МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії
програмного забезпечення

_____ Євген ДАВИДЕНКО

«___» _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ГЕНЕРАЦІЇ ПРОТОКОЛІВ ЗУСТРІЧЕЙ НА ОСНОВІ АУДІОЗАПИСІВ

Спеціальність 121 Інженерія програмного забезпечення
Освітня програма «Інженерія програмного забезпечення»

Здобувач _____ **Дан ФРИЧ**

«__» _____ 2025 р.

Керівник роботи
канд. техн. наук,
доцент _____ **Євген ДАВИДЕНКО**

«__» _____ 2025 р.

Миколаїв – 2025

Завдання на виконання кваліфікаційної роботи

Чорноморський національний університет імені Петра Могили

Факультет	Комп'ютерних наук
Кафедра	Інженерії програмного забезпечення
Рівень вищої освіти	Другий (магістерський)
Освітній ступінь	Магістр
Спеціальність	121 Інженерія програмного забезпечення
Освітня програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
«___» _____ 2025 р.

ЗАВДАННЯ

на кваліфікаційну магістерську роботу здобувача вищої освіти

Фрича Дана

1. Тема кваліфікаційної роботи «Програмне забезпечення генерації протоколів зустрічей на основі аудіозаписів» затверджена наказом ректора ЧНУ ім. Петра Могили №182 від «02» липня 2025 р.
2. Строк представлення кваліфікаційної роботи «22» грудня 2025 р.
3. Очікуваний результат роботи та початкові дані якщо такі потрібні.
4. Перелік питань, що підлягають розробці:

— аналіз предметної області автоматичного створення протоколів;`

— дослідження існуючих методів розпізнавання мовлення та генерації текстів;

— розробка моделі процесу обробки аудіо та формування структурованого протоколу;
— побудува архітектурної моделі програмного забезпечення;
— формування специфікації функціональних і нефункціональних вимог;
— реалізація програмного забезпечення;
— оцінка працездатність розробленого рішення.

5. Перелік графічних матеріалів: презентація
6. Консультанти:

Консультант	**Кафедра (організація)**	**Частина роботи**

Дата видачі завдання «23» червня 2025 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: **Програмне забезпечення генерації протоколів зустрічей на основі аудіозаписів**

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КМР	01.09.2025	08.09.2025	Виконано
2.	Огляд літератури за темою роботи	08.09.2025	19.09.2025	Виконано
3.	Складання календарного плану КМР	08.09.2025	10.09.2025	Виконано
4.	Аналіз предметної області	10.09.2025	26.09.2025	Виконано
5.	Розробка проєктних рішень	29.09.2025	17.10.2025	Виконано
6.	Моделювання та конструювання ПЗ	20.10.2025	03.11.2025	Виконано
7.	Кодування, тестування та апробація розробленого ПЗ, аналіз результатів тестування, розробка керівництва користувача	04.11.2025	17.11.2025	Виконано
8.	Оформлення КМР та презентації	17.11.2025	24.11.2025	Виконано
9.	Попередній захист КРМ	24.11.2025	24.11.2025	Виконано
10.	Завершення оформлення КМР та презентації	25.11.2025	05.12.2025	Виконано
11.	Відгук керівника КМР	08.12.2025	12.12.2025	Виконано
12.	Рецензування КМР	12.12.2025	15.12.2025	Виконано
13.	Захист кваліфікаційної роботи	22.12.2025	22.12.2025	Виконано

Здобувач _____ **Дан ФРИЧ**

«__» _____ 2025 р.

Керівник роботи

канд. техн. наук,

доцент _____ **Євген ДАВИДЕНКО**

«__» _____ 2025 р.

АНОТАЦІЯ

до кваліфікаційної магістерської роботи

«Програмне забезпечення генерації протоколів зустрічей на основі аудіозаписів»

Здобувач 608 гр.: Фрич Дан

Керівник: канд. техн. наук, доцент Давиденко Євген

Магістерська робота присвячена розвитку методів автоматизованої обробки аудіозаписів зустрічей для формування структурованих протоколів із використанням технологій розпізнавання мовлення та узагальнення текстів.

Актуальність дослідження зумовлена потребою підвищення ефективності документообігу та комунікації в організаціях шляхом автоматизації створення протоколів офлайн-зустрічей. Запропонований підхід дозволяє скоротити час підготовки документів, зменшити людський фактор і забезпечити точність переданої інформації.

Об’єктом дослідження є процес автоматичного створення текстових документів на основі аудіозаписів зустрічей.

Предметом дослідження є методи та алгоритми обробки звукової інформації, розпізнавання мовлення й узагальнення текстів, що використовуються при створенні протоколів.

Метою кваліфікаційної роботи є розвиток методів автоматизованої обробки аудіозаписів зустрічей для формування структурованих протоколів шляхом поєднання технологій розпізнавання мовлення та узагальнення текстів.

Для досягнення поставленої мети у роботі необхідно вирішити такі завдання:

— проаналізувати предметну область автоматичного створення протоколів;`

— дослідити існуючі методи розпізнавання мовлення та генерації текстів;

— розробити модель процесу обробки аудіо та формування структурованого протоколу;

— побудувати архітектурну модель програмного забезпечення;

— сформувати специфікацію функціональних і нефункціональних вимог;

— реалізувати програмне забезпечення та оцінити працездатність розробленого рішення.

Пояснювальна записка магістерської роботи складається зі вступу, чотирьох розділів, висновків, переліку використаних джерел та додатків. У вступі обґрунтовано актуальність теми, визначено мету, завдання, об’єкт і предмет дослідження. У першому розділі проведено аналіз предметної області, розглянуто значення протоколів зустрічей, сучасні технології розпізнавання мовлення, методи обробки природної мови та існуючі наукові й програмні рішення у сфері автоматичного протоколювання.

У другому розділі виконано моделювання програмного забезпечення, зокрема визначено вимоги до системи, побудовано функціональну та інформаційну моделі, а також описано основні сценарії використання платформи.

У третьому розділі розроблено архітектуру та виконано проєктування програмного забезпечення, обґрунтовано вибір технологій і мов програмування, а також визначено структуру бази даних і програмну архітектуру платформи.

У четвертому розділі описано програмну реалізацію вебзастосунку, включаючи підготовку інфраструктури, реалізацію серверної та клієнтської частин, інтеграцію з сервісами штучного інтелекту та тестування програмного забезпечення.У першому розділі проведено аналіз предметної області та сучасних технологій автоматичної обробки мовлення. У другому розділі зроблено моделювання програмного забезпечення. У третьому розділі описано реалізацію програмного забезпечення застосунку. У четвертому розділі розглянуто питання тестування та аналізу отриманих результатів.

Кваліфікаційна робота викладена на 101 сторінці машинописного тексту, складається із вступу, 4 розділів, загальних висновків, переліку джерел посилання з 40 найменувань та 2 додатків. Праця містить 3 таблиці та 21 рисунок.

Ключові слова: автоматизація, протокол зустрічі, розпізнавання мовлення, обробка природної мови, штучний інтелект, вебзастосунок, FastAPI, Angular.

ABSTRACT

to the qualifying master's thesis

«Software for generating meeting minutes based on audio recordings»

Student of 608 group: Frych Dan

Supervisor: Candidate of Technical Sciences (Ph. D.), Associate Professor

Davydenko Y. O

The master's thesis is devoted to the development of methods for automated processing of audio recordings of meetings to form structured protocols using speech recognition and text summarization technologies.

The relevance of the research is determined by the need to improve the efficiency of document flow and communication in organizations by automating the creation of offline meeting minutes. The proposed approach allows reducing the time required for document preparation, minimizing the human factor, and ensuring the accuracy of the information transmitted.

The object of the research is the process of automatic creation of text documents based on audio recordings of meetings.

The subject of the study is the methods and algorithms for processing audio information, speech recognition, and text summarization used in the creation of minutes.

The purpose of the qualification work is to develop methods for the automated processing of audio recordings of meetings to form structured minutes by combining speech recognition and text summarization technologies.

To achieve this goal, the following tasks must be solved in the work:

— analyze the subject area of automatic protocol creation;

— research existing methods of speech recognition and text generation;

— develop a model of the audio processing and structured protocol formation process;

— build an architectural model of the software;

— formulate specifications for functional and non-functional requirements;

— implement the software and evaluate the performance of the developed solution.

The explanatory note of the master's thesis consists of an introduction, four chapters, conclusions, a list of references, and appendices. The introduction justifies the relevance of the topic and defines the purpose, objectives, object, and subject of the study. The first chapter analyzes the subject area, considers the importance of meeting minutes, modern speech recognition technologies, natural language processing methods, and existing scientific and software solutions in the field of automatic minute-taking.

The second chapter models the software, specifically defining the system requirements, building functional and information models, and describing the main scenarios for using the platform.

The third chapter develops the architecture and designs the software, justifies the choice of technologies and programming languages, and defines the database structure and software architecture of the platform.

The fourth chapter describes the software implementation of the web application, including infrastructure preparation, implementation of the server and client parts, integration with artificial intelligence services, and software testing. The first chapter analyzes the subject area and modern technologies for automatic speech processing. The second chapter models the software. The third chapter describes the implementation of the application software. The fourth chapter discusses testing and analysis of the results obtained.

The thesis is presented on 101 pages of typed text and consists of an introduction, 4 chapters, general conclusions, a list of 40 references, and 2 appendices. The work contains 3 tables and 21 figures.

Keywords: automation, meeting protocol, speech recognition, natural language processing, artificial intelligence, web application, FastAPI, Angular.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД — база даних

ПЗ — програмне забезпечення

ТЗ — технічне завдання

ШІ — штучний інтелект

API — Application Programming Interface,

ASR — Automatic Speech Recognition

CRUD — Create, Read, Update, Delete

DFD — Data Flow Diagram

DTO — Data Transfer Object

ERD — Entity-Relationship Diagram

HTTP — HyperText Transfer Protocol

HTTPS — HyperText Transfer Protocol Secure

JSON — JavaScript Object Notation

JWT — JSON Web Token

LLM — Large Language Model

ORM — Object-Relational Mapping

REST — Representational State Transfer

SQL — Structured Query Language

UI — User Interface

UUID — Universally Unique Identifier

ВСТУП

Сучасні організації, підприємства та навчальні заклади проводять значну кількість зустрічей, нарад і робочих обговорень, результати яких необхідно фіксувати у вигляді протоколів або звітів.

Процес підготовки таких документів традиційно вимагає значних часових і людських ресурсів, оскільки здійснюється вручну. Особливо складним є створення протоколів на основі офлайн-зустрічей, де немає автоматичних інструментів фіксації та розпізнавання мовлення. Це зумовлює потребу у розробленні програмних засобів, здатних автоматизувати перетворення аудіозаписів у структуровані текстові документи.

Актуальність теми полягає у необхідності підвищення ефективності документообігу та комунікації в організаціях за рахунок автоматизації створення протоколів.

Використання технологій розпізнавання мовлення, обробки природної мови та штучного інтелекту відкриває нові можливості для оптимізації цього процесу. Програмне забезпечення генерації протоколів зустрічей на основі аудіозаписів зможе зменшити витрати часу на підготовку документів, мінімізувати помилки та забезпечити об’єктивність зафіксованої інформації.

Метою кваліфікаційної роботи є розвиток методів автоматизованої обробки аудіозаписів зустрічей для формування структурованих протоколів шляхом поєднання технологій розпізнавання мовлення та узагальнення текстів.

Для досягнення поставленої мети у роботі необхідно вирішити такі *завдання*:

— проаналізувати предметну область автоматичного створення протоколів;`

— дослідити існуючі методи розпізнавання мовлення та генерації текстів;

— розробити модель процесу обробки аудіо та формування структурованого протоколу;

— побудувати архітектурну модель програмного забезпечення;

— сформувати специфікацію функціональних і нефункціональних вимог;

— реалізувати програмне забезпечення та оцінити працездатність розробленого рішення.

Об'єктом дослідження є процес автоматичного створення текстових документів на основі аудіозаписів зустрічей.

Предметом дослідження є методи та алгоритми обробки звукової інформації, розпізнавання мовлення й узагальнення текстів, що використовуються при створенні протоколів.

У роботі застосовуються такі методи дослідження: системний аналіз, методи обробки аудіоінформації, алгоритми машинного навчання та мовних моделей, а також методи структурного проєктування програмного забезпечення.

Практична цінність полягає у можливості використання розробленого програмного забезпечення для автоматизації документообігу у громадських організаціях, бізнес-структурах, навчальних закладах та інших установах, де регулярно проводяться зустрічі й наради.

Апробація результатів КМР відбулась під час XXVII Всеукраїнської науково-практичної конференції «Могилянські читання 2025», Миколаїв, 10-14 листопада, 2025 р. [40] (Додаток А).

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Значення протоколів зустрічей у сучасних організаціях

Управління проєктами та операційна діяльність будь-якої організації значною мірою спираються на коректну фіксацію результатів нарад, засідань і робочих зустрічей. Протокол зустрічі – це офіційний документ, який відображає склад учасників, порядок денний, ключові обговорення, прийняті рішення та визначених відповідальних осіб. Наявність структурованих протоколів є необхідною умовою збереження логіки ухвалення рішень, забезпечення прозорості комунікацій і можливості контролю за виконанням поставлених завдань[9].

У практиці ручного протоколювання часто спостерігаються типові проблеми, які знижують ефективність цього процесу. До них належать суб’єктивність викладу, ризик втрати важливих деталей під час інтенсивних дискусій, неоднорідність стилю документів та значні часові витрати на підготовку остаточного тексту. Особливо це характерно для офлайн-зустрічей, які проходять у динамічному форматі – конференції, круглі столи, семінари, робочі групи, де кількість спікерів і обсяг інформації не дозволяють одному секретарю або модератору повноцінно зафіксувати всі деталі. Після завершення таких подій учасникам доводиться вручну прослуховувати аудіозаписи, вибирати основні тези та заново оформлювати їх у документ [37, 38].

Проблема ускладнюється тим, що, на відміну від онлайн-платформ, офлайн-зустрічі не мають вбудованих інструментів автоматичного протоколювання. У той час як Zoom, Microsoft Teams чи Google Meet вже пропонують базову транскрипцію і створення нотаток, офлайн-події досі потребують ручного підходу. Тому питання автоматизації фіксації змісту саме офлайн-зустрічей є надзвичайно актуальним [5].

Ситуація особливо загострилася після пандемії COVID-19 та початку повномасштабного вторгнення в Україну. Ці події суттєво змінили формати

комунікації – частина зустрічей перейшла в онлайн, але велика кількість важливих заходів (зокрема публічні форуми, освітні тренінги, стратегічні сесії, консультації з партнерами чи громадами) проводиться в офлайн або гібридному форматі [15, 17]. У таких умовах виникає потреба у системах, які можуть використовувати зовнішні мікрофони для конференцій або диктофони, автоматично розпізнавати мовлення під час події та створювати структурований протокол.

Автоматизація цього процесу базується на поєднанні двох ключових технологічних компонентів: автоматичного розпізнавання мовлення (ASR) та методів обробки природної мови (NLP). Перша технологія забезпечує перетворення аудіосигналу в текстову форму, друга – структуризацію, узагальнення та оформлення змісту у вигляді логічно зв’язаного документа [10]. Такий підхід дає змогу суттєво зменшити навантаження на організаторів заходів, скоротити час на підготовку протоколів і підвищити точність відтворення змісту обговорень.

Типовий процес формування протоколу в автоматизованій системі передбачає кілька послідовних етапів. Спочатку здійснюється запис зустрічі за допомогою конференційного мікрофона або іншого пристрою збору аудіо. Далі проводиться попередня обробка сигналу, що включає зменшення фонового шуму, вирівнювання гучності та виділення мовних сегментів. Після цього відбувається розпізнавання мовлення за допомогою нейромережевих моделей, у результаті чого формується текстова транскрипція. Наступним кроком є очищення тексту від повторів, нерелевантних фраз і мовних паразитів, після чого система виконує ідентифікацію ключових тем та узагальнення змісту. Завершальний етап – формування структурованого документа, у якому автоматично виокремлюються пункти порядку денного, обговорення, рішення й відповідальні особи.

Розроблення подібного програмного забезпечення дає змогу об’єднати переваги офлайн-та онлайн-режимів. У майбутньому система може працювати як локальний інструмент для конференцій і нарад, так і як модуль інтеграції з

онлайн-платформами, що дозволить застосовувати її для гібридних форматів взаємодії [22].

Крім технічних переваг, автоматизація протоколювання має значне управлінське значення. Вона забезпечує єдиний формат ведення документації, підвищує прозорість ухвалення рішень, спрощує звітність і створює внутрішню базу знань організації. Протоколи, створені автоматично, можна швидко аналізувати, індексувати за темами, повертатися до окремих обговорень і використовувати як навчальні або аналітичні матеріали [9].

Таким чином, протокол зустрічі розглядається не лише як документ, а як важливий елемент інформаційно-комунікаційної системи управління. Використання технологій розпізнавання мовлення та обробки природної мови відкриває можливості для підвищення ефективності роботи організацій, особливо у сфері офлайн- та гібридних подій. Це визначає предмет дослідження даної роботи – створення програмного забезпечення для автоматичної генерації протоколів зустрічей на основі аудіозаписів, що поєднує точність, зручність і гнучкість використання в різних форматах комунікації.

1.2 Технології розпізнавання мовлення

Розпізнавання мовлення є одним із ключових напрямів розвитку сучасних інтелектуальних систем та невід’ємною складовою більшості продуктів, що працюють із людською комунікацією. Суть цієї технології полягає у перетворенні акустичного сигналу, який містить мовлення людини, у структурований текст, придатний для подальшої комп’ютерної обробки. Системи автоматичного розпізнавання мовлення (Automatic Speech Recognition, ASR) широко застосовуються у голосових асистентах, сервісах субтитрування, системах перекладу в реальному часі, медичних та юридичних стенографах, а також у програмному забезпеченні, що автоматично формує протоколи зустрічей [7]. У контексті даної роботи саме ASR-технології забезпечують первинний, критично важливий етап – перехід від аудіозапису

заходів до точного текстового представлення, яке надалі узагальнюється та структурується у вигляді протоколу.

Складність задачі розпізнавання мовлення зумовлена природою усної комунікації. Мовлення є непостійним, різнорідним і має численні варіації: акценти, інтонаційні відтінки, різну швидкість вимови, редукції звуків, слова-паразити, самоповтори, паралельні репліки кількох учасників. Додатково на аудіосигнал впливають зовнішні фактори – фоновий шум, реверберація приміщення, якість мікрофона та відстань до джерела звуку. Усе це робить задачу перетворення аудіо на текст нетривіальною й вимагає складних алгоритмів, здатних адаптуватися до умов реального середовища, що особливо актуально для зустрічей, семінарів чи конференцій.

Процес автоматичного розпізнавання мовлення традиційно включає кілька етапів. На першому етапі виконується попередня обробка аудіосигналу – шумоприглушення, нормалізація гучності, видалення статичних перешкод, фільтрація зайвих частот. Важливою складовою є виявлення мовленнєвої активності (Voice Activity Detection, VAD), коли система визначає, де людина говорить, а де присутні тиша або сторонні звуки. Це дозволяє сфокусувати обробку на інформативних ділянках.

Далі сигнал розбивається на короткі фрейми тривалістю 20–30 мс. Для кожного фрейму обчислюються акустичні ознаки – числові характеристики, що описують спектральні властивості звуку. Найпоширенішим набором таких ознак є мел-частотні кепстральні коефіцієнти (MFCC), які моделюють звуковий спектр відповідно до психоакустичних особливостей людського слуху: низькі частоти подаються з вищою роздільністю, ніж високі. У результаті кожен фрейм аудіосигналу представляється як вектор параметрів, який слугує компактним «відбитком» сигналу.

У класичних системах ASR ці ознаки подавались на акустичну модель, побудовану на основі прихованих марковських моделей (Hidden Markov Models, HMM) [23] у поєднанні з гаусовими сумішами (Gaussian Mixture Models, GMM). Мовлення розглядалося як послідовність прихованих станів,

що відповідають фонемам або іншим елементарним одиницям, а кожен стан породжує спостережувані ознаки з певною ймовірністю. Паралельно використовувалась мовна модель, яка описувала статистичну структуру мови – ймовірність появи певних послідовностей слів. Формально задачу розпізнавання можна подати як пошук послідовності слів W, яка з найбільшою ймовірністю відповідає вхідному сигналу X:

$$\mathrm{W}^* = \arg\max_{\mathrm{W}} \mathrm{P}(\mathrm{W}|\mathrm{X}) = \arg\max_{\mathrm{W}} \mathrm{P}(\mathrm{W}|\mathrm{X}) \cdot \mathrm{P}(\mathrm{W}),$$

де $P(W|X)$ – це акустична модель (ймовірність того, що слова W породили сигнал X);

$P(W)$ – мовна модель, яка описує статистичну структуру мови.

Попри те, що GMM–HMM системи стали основою перших промислових рішень, вони мали низку суттєвих обмежень: потребу у великих обсягах вручну розмічених даних, обмежену здатність моделювати складні нелінійні залежності між ознаками, відносно низьку стійкість до шуму та до варіативності дикторів. У реальних умовах багатокористувацьких зустрічей їх точність часто виявлялася недостатньою.

Ситуація кардинально змінилася з появою глибинного навчання. Глибокі нейронні мережі (Deep Neural Networks, DNN), а згодом рекурентні архітектури RNN, LSTM і GRU дали змогу значно підвищити точність розпізнавання завдяки здатності моделювати часові залежності у мовленні. Важливим етапом став метод Connectionist Temporal Classification (CTC), який дозволив навчати моделі на довгих аудіозаписах без точного вирівнювання між сигналом і текстом: система отримує лише аудіо та відповідний йому транскрипт, а алгоритм самостійно вчиться зіставляти часові фрагменти із символами чи словами [3]. Це фактично відкрило шлях до end-to-end систем, які без проміжних вручну сконструйованих рівнів напряму перетворюють аудіо у текст.

Наступним кроком розвитку стали трансформерні архітектури (Transformers), що базуються на механізмі самоуваги (self-attention). На

відміну від рекурентних мереж, які обробляють послідовність покроково, трансформери аналізують контекст висловлювання одразу в широкому часовому діапазоні. Це дозволяє моделі враховувати як локальні, так і глобальні залежності у мовленні, підвищуючи точність розпізнавання, особливо для довгих фрагментів аудіо. Трансформери добре масштабуються на великі багатомовні корпуси, що зробило їх основою для більшості сучасних ASR-систем. На базі таких підходів були створені моделі на кшталт wav2vec 2.0 (Meta AI), SpeechBrain, Vosk, open-source-версії Whisper та інші. Вони навчені на сотнях тисяч годин аудіоданих і демонструють рівень Word Error Rate (WER), який для окремих мов та умов наближається до людської точності.

Подальший розвиток ASR у 2024–2025 роках пов’язаний із появою мультимодальних моделей нового покоління, де розпізнавання аудіо інтегроване в ширшу архітектуру, здатну працювати з різними типами даних. Прикладом є моделі сімейства gpt-4o-transcribe, які замінили Whisper як основне хмарне рішення розпізнавання мовлення в екосистемі OpenAI [21]. На відміну від попередніх поколінь, такі моделі демонструють нижчий WER на багатомовних та шумних аудіозаписах, краще працюють із природним розмовним мовленням, підтримують довші аудіофайли та забезпечують коректну пунктуацію і базову діаризацію (визначення зміни дикторів). За результатами відкритих тестувань, gpt-4o-transcribe дає суттєве зменшення помилок порівняно з Whisper-large-v3, що особливо важливо для сценаріїв, де точність розпізнавання прямо впливає на коректність ділової документації [19].

Важливим аспектом сучасних моделей є те, що вони одразу формують текст, який за структурою наближений до природного письмового мовлення: із розставленою пунктуацією, абзацами, інтонаційними паузами. Це значно спрощує подальший етап узагальнення та перетворення транскрипту на протокол, оскільки системі обробки тексту не потрібно додатково відновлювати базову структуру речень. Разом з тим використання хмарних ASR-сервісів має свої обмеження: це платні рішення, які потребують

стабільного доступу до інтернету й інтеграції через API, а також вимагають уваги до питань конфіденційності даних [18].

Задача автоматичного створення протоколів зустрічей ставить до систем розпізнавання мовлення підвищені вимоги. На відміну від, наприклад, субтитрування чи голосових асистентів, де допустима втрата окремих слів, протоколи мають точно передавати ключові формулювання, рішення, формати голосування та позиції учасників. Помилка в одному слові може змінити юридичний або організаційний зміст документа. Тому для таких застосувань особливо важливі низький WER, стійкість до шуму, коректна обробка довгих записів, підтримка потрібної мови (зокрема української) та можливість інтеграції в клієнт–серверні системи через API.

Таким чином, технології розпізнавання мовлення пройшли шлях від класичних статистичних моделей GMM–HMM до складних глибинних трансформерних і мультимодальних архітектур, здатних працювати в реальних умовах і забезпечувати високу точність навіть у складних акустичних сценах [35]. Сучасні ASR-системи стали фундаментальною частиною рішень, що потребують достовірної транскрипції, – зокрема й автоматичного протоколювання зустрічей. У подальших розділах саме на таких моделях базуватиметься програмне забезпечення для перетворення аудіозаписів подій у структуровані текстові протоколи.

1.3 Методи обробки природної мови для генерації протоколів

Після того як система розпізнавання мовлення перетворює аудіозапис зустрічі на текст, цей текст потребує додаткової обробки. Сирий транскрибований текст не може бути використаний як протокол, оскільки він містить численні властивості усного мовлення: слова-паразити, повтори, неповні або еліптичні речення, емоційні вигуки, інтонаційні паузи, а також не має формальної структури. У такому вигляді текст не відображає логіку обговорення, не поділений на змістові блоки та не містить виділених рішень чи відповідальних осіб. Тому на наступному етапі застосовуються методи

обробки природної мови (NLP), які забезпечують нормалізацію тексту, виділення головних змістових фрагментів і побудову стислого, логічно впорядкованого документа [35].

У контексті автоматичного створення протоколів NLP виконує дві ключові функції. Перша – очищення та нормалізація тексту, що дає змогу усунути властивості усного мовлення та привести транскрипт до формату, наближеного до письмової форми. Друга – узагальнення змісту, тобто побудова короткого формального викладу, який містить основні тези та рішення. Сукупність цих функцій утворює повний цикл трансформації необробленої транскрипції у готовий протокол [5,12].

Першим етапом NLP є підготовка транскрипту. Метою цього процесу є усунення зайвих елементів, відновлення логічної структури та вирівнювання тексту до форми, з якою може працювати алгоритм узагальнення. Хоча конкретні методи можуть різнитися залежно від системи, у більшості випадків процес включає такі процедури.

Передусім виконується токенізація – поділ тексту на окремі елементи: слова, символи, знаки пунктуації. Це необхідно для подальших синтаксичних та семантичних аналізів. Після токенізації може виконуватися нормалізація слів, яка спрямована на зниження варіативності формулювань та приведення тексту до більш однорідного стану. Цей етап часто включає перетворення числових або службових конструкцій у стандартизований вигляд.

Наступною важливою процедурою є видалення шумових елементів. Усне мовлення містить значну кількість вставних слів («ну», «тобто», «як би»), повторів, емоційних реакцій, незавершених фраз і корекцій. Усі ці елементи не мають цінності в офіційному документі, тому системи автоматично очищають текст, усуваючи їх. У разі діалогів та нарад до очищення також належить маркування мовців, що створює основу для правильної атрибуції висловлювань. Розподіл тексту між учасниками може виконуватися з використанням алгоритмів діаризації, які розпізнають зміни мовця за акустичними або структурними ознаками.

Після видалення шумів текст проходить синтаксичний аналіз. Цей етап дозволяє визначити структуру речень, встановити зв’язки між словам, виокремити підмет, присудок, об’єкти та обставинні компоненти. Завдяки цьому текст стає більш формалізованим, а наступні алгоритми можуть працювати не лише на рівні слів, а й на рівні смислових зв’язків [35].

Важливою складовою підготовки є також розпізнавання іменованих сутностей (Named Entity Recognition), що включає виявлення імен людей, назв організацій, дат, документів або ключових об’єктів зустрічі. Для протоколювання це має особливе значення, оскільки дозволяє правильно позначати відповідальних осіб та структурувати зміст навколо ключових подій. Завдяки NER система може формувати структуровані фрагменти протоколу, що містять блоки «Учасники», «Питання порядку денного», «Рішення» та інші [36].

У результаті очищення і нормалізації система отримує граматично й структурно впорядкований текст, позбавлений шумів і суперечностей, що дає змогу перейти до етапу узагальнення.

Узагальнення змісту – це процес побудови короткого викладу основних тез обговорення. На відміну від очищення, яке має механічний характер, узагальнення пов’язане з інтерпретацією змісту та визначенням того, які частини тексту є центральними.

У NLP існує два підходи до узагальнення: екстрактивний та абстрактивний [10].

Екстрактивне узагальнення полягає у виборі найбільш значущих фрагментів тексту. Для цього система аналізує частоту ключових слів, визначає важливість речень за статистичними або семантичними критеріями, а також оцінює їх роль у контексті. Такий підхід є простішим, проте має суттєвий недолік: відібрані речення зберігають властивості усного мовлення і часто не утворюють цілісного формального викладу.

Абстрактивне узагальнення є складнішим, оскільки націлене на побудову нового тексту. Система не просто вибирає речення, а формує нові,

більш короткі та структуровані формулювання, які передають сутність дискусії. Абстрактивні методи дозволяють враховувати логіку розвитку обговорення, усувати повтори, змінювати порядок подання інформації та формувати текст відповідно до вимог офіційного стилю [10, 26].

Для реалізації абстрактивного узагальнення застосовуються архітектури нейронних мереж типу Transformer, які працюють за принципом «encoder–decoder». На етапі кодування модель сприймає текст і створює його контекстне представлення у вигляді числових векторів, а на етапі декодування – формує узагальнений текст. До найвідоміших моделей цього типу належать BERT, T5, BART, PEGASUS, а також сучасні великі мовні моделі (Large Language Models, LLM), зокрема GPT-3.5, GPT-4 та інші [16, 33].

У задачі автоматичного створення протоколів NLP виконує специфічну роль: система має не просто скоротити текст, а відтворити логіку обговорення у формальному вигляді. На відміну від класичного узагальнення, метою якого є стислий переказ, протокол вимагає збереження причинно-наслідкових зв’язків, позицій різних учасників, а також фіксації рішень та доручень.

Система повинна:

— правильно розмежовувати змістові блоки зустрічі;

— визначати, які тези є ключовими, а які другорядними;

— встановлювати зв’язок між пропозиціями та рішеннями;

— забезпечувати формальний стиль викладу.

Особливо важливим є контроль глибини узагальнення: якщо текст буде надто деталізованим, це перетвориться на транскрипт; якщо надто стислим – може бути втрачені важливі рішення. Тому алгоритми узагальнення мають адаптуватися до типу зустрічі та бажаного рівня деталізації.

У практичних системах NLP-компоненти організують у вигляді конвеєра: окремі модулі відповідають за очищення тексту, узагальнення, визначення структури документа та форматування. Така архітектура дозволяє масштабувати систему й забезпечує стабільність результатів [11].

Застосування NLP у процесі створення протоколів має вагоме практичне значення. По-перше, автоматизація дозволяє суттєво скоротити час між проведенням зустрічі та отриманням готового документа. Ретельне ручне створення протоколу може займати години, тоді як автоматизована система формує результат практично миттєво. По-друге, NLP зменшує ймовірність пропуску важливої інформації, оскільки алгоритм аналізує весь транскрибований матеріал, не покладаючись на вибіркову увагу людини.

Ще однією перевагою є стандартизація документів: система автоматично формує протоколи в єдиному стилі, що забезпечує їх порівнянність та зручність використання. Крім того, структуровані протоколи можуть автоматично індексуватися, що дозволяє організовувати пошук за темами, прізвищами учасників або ключовими словами, перетворюючи протоколи на інструмент управління знаннями.

Таким чином, методи обробки природної мови є ключовим компонентом систем автоматичного протоколювання. Вони забезпечують повний цикл перетворення транскрипції на зрозумілий, структурований та формально коректний документ, що підвищує ефективність організаційної роботи та якість управлінських процесів.

1.4 Огляд сучасних наукових робіт та програмних рішень у сфері автоматичного протоколювання

Проблематика автоматичного створення протоколів зустрічей активно досліджується вже понад два десятиліття. Перші роботи у цій галузі зосереджувалися на задачах транскрибування та структурування аудіозаписів. Згодом, із розвитком глибинного навчання та великих мовних моделей, фокус змістився до інтегрованих систем, здатних повністю автоматизувати шлях від аудіо до структурованого узагальнення.

Початок систематичних досліджень у сфері автоматичного протоколювання пов’язують із появою спеціалізованих корпусів зустрічей. Одним із перших став ICSI Meeting Corpus (Janin et al., 2003). У ньому зібрано

75 технічних зустрічей загальною тривалістю приблизно 72 години англомовного мовлення, записаного в реальних умовах у Міжнародному комп'ютерному науковому інституті (ICSI, Берклі).

Характерна риса ICSI полягає у багатоканальному записі (головні мікрофони, настільні мікрофони) та природності ситуацій: зустрічі не інсценовані, учасники вільно переривають одне одного, говорять одночасно, змінюють тему. Це створює складні умови для алгоритмів ASR і діаризації, але робить корпус дуже цінним для досліджень. На його основі пізніше було створено додаткові ресурси: корпус діалогових актів MRDA (приблизно ті ж 72 години анотованих діалогових актів) та спеціалізовані анотації «action items», тем і «гарячих моментів» зустрічей [11].

Наступним важливим ресурсом став AMI Meeting Corpus [6]. Цей корпус – багатомодальний: він містить близько 100 годин записів зустрічей, включно з аудіо, відео, записом слайдів, роботи з дошкою, а також детальною анотацією діалогів. Частина зустрічей є природними, але значну частину становлять сценарно організовані наради, де учасники відіграють ролі в команді з розробки умовного продукту; це дає змогу отримувати повторювані сценарії для порівняння алгоритмів.

AMI став базою для великої кількості досліджень:

— сегментації зустрічей на тематичні блоки;

— діаризації (визначення мовців);

— класифікації діалогових актів (запитання, уточнення, пропозиція, підсумок);

— автоматичного узагальнення та формування протоколів.

Як ICSI, так і AMI ввели стандартні форматі анотації, які надалі використовувалися для порівняння систем протоколювання [6, 11].

Подальший розвиток теми відображено у низці оглядових робіт. Один із систематичних оглядів – стаття Zhang та співавт. "Automatic Meeting Summarization: A Survey", де проаналізовано підходи до узагальнення зустрічей: від класичних екстрактивних методів (вибір важливих речень) до

абстрактивних моделей, що генерують новий текст на основі розуміння контексту. Автори звертають увагу, що для реальних нарад критично важливо враховувати: довжину транскриптів, рольову структуру (ведучий, доповідач, учасник), наявність паралельних реплік та необхідність формувати не просто «резюме», а саме структурований протокол із виділеними рішеннями та завданнями.

Починаючи з 2020-х років, усе більше робіт пропонують інтегровані конвеєри, де ланцюжок обробки виглядає як єдина система:

1) аудіо;
2) розпізнавання мовлення;
3) діаризація;
4) очищення й сегментація тексту;
5) абстрактивне узагальнення;
6) формування протоколу заданого формату.

З’являються ієрархічні моделі, які спочатку узагальнюють окремі фрагменти зустрічі (наприклад, блоки по 1–2 хвилини), а потім будують «узагальнення узагальнень», зберігаючи структуру порядку денного.

Окрему хвилю інтересу сформували роботи, пов’язані з великими мовними моделями (LLM). Дослідники показують, що LLM здатні створювати більш зв’язні й «людяні» протоколи, краще виділяти завдання та рішення, але водночас можуть «дописувати» те, чого не було сказано (проблема галюцинацій). Тому в сучасних роботах активно досліджуються методи контролю достовірності: прив’язка до вихідної транскрипції, обмеження на перефразування фактів, використання додаткових систем верифікації [20].

Ще однією тенденцією є поява дослідницьких змагань (наприклад, AutoMin – Automatic Minuting Challenge), де системи оцінюють не лише за якістю резюме (ROUGE тощо), а й за тим, наскільки добре вони формують повноцінні протоколи: чи всі рішення відображені, чи правильно визначено відповідальних, чи витримано структуру документа. Такі конкурси підштовхують розробників до створення практично орієнтованих систем, які

видають результат у форматі, близькому до реальної управлінської документації [32, 35].

Попри великий прогрес, ключова особливість більшості корпусів і моделей – орієнтація на англійську мову. Для української наразі немає настільки ж великих і детально анотованих корпусів реальних нарад, що ускладнює навчання спеціалізованих моделей «з нуля». Це робить актуальним використання багатомовних моделей (ASR і LLM), здатних працювати з українською без повного перенавчання, а також побудову клієнт–серверних рішень, де можна комбінувати різні сервіси розпізнавання й узагальнення [3, 25, 26, 35].

Паралельно з розвитком наукових досліджень у сфері автоматичного протоколювання активно з’являються й прикладні програмні рішення, які пропонують можливість автоматичного створення транскриптів та коротких підсумків зустрічей. Найбільш поширеними є комерційні хмарні сервіси, вбудовані функції у корпоративні платформи та відкриті інструменти, орієнтовані на локальне розгортання. Кожен із цих підходів має свої переваги, обмеження і сфери застосування.

Одним із найвідоміших комерційних продуктів є Otter.ai, який автоматично записує онлайн-зустрічі, створює транскрипти та формує короткі резюме. Сервіс інтегрується з основними платформами відеоконференцій (Zoom, Google Meet, Microsoft Teams) і здатний у реальному часі виділяти ключові тези. Попри високий рівень автоматизації, його можливості обмежені мовною підтримкою: українська станом на 2024–2025 роки відсутня, а сама система повністю залежить від хмарної інфраструктури.

Схожий за ідеологією сервіс Fireflies.ai орієнтований насамперед на корпоративне середовище. Він автоматично виділяє рішення, завдання, ключові висловлювання та дозволяє інтегрувати їх із CRM-системами. Основний недолік аналогічний – відсутність підтримки української мови та неможливість роботи офлайн, що обмежує застосування в середовищах із підвищеними вимогами до конфіденційності.

Такі сервіси, як Trint і Sonix, орієнтовані переважно на журналістику та медіавиробництво. Вони забезпечують інструменти для точного транскрибування інтерв'ю, мають розвинуті веб-редактори з підтримкою тайм-кодів та інструменти спільного редагування. Однак вони також залежать від хмарних серверів, мають високу вартість і не є спеціалізованими рішеннями для автоматичного протоколювання офіційних зустрічей.

У 2023–2024 роках провідні технологічні компанії інтегрували функції автоматичного створення підсумків зустрічей безпосередньо у власні платформи. Microsoft Teams Intelligent Recap формує нотатки, визначає ключові теми, виділяє доручення та структурує матеріал відповідно до логіки зустрічі. Google Meet AI Notes створює короткі резюме та список завдань у форматі Google Docs. Проте обидва рішення працюють виключно в межах відповідних екосистем і не призначені для офлайн-використання, що лишає питання автономності та конфіденційності відкритим.

На відміну від комерційних сервісів, відкриті інструменти на кшталт Vosk, Kaldi, DeepSpeech, SpeechBrain та Whisper орієнтовані на локальне розгортання і можуть бути використані як компоненти власних систем [19, 25, 33]. Їх основні переваги – можливість повної автономності, контроль над даними та підтримка української мови (особливо у Vosk і Whisper)[19, 25]. Whisper, хоча й залишається одним із найточніших відкритих рішень, більше не є основною моделлю для хмарних сервісів OpenAI, поступившись спеціалізованим моделям розпізнавання мовлення нового покоління [21]. Натомість Vosk і Kaldi показують стабільні результати при офлайн-роботі, що робить їх придатними для застосувань, де не допускається передача аудіо до сторонніх серверів.

Окремо варто зазначити появу нових хмарних рішень для розпізнавання мовлення – зокрема моделей gpt-4o-transcribe, які у 2024–2025 роках замінили Whisper API у сервісах OpenAI [21]. Ці моделі пропонують багатомовну підтримку, підвищену точність та можливість автоматичної діаризації, проте

залишаються суто хмарними й не можуть бути розгорнуті локально, що створює ті ж обмеження щодо конфіденційності та автономності.

У таблиці 1.1 представлено узагальнене порівняння поширених рішень за основними критеріями: тип, підтримка української мови, можливість офлайн-роботи, точність розпізнавання та здатність виконувати автоматичне узагальнення або формувати протоколи [3]. Узагальнення точності наведене на основі відкритих тестувань, документації розробників та незалежних оглядів; реальні показники можуть відрізнятися залежно від умов запису.

Таблиця 1.1 – Порівняння програмних рішень

№	Назва системи / моделі	Тип (комерційна / відкрита)	Підтримка української	Офлайн-робота	Word Error Rate, %	Узагальнення / протоколювання	Примітки
1	Otter.ai	Комерційна (SaaS)	Відсутня	Відсутня	10–12	Так	Працює лише онлайн
2	Fireflies.ai	Комерційна (SaaS)	Відсутня	Відсутня	10	Так	Орієнтація на бізнес-зустрічі
3	Microsoft Teams Recap	Корпоративна	Відсутня	Відсутня	8–10	Так	Тісна інтеграція з Office 365
4	Google Meet AI Notes	Корпоративна	Відсутня	Відсутня	8–10	Так	Потребує Google Workspace
5	Trint / Sonix	Комерційна	Відсутня	Відсутня	12–15	Частково	Для журналістики
6	DeepSpeech (Mozilla)	Відкрита	Частково	Так	15	Ні	Застаріла модель
7	Vosk / Kaldi	Відкрита	Так	Так	10–12	Ні	Працює офлайн
8	wav2vec 2.0 (Meta AI)	Відкрита / Research	Так	Так	7–9	Ні	Потребує інтеграції NLP
9	Whisper (OpenAI)	Відкрита	Так	Так	5–7	Ні	Найвища точність, стійкість до шуму
10	gpt-4o-transcribe	Комерційна (API)	Так	Ні	4-7	Частково	Нова модель ASR від OpenAI

Як видно з наведеного порівняння, сучасні програмні рішення переважно орієнтовані на англомовний ринок і тісно інтегровані у власні хмарні екосистеми. Хоча вони забезпечують високий рівень автоматизації, їхнє використання часто обмежене відсутністю гнучкого налаштування формату протоколу та слабкою підтримкою української мови. Відкриті інструменти пропонують більшу свободу, але вимагають суттєвих зусиль для побудови повного конвеєра обробки та узагальнення тексту. Таким чином, на ринку немає універсальної системи, яка б одночасно поєднувала високу точність багатомовного розпізнавання, підтримку сучасних моделей NLP, можливість інтеграції через API та адаптацію під вимоги конкретної організації. Це підкреслює актуальність створення власного клієнт–серверного рішення для автоматичного протоколювання зустрічей українською мовою.

Висновки до розділу 1

У першому розділі було розглянуто роль протоколів у роботі сучасних організацій та специфіку їх створення під час офлайн- і гібридних заходів. Показано, що ручне протоколювання є трудомістким процесом, який залежить від уважності та досвіду окремої людини, часто призводить до втрати деталей і потребує додаткового часу на прослуховування аудіозаписів та оформлення тексту.

Було описано загальну схему автоматизованого формування протоколу: від запису зустрічі та попередньої обробки аудіосигналу до розпізнавання мовлення, очищення транскрипту, узагальнення змісту й формування структурованого документа. Проаналізовано еволюцію технологій автоматичного розпізнавання мовлення від моделей GMM–HMM до сучасних трансформерних і мультимодальних рішень, зокрема gpt-4o-transcribe, які забезпечують нижчий рівень помилок, підтримку української мови та придатні для інтеграції через API у вебзастосунки.

Окремо розглянуто методи обробки природної мови, які дозволяють очищати транскрибований текст від властивостей усного мовлення, виділяти ключові теми, рішення та доручення й формувати протокол у формальному стилі. Наведено огляд наукових робіт і програмних рішень (корпуси ICSI, AMI, комерційні сервіси Otter.ai, Fireflies.ai, інструменти в Microsoft Teams і Google Meet, а також відкриті ASR-моделі), що показав: існуючі системи або орієнтовані переважно на англомовний ринок, або не дають можливості гнучко налаштовувати формат протоколу та використовувати українську мову.

Таким чином, сформульовано проблему, яка є предметом даної роботи: відсутність програмного забезпечення, що автоматично генерує протоколи зустрічей українською мовою на основі аудіозаписів, поєднуючи сучасні ASR- та NLP-технології та підтримуючи використання у форматі вебзастосунку. У подальших розділах буде визначено вимоги до такої системи, спроєктовано її архітектуру та обґрунтовано вибір конкретних технологій.

2 МОДЕЛЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Вибір та обґрунтування підходів до моделювання

Моделювання є одним із ключових етапів проєктування програмного забезпечення, оскільки воно дозволяє формалізувати роботу системи, встановити взаємозв’язки між її компонентами та визначити логіку обробки даних [30]. Для системи автоматичного створення протоколів зустрічей на основі аудіозаписів моделювання має особливе значення, адже така система поєднує різнорідні елементи: модулі обробки аудіосигналу, сервіси розпізнавання мовлення, алгоритми обробки природної мови та інструменти генерації структурованого документа. Їх коректна взаємодія визначає якість кінцевого результату, тому вибір відповідних підходів до моделювання є обґрунтованою необхідністю [9].

Основна мета моделювання полягає у створенні концептуального опису системи, який відображає її функціональну структуру, інформаційні потоки та правила взаємодії між компонентами. Такий опис дозволяє проаналізувати майбутню систему до її реалізації, виявити потенційні недоліки, оптимізувати архітектуру та забезпечити узгодженість між вимогами й технічною реалізацією [9, 30].

Для розроблення програмного забезпечення, що поєднує клієнтську частину, серверну логіку, зовнішні API для розпізнавання мовлення та модулі NLP-обробки, доцільно застосовувати комбінацію кількох підходів моделювання. Зокрема, функціональне моделювання дає змогу описати послідовність операцій, які виконує система; а інформаційне моделювання формалізує структуру даних [30].

Одним із базових інструментів є DFD-діаграми (Data Flow Diagrams), які використовуються для відображення потоків даних між модулями системи. Цей підхід дозволяє чітко описати, як аудіозапис проходить шлях від завантаження до формування текстового протоколу, які проміжні дані генеруються, які процеси здійснюють трансформацію інформації та де вона

зберігається. Перевагою DFD є можливість представити складну обробку у вигляді послідовності логічних операцій без занурення у внутрішню реалізацію алгоритмів.

Для опису алгоритмічної логіки роботи окремих модулів можуть бути використані блок-схеми, які забезпечують наочне подання умов, циклів і послідовних дій. Вони зручні для моделювання процесів попередньої обробки аудіо, фільтрації шумів або підготовки тексту до NLP-аналізу. Проте для складніших сценаріїв вони швидко втрачають читабельність, тому застосовуються лише на локальному рівні.

Важливою групою інструментів є діаграми UML (Unified Modeling Language) [14, 30].

Діаграми прецедентів дозволяють описати взаємодію користувача із системою на концептуальному рівні: завантаження аудіофайлу, вибір шаблону протоколу, генерація документу тощо [14].

Діаграми класів дають змогу відобразити внутрішню структуру програмних модулів – моделі документів, сутності бази даних, взаємодію API-сервісів [22].

Діаграми послідовності використовуються для опису обміну даними між компонентами: клієнтською частиною, сервером, мікросервісом транскрипції та модулем узагальнення тексту.

Перевагою UML є здатність системно описати архітектуру у вигляді взаємопов’язаних моделей, що підвищує прозорість проєктування і спрощує подальшу реалізацію.

Для формального опису структури даних застосовуються ER-діаграми (Entity–Relationship). Вони дозволяють моделювати логічні зв’язки між сутностями: транскрипт, протокол, шаблон оформлення, користувач, історія обробок. ER-моделі допомагають оптимізувати структуру бази даних, уникнути надлишковості та забезпечити цілісність даних під час роботи системи.

Таким чином, вибір методів моделювання зумовлений необхідністю відобразити різні аспекти функціонування системи: логіку процесів, взаємодію компонентів, структуру даних та користувацькі сценарії. Комбіноване використання DFD-діаграм, UML-нотацій та ER-моделей забезпечує всебічне проєктування програмного забезпечення й дозволяє створити узгоджену архітектуру, що відповідатиме вимогам системи автоматичного створення протоколів зустрічей на основі аудіозаписів.

2.2 Специфікація вимог

ПРИЗНАЧЕННЯ ПРОЄКТУ

Розроблюване програмне забезпечення призначене для автоматичного створення протоколів зустрічей на основі аудіозаписів, отриманих у ході офлайн- або гібридних подій. Система забезпечує повний цикл обробки: від завантаження або запису аудіофайлу в браузері до формування структурованого протоколу у форматі PDF чи DOCX. У межах програмної документації погоджується, що система має забезпечувати коректну інтеграцію з хмарними сервісами розпізнавання мовлення та засобами узагальнення тексту. Передбачається клієнт–серверна архітектура із взаємодією через REST API.

ЗАГАЛЬНИЙ ОПИС

Сфера застосування охоплює організації, установи та команди, які регулярно проводять зустрічі й потребують швидкого та точного документування їх результатів. Програмне забезпечення автоматизує рутинну частину роботи, зменшує залежність від людського фактору та забезпечує стандартизований формат протоколів.

Користувачами системи є співробітники, які відповідають за підготовку протоколів зустрічей: менеджери, координатори, секретарі або інші особи, що виконують адміністративні функції.

Система складається з користувацького вебінтерфейсу, серверної частини для обробки запитів, модулів автоматичного розпізнавання мовлення

та узагальнення тексту, сховища даних для транскриптів і протоколів, а також засобів експорту документів у PDF та DOCX. Компоненти працюють у межах клієнт–серверної взаємодії, забезпечуючи повний цикл обробки аудіоматеріалу від завантаження до формування структурованого протоколу.

Основними обмеженнями системи є необхідність постійного доступу до Інтернету, підтримка лише української мови інтерфейсу та обмеження на розмір аудіофайлів. Швидкість роботи залежить від якості мережевого з'єднання та часу обробки зовнішніми сервісами.

ФУНКЦІЇ СИСТЕМИ

Функція транскрипції аудіо

Опис функції: забезпечує перетворення аудіозаписів на текстову транскрипцію з можливістю попереднього запису звуку через браузер.

Вхідна інформація: аудіофайл у форматах MP3, WAV або M4A.

Вихідна інформація: текст транскрипції.

Функціональні вимоги: коректне завантаження аудіо, передача на API, робота з файлами до 100 МБ, відображення статусу обробки.

Функція генерації протоколу

Опис функції: формування структурованого протоколу на основі транскрипції, включаючи тези обговорення, рішення та відповідальних осіб.

Вхідна інформація: транскрибований текст.

Вихідна інформація: протокол встановленої структури.

Функціональні вимоги: аналіз і структурування змісту, підтримка шаблонів протоколів, можливість редагування перед збереженням.

Функція управління учасниками

Опис функції: створення, редагування та видалення записів про учасників зустрічі із зазначенням ролей.

Вхідна інформація: дані про учасників.

Вихідна інформація: оновлені списки учасників.

Функціональні вимоги: валідація даних, зв'язування учасників із зустрічами, збереження у базі.

Функція роботи з документами

Опис функції: забезпечення збереження транскрипцій і протоколів, повторної генерації документів та їх експорт.

Вхідна інформація: дані зустрічі, транскрипт, протокол.

Вихідна інформація: файли PDF або DOCX.

Функціональні вимоги: формування документів, ведення історії зустрічей, пошук та фільтрація.

Функція аутентифікації користувачів

Опис функції: забезпечення реєстрації, вхід до системи та контроль доступу.

Вхідна інформація: логін і пароль.

Вихідна інформація: токен доступу та дані профілю.

Функціональні вимоги: JWT-автентифікація, шифрування даних, рольова модель доступу.

ВИМОГИ ДО ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

Вхідними даними є аудіофайли зустрічей, інформація про учасників, параметри подій та транскрипції. Вихідними даними виступають готові протоколи, транскрипти та метадані зустрічей.

Нормативно-довідкова інформація включає шаблони структури протоколів, внутрішні довідники ролей та можливі класифікації типів подій.

Інформація зберігається у базі даних PostgreSQL з використанням нормалізованої структури та забезпеченням підтримки цілісності зв’язків між сутностями.

ВИМОГИ ДО ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ

Користувачу потрібен персональний комп’ютер або мобільний пристрій із сучасним веббраузером. Сервер повинен мати достатні ресурси для обробки запитів, стабільне інтернет-підключення та доступ до API OpenAI.

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Архітектура програмної системи

Система повинна мати клієнт–серверну архітектуру. Клієнт працює у веббраузері, сервер відповідає за обробку аудіо, транскрипцію, узагальнення тексту, формування протоколів і роботу з базою даних.

Системне програмне забезпечення

Сервер повинен підтримувати роботу вебзастосунку, можливість обробки файлів та взаємодії з базою даних. Роботу необхідно забезпечити на операційних системах, сумісних із серверними вебтехнологіями.

Мережеве програмне забезпечення

Передбачає використання захищеного з'єднання через HTTPS та підтримку передачі даних у стандартизованому форматі.

Програмне забезпечення ведення інформаційної бази

База даних повинна забезпечувати зберігання транскриптів, протоколів, інформації про зустрічі та користувачів, а також підтримувати виконання резервного копіювання.

Мова і технологія розробки ПЗ

Для серверної та клієнтської частин повинні використовуватися сучасні вебтехнології, що забезпечують масштабованість, стабільність і можливість подальшого розвитку системи.

ВИМОГИ ДО ЗОВНІШНІХ ІНТЕРФЕЙСІВ

Інтерфейс користувача

Застосунок повинен мати зрозумілий вебінтерфейс для завантаження аудіо, перегляду транскрипту, редагування протоколу та його збереження. Інтерфейс орієнтований на користувачів без спеціальної технічної підготовки.

Апаратний інтерфейс

За потреби користувач може використовувати мікрофон для запису аудіо безпосередньо у браузері.

Програмний інтерфейс

Програмний інтерфейс системи реалізовано за допомогою вебклієнта на Angular та серверної частини, побудованої на FastAPI. Клієнтська частина

формує запити до серверу через REST API, а FastAPI забезпечує обробку цих запитів, виконання транскрипції, узагальнення та взаємодію з базою даних.

Такий підхід дає змогу підтримувати модульність, швидке внесення змін у серверні ендпоінти та розвиток клієнтського інтерфейсу без порушення основної логіки роботи системи. Програмний інтерфейс є розширюваним і дозволяє інтегрувати додаткові сервіси чи модулі обробки у майбутньому.

Комунікаційний протокол

Усі дані повинні передаватися за протоколом HTTPS для забезпечення конфіденційності та цілісності інформації.

ВЛАСТИВОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Доступність

Застосунок повинен бути доступним усім користувачам, які мають доступ до браузера та Інтернету, незалежно від операційної системи.

Супроводжуваність

Програмне забезпечення має бути придатним до подальшого оновлення, додавання нових модулів та адаптації під потреби організації.

Переносимість

Веборієнтованість дозволяє запускати застосунок на різних операційних системах. Можливий перенос серверної частини на інші середовища без значної зміни логіки роботи.

Продуктивність

Система повинна забезпечувати обробку аудіофайлів у прийнятний час, залежно від тривалості запису та навантаження сервера.

Надійність

Застосунок повинен забезпечувати стабільну роботу, коректне збереження даних та відновлення після можливих збоїв.

Безпека

Доступ до протоколів і транскриптів має бути обмежений автентифікацією. Усі файли та дані передаються у зашифрованому вигляді.

2.3 Функціональна модель системи

Програмне забезпечення автоматичного створення протоколів зустрічей виконує низку ключових функцій, спрямованих на обробку аудіозаписів офлайн- або гібридних заходів, перетворення усного мовлення у текстову форму та формування структурованого протоколу зустрічі. Реалізовані функції охоплюють повний цикл роботи системи – від моменту завантаження аудіозапису та даних про зустріч до збереження готового протоколу та надання доступу користувачу у зручному форматі. Далі розглянуто основні функції системи та їх роль у процесі автоматизованого створення протоколів.

Завантаження аудіозапису та даних про зустріч. Дана функція забезпечує початковий етап роботи системи та відповідає за приймання вхідних даних. Користувач завантажує аудіозапис зустрічі та вводить основну інформацію про подію, зокрема назву, дату, тип зустрічі та дані про учасників. У межах цієї функції здійснюється перевірка відповідності аудіофайлу встановленим вимогам щодо формату та розміру, а також перевірка заповнення обов’язкових полів. Результатом виконання функції є збережений аудіозапис разом із пов’язаними метаданими, які передаються на наступний етап обробки.

Розпізнавання мовлення та формування текстового представлення. Ця функція відповідає за перетворення аудіозапису зустрічі у текстову форму. Система формує запит до зовнішнього сервісу автоматичного розпізнавання мовлення (ASR), передає аудіофайл та отримує результат у вигляді текстового представлення мовлення. Розпізнавання мовлення виконується на стороні зовнішнього API, тоді як програмне забезпечення забезпечує коректну передачу даних та збереження отриманого результату. Сформований текст зустрічі зберігається у базі даних і використовується для подальшої інтелектуальної обробки.

Узагальнення тексту та формування протоколу. Основна інтелектуальна функція системи, яка забезпечує автоматичне створення структурованого протоколу зустрічі. На цьому етапі текстове представлення

мовлення, а також дані про зустріч і її учасників використовуються для формування запиту до мовної моделі gpt-5-nano. Запит містить інструкції щодо структури протоколу, правил узагальнення та вимог до стилю викладу. У відповідь система отримує вже структурований текст протоколу, який містить основні пункти зустрічі, обговорювані питання, прийняті рішення та доручення. Отриманий текст додатково форматується відповідно до встановлених правил і готується до збереження.

*Збереження протоколу та надання доступу користувачу. З*авершальна функція системи відповідає за довготривале збереження результатів обробки та організацію доступу до них. Сформований та відформатований текст протоколу зберігається у вигляді файлу встановленого формату, а відповідні метадані – у базі даних. Система забезпечує можливість перегляду та завантаження протоколу відповідно до політик доступу та авторизації. Таким чином, користувач отримує готовий протокол зустрічі у зручному для подальшого використання вигляді.

На рис. 2.1 наведено контекстну діаграму IDEF0 нульового рівня, яка відображає загальну функціональну структуру системи та взаємозв’язки між основними етапами обробки даних.

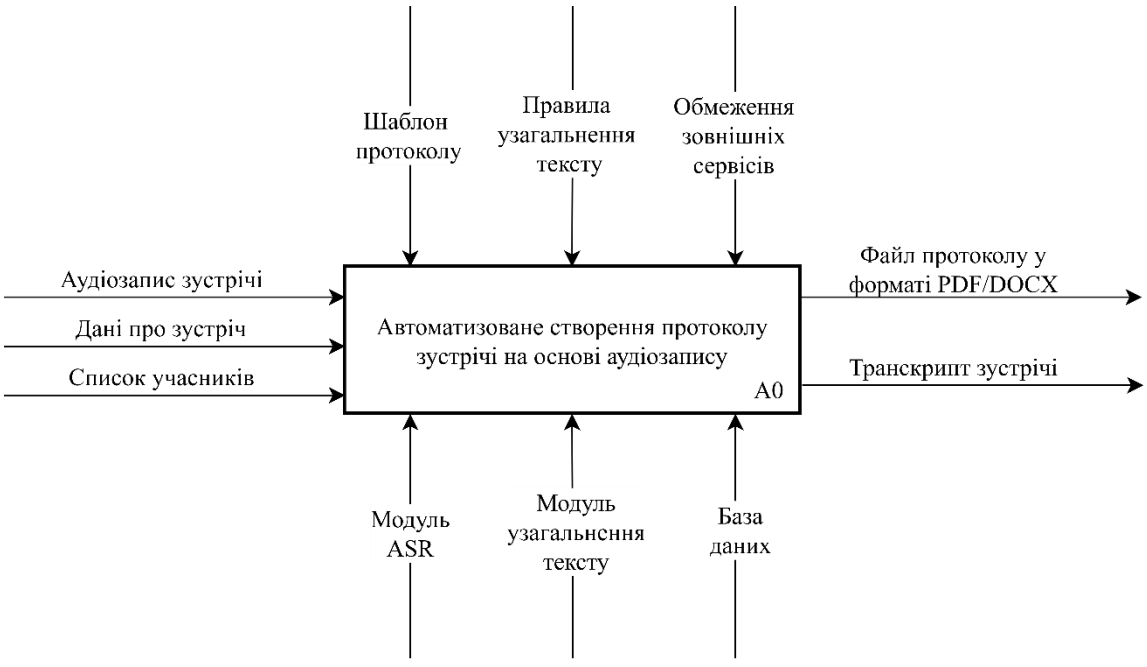


Рисунок 2.1 –Контекстна діаграма IDEF0 A-0

Система має 3 входи, 2 виходи, 3 механізми та 3 контролі. На вході система приймає аудіозапис зустрічі разом із даними про подію, після чого ініціюється процес автоматизованої обробки інформації. На виході система формує структурований протокол зустрічі, готовий до збереження та подальшого використання у вигляді файлу встановленого формату.

На рис. 2.2 наведено діаграму IDEF0 1-го рівня для програмного забезпечення генерації протоколів зустрічей на основі аудіозаписів.

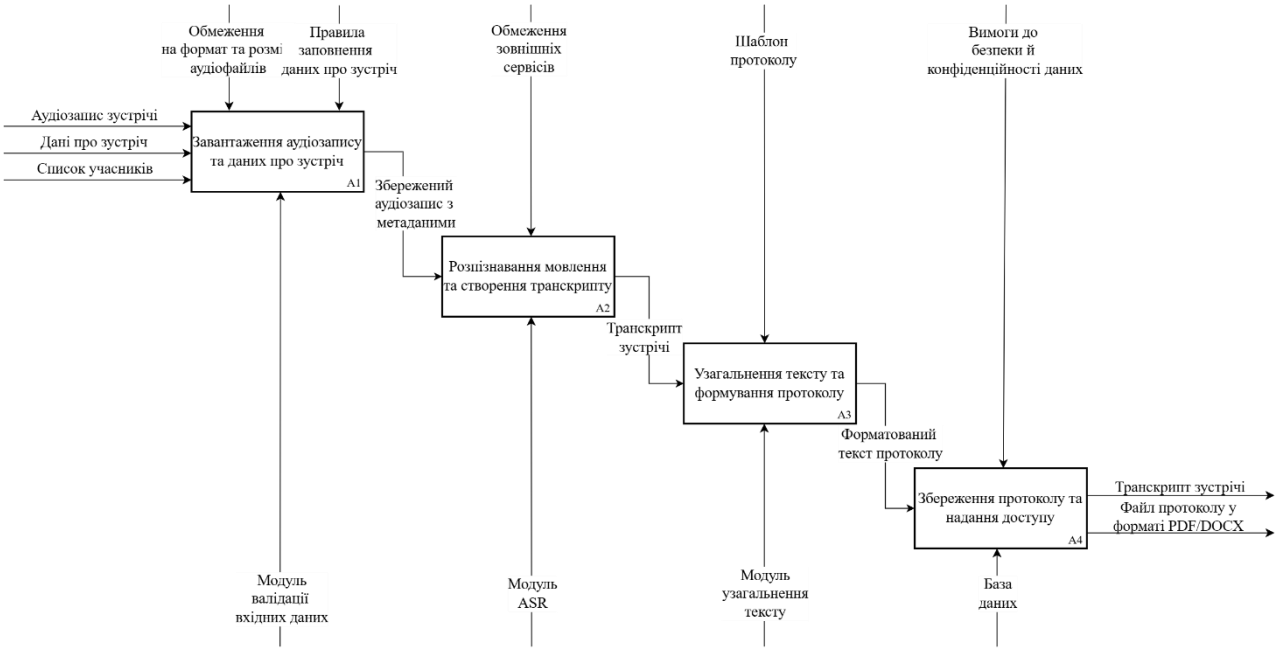


Рисунок 2.2 – Діаграма IDEF0 A0

На рис. 2.3 наведено діаграму IDEF0 2-го рівня для функції «Завантаження аудіозапису та даних про зустріч».

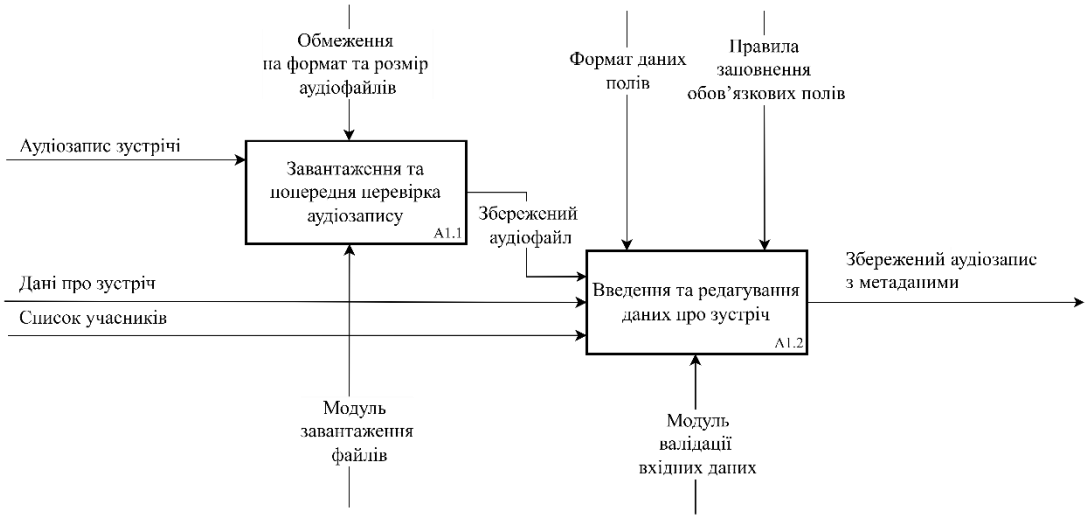


Рисунок 2.3 – Діаграма IDEF0 A1

На рис. 2.4 наведено діаграму IDEF0 2-го рівня для функції «Розпізнавання мовлення та створення транскрипту».

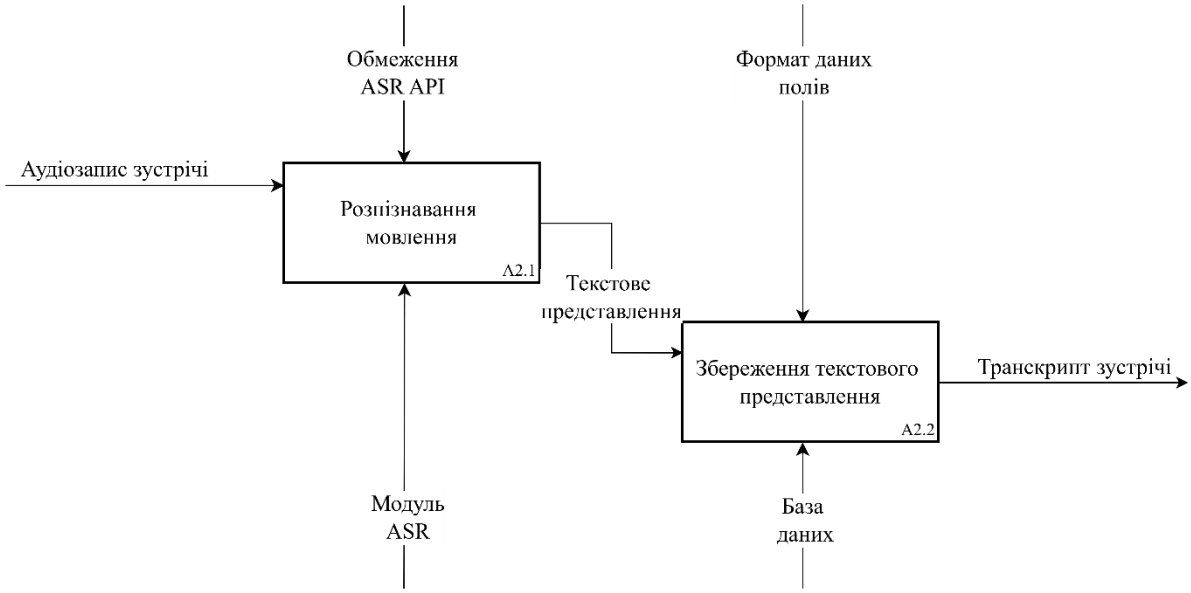


Рисунок 2.4 – Діаграма IDEF0 A2

На рис. 2.5 наведено діаграму IDEF0 2-го рівня для функції «Узагальнення тексту та формування протоколу».

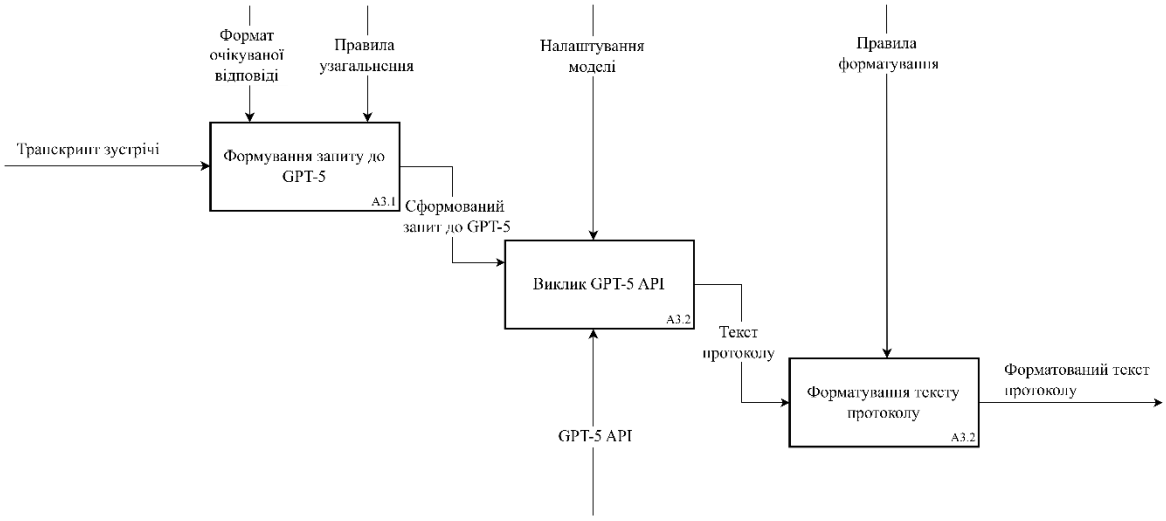


Рисунок 2.5 – Діаграма IDEF0 A3

На рис. 2.6 наведено діаграму IDEF0 2-го рівня для функції «Збереження протоколу та надання доступу».

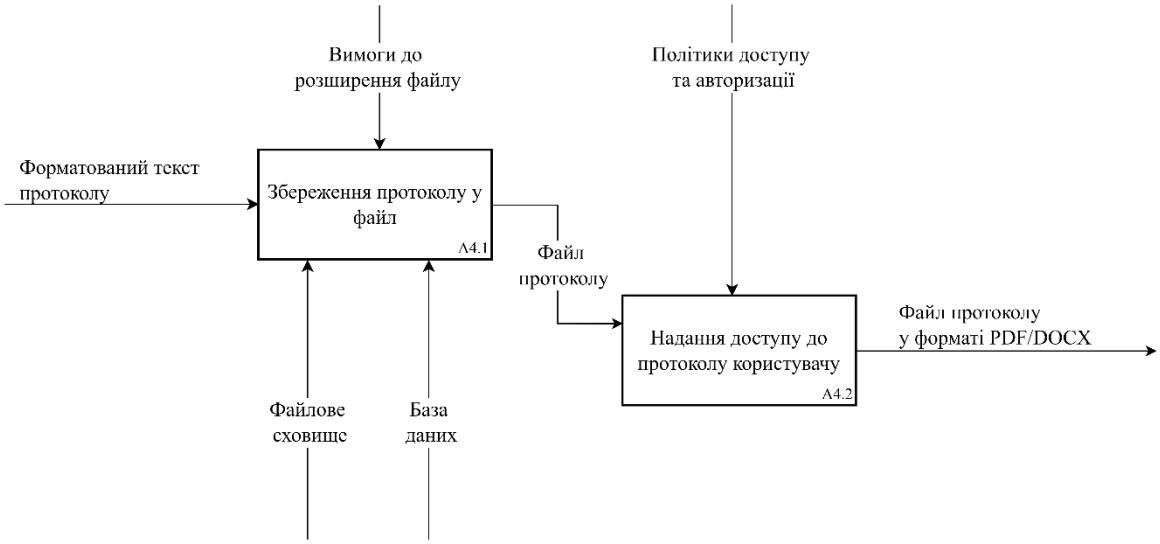


Рисунок 2.6 – Діаграма IDEF0 A4

Побудована функціональна модель дозволяє формально описати логіку роботи програмного забезпечення автоматичного створення протоколів зустрічей. Використання методології IDEF0 забезпечує чітке розмежування функцій системи, визначення потоків даних та залежностей між етапами обробки, що є основою для подальшого проєктування архітектури та програмної реалізації.

2.4 Інформаційна модель системи

Інформаційна модель системи автоматичного створення протоколів зустрічей призначена для формалізації потоків даних, що циркулюють між користувачем, серверною частиною платформи та зовнішніми сервісами розпізнавання мовлення й інтелектуального узагальнення тексту. Побудова інформаційної моделі дозволяє відобразити, які саме дані надходять у систему, як вони трансформуються під час обробки та у якому вигляді повертаються користувачу.

Для опису інформаційної взаємодії в межах платформи використано методологію DFD (Data Flow Diagram). DFD-діаграми наочно демонструють рух інформаційних потоків між процесами, зовнішніми сутностями та сховищами даних, не заглиблюючись у внутрішню реалізацію алгоритмів. Такий підхід є доцільним на етапі аналітичного та архітектурного

проєктування, оскільки дозволяє зосередитись саме на логіці обробки інформації.

На рисунку 2.7 наведено контекстну DFD-діаграму системи автоматичного створення протоколів зустрічей, яка відображає взаємодію платформи з основними зовнішніми сутностями.

Зовнішніми сутностями системи є:

— користувач, який ініціює процес обробки, надає вхідні дані та отримує результат;

— ASR API, що виконує автоматичне розпізнавання мовлення та повертає текстову транскрипцію аудіозапису;

— API узагальнення (gpt-5-nano), яке формує структурований текст протоколу на основі транскрипту та контексту зустрічі.

Центральним процесом контекстної діаграми є «Система забезпечення генерації протоколів зустрічей на основі аудіозаписів», яка виконує роль координатора всіх етапів обробки.

Згідно з контекстною діаграмою, користувач передає до системи аудіозапис та дані про зустріч, після чого система ініціює обробку. Аудіофайл надсилається до зовнішнього сервісу розпізнавання мовлення, у відповідь на що система отримує транскрипт. Отриманий текст використовується для формування запиту на узагальнення, який передається до мовної моделі. Результатом роботи системи є структурований протокол зустрічі, що повертається користувачу.

Таким чином, контекстна DFD-діаграма відображає систему як єдиний логічний процес і дозволяє визначити основні вхідні та вихідні інформаційні потоки.

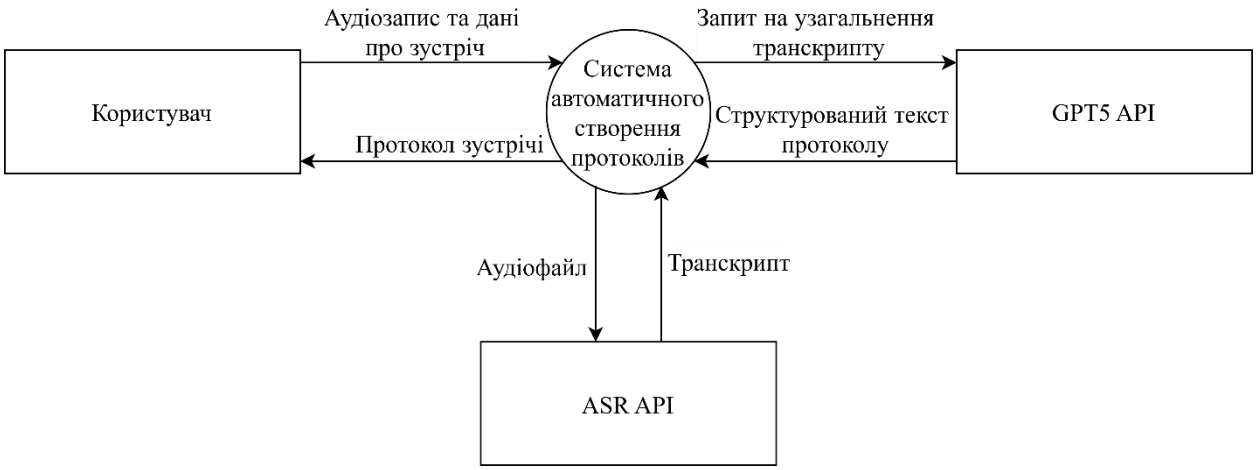


Рисунок 2.7 – Контекстна діаграма DFD

Побудована інформаційна модель у вигляді DFD-діаграм дозволяє наочно представити логіку руху даних у системі автоматичного створення протоколів зустрічей. Вона демонструє, як первинні аудіодані перетворюються у структурований протокол шляхом послідовної обробки із залученням зовнішніх сервісів.

Застосування DFD у межах даного проєкту забезпечує чітке розмежування відповідальностей між процесами, спрощує аналіз інформаційних потоків і створює основу для подальшої реалізації серверної частини системи відповідно до визначеної архітектури.

2.5 Сценарії використання

Сценарії використання застосовуються для опису функціональної поведінки програмного забезпечення з точки зору взаємодії користувача із системою. Вони дозволяють формалізувати послідовність дій, необхідних для досягнення певної мети, а також врахувати можливі альтернативні варіанти виконання операцій і помилки, що можуть виникати під час роботи системи.

Для опису сценаріїв використання застосовуються різні рівні деталізації, вибір яких залежить від етапу проєктування та важливості відповідної функціональності:

— коротка форма – коротке представлення основного сценарію взаємодії без деталізації окремих кроків;

– поверхнева форма – текстовий опис основного та альтернативних сценаріїв у довільній формі;

– повна форма – повний перелік кроків виконання сценарію із зазначенням передумов, результатів та можливих відхилень, який використовується для аналізу ключових функцій системи.

Короткий use case

Користувач завантажує аудіозапис зустрічі та вводить основні дані про подію, після чого система автоматично виконує розпізнавання мовлення, узагальнює отриманий текст і формує структурований протокол зустрічі. У результаті користувач отримує готовий протокол у встановленому форматі з можливістю перегляду та завантаження.

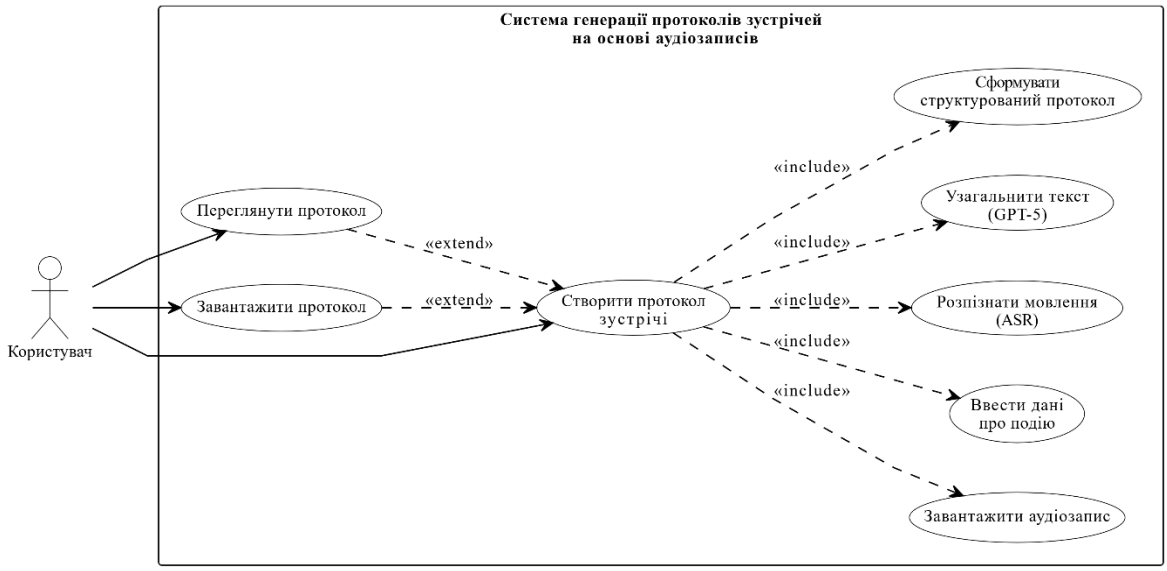


Рисунок 2.8 – Коротка use-case діаграма

Поверхневий use case

Користувач створює нову зустріч у системі, завантажує аудіозапис та вводить інформацію про учасників і параметри події. Після цього система передає аудіофайл до зовнішнього сервісу розпізнавання мовлення, отримує текстове представлення зустрічі та використовує мовну модель для формування структурованого протоколу. Готовий протокол зберігається у системі та стає доступним для перегляду і завантаження.

Альтернативні сценарії:

1) Аудіофайл має некоректний формат або перевищує допустимий розмір.

2) Сервіс розпізнавання мовлення повертає помилку або недоступний.

3) Узагальнення тексту не може бути виконане через обмеження зовнішнього API.

4) Користувач скасовує процес обробки до його завершення.

Повний use case

Таблиця 2.1 – Повний use case

Primary Actor	Користувач (організатор або секретар зустрічі)
Scope	Програмне забезпечення генерації протоколів зустрічей на основі аудіозаписів
Level	Формування структурованого протоколу зустрічі
Preconditions	Користувач має доступ до системи та підготовлений аудіозапис зустрічі
Stakeholders and interests	─ користувач – зацікавлений у швидкому отриманні коректного протоколу зустрічі; ─ організація – зацікавлена у стандартизованому документуванні зустрічей.
Main Success Scenario	1) тренування зі швидкості введення тексту на клавіатурі; 2) користувач проходить процес реєстрації для створення облікового запису; 3) користувач успішно виконує вхід у систему через аутентифікацію та авторизацію; 4) користувач створює нову зустріч у системі; 5) користувач завантажує аудіозапис та вводить дані про зустріч; 6) система зберігає аудіофайл і метадані;

Продовження таблиці 2.1

Main Success Scenario	7) система передає аудіозапис до сервісу ASR; 8) система отримує текстове представлення мовлення; 9) система формує запит до мовної моделі gpt-5-nano; 10) система отримує структурований текст протоколу; 11) система зберігає протокол та формує файл документа; 12) користувач отримує доступ до готового протоколу.
Result	Користувач отримав структурований протокол зустрічі у потрібному форматі.
Extensions	1. Користувач не автентифікований у системі: 1.1. користувач намагається створити зустріч або завантажити аудіозапис без попередньої автентифікації; 1.2. система перевіряє стан сесії та виявляє відсутність активної автентифікації; 1.3. система перенаправляє користувача на сторінку входу або реєстрації; 1.4. після успішної автентифікації користувач може повторити виконання сценарію з початкового кроку. 2. Помилка завантаження аудіофайлу: 2.1. користувач завантажує аудіофайл, що не відповідає вимогам до формату або перевищує допустимий розмір; 2.2. система виконує перевірку параметрів файлу та виявляє невідповідність;

Продовження таблиці 2.1

	2.3. система повідомляє користувача про причину відмови у завантаженні; 2.4. користувач має можливість завантажити інший аудіофайл або скасувати виконання сценарію. 3. Помилка під час розпізнавання мовлення: 3.1. зовнішній сервіс розпізнавання мовлення тимчасово недоступний або повертає помилку обробки; 3.2. система фіксує помилку та припиняє подальше виконання сценарію; 3.3. користувач отримує повідомлення про неможливість виконання розпізнавання мовлення у поточний момент; 3.4. користувач може повторити спробу пізніше або використати інший аудіозапис. 4. Помилка узагальнення тексту та формування протоколу: 4.1. під час виконання запиту до мовної моделі перевищено допустимі обмеження API або виникла помилка обробки; 4.2. система реєструє помилку та зберігає проміжні результати обробки; 4.3. користувач отримує повідомлення про неможливість формування протоколу; 4.4. користувач може повторно ініціювати формування протоколу або змінити параметри узагальнення. 5. Переривання сценарію з боку користувача:

Продовження таблиці 2.1

Extensions	5.1. користувач скасовує виконання сценарію до його завершення; 5.2. система припиняє обробку даних та зберігає поточний стан зустрічі; 5.3. користувач може повернутися до сценарію пізніше та продовжити роботу з тієї ж зустрічі.
Special Requirements	1. підтримка різних форматів аудіофайлів; 2. збереження конфіденційності даних зустрічей; 3. можливість формування протоколів у стандартизованому вигляді.
Frequency of Occurrence	Сценарій використання виконується щоразу під час створення нового протоколу зустрічі.

Таким чином, сценарії використання дозволяють наочно описати основні варіанти взаємодії користувача із системою та послідовність виконання ключових операцій. Представлення функціональності у вигляді UML-сценаріїв спрощує розуміння логіки роботи системи та дає змогу виявити можливі альтернативні шляхи виконання процесів і помилки.

Висновки до розділу 2

У другому розділі виконано моделювання програмного забезпечення автоматичного створення протоколів зустрічей на основі аудіозаписів. Обґрунтовано вибір підходів до моделювання та сформовано специфікацію вимог, що визначає призначення системи, її функціональні можливості, обмеження й вимоги до програмного, технічного та інформаційного забезпечення.

Функціональну та інформаційну структуру системи описано з використанням діаграм IDEF0 і DFD. Функціональна модель відображає основні етапи обробки даних – від завантаження аудіозапису та даних про зустріч до формування і збереження готового протоколу. Інформаційні моделі

дозволили формалізувати потоки даних між компонентами системи та визначити місця їх зберігання, що забезпечує цілісність і узгодженість обробки інформації.

У розділі описано сценарії використання системи у вигляді коротких, поверхневих та повних use case, які відображають взаємодію користувача із системою та можливі альтернативні сценарії. Отримані результати моделювання створюють основу для переходу до наступного етапу – проєктування архітектури програмного забезпечення та вибору конкретних технологічних рішень для реалізації системи.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Розробка архітектури програмного забезпечення

Архітектура програмного забезпечення автоматичного створення протоколів зустрічей розроблялася з урахуванням функціональних вимог, визначених у попередньому розділі, а також специфіки предметної області, що пов’язана з обробкою аудіозаписів, використанням хмарних сервісів розпізнавання мовлення та інтелектуального узагальнення тексту. Основною метою архітектурного проєктування є створення цілісної та логічно впорядкованої структури системи, яка забезпечує автоматизацію процесу підготовки протоколів зустрічей, а також можливість подальшого розвитку програмного забезпечення.

Для реалізації платформи обрано клієнт–серверну архітектуру, що є типовим підходом для сучасних веборієнтованих інформаційних систем. Такий підхід дозволяє відокремити інтерфейс користувача від серверної логіки обробки даних і централізувати виконання ресурсоємних операцій, зокрема обробку аудіофайлів та взаємодію з хмарними сервісами.

Використання клієнт–серверної архітектури обґрунтоване необхідністю обробки аудіозаписів значного розміру, інтеграції з зовнішніми API та забезпечення доступу до системи з різних пристроїв без встановлення додаткового програмного забезпечення. Крім того, такий підхід спрощує супроводження системи та дає змогу незалежно розвивати клієнтську і серверну частини.

Альтернативні архітектурні рішення, зокрема створення настільного застосунку або повністю автономної системи без серверної частини, були відхилені через обмежені можливості масштабування та складність інтеграції з хмарними сервісами розпізнавання мовлення й обробки природної мови.

Архітектура програмного забезпечення розроблялася з урахуванням низки технічних і організаційних обмежень. Зокрема, система орієнтована на

роботу у вебсередовищі та передбачає постійний доступ до мережі Інтернет, оскільки ключові етапи обробки аудіозаписів виконуються із залученням зовнішніх хмарних сервісів. Це обмеження безпосередньо впливає на вибір архітектурного підходу та унеможливлює повністю автономну роботу системи.

Також передбачається, що основне навантаження на систему має нерівномірний характер і залежить від кількості одночасних запитів на обробку аудіозаписів. У зв’язку з цим архітектура повинна забезпечувати стабільну роботу за умов пікових навантажень, зокрема під час масового завантаження аудіофайлів. При цьому обсяги даних, що передаються між клієнтською і серверною частинами, можуть бути значними, що потребує централізованої обробки та контролю ресурсів на стороні сервера.

Ще одним припущенням є використання зовнішніх API як “чорних скриньок”, внутрішня реалізація яких не контролюється розробником системи. Це означає, що архітектура має бути стійкою до можливих затримок або тимчасової недоступності таких сервісів і передбачати коректну обробку помилок.

Архітектурно система складається з трьох основних компонентів: клієнтської частини, серверної частини та сховища даних. Такий поділ є стандартним для вебзастосунків і дозволяє чітко розмежувати відповідальності між окремими частинами системи.

Клієнтська частина забезпечує взаємодію користувача із системою. Вона відповідає за завантаження або запис аудіо, введення даних про зустрічі та учасників, перегляд транскриптів і сформованих протоколів, а також ініціювання процесу автоматичної обробки. Уся взаємодія з серверною частиною відбувається через REST API, що дозволяє зберігати незалежність між інтерфейсом користувача та бізнес-логікою.

Серверна частина реалізує основну логіку роботи системи. Вона відповідає за приймання і валідацію аудіофайлів, керування процесом транскрипції, виклики зовнішніх сервісів розпізнавання мовлення та

узагальнення тексту, формування структурованого протоколу і збереження результатів обробки. Сервер виступає центральним координатором усіх етапів обробки інформації.

Сховище даних використовується для довготривалого збереження інформації про користувачів, зустрічі, аудіофайли, транскрипти та сформовані протоколи. Централізоване зберігання даних забезпечує цілісність інформації та можливість повторного доступу до результатів обробки.

На рисунку 3.1 наведено загальну архітектуру платформи автоматичного створення протоколів зустрічей.

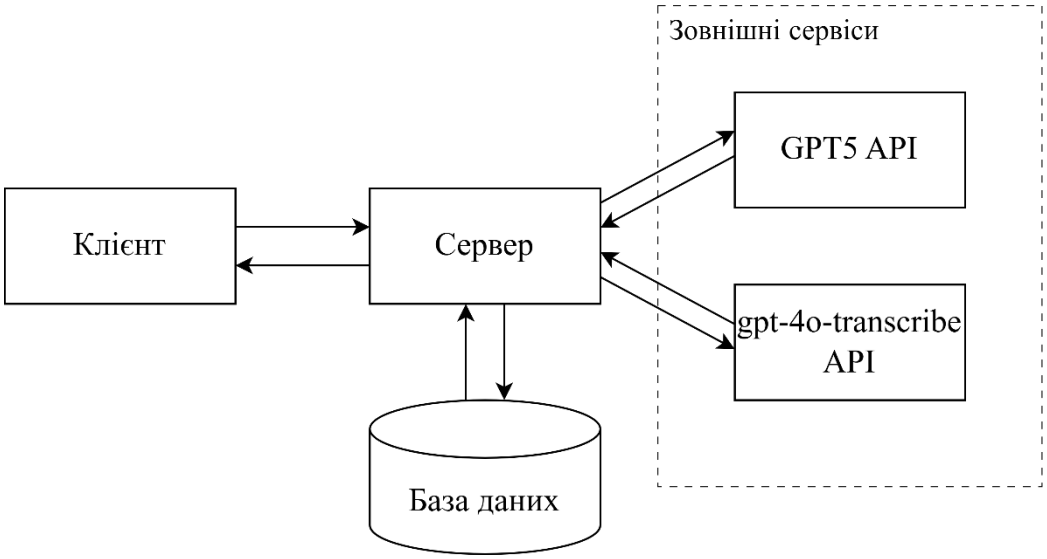


Рисунок 3.1 – Загальна архітектура платформи автоматичного створення протоколів зустрічей

Взаємодія між клієнтською та серверною частинами здійснюється за допомогою REST API, що дозволяє стандартизувати обмін даними. Враховуючи, що процеси транскрипції та інтелектуального узагальнення тексту є тривалими у часі операціями, архітектура передбачає асинхронну модель обробки запитів.

Після ініціювання користувачем створення нової зустрічі клієнтська частина формує HTTP-запит до сервера, який містить аудіофайл та супровідні метадані. Серверна частина приймає цей запит, виконує валідацію вхідних даних, зберігає файл у сховищі та реєструє завдання в черзі обробки. Після цього сервер миттєво повертає клієнту відповідь про успішний початок

операції (HTTP 202 Accepted) разом з ідентифікатором завдання, не змушуючи користувача чекати завершення всього процесу.

Безпосередня обробка, що включає взаємодію із зовнішніми сервісами розпізнавання мовлення та узагальнення тексту, виконується сервером у фоновому режимі. Клієнтська частина періодично звертається до сервера для перевірки статусу завдання (polling) або очікує повідомлення про завершення. Після фіналізації всіх етапів результати зберігаються в базі даних, і клієнт отримує доступ до готового протоколу.Пайплайн обробки аудіозаписів

Обробка аудіозаписів у системі здійснюється послідовно, що дозволяє контролювати кожен етап обробки та спрощує діагностику помилок. Після завантаження аудіофайлу сервер виконує первинну перевірку його параметрів і зберігає пов’язані метадані. Далі аудіозапис передається до зовнішнього сервісу автоматичного розпізнавання мовлення, який формує текстове представлення зустрічі.

Отриманий текст проходить етап попередньої обробки та нормалізації, після чого передається до модуля узагальнення. На цьому етапі використовується мовна модель, яка формує структурований протокол відповідно до заданого шаблону. Готовий документ зберігається у базі даних і стає доступним користувачу для перегляду та експорту.

Такий поділ процесу на логічні етапи дозволяє змінювати або оптимізувати окремі частини обробки без необхідності модифікації всієї системи.

Серверна частина побудована за модульним принципом, що забезпечує чітке розмежування відповідальностей між окремими складовими. Кожен модуль відповідає за окрему функціональну область: керування зустрічами, обробку аудіо, транскрипцію, узагальнення тексту, формування документів та контроль доступу. Такий підхід спрощує підтримку системи, підвищує її зрозумілість і дозволяє в майбутньому розширювати функціонал без суттєвих змін у загальній архітектурі.

З логічної точки зору серверна частина системи може бути поділена на кілька рівнів відповідальності. Рівень представлення відповідає за приймання HTTP-запитів від клієнта та формування відповідей у стандартизованому форматі. Саме на цьому рівні реалізовані REST-ендпоінти, які визначають доступні операції системи.

Рівень прикладної логіки відповідає за координацію процесів обробки аудіозаписів, виклики модулів транскрипції та узагальнення, а також керування життєвим циклом зустрічей і протоколів. На цьому рівні реалізуються основні правила роботи системи та сценарії використання, описані у попередньому розділі.

Рівень інтеграції забезпечує взаємодію із зовнішніми сервісами та базою даних. Він інкапсулює логіку роботи з API розпізнавання мовлення, мовною моделлю та механізмами збереження даних, що дозволяє ізолювати ці аспекти від прикладної логіки.

У процесі проєктування було розглянуто можливість використання мікросервісної архітектури, проте для даного проєкту такий підхід визнано надмірним. Враховуючи масштаби системи та характер навантаження, централізована серверна архітектура з модульною структурою є більш доцільною. Вона спрощує розгортання, зменшує інфраструктурні витрати та забезпечує достатню гнучкість для подальшого розвитку платформи.

Запропонована архітектура передбачає можливість подальшого розвитку системи без кардинальних змін її структури. У разі зростання навантаження серверна частина може бути масштабована горизонтально шляхом розгортання кількох екземплярів застосунку за наявності спільного сховища даних.

Крім того, модульна організація серверної частини дозволяє в майбутньому виділити окремі функціональні компоненти у незалежні сервіси, якщо це буде обґрунтовано масштабами проєкту або вимогами до продуктивності. Таким чином, обрана архітектура не обмежує можливості еволюції системи та забезпечує баланс між простотою реалізації та гнучкістю.

Таким чином, обрана архітектура створює надійну основу для реалізації програмного забезпечення автоматичного створення протоколів зустрічей та забезпечує відповідність функціональним і нефункціональним вимогам системи.

3.2 Вибір технологій та мов програмування для розробки програмного забезпечення

До вибору технологій і мов програмування для реалізації системи автоматичного створення протоколів зустрічей на основі аудіозаписів необхідно підходити виважено. Формально систему можна реалізувати на різних мовах програмування й фреймворках, однак невдалий вибір на початковому етапі призводить до зростання складності розробки, проблем із підтримкою, обмежень у масштабуванні та ускладнення інтеграції із зовнішніми сервісами [39]. У межах даного проєкту технологічний стек обирався з урахуванням задач системи: приймання й обробка аудіофайлів, інтеграція з ASR-сервісом та мовною моделлю, формування документів, робота з базою даних, а також реалізація безпечного доступу користувачів до матеріалів зустрічей.

Логічно спочатку визначити технології серверної частини, оскільки саме Back-End реалізує основну бізнес-логіку: приймає запити клієнта, виконує взаємодію із зовнішніми API, керує станами обробки, забезпечує збереження результатів і контроль доступу. Сьогодні для створення вебсерверів та REST API використовують різні мови програмування й екосистеми. Найпоширеніші підходи можна узагальнити таким переліком:

— C# / .NET (ASP.NET Core) – продуктивна платформа з розвинутими інструментами для створення веб API, зручна для корпоративних застосунків та великих команд.

— Java (Spring Boot) – класичний вибір для високонавантажених систем, має потужну екосистему, розвинуті інструменти та зрілу інфраструктуру.

— JavaScript/TypeScript (Node.js, Express/NestJS) – зручний для швидкої розробки веб API, широко застосовується у вебпроєктах, добре підходить для асинхронних операцій.

— Go (net/http, Gin) – орієнтований на продуктивність і простоту деплою, часто застосовується для мінімалістичних сервісів та високої пропускної здатності.

— Python (FastAPI/Flask/Django) – має сильну екосистему для інтеграцій, роботи з файлами та текстом, а також є природним вибором для задач, пов’язаних з NLP та сервісами ШІ.

З огляду на специфіку даного проєкту, доцільним є вибір Python як основної мови програмування серверної частини. Програмне забезпечення генерації протоколів зустрічей на основі аудіозаписів зустрічей безпосередньо інтегрується із зовнішніми сервісами розпізнавання мовлення та узагальнення тексту, виконує обробку великих текстових масивів, формує структуровані запити до мовних моделей та здійснює подальшу підготовку результатів до збереження і експорту. Python історично та практично є однією з основних мов у сфері обробки природної мови, машинного навчання та інтеграції з API сервісами штучного інтелекту, що робить його логічним вибором для реалізації серверної логіки даної системи.

Важливою перевагою Python є наявність розвиненої екосистеми бібліотек, орієнтованих на роботу з текстом, файлами та мережевими запитами [1]. Це дозволяє реалізовувати інтеграцію з ASR-сервісами та мовними моделями без надмірної складності коду, зберігаючи при цьому читабельність і підтримуваність серверної частини. Для системи, яка виконує багатокрокову обробку даних (приймання аудіо, отримання транскрипту, узагальнення, форматування, збереження), критично важливою є можливість швидко модифікувати логіку обробки та адаптувати її під зміни зовнішніх API. Python добре підходить для таких сценаріїв завдяки своїй гнучкості та високому рівню абстракції.

Ще одним суттєвим аргументом на користь Python є ефективна підтримка асинхронної моделі виконання. Взаємодія із зовнішніми сервісами розпізнавання мовлення та мовними моделями є мережево-залежною операцією, час виконання якої може змінюватися залежно від навантаження або розміру вхідних даних. Python у поєднанні з асинхронними фреймворками дозволяє обробляти такі запити неблокуючим чином, що підвищує загальну продуктивність системи та покращує масштабованість серверної частини без необхідності ускладнення архітектури.

Крім того, Python широко використовується у прототипуванні та розвитку інтелектуальних систем, що є важливим у контексті даного проєкту. Архітектура платформи передбачає можливість подальшого розширення функціональності, зокрема додавання нових алгоритмів аналізу тексту, альтернативних моделей узагальнення або додаткових етапів обробки протоколів. Використання Python спрощує впровадження таких змін, оскільки більшість сучасних рішень у сфері NLP та ШІ мають нативну або першочергову підтримку саме цієї мови [4].

Таким чином, вибір Python як мови програмування серверної частини зумовлений не лише зручністю реалізації REST API, а й його придатністю для задач обробки мовлення, роботи з текстом та інтеграції з хмарними сервісами штучного інтелекту. Це дозволяє реалізувати серверну логіку платформи як цілісну, гнучку та розширювану систему, орієнтовану на подальший розвиток і адаптацію до змін вимог.

Для реалізації серверного веб API обрано FastAPI, оскільки цей фреймворк забезпечує швидку розробку REST-ендпоінтів, підтримує асинхронну обробку запитів, має вбудовані засоби валідації вхідних даних та дозволяє автоматично генерувати технічну документацію до API. Це є важливим для даного проєкту, оскільки система працює з різними типами запитів: завантаження файлів, створення/оновлення сутностей зустрічі, запуск транскрипції, запуск генерації протоколу, отримання та завантаження результатів [8]. Крім того, FastAPI органічно підтримує побудову серверної

логіки за модульним підходом: окремі компоненти можуть відповідати за автентифікацію, керування зустрічами, інтеграцію з ASR, інтеграцію з gpt-5-nano та керування документами.

Окремо враховувалась задача збереження даних. Система працює з сутностями, між якими існують чіткі зв'язки: користувачі, зустрічі, учасники, аудіофайли, транскрипти, протоколи, а також службові стани обробки. Для такого типу даних природним є застосування реляційної моделі, де важливими є цілісність, транзакційність і можливість будувати вибірки для пошуку та фільтрації. У зв'язку з цим як СУБД обрано PostgreSQL – надійну open-source систему, яка підтримує складні запити, індексацію, обмеження цілісності та добре працює як із структурованими даними, так і з великими текстовими полями (транскрипти та тексти протоколів) [30].

Використання NoSQL у межах даного проєкту не є доцільним. NoSQL-рішення зазвичай застосовуються тоді, коли структура даних постійно змінюється, відсутні чіткі зв'язки між сутностями або потрібна горизонтальна масштабованість під надвисокі навантаження. Натомість у даній системі модель даних є визначеною, зв'язки між сутностями суттєві, а коректність і узгодженість (наприклад, відповідність протоколу конкретній зустрічі та користувачу) є критичними. Тому реляційна СУБД забезпечує більш прогнозовану та керовану реалізацію з точки зору проєктування й супроводу.

Для роботи з базою даних у Python-застосунках доцільним є використання ORM, оскільки це спрощує доступ до даних, зменшує обсяг ручного SQL і підвищує підтримуваність коду. ORM також зручний у контексті валідації й еволюції структури даних (міграції), що важливо для проєкту, який розвивається поступово та може отримувати нові сутності (наприклад, шаблони протоколів, додаткові поля зустрічі, статуси етапів обробки) [27].

Після визначення технологій серверної частини обирались технології клієнтського вебінтерфейсу. Оскільки система реалізована як вебзастосунок, базовими мовами Front-End є JavaScript та TypeScript, а вибір фактично

зводиться до підходу та фреймворку. Найпоширеніші рішення для створення односторінкових застосунків включають:

— React – бібліотека з компонентним підходом і великою екосистемою, часто використовується для гнучкої побудови UI, але значна частина інфраструктурних рішень (маршрутизація, архітектура проєкту) добирається додатково.

— Angular – повноцінний фреймворк із готовою архітектурною структурою, TypeScript як основою, вбудованими механізмами маршрутизації, форм, сервісів та організації проєкту.

— Vue – зручний і порівняно простий у входженні, добре підходить для застосунків різного масштабу, однак вибір архітектурних рішень часто залежить від додаткових бібліотек.

Для реалізації клієнтської частини обрано Angular, оскільки він надає цілісну структуру проєкту «з коробки», підтримує TypeScript та дозволяє будувати масштабовані інтерфейси з чітким розділенням компонентів, сервісів і моделей даних. Для даного застосунку це важливо, адже клієнтська частина має не лише відображати дані, а й керувати сценаріями: створення зустрічі, завантаження аудіо, відображення станів обробки, показ транскриптів, перегляд і завантаження протоколів. Angular також добре підходить для роботи з формами та валідацією, що напряму відповідає функціям введення даних про зустріч та учасників [30] .

Окремим фактором при виборі була практична ефективність розробки інтерфейсу. Для прискорення реалізації типових UI-елементів (форми, таблиці, кнопки, модальні вікна, сповіщення) доцільно застосовувати бібліотеки компонентів. У контексті Angular таким рішенням є Angular Material, який реалізує набір уніфікованих компонентів та спрощує забезпечення однакового стилю інтерфейсу, що позитивно впливає на швидкість розробки та якість користувацького досвіду.

Ключовою характеристикою даного проєкту є використання зовнішніх сервісів штучного інтелекту. Для автоматичного розпізнавання мовлення

застосовується gpt-4o-transcribe API, який приймає аудіофайл та повертає текстове представлення змісту [21]. Для формування структурованого протоколу використовується gpt-5-nano API, який на основі транскрипту й метаданих зустрічі генерує узагальнений текст відповідної структури. Важливо, що сама система не «навчає» модель і не реалізує власні алгоритми ASR чи NLP, а виступає як керований програмний шар: формує запит, передає вхідні дані, контролює обмеження, отримує відповідь, зберігає результат і перетворює його у документ встановленого формату.

У якості альтернатив для розпізнавання мовлення можуть застосовуватися open-source рішення, зокрема Whisper, Vosk або Kaldi [12]. Дані системи дозволяють виконувати ASR без використання платних API та можуть бути розгорнуті на власній інфраструктурі. Проте їх застосування має низку обмежень у контексті поставленої задачі. По-перше, якість розпізнавання мовлення в реальних умовах значною мірою залежить від параметрів запису, мови, акцентів і наявності фонових шумів. Для досягнення прийнятного рівня Word Error Rate (WER) такі рішення часто потребують додаткового донавчання моделей або складної конфігурації, що ускладнює впровадження та супровід системи. По-друге, розгортання власного ASR-сервісу вимагає значних обчислювальних ресурсів, що суперечить вимогам до простоти архітектури та масштабованості платформи.

У порівнянні з цими підходами gpt-4o-transcribe надає стабільну якість розпізнавання мовлення без необхідності налаштування моделей або підтримки окремої інфраструктури. Низький показник WER дозволяє отримувати транскрипти, придатні для подальшого семантичного аналізу без суттєвого ручного редагування. Саме цей аспект є критично важливим, оскільки помилки транскрипції напряму впливають на якість сформованого протоколу.

Таким чином, остаточно сформований технологічний стек включає: Angular для клієнтської частини, Python + FastAPI для серверної частини, PostgreSQL як реляційну СУБД, а також зовнішні API для транскрипції та

узагальнення. Обрані технології узгоджуються з клієнт–серверною архітектурою, забезпечують надійне зберігання даних, зручну інтеграцію з сервісами ШІ та дозволяють підтримувати й розширювати систему без надлишкового ускладнення її структури.

3.3 Структура бази даних платформи

Діаграма класів є одним із ключових інструментів у процесі проєктування програмного забезпечення та слугує основою для формування структури бази даних і подальшої реалізації системи засобами об’єктно-орієнтованого програмування. Вона дозволяє формалізувати внутрішню будову програмного забезпечення, визначити основні сутності предметної області, їх атрибути та зв’язки між ними.

У контексті платформи автоматичного створення протоколів зустрічей діаграма класів відображає логічну модель даних, яка безпосередньо пов’язана з реляційною структурою бази даних PostgreSQL. Кожен клас діаграми відповідає окремій таблиці бази даних, а зв’язки між класами відображають відношення між відповідними сутностями системи.

Поля класів описують стан об’єктів і зберігають дані, необхідні для роботи платформи, тоді як зв’язки між класами визначають правила взаємодії між компонентами системи. Такий підхід дозволяє забезпечити узгодженість між логічною моделлю, реалізацією серверної частини та структурою бази даних.

На рисунку 3.2 наведено діаграму класів, яка охоплює основні сутності платформи. Нижче наведено детальний опис ключових класів та їх призначення.

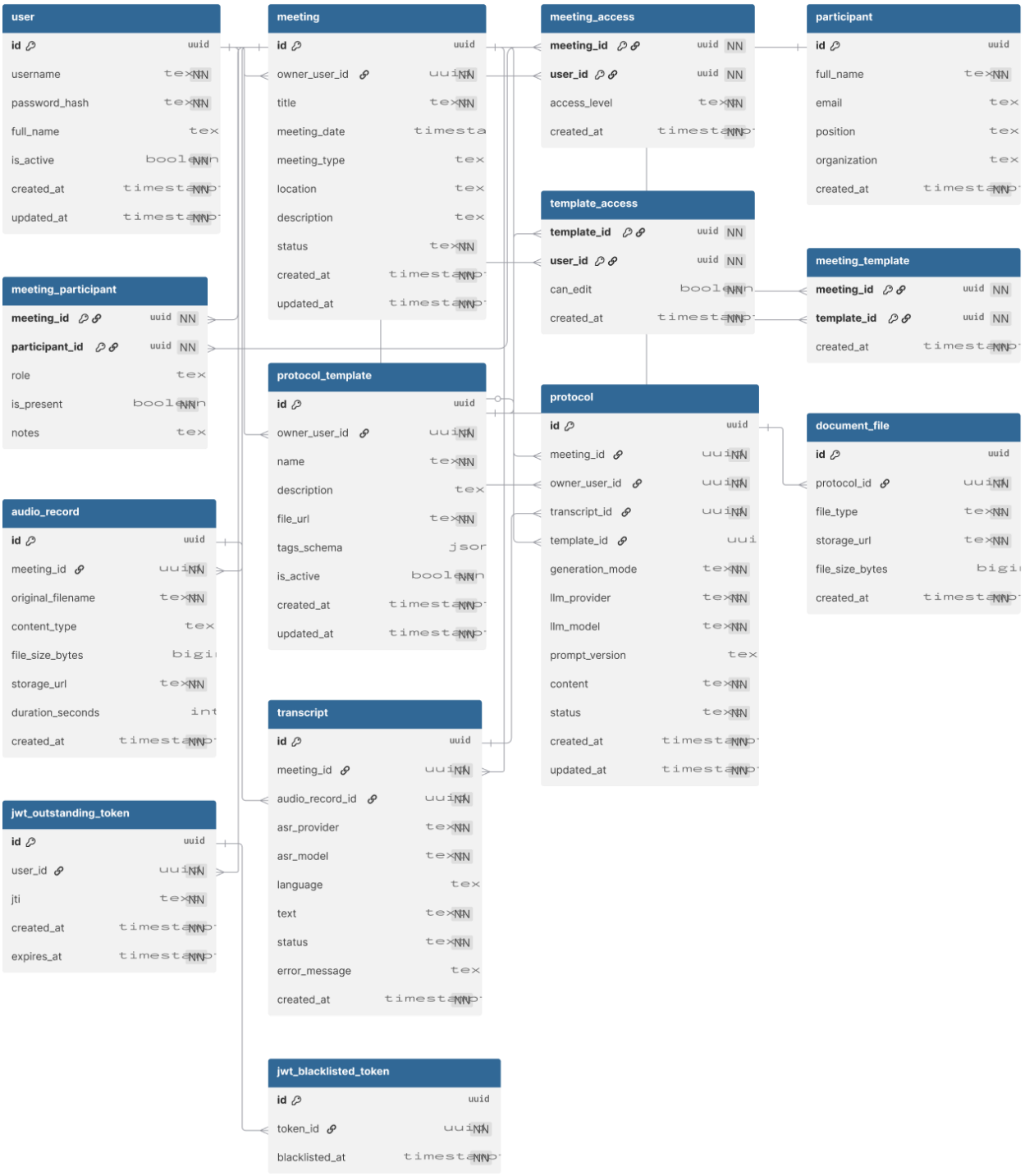


Рисунок 3.2 – ER-діаграма

Клас User

Клас User представляє користувача платформи та є базовою сутністю системи з точки зору доступу до функціоналу. Він містить ідентифікаційні дані користувача, зокрема унікальне ім’я користувача, хеш пароля та службову інформацію про дату створення облікового запису.

Цей клас використовується у механізмах реєстрації, автентифікації та авторизації. З об’єктами класу User пов’язані зустрічі, шаблони протоколів та згенеровані результати. Користувач може бути власником зустрічі або мати спільний доступ до неї.

Класи OutstandingToken та BlacklistedToken

Для реалізації JWT-автентифікації у системі використовується окрема група класів, пов’язаних з керуванням токенами доступу.

Клас OutstandingToken зберігає інформацію про видані refresh-токени, включаючи їх унікальний ідентифікатор та термін дії.

Клас BlacklistedToken використовується для збереження токенів, які були відкликані та не можуть бути використані повторно.

Розділення логіки токенів у окремі класи дозволяє реалізувати контроль сесій користувачів, що підвищує рівень безпеки платформи та відповідає сучасним вимогам до автентифікації вебзастосунків.

Клас Meeting

Клас Meeting описує окрему зустріч, яка є центральним елементом предметної області системи. Він містить основні метадані події: назву, дату проведення, тип зустрічі та службовий статус обробки.

Один об’єкт класу Meeting може бути пов’язаний з кількома аудіозаписами, транскриптами, шаблонами та протоколами. Така модель дозволяє повторно обробляти одну й ту саму зустріч із використанням різних шаблонів або параметрів узагальнення.

Клас MeetingAccess

Клас MeetingAccess реалізує зв’язок між користувачами та зустрічами і використовується для підтримки спільної роботи. Він містить посилання на користувача та зустріч, а також роль доступу, що визначає рівень прав користувача.

Застосування окремого класу для керування доступом дозволяє гнучко розширювати систему, додаючи нові ролі або правила взаємодії без зміни основної логіки платформи.

Клас Participant та MeetingParticipant

Клас Participant використовується для опису учасників конкретної зустрічі. Він відокремлює поняття користувача платформи від реального учасника події, що є важливим для коректного формування протоколів.

Зв’язок між зустрічами та учасниками реалізується через клас MeetingParticipant, який зберігає додаткові атрибути, зокрема роль учасника у зустрічі. Ця інформація використовується під час автоматичного формування рішень та доручень у протоколі.

Класи AudioRecord та Transcript

Клас AudioRecord відповідає за збереження інформації про аудіозапис зустрічі, включаючи формат файлу, тривалість та шлях до збереження.

Клас Transcript містить текстову транскрипцію, отриману в результаті автоматичного розпізнавання мовлення, а також службові параметри обробки.

Розділення цих сутностей дозволяє зберігати первинні дані незалежно від результатів обробки та повторно використовувати аудіозапис у разі потреби.

Клас ProtocolTemplate

Клас ProtocolTemplate описує шаблони протоколів, створені користувачем. Шаблон являє собою документ формату DOCX із вбудованими тегами, які використовуються для автоматичного заповнення структурованим текстом протоколу.

Кожен шаблон належить конкретному користувачу, але може бути пов’язаний із кількома зустрічами, що дозволяє використовувати його у спільному середовищі.

Клас Protocol

Клас Protocol представляє результат автоматичної генерації протоколу. Він містить структурований текст, посилання на відповідну зустріч, транскрипт і використаний шаблон (за наявності).

Особливістю даної моделі є те, що протокол створюється для конкретного користувача, що дозволяє формувати кілька варіантів протоколів однієї зустрічі з різними шаблонами або параметрами узагальнення.

Запропонована модель класів відображає логіку роботи платформи та забезпечує узгодженість між програмною реалізацією та структурою бази даних. Чітке розділення відповідальностей між класами спрощує підтримку системи, дозволяє масштабувати функціонал і створює основу для подальшого розвитку програмного забезпечення.

3.4 Програмна архітектура платформи

Програмна архітектура back-end частини платформи генерації протоколів зустрічей на основі аудіозаписів побудована за модульним принципом із розподілом відповідальностей між компонентами. Основною метою такого підходу є спрощення реалізації та супроводження коду, а також забезпечення зрозумілої структури проєкту, яку можна швидко розгорнути й підтримувати навіть при невеликому обсязі серверної логіки. Архітектура реалізована у межах одного застосунку FastAPI без використання мікросервісного підходу, оскільки для задач платформи достатньо централізованого сервера з REST API та єдиною базою даних.

Back-end платформи виконує такі ключові функції: аутентифікацію користувачів (JWT), керування зустрічами та доступами, приймання і збереження аудіофайлів, виклик зовнішнього сервісу розпізнавання мовлення для отримання текстового представлення, формування запиту до gpt-5-nano для узагальнення та генерації протоколу, а також збереження результатів у базі даних і надання доступу до сформованих документів. Взаємодія з клієнтською частиною здійснюється через HTTP-запити, а обмін даними відбувається переважно у форматі JSON; окремо підтримується передавання файлів у форматі multipart/form-data.

Для забезпечення керованості коду серверна частина поділяється на кілька програмних модулів, що відповідають основним підсистемам: core,

database, auth, meetings, templates, processing та integrations. Такий поділ є практичним компромісом між «надмірною» шаруватістю та повною відсутністю структури: модулі відповідають бізнес-функціям платформи і можуть розвиватися незалежно, не створюючи великої кількості дрібних директорій.

Модуль core містить загальні налаштування застосунку та інфраструктурні компоненти, що використовуються у всіх підсистемах.

Модуль database відповідає за підключення до PostgreSQL та організацію доступу до даних. У реалізації використовується SQLModel як зручний підхід, що поєднує можливості Pydantic та SQLAlchemy: сутності бази даних описуються у вигляді класів, які одночасно виконують роль ORM-моделей. Для підтримки еволюції структури бази даних використовується Alembic, що забезпечує повторюваність розгортання схеми та контроль змін під час розробки.

Модуль auth реалізує реєстрацію, вхід та видачу JWT-токенів. Паролі зберігаються лише у вигляді хешів із використанням алгоритму bcrypt (через passlib), а підпис і перевірка JWT виконуються за допомогою бібліотеки python-jose. У межах цього модуля також реалізуються залежності FastAPI для перевірки токена при доступі до захищених ресурсів. Обраний підхід дозволяє централізовано контролювати доступ до зустрічей, шаблонів та сформованих протоколів без дублювання коду перевірок у кожному ендпоінті.

Модуль meetings відповідає за керування зустрічами, учасниками та правами доступу. Саме в цьому модулі реалізується логіка “спільної зустрічі”: зустріч може належати одному користувачу, але бути доступною іншим користувачам за роллю (наприклад, перегляд або редагування). Модуль містить операції створення зустрічі, оновлення метаданих, додавання учасників та керування списком користувачів, які мають доступ до зустрічі.

Модуль templates реалізує роботу з користувацькими шаблонами протоколів. За умовами платформи кожен користувач має власні шаблони, однак при спільних зустрічах доступ до шаблонів також може бути спільним.

Шаблон зберігається як DOCX-файл із тегами (плейсхолдерами), які під час формування протоколу заповнюються структурованими даними. Якщо шаблон не обрано, система використовує стандартний режим генерації, у якому структура протоколу формується моделлю gpt-5-nano згідно з системними правилами та інструкціями.

Модуль processing є центральним з точки зору основної бізнес-функції платформи, оскільки відповідає за повний цикл обробки: збереження аудіо, отримання текстового представлення мовлення, узагальнення та генерацію протоколу, формування документа та запис результатів у базу даних. Логіка реалізується як оркестрація послідовних кроків (pipeline), що дозволяє фіксувати проміжні стани, повторно запускати окремі етапи (наприклад, лише генерацію протоколу) та коректно обробляти помилки зовнішніх сервісів.

Модуль integrations інкапсулює взаємодію із зовнішніми API. Окремо виділяються клієнти для сервісу розпізнавання мовлення та для генерації протоколу (gpt-5-nano). Такий підхід дозволяє централізувати роботу з ключами API, таймаутами, логуванням та повторними спробами при помилках. Для інтеграції використовується офіційний SDK OpenAI або HTTP-клієнт httpx; повторні спроби при тимчасових збоях реалізуються через tenacity.

Для реалізації back-end частини платформи застосовано набір бібліотек, що покриває ключові задачі вебзастосунку та інтеграцій:

— FastAPI – реалізація REST API, маршрутизація та залежності;

— Uvicorn – ASGI-сервер для запуску застосунку;

— SQLModel – опис сутностей бази даних у вигляді класів та робота з ORM;

— psycopg – драйвер для PostgreSQL;

— Alembic – міграції схеми бази даних;

— python-jose – формування та перевірка JWT;

— passlib – хешування паролів;

— python-multipart – підтримка завантаження файлів;

— openai – інтеграція з сервісами ASR та gpt-5-nano;

— httpx – HTTP-клієнт для зовнішніх запитів;

— tenacity – повторні спроби запитів при тимчасових помилках.

Для організації коду застосовано доменно-орієнтований підхід до структурування проєкту, подібний до того, який використовується в open-source платформі Netflix Dispatch. Суть підходу полягає в тому, що код групується не за «типами» файлів (окремо routers, models, services тощо), а за предметними підсистемами (domains): аутентифікація, зустрічі, шаблони, обробка. У межах кожного домену зберігаються всі необхідні компоненти (API, моделі, схеми та сервіс), що дозволяє швидко знаходити потрібний функціонал, зменшує кількість перехресних залежностей і спрощує масштабування проєкту. Практична перевага такого підходу особливо помітна під час розвитку системи: при додаванні нового модуля достатньо створити окремий домен-каталог із типовим набором файлів, не змінюючи загальну структуру проєкту.

Структура серверної частини проєкту представлена на рисунку 3.3

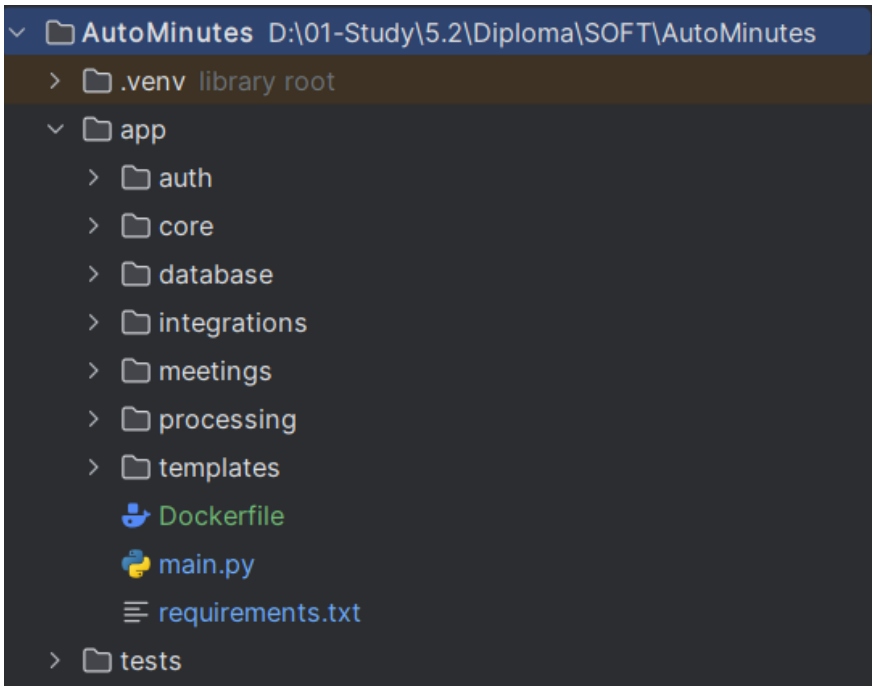


Рисунок 3.3 – Структура серверної частини проєкту

У наведеній структурі main.py виконує роль точки входу та підключає маршрути з модулів auth, meetings, templates і processing. Загальні компоненти

винесені в core, а доступ до бази даних та ORM-моделі сконцентровані в database. Окреме винесення integrations забезпечує ізоляцію зовнішніх API від бізнес-логіки та спрощує подальшу заміну або розширення інтеграцій.

Таким чином, програмна архітектура back-end частини платформи є достатньо простою для практичної реалізації, але водночас структурованою для підтримки й розвитку. Модульний поділ відповідає основним функціям системи та дозволяє реалізувати повний цикл обробки аудіозаписів і формування протоколів без надмірного ускладнення коду.

Висновки до розділу 3

У розділі 3 виконано архітектурне проєктування платформи автоматичного створення протоколів зустрічей з урахуванням функціональних вимог і специфіки обробки аудіозаписів та інтеграції з хмарними сервісами. Обґрунтовано вибір клієнт–серверної архітектури як найбільш придатної для вебзастосунку, що працює з великими файлами, зовнішніми API та потребує доступу з різних пристроїв. Сформовано загальну модель взаємодії компонентів системи (клієнт, сервер, база даних) і визначено асинхронний характер обробки запитів для тривалих операцій транскрипції й генерації протоколу.

У межах розділу також обґрунтовано технологічний стек платформи. Для серверної частини обрано Python та FastAPI як основу для реалізації REST API і підтримки асинхронної моделі виконання; для сховища даних – PostgreSQL як реляційну СУБД, що забезпечує цілісність і зв’язність даних предметної області. Окремо визначено доцільність використання зовнішніх AI-сервісів для транскрипції та узагальнення (gpt-4o-transcribe та gpt-5-nano), а також обґрунтовано відмову від NoSQL-рішень у зв’язку з наявністю чітких зв’язків між сутностями та потребою в транзакційності. Для клієнтської частини обрано Angular як фреймворк із готовою структурою, що відповідає сценаріям роботи системи та спрощує реалізацію інтерфейсу.

Крім того, спроєктовано модель даних платформи у вигляді набору класів/сутностей, на основі яких сформовано ER-діаграму, і визначено їх зв’язки (користувачі, зустрічі, доступи, учасники, аудіозаписи, транскрипти, шаблони та протоколи). Для реалізації безпечного доступу передбачено JWT-аутентифікацію з керуванням токенами та можливістю відкликання. Завершальним етапом сформовано програмну архітектуру back-end як монолітного застосунку FastAPI з модульною, доменно-орієнтованою структурою (у стилі Netflix Dispatch), що забезпечує практичну реалізованість, зрозумілу організацію коду та основу для подальшого розвитку системи.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Контейнеризація та підготовка інфраструктури застосунку

На першому етапі реалізації застосунку AutoMinutes (програмне забезпечення генерації протоколів зустрічей на основі аудіозаписів) було виконано підготовку інфраструктури запуску, щоб забезпечити відтворюваність середовища розробки, уніфікований процес старту компонентів і можливість швидкого розгортання на інших робочих станціях або в хмарі. Для цього використано технологію контейнеризації Docker та інструмент оркестрації Docker Compose.

Інфраструктура AutoMinutes на етапі розробки складається з трьох контейнерів:

— db – база даних PostgreSQL 16 для зберігання даних системи (користувачі, зустрічі, доступи, аудіозаписи, транскрипції, протоколи тощо);

— backend – серверна частина на FastAPI, що реалізує REST API та бізнес-логіку;

— frontend – клієнтська частина на Angular, запущена у режимі розробки (dev server) для перевірки інтеграції з API.

Такий підхід дозволяє запускати всю систему одночасно єдиною командою, гарантуючи коректний порядок старту компонентів та стабільність конфігурації.

Для централізації параметрів запуску створено файл .env. Використання змінних середовища дозволяє уникнути жорсткого задання конфігурації в коді та забезпечує просте перенесення проєкту між середовищами (dev/test/prod).

Приклад змісту .env (основні параметри):

```
POSTGRES_DB=autominutes
POSTGRES_USER=postgres
POSTGRES_PASSWORD=postgres
POSTGRES_PORT=5432
```

```
BACKEND_PORT=8000
FRONTEND_PORT=4200

ENV=dev
```

У подальшому ці змінні використовуються Docker Compose для ініціалізації PostgreSQL та формування рядка підключення DATABASE_URL для серверної частини.

Для контейнеризації серверної частини AutoMinutes створено файл backend/Dockerfile, який визначає процес побудови образу. На етапі побудови:

1) обирається базовий образ Python;
2) встановлюються залежності з requirements.txt;
3) копіюється вихідний код серверного застосунку;
4) задаються налаштування для коректної роботи імпортів Python-модулів.

Файл backend/Dockerfile:

```
FROM python:3.12-slim

WORKDIR /app

RUN apt-get update && apt-get install -y --no-install-recommends
\    build-essential \
        && rm -rf /var/lib/apt/lists/*

COPY backend/requirements.txt /app/requirements.txt
RUN pip install --no-cache-dir -r /app/requirements.txt

COPY backend/app /app/app
COPY backend/alembic.ini /app/alembic.ini
ENV PYTHONPATH=/app

EXPOSE 8000
```

Запуск сервера виконується командою Uvicorn через Docker Compose у режимі --reload, що дозволяє застосовувати зміни коду без повторної побудови образу.

Для запуску всіх компонентів застосунку створено конфігураційний файл docker-compose.yml, який описує сервіси, їх змінні середовища, порти, volumes та залежності запуску.

Файл docker-compose.yml:

```
services:
  db:
    image: postgres:16
    container_name: autominutes_db
    restart: unless-stopped
    env_file: .env
    environment:
      POSTGRES_DB: ${POSTGRES_DB}
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
    ports:
      - "${POSTGRES_PORT}:5432"
    volumes:
      - autominutes_postgres_data:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U $POSTGRES_USER -d $POSTGRES_DB"]
      interval: 5s
      timeout: 3s
      retries: 20
  backend:
    build:
      context: .
      dockerfile: backend/Dockerfile
    container_name: autominutes_backend
    restart: unless-stopped
```

```
    env_file: .env
    environment:
      DATABASE_URL:
postgresql+psycopg://${POSTGRES_USER}:${POSTGRES_PASSWORD}@db:5432/${P
OSTGRES_DB}
    ports:
      - "${BACKEND_PORT}:8000"
    depends_on:
      db:
        condition: service_healthy
    volumes:
      - ./backend/app:/app/app
    command: >
      uvicorn app.main:app
      --host 0.0.0.0
      --port 8000
      --reload
  frontend:
    image: node:20-alpine
    container_name: autominutes_frontend
    working_dir: /frontend
    restart: unless-stopped
    env_file: .env
    ports:
      - "${FRONTEND_PORT}:4200"
    volumes:
      - ./frontend:/frontend
    command: >
      sh -c "npm ci && npm start -- --host 0.0.0.0 --port 4200"
    depends_on:
      - backend
volumes:
  autominutes_postgres_data:
```

PostgreSQL використовує volume autominutes_postgres_data, що забезпечує збереження даних між перезапусками. Механізм healthcheck для db перевіряє готовність БД через pg_isready. backend запускається тільки після переходу db у стан healthy (через depends_on: condition: service_healthy), що усуває проблему раннього старту API без доступної БД. Код серверної частини монтується у контейнер (./backend/app:/app/app), а сервер запускається з --reload, що прискорює цикл розробки. Frontend запускається як dev server для Angular з автоматичним встановленням залежностей через npm ci.

Після підготовки файлів Docker виконується побудова образів і запуск контейнерів:

```
docker compose up --build
```

У результаті команда:

— завантажує образ PostgreSQL 16;

— будує образ серверної частини AutoMinutes із backend/Dockerfile;

— запускає всі три контейнери відповідно до описаного порядку.

Результат запуску контейнерів у терміналі необхідно зафіксувати у звіті.

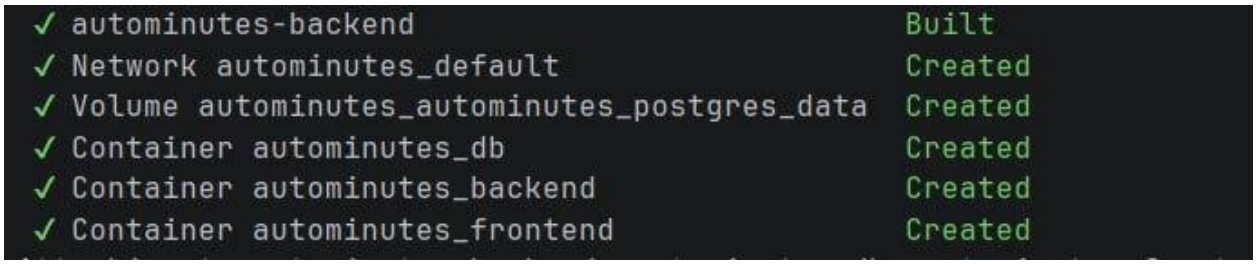


Рисунок 4.1 – Запуск AutoMinutes через команду docker compose up --build

Для перевірки стану контейнерів використовується команда:

```
docker compose ps
```

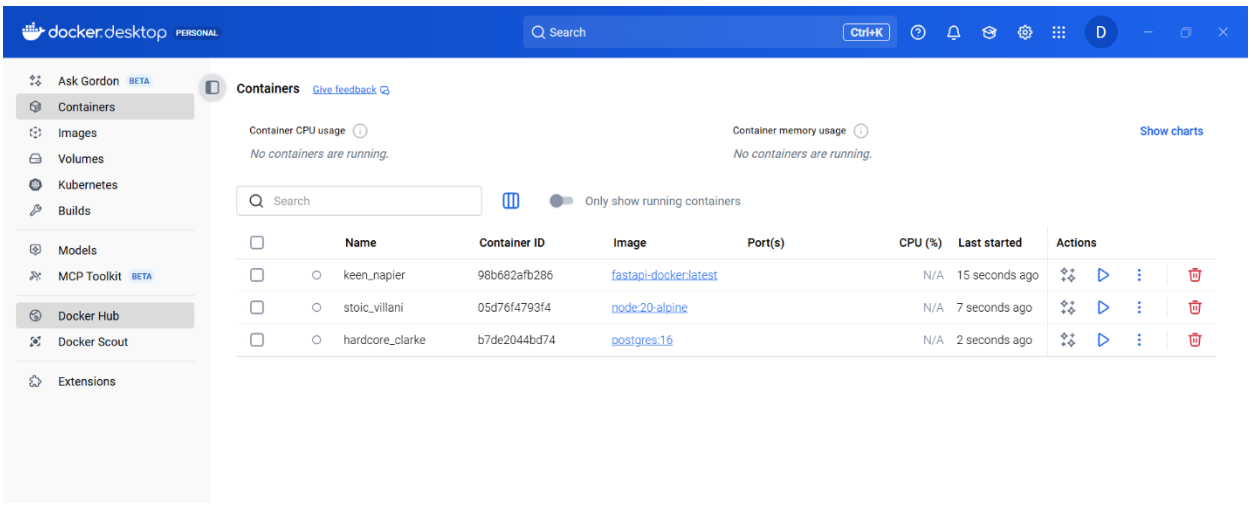


Рисунок 4.2 – Стан контейнерів AutoMinutes після запуску

Таким чином виконано контейнеризацію інфраструктури застосунку AutoMinutes за допомогою Docker та Docker Compose. Налаштовано три контейнери (PostgreSQL, FastAPI, Angular), визначено змінні середовища через .env, реалізовано механізм перевірки готовності бази даних (healthcheck) та керований порядок старту сервісів. Створене контейнеризоване середовище є основою для подальших етапів реалізації – підключення ORM, налаштування міграцій та створення моделей відповідно до структури бази даних.

4.2 Реалізація серверної частини

Після підготовки контейнеризованого середовища виконано реалізацію серверної частини застосунку AutoMinutes. Серверна частина виконує роль центрального керуючого компонента системи, оскільки саме вона відповідає за збереження даних, контроль доступу, виконання бізнес-правил та взаємодію з зовнішніми сервісами.

На етапі проєктування серверної частини було прийнято рішення використовувати FastAPI як вебфреймворк, оскільки він забезпечує високу продуктивність, підтримує асинхронну обробку запитів та автоматично генерує документацію API. Для роботи з базою даних обрано SQLModel, що дозволяє в єдиному стилі описувати структуру таблиць, типи даних та схеми обміну інформацією.

Серверна частина AutoMinutes побудована за принципом чіткого розділення відповідальностей. Кожен логічний компонент системи реалізований у вигляді окремого модуля, що дозволяє ізолювати бізнес-логіку від HTTP-рівня та забезпечити можливість подальшого розширення системи. У межах серверної частини виділено рівень доступу до даних (моделі та сесії БД), рівень бізнес-логіки (сервіси) та рівень представлення (REST API). Такий підхід мінімізує дублювання коду і спрощує тестування компонентів.

Першим практичним кроком реалізації серверної частини стало налаштування з'єднання з базою даних PostgreSQL. Для цього використовується змінна середовища DATABASE_URL, яка формується на етапі запуску Docker-контейнера. Такий підхід дозволяє не фіксувати конфігураційні параметри безпосередньо у вихідному коді, що є важливим з точки зору безпеки та переносимості застосунку. Модуль database/session.py відповідає за ініціалізацію engine SQLAlchemy, створення сесій для роботи з БД та коректне завершення з'єднання після обробки запиту.

Файл backend/app/database/session.py

```
import os
from sqlmodel import Session, create_engine
DATABASE_URL = os.getenv("DATABASE_URL", "")
engine = create_engine(
    DATABASE_URL,
    pool_pre_ping=True,
)
def get_session():
    with Session(engine) as session:
        yield session
```

Використання параметра pool_pre_ping=True дозволяє уникати помилок під час роботи з «простроченими» з'єднаннями (наприклад, після перезапуску контейнера БД), що є типово важливим у контейнеризованому середовищі.

Для опису структури бази даних використано SQLModel, оскільки ця бібліотека поєднує можливості ORM та механізми валідації даних. Це

дозволяє застосовувати типізацію Python для опису структури таблиць, автоматично перевіряти коректність даних та повторно використовувати класи як для роботи з БД, так і для обміну даними через API. У результаті кожна модель SQLModel одночасно є відображенням таблиці бази даних і типовою структурою даних для серверної логіки.

Розробку моделей даних розпочато з ключової сутності системи – користувача. Саме користувач є власником зустрічей, шаблонів протоколів та згенерованих документів.

Файл backend/app/auth/models.py

```
class User(SQLModel, table=True):
    __tablename__ = "user"
    id: UUID = Field(default_factory=uuid.uuid4,
primary_key=True)
    username: str = Field(nullable=False, unique=True)
    password_hash: str = Field(nullable=False)
    full_name: Optional[str] = None
    is_active: bool = Field(default=True, nullable=False)
    created_at: datetime = Field(
        sa_column=Column(DateTime(timezone=True),
server_default=func.now(), nullable=False)
    )
    updated_at: datetime = Field(
        sa_column=Column(
            DateTime(timezone=True),
            server_default=func.now(),
            onupdate=func.now(),
            nullable=False,
        )
    )
```

Поле id реалізовано у вигляді UUID, що дозволяє гарантувати унікальність ідентифікаторів у розподіленому середовищі. Поле password_hash використовується замість зберігання пароля у відкритому вигляді, що відповідає базовим вимогам інформаційної безпеки. Часові поля

created_at та updated_at заповнюються на рівні бази даних, що усуває потребу вручну підтримувати часові мітки в бізнес-логіці.

Сутність зустрічі є центральною в предметній області застосунку AutoMinutes. Вона пов’язує між собою користувачів, учасників, аудіозаписи та результати обробки, а також забезпечує контекст для генерації протоколів.

Файл backend/app/meetings/models.py

```
class Meeting(SQLModel, table=True):
    __tablename__ = "meeting"
    id: UUID = Field(default_factory=uuid.uuid4,
primary_key=True)
    owner_user_id: UUID = Field(foreign_key="user.id",
nullable=False)
    title: str = Field(nullable=False)
    meeting_date: Optional[datetime] = None
    meeting_type: Optional[str] = None
    location: Optional[str] = None
    description: Optional[str] = None
    status: str = Field(default="DRAFT", nullable=False)
    created_at: datetime = Field(
        sa_column=Column(DateTime(timezone=True),
server_default=func.now(), nullable=False)
    )
    updated_at: datetime = Field(
        sa_column=Column(
            DateTime(timezone=True),
            server_default=func.now(),
            onupdate=func.now(),
            nullable=False,
        )
    )
```

Поле owner_user_id визначає власника зустрічі та використовується для контролю доступу. Поле status дозволяє відстежувати стан зустрічі на різних етапах її обробки та керувати допустимими діями в бізнес-логіці.

Безпосередня робота з моделями БД винесена у сервісний шар. Основною причиною такого рішення є необхідність централізувати бізнес-логіку та уникнути її дублювання у різних API-ендпоінтах. Сервіс відповідає за створення та оновлення об’єктів, перевірку бізнес-умов і узгоджену взаємодію між сутностями.

Файл backend/app/meetings/service.py

```
class MeetingService:
    def create_meeting(self, session: Session, owner_user_id:
UUID, data) -> Meeting:
        meeting = Meeting(
            owner_user_id=owner_user_id,
            title=data.title,
            description=getattr(data, "description", None),
            meeting_date=getattr(data, "meeting_date", None),
            meeting_type=getattr(data, "meeting_type", None),
            location=getattr(data, "location", None),
        )
        session.add(meeting)
        session.commit()
        session.refresh(meeting)
        return meeting
```

Використання session.refresh(meeting) після commit() забезпечує повернення повного об’єкта з актуальними значеннями (зокрема id та часовими полями), що важливо для подальших операцій у межах одного запиту.

HTTP-рівень реалізовано з використанням FastAPI Router. Кожен ендпоінт виконує одну задачу – приймає запит, передає дані у відповідний сервіс та повертає результат. Таким чином API не містить бізнес-логіки, а виконує роль проміжного шару між клієнтом і серверною логікою.

Файл backend/app/meetings/api.py

```
router = APIRouter(prefix="/meetings", tags=["meetings"])
meeting_service = MeetingService()
```

```
@router.post("", response_model=MeetingRead)
def create_meeting(
    data: MeetingCreate,
    session: Session = Depends(get_session),
    current_user: User = Depends(get_current_user),
):
    return meeting_service.create_meeting(
        session=session,
        owner_user_id=current_user.id,
        data=data,
    )
```

Оскільки застосунок AutoMinutes передбачає роботу з конфіденційною інформацією (аудіозаписи зустрічей, транскрипції, протоколи), важливим етапом реалізації серверної частини стало впровадження механізмів контролю доступу. На рівні предметної області визначено, що кожна зустріч має власника, до зустрічі можуть мати доступ інші користувачі, а рівень доступу може відрізнятися (перегляд або редагування). Для цього у базі даних передбачено таблицю meeting_access, яка реалізує зв’язок «багато-до-багатьох» між користувачами та зустрічами.

Файл backend/app/meetings/models.py

```
class MeetingAccess(SQLModel, table=True):
    __tablename__ = "meeting_access"
    meeting_id: UUID = Field(foreign_key="meeting.id",
primary_key=True)
    user_id: UUID = Field(foreign_key="user.id",
primary_key=True)
    access_level: str = Field(default="VIEWER", nullable=False)
    created_at: datetime = Field(
        sa_column=Column(DateTime(timezone=True),
server_default=func.now(), nullable=False)
    )
```

Поле access_level використовується для визначення прав користувача. Такий підхід дозволяє гнучко розширювати систему ролей без зміни загальної архітектури.

Для ідентифікації користувачів у серверній частині використовується токенна автентифікація на основі JWT. Це рішення обрано з огляду на простоту інтеграції з клієнтською частиною, відсутність необхідності зберігати стан сесій на сервері та можливість масштабування. У модулі core/security.py визначаються базові параметри безпеки (секретний ключ, алгоритм, час життя токена). Токен містить ідентифікатор користувача та використовується у захищених API-ендпоінтах, що дозволяє серверу визначати поточного користувача без повторних запитів до бази даних під час обробки одного запиту.

Окрему роль у системі відіграють учасники зустрічей. На відміну від користувачів, учасники можуть не мати облікового запису в AutoMinutes, однак беруть участь у конкретній зустрічі. Для цього реалізовано сутності Participant та MeetingParticipant.

Файл backend/app/meetings/models.py

```
class Participant(SQLModel, table=True):
    __tablename__ = "participant"
    id: UUID = Field(default_factory=uuid.uuid4,
primary_key=True)
    full_name: str = Field(nullable=False)
    email: Optional[str] = None
    position: Optional[str] = None
    organization: Optional[str] = None
    created_at: datetime = Field(
        sa_column=Column(DateTime(timezone=True),
server_default=func.now(), nullable=False)
    )
class MeetingParticipant(SQLModel, table=True):
    __tablename__ = "meeting_participant"
```

```
        meeting_id: UUID = Field(foreign_key="meeting.id",
primary_key=True)
        participant_id: UUID = Field(foreign_key="participant.id",
primary_key=True)
        role: Optional[str] = None
        is_present: bool = Field(default=True, nullable=False)
        notes: Optional[str] = None
```

Такий підхід дозволяє повторно використовувати учасників у різних зустрічах і зберігати контекстну інформацію про роль учасника в межах конкретної події.

На завершальному етапі реалізації базового бекенду було закладено основу для подальшої обробки аудіозаписів та генерації протоколів. Для цього передбачено сутність audio_record, яка зберігає метадані про файл (назва, тип, розмір, шлях збереження тощо) і прив’язується до конкретної зустрічі.

Файл backend/app/processing/models.py

```
class AudioRecord(SQLModel, table=True):
    __tablename__ = "audio_record"
    id: UUID = Field(default_factory=uuid.uuid4,
primary_key=True)
    meeting_id: UUID = Field(foreign_key="meeting.id",
nullable=False)
    original_filename: str = Field(nullable=False)
    content_type: Optional[str] = None
    file_size_bytes: Optional[int] = None
    storage_url: str = Field(nullable=False)
    duration_seconds: Optional[int] = None
    created_at: datetime = Field(
        sa_column=Column(DateTime(timezone=True),
server_default=func.now(), nullable=False)
    )
```

Реалізація безпосередньої обробки аудіо та генерації тексту винесена в окремий підрозділ, оскільки вона пов’язана з інтеграцією зовнішніх сервісів розпізнавання мовлення та мовних моделей.

Таким чином розглянуто цикл реалізації серверної частини застосунку AutoMinutes. Описано архітектурні принципи побудови бекенду, налаштування підключення до бази даних, реалізацію моделей предметної області відповідно до затвердженої структури БД, побудову сервісного шару бізнес-логіки, механізми контролю доступу та підготовку основи для подальшої обробки аудіоданих. Запропонована архітектура забезпечує чітке розділення відповідальностей між компонентами системи та створює основу для інтеграції із сервісами розпізнавання мовлення та генерації протоколів зустрічей.

4.3 Інтеграція з сервісами OpenAI

Наступним етапом стала інтеграція застосунку AutoMinutes із сервісами платформи OpenAI. OpenAI використовується в системі для виконання двох основних задач: автоматичного розпізнавання мовлення з аудіозаписів зустрічей (ASR) та генерації структурованого протоколу зустрічі на основі отриманої транскрипції. Інтеграція реалізована таким чином, щоб усі зовнішні виклики до API OpenAI були ізольовані в окремому модулі integrations, тоді як бізнес-логіка (ініціація процесів обробки, зміна статусів, збереження результатів у базі даних) зосереджена в сервісному шарі модуля processing.

Платформа OpenAI є платним сервісом. Для виконання розробки та тестування функціональності застосунку було поповнено баланс на суму 5 доларів США, чого більш ніж достатньо для демонстраційних сценаріїв. Це пояснюється тим, що в межах навчального проєкту кількість оброблюваних зустрічей та тривалість аудіозаписів є обмеженими, а витрати контролюються через механізми білінгу платформи.

Для роботи із сервісами OpenAI необхідно створити окремий проєкт та згенерувати API-ключ доступу. API-ключ є конфіденційним параметром і не зберігається безпосередньо у вихідному коді застосунку. Замість цього він передається через змінні середовища (наприклад, за допомогою файлу .env у

конфігурації Docker Compose), що відповідає принципам безпечної розробки програмного забезпечення та зменшує ризик компрометації доступу.

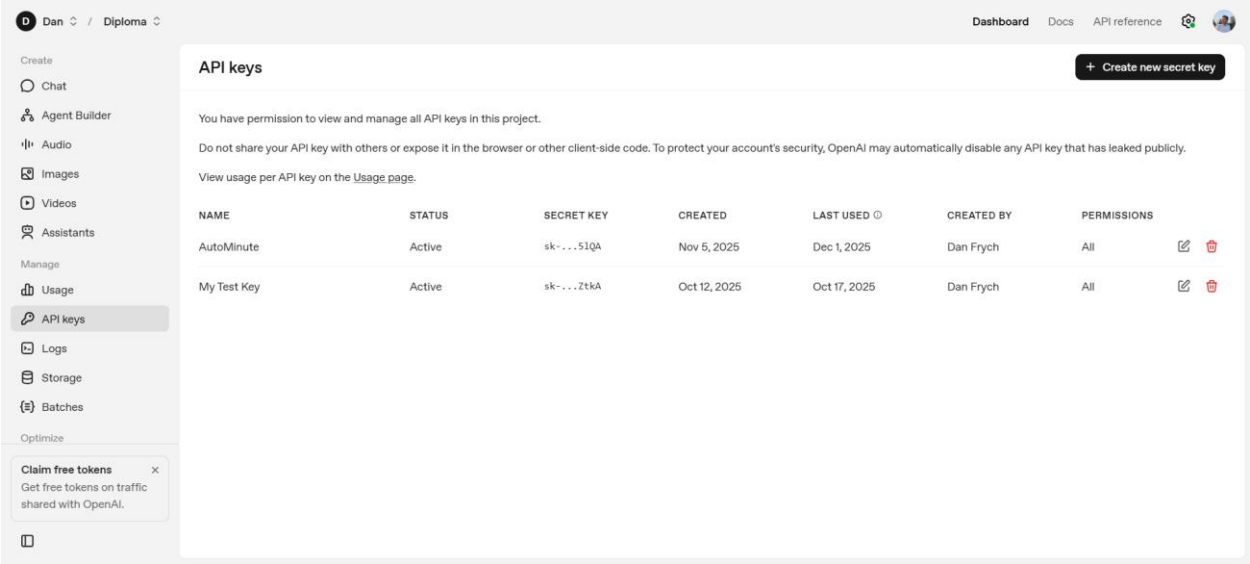


Рисунок 4.3 – Інтерфейс створення API-ключа в OpenAI Dashboard

Для уніфікованого доступу до параметрів конфігурації в серверній частині визначено окремий модуль налаштувань, який включає API-ключ OpenAI та ідентифікатори моделей, що використовуються для розпізнавання мовлення та генерації тексту. Такий підхід дозволяє централізовано змінювати моделі або параметри інтеграції без модифікації бізнес-логіки застосунку.

Файл backend/app/core/config.py (фрагмент):

```
import os
from dataclasses import dataclass
@dataclass(frozen=True)
class Settings:
    DATABASE_URL: str = os.getenv("DATABASE_URL", "")
    OPENAI_API_KEY: str = os.getenv("OPENAI_API_KEY", "")
    OPENAI_ASR_MODEL: str = os.getenv("OPENAI_ASR_MODEL", "gpt-
4o-transcribe")
    OPENAI_LLM_MODEL: str = os.getenv("OPENAI_LLM_MODEL", "gpt-5-
nano")
settings = Settings()
```

Щоб інтеграція з OpenAI була ізольованою та придатною до тестування, доступ до API реалізовано у вигляді двох окремих клієнтів: asr_client.py для виконання транскрипції аудіо та gpt_client.py для генерації текстових протоколів. Такий поділ відповідає принципу єдиної відповідальності та дозволяє незалежно розвивати або замінювати кожний компонент інтеграції.

Під час реалізації клієнта автоматичного розпізнавання мовлення було враховано необхідність прийому аудіофайлу, передачі його до моделі транскрибування та повернення нормалізованого результату, що включає текст транскрипції та, за потреби, визначену мову мовлення. Нижче наведено спрощений, але повністю працездатний каркас клієнта.

Файл backend/app/integrations/asr_client.py:

```
from dataclasses import dataclass
from typing import Optional
from openai import OpenAI
from app.core.config import settings
@dataclass
class AsrResult:
    text: str
    language: Optional[str] = None
class AsrClient:
    def __init__(self) -> None:
        if not settings.OPENAI_API_KEY:
            raise RuntimeError("OPENAI_API_KEY is not set")
        self._client = OpenAI(api_key=settings.OPENAI_API_KEY)
    def transcribe(self, audio_path: str) -> AsrResult:
        with open(audio_path, "rb") as f:
            resp = self._client.audio.transcriptions.create(
                model=settings.OPENAI_ASR_MODEL,
                file=f,
            )
        return AsrResult(
            text=resp.text,
            language=getattr(resp, "language", None)
```

```
        )
```

Після отримання транскрипції система переходить до генерації протоколу зустрічі. Ключовим аспектом на цьому етапі є формування результату таким чином, щоб він був прогнозованим, структурованим і відповідав вимогам протоколювання. Для цього в AutoMinutes використовується керований промпт, який поєднує метадані зустрічі (meeting), перелік учасників (meeting_participant та participant), текст транскрипції (transcript.text) та опис шаблону протоколу (protocol_template). Поле tags_schema, що зберігається у форматі JSON, виступає специфікацією структури вихідного документа.

Промпт сформовано так, щоб мовна модель повертала результат у структурованому форматі Markdown із заголовками, списками та логічними секціями. У межах навчального проєкту цього підходу достатньо, а збереження згенерованого тексту здійснюється у полі protocol.content бази даних.

Для виконання генерації використовується окремий клієнт мовної моделі.

Файл backend/app/integrations/gpt_client.py:

```
from openai import OpenAI
from app.core.config import settings
class GptClient:
    def __init__(self) -> None:
        if not settings.OPENAI_API_KEY:
            raise RuntimeError("OPENAI_API_KEY is not set")
        self._client = OpenAI(api_key=settings.OPENAI_API_KEY)
    def generate_protocol(self, system_prompt: str, user_prompt:
str) -> str:
        resp = self._client.chat.completions.create(
            model=settings.OPENAI_LLM_MODEL,
            messages=[
                {"role": "system", "content": system_prompt},
```

```
                {"role": "user", "content": user_prompt},
            ],
        )
        return resp.choices[0].message.content or ""
```

Для зручності та повторного використання логіки побудови промптів у модулі templates реалізовано окремі функції формування системного та користувацького повідомлень. Важливо, що система підтримує роботу як у типовому режимі (без шаблону), так і з використанням користувацького шаблону, коли структура протоколу визначається полем tags_schema.

Файл backend/app/templates/service.py (фрагмент):

```
import json
from typing import Optional
def build_system_prompt() -> str:
    return (
        "Ти – асистент для протоколювання зустрічей. "
        "Твоє завдання – на основі транскрипції та метаданих зустрічі "
        "сформувати чіткий структурований протокол українською мовою. "
        "Пиши лаконічно, без вигаданих фактів. "
    )
def build_user_prompt(
    meeting_title: str,
    meeting_date_iso: Optional[str],
    participants_text: str,
    transcript_text: str,
    tags_schema: Optional[dict],
) -> str:
    schema_block = ""
    if tags_schema:
        schema_block = (
            "\n\nСтруктура протоколу (tags_schema):\n"
```

```
                f"{json.dumps(tags_schema, ensure_ascii=False,
indent=2)}\n"
                "Дотримуйся цієї структури як орієнтира для секцій."
            )
        return (
            f"Назва зустрічі: {meeting_title}\n"
            f"Дата/час: {meeting_date_iso or 'не вказано'}\n\n"
            f"Учасники:\n{participants_text}\n\n"
            f"Транскрипція:\n{transcript_text}"
            f"{schema_block}\n\n"
            "Вихід: сформуй протокол у Markdown зі стандартними
секціями: "
            "Порядок денний, Обговорення, Рішення, Завдання/Action
items, Підсумок."
        )
```

Після підготовки клієнтів і промптів реалізовано сервіс обробки, який об’єднує всі етапи в єдиний сценарій: створення запису аудіо (audio_record), запуск транскрипції (transcript зі статусом PROCESSING), збереження тексту транскрипції, генерацію протоколу (protocol) та оновлення статусів. Це дозволяє ініціювати обробку однією операцією на рівні API та отримувати результат через подальші запити клієнта.

Таким чином, у межах цього підрозділу реалізовано повноцінну інтеграцію AutoMinutes з сервісами OpenAI. Серверна частина отримала можливість автоматично перетворювати аудіозаписи зустрічей у текстові транскрипції та генерувати структуровані протоколи. Побудована архітектура дозволяє легко змінювати моделі або параметри генерації та є логічним продовженням серверної частини, описаної у попередньому підрозділі.

4.4 Реалізація користувацького інтерфейсу

Після реалізації серверної частини та інтеграції з сервісами штучного інтелекту наступним етапом стала розробка клієнтської частини застосунку

AutoMinutes. Основне призначення клієнтської частини – забезпечити користувачу зрозумілий та послідовний сценарій роботи із системою: автентифікація, створення зустрічі, керування учасниками, завантаження аудіозапису, запуск обробки, перегляд транскрипції та перегляд/завантаження згенерованого протоколу.

Для реалізації інтерфейсу використано фреймворк Angular, оскільки він надає компонентну архітектуру, розвинену систему маршрутизації, інструменти для роботи з формами та зручні механізми для інтеграції з REST API. У результаті клієнтська частина виступає як окремий модуль у клієнт–серверній архітектурі та взаємодіє з бекендом виключно через HTTP-запити до API.

Під час реалізації UI важливо було зберегти чітке розділення відповідальностей: компоненти відповідають за відображення та взаємодію з користувачем, сервіси – за звернення до API, а спільні механізми (зокрема автентифікація) – винесені в окремі утиліти та перехоплювачі. Такий підхід дозволяє забезпечити супровідність коду та спрощує розширення функціоналу.

Початковим кроком було створення Angular-проєкту та підготовка середовища виконання. У контексті Docker-розгортання застосунок запускається в контейнері autominutes_frontend, де виконуються команди встановлення залежностей і старту dev-сервера (порт 4200). Це дозволяє розробляти інтерфейс незалежно від ОС, узгодити версії Node.js та залежностей, а також зробити запуск проєкту відтворюваним на будь-якому середовищі.

Далі було реалізовано конфігурацію взаємодії з сервером. Для цього визначено базову адресу API (наприклад, через environment.ts), щоб у всіх сервісах використовувався єдиний endpoint. Це критично важливо для подальшого розгортання, оскільки при деплої може змінюватися домен або порт бекенду, але код клієнтської частини при цьому не повинен переписуватися.

Файл src/environments/environment.ts

```
export const environment = {
  production: false,
  apiBaseUrl: 'http://localhost:8000'
};
```

Після налаштування базової конфігурації було реалізовано механізм автентифікації користувача. Оскільки на серверній частині застосовано JWT, клієнтський застосунок має виконувати дві ключові задачі: отримати токен після входу; автоматично додавати токен до всіх захищених запитів. Для цього створено сервіс авторизації, який відповідає за збереження токена (наприклад, у LocalStorage) та за надання поточного стану автентифікації іншим компонентам.

Файл src/app/core/auth/auth.service.ts

```
@Injectable({ providedIn: 'root' })
export class AuthService {
  private readonly tokenKey = 'autominutes_token';
  setToken(token: string): void {
    localStorage.setItem(this.tokenKey, token);
  }
  getToken(): string | null {
    return localStorage.getItem(this.tokenKey);
  }
  clear(): void {
    localStorage.removeItem(this.tokenKey);
  }
  isAuthenticated(): boolean {
    return !!this.getToken();
  }
}
```

Щоб не дублювати додавання токена вручну в кожному запиті, було реалізовано HTTP-interceptor, який перехоплює вихідні запити й додає заголовок Authorization: Bearer <token> для звернень до API.

Файл src/app/core/http/auth.interceptor.ts

```
@Injectable()
export class AuthInterceptor implements HttpInterceptor {
  constructor(private auth: AuthService) {}
  intercept(req: HttpRequest<any>, next: HttpHandler) {
    const token = this.auth.getToken();
    if (!token) return next.handle(req);
    const authReq = req.clone({
      setHeaders: { Authorization: `Bearer ${token}` }
    });
    return next.handle(authReq);
  }
}
```

Після цього була налаштована маршрутизація застосунку. Основна логіка маршрутизації полягає в тому, що неавтентифікований користувач має доступ лише до сторінки входу, тоді як решта сторінок (зустрічі, шаблони, обробка, протоколи) – доступні лише після успішного входу. Для цього реалізовано Guard, який перевіряє стан автентифікації та у разі потреби перенаправляє користувача на екран входу.

Далі було реалізовано ключові сторінки інтерфейсу, які відповідають основним сценаріям роботи користувача в AutoMinutes.

Першою створювалася сторінка входу. Вона є початковою точкою роботи системи та містить форму введення облікових даних. Після успішної авторизації токен зберігається локально, а користувач перенаправляється до розділу зі списком зустрічей.

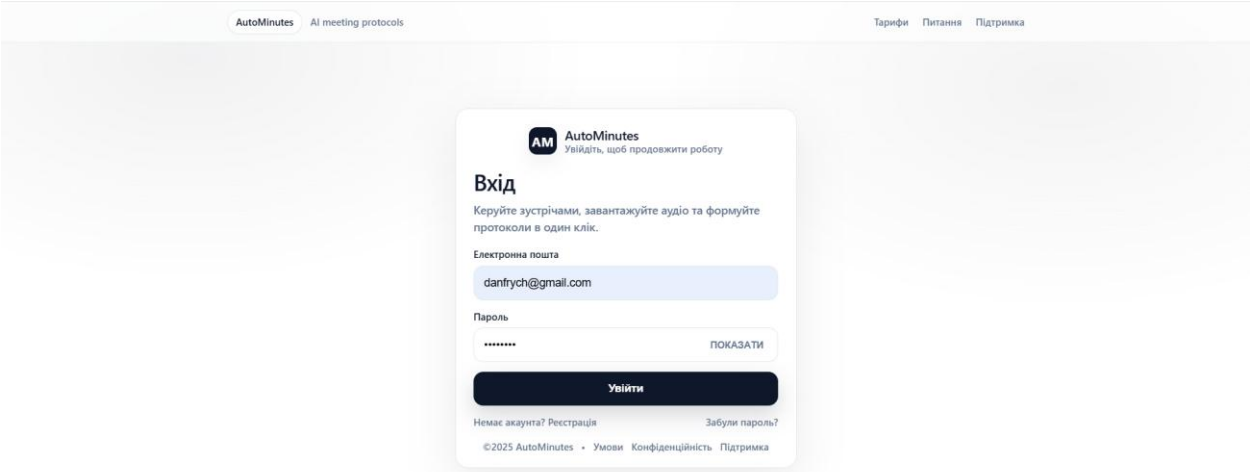


Рисунок 4.4 – Сторінка входу в AutoMinutes

Після входу користувач потрапляє на сторінку зі списком зустрічей. На цій сторінці відображається перелік зустрічей, що належать користувачу, а також елементи керування для створення нової зустрічі. Ця сторінка є центральною навігаційною точкою, оскільки саме з неї користувач переходить до конкретної зустрічі та запускає процес обробки аудіо.

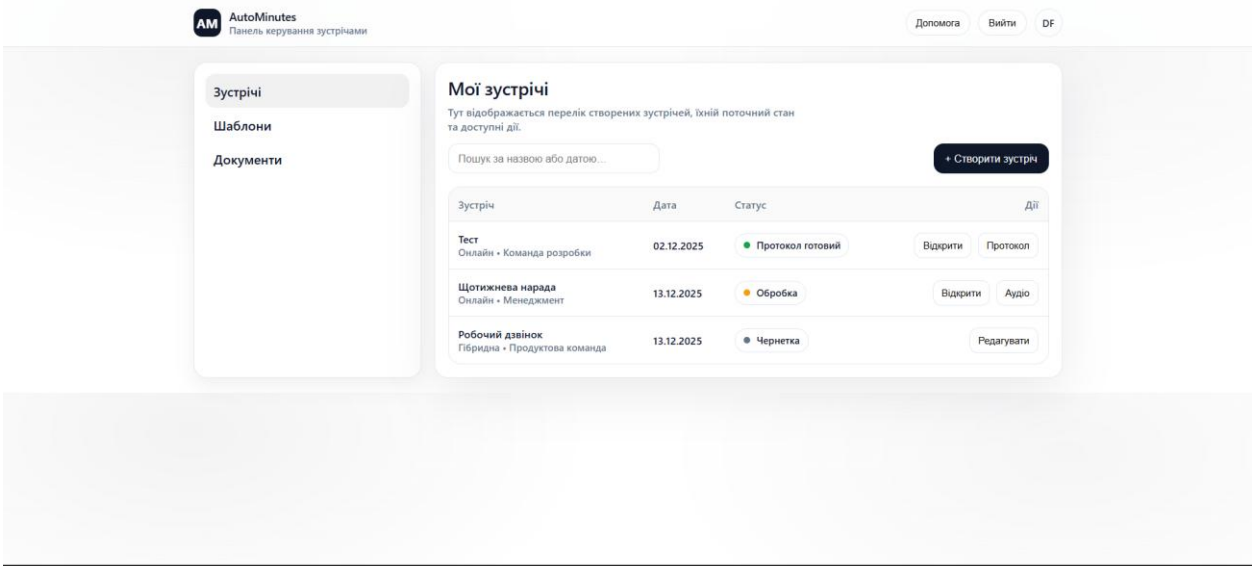


Рисунок 4.5 – Список зустрічей користувача

Для взаємодії зі списком зустрічей створено Angular-сервіс, який викликає бекенд-ендпоінти. Рішення винести HTTP-логіку в сервіси дозволяє компонентам залишатися “тонкими”, тобто відповідальними лише за відображення та реакцію на дії користувача.

Файл src/app/features/meetings/meetings.api.ts

```
@Injectable({ providedIn: 'root' })
export class MeetingsApi {
  constructor(private http: HttpClient) {}
  list() {
    return
this.http.get<any[]>(`${environment.apiBaseUrl}/meetings`);
  }
  create(payload: { title: string; description?: string }) {
    return this.http.post(`${environment.apiBaseUrl}/meetings`,
payload);
  }
}
```

Наступною реалізовано сторінку деталей зустрічі. Вона містить основні атрибути зустрічі (назва, дата, місце, опис), а також блоки керування учасниками, аудіозаписом і результатами обробки. Така сторінка побудована за принципом “єдиного робочого простору”: користувач не повинен переходити між багатьма екранами для виконання основного сценарію.

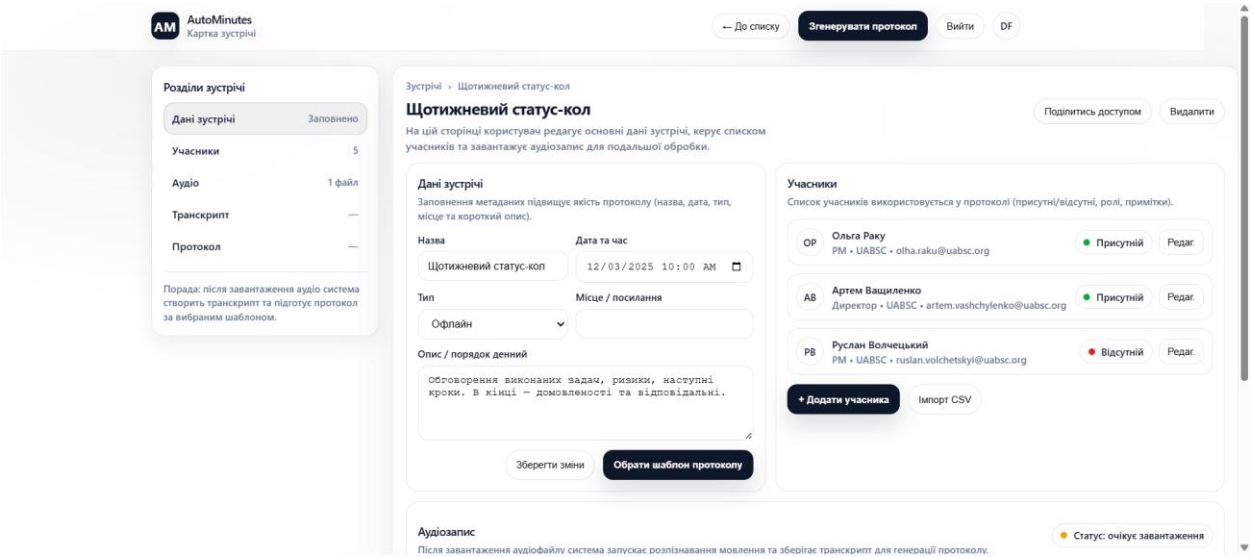


Рисунок 4.6 – Сторінка зустрічі: дані, учасники та блок аудіо

Важливою частиною UI став сценарій завантаження аудіозапису. На цій сторінці/панелі користувач вибирає файл, завантажує його на сервер і після цього запускає обробку. На інтерфейсному рівні це реалізовано як послідовність станів:

1) файл не вибрано;
2) файл завантажується;
3) файл завантажено;
4) обробка виконується;
5) готово / помилка.

Такий підхід робить поведінку застосунку прозорою, а користувач чітко розуміє, що відбувається в системі.

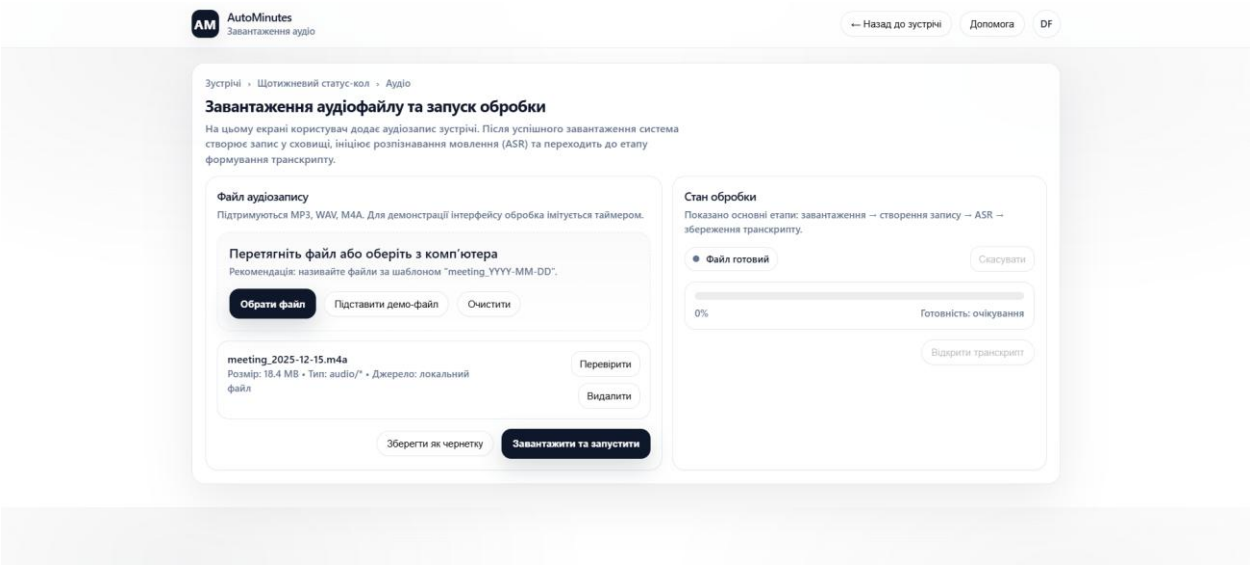


Рисунок 4.7 – Завантаження аудіофайлу та запуск обробки

Після завершення розпізнавання мовлення користувач отримує доступ до транскрипції. У UI транскрипція відображається як окремий блок або вкладка. Це важливо з практичної точки зору: користувач повинен мати можливість переглянути текст до генерації протоколу або принаймні переконатися в якості розпізнавання. Якщо транскрипція не створена або під час обробки сталася помилка, інтерфейс відображає відповідний статус і повідомлення.

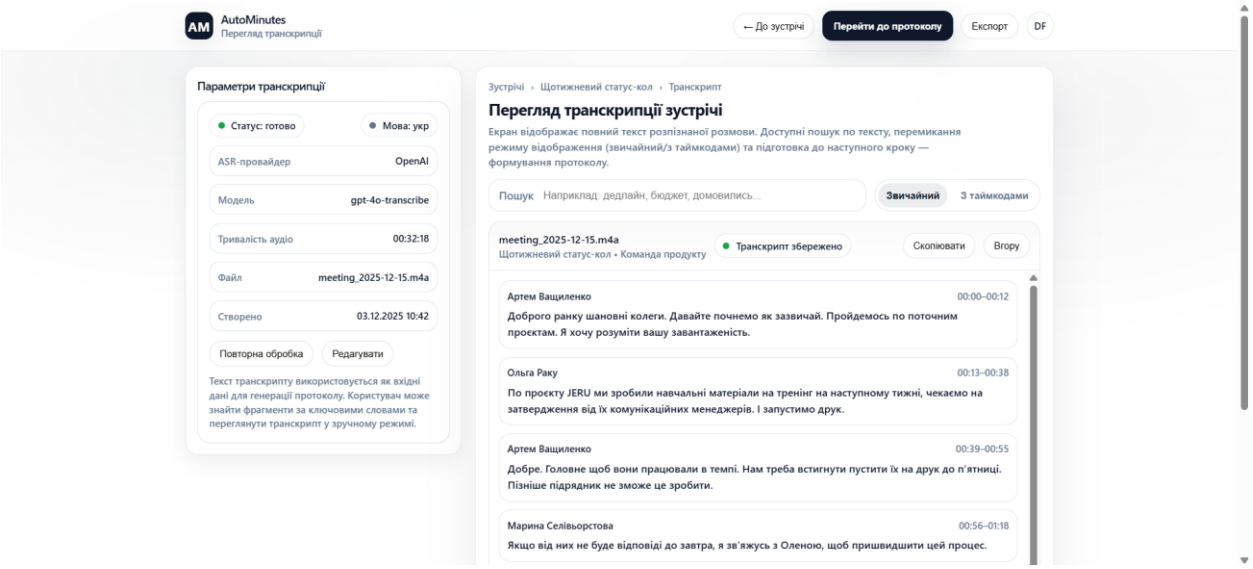


Рисунок 4.8 – Перегляд транскрипції зустрічі

Наступний етап – генерація протоколу. Користувач запускає генерацію, після чого система відображає результат у структурованому вигляді. UI передбачає як перегляд протоколу в браузері, так і можливість його експорту/завантаження (надалі – в PDF/DOCX після реалізації відповідного блоку серверної генерації файлів). Результат генерації має зберігатися в системі, тому інтерфейс дозволяє повертатися до вже створених протоколів без повторного запуску обробки.

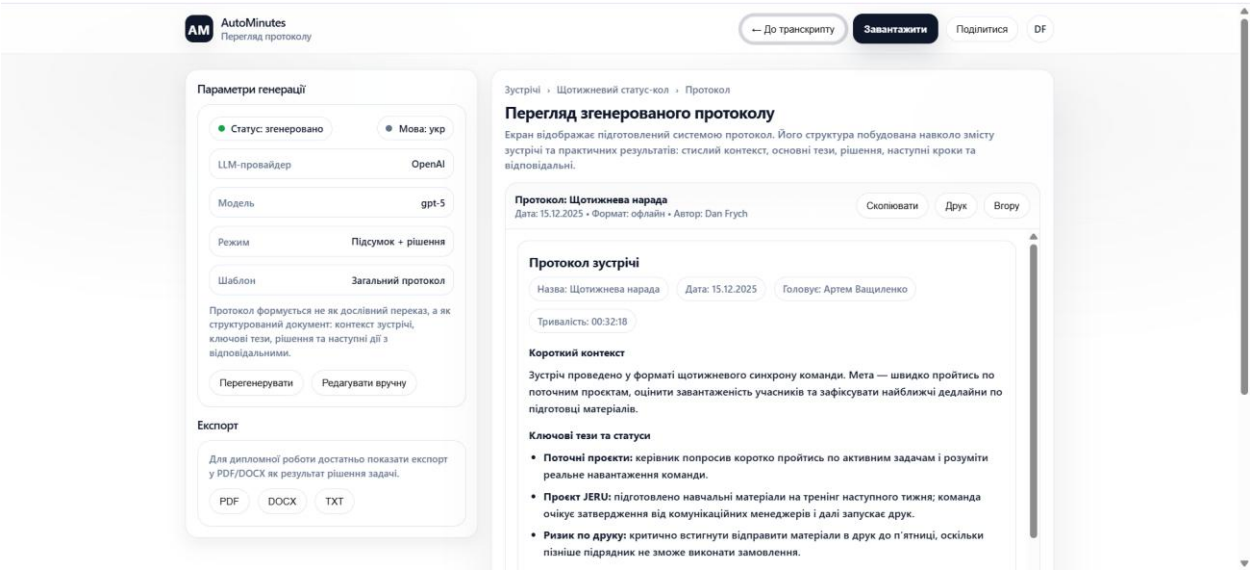


Рисунок 4.9 – Перегляд згенерованого протоколу

Окремо реалізовано розділ шаблонів протоколів. У ньому користувач може переглядати доступні шаблони, завантажувати нові, керувати доступом

та прив’язувати шаблон до конкретної зустрічі. Такий розділ є важливим, оскільки в реальних сценаріях різні організації використовують різні формати протоколів, а шаблони дозволяють адаптувати систему під конкретні вимоги.

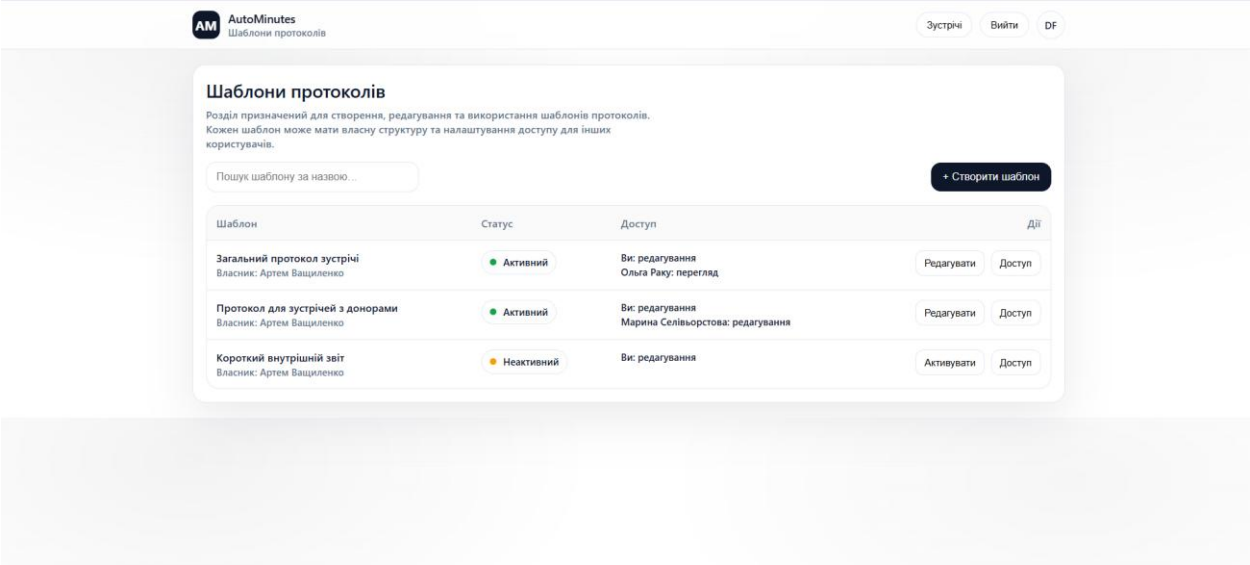


Рисунок 4.10 – Розділ шаблонів протоколів і керування доступом

У результаті виконаної роботи клієнтська частина AutoMinutes реалізує повний базовий сценарій користувача: від входу та створення зустрічі до запуску обробки аудіо й отримання результатів. Побудована структура Angular-застосунку з поділом на компоненти, сервіси та спільні механізми (interceptor/guard) забезпечує підтримуваність коду та створює основу для подальшого розвитку інтерфейсу, зокрема додавання розширених сценаріїв (спільна робота, редагування протоколів, історія генерацій, управління файлами документів).

4.5 Тестування програмного забезпечення

Тестування програмного забезпечення AutoMinutes проводилось з метою перевірки коректності реалізованих функціональних можливостей системи, а також підтвердження узгодженої взаємодії між серверною частиною, базою даних і користувацьким інтерфейсом. Основна увага приділялася перевірці роботи системи в межах типових сценаріїв використання з позиції кінцевого користувача.

З огляду на прикладний характер розроблюваного програмного забезпечення та його орієнтацію на автоматизацію процесу створення протоколів зустрічей, у межах даної роботи було застосовано функціональне (сценарне) тестування. Такий підхід передбачає перевірку коректності роботи системи шляхом виконання основних користувацьких дій без застосування окремих автоматизованих тестових фреймворків. Обраний метод є доцільним для вебзастосунків прототипного та сервісного типу, оскільки дозволяє оцінити працездатність системи в реальних умовах її експлуатації.

Тестування проводилося у середовищі, розгорнутому за допомогою Docker, що включало серверну частину на базі FastAPI, реляційну базу даних PostgreSQL та клієнтський інтерфейс. Це забезпечило стабільність тестового середовища та відтворюваність результатів перевірки.

У процесі тестування було перевірено основні функціональні сценарії роботи системи, які охоплюють повний цикл обробки даних, від авторизації користувача до формування та перегляду згенерованого протоколу. Перелік перевірених сценаріїв наведено в таблиці 4.1.

Таблиця 4.1 – Результати функціонального тестування

№	Функціональний сценарій	Вхідні дані	Очікуваний результат	Результат
1	Авторизація користувача	Логін, пароль	Успішний вхід у систему	Успішно
2	Перегляд списку зустрічей	Дані користувача	Відображення зустрічей користувача	Успішно
3	Створення нової зустрічі	Назва, дата, опис	Створення запису в БД	Успішно
4	Додавання учасників	ПІБ, роль	Збереження учасників	Успішно
5	Завантаження аудіофайлу	Файл формату mp3	Файл збережено	Успішно
6	Запуск транскрипції	Аудіозапис	Створення транскрипції	Успішно
7	Перегляд транскрипції	Дані транскрипції	Коректне відображення тексту	Успішно
8	Генерація протоколу	Транскрипція, шаблон	Створення протоколу	Успішно

Продовження таблиці 4.2

№	Функціональний сценарій	Вхідні дані	Очікуваний результат	Результат
9	Перегляд протоколу	Згенерований текст	Коректне відображення	Успішно
10	Керування шаблонами	Дані шаблонів	Відображення та редагування	Успішно
11	Керування доступом	Користувач, рівень доступу	Зміна прав доступу	Успішно

У результаті проведеного тестування підтверджено коректну роботу основних функціональних модулів системи AutoMinutes. Усі перевірені сценарії виконувалися відповідно до очікуваних результатів, що свідчить про правильність реалізації бізнес-логіки, узгодженість структури бази даних та стабільну взаємодію між компонентами системи.

Отримані результати дозволяють зробити висновок, що розроблене програмне забезпечення є працездатним, відповідає поставленим вимогам та може бути використане для автоматизації процесу створення протоколів зустрічей у реальних умовах.

Висновки до розділу 4

У розділі 4 подано практичні результати програмної реалізації застосунку AutoMinutes та послідовність виконаних робіт від підготовки середовища до отримання кінцевого результату. Розгорнуто контейнеризовану інфраструктуру на базі Docker і Docker Compose, що забезпечує відтворюваний запуск компонентів системи (PostgreSQL, FastAPI, Angular), керований порядок старту сервісів та зручний цикл розробки.

Описано реалізацію серверної частини на FastAPI з використанням SQLModel відповідно до затвердженої структури бази даних. Впроваджено розподіл на модулі та шари (доступ до даних, бізнес-логіка, API), що забезпечує зрозумілу організацію коду, спрощує розширення функціоналу та підтримує контроль доступу до ключових об’єктів системи. Також виконано

інтеграцію із сервісами OpenAI для розпізнавання мовлення з аудіозаписів і генерації структурованого протоколу на основі транскрипції.

Окремо представлено реалізацію користувацького інтерфейсу та відображення повного сценарію роботи користувача: вхід у систему, керування зустрічами й учасниками, завантаження аудіо, перегляд транскрипції та перегляд згенерованого протоколу, а також роботу із шаблонами протоколів і доступами. Проведене функціональне тестування за ключовими сценаріями підтвердило коректність взаємодії між компонентами системи та працездатність AutoMinutes у межах поставлених вимог.

ВИСНОВКИ

Порівняно з існуючими рішеннями для ведення протоколів зустрічей, розроблене Програмне забезпечення генерації протоколів зустрічей на основі аудіозаписів має такі переваги: автоматизація повного циклу обробки аудіозаписів без ручного втручання, використання сучасних сервісів автоматичного розпізнавання мовлення та великих мовних моделей, гнучке формування структурованих протоколів відповідно до заданих правил, а також можливість інтеграції в існуючі робочі процеси.

Практична цінність результатів роботи полягає у зменшенні часових витрат на документування зустрічей, зниженні впливу людського фактора та підвищенні якості фіксації домовленостей. Розроблений застосунок може бути використаний у бізнес-середовищі, IT-командах, освітніх закладах, громадських та управлінських організаціях для протоколювання нарад, інтерв'ю, робочих зустрічей та інших форм усної комунікації. Запропоноване рішення дозволяє створювати ефективні інформаційні системи без потреби у спеціалізованому обладнанні, використовуючи лише наявні цифрові ресурси.

Основні результати виконання кваліфікаційної роботи відповідають поставленим завданням:

проаналізовано предметну область автоматизованого протоколювання зустрічей та існуючі підходи до обробки аудіоінформації;

сформульовано функціональні та нефункціональні вимоги до системи автоматичного створення протоколів;

— спроєктовано клієнт-серверну архітектуру вебзастосунку;

— реалізовано back-end частину системи з використанням фреймворку FastAPI та СКБД PostgreSQL;

— реалізовано процес завантаження, валідації та зберігання аудіозаписів і результатів їх обробки;

— виконано інтеграцію з API автоматичного розпізнавання мовлення та мовною моделлю для генерації протоколів;

— забезпечено контейнеризацію застосунку з використанням Docker для уніфікації середовища розгортання;

— наведено приклади можливих напрямів удосконалення розробленого рішення.

Подальший розвиток предмету дослідження може включати розширення функціональності системи шляхом інтеграції з календарними та корпоративними сервісами, а також удосконалення алгоритмів аналізу тексту для автоматичного виділення ключових рішень, завдань і відповідальних осіб.

У майбутньому розроблений застосунок автоматичного створення протоколів може бути впроваджений у різних сферах діяльності, де важливими є точність фіксації інформації та ефективність комунікації. Забезпечення масштабування системи, підвищення рівня безпеки даних і оптимізація обробки великих обсягів аудіоінформації дозволить використовувати рішення у корпоративних і промислових середовищах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Anthati Y., Mudde K., Damuluri H. Multilingual Video Summarizer. IEEE. 2025. URL: https://ieeexplore.ieee.org/document/11053084 (дата звернення: 15.12.2025).

2. Automated meeting summary generation using ASR and NLP pipelines : пат. US11593812B2. Опубл. 2023.

3. Baevski A., Zhou Y., Mohamed A., Auli M. wav2vec 2.0: A framework for self-supervised learning of speech representations. NeurIPS. 2020.

4. Banda S. K., Shareef M. R., Bavirthi S. S. Automatic Generation of Executive Summaries. Smart Trends in Computing. Springer, 2025. URL: https://link.springer.com/chapter/10.1007/978-981-96-7508-1_11 (дата звернення: 15.12.2025).

5. Banda S. K., Shareef M. R., Bavirthi S. S. Summaries for Online Meetings Using NLP and ASR. Smart Trends in Computing and Communications. 2025.

6. Carletta J. та ін. The AMI Meeting Corpus: A Multimodal Database. Machine Learning Journal. 2005.

7. Deshmukh S., Pacharaney U. Automated Speech-to-Text in Healthcare Communication. IEEE. 2025. URL: https://ieeexplore.ieee.org/document/10933272 (дата звернення: 15.12.2025).

8. FastAPI Documentation. URL: https://fastapi.tiangolo.com/ (дата звернення: 15.12.2025).

9. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2021. 560 с.

10. Jadhav A., Channe H. Extractive vs. Abstractive Summarization in NLP: Comparative Survey. Procedia Computer Science. 2021.

11. Janin A. та ін. The ICSI Meeting Corpus. ICASSP. 2003.

12. Jerusha B., Sunil A. R. K., Yogish D. Speech-to-Text Summarization Pipeline. IEEE. 2025. URL: https://ieeexplore.ieee.org/document/11036034 (дата звернення: 15.12.2025).

13. Khalil Z. та ін. Abstractive Summarization of Spoken Dialogues Using Hierarchical Transformers. 2023.

14. Kruchten P. The Rational Unified Process: An Introduction. Boston : Addison-Wesley, 2020. 360 с.

15. LangChain Documentation. URL: https://docs.langchain.com/ (дата звернення: 15.12.2025).

16. Lewis M., Liu Y., Goyal N. та ін. BART: Denoising Sequence-to-Sequence Pre-training. ACL. 2020.

17. Mirge A. S., Dharmale G. Automated MoM Generation with IoT-Based Audio Capture. IJACECT. 2025. URL: https://journals.mriindia.com/index.php/ijacect/article/download/345/385 (дата звернення: 15.12.2025).

18. OpenAI API Docs: Speech-to-text. URL: https://platform.openai.com/docs/guides/speech-to-text (дата звернення: 15.12.2025).

19. OpenAI Whisper. URL: https://github.com/openai/whisper (дата звернення: 15.12.2025).

20. OpenAI. Fine-tuning LLMs for Structured Document Generation. OpenAI Research. 2025.

21. OpenAI. gpt-4o-transcribe: Multimodal ASR Solution. OpenAI Docs. 2024.

22. Palanisamy B. та ін. Microservices Architecture for AI-based Applications. IEEE Software. 2023.

23. Panayotov V. та ін. LibriSpeech: An ASR corpus. ICASSP. 2015.

24. Perumal I., Kalaivani P. AI Chrome Extension for Meeting Summary. IEEE. 2025. URL: https://ieeexplore.ieee.org/document/10940198 (дата звернення: 15.12.2025).

25. Radford A. та ін. Whisper: Robust Speech Recognition via Large-Scale Weak Supervision. OpenAI, 2022. URL: https://github.com/openai/whisper (дата звернення: 15.12.2025).

26. See A., Liu P. J., Manning C. D. Get To The Point: Abstractive Summarization with Pointer-Generator Networks. ACL. 2017.

27. SQLModel Documentation. URL: https://sqlmodel.tiangolo.com/ (дата звернення: 15.12.2025).

28. System and method for generating structured meeting protocols from speech audio : пат. WO2024011276A1. Опубл. 2024.

29. Thanam A., Kamalanaban E., Devi M. M. Y. An NLP-Driven Approach for Automated Transcription and Summarization of YouTube Videos. IEEE. 2025. URL: https://ieeexplore.ieee.org/document/11176490 (дата звернення: 15.12.2025).

30. Van Vliet H. Software Engineering: Principles and Practice. Hoboken : Wiley, 2020. 550 с.

31. Varalatchoumy M., Hayath S., Dinesh D. Generative AI-Powered Video Summarization. Springer, 2025. URL: https://link.springer.com/chapter/10.1007/978-981-96-7502-9_10 (дата звернення: 15.12.2025).

32. Wang X. та ін. AutoMin Challenge: Benchmarking Minuting Systems. 2024.

33. Zhang B. та ін. Meta’s SpeechBrain: Open-Source Toolkit for End-to-End Speech Processing. Meta AI. 2023.

34. Zhang J., Zhao Y., Litman D. Meeting Summarization with Disentangled Role Modeling. EMNLP. 2022.

35. Zhang Y. та ін. Automatic Meeting Summarization: A Survey. ACM Computing Surveys. 2021.

36. Гринь В. С., Пєтухов В. В. Інтелектуальні системи обробки природної мови українською мовою. Системи обробки інформації. 2021. № 3(165). С. 21–25.

37. Колодяжний В. І. Автоматизація створення протоколів нарад за допомогою трансформації голосу в текст. Сучасні інформаційні системи. 2023.

№ 2. URL: https://openarchive.nure.ua/handle/document/20322 (дата звернення: 15.12.2025).

38. Кондратенко Ю. П., Литвиненко А. В. Моделі та методи розпізнавання українського усного мовлення на основі нейронних мереж. Вісник НТУ «ХПІ». Серія: Інформатика та моделювання. 2022. № 3. URL: http://repository.kpi.kharkov.ua/handle/KhPI-Press/58802 (дата звернення: 15.12.2025).

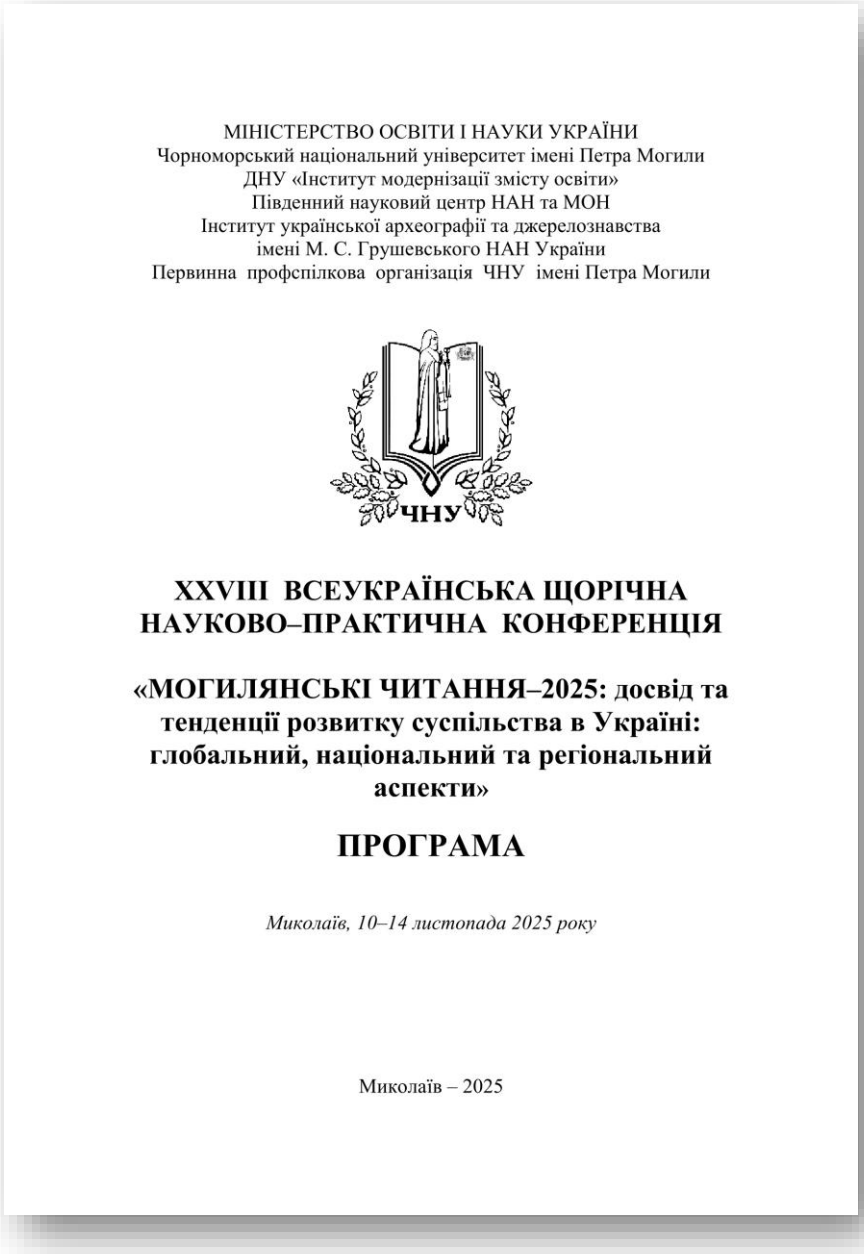
39. Мозговий М. І., Дворецький Ю. І. Система трансформації аудіо в текст з елементами NLP для української мови. Наукові праці ОНАЗ ім. О. С. Попова. 2021. URL: http://journals.uran.ua/index.php/1998-5748/article/view/260258 (дата звернення: 15.12.2025).

40. Фрич Д. О., Давиденко Є. О. Автоматизоване створення протоколів зустрічей на основі аудіозаписів. Могилянські читання – 2025 : тези доп. XXVIII Всеукр. наук.-практ. конф. Миколаїв, 10–14 листоп. 2024 р. Миколаїв : Чорном. нац. ун-т ім. Петра Могили, 2025.

ДОДАТОК А

Матеріали апробації роботи

Результати досліджень були представлені на конференції -Могилянські читання 2025, Досвід та тенденції розвитку суспільства в Україні: глобальний, національний та регіональний аспекти: XXVIII Всеукр. наук.-практ. конф. тези доповідей: Комп'ютерні науки. Технічні науки, Миколаїв, 10-14 листоп. 2025 р. / ЧНУ ім. Петра Могили. Миколаїв: Вид-во ЧНУ ім. Петра Могили, 2025.



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
ДНУ «Інститут модернізації змісту освіти»
Південний науковий центр НАН та МОН
Інститут української археографії та джерелознавства
імені М. С. Грушевського НАН України
Первинна профспілкова організація ЧНУ імені Петра Могили

ЧНУ

XXVIII ВСЕУКРАЇНСЬКА ЩОРІЧНА НАУКОВО–ПРАКТИЧНА КОНФЕРЕНЦІЯ

«МОГИЛЯНСЬКІ ЧИТАННЯ–2025: досвід та тенденції розвитку суспільства в Україні: глобальний, національний та регіональний аспекти»

ПРОГРАМА

Миколаїв, 10–14 листопада 2025 року

Миколаїв – 2025

Рисунок А.1 – Обкладинка збірника тез доповідей конференції