

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»**

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

«До захисту допущено»

Завідувач кафедри

Наталія АУШЕВА

“ ” 2026 р.

**Дипломна робота
на здобуття ступеня бакалавр**

За освітньою програмою “Цифрові технології в енергетиці”

Спеціальності 122 “Комп’ютерні науки”

на тему: “ Веб-система автоматизованого перекладу мультимедійного контенту ”

Виконав: студентка 4 курсу, групи ТР-25

Мовчан Анастасія Анатоліївна

(прізвище, ім’я, по батькові)

_____ (підпис)

Керівник доцент каф. ЦТЕ, к.т.н. Володимир ТИХОХОД

(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ)

_____ (підпис)

Рецензент доктор філософії, ст. викладач Ольга ВЛАСЕНКО

(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ)

_____ (підпис)

Н.контроль асистент Костянтин ЗДОР

(посада, ім’я, ПРІЗВИЩЕ)

_____ (підпис)

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів без
відповідних посилань.

Студент _____

Київ - 2026

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”**

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)

спеціальність 122 “Комп’ютерні науки”

Освітньо-професійна програма “Цифрові технології в енергетиці”

«До захисту допущено»

Завідувач кафедри

Наталія АУШЕВА

“ ” 2026 р.

ЗАВДАННЯ

на дипломну роботу студенту

Мовчан Анастасії Анатоліївни

(прізвище, ім’я, по батькові)

1. Тема роботи “ Веб-система автоматизованого перекладу мультимедійного контенту

Науковий керівник доцент каф. ЦТЕ, к.т.н. Володимир ТИХОХОД

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “28” травня 2026 року № 1992-с

2. Термін подання студентом роботи 04.06.2026

3. Вихідні дані до роботи Python 3.10, FastAPI, Uvicorn, PostgreSQL 13+, OpenAI Whisper (base), Tesseract OCR 5.x, FFmpeg, DeepL API, psycopg2, PyJWT, bcrypt, Pillow

4. Перелік питань, які потрібно розробити _____

1) провести аналіз серед наявних аналогів

2) визначити головні функції веб-застосунку, що керує послугами та ресурсами

3) аргументувати вибрані інструменти, алгоритми та технології для реалізації веб-системи

- 4) побудувати архітектуру програмного забезпечення та бази даних
- 5) створити прикладний програмний веб-інтерфейс
- 6) представити процес застосування веб-інтерфейсу
5. Орієнтований перелік ілюстративного матеріалу матеріалу діаграма прецедентів (варіантів використання), концептуальна модель бази даних; інтерфейси користувача.
6. Дата видачі завдання 16.09.2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вибір теми роботи	16.09.2025	виконано
2.	Аналіз методів та засобів розв'язання задачі	20.04.2026-21.04.2026	виконано
3.	Розробка архітектури та загальної структури системи	23.04.2026-26.04.2026	виконано
4.	Розробка окремих підсистем	30.04.2026-06.05.2026	виконано
5.	Програмна реалізація системи	07.05.2026-10.05.2026	виконано
6.	Оформлення пояснювальної записки	11.05.2026-17.05.2026	виконано
7.	Захист програмного забезпечення	14.05.2026	виконано
8.	Передзахист	04.06.2026	виконано
9.	Захист	15.05.2026	виконано

Студентка

(підпис)

Анастасія МОВЧАН

(прізвище та ініціали)

Керівник

(підпис)

Володимир ТИХОХОД

(прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота виконана на 64 сторінках, містить 12 ілюстрацій, 10 таблиць, 1 додаток, 21 джерел у переліку посилань.

Метою роботи є вирішення прикладної задачі автоматизованого перекладу мультимедійного контенту шляхом розробки веб-системи з уніфікованим конвеєром обробки файлів різних форматів.

У процесі розробки застосовано клієнт-серверну архітектуру з програмним інтерфейсом на основі принципів REST, методи автоматичного розпізнавання мовлення на основі нейромережевої трансформерної моделі, оптичного розпізнавання символів та нейромережевого машинного перекладу. Для захисту даних використано адаптивне хешування паролів та автентифікацію на основі підписаних токенів. Зберігання даних реалізовано засобами реляційної бази даних з підтримкою каскадних операцій та індексування.

Результатом роботи є веб-система, що автоматично визначає тип завантаженого файлу та виконує повний цикл обробки: розпізнавання тексту або мовлення, переклад на обрану мову та збереження результату у відповідному форматі. Система підтримує вісім форматів вхідних файлів, реалізує механізм кешування повторних перекладів та персональний журнал операцій для кожного користувача. Працездатність системи підтверджено функціональним та нефункціональним тестуванням

Практичне значення роботи полягає у створенні готового до використання програмного продукту, що усуває необхідність застосування кількох розрізнених інструментів при роботі з різнотипним мультимедійним контентом і не потребує від користувача технічної підготовки.

Ключові слова: АВТОМАТИЗОВАНИЙ ПЕРЕКЛАД, МУЛЬТИМЕДІЙНИЙ КОНТЕНТ, АВТОМАТИЧНЕ РОЗПІЗНАВАННЯ МОВЛЕННЯ, ОПТИЧНЕ РОЗПІЗНАВАННЯ СИМВОЛІВ, НЕЙРОМЕРЕЖЕВИЙ МАШИННИЙ ПЕРЕКЛАД, ВЕБ-СИСТЕМА, КЕШУВАННЯ, АВТЕНТИФІКАЦІЯ.

ABSTRACT

The thesis consists of 64 pages, includes 12 figures, 10 tables, 1 appendix, and 21 references.

The purpose of this work is to solve the applied problem of automated multimedia content translation by developing a web-based system with a unified processing pipeline for files of various formats.

During the development process, a client-server architecture with a REST-based application programming interface was implemented. The system employs automatic speech recognition methods based on a neural transformer model, optical character recognition, and neural machine translation. Data security is ensured through adaptive password hashing and authentication based on signed tokens. Data storage is implemented using a relational database with support for cascading operations and indexing.

The result of the work is a web-based system that automatically determines the type of the uploaded file and performs the complete processing cycle: text or speech recognition, translation into the selected language, and saving the result in the appropriate format. The system supports eight input file formats, implements a translation caching mechanism for repeated requests, and provides a personal operation history for each user. The functionality and reliability of the system have been confirmed through functional and non-functional testing.

The practical significance of the work lies in the development of a ready-to-use software product that eliminates the need to use multiple separate tools when working with different types of multimedia content and does not require technical expertise from the user.

Keywords: AUTOMATED TRANSLATION, MULTIMEDIA CONTENT, AUTOMATIC SPEECH RECOGNITION, OPTICAL CHARACTER RECOGNITION, NEURAL MACHINE TRANSLATION, WEB SYSTEM, CACHING, AUTHENTICATION.

ЗМІСТ

АНОТАЦІЯ.....	4
ABSTRACT	5
ВСТУП	8
1 ЗАДАЧА АВТОМАТИЗОВАНОГО ПЕРЕКЛАДУ МУЛЬТИМЕДІЙНОГО КОНТЕНТУ ТА ЗАСОБИ ЇЇ РОЗВ'ЯЗАННЯ	10
1.1 Постановка задачі	10
1.2 Аналіз існуючих рішень та обґрунтування підходу	15
1.3 Огляд методів обробки мультимедійного контенту.....	16
2 МЕТОДИ ТА АЛГОРИТМИ АВТОМАТИЗОВАНОГО ПЕРЕКЛАДУ МУЛЬТИМЕДІЙНОГО КОНТЕНТУ.....	21
2.1 Архітектурний підхід та патерни проєктування.....	21
2.2 Принципи організації моделі даних.....	24
2.3 Принципи організації програмного інтерфейсу	26
2.4 Алгоритм конвеєра обробки мультимедійних файлів	27
3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-СИСТЕМИ АВТОМАТИЗОВАНОГО ПЕРЕКЛАДУ МУЛЬТИМЕДІЙНОГО КОНТЕНТУ.....	35
3.1 Функціональна модель системи	35
3.2 Структура програмних компонентів та модель даних.....	38
3.3 Реалізація модулів конвеєра обробки мультимедіа.....	39
3.5 Реалізація інструментів адміністрування	42
4 РОЗГОРТАННЯ, ДЕМОНСТРАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ.....	44
4.1 Інсталяція та вимоги до обчислювальної техніки	44
4.2 Демонстрація функціоналу та сценарії роботи.....	46
4.3 Тестування системи та результати	57

ВИСНОВКИ.....	64
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	66
ДОДАТОК А ПРОГРАМНИЙ КОД СИСТЕМИ.....	68

ВСТУП

Стрімкий розвиток цифрових технологій та глобалізація інформаційного простору зумовили суттєве зростання обсягів мультимедійного контенту, що циркулює у мережі Інтернет. Щодня мільйони відеоматеріалів, аудіозаписів, зображень та документів різними мовами стають доступними для широкої аудиторії. Для фахівців, що працюють з іншомовними матеріалами — перекладачів, журналістів, науковців, студентів — виникає постійна потреба в інструментах автоматизованого перекладу різнотипного контенту.

Аналіз існуючих рішень показав, що розглянуті сервіси мають суттєві обмеження щодо підтримки окремих типів мультимедійного контенту. Більшість з них орієнтовані на один конкретний тип контенту або є платними з обмеженим функціоналом у безкоштовній версії. Традиційний підхід до перекладу мультимедіа є багатоетапним і трудомістким: для кожного типу файлу потрібен окремий інструмент, що вимагає технічних знань та значних часових витрат. Таким чином, актуальною є задача створення веб-системи, що забезпечує автоматизовану обробку та переклад мультимедійного контенту різних форматів у межах єдиного робочого процесу.

Метою дипломної роботи є розв'язання прикладної задачі автоматизованого перекладу мультимедійного контенту шляхом розробки веб-системи для обробки та перекладу мультимедійних файлів різних форматів.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати предметну область та сформулювати вимоги до системи;
- дослідити та порівняти існуючі рішення у сфері автоматизованого перекладу мультимедіа;
- обрати програмні засоби реалізації системи;
- спроектувати архітектуру системи та схему бази даних;
- реалізувати серверну частину з програмним інтерфейсом та системою автентифікації;

- реалізувати конвеєр обробки мультимедійних файлів різних форматів;
- розробити клієнтський інтерфейс;
- провести тестування системи та усунути виявлені недоліки.

Об'єктом дослідження є процес автоматизованого перекладу мультимедійного контенту різних форматів. Предметом дослідження є методи та засоби розробки веб-систем для автоматизованої обробки та перекладу мультимедійних файлів.

Практична цінність розробленої системи полягає в тому, що вона надає єдиний інструмент для автоматизованого перекладу мультимедійних матеріалів різних форматів. Система може використовуватись перекладачами та редакторами для прискорення робочого процесу, студентами та викладачами для опрацювання іншомовних навчальних матеріалів, журналістами для роботи з іноземними джерелами, а також фахівцями з локалізації відеоконтенту.

Дипломна робота складається з чотирьох розділів. У першому розділі сформульовано прикладну задачу, проведено аналіз існуючих рішень та огляд методів обробки мультимедійного контенту. У другому розділі обґрунтовано архітектурний підхід, описано патерни проектування, модель даних та алгоритм конвеєра обробки. У третьому розділі описано програмну реалізацію серверного модуля, підсистеми автентифікації, модулів конвеєра та клієнтської частини. У четвертому розділі наведено вимоги до розгортання, демонстрацію функціоналу та результати тестування системи.

1 ЗАДАЧА АВТОМАТИЗОВАНОГО ПЕРЕКЛАДУ МУЛЬТИМЕДІЙНОГО КОНТЕНТУ ТА ЗАСОБИ ЇЇ РОЗВ'ЯЗАННЯ

У розділі сформульовано прикладну задачу автоматизованого перекладу мультимедійного контенту та визначено вимоги до системи, що її розв'язує. Проведено аналіз існуючих рішень у сфері автоматизованого перекладу мультимедіа та огляд ключових методів обробки різних типів контенту: автоматичного розпізнавання мовлення, оптичного розпізнавання символів та нейромережевого машинного перекладу. На основі порівняльного аналізу обґрунтовано вибір програмних засобів для розв'язання задачі.

1.1 Постановка задачі

1.1.1 Опис прикладної задачі та цільова аудиторія

Прикладна задача полягає у забезпеченні автоматизованого перекладу мультимедійного контенту різних форматів — аудіофайлів, відеофайлів, зображень із текстом та файлів субтитрів — у межах єдиного робочого процесу без необхідності технічної підготовки з боку користувача.

Проблема, що вирішується, обумовлена фрагментованістю існуючих інструментів. Переклад різнотипного мультимедійного контенту наразі вимагає послідовного використання кількох спеціалізованих засобів: для перекладу відеолекції необхідно спочатку витягти аудіодоріжку, потім виконати розпізнавання мовлення окремим сервісом, після чого передати текст до перекладача. Переклад файлів субтитрів додатково ускладнюється необхідністю збереження часових міток, а розпізнавання тексту на зображеннях потребує встановлення та налаштування спеціалізованого програмного забезпечення. Такий підхід є трудомістким, вимагає технічних знань та обмежує доступність інструментів для широкого кола користувачів.

Таким чином, задача формулюється як: забезпечити уніфікований, технічно доступний спосіб автоматизованого перекладу мультимедійного контенту різних форматів з автоматичним визначенням типу файлу, мови оригіналу та збереженням відповідного формату результату.

Цільова аудиторія системи охоплює кілька груп користувачів. Перекладачі та редактори, що регулярно працюють з іншомовними матеріалами, потребують автоматизації рутинних операцій: замість ручного багатоетапного процесу вони мають отримувати готовий результат за одну дію. Студенти та викладачі потребують можливості швидко опрацьовувати іншомовні навчальні відеоматеріали без технічної підготовки. Журналісти потребують оперативного перекладу іноземних джерел у будь-якому форматі. Фахівці з локалізації потребують автоматизованого перекладу файлів субтитрів зі збереженням синхронізації. Спільною потребою всіх груп є єдиний інструмент, що усуває необхідність перемикання між різними сервісами та знижує технічний поріг входу.

1.1.2 Функціональні вимоги

Функціональні вимоги визначають, що система повинна вміти робити з точки зору користувача, незалежно від способу технічної реалізації.

Система повинна забезпечувати реєстрацію та автентифікацію користувачів з валідацією введених даних і захищеним зберіганням облікової інформації. Після входу користувач повинен мати доступ до перегляду власного профілю та журналу виконаних перекладів.

Центральною функцією системи є обробка та переклад файлів різних форматів. Система повинна приймати аудіофайли поширених форматів і повертати перекладений текст з автоматичним визначенням мови оригіналу. Для відеофайлів система повинна витягувати звуковий супровід та виконувати його транскрипцію з подальшим перекладом. Зображення, що містять текст, мають оброблятися шляхом розпізнавання і перекладу виявленого тексту. Файли субтитрів повинні перекладатися із суворим збереженням часових міток та форматування, щоб результат можна було використати безпосередньо у відеоплеєрі.

Система повинна автоматично визначати тип завантаженого файлу та застосовувати відповідний алгоритм обробки без додаткових дій з боку користувача. Цільова мова перекладу обирається з переліку підтримуваних мов. Після завершення обробки система повинна надавати посилання для завантаження результату та відображати метадані: визначену мову оригіналу, кількість оброблених фрагментів і час обробки.

Система повинна зберігати журнал перекладів кожного користувача та надавати можливість повторного завантаження вже перекладених файлів без повторної обробки. Для адміністратора повинні бути доступні інструменти керування обліковими записами та перегляду статистики системи.

1.1.3 Нефункціональні вимоги

Нефункціональні вимоги визначають якісні характеристики системи та обмеження на її функціонування.

Продуктивність. Відповіді на запити автентифікації та отримання довідникових даних повинні надходити протягом не більше 500 мілісекунд. Це значення відповідає порогу «миттєвої» реакції інтерфейсу за моделлю Дж. Нільсена [1], що забезпечує суб'єктивне відчуття плавності роботи. Обробка аудіофайлів тривалістю до 5 хвилин повинна завершуватися не більш ніж за 3 хвилини. Це обмеження визначено виходячи з практичних замірів продуктивності моделі Whisper base на типовому серверному обладнанні: швидкість транскрипції моделі базового розміру складає приблизно $1,5\text{--}2\times$ реального часу аудіо [2], отже для 5-хвилинного файлу потрібно близько 2,5–3,3 хвилини. Переклад файлу субтитрів із 50 фрагментами повинен завершуватися не більш ніж за 30 секунд: цей показник враховує середній розмір фрагмента субтитрів (~10 слів) та типову затримку API перекладу (~100–200 мс на запит) при послідовній обробці пакетами.

Безпека. Паролі користувачів не повинні зберігатися у відкритому вигляді. Доступ до захищених функцій системи повинен надаватися лише після підтвердження особи користувача. Усі запити до бази даних повинні бути захищені

від ін'єкцій, а завантажені файли — зберігатися у спосіб, що унеможливорює несанкціонований доступ.

Надійність. Відсутність або тимчасова недоступність одного із зовнішніх компонентів не повинна призводити до повної непрацездатності системи. Усі помилкові ситуації повинні оброблятися коректно з поверненням зрозумілих повідомлень.

Зручність використання. Веб-інтерфейс повинен підтримувати зручне завантаження файлів та динамічне оновлення результатів без перезавантаження сторінки. Система повинна коректно працювати у сучасних браузерях без встановлення додаткових компонентів. Конфігурація системи повинна зберігатися окремо від програмного коду.

1.1.4 Діаграма варіантів використання

Діаграма варіантів використання відображає взаємодію між акторами системи та функціональними сценаріями. Система передбачає три типи акторів: незареєстрований відвідувач, зареєстрований користувач та адміністратор.

Незареєстрований відвідувач має доступ лише до сценаріїв реєстрації нового облікового запису та входу до системи. Це рішення забезпечує персоналізацію сервісу та можливість ведення індивідуальної історії перекладів.

Зареєстрований користувач отримує доступ до повного функціоналу системи. Центральним сценарієм є завантаження файлу для перекладу, що залежно від формату файлу розгалужується на три підсценарії: переклад аудіо та відео, переклад зображення та переклад субтитрів. Крім того, користувач може переглядати власний журнал перекладів та повторно завантажувати раніше отримані результати.

Адміністратор взаємодіє з системою через інтерфейс командного рядка та має доступ до сценаріїв керування обліковими записами: перегляд списку користувачів, перегляд статистики, скидання пароля та видалення облікового запису. (Рис.1.)

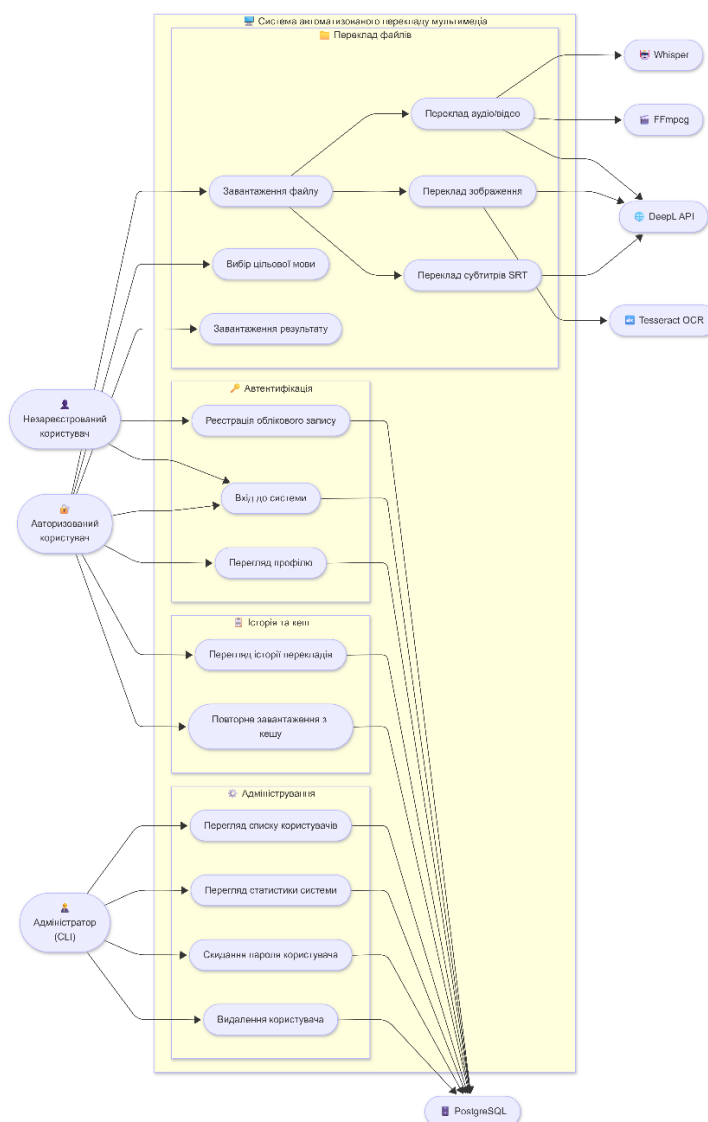


Рис.1. Діаграма варіантів використання

Система передбачає три типи акторів: незареєстрований користувач, авторизований користувач та адміністратор.

Незареєстрований користувач має доступ лише до двох сценаріїв: реєстрація нового облікового запису та вхід до системи. Це архітектурне рішення забезпечує персоналізацію сервісу та можливість збереження індивідуальної історії перекладів.

Авторизований користувач отримує доступ до повного функціоналу системи. Центральним сценарієм є завантаження файлу для перекладу, який є точкою входу для трьох підсценаріїв відповідно до типу файлу: переклад аудіо та відео, переклад зображення та переклад субтитрів SRT. Це відношення типу «include» - залежно від

формату файлу система автоматично обирає відповідний конвертер обробки. Авторизований суб'єкт доступу також може переглядати історію своїх перекладів та повторно завантажувати вже перекладені файли з кешу без повторної обробки.

Адміністратор взаємодіє з системою через командний рядок і має доступ до чотирьох сценаріїв: перегляд списку облікових записів, перегляд статистики системи, скидання пароля та видалення облікового запису.

1.2 Аналіз існуючих рішень та обґрунтування підходу

сфері автоматизованого перекладу мультимедійного контенту. Розглянуто як комерційні онлайн-сервіси, так і програмне забезпечення з відкритим кодом.

Google Translate є найбільш відомим сервісом машинного перекладу. Він підтримує переклад тексту та зображень через мобільний застосунок, однак не надає можливості перекладу аудіо- та відеофайлів або файлів субтитрів. Веб-версія не зберігає журнал перекладів файлів, а якість перекладу для низки мовних пар поступається спеціалізованим нейромережевим системам.

DeepL є одним з найякісніших сервісів нейромережевого перекладу. Веб-версія підтримує переклад текстових документів, однак не підтримує аудіо, відео та зображення. Безкоштовна версія має обмеження на обсяг тексту та кількість документів на місяць. Відсутня можливість збереження та повторного використання результатів.

Otter.ai — сервіс транскрипції та перекладу аудіо. Підтримує завантаження аудіофайлів та відеозаписів, однак орієнтований переважно на англійську мову та має суттєві обмеження у безкоштовній версії. Переклад зображень та субтитрів не підтримується.

Subtitle Edit — безкоштовна програма для роботи з субтитрами. Підтримує переклад SRT-файлів через інтеграцію з онлайн-перекладачами, однак є настільним застосунком, що вимагає встановлення. Обробка аудіо та зображень не підтримується.

Happy Scribe та Sonix — комерційні сервіси транскрипції та перекладу відео. Забезпечують якісну обробку аудіо та відео, однак є повністю платними та не підтримують переклад зображень.

Таблиця 1.1 — Порівняльний аналіз існуючих рішень

Рішення	Аудіо/Відео	Зображення	Субтитри SRT	Авторизація	Історія	Безкоштовно
Google Translate	×	Частково	×	✓	×	✓
DeepL (веб)	×	×	×	✓	×	Частково
Otter.ai	Частково	×	×	✓	✓	Частково
Subtitle Edit	×	×	✓	×	×	✓
Happy Scribe	✓	×	✓	✓	✓	×
Розроблена система	✓	✓	✓	✓	✓	✓

З таблиці 1.1 видно, що жоден з розглянутих сервісів не забезпечує одночасної підтримки всіх трьох типів мультимедійного контенту в рамках єдиної безкоштовної платформи зі збереженням журналу перекладів. Кожне з розглянутих рішень орієнтоване або на один тип контенту, або є повністю платним, або не зберігає результати для подальшого використання. Цей висновок підтверджує доцільність розробки спеціалізованої системи, що поєднує обробку різнотипного мультимедійного контенту в єдиному інтерфейсі.

1.3 Огляд методів обробки мультимедійного контенту

Для розв'язання поставленої задачі необхідно застосувати три ключові методи обробки даних: автоматичне розпізнавання мовлення для аудіо- та відеофайлів, оптичне розпізнавання символів для зображень та нейромережевий машинний переклад для всіх типів контенту. У цьому підрозділі розглянуто

принципи роботи кожного методу та проведено порівняльний аналіз доступних програмних засобів реалізації з обґрунтуванням.

1.3.1 Автоматичне розпізнавання мовлення

Автоматичне розпізнавання мовлення (Automatic Speech Recognition, ASR) — це галузь штучного інтелекту, що займається перетворенням усного мовлення у текстову форму. Сучасні системи засновані на нейромережевих архітектурах трансформерного типу та досягли рівня точності, що наближається до людського сприйняття для більшості мов і умов запису.

Принцип роботи трансформерних ASR-моделей полягає у перетворенні акустичного сигналу на послідовність мел-спектрограм, які потім кодуються нейромережею в приховане представлення. Декодер на основі механізму уваги генерує текстову послідовність, враховуючи весь контекст звукового фрагменту, що дозволяє коректно обробляти омоніми та відновлювати слова зі спотвореним звучанням [3].

Для задачі обробки аудіо- та відеофайлів розглянуто кілька програмних засобів. Хмарні сервіси Google Speech-to-Text та Azure Speech забезпечують дуже високу точність розпізнавання та підтримують понад 100 мов, однак вимагають передачі аудіоданих на зовнішні сервери та потребують оплати за кожну хвилину обробки. Відкриті рішення для локального запуску — Mozilla DeepSpeech та Vosk — підтримують значно менше мов і не виконують автоматичного визначення мови оригіналу. Модель Whisper від OpenAI поєднує переваги обох підходів: вона є відкритою, підтримує 99 мов з автоматичним визначенням мови та може виконуватись локально без передачі даних назовні [4]. Модель базового розміру забезпечує прийнятний баланс між точністю та швидкістю обробки, що робить її оптимальним вибором для задач системи.

Таблиця 1.2 — Порівняльний аналіз засобів автоматичного розпізнавання мовлення

Рішення	Мов	Автовизначення мови	Відкритий код	Локальний запуск	Точність (WER)
OpenAI Whisper (base)	99	✓	✓	✓	~15%
Google Speech-to-Text	125	✓	✗	✗	~10%
Azure Speech	100+	✓	✗	✗	~10%
Mozilla DeepSpeech	10	✗	✓	✓	~20%
Vosk	20	✗	✓	✓	~20%

Висновок: для розв'язання задачі обрано модель Whisper (base) як відкрите рішення з найширшою підтримкою мов серед локально розгорнутих систем та автоматичним визначенням мови оригіналу.

1.3.2 Оптичне розпізнавання символів

Оптичне розпізнавання символів (Optical Character Recognition, OCR) — це технологія перетворення зображень, що містять текст, у машинозчитуваний формат. Сучасні системи використовують нейромережеві підходи на основі рекурентних мереж та трансформерних архітектур. Принцип роботи більшості OCR-систем включає два етапи: сегментацію зображення для виділення рядків і символів та класифікацію виділених фрагментів нейромережею. Якість розпізнавання суттєво залежить від якості вхідного зображення: роздільної здатності, контрастності та рівномірності освітлення [5].

Для задачі розпізнавання тексту на зображеннях розглянуто кілька програмних засобів. Хмарні сервіси Google Vision API та Azure Computer Vision забезпечують найвищу точність розпізнавання, проте є платними та вимагають передачі зображень на зовнішні сервери. Серед відкритих рішень Tesseract є одним з найбільш зрілих рушіїв: починаючи з версії 4.0 він використовує нейромережевий рушій на основі LSTM, підтримує понад 100 мов і має розвинену документацію та широку спільноту розробників [6]. EasyOCR та PaddleOCR також є відкритими і

забезпечують порівнянню якості розпізнавання, однак є менш зрілими проектами з меншим обсягом документації.

Таблиця 1.3 — Порівняльний аналіз засобів оптичного розпізнавання символів

Рішення	Підтримка мов	Відкритий код	Нейромережевий рушій
Tesseract 5.x	100+	✓	LSTM
Google Vision API	50+	✗	✓
Azure Computer Vision	50+	✗	✓
EasyOCR	80+	✓	✓
PaddleOCR	80+	✓	✓

Висновок: для розв'язання задачі обрано Tesseract 5.x з огляду на зрілість рішення, широку підтримку мов, відкритий код та можливість локального розгортання без передачі даних назовні.

1.3.3 Нейромережевий машинний переклад

Машинний переклад — це галузь обчислювальної лінгвістики, що займається автоматичним перекладом тексту між природними мовами. Сучасні нейромережеві системи перекладу засновані на трансформерній архітектурі з механізмом уваги (attention mechanism), що дозволяє враховувати контекст усього речення при перекладі кожного слова. Завдяки цьому якість перекладу довгих речень та збереження змістовного контексту суттєво покращились порівняно зі статистичними підходами [7].

Для задачі перекладу розпізнаного тексту розглянуто кілька програмних інтерфейсів. DeepL демонструє найвищу якість перекладу за результатами незалежних порівняльних досліджень, особливо для європейських мов, підтримує понад 30 мов і надає безкоштовний план з лімітом 500 тисяч символів на місяць [8]. Google Translate підтримує більшу кількість мов, однак якість перекладу для більшості мовних пар є нижчою порівняно з DeepL. Microsoft Translator пропонує

ширший безкоштовний ліміт, проте також поступається DeepL за якістю. LibreTranslate є повністю відкритим рішенням для самостійного розгортання, однак його якість перекладу суттєво поступається комерційним сервісам.

Таблиця 1.4 — Порівняльний аналіз сервісів машинного перекладу

Сервіс	Підтримка мов	Програмний інтерфейс	Безкоштовний план	Якість
DeepL	30+	✓	500 тис. символів/міс	Найвища
Google Translate	130+	✓	500 тис. символів/міс	Висока
Microsoft Translator	100+	✓	2 млн символів/міс	Висока
LibreTranslate	30+	✓	Необмежено (self-hosted)	Середня

Висновок: для розв'язання задачі обрано DeepL API як сервіс з найвищою якістю перекладу серед розглянутих варіантів з безкоштовним планом, що покриває потреби системи.

2 МЕТОДИ ТА АЛГОРИТМИ АВТОМАТИЗОВАНОГО ПЕРЕКЛАДУ МУЛЬТИМЕДІЙНОГО КОНТЕНТУ

У розділі обґрунтовано архітектурний підхід та патерни проєктування, обрані для розв'язання задачі автоматизованого перекладу мультимедійного контенту. Описано принципи організації моделі даних та програмного інтерфейсу системи. Детально розглянуто алгоритм конвеєра обробки мультимедійних файлів та алгоритми кожної з гілок обробки — аудіо/відео, зображень та субтитрів.

2.1 Архітектурний підхід та патерни проєктування

2.1.1 Обґрунтування архітектурного підходу

Для розв'язання задачі автоматизованого перекладу мультимедійного контенту обрано клієнт-серверну архітектуру з програмним інтерфейсом на основі принципів REST. Таблиця 2.1 узагальнює альтернативні архітектурні підходи та обґрунтовує зроблений вибір.

Таблиця 2.1 — Порівняльний аналіз архітектурних підходів

Підхід	Ресурсомісткі операції на клієнті	Приховування ключів API	Персональний журнал	Доступ з браузера без встановлення	Рішення
Клієнт-серверна (REST)	×	✓	✓	✓	Обрано
Настільний застосунок	✓	×	Локально	×	Відхилено
Безсерверна (serverless)	×	✓	✓	✓	Ускладнює стан
Однорангова (P2P)	✓	×	×	×	Відхилено

Клієнт-серверна архітектура задовольняє всі чотири ключові вимоги задачі. По-перше, обробка мультимедійних файлів є ресурсомісткою операцією —

розміщення її на сервері дозволяє виконувати незалежно від обчислювальних можливостей пристрою користувача. По-друге, інтеграція з зовнішніми сервісами перекладу вимагає приховування ключів доступу від клієнта, що можливо лише при серверній централізації. По-третє, ведення персонального журналу перекладів потребує серверного сховища даних. По-четверте, клієнт-серверна модель забезпечує доступ через будь-який сучасний браузер без встановлення додаткового програмного забезпечення.

Сервер є stateless з точки зору бізнес-логіки: стан зберігається виключно в реляційній базі даних, що спрощує масштабування та підвищує надійність. Взаємодія між клієнтом та сервером здійснюється через стандартизований інтерфейс у форматі JSON або multipart/form-data для завантаження файлів.



Рис. 2.1 Загальна архітектура системи

Чітке розділення між клієнтом і сервером забезпечує незалежність їх розробки та дозволяє надалі замінити клієнтський інтерфейс, не змінюючи серверну частину. Взаємодія між клієнтом та сервером здійснюється через стандартизований програмний інтерфейс у форматі JSON або multipart/form-data для завантаження файлів.

2.1.2 Патерни проєктування та їх роль у розв'язанні задачі

Для вирішення конкретних архітектурних проблем системи застосовано низку патернів проєктування. Таблиця 2.2 узагальнює кожен патерн, задачу, яку він вирішує, та результат застосування.

Таблиця 2.2 — Застосовані патерни проєктування

Патерн	Задача, що вирішується	Результат застосування
Стратегія	Обробка різнотипних файлів одним центральним алгоритмом	Вибір гілки алгоритму за типом файлу під час виконання; додавання нових типів без зміни основної логіки
Впровадження залежностей	Однорідна перевірка автентифікації для захищених запитів	Функція перевірки токена декларується як залежність; відсутність дублювання перевірок
Репозиторій	Ізоляція логіки зберігання від бізнес-логіки	Всі операції з БД виокремлені в окремий шар; можливість заміни сховища незалежно від логіки обробки
Відкладене завантаження	Нейромережева модель потребує тривалої ініціалізації та значної пам'яті	Модель завантажується лише при першому зверненні; прискорення запуску сервера
Кешування на стороні застосунку	Уникнення повторних витратних операцій транскрипції та перекладу	Перевірка журналу перед кожним перекладом; повернення готового результату без зовнішніх запитів
Пул з'єднань	Ефективне управління підключеннями до БД при конкурентній обробці	З'єднання ініціалізуються заздалегідь і повертаються до пулу після використання

Ключовим для розв'язання задачі є патерн «Стратегія»: оскільки кожен тип мультимедійного контенту вимагає принципово різного алгоритму підготовки тексту до перекладу, необхідний механізм вибору алгоритму під час виконання програми. Центральна функція обробки визначає тип файлу та делегує виконання відповідній гілці, що дозволяє розширювати систему без зміни її основної логіки.

2.2 Принципи організації моделі даних

2.2.1 Обґрунтування вибору реляційної моделі та СКБД

Для зберігання даних системи обрано реляційну модель, оскільки вона найкраще відповідає природі оброблюваних даних: записи журналу перекладів мають фіксовану структуру, між обліковими записами та журналами існують чіткі зв'язки, а запити переважно вибіркові з фільтрацією та сортуванням.

Таблиця 2.3 — Порівняльний аналіз систем управління базою даних

Критерій	PostgreSQL	MySQL	SQLite
Підтримка складних запитів	Повна	Часткова	Обмежена
Каскадне видалення	Так	Так	Так
Пул з'єднань	Так	Так	Ні
Конкурентний запис	Версіонування рядків (MVCC)	Блокування	Блокування
Відповідність стандарту SQL	Повна	Часткова	Часткова

SQLite відхилено через обмеження на конкурентний запис, що є критичним для веб-застосунку з кількома одночасними користувачами. PostgreSQL обрано над MySQL завдяки повнішій відповідності стандарту SQL, механізму конкурентного доступу на основі версіонування рядків (MVCC) та повноцінній підтримці каскадних операцій.

2.2.2 Концептуальна модель даних

Модель даних системи побудована навколо центральної сутності — облікового запису користувача. З нею пов'язані три залежні сутності: журнал перекладів, активні сесії та токени відновлення пароля. Зв'язки між сутностями реалізовано через зовнішні ключі з каскадним видаленням — при видаленні облікового запису автоматично видаляються всі пов'язані записи, що забезпечує цілісність даних без додаткової логіки в застосунку.

Журнал перекладів виконує подвійну функцію. По-перше, він зберігає повну історію перекладів для відображення користувачу. По-друге, слугує механізмом кешування: перед кожним новим перекладом система шукає в журналі запис з ідентичним поєднанням ідентифікатора користувача, імені файлу та цільової мови. Поєднання цих двох функцій в одній таблиці дозволяє уникнути дублювання даних та спростити архітектуру.

Таблиця 2.4 — Сутності моделі даних та їх призначення

Сутність	Призначення	Зв'язок з користувачем
Обліковий запис (users)	Центральна сутність; ідентифікація та автентифікація	—
Журнал перекладів (translations)	Зберігання результатів; механізм кешування	Зовнішній ключ, каскадне видалення
Активні сесії (sessions)	Відстеження виданих токенів доступу	Зовнішній ключ, каскадне видалення
Токени відновлення (reset_tokens)	Підтримка процедури скидання пароля	Зовнішній ключ, каскадне видалення

Нижче наведена діаграма «сутність-зв'язок»:

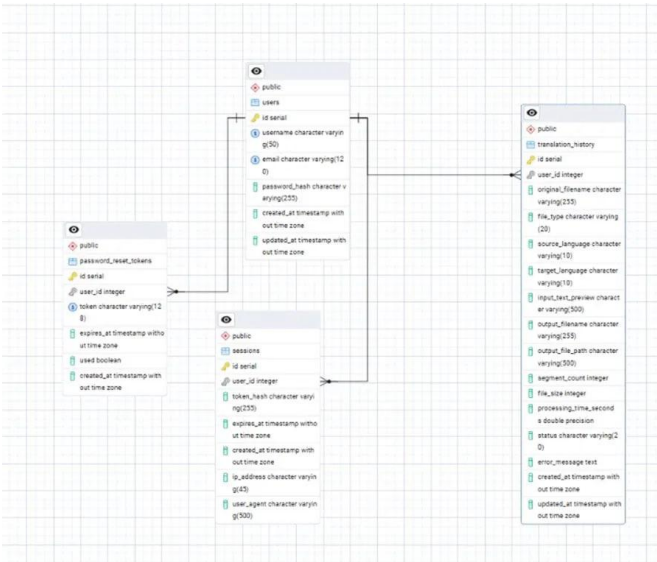


Рисунок 2.2 — Діаграма «сутність-зв'язок»

2.2.3 Підходи до оптимізації запитів

Продуктивність системи залежить від швидкості двох ключових операцій: пошуку кешованого результату та вибірки журналу для відображення. Індукування вирішує задачу прискорення цих операцій за рахунок попередньо впорядкованих структур доступу.

Таблиця 2.5 — Індеси бази даних та їх обґрунтування

Поле (таблиця)	Тип індексу	Операція, що прискорюється
login, email (users)	B-tree, унікальний	Пошук облікового запису при автентифікації
user_id (translations)	B-tree	Вибір журналу конкретного користувача
created_at DESC (translations)	B-tree	Отримання найновіших записів журналу
status (translations)	B-tree	Фільтрація журналу за результатом обробки
(user_id, filename, target_lang) (translations)	B-tree, складений	Пошук кешованого результату (виконується при кожному запиті перекладу)

Складений індекс за триплетом (user_id, filename, target_lang) є критично важливим, оскільки відповідний запит виконується при кожному виклику конвеєра обробки. Без цього індексу кожна перевірка кешу потребувала б повного перебору таблиці журналу.

2.3 Принципи організації програмного інтерфейсу

Програмний інтерфейс системи організовано відповідно до принципів REST: кожен ресурс адресується окремим шляхом, HTTP-метод відображає тип операції, а відповіді містять стандартизовані коди стану. Детальний перелік кінцевих точок та їх технічні характеристики наведено у розділі 3.

Ендпоінти згруповано в три функціональні групи: операції автентифікації (реєстрація, вхід, профіль), операції з перекладами (завантаження файлу, перегляд журналу, отримання результату) та допоміжні операції (перелік мов, перевірка стану системи).

Таблиця 2.6 — Функціональні групи програмного інтерфейсу

Група	Кількість ендпоінтів	Захист токеном	Призначення
Автентифікація	3	Частково	Реєстрація, вхід, отримання профілю
Переклади	3	Так / Ні	Завантаження файлу, журнал, отримання файлу
Допоміжні	2	Ні	Довідник мов, стан компонентів

Для підтвердження особи користувача застосовано підхід на основі токенів без збереження стану сесії на сервері (stateless JWT). Після успішного входу сервер видає підписаний токен, що містить ідентифікатор та логін користувача і має обмежений термін дії. Клієнт передає токен у заголовку кожного захищеного запиту, що дозволяє серверу перевіряти особу користувача без звернення до бази даних при кожному запиті.

Розмежування між захищеними та публічними ендпоінтами обумовлено логікою системи: операції з перекладами є персональними і вимагають ідентифікації, тоді як завантаження готового файлу результату є публічним, оскільки файли зберігаються з невгадуваними унікальними ідентифікаторами.

2.4 Алгоритм конвеєра обробки мультимедійних файлів

2.4.1 Загальна структура алгоритму

Конвеєр обробки є центральним алгоритмом системи. Він приймає завантажений файл та код цільової мови, виконує повний цикл перетворення та повертає перекладений результат у форматі, що відповідає типу вхідного файлу. Алгоритм складається з чотирьох послідовних фаз.

Фаза 1. Перевірка кешу. До запуску будь-яких витратних операцій алгоритм виконує пошук у журналі перекладів за трьома параметрами: ідентифікатор поточного користувача, оригінальне ім'я файлу та цільова мова. Якщо відповідний запис знайдено і файл результату існує на диску, алгоритм завершується на цій фазі — повторної обробки не відбувається.

Фаза 2. Визначення типу файлу та маршрутизація. Тип файлу визначається за розширенням та зіставляється з таблицею підтримуваних форматів. За результатом виконання системою патерну «Стратегія» управління передається відповідній гілці алгоритму. При невідомому розширенні алгоритм завершується з повідомленням про помилку.

Фаза 3. Обробка за типом файлу та переклад. Кожна гілка виконує специфічну для свого типу підготовку тексту, після чого передає отриманий текст до служби перекладу.

Фаза 4. Збереження результату та метаданих. Незалежно від гілки обробки, після отримання перекладеного результату алгоритм зберігає файл з унікальним ідентифікатором та записує метадані операції до журналу: час обробки, кількість оброблених фрагментів, визначена мова оригіналу, статус виконання.

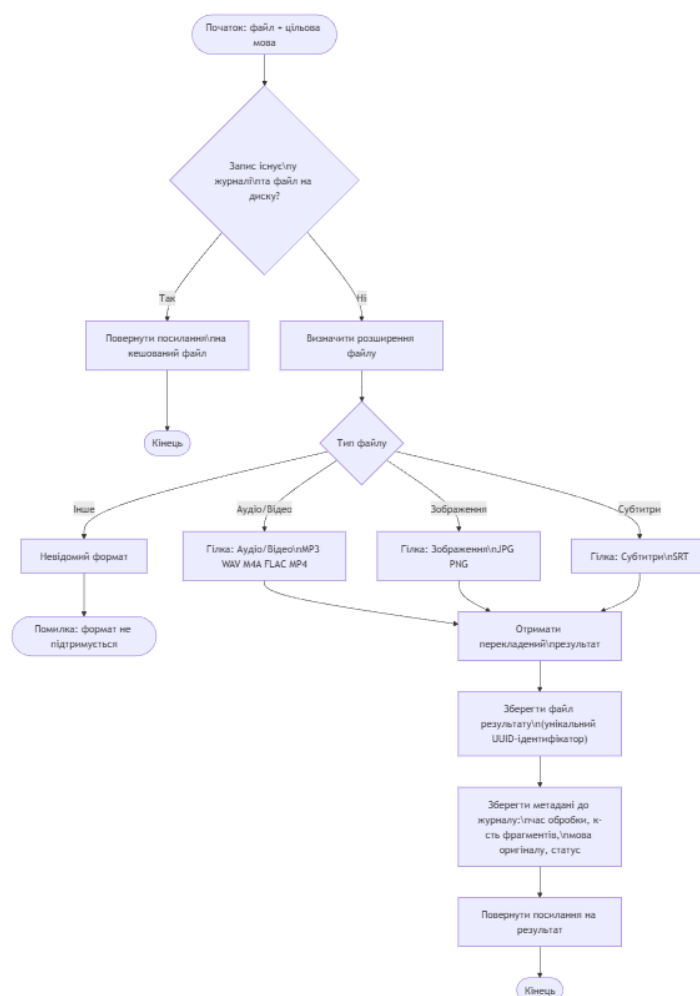


Рисунок 2.3 — UML-діаграма діяльності конвеєра обробки мультимедійних файлів

Таблиця 2.7 — Підтримувані формати та відповідні гілки алгоритму

Гілка алгоритму	Вхідні формати	Метод отримання тексту	Формат результату
Аудіо/Відео	MP3, WAV, M4A, FLAC, MP4	Транскрипція (ASR)	TXT
Зображення	JPG, PNG	Оптичне розпізнавання (OCR)	TXT
Субтитри	SRT	Синтаксичний розбір сегментів	SRT (зі збереженням тайм-кодів)

З таблиці 2.7 видно, що лише гілка обробки субтитрів повертає структурований вихідний формат зі збереженням тайм-кодів; гілки аудіо/відео та зображень повертають плаский текстовий файл.

2.4.2 Алгоритм обробки аудіо та відеофайлів

Гілка обробки аудіо та відео включає дві послідовні операції для відеофайлів та одну для аудіофайлів.

Крок 1 (лише для відео). Витягування звукової доріжки. Інструмент обробки мультимедіа запускається як зовнішній процес з параметрами вхідного відеофайлу та шляху до тимчасового аудіофайлу. Встановлюється часовий ліміт для запобігання зависанню при обробці пошкоджених файлів. Після завершення транскрипції тимчасовий файл видаляється.

Крок 2. Транскрипція мовлення. Нейромережева модель ASR перетворює аудіосигнал у текст. Модель автоматично визначає мову аудіо без потреби у попередньому налаштуванні. Результатом є розпізнаний текст та код визначеної мови оригіналу.

Крок 3. Переклад. Розпізнаний текст передається до служби перекладу. Результат зберігається як текстовий файл.

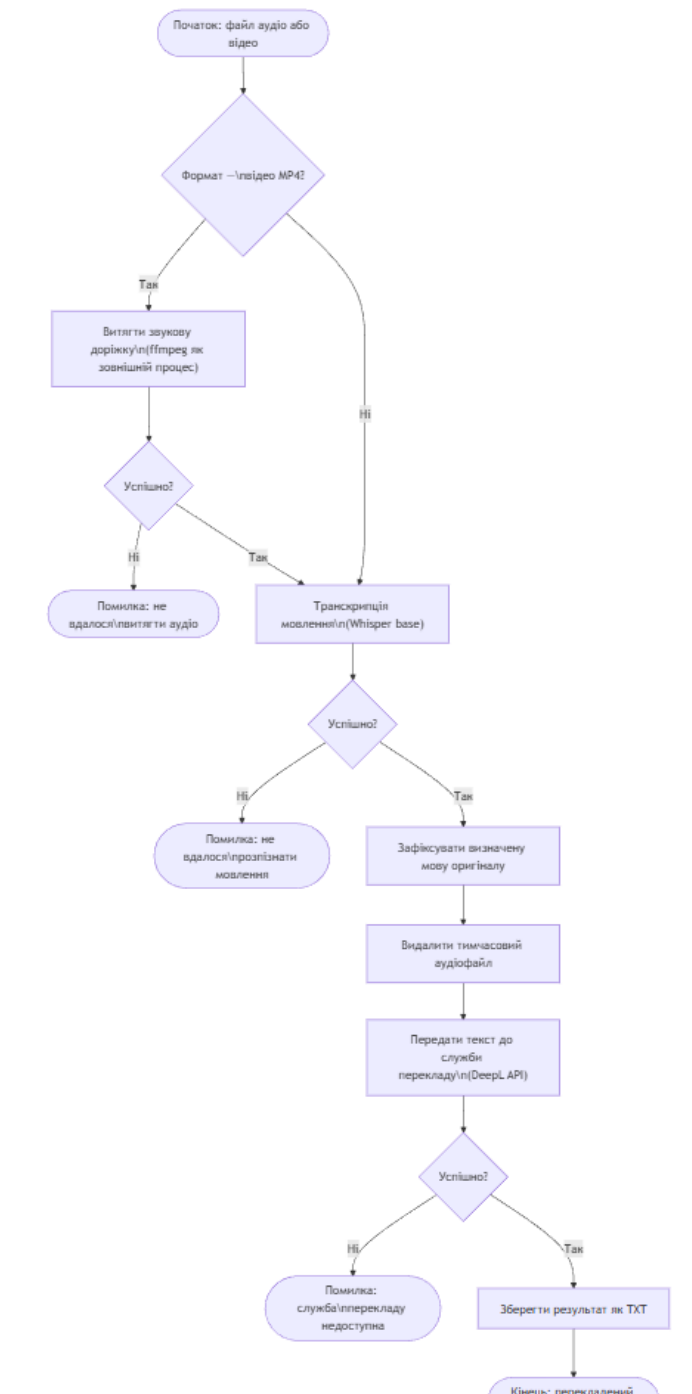


Рисунок 2.4 — UML-діаграма діяльності алгоритму обробки аудіо та відеофайлів

На рисунку 2.4 показано розгалуження між обробкою відеофайлу (з кроком витягування аудіодоріжки) та аудіофайлу (безпосередня транскрипція), а також точки виходу з помилкою для кожного з можливих збоїв.

2.4.3 Алгоритм обробки зображень

Гілка обробки зображень перед виконанням розпізнавання перевіряє доступність рушія OCR. Відсутність компонента обробляється явно: користувач отримує зрозуміле повідомлення замість технічної помилки виконання.

Алгоритм складається з трьох кроків

Крок 1. Перевірка наявності OCR-рушія. Якщо рушій недоступний, алгоритм завершується з діагностичним повідомленням.

Крок 2. Розпізнавання тексту. Зображення завантажується та передається до рушія OCR. Результатом є виявлений текст. Якість розпізнавання залежить від характеристик вхідного зображення (роздільна здатність, контрастність, рівномірність освітлення) і є властивістю вхідних даних, а не алгоритму.

Крок 3. Переклад. Розпізнаний текст передається до служби перекладу. Результат зберігається як текстовий файл.

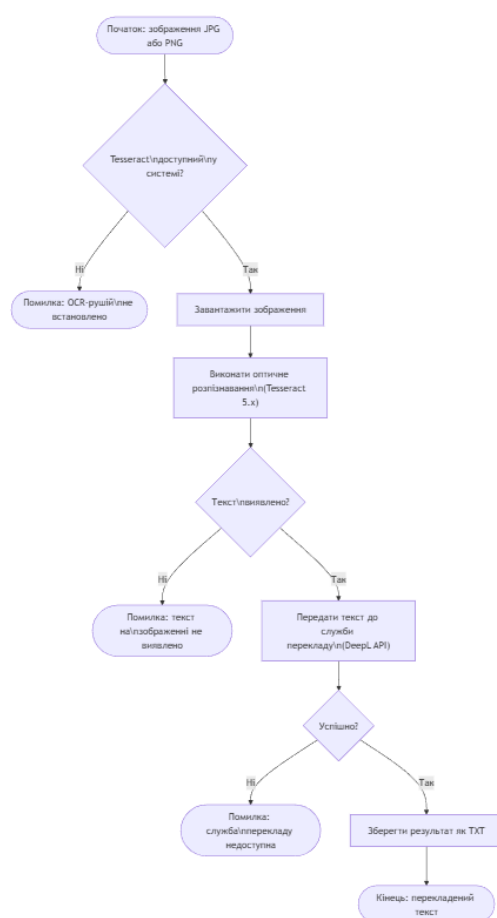


Рисунок 2.5 — UML-діаграма діяльності алгоритму обробки зображень

На рисунку 2.5 відображено початкову перевірку наявності OCR-рушія як обов'язкову умову, без виконання якої подальша обробка неможлива, а також залежність якості результату від характеристик вхідного зображення.

2.4.4 Алгоритм обробки файлів субтитрів

Гілка обробки файлів субтитрів є найбільш складною з огляду на вимогу збереження структури вхідних даних. Формат SRT передбачає нумеровані сегменти з тайм-кодами початку та кінця. Ці тайм-коди визначають синхронізацію субтитрів з відеорядом і не повинні змінюватись у процесі перекладу.

Алгоритм складається з трьох кроків.

Крок 1. Синтаксичний розбір файлу. Файл розбирається на окремі сегменти. Для кожного сегмента виділяються три компоненти: порядковий номер, рядок тайм-кодів та текстовий вміст. Підтримуються багаторядкові субтитри.

Крок 2. Переклад сегментів. Текстовий вміст кожного сегмента окремо передається до служби перекладу. Ізольована передача кожного сегмента забезпечує коректний переклад у межах природного контексту субтитру. Порядкові номери та тайм-коди не передаються до служби перекладу і залишаються незмінними.

Крок 3. Збирання результату. Файл збирається у стандартному форматі SRT: для кожного сегмента виводиться оригінальний порядковий номер, оригінальний рядок тайм-кодів, перекладений текст та порожній рядок-роздільник.

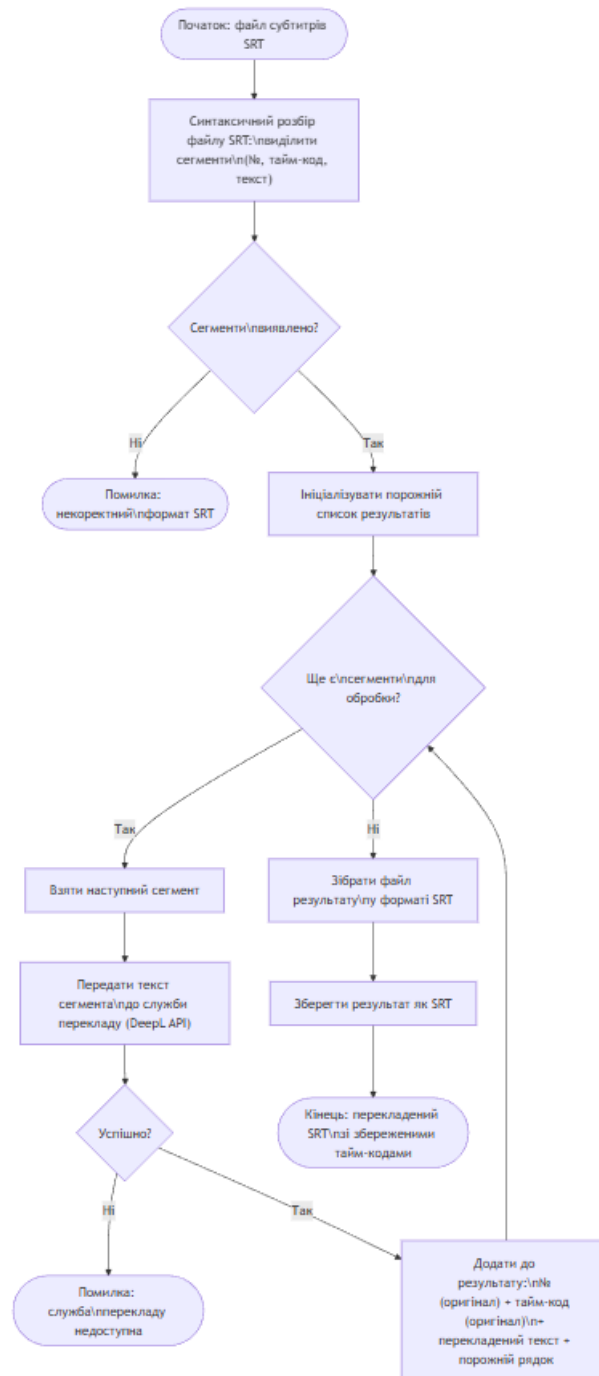


Рисунок 2.6 — UML-діаграма діяльності алгоритму обробки файлів субтитрів

На рисунку 2.6 відображено цикл посегментної обробки та принцип незмінності тайм-кодів: рядки часових міток проходять крізь алгоритм без передачі до служби перекладу і з'являються у вихідному файлі незмінними.

Таблиця 2.8 — Порівняльна характеристика алгоритмів гілок обробки

Характеристика	Аудіо/Відео	Зображення	Субтитри
Кількість кроків	2–3	3	3
Зовнішній компонент	ASR-модель	OCR-рушій	—
Залежність від якості вхідних даних	Якість звуку	Якість зображення	Коректний формат SRT
Збереження структури вхідного формату	Ні (текстовий вивід)	Ні (текстовий вивід)	Так (тайм-коди)
Кількість запитів до служби перекладу	1	1	N (по одному на сегмент)

З таблиці 2.8 видно, що гілка субтитрів є єдиною, де кількість запитів до служби перекладу залежить від вмісту вхідного файлу і може суттєво впливати на загальний час обробки при великій кількості сегментів.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-СИСТЕМИ АВТОМАТИЗОВАНОГО ПЕРЕКЛАДУ МУЛЬТИМЕДІЙНОГО КОНТЕНТУ

У розділі описано програмну реалізацію веб-системи автоматизованого перекладу мультимедійного контенту. У підрозділі 3.1 наведено функціональну модель системи у вигляді діаграми варіантів використання. У підрозділі 3.2 описано структуру серверного модуля, організацію програмних компонентів та модель даних. У підрозділі 3.3 описано реалізацію програмного інтерфейсу та підсистеми автентифікації. У підрозділі 3.4 описано програмні модулі конвеєра обробки мультимедійних файлів та схему їх взаємодії. У підрозділі 3.5 описано структуру клієнтської частини та інструменти адміністрування.

3.1 Функціональна модель системи

Функціональна модель системи описує взаємодію між акторами та функціональними сценаріями без прив'язки до деталей реалізації. Вона є основою для визначення меж системи та переліку реалізованих можливостей.

Система передбачає три типи акторів. **Неzareєстрований відвідувач** — будь-який користувач, що відкрив веб-інтерфейс без активної сесії. Він має доступ лише до публічних сценаріїв: реєстрації нового облікового запису та входу до системи. **Zareєстрований користувач** — ідентифікований учасник з активним токеном сесії. Він має доступ до повного функціоналу перекладу та персонального журналу. **Адміністратор** — технічний оператор системи, що взаємодіє з нею через інтерфейс командного рядка поза межами веб-інтерфейсу.

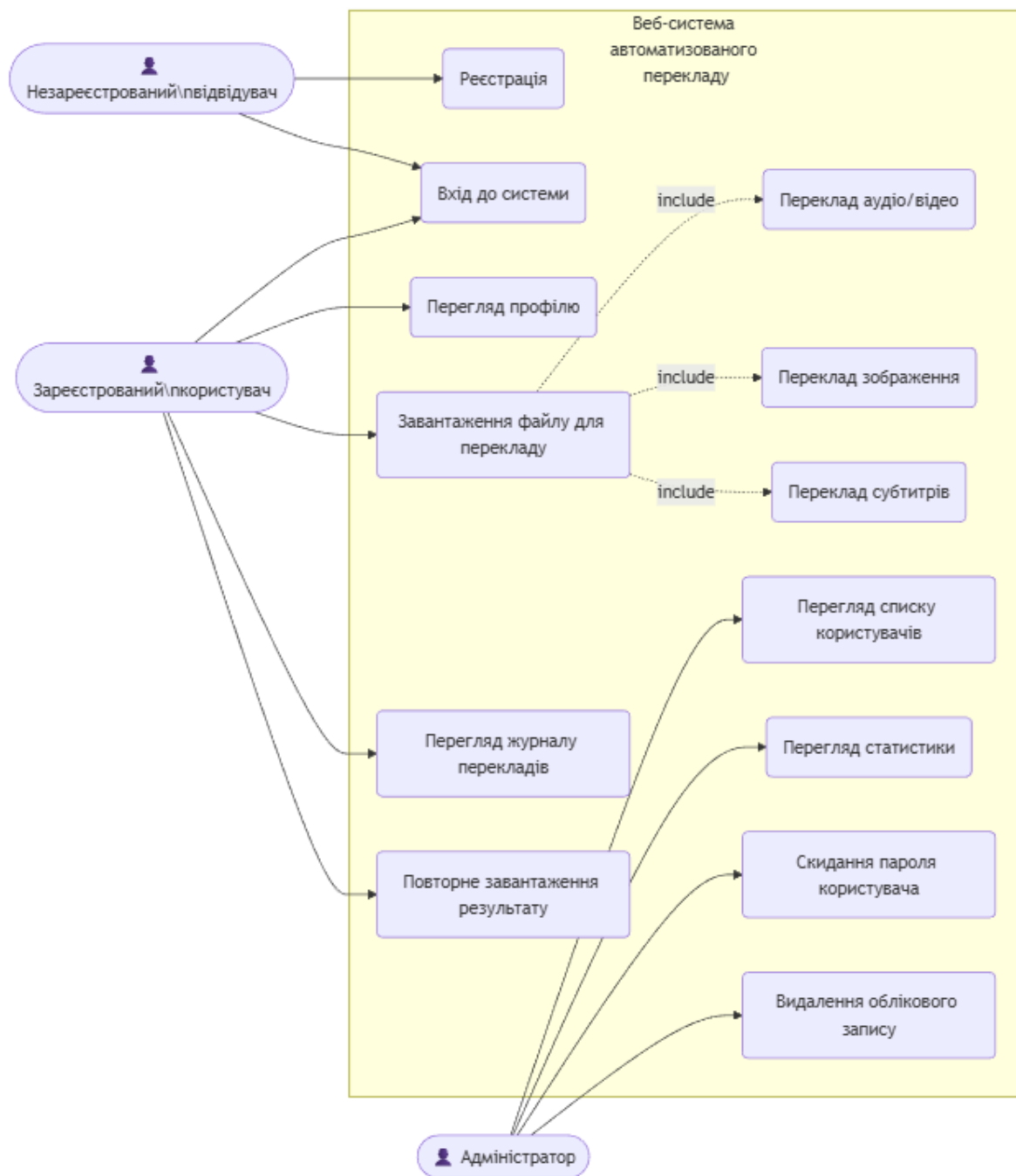


Рисунок 3.1 — Діаграма варіантів використання

На рисунку 3.1 наведено UML-діаграму варіантів використання системи з позначенням акторів у стандартній нотації (stickman) та зв'язками включення між варіантом «Завантажити файл для перекладу» та трьома спеціалізованими підсценаріями.

Таблиця 3.1 — Варіанти використання системи за акторами

Актор	Варіант використання	Передумова
Незареєстрований відвідувач	Реєстрація нового облікового запису	—
Незареєстрований відвідувач	Вхід до системи	Обліковий запис існує
Зареєстрований користувач	Вхід до системи	—
Зареєстрований користувач	Перегляд профілю	Активна сесія
Зареєстрований користувач	Завантаження файлу для перекладу	Активна сесія
Зареєстрований користувач	Переклад аудіо/відео	Файл формату MP3/WAV/M4A/FLAC/MP4
Зареєстрований користувач	Переклад зображення	Файл формату JPG/PNG
Зареєстрований користувач	Переклад субтитрів	Файл формату SRT
Зареєстрований користувач	Перегляд журналу перекладів	Активна сесія
Зареєстрований користувач	Повторне завантаження результату	Запис у журналі
Адміністратор	Перегляд списку користувачів	Доступ до CLI
Адміністратор	Перегляд статистики системи	Доступ до CLI
Адміністратор	Скидання пароля користувача	Доступ до CLI
Адміністратор	Видалення облікового запису	Доступ до CLI

З таблиці 3.1 видно, що система підтримує 14 варіантів використання, розподілених між трьома акторами. Центральний сценарій — «Завантаження файлу для перекладу» — розгалужується на три спеціалізовані підсценарії залежно від типу файлу.

3.2 Структура програмних компонентів та модель даних

3.2.1 Структура серверного модуля та організація компонентів

Серверний модуль організовано за принципом розділення відповідальності.

Таблиця 3.2 описує програмні модулі системи та їх функціональні обов'язки.

Таблиця 3.2 — Програмні модулі системи та їх функції

Модуль / компонент	Файл	Функції
Серверне ядро	main.py	Конфігурація, маршрутизація, обробники запитів, ініціалізація залежностей
Шар роботи з БД	main.py (db layer)	Операції з обліковими записами, журналом, кешем (патерн Репозиторій)
Модуль транскрипції	main.py (asr module)	Витягування аудіо з відео, транскрипція мовлення (Whisper)
Модуль OCR	main.py (ocr module)	Розпізнавання тексту на зображеннях (Tesseract)
Модуль субтитрів	main.py (srt module)	Розбір SRT, посегментний переклад, збирання результату
Модуль перекладу	main.py (translate module)	Взаємодія з DeepL API
Схема бази даних	db/init.sql	DDL-визначення таблиць, індексів, зовнішніх ключів
Адміністрування БД	db/manage_db.py	Ініціалізація, скидання, перевірка схеми
Адміністрування облікових записів	db/admin.py	Перегляд, скидання пароля, видалення користувачів
Клієнтський інтерфейс	index.html	Односторінковий застосунок (HTML/CSS/JS)

Серверна логіка зосереджена в основному модулі main.py, що містить чотири функціональні шари: конфігурацію (зчитування змінних середовища), моделі валідації вхідних даних (Pydantic), шар роботи з базою даних (патерн Репозиторій) та обробники запитів REST API. Схема бази даних зберігається в окремому SQL-файлі та виконується при ініціалізації. Інструменти адміністрування розміщено в окремій директорії db/ і не входять до складу серверного застосунку. Конфігурація системи зберігається у файлі .env окремо від коду.

3.2.2 Реалізація токенів сеансу та захист файлів

Для підтвердження особи користувача при кожному запиті використовується механізм підписаних токенів. Токен формується при успішному вході та містить три поля: числовий ідентифікатор користувача, його логін та часову мітку закінчення дії. Токен підписується алгоритмом HS256 з використанням секретного ключа, що зберігається у змінних середовища. Термін дії токена обмежено 24 годинами [19].

При кожному запиті до захищеного маршруту функція перевірки декодує токен та верифікує підпис. Пошкоджений або прострочений токен призводить до відповіді з кодом 401 без доступу до ресурсу. Такий підхід не потребує зберігання стану сесії на сервері — вся необхідна інформація міститься в самому токені.

Завантажені файли зберігаються під унікальними системними іменами, що генеруються випадково при кожному завантаженні. Оригінальне ім'я файлу зберігається лише в базі даних для відображення в журналі. Такий підхід виключає можливість вгадування шляху до чужого файлу та запобігає конфліктам імен. Тимчасові файли, що утворюються під час обробки (зокрема, аудіофайл, витягнутий з відео), видаляються одразу після завершення відповідного кроку.

3.3 Реалізація модулів конвеєра обробки мультимедіа

Конвеєр обробки реалізовано як центральну функцію серверного модуля. Алгоритм конвеєра описано у розділі 2; у цьому підрозділі розглянуто програмну реалізацію кожного модуля конвеєра: структуру компонентів, особливості реалізації та взаємодію з зовнішніми залежностями.

3.3.1 Модуль транскрипції аудіо та відео

Модуль транскрипції складається з двох функцій: витягування звукової доріжки та власне транскрипції.

Функція витягування звукової доріжки реалізована через виклик зовнішньої утиліти обробки мультимедіа як дочірнього процесу. Виклик налаштований на

заборону перезапису існуючих файлів та обмежений часовим лімітом 30 секунд. Результатом є тимчасовий аудіофайл у форматі WAV, що видається після завершення транскрипції.

Функція транскрипції реалізує відкладене завантаження нейромережевої моделі. Глобальний стан модуля ініціалізується як «не завантажено»; при першому виклику завантажується модель базового розміру та кешується в оперативній пам'яті для подальших викликів. Завдяки цьому час першого запуску сервера не залежить від розміру моделі. Функція повертає пару значень: розпізнаний текст та код мови, визначений автоматично.

Взаємодія модуля з рештою системи: функція транскрипції викликається центральною функцією конвеєра, що передає їй шлях до аудіофайлу; результат передається у функцію перекладу.

3.3.2 Модуль оптичного розпізнавання тексту

Модуль розпізнавання тексту на зображеннях реалізовано з явною перевіркою доступності зовнішнього рушія розпізнавання на старті системи. Результат перевірки зберігається у змінній стану та відображається в ендпоінті діагностики. При спробі обробити зображення за умови відсутності рушія система повертає код 503 з описовим повідомленням — замість нечитабельної технічної помилки.

Обробка зображення відбувається через ланцюжок двох компонентів: бібліотека роботи із зображеннями відкриває файл та надає уніфікований об'єкт незалежно від формату (JPEG або PNG); рушій оптичного розпізнавання приймає цей об'єкт та повертає рядок розпізнаного тексту. Мова оригіналу в поточній конфігурації не визначається автоматично й позначається як невідома.

3.3.3 Модуль обробки файлів субтитрів

Модуль обробки субтитрів реалізовано як два послідовних кроки: синтаксичний розбір вхідного файлу та збирання вихідного файлу.

На кроці розбору кожен субтитр виділяється у структурований об'єкт з трьома полями: порядковий номер, рядок часових міток та текст. Важливою особливістю реалізації є те, що часові мітки зберігаються як рядки без перетворення — це гарантує їх точне відтворення у вихідному файлі без ризику помилок форматування. Розбір враховує можливість багаторядкового тексту субтитру.

Переклад виконується поокремо для кожного субтитру: текст передається до служби перекладу, а відповідь замінює оригінальний текст у структурованому об'єкті. Такий підхід забезпечує природність перекладу в межах короткого контексту субтитру.

На кроці збирання файл формується послідовним записом кожного об'єкта: порядковий номер, незмінений рядок часових міток, перекладений текст та порожній рядок-роздільник. Вихідний файл зберігається з системним унікальним іменем та розширенням .srt.

3.4 Структура клієнтської частини та взаємодія з сервером

Клієнтська частина реалізована як односторінковий застосунок у єдиному HTML-файлі, що містить розмітку, стилі та скриптовий код. Таке рішення усуває необхідність кроку збірки та спрощує розгортання — клієнтський файл обслуговується безпосередньо серверним модулем як статичний ресурс.

З архітектурної точки зору клієнтська частина побудована навколо двох станів: стан «не авторизовано» та стан «авторизовано». Перемикання між станами виконується на основі наявності токена в локальному сховищі браузера. При завантаженні сторінки відбувається перевірка: якщо токен присутній, відображається головний інтерфейс; якщо відсутній — форма автентифікації.

Взаємодія з серверним програмним інтерфейсом реалізована через асинхронні HTTP-запити. Токен сеансу автоматично додається до заголовка авторизації при кожному запиті до захищених маршрутів — ця логіка централізована в допоміжній функції, що обгортає базовий механізм HTTP-запитів. Файл та цільова мова передаються у форматі multipart/form-data.

Клієнтська частина складається з кількох функціональних блоків. Блок автентифікації містить форми реєстрації та входу з клієнтською валідацією полів до відправлення запиту. При успішній відповіді токен зберігається в локальному сховищі та відбувається перехід до головного інтерфейсу. Блок завантаження файлів реалізує перетягування файлів у зону скидання та стандартний вибір через діалог; підтримані формати перевіряються на клієнті до відправлення; індикатор завантаження блокує повторні відправлення під час обробки. Блок результатів відображає посилання для завантаження перекладеного файлу та метадані операції: визначена мова оригіналу, кількість оброблених фрагментів, ознака кешованого результату. Блок журналу перекладів відображається на бічній панелі та завантажується при вході в систему; кожен запис містить ім'я файлу, тип, мови та дату операції. Список доступних мов перекладу завантажується одноразово при ініціалізації застосунку та кешується в пам'яті браузера для формування меню вибору.

3.5 Реалізація інструментів адміністрування

Інструменти адміністрування є окремим програмним компонентом, що взаємодіє безпосередньо з базою даних через той самий рядок підключення, що і серверний модуль. Це забезпечує узгодженість операцій: адміністративні скрипти бачать актуальний стан бази даних та дотримуються тих самих правил цілісності, що й основна система.

Компонент управління схемою бази даних відповідає за ініціалізацію, скидання та перевірку структури таблиць. Він зчитує та виконує SQL-файл схеми, що дозволяє вносити зміни до структури бази даних через редагування одного файлу без правки скриптового коду. Перевірка з'єднання виконує тестовий запит та повідомляє про доступність бази даних — це зручно при первинному налаштуванні системи.

Компонент адміністрування облікових записів реалізований як клас з методами для кожної операції. Методи перегляду та статистики виконують агрегуючі запити з об'єднанням таблиць облікових записів та журналу перекладів.

Метод видалення виконує одну операцію видалення запису користувача; каскадне видалення на рівні схеми бази даних гарантує автоматичне видалення всіх пов'язаних записів журналу. Перед виконанням деструктивних операцій компонент запитує підтвердження від адміністратора.

Обидва компоненти підтримують запуск з аргументами командного рядка, що дозволяє вбудовувати їх у скрипти автоматизації та виконувати у неінтерактивному режимі за потреби.

4 РОЗГОРТАННЯ, ДЕМОНСТРАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

У розділі наведено вимоги до програмного й апаратного забезпечення системи автоматизованого перекладу мультимедійного контенту, описано процес інсталяції та розгортання застосунку, а також продемонстровано основні сценарії його використання. Окрему увагу приділено результатам тестування та перевірці відповідності реалізованої системи функціональним і нефункціональним вимогам, зокрема коректності обробки файлів різних форматів, роботі механізму кешування та захисту від несанкціонованого доступу.

4.1 Інсталяція та вимоги до обчислювальної техніки

Для розгортання системи необхідне стандартне апаратне забезпечення з достатнім обсягом оперативної пам'яті для роботи нейромережевої моделі Whisper та встановлений набір програмних компонентів: Python, PostgreSQL, FFmpeg і Tesseract OCR. Процес інсталяції зводиться до налаштування змінних середовища, ініціалізації схеми бази даних та запуску серверного процесу, що робить розгортання системи доступним без спеціальних технічних знань.

4.1.1 Вимоги до апаратного та програмного забезпечення

Для розгортання та коректної роботи системи необхідне таке апаратне та програмне забезпечення.

Мінімальні вимоги до апаратного забезпечення:

- Процесор: 4-ядерний, тактова частота від 2.0 ГГц
- Оперативна пам'ять: 4 ГБ (рекомендовано 8 ГБ для роботи з моделлю Whisper)
- Дисковий простір: 2 ГБ для системи та моделей (модель Whisper base - 74 МБ)
- Мережеве з'єднання: необхідне для звернень до DeepL API

Вимоги до програмного забезпечення:

- Операційна система: Windows 10/11, Ubuntu 20.04+, macOS 12+
- Python 3.10 або вище
- PostgreSQL 13 або вище
- FFmpeg (для обробки відеофайлів MP4)
- Tesseract OCR 5.x (для обробки зображень)

4.1.2 Порядок інсталяції

Процес інсталяції системи складається з чотирьох послідовних кроків і не потребує спеціальних технічних знань за умови попереднього встановлення Python та PostgreSQL.

На першому кроці необхідно створити та активувати віртуальне середовище Python, після чого встановити всі залежності з файлу `requirements.txt`. Використання віртуального середовища є обов'язковим для ізоляції залежностей проекту від системного Python та уникнення конфліктів версій пакетів. Встановлення залежностей виконується одноразово і займає кілька хвилин залежно від швидкості мережевого з'єднання, оскільки пакет `openai-whisper` при першому запуску також завантажує модель розміром 74 МБ.

На другому кроці виконується налаштування змінних середовища. Для цього необхідно скопіювати файл-шаблон `.env.example` у файл `.env` та заповнити три обов'язкові значення: ключ DeepL API, секретний ключ для підпису JWT-токенів та рядок підключення до бази даних PostgreSQL. Ключ DeepL API отримується безкоштовно після реєстрації на офіційному сайті сервісу - безкоштовний план надає 500 тисяч символів на місяць, що є достатнім для розробки та тестування. Секретний ключ JWT повинен бути випадковим рядком довжиною не менше 32 символів; для його генерації можна скористатися стандартними засобами Python. Рядок підключення до бази даних містить ім'я користувача, пароль, адресу сервера та назву бази даних у стандартному форматі URI.

На третьому кроці виконується ініціалізація схеми бази даних. Для цього необхідно попередньо створити порожню базу даних PostgreSQL з назвою, що відповідає значенню в `.env`, після чого запустити скрипт `db/manage_db.py` з параметром `--init`. Скрипт зчитує SQL-файл `db/init.sql` та виконує його, створюючи всі необхідні таблиці, зовнішні ключі та індекси. Після успішного виконання скрипт виводить підтвердження про створення кожної таблиці. Для перевірки коректності ініціалізації можна скористатися параметром `--schema`, який виводить поточну структуру всіх таблиць.

На четвертому кроці запускається серверний процес командою `python main_auth.py`. Після запуску сервер виводить у консоль інформацію про доступність кожного компонента: Whisper, Tesseract, DeepL та бази даних. Сервер запускається на адресі `http://127.0.0.1:8001` і готовий до прийому запитів. Веб-інтерфейс доступний за цією ж адресою в будь-якому сучасному браузері. Інтерактивна документація Swagger UI доступна за адресою `http://127.0.0.1:8001/docs` і дозволяє тестувати всі ендпоінти API безпосередньо в браузері.

4.2 Демонстрація функціоналу та сценарії роботи

У цьому підрозділі наведено демонстрацію функціоналу системи через сім сценаріїв роботи, що охоплюють усі реалізовані варіанти використання. Кожен сценарій описує мету, передумови та послідовність дій суб'єкта доступу з відповідною реакцією системи на кожному кроці. Такий підхід дозволяє оцінити практичну цінність реалізованого функціоналу та переконатися у відповідності системи сформованим вимогам.

4.2.1 Сценарій 1. Реєстрація та автентифікація

Мета сценарію полягає в отриманні доступу до функціоналу системи шляхом створення нового облікового запису та подальшого входу. Передумовою є запуснена система та відкритий браузер за адресою сервера.

Суб'єкт доступу відкриває головну сторінку застосунку. Оскільки JWT-токен відсутній у локальному сховищі браузера, система автоматично відображає форму автентифікації з двома вкладками: «Вхід» та «Реєстрація». Клієнт системи переходить на вкладку «Реєстрація» та заповнює три поля: логін, електронна адреса та пароль.(Рис.6)

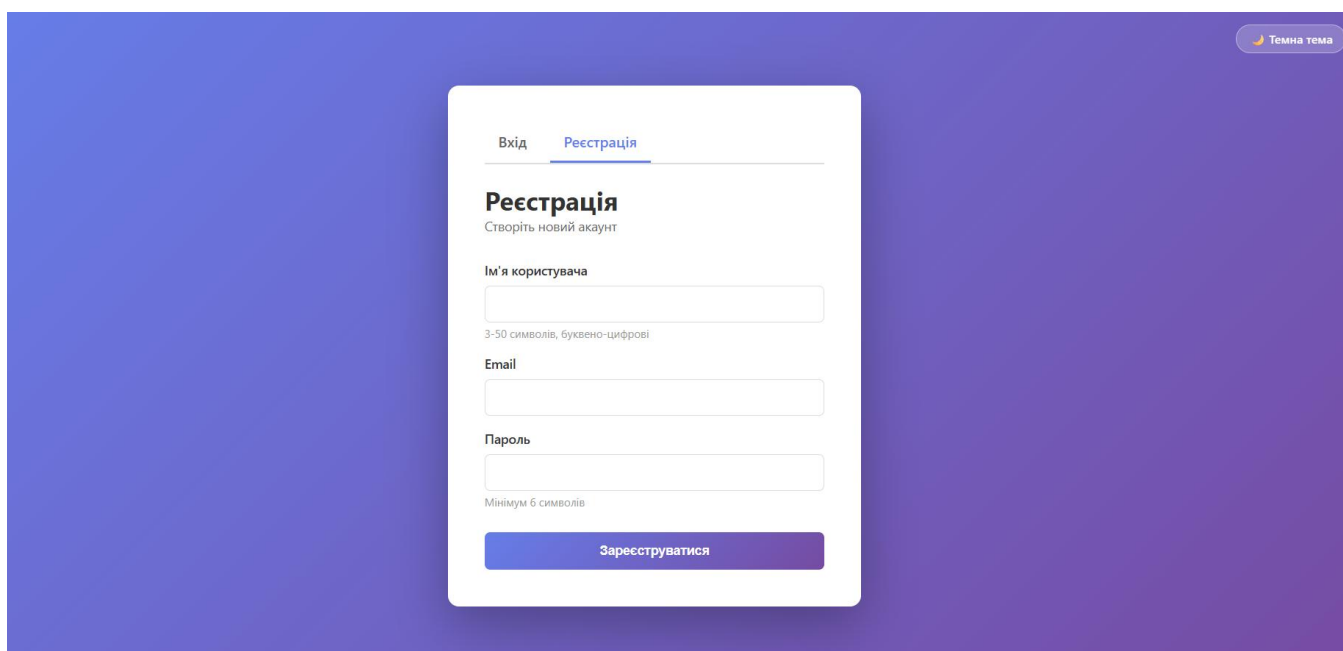
The image shows a web application interface with a dark blue gradient background. In the top right corner, there is a small button labeled "Темна тема" (Dark theme) with a sun icon. Centered on the screen is a white registration form. At the top of the form, there are two tabs: "Вхід" (Login) and "Реєстрація" (Registration), with "Реєстрація" being the active tab. Below the tabs, the title "Реєстрація" is displayed in bold, followed by the subtitle "Створіть новий акаунт" (Create a new account). The form contains three input fields: "Ім'я користувача" (Username) with a hint "3-50 символів, буквено-цифрові" (3-50 characters, alphanumeric), "Email", and "Пароль" (Password) with a hint "Мінімум 6 символів" (Minimum 6 characters). At the bottom of the form is a blue button labeled "Зареєструватися" (Register).

Рис.6. Вкладка реєстрації користувача у системі

Після натискання кнопки «Зареєструватися» фронтенд надсилає POST-запит на ендпоінт `/api/auth/register` із даними форми у форматі JSON. На сервері виконується послідовна багаторівнева перевірка вхідних даних: спочатку Pydantic-валідатор перевіряє коректність кожного поля відповідно до встановлених правил - формат електронної адреси, мінімальна довжина пароля та обов'язковість імені користувача. Після успішної валідації формату система звертається до бази даних для перевірки унікальності логіна та електронної адреси, оскільки дублювання облікових записів є неприпустимим.

Якщо всі перевірки пройдено успішно, пароль хешується через двоетапний алгоритм - спочатку SHA-256, а потім bcrypt із вбудованим сіллю, що унеможливорює відновлення оригінального пароля навіть у разі витоку бази даних. Новий запис зберігається в таблиці `users`, після чого сервер генерує JWT-токен із

терміном дії 24 години, підписаний секретним ключем із файлу конфігурації, та повертає його клієнту у відповіді.

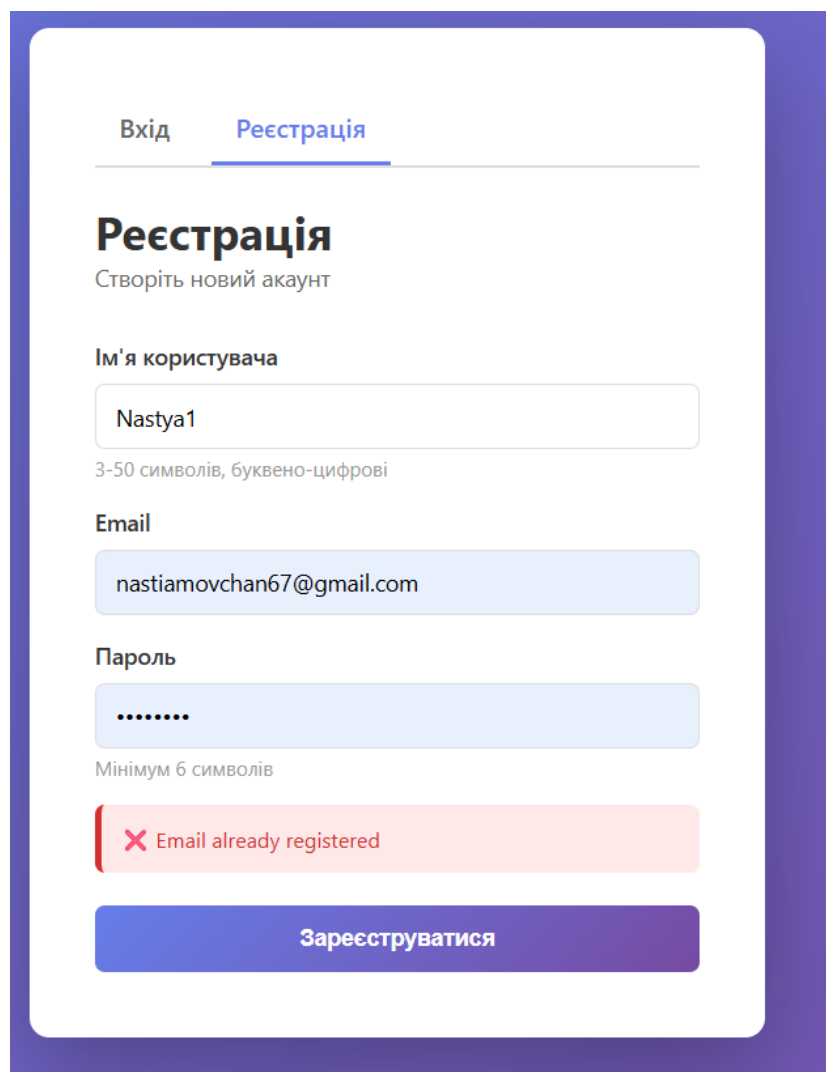
Фронтенд зберігає отриманий токен у локальному сховищі браузера (localStorage) та автоматично додає його до заголовка Authorization: Bearer КОЖНОГО наступного запиту до захищених ендпоінтів. Після успішної авторизації інтерфейс перемикається на головну сторінку з формою завантаження файлу, де користувач може обрати мультимедійний файл підтримуваного формату - MP4, MP3, WAV, JPG, PNG або SRT - та вказати цільову мову перекладу. У верхній частині сторінки відображається ім'я авторизованого суб'єкта доступу та кнопка виходу з системи, натискання якої очищає токен зі сховища та повертає користувача на сторінку входу. (Рис.7)

The screenshot displays the 'Multimedia Translation' web application. At the top, the title 'Multimedia Translation' is followed by a user greeting 'Привіт, Nastya !' and a 'Вийти' (Logout) button. Below this is a navigation bar with three tabs: 'Переклад' (Translation), 'Історія' (History), and 'Профіль' (Profile). The main heading is 'Завантажте файл для перекладу' (Upload file for translation). Underneath, it lists supported formats: 'Підтримуються: MP4, MP3, WAV, JPG, PNG, SRT'. A dashed box labeled 'Файл' (File) contains the instruction 'Перетягніть файл сюди або натисніть для вибору' (Drag file here or click to select). Below this is a 'Мова перекладу' (Translation language) dropdown menu currently set to 'Afrikaans'. A large blue button labeled 'Перекласти' (Translate) is positioned below the dropdown. At the bottom, the section 'Результат перекладу' (Translation result) is visible.

Рис.7 Головна сторінка

При наступному відвідуванні суб'єкт доступу вводить логін та пароль на вкладці «Вхід». Сервер знаходить запис у базі даних за логіном, хешує введений пароль тим самим двоетапним способом та порівнює з збереженим хешем. При збігу генерується новий JWT-токен і повертається клієнту.

Система коректно обробляє всі помилкові сценарії. При спробі реєстрації з вже зайнятим логіном повертається повідомлення про дублікат. При введенні невірного пароля під час входу повертається повідомлення про невірні облікові дані без уточнення, яке саме поле є невірним - це стандартна практика безпеки, що унеможливорює перебір існуючих логінів. (Рис.8)



The image shows a web registration form titled "Реєстрація" (Registration) with a subtitle "Створіть новий акаунт" (Create a new account). At the top, there are two tabs: "Вхід" (Login) and "Реєстрація" (Registration), with the latter being active. The form contains three input fields: "Ім'я користувача" (Username) with the value "Nastya1", "Email" with the value "nastiamovchan67@gmail.com", and "Пароль" (Password) with masked characters ".....". Below the password field, a red error message "✗ Email already registered" is displayed. At the bottom of the form is a blue button labeled "Зареєструватися" (Register).

Рис.8. Помилка реєстрації: обліковий запис із таким email вже існує

4.2.2 Сценарій 2. Переклад аудіофайлу

Мета сценарію - завантажити аудіофайл та отримати перекладений текст. Передумовою є авторизований суб'єкт доступу та встановлені компоненти FFmpeg і Whisper.

Авторизований суб'єкт доступу перетягує аудіофайл формату MP3 у зону drag&drop або натискає на неї для відкриття стандартного діалогу вибору файлу. Після вибору файлу його назва відображається у зоні завантаження, що підтверджує успішне прикріплення. Потім суб'єкт доступу обирає цільову мову перекладу з випадаючого меню. Список мов завантажується динамічно з DeepL API при відкритті сторінки та містить понад 30 мов, включаючи українську.(Рис.8)

The screenshot shows the 'Multimedia Translation' web application. At the top, there is a header with the title 'Multimedia Translation', a greeting 'Привіт, Nastya !', and a 'Вийти' (Logout) button. Below the header, there are three tabs: 'Переклад' (Translation), 'Історія' (History), and 'Профіль' (Profile). The 'Переклад' tab is active. The main content area has a heading 'Завантажте файл для перекладу' (Upload file for translation). Below this, it lists supported formats: 'Підтримуються: MP4, MP3, WAV, JPG, PNG, SRT'. There is a section labeled 'Файл' (File) with a dashed blue box containing the text 'Перетягніть файл сюди або натисніть для вибору' (Drag file here or click to select). Below this box, a message indicates a file has been selected: '✓ Файл обрано: A1-050-JULIE-WHO-IS-YOUR-FRIEND.mp3 (0.87 MB)'. Further down, there is a 'Мова перекладу' (Translation language) dropdown menu currently set to 'Ukrainian'. At the bottom, there is a large blue button labeled 'Перекласти' (Translate).

Рис.8. Завантажили аудіофайл та обрали мову перекладу

Після натискання кнопки «Перекласти» фронтенд формує запит у форматі multipart/form-data та надсилає його на сервер з JWT-токеном у заголовку авторизації. Інтерфейс відображає індикатор завантаження, що інформує користувача про тривалий процес обробки.

На сервері виконується така послідовність дій. Спочатку перевіряється дійсність JWT-токена та витягуються дані поточного суб'єкта доступу. Потім перевіряється розмір завантаженого файлу - він не повинен перевищувати 50 МБ - та коректність розширення. Файл зберігається у папці завантажень з UUID-іменем для забезпечення унікальності. Далі система перевіряє наявність кешованого результату в журналі перекладів: якщо цей файл з цією мовою вже перекладався поточним суб'єктом доступу, повертається готовий результат без повторної обробки.

Якщо кеш відсутній, запускається конвеєр обробки. Перевіряється доступність FFmpeg у системі. Модель Whisper завантажується при першому зверненні та виконує транскрипцію аудіофайлу, автоматично визначаючи мову оригіналу. Отриманий текст передається в DeerpL API для перекладу на обрану мову. Перекладений текст зберігається у папці результатів з UUID-іменем та розширенням .txt. Запис про виконаний переклад зберігається в журналі перекладів з усіма метаданими: ім'я файлу, тип, мови, кількість сегментів, час обробки. Тимчасовий файл завантаження видаляється. (Рис.9)

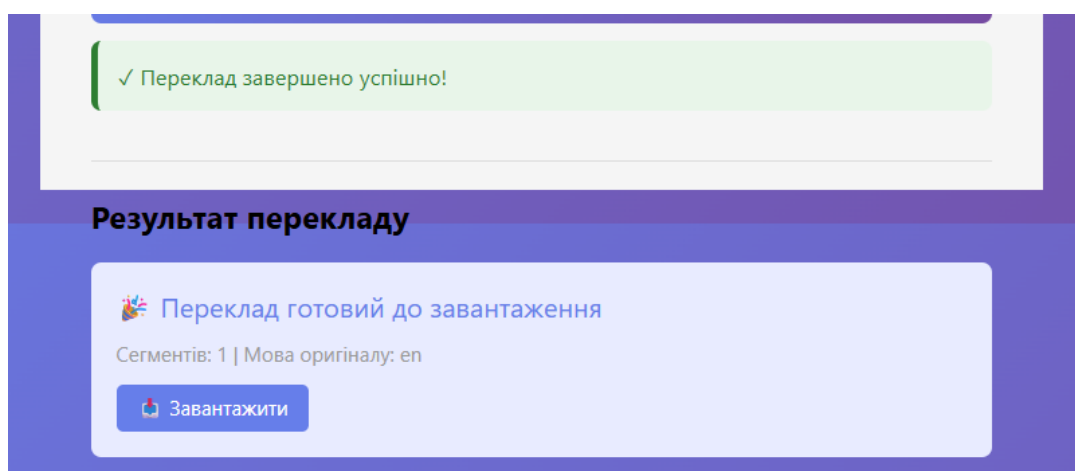


Рис.9 Можливість завантажити перекладений файл

Сервер повертає клієнту посилання для завантаження перекладеного файлу, код виявленої мови оригіналу, кількість оброблених сегментів та прапорець кешування. Фронтенд відображає ці метадані та кнопку завантаження. Запис про переклад з'являється в сайдбарі з журналом перекладів.

4.2.3 Сценарій 3. Переклад відеофайлу

Мета сценарію - завантажити відеофайл та отримати перекладений текст звукового супроводу. Передумовою є авторизований суб'єкт доступу та встановлені компоненти FFmpeg і Whisper.

Суб'єкт доступу завантажує відеофайл формату MP4. Система автоматично визначає тип файлу за розширенням та застосовує відповідний конвеєр обробки. Відмінність від обробки аудіофайлу полягає в додатковому кроці: перед транскрипцією FFmpeg витягує аудіодоріжку з відеофайлу та зберігає її у форматі WAV. Цей крок є прозорим для користувача - він не потребує жодних додаткових дій.

Після витягування аудіодоріжки подальша обробка відбувається ідентично до аудіофайлу: Whisper виконує транскрипцію, DeepL перекладає отриманий текст, результат зберігається як TXT-файл. Тимчасовий WAV-файл видаляється після завершення транскрипції. Суб'єкт доступу отримує перекладений текст звукового супроводу відео у вигляді текстового файлу для завантаження.

Важливою особливістю є встановлений таймаут 30 секунд для операції витягування аудіо через FFmpeg. Це запобігає зависанню сервера при обробці пошкоджених або надто великих відеофайлів - у разі перевищення таймауту повертається зрозуміле повідомлення про помилку.

4.2.4 Сценарій 4. Переклад файлу субтитрів SRT

Мета сценарію - перекласти файл субтитрів зі збереженням часових міток та структури. Передумовою є авторизований суб'єкт доступу та наявний SRT-файл для перекладу.

Суб'єкт доступу завантажує файл з розширенням .srt. Система автоматично визначає тип файлу та застосовує спеціалізований конвеєр обробки субтитрів без будь-яких додаткових дій з боку користувача.

Після отримання файлу сервер зчитує його вміст з кодуванням UTF-8 та розбиває на окремі сегменти за допомогою регулярного виразу. Кожен сегмент стандартного SRT-файлу складається з трьох частин: порядкового номера, рядка з часовими мітками початку та кінця у форматі «години:хвилини:секунди,мілісекунди», та тексту субтитру, що може займати один або кілька рядків. Регулярний вираз коректно обробляє багаторядкові субтитри завдяки відповідному прапорцю режиму відповідності.

Текст кожного сегмента окремо передається в DeepL API для перекладу. Такий підхід забезпечує коректний переклад кожного субтитру в межах його власного контексту та дозволяє зберегти природність мови. Після перекладу всіх сегментів файл збирається назад у стандартному SRT-форматі: порядковий номер, рядок з часовими мітками, перекладений текст та порожній рядок-роздільник між сегментами. Оригінальні порядкові номери та часові мітки зберігаються без жодних змін.

Суб'єкт доступу отримує готовий SRT-файл з перекладеними субтитрами та оригінальними часовими мітками. Цей файл можна безпосередньо використовувати у будь-якому відеоплеєрі, що підтримує зовнішні субтитри, без жодного додаткового редагування. У метаданих відповіді відображається кількість оброблених сегментів, що дозволяє суб'єкту доступу переконатися в повноті перекладу.

4.2.5 Сценарій 5. Переклад зображення з текстом

Мета сценарію - розпізнати та перекласти текст, що міститься на зображенні. Передумовою є авторизований суб'єкт доступу та встановлений Tesseract OCR.

Суб'єкт доступу завантажує файл формату JPG або PNG, що містить текст. Це може бути скан документа, знімок екрана з текстом, фотографія таблички або

будь-яке інше зображення з читабельним текстом. Система визначає тип файлу та застосовує OCR-конвеєр.

Перед запуском розпізнавання система перевіряє доступність Tesseract у системі. Якщо Tesseract не встановлено або недоступний у системному PATH, сервер повертає HTTP 503 з відповідним повідомленням та інструкцією щодо встановлення. Це забезпечує зрозумілу діагностику замість технічного виключення.

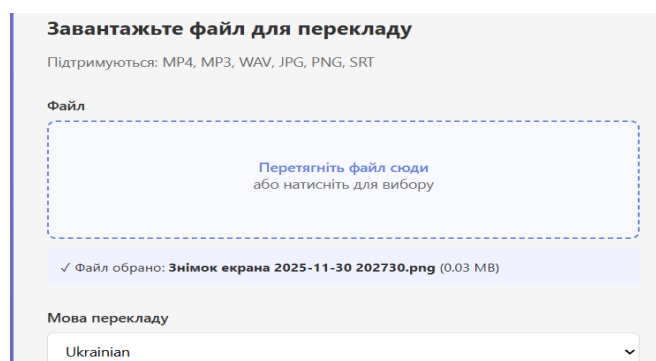


Рис.10 Переклад зображення

Бібліотека Pillow відкриває файл зображення та передає його в pytesseract для розпізнавання. Tesseract аналізує зображення та повертає розпізнаний текст. Якість розпізнавання суттєво залежить від якості вхідного зображення: зображення з чітким текстом на контрастному фоні та достатньою роздільною здатністю розпізнаються з високою точністю, тоді як зображення з низькою контрастністю, розмитим текстом або складним фоном можуть давати гірші результати. Розпізнаний текст передається в DeepL для перекладу на обрану мову, після чого результат зберігається як TXT-файл та надається для завантаження.

4.2.6 Сценарій 6. Повторний переклад та кешування

Мета сценарію - продемонструвати роботу механізму кешування при повторному завантаженні файлу. Передумовою є авторизований суб'єкт доступу та файл, що вже перекладався раніше.

Суб'єкт доступу завантажує той самий файл з тією ж цільовою мовою, що й раніше. Після збереження файлу з UUID-іменем сервер виконує запит до журналу перекладів, шукаючи запис з відповідним ідентифікатором суб'єкта доступу, оригінальним іменем файлу та цільовою мовою. Якщо такий запис знайдено і відповідний вихідний файл існує на диску, сервер повертає кешований результат без запуску конвеєра обробки.(Рис.11)

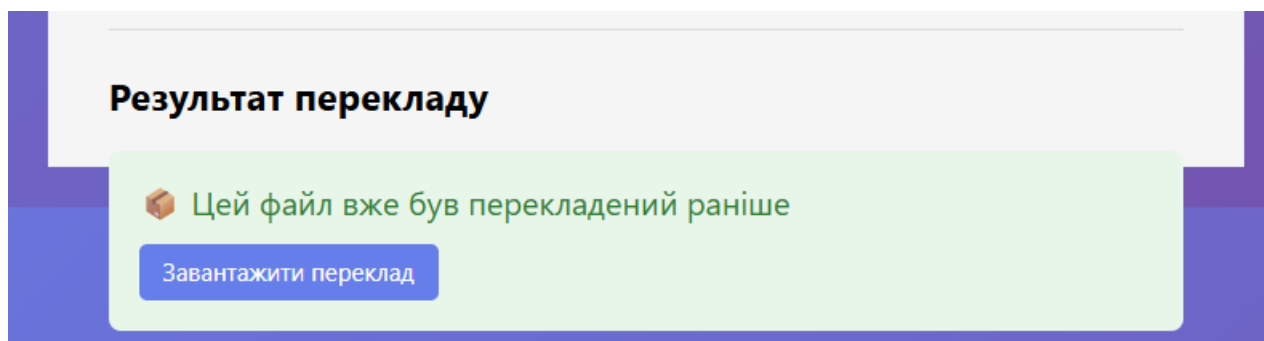


Рис.11. Відображення кешованого результату перекладу

Час відповіді при кешованому результаті становить менше 100 мілісекунд порівняно з кількома хвилинами при повній обробці аудіофайлу. Це суттєво економить час очікування та ліміти зовнішніх API, зокрема безкоштовний ліміт DeepL у 500 тисяч символів на місяць. У відповіді сервера прапорець кешування має значення «true», що відображається у фронтенді поряд з посиланням для завантаження.

Важливою деталлю реалізації є те, що кеш є персональним: результати перекладів одного суб'єкта доступу не доступні іншим. Крім того, зміна цільової мови при повторному завантаженні того самого файлу призводить до повного запуску конвеєра обробки, оскільки кеш прив'язаний до конкретної комбінації файлу та мови.

4.2.7 Сценарій 7. Перегляд журналу перекладів

Мета сценарію - переглянути історію виконаних перекладів та повторно завантажити раніше перекладений файл. Передумовою є авторизований суб'єкт доступу з щонайменше одним виконаним перекладом.

Журнал перекладів відображається у сайдбарі праворуч від основної форми завантаження. Він завантажується автоматично при вході в систему через запит до ендпоінту отримання історії. Сервер повертає до 50 останніх перекладів поточного суб'єкта доступу у зворотному хронологічному порядку - від найновішого до найстарішого.(Рис.12)

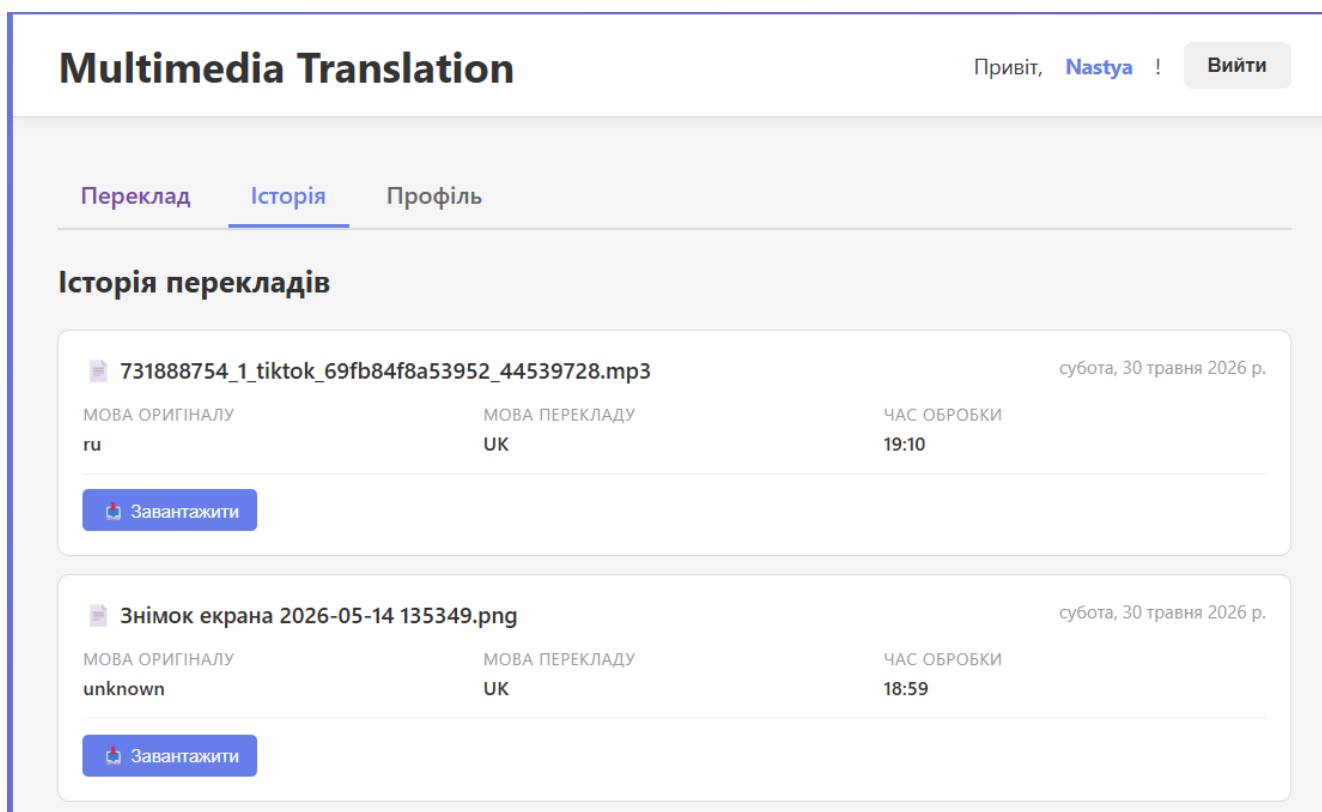


Рис.12. Вкладка історії перекладів

Для кожного запису в журналі відображається оригінальне ім'я файлу, тип файлу у вигляді іконки або мітки, мова оригіналу та цільова мова перекладу, а також дата та час виконання. Ця інформація дозволяє суб'єкту доступу швидко знайти потрібний переклад серед усіх виконаних.

Суб'єкт доступу може натиснути на запис у журналі для повторного завантаження вже перекладеного файлу. Оскільки файли зберігаються у папці результатів з UUID-іменами, вони залишаються доступними для завантаження протягом усього часу роботи системи. Ця функція є особливо зручною у поєднанні з механізмом кешування: навіть якщо суб'єкт доступу закрив вкладку браузера після перекладу, він може знайти результат у журналі та завантажити його пізніше.

4.2.8 Сценарій 8. Перевірка стану системи

Мета сценарію - перевірити доступність усіх компонентів системи. Цей сценарій є корисним для адміністратора при діагностиці проблем або перевірці готовності системи до роботи.

Ендпоінт перевірки стану є публічним і не потребує автентифікації. Він повертає поточний стан чотирьох компонентів: Whisper, Tesseract, DeepL та бази даних. Для кожного компонента перевіряється його фактична доступність: для Whisper - можливість завантаження модуля, для Tesseract - наявність виконуваного файлу в системі, для DeepL - наявність ключа API у змінних середовища, для бази даних - можливість отримання з'єднання з пулу.

Якщо один з компонентів недоступний, відповідний прапорець у відповіді матиме значення «false». Це дозволяє адміністратору швидко діагностувати проблему без перегляду логів сервера. Наприклад, якщо Whisper недоступний, система не зможе обробляти аудіо- та відеофайли, але продовжить коректно обробляти зображення та SRT-файли. Якщо DeepL недоступний через відсутність ключа API, жоден тип файлів не може бути перекладений. Якщо база даних недоступна, система не зможе зберігати результати та перевіряти автентифікацію.

4.3 Тестування системи та результати

Тестування системи проводилось паралельно з розробкою. Застосовувались кілька видів тестування: функціональне тестування ендпоінтів, тестування обробки файлів різних форматів, тестування механізму кешування та тестування безпеки. Основним інструментом тестування API слугував Swagger UI, доступний за адресою /docs, що дозволяє виконувати запити до всіх ендпоінтів безпосередньо в браузері. Додатково розроблено скрипт автоматизованого тестування основних сценаріїв.

4.3.1 Тестування автентифікації

Тестування ендпоінту реєстрації включало перевірку шести сценаріїв, що охоплюють як успішні, так і помилкові випадки. Результати тестування наведено в таблиці 9.

Таблиця 9-Результати тестування ендпоінту реєстрації

Сценарій	Вхідні дані	Очікуваний результат	Фактичний результат
Успішна реєстрація	Валідні username, email, password	HTTP 200, JWT-токен	✓ Відповідає
Дублікат username	Існуючий username	HTTP 400, повідомлення	✓ Відповідає
Дублікат email	Існуючий email	HTTP 400, повідомлення	✓ Відповідає
Короткий username	username менше 3 символів	HTTP 422, деталі помилки	✓ Відповідає
Невалідний email	email без символу @	HTTP 422, деталі помилки	✓ Відповідає
Короткий пароль	password менше 6 символів	HTTP 422, деталі помилки	✓ Відповідає

Тестування ендпоінту входу включало перевірку успішного входу з коректними обліковими даними, спроби входу з невірним паролем та спроби входу з неіснуючим логіном. В усіх помилкових випадках сервер повертав HTTP 401 з однаковим повідомленням «Invalid username or password» без уточнення, яке саме поле є невірним. Це є навмисним рішенням безпеки, що унеможливорює перебір існуючих логінів.

Тестування захисту ендпоінтів включало перевірку поведінки при відсутньому токені, простроченому токені та підробленому токені. В усіх трьох випадках сервер коректно повертав HTTP 401 з відповідним повідомленням про помилку.

4.3.2 Тестування обробки файлів

Тестування ендпоінту перекладу проводилось для кожного підтримуваного формату файлів окремо. Результати тестування наведено в таблиці 10.

Таблиця 10 - Результати тестування обробки файлів різних форматів

Формат	Тестовий файл	Очікуваний результат	Фактичний результат
MP3	Аудіо 30 сек, англійська	Транскрипція + переклад, мова «en»	✓ Відповідає
MP4	Відео 1 хв, англійська	Витягування аудіо + транскрипція + переклад	✓ Відповідає
SRT	10 сегментів, англійська	Переклад з збереженням міток	✓ Відповідає
JPG	Чіткий текст на білому фоні	OCR + переклад	✓ Відповідає
PNG	Текст з низькою контрастністю	OCR (знижена точність) + переклад	✓ Часткова точність OCR
Невідомий формат	Файл .docx	HTTP 400, повідомлення	✓ Відповідає
Перевищення розміру	Файл понад 50 МБ	HTTP 400, повідомлення	✓ Відповідає

Для аудіофайлів перевірялась коректність транскрипції та перекладу для файлів різної тривалості та якості запису. Тестовий файл MP3 тривалістю 30 секунд з чітким мовленням успішно транскрибовано та перекладено з англійської на українську мову. Виявлена мова оригіналу коректно визначена як «en». Файл з фоновим шумом також успішно транскрибовано, хоча з дещо нижчою точністю розпізнавання окремих слів

Для SRT-файлів перевірялась коректність збереження часових міток та форматування після перекладу. Тестовий файл з десятима сегментами успішно перекладено, всі часові мітки збережено без змін. Перекладений файл коректно відтворювався у відеоплеєрі з синхронізацією субтитрів.

Для зображень перевірялась якість розпізнавання тексту при різних умовах. Зображення з чітким текстом на білому фоні та роздільною здатністю не менше 150 DPI розпізнавалось з високою точністю. Зображення з низькою контрастністю або розмитим текстом давало гірші результати розпізнавання, що є очікуваною поведінкою для будь-якого OCR-рушія.

4.3.3 Тестування механізму кешування

Тестування механізму кешування проводилось шляхом повторного завантаження ідентичних файлів у різних комбінаціях. Результати наведено в таблиці 11.

Таблиця 11 - Результати тестування механізму кешування

Сценарій	Очікуваний результат	Фактичний результат
Перше завантаження файлу	cached: false, повна обробка	✓ Відповідає
Повторне завантаження того самого файлу, та сама мова	cached: true, миттєва відповідь	✓ Відповідає
Той самий файл, інша цільова мова	cached: false, повна обробка	✓ Відповідає
Той самий файл, інший суб'єкт доступу	cached: false, повна обробка	✓ Відповідає
Файл видалено з диска, запис є в БД	cached: false, повна обробка	✓ Відповідає

При першому завантаженні файлу відповідь містила прапорець кешування зі значенням «false». При повторному завантаженні того самого файлу з тією ж цільовою мовою відповідь містила прапорець зі значенням «true», а час відповіді скоротився з кількох десятків секунд до менш ніж 100 мілісекунд. При зміні цільової мови кеш не використовувався, що є коректною поведінкою. Кеш також не використовувався при завантаженні того самого файлу іншим суб'єктом доступу, що підтверджує персональний характер кешування.

4.3.4 Тестування безпеки

Тестування безпеки охоплювало три аспекти: захист ендпоінтів, валідацію вхідних даних та захист від SQL-ін'єкцій.

Захист ендпоінтів перевірявся шляхом надсилання запитів до захищених ендпоінтів без токена, з простроченим токеном та з підробленим токеном. В усіх трьох випадках сервер коректно повертав HTTP 401 з відповідним повідомленням. Спроба доступу до профілю іншого суб'єкта доступу через підміну ідентифікатора в токені також коректно відхилялась.

Валідація вхідних даних перевірялась шляхом передачі граничних значень та некоректних форматів у полях форми реєстрації. Pydantic-валідатор коректно відхиляв усі невалідні значення та повертав детальні повідомлення про помилки з HTTP 422.

Захист від SQL-ін'єкцій перевірявся шляхом передачі рядків, що містять типові SQL-ін'єкції, у полях форми реєстрації та параметрах запитів. Параметризовані запити psycopg2 коректно обробляли ці рядки як звичайні текстові значення, не допускаючи їх інтерпретації як SQL-команд. Жодна спроба SQL-ін'єкції не призвела до помилки бази даних або несанкціонованого доступу до даних.

Обмеження розміру файлів перевірялось шляхом спроби завантажити файл розміром понад 50 МБ. Сервер коректно повертав HTTP 400 з повідомленням про перевищення ліміту до початку обробки файлу.

4.3.5 Тестування нефункціональних вимог

Тестування нефункціональних вимог охоплювало перевірку продуктивності, надійності та сумісності системи.

Перевірка продуктивності проводилась шляхом вимірювання часу відповіді для різних типів запитів. Запити автентифікації, отримання профілю та списку мов виконувались менш ніж за 200 мілісекунд, що відповідає вимозі НФВ-01 (не більше 500 мс). Час обробки аудіофайлу тривалістю 30 секунд становив близько 15–20

секунд, що є прийнятним для даного типу операцій. Час перекладу SRT-файлу з 10 сегментами становив близько 3–5 секунд.

Перевірка надійності проводилась шляхом симуляції недоступності зовнішніх компонентів. При відключенні DeepL API (видаленні ключа з .env) система коректно повертала HTTP 503 з повідомленням про недоступність сервісу перекладу, не падаючи повністю. При відключенні бази даних сервер продовжував запускатись, але повертав помилку при спробі автентифікації. Ендпоінт перевірки стану коректно відображав недоступні компоненти.

Перевірка сумісності проводилась у трьох браузерях: Google Chrome версії 120, Mozilla Firefox версії 121 та Microsoft Edge версії 120. В усіх браузерах інтерфейс відображався коректно, drag&drop завантаження файлів працювало без помилок, а всі функції системи були доступні без встановлення додаткових плагінів.

4.3.6 Виявлені недоліки та їх усунення

За результатами тестування виявлено та усунуто чотири недоліки, що впливали на коректність роботи системи.

Перший недолік стосувався некоректної обробки файлів з кириличними символами в імені. При збереженні файлу з кириличним іменем у папці завантажень виникала помилка кодування на деяких операційних системах. Недолік усунуто шляхом переходу на UUID-іменування при збереженні файлів - оригінальне ім'я зберігається лише в базі даних і не використовується як ім'я файлу на диску.

Другий недолік полягав у відсутності перевірки доступності FFmpeg перед обробкою відеофайлів. При спробі обробити відеофайл на системі без встановленого FFmpeg виникало технічне виключення Python замість зрозумілого повідомлення про помилку. Недолік усунуто шляхом додавання попередньої перевірки доступності FFmpeg з поверненням HTTP 503 та інструкцією щодо встановлення.

Третій недолік стосувався некоректного повідомлення про помилку при перевищенні ліміту DeepL API. При вичерпанні безкоштовного ліміту символів DeepL повертав специфічне виключення, яке не оброблялось окремо і призводило до HTTP 500 з технічним повідомленням. Недолік усунуто шляхом додавання обробки відповідного виключення з поверненням HTTP 503 та зрозумілим повідомленням про вичерпання ліміту API.

Четвертий недолік полягав у відсутності логування при першому завантаженні моделі Whisper. Оскільки завантаження моделі займає кілька секунд, користувач не отримував жодної інформації про стан обробки в цей час. Недолік усунуто шляхом додавання відповідних записів у лог при ініціалізації моделі, що дозволяє адміністратору відстежувати цей процес у консолі сервера.

Усі виявлені недоліки усунуто до завершення розробки. Система успішно пройшла повторне тестування після внесення виправлень. Ендпоінт перевірки стану підтверджує доступність усіх компонентів системи в штатному режимі роботи.

ВИСНОВКИ

У результаті виконання бакалаврської роботи розроблено веб-систему автоматизованого перекладу мультимедійного контенту, що забезпечує єдиний інструмент для обробки аудіо-, відео-, графічних файлів та файлів субтитрів без необхідності залучення кількох спеціалізованих сервісів.

На основі аналізу існуючих підходів та рішень у сфері автоматизованого перекладу мультимедіа обґрунтовано використання клієнт-серверної архітектури з програмним інтерфейсом на основі принципів REST. Таке рішення дозволяє виконувати ресурсомісткі операції транскрипції та розпізнавання на стороні сервера незалежно від обчислювальних можливостей пристрою користувача, а також централізовано управляти інтеграцією з зовнішніми сервісами. Для захисту даних застосовано двоетапне хешування паролів через SHA-256 та bcrypt, підписані токени сеансу з обмеженим терміном дії, параметризацію SQL-запитів та унікальне іменування файлів, що разом унеможливляють несанкціонований доступ до персональних даних.

За результатами порівняльного аналізу програмних засобів для реалізації системи обрано Python 3.10 та FastAPI для серверної частини, PostgreSQL як систему зберігання даних, нейромережеву модель Whisper для транскрипції мовлення, Tesseract OCR для розпізнавання тексту на зображеннях, FFmpeg для обробки відеофайлів та DeepL як сервіс нейромережевого перекладу. Вибір кожного компонента обґрунтовано порівняльним аналізом доступних альтернатив за критеріями підтримки мов, наявності відкритого коду, можливості локального запуску та якості результату.

У межах роботи розроблено програмну систему, що включає серверний модуль з вісьмома маршрутами програмного інтерфейсу, чотириступеневий конвеєр обробки файлів з трьома спеціалізованими гілками (транскрипція, оптичне розпізнавання, синтаксичний розбір субтитрів), підсистему автентифікації з двоетапним хешуванням та токенами сеансу, реляційну базу даних з чотирма таблицями та п'ятьма індексами, односторінковий клієнтський застосунок з

двостановою архітектурою, а також окремий компонент інструментів адміністрування.

Проведено тестування розробленого програмного забезпечення за функціональними сценаріями, що охоплюють автентифікацію, обробку файлів семи форматів, механізм кешування та захист від несанкціонованого доступу. Усі перевірки виконано успішно. Тестування нефункціональних характеристик підтвердило відповідність встановленим вимогам до продуктивності та сумісності з трьома сучасними браузерами. Виявлено та усунуто чотири недоліки, що виникли в ході тестування.

Практична цінність роботи полягає у створенні готового до розгортання застосунку з відкритим кодом, що дозволяє перекладачам, науковцям, журналістам та іншим фахівцям автоматизувати обробку різнотипного мультимедійного контенту без технічних знань і без залежності від платних сервісів. Система забезпечує конфіденційність завантажених матеріалів завдяки локальному виконанню транскрипції та розпізнавання, зберігає персональний журнал перекладів та підтримує кешування повторних запитів для економії ресурсів.

Перспективними напрямками подальшого вдосконалення системи є реалізація сповіщень про прогрес обробки в реальному часі через WebSocket для інформування користувача під час тривалих операцій; додавання підтримки пакетної обробки кількох файлів одночасно; впровадження попередньої обробки зображень для підвищення точності оптичного розпізнавання; розширення переліку підтримуваних форматів файлів, зокрема DOCX та PDF; реалізація обмежень частоти запитів для захисту від зловживань; інтеграція з додатковими сервісами машинного перекладу для збільшення переліку підтримуваних мов.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Géron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. — 3rd ed. — Sebastopol: O'Reilly Media, 2022. 861 с.
2. Goodfellow I., Bengio Y., Courville A. Deep Learning. — Cambridge: MIT Press, 2016. 800 с.
3. Raschka S., Liu Y., Mirjalili V. Machine Learning with PyTorch and Scikit-Learn. — Birmingham: Packt Publishing, 2022. — 770 p.
4. Beazley D., Jones B. K. Python Cookbook. — 3rd ed. — Sebastopol: O'Reilly Media, 2013. 706 с.
5. Lutz M. Learning Python. — 5th ed. — Sebastopol: O'Reilly Media, 2013. 1594 с.
6. Percival H., Gregory B. Architecture Patterns with Python. — Sebastopol: O'Reilly Media, 2020. 280 с.
7. Kleppmann M. Designing Data-Intensive Applications. — Sebastopol: O'Reilly Media, 2017. 611 с.
8. Fowler M. Patterns of Enterprise Application Architecture. — Boston: Addison-Wesley, 2002. 560 с.
9. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. — Upper Saddle River: Prentice Hall, 2017. 432 с.
10. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. — Reading: Addison-Wesley, 1994. 395 с.
11. Momjian B. PostgreSQL: Introduction and Concepts. — New York: Addison-Wesley, 2001. 461 с.
12. Obe R. O., Hsu L. S. PostgreSQL: Up and Running. — 3rd ed. — Sebastopol: O'Reilly Media, 2017. 334 с.
13. Jurafsky D., Martin J. H. Speech and Language Processing. — 3rd ed. draft. — Stanford University, 2023. 654 с.
14. Manning C. D., Schütze H. Foundations of Statistical Natural Language Processing. — Cambridge: MIT Press, 1999. 680 с.

15. Radford A. et al. Robust Speech Recognition via Large-Scale Weak Supervision. — Technical Report, OpenAI, 2022. URL: <https://arxiv.org/abs/2212.04356>
16. Smith R. An Overview of the Tesseract OCR Engine. — Proceedings of the Ninth International Conference on Document Analysis and Recognition. — IEEE, 2007. C. 629–633.
17. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures: PhD Dissertation. — University of California, Irvine, 2000. 162 с.
18. Provos N., Mazières D. A Future-Adaptable Password Scheme. — Proceedings of the USENIX Annual Technical Conference, 1999. — с. 81–91.
19. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). RFC 7519. — Internet Engineering Task Force, 2015. URL: <https://datatracker.ietf.org/doc/html/rfc7519>
20. Wiggins A. The Twelve-Factor App. URL: <https://12factor.net/>
21. Тихоход В., Гурін А., Беспала О. Технології розробки програмного забезпечення. Курс лекцій : навч. посіб. (електрон.). Київ : КПІ ім. Ігоря Сікорського, 2024. 230 с.

ДОДАТОК А ПРОГРАМНИЙ КОД СИСТЕМИ

A.1 Конфігурація та ініціалізація серверного застосунку (main_auth.py)

```
app = FastAPI(title="Multimedia Translation MVP")
app.add_middleware(
    CORSMiddleware
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"]
    allow_headers=["*"],
)
app.mount("/static", StaticFiles(directory="."), name="static")
security = HTTPBearer()
@app.on_event("startup")
async def startup_event():
    init_db_pool()
    logger.info("Database pool initialized")
```

A.2 Функції хешування та перевірки пароля (main_auth.py)

```
def hash_password(password: str) -> str:

    """Двоетапне хешування: SHA-256 для нормалізації + bcrypt для захисту"""
    pwd_hash = hashlib.sha256(password.encode('utf-8')).hexdigest()
    bcrypt_hash = bcrypt.hashpw(pwd_hash.encode('utf-8'), bcrypt.gensalt())
    return bcrypt_hash.decode('utf-8')

def verify_password(plain: str, hashed: str) -> bool:

    """Перевірка пароля за тією самою двоетапною схемою"""
    pwd_hash = hashlib.sha256(plain.encode('utf-8')).hexdigest()
    return bcrypt.checkpw(pwd_hash.encode('utf-8'), hashed.encode('utf-8'))
```

A.3 Функція транскрипції аудіо з відкладеним завантаженням моделі (main_auth.py)

```

whisper = None # ініціалізується при першому зверненні
def transcribe_audio(audio_path: str) -> tuple[str, str]:

    """Транскрипція аудіо через Whisper з відкладеним завантаженням"""
    global whisper
    if whisper is None:
        import whisper as whisper_module
        whisper = whisper_module
        logger.info("Whisper model loaded")
    model = whisper.load_model("base")
    result = model.transcribe(audio_path)
    return result["text"], result.get("language", "unknown")

```

A.4 Обробник запиту на переклад файлу (main_auth.py)

```

@app.post("/api/translate")
async def translate(
    file: UploadFile = File(...),
    target_language: str = Form(...),
    current_user: dict = Depends(get_current_user)
):

    ext = get_file_extension(file.filename)
    if not (is_audio_file(ext) or is_image_file(ext) or is_srt_file(ext)):
        raise HTTPException(status_code=400, detail="Unsupported file type")
    upload_path = UPLOAD_DIR / f"{uuid.uuid4()}_{file.filename}"
    with open(upload_path, "wb") as f:
        content = await file.read()
        f.write(content)

```

```

cached = get_cached_translation
    current_user['user_id'], file.filename, target_language
)

if cached and (OUTPUT_DIR / cached['output_filename']).exists():
    upload_path.unlink()
    return {
        "download_url": f"/download/{cached['output_filename']}",
        "cached": True
    }
result = process_file(str(upload_path), target_language)
save_translation_history(current_user['user_id'], result)
upload_path.unlink()
return {"download_url": result['download_url'], "cached": False}

```

A.5 Схема бази даних (db/init.sql)

```

CREATE TABLE users (
    id          SERIAL PRIMARY KEY,
    username    VARCHAR(50) UNIQUE NOT NULL,
    email       VARCHAR(120) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE translation_history
    id          SERIAL PRIMARY KEY,
    user_id     INTEGER NOT NULL
                REFERENCES users(id) ON DELETE CASCADE,
    original_filename VARCHAR(255) NOT NULL,
    file_type   VARCHAR(20) NOT NULL,

```

```

source_language    VARCHAR(10),
target_language    VARCHAR(10) NOT NULL,
output_filename    VARCHAR(255) NOT NULL,
segment_count      INTEGER,
processing_time_seconds  FLOAT,
status             VARCHAR(20) DEFAULT 'completed',
created_at         TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE sessions (
    id      SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    token_hash VARCHAR(255) NOT NULL,
    expires_at TIMESTAMP NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_users_username
    ON users(username);
CREATE INDEX idx_users_email
    ON users(email);
CREATE INDEX idx_translation_history_user_id
    ON translation_history(user_id);
CREATE INDEX idx_translation_history_created_at
    ON translation_history(created_at DESC);
CREATE INDEX idx_translation_history_status
    ON translation_history(status);

```

A.6 Функції роботи з базою даних — шар репозиторію (main_auth.py)

def get_user_by_username(username: str) -> dict | None:

```

    """Отримання користувача за логіном для автентифікації"""
    conn = get_db()
    try
        cursor = conn.cursor(cursor_factory=RealDictCursor)
        cursor.execute(
            "SELECT id, username, email, password_hash, created_at "
            "FROM users WHERE username = %s",
            (username,)
        )
        return cursor.fetchone()
    except psycopg2.Error as e:
        logger.error(f"DB error in get_user_by_username: {e}")
        raise
    finally:
        return_db(conn)

```

def save_user(username: str, email: str, password_hash: str) -> int:

```

    """Збереження нового облікового запису"""
    conn = get_db()
    try:
        cursor = conn.cursor()
        cursor.execute(
            "INSERT INTO users (username, email, password_hash) "
            "VALUES (%s, %s, %s) RETURNING id",
            (username, email, password_hash)
        )

```

```

        user_id = cursor.fetchone()[0]
        conn.commit()
        return user_id
    except psycopg2.Error as e:
        conn.rollback()
        raise
    finally:
        return_db(conn)

def save_translation_history(user_id: int, file_info: dict) -> int:

    """Збереження запису в журнал перекладів"""
    conn = get_db()
    try:
        cursor = conn.cursor()
        cursor.execute
        """INSERT INTO translation_history
        (user_id, original_filename, file_type, source_language,
        target_language, output_filename, segment_count,
        processing_time_seconds, status)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
        RETURNING id""",
        (
            user_id,
            file_info.get('original_filename'),
            file_info.get('file_type'),
            file_info.get('source_language'),
            file_info.get('target_language'),
            file_info.get('output_filename'),
            file_info.get('segment_count'),
            file_info.get('processing_time'),

```

```

        'completed'
    )
)
record_id = cursor.fetchone()[0]
conn.commit()
return record_id
except psycopg2.Error as e:
    conn.rollback()
    raise
finally:
    return_db(conn)

def get_cached_translation(user_id: int, filename: str,
                           target_language: str) -> dict | None:

    """Пошук кешованого результату перед запуском конвєсра"""
    conn = get_db()
    try:
        cursor = conn.cursor(cursor_factory=RealDictCursor)
        cursor.execute(
            """SELECT id, output_filename
            FROM translation_history
            WHERE user_id = %s
            AND original_filename = %
            AND target_language = %s
            AND status = 'completed'
            ORDER BY created_at DESC
            LIMIT 1""",
            (user_id, filename, target_language)

        )

```

```

        return cursor.fetchone()
    finally:
        return_db(conn)
def get_user_translations(user_id: int, limit: int = 50) -> list:

    """Отримання журналу перекладів поточного користувача"""
    conn = get_db()
    try:
        cursor = conn.cursor(cursor_factory=RealDictCursor)
        cursor.execute
        """SELECT id, original_filename, file_type, source_language,
            target_language, output_filename, segment_count,
            processing_time_seconds, status, created_at
        FROM translation_histor
        WHERE user_id = %s
        ORDER BY created_at DESC
        LIMIT %s""",
        (user_id, limit)
    )
    return cursor.fetchall()
    finally:
        return_db(conn)

```

A.7 Функція витягування аудіодоріжки з відеофайлу (main_auth.py)

```

def extract_audio_from_video(video_path: str) -> str:
    """Витягування аудіодоріжки через FFmpeg як зовнішній процес"""
    output_audio = str(video_path) + ".wav"
    try:
        subprocess.run(

```

```

["ffmpeg", "-i", video_path
 "-vn",      # без відеопотоку
 "-acodec", "pcm_s16le", # PCM 16-bit
 "-ar", "16000",      # частота дискретизації 16 кГц
 "-ac", "1",          # моно
 "-n",              # не перезаписувати існуючі файли
 output_audio],
check=True
capture_output=True,
timeout=30
)

logger.info(f"Audio extracted from video: {output_audio}")
return output_audio
except subprocess.TimeoutExpired:
    raise HTTPException(
        status_code=408,
        detail="Video processing timeout exceeded (30s)"
    )

except subprocess.CalledProcessError as e:
    raise HTTPException(
        status_code=500,
        detail=f"FFmpeg error: {e.stderr.decode()}"
    )

```

A.8 Функція розпізнавання тексту на зображенні (main_auth.py)

```
def ocr_image(image_path: str) -> tuple[str, str]:
```

```

"""Розпізнавання тексту на зображенні через Tesseract OCR"""
try:
    pytesseract.pytesseract.get_tesseract_version()
except Exception:
    raise HTTPException(
        status_code=503,
        detail="Tesseract OCR is not installed or not found in PATH. "
        "Install: https://tesseract-ocr.github.io/tessdoc/Installation.html"
    )
try:
    img = Image.open(image_path)
    text = pytesseract.image_to_string(img)
    if not text.strip():
        raise HTTPException(
            status_code=422,
            detail="No text detected in the image"
        )
    logger.info(f"OCR extracted {len(text)} characters from {image_path}")
    return text, "unknown"
except HTTPException:
    raise
except Exception as e:
    raise HTTPException(
        status_code=500,
        detail=f"OCR processing error: {str(e)}"
    )

```

A.9 Функція розбору та збирання SRT-файлу (main_auth.py)

```
import re
```

```

def parse_srt(srt_path: str) -> tuple[list[dict], str]:
    """Синтаксичний розбір SRT-файлу на окремі сегменти"""
    with open(srt_path, "r", encoding="utf-8") as f:
        content = f.read()

    pattern = (
        r"(\d+)\n"
        r"(\d{2}:\d{2}:\d{2},\d{3})"
        r"--> "
        r"(\d{2}:\d{2}:\d{2},\d{3})\n"
        r"(.*)?(?=\n\n|\Z)"
    )

    matches = re.findall(pattern, content, re.DOTALL)
    segments = [
        {
            "index": m[0],
            "start": m[1],
            "end": m[2],
            "text": m[3].strip()
        }
        for m in matches
    ]

    logger.info(f"Parsed {len(segments)} SRT segments from {srt_path}")
    return segments, "unknown"

def build_srt(segments: list[dict]) -> str:

    """Збирання SRT-файлу зі збереженням оригінальних часових міток"""
    lines = []

```

```

for seg in segments:
    lines.append(seg["index"])
    lines.append(f'{seg["start"]} --> {seg["end"]}')
    lines.append(seg["text"])
    lines.append("") # порожній рядок-роздільник
return "\n".join(lines)

```

A.10 Функція реєстрації нового облікового запису (main_auth.py)

```

@app.post("/api/auth/register")
async def register(user_data: UserRegister):
    """Реєстрація нового користувача з валідацією та збереженням"""
    # перевірка унікальності логіна
    existing = get_user_by_username(user_data.username)
    if existing:
        raise HTTPException(
            status_code=400,
            detail="Username already exists"
        )
    # перевірка унікальності email
    conn = get_db()
    try:
        cursor = conn.cursor()
        cursor.execute(
            "SELECT id FROM users WHERE email = %s",
            (user_data.email,)
        )
        if cursor.fetchone():
            raise HTTPException(
                status_code=400,

```

```

        detail="Email already registered"
    )
finally:
    return_db(conn)
password_hash = hash_password(user_data.password)
user_id = save_user(user_data.username, user_data.email, password_hash)
token = create_jwt_token(user_id, user_data.username)
logger.info(f"New user registered: {user_data.username} (id={user_id})")
return TokenResponse(
    access_token=token,
    token_type="bearer",
    user_id=user_id,
    username=user_data.username
)

```

A.11 Ендпоінт перевірки стану компонентів системи (main_auth.py)

```

@app.get("/api/health")
async def health_check():
    """Перевірка доступності всіх компонентів системи"""
    # перевірка Whisper
    whisper_ok = False
    try:
        import importlib
        importlib.util.find_spec("whisper")
        whisper_ok = True
    except Exception:
        pass
    # перевірка Tesseract
    tesseract_ok = False
    try:

```

```

pytesseract.pytesseract.get_tesseract_version()
tesseract_ok = True
except Exception:
    pass
# перевірка DeepL
deepl_ok = bool(os.getenv("DEEPL_API_KEY"))
# перевірка бази даних
db_ok = False
try:
    conn = get_db()
    cursor = conn.cursor()
    cursor.execute("SELECT 1")
    db_ok = True
    return_db(conn)
except Exception:
    pass
return {
    "status": "ok" if all([whisper_ok, tesseract_ok, deepl_ok, db_ok])
        else "degraded",
    "whisper_available": whisper_ok,
    "tesseract_available": tesseract_ok,
    "deepl_available": deepl_ok,
    "database_available": db_ok
}

```

A.12 Скрипт управління схемою бази даних (db/manage_db.py)

```

import argpars
import psycopg2
import os
from dotenv import load_dotenv

```

```

load_dotenv()
DATABASE_URL = os.getenv("DATABASE_URL")
def init_schema():
    """Ініціалізація схеми з SQL-файлу"""
    conn = psycopg2.connect(DATABASE_URL)
    try:
        with open("db/init.sql", "r", encoding="utf-8") as f:
            sql = f.read()
        cursor = conn.cursor()
        cursor.execute(sql)
        conn.commit()
        print("Schema initialized successfully.")
    except Exception as e:
        conn.rollback()
        print(f"Error: {e}")
    finally:
        conn.close()
def reset_schema():
    """Скидання та повторне створення схеми"""
    conn = psycopg2.connect(DATABASE_URL)
    try:
        cursor = conn.cursor()
        cursor.execute(
            "DROP TABLE IF EXISTS sessions, translation_history, users CASCADE
        )
        conn.commit()
        print("All tables dropped.")
    finally:
        conn.close()

```

```

init_schema()

def check_connection():

    """Перевірка з'єднання з базою даних"""
    try:
        conn = psycopg2.connect(DATABASE_URL)
        cursor = conn.cursor()
        cursor.execute("SELECT version()")
        version = cursor.fetchone()[0]
        print(f"Connection OK. PostgreSQL: {version}")
        conn.close()
    except Exception as e:

        print(f"Connection failed: {e}")

def show_schema():

    """Виведення поточної структури таблиць"""
    conn = psycopg2.connect(DATABASE_URL)
    try:
        cursor = conn.cursor(
            cursor.execute(
                """SELECT table_name, column_name, data_type
                FROM information_schema.columns
                WHERE table_schema = 'public'
                ORDER BY table_name, ordinal_position"""
            )

        current_table = None
        for row in cursor.fetchall():

```

```

    if row[0] != current_table:
        current_table = row[0]
        print(f"\nТаблица: {current_table}")
        print("-" * 40)
        print(f" {row[1]:30s} {row[2]}")
    finally:
        conn.close()

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Database management tool")
    parser.add_argument("--init", action="store_true", help="Initialize schema")
    parser.add_argument("--reset", action="store_true", help="Reset schema")
    parser.add_argument("--check", action="store_true", help="Check connection")
    parser.add_argument("--schema", action="store_true", help="Show schema")
    args = parser.parse_args()
    if args.init: init_schema()
    if args.reset: reset_schema()
    if args.check: check_connection()
    if args.schema: show_schema()

```