванням сучасного стеку технологій, таких як Python, Flask, React, Celery тощо;

– проведення тестування та оцінка точності отриманих транскрипцій.

Об'єктом роботи є процес автоматизованого перетворення аудіоінформації у текстовий формат.

Предмет роботи – програмні засоби та алгоритми для транскрипції мовлення.

Мета роботи – підвищення ефективності обробки аудіофайлів у процесі транскрипції.

1 ТЕХНОЛОГІЧНІ ЗАСАДИ ТРАНСКРИБУВАННЯ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Сфера застосування технологій автоматичного транскрибування

Технології автоматичного транскрибування мовлення знайшли широке впровадження у численних галузях діяльності, де виникає потреба оперативного та точного перетворення аудіоінформації на текстові дані. Однією з найбільш розповсюджених галузей використання є сфера обслуговування клієнтів: транскрибування телефонних розмов забезпечує можливість аналізу якості роботи операторів, спостереження за дотриманням стандартів обслуговування, а також створення бази знань для навчання персоналу.

У медичному контексті автоматичне транскрибування використовується для документування консультацій лікарів, що полегшує ведення електронних медичних записів. У сфері юриспруденції ця технологія слугує для запису свідчень і розмов у процесі судових засідань або розслідувань. Значущою є роль транскрибування і в журналістиці, де воно дає можливість швидко обробляти інтерв'ю та публічні виступи.

Крім того, автоматичне транскрибування має застосування в освіті, маркетингових дослідженнях, аналітичній діяльності, а також у системах штучного інтелекту для навчання моделей, що займаються обробкою природної мови. Усі ці сфери потребують точних, масштабованих та ефективних рішень для перетворення мовлення у текст.

1.2 Аналіз існуючих програмних засобів та API для транскрибування

Розвиток сучасних технологій автоматичного транскрибування зазнає значних змін завдяки успіхам у галузі глибокого машинного навчання та обробки природної мови.

Існує чимало популярних рішень, що забезпечують високу точність перетворення мовлення на текст і підтримують різні мови, акценти, а також аудіоформати.

Одним із популярних рішень є Otter.ai — онлайн-платформа, що надає можливість завантаження аудіофайлів, автоматичного створення транскрипцій та спільного редагування тексту в режимі реального часу [28]. Otter підтримує англійську мову, дозволяє створювати позначки, додавати коментарі й має функцію автоматичного розпізнавання мовців. Інтерфейс цього сервісу зображений на рис. 1.1.

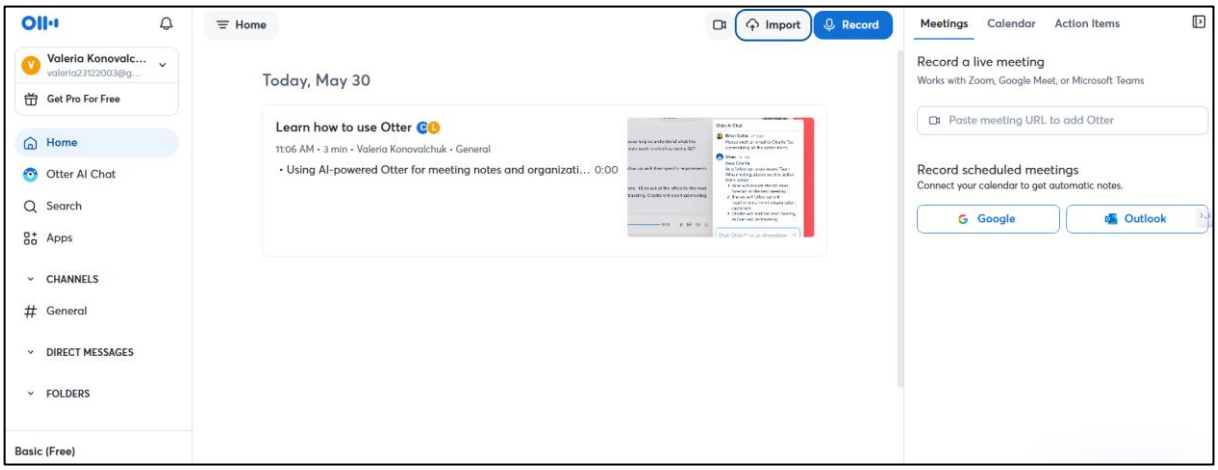


Рисунок 1.1 – Інтерфейс Otter.ai

Ще одним прикладом є Descript — потужний інструмент, що поєднує функції транскрибування, редагування відео та аудіо [29]. Платформа пропонує функціонал видалення шумів, зміни голосу та редагування транскрипції як звичайного тексту, що автоматично змінює відповідний аудіофайл (рис. 1.2).

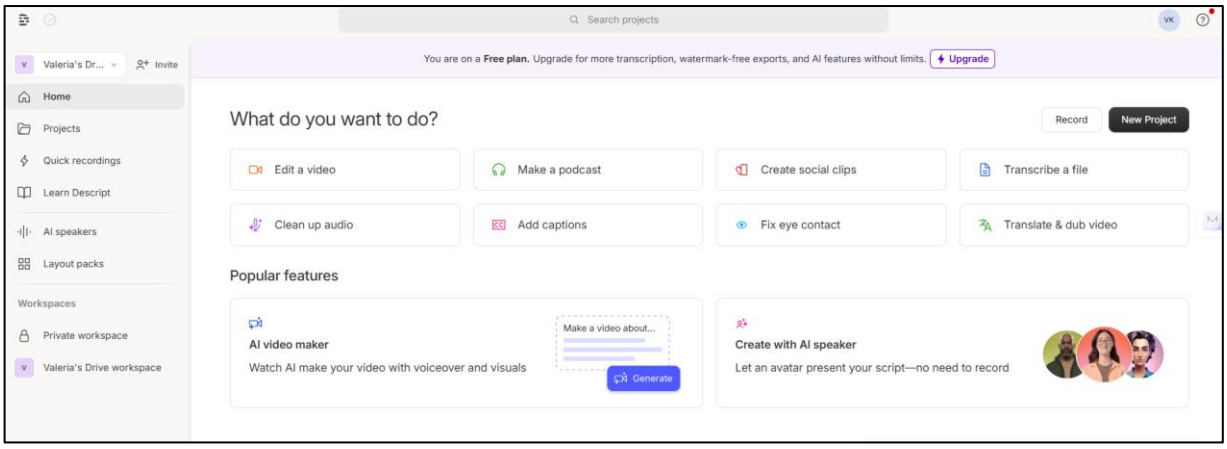


Рисунок 1.2 – Інтерфейс Descript

Sonix.ai — багатомовна система транскрибування, яка підтримує понад 40

мов [30]. Вона надає користувачам зручний редактор, можливість перекладу транскрипцій та інтеграцію з хмарними сервісами для зберігання файлів (рис. 1.3).

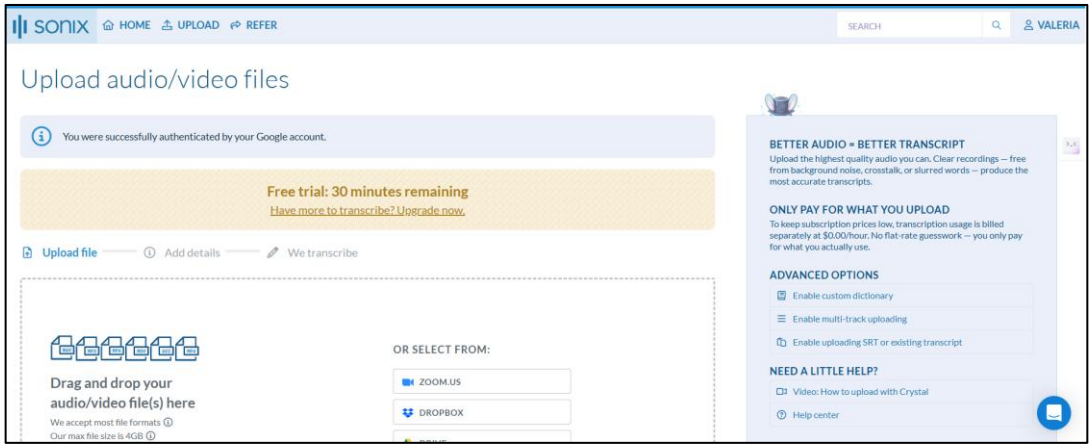


Рисунок 1.3 – Інтерфейс Sonix.ai

Також варто згадати Trint — сервіс, який поєднує автоматичне розпізнавання мовлення з інструментами для редагування, аналізу й співпраці [31]. Trint особливо корисний у медіа-секторі завдяки функціям пошуку за словами в аудіо та відео. Інтерфейс сервісу представлений на рис. 1.4.

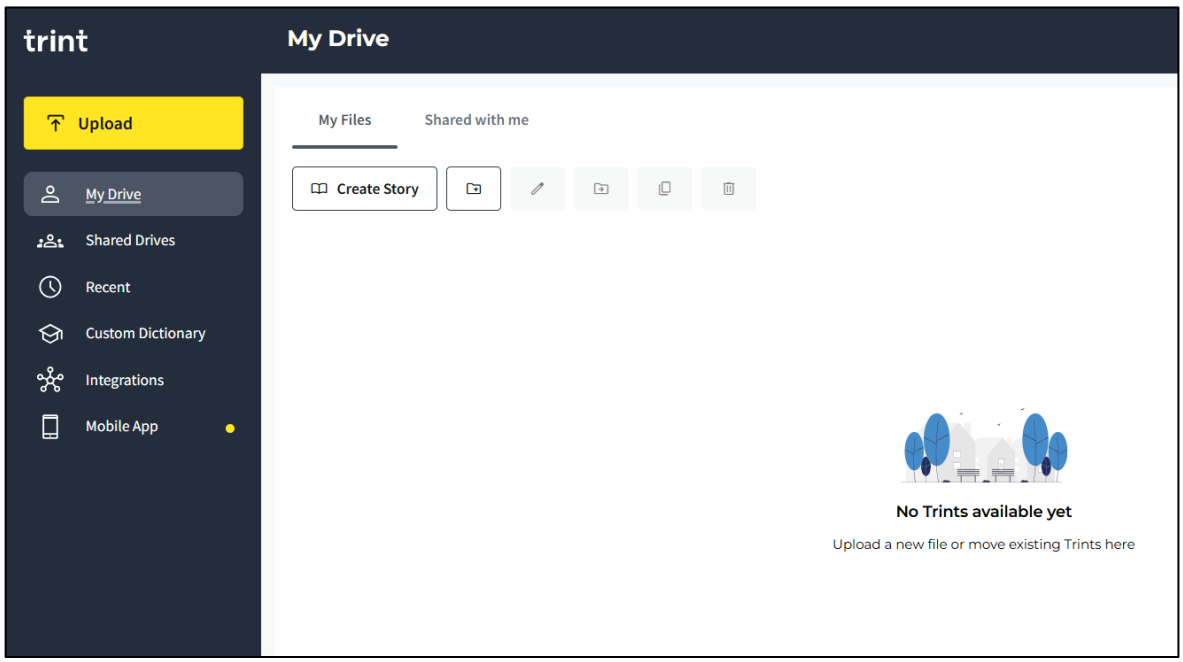


Рисунок 1.4 – Інтерфейс Trint

В табл. 1.1 описано порівняння функцій чотирьох популярних систем транскрибування: Otter.ai, Descript, Sonix.ai та Trint.

Таблиця 1.1 – Порівняння чотирьох популярних систем транскрибування

Функція / Сервіс	Otter.ai	Descript	Sonix.ai	Trint
Підтримка мов	Лише англійська	Переважно англійська	40+ мов	30+ мов
Розпізнавання мовців	Автоматичне	Автоматичне	Автоматичне	Автоматичне
Редактор транскрипції	Інтерактивний редактор	Повне редагування як тексту	Зручний текстовий редактор	З інтегрованим відео/аудіо плеєром
Формати аудіо/відео	MP3, WAV, MP4	MP3, WAV, MP4, MOV, AVI	MP3, WAV, M4A, MP4, AIFF, AAC	MP3, MP4, MOV, AVI, WAV
Спільна робота	Підтримується	Підтримується	Спільний доступ до файлів	Командна робота
Підтримка перекладу	Немає	Немає	Автоматичний переклад	Автоматичний переклад
Інтеграції	Google Meet, Dropbox, Zoom	Google Drive, Zoom, Slack	Google Drive, Dropbox, Zoom	Adobe Premiere, Zoom, Dropbox
Додаткові функції	Субтитри, виявлення ключових слів	Відео-/аудіоредактор, Overdub (зміна голосу)	Таймкоди, переклад, словник	Пошук за словами в медіа, словник термінів
Тип платформи	Веб, мобільний застосунок	Десктопна програма + веб	Веб	Веб

Ці системи демонструють широкий функціонал та орієнтацію на користувацький досвід, дозволяючи не лише конвертувати аудіо в текст, а й ефективно керувати отриманою інформацією.

Google Speech-to-Text є одним із найзнаніших хмарних сервісів для розпізнавання мовлення [1]. Його ключовими перевагами є висока точність, підтримка понад 120 мов і діалектів, можливість потокового розпізнавання та адаптації до специфічної лексики. Інтеграція з Google Cloud дозволяє ефективно масштабувати проекти й зберігати аудіоматеріали у хмарі.

OpenAI Whisper представляє собою відкритий проект для транскрибування, що демонструє високу точність навіть за умов зашумлених або низькоякісних

аудіо [3]. Whisper функціонує локально або на сервері, що дозволяє уникати передавання конфіденційної інформації стороннім сервісам. Він характеризується підтримкою багатомовності, автоматичним визначенням мови, сегментацією мовлення, а також функціоналом перекладу.

Deepgram – це комерційний API-сервіс, який забезпечує транскрибування в режимі реального часу [4]. Він характеризується високою продуктивністю, підтримкою різних моделей, включаю чи моделі для телефонного мовлення, можливістю навчання на власних даних та автоматичним виділенням мовців (speaker diarization). В табл. 1.1 порівняно ці три системи.

На рис. 1.5 зображено результати порівняння коефіцієнта помилок слів (WER) та коефіцієнта помилок символів (CER) з використанням двох різних систем STT, описаних вище [5, 6]. Синя лінія позначає вхідні дані, жовта лінія представляє модель лише з текстом, а червона лінія представляє результати, отримані запропонованим методом.

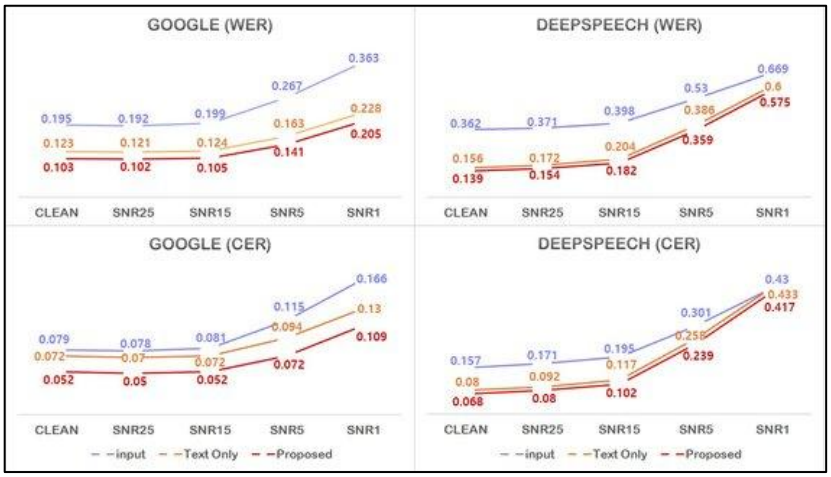


Рисунок 1.5 – Результати порівняння коефіцієнтів помилок слів та помилок символів двох різних систем STT

Таблиця 1.2 – Порівняння трьох різних API STT

Назва	**Тип доступу**	**Багатомовність**	**Локальна робота**	**Ведення діалогу**	**API**
Google STT [1]	Хмарний (платний)	Так	Ні	Так	Так
Whisper [3]	Open-source	Так	Так	Обмежено	Ні
Deepgram [4]	Хмарний (платний)	Так	Ні	Так	Так

Інші рішення, такі як AssemblyAI, IBM Watson Speech-to-Text, Amazon Transcribe та Speechmatics, також займають значне місце на ринку. Більшість із них пропонують SDK, REST API, функціонал виділення ключових слів, автоматичну пунктуацію та атрибуцію мовців, що значно покращує якість транскрибування.

1.3 Постановка задачі: розробка вебзастосунку для транскрибування

Автоматичне транскрибування телефонних розмов є важливим завданням, яке має застосування в різних галузях, таких як клієнтська підтримка, юридичні фірми, медичні заклади, служби безпеки та маркетингові агентства. Враховуючи, що ручна обробка великих обсягів аудіозаписів є трудомісткою і неефективною, виникає потреба у створенні вебзастосунку, здатного швидко і точно перетворювати мовлення на текст.

Мета проєкту – розробити вебзастосунок, який дає змогу завантажувати аудіофайли телефонних розмов, автоматично їх транскрибувати, зберігати результати у зручному форматі та забезпечувати інтуїтивно зрозумілий інтерфейс для користувачів. Це включає інтеграцію сучасних інструментів розпізнавання мовлення, таких як Google Speech-to-Text або OpenAI Whisper [1, 3].

Для досягнення цієї мети необхідно вирішити наступні завдання:

– проаналізувати доступні API та бібліотеки для транскрибування аудіо;

– розробити архітектуру вебзастосунку з поділом на клієнтську і серверну частини;

– реалізувати функціонал для завантаження аудіофайлів користувачем;

– інтегрувати обрану модель транскрибування і налаштувати її для обробки телефонних записів;

– забезпечити відображення транскрибованого тексту в зручному для читання вигляді (включаючи часові позначки, імена мовців тощо);

– створити базову систему збереження та експорту транскриптів (у форматах PDF, TXT).

Блок-схема алгоритму системи транскрибування зображена на рис. 1.6.

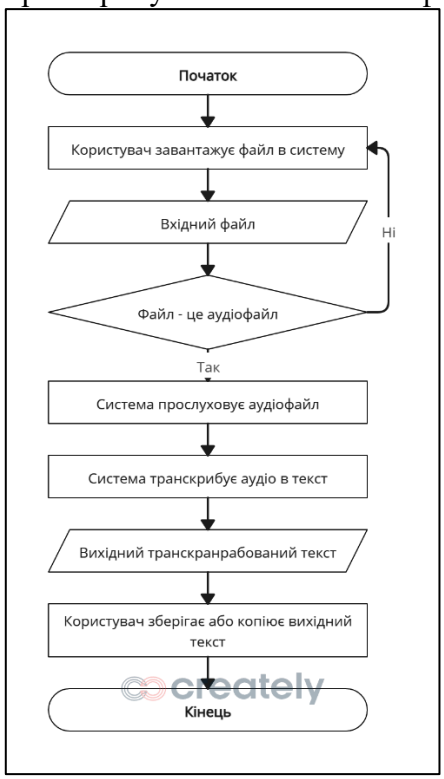


Рисунок 1.6 – Блок-схема системи транскрибування

Об’єктом дослідження є процес транскрибування телефонних розмов. Предметом виступає інформаційна система, що автоматизує цей процес на базі вебтехнологій.

Методи: для реалізації буде використано стек технологій, зокрема React.js (для клієнтської частини), Python (для серверної логіки), а також API для мовного розпізнавання (наприклад, Whisper API).

1.4 Вимоги до функціоналу системи

Розробка вебзастосунку для транскрибування телефонних дзвінків вимагає ретельного визначення функціональних та нефункціональних вимог. Це дозволяє сформувати чітке уявлення про передбачувану поведінку системи, її функціональні можливості, інтерфейсну взаємодію та технічні обмеження.

Основні функціональні вимоги включають:

– авторизацію користувача (опційно), яка охоплює процеси реєстрації, входу в систему та управління обліковими даними;

– завантаження аудіофайлів зі підтримкою основних форматів (mp3, wav, ogg) та встановлення обмежень на розмір файлів;

– обробку аудіо шляхом передачі файлів на сервер для подальшого транскрибування з використанням обраного API або мовної моделі, наприклад Whisper або Google Speech-to-Text;

– відображення транскрипції з можливістю копіювання, пошуку та виділення фрагментів тексту;

– реалізацію міток часу та розпізнавання мовців для визначення учасників діалогу та їхньої ролі (за наявності підтримки в обраній моделі) ;

– збереження результатів з можливістю експорту транскрипцій у формати TXT, DOCX чи PDF;

– ведення історії транскрипцій з можливістю перегляду та редагування попередніх результатів (опційно).

Нефункціональні вимоги охоплюють:

– продуктивність системи, що передбачає обробку файлів тривалістю до 30 хвилин упродовж кількох хвилин, залежно від потужності сервера;

– створення інтуїтивно зрозумілого інтерфейсу з адаптивним дизайном для різних пристроїв;

– забезпечення надійності системи при обробці помилок під час завантаження чи розпізнавання;

– гарантування безпеки через забезпечення конфіденційності користувацьких даних та шифрування з'єднань.

1.5 Вибір архітектурного підходу

Для створення вебзастосунку, який забезпечує транскрибування телефонних дзвінків, раціонально скористатися клієнт-серверною архітектурою [7]. Цей підхід припускає чітке розмежування функцій між клієнтською частиною, яка відповідає за інтерфейс користувача, та серверною, яка займається обробкою логіки, збереженням даних і транскрипцією.

Клієнтська частина займається збором вхідних даних (аудіофайлів),

візуалізацією процесу транскрибування та відображенням результатів. Серверна частина здійснює складні обчислення, надсилає запити до сторонніх API, таких як Google Speech-to-Text чи OpenAI Whisper, або використовує локальні моделі розпізнавання мовлення для забезпечення збереження результатів у базі даних. На рис. 1.7 можна побачити порівняння трьох архітектур.

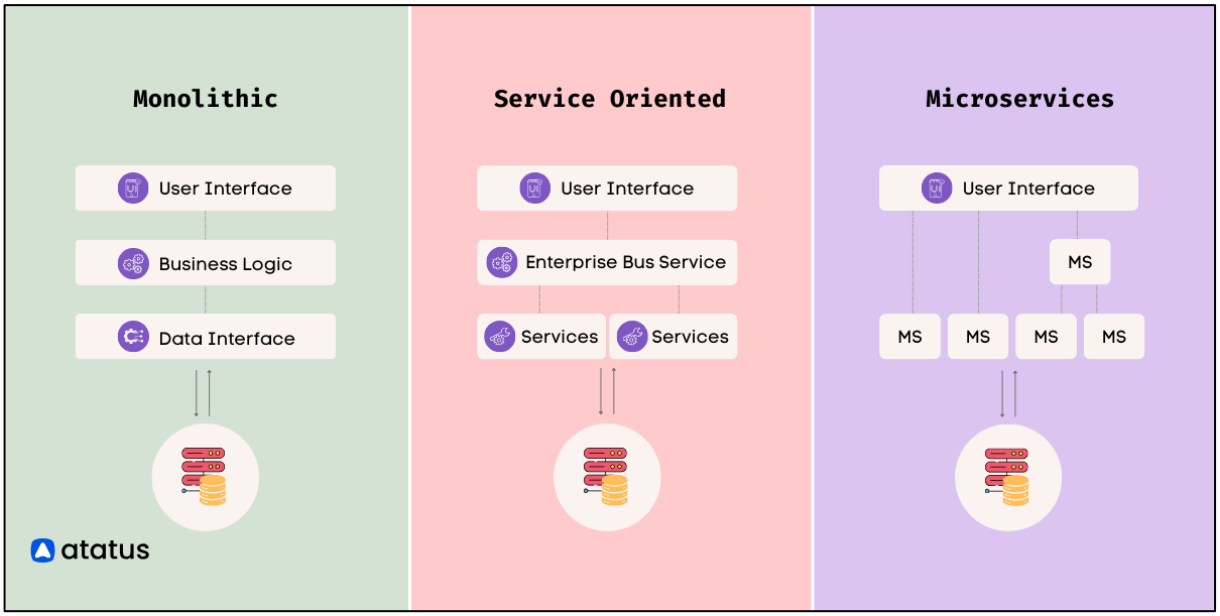


Рисунок 1.7 – Порівняння архітектур

З огляду на потенційні навантаження та потребу в масштабуванні системи, може бути доречно запровадити розподілену архітектуру. Вона дає змогу розділити обробку аудіо, зберігання файлів, управління чергами запитів та користувацькі сесії через різні мікросервіси або окремі модулі. Це забезпечує:

– підвищення продуктивності;
– стійкість до збоїв;
– гнучке масштабування системи.

Висновки до розділу 1

У першому розділі була здійснена аналітична оцінка основних аспектів, пов'язаних з автоматизованим транскрибуванням телефонних дзвінків, що є надзвичайно важливою проблемою в контексті сучасного інформаційного суспільства. Автоматичне перетворення мовленнєвого контенту у текст знайшло

широке застосування у різноманітних галузях, включаючи аналітику дзвінків в службах підтримки та формування текстових протоколів для медичних, юридичних і адміністративних цілей.

Проведено всебічний огляд актуальних технологічних рішень, серед яких особливе місце займають Google Speech-to-Text, OpenAI Whisper і Deepgram [1, 3, 4]. Ці інструменти забезпечують високий ступінь точності транскрипції та підтримують численні мови, однак кожен з них має специфічні обмеження у швидкодії, вартості чи вимогах до ресурсів.

У результаті проведеного аналізу сформульовано завдання створення вебзастосунку для транскрибування аудіоматеріалів. Зазначено функціональні та нефункціональні вимоги до системи, зокрема: зручність користування, можливість масштабування, точність транскрипції та підтримка різних форматів аудіофайлів.

З огляду на потреби користувачів та технічні вимоги, обґрунтовано вибір клієнт-серверної архітектури з потенціалом для подальшої розподіленої обробки даних. Такий підхід дозволяє забезпечити гнучкість, високу масштабованість і стабільність функціонування системи.

Таким чином, результати першого розділу слугують фундаментом для подальшого проектування та впровадження вебзастосунку.

2 ТЕХНОЛОГІЇ, ЩО ВИКОРИСТОВУЮТЬСЯ ДЛЯ РОЗРОБКИ ВЕБЗАСТОСУНКУ ДЛЯ ТРАНСКРИБУВАННЯ

2.1 Технології backend-у

Розробка надійної серверної частини для вебзастосунку з транскрибування аудіо відіграє ключову роль у його створенні. Цей компонент відповідає за обробку запитів від користувачів, інтеграцію з сервісами транскрипції, збереження даних, авторизацію користувачів і виконання транскрипційних функцій. У даному розділі буде розглянуто основні технології, які були обрані для реалізації серверної сторони цього проєкту.

2.1.1 Python як мова для обробки аудіо та реалізації логіки

Python є однією з найбільш популярних мов програмування у світі, і не випадково [8].

Гнучкість цієї мови програмування, читабельність коду та розвинута екосистема роблять його ідеальним вибором для розробки систем, що комбінують роботу з аудіо, машинне навчання, обробку даних та серверну логіку. Саме тому Python використовується в цьому проєкті як основна мова для створення серверної частини вебзастосунку з функцією автоматичного транскрибування аудіо.

На рис. 2.2 зображено Sequence-діаграму, тобто діаграму послідовності, яка візуалізує взаємодію між компонентами системи у часовому порядку [9].

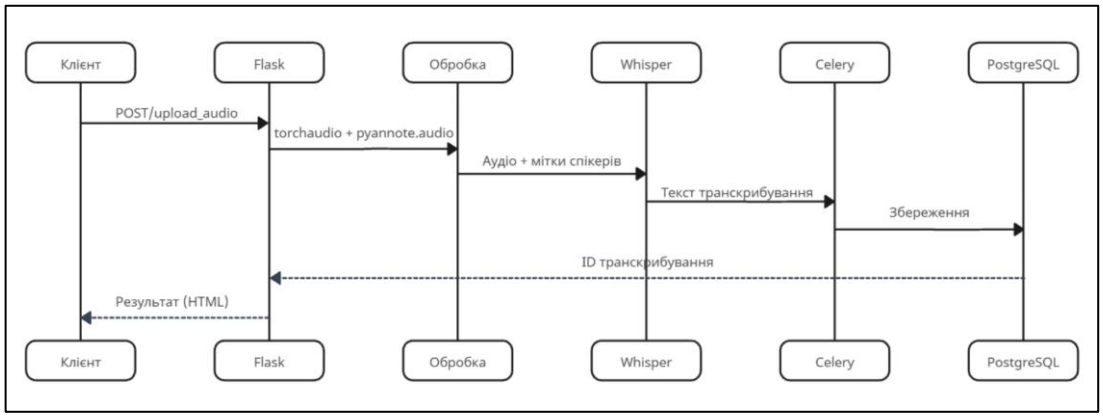


Рисунок 2.2 – Sequence-діаграма системи

Головною перевагою Python є зручний синтаксис, що дозволяє швидко реалізовувати навіть складні алгоритми, зберігаючи високу читабельність і підтримуваність коду. Це особливо важливо для командної роботи: код легко зрозумілий іншим членам команди і може бути швидко змінений чи доповнений.

Python активно використовується у наукових дослідженнях, розробці штучного інтелекту, обробці природної мови, комп'ютерному зорі, фінансовому аналізі, веброзробці, а також у сфері аудіоаналізу. Завдяки підтримці спільноти існує багато бібліотек і фреймворків для Python, які сприяють ефективному виконанню найрізноманітніших задач, включаючи асинхронну обробку, криптографію, взаємодію з базами даних та хмарними сервісами.

Python має важливу перевагу міжплатформеності. Код може виконуватися як на Windows, так і на Linux чи macOS, що спрощує процес впровадження та масштабування системи. Python добре інтегрується з іншими мовами та сервісами, що дозволяє будувати модульні гнучкі архітектури.

У контексті транскрибування, Python забезпечує надійну основу для обробки аудіо та побудови серверної логіки. В табл. 2.1 зазначені переваги Python для серверної логіки.

Таблиця 2.1 – Переваги використання Python для серверної логіки

Перевага	**Опис**
Читабельність	*Легко зрозумілий код*
Велика екосистема	*Багато бібліотек для будь-яких задач*
Кросплатформеність	*Працює на всіх основних ОС*
Продуктивність у розробці	*Швидкий розвиток функціоналу*
Підтримка асинхронності	*Можливість фонової обробки даних*

Крім того, Python підтримує інтеграцію з базами даних, хмарними сервісами та інтерфейсами користувача, дозволяючи створювати комплексні вебзастосунки зручні у використанні. Отже, Python – це не лише ефективний інструмент для швидкої розробки, але й потужна технологічна платформа для реалізації проєктів будь-якої складності.

2.1.2 Flask як вебфреймворк

Flask – це легкий і гнучкий вебфреймворк для мови Python, який дає змогу швидко створювати вебзастосунки різної складності [10]. Завдяки своїй мінімалістичній архітектурі, Flask пропонує лише основні функції, залишивши розробнику можливість обирати додаткові інструменти та бібліотеки для побудови комплексних систем. Це робить Flask відмінним вибором для проєктів, що потребують високої кастомізації, а також для навчальних або дослідницьких задач.

Одна з головних переваг Flask – зрозумілий синтаксис і зручна структура проєкту. Розробник може легко організовувати маршрути для обробки запитів користувача та повернення відповідей. Flask підтримує шаблонізатор Jinja2, що дозволяє динамічно формувати HTML-сторінки на основі наданих змінних та зручно реалізовувати контролери й логіку обробки даних.

Послідовність обробки запиту завантаження файлу, обробки та повернення зміненого файлу в Flask зображено на рис. 2.2.

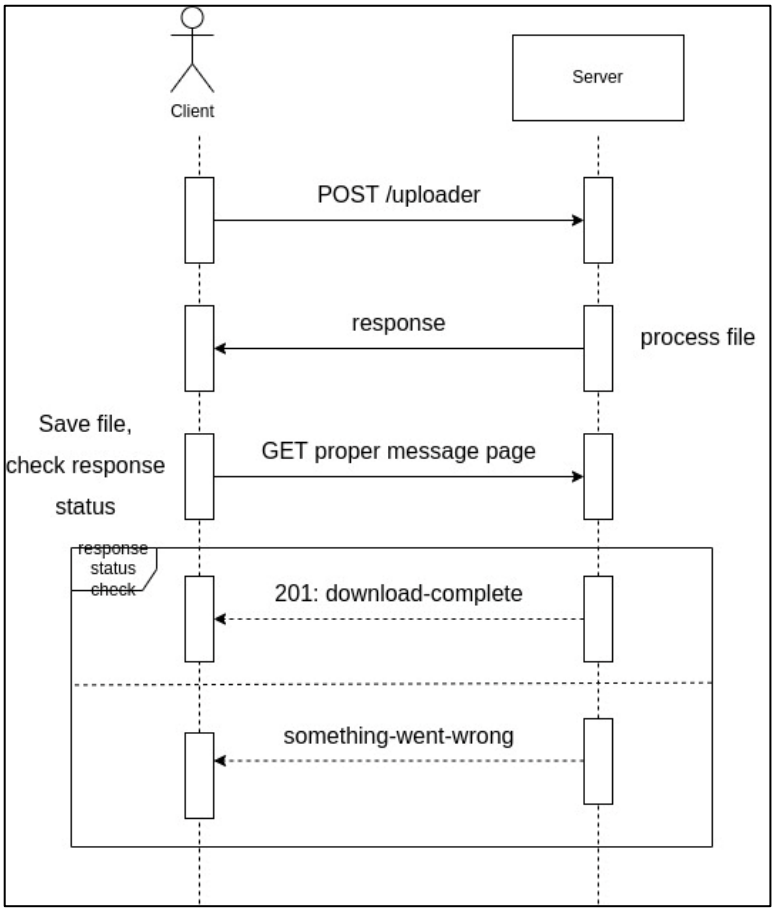


Рисунок 2.2 – Послідовність обробки запиту в Flask

Flask також добре підходить для взаємодії з асинхронними чергами задач, такими як Celery [11]. Це дає змогу передавати ресурсомісткі операції, наприклад, транскрибування аудіо, у фонові процеси, не блокуючи основний серверний потік.

Крім того, Flask полегшує роботу з оточенням через використання змінних середовища за допомогою бібліотеки python-dotenv. Це забезпечує безпечне зберігання конфіденційних даних, таких як ключі доступу до хмарного сховища або параметри підключення до бази даних. Переваги Flask описані в табл. 2.2.

Таблиця 2.2 – Переваги Flask у порівнянні з іншими популярними вебфреймворками

Критерій	Flask	Django	FastAPI
Простота та легкість	Так	Ні	Так
Гнучкість	Так	Ні	Так
Швидкість розробки	Так	Так	Так
Підтримка асинхронності	Обмежено	Ні	Так
Розширюваність	Так	Так	Так
ORM	Обмежено	Так	Обмежено
Автоматичне створення API	Ні	Обмежено	Так
Документація	Так	Так	Так
Придатність для великих систем	Так	Так	Так

2.1.3 OpenAI Whisper для транскрибування аудіофайлів

Whisper – це нейронна модель від OpenAI, що забезпечує автоматичне розпізнавання мовлення [3]. Вона вміє точно транскрибувати аудіо на більш ніж 50 мовах, включаючи українську, завдяки масштабному набору даних, на яких була навчена. Whisper побудовано на основі архітектури трансформерів, що гарантує високу точність розпізнавання навіть за складних умов, як-от високий рівень шуму, різні акценти або низька якість запису. На рис. 2.3 зображено схему роботи цієї моделі.

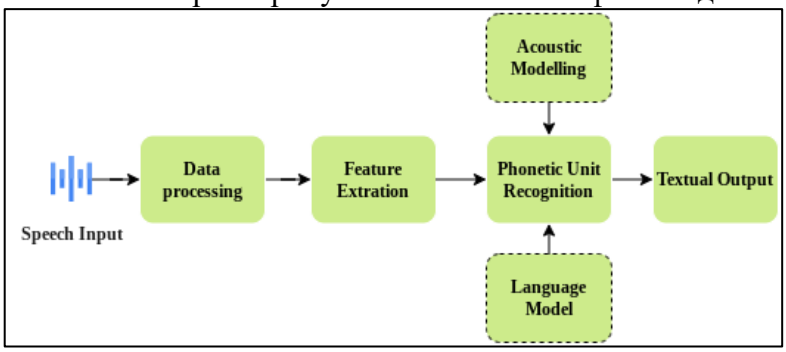


Рисунок 2.3 – Схема роботи Whisper

Whisper інтегрується з Python і бібліотеками машинного навчання, забезпечуючи ефективну серверну частину для вебзастосунків, які потребують транскрибування. Модель може бути розгорнута як локально на сервері (за допомогою CPU або GPU), так і в хмарних середовищах.

Whisper також має функцію автоматичного визначення мови та поділу транскрипту на частини з часовими мітками, що полегшує створення синхронізованих субтитрів або пошук за текстом.

2.1.4 Torchaudio та pyannote.audio для попередньої обробки аудіо та сегментації мовлення

Бібліотеки torchaudio і pyannote.audio є ключовими засобами для підготовки аудіосигналів та сегментації мовлення перед транскрипцією [12, 13].

Torchaudio, створена розробниками PyTorch, є потужним інструментом, який ефективно завантажує, обробляє та перетворює аудіодані. Вона пропонує зручні методи для нормалізації, перетворення в спектрограми, зменшення шумів та зміни частоти дискретизації.

Pyannote.audio, навпаки, зосереджена на діаризації мовлення – тобто визначенні та класифікації сегментів аудіо, що належать різним мовцям. Це особливо корисно для багатомовних чи діалогових аудіозаписів, де важливо встановити, хто говорить що.

Ці інструменти у поєднанні значно покращують якість транскрипцій, створюючи чіткі та розділені сегменти мовлення, які легко піддаються подальшому аналізу.

2.1.5 FFmpeg та ffmpeg-normalize для обробки аудіофайлів

FFmpeg є одним з найпотужніших інструментів для обробки мультимедійних файлів, особливо аудіо [14]. Він надає широкий спектр функцій: зміну формату, обрізку, об'єднання, ресемплінг, конвертацію частоти дискретизації і вирівнювання рівнів гучності. При транскрибуванні важливо мати якісно нормалізований аудіосигнал, щоб запобігти втраті точності при розпізнаванні мовлення.

У цьому процесі допомагає утиліта ffmpeg-normalize, яка автоматично регулює гучність всіх аудіофрагментів до встановленого рівня. Це особливо корисно для обробки аудіо, записаного в різних умовах або з різними мікрофонами, де рівень сигналу може бути неоднорідним. Таким чином, забезпечується стабільне введення для подальшої обробки та транскрипції.

2.1.6 Celery для асинхронної обробки задач транскрибування

У процесі транскрибування аудіофайлів важливо забезпечити ефективну обробку завдань без перешкоджання діяльності основного сервера. Для досягнення цього використовується Celery – потужний фреймворк для асинхронного виконання завдань на Python [11]. Він надає можливість запуску фонового процесу, здатного паралельно обробляти значні обсяги аудіо, зберігати результати і оновлювати статус завдання в базах даних.

На рис. 2.4 зображено архітектуру Celery.

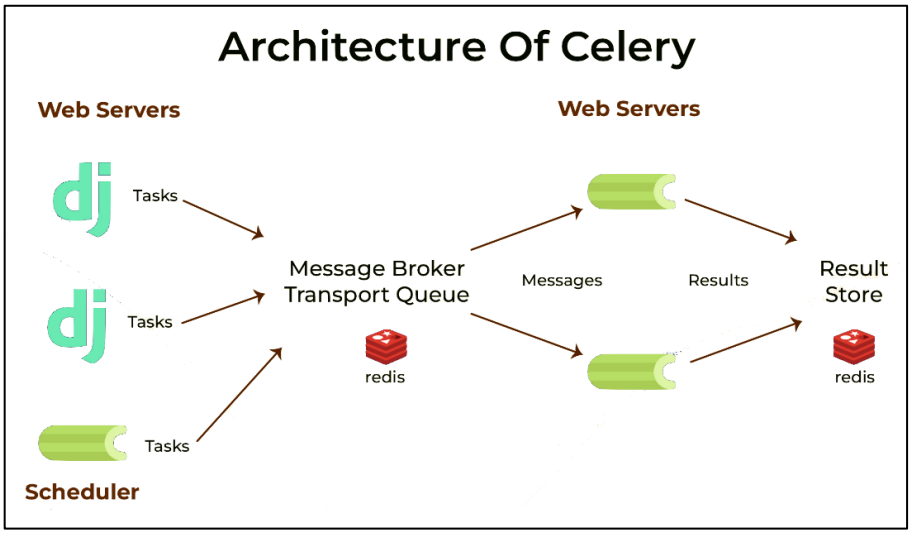


Рисунок 2.4 – Робочий процес Celery

У типовій ситуації користувач за допомогою вебінтерфейсу завантажує аудіофайл, після чого завдання передається на обробку до Celery через систему управління чергами повідомлень (таку як Redis або RabbitMQ). Під час виконання транскрибування у фоновому режимі клієнтський інтерфейс здатний показувати поточний статус завдання, без вимог до очікування завершення обробки.

Цей підхід оптимізує масштабованість системи, забезпечуючи високу ефективність під час обробки великої кількості запитів одночасно.

2.1.7 Flask-CORS для взаємодії з фронтендом

У сучасних вебзастосунках фронтенд і бекенд часто розміщуються на різних доменах або портах, що викликає проблему крос-доменного доступу, регульовану політикою безпеки браузерів – CORS (Cross-Origin Resource Sharing). Для того щоб клієнтський застосунок міг взаємодіяти з сервером Flask, використовують розширення Flask-CORS. Схема запиту між фронтендом і бекендом із CORS-заголовками зображено на рис. 2.5.

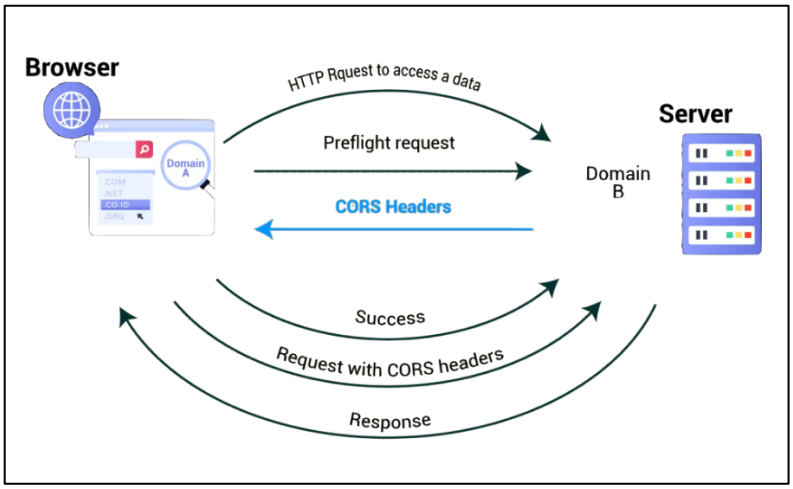


Рисунок 2.5 – Схема запиту із CORS-заголовками

Flask-CORS автоматично додає необхідні HTTP-заголовки до відповідей сервера, що дозволяє фронтенду безпечно надсилати запити, отримувати відповіді та обробляти помилки. Це надзвичайно важливо під час розробки односторінкових застосунків (SPA) або при використанні сучасних JavaScript-фреймворків, таких як React, Vue чи Angular, які функціонують окремо від API.

Конфігурація Flask-CORS є гнучкою: можна дозволити або обмежити доступ

лише до певних маршрутів, методів або доменів, забезпечуючи баланс між безпекою та функціональністю.

2.2 Технології frontend-у

Frontend є вкрай важливою частиною вебзастосунку, оскільки він надає користувачеві доступ до взаємодії із системою транскрибування. Основне завдання фронтенду полягає в розробці зручного, інтуїтивного і адаптивного інтерфейсу, що дозволяє завантажувати аудіофайли, відслідковувати процес транскрибування, отримувати текстові результати та керувати історією обробок.

Схема взаємодії фронтенду з бекендом відображена на рис. 2.6.

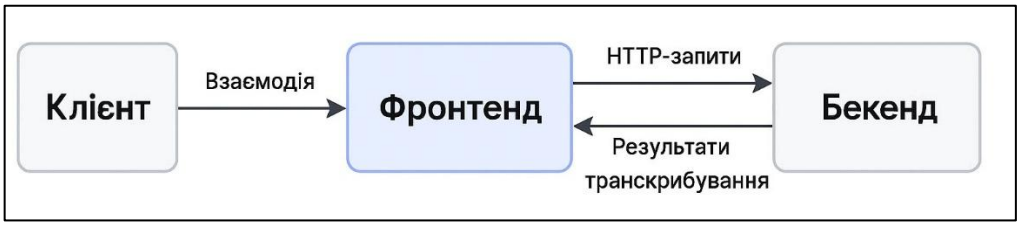


Рисунок 2.6 – Проста схема взаємодії фронтенду з бекендом

Приблизний макет інтерфейсу користувача для платформ ПК та мобільних пристроїв наведено на рис. 2.7.

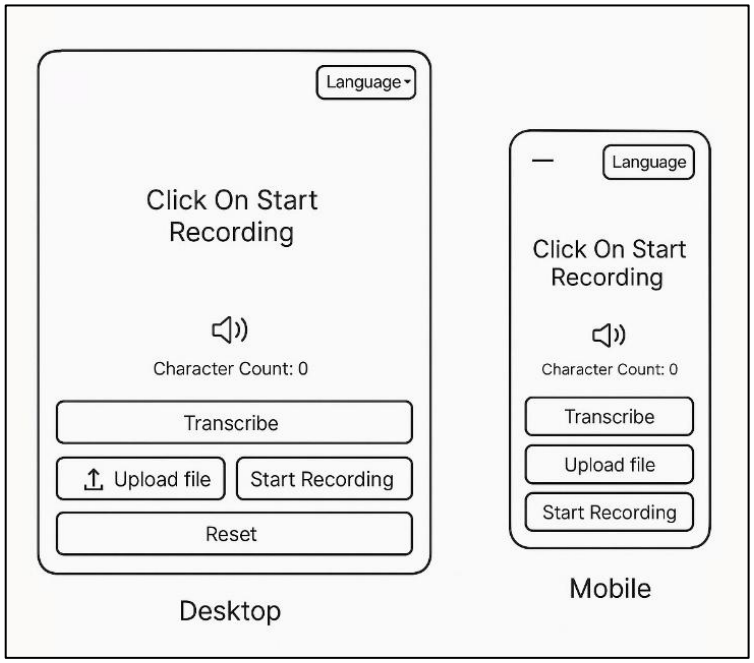


Рисунок 2.7 – Макет інтерфейсу користувача

Для реалізації клієнтської частини зазвичай використовують сучасні вебтехнології, такі як HTML, CSS3 та JavaScript, разом із фреймворками на зразок React або Vue.js. Важливо, що фронтенд забезпечує динамічну взаємодію з бекендом через HTTP-запити.

Ключові функції інтерфейсу включають підтримку перетягування файлів, візуалізацію прогресу транскрибування, індикацію помилок і повідомлень, а також забезпечення доступності для різних типів пристроїв. В табл. 2.3 описано основні функціональні блоки фронтенду вебзастосунку для транскрибування аудіо.

Таблиця 2.3 – Опис основних функціональних блоків фронтенду вебзастосунку для транскрибування аудіо

Функціональний блок	**Опис**	**Основні елементи UI**
Завантаження	Дає змогу користувачеві обрати та завантажити аудіофайл для обробки.	Кнопка «Оберіть файл», поле drag & drop.
Прогрес	Відображає стан обробки завантаженого файлу у реальному часі.	Прогрес-бар, індикатори статусу
Результат	Показує транскрибований текст після завершення обробки.	Вікно з текстом
Збереження	Можливість зберегти вихідний текст користувачем	Кнопка «Зберегти»

Функціональні блоки, представлені в системі, відіграють вирішальну роль у забезпеченні комфорту та ефективності користування. Чітке розмежування цих блоків сприяє покращенню структури інтерфейсу, полегшуючи навігацію для користувача. Завдяки такій організації, користувач може без труднощів завантажити файл, відстежити процес його обробки, ознайомитися з результатами та зберегти їх.

2.2.1 React і TypeScript як основа інтерфейсу

У сучасних веб-додатках важливо застосовувати технології, які є гнучкими, ефективними та здатними підтримувати розвиток.

В даному проєкті для експертної реалізації клієнтської частини було обрано React у поєднанні з TypeScript [15, 16].

React, як бібліотека JavaScript, створена корпорацією Meta, надає можливість будувати інтерактивні, компонентно-орієнтовані користувальницькі інтерфейси, що легко масштабуються та підтримуються.

TypeScript слугує надбудовою над JavaScript, надаючи статичну типізацію, яка оптимізує процес налагодження коду та сприяє ефективній командній співпраці.

Застосування React разом із TypeScript дозволяє розробити інтерфейс, що відзначається надійністю і гнучкістю у розширенні. Типізація допомагає уникати помилок ще на стадії розробки, тоді як компонентна структура React забезпечує повторне використання певних логічних елементів і стилів.

У межах цього проєкту React-компоненти реалізують основні елементи інтерфейсу: завантаження аудіофайлів, відображення прогресу транскрибування, демонстрацію результатів та історію оброблених аудіозаписів.

В табл. 2.4 демонструються переваги використання React + TypeScript у порівнянні з іншими популярними фронтенд-технологіями.

Таблиця 2.4 – Переваги використання React + TypeScript

Критерій	**React + TypeScript**	**Plain JavaScript**	**Vue.js**	**Angular**
Типізація	✓	×	!	✓
Крива навчання	!	✓	✓	×
Гнучкість	✓	✓	!	×

Продовження таблиці 2.4

Критерій	React + TypeScript	Plain JavaScript	Vue.js	Angular
Компонентна модель	✓	!	✓	✓
Інструменти розробника	✓	!	✓	✓
Сумісність і екосистема	✓	✓	!	✓
Тестування	✓	!	✓	✓

Крім того, TypeScript спрощує взаємодію з API через чітке визначення типів відповідей сервера. Це суттєво зменшує ймовірність помилок при обміні даними між клієнтом і сервером, що є критично важливим для систем із високими вимогами до надійності.

Схема компонентної структури інтерфейсу з ключовими елементами React відображена на рис. 2.8.

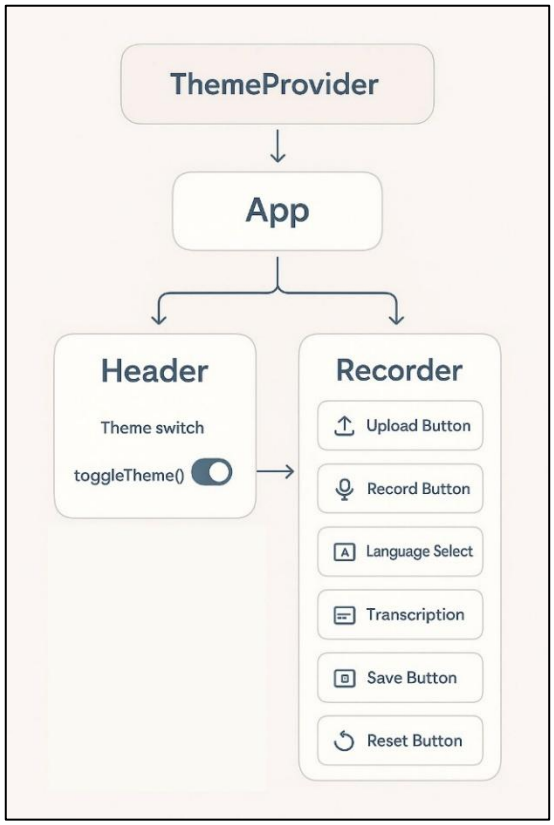


Рисунок 2.8 – Схема компонентної структури інтерфейсу з ключовими елементами React

Компоненти відповідають за керування темою, вибором мови інтерфейсу, за обгортку всього інтерфейсу та за кнопки управління: завантаження аудіофайлу, запуск транскрибування.

2.2.2 Styled-components і Tailwind CSS для стилізації

У цьому проєкті застосовуються дві сучасні технології для стилізації інтерфейсу користувача, а саме Styled-components та Tailwind CSS [17]. Їх інтеграція забезпечує гнучкість, повторне використання стилів та швидку розробку інтерфейсу.

Styled-components представляє собою бібліотеку для стилізації компонентів у React, яка ґрунтується на підході CSS-in-JS. Ця технологія дозволяє розробникам створювати стилі безпосередньо у JavaScript або TypeScript файлах, асоціюючи їх із певними компонентами. Таке рішення зменшує ймовірність виникнення конфліктів класів, пропонує покращену підтримку тем та спрощує логіку умовного рендерингу стилів, наприклад, залежно від тематики чи стану компонента.

Основними перевагами Styled-components є:

– інкапсуляція стилів у межах компонентів;
– підтримка тем через ThemeProvider;
– динамічне налаштування стилів за допомогою параметрів (props);
– повна інтеграція з TypeScript для типізованої розробки.

Tailwind CSS виступає утилітарним CSS-фреймворком, який надає набір готових класів для створення адаптивного та сучасного дизайну, що дозволяє уникнути написання власного CSS. У даному проєкті Tailwind застосовується для швидкої розробки інтерфейсу, особливо для компонентів, які не потребують складної логіки стилізації, як от кнопки, контейнери або вирівнювання елементів.

Tailwind CSS надає можливість:

– значно скоротити обсяг написаного CSS-коду;
– забезпечити послідовний дизайн інтерфейсу;
– швидко створювати прототипи та візуальні рішення;

– легко адаптувати інтерфейс під різні пристрої завдяки вбудованій підтримці адаптивного дизайну.

Комбінування Styled-components для компонентної стилізації з підтримкою тем та Tailwind CSS для швидкого створення макету забезпечує ефективну реалізацію користувацького інтерфейсу з високим рівнем контрольованості та гнучкості.

2.2.3 React-Speech-Recognition для інтеграції з мікрофоном

Для впровадження голосового введення в інтерфейсі була застосована бібліотека React-Speech-Recognition, яка служить обгорткою для Web Speech API [18]. Вона забезпечує зручний доступ до функцій розпізнавання мови без необхідності занурення в низькорівневі API браузера. Це дозволяє легко інтегрувати можливість перетворення голосу на текст безпосередньо в React-додаток.

Однією з основних переваг цієї бібліотеки є її сумісність з React-компонентами та підтримка хука useSpeechRecognition(), який дозволяє отримувати поточний текст, статус прослуховування та інші параметри. У нашому проекті цей функціонал може бути розширено для команд голосового управління або автозаповнення текстового поля під час запису.

Бібліотека підтримує кілька мов розпізнавання, управління паузами, автоматичне завершення розпізнавання і можливість почати прослуховування за подією, наприклад, натисканням кнопки "Start Recording". Це створює комфортний та сучасний досвід взаємодії користувача з інтерфейсом.

Завдяки використанню React-Speech-Recognition вдалося реалізувати зручний, доступний і ефективний спосіб голосового введення, що є особливо важливим для мобільних користувачів або людей із обмеженими можливостями.

2.2.4 Lucide-react для іконок

У процесі розробки інтерфейсу користувача було обрано бібліотеку Lucide-react для інтеграції сучасних і адаптивних SVG-іконок [19]. Ця бібліотека є

відкритою версією популярного набору Feather Icons, але вирізняється частішими оновленнями та ширшим асортиментом іконок, оптимізованих для використання у React-середовищі.

Lucide-react забезпечує простоту імпортування іконок як React-компонентів. Наприклад, компоненти на кшталт <Mic />, <Save />, <Upload /> використовуються в програмі для представлення функцій запису, збереження та завантаження аудіофайлів. Такий підхід до роботи з компонентами створює високу гнучкість у стилізації, дозволяючи змінювати колір, розміри та анімацію за допомогою стандартних методів CSS або styled-components.

Однією з ключових переваг бібліотеки Lucide є її компактність і відсутність зовнішніх залежностей, що сприяє швидшому завантаженню інтерфейсу. Крім того, всі іконки виконані у форматі SVG, що дає змогу масштабувати їх без втрати якості – це особливо важливо для побудови адаптивного дизайну.

Використання єдиної бібліотеки для набору іконок забезпечує стильову єдність всього інтерфейсу, покращує сприйняття користувачем представлених дій та загальне враження від взаємодії із додатком.

Висновки до розділу 2

У другому розділі було детально розглянуто архітектуру та реалізацію інтерфейсної частини вебдодатку, призначеного для обробки голосових даних. Аналіз почався з функціонального поділу інтерфейсу на основні блоки: завантаження файлу, прогрес обробки, результат транскрипції та історія. Такий підхід забезпечує логічну структуру та зручну взаємодію користувача з додатком.

Було обґрунтовано вибір React як основного фреймворку завдяки його компонентному підходу, ефективній обробці стану та широкій екосистемі. TypeScript додатково підвищив надійність і масштабованість коду за рахунок статичної типізації. У підрозділі 2.2.1 було продемонстровано приклад структури компонентів (App, Header, Recorder), яка сприяє чіткому розділенню відповідальностей.

Для стилізації застосовано комбінований підхід – Styled-components для

гнучкої стилізації компонентів із підтримкою тем, та Tailwind CSS для швидкого прототипування класами. Це дозволило ефективно реалізувати адаптивний дизайн та підтримку темної/світлої тем.

У підрозділі 2.2.3 розглянуто використання React-Speech-Recognition для інтеграції мікрофона, що відкриває можливості для роботи з живою мовою, а не лише з файлами. Нарешті, завдяки Lucide-react інтерфейс отримав сучасні SVG-іконки, які посилили візуальну інтуїтивність функцій.

На рис. 2.9 зображено архітектуру клієнт-серверного вебзастосунку для обробки аудіо та транскрипції.

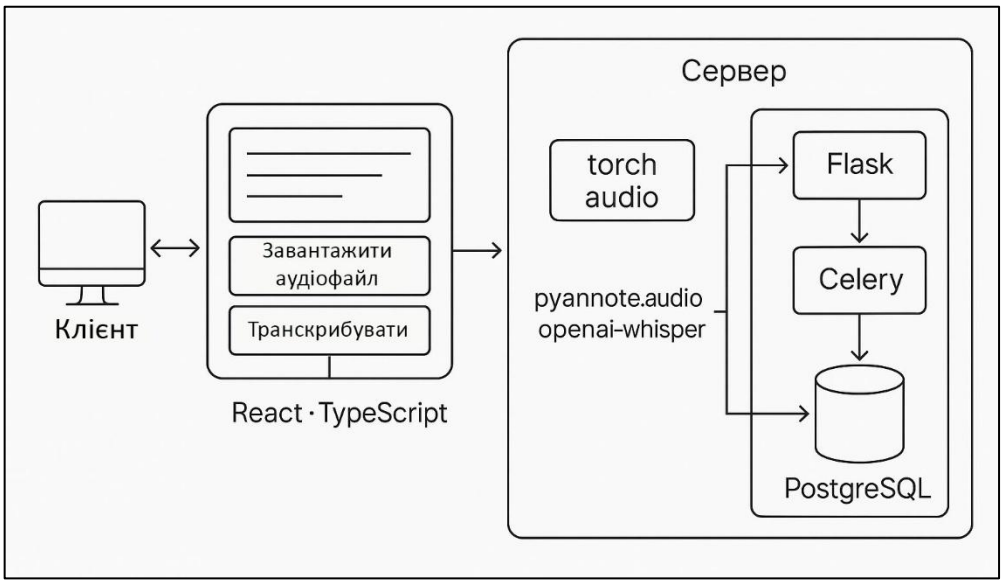


Рисунок 2.9 – Архітектурна схема вебзастосунку

Загалом реалізація фронтенду базується на сучасних, перевірених технологіях, що забезпечують масштабованість, зручність підтримки та якісний користувацький досвід. Така структура дозволяє розширювати додаток новими можливостями без шкоди стабільності.

3 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ТА АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

3.1 Опис вхідних даних та структура системи

3.1.1 Типи вхідних даних

У сучасних інформаційних системах для транскрибування мовлення вхідні дані відіграють вирішальну роль у процесі обробки. Якість, формат, тривалість та інші параметри вхідного аудіо не лише впливають на точність транскрипції, але й визначають загальну стабільність роботи системи. Тому на етапі проектування надзвичайно важливо ретельно проаналізувати можливі типи вхідних даних, які можуть використовуватися кінцевими користувачами, і забезпечити їх коректне опрацювання в різних сценаріях.

Очікується, що користувачі системи можуть надавати аудіоінформацію у трьох основних формах: завантаження файлів з локальних пристроїв, запис голосу в реальному часі через мікрофон, а також надання посилань на зовнішні джерела аудіо. Останній метод – це майбутня функція вебсервісу. На рис. 3.1 відображена проста схема двох реалізованих основних шляхів, якими аудіо може потрапити до інформаційної системи. Кожен з цих методів має свої специфічні особливості, переваги та технічні виклики, що вимагає відповідної реалізації як на серверній, так і на клієнтській стороні ПЗ.

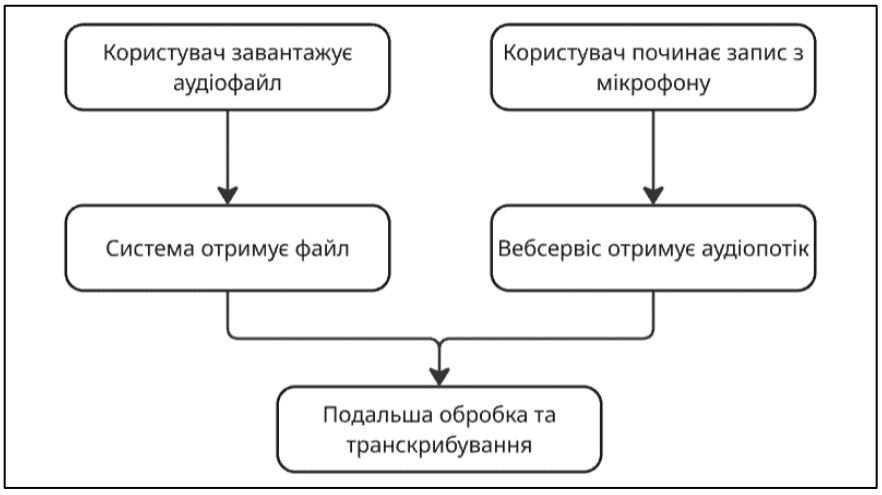


Рисунок 3.1 – Шляхи надходження аудіоданих до системи

Завантаження аудіофайлів із локальних пристроїв є найпоширенішим способом. Це дозволяє працювати з уже записаними матеріалами, такими як інтерв'ю, конференції або лекції. Завантаження відбувається через вебінтерфейс системи, після чого файли передаються на сервер для перевірки та подальшої обробки. Важливим аспектом є підтримка основних форматів аудіофайлів: WAV, MP3, M4A, OGG, FLAC тощо. В табл. 3.1 зазначено основні характеристики підтримуваних форматів аудіофайлів. Формати WAV і FLAC зазвичай застосовуються для збереження аудіо без втрат якості, тоді як MP3 і M4A є стисненими і сприяють економії пам’яті, але можуть призводити до втрати деталей мовлення, що ускладнює розпізнавання.

Таблиця 3.1 – Підтримувані формати аудіофайлів із характеристиками

Формат	**Розширення**	**Тип стиснення**	**Підтримка системою**	**Особливості**
WAV	.wav	Без стиснення	Повна	Висока якість, великий розмір, ідеальний для обробки мовлення
MP3	.mp3	З втратами	Повна	Популярний формат, менший розмір, можлива втрата якості
FLAC	.flac	Без втрат	Часткова	Висока якість, збереження оригінального звуку, більший розмір
AAC	.aac	З втратами	Часткова	Підвищена ефективність стиснення порівняно з MP3
OGG	.ogg	З втратами	Часткова	Вільний формат, підтримка залежить від ПЗ
M4A	.m4a	З втратами/без	Часткова	Використовується в Apple-пристроях, іноді має захист DRM
WEBM	.webm	З втратами	Часткова	Часто використовується у веб-середовищі, може містити аудіо/відео
AMR	.amr	З втратами	Обмежена	Оптимізований для запису мови, використовується в мобільних пристроях

Реалізація запису аудіо через мікрофон відбувається за допомогою сучасних браузерних API для обробки мультимедійного контенту. Це особливо корисно для

користувачів, які бажають швидко надати голосову інформацію без попередньої підготовки файлу. У таких випадках система забезпечує або потокову передачу даних на сервер, або буферизоване збереження до завершення запису. Подальший аналіз подібних даних здійснюється аналогічно до обробки локальних файлів, хоча потік зазвичай перетворюється у стандартний формат для подальшої сумісності із вибраною моделлю транскрипції.

Обробка аудіо, отриманого з зовнішніх ресурсів за URL-посиланням, є ще одним важливим аспектом. Такий підхід часто використовується для транскрибування подкастів чи відеозаписів з YouTube. Система повинна мати засоби для попереднього аналізу та завантаження звукового потоку з ресурсів та подальшої його обробки у стандартному форматі. Це завдання ускладнюється юридичними та технічними аспектами, включаючи питання авторських прав і ефективне витягнення звуку з відео, що робить цю функцію додатковою та реалізується лише за наявності відповідної інфраструктури.

На додачу до форм подачі, система повинна враховувати технічні характеристики аудіо. Частота дискретизації є ключовим параметром, адже для сучасних моделей розпізнавання мовлення рекомендується частота не нижче 16 кГц. Низькі значення частоти призводять до погіршення якості сигналу та зниження точності розпізнавання. Окрім того, важливе значення має кількість каналів: більшість моделей орієнтована на монофайли. У разі стереоформатів система має бути здатною автоматично конвертувати їх, щоб запобігти потенційним конфліктам.

Тривалість аудіозапису становить значущий параметр в опрацюванні даних. На одному кінці спектра користувачі можуть подавати стислі аудіофайли тривалістю лише декілька секунд, тоді як на іншому кінці розташовані великі записи, що тривають більше години. У таких випадках виникає потреба в реалізації механізму, що дозволяє поділ файлу на логічні сегменти для окремої обробки, після чого результати транскрипції повинні бути зведені в цілісний текстовий блок. Система повинна також мати можливість виявляти надмірно великі або порожні

файли, попереджати користувача про неприйнятні розміри чи брак звукового сигналу, і надавати чіткі інструкції щодо їхнього виправлення.

Таким чином, типовий процес введення даних у системі, що розробляється, охоплює широкий спектр форматів і методів отримання аудіоданих. Урахування всіх можливих сценаріїв подачі інформації та її стандартизація є ключем до забезпечення коректності роботи системи транскрипції, а також її адаптивності та зручності для кінцевого користувача.

3.1.2 Вимоги до аудіоформатів та обмеження

Щоб забезпечити максимальну точність транскрипції мовлення, аудіофайли, що надходять до системи, мають відповідати певним технічним вимогам. Ці вимоги стосуються як формату, так і якості аудіосигналу, оскільки саме від цих параметрів залежить якість розпізнавання мови, швидкість обробки та стабільна робота системи.

Важливо зазначити, що підтримуються лише формати, які можна декодувати за допомогою стандартних бібліотек для обробки аудіо, таких як pydub, ffmpeg, librosa тощо. Обов’язковою умовою є наявність монофонічного каналу (1 канал), оскільки багатоканальні записи ускладнюють попередню обробку і потребують додаткової обчислювальної потужності. Якщо вхідний файл має стереозвук або більше каналів, система автоматично перетворює його на монофон, об'єднуючи канали.

Серед ключових технічних вимог виділяється частота дискретизації. У дослідженні "Automatic Speech Recognition in Noise for Parkinson's Disease: A Pilot Study" порівнюється точність розпізнавання мовлення при частотах дискретизації 8 кГц та 48 кГц [21]. Результати показують, що вищі частоти дискретизації можуть покращити точність розпізнавання, особливо в умовах шуму. Детальні дані представлені у таблиці дослідження.

Рекомендована частота – 16 кГц або 22,05 кГц. Це забезпечує достатній рівень для розпізнавання мовлення, зберігає інформативність сигналу і водночас дозволяє зменшити розмір файлу та навантаження на обчислювальну систему.

Занадто висока частота, як-от 48 кГц, не дає значущих покращень якості, але збільшує обсяг обробки. На рис. 3.2 зображено графік залежності точності транскрибування від частоти шуму (в кГц).

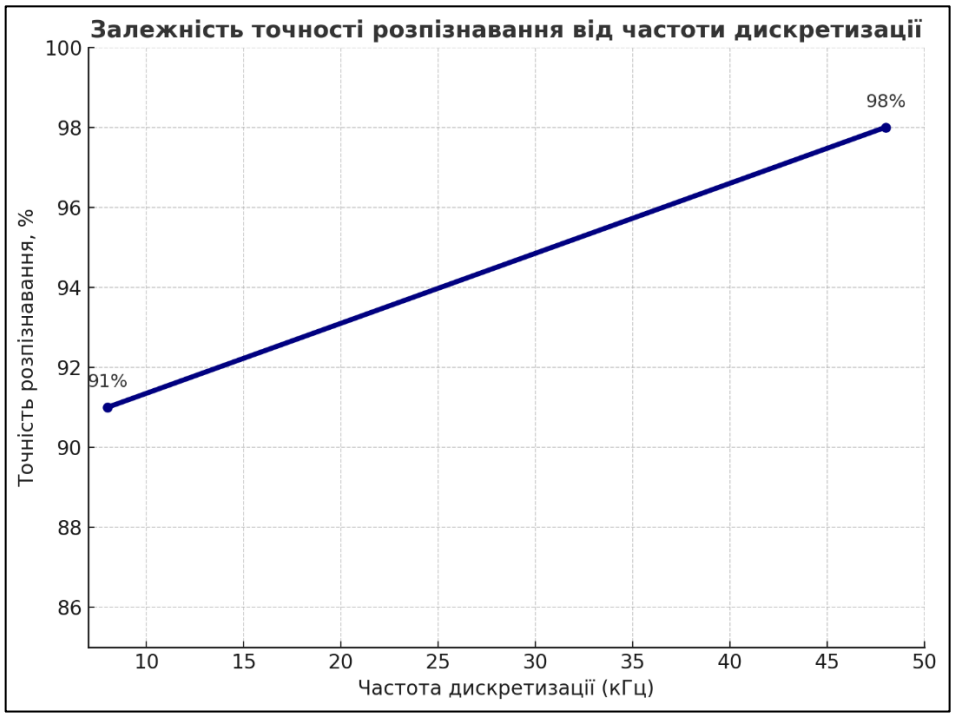


Рисунок 3.2 – Графік залежності точності транскрибування від частоти шуму

Крім того, аудіофайли мають бути заздалегідь очищені від шумів. Фонові звуки, музика, перешкоди, переривання мовлення або одночасне звучання кількох голосів знижують точність розпізнавання. У складних звукових умовах використовується попередня обробка, яка включає фільтрацію шумів, нормалізацію гучності і, при необхідності, сегментацію мовців.

Відношення сигнал/шум (SNR), як правило, є важливим фактором для точності оцінки параметрів. За допомогою моделювання методом Монте-Карло, співвідношення між SNR відлуння-сигналу та помилками оцінки параметрів якісно показано на рис. 3.3. SNR відлуння-сигналу коливається від -30 до 0 дБ. Як видно з рис. 3.3, найкращі значення, які мають хороші протишумові властивості, – це значення, які перевищують -17 дБ.

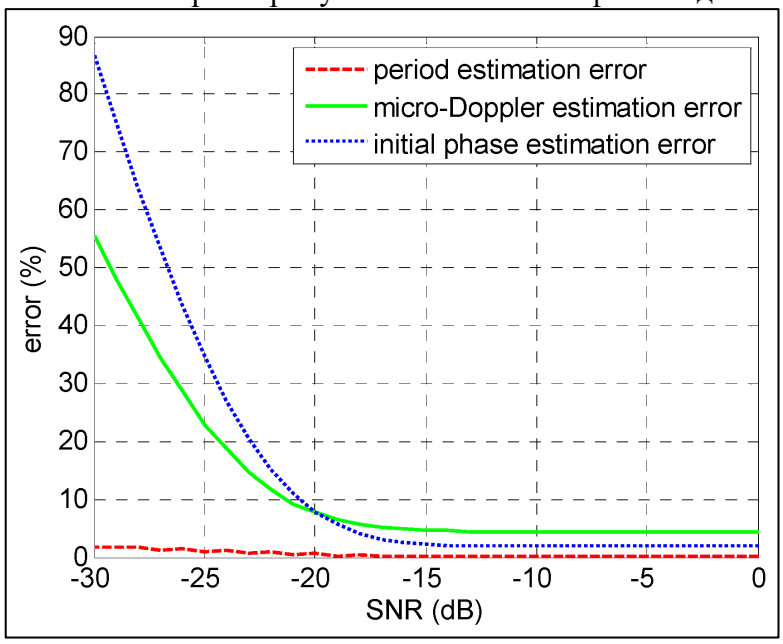


Рисунок 3.3 – Співвідношення сигнал/шум (SNR) та помилками оцінки параметрів

Як зазначається в статті "Getting Started with Audio Data Processing Techniques and Key Challenges, ефективні методи обробки аудіо можуть значно підвищити якість результатів транскрибування, особливо у складних акустичних умовах [20].

Попередня обробка є ключовим початковим етапом в аналізі аудіо. Вона трансформує необроблені, неструктуровані аудіодані у чисті, стандартизовані формати, придатні для подальшого аналізу. Етапи, такі як очищення даних, повторна дискретизація, нормалізація та сегментація, забезпечують готовність даних до використання в надійних моделях штучного інтелекту. На рис. 3.4 зображено графічне представлення аудіоданих без обробки.

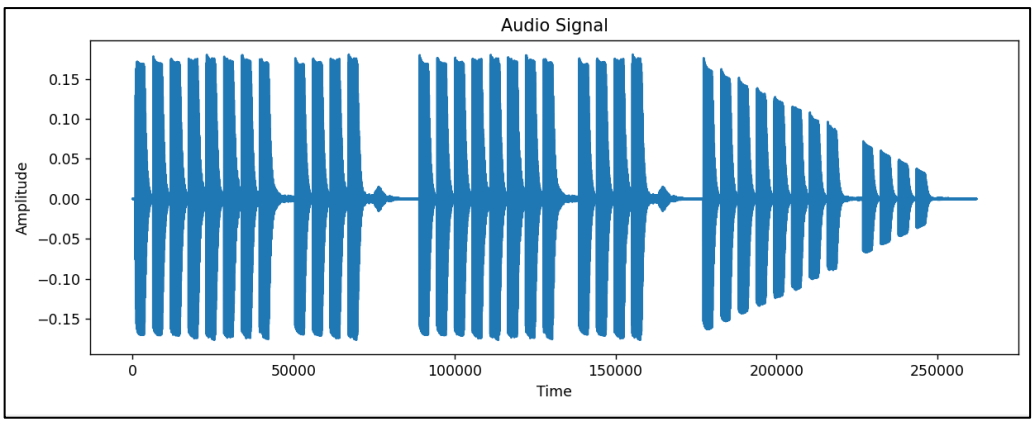


Рисунок 3.4 – Графічне представлення аудіоданих без обробки

Очищення даних передбачає видалення небажаного шуму й артефактів з первинного аудіо, що гарантує найвищу якість звуку. На цьому етапі слід особливо приділити увагу усуненню фонового шуму, клацань та спотворень, які можуть ускладнити подальший аналіз. На рис. 3.5 відображен фрагмент коду для очищення даних з аудіофайлу, а на рис. 3.6 – графічне відображення даних після чистки.

```
waveform, sample_rate = torchaudio.load("example.wav")
y = waveform.numpy()[0]
cleaned_audio = nr.reduce_noise(y=y, sr=sample_rate)
plt.figure(figsize=(10, 6))
plt.plot(cleaned_audio)
plt.title("Cleaned Audio Signal")
plt.xlabel("Time (samples)")
plt.ylabel("Amplitude")
plt.tight_layout()
plt.show()
```

Рисунок 3.5 – Фрагмент коду для очистки даних з аудіофайлу

Наприклад, при роботі з шумним аудіокліпом, його очищення дозволяє зосередитися на основних особливостях аудіо без відволікаючих факторів. Застосовуючи техніки шумозаглушення, ми покращуємо співвідношення сигнал/шум і підвищуємо чіткість звучання для таких завдань, як розпізнавання мовлення або музичний аналіз.

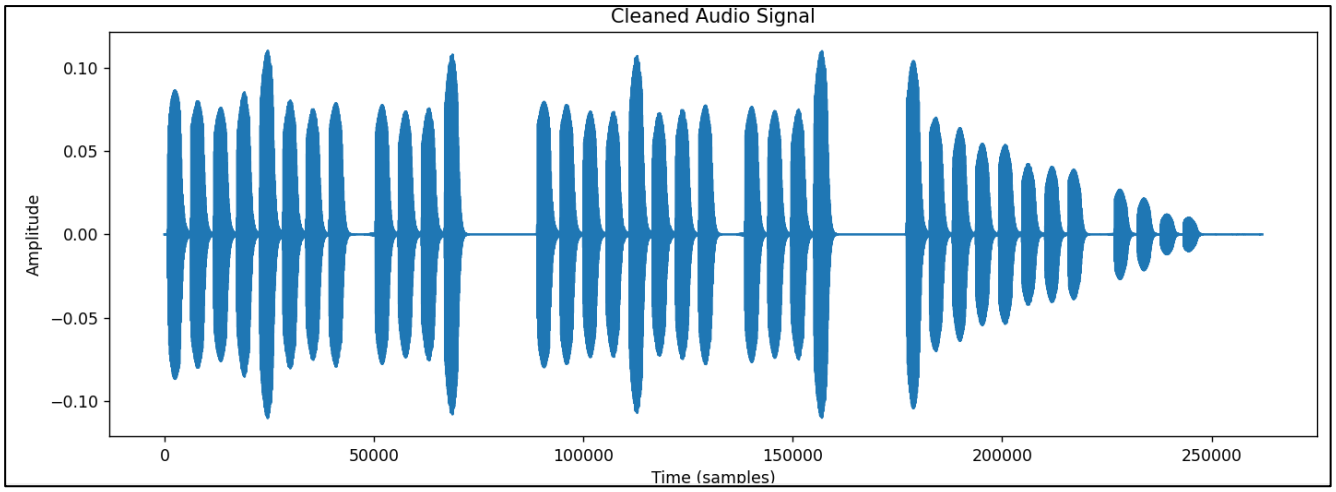


Рисунок 3.6 – Графічне відображення аудіоданих після очистки даних

Передискретизація є процесом коригування частоти дискретизації аудіосигналу з метою забезпечення сумісності між різними наборами даних. Це має особливе значення при роботі з аудіоінформацією, отриманою з різних джерел, які можуть характеризуватися різними частотами дискретизації. На рис. 3.7 відображен фрагмент коду для передискретизації файлу, а на рис. 3.8 – графічне відображення даних після цього процесу.

```python
import torchaudio
import scipy.signal
import matplotlib.pyplot as plt

waveform, sample_rate = torchaudio.load("example.wav")
audio_np = waveform.numpy()[0]
sos = scipy.signal.butter(10, 3000, btype='lowpass', fs=sample_rate, output='sos')
filtered = scipy.signal.sosfilt(sos, audio_np)
plt.figure(figsize=(10, 4))
plt.plot(filtered)
plt.title("Resampled Audio Signal")
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.tight_layout()
plt.show()
```

Рисунок 3.7 – Фрагмент коду з передискретизації аудіоданих

Наприклад, коли деякі аудіокліпи записані з частотою 44,1 кГц, а інші — з частотою 22,05 кГц, доцільно здійснити передискретизацію до стандартної частоти, такої як 22,05 кГц, для забезпечення однорідності даних у всьому наборі. Передискретизація забезпечує узгоджену обробку всіх аудіофайлів, що має вирішальне значення при виконанні задач, таких як екстракція ознак або класифікація.

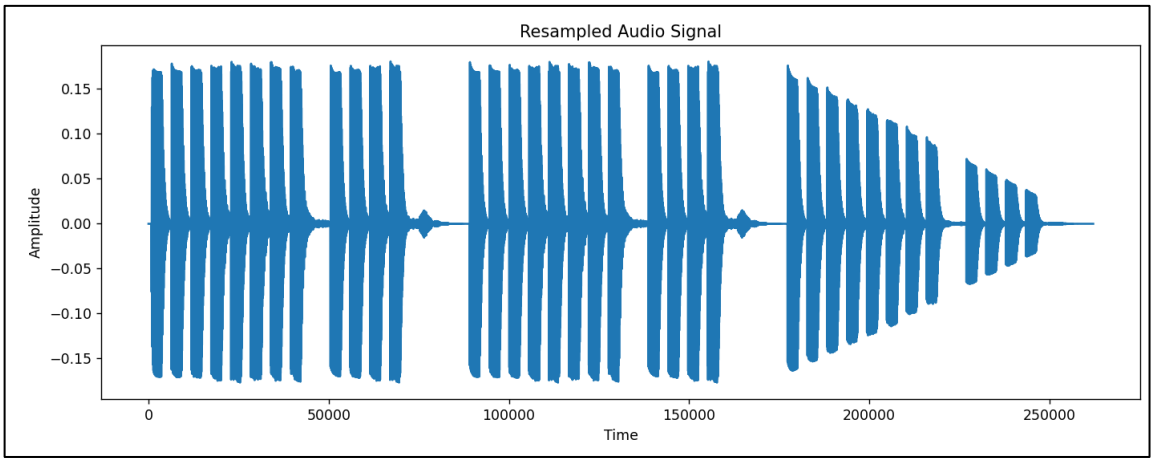


Рисунок 3.8 – Графічне відображення даних після передискретизації

Нормалізація — це метод, який дозволяє оптимізувати амплітуду аудіосигналу, приводячи її до стандартного діапазону, найчастіше від -1 до 1. Цей процес забезпечує баланс між надмірною гучністю, яка може спричинити кліпінг, і надмірною тишею, що погіршує якість звучання. На рис. 3.9 відображен фрагмент коду з нормалізацією аудіоданих. Це один із ключових кроків при роботі з аудіофайлами різних джерел, адже рівень їхньої гучності може значно відрізнятися.

```python
import torch
import torchaudio
import matplotlib.pyplot as plt

waveform, sample_rate = torchaudio.load("example.wav")
waveform_normalized = waveform / waveform.abs().max()
plt.figure(figsize=(10, 6))
plt.plot(waveform_normalized[0].numpy())
plt.title("Normalized Audio Signal")
plt.xlabel("Time (samples)")
plt.ylabel("Amplitude")
plt.tight_layout()
plt.show()
```

Рисунок 3.9 – Фрагмент коду з нормалізацією аудіоданих

Застосування нормалізації дозволяє досягти однаковості гучності в усіх аудіокліпах, що особливо корисно у створенні списків відтворення чи підготовці матеріалів для подальшого використання. На рис. 3.10 зображено графічне відображення аудіоданих після нормалізації.

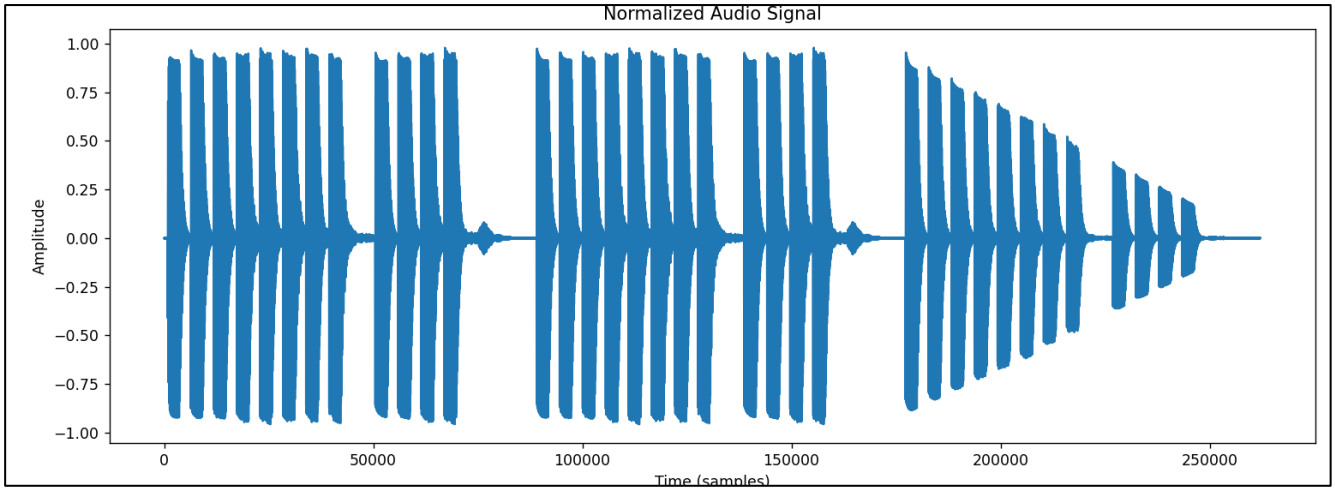


Рисунок 3.10 – Графічне відображення аудіоданих після нормалізації

Система також обмежує розмір вхідного файлу. Для стабільної роботи не рекомендується завантажувати файли довжиною більше 30 хвилин або обсягом понад 100 МБ. У випадку великих записів автоматично застосовується поділ на сегменти з поетапним транскрибуванням, що допомагає уникнути перевантаження пам'яті.

Отже, дотримання вимог до формату, якості та розміру аудіо гарантує ефективну та надійну роботу системи транскрипції. Всі обмеження і вимоги формалізуються через відповідний модуль валідації під час завантаження файлу.

3.1.3 Загальна архітектура інформаційної системи

Архітектура інформаційної системи для транскрибування аудіо базується на принципах модульності, масштабованості та розширюваності. Завдяки такому підходу система легко адаптується до змін у вимогах, обсягах даних або технологічній платформі.

Основні компоненти архітектури включають модуль введення аудіо, блок попередньої обробки, ядро транскрипції, модуль діаризації мовців, систему зберігання результатів і користувацький інтерфейс. Кожен із цих елементів виконує окрему функцію, забезпечуючи можливість гнучкого оновлення або заміни модулів без впливу на загальну працездатність системи.

На початковому етапі система отримує аудіофайли з трьох основних джерел: завантаження локальних файлів, запис через мікрофон або посилання на URL. Модуль введення перевіряє відповідність формату, частоти дискретизації та інших технічних параметрів, готуючи дані для подальшої обробки.

Аудіосигнал надходить до модуля попередньої обробки, де проводиться нормалізація рівня гучності, усунення шумів і, за необхідності, сегментація довгих файлів на окремі частини. Завдяки цьому сегменти можуть оброблятися одночасно, що сприяє підвищенню продуктивності системи.

Основним компонентом є модуль транскрипції, який тісно взаємодіє з нейронними мережами. Ці моделі штучного інтелекту перетворюють усне мовлення на текстовий формат. Згенерований текст синхронізується з часовими

мітками, а в разі потреби проходить подальшу обробку через модуль діаризації для визначення різних мовців.

На рис. 3.11 відображено графік часу обробки аудіофайлу розміром 50 МБ, що відповідає приблизно 30 хвилинам аудіо у форматі WAV із частотою дискретизації 16 кГц і 16-бітною глибиною, залежно від кількості паралельних потоків.

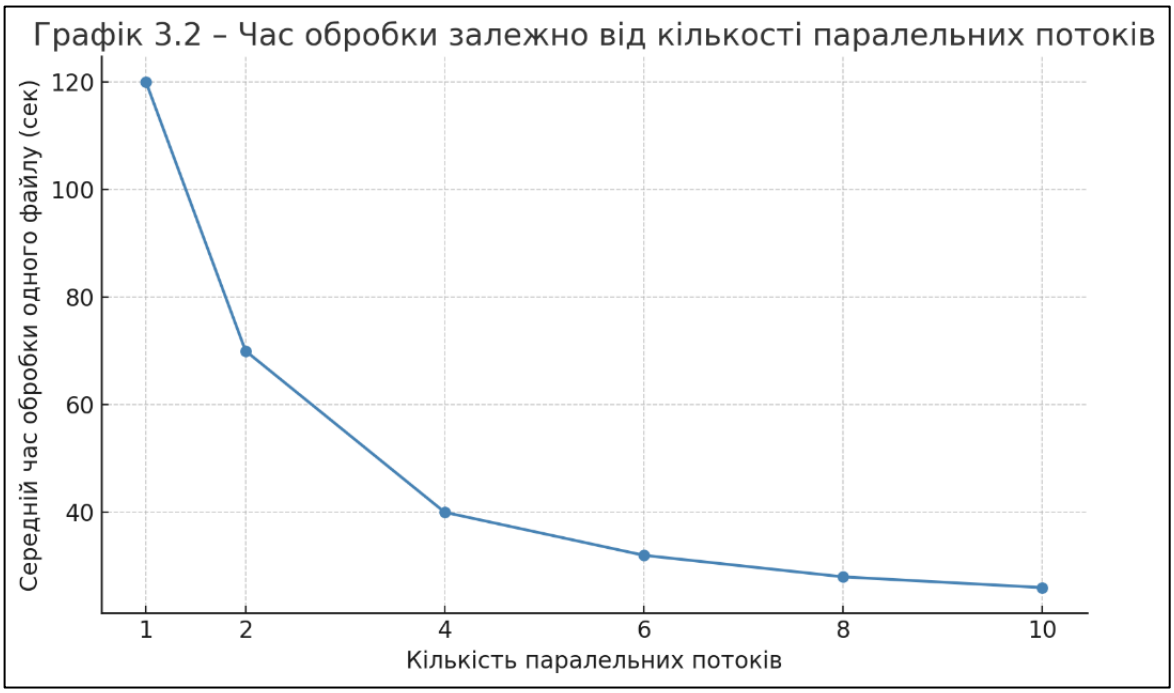


Рисунок 3.11 – Час обробки аудіофайлу залежно від кількості паралельних потоків

Завершений процес транскрипції призводить до збереження результатів у базі даних, звідки їх можна переглянути через вебінтерфейс або експортувати у форматах TXT, DOCX, JSON і подібних.

3.2 Проєктування та моделювання

3.2.1 Діаграма класів для ключових сутностей

У процесі розробки інформаційної системи для транскрибування аудіофайлів одним із ключових етапів є створення діаграми класів, яка відображає основні сутності, їх атрибути та взаємозв’язки.

Цей елемент об’єктно-орієнтованого аналізу забезпечує краще розуміння структури програмної системи, її логічну цілісність, масштабованість і легкість підтримки. В табл. 3.2 короткий опис основних класів системи, які детально будуть розглянуті пізніше.

Таблиця 3.2 – Опис основних класів системи

Назва класу	Призначення	Ключові атрибути
`AudioFile`	Представлення аудіофайлу	`id`, `name`, `format`, `duration`, `status`
`TranscriptionTask`	Обробка транскрипції	`taskId`, `status`, `startTime`, `error`
`User`	Користувач системи	`userId`, `email`, `role`
`AudioSegment`	Частина аудіофайлу	`segmentId`, `startTime`, `transcriptText`
`Recognizer`	Модуль розпізнавання	`transcribe()`, `mergeResults()`
`ErrorLog`	Запис помилок	`errorId`, `message`, `timestamp`

Головною сутністю виступає клас AudioFile, що відповідає за представлення файлу, завантаженого користувачем або записаного системою. Його атрибути включають id, name, format, duration, uploadDate, status і sourceType (наприклад, локальний файл, мікрофон чи URL). Для цього класу можуть бути передбачені методи перевірки валідності формату файлу та ініціалізації процесу його обробки.

Клас AudioFile має зв’язок із класом TranscriptionTask, який управляє процесом обробки файлу. Серед його атрибутів — посилання на відповідний аудіофайл, статус обробки (queued, processing, done, error), часові мітки, помилки, що виникли під час транскрипції, та результат у вигляді тексту. Завдяки цьому класу можна зручно відслідковувати виконання завдань і організовувати чергу їхньої обробки.

Окремою сутністю є клас User, котрий представляє користувача системи. Він може завантажувати файли, виконувати транскрипцію та переглядати результати. Основні атрибути включають userId, email, passwordHash і role (звичайний користувач або адміністратор). Між класами User і AudioFile існує взаємозв'язок один-до-багатьох, що дозволяє одному користувачеві мати доступ до кількох аудіофайлів.

Для обробки розпізнавання мовлення використовується спеціалізований клас Recognizer, який виступає як інтерфейс до зовнішніх або вбудованих API. Цей клас надає методи, зокрема transcribeSegment(segment) та mergeResults(results), які забезпечують поетапну обробку сегментів великого аудіофайлу, дозволяючи ефективно розподіляти процес розшифрування.

Ключову підтримувальну функцію виконує клас AudioSegment, який містить детальну інформацію про відповідний фрагмент аудіо, включаючи початковий та кінцевий час (startTime, endTime), унікальний ідентифікатор сегмента (segmentId) та текст транскрипції (transcriptText).

Завдяки даному класу забезпечується можливість паралельної обробки великих файлів, а також зручність у повторній обробці окремих сегментів у випадку виникнення помилок або неточностей.

Також у системі передбачено клас ErrorLog, який фіксує критичні помилки, що можуть виникати протягом виконання операцій. На основі записів в цьому журналі адміністратор може проводити діагностику роботи системи й покращувати її стабільність, що сприяє забезпеченню надійності та безперервності процесу.

Рис. 3.12 відображає основні класи backend-системи, їхні атрибути та методи, а також зв'язки між цими класами.

Узаємозв'язки між цими сутностями відображають основну логіку функціонування системи. Користувач (User) володіє одним або декількома аудіофайлами (AudioFile), кожен з яких складається з одного чи кількох сегментів (AudioSegment). Кожен сегмент проходить через Recognizer, створюючи завдання транскрипції (TranscriptionTask). У разі виникнення помилки відповідний запис автоматично додається до ErrorLog для подальшого аналізу та корекції.

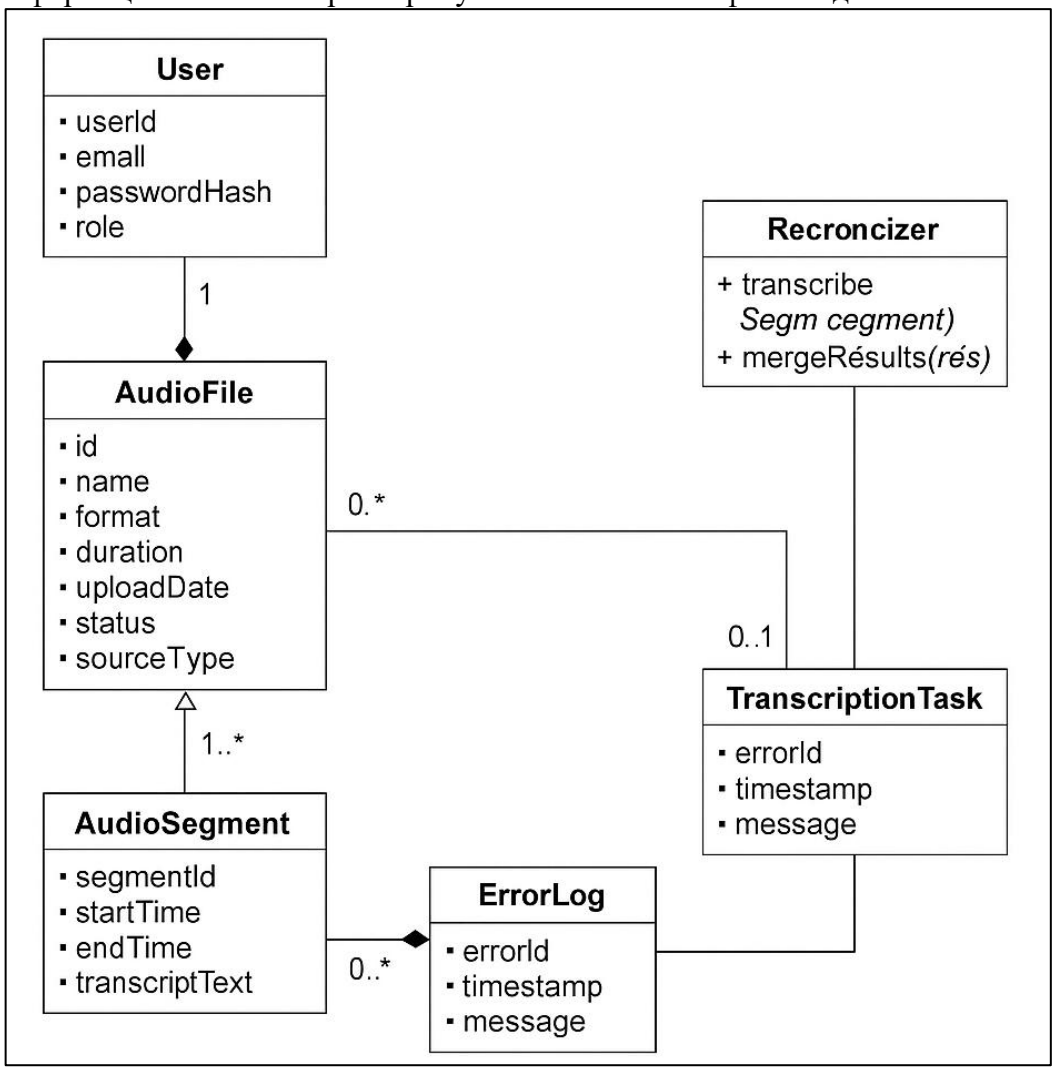


Рисунок 3.12 – Діаграма класів ключових сутностей

Представлена модель забезпечує високоефективне управління процесом транскрипції аудіофайлів – від моменту завантаження файлу до отримання результатів. Вона добре адаптована до масштабування, оскільки підтримує одночасну обробку багатьох файлів для різних користувачів, сприяючи оптимізації робочих процесів і підвищенню загальної продуктивності системи.

3.2.2 Схема бази даних

Для забезпечення ефективного зберігання, обробки та управління даними, що використовуються у процесі транскрибування аудіо, інформаційна система має мати ретельно спроєктовану та структуровану базу даних. Навіть у тих випадках, коли система не передбачає довготривалого збереження даних користувачів,

наявність бази даних є критичною для організації тимчасових сесій, відстежування історії запитів, управління чергами обробки та ведення журналів операцій.

Концептуальна модель бази даних базується на реляційному підході і включає кілька ключових таблиць, таких як Users, Audio, Transcriptions. Структура бази забезпечує гнучкість у збереженні метаінформації про аудіофайли та результатів обробки, зокрема розподілу за мовцями, фіксації помилок, тривалості аудіофайлів, часу їх обробки та інших параметрів. На рис. 3.13 зображена неповна схема бази даних.

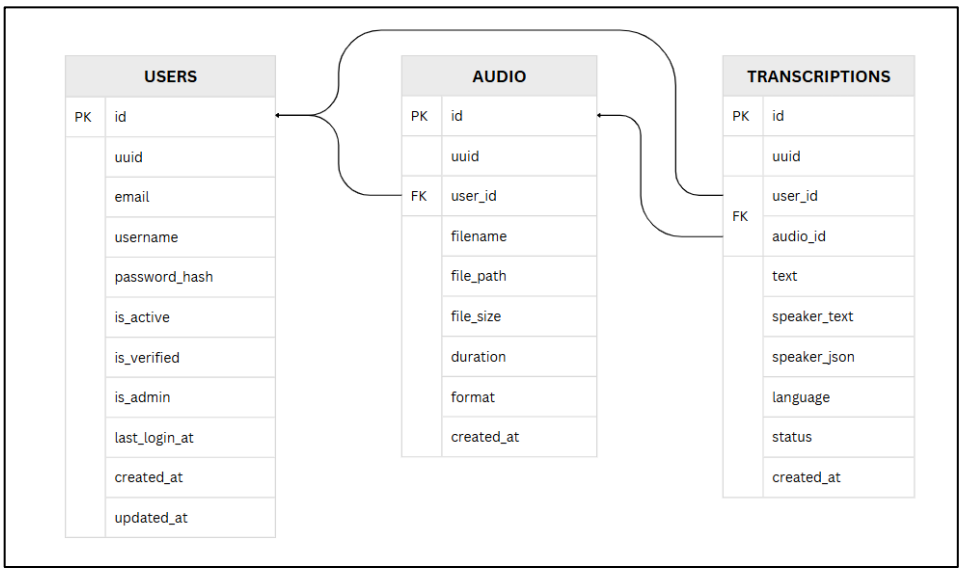


Рисунок 3.13 – Схема бази даних

Таблиця ***users*** містить дані про зареєстрованих користувачів. Для кожного користувача зберігається унікальний ідентифікатор id, автоматично згенерований uuid, адреса електронної пошти (email), ім’я користувача (username), відображуване ім’я (display_name) та хеш паролю (password_hash). Також передбачено логічні поля, що визначають стан облікового запису: активність (is_active), верифікацію (is_verified) та права адміністратора (is_admin). Для контролю часу останнього входу і моменту створення або оновлення облікового запису зберігаються відповідні дати: last_login_at, created_at, updated_at.

Таблиця ***audio*** використовується для зберігання інформації про завантажені аудіофайли. Вона містить id, uuid, а також зовнішній ключ user_id, що вказує на

користувача, якому належить файл. Зберігаються такі характеристики файлу, як ім’я (filename), шлях до збереження (file_path), розмір у байтах (file_size), тривалість (duration) та формат файлу (format). Дата завантаження фіксується у полі created_at.

Таблиця ***transcriptions*** відповідає за зберігання результатів розпізнавання мовлення. Кожен запис містить id, uuid, а також зовнішні ключі user_id та audio_id, які пов’язують транскрипцію з конкретним користувачем і аудіофайлом. Основна інформація включає розпізнаний текст у полі text. Додатково зберігається текст із поділом за мовцями (speakers_text) і його представлення у форматі JSON (speakers_json_text). Вказується мова транскрипції (language) та поточний статус (status), який може бути pending, processing, completed або failed. Дата створення транскрипції записується в полі created_at.

На рис. 3.14 відображено графік зростання об’єму бази даних залежно від кількості оброблених файлів. Дані для побудови графіка отримано на основі оцінки в середньому обсягу інформації, що зберігається в базі даних для одного обробленого аудіофайлу: один аудіофайл важить приблизно 200 байт, текст до нього – 500 байт, записи про мовців та окремі сегменти – 2,1 Кбайт. Сумарно приблизно 2,8 КБайб на один файл. Графік ілюструє лінійне зростання обсягу бази даних відповідно до кількості оброблених файлів.

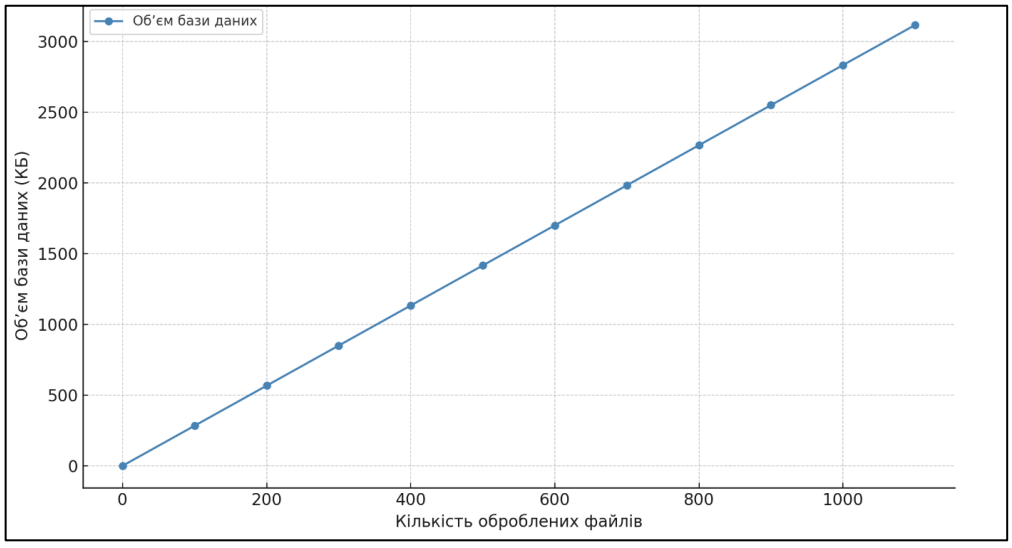


Рисунок 3.14 – Зростання об’єму бази даних залежно від кількості оброблених файлів

Окрім основних таблиць, у базі даних передбачена таблиця Logs, призначена для відстеження внутрішніх подій системи. Вона містить інформацію про статуси обробки, помилки, типи виконаних запитів, що є важливим для налагодження роботи або масштабування системи.

3.2.3 Модель обробки помилок

Система обробки помилок є ключовим компонентом будь-якої сучасної інформаційної платформи, особливо такої, що працює з аудіоданими та використовує алгоритми розпізнавання мовлення. Вона забезпечує стійкість до технічних збоїв, адекватну реакцію на нестандартні обставини, збереження цілісності даних, а також оперативне інформування користувачів про виниклі проблеми. У межах створеної системи транскрибування впроваджено багаторівневу архітектуру для обробки помилок, яка охоплює як користувацький інтерфейс, так і внутрішні серверні процеси, опрацювання аудіофайлів, взаємодію з базою даних і реєстрацію подій.

На користувацькому рівні система перевіряє коректність введених даних, відповідність типу й розміру завантажених файлів, а також аналізує помилки у взаємодії з інтерфейсом. Наприклад, якщо користувач завантажує файл у невідповідному форматі чи не заповнює обов’язкові поля, система відображає відповідне повідомлення. Це дозволяє мінімізувати кількість некоректних запитів, які можуть спричинити помилки на серверній стороні.

На рівні сервера реалізовано перевірку запитів щодо наявності всіх необхідних параметрів, відповідності формату, перевірку автентифікації користувача та обробку виняткових ситуацій, які можуть виникнути під час виконання бізнес-логіки. У разі виявлення помилки сервер повертає структуровану відповідь у форматі JSON із зазначенням коду помилки, описом проблеми та рекомендаціями щодо її виправлення.

Під час обробки аудіофайлів можуть виникати помилки, зумовлені такими факторами, як низька якість запису, відсутність мовного сигналу, пошкоджені фрагменти або інші аномальні ситуації. Для вирішення подібних випадків у системі

передбачено спеціальний журнал подій, де реєструються тип виявленої помилки, час її виникнення, ідентифікатор відповідного користувача та аудіофайлу, а також надається стислий опис проблеми. Такий підхід сприяє діагностиці функціонування системи та вдосконаленню алгоритмів її обробки.

На рис. 3.15 представлена блок-схема, що описує, як система обробляє різні типи помилок, які можуть виникнути під час процесу транскрибування. Ця схема показує основні етапи та можливі точки виникнення помилок з відповідними діями системи.

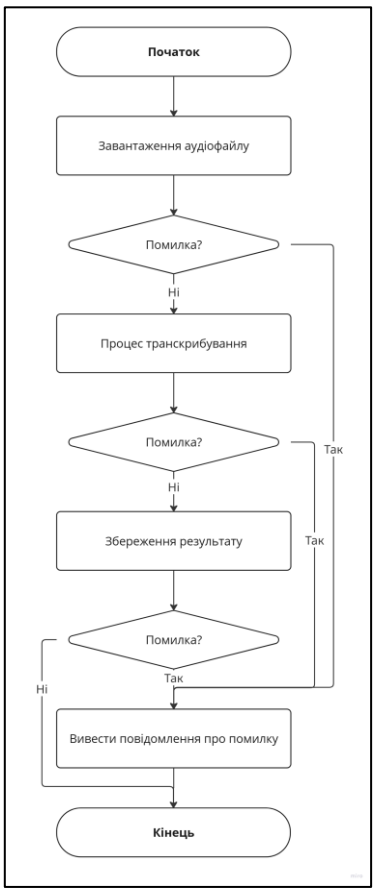


Рисунок 3.15 – Блок-схема моделі обробки помилок

З точки зору роботи з базою даних усі операції виконуються, дотримуючись принципу транзакційності. У разі виникнення збоїв жодна часткова зміна не зберігається: транзакція анулюється повністю, що забезпечує цілісність та узгодженість інформації, яку зберігає система. Окрім того, реалізований механізм фіксації критичних помилок у спеціалізованій таблиці Logs, що додатково підвищує стабільність і надійність обробки даних.

3.2.4 Обробка сегментації мовців (діаризація)

Діаризація мовлення – це метод, який дозволяє визначати, які частини аудіозапису належать різним мовцям. Важливо зауважити, що цей процес не спрямований на встановлення імен чи ідентичності осіб, а лише забезпечує сегментацію аудіо за принципом "хто і коли говорить". Таке розмежування є ключовим для подальшої роботи з транскрипціями, аналізу розмов чи створення текстів із чітким поділом між мовцями.

Процес діаризації зазвичай включає кілька етапів, серед яких: виявлення активності мовлення (VAD), розділення аудіофайлу на фрагменти, екстракція ознак мовлення (наприклад, x-vector або embeddings), кластеризація отриманих даних та об'єднання результатів. Одним із сучасних рішень для якісної діаризації виступає бібліотека PyAnnote.audio — універсальний інструмент для роботи з аудіо, який підтримує діаризацію, верифікацію мовців та інші пов'язані завдання. Фрагмент коду нижче – це простий приклад використання PyAnnote.audio для діаризації.

```
from pyannote.audio import Pipeline
# завантаження попередньо натренованої моделі
pipeline = Pipeline.from_pretrained("pyannote/speaker-diarization")
# діаризація аудіофайлу
diarization = pipeline("example.wav")
# виведення результатів
for turn, _, speaker in diarization.itertracks(yield_label=True):
    print(f"{turn.start:.1f}s - {turn.end:.1f}s: {speaker}")
```

Цей код використовує модель для автоматичного визначення кількості мовців і сегментування аудіо.

3.3 Аналіз результатів

3.3.1 Метрики оцінки якості транскрибування

Оцінка якості транскрибування є ключовим етапом у перевірці ефективності систем розпізнавання мовлення. Надійність і точність результатів безпосередньо залежать від обраних метрик, які дозволяють кількісно охарактеризувати якість транскрипції, порівнюючи її з еталонним текстом (ground truth).

Найбільш поширеною та загальноприйнятою метрикою є Word Error Rate (WER) – показник, який визначає частку помилок у транскрипції. Вона розраховується як сума вставок (insertions), видалень (deletions) та замін (substitutions) слів, поділена на загальну кількість слів в еталонному тексті. Формально:

$$WER = \frac{S+D+I}{N}, \quad (3.1)$$

де S – кількість замін слів;

D – кількість видалень слів;

I – кількість вставок слів;

N – кількість слів у еталонному реченні.

Чим менше значення WER, тим якісніша транскрипція.

Іншою важливою метрикою є Character Error Rate (CER) — вона працює аналогічно до WER, але оцінює помилки на рівні символів. CER особливо корисна при роботі з мовами, де слова мають складну морфологію або відсутні чіткі пробіли (наприклад, китайська чи японська).

Додатково, у практиці оцінювання можуть застосовуватися метрики Match Error Rate (MER), Word Information Lost (WIL) та Word Information Preserved (WIP), які враховують специфіку спотворень інформації. Наприклад, WIL дозволяє оцінити, скільки інформації було втрачено внаслідок неправильного розпізнавання.

Також можна порівнювати середню довжину правильного сегменту, коефіцієнт точності розділення мовців (для діаризації) або індекс узгодженості розмітки.

3.3.2 Аналіз потенційних проблем та обмежень

Система автоматичного транскрибування та діаризації мовлення, хоча й демонструє високий рівень автоматизації та ефективності, має низку потенційних викликів і обмежень, які можуть вплинути на якість результатів, стабільність функціонування та точність розпізнавання. В табл. 3.3 надано короткий опис про

вплив різних факторів на WER/CER. Ці проаналізовані дані допоможуть ідентифікувати слабкі сторони системи та напрямки для її покращення.

Таблиця 3.3 – Вплив різних факторів на WER/CER

Фактор	**Вплив на WER/CER**	**Запропоновані шляхи вирішення**
Фоновий шум	Високий	Використання алгоритмів шумозаглушення; фільтрація аудіосигналу
Акцент мовця	Середній / Високий	Розширення навчальних даних моделями з різними акцентами
Швидкість мовлення	Середній	Адаптація моделей до змін темпу мовлення; використання data augmentation
Якість вхідного аудіо	Високий	Визначення мінімальних технічних вимог до запису; контроль якості
Перекриття мовлення (діалоги)	Високий	Впровадження діаризації; використання моделей, здатних до обробки перехресного мовлення
Відлуння або реверберація	Середній / Високий	Акустична обробка приміщень; попереднє очищення сигналу
Наявність жаргону або спеціальних термінів	Середній	Розширення словника; тонке донавчання моделі на доменних даних
Обмеження словника моделі	Середній	Використання субсловникових моделей (BPE, SentencePiece)
Мовна багатоманітність	Середній	Підтримка багатомовних моделей або мультизадачне навчання

Найбільшою проблемою є якість аудіозаписів. Фактори на зразок шумового фону, низької частоти дискретизації, відлуння, міжмовних перешкод або одночасного мовлення кількох осіб серйозно погіршують результати розпізнавання. У таких умовах моделі часто неправильно інтерпретують сказане, що призводить до підвищеного рівня помилок (Word Error Rate). Тому в багатьох виробничих системах застосовують попередню обробку звуку, зокрема фільтрацію шумів чи нормалізацію гучності, для покращення якості вхідного сигналу.

Однак дослідження, опубліковане у журналі EURASIP Journal on Audio, Speech, and Music Processing, демонструє, що ефективність розпізнавання мовлення у зашумлених умовах залежить не лише від рівня шуму в аудіофайлі, але й від того, за яких умов був підвищений рівень SNR [26]. Графік зміни WER при різних значеннях SNR зображений на рис. 3.16.

Подібна поведінка була підтверджена і для трьох інших типів шуму, що дозволяє зробити загальний висновок, що фоновий шум не завжди буде підвищувати кількість помилок при транскрибуванні.

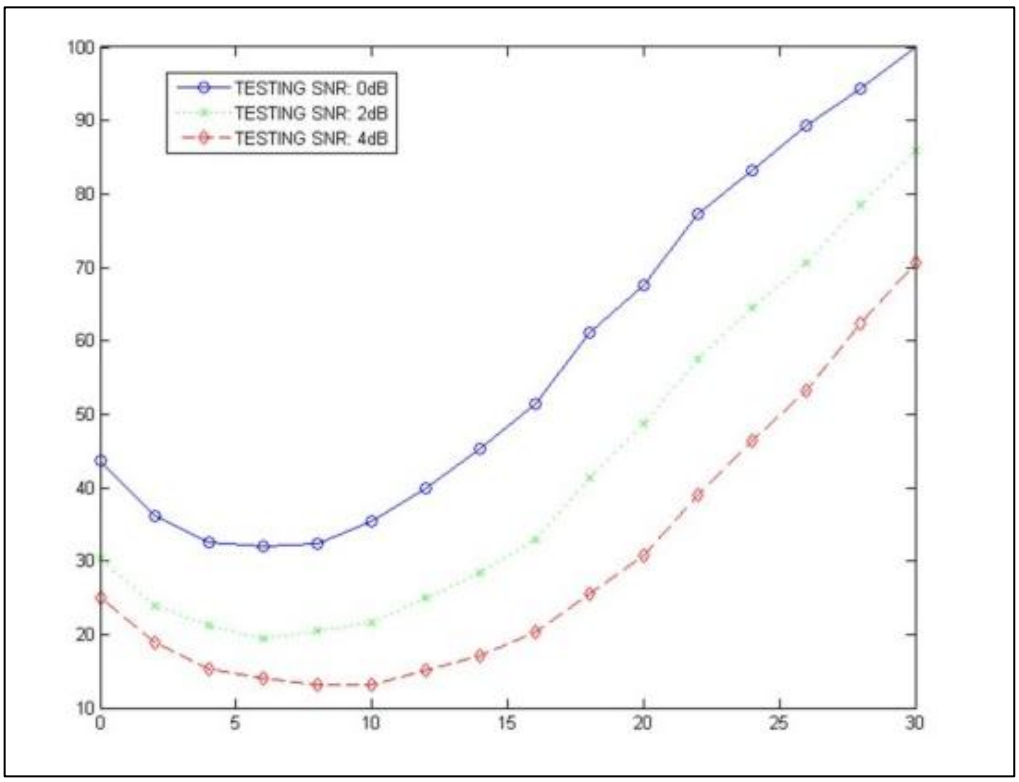


Рисунок 3.16 – Графік зміни WER при різних значеннях SNR в умовах кращого розпізнавання мовлення

Ще одне важливе обмеження стосується підготовленості моделі, тобто якості та обсягу навчальних даних. Якщо модель навчали на недостатній кількості прикладів певної мови, діалекту чи специфічного лексикону, її точність у розпізнаванні таких мовних варіацій значно знижується. Це особливо актуально для рідкісних мов, вузькопрофесійного жаргону або швидкої розмовної мови. Тому

важливо уважно підійти до вибору бібліотеки для розпізнавання мовлення для реалізації бекенду.

У процесі діаризації основним викликом залишається ідентифікація мовців в умовах підвищеної складності, зокрема коли голоси мають схожі тембри або загальну акустичну характеристику. Додаткову складність створює високий темп чергування реплік та ситуації, коли виникає перехресне мовлення, що ускладнює коректний розподіл сегментів. Це призводить до зниження точності сегментації та спричиняє помилкову атрибуцію реплік конкретним мовцям.

Серед технічних обмежень варто відзначити значні обчислювальні витрати, особливо при роботі з великими обсягами аудіоданих або під час застосування нейромережевих моделей у режимі реального часу. Такі вимоги можуть суттєво ускладнити масштабування систем у продуктивному середовищі без попередньої оптимізації ресурсів.

Висновки до розділу 3

Третій розділ детально охоплює архітектуру та основні компоненти програмної системи для автоматичного транскрибування мовлення. Спершу, у підрозділі 3.1, була представлена загальна структура системи, що включає модулі завантаження аудіофайлів, обробки даних, транскрипції, діаризації мовців та збереження результатів. Такий підхід забезпечує масштабованість, модульність і гнучкість системи, що є важливим для розширення функціоналу чи інтеграції з іншими сервісами.

У підрозділі 3.2 було здійснено глибокий аналіз технічної реалізації системи. У межах підрозділу 3.2.1 наведено діаграму класів, яка демонструє взаємозв’язки між ключовими сутностями, такими як користувачі, аудіофайли, транскрипти, мовці та сегменти. Це дозволило створити чітке розуміння об’єктно-орієнтованої структури проєкту та логіки його роботи. Подальший опис у підрозділі 3.2.2 стосується реляційної моделі бази даних, яка включає основні таблиці та зв’язки між ними.

Підрозділи 3.2.3 і 3.2.4 детально розглядають динамічні аспекти роботи системи: процес транскрибування мовлення та обробку діаризації мовців. Діаграма послідовності ілюструє взаємодію між компонентами системи під час транскрипції, починаючи з дії користувача й завершуючи збереженням результатів. Окрім цього, описано теоретичні основи діаризації мовців і наведено приклади реалізації з використанням бібліотеки pyannote.audio.

Загальний зміст розділу 3 формує міцну архітектурну і технічну базу для розробки програмної системи. Представлена структура забезпечує надійність системи, її адаптивність до змін і закладає основу для подальшого розвитку через додавання нових моделей або методів обробки аудіо.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ТРАНСКРИБУВАННЯ

4.1 Реалізація backend-частини

4.1.1 Архітектура та основні модулі backend

Backend-частина розробленої системи для транскрибування та аналізу аудіофайлів реалізована на мові програмування Python із використанням Flask — легкого фреймворку для створення REST API. Основна архітектура базується на модульному підході: кожен функціональний блок винесений у відповідний окремий модуль, що значно полегшує розробку, масштабування та тестування.

Серверна частина забезпечує завантаження аудіофайлів, їх попередню обробку, транскрипцію, діаризацію мовців, подальший семантичний аналіз за допомогою мовної моделі (LLM), а також зберігання проміжних і фінальних результатів у базі даних. На рис. 4.1 зображена блок-схема, яка показує взаємодію між модулями app.py, transcribe.py, llmworker.py, models.py, БД, LLM, та frontend.

Основні модулі системи:

– *app.py* — основний файл, що запускає Flask-додаток. Тут здійснюється ініціалізація серверу, підключення бази даних через SQLAlchemy, налаштування CORS, обробка HTTP-запитів, а також виклик транскрипційного та аналітичного процесу. Передбачене використання Celery для асинхронної обробки даних. Додатково реалізовані функції шифрування/дешифрування даних за допомогою бібліотек Crypto.Hash, Crypto.Cipher;

– *transcribe.py* — модуль для обробки аудіофайлів. Він використовує бібліотеку Whisper для автоматичного розпізнавання мовлення, pyannote.audio для діаризації мовців, а також pydub для перетворення аудіоформатів. Передбачено поділ на сегменти, обробка мовчання, збереження транскрипту, визначення структурованих частин розмови тощо;

– *llmworker.py* — модуль для взаємодії з LLM (великими мовними моделями) через сервіс Amazon Bedrock. Він формує запити до LLM, передає

оброблений текст транскрипту та отримує узагальнення або аналіз діалогів. Тут застосовується бібліотека boto3 для роботи з AWS API;

– *models.py* — файл із моделями бази даних, створеними за допомогою SQLAlchemy. Визначені сутності для збереження інформації про виклики (Call), транскрипти (Transcription), результати аналізу (ComprehendMedicalResult) та LLM-висновки (LLMSummary). Застосовується PostgreSQL як СКБД.

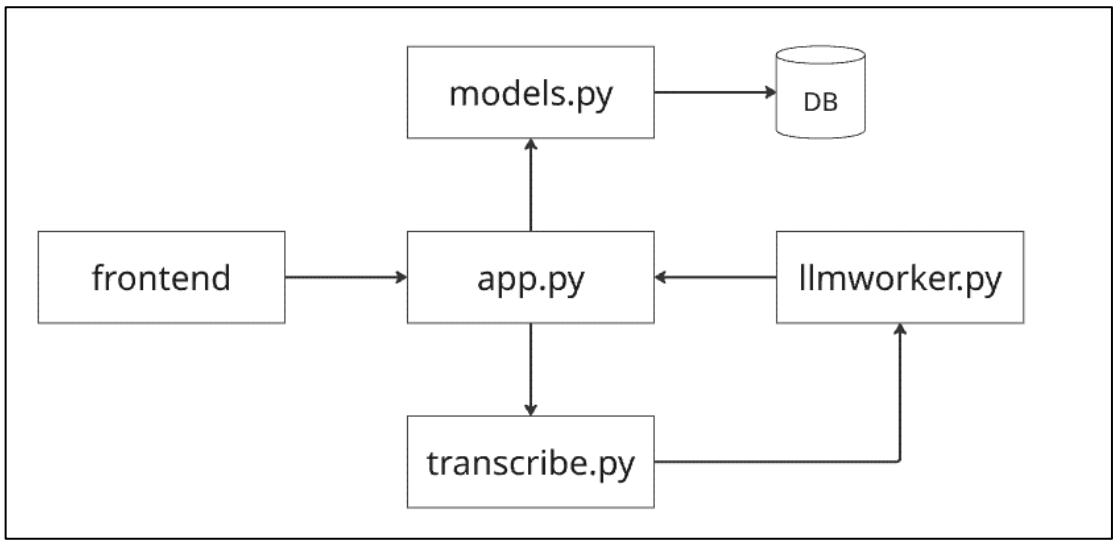


Рисунок 4.1 – Загальна архітектура backend-системи

Усі частини backend системи пов’язані між собою за допомогою чітко визначених API та ORM-зв’язків, що забезпечує логічну послідовність обробки даних та збереження результатів. Завдяки використанню таких бібліотек, як torch, tempfile, uuid, dotenv, numpy тощо, реалізовано обробку аудіофайлів у різних форматах, генерацію унікальних ідентифікаторів, використання тимчасових файлів та інтеграцію з хмарними сервісами.

4.1.2 Розробка API Flask для взаємодії з frontend

Одним із ключових етапів розробки вебзастосунку стало формування серверної частини на основі мікрофреймворку Flask. Основною метою створеного API було забезпечення ефективної інтеракції між фронтенд-частиною та бекендом. Зокрема, це включало обробку запитів, маніпуляції з аудіофайлами, виконання асинхронних завдань і повернення результатів транскрипції та узагальнення.

На початкових етапах розробки було створено застосунок Flask, який був спеціально налаштований для інтеграції з клієнтською частиною, що функціонує на http://localhost:3000. Це здійснено через конфігурацію CORS (Cross-Origin Resource Sharing), яка відкриває можливість фронтенду надсилати запити до Flask-серверу. Ініціалізація цих бібліотек зображена на рис. 4.2. Такі запити охоплюють завантаження аудіофайлів і отримання статусу виконання транскрипції.

```
app = Flask(__name__)

CORS(app, resources={
    r"/upload": {"origins": "http://localhost:3000"},
    r"/transcribe": {"origins": "http://localhost:3000"},
    r"/auth/*": {"origins": "http://localhost:3000"},
    r"/transcriptions/*": {"origins": "http://localhost:3000"},
    r"/audio/*": {"origins": "http://localhost:3000"}
    })
```

Рисунок 4.2 – Ініціалізація Flask та CORS

Одною з ключових функцій створеного API є ефективна обробка аудіофайлів. Користувачі можуть передавати аудіофайли через маршрут /transcribe. Обробку маршруту відображено на рис. 4.3.

```
@app.route('/transcribe', methods=['GET'])
def transcribe_immediately():
    if 'file' not in request.files:
        return jsonify({'error': 'No file part'}), 400

    file = request.files['file']
    if file.filename == '':
        return jsonify({'error': 'No selected file'}), 400

    filename = secure_filename(file.filename)
    file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
    file.save(file_path)
    try:
        result = model.transcribe(file_path)
        transcription = result['text']

        os.remove(file_path)

        return jsonify({
            'status': 'success',
            'text': transcription
        })
    except Exception as e:
        app.logger.error(f"Помилка транскрибації: {str(e)}")
        return jsonify({'error': 'Internal server error'}), 500
```

Рисунок 4.3 – Обробка маршруту /transcribe

Після їх отримання файли тимчасово зберігаються на диск у призначену для цього директорію. Далі здійснюється транскрипція мовлення за допомогою моделі Whisper, а результати надаються у вигляді JSON-об'єкта. На рис. 4.4 зображено відповідь API сервера у форматі JSON, отриманої за допомогою інструментів розробника браузера. У випадках виникнення помилок система генерує відповідне повідомлення із детальним описом проблеми, яке повертається користувачеві для подальшого вирішення.

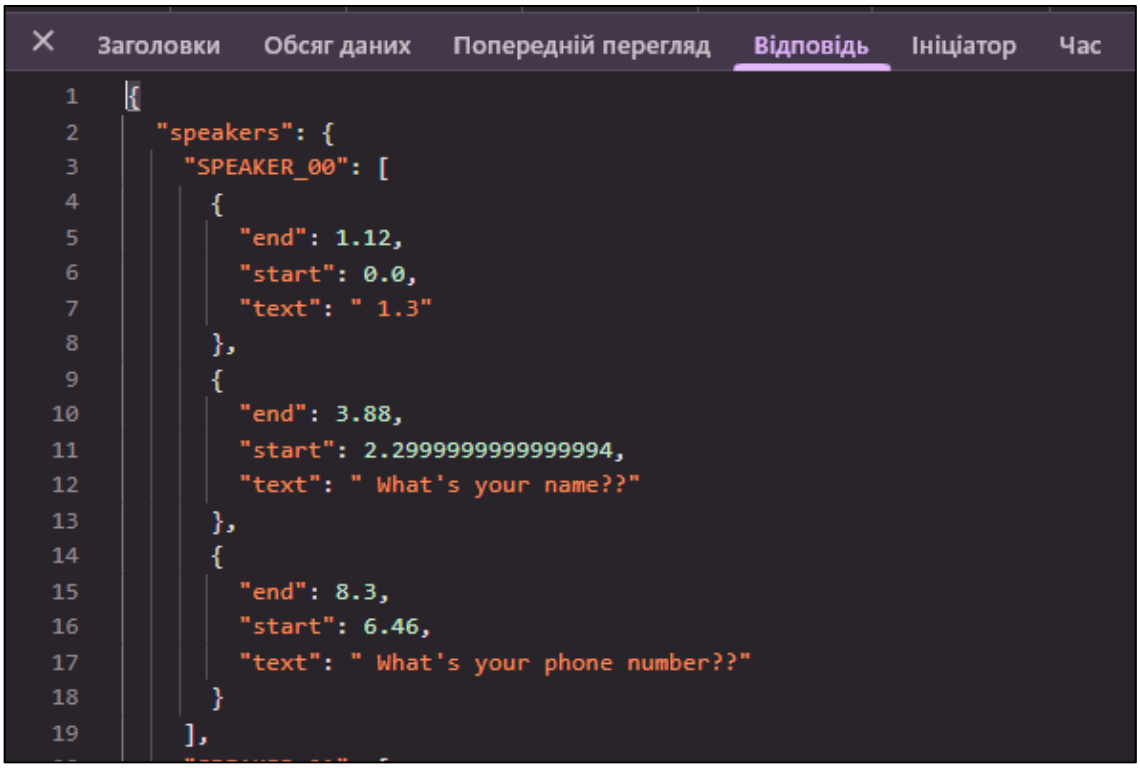


Рисунок 4.4 – Відповідь сервера у вигляді JSON

Іншим важливим аспектом функціонування системи є впровадження асинхронного підходу до виконання завдань через використання інструментів Celery та Redis. На рис. 4.5 зображена реалізація використання Celery для асинхронної задачі. Даний підхід забезпечує підвищення масштабованості системи та оптимізацію обробки великих обсягів даних. Завдяки винесенню задач, таких як транскрибування або узагальнення значних текстових масивів, у фонові процеси, вдалося уникнути блокування головного потоку додатка, що, своєю чергою, суттєво зменшує час очікування для кінцевого користувача.

```python
@celery.task(bind=True)
def transcribe_process(self, file_path, tr_uuid, model_type = 'large-v3'):
    with app.app_context():
        try:
            print(f"Starting transcription for {tr_uuid}")
            transcription = Transcription.query.filter_by(uuid=tr_uuid).first()
            if not transcription:
                raise Exception(f"Transcription with UUID {tr_uuid} not found")

            transcription.status = "processing"
            db.session.commit()

            transcribe = Transcribe()

            pre_loaded_file = transcribe.get_audio_data(file_path)
            normalized_file = transcribe.audio_normalize(pre_loaded_file)

            try:
                result = transcribe.audio_to_text(model=model, mediafile=normalized_file)
            except Exception as e:
                print(f"Model {model} failed, trying base model: {str(e)}")
                result = transcribe.audio_to_text(model='base', mediafile=normalized_file)

            try:
                diarization_result = transcribe.diarization(audio_location=normalized_file)
                speakers_json, speakers_text = transcribe.match_transcription_diarization(
                    diarization_result, result, normalized_file)
```

Рисунок 4.5 – Використання Celery для асинхронної задачі

Для роботи бібліотеки Celery обов'язково потрібен Redis. Підключення його ілюструється на рис. 4.6-4.7.

```
C:\Users\Admin\Lera>docker run -d -p 6379:6379 --name redis-server redis
Unable to find image 'redis:latest' locally
latest: Pulling from library/redis
61320b01ae5e: Pull complete
f8f9e5369fdd: Pull complete
b8f8315b617a: Pull complete
eb61abf5b105: Pull complete
03bc4ee78aa8: Pull complete
4f4fb700ef54: Pull complete
93cd3c5153ab: Pull complete
Digest: sha256:b3ad79880c88e302deb5e0fed6cee3e90c0031eb90cd936b01ef2f83ff5b3ff2
Status: Downloaded newer image for redis:latest
ec48772183965ca073d889f8535c4c5dc499ddaed87720a4fae9c8ecd8ca3f21

C:\Users\Admin\Lera>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS                    NAMES
ec4877218396   redis     "docker-entrypoint.s…"   20 seconds ago   Up 19 seconds   0.0.0.0:6379->6379/tcp   redis-server
```

Рисунок 4.6 – Завантаження Redis

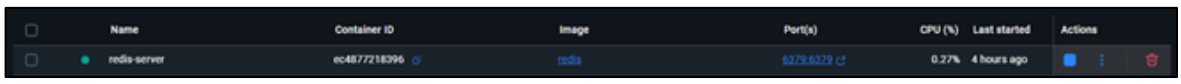


Рисунок 4.7 – Активний Redis

На рис. 4.8 показано підключення Celery, яка виконує розподілені асинхронні задачі за допомогою брокера повідомлень (наприклад, Redis або RabbitMQ), а Flower (рис. 4.10) використовується як веб-інтерфейс для моніторингу статусу

виконання цих задач у реальному часі. На рис. 4.7=9 показано підключення Celery до Redis для аналізу результатів асинхронності.

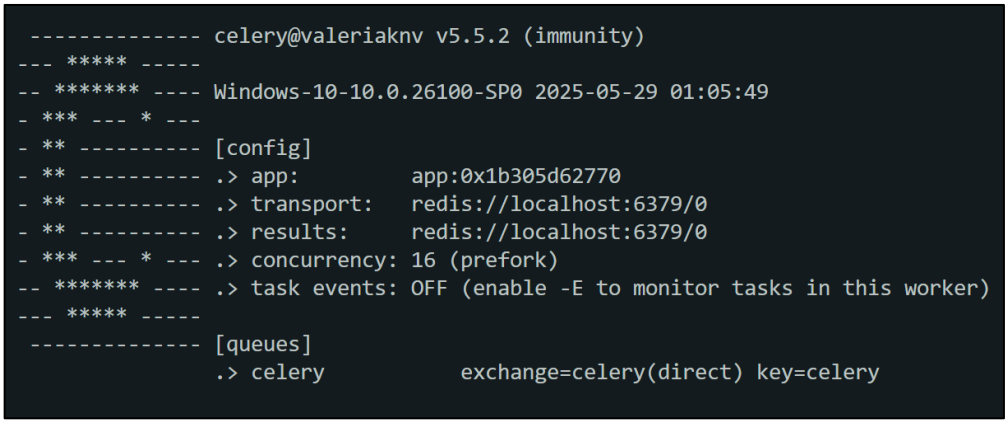


Рисунок 4.8 – Процес запуску Celery

Рисунок 4.9 – Підключення до Celery до Redis

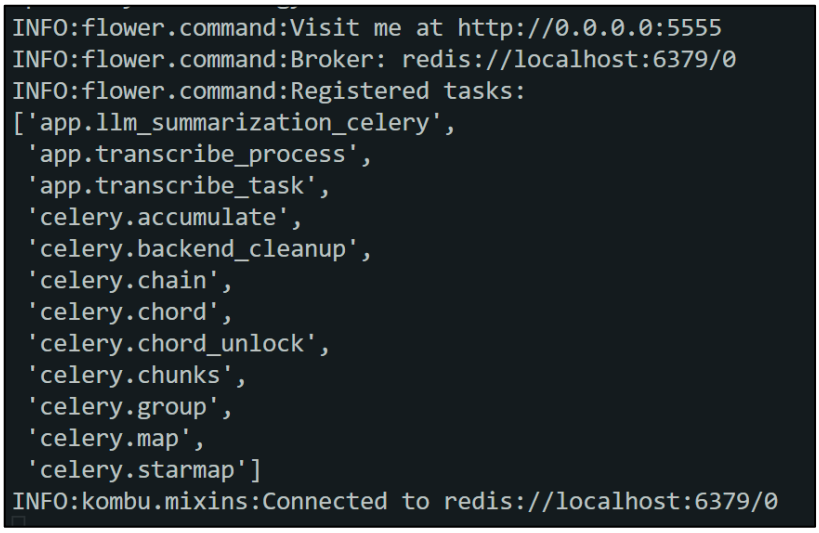


Рисунок 4.10 – Процес запуску Flower

Celery відповідає за постановку та виконання задач, у той час як Flower збирає та відображає інформацію про їхній стан, статистику та історію виконання.

Такий підхід забезпечує прозорість процесу обробки задач та полегшує діагностику помилок.

На рис. 4.11 наведено скріншот інтерфейсу Celery Flower, на якому відображено активну та успішно виконану задачу, що підтверджує коректну роботу системи та можливість ефективного моніторингу.

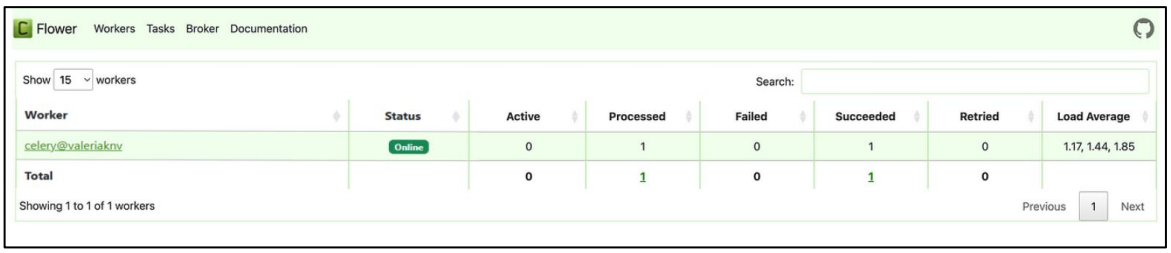


Рисунок 4.11 - Скріншот інтерфейсу Celery Flower з активною задачею

Окрім транскрипції, функціонал API включає можливість генерації коротких узагальнень на основі транскрибованого тексту, що виконується за допомогою модуля LLMWorker. Такий модуль обробляє вихідний результат транскрипції та створює лаконічний і змістовний зміст тексту. Ця функція стає особливо актуальною під час роботи з великими мовними ресурсами, наприклад, у разі аналізу записів лекцій чи зустрічей. Фрагмент коду з генерацією узагальнення тексту відображено на рис. 4.12. Процес узагальнення реалізується через поетапне розбиття тексту на окремі частини з подальшою їх обробкою і з'єднанням отриманих підсумків у єдиний інтегрований документ.

```python
def llm_summarization(tr_uuid):
    with app.app_context():
        print("summarization process has started")
        value_ = Transcription.query.filter_by(uuid = str(tr_uuid)).first()
        uuid_=value_.uuid
        transcription_speakers = value_.transcription_speakers
        llmworker = LLMWorker()
        summary = llmworker.create_summary_prompt(transcription_speakers)
        llmobject = LLMSummary.query.filter_by(uuid = str(tr_uuid)).first()
        if llmworker:
            llmobject.summary = summary
            llmobject.status = True
            db.session.commit()
            print("Summary is done! ")
        else:
            print("We are screwed, Bedrock is dead")
```

Рисунок 4.12 – Генерація узагальнення тексту

Важливим елементом архітектури системи є також забезпечення безпеки даних. Для цього був імплементований механізм розшифрування інформації, який дозволяє працювати із захищеними об'єктами (рис. 4.13). Вхідні шифровані рядки обробляються за допомогою алгоритму AES у режимі CBC, при цьому для генерації ключів використовується хешування за методом MD5. Розшифровані дані інтегруються у систему шляхом оновлення відповідних записів у базі даних.

```python
def decrypt_payload(ciphertext_b64,password):
    encryptedData = base64.b64decode(ciphertext_b64)
    salt = encryptedData[8:16]
    ciphertext = encryptedData[16:]
    derived = b""
    while len(derived) < 48: # "key size" + "iv size" (8 + 4 magical units = 12 * 4 = 48)
        hasher = MD5.new()
        hasher.update(derived[-16:] + password.encode('utf-8') + salt)
        derived += hasher.digest()
    key = derived[0:32]
    iv = derived[32:48]
    cipher = AES.new(key, AES.MODE_CBC, iv)
    decrypted = unpad(cipher.decrypt(ciphertext), 16).decode('utf-8')
    dc_dict = decrypted.split('&')
    print(dc_dict)
    res_list = []
    for i in range(len(dc_dict)):
        temp = dc_dict[i].split('=')
        res_list.append(temp[0])
        res_list.append(temp[1])
    print(res_list)
    return res_list
```

Рисунок 4.13 – Дешифрування вхідного запиту

Усі транскрипції та їх узагальнення зберігаються у централізованій базі даних, яка інтегрована з веб-фреймворком Flask за допомогою бібліотеки SQLAlchemy. У процесі розробки створено моделі даних Transcription і LLMSummary, що відповідають за збереження тексту транскрипцій та відповідних узагальнень. На рис. 4.14 зображено у вигляді простої схеми взаємодія цих моделей. Такий підхід дає змогу ефективно отримувати необхідну інформацію для подальшого відображення в інтерфейсі користувача.

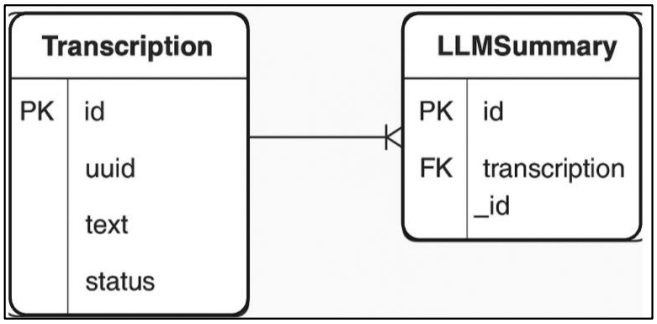


Рисунок 4.14 – ER-діаграма бази даних

На виході цього етапу було створено гнучкий, масштабований і захищений API, який забезпечує всі основні функціональні можливості для належної інтеграції та роботи з frontend-інтерфейсом застосунку.

4.1.3 Інтеграція з сервісами транскрибування та обробки аудіо

Інтеграція сервісів транскрибування та діаризації спікерів виконана у формі класу Transcribe, який забезпечує повний процес обробки вхідного аудіо: від оцінки його якості до створення структурованого текстового результату. Основна ціль полягає в отриманні точного транскрипту, сегментованого за спікерами, із чітким таймінгом, що має особливе значення для аналізу діалогів, інтерв'ю чи судових протоколів.

На першому етапі проводиться діагностика аудіофайлу. Для цього використовується метод diagnose_diarization(), який оцінює якість звуку, виконує нормалізацію, запускає стандартну діаризацію та передбачає резервні сценарії з примусовими параметрами або альтернативними алгоритмами кластеризації. Фрагмент коду реалізації зображено на рис. 4.15.

```
# Standaty diarization
try:
    diarization = self.diarization(normalized_file)
    speakers = set()
    for _, _, speaker in diarization.itertracks(yield_label=True):
        speakers.add(speaker)
    print(f"   Detected {len(speakers)} speakers: {', '.join(speakers)}")
except Exception as e:
    print(f"   Standard diarization failed: {str(e)}")

try:
    diarization = self.diarization_pipeline(
        normalized_file,
        num_speakers=2,
        clustering="AgglomerativeClustering"
    )
    speakers = set()
    for _, _, speaker in diarization.itertracks(yield_label=True):
        speakers.add(speaker)
    print(f"   Detected {len(speakers)} speakers: {', '.join(speakers)}")
```

Рисунок 4.15 – Фрагмент коду з діагностикою аудіофайлу

Це дозволяє ідентифікувати можливі проблеми ще до початку основної обробки. На рис. 4.16 зображена схема логіки цього методу, а на рис. 4.17 – результати аналізу. Наприклад, якщо стандартна діаризація не виявляє більше

одного спікера, система автоматично переходить до повторного аналізу із примусовим параметром num_speakers=2.

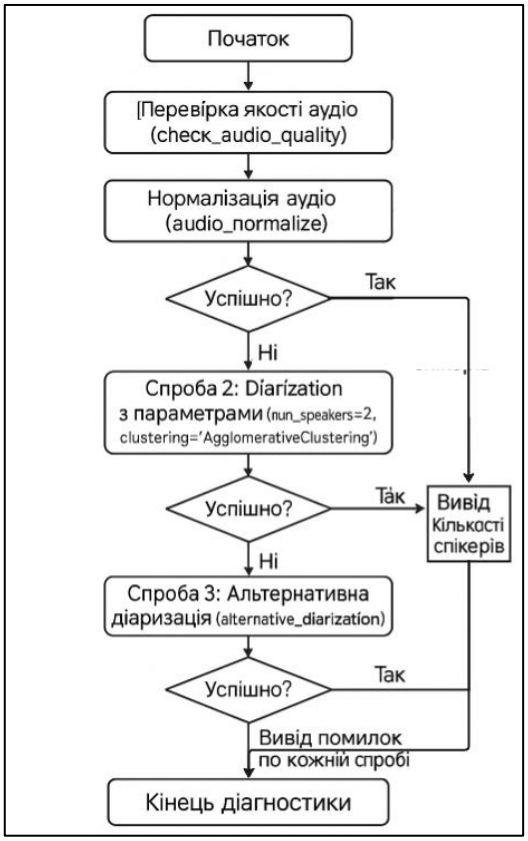


Рисунок 4.16 – Схема логіки diagnose_diarization()

```
Basic diarization completed
Segment type: <class 'pyannote.core.segment.Segment'>, Speaker: SPEAKER_03
Valid segment: 0.15-1.43, Speaker: SPEAKER_03
Segment type: <class 'pyannote.core.segment.Segment'>, Speaker: SPEAKER_03
Valid segment: 2.68-3.61, Speaker: SPEAKER_03
Segment type: <class 'pyannote.core.segment.Segment'>, Speaker: SPEAKER_03
Valid segment: 6.53-7.78, Speaker: SPEAKER_03
Segment type: <class 'pyannote.core.segment.Segment'>, Speaker: SPEAKER_02
Valid segment: 10.95-12.08, Speaker: SPEAKER_02
Segment type: <class 'pyannote.core.segment.Segment'>, Speaker: SPEAKER_02
Valid segment: 15.13-15.98, Speaker: SPEAKER_02
Segment type: <class 'pyannote.core.segment.Segment'>, Speaker: SPEAKER_01
Valid segment: 19.03-20.18, Speaker: SPEAKER_01
Segment type: <class 'pyannote.core.segment.Segment'>, Speaker: SPEAKER_01
Valid segment: 23.23-24.08, Speaker: SPEAKER_01
Segment type: <class 'pyannote.core.segment.Segment'>, Speaker: SPEAKER_04
Valid segment: 27.17-28.52, Speaker: SPEAKER_04
Segment type: <class 'pyannote.core.segment.Segment'>, Speaker: SPEAKER_04
Valid segment: 31.60-32.16, Speaker: SPEAKER_04
Segment type: <class 'pyannote.core.segment.Segment'>, Speaker: SPEAKER_00
Valid segment: 35.28-36.23, Speaker: SPEAKER_00
Found 5 speakers: SPEAKER_01, SPEAKER_03, SPEAKER_02, SPEAKER_00, SPEAKER_04
Post-processing: remapped 5 speakers
```

Рисунок 4.17 – Результати діагностики diagnose_diarization()

Метод diarization() грає ключову роль у забезпеченні адаптивності та надійності системи. На рис. 4.18 відображена його реалізація. Його основна задача — обробка різноманітних форматів результатів, які можуть бути представлені як об'єкти з атрибутами start і end або у вигляді звичайних tuple. Метод також відповідає за перевірку валідності отриманих сегментів і передбачає виконання fallback-дій у випадках, коли якість результатів виявляється недостатньою. Завдяки цьому система демонструє високу гнучкість до змін у форматах вихідних даних від різних моделей.

```
# Basic diarization
try:
    diarization = self.diarization_pipeline(audio_location)
    print("Basic diarization completed")
except Exception as e:
    print(f"Basic diarization failed: {str(e)}")
    # Min params try
    try:
        diarization = self.diarization_pipeline(
            audio_location,
            min_speakers=1,
            max_speakers=6
        )
        print("Diarization with min/max speakers completed")
    except Exception as e2:
        print(f"Diarization with parameters also failed: {str(e2)}")
        raise Exception(f"All diarization attempts failed: {str(e2)}")
```

Рисунок 4.18 – Фрагмент коду діаризації

Після завершення діаризації метод match_transcription_diarization() інтегрує її результати з текстовими транскрипціями (рис. 4.19). Він визначає відповідність між таймінгами діаризованих реплік спікерів і текстових сегментів, аналізує ступінь їхнього перекриття, а також ідентифікує домінуючого спікера в кожному часовому інтервалі.

Кожен сегмент отримує відповідну мітку (наприклад, SPEAKER_00, SPEAKER_01). У випадках, коли перекриття відсутнє, система знаходить найближчого за часом спікера. Це дозволяє точно відновлювати послідовність мовлення навіть у складних сценаріях, коли спікери перебивають один одного чи говорять одночасно.

```python
for segment in sorted(transcription['segments'], key=lambda x: x['start']):
    seg_start = segment['start']
    seg_end = segment['end']
    text = segment['text']

    # Знаходимо всі перекриття з поворотами спікерів
    overlaps = []
    for turn in speaker_turns:
        overlap_start = max(seg_start, turn['start'])
        overlap_end = min(seg_end, turn['end'])
        if overlap_start < overlap_end:  # Реальне перекриття
            overlaps.append({
                'speaker': turn['speaker'],
                'duration': overlap_end - overlap_start
            })

    # Призначаємо домінуючому спікеру, якщо є перекриття
    if overlaps:
        # Обчислюємо загальну тривалість для кожного спікера
        speaker_durations = defaultdict(float)
        for ov in overlaps:
            speaker_durations[ov['speaker']] += ov['duration']

        # Отримуємо домінуючого спікера
        dominant_speaker = max(speaker_durations.items(), key=lambda x: x[1])[0]
        norm_speaker = speaker_map[dominant_speaker]
```

Рисунок 4.19 – Фрагмент коду формування результатів транскрибування та діаризації

Фінальний результат надається у двох форматах: структурованому JSON для подальшої аналітики чи інтеграції та текстовому форматі з часовими позначками і маркуванням спікерів. Реалізація формування JSON зображена на рис. 4.20. Це забезпечує багатофункціональність рішення, роблячи його однаково придатним як для візуалізації в інтерфейсі користувача, так і для експорту в сторонні системи.

```python
for seg in sorted(all_segments, key=lambda x: x['start']):
    if seg['speaker'] != current_speaker:
        text_output += f"\n=== {seg['speaker']} ===\n"
        current_speaker = seg['speaker']
    text_output += f"[{seg['start']:.2f}-{seg['end']:.2f}] {seg['text']}\n"

for seg in all_segments:
    speakers[seg['speaker']].append({
        'start': seg['start'],
        'end': seg['end'],
        'text': seg['text']
    })
```

Рисунок 4.20 – Формування результатів у вигляді JSON

4.1.4 Реалізація аутентифікації користувачів

Аутентифікація користувачів є невід'ємним компонентом забезпечення безпеки веб-застосунків. У представленій системі процес аутентифікації реалізовано на стороні бекенду із застосуванням мікрофреймворку Flask у поєднанні з технологією JWT (JSON Web Token) для створення безпечних токенів доступу. Зазначений підхід дозволяє забезпечити захищений доступ до приватних ресурсів користувача без необхідності збереження сеансів на сервері.

Фундамент функціоналу забезпечує модуль *auth_routes.py*, в якому детально визначено маршрути для операцій реєстрації, авторизації, виходу користувача, а також отримання даних про поточного користувача. Крім того, модуль включає спеціалізований декоратор *@token_required*, що здійснює перевірку наявності та валідності токена перед виконанням запитів до маршрутів, які потребують авторизації (рис. 4.21-4.22).

```python
def token_required(f):
    """Декоратор для захищених маршрутів"""
    @wraps(f)
    def decorated(*args, **kwargs):
        token = request.headers.get('Authorization')

        if not token:
            return jsonify({'message': 'Token is missing'}), 401

        try:
            if token.startswith('Bearer '):
                token = token[7:]

            data = jwt.decode(token, JWT_SECRET, algorithms=['HS256'])
            user_id = data['user_id']

            current_user = User.query.filter_by(id=user_id).first()

            if not current_user:
                return jsonify({'message': 'User not found'}), 401

        except jwt.ExpiredSignatureError:
            return jsonify({'message': 'Token has expired'}), 401
        except jwt.InvalidTokenError:
            return jsonify({'message': 'Token is invalid'}), 401
        except Exception as e:
            print(f"Token validation error: {str(e)}")
            return jsonify({'message': 'Token validation failed'}), 401

        return f(current_user, *args, **kwargs)

    return decorated
```

Рисунок 4.21 – Декоратор token_required для перевірки валідації токенів

```python
@auth_bp.route('/me', methods=['GET'])
@token_required
def get_current_user(current_user):
    """Отримує інформацію про поточного користувача"""
    return jsonify({
        'status': 'success',
        'user': current_user.to_dict()
    }), 200
```

Рисунок 4.22 – Приклад використання @token_required

На рис. 4.23 зображена проста схема аутентифікації, яка ілюструє послідовність дій під час реєстрації, входу та перевірки доступу до захищених маршрутів. Цей алгоритм включає запит клієнта, перевірку даних, хешування паролю, генерацію токену та передачу токену назад.

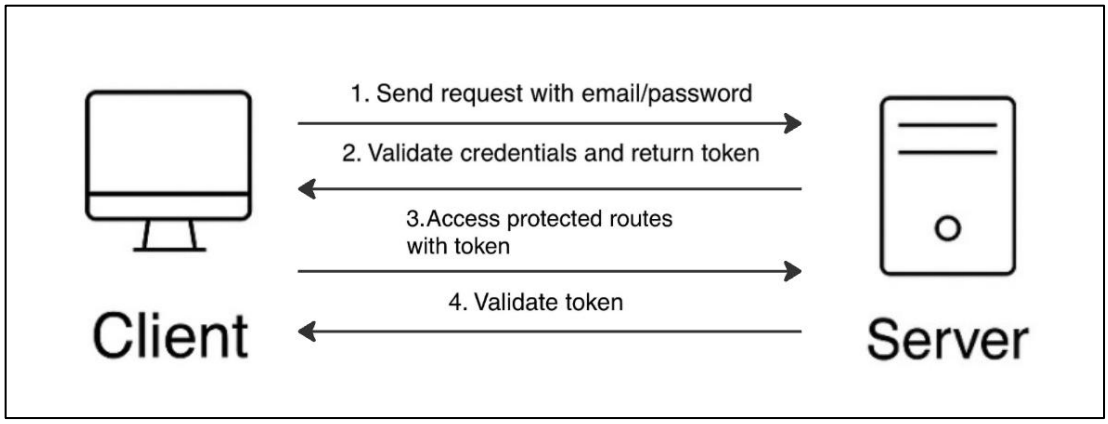


Рисунок 4.23 – Схема аутентифікації

У процесі реєстрації система очікує від клієнта надання значень *email, password* і необов'язкового поля *username*. Якщо користувач вирішить не вводити username, то замість нього буде використовуватись пошта користувача. Після проходження базової валідації введених даних (включаючи перевірку заповнення обов'язкових полів, відповідності мінімальним вимогам до довжини паролю, а також унікальності email й username), пароль хешується із застосуванням функції *generate_password_hash* (рис. 4.24). Далі створюється об'єкт класу *User*, який зберігається в базі даних за допомогою SQLAlchemy ORM. У разі успішного

завершення процесу створення користувача одразу генерується JWT-токен (рис. 4.25), який разом із інформацією про користувача надсилається клієнту.

```python
def set_password(self, password):
    """Встановлює хешований пароль"""
    self.password_hash = generate_password_hash(password)
```

Рисунок 4.24 – Хешування паролю

```python
def generate_token(user):
    """Генерує JWT токен для користувача"""
    try:
        expiration_time = datetime.datetime.utcnow() + datetime.timedelta(days=7)
        exp_timestamp = int(expiration_time.timestamp())

        payload = {
            'user_id': user.id,
            'email': user.email,
            'exp': exp_timestamp
        }

        print(f"Payload для JWT: {payload}")

        token = jwt.encode(payload, JWT_SECRET, algorithm='HS256')

        if isinstance(token, bytes):
            token = token.decode('utf-8')

        print(f"Token generated successfully: {token[:50]}...")
        return token
```

Рисунок 4.25 – Фрагмент коду з генерацією JWT-токену

Також реалізовано верифікацію користувача через підтвердження електронної пошти: на пошту користувачу надсилається лист (рис. 4.26), де необхідно натиснути кнопку підтвердження пошти (рис. 4.27). Фрагмент коду з перевіркою верифікації зображено на рис. 4.28.

```python
def send_email(self, to_email, subject, html_content, text_content=None):
    """Відправляє електронний лист"""
    try:
        message = MIMEMultipart("alternative")
        message["Subject"] = subject
        message["From"] = f"{self.app_name} <{self.email_address}>"
        message["To"] = to_email

        if text_content:
            text_part = MIMEText(text_content, "plain", "utf-8")
            message.attach(text_part)

        html_part = MIMEText(html_content, "html", "utf-8")
        message.attach(html_part)

        context = ssl.create_default_context()
        with smtplib.SMTP(self.smtp_server, self.smtp_port) as server:
            server.starttls(context=context)
            server.login(self.email_address, self.email_password)
            server.sendmail(self.email_address, to_email, message.as_string())

        print(f"✅ Email sent successfully to {to_email}")
        return True
```

Рисунок 4.26 – Фрагмент коду з надсиланням користувачу на пошту листа

```
<div class="content">
    <div class="greeting">Привіт, {display_name}!</div>

    <div class="message">
        Дякуємо за реєстрацію в {self.app_name}! Для завершення процесу реєстрації,
        будь ласка, підтвердіть вашу електронну пошту, натиснувши на кнопку нижче.
    </div>

    <div style="text-align: center;">
        <a href="{verification_url}" class="verify-button">
            ✅ Підтвердити електронну пошту
        </a>
    </div>
```

Рисунок 4.27 – HTML-код надсилаємого листа з підтвердженням пошти

```
if user.is_verified:
    return render_template_string("""
    <!DOCTYPE html>
    <html>
    <head>
        <title>Вже верифіковано</title>
        <meta charset="UTF-8">
        <style>
            body { font-family: Arial, sans-serif; text-align: center; padding: 50px; }
            .success { color: #27ae60; }
        </style>
    </head>
    <body>
        <h1 class="success">✅ Вже верифіковано</h1>
        <p>Ваша електронна пошта вже підтверджена.</p>
        <a href="http://localhost:3000">Перейти до додатку</a>
    </body>
    </html>
    """)

user.verify_email()
db.session.commit()
```

Рисунок 4.28 – Перевірка верифікації

Маршрут */login* виконує аналогічні перевірки. Користувач вводить email та пароль, які звіряються з даними в базі. Для перевірки пароля використовується функція *check_password_hash*. У випадку успішного проходження перевірки створюється токен, як і під час реєстрації. JWT містить інформацію про *user*_id, *email* та *exp* — час закінчення дії токена. В табл. 4.1 представлено опис JWT-токену. Шифрування токена здійснюється за допомогою симетричного алгоритму *HS256* із застосуванням секретного ключа, що зберігається в середовищній змінній *JWT_SECRET*.

Таблиця 4.1 – Опис полів JWT-токена

Поле	Опис
`user_id`	Унікальний ідентифікатор користувача
`email`	Електронна адреса
`exp`	Час закінчення дії токена

Декоратор *@token_required* є ключовим елементом авторизації. Його додають до всіх маршрутів, для яких потрібне підтвердження доступу. Цей декоратор аналізує заголовок *Authorization*, витягує з нього токен (прибираючи префікс *'Bearer'*), після чого розшифровує його через бібліотеку *jwt*. У разі недійсного токена, завершення терміну його дії або відсутності відповідного до нього користувача, запит блокується із зазначенням відповідного коду помилки. На рис. 4.29 продемонстровано скріншот відповіді сервера на успішну реєстрацію користувача.

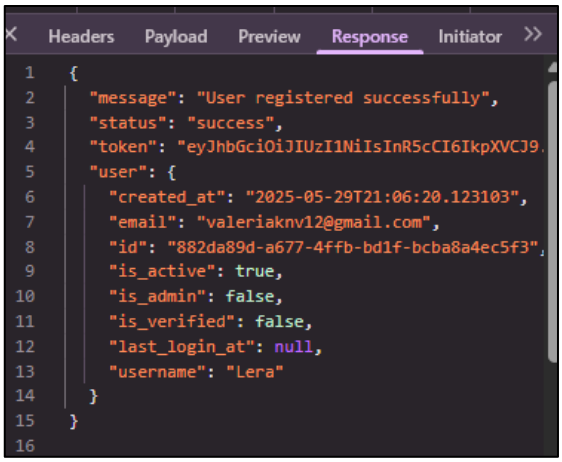


Рисунок 4.29 – Відповідь сервера на успішну реєстрацію

Окремі маршрути, як-от */me*, надають можливість отримати дані про поточного авторизованого користувача, використовуючи попередньо перевірений токен. У випадку маршруту */logout* система не зберігає стану, тому процедура виходу зводиться до видалення токена на боці клієнта.

Додатково впроваджено тестові маршрути */test-jwt* та */simple-test*, які забезпечують засоби для розробників перевірити процес створення, декодування та валідації JWT-токенів.

Також приділено увагу гнучкості архітектури: секретний ключ задається через змінні середовища, що дозволяє мінімізувати ризики його компрометації під час публікації коду. Також враховано особливості різних версій бібліотеки *PyJWT*, яка може повертати представлення токена як у вигляді байтів, так і тексту.

4.1.5 Обробка аудіоданих для якісного транскрибування

Аналіз механізму обробки даних і взаємодії з базою даних є ключовим аспектом роботи системи, що спеціалізується на транскрипції й діаризації аудіофайлів. Основний акцент у даній системі зроблено на трансформації вхідного аудіо у стандартизований формат, а також на застосуванні алгоритмів машинного навчання для генерації текстових розшифровок і ідентифікації мовців.

На початковому етапі обробки аудіофайл піддається нормалізації. Метод audio_normalize здійснює конвертацію звукового сигналу в монофонічний формат WAV із частотою дискретизації 16 кГц. Крім цього, застосовується нормалізація рівня гучності, усунення низько- та високочастотних шумів, а також компресія динамічного діапазону, що сприяє досягненню максимальної чіткості звучання голосів навіть за несприятливих акустичних умов. Така передобробка забезпечує високу якість вхідних даних для наступних етапів обробки. Фрагмент коду реалізації цієї функції проілюстровано на рис. 4.30.

```python
def audio_normalize(self, audio_file: str) -> str:
    """Normalize audio to 16kHz mono WAV format"""
    try:
        audio = AudioSegment.from_file(audio_file)
        audio = audio.set_frame_rate(16000).set_channels(1)

        audio = audio.normalize()

        if len(audio) > 5000:
            audio = audio.low_pass_filter(4000)
            audio = audio.high_pass_filter(80)

        audio = audio.compress_dynamic_range(threshold=-20.0, ratio=2.0)

        normalized_file = tempfile.NamedTemporaryFile(delete=False, suffix='.wav')
        audio.export(normalized_file.name,
                     format="wav",
                     codec="pcm_s16le",
                     parameters=["-ar", "16000"])
```

Рисунок 4.30 – Реалізація функції нормалізації аудіо

Після завершення нормалізації відбувається процес транскрибування, тобто перетворення аудіо у текстовий формат. Для цього функція audio_to_text використовує модель Whisper, яка володіє багатомовною підтримкою та здатна працювати з часовими мітками слів (рис. 4.31). Завдяки оптимізації для роботи на GPU (перевіряється наявність CUDA), модель забезпечує високу швидкість і точність. Важливою особливістю цього підходу є збереження часових меж для кожного сегмента, що дозволяє чітко співвідносити слова із відповідними моментами запису.

```python
def audio_to_text(self, mediafile: str, model: str = 'base') -> Dict[str, Any]:
    """Transcribe audio to text using Whisper"""
    try:
        result = self.model.transcribe(
            mediafile,
            word_timestamps=True,
            fp16=(self.device == "cuda"),
            temperature=1,
        )

        # Verify we got the full audio duration
        audio = AudioSegment.from_file(mediafile)
        audio_duration = len(audio) / 1000.0  # in seconds

        return {
            'text': result['text'],
            'segments': result['segments'],
            'language': result['language']
        }
```

Рисунок 4.31 – Фрагмент коду з процесом транскрибування

Окрему увагу варто приділити діаризації — процесу, який забезпечує ідентифікацію різних мовців у запису. Ініціалізація пайплайну у self.diarization_pipeline здійснюється на базі моделі pyannote/speaker-diarization із вручну налаштованими параметрами сегментації, кластеризації та вікон ембедінгів. Цей підхід забезпечує точніше розпізнавання навіть у випадках коротких висловлювань чи накладання голосів. У методі diarization реалізовано механізм перевірки формату файлу, його наявності та обробки за допомогою попередньо підготовленого пайплайну. Як результат, формується об'єкт типу Annotation, який

містить часові інтервали та інформацію про ідентифікованих мовців. Фрагмент коду представлений на рис. 4.32.

```python
def diarization(self, audio_location: str) -> Annotation:
    """Perform speaker diarization on audio file"""
    if not self.diarization_pipeline:
        raise Exception("Diarization pipeline not initialized")

    try:
        print("Starting diarization...")

        if not isinstance(audio_location, (str, os.PathLike)):
            raise Exception(f"Expected file path, got {type(audio_location)}")

        audio_location = str(audio_location)

        if not os.path.exists(audio_location):
            raise Exception(f"Audio file not found: {audio_location}")

        # Basic diarization
        try:
            diarization = self.diarization_pipeline(audio_location)
            print("Basic diarization completed")
        except Exception as e:
            print(f"Basic diarization failed: {str(e)}")
            # Min params try
            try:
                diarization = self.diarization_pipeline(
                    audio_location,
                    min_speakers=1,
                    max_speakers=6
                )
```

Рисунок 4.32 – Реалізація діаризації

Клас Transcribe пропонує додаткові функції для оцінки якості аудіо. Метод check_audio_quality аналізує основні параметри запису, такі як тривалість, рівень гучності, динамічний діапазон, частоту дискретизації та кількість каналів. Окрім цього, він ідентифікує потенційні проблеми, які можуть негативно вплинути на якість діаризації, наприклад надто тихий або короткий аудіофайл. Застосування цього методу допомагає мінімізувати ризик виникнення помилок на наступних етапах обробки даних.

Метод diagnose_diarization реалізований для покрокової перевірки аудіо, його нормалізації та тестування різних конфігурацій діаризації (рис. 4.33).

```python
def diagnose_diarization(self, audio_file: str) -> None:
    """Run diagnostics on diarization to identify issues"""
    try:
        print("\n=== DIARIZATION DIAGNOSTICS ===\n")
        quality_report = self.check_audio_quality(audio_file)

        normalized_file = self.audio_normalize(audio_file)

        # Standaty diarization
        try:
            diarization = self.diarization(normalized_file)
            speakers = set()
            for _, _, speaker in diarization.itertracks(yield_label=True):
                speakers.add(speaker)
            print(f"   Detected {len(speakers)} speakers: {', '.join(speakers)}")
        except Exception as e:
            print(f"   Standard diarization failed: {str(e)}")

        try:
            diarization = self.diarization_pipeline(
                normalized_file,
                num_speakers=2,
                clustering="AgglomerativeClustering"
            )
            speakers = set()
            for _, _, speaker in diarization.itertracks(yield_label=True):
                speakers.add(speaker)
            print(f"   Detected {len(speakers)} speakers: {', '.join(speakers)}")
```

Рисунок 4.33 – Метод діагностики діаризації

Цей підхід є особливо важливим для виявлення недоліків у процесі розділення мовців. Крім того, він дозволяє порівнювати результати різних кластеризацій та обрати оптимальний варіант.

4.1.6 Взаємодія з базою даних

У межах створення вебсервісу для транскрипції аудіо особлива увага приділяється збереженню, обробці та подальшому використанню даних. Для цього була впроваджена повноцінна база даних, яка забезпечує надійне зберігання інформації про користувачів, завантажені аудіофайли та результати транскрипції. Усі дані організовані зі зручною структурою, включно з метаданими та розподілом за спікерами. У процес взаємодії з базою даних інтегровано фреймворк SQLAlchemy, який реалізує ORM-рівень і дає змогу працювати з об’єктами Python, уникаючи написання сирих SQL-запитів.

База даних проєкту побудована на основі PostgreSQL, що вирізняється стабільністю та підтримкою складних типів даних, таких як JSONB. Ця функціональність була використана для зберігання структурованої інформації про

спікерів. У модулі models.py представлені три основні сутності: User, Audio та Transcription. Кожна з цих моделей має необхідні поля для опису даних, встановлені взаємозв'язки, методи обробки інформації, а також функціонал для серіалізації, що спрощує передачу даних через API. Наприклад, модель User включає такі додаткові методи як set_password, check_password і to_dict, які використовуються в процесах аутентифікації та підготовки даних до передачі. На рис. 4.34 зображено фрагмент коду реалізації сутності User.

```python
class User(db.Model):
    """Таблиця користувачів для авторизації"""
    __tablename__ = 'users'

    id = db.Column(db.Integer, primary_key=True)
    uuid = db.Column(UUID(as_uuid=True), unique=True, nullable=False, default=uuid_lib.uuid4)

    # Основна інформація
    email = db.Column(db.String(120), unique=True, nullable=False, index=True)
    username = db.Column(db.String(80), unique=True, nullable=True)

    # Аутентифікація
    password_hash = db.Column(db.String(255), nullable=False)

    # Статус акаунту
    is_active = db.Column(db.Boolean, default=True)
    is_verified = db.Column(db.Boolean, default=False)
    is_admin = db.Column(db.Boolean, default=False)

    # Метадані
    last_login_at = db.Column(db.DateTime, nullable=True)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
    updated_at = db.Column(db.DateTime, default=datetime.utcnow, onupdate=datetime.utcnow)

    # Зв'язки з іншими таблицями
    audio_files = db.relationship('Audio', backref='user', lazy=True, cascade='all, delete-orphan')
    transcriptions = db.relationship('Transcription', backref='user', lazy=True, cascade='all, delete-orphan')
```

Рисунок 4.34 – Реалізація сутності User

Структура бази даних визначається за допомогою модулів SQLAlchemy, а всі зміни в її схемі керуються засобами Flask-Migrate. На рис. 4.35 проілюстровано використання Flask-Migrate для керування базою даних, а на рис. 4.36 представлено створена сутність User через перегляд БД PostgreSQL PgAdmin.

```
(venv) C:\Users\User\Documents\GitHub\Diploma>flask db migrate -m "Add speakers_json colum"
Summary:
JWT_SECRET type: <class 'str'>, value: valeria...
INFO  [alembic.runtime.migration] Context impl PostgresqlImpl.
INFO  [alembic.runtime.migration] Will assume transactional DDL.
INFO  [alembic.autogenerate.compare] Detected added table 'transcription'
INFO  [alembic.ddl.postgresql] Detected sequence named 'users_id_seq' as owned by integer column 'users(id)', assuming SERIAL and omitting
INFO  [alembic.ddl.postgresql] Detected sequence named 'audio_id_seq' as owned by integer column 'audio(id)', assuming SERIAL and omitting
Generating C:\Users\User\Documents\GitHub\Diploma\migrations\versions\c0d14b99d49c_add_speakers_json_colum.py ...  done

(venv) C:\Users\User\Documents\GitHub\Diploma>flask db upgrade
Summary:
Generating C:\Users\User\Documents\GitHub\Diploma\migrations\versions\c0d14b99d49c_add_speakers_json_colum.py ...  done
```

Рисунок 4.35 – Керування БД через Flask-Migrate

	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
	id	integer			✓	✓	nextval('users_i
	uuid	uuid			✓		
	email	character varying	120		✓		
	username	character varying	80				
	password_hash	character varying	255		✓		
	is_active	boolean					
	is_verified	boolean					
	is_admin	boolean					
	last_login_at	timestamp without time zone					
	created_at	timestamp without time zone					
	updated_at	timestamp without time zone					

Рисунок 4.36 – Модель User

У процесі транскрибування аудіофайли завантажуються користувачами, після чого створюється об'єкт моделі Audio, до якого додаються атрибути, зокрема шлях до файлу, розмір, тривалість, формат, дата створення та інші метадані. На рис. 4.37 представлено скріншот сутності Audio. Після завершення процесу транскрипції в базі даних автоматично створюється запис у таблиці Transcription, який містить текст транскрипції, його версію з розміткою за спікерами, JSON-репрезентацію часових міток та інформацію про використану мову. Зв’язок між моделями Audio та Transcription реалізовано через зовнішні ключі (рис. 4.38).

	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
	id	integer			✓	✓	nextval('aud
	uuid	uuid			✓		
	user_id	integer			✓		
	filename	character varying	255		✓		
	file_path	character varying	500		✓		
	file_size	integer					
	duration	double precision					
	format	character varying	50				
	created_at	timestamp without time z					

Рисунок 4.37 – Сутність Audio

```
# Зв'язки з іншими таблицями
transcriptions = db.relationship('Transcription', backref='audio', lazy=True, cascade='all, delete-orphan')
```

Рисунок 4.38 – Зв’язок між моделями Audio та Transcription

Логіка автоматичної реєстрації результатів у базі даних реалізована на серверній стороні, зокрема у файлі app.py, де прописано механізм обробки даних

(рис. 4.39). Після завершення транскрибування викликається функція, яка створює об'єкт Transcription, асоціює його із відповідним користувачем та аудіофайлом, додає необхідні метадані й ініціює збереження через db.session.add(...) та db.session.commit().

```python
# Створюємо запис для транскрипції
transcription = Transcription(
    uuid=tr_uuid,
    user_id=current_user.id,
    audio_id=audio.id,
    status="pending"
)
db.session.add(transcription)
db.session.commit()
```

Рисунок 4.39 – Фрагмент коду з реалізацією логіки запису результатів транскрипції

Одним із важливих технічних викликів було збереження JSON-структури з розміткою спікерів. Для цих потреб було обрано тип даних JSONB, що забезпечує ефективну фільтрацію, пошук та індексацію вкладених структур. Ця характеристика є критично важливою для оптимізації відображення результатів транскрипції в клієнтському інтерфейсі.

На рис. 4.40-4.42 проілюстровано скріншоти записів даних у сутності.

	id [PK] integer	uuid uuid	email character varying (120)	username character varying (80)	password_hash character varying (255)
1	2	0220c004-890a-415d-ae5a-21a147bf5...	valeria23122003@gmail.com	valeria_knv	scrypt:32768:8:1$iorvBTamkdGQX5M5$63cff290de4
2	3	882da89d-a677-4ffb-bd1f-bcba8a4ec5f3	valeriaknv12@gmail.com	Lera	scrypt:32768:8:1$0i85KD4ESXDx1P7a$ef8b4d018ae

Рисунок 4.40 – Записи полів сутності User

	id [PK] integer	uuid uuid	user_id integer	filename character varying (255)	file_path character varying (500)
1	3	71f84ec2-a305-4be3-93ec-b8b35c8a2c92	2	ef3e_elem_01a_1-03.mp3	uploads\71f84ec2-a305-4be3-93ec-b8b35c8a2c92_ef3e_e
2	5	0dd768b3-95a4-4cbe-a994-f063f6abb24f	2	ef3e_elem_01a_1-02.mp3	uploads\0dd768b3-95a4-4cbe-a994-f063f6abb24f_ef3e_el
3	6	88360264-def4-4d4d-a782-d28a48360017	2	ef3e_elem_02a_1-55.mp3	uploads\88360264-def4-4d4d-a782-d28a48360017_ef3e_e

Рисунок 4.41 – Записи полів сутності Audio

	id [PK] integer	uuid uuid	user_id integer	audio_id integer	text text
1	1	71f84ec2-a305-4be3-93ec-b8b35c8a2c92	2	3	1.3 What's your name? What's your phone number? ...see
2	3	0dd768b3-95a4-4cbe-a994-f063f6abb24f	2	5	1.2 1 Hi! I'm Mike What's your name? Hannah Sorry? Hanr
3	4	88360264-def4-4d4d-a782-d28a48360017	2	6	[null]

Рисунок 4.42 – Записи полів сутності Transcription

Для забезпечення гнучкості в роботі з моделями та налагодження взаємодії з базою даних були використані утиліти Flask-Shell і SQLAlchemy Query API. Це дозволило оперативно перевіряти цілісність даних, взаємозв'язки між таблицями та виконувати індивідуальні запити під час розробки.

4.2 Реалізація frontend-частини

4.2.1 Структура компонентів React та їх функціональність

Клієнтська частина застосунку реалізована з використанням бібліотеки React, що забезпечує компонентно-орієнтований підхід до побудови інтерфейсу. Архітектура програми є модульною та включає ключові компоненти: App, Header, Recorder і контекст мовної локалізації LanguageContext.

Компонент App є кореневим і забезпечує контекстне оточення для всього застосунку. Він ініціалізує тему оформлення (світлу або темну) за допомогою кастомного хука usePersistedState, що зберігає тему в localStorage. На рис. 4.43-4.44 відображений загальний інтерфейс користувача у різних темах, а на рис. 4.45 – фрагмент коду для визначення темної теми. Обраний дизайн передається через ThemeProvider з бібліотеки styled-components. Також застосунок обгортається в LanguageProvider, який забезпечує мультимовну підтримку.

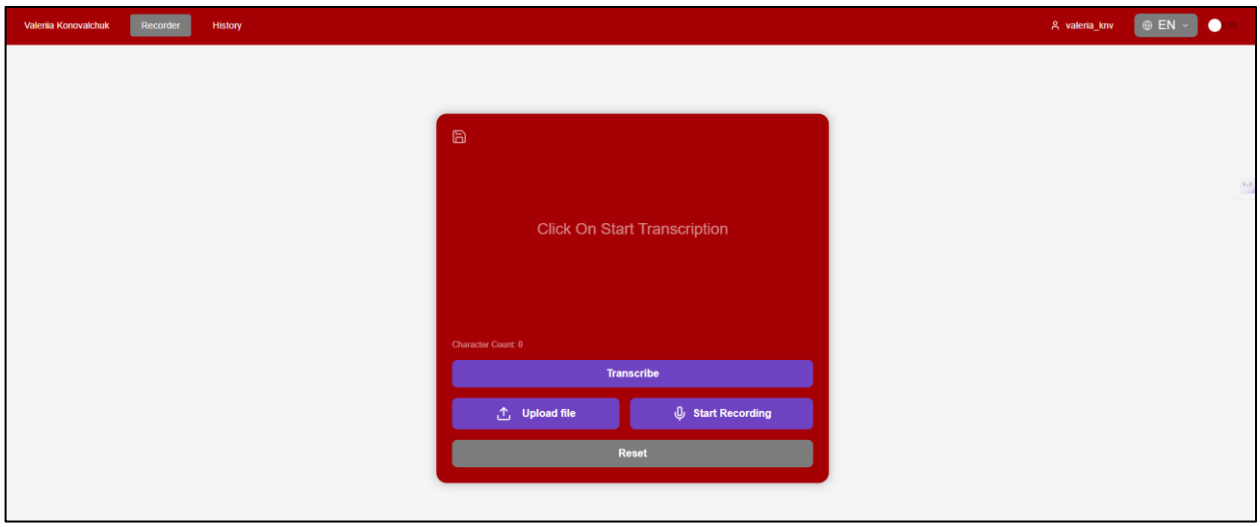


Рисунок 4.43 – Загальний інтерфейс користувача у світлій темі

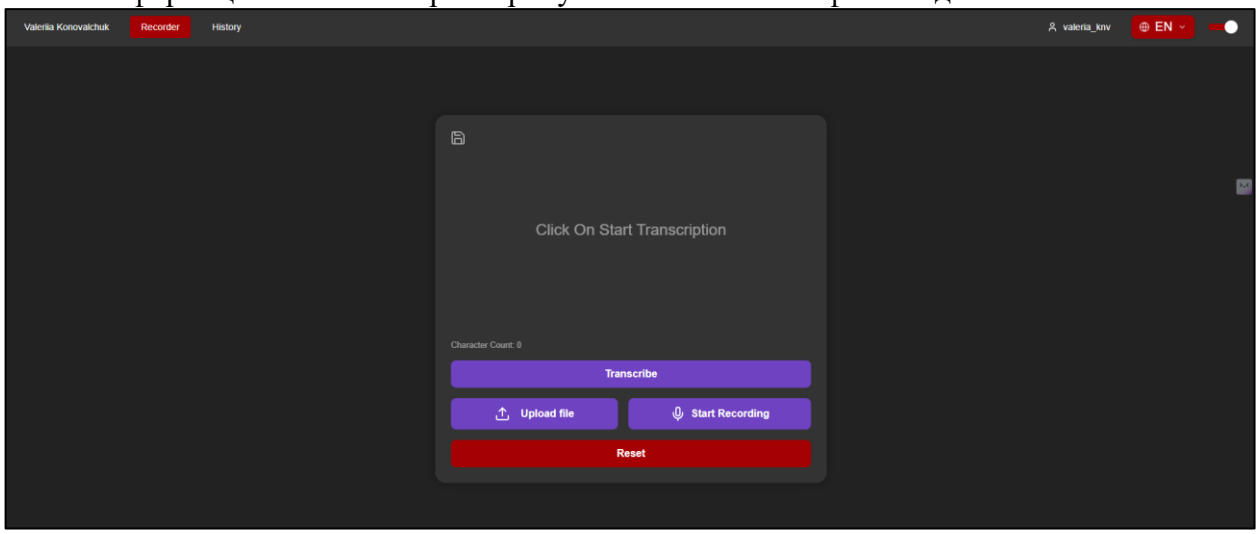


Рисунок 4.44 – Загальний інтерфейс користувача у темній темі

```
export default {
  title: 'dark',

  colors: {
    primary: '#333',
    primaryHover: '#444',
    secundary: '#A50104',
    secundaryHover: '#B91414',

    background: '#222',
    text: '#fff',

    accent: '#6f42c1',
    accentDark: '#5936a2',
  }
```

Рисунок 4.45 – Фрагмент коду визначення темної теми

У JSX-структурі App підключає компоненти Header (верхня панель з налаштуваннями) та Recorder (основна робоча зона для запису і транскрипції аудіо). Фрагмент коду: App.tsx з підключенням компонентів зображено на рис. 4.46.

```tsx
function App() {
  const [theme, setTheme] = usePersistedState<DefaultTheme>('theme', light);
  const [activeTab, setActiveTab] = useState<ActiveTab>("recorder")

  const toggleTheme = () => {
    setTheme(theme.title === 'light' ? dark : light);
  };

  return (
    <LanguageProvider>
      <AuthProvider>
        <ThemeProvider theme={theme}>
          <div className="App">
            <GlobalStyle />
            <ProtectedRoute>
            <Header toggleTheme={toggleTheme} activeTab={activeTab} setActiveTab={setActiveTab} />
              {activeTab === "recorder" && <Recorder />}
              {activeTab === "history" && <TranscriptionHistory />}
            </ProtectedRoute>
          </div>
        </ThemeProvider>
      </AuthProvider>
    </LanguageProvider>
  );
```

Рисунок 4.46 – Реалізація підключення головних компонентів у App.tsx

Header відповідає за елементи керування інтерфейсом: перемикач темної/світлої теми та селектор мови. Для перемикання теми використовується компонент Switch з бібліотеки react-switch, а кольори налаштовуються динамічно відповідно до активної теми через контекст ThemeContext. Мовна підтримка реалізована через LanguageSelector, який взаємодіє з LanguageContext. Реалізація цього функціоналу відображена на рис. 4.47.

```tsx
const Header: React.FC<Props> = ({toggleTheme, activeTab, setActiveTab }) => {
  const { colors, title } = useContext(ThemeContext) as DefaultTheme;
  const { t } = useLanguage()

  return (
    <Container>
      <div style={{ display: "flex", alignItems: "center", gap: "2rem" }}>
        {t("header.title")}

        <Navigation>
          <NavButton active={activeTab === "recorder"} onClick={() => setActiveTab("recorder")}>
            {t("navigation.recorder")}
          </NavButton>
          <NavButton active={activeTab === "history"} onClick={() => setActiveTab("history")}>
            {t("navigation.history")}
          </NavButton>
        </Navigation>
      </div>

      <ControlsWrapper>
        <UserProfile />
        <LanguageSelector />
```

Рисунок 4.47 – Реалізація зміни теми та мови у Header

Зміна мови відбувається за допомогою створеного компонента LanguageProvider та хука useLanguage (рис. 4.48). На рис. 4.49 зображено інтерфейс користувача українською мовою.

```tsx
export const useLanguage = () => {
  const context = useContext(LanguageContext)
  if (!context) {
    throw new Error("useLanguage must be used within a LanguageProvider")
  }
  return context
}

interface LanguageProviderProps {
  children: ReactNode
}

export const LanguageProvider: React.FC<LanguageProviderProps> = ({ children }) => {
  const [language, setLanguage] = useState<Language>("en")

  const t = (key: string): string => {
    const translationsForLanguage = translations[language] as Record<string, string>
    return translationsForLanguage[key] || key
  }

  const value = {
    language,
    setLanguage,
    t,
  }

  return <LanguageContext.Provider value={value}>{children}</LanguageContext.Provider>
```

Рисунок 4.48 – Фрагмент коду з реалізацією зміни мови

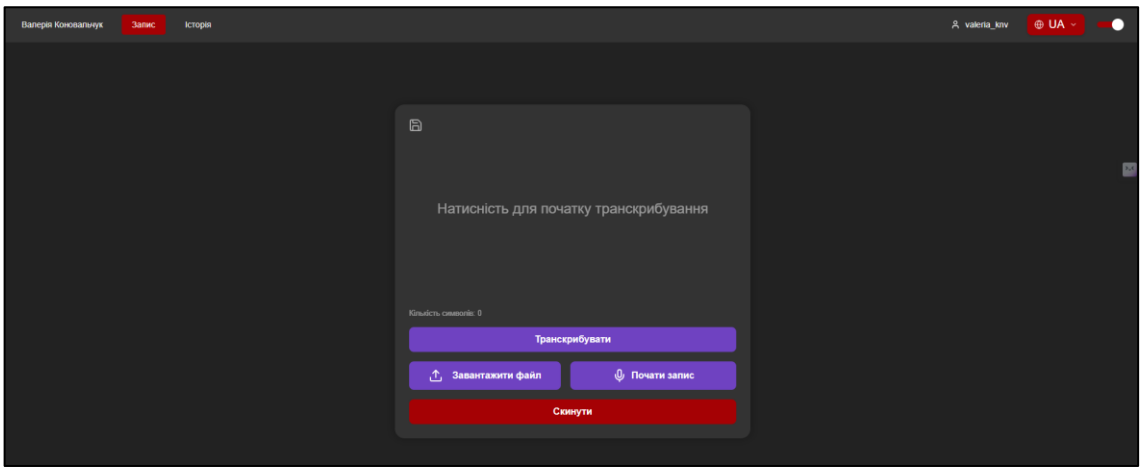


Рисунок 4.49 – Інтерфейс користувача українською мовою

Компонент Recorder є ключовим функціональним блоком застосунку. Він реалізує такі можливості, як:

– *запис аудіо*: через API MediaRecorder користувач може записати аудіо прямо в браузері. Отриманий запис зберігається як об’єкт Blob, потім перетворюється на файл File (рис. 4.50). На рис. 4.51 відображено інтерфейс користувача під час запису з мікрофону.

```
const startRecording = async () => {
  try {
    const stream = await navigator.mediaDevices.getUserMedia({ audio: true });
    const mediaRecorder = new MediaRecorder(stream);
    mediaRecorderRef.current = mediaRecorder;
    audioChunksRef.current = [];

    mediaRecorder.ondataavailable = (event) => {
      if (event.data.size > 0) {
        audioChunksRef.current.push(event.data);
      }
    };

    mediaRecorder.onstop = () => {
      const audioBlob = new Blob(audioChunksRef.current, { type: 'audio/wav' });
      setRecordedAudio(audioBlob);

      // Создаем файл из записанного аудио
      const audioFile = new File([audioBlob], `recording_${Date.now()}.wav`, { type: 'audio/wav' });
      setUploadedFile(audioFile);
      setUploadedFileName(audioFile.name);
      setTranscription({text: ""});
```

Рисунок 4.50 – Фрагмент коду з реалізацією запису з мікрофону

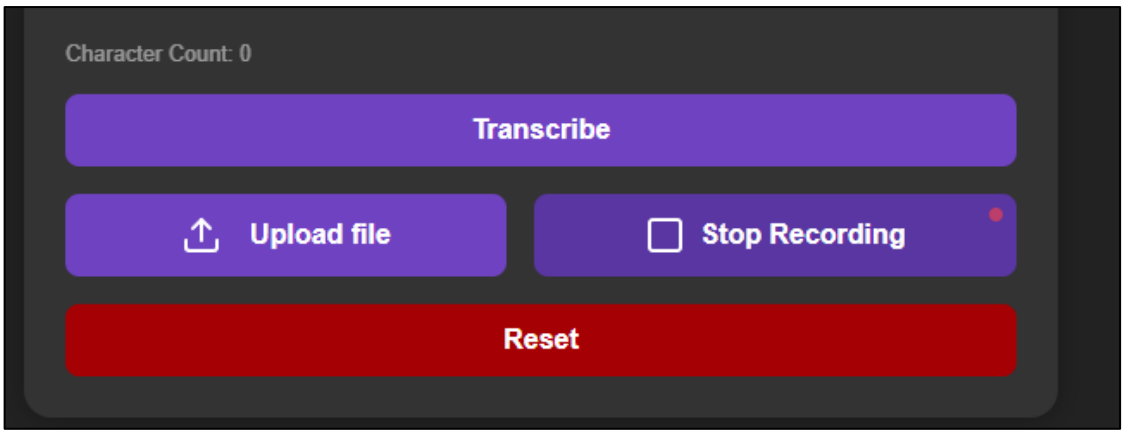


Рисунок 4.51 – Інтерфейс користувача під час запису з мікрофону

– *завантаження аудіофайлу*: підтримується завантаження файлів формату audio/*. Обраний файл зберігається у стані uploadedFile, після чого його можна відправити на сервер (рис. 4.52).

```
const handleUpload = (e: React.ChangeEvent<HTMLInputElement>) => {
  const file = e.target.files?.[0];
  if (file) {
    setUploadedFile(file);
    setUploadedFileName(file.name);
    setTranscription({text: ""});
    setRecordedAudio(null);
  }
};
```

Рисунок 4.52 – Фрагмент коду з реалізацією завантаження аудіофайлу

– *транскрибування*: вбудована функція handleTranscribe надсилає аудіофайл через fetch на бекенд. Отримана відповідь з текстом транскрипції зберігається у стані transcription (рис. 4.53). На рис. 4.54 відображено результат транскрибування.

```
try {
  const response = await fetch("http://localhost:5070/upload", {
    method: "POST",
    body: formData,
    headers: {
      'Accept': 'application/json',
      Authorization: `Bearer ${token}`,
    },
  })

  console.log("Response status:", response.status)

  if (!response.ok) {
    const errorData = await response.json();
    throw new Error(errorData.error || t("alert.transcriptionFailed"));
  }

  const data = await response.json();
  setTranscription(data);
```

Рисунок 4.53 – Фрагмент коду з реалізацією відображення результатів запиту транскрибування

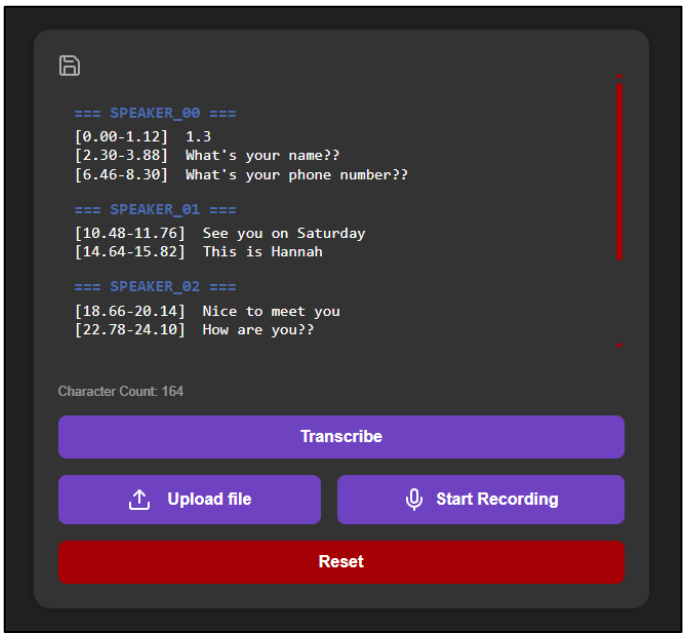


Рисунок 4.54 – Відображення результатів транскрибування

– *збереження*: дані можна завантажити у форматі .txt або .json (рис. 4.55). Ця опція реалізована через модальне вікно (ModalOverlay, ModalContent), що

відкривається при натисканні на кнопку завантаження (рис. 4.56). На рис 4.57 зображено створений текстовий файл з результатами транскрибування, а на рис. 4.58 – створений файл .json.

```
const downloadFile = (format: 'txt' | 'json') => {
  if (!transcription.text && !transcription.speakers_text) {
    alert(t("alert.noDataToDownload"));
    return;
  }

  let content: string;
  let filename: string;
  let mimeType: string;

  if (format === 'txt') {
    content = transcription.speakers_text || transcription.text;
    filename = `transcription_${Date.now()}.txt`;
    mimeType = 'text/plain';
  } else {
    content = JSON.stringify(transcription, null, 2);
    filename = `transcription_${Date.now()}.json`;
    mimeType = 'application/json';
  }

  const blob = new Blob([content], { type: mimeType });
  const url = URL.createObjectURL(blob);
  const link = document.createElement('a');
  link.href = url;
  link.download = filename;
  document.body.appendChild(link);
```

Рисунок 4.55 – Фрагмент коду з реалізацією модального вікна

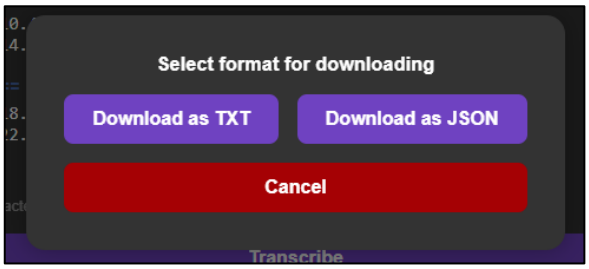


Рисунок 4.56 – Відображення модального вікна

```
=== SPEAKER_00 ===
[0.00-1.12]  1.3
[2.30-3.88]  What's your name??
[6.46-8.30]  What's your phone number??

=== SPEAKER_01 ===
[10.48-11.76]  See you on Saturday
[14.64-15.82]  This is Hannah

=== SPEAKER_02 ===
[18.66-20.14]  Nice to meet you
[22.78-24.10]  How are you??

=== SPEAKER_03 ===
[26.76-28.46]  I'm very well, thank you
[30.38-32.02]  And you?

=== SPEAKER_04 ===
[34.20-36.08]  Fine, thanks.
```

Рисунок 4.57 – Створений текстовий файл з результатами транскрибування

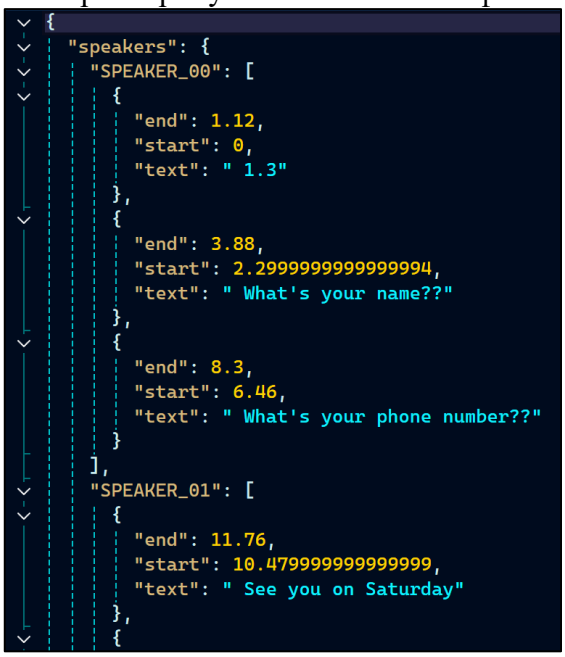


Рисунок 4.58 – Створений файл .json з результатами транскрибування

– *інтерфейсна логіка*: додатково реалізовано підрахунок символів, візуальні індикатори запису (RecordingIndicator) та обробку прокрутки (checkScroll), що визначає, чи потрібно показати скролбар. На рис. 4.59 відображено описані елементи.

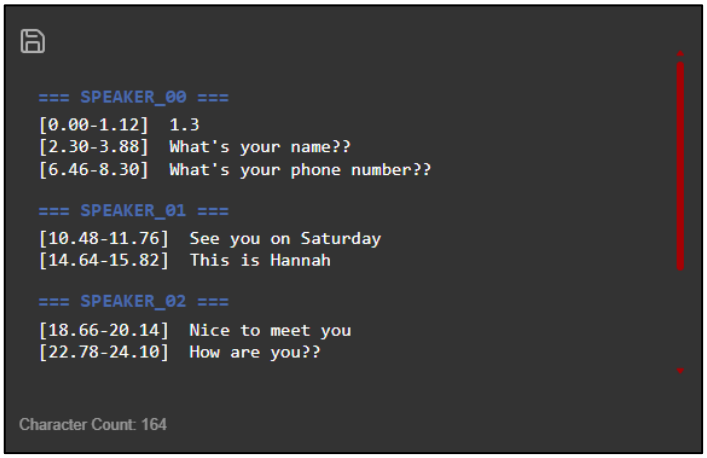


Рисунок 4.59 – Відображення інтерфейсної логіки

4.2.2 Аутентифікація користувачів

Аутентифікація користувачів — це один із ключових механізмів безпеки вебзастосунку, який забезпечує доступ до приватного функціоналу лише зареєстрованим користувачам. У розробленому застосунку реалізована повноцінна

система аутентифікації, яка включає можливість реєстрації нового користувача, авторизації вже зареєстрованого, а також збереження сесії користувача за допомогою токена. Уся логіка аутентифікації реалізована з використанням контексту auth-context.tsx, що дозволяє централізовано керувати станом користувача та його токеном у межах усього застосунку (фрагмент коду для вставки: auth-context.tsx, частина створення AuthContext, збереження токена та користувача).

Під час першого завантаження застосунку здійснюється перевірка наявності збережених у localStorage токена та даних користувача, що дозволяє зберігати сесію навіть після оновлення сторінки. Це реалізовано у useEffect хука всередині AuthProvider (рис. 4.60). Якщо токен знайдено, користувач автоматично вважається автентифікованим. Інакше йому пропонується пройти авторизацію.

```
useEffect(() => {
  // Перевіряємо збережений токен при завантаженні
  const savedToken = localStorage.getItem("auth_token")
  const savedUser = localStorage.getItem("auth_user")

  if (savedToken && savedUser) {
    try {
      setToken(savedToken)
      setUser(JSON.parse(savedUser))
    } catch (error) {
      console.error("Error parsing saved user data:", error)
      localStorage.removeItem("auth_token")
      localStorage.removeItem("auth_user")
    }
  }
  setIsLoading(false)
}, [])
```

Рисунок 4.60 – Фрагмент коду AuthProvider

Інтерфейс користувача представлений сторінкою AuthPage, яка динамічно перемикається між компонентами LoginForm та RegisterForm залежно від стану isLogin. Ці форми реалізовані як окремі компоненти, що дозволяє логічно розділити функціональність авторизації та реєстрації. У LoginForm.tsx при натисканні кнопки входу перевіряється коректність введених даних, і, якщо все вірно, викликається функція login із контексту Аналогічна логіка реалізована у компоненті

RegisterForm.tsx для створення нового акаунта, з додатковими перевірками відповідності пароля та підтвердження пароля. На рис. 61 зображено логіку зміни реєстрації та аутентифікації.

```
const handleSubmit = async (e: React.FormEvent) => {
  e.preventDefault()
  setError(null)

  if (!credentials.email || !credentials.password) {
    setError(t("auth.fillAllFields"))
    return
  }

  setIsLoading(true)
  try {
    await login(credentials)
  } catch (error) {
    setError(error instanceof Error ? error.message : t("auth.loginFailed"))
  } finally {
    setIsLoading(false)
  }
}
```

Рисунок 4.61 – Логіка зміни реєстрації та аутентифікації

На рис. 4.62-4.63 зображені форми реєстрації та аутентифікації.

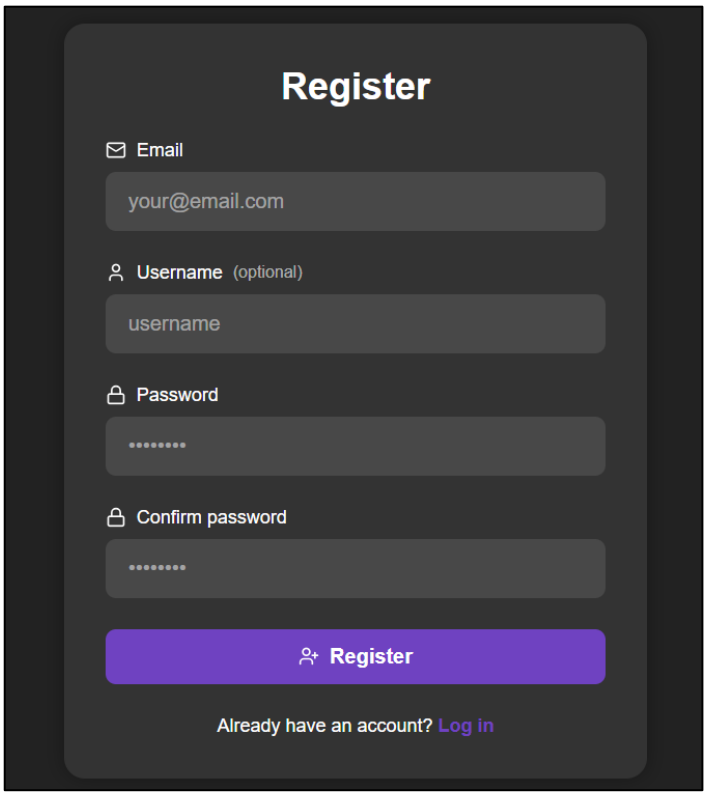


Рисунок 4.62 – Форма реєстрації

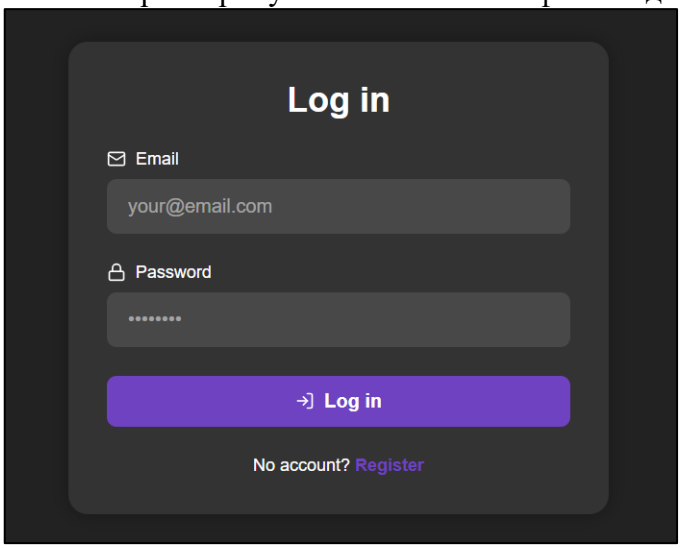


Рисунок 4.63 – Форма аутентифікації

Доступу до головної сторінки у зареєстрованого користувача не буде, поки він не верифікує електрону пошту. Після введення даних, відкривається сторінка з успішною реєстрацією, але повідомляється, що необхідно підтвердити пошту (рис. 4.64).

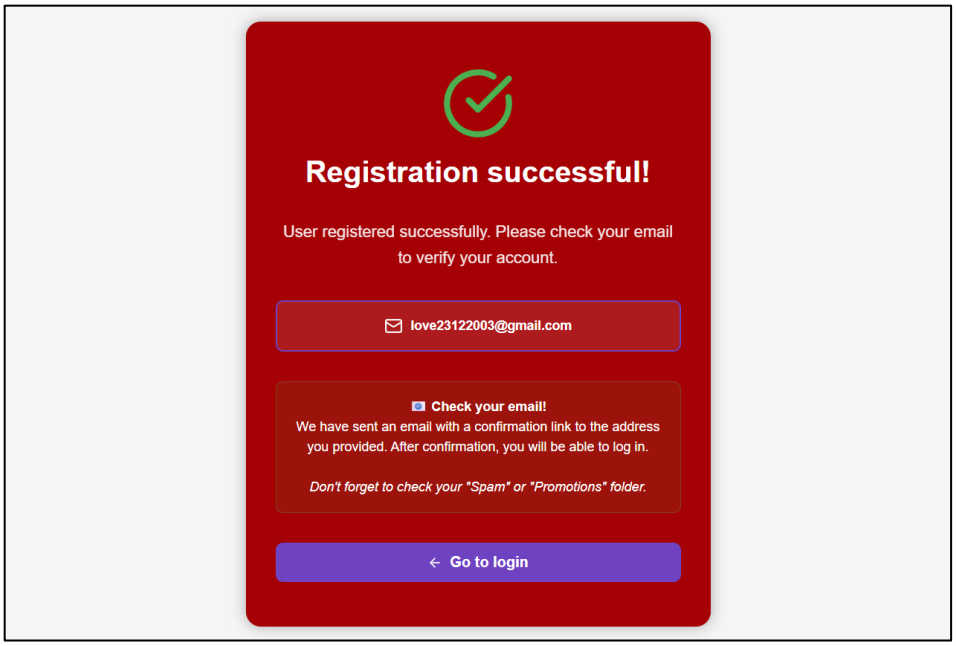


Рисунок 4.64 – Форма необхідності підтвердження пошти

Відповідно на пошту надсилається лист (рис. 4.65) з кнопкою підтвердження, на яку користувач повинен натиснути для верифікації. Після цього відкривається сторінка успішної верифікації (рис. 4.66).

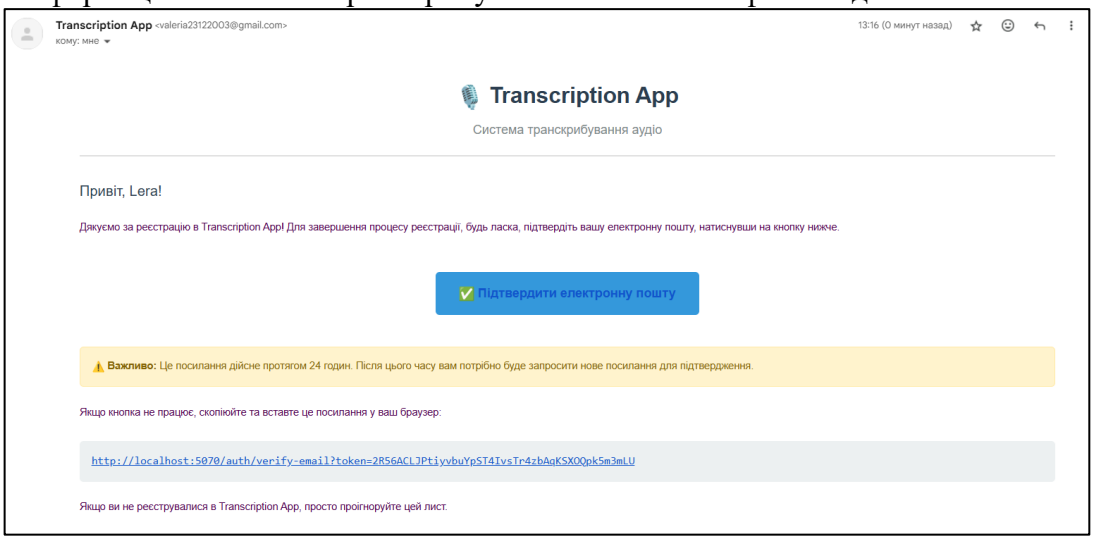


Рисунок 4.65 – Лист з підтвердженням пошти

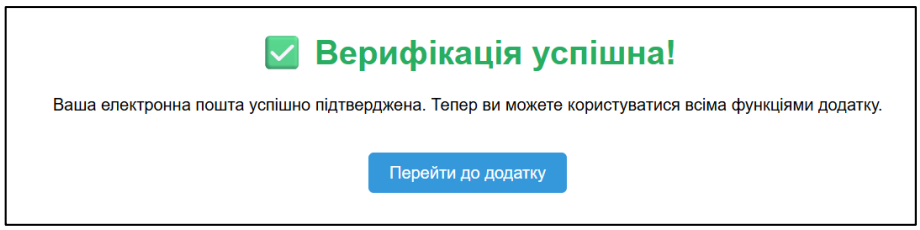
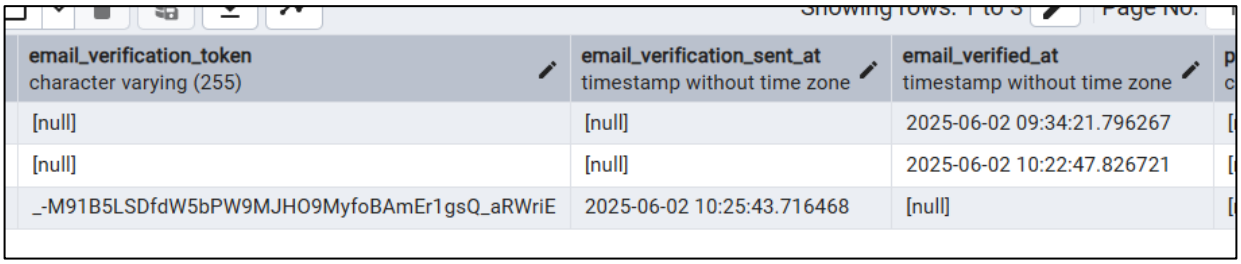


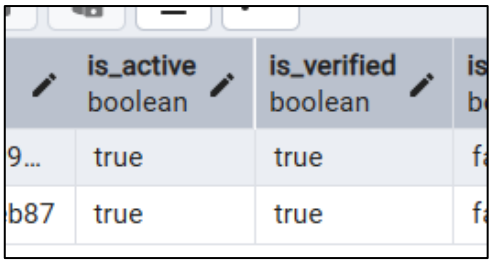
Рисунок 4.66 – Верифікація успішна

Відповідні зміни зберігаються в БД. На рис. 4.67 зображено БД з згенерованим токеном для верифікації, а на рис. 4.68 – відмітка з тим, що користувач верифікований.



email_verification_token character varying (255)	email_verification_sent_at timestamp without time zone	email_verified_at timestamp without time zone
[null]	[null]	2025-06-02 09:34:21.796267
[null]	[null]	2025-06-02 10:22:47.826721
_-M91B5LSDfdW5bPW9MJHO9MyfoBAmEr1gsQ_aRWriE	2025-06-02 10:25:43.716468	[null]

Рисунок 4.67 – Згенерований токен верифікації в БД



	is_active boolean	is_verified boolean	is
9...	true	true	f
b87	true	true	f

Рисунок 4.68 – Верифіковані користувачі

Доступ до закритих сторінок реалізований через компонент ProtectedRoute. Він перевіряє, чи автентифікований користувач, і, якщо ні — перенаправляє його на сторінку авторизації (рис. 4.69). Це дозволяє легко обмежити доступ до будь-якого маршруту, просто обгорнувши його у ProtectedRoute.

```tsx
const ProtectedRoute: React.FC<ProtectedRouteProps> = ({ children }) => {
  const { isAuthenticated, isLoading } = useAuth()
  const { t } = useLanguage()

  if (isLoading) {
    return <div>Loading...</div>
  }

  if (!isAuthenticated) {
    return <AuthPage />
  }

  return <>{children}</>
}
```

Рисунок 4.69 – Логіка ProtectedRoute

Після входу користувач бачить свій профіль, реалізований у компоненті UserProfile. Тут можна переглянути email та ім’я користувача (якщо воно задано), вийти з акаунта, а також перейти до історії, якщо вона реалізована. Це візуальне підтвердження авторизованого стану, яке водночас дає змогу керувати сесією (фрагмент коду для вставки: UserProfile.tsx, частина з logout та відображенням email користувача).

Аутентифікація також тісно пов’язана з локалізацією, оскільки всі повідомлення про помилки, підказки та кнопки перекладаються відповідно до вибраної мови за допомогою хука useLanguage. Це підвищує зручність користування системою авторизації для широкої аудиторії.

4.2.3 Реалізація інтерфейсу користувача для роботи з аудіо

Основна мета інтерфейсу — забезпечити користувачу зручний та інтуїтивно зрозумілий спосіб завантаження, перегляду та керування аудіозаписами, а також доступ до транскрипцій.

Інтерфейс реалізовано у вигляді веб-додатку з адаптивним дизайном, що забезпечує доступ із різних пристроїв. На рис. 4.70 зображено скріншот інтерфейсу користувача для мобільних пристроїв.

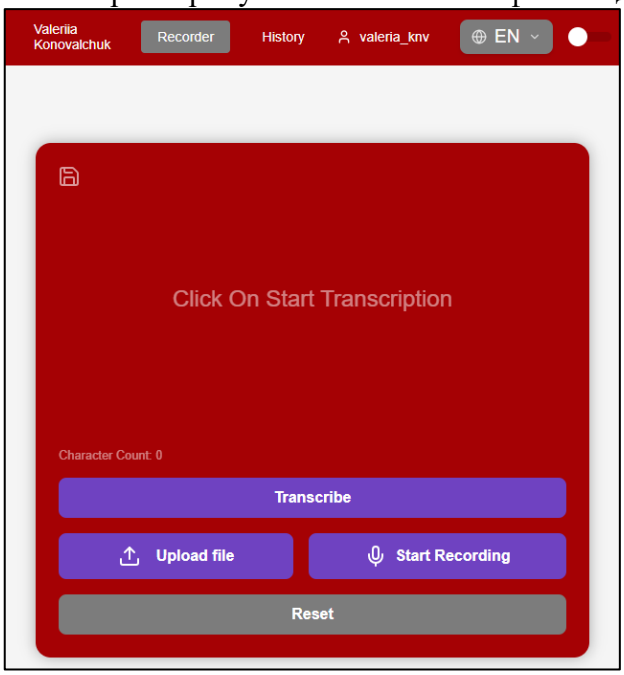


Рисунок 4.70 – Адаптація інтерфейсу для мобільних пристроїв

Користувач після авторизації потрапляє на головну сторінку. Звідти у хедері можна відкрити особистий кабінет (рис. 4.71).

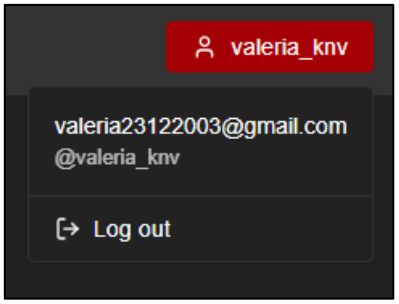


Рисунок 4.71 – Модальне вікно особистого кабінету

Також користувач має можливість з головної сторінки відкрити історію транскрибування, де розміщено список усіх завантажених аудіофайлів (рис. 4.64). Кожен запис у списку містить базову інформацію: назву файлу, дату завантаження, тривалість, розмір та статус транскрипції (наприклад, «обробляється», «завершено», «помилка»).

Також реалізований пошук за ключовими словами та статусом транскрибування. Скріншоти виконання цих операцій можна побачити на рис. 4.72-4.74.

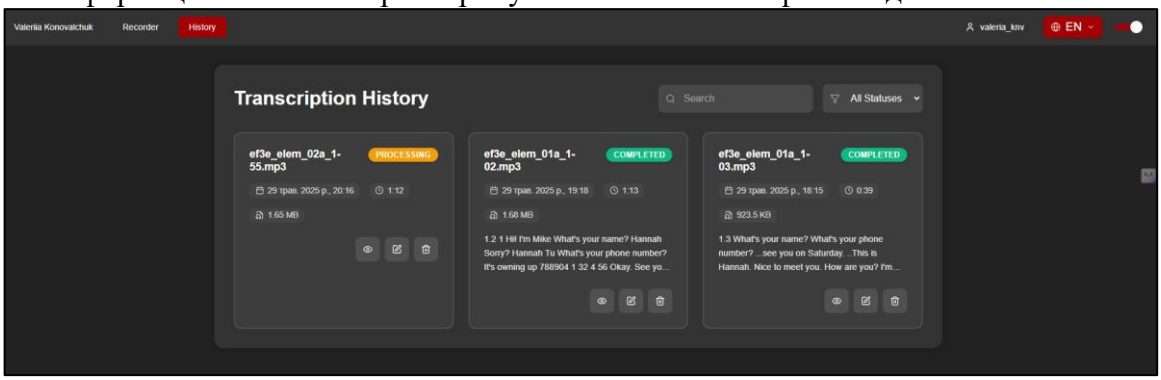


Рисунок 4.72 – Історія транскрибування в інтерфейсі користувача

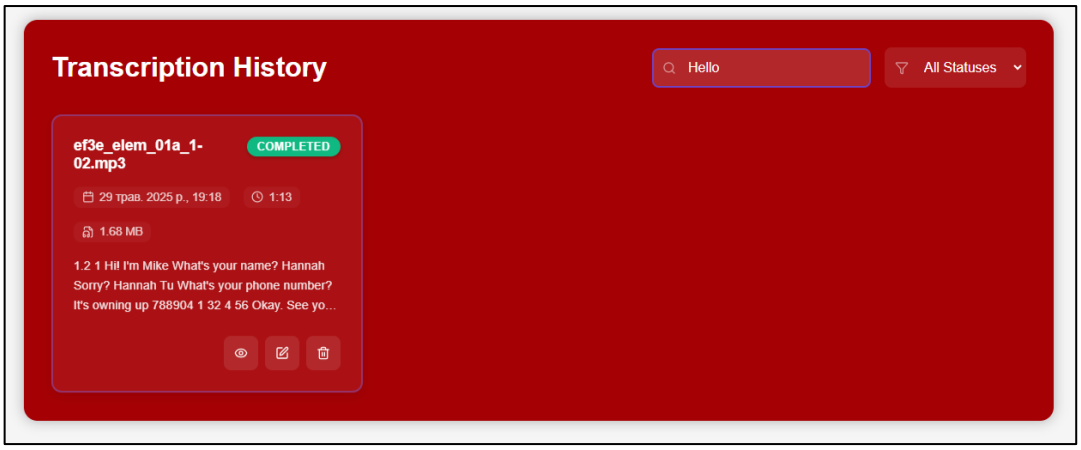


Рисунок 4.73 – Пошук за ключовими словами в історії

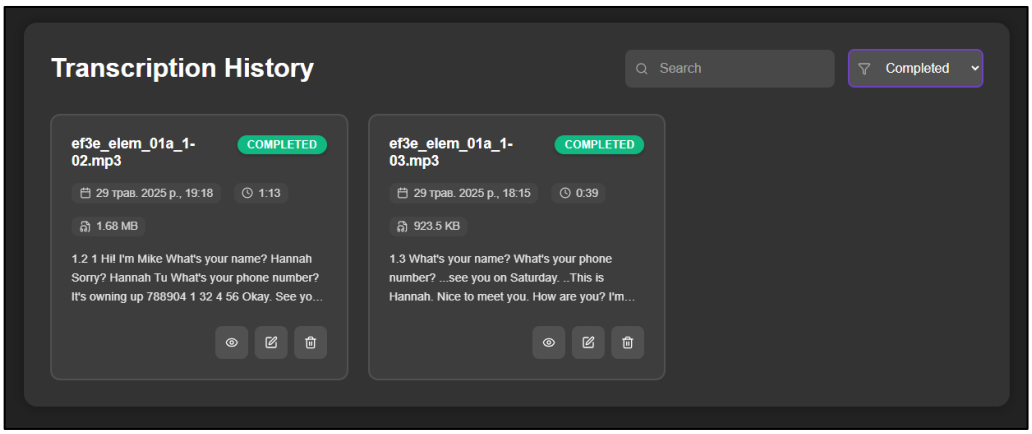


Рисунок 4.74 – Пошук за статусом виконання транскрибування

Також користувач може, окрім як зберегти, переглянути створений текст, змінити його та видалити зі своєї історії. Для видалення додатково реалізовано модальне вікно підтвердження видалення результату транскрибування з історії. На рис. 4.75 відображено модальне вікно підтвердження видалення результату транскрибування, а на рис. 4.76 – результат видалення.

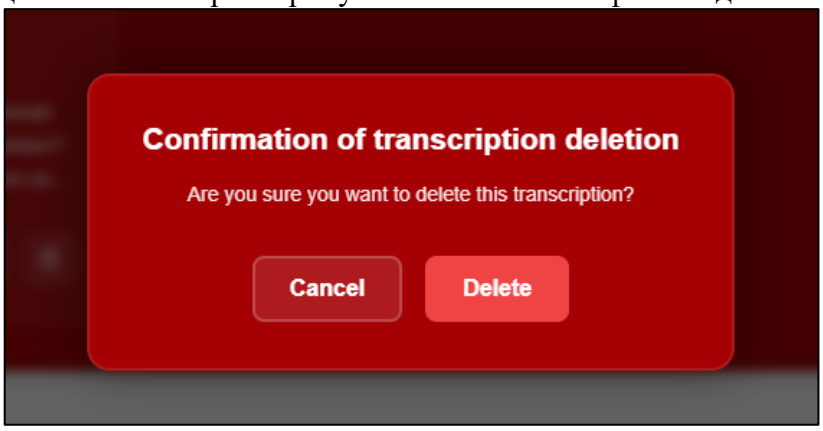


Рисунок 4.75 – Модальне вікно підтвердження видалення результату транскрибування

Рисунок 4.76 – Результат видалення

У компоненті з транскрибування розміщено кнопку для завантаження нового файлу. Після вибору файлу необхідно натиснути на кнопку «Transcribe», яка розпочинає процес обробки, про що інформує користувача відповідним індикатором. Успішна транскрипція виводить текст транскрибування для перегляду з розділенням за мовцями, якщо така функція активна.

Також користувач має можливість записувати з мікрофону аудіофайл, який пізніше можна транскрибувати і побачити створений текст. Додатково реалізовано можливість завантаження отриманого тексту у форматах TXT або DOCX.

Інтерфейс містить також індикатори стану (кольорові мітки або піктограми) та випадаюче меню для керування кожним записом — перегляду, редагування, видалення або повторної обробки.

4.2.4 Взаємодія з backend API

Фронтенд-застосунок реалізовує взаємодію з бекенд-сервером через HTTP-запити, які виконуються за допомогою вбудованої функції fetch. Основний процес комунікації відбувається під час передачі аудіофайлу на сервер для обробки та отримання результату транскрипції. Цей етап є ключовим у загальній архітектурі системи, оскільки забезпечує зв’язок між інтерфейсом користувача та обчислювальним ядром.

У компоненті Recorder.tsx реалізовано метод handleTranscribe, який:

– формує об’єкт FormData, що містить аудіофайл (цей файл може бути як завантажений вручну, так і записаний безпосередньо через браузер);

– надсилає POST-запит на бекенд-ендпоінт /api/transcribe із сформованим об'єктом FormData як тілом запиту;

– очікує на відповідь у форматі JSON, яка включає транскрибований текст, а також метаінформацію (наприклад, дані про спікерів або часові мітки);

– зберігає отриманий результат у локальному стані transcription і виводить його на екран для перегляду користувачем.

Під час запиту можуть передаватися параметри, які впливають на спосіб обробки. У майбутньому ці параметри можуть бути винесені в окремі форми-контролі для зручності користувача. На рис. 4.77 відображено час обробки запиту транскрибування аудіофайлу довжиною 46 секунд. Час обробки становив менше 5 секунд.

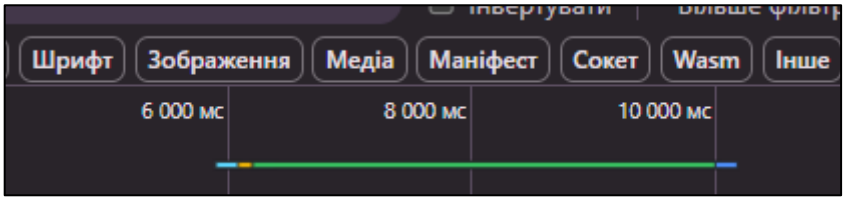


Рисунок 4.77 – Час обробки запиту транскрибування аудіо довжиною 46 секунд.

У разі виникнення помилки серверного запиту передбачено базову обробку винятків: інтерфейс сповіщає користувача про проблему у вигляді повідомлення,

яке може бути реалізовано через alert, Toast або модальне вікно. На рис. 4.78 відображено інформацію про результати запиту upload (статус, тип та час).

Рисунок 4.78 – Результати запиту

Загалом бекенд функціонує через API у REST-стилі. Аудіо передається як мультичастинний запит, а відповідь надходить у структурованому форматі, готовому для відображення на фронтенді.

4.3 Тестування системи

4.3.1 Опис процесу та стратегії тестування

Тестування вебзастосунку охоплювало декілька рівнів контролю якості з метою забезпечення надійності функціоналу, коректності інтерфейсу, стабільної взаємодії з бекендом та адаптивності на різних пристроях. Процес був розбитий на етапи модульного, інтеграційного, функціонального та візуального тестування. Табл. 4.2 надає зведену інформацію про покриття функціоналу тестами.

Таблиця 4.2 – Зведена інформація про тести

Функція	**Тестується?**	**Метод тестування**
Завантаження файлу	Так	Модульне
Запис аудіо	Так	Інтеграційне
Вивід транскрибування	Так	Модульне
Темна/світла тема	Так	Модульне
Помилка API	Так	Функціональне

На початковому етапі використовувалося модульне тестування компонентів за допомогою бібліотек Jest і React Testing Library. Для кожного ключового компоненту – Recorder, Header, App – створювалися тести, які перевіряли базову функціональність. Зокрема, тестувалося натискання кнопок, відображення тексту,

виклик функцій зміни теми або мови, обробка подій запису голосу та завантаження файлів. Також перевірялися нештатні ситуації – наприклад, відсутність мікрофона або невідповідний формат файлу. Важливим аспектом було тестування поведінки застосунку у випадку помилок з боку API, зокрема при отриманні некоректної відповіді чи повній її відсутності.

На наступному етапі виконувалося інтеграційне тестування. Воно включало повну симуляцію користувацького сценарію: запис аудіо або завантаження файлу, відправка даних на сервер, отримання транскрипції та збереження її у форматах .txt і .json. Таким чином перевірялась взаємодія між компонентами інтерфейсу та серверною логікою.

Окрему увагу було приділено перевірці адаптивності інтерфейсу. Тестування проводилося у сучасних браузерах – Google Chrome, Mozilla Firefox, Microsoft Edge. Рис. 4.79 показує відображення застосунку в Microsoft Edge.

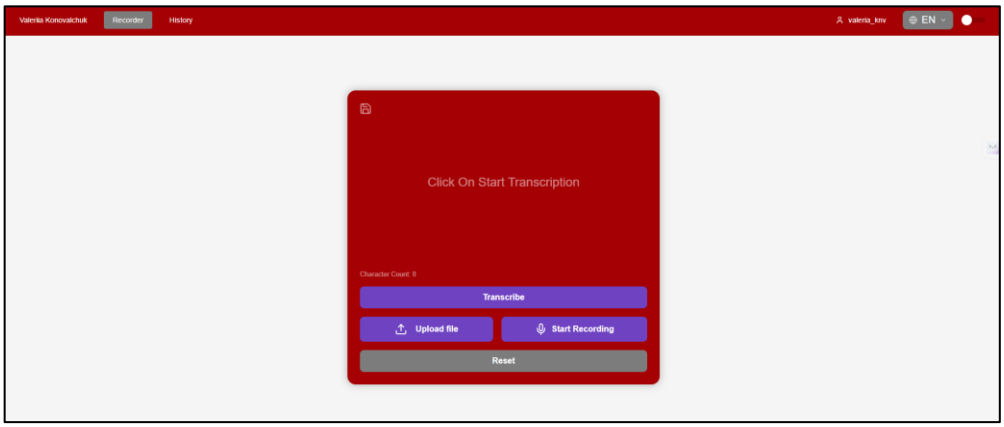


Рисунок 4.79 – Відображення системи в Microsoft Edge

Інтерфейс програми перевірявся на різних розмірах екранів, включаючи мобільні телефони, планшети та десктопи. Це дозволило виявити та усунути помилки відображення, забезпечивши однакову зручність користування незалежно від пристрою.

З метою фіксації результатів тестування та підтвердження його повноти додається скріншот із результатами запуску тестів у терміналі, де показано статус виконання, кількість тестів та покриття коду. Окрім того, у таблиці наведено матрицю покриття функціональностей тестами з вказанням типу перевірки. Для

ілюстрації кросбраузерного тестування додаються знімки інтерфейсу у браузерах Chrome і Safari. Також представлений графік тривалості проходження основних сценаріїв тестування, що дозволяє оцінити ефективність та швидкодію системи.

4.3.2 Результати інтеграційного та системного тестування

Після успішного завершення юніт-тестування окремих модулів наступним ключовим етапом стає інтеграційне та системне тестування. Ці види тестування дають змогу перевірити, наскільки злагоджено всі частини системи функціонують разом у середовищі, максимально наближеному до реальних умов використання. Головна мета інтеграційного тестування полягає в переконанні, що бекенд, фронтенд і база даних бездоганно взаємодіють, формуючи єдину цілісну систему. Системне тестування, своєю чергою, спрямоване на оцінку загальної працездатності додатка, його стабільності, продуктивності та поведінки як у звичайних, так і у складних сценаріях.

В ході інтеграційного тестування проводився повний цикл перевірки: завантаження аудіофайлу через інтерфейс, передача цього файлу на сервер, обробка класами Transcribe, збереження інформації у відповідних таблицях бази даних і повернення результату на фронтенд. Одним із ключових сценаріїв стало тестування маршруту POST /api/upload, який відповідає за передачу файлу, ініціювання транскрипції та створення відповідних записів у таблицях Audio і Transcription. Після завершення процесу транскрипції перевірялася відповідність тексту, збереженого в базі даних, тому тексту, який повертається клієнту, а також правильність структури JSON для визначених спікерів. Тестування виконувалося із застосуванням бібліотеки pytest і тестового клієнта Flask із додатковим використанням мокування для симуляції тривалих операцій.

Прикладом системного тестування може стати тест роботи бібліотеки JWT. На рис. 4.80-4.81 зображено реалізацію маршрутів для перевірки роботи JWT при створенні токену для користувачів.

```python
@auth_bp.route('/test-jwt', methods=['GET'])
def test_jwt():
    """Тестовий маршрут для перевірки роботи JWT"""
    try:
        # Створюємо тестовий токен з правильним timestamp
        exp_time = datetime.datetime.utcnow() + datetime.timedelta(minutes=5)
        test_payload = {
            'user_id': 1,
            'test': 'data',
            'exp': int(exp_time.timestamp())  # Використовуємо timestamp
        }

        print(f"Test payload: {test_payload}")

        test_token = jwt.encode(test_payload, JWT_SECRET, algorithm='HS256')

        # Декодуємо токен
        decoded = jwt.decode(test_token, JWT_SECRET, algorithms=['HS256'])
```

Рисунок 4.80 – Фрагмент коду з реалізацією тесту маршруту для перевірки JWT

```python
@auth_bp.route('/simple-test', methods=['GET'])
def simple_test():
    """Найпростіший тест JWT"""
    try:
        # Найпростіший payload
        payload = {
            'user_id': 123,
            'exp': int(time.time()) + 3600  # Через годину
        }

        # Кодуємо
        token = jwt.encode(payload, 'secret', algorithm='HS256')

        # Декодуємо
        decoded = jwt.decode(token, 'secret', algorithms=['HS256'])

        return jsonify({
            'status': 'success',
            'original': payload,
            'token': token,
            'decoded': decoded
        })
```

Рисунок 4.81 – Реалізація простого тесту генерації токенів

На рис. 4. 82 відображено скріншот фрагменту коду тестування маршрутів, а на рис. 4.83 – результати тестування.

```python
def test_jwt_with_different_types():
    """Тестує JWT з різними типами даних"""
    test_cases = [
        ('string', 'test_string'),
        ('number', 123),
        ('boolean', True),
        ('list', [1, 2, 3]),
        ('dict', {'key': 'value'}),
        ('none', None),
    ]

    results = {}

    for name, value in test_cases:
        try:
            payload = {'test_key': value}
            token = jwt.encode(payload, SECRET_KEY, algorithm='HS256')
            decoded = jwt.decode(token, SECRET_KEY, algorithms=['HS256'])
```

Рисунок 4.82 – Фрагмент коду тестування JWT для різних типів даних

```
=== Тестування JWT ===
Оригінальний payload: {'user_id': 123, 'email': 'test@example.com', 'exp': datetime.datetime(2025, 5, 31, 0, 35, 49, 967830)}
Закодований токен: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoxMjMsImVtYWlsIjoidGVzdEBleGFtcGxlLmNvbSIsImV4cCI6MTc0ODY
PR572ftQ4unXSjvKf7hgk
Декодований payload: {'user_id': 123, 'email': 'test@example.com', 'exp': 1748651749}
Базовий тест пройдено успішно!

=== Тестування різних типів даних ===
Тест для типу string пройдено успішно!
Тест для типу number пройдено успішно!
Тест для типу boolean пройдено успішно!
Тест для типу list пройдено успішно!
Тест для типу dict пройдено успішно!
Тест для типу none пройдено успішно!

=== Тестування завершено ===
```

Рисунок 4.83 – Результати тестування

У рамках системного тестування здійснювалася перевірка основних функціональних можливостей з позиції кінцевого користувача. Було перевірено роботу авторизації, реєстрації, завантаження файлів, перегляду транскрипцій, розпізнавання спікерів, редагування тексту та збереження змін. Особливий акцент зроблено на тестування застосунку для різних типів доступу: анонімних користувачів, зареєстрованих користувачів та адміністраторів. Для моделювання взаємодії з інтерфейсом використовувалися браузерні інструменти автоматизації, такі як Playwright або Selenium. Це забезпечило можливість протестувати не лише бізнес-логіку, але й стабільність інтерфейсу під навантаженням, а також відповідність дизайну очікуваному функціоналу.

У ході тестування було виявлено та усунено кілька критичних недоліків. Зокрема, траплялися випадки, коли у разі збою транскрипції в таблиці Transcription

залишався "порожній" запис. Ця проблема не впливала на відображення в інтерфейсі користувача, але спричиняла плутанину в базі даних. Для вирішення було додано перевірку статусу транскрипції перед записом в базу даних, що дозволило уникнути подібних помилок у майбутньому.

4.3.3 Оцінка продуктивності та стабільності системи

Оцінювання продуктивності та стабільності програмної системи становить критично важливий аспект тестування, що спрямований на виявлення потенційних вузьких місць у її функціонуванні під впливом різного рівня навантаження. Це дозволяє забезпечити оптимальну швидкодію, а також передбачити поведінку системи за умов інтенсивної експлуатації. У ході проведених випробувань було здійснено аналіз часу обробки запитів та витрат ресурсів під час виконання ключових операцій, зокрема таких, як завантаження аудіофайлів, їх обробка, транскрипція, збереження отриманих результатів та їх відтворення у фронтенд-інтерфейсі. Крім цього, увага приділялася тривалій стабільності роботи системи без потреби у перезапуску сервера, а також її здатності реагувати на багаторазові однотипні запити, що відрізнялися за інтенсивністю.

У межах тестувального етапу було розроблено кілька сценаріїв навантаження із використанням утиліти Locust, яка забезпечує моделювання великої кількості одночасних користувачів, котрі надсилають запити до API. Результати показали середні та максимальні значення часу відповіді для різних ендпоінтів. Зокрема, тестуванню підлягала швидкість реакції сервера на запити до маршруту /api/upload, а також його стабільність за участі 50, 100 та 200 паралельних користувачів. Отримані показники часу відповіді коливалися в діапазоні від 500 мс до 2 секунд залежно від розміру аудіофайлу та рівня навантаження системи, що відповідає прийнятним стандартам для вебзастосунків відповідного класу.

Особливу увагу було зосереджено на процесі транскрипції, який виявився найбільш ресурсоємким серед усіх етапів. При паралельній обробці декількох файлів середньої тривалості (3–5 хвилин) система потребувала від 10 до 20 секунд для завершення обробки одного запиту, включно з діаризацією та збереженням у

базу даних. Аналіз демонструє, що за допомогою паралельної обробки запитів із Flask при налаштуванні threaded=True сервер здатний підтримувати до 20 транскрипцій одночасно без збоїв і зниження продуктивності.

Стабільність системи була ретельно випробувана шляхом тривалого запуску бекенду із постійним потоком запитів. Протягом 8 годин безперервної роботи не було виявлено жодних витоків пам’яті або збоїв процесів. Це свідчить про високу надійність обраної архітектури, а також про ефективне управління такими ресурсами, як бази даних, зовнішні бібліотеки для транскрипції (Whisper) і файлові операції.

Висновки до розділу 4

У цьому розділі було детально розглянуто процес програмної реалізації та тестування інформаційної системи транскрибування. Було представлено архітектуру backend-частини, що включає розробку REST API за допомогою Flask, інтеграцію з зовнішніми сервісами транскрибування, реалізацію системи аутентифікації користувачів та обробку аудіофайлів для покращення якості розпізнавання. Окрема увага приділялася взаємодії з базою даних, де описано логіку зберігання, вибірки та оновлення інформації, необхідної для роботи системи.

Frontend-частина проєкту реалізована з використанням React, і в розділі наведено структуру компонентів, їхню функціональність та особливості взаємодії з backend API. Також описано користувацький інтерфейс для завантаження, прослуховування та перегляду результатів транскрипції.

У підрозділах щодо тестування висвітлено застосовану стратегію перевірки системи, результати інтеграційного та системного тестування, а також оцінено продуктивність і стабільність роботи застосунку.

ВИСНОВКИ

У результаті виконання дипломної роботи було проведено всебічний аналіз предметної області автоматичного транскрибування аудіозаписів, зокрема телефонних дзвінків. Було вивчено існуючі рішення на ринку, їхні переваги та недоліки, що дозволило визначити актуальність розробки власної інформаційної системи з фокусом на зручність, точність та швидкість обробки мовленнєвих даних.

На основі сформульованих вимог було спроєктовано вебзастосунок, що використовує сучасні технології на backend-і та frontend-і. Зокрема, для реалізації серверної частини обрано мову Python та фреймворк Flask, інтегровано бібліотеку OpenAI Whisper для якісного транскрибування, а також інструменти Torchaudio та pyannote.audio для сегментації мовлення. Забезпечено асинхронну обробку задач за допомогою Celery, що дозволяє ефективно працювати з великими обсягами аудіоданих.

Клієнтська частина застосунку реалізована на базі React і TypeScript, що забезпечує гнучкість і масштабованість інтерфейсу. Використано сучасні засоби стилізації (Tailwind CSS, styled-components), а також додаткові бібліотеки для роботи з мікрофоном і відображення графічних елементів. Система підтримує завантаження аудіо, їх обробку, відображення тексту, ідентифікацію мовців та зберігання результатів у базі даних.

Було виконано тестування системи, яке засвідчило її стабільність, прийнятну точність транскрибування та зручність у використанні. Оцінено продуктивність, ідентифіковано потенційні вузькі місця та надано рекомендації для майбутнього вдосконалення.

Отже, розроблена інформаційна система є практичним інструментом для автоматизованого транскрибування телефонних дзвінків і може бути ефективно впроваджена у службах підтримки, кол-центрах, юридичних установах тощо. Вона відповідає сучасним вимогам до ПЗ, є масштабованою, надійною та розширюваною.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Speech-to-Text documentation. *Google Cloud*. URL: https://cloud.google.com/speech-to-text/docs (Last accessed: 30.04.2025).

2. Шестакевич, Т., МИХАЙЛО, Л., & КОБИЛЮХ, Л. (2024). МОДЕЛЬ СИСТЕМИ МОНІТОРИНГУ МОВЛЕННЯ З ВИКОРИСТАННЯМ СИСТЕМ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ РОЗПІЗНАВАННЯ УКРАЇНСЬКОЇ МОВИ. Herald of Khmelnytskyi National University. Technical sciences, 335(3 (1)), 387-392. DOI: 10.31891/2307-5732-2024-335-3-52.

3. OpenAI. (2023). *Introducing ChatGPT and Whisper APIs*. URL: https://openai.com/index/introducing-chatgpt-and-whisper-apis/ (Last accessed: 30.04.2025).

4. Welcome to Deepgram Docs! – *Deepgram | Documentation*. URL: https://developers.deepgram.com/home/introduction (Last accessed: 30.04.2025).

5. Word Error Rate – *PyTorch-Metrics 1.7.1 documentation. Lightning AI*. URL: https://lightning.ai/docs/torchmetrics/stable/text/word_error_rate.html (Last accessed: 30.04.2025).

6. Char Error Rate – *PyTorch-Metrics 1.7.1 documentation. Lightning AI*. URL: https://lightning.ai/docs/torchmetrics/stable/text/char_error_rate.html (Last accessed: 30.04.2025).

7. GeeksforGeeks. Client-Server Model. URL: https://www.geeksforgeeks.org/client-server-model/ (Last accessed: 30.04.2025).

8. Python 3.13 documentation. *Python documentation*. URL: https://docs.python.org/3/ (Last accessed: 30.04.2025).

9. What is Sequence Diagram?. *Ideal Modeling & Diagramming Tool for Agile Team Collaboration*. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/ (Last accessed: 30.04.2025).

10. Welcome to flask. *Flask Documentation (3.1.x)*. URL: https://flask.palletsprojects.com/en/stable/ (Last accessed: 30.04.2025).

11. Celery - Distributed Task Queue – *Celery 5.5.2 documentation*. URL: https://docs.celeryq.dev/en/stable/ (Last accessed: 30.04.2025).

12. Torchaudio Documentation – *Torchaudio 2.7.0 documentation*. URL: https://pytorch.org/audio/stable/index.html (Last accessed: 30.04.2025).

13. GitHub - pyannote/pyannote-audio. URL: https://github.com/pyannote/pyannote-audio (Last accessed: 30.04.2025).

14. Ffmpeg documentation. URL: https://ffmpeg.org/ffmpeg.html (Last accessed: 30.04.2025).

15. Початок роботи – React – *JavaScript-бібліотека для створення користувацьких інтерфейсів*. URL: https://uk.legacy.reactjs.org/docs/getting-started.html (дата звернення: 30.04.2025).

16. The starting point for learning TypeScript. URL: https://www.typescriptlang.org/docs/ (Last accessed: 30.04.2025).

17. Documentation - tailwind CSS. - *Rapidly build modern websites without ever leaving your HTML*. URL: https://v2.tailwindcss.com/docs (Last accessed: 30.04.2025).

18. React-speech-recognition. URL: https://www.npmjs.com/package/react-speech-recognition (Last accessed: 30.04.2025).

19. Lucide icons. *Lucide*. URL: https://lucide.dev/guide/packages/lucide-react (Last accessed: 30.04.2025).

20. Zilliz. Getting started with audio data: processing techniques and key challenges. URL: https://medium.com/@zilliz_learn/getting-started-with-audio-data-processing-techniques-and-key-challenges-420dc5233163 (Last accessed: 16.05.2025).

21. Goudarzi A., Moya-Galé G. Automatic speech recognition in noise for parkinson's disease: a pilot study. Frontiers in artificial intelligence. 2021. Vol. 4. DOI: 10.3389/frai.2021.809321.

22. Hirsch H. G., Hellwig K., Dobler S. Speech recognition at multiple sampling rates. 7th european conference on speech communication and technology (eurospeech 2001). ISCA, 2001. DOI: 10.21437/Eurospeech.2001-434.

23. Poor transcription accuracy on voice calls is a catalyst for trouble. Observe.AI | Contact Center AI Software. URL: https://www.observe.ai/blog/bad-voice-call-transcription-accuracy-is-a-catalyst-for-trouble (Last accessed: 16.05.2025).

24. A comparative study for some characteristics between two species of okra abelmoschus esculentus L. under al-najaf province conditions. International journal for sciences and technology. 2017. Т. 12, № 1. С. 72–75. DOI: 10.12816/0040724.

25. High-Frequency dependence of acoustic properties of three typical sediments in the south china sea / J. Wang et al. Journal of marine science and engineering. 2022. Vol. 10, no. 9. P. 1295. DOI: 10.3390/jmse10091295.

26. Chung Y., Hansen J. H. Compensation of SNR and noise type mismatch using an environmental sniffing based speech recognition solution. EURASIP journal on audio, speech, and music processing. 2013. Vol. 2013, no. 1. DOI: 10.1186/1687-4722-2013-12.

27. Bu R., Weber J. H. Maximum likelihood decoding for multi-level cell memories with scaling and offset mismatch. ICC 2019 - 2019 IEEE international conference on communications (ICC), Shanghai, China, 20–24 May 2019. 2019. DOI: 10.1109/ICC.2019.8761978.

28. Otter.ai Help. URL: https://help.otter.ai/hc/en-us (Last accessed: 30.05.2025).

29. Descript Help Center. URL: https://help.descript.com/hc/en-us (Last accessed: 30.05.2025).

30. Sonix. API documentation. URL: https://sonix.ai/docs/api (Last accessed: 30.05.2025).

31. Trint API keys and authentication. URL: https://dev.trint.com/docs (Last accessed: 30.05.2025).

ДОДАТОК А

Код програмної реалізації

Лістинг коду головного серверного файлу app.py

```
import base64
import logging
from flask import Flask, redirect, url_for, request, jsonify
from flask_sqlalchemy import SQLAlchemy
from flask_migrate import Migrate
import json
import boto3
from botocore.exceptions import ClientError
import time
from flask_cors import CORS
from transcribe import Transcribe
from models import db, User, Audio, Transcription
import threading
import uuid as uuid_lib
from llmworker import LLMWorker

#Encryption libs
from Crypto.Hash import MD5
from Crypto.Util.Padding import unpad
from Crypto.Cipher import AES

import os
import uuid
import whisper
from werkzeug.utils import secure_filename
from dotenv import load_dotenv
from pydub import AudioSegment

#request
from urllib import request as urlrequest

#auth
from auth_routes import auth_bp, token_required
from transcription_routes import transcription_bp

app = Flask(__name__)

CORS(app,
     origins=["http://localhost:3000"],
     methods=["GET", "POST", "PUT", "DELETE", "OPTIONS"],
     allow_headers=["Content-Type", "Authorization"],
     supports_credentials=True)

app.register_blueprint(auth_bp)
app.register_blueprint(transcription_bp)

load_dotenv()

#PostgreSQL
database_url = os.getenv('DATABASE_URL')
if not database_url:
    raise ValueError("DATABASE_URL не знайдено в .env файлі")
if database_url.startswith('postgres://'):
    database_url = database_url.replace('postgres://', 'postgresql://', 1)
```

```
app.config['SQLALCHEMY_DATABASE_URI'] = database_url
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

app.config['SQLALCHEMY_ENGINE_OPTIONS'] = {
    'pool_pre_ping': True,
    'pool_recycle': 300,
}

db.init_app(app)
migrate = Migrate(app, db)

UPLOAD_FOLDER = 'uploads'
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

transcription_tasks = {}


def get_audio_duration(file_path):
    """Отримує тривалість аудіофайлу"""
    try:
        audio = AudioSegment.from_file(file_path)
        return len(audio) / 1000.0
    except Exception as e:
        app.logger.error(f"Error getting audio duration: {str(e)}")
        return None


def transcribe_process_thread(file_path, tr_uuid, model_type='base'):
    """Функція для асинхронної транскрипції в окремому потоці"""
    try:
        print(f"Starting transcription for {tr_uuid}")

        transcription_tasks[str(tr_uuid)] = {
            'status': 'processing',
            'progress': 0,
            'message': 'Starting transcription...'
        }

        with app.app_context():
            transcription = Transcription.query.filter_by(uuid=tr_uuid).first()
            if not transcription:
                raise Exception(f"Transcription with UUID {tr_uuid} not found")

            transcription.status = "processing"
            db.session.commit()

            transcribe = Transcribe()

            transcription_tasks[str(tr_uuid)]['progress'] = 20
            transcription_tasks[str(tr_uuid)]['message'] = 'Normalizing audio...'

            pre_loaded_file = transcribe.get_audio_data(file_path)
            normalized_file = transcribe.audio_normalize(pre_loaded_file)

            transcription_tasks[str(tr_uuid)]['progress'] = 40
            transcription_tasks[str(tr_uuid)]['message'] = 'Transcribing audio...'

            try:
                result = transcribe.audio_to_text(model=model_type,
mediafile=normalized_file)
            except Exception as e:
                print(f"Model {model_type} failed, trying base model: {str(e)}")
```

```
                result = transcribe.audio_to_text(model='base',
mediafile=normalized_file)

            transcription_tasks[str(tr_uuid)]['progress'] = 70
            transcription_tasks[str(tr_uuid)]['message'] = 'Analyzing speakers...'

            try:
                diarization_result =
transcribe.diarization(audio_location=normalized_file)
                speakers_json, speakers_text =
transcribe.match_transcription_diarization(
                    diarization_result, result, normalized_file)
            except Exception as e:
                print(f"Diarization failed: {str(e)}")
                speakers_json = {}
                speakers_text = result['text']

            transcription_tasks[str(tr_uuid)]['progress'] = 90
            transcription_tasks[str(tr_uuid)]['message'] = 'Saving results...'

            transcription.text = result['text']
            transcription.speakers_text = speakers_text
            transcription.speakers_json = speakers_json  # PostgreSQL підтримує
JSON
            transcription.language = result.get('language', 'unknown')
            transcription.status = "completed"
            db.session.commit()

            for f in [file_path, pre_loaded_file, normalized_file]:
                try:
                    if os.path.exists(f):
                        os.remove(f)
                except Exception as e:
                    print(f"Error removing file {f}: {str(e)}")

            transcription_tasks[str(tr_uuid)] = {
                'status': 'completed',
                'progress': 100,
                'message': 'Transcription completed successfully',
                'result': {
                    'text': result['text'],
                    'speakers_text': speakers_text,
                    'speakers_json': speakers_json,
                    'language': result.get('language', 'unknown')
                }
            }

            print(f"Transcription completed for {tr_uuid}")

    except Exception as e:
        print(f"Transcription error for {tr_uuid}: {str(e)}")

        try:
            with app.app_context():
                transcription =
Transcription.query.filter_by(uuid=tr_uuid).first()
                if transcription:
                    transcription.status = "failed"
                    db.session.commit()
        except Exception as db_error:
            print(f"Database error: {str(db_error)}")

        transcription_tasks[str(tr_uuid)] = {
```

```
            'status': 'failed',
            'progress': 0,
            'message': f'Transcription failed: {str(e)}'
        }


def get_current_user_from_token():
    """Отримує поточного користувача з JWT токена"""
    try:
        token = request.headers.get('Authorization')

        if not token:
            return None

        if token.startswith('Bearer '):
            token = token[7:]

        from auth_routes import JWT_SECRET
        import jwt

        data = jwt.decode(token, JWT_SECRET, algorithms=['HS256'])
        current_user = User.query.filter_by(id=data['user_id']).first()

        return current_user

    except Exception as e:
        print(f"Error getting user from token: {str(e)}")
        return None


@app.route('/', methods=['GET'])
def index():
    return jsonify(
        status='Index Page',
        description='Transcription API v1.0',
        database='PostgreSQL'
    )


@app.route('/upload', methods=['POST'])
def upload_file():
    """Завантаження файлу та запуск асинхронної транскрипції"""
    current_user = get_current_user_from_token()

    if not current_user:
        return jsonify({'error': 'Authentication required'}), 401

    print(f"Upload request from user: {current_user.email}")

    if 'file' not in request.files:
        return jsonify({'error': 'No file part'}), 400

    file = request.files['file']
    if file.filename == '':
        return jsonify({'error': 'No selected file'}), 400

    try:
        tr_uuid = uuid_lib.uuid4()

        filename = secure_filename(file.filename)
        unique_filename = f"{tr_uuid}_{filename}"
        file_path = os.path.join(app.config['UPLOAD_FOLDER'], unique_filename)
```

```
        file.save(file_path)

        if not os.path.exists(file_path):
            return jsonify({'error': 'Failed to save file'}), 500
        print(f"File saved to: {file_path}")
        print(f"File size: {os.path.getsize(file_path)} bytes")
        file_size = os.path.getsize(file_path)
        duration = get_audio_duration(file_path)
        file_format = filename.split('.')[-1] if '.' in filename else None

        audio = Audio(
            uuid=tr_uuid,
            user_id=current_user.id,
            filename=filename,
            file_path=file_path,
            file_size=file_size,
            duration=duration,
            format=file_format
        )
        db.session.add(audio)
        db.session.flush()

        transcription = Transcription(
            uuid=tr_uuid,
            user_id=current_user.id,
            audio_id=audio.id,
            status="pending"
        )
        db.session.add(transcription)
        db.session.commit()

        transcription_tasks[str(tr_uuid)] = {
            'status': 'pending',
            'progress': 0,
            'message': 'Task queued for processing'
        }

        thread = threading.Thread(
            target=transcribe_process_thread,
            args=(file_path, tr_uuid),
            daemon=True
        )
        thread.start()
        return jsonify({
            'status': 'success',
            'message': 'File uploaded successfully. Transcription started.',
            'uuid': str(tr_uuid),
            'transcription_status': 'pending'
        })
    except Exception as e:
        app.logger.error(f"Upload error: {str(e)}")
        try:
            if 'file_path' in locals() and os.path.exists(file_path):
                os.remove(file_path)
        except:
            pass
        return jsonify({'error': str(e)}), 500

@app.route('/status/<transcription_uuid>', methods=['GET'])
def get_transcription_status(transcription_uuid):
    """Отримання статусу транскрипції"""
    current_user = get_current_user_from_token()
```

```
    if not current_user:
        return jsonify({'error': 'Authentication required'}), 401
    try:
        transcription = Transcription.query.filter_by(
            uuid=transcription_uuid,
            user_id=current_user.id
        ).first()

        if not transcription:
            return jsonify({'error': 'Transcription not found'}), 404
        task_status = transcription_tasks.get(transcription_uuid, {
            'status': transcription.status,
            'progress': 100 if transcription.status == 'completed' else 0,
            'message': f'Status: {transcription.status}'
        })
        response_data = {
            'status': task_status['status'],
            'progress': task_status.get('progress', 0),
            'message': task_status.get('message', ''),
            'uuid': str(transcription.uuid),
            'created_at': transcription.created_at.isoformat() if
transcription.created_at else None
        }
        if task_status['status'] == 'completed':
            if 'result' in task_status:
                response_data.update(task_status['result'])
            else:
                response_data.update({
                    'text': transcription.text,
                    'speakers_text': transcription.speakers_text,
                    'speakers': transcription.speakers_json,
                    'language': transcription.language
                })
        return jsonify(response_data)

    except Exception as e:
        app.logger.error(f"Error getting transcription status: {str(e)}")
        return jsonify({'error': str(e)}), 500

@app.route('/audio/<audio_uuid>', methods=['GET'])
def get_audio(audio_uuid):
    """Отримання інформації про аудіофайл"""
    current_user = get_current_user_from_token()

    if not current_user:
        return jsonify({'error': 'Authentication required'}), 401

    try:
        audio = Audio.query.filter_by(
            uuid=audio_uuid,
            user_id=current_user.id
        ).first()

        if not audio:
            return jsonify({'error': 'Audio not found'}), 404
        transcriptions = [t.to_dict() for t in audio.transcriptions]
        return jsonify({
            'status': 'success',
            'audio': audio.to_dict(),
            'transcriptions': transcriptions
        })
    except Exception as e:
        app.logger.error(f"Error getting audio: {str(e)}")
```

```
        return jsonify({'error': str(e)}), 500

@app.route('/transcriptions', methods=['GET'])
def get_all_transcriptions():
    """Отримання всіх транскрипцій користувача"""
    current_user = get_current_user_from_token()

    if not current_user:
        return jsonify({'error': 'Authentication required'}), 401
    try:
        page = request.args.get('page', 1, type=int)
        per_page = request.args.get('per_page', 10, type=int)
        status = request.args.get('status')
        query = Transcription.query.filter_by(user_id=current_user.id)
        if status:
            query = query.filter_by(status=status)
        transcriptions = query.order_by(Transcription.created_at.desc()).paginate(
            page=page, per_page=per_page, error_out=False
        )
        return jsonify({
            'status': 'success',
            'transcriptions': [t.to_dict() for t in transcriptions.items],
            'pagination': {
                'page': page,
                'per_page': per_page,
                'total': transcriptions.total,
                'pages': transcriptions.pages
            }
        })
    except Exception as e:
        app.logger.error(f"Error getting transcriptions: {str(e)}")
        return jsonify({'error': str(e)}), 500

@app.route('/transcriptions/<transcription_uuid>', methods=['GET'])
def get_transcription(transcription_uuid):
    """Отримання конкретної транскрипції"""
    current_user = get_current_user_from_token()

    if not current_user:
        return jsonify({'error': 'Authentication required'}), 401

    try:
        transcription = Transcription.query.filter_by(
            uuid=transcription_uuid,
            user_id=current_user.id
        ).first()

        if not transcription:
            return jsonify({'error': 'Transcription not found'}), 404

        return jsonify({
            'status': 'success',
            'transcription': transcription.to_dict()
        })
    except Exception as e:
        app.logger.error(f"Error getting transcription: {str(e)}")
        return jsonify({'error': str(e)}), 500

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5070)
```

Лістинг коду файлу з діагностикою та транскрибуванням аудіо transcribe.py

```
import os
import time
import whisper
import tempfile
import numpy as np
from pydub import AudioSegment
import json
from pyannote.audio import Pipeline
from pyannote.core import Annotation
from typing import Dict, Any, Tuple, List
import logging
from collections import defaultdict
import pprint
import torch


logger = logging.getLogger(__name__)


class Transcribe:
    def __init__(self):
        self.device = "cuda" if torch.cuda.is_available() else "cpu"
        logger.info(f"Using device: {self.device}")

        self.model = whisper.load_model("base")

        self.diarization_pipeline = None
        try:
            self.diarization_pipeline = Pipeline.from_pretrained(
                "pyannote/speaker-diarization",
                use_auth_token=os.getenv('PYANNOTE_TOKEN')
            )

            if hasattr(self.diarization_pipeline, 'instantiate_params'):
                self.diarization_pipeline.instantiate_params({
                    "segmentation": {
                        "threshold": 0.25,
                        "min_duration": 0.1
                    },
                    "embedding": {
                        "window": 0.75,
                        "step": 0.2
                    },
                    "clustering": {
                        "method": "affinity_propagation",
                        "min_cluster_size": 2,
                        "threshold": 0.65
                    }
                })
        except Exception as e:
            print(f"Failed to initialize diarization pipeline: {str(e)}")


    def get_audio_data(self, audio_location: str) -> str:
        """Download or get local audio file"""
        try:
            if audio_location.startswith(('http://', 'https://')):
                import requests
                response = requests.get(audio_location)
                response.raise_for_status()
```

```
                temp_file = tempfile.NamedTemporaryFile(delete=False,
suffix='.wav')
                temp_file.write(response.content)
                temp_file.close()
                return temp_file.name
            return audio_location
        except Exception as e:
            logger.error(f"Error getting audio data: {str(e)}")
            raise


    def audio_normalize(self, audio_file: str) -> str:
        """Normalize audio to 16kHz mono WAV format"""
        try:
            audio = AudioSegment.from_file(audio_file)
            audio = audio.set_frame_rate(16000).set_channels(1)

            audio = audio.normalize()

            if len(audio) > 5000:
                audio = audio.low_pass_filter(4000)
                audio = audio.high_pass_filter(80)

            audio = audio.compress_dynamic_range(threshold=-20.0, ratio=2.0)

            normalized_file = tempfile.NamedTemporaryFile(delete=False,
suffix='.wav')
            audio.export(normalized_file.name,
                         format="wav",
                         codec="pcm_s16le",
                         parameters=["-ar", "16000"])

            return normalized_file.name
        except Exception as e:
            logger.error(f"Error normalizing audio: {str(e)}")
            raise


    def audio_to_text(self, mediafile: str, model: str = 'base') -> Dict[str,
Any]:
        """Transcribe audio to text using Whisper"""
        try:
            result = self.model.transcribe(
                mediafile,
                word_timestamps=True,
                fp16=(self.device == "cuda"),
                temperature=1,
            )

            audio = AudioSegment.from_file(mediafile)
            audio_duration = len(audio) / 1000.0

            return {
                'text': result['text'],
                'segments': result['segments'],
                'language': result['language']
            }
        except Exception as e:
            logger.error(f"Transcription error: {str(e)}")
            raise


    def audio_to_vtt(self, transcription: Dict[str, Any]) -> str:
```

```
        """Convert transcription to WebVTT format"""
        vtt = "WEBVTT\n\n"
        for segment in transcription['segments']:
            start = self.format_time(segment['start'])
            end = self.format_time(segment['end'])
            text = segment['text']
            vtt += f"{start} --> {end}\n{text}\n\n"
        return vtt


    def format_time(self, seconds: float) -> str:
        """Format seconds to VTT time format"""
        hours = int(seconds // 3600)
        minutes = int((seconds % 3600) // 60)
        seconds = seconds % 60
        return f"{hours:02d}:{minutes:02d}:{seconds:06.3f}"


    def check_audio_quality(self, audio_file: str) -> dict:
        """Check audio for potential issues affecting diarization"""
        try:
            if not isinstance(audio_file, (str, os.PathLike)):
                print(f"Invalid audio_file type: {type(audio_file)}")
                return {"error": f"Expected file path, got {type(audio_file)}"}

            audio_file = str(audio_file)

            if not os.path.exists(audio_file):
                print(f"Audio file does not exist: {audio_file}")
                return {"error": "File not found"}

            file_size = os.path.getsize(audio_file)
            if file_size == 0:
                print("Audio file is empty")
                return {"error": "Empty file"}

            print(f"Processing audio file: {audio_file} (size: {file_size}
bytes)")

            try:
                audio = AudioSegment.from_file(audio_file)
                print("Successfully loaded audio with AudioSegment")
            except Exception as e:
                print(f"AudioSegment error: {str(e)}")
                return {"error": f"Failed to load audio: {str(e)}"}

            duration = len(audio) / 1000.0
            loudness = audio.dBFS

            try:
                samples = np.array(audio.get_array_of_samples())
                if len(samples) > 0:
                    dynamic_range = float(np.max(samples) - np.min(samples))
                else:
                    dynamic_range = 0
            except Exception as e:
                print(f"Sample analysis error: {str(e)}")
                dynamic_range = 0

            sample_rate = audio.frame_rate
            channels = audio.channels

            report = {
```

```
                "duration": duration,
                "loudness": loudness,
                "dynamic_range": dynamic_range,
                "sample_rate": sample_rate,
                "channels": channels,
                "potential_issues": []
            }

            if duration < 10:
                report["potential_issues"].append("Audio too short for reliable
diarization")

            if loudness > -20:
                report["potential_issues"].append("Audio might be too loud")

            if loudness < -35:
                report["potential_issues"].append("Audio might be too quiet")

            if dynamic_range < 1000:
                report["potential_issues"].append("Low dynamic range might affect
speaker separation")

            if sample_rate < 16000:
                report["potential_issues"].append("Sample rate too low for optimal
diarization")

            print("\n--- Audio Quality Report ---")
            print(f"Duration: {duration:.2f} seconds")
            print(f"Loudness: {loudness:.2f} dB")
            print(f"Sample rate: {sample_rate} Hz")
            print(f"Channels: {channels}")
            if report["potential_issues"]:
                print("Potential issues:")
                for issue in report["potential_issues"]:
                    print(f"- {issue}")

            return report
        except Exception as e:
            print(f"Audio quality check error: {str(e)}")
            return {"error": str(e)}


    def post_process_diarization(self, diarization: Annotation) -> Annotation:
        """Покращує послідовність ідентифікації спікерів"""
        try:
            from pyannote.core import Segment

            segments = []
            for segment, track, speaker in
diarization.itertracks(yield_label=True):
                try:
                    if hasattr(segment, 'start') and hasattr(segment, 'end'):
                        segments.append({
                            'start': float(segment.start),
                            'end': float(segment.end),
                            'speaker': speaker
                        })
                    else:
                        print(f"Invalid segment type in post_process:
{type(segment)}")
                except Exception as e:
                    print(f"Error processing segment in post_process: {str(e)}")
                    continue
```

```
            if not segments:
                print("No valid segments found in post_process_diarization")
                return diarization

            segments.sort(key=lambda x: x['start'])

            speaker_order = {}
            current_index = 0

            for segment in segments:
                if segment['speaker'] not in speaker_order:
                    speaker_order[segment['speaker']] =
f"SPEAKER_{current_index:02d}"
                    current_index += 1

            new_diarization = Annotation()
            for segment in segments:
                new_speaker = speaker_order[segment['speaker']]
                segment_obj = Segment(segment['start'], segment['end'])
                new_diarization[segment_obj] = new_speaker

            print(f"Post-processing: remapped {len(speaker_order)} speakers")
            for old_speaker, new_speaker in speaker_order.items():
                print(f"  {old_speaker} -> {new_speaker}")

            return new_diarization

        except Exception as e:
            print(f"Error in post_process_diarization: {str(e)}")
            return diarization


    def _calculate_speaker_similarity(self, features1, features2):
        """Обчислює схожість між двома наборами характеристик спікерів"""
        if not features1 or not features2:
            return 0.0

        avg1 = {k: np.mean([f[k] for f in features1]) for k in features1[0] if k
!= 'duration'}
        avg2 = {k: np.mean([f[k] for f in features2]) for k in features2[0] if k
!= 'duration'}

        squared_diff = sum((avg1[k] - avg2[k])**2 for k in avg1)
        distance = np.sqrt(squared_diff)

        max_possible_distance = 65535
        similarity = 1.0 - min(distance / max_possible_distance, 1.0)

        return similarity


    def alternative_diarization(self, audio_location: str) -> Annotation:
        """Alternative approach using direct model access"""
        try:
            from pyannote.audio.pipelines import SpeakerDiarization
            from pyannote.audio import Model

            segmentation = Model.from_pretrained("pyannote/segmentation-3.0",

use_auth_token='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')
            embedding = Model.from_pretrained("pyannote/embedding-3.0",
```

```
use_auth_token='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')

            pipeline = SpeakerDiarization(segmentation=segmentation,
embedding=embedding)

            pipeline.instantiate({
                "segmentation": {"threshold": 0.2},
                "clustering": {"method": "spectral", "min_clusters": 2,
"max_clusters": 5}
            })

            diarization = pipeline(audio_location)

            return diarization
        except Exception as e:
            logger.error(f"Alternative diarization error: {str(e)}")
            raise


    def diagnose_diarization(self, audio_file: str) -> None:
        """Run diagnostics on diarization to identify issues"""
        try:
            print("\n=== DIARIZATION DIAGNOSTICS ===\n")
            quality_report = self.check_audio_quality(audio_file)

            normalized_file = self.audio_normalize(audio_file)

            try:
                diarization = self.diarization(normalized_file)
                speakers = set()
                for _, _, speaker in diarization.itertracks(yield_label=True):
                    speakers.add(speaker)
                print(f"   Detected {len(speakers)} speakers: {',
'.join(speakers)}")
            except Exception as e:
                print(f"   Standard diarization failed: {str(e)}")

            try:
                diarization = self.diarization_pipeline(
                    normalized_file,
                    num_speakers=2,
                    clustering="AgglomerativeClustering"
                )
                speakers = set()
                for _, _, speaker in diarization.itertracks(yield_label=True):
                    speakers.add(speaker)
                print(f"   Detected {len(speakers)} speakers: {',
'.join(speakers)}")
            except Exception as e:
                print(f"   Forced parameters failed: {str(e)}")

            try:
                result = self.alternative_diarization(normalized_file)
                speakers = set()
                for _, _, speaker in result.itertracks(yield_label=True):
                    speakers.add(speaker)
                print(f"   Detected {len(speakers)} speakers: {',
'.join(speakers)}")
            except Exception as e:
                print(f"   Alternative method failed: {str(e)}")

            print("\n=== DIAGNOSTICS COMPLETE ===\n")
```

```
        except Exception as e:
            print(f"Diagnostics failed: {str(e)}")


    def diarization(self, audio_location: str) -> Annotation:
        """Perform speaker diarization on audio file"""
        if not self.diarization_pipeline:
            raise Exception("Diarization pipeline not initialized")

        try:
            print("Starting diarization...")

            if not isinstance(audio_location, (str, os.PathLike)):
                raise Exception(f"Expected file path, got {type(audio_location)}")

            audio_location = str(audio_location)

            if not os.path.exists(audio_location):
                raise Exception(f"Audio file not found: {audio_location}")

            # Basic diarization
            try:
                diarization = self.diarization_pipeline(audio_location)
                print("Basic diarization completed")
            except Exception as e:
                print(f"Basic diarization failed: {str(e)}")
                # Min params try
                try:
                    diarization = self.diarization_pipeline(
                        audio_location,
                        min_speakers=1,
                        max_speakers=6
                    )
                    print("Diarization with min/max speakers completed")
                except Exception as e2:
                    print(f"Diarization with parameters also failed: {str(e2)}")
                    raise Exception(f"All diarization attempts failed: {str(e2)}")

            # Check valid
            valid_segments = []
            speakers = set()

            try:
                for segment, track, speaker in
diarization.itertracks(yield_label=True):
                    print(f"Segment type: {type(segment)}, Speaker: {speaker}")

                    if hasattr(segment, 'start') and hasattr(segment, 'end'):
                        valid_segments.append((segment, track, speaker))
                        speakers.add(speaker)
                        print(f"Valid segment: {segment.start:.2f}-
{segment.end:.2f}, Speaker: {speaker}")
                    elif isinstance(segment, tuple) and len(segment) >= 2:
                        start_time, end_time = segment[0], segment[1]
                        print(f"Tuple segment: {start_time:.2f}-{end_time:.2f},
Speaker: {speaker}")
                        class PseudoSegment:
                            def __init__(self, start, end):
                                self.start = start
                                self.end = end
                        pseudo_segment = PseudoSegment(start_time, end_time)
                        valid_segments.append((pseudo_segment, track, speaker))
```

```
                        speakers.add(speaker)
                    else:
                        print(f"Unknown segment format: {type(segment)}, value:
{segment}")
            except Exception as e:
                print(f"Error iterating through diarization: {str(e)}")
                raise Exception(f"Failed to process diarization results:
{str(e)}")

            if not valid_segments:
                raise Exception("No valid segments found in diarization result")

            print(f"Found {len(speakers)} speakers: {', '.join(speakers)}")

            #If only one spealer
            if len(speakers) <= 1:
                print("Only one speaker detected, trying with forced
parameters...")
                try:
                    diarization = self.diarization_pipeline(
                        audio_location,
                        num_speakers=2
                    )
                    print("Forced diarization with num_speakers=2 completed")

                    valid_segments = []
                    speakers = set()

                    for segment, track, speaker in
diarization.itertracks(yield_label=True):
                        if hasattr(segment, 'start') and hasattr(segment, 'end'):
                            valid_segments.append((segment, track, speaker))
                            speakers.add(speaker)

                    if len(speakers) <= 1:
                        print("Still only one speaker detected after forced
parameters")

                except Exception as e:
                    print(f"Forced diarization failed: {str(e)}")

            try:
                diarization = self.post_process_diarization(diarization)
                print("Post-processing completed")
            except Exception as e:
                print(f"Post-processing failed: {str(e)}")

            print("\n--- Diarization Segments ---")
            segment_count = 0
            for turn, _, speaker in diarization.itertracks(yield_label=True):
                try:
                    if hasattr(turn, 'start') and hasattr(turn, 'end'):
                        print(f"Speaker: {speaker}, Start:
{float(turn.start):.2f}, End: {float(turn.end):.2f}")
                        segment_count += 1
                    elif isinstance(turn, tuple) and len(turn) >= 2:
                        print(f"Speaker: {speaker}, Start: {float(turn[0]):.2f},
End: {float(turn[1]):.2f}")
                        segment_count += 1
                    else:
                        print(f"Unknown turn format: {type(turn)}")
                except Exception as e:
                    print(f"Error displaying segment: {str(e)}")
```

```
            print(f"Total segments: {segment_count}")
            return diarization

        except Exception as e:
            logger.error(f"Diarization error: {str(e)}")
            raise


    def match_transcription_diarization(self,
                                        diarization: Annotation,
                                        transcription: Dict[str, Any],
                                        audio_file: str
                                        ) -> Tuple[Dict[str, List[Dict[str,
Any]]], str]:
        """Match transcription segments with speaker diarization"""
        try:
            if not diarization:
                print("No diarization available")
                return {}, "No diarization available"

            speaker_turns = []
            try:
                for turn, _, speaker in diarization.itertracks(yield_label=True):
                    try:
                        if hasattr(turn, 'start') and hasattr(turn, 'end'):
                            speaker_turns.append({
                                'start': float(turn.start),
                                'end': float(turn.end),
                                'speaker': speaker
                            })
                        else:
                            print(f"Invalid turn format: {type(turn)}")
                    except Exception as e:
                        print(f"Error processing turn: {str(e)}")
                        continue
            except Exception as e:
                print(f"Error iterating through diarization: {str(e)}")
                return {}, f"Error processing diarization: {str(e)}"
            if not speaker_turns:
                print("No valid speaker turns found")
                return {}, "No valid speaker turns found"

            speaker_turns.sort(key=lambda x: x['start'])

            unique_speakers_by_time = []
            seen_speakers = set()

            for turn in speaker_turns:
                if turn['speaker'] not in seen_speakers:
                    unique_speakers_by_time.append(turn['speaker'])
                    seen_speakers.add(turn['speaker'])

            speaker_map = {spkr: f"SPEAKER_{i:02d}" for i, spkr in
enumerate(unique_speakers_by_time)}
            print(f"Found {len(unique_speakers_by_time)} unique speakers in order
of appearance:")
            for old_speaker, new_speaker in speaker_map.items():
                print(f"  {old_speaker} -> {new_speaker}")
            speakers = defaultdict(list)
            all_segments = []
            for segment in sorted(transcription['segments'], key=lambda x:
x['start']):
```

```
                seg_start = segment['start']
                seg_end = segment['end']
                text = segment['text']
                overlaps = []
                for turn in speaker_turns:
                    overlap_start = max(seg_start, turn['start'])
                    overlap_end = min(seg_end, turn['end'])
                    if overlap_start < overlap_end:
                        overlaps.append({
                            'speaker': turn['speaker'],
                            'duration': overlap_end - overlap_start
                        })
                if overlaps:
                    speaker_durations = defaultdict(float)
                    for ov in overlaps:
                        speaker_durations[ov['speaker']] += ov['duration']
                    dominant_speaker = max(speaker_durations.items(), key=lambda
x: x[1])[0]
                    norm_speaker = speaker_map[dominant_speaker]

                    all_segments.append({
                        'start': seg_start,
                        'end': seg_end,
                        'speaker': norm_speaker,
                        'text': text
                    })
                else:
                    closest_speaker = min(speaker_turns,
                                          key=lambda x: min(abs(x['start'] -
seg_start),
                                                            abs(x['end'] - seg_end)))
                    norm_speaker = speaker_map[closest_speaker['speaker']]

                    all_segments.append({
                        'start': seg_start,
                        'end': seg_end,
                        'speaker': norm_speaker,
                        'text': text
                    })

            text_output = ""
            current_speaker = None

            for seg in sorted(all_segments, key=lambda x: x['start']):
                if seg['speaker'] != current_speaker:
                    text_output += f"\n=== {seg['speaker']} ===\n"
                    current_speaker = seg['speaker']
                text_output += f"[{seg['start']:.2f}-{seg['end']:.2f}]
{seg['text']}\n"

            for seg in all_segments:
                speakers[seg['speaker']].append({
                    'start': seg['start'],
                    'end': seg['end'],
                    'text': seg['text']
                })

            return dict(speakers), text_output.strip()

        except Exception as e:
            print(f"Error in match_transcription_diarization: {str(e)}")
            return {}, f"Error matching transcription with diarization: {str(e)}"
```

Лістинг коду файлу з обробкою транскрипції LLM llmworker.py

```
import os
import time
import logging
import boto3
from botocore.exceptions import ClientError


logger = logging.getLogger(__name__)
logging.basicConfig(level=logging.INFO)


class LLMWorker():
      def __init__(self, aws_access_key_id = os.getenv('AWS_ACCESS_KEY_ID'),
                   aws_secret_access_key=os.getenv('AWS_SECRET_ACCESS_KEY'),
                   region_name= os.getenv('AWS_REGION')):
            self.client = boto3.client(
                  "bedrock-runtime",
                  aws_access_key_id = aws_access_key_id,
                  aws_secret_access_key = aws_secret_access_key,
                  region_name = region_name
                  )
            self.max_retries = 3
            self.initial_retry_delay = 2

      def _invoke_model_with_retry(self, conversation, model_id,
inference_config):
            retry_delay = self.initial_retry_delay

            for attempt in range(self.max_retries):
                  try:
                        response = self.client.converse(
                              modelId=model_id,
                              messages=conversation,
                              inferenceConfig=inference_config,
                              additionalModelRequestFields={}
                        )
                        return response["output"]["message"]["content"][0]["text"]

                  except ClientError as e:
                        if e.response['Error']['Code'] == 'ThrottlingException':
                              logger.warning(
                                    f"Throttling detected (attempt {attempt +
1}/{self.max_retries}), "
                                    f"retrying in {retry_delay} seconds..."
                              )
                              time.sleep(retry_delay)
                              retry_delay *= 2  # Exponential backoff
                              continue
                        logger.error(f"AWS ClientError: {str(e)}")
                        raise
                  except Exception as e:
                        logger.error(f"Unexpected error: {str(e)}")
                        raise

            raise Exception(f"Max retries ({self.max_retries}) reached for model
{model_id}")


      def _truncate_text(self, text, max_length=8000):
            if len(text) > max_length:
```

```
                logger.warning(f"Truncating text from {len(text)} to
{max_length} characters")
                return text[:max_length] + "\n...[текст обрізано через обмеження
довжини]"
            return text


      def create_summary_prompt(self, conversation_text, model_id =
os.getenv('AWS_MODEL_NAME')):
            conversation_text = self._truncate_text(conversation_text)

            prompt_text = (
            "Please create a detailed summary of this conversation. "
            "Structure the summary as follows:\n\n"
            "1. First, describe each speaker briefly (e.g., SPEAKER_00: [brief
description based on their speech patterns and topics])\n"
            "2. Then provide a comprehensive summary of the conversation
covering:\n"
            "   - Main topics discussed\n"
            "   - Key decisions or agreements made\n"
            "   - Important information shared\n"
            "   - Action items or next steps mentioned\n"
            "   - Overall tone and context of the conversation\n\n"
            "Make each paragraph at least 3-4 sentences and focus on the most
important content. "
            "Write in a clear, professional style suitable for general business or
personal conversations."
        )
            conversation = [
            {
                "role": "user",
                "content": [{"text": prompt_text}, {"text": conversation_text}],
            }
            ]

            inference_config = {
            "maxTokens": 2048,
            "stopSequences": ["User:"],
            "temperature": 0.7,
            "topP": 0.9
        }
            try:
                return self._invoke_model_with_retry(conversation, model_id,
inference_config)
            except Exception as e:
                logger.error(f"Failed to create summary: {str(e)}")
                return None


      def analyze_conversation_text(self, conversation_text, model_id =
os.getenv('AWS_MODEL_NAME')):
            conversation_text = self._truncate_text(conversation_text)

            prompt_text = (
                "Analyze this conversation and extract key information in the
following structured format:\n\n"
                "CONVERSATION DETAILS\n"
                "Date/Time: [Extract if mentioned]\n"
                "Duration: [Estimate based on content]\n"
                "Type: [Meeting, Phone call, Casual conversation, etc.]\n"
                "Language: [Primary language used]\n\n"
                "PARTICIPANTS\n"
                "Speaker 1: [Name if mentioned, otherwise description]\n"
```

```
                  "Speaker 2: [Name if mentioned, otherwise description]\n"
                  "[Additional speakers if present]\n\n"
                  "MAIN TOPICS\n"
                  "Primary Topic: [Main subject discussed]\n"
                  "Secondary Topics: [Other important subjects]\n"
                  "Keywords: [Important terms, names, places mentioned]\n\n"
                  "KEY INFORMATION\n"
                  "Decisions Made: [Any decisions or agreements]\n"
                  "Action Items: [Tasks or follow-ups mentioned]\n"
                  "Important Dates: [Dates, deadlines, appointments]\n"
                  "Contact Information: [Phone numbers, emails, addresses if
mentioned]\n"
                  "References: [Documents, websites, other resources
mentioned]\n\n"
                  "CONVERSATION TONE\n"
                  "Overall Mood: [Professional, casual, friendly, tense, etc.]\n"
                  "Relationship: [Business partners, friends, family, etc.]\n"
                  "Purpose: [Information sharing, planning, problem-solving,
etc.]"
            )

            conversation = [
                  {
                        "role": "user",
                        "content": [{"text": prompt_text}, {"text":
conversation_text}],
                  }
            ]

            inference_config = {
                  "maxTokens": 2048,
                  "stopSequences": ["User:"],
                  "temperature": 0.5,
                  "topP": 0.8
            }

            try:
                  return self._invoke_model_with_retry(conversation, model_id,
inference_config)
            except Exception as e:
                  logger.error(f"Failed to analyze medical text: {str(e)}")
                  return None


      def extract_action_items(self, conversation_text, model_id =
os.getenv('AWS_MODEL_NAME')):
            conversation_text = self._truncate_text(conversation_text)

            prompt_text = (
                  "Extract all action items, tasks, and follow-ups from this
conversation. "
                  "Format the response as a clear list:\n\n"
                  "ACTION ITEMS:\n"
                  "1. [Task description] - Assigned to: [Person] - Deadline: [Date
if mentioned]\n"
                  "2. [Task description] - Assigned to: [Person] - Deadline: [Date
if mentioned]\n\n"
                  "FOLLOW-UP MEETINGS:\n"
                  "- [Meeting description] - Date: [Date if mentioned] -
Participants: [Who should attend]\n\n"
                  "DEADLINES AND IMPORTANT DATES:\n"
                  "- [Date] - [What needs to be done]\n\n"
                  "CONTACT INFORMATION TO FOLLOW UP:\n"
```

```
                  "- [Name] - [Contact details if mentioned] - [Reason for
contact]\n\n"
                  "If no action items are found in any category, write 'None
identified' for that section."
            )

            conversation = [
                  {
                        "role": "user",
                        "content": [{"text": prompt_text}, {"text":
conversation_text}],
                  }
            ]

            inference_config = {
                  "maxTokens": 1024,
                  "stopSequences": ["User:"],
                  "temperature": 0.3,
                  "topP": 0.7
            }

            try:
                  return self._invoke_model_with_retry(conversation, model_id,
inference_config)
            except Exception as e:
                  logger.error(f"Failed to extract action items: {str(e)}")
                  return None


      def generate_meeting_minutes(self, conversation_text, model_id =
os.getenv('AWS_MODEL_NAME')):
            conversation_text = self._truncate_text(conversation_text)

            prompt_text = (
                  "Create professional meeting minutes from this conversation in
the following format:\n\n"
                  "MEETING MINUTES\n"
                  "Date: [Extract or estimate]\n"
                  "Participants: [List all speakers]\n"
                  "Duration: [Estimate based on content]\n\n"
                  "AGENDA ITEMS DISCUSSED:\n"
                  "1. [Topic 1]\n"
                  "   - Discussion points\n"
                  "   - Decisions made\n"
                  "   - Action items\n\n"
                  "2. [Topic 2]\n"
                  "   - Discussion points\n"
                  "   - Decisions made\n"
                  "   - Action items\n\n"
                  "DECISIONS MADE:\n"
                  "- [Decision 1]\n"
                  "- [Decision 2]\n\n"
                  "ACTION ITEMS:\n"
                  "- [Action item] - Responsible: [Person] - Due: [Date]\n\n"
                  "NEXT STEPS:\n"
                  "- [Next step 1]\n"
                  "- [Next step 2]\n\n"
                  "Write in a professional, clear style suitable for business
documentation."
            )

            conversation = [
                  {
```

```
                        "role": "user",
                        "content": [{"text": prompt_text}, {"text":
conversation_text}],
                  }
            ]

            inference_config = {
                  "maxTokens": 2048,
                  "stopSequences": ["User:"],
                  "temperature": 0.4,
                  "topP": 0.8
            }

            try:
                  return self._invoke_model_with_retry(conversation, model_id,
inference_config)
            except Exception as e:
                  logger.error(f"Failed to generate meeting minutes: {str(e)}")
                  return None


      def analyze_conversation_sentiment(self, conversation_text, model_id =
os.getenv('AWS_MODEL_NAME')):
            conversation_text = self._truncate_text(conversation_text)

            prompt_text = (
                  "Analyze the emotional tone and mood of this conversation:\n\n"
                  "MOOD ANALYSIS\n"
                  "Overall Tone: [Positive/Neutral/Negative]\n"
                  "Formality Level: [Formal/Informal/Mixed]\n"
                  "Emotional Intensity: [Low/Medium/High]\n\n"
                  "Each Speaker's MOOD:\n"
                  "SPEAKER_00: [Description of mood and emotional state]\n"
                  "SPEAKER_01: [Description of mood and emotional state]\n\n"
                  "KEY EMOTIONAL MOMENTS:\n"
                  "- [Moment 1]: [Description of emotional change or important
moment]\n"
                  "- [Moment 2]: [Description of emotional change or important
moment]\n\n"
                  "RECOMMENDATIONS:\n"
                  "- [Recommendation for further communication or action based on
analysis]"
            )

            conversation = [
                  {
                        "role": "user",
                        "content": [{"text": prompt_text}, {"text":
conversation_text}],
                  }
            ]

            inference_config = {
                  "maxTokens": 2048,
                  "stopSequences": ["User:"],
                  "temperature": 0.6,
                  "topP": 0.8
            }

            try:
                  return self._invoke_model_with_retry(conversation, model_id,
inference_config)
            except Exception as e:
```

```
            logger.error(f"Failed to analyze medical text: {str(e)}")
            return None


conversation_text = '''
Good morning, this is Dr. Emily Hart from the Respiratory Medicine Department at
St. Thomas' Hospital in London. I am calling to make a patient referral that
requires urgent attention.

The patient's name is Mr. James Edward Cartwright. He is a 64-year-old male, born
on 3rd February 1959, with NHS number 2345678901. This referral has been flagged
as urgent due to the severity of his symptoms and the need for immediate
specialist intervention.

Mr. Cartwright presented to our A&E department earlier today with worsening
shortness of breath, persistent cough with blood-streaked sputum, and pleuritic
chest pain over the past five days. A CT scan performed earlier revealed a large
left-sided pleural effusion, likely malignant, and mediastinal lymphadenopathy.
His oxygen saturation on room air was 88%, and he is currently on supplemental
oxygen. His medical history includes chronic obstructive pulmonary disease, a 40-
pack-year smoking history, and hypertension.
I have initiated a pleural tap, and 50ml of fluid has been sent for cytology and
biochemistry. However, he urgently requires a respiratory specialist for further
management, including consideration for thoracoscopy and biopsy.
For your reference, he is currently stable but remains at risk of clinical
deterioration. I strongly recommend his admission under the respiratory team for
immediate assessment and intervention.
You can contact me directly if there are any questions or additional details
required. My bleep number is 5678, and for urgent matters, I can be reached on my
personal mobile number, 07912 345678. You may also email me at
emily.hart@stthomas.nhs.uk for follow-up or documentation.
For the record, my full name is Dr. Emily Hart. I am a Consultant in Respiratory
Medicine at St. Thomas' Hospital. The referral has been initiated from our A&E
Department under the Respiratory Medicine Service. Please ensure this is actioned
promptly, as this is a high-priority case.

Thank you for your attention, and please do not hesitate to get in touch if you
require further clarification or documentation. I will follow up tomorrow to
ensure the referral has been processed."
'''

worker = LLMWorker()

summary = worker.create_summary_prompt(conversation_text)
print("Summary:")
print(summary)

conversation_structure = worker.analyze_conversation_text(conversation_text)
print("\nConversation Structure:")
print(conversation_structure)

action_items = worker.extract_action_items(conversation_text)
print("\nAction Items:")
print(action_items)

meeting_minutes = worker.generate_meeting_minutes(conversation_text)
print("\nMeeting Minutes:")
print(meeting_minutes)

conversation_sentiment = worker.analyze_conversation_sentiment(conversation_text)
print("\nConversation Sentiment:")
print(conversation_sentiment)
```

Лістинг коду файлу з маршрутами авторизації auth_routes.py

```
from flask import Blueprint, request, jsonify, redirect, url_for,
render_template_string
from werkzeug.security import generate_password_hash, check_password_hash
from models import db, User
from email_service import email_service
import jwt
import os
from datetime import datetime, timedelta
from functools import wraps


auth_bp = Blueprint('auth', __name__, url_prefix='/auth')

JWT_SECRET_RAM = os.getenv('JWT_SECRET')
JWT_SECRET = str(JWT_SECRET_RAM)

print(f"JWT_SECRET type: {type(JWT_SECRET)}, value: {JWT_SECRET[:10]}...")

def token_required(f):
    """Декоратор для захищених маршрутів"""
    @wraps(f)
    def decorated(*args, **kwargs):
        token = request.headers.get('Authorization')

        if not token:
            return jsonify({'message': 'Token is missing'}), 401

        try:
            if token.startswith('Bearer '):
                token = token[7:]

            data = jwt.decode(token, JWT_SECRET, algorithms=['HS256'])
            user_id = data['user_id']

            current_user = User.query.filter_by(id=user_id).first()

            if not current_user:
                return jsonify({'message': 'User not found'}), 401

            if not current_user.is_active:
                return jsonify({'error': 'User account is deactivated'}), 401

        except jwt.ExpiredSignatureError:
            return jsonify({'message': 'Token has expired'}), 401
        except jwt.InvalidTokenError:
            return jsonify({'message': 'Token is invalid'}), 401
        except Exception as e:
            print(f"Token validation error: {str(e)}")
            return jsonify({'message': 'Token validation failed'}), 401

        return f(current_user, *args, **kwargs)

    return decorated


def verified_required(f):
    """Декоратор для перевірки верифікації електронної пошти"""
    @wraps(f)
    def decorated(current_user, *args, **kwargs):
        if not current_user.is_verified:
```

```
            return jsonify({
                'error': 'Email verification required',
                'message': 'Please verify your email address before using this
feature'
            }), 403

        return f(current_user, *args, **kwargs)

    return decorated


def generate_token(user):
    """Генерує JWT токен для користувача"""
    try:
        expiration_time = datetime.datetime.utcnow() + datetime.timedelta(days=7)
        exp_timestamp = int(expiration_time.timestamp())

        payload = {
            'user_id': user.id,
            'email': user.email,
            'exp': exp_timestamp
        }

        print(f"Payload для JWT: {payload}")

        token = jwt.encode(payload, JWT_SECRET, algorithm='HS256')

        if isinstance(token, bytes):
            token = token.decode('utf-8')

        print(f"Token generated successfully: {token[:50]}...")
        return token
    except Exception as e:
        print(f"Token generation error: {str(e)}")
        import traceback
        traceback.print_exc()
        raise

@auth_bp.route('/register', methods=['POST'])
def register():
    try:
        data = request.get_json()

        if not data:
            return jsonify({
                'status': 'error',
                'message': 'No data provided'
            }), 400

        email = data.get('email', '').strip().lower()
        password = data.get('password', '')
        username = data.get('username', '').strip()

        print(f"Registration attempt for email: {email}")

        if not email or not password:
            return jsonify({
                'status': 'error',
                'message': 'Email and password are required'
            }), 400

        if len(password) < 6:
            return jsonify({
```

```
                'status': 'error',
                'message': 'Password must be at least 6 characters long'
            }), 400

        existing_user = User.query.filter_by(email=email).first()
        if existing_user:
            return jsonify({
                'status': 'error',
                'message': 'User with this email already exists'
            }), 409

        if username:
            existing_username = User.query.filter_by(username=username).first()
            if existing_username:
                return jsonify({
                    'status': 'error',
                    'message': 'Username already taken'
                }), 409

        user = User(
            email=email,
            username=username if username else None
        )
        user.set_password(password)

        verification_token = user.generate_verification_token()

        db.session.add(user)
        db.session.commit()

        email_sent = email_service.send_verification_email(
            user.email,
            verification_token,
            user.username
        )

        if not email_sent:
            print(f"Warning: Failed to send verification email to {user.email}")

        return jsonify({
            'status': 'success',
            'message': 'User registered successfully. Please check your email to
verify your account.',
            'user': {
                'email': user.email,
                'username': user.username,
                'is_verified': user.is_verified
            },
            'email_sent': email_sent,
            'redirect_to_login': True
        }), 201

    except Exception as e:
        db.session.rollback()
        print(f"Registration error: {str(e)}")
        import traceback
        traceback.print_exc()
        return jsonify({
            'status': 'error',
            'message': f'Registration failed: {str(e)}'
        }), 500

@auth_bp.route('/login', methods=['POST'])
```

```
def login():
    try:
        data = request.get_json()

        if not data:
            return jsonify({
                'status': 'error',
                'message': 'No data provided'
            }), 400

        email = data.get('email', '').strip().lower()
        password = data.get('password', '')

        print(f"Login attempt for email: {email}")

        if not email or not password:
            return jsonify({
                'status': 'error',
                'message': 'Email and password are required'
            }), 400

        user = User.query.filter_by(email=email).first()

        if not user:
            print(f"User not found for email: {email}")
            return jsonify({
                'status': 'error',
                'message': 'Invalid email or password'
            }), 401

        print(f"User found: {user.email}, ID: {user.id}")

        if not user.check_password(password):
            print("Password check failed")
            return jsonify({
                'status': 'error',
                'message': 'Invalid email or password'
            }), 401

        if not user.is_active:
            return jsonify({
                'status': 'error',
                'message': 'Account is deactivated'
            }), 401

        user.last_login_at = datetime.utcnow()
        db.session.commit()

        token_payload = {
            'user_id': user.id,
            'email': user.email,
            'exp': datetime.utcnow() + timedelta(days=7)  # Токен дійсний 7 днів
        }

        token = jwt.encode(token_payload, JWT_SECRET, algorithm='HS256')

        return jsonify({
            'status': 'success',
            'message': 'Login successful',
            'user': user.to_dict(),
            'token': token
        }), 200
```

```
    except Exception as e:
        print(f"Login error: {str(e)}")
        import traceback
        traceback.print_exc()
        return jsonify({
            'status': 'error',
            'message': f'Login failed: {str(e)}'
        }), 500


@auth_bp.route('/verify-email', methods=['GET'])
def verify_email():
    """Верифікація електронної пошти"""
    try:
        token = request.args.get('token')

        if not token:
            return render_template_string("""
            <!DOCTYPE html>
            <html>
            <head>
                <title>Помилка верифікації</title>
                <meta charset="UTF-8">
                <style>
                    body { font-family: Arial, sans-serif; text-align: center;
padding: 50px; }
                    .error { color: #e74c3c; }
                </style>
            </head>
            <body>
                <h1 class="error">❌ Помилка верифікації</h1>
                <p>Токен верифікації не знайдено.</p>
                <a href="http://localhost:3000">Повернутися на головну</a>
            </body>
            </html>
            """), 400

        user = User.query.filter_by(email_verification_token=token).first()

        if not user:
            return render_template_string("""
            <!DOCTYPE html>
            <html>
            <head>
                <title>Помилка верифікації</title>
                <meta charset="UTF-8">
                <style>
                    body { font-family: Arial, sans-serif; text-align: center;
padding: 50px; }
                    .error { color: #e74c3c; }
                </style>
            </head>
            <body>
                <h1 class="error">❌ Невірний токен</h1>
                <p>Токен верифікації недійсний або застарілий.</p>
                <a href="http://localhost:3000">Повернутися на головну</a>
            </body>
            </html>
            """), 400

        if not user.is_verification_token_valid(token):
            return render_template_string("""
            <!DOCTYPE html>
```

```
            <html>
            <head>
                <title>Токен застарілий</title>
                <meta charset="UTF-8">
                <style>
                    body { font-family: Arial, sans-serif; text-align: center;
padding: 50px; }
                    .error { color: #e74c3c; }
                </style>
            </head>
            <body>
                <h1 class="error">⏰ Токен застарілий</h1>
                <p>Токен верифікації застарілий. Будь ласка, запросіть новий лист
верифікації.</p>
                <a href="http://localhost:3000">Повернутися на головну</a>
            </body>
            </html>
            """), 400

        if user.is_verified:
            return render_template_string("""
            <!DOCTYPE html>
            <html>
            <head>
                <title>Вже верифіковано</title>
                <meta charset="UTF-8">
                <style>
                    body { font-family: Arial, sans-serif; text-align: center;
padding: 50px; }
                    .success { color: #27ae60; }
                </style>
            </head>
            <body>
                <h1 class="success">☑ Вже верифіковано</h1>
                <p>Ваша електронна пошта вже підтверджена.</p>
                <a href="http://localhost:3000">Перейти до додатку</a>
            </body>
            </html>
            """)

        user.verify_email()
        db.session.commit()

        return render_template_string("""
        <!DOCTYPE html>
        <html>
        <head>
            <title>Верифікація успішна</title>
            <meta charset="UTF-8">
            <style>
                body { font-family: Arial, sans-serif; text-align: center;
padding: 50px; }
                .success { color: #27ae60; }
                .button {
                    display: inline-block;
                    background: #3498db;
                    color: white;
                    padding: 10px 20px;
                    text-decoration: none;
                    border-radius: 5px;
                    margin-top: 20px;
                }
            </style>
```

```
        </head>
        <body>
            <h1 class="success">☑ Верифікація успішна!</h1>
            <p>Ваша електронна пошта успішно підтверджена. Тепер ви можете
користуватися всіма функціями додатку.</p>
            <a href="http://localhost:3000" class="button">Перейти до додатку</a>
        </body>
        </html>
        """)

    except Exception as e:
        print(f"Email verification error: {str(e)}")
        return render_template_string("""
        <!DOCTYPE html>
        <html>
        <head>
            <title>Помилка сервера</title>
            <meta charset="UTF-8">
            <style>
                body { font-family: Arial, sans-serif; text-align: center;
padding: 50px; }
                .error { color: #e74c3c; }
            </style>
        </head>
        <body>
            <h1 class="error">✗ Помилка сервера</h1>
            <p>Виникла помилка при верифікації. Спробуйте пізніше.</p>
            <a href="http://localhost:3000">Повернутися на головну</a>
        </body>
        </html>
        """), 500

@auth_bp.route('/resend-verification', methods=['POST'])
def resend_verification():
    """Повторна відправка листа верифікації"""
    try:
        data = request.get_json()

        if not data:
            return jsonify({'error': 'No data provided'}), 400

        email = data.get('email', '').strip().lower()

        if not email:
            return jsonify({'error': 'Email is required'}), 400

        user = User.query.filter_by(email=email).first()

        if not user:
            return jsonify({'error': 'User not found'}), 404

        if user.is_verified:
            return jsonify({'error': 'Email is already verified'}), 400

        verification_token = user.generate_verification_token()
        db.session.commit()

        email_sent = email_service.send_verification_email(
            user.email,
            verification_token,
            user.username
        )
```

```
        if not email_sent:
            return jsonify({'error': 'Failed to send verification email'}), 500

        return jsonify({
            'status': 'success',
            'message': 'Verification email sent successfully'
        })

    except Exception as e:
        print(f"Resend verification error: {str(e)}")
        return jsonify({'error': 'Failed to resend verification email'}), 500


@auth_bp.route('/me', methods=['GET'])
@token_required
def get_current_user(current_user):
    """Отримує інформацію про поточного користувача"""
    return jsonify({
        'status': 'success',
        'user': current_user.to_dict()
    }), 200

@auth_bp.route('/logout', methods=['POST'])
@token_required
def logout(current_user):
    """Вихід з системи (на клієнті потрібно видалити токен)"""
    return jsonify({
        'status': 'success',
        'message': 'Logged out successfully'
    }), 200

@auth_bp.route('/test-jwt', methods=['GET'])
def test_jwt():
    """Тестовий маршрут для перевірки роботи JWT"""
    try:
        exp_time = datetime.datetime.utcnow() + datetime.timedelta(minutes=5)
        test_payload = {
            'user_id': 1,
            'test': 'data',
            'exp': int(exp_time.timestamp())
        }

        print(f"Test payload: {test_payload}")

        test_token = jwt.encode(test_payload, JWT_SECRET, algorithm='HS256')

        decoded = jwt.decode(test_token, JWT_SECRET, algorithms=['HS256'])

        return jsonify({
            'status': 'success',
            'message': 'JWT working correctly',
            'jwt_version': getattr(jwt, '__version__', 'unknown'),
            'test_token': test_token if isinstance(test_token, str) else
test_token.decode('utf-8'),
            'decoded': decoded
        })
    except Exception as e:
        import traceback
        traceback.print_exc()
        return jsonify({
            'status': 'error',
            'message': f'JWT test failed: {str(e)}',
            'jwt_available': 'jwt' in globals()
```

```
        }), 500


@auth_bp.route('/simple-test', methods=['GET'])
def simple_test():
    """Найпростіший тест JWT"""
    try:
        payload = {
            'user_id': 123,
            'exp': int(datetime.time()) + 3600
        }

        token = jwt.encode(payload, 'secret', algorithm='HS256')

        decoded = jwt.decode(token, 'secret', algorithms=['HS256'])

        return jsonify({
            'status': 'success',
            'original': payload,
            'token': token,
            'decoded': decoded
        })
    except Exception as e:
        return jsonify({
            'status': 'error',
            'error': str(e)
        }), 500
```

Лістинг коду файлу з API історії транскрибування transcription_routes.py

```
from flask import Blueprint, request, jsonify
from models import db, User, Audio, Transcription
from auth_routes import token_required
from sqlalchemy import desc

transcription_bp = Blueprint('transcriptions', __name__,
url_prefix='/transcriptions')

@transcription_bp.route('/history', methods=['GET'])
@token_required
def get_user_transcription_history(current_user):
    """Отримує історію транскрибування для поточного користувача"""
    try:
        page = request.args.get('page', 1, type=int)
        per_page = request.args.get('per_page', 10, type=int)
        status = request.args.get('status')

        query = Transcription.query.filter_by(user_id=current_user.id)

        if status:
            query = query.filter_by(status=status)

        query = query.order_by(desc(Transcription.created_at))

        transcriptions = query.paginate(
            page=page,
            per_page=per_page,
            error_out=False
        )

        result = []
        for transcription in transcriptions.items:
            transcription_data = transcription.to_dict()

            if transcription.audio:
                transcription_data['audio'] = transcription.audio.to_dict()

            result.append(transcription_data)

        return jsonify({
            'status': 'success',
            'transcriptions': result,
            'pagination': {
                'page': page,
                'per_page': per_page,
                'total': transcriptions.total,
                'pages': transcriptions.pages,
                'has_next': transcriptions.has_next,
                'has_prev': transcriptions.has_prev
            }
        })

    except Exception as e:
        print(f"Error getting transcription history: {str(e)}")
        return jsonify({
            'status': 'error',
            'message': str(e)
        }), 500

@transcription_bp.route('/<transcription_uuid>', methods=['GET'])
```

```
@token_required
def get_transcription_details(current_user, transcription_uuid):
    """Отримує детальну інформацію про конкретну транскрипцію"""
    try:
        transcription = Transcription.query.filter_by(
            uuid=transcription_uuid,
            user_id=current_user.id
        ).first()

        if not transcription:
            return jsonify({
                'status': 'error',
                'message': 'Transcription not found'
            }), 404

        transcription_data = transcription.to_dict()

        if transcription.audio:
            transcription_data['audio'] = transcription.audio.to_dict()

        return jsonify({
            'status': 'success',
            'transcription': transcription_data
        })

    except Exception as e:
        print(f"Error getting transcription details: {str(e)}")
        return jsonify({
            'status': 'error',
            'message': str(e)
        }), 500

@transcription_bp.route('/<transcription_uuid>', methods=['PUT'])
@token_required
def update_transcription(current_user, transcription_uuid):
    """Оновлює текст транскрипції"""
    try:
        transcription = Transcription.query.filter_by(
            uuid=transcription_uuid,
            user_id=current_user.id
        ).first()

        if not transcription:
            return jsonify({
                'status': 'error',
                'message': 'Transcription not found'
            }), 404

        data = request.get_json()

        if 'text' in data:
            transcription.text = data['text']
            transcription.is_edited = True

        if 'speakers_text' in data:
            transcription.speakers_text = data['speakers_text']
            transcription.is_edited = True

        db.session.commit()

        return jsonify({
            'status': 'success',
            'message': 'Transcription updated successfully',
```

```
            'transcription': transcription.to_dict()
        })

    except Exception as e:
        print(f"Error updating transcription: {str(e)}")
        return jsonify({
            'status': 'error',
            'message': str(e)
        }), 500

@transcription_bp.route('/<transcription_uuid>', methods=['DELETE'])
@token_required
def delete_transcription(current_user, transcription_uuid):
    """Видаляє транскрипцію"""
    try:
        transcription = Transcription.query.filter_by(
            uuid=transcription_uuid,
            user_id=current_user.id
        ).first()

        if not transcription:
            return jsonify({
                'status': 'error',
                'message': 'Transcription not found'
            }), 404

        if transcription.audio:
            audio = transcription.audio
            import os
            if audio.file_path and os.path.exists(audio.file_path):
                try:
                    os.remove(audio.file_path)
                except Exception as e:
                    print(f"Error removing audio file: {str(e)}")

            db.session.delete(audio)

        db.session.delete(transcription)
        db.session.commit()

        return jsonify({
            'status': 'success',
            'message': 'Transcription deleted successfully'
        })

    except Exception as e:
        print(f"Error deleting transcription: {str(e)}")
        return jsonify({
            'status': 'error',
            'message': str(e)
        }), 500
```

Лістинг коду головного файлу фронтенду App.tsx

```
import React, { useState } from 'react';
import { ThemeProvider, DefaultTheme } from 'styled-components';
import usePersistedState from './utils/usePersistedState';
import { LanguageProvider } from "./contexts/language-context"

import light from './styles/themes/light';
import dark from './styles/themes/dark';

import GlobalStyle from './styles/global';
import Header from './components/Header';
import Recorder from './components/Recorder';
import TranscriptionHistory from "./components/TranscriptionHistory"
import ProtectedRoute from "./components/Auth/ProtectedRoute"
import { AuthProvider } from "./contexts/auth-context"

type ActiveTab = "recorder" | "history"

function App() {
  const [theme, setTheme] = usePersistedState<DefaultTheme>('theme', light);
  const [activeTab, setActiveTab] = useState<ActiveTab>("recorder")

  const toggleTheme = () => {
    setTheme(theme.title === 'light' ? dark : light);
  };

  return (
    <LanguageProvider>
      <AuthProvider>
        <ThemeProvider theme={theme}>
          <div className="App">
            <GlobalStyle />
            <ProtectedRoute>
            <Header toggleTheme={toggleTheme} activeTab={activeTab}
setActiveTab={setActiveTab} />
              {activeTab === "recorder" && <Recorder />}
              {activeTab === "history" && <TranscriptionHistory />}
            </ProtectedRoute>
          </div>
        </ThemeProvider>
      </AuthProvider>
    </LanguageProvider>
  );
}

export default App;
```

Лістинг коду файлу зміни мови context/language-context.tsx

```
"use client"

import type React from "react"
import { createContext, useContext, useState, type ReactNode } from "react"

export type Language = "en" | "uk"

interface LanguageContextType {
  language: Language
  setLanguage: (lang: Language) => void
  t: (key: string) => string
}

const translations = {
  en: {
    // Header
    "header.title": "Valeriia Konovalchuk",

    // Recorder
    "recorder.placeholder": "Click On Start Transcription",
    "recorder.characterCount": "Character Count",
    "recorder.transcribe": "Transcribe",
    "recorder.transcribing": "Transcribing...",
    "recorder.uploadFile": "Upload file",
    "recorder.startRecording": "Start Recording",
    "recorder.stopRecording": "Stop Recording",
    "recorder.reset": "Reset",

    // Alerts and errors
    "alert.uploadFileFirst": "Please upload an audio file before transcription.",
    "alert.microphoneError": "Unable to access microphone. Check permissions.",
    "alert.unknownError": "An unknown error occurred",
    "alert.noDataToDownload": "No data to download",
    "alert.transcriptionFailed": "Transcription failed",

    // Download modal
    "download.title": "Select format for downloading",
    "download.txt": "Download as TXT",
    "download.json": "Download as JSON",
    "download.cancel": "Cancel",

    // Navigation
    "navigation.recorder": "Recorder",
    "navigation.history": "History",

    // Transcription
    "transcription.history": "Transcription History",
    "transcription.search": "Search",
    "transcription.allStatuses": "All Statuses",
    "transcription.completed": "Completed",
    "transcription.processing": "Processing",
    "transcription.pending": "Pending",
    "transcription.failed": "Failed",
    "transcription.noTranscriptions": "No transcriptions found",
    "transcription.confirmDelete": "Are you sure you want to delete this
transcription?",
```

```
    "transcription.deleteTitle": "Confirmation of transcription deletion",
    "transcription.delete": "Delete",
    "transcription.deleteError": "Error deleting transcription",

    // Common
    "common.loading": "Loading...",
    "common.previous": "Previous",
    "common.next": "Next",
    "common.cancel": "Cancel",

    // Auth
    "auth.login": "Log in",
    "auth.register": "Register",
    "auth.logout": "Log out",
    "auth.pleaseLogin": "Please log in system",
    "auth.email": "Email",
    "auth.username": "Username",
    "auth.optional": "optional",
    "auth.password": "Password",
    "auth.confirmPassword": "Confirm password",
    "auth.noAccount": "No account?",
    "auth.haveAccount": "Already have an account?",

    //VerificationMessage
    "verificationMessage.title": "Confirm Email",
    "verificationMessage.message": "To use all the features of the application,
you must confirm your email. We have sent an email with a confirmation link to:",
    "verificationMessage.hint": "Hint:",
    "verificationMessage.hintMessage": "Check your \"Spam\" or \"Promotions\"
folder if you do not see the email in your inbox.",
    "verificationMessage.logout": "Log out",
    "verificationMessage.function": "After confirming your email, you will be able
to use all the features of the application.",
    "verificationMessage.time": "The confirmation link is valid for 24 hours.",

    //VerificationSuccess
    "verificationSuccess.title": "Registration successful! ",
    "verificationSuccess.checkEmail": "Check your email!",
    "verificationSuccess.message": "We have sent an email with a confirmation link
to the address you provided. After confirmation, you will be able to log in.",
    "verificationSuccess.resendEmail": "Resend",
    "verificationSuccess.emphasis": "Don't forget to check your \"Spam\" or
\"Promotions\" folder.",
    "verificationSuccess.login": "Go to login",
  },
  uk: {
    // Header
    "header.title": "Валерія Коновальчук",

    // Recorder
    "recorder.placeholder": "Натисність для початку транскрибування",
    "recorder.characterCount": "Кількість символів",
    "recorder.transcribe": "Транскрибувати",
    "recorder.transcribing": "Транскрибування...",
    "recorder.uploadFile": "Завантажити файл",
    "recorder.startRecording": "Почати запис",
    "recorder.stopRecording": "Зупинити запис",
    "recorder.reset": "Скинути",
```

```
    // Alerts and errors
    "alert.uploadFileFirst": "Будь ласка, завантажте аудіофайл перед
транскрибуванням.",
    "alert.microphoneError": "Не вдалося отримати доступ до мікрофону. Перевірте
дозволи.",
    "alert.unknownError": "Сталася невідома помилка",
    "alert.noDataToDownload": "Немає даних для завантаження",
    "alert.transcriptionFailed": "Транскрибування не вдалася",

    // Download modal
    "download.title": "Виберіть формат для завантаження",
    "download.txt": "Завантажити як TXT",
    "download.json": "Завантажити як JSON",
    "download.cancel": "Скасувати",

    // Navigation
    "navigation.recorder": "Запис",
    "navigation.history": "Історія",

    // Transcription
    "transcription.history": "Історія транскрипцій",
    "transcription.search": "Пошук",
    "transcription.allStatuses": "Всі статуси",
    "transcription.completed": "Завершено",
    "transcription.processing": "Обробляється",
    "transcription.pending": "Очікує",
    "transcription.failed": "Помилка",
    "transcription.noTranscriptions": "Транскрипції не знайдено",
    "transcription.deleteTitle": "Підтвердження видалення транскрипції",
    "transcription.confirmDelete": "Ви впевнені, що хочете видалити цю
транскрипцію?",
    "transcription.delete": "Видалити",
    "transcription.deleteError": "Помилка при видаленні транскрипції",

    // Common
    "common.loading": "Завантаження...",
    "common.previous": "Попередня",
    "common.next": "Наступна",
    "common.cancel": "Скасувати",

    // Auth
    "auth.login": "Авторизація",
    "auth.register": "Реєстрація",
    "auth.logout": "Вийти",
    "auth.pleaseLogin": "Будь ласка, увійдіть в систему",
    "auth.email": "Електронна пошта",
    "auth.username": "Ім'я користувача",
    "auth.optional": "необов'язково",
    "auth.password": "Пароль",
    "auth.confirmPassword": "Підтвердіть пароль",
    "auth.noAccount": "Немає акаунту?",
    "auth.haveAccount": "Вже є акаунт?",

    //VerificationMessage
    "verificationMessage.title": "Підтвердіть електронну пошту",
    "verificationMessage.message": "Для використання всіх функцій додатку
необхідно підтвердити вашу електронну пошту. Ми відправили лист з посиланням для
підтвердження на адресу:",
    "verificationMessage.hint": "Підказка:",
```

```
    "verificationMessage.hintMessage": "Перевірте папку \"Спам\" або
\"Промоакції\", якщо не бачите лист у вхідних повідомленнях.",
    "verificationMessage.logout": "Вийти з акаунту",
    "verificationMessage.function": "Після підтвердження електронної пошти ви
зможете користуватися всіма функціями додатку.",
    "verificationMessage.time": "Посилання для підтвердження дійсне протягом 24
годин.",

    //VerificationSuccess
    "verificationSuccess.title": "Реєстрація успішна!",
    "verificationSuccess.checkEmail": "Перевірте вашу пошту!",
    "verificationSuccess.message": "Ми відправили лист з посиланням для
підтвердження на вказану адресу. Після підтвердження ви зможете увійти в
систему.",
    "verificationSuccess.resendEmail": "Надіслати повторно",
    "verificationSuccess.emphasis": "Не забудьте перевірити папку \"Спам\" або
\"Промоакції\".",
    "verificationSuccess.login": "Перейти до входу",
  },
}

const LanguageContext = createContext<LanguageContextType | undefined>(undefined)

export const useLanguage = () => {
  const context = useContext(LanguageContext)
  if (!context) {
    throw new Error("useLanguage must be used within a LanguageProvider")
  }
  return context
}

interface LanguageProviderProps {
  children: ReactNode
}

export const LanguageProvider: React.FC<LanguageProviderProps> = ({ children }) =>
{
  const [language, setLanguage] = useState<Language>("en")

  const t = (key: string): string => {
    const translationsForLanguage = translations[language] as Record<string,
string>
    return translationsForLanguage[key] || key
  }

  const value = {
    language,
    setLanguage,
    t,
  }

  return <LanguageContext.Provider
value={value}>{children}</LanguageContext.Provider>
}
```

Лістинг коду файлу контексту аутентифікації context/auth-context.tsx

```
"use client"

import type React from "react"
import { createContext, useContext, useState, useEffect, type ReactNode } from
"react"
import type { User, AuthContextType, LoginCredentials, RegisterCredentials,
AuthResponse } from "../types/auth"

const AuthContext = createContext<AuthContextType | undefined>(undefined)

interface AuthProviderProps {
  children: ReactNode
}

export const AuthProvider: React.FC<AuthProviderProps> = ({ children }) => {
  const [user, setUser] = useState<User | null>(null)
  const [token, setToken] = useState<string | null>(null)
  const [isLoading, setIsLoading] = useState(true)

  useEffect(() => {
    const savedToken = localStorage.getItem("auth_token")
    const savedUser = localStorage.getItem("auth_user")

    if (savedToken && savedUser) {
      try {
        setToken(savedToken)
        setUser(JSON.parse(savedUser))
        checkUserStatus(savedToken)
      } catch (error) {
        console.error("Error parsing saved user data:", error)
        localStorage.removeItem("auth_token")
        localStorage.removeItem("auth_user")
      }
    }
    setIsLoading(false)
  }, [])

  const checkUserStatus = async (authToken: string) => {
    try {
      const response = await fetch("http://localhost:5070/auth/me", {
        headers: {
          Authorization: `Bearer ${authToken}`,
        },
      })

      if (response.ok) {
        const data = await response.json()
        if (data.status === "success") {
          setUser(data.user)
          localStorage.setItem("auth_user", JSON.stringify(data.user))
        }
      }
    } catch (error) {
      console.error("Error checking user status:", error)
    }
  }
```

```
const login = async (credentials: LoginCredentials): Promise<void> => {
  try {
    const response = await fetch("http://localhost:5070/auth/login", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(credentials),
    })

    const data: AuthResponse = await response.json()

    if (!response.ok) {
      throw new Error(data.message || "Login failed")
    }

    if (data.user && data.token) {
      setUser(data.user)
      setToken(data.token)
      localStorage.setItem("auth_token", data.token)
      localStorage.setItem("auth_user", JSON.stringify(data.user))
    }
  } catch (error) {
    console.error("Login error:", error)
    throw error
  }
}

const register = async (credentials: RegisterCredentials): Promise<void> => {
  if (credentials.password !== credentials.confirmPassword) {
    throw new Error("Passwords do not match")
  }

  try {
    const response = await fetch("http://localhost:5070/auth/register", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        email: credentials.email,
        password: credentials.password,
        username: credentials.username,
      }),
    })

    const data: AuthResponse = await response.json()

    if (!response.ok) {
      throw new Error(data.message || "Registration failed")
    }

    // if (data.user && data.token) {
    //   setUser(data.user)
    //   setToken(data.token)
    //   localStorage.setItem("auth_token", data.token)
    //   localStorage.setItem("auth_user", JSON.stringify(data.user))
    // }
```

```
    } catch (error) {
      console.error("Registration error:", error)
      throw error
    }
  }

  const logout = (): void => {
    setUser(null)
    setToken(null)
    localStorage.removeItem("auth_token")
    localStorage.removeItem("auth_user")
  }

  const value: AuthContextType = {
    user,
    token,
    isAuthenticated: !!user && !!token,
    isLoading,
    login,
    register,
    logout,
  }

  return <AuthContext.Provider value={value}>{children}</AuthContext.Provider>
}

export const useAuth = (): AuthContextType => {
  const context = useContext(AuthContext)
  if (context === undefined) {
    throw new Error("useAuth must be used within an AuthProvider")
  }
  return context
}
```

Лістинг коду файлу захисту маршрутів аутентифікації

Auth/ProtectedRoute.tsx

```
"use client"

import type React from "react"
import { useAuth } from "../../contexts/auth-context"
import { useLanguage } from "../../contexts/language-context"
import AuthPage from "./AuthPage"
import VerificationMessage from "./VerificationMessage"

interface ProtectedRouteProps {
  children: React.ReactNode
}
const ProtectedRoute: React.FC<ProtectedRouteProps> = ({ children }) => {
  const { user, isAuthenticated, isLoading, logout } = useAuth()
  const { t } = useLanguage()

  if (isLoading) {
    return (
      <div
        style={{
          display: "flex",
          justifyContent: "center",
          alignItems: "center",
          minHeight: "100vh",
          fontSize: "1.2rem",
        }}
      >
        Завантаження...
      </div>
    )
  }
  if (!isAuthenticated) {
    return <AuthPage />
  }
  if (user && !user.is_verified) {
    const handleResendEmail = async () => {
      const response = await fetch("http://localhost:5070/auth/resend-
verification", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({ email: user.email }),
      })
      if (!response.ok) {
        const data = await response.json()
        throw new Error(data.error || "Failed to resend verification email")
      }
    }

    return <VerificationMessage userEmail={user.email}
onResendEmail={handleResendEmail} onLogout={logout} />
  }

  return <>{children}</>
}

export default ProtectedRoute
```

Лістинг коду компоненту зміни мови Language/index.tsx

```
"use client"

import type React from "react"
import { useState, useRef, useEffect } from "react"
import { Globe, ChevronDown } from "lucide-react"
import { useLanguage, type Language } from "../../contexts/language-context"
import {
  LanguageSelectorContainer,
  LanguageButton,
  LanguageIcon,
  DropdownIcon,
  DropdownMenu,
  DropdownItem,
  FlagIcon,
} from "./styles"

const LanguageSelector: React.FC = () => {
  const { language, setLanguage } = useLanguage()
  const [isOpen, setIsOpen] = useState(false)
  const containerRef = useRef<HTMLDivElement>(null)

  const languages = [
    { code: "en" as Language, flag: "EN" },
    { code: "uk" as Language, flag: "UA" },
  ]

  const currentLanguage = languages.find((lang) => lang.code === language)

  const handleLanguageSelect = (langCode: Language) => {
    setLanguage(langCode)
    setIsOpen(false)
  }

  const toggleDropdown = () => {
    setIsOpen(!isOpen)
  }

  useEffect(() => {
    const handleClickOutside = (event: MouseEvent) => {
      if (containerRef.current && !containerRef.current.contains(event.target as
Node)) {
        setIsOpen(false)
      }
    }

    document.addEventListener("mousedown", handleClickOutside)
    return () => {
      document.removeEventListener("mousedown", handleClickOutside)
    }
  }, [])

  return (
    <LanguageSelectorContainer ref={containerRef}>
      <LanguageButton onClick={toggleDropdown} isOpen={isOpen}>
        <LanguageIcon>
          <Globe size={16} />
        </LanguageIcon>
        <FlagIcon>{currentLanguage?.flag}</FlagIcon>
        <DropdownIcon isOpen={isOpen}>
          <ChevronDown size={14} />
```

```
        </DropdownIcon>
      </LanguageButton>

      {isOpen && (
        <DropdownMenu>
          {languages.map((lang) => (
            <DropdownItem
              key={lang.code}
              onClick={() => handleLanguageSelect(lang.code)}
              isSelected={lang.code === language}
            >
              <FlagIcon>{lang.flag}</FlagIcon>
            </DropdownItem>
          ))}
        </DropdownMenu>
      )}
    </LanguageSelectorContainer>
  )
}

export default LanguageSelector
```

Лістинг коду компоненту транскрибування Recorder/index.tsx

```
"use client";

import React, { useState, useRef, useCallback, useEffect } from "react";
import { Mic, Save, Upload, Square } from "lucide-react";
import { useLanguage } from "../../contexts/language-context"
import { useAuth } from "../../contexts/auth-context"
import {
  RecorderContainer,
  RecorderCard,
  IconButton,
  TranscriptBox,
  Placeholder,
  CharCount,
  ButtonGroup,
  Button,
  ModalOverlay,
  ModalContent,
  ModalButton,
  RecordingIndicator,
} from './styles';

interface TranscriptionData {
  text: string;
  speakers_text?: string;
  speakers?: Record<string, Array<{start: number; end: number; text: string}>>;
}

const Recorder: React.FC = () => {
  const { t } = useLanguage()
  const { user, token } = useAuth()
  const [uploadedFileName, setUploadedFileName] = useState<string | null>(null);
  const [loading, setLoading] = useState(false);
  const [uploadedFile, setUploadedFile] = useState<File | null>(null);
  const [transcription, setTranscription] = useState<TranscriptionData>({text:
""});
  const [hasScroll, setHasScroll] = useState(false);
  const [isRecording, setIsRecording] = useState(false);
  const [recordedAudio, setRecordedAudio] = useState<Blob | null>(null);
  const [showDownloadModal, setShowDownloadModal] = useState(false);

  const transcriptBoxRef = useRef<HTMLDivElement>(null);
  const mediaRecorderRef = useRef<MediaRecorder | null>(null);
  const audioChunksRef = useRef<Blob[]>([]);

  const checkScroll = useCallback(() => {
    if (transcriptBoxRef.current) {
      setHasScroll(
        transcriptBoxRef.current.scrollHeight >
transcriptBoxRef.current.clientHeight
      );
    }
  }, []);

  useEffect(() => {
    checkScroll();
    window.addEventListener('resize', checkScroll);
```

```
    return () => window.removeEventListener('resize', checkScroll);
  }, [checkScroll, transcription]);

  useEffect(() => {
    console.log("Auth state:", { user, token: token ? "present" : "missing" })
    if (!user || !token) {
      console.warn("User not authenticated")
    }
  }, [user, token])

  const handleReset = () => {
    setTranscription({text: ""});
    setUploadedFileName(null);
    setUploadedFile(null);
    setRecordedAudio(null);
    setIsRecording(false);

    if (mediaRecorderRef.current && isRecording) {
      mediaRecorderRef.current.stop();
    }
  };

  const startRecording = async () => {
    try {
      const stream = await navigator.mediaDevices.getUserMedia({ audio: true });
      const mediaRecorder = new MediaRecorder(stream);
      mediaRecorderRef.current = mediaRecorder;
      audioChunksRef.current = [];

      mediaRecorder.ondataavailable = (event) => {
        if (event.data.size > 0) {
          audioChunksRef.current.push(event.data);
        }
      };

      mediaRecorder.onstop = () => {
        const audioBlob = new Blob(audioChunksRef.current, { type: 'audio/wav' });
        setRecordedAudio(audioBlob);

        const audioFile = new File([audioBlob], `recording_${Date.now()}.wav`, {
type: 'audio/wav' });
        setUploadedFile(audioFile);
        setUploadedFileName(audioFile.name);
        setTranscription({text: ""});

        stream.getTracks().forEach(track => track.stop());
      };

      mediaRecorder.start();
      setIsRecording(true);
    } catch (error) {
      console.error('Error accessing microphone:', error);
      alert(t("alert.microphoneError"));
    }
  };

  const stopRecording = () => {
    if (mediaRecorderRef.current && isRecording) {
      mediaRecorderRef.current.stop();
```

```
    setIsRecording(false);
  }
};

const handleRecordingToggle = () => {
  if (isRecording) {
    stopRecording();
  } else {
    startRecording();
  }
};

const handleUpload = (e: React.ChangeEvent<HTMLInputElement>) => {
  const file = e.target.files?.[0];
  if (file) {
    setUploadedFile(file);
    setUploadedFileName(file.name);
    setTranscription({text: ""});
    setRecordedAudio(null);
  }
};

const handleTranscribe = async () => {
  if (!user || !token) {
    alert(t("auth.loginRequired") || "Please log in to transcribe files")
    return
  }

  if (!uploadedFile) {
    alert(t("alert.uploadFileFirst"));
    return;
  }

  setLoading(true);

  const formData = new FormData();
  formData.append("file", uploadedFile);

  try {
    const response = await fetch("http://localhost:5070/upload", {
      method: "POST",
      body: formData,
      headers: {
        // 'Accept': 'application/json',
        Authorization: `Bearer ${token}`,
      },
    })

    console.log("Response status:", response.status)

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.error || t("alert.transcriptionFailed"));
    }

    const uploadData = await response.json();
    const transcriptionUuid = uploadData.uuid;

    // Перевіряємо статус транскрипції кожні 2 секунди
```

```
      const checkStatus = async () => {
        try {
          const statusResponse = await
fetch(`http://localhost:5070/status/${transcriptionUuid}`, {
            headers: {
              Authorization: `Bearer ${token}`,
            },
          });

          const statusData = await statusResponse.json();

          if (statusData.status === 'completed') {
            setTranscription({
              text: statusData.text,
              speakers_text: statusData.speakers_text,
              speakers: statusData.speakers
            });
            setLoading(false);
          } else if (statusData.status === 'failed') {
            throw new Error("Transcription failed");
          } else {
            // Продовжуємо перевіряти статус
            setTimeout(checkStatus, 2000);
          }
        } catch (error) {
          console.error("Status check error:", error);
          setLoading(false);
          alert("Error checking transcription status");
        }
      };

      // Починаємо перевіряти статус
      setTimeout(checkStatus, 2000);

    } catch (error) {
      console.error("Transcription error:", error);
      setLoading(false);
      alert("Transcription failed");
    }
  };

  const downloadFile = (format: 'txt' | 'json') => {
    if (!transcription.text && !transcription.speakers_text) {
      alert(t("alert.noDataToDownload"));
      return;
    }

    let content: string;
    let filename: string;
    let mimeType: string;

    if (format === 'txt') {
      content = transcription.speakers_text || transcription.text;
      filename = `transcription_${Date.now()}.txt`;
      mimeType = 'text/plain';
    } else {
      content = JSON.stringify(transcription, null, 2);
      filename = `transcription_${Date.now()}.json`;
      mimeType = 'application/json';
```

```
  }

  const blob = new Blob([content], { type: mimeType });
  const url = URL.createObjectURL(blob);
  const link = document.createElement('a');
  link.href = url;
  link.download = filename;
  document.body.appendChild(link);
  link.click();
  document.body.removeChild(link);
  URL.revokeObjectURL(url);

  setShowDownloadModal(false);
};

const handleDownloadClick = () => {
  if (!transcription.text && !transcription.speakers_text) {
    alert(t("alert.noDataToDownload"));
    return;
  }
  setShowDownloadModal(true);
};

const renderTranscription = () => {
  if (transcription.speakers_text) {
    return (
      <div style={{
        whiteSpace: 'pre-wrap',
        textAlign: 'left',
        fontFamily: 'monospace',
        fontSize: '0.95rem'
      }}>
        {transcription.speakers_text.split('\n').map((line, i) => {
            if (line.startsWith('===')) {
                return (
                    <div key={i} style={{
                        fontWeight: 'bold',
                        margin: '15px 0 5px',
                        color: '#4a6baf',
                        fontSize: '1.05em'
                    }}>
                        {line}
                    </div>
                );
            }
            return <div key={i}>{line}</div>;
        })}
      </div>
    );
  }
  if (transcription.text) {
    return (
      <div style={{ fontSize: '0.95rem' }}>
        {transcription.text}
      </div>
    );
  }
  if (uploadedFileName) {
    return uploadedFileName;
```

```
    }
    return t("recorder.placeholder");
  };

  const hasTranscriptionText = !!(transcription.text ||
transcription.speakers_text);

  if (!user) {
    return (
      <RecorderContainer>
        <RecorderCard>
          <div style={{ textAlign: "center", padding: "2rem" }}>
            <h3>{t("auth.loginRequired") || "Please log in to use
transcription"}</h3>
            <p>{t("auth.loginToTranscribe") || "You need to be logged in to
transcribe audio files."}</p>
          </div>
        </RecorderCard>
      </RecorderContainer>
    )
  }

  return (
    <RecorderContainer>
      <RecorderCard>
        <IconButton top onClick={handleDownloadClick}>
          <Save />
        </IconButton>

        <TranscriptBox ref={transcriptBoxRef}>
          <Placeholder
            hasText={hasTranscriptionText || !!uploadedFileName}
            isPlaceholder={!hasTranscriptionText}
          >
          {renderTranscription()}
          </Placeholder>
        </TranscriptBox>

        <CharCount>
          {t("recorder.characterCount")}: {transcription.text?.length || 0}
        </CharCount>

        <ButtonGroup>
          <Button variant="main" onClick={handleTranscribe} disabled={loading}>
            {loading ? t("recorder.transcribing") : t("recorder.transcribe")}
          </Button>
        </ButtonGroup>

        <ButtonGroup>
          <label htmlFor="audio-upload" style={{ flex: 1 }}>
            <Button as="span" variant="upload" style={{ width: "100%" }}>
              <Upload style={{ marginRight: "0.5rem" }} />
              {t("recorder.uploadFile")}
            </Button>
          </label>
          <input
            type="file"
            accept="audio/*"
            id="audio-upload"
```

```
            style={{ display: "none" }}
            onChange={handleUpload}
          />
          <Button
            variant={isRecording ? "recording" : "main"}
            style={{ flex: 1, position: 'relative' }}
            onClick={handleRecordingToggle}
          >
            {isRecording ? <Square /> : <Mic />}
            {isRecording ? t("recorder.stopRecording") :
t("recorder.startRecording")}
            {isRecording && <RecordingIndicator />}
          </Button>
        </ButtonGroup>

        <ButtonGroup>
          <Button
            variant="secondary"
            style={{ width: "100%" }}
            onClick={handleReset}
          >
            {t("recorder.reset")}
          </Button>
        </ButtonGroup>
      </RecorderCard>

      {showDownloadModal && (
        <ModalOverlay onClick={() => setShowDownloadModal(false)}>
          <ModalContent onClick={(e) => e.stopPropagation()}>
            <h3>{t("download.title")}</h3>
            <div style={{ display: 'flex', gap: '1rem', marginTop: '1rem' }}>
            <ModalButton onClick={() => downloadFile("txt")}>
              {t("download.txt")}
            </ModalButton>
            <ModalButton onClick={() => downloadFile("json")}>
              {t("download.json")}
            </ModalButton>
            </div>
            <ModalButton
              variant="secondary"
              onClick={() => setShowDownloadModal(false)}
              style={{ marginTop: '1rem', width: '100%' }}
            >
              {t("download.cancel")}
            </ModalButton>
          </ModalContent>
        </ModalOverlay>
      )}
    </RecorderContainer>
  );
};

export default Recorder;
```

Лістинг коду компоненту історії транскрибування

TranscriptionHistory/index.tsx

```
"use client"

import type React from "react"
import { useState, useEffect } from "react"
import styled from "styled-components"
import { FileAudio, Clock, Calendar, Edit, Trash2, Eye, ChevronLeft, ChevronRight,
Search, Filter } from "lucide-react"
import { useAuth } from "../../contexts/auth-context"
import { useLanguage } from "../../contexts/language-context"
import ConfirmationModal from "../ConfirmationModal"

// Типи даних
interface AudioData {
  id: string
  filename: string
  file_size: number
  duration: number
  format: string
  created_at: string
}

interface TranscriptionData {
  id: string
  text: string
  speakers_text: string
  speakers: any
  language: string
  status: "pending" | "processing" | "completed" | "failed"
  is_edited: boolean
  created_at: string
  updated_at: string
  audio: AudioData
}

interface PaginationData {
  page: number
  per_page: number
  total: number
  pages: number
  has_next: boolean
  has_prev: boolean
}


const TranscriptionHistory: React.FC = () => {
  const { user, token } = useAuth()
  const { t } = useLanguage()

  const [transcriptions, setTranscriptions] = useState<TranscriptionData[]>([])
  const [pagination, setPagination] = useState<PaginationData | null>(null)
  const [loading, setLoading] = useState(true)
  const [error, setError] = useState<string | null>(null)
  const [currentPage, setCurrentPage] = useState(1)
  const [statusFilter, setStatusFilter] = useState<string>("")
  const [searchTerm, setSearchTerm] = useState("")

  const [confirmModalOpen, setConfirmModalOpen] = useState(false)
```

```
  const [transcriptionToDelete, setTranscriptionToDelete] =
useState<TranscriptionData | null>(null)

  const fetchTranscriptions = async (page = 1) => {
    try {
      setLoading(true)
      setError(null)

      if (!token) {
        throw new Error("No authentication token")
      }

      const params = new URLSearchParams({
        page: page.toString(),
        per_page: "10",
      })

      if (statusFilter) {
        params.append("status", statusFilter)
      }

      const response = await
fetch(`http://localhost:5070/transcriptions/history?${params}`, {
        headers: {
          Authorization: `Bearer ${token}`,
          "Content-Type": "application/json",
        },
      })

      if (!response.ok) {
        throw new Error(`HTTP error! status: ${response.status}`)
      }

      const data = await response.json()

      if (data.status === "success") {
        setTranscriptions(data.transcriptions)
        setPagination(data.pagination)
      } else {
        throw new Error(data.message || "Failed to fetch transcriptions")
      }
    } catch (err) {
      console.error("Error fetching transcriptions:", err)
      setError(err instanceof Error ? err.message : "Unknown error occurred")
    } finally {
      setLoading(false)
    }
  }

  useEffect(() => {
    if (user) {
      fetchTranscriptions(currentPage)
    }
  }, [user, currentPage, statusFilter])

  const formatDate = (dateString: string) => {
    return new Date(dateString).toLocaleDateString("uk-UA", {
      year: "numeric",
      month: "short",
      day: "numeric",
      hour: "2-digit",
      minute: "2-digit",
    })
```

```
  }

  const formatDuration = (seconds: number) => {
    const minutes = Math.floor(seconds / 60)
    const remainingSeconds = Math.floor(seconds % 60)
    return `${minutes}:${remainingSeconds.toString().padStart(2, "0")}`
  }

  const formatFileSize = (bytes: number) => {
    const sizes = ["Bytes", "KB", "MB", "GB"]
    if (bytes === 0) return "0 Bytes"
    const i = Math.floor(Math.log(bytes) / Math.log(1024))
    return Math.round((bytes / Math.pow(1024, i)) * 100) / 100 + " " + sizes[i]
  }

  const handlePageChange = (newPage: number) => {
    setCurrentPage(newPage)
  }

  const handleStatusFilterChange = (event: React.ChangeEvent<HTMLSelectElement>)
=> {
    setStatusFilter(event.target.value)
    setCurrentPage(1)
  }

  const handleViewTranscription = (transcription: TranscriptionData) => {
    console.log("View transcription:", transcription)
  }

  const handleEditTranscription = (transcription: TranscriptionData) => {
    console.log("Edit transcription:", transcription)
  }

  const handleDeleteClick = (transcription: TranscriptionData) => {
    setTranscriptionToDelete(transcription)
    setConfirmModalOpen(true)
  }

  const confirmDelete = async () => {
    if (!transcriptionToDelete) return

    try {
      const response = await
fetch(`http://localhost:5070/transcriptions/${transcriptionToDelete.id}`, {
        method: "DELETE",
        headers: {
          Authorization: `Bearer ${token}`,
        },
      })

      if (response.ok) {
        fetchTranscriptions(currentPage)
      } else {
        throw new Error("Failed to delete transcription")
      }
    } catch (err) {
      console.error("Error deleting transcription:", err)
      alert(t("transcription.deleteError"))
    } finally {
      setConfirmModalOpen(false)
      setTranscriptionToDelete(null)
    }
  }
```

```
  const cancelDelete = () => {
    setConfirmModalOpen(false)
    setTranscriptionToDelete(null)
  }

  const filteredTranscriptions = transcriptions.filter(
    (transcription) =>

transcription.audio.filename.toLowerCase().includes(searchTerm.toLowerCase()) ||
      (transcription.text &&
transcription.text.toLowerCase().includes(searchTerm.toLowerCase())),
  )

  if (!user) {
    return (
      <Container>
        <HistoryCard>
          <LoadingMessage>{t("auth.pleaseLogin")}</LoadingMessage>
        </HistoryCard>
      </Container>
    )
  }

  return (
    <Container>
      <HistoryCard>
        <Header>
          <Title>{t("transcription.history")}</Title>
          <Controls>
            <div style={{ position: "relative" }}>
              <Search
                size={16}
                style={{ position: "absolute", left: "12px", top: "50%",
transform: "translateY(-50%)", opacity: 0.5 }}
              />
              <SearchInput
                type="text"
                placeholder={t("transcription.search")}
                value={searchTerm}
                onChange={(e) => setSearchTerm(e.target.value)}
                style={{ paddingLeft: "40px" }}
              />
            </div>
            <div style={{ position: "relative" }}>
              <Filter
                size={16}
                style={{ position: "absolute", left: "12px", top: "50%",
transform: "translateY(-50%)", opacity: 0.5 }}
              />
              <FilterSelect value={statusFilter}
onChange={handleStatusFilterChange} style={{ paddingLeft: "40px" }}>
                <option value="">{t("transcription.allStatuses")}</option>
                <option value="completed">{t("transcription.completed")}</option>
                <option
value="processing">{t("transcription.processing")}</option>
                <option value="pending">{t("transcription.pending")}</option>
                <option value="failed">{t("transcription.failed")}</option>
              </FilterSelect>
            </div>
          </Controls>
        </Header>
```

```
        {loading && <LoadingMessage>{t("common.loading")}</LoadingMessage>}

        {error && <ErrorMessage>{error}</ErrorMessage>}

        {!loading && !error && filteredTranscriptions.length === 0 && (
          <EmptyMessage>{t("transcription.noTranscriptions")}</EmptyMessage>
        )}

        {!loading && !error && filteredTranscriptions.length > 0 && (
          <>
            <TranscriptionGrid>
              {filteredTranscriptions.map((transcription) => (
                <TranscriptionCard key={transcription.id}>
                  <CardHeader>
                    <FileName>{transcription.audio.filename}</FileName>
                    <StatusBadge status={transcription.status}>
                      {t(`transcription.${transcription.status}`)}
                    </StatusBadge>
                  </CardHeader>

                  <CardContent>
                    <MetaInfo>
                      <MetaItem>
                        <Calendar size={14} />
                        {formatDate(transcription.created_at)}
                      </MetaItem>
                      {transcription.audio.duration && (
                        <MetaItem>
                          <Clock size={14} />
                          {formatDuration(transcription.audio.duration)}
                        </MetaItem>
                      )}
                      <MetaItem>
                        <FileAudio size={14} />
                        {formatFileSize(transcription.audio.file_size)}
                      </MetaItem>
                    </MetaInfo>

                    {transcription.text &&
<TextPreview>{transcription.text}</TextPreview>}
                  </CardContent>

                  <CardActions>
                    <ActionButton onClick={() =>
handleViewTranscription(transcription)}>
                      <Eye size={16} />
                    </ActionButton>
                    <ActionButton onClick={() =>
handleEditTranscription(transcription)}>
                      <Edit size={16} />
                    </ActionButton>
                    <ActionButton onClick={() =>
handleDeleteClick(transcription)}>
                      <Trash2 size={16} />
                    </ActionButton>
                  </CardActions>
                </TranscriptionCard>
              ))}
            </TranscriptionGrid>

            {pagination && pagination.pages > 1 && (
              <Pagination>
```

```
                <PaginationButton disabled={!pagination.has_prev} onClick={() =>
handlePageChange(currentPage - 1)}>
                  <ChevronLeft size={16} />
                  {t("common.previous")}
                </PaginationButton>

                <PageInfo>{`${pagination.page} з ${pagination.pages}`}</PageInfo>

                <PaginationButton disabled={!pagination.has_next} onClick={() =>
handlePageChange(currentPage + 1)}>
                  {t("common.next")}
                  <ChevronRight size={16} />
                </PaginationButton>
              </Pagination>
            )}
          </>
        )}

        <ConfirmationModal
          isOpen={confirmModalOpen}
          title={t("transcription.deleteTitle")}
          message={t("transcription.confirmDelete")}
          confirmText={t("transcription.delete")}
          cancelText={t("common.cancel")}
          onConfirm={confirmDelete}
          onCancel={cancelDelete}
          isDanger
        />
      </HistoryCard>
    </Container>
  )
}

export default TranscriptionHistory
```

Лістинг коду компоненту реєстрації Auth/RegisterForm.tsx

```
"use client"

import type React from "react"
import { useState } from "react"
import styled from "styled-components"
import { UserPlus, Mail, Lock, User } from "lucide-react"
import { useAuth } from "../../contexts/auth-context"
import { useLanguage } from "../../contexts/language-context"
import type { RegisterCredentials } from "../../types/auth"

const FormCard = styled.div`
  width: 90%;
  max-width: 450px;
  min-width: 350px;
  background-color: ${(props) => props.theme.colors.primary};
  color: ${(props) => props.theme.colors.text};
  padding: 2rem;
  border-radius: 1rem;
  box-shadow: 0 0 15px rgba(0, 0, 0, 0.3);
  position: relative;
  display: flex;
  flex-direction: column;
  gap: 1.5rem;
`

const Title = styled.h2`
  text-align: center;
  margin: 0;
  color: ${(props) => props.theme.colors.text};
  font-size: 1.8rem;
  font-weight: 600;
`

const Form = styled.form`
  display: flex;
  flex-direction: column;
  gap: 1.5rem;
`

const InputGroup = styled.div`
  display: flex;
  flex-direction: column;
  gap: 0.5rem;
`

const Label = styled.label`
  font-weight: 500;
  color: ${(props) => props.theme.colors.text};
  font-size: 0.9rem;
  display: flex;
  align-items: center;
  gap: 0.5rem;
`

const Input = styled.input`
  padding: 0.75rem 1rem;
  border: 2px solid transparent;
  border-radius: 0.5rem;
  font-size: 1rem;
  background: rgba(255, 255, 255, 0.1);
```

```
  color: ${(props) => props.theme.colors.text};
  transition: all 0.2s ease;

  &::placeholder {
    color: rgba(255, 255, 255, 0.5);
  }

  &:focus {
    outline: none;
    border-color: ${(props) => props.theme.colors.accent};
    background: rgba(255, 255, 255, 0.15);
  }

  &:hover {
    background: rgba(255, 255, 255, 0.12);
  }
`

const Button = styled.button`
  padding: 0.75rem 1.5rem;
  border-radius: 0.5rem;
  font-weight: 600;
  font-size: 1rem;
  color: #fff;
  background-color: ${(props) => props.theme.colors.accent};
  border: none;
  cursor: pointer;
  transition: all 0.2s ease;
  display: flex;
  align-items: center;
  justify-content: center;
  gap: 0.5rem;

  &:hover {
    background-color: ${(props) => props.theme.colors.accentDark};
    transform: translateY(-1px);
  }

  &:disabled {
    opacity: 0.6;
    cursor: not-allowed;
    transform: none;
  }
`

const ErrorMessage = styled.div`
  color: #ff6b6b;
  text-align: center;
  font-size: 0.9rem;
  padding: 0.5rem;
  background: rgba(255, 107, 107, 0.1);
  border-radius: 0.5rem;
  border: 1px solid rgba(255, 107, 107, 0.3);
`

const SwitchText = styled.p`
  text-align: center;
  margin: 0;
  color: ${(props) => props.theme.colors.text};
  font-size: 0.9rem;
`

const SwitchLink = styled.span`
```

```
  color: ${(props) => props.theme.colors.accent};
  cursor: pointer;
  font-weight: 600;
  transition: color 0.2s ease;

  &:hover {
    color: ${(props) => props.theme.colors.accentDark};
    text-decoration: underline;
  }
`

const OptionalText = styled.span`
  font-size: 0.8rem;
  color: rgba(255, 255, 255, 0.6);
  font-weight: 400;
`

interface RegisterFormProps {
  onSwitchToLogin: () => void
  onRegistrationSuccess: (email: string, message: string) => void
}

const RegisterForm: React.FC<RegisterFormProps> = ({ onSwitchToLogin,
onRegistrationSuccess }) => {
  const { t } = useLanguage()
  const [credentials, setCredentials] = useState<RegisterCredentials>({
    email: "",
    password: "",
    confirmPassword: "",
    username: "",
  })
  const [isLoading, setIsLoading] = useState(false)
  const [error, setError] = useState<string | null>(null)

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault()
    setError(null)

    if (!credentials.email || !credentials.password ||
!credentials.confirmPassword) {
      setError(t("auth.fillAllFields"))
      return
    }

    if (credentials.password.length < 6) {
      setError(t("auth.passwordTooShort"))
      return
    }

    if (credentials.password !== credentials.confirmPassword) {
      setError(t("auth.passwordsDoNotMatch"))
      return
    }

    setIsLoading(true)
    try {
      const response = await fetch("http://localhost:5070/auth/register", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          email: credentials.email,
```

```
          password: credentials.password,
          username: credentials.username,
        }),
      })

      const data = await response.json()

      if (!response.ok) {
        throw new Error(data.message || "Registration failed")
      }

      if (data.status === "success") {
        // Показуємо повідомлення про успішну реєстрацію
        onRegistrationSuccess(credentials.email, data.message)
      }
    } catch (error) {
      setError(error instanceof Error ? error.message :
t("auth.registrationFailed"))
    } finally {
      setIsLoading(false)
    }
  }

  const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    const { name, value } = e.target
    setCredentials((prev) => ({
      ...prev,
      [name]: value,
    }))
  }

  return (
    <FormCard>
      <Title>{t("auth.register")}</Title>
      <Form onSubmit={handleSubmit}>
        <InputGroup>
          <Label htmlFor="email">
            <Mail size={16} />
            {t("auth.email")}
          </Label>
          <Input
            type="email"
            id="email"
            name="email"
            value={credentials.email}
            onChange={handleChange}
            placeholder="your@email.com"
            required
          />
        </InputGroup>
        <InputGroup>
          <Label htmlFor="username">
            <User size={16} />
            {t("auth.username")}
<OptionalText>({t("auth.optional")})</OptionalText>
          </Label>
          <Input
            type="text"
            id="username"
            name="username"
            value={credentials.username}
            onChange={handleChange}
            placeholder="username"
```

```
            />
          </InputGroup>
          <InputGroup>
            <Label htmlFor="password">
              <Lock size={16} />
              {t("auth.password")}
            </Label>
            <Input
              type="password"
              id="password"
              name="password"
              value={credentials.password}
              onChange={handleChange}
              placeholder="••••••••"
              required
              minLength={6}
            />
          </InputGroup>
          <InputGroup>
            <Label htmlFor="confirmPassword">
              <Lock size={16} />
              {t("auth.confirmPassword")}
            </Label>
            <Input
              type="password"
              id="confirmPassword"
              name="confirmPassword"
              value={credentials.confirmPassword}
              onChange={handleChange}
              placeholder="••••••••"
              required
            />
          </InputGroup>
          <Button type="submit" disabled={isLoading}>
            {isLoading ? (
              "..."
            ) : (
              <>
                <UserPlus size={16} />
                {t("auth.register")}
              </>
            )}
          </Button>
          {error && <ErrorMessage>{error}</ErrorMessage>}
        </Form>
        <SwitchText>
          {t("auth.haveAccount")} <SwitchLink
onClick={onSwitchToLogin}>{t("auth.login")}</SwitchLink>
        </SwitchText>
      </FormCard>
    )
}

export default RegisterForm
```