

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій  
(повне найменування факультету)

Кафедра програмних засобів  
(повне найменування кафедри)

## Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему РОЗРОБКА АВТОМАТИЗОВАНОЇ СИСТЕМИ РОЗПІЗНАВАННЯ  
АУДІО СИГНАЛІВ  
DEVELOPMENT OF AN AUTOMATIC AUDIO SIGNAL  
RECOGNITION SYSTEM

Виконав: студент 2 курсу, групи КНТ-123м  
Спеціальності 121 Інженерія програмного  
забезпечення  
(код і найменування спеціальності)

Освітня програма (спеціалізація)  
Інженерія програмного забезпечення

МОРОЗОВ Д.М.  
(ПРИЗВИЩЕ та ініціали)

Керівник ЛЕОЩЕНКО С.Д.  
(ПРИЗВИЩЕ та ініціали)

Рецензент СКРУПСЬКИЙ С.Ю.  
(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
Національний університет «Запорізька політехніка»  
(повне найменування закладу вищої освіти)

Факультет КНТ  
Кафедра програмних засобів  
Ступінь вищої освіти магістр  
Спеціальність 121 Інженерія програмного забезпечення  
(код і найменування)  
Освітня програма (спеціалізація) Інженерія програмного забезпечення  
(назва освітньої програми (спеціалізації))

**ЗАТВЕРДЖУЮ**

Завідувач кафедри ПЗ, д.т.н, проф.  
Сергій СУББОТІН  
“      ”        2024 року

**З А В Д А Н Н Я**

**НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)**

МОРОЗОВА Дениса Миколайовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка автоматизованої системи розпізнавання аудіо сигналів. Development of an Automatic Audio Signal Recognition System

керівник проєкту (роботи) д-р.філос., ЛЕОЩЕНКО Сергій Дмитрович,  
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “07” жовтня 2024 року № 398

2. Строк подання студентом проєкту (роботи) 5 грудня 2024 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області, 2. Матеріали та методи, 3. Основні рішення щодо реалізації компонентів. 4. Тестування реалізації та результати.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації



## РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:  
114 с., 26 рис., 11 табл., 4 дод., 23 джерела.

РОЗПІЗНАВАННЯ АУДІО, ВЕБ-ЗАСТОСУНОК, DJANGO, REACT.JS,  
МАШИННЕ НАВЧАННЯ, API, КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС,  
АВТОМАТИЗАЦІЯ.

Об'єкт дослідження – процес розробки автоматизованої системи розпізнавання аудіо сигналів із використанням сучасних веб-технологій.

Предмет дослідження – методи та засоби реалізації веб-систем розпізнавання аудіо з використанням сучасних веб-технологій.

Мета роботи – розробка системи розпізнавання аудіо сигналів з використанням сучасних веб-технологій для забезпечення високої точності розпізнавання та зручного користувацького інтерфейсу.

Матеріали, методи та технічні засоби: методи моделювання, проектування програмного забезпечення, програмування; фреймворк Django, бібліотека React.js, база даних SQLite; системи розпізнавання Google Web Speech API та PocketSphinx.

Результати. Розроблено веб-систему розпізнавання аудіо сигналів з можливістю вибору методу розпізнавання, що забезпечує зручний інтерфейс завантаження та обробки аудіофайлів, а також отримання і збереження результатів розпізнавання.

Висновки. Створено автоматизовану систему розпізнавання аудіо з модульною архітектурою, що поєднує переваги хмарних та локальних рішень, забезпечуючи високу точність розпізнавання та зручність використання через веб-інтерфейс.

Галузь використання - організації та установи, що потребують автоматизованого розпізнавання аудіо: освітні заклади, медичні установи, системи безпеки, а також індивідуальні користувачі.

## ABSTRACT

Explanatory note to the diploma qualifying work of the master: 114 pages, 26 figures, 11 tables, 4 appendixes, 23 sources.

AUDIO RECOGNITION, WEB APPLICATION, DJANGO, REACT.JS, MACHINE LEARNING, API, USER INTERFACE, AUTOMATION.

Research object is the process of developing an automated audio signal recognition system using modern web technologies.

Research subject is methods and tools for implementing web-based audio recognition systems using modern web technologies.

Purpose is the development of an audio signal recognition system using modern web technologies to ensure high recognition accuracy and user-friendly interface.

Materials, methods and technical tools: modeling methods, software design, programming; Django framework, React.js library, SQLite database; recognition systems Google Web Speech API and PocketSphinx.

Results. Developed a web-based audio signal recognition system with the ability to choose the recognition method, providing a convenient interface for uploading and processing audio files, as well as obtaining and storing recognition results.

Conclusions. Created an automated audio recognition system with modular architecture that combines the advantages of cloud and local solutions, providing high recognition accuracy and ease of use through a web interface.

Field of application - organizations and institutions requiring automated audio recognition: educational institutions, medical facilities, security systems, as well as individual users.

## ЗМІСТ

|                                                                                                    |    |
|----------------------------------------------------------------------------------------------------|----|
|                                                                                                    | С. |
| Вступ.....                                                                                         | 9  |
| 1 Аналіз предметної області .....                                                                  | 11 |
| 1.1 Сучасні технології розпізнавання аудіо сигналів: поточний стан та<br>перспективи розвитку..... | 11 |
| 1.2 Ринок існуючих рішень для розпізнавання аудіо сигналів .....                                   | 17 |
| 1.3 Огляд методів та алгоритмів розпізнавання аудіо .....                                          | 22 |
| 1.4 Архітектурний підхід до побудови системи .....                                                 | 27 |
| 1.5 Висновки за розділом 1 .....                                                                   | 32 |
| 2 Матеріали та методи.....                                                                         | 34 |
| 2.1 Терміни та визначення .....                                                                    | 35 |
| 2.2 Функціональні вимоги .....                                                                     | 36 |
| 2.3 Нефункціональні вимоги .....                                                                   | 37 |
| 2.4 Характеристика досліджуваної сфери (предметної області) .....                                  | 38 |
| 2.5 Опис системи .....                                                                             | 38 |
| 2.6 Огляд інструментів розробки .....                                                              | 39 |
| 2.6.1 Огляд Django .....                                                                           | 40 |
| 2.6.2 Огляд React.js .....                                                                         | 43 |
| 2.6.3 Огляд SQLite .....                                                                           | 46 |
| 2.6.4 Розпізнавання та обробка аудіо сигналів .....                                                | 47 |
| 2.7 Висновки за розділом 2 .....                                                                   | 50 |
| 3 Основні рішення щодо реалізації компонентів системи .....                                        | 52 |
| 3.1 Моделювання архітектури системи засобами UML .....                                             | 52 |
| 3.2 Діаграма варіантів використання .....                                                          | 54 |
| 3.3 Діаграма діяльності .....                                                                      | 58 |
| 3.4 Діаграма послідовності .....                                                                   | 62 |
| 3.5 Діаграма класів .....                                                                          | 65 |
| 3.6 Діаграма розгортання .....                                                                     | 67 |
| 3.7 Висновки за розділом 3 .....                                                                   | 69 |

|                                                       |     |
|-------------------------------------------------------|-----|
|                                                       | 7   |
| 4 Тестування реалізації та результати .....           | 71  |
| 4.1 Вибір та обґрунтування засобів реалізації .....   | 71  |
| 4.2 Розробка клієнтської частини.....                 | 73  |
| 4.3 Розробка серверної частини .....                  | 74  |
| 4.4 Основні класи та функції системи .....            | 76  |
| 4.5 Результати навчання та оцінка якості моделі ..... | 77  |
| 4.6 Тестування проєкту .....                          | 84  |
| 4.7 Керівництво користувача.....                      | 86  |
| 4.7.1 Підготовка до роботи .....                      | 86  |
| 4.7.2 Загальний опис інтерфейсу.....                  | 87  |
| 4.7.3 Функціональні елементи інтерфейсу.....          | 88  |
| 4.8 Висновки за розділом 4.....                       | 90  |
| Висновки.....                                         | 91  |
| Перелік джерел посилання .....                        | 93  |
| Додаток А Технічне завдання.....                      | 96  |
| Додаток Б Програмний код.....                         | 101 |
| Додаток В Слайди презентації .....                    | 106 |

## ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

ACID – Atomicity, Consistency, Isolation, Durability (атомарність, узгодженість, ізольованість, довговічність);

API – Application Programming Interface (інтерфейс програмування застосунків);

CNN – Convolutional Neural Networks (згорткові нейронні мережі);

CSRF – Cross-Site Request Forgery (міжсайтове підроблення запитів);

DOM – Document Object Model (об'єктна модель документа);

DTW – Dynamic Time Warping (динамічна трансформація часової шкали);

HMM – Hidden Markov Models (приховані марковські моделі);

JWT – JSON Web Token (веб-токен у форматі JSON);

MFCC – Mel-Frequency Cepstral Coefficients (мел-частотні кепстральні коефіцієнти);

MTV – Model-Template-View (модель-шаблон-представлення);

MVC – Model-View-Controller (модель-представлення-контролер);

ORM – Object-Relational Mapping (об'єктно-реляційне відображення);

RNN – Recurrent Neural Networks (рекурентні нейронні мережі);

SPA – Single Page Application (односторінковий додаток);

UML – Unified Modeling Language (уніфікована мова моделювання).

XSS – Cross-Site Scripting (міжсайтове виконання сценаріїв);

## ВСТУП

У сучасному світі, де інформаційні технології активно впливають на всі аспекти життя, розпізнавання аудіо сигналів займає особливе місце серед ключових технологій. Це обумовлено не лише необхідністю автоматизації процесів, пов'язаних з обробкою звуку, але й широтою застосувань цієї технології. Від медичної діагностики до систем безпеки, від інтерактивних голосових асистентів до інструментів для створення субтитрів – системи розпізнавання аудіо сигналів значно розширюють можливості цифрових рішень.

В основі сучасних систем розпізнавання лежать досягнення машинного навчання, зокрема використання глибоких нейронних мереж, таких як згорткові нейронні мережі (CNN), рекурентні мережі (RNN), а також архітектури на базі трансформерів. Завдяки їм, рівень точності розпізнавання досягає 95–98% у сприятливих умовах, що наближається до рівня людського сприйняття.

Водночас залишається низка викликів, зокрема адаптація систем до складних акустичних середовищ, підвищення продуктивності при обмежених обчислювальних ресурсах, а також забезпечення зручності використання для кінцевих споживачів.

Актуальність даної теми зумовлена необхідністю створення ефективних та гнучких інструментів для обробки аудіо сигналів. Така розробка має важливе значення для впровадження інноваційних рішень у сфері автоматизації процесів, підвищення якості взаємодії між користувачами та інформаційними системами, а також розширення можливостей аналізу великих обсягів аудіоданих.

Метою роботи є розробка системи розпізнавання аудіо сигналів з використанням сучасних веб-технологій для забезпечення високої точності розпізнавання та зручного користувацького інтерфейсу.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

- провести аналіз існуючих рішень та методів розпізнавання аудіо сигналів, визначити їх переваги та обмеження;

- сформулювати вимоги до системи розпізнавання аудіо сигналів та обрати оптимальні технології для її реалізації;
- спроектувати архітектуру системи, визначити основні компоненти та взаємозв'язки між ними;
- розробити серверну частину системи з використанням Django, реалізувати необхідні API та функції обробки аудіо;
- створити клієнтську частину на базі React.js з інтуїтивно зрозумілим інтерфейсом користувача;
- провести тестування розробленої системи.

Практична цінність роботи полягає у створенні готового до використання веб-застосунку для розпізнавання аудіо сигналів, що може бути впроваджений як самостійний продукт або інтегрований в існуючі інформаційні системи.

## 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

### 1.1 Сучасні технології розпізнавання аудіо сигналів: поточний стан та перспективи розвитку

Розпізнавання аудіо сигналів є ключовою складовою сучасних інформаційних технологій, що знаходить застосування в різних галузях, таких як безпека, медицина, розваги та інші. Сучасні методи розпізнавання аудіо сигналів базуються на поєднанні класичних алгоритмів обробки сигналів та новітніх підходів машинного навчання, зокрема глибоких нейронних мереж [1].

Аналіз історичного розвитку технологій розпізнавання аудіо сигналів, представлений у таблиці 1.1, демонструє значний прогрес від простих систем розпізнавання окремих слів до складних мультимодальних систем з точністю розпізнавання до 98%. Історія розвитку систем розпізнавання аудіо сигналів почалася ще в 1950-х роках з простих систем розпізнавання окремих слів. Перші системи базувалися на порівнянні шаблонів та мали дуже обмежені можливості. Прорив відбувся в 1970-х роках з появою прихованих марковських моделей, які стали стандартом для розпізнавання мовлення на наступні десятиліття [2].

Сучасний етап розвитку характеризується революційними змінами, пов'язаними з впровадженням технологій глибокого навчання. Починаючи з 2010-х років, використання глибоких нейронних мереж дозволило досягти безпрецедентної точності розпізнавання. Проривом стало застосування згорткових та рекурентних нейронних мереж, які здатні ефективно обробляти часові послідовності та виявляти складні патерни в аудіо сигналах [3].

Останні роки відзначилися появою трансформерних архітектур, які забезпечили значний прогрес у якості розпізнавання. Важливим досягненням стала можливість end-to-end навчання систем, що дозволяє оптимізувати весь процес розпізнавання як єдине ціле. Розвиток технологій самонавчання та навчання з малою кількістю розмічених даних відкрив нові можливості для створення більш універсальних систем [4].

Таблиця 1.1 - Етапи розвитку технологій розпізнавання аудіо сигналів

| Період    | Технології та методи                                           | Точність | Обмеження                                                                                  | Ключові досягнення                                                      | Основні застосування                                     |
|-----------|----------------------------------------------------------------|----------|--------------------------------------------------------------------------------------------|-------------------------------------------------------------------------|----------------------------------------------------------|
| 1950-1960 | Аналогові фільтри.<br>Частотний аналіз.                        | 10-20%   | - розпізнавання окремих слів;<br>- залежність від диктора;<br>- висока чутливість до шуму. | - перші системи розпізнавання цифр;<br>- базовий частотний аналіз.      | - військові застосування;<br>- експериментальні системи. |
| 1970-1980 | Приховані марковські моделі (НММ).<br>Динамічне програмування. | 40-50%   | - обмежений словник;<br>- значна кількості навчальних даних.                               | - алгоритм Вітербі;<br>- перші комерційні системи.                      | - голосове управління;<br>- диктування тексту.           |
| 1990-2000 | Статистичні методи.<br>Нейронні мережі першого покоління.      | 60-70%   | - висока обчислювальна складність;<br>- залежність від якості запису.                      | - гібридні системи НММ-NN;<br>- покращені акустичні моделі.             | - телефонні системи;<br>- автоматизовані сервіси.        |
| 2000-2010 | Глибокі нейронні мережі (DNN).<br>Згорткові мережі (CNN).      | 80-85%   | - потреба у великих обсягах даних;<br>- високі вимоги до обчислень.                        | - End-to-end навчання;<br>- автоматичне виділення ознак.                | - мобільні асистенти;<br>- системи транскрибації.        |
| 2010-2020 | Рекурентні мережі (RNN).<br>LSTM.<br>Attention механізми.      | 90-95%   | - складність навчання;<br>- потреба в GPU.                                                 | - Bidirectional LSTM;<br>- Attention механізми;<br>- Transfer learning. | - голосові помічники;<br>- реально-часові системи.       |

Кінець таблиці 1.1

| 1          | 2                                                   | 3      | 4                                                             | 5                                                                              | 6                                                                   |
|------------|-----------------------------------------------------|--------|---------------------------------------------------------------|--------------------------------------------------------------------------------|---------------------------------------------------------------------|
| 2020-тепер | Трансформери. Самонавчання. Мульти-модальні моделі. | 95-98% | - великі обчислювальні вимоги;<br>- складність інтерпретації. | - Few-shot learning;<br>- крос-лінгвістичні моделі;<br>- контекстне розуміння. | - розумні пристрої;<br>- медична діагностика;<br>- системи безпеки. |

Поточний стан галузі характеризується досягненням точності розпізнавання 95-98% у сприятливих акустичних умовах, що наближається до людського рівня сприйняття. Сьогодні системи здатні працювати в режимі реального часу, адаптуватися до різних акустичних умов та розпізнавати мовлення різних дикторів без попереднього налаштування [5].

Сучасний етап розвитку технологій характеризується активним впровадженням у різних галузях, що підтверджується даними таблиці 1.2. Особливо варто відзначити досягнення в медичній галузі, де системи розпізнавання аудіо демонструють точність діагностики на рівні 92-95%, що дозволило знизити час обстеження пацієнтів на 60%. У сфері безпеки досягнуто значних результатів у швидкості реакції систем (менше 1 секунди) та точності ідентифікації голосу (98%) [6].

Сфери застосування систем розпізнавання аудіо постійно розширюються. У медицині вони використовуються для ранньої діагностики неврологічних захворювань, моніторингу стану пацієнтів через аналіз голосу, автоматизації ведення медичної документації та допомоги пацієнтам з вадами слуху. В області безпеки системи розпізнавання застосовуються для виявлення потенційних загроз через акустичний моніторинг, ідентифікації осіб за голосом, розпізнавання небезпечних звуків та аналізу аудіозаписів у криміналістиці. Значного поширення набули голосові помічники та системи керування, які стали невід'ємною частиною сучасних смартфонів та розумних пристроїв [6].

Таблиця 1.2 - Застосування технологій в різних галузях

| Галузь      | Приклади застосування                                                                                   | Особливості впровадження                                                     | Досягнуті результати                                                                                 |
|-------------|---------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| Медицина    | - діагностика респіраторних захворювань;<br>- моніторинг серцевих ритмів;<br>- аналіз психічного стану. | - висока точність;<br>- спеціалізовані датасети;<br>- реальночасова обробка. | - точність діагностики 92-95%;<br>- зниження часу обстеження на 60%;<br>- раннє виявлення патологій. |
| Безпека     | - виявлення пострілів;<br>- ідентифікація голосу;<br>- моніторинг натовпу.                              | - стійкість до шумів;<br>- швидка реакція;<br>- цілодобова робота.           | - час реакції < 1 сек;<br>- точність ідентифікації 98%;<br>- зниження помилкових тривог на 75%.      |
| Освіта      | - транскрибація лекцій;<br>- вивчення мов;<br>- оцінка вимови.                                          | - багатомовність;<br>- адаптація до акцентів;<br>- інтерактивність.          | - точність транскрибації 95%;<br>- покращення засвоєння на 40%;<br>- персоналізоване навчання.       |
| Виробництво | - контроль якості;<br>- моніторинг обладнання;<br>- голосове управління.                                | - промислові шуми;<br>- інтеграція з IoT;<br>- безперервна робота.           | - виявлення несправностей 90%;<br>- зниження простоїв на 35%;<br>- оптимізація процесів.             |

Втім, незважаючи на переваги, сучасні рішення стикаються з низкою недоліків. Однією з основних проблем є залежність від якості даних. Системи дуже чутливі до шуму, перешкод та низької якості запису, що може суттєво знижувати точність результатів (таблиця 1.3). Варто зазначити, що розробка таких систем є ресурсозатратною через високу обчислювальну складність. Навчання глибоких моделей потребує значних обчислювальних потужностей, що може стати серйозною перешкодою для розробників із обмеженими ресурсами.

Таблиця 1.3 - Вплив якості даних на точність

| Фактор якості       | Вплив на точність | Методи покращення                                                              | Типові проблеми                                      |
|---------------------|-------------------|--------------------------------------------------------------------------------|------------------------------------------------------|
| Шум                 | -10-30%           | - фільтрація;<br>- адаптивне шумопридушення;<br>- багатоканальна обробка.      | - втрата корисного сигналу;<br>- спотворення частот. |
| Акустика приміщення | -5-15%            | - деревербрація;<br>- просторова фільтрація;<br>- адаптивні моделі.            | - відлуння;<br>- інтерференція.                      |
| Якість запису       | -15-25%           | - попередня обробка;<br>- нормалізація;<br>- підвищення частоти дискретизації. | - кліппінг;<br>- артефакти стиснення.                |
| Розмір датасету     | -20-40%           | - аугментація даних;<br>- синтетичні дані;<br>- трансферне навчання.           | - перенавчання;<br>- недостатня варіативність.       |

Ще одним викликом є обмежена інтерпретованість моделей, зокрема нейронних мереж, які часто працюють як «чорні скриньки». Це ускладнює пояснення отриманих результатів і діагностику можливих помилок. Для мовних задач складнощі можуть виникати через специфіку мов або діалектів, які недостатньо представлені у навчальних даних. Окрім того, використання сторонніх хмарних сервісів, таких як Google Cloud чи IBM Watson, може бути фінансово обтяжливим для малих компаній або індивідуальних розробників.

Актуальним є питання забезпечення конфіденційності при обробці аудіо даних, особливо в контексті голосових помічників та систем моніторингу. Крім того, існує потреба у вдосконаленні роботи систем в умовах сильного шуму та реверберації.

Потенціал впровадження автоматизованої системи розпізнавання аудіо сигналів є значним, з огляду на різноманіття потенційних сфер використання у багатьох галузях. Технології, що базуються на розпізнаванні аудіо сигналів, пропонують ефективні рішення для автоматизації процесів, підвищення точності аналізу даних та забезпечення інтерактивності між користувачами та системами.

У бізнесі такі системи можуть покращити якість обслуговування клієнтів, наприклад, через впровадження голосових помічників або чат-ботів, що здатні розуміти мовлення та реагувати на запити в реальному часі. У службах підтримки клієнтів автоматизовані системи здатні записувати та аналізувати дзвінки, виявляти типові проблеми та навіть оцінювати емоційний стан клієнтів для більш ефективної взаємодії [6].

У сфері освіти такі системи дозволяють трансформувати процес навчання. Вони здатні автоматично транскрибувати лекції в текст, генерувати субтитри до відео та навіть пропонувати негайний зворотний зв'язок під час онлайн-занять. Завдяки інтеграції з інтерфейсами користувача, ці системи можуть підтримувати багатомовність, що розширює їхню доступність для аудиторій з різних країн.

У сфері медицини аналіз аудіосигналів відкриває можливості для спрощення процесу діагностики та контролю за станом здоров'я пацієнтів. Наприклад, системи можуть аналізувати звуки дихання або серцевого ритму для раннього виявлення захворювань. Голосові асистенти також можуть допомагати лікарям вести медичну документацію, забезпечуючи точність і економію часу.

Індустрія розваг та медіа також є перспективною сферою впровадження таких технологій. Розпізнавання аудіо сигналів дозволяє створювати інтерактивні ігри, забезпечувати персоналізовані музичні рекомендації, автоматично створювати субтитри та перекладати мовлення у реальному часі. Голосове управління в ігрових платформах або мультимедійних пристроях є однією з тенденцій, що швидко набирає популярності.

Потенціал розпізнавання аудіо сигналів також значний у державному секторі та системах безпеки. Автоматизовані системи можуть аналізувати звуки в громадських місцях для виявлення потенційних загроз, таких як постріли чи звуки вибухів, що дозволяє оперативно реагувати на надзвичайні ситуації. У судовій системі ці технології можуть використовуватися для транскрибування свідчень та аналізу аудіозаписів.

Аналіз перспективних напрямків розвитку технологій вказує на зростаючу роль мультимодальних систем та федеративного навчання. Особливий інтерес представляють нейроморфні обчислення, які обіцяють значне підвищення енергоефективності систем розпізнавання. Розвиток квантових обчислень може відкрити нові можливості для оптимізації алгоритмів навчання. Ключовими факторами, що вплинули на розвиток технологій, стали: стрімке зростання обчислювальної потужності (збільшення у 100 разів за останні 10 років), розширення доступності даних через відкриті датасети та краудсорсинг, а також алгоритмічні інновації, зокрема впровадження механізмів уваги та методів самонавчання [7].

Таким чином, сучасні досягнення в області розпізнавання аудіо сигналів демонструють стабільний прогрес у підвищенні точності та розширенні сфер застосування. Завдяки постійному вдосконаленню алгоритмів та технологій їхнє впровадження стає дедалі доступнішим, відкриваючи нові можливості для інновацій [7].

## **1.2 Ринок існуючих рішень для розпізнавання аудіо сигналів**

Для реалізації систем розпізнавання аудіо сигналів існує низка бібліотек та фреймворків. Наприклад, бібліотека Librosa є однією з найпопулярніших Python-бібліотек для обробки аудіо. Вона забезпечує широкий набір функцій для аналізу, обробки, синтезу аудіосигналів, а також роботи зі спектральними характеристиками звуку. Librosa часто використовується в задачах машинного навчання, які потребують роботи з аудіоданими, таких як класифікація жанрів, розпізнавання мовлення чи музики, а також створення музичних додатків. Фреймворки машинного навчання, такі як TensorFlow та PyTorch, забезпечують засоби для побудови та навчання нейронних мереж, що використовуються для розпізнавання аудіо [8].

Аналіз сучасних бібліотек для обробки аудіо сигналів (таблиця 1.4) показує різноманітність доступних інструментів, кожен з яких має свої особливості та сфери застосування.

Librosa залишається найбільш популярним вибором для задач машинного навчання та аналізу аудіо, пропонуючи широкий спектр функцій та відмінну документацію. Для високопродуктивної обробки даних перевагу має SoundFile, що забезпечує ефективну роботу з низькорівневими аудіо даними. TorchAudio виділяється можливістю використання GPU-прискорення, що особливо важливо для проектів глибокого навчання [9].

Вибір конкретної бібліотеки залежить від специфіки проекту, вимог до продуктивності та необхідного функціоналу. Для простих задач обробки та конвертації форматів оптимальним вибором є PyDub, тоді як для наукових досліджень та складного аналізу краще підходить комбінація Librosa та SciPy Audio. При розробці систем реального часу перевагу слід віддавати TorchAudio або SoundFile через їх високу продуктивність. Комбінація класичних методів обробки сигналів з сучасними підходами машинного навчання дозволяє створювати ефективні системи розпізнавання аудіо сигналів, здатні адаптуватися до різних умов та забезпечувати високу якість роботи [9].

Ринок застосунків для розпізнавання аудіо сигналів стрімко розвивається, охоплюючи різноманітні галузі та задовольняючи потреби як індивідуальних користувачів, так і корпоративних клієнтів. Сучасні технології дозволяють створювати застосунки, здатні аналізувати та інтерпретувати аудіодані в режимі реального часу, що відкриває широкі можливості для їх використання. Аналіз існуючих рішень дозволяє визначити їх переваги, обмеження та оптимальні сфери застосування. На ринку систем розпізнавання аудіо провідні позиції займають кілька основних гравців.

OpenAI Whisper представляє сучасне рішення з відкритим кодом, що забезпечує високу точність розпізнавання для багатьох мов та може працювати як локально, так і через API. Система демонструє стійкість до фонового шуму та акцентів, що робить її особливо ефективною для реальних умов використання.

Таблиця 1.4 - Порівняльна характеристика бібліотек обробки аудіо

| Бібліотека  | Переваги                                                                                                                                                          | Недоліки                                                                                                                                            | Особливості використання                                                                                                                                     |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Librosa     | <ul style="list-style-type: none"> <li>- широкий набір функцій аналізу;</li> <li>- відмінна документація;</li> <li>- потужні інструменти візуалізації.</li> </ul> | <ul style="list-style-type: none"> <li>- повільніша обробка;</li> <li>- висока затрата пам'яті;</li> <li>- обмежена підтримка запису.</li> </ul>    | <ul style="list-style-type: none"> <li>- найкраще підходить для аналізу;</li> <li>- популярна в ML проектах;</li> <li>- багато готових прикладів.</li> </ul> |
| PyDub       | <ul style="list-style-type: none"> <li>- простий інтерфейс;</li> <li>- ефективна обробка;</li> <li>- підтримка багатьох форматів.</li> </ul>                      | <ul style="list-style-type: none"> <li>- обмежені функції аналізу;</li> <li>- базова функціональність;</li> <li>- залежність від ffmpeg.</li> </ul> | <ul style="list-style-type: none"> <li>- ідеальна для базової обробки;</li> <li>- конвертація форматів;</li> <li>- проста маніпуляція аудіо.</li> </ul>      |
| SoundFile   | <ul style="list-style-type: none"> <li>- висока продуктивність;</li> <li>- низькорівневий доступ;</li> <li>- мала затрата пам'яті.</li> </ul>                     | <ul style="list-style-type: none"> <li>- обмежена функціональність;</li> <li>- мінімальна обробка;</li> <li>- складніший API.</li> </ul>            | <ul style="list-style-type: none"> <li>- ефективне читання/запис;</li> <li>- потокова обробка;</li> <li>- робота з raw-даними.</li> </ul>                    |
| Audioread   | <ul style="list-style-type: none"> <li>- універсальний доступ;</li> <li>- стабільна робота;</li> <li>- простий API.</li> </ul>                                    | <ul style="list-style-type: none"> <li>- тільки читання;</li> <li>- немає функцій обробки;</li> <li>- обмежена підтримка.</li> </ul>                | <ul style="list-style-type: none"> <li>- надійне читання файлів;</li> <li>- кросплатформність;</li> <li>- базове декодування.</li> </ul>                     |
| SciPy Audio | <ul style="list-style-type: none"> <li>- наукові обчислення</li> <li>- інтеграція з numpy;</li> <li>- швидка обробка.</li> </ul>                                  | <ul style="list-style-type: none"> <li>- базові функції;</li> <li>- обмежені формати;</li> <li>- мінімальний функціонал.</li> </ul>                 | <ul style="list-style-type: none"> <li>- низькорівнева обробка;</li> <li>- наукові дослідження;</li> <li>- швидкі перетворення.</li> </ul>                   |
| TorchAudio  | <ul style="list-style-type: none"> <li>- GPU прискорення;</li> <li>- інтеграція з pytorch;</li> <li>- оптимізована обробка.</li> </ul>                            | <ul style="list-style-type: none"> <li>- складний API;</li> <li>- вимагає pytorch;</li> <li>- висока крива навчання.</li> </ul>                     | <ul style="list-style-type: none"> <li>- ML проекти;</li> <li>- глибоке навчання;</li> <li>- реальночасова обробка.</li> </ul>                               |

Google Cloud Speech-to-Text V2 представляє передове рішення для розпізнавання мовлення. Система підтримує понад 120 мов та діалектів, забезпечуючи точність розпізнавання до 98%. Сервіс підтримує автоматичне визначення мови, адаптивну фільтрацію шумів та оптимізацію для специфічних предметних областей.

Amazon Transcribe від Amazon Web Services (AWS) є ще одним потужним інструментом для автоматичного перетворення мовлення в текст. Сервіс підтримує обробку потокового аудіо, що робить його зручним для застосування в реальному часі, наприклад, у службах підтримки клієнтів або під час проведення вебінарів. Сервіс пропонує спеціалізовані моделі для різних галузей та можливість створення власних словників.

Для локального розгортання актуальним є рішення PocketSphinx - легка версія CMUSphinx, оптимізована для мобільних та вбудованих систем. Хоча точність дещо нижча порівняно з хмарними сервісами, система забезпечує повний контроль над даними та не потребує підключення до інтернету.

Серед нових рішень варто відзначити Coqui TTS/STT - проект, що розвинувся з Mozilla DeepSpeech та пропонує сучасні моделі для розпізнавання та синтезу мовлення. Система підтримує багатомовність та може бути розгорнута локально.

Порівняльний аналіз систем за ключовими характеристиками, наведений в таблиці 1.5, показує тенденцію до балансування між хмарними та локальними рішеннями, де кожен підхід має свої переваги залежно від конкретних вимог до системи розпізнавання аудіо. Хмарні сервіси забезпечують найвищу точність та постійні оновлення моделей, тоді як локальні рішення гарантують конфіденційність та незалежність від підключення до мережі.

При виборі системи слід враховувати фактори, наведені на рисунку 1.1. Загалом, ринок застосунків для розпізнавання аудіо сигналів демонструє стійке зростання, обумовлене розвитком технологій машинного навчання та збільшенням обчислювальних потужностей.

Таблиця 1.5 - Порівняльна таблиця систем розпізнавання аудіо сигналів

| Система                        | Тип розгортання | Точність | Мови | Розпізнавання в реальному часі | Переваги                                                |
|--------------------------------|-----------------|----------|------|--------------------------------|---------------------------------------------------------|
| OpenAI Whisper                 | Локальне + API  | 85-95%   | 90+  | Ні                             | Стійкість до шуму, відкритий код, GPU підтримка         |
| Google Cloud Speech-to-Text V2 | Хмарне          | 95-98%   | 120+ | Так                            | Фільтрація шумів, визначення мови, галузева оптимізація |
| Amazon Transcribe              | Хмарне          | 90-95%   | 100+ | Так                            | Власні словники, галузеві моделі, аналітика             |
| Pocket Sphinx                  | Локальне        | 75-85%   | 20+  | Так                            | Низькі вимоги, повний контроль, офлайн режим            |
| Coqui TTS/STT                  | Локальне + API  | 80-90%   | 30+  | Так                            | Відкритий код, модульність, легка інтеграція            |

| Точність розпізнавання                                                                                                                                             | Технічні обмеження                                                                                                                                | Особливості експлуатації                                                                                                                                         | Економічні аспекти                                                                                                                                           |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"><li>• специфіка предметної області</li><li>• якість вхідних аудіо даних</li><li>• необхідність роботи з різними мовами</li></ul> | <ul style="list-style-type: none"><li>• доступні обчислювальні ресурси</li><li>• вимоги до швидкодії</li><li>• можливості масштабування</li></ul> | <ul style="list-style-type: none"><li>• необхідність роботи в реальному часі</li><li>• вимоги до конфіденційності даних</li><li>• обсяги обробки аудіо</li></ul> | <ul style="list-style-type: none"><li>• вартість впровадження та підтримки</li><li>• модель ліцензування</li><li>• потенційна окупність інвестицій</li></ul> |

Рисунок 1.1 – Фактори, що впливають на вибір системи розпізнавання аудіо сигналів

### 1.3 Огляд методів та алгоритмів розпізнавання аудіо

Розпізнавання аудіо сигналів представляє собою складний багатоетапний процес обробки інформації, що вимагає застосування комплексу різноманітних методів та алгоритмів. На першому етапі обробки здійснюється попередня підготовка сигналу, що включає фільтрацію шумів, нормалізацію амплітуди та сегментацію на часові вікна. Фільтрація шумів здійснюється за допомогою низькочастотних та високочастотних фільтрів, які дозволяють видалити небажані частотні компоненти сигналу. При цьому особлива увага приділяється збереженню корисної інформації та мінімізації спотворень сигналу. Нормалізація амплітуди забезпечує вирівнювання гучності та компресію динамічного діапазону, що покращує подальшу обробку сигналу.

Сегментація на часові вікна є критично важливим етапом, під час якого сигнал розбивається на короткі фрагменти з використанням спеціальних віконних функцій, таких як вікна Хеммінга або Ханна. Ці функції забезпечують плавні переходи між сегментами та мінімізують крайові ефекти. Важливим аспектом є вибір розміру вікна та ступеня перекриття між сусідніми вікнами, що впливає на якість подальшого частотного аналізу [10].

Наступним ключовим етапом є виділення акустичних ознак, серед яких найбільшого поширення набули мел-частотні кепстральні коефіцієнти (MFCC). Ці коефіцієнти ефективно моделюють особливості сприйняття звуку людським вухом, враховуючи нелінійність слухової системи людини. Процес обчислення MFCC включає застосування перетворення Фур'є, перехід до мел-шкали частот та косинусне перетворення. Окрім MFCC, важливими є спектральні ознаки, такі як спектральний центроїд, що характеризує "яскравість" звуку, спектральний нахил та спектральний потік, які відображають зміни в спектральному складі сигналу.

Класичні методи розпізнавання базуються на використанні прихованих марковських моделей (НММ) та динамічній трансформації часової шкали (DTW). НММ дозволяють моделювати послідовність звуків як ланцюжок

прихованих станів, що генерують спостережувані акустичні ознаки. Важливою перевагою НММ є можливість враховувати часову динаміку сигналу та варіативність вимови. Для навчання НММ використовується алгоритм Баума-Велша, а для розпізнавання - алгоритм Вітербі, який знаходить найбільш ймовірну послідовність станів [11].

DTW забезпечує можливість порівняння часових послідовностей різної довжини, що особливо важливо при розпізнаванні слів різної тривалості. Алгоритм знаходить оптимальне відображення між двома послідовностями шляхом мінімізації відстані між ними. Це дозволяє враховувати природні варіації у швидкості вимови та тривалості звуків.

На сучасному етапі розвитку методів розпізнавання аудіо широко застосовуються глибокі нейронні мережі. CNN – згорткові нейронні мережі демонструють високу ефективність при обробці спектрограм аудіо сигналів, дозволяючи виявляти локальні частотно-часові патерни. Архітектури типу ResNet та DenseNet, які були спочатку розроблені для обробки зображень, успішно адаптовані для задач аудіо розпізнавання [12].

RNN – рекурентні нейронні мережі, зокрема їх модифікації з LSTM та GRU комірками, ефективно моделюють довготривалі залежності в послідовностях аудіо ознак. Двонаправлені архітектури дозволяють враховувати як попередній, так і наступний контекст при аналізі кожного елемента послідовності. Механізми уваги в RNN підвищують здатність мережі фокусуватися на найбільш релевантних частинах вхідного сигналу [13].

Архітектура Transformer, що використовує механізм самоуваги, стала проривом у області розпізнавання мовлення. Здатність цієї архітектури паралельно обробляти всі елементи послідовності та встановлювати зв'язки між довільними позиціями забезпечує високу якість розпізнавання. Позиційне кодування дозволяє зберігати інформацію про порядок елементів у послідовності.

Для підвищення якості розпізнавання застосовуються різноманітні методи адаптації моделей. Адаптація до диктора дозволяє враховувати

індивідуальні особливості голосу, адаптація мовної моделі покращує розпізнавання специфічної лексики, а адаптація до акустичного оточення підвищує стійкість до шумів та реверберації. Доменна адаптація забезпечує ефективне перенесення знань між різними предметними областями.

Важливу роль відіграють методи розширення навчальних даних (data augmentation). Створення синтетичних варіацій навчальних прикладів шляхом додавання шуму, зміни швидкості відтворення та висоти звуку підвищує стійкість моделей до різних типів спотворень. Симуляція реверберації дозволяє краще враховувати акустичні особливості різних приміщень.

Мовне моделювання є критичним компонентом сучасних систем розпізнавання. Статистичні та нейромережеві мовні моделі враховують граматичні та семантичні закономірності мови, що дозволяє виправляти помилки акустичного розпізнавання. Контекстно-залежне моделювання забезпечує більш точне розпізнавання слів з урахуванням їх оточення [14].

Сучасні тенденції розвитку включають створення end-to-end систем, які здійснюють пряме перетворення аудіо в текст без проміжних етапів. Такий підхід спрощує архітектуру системи та часто демонструє кращі результати порівняно з традиційними гібридними підходами. Уніфікована архітектура дозволяє оптимізувати всі компоненти системи спільно [5].

Значна увага приділяється методам самонавчання та навчання без учителя, які дозволяють ефективно використовувати великі обсяги нерозмічених даних. Контрастивне навчання та автоенкодери забезпечують вилучення інформативних ознак з аудіо сигналів, а методи псевдо-розмітки даних розширюють можливості навчання моделей.

Багатозадачне навчання дозволяє одночасно вирішувати кілька пов'язаних задач, таких як розпізнавання мовлення, ідентифікація диктора та визначення емоційного стану. Це підвищує ефективність використання навчальних даних та покращує загальну якість розпізнавання. Сегментація дикторів забезпечує можливість роботи з багатоголосними записами.

В таблиці 1.6 представлено порівняльний аналіз основних методів та алгоритмів розпізнавання аудіо сигналів. З таблиці видно, що кожен метод має свої переваги та обмеження, що визначає його оптимальну сферу застосування.

Таблиця 1.6 - Порівняння методів та алгоритмів розпізнавання аудіо

| Метод/<br>Алгоритм                                   | Принцип<br>роботи                                                             | Переваги                                                                                                                     | Недоліки                                                                                                | Ефективність<br>застосування                                                                                     |
|------------------------------------------------------|-------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------|
| 1                                                    | 2                                                                             | 3                                                                                                                            | 4                                                                                                       | 5                                                                                                                |
| Приховані<br>марковські<br>моделі (НММ)              | Моделювання<br>послідовності<br>звуків як<br>ланцюжка<br>прихованих<br>станів | - врахування<br>часової<br>динаміки;<br>- моделювання<br>варіативності<br>вимови;<br>- використання<br>алгоритму<br>Вітербі. | - складність<br>навчання;<br>- потреба у<br>великій<br>кількості<br>даних;<br>- обмежена<br>точність.   | - базові системи<br>розпізнавання;<br>- класифікація<br>послідовностей;<br>- інтеграція з<br>іншими<br>методами. |
| Динамічна<br>трансформація<br>часової шкали<br>(DTW) | Порівняння<br>часових<br>послідовностей<br>різної довжини                     | - простота<br>реалізації;<br>- робота з<br>різною<br>тривалістю;<br>- незалежність<br>від диктора.                           | - висока<br>обчислювальна<br>складність;<br>- чутливість до<br>шуму;<br>- обмежена<br>масштабованість.  | - розпізнавання<br>окремих слів;<br>- порівняння<br>патернів;<br>- прототипні<br>системи.                        |
| Згорткові<br>нейронні<br>мережі (CNN)                | Обробка<br>спектрограм та<br>виявлення<br>частотно-<br>часових<br>патернів    | - ефективна<br>обробка<br>локальних<br>патернів;<br>- стійкість до<br>зсувів;<br>- паралельна<br>обробка.                    | - потреба у<br>великих<br>датасетах;<br>- ресурсоемність<br>навчання;<br>- складність<br>інтерпретації. | - аналіз<br>спектрограм;<br>- класифікація<br>звуків;<br>- виділення<br>ознак.                                   |

Кінець таблиці 1.6

| 1                                     | 2                                                       | 3                                                                                               | 4                                                                                   | 5                                                                                 |
|---------------------------------------|---------------------------------------------------------|-------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|
| Рекурентні нейронні мережі (RNN/LSTM) | Моделювання довготривалих залежностей в послідовностях  | - обробка послідовностей довільної довжини;<br>- врахування контексту;<br>- адаптивне навчання. | - проблема зникаючого градієнта;<br>- складність навчання;<br>- послідовна обробка. | - розпізнавання мовлення;<br>- моделювання контексту;<br>- реальночасові системи. |
| Transformer                           | Паралельна обробка послідовності з механізмом самоуваги | - ефективна паралельна обробка;<br>- глобальний контекст;<br>- висока точність.                 | - квадратична складність;<br>- великі вимоги до пам'яті;<br>- складна архітектура.  | - сучасні системи розпізнавання;<br>- багатомовні системи;<br>- складні завдання. |
| End-to-end системи                    | Пряме перетворення аудіо в текст                        | - оптимізація всього процесу;<br>- спрощена архітектура;<br>- краща продуктивність.             | - потреба у великих датасетах;<br>- складність навчання;<br>- менша гнучкість.      | - промислові системи;<br>- інтегровані рішення;<br>- спеціалізовані застосування. |

Класичні методи, такі як HMM та DTW, залишаються актуальними для певних задач, особливо в умовах обмежених навчальних даних. Сучасні нейромережеві підходи, зокрема архітектури на основі CNN, RNN та Transformer, забезпечують вищу точність розпізнавання, але вимагають значних обчислювальних ресурсів та великих наборів даних для навчання [13].

End-to-end системи представляють новий етап у розвитку технологій розпізнавання, пропонуючи уніфікований підхід до обробки аудіо сигналів. Незважаючи на певні обмеження, такі системи демонструють високу

ефективність у практичних застосуваннях та визначають напрямок подальшого розвитку галузі [14].

Основними викликами в області розпізнавання аудіо залишаються забезпечення надійної роботи в умовах сильного шуму, обробка спонтанного мовлення з перекриттями та виправленнями, ефективне використання обчислювальних ресурсів та зменшення потреби в розмічених навчальних даних. Перспективними напрямками є розробка легких моделей для мобільних пристроїв, створення багатомовних систем, вдосконалення емоційного розпізнавання та розробка адаптивних систем реального часу.

Підсумовуючи, можна сказати, що сучасні методи та алгоритми розпізнавання аудіо сигналів представляють собою складний комплекс технологій, що постійно вдосконалюється. Інтеграція класичних методів з сучасними нейромережевими підходами дозволяє досягати високої якості розпізнавання в різноманітних умовах застосування. Основними тенденціями розвитку є підвищення ефективності використання обчислювальних ресурсів, покращення роботи в реальних умовах та розширення функціональності систем розпізнавання.

#### **1.4 Архітектурний підхід до побудови системи**

Архітектурний підхід до побудови системи розпізнавання аудіо сигналів базується на багаторівневій структурі, що забезпечує модульність, масштабованість та зручність у підтримці. Основною моделлю, яка використовується в сучасних веб-додатках, є архітектура клієнт-сервер з елементами розподілених систем. Для даної системи доцільно використовувати підхід SPA (Single Page Application), що поєднує інтерактивний фронтенд та продуктивний бекенд. У таблиці 1.7 представлено порівняльний аналіз різних архітектурних підходів, які можуть бути застосовані для побудови такої системи.

Таблиця 1.7 - Порівняння архітектурних підходів до побудови системи розпізнавання аудіо

| Архітектурний підхід      | Особливості                                                                                                                                     | Переваги                                                                                                                                      | Недоліки                                                                                                                                       | Сценарії використання                                                                                                                 |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| Монолітна архітектура     | <ul style="list-style-type: none"> <li>- єдина кодова база;</li> <li>- спільна база даних;</li> <li>- тісна інтеграція компонентів.</li> </ul>  | <ul style="list-style-type: none"> <li>- простота розробки;</li> <li>- легке тестування;</li> <li>- швидка початкова розробка.</li> </ul>     | <ul style="list-style-type: none"> <li>- складність масштабування;</li> <li>- важка підтримка;</li> <li>- обмежена гнучкість.</li> </ul>       | <ul style="list-style-type: none"> <li>- малі проекти;</li> <li>- прототипи;</li> <li>- обмежені ресурси.</li> </ul>                  |
| Мікросервісна архітектура | <ul style="list-style-type: none"> <li>- незалежні сервіси;</li> <li>- розподілені дані;</li> <li>- API-взаємодія.</li> </ul>                   | <ul style="list-style-type: none"> <li>- висока масштабованість;</li> <li>- гнучкість технологій;</li> <li>- ізоляція компонентів.</li> </ul> | <ul style="list-style-type: none"> <li>- складність розробки;</li> <li>- витрати на інфраструктуру;</li> <li>- складна координація.</li> </ul> | <ul style="list-style-type: none"> <li>- великі системи;</li> <li>- високе навантаження;</li> <li>- складна бізнес-логіка.</li> </ul> |
| SPA + REST API            | <ul style="list-style-type: none"> <li>- розділення frontend/backend;</li> <li>- rest комунікація;</li> <li>- єдина сторінка.</li> </ul>        | <ul style="list-style-type: none"> <li>- швидкий інтерфейс;</li> <li>- зручність використання;</li> <li>- чітке розділення.</li> </ul>        | <ul style="list-style-type: none"> <li>- початкове навантаження;</li> <li>- SEO складності;</li> <li>- стан клієнта.</li> </ul>                | <ul style="list-style-type: none"> <li>- веб-застосунки;</li> <li>- інтерактивні системи;</li> <li>- B2B рішення.</li> </ul>          |
| Serverless архітектура    | <ul style="list-style-type: none"> <li>- функції як сервіс;</li> <li>- автоматичне масштабування;</li> <li>- оплата за використання.</li> </ul> | <ul style="list-style-type: none"> <li>- низька вартість;</li> <li>- автоматичне масштабування;</li> <li>- швидке розгортання.</li> </ul>     | <ul style="list-style-type: none"> <li>- обмеження виконання;</li> <li>- залежність від провайдера;</li> <li>- cold start.</li> </ul>          | <ul style="list-style-type: none"> <li>- змінне навантаження;</li> <li>- MVP;</li> <li>- стартапи.</li> </ul>                         |
| Гібридна архітектура      | <ul style="list-style-type: none"> <li>- комбінація підходів;</li> <li>- оптимізація під задачі;</li> <li>- гнучка структура.</li> </ul>        | <ul style="list-style-type: none"> <li>- оптимальна продуктивність;</li> <li>- баланс ресурсів;</li> <li>- адаптивність.</li> </ul>           | <ul style="list-style-type: none"> <li>- складність управління;</li> <li>- потреба в експертизі;</li> <li>- вартість розробки.</li> </ul>      | <ul style="list-style-type: none"> <li>- складні системи;</li> <li>- специфічні вимоги;</li> <li>- enterprise рішення.</li> </ul>     |

Кожен підхід має свої переваги та обмеження. Монолітна архітектура, незважаючи на простоту розробки, має суттєві обмеження щодо масштабування.

Мікросервісна архітектура забезпечує високу гнучкість та масштабованість, але вимагає значних ресурсів на розробку та підтримку інфраструктури. Підхід SPA з REST API, обраний для даної системи, представляє оптимальний баланс між функціональністю та складністю реалізації.

Ефективна взаємодія технологій фронтенду та бекенду забезпечує злагоджену роботу системи, високу продуктивність і зручність використання для кінцевого користувача. У сучасних розробках фронтенд і бекенд розділяються, що дозволяє незалежно працювати над кожним компонентом та спрощує масштабування.

Фронтенд, розроблений за допомогою таких фреймворків, як React.js, відповідає за створення інтерактивного інтерфейсу користувача. Його основною задачею є забезпечення швидкої та динамічної взаємодії, де зміни вмісту сторінки можуть відбуватися без необхідності її повного перезавантаження. React.js також надає можливості для роботи зі станом програми, управління подіями та динамічного оновлення даних, що робить його ідеальним вибором для побудови односторінкових додатків (SPA).

Бекенд, реалізований за допомогою Django, виконує роль сервера, який обробляє запити від фронтенду, взаємодіє з базами даних і виконує основну бізнес-логіку. Django відзначається високим рівнем безпеки, гнучкістю та можливістю швидкої розробки, що дозволяє забезпечити ефективну обробку складних алгоритмів розпізнавання аудіо сигналів. Django добре підходить для швидкої розробки завдяки своєму вбудованому ORM (Object-Relational Mapping), що спрощує роботу з базами даних [15].

Крім того, можливості розширення Django дозволяють інтегрувати сторонні бібліотеки для аналізу аудіо сигналів, наприклад, Librosa або PyTorch. Використання REST API в Django дозволяє фронтенду і бекенду комунікувати через HTTP-запити, що спрощує інтеграцію та підтримує модульність системи [15].

Особливості інтеграції компонентів полягають у необхідності чітко визначити кінцеві точки API, які фронтенд буде використовувати для взаємодії з

бекендом. Для цього зазвичай створюється набір RESTful API, який включає маршрути для надсилання аудіо сигналів на сервер, отримання результатів розпізнавання, управління даними користувачів тощо. Ці API забезпечують ефективну передачу даних між клієнтом і сервером у форматі JSON, що є стандартом у сучасних веб-додатках.

Важливим аспектом є також обробка даних у реальному часі, яка може вимагати використання вебсокетів або технологій потокової передачі даних, щоб мінімізувати затримку між надсиланням аудіо користувачем і отриманням результатів аналізу. У таких випадках фронтенд забезпечує зручний запис аудіо або його завантаження, тоді як бекенд виконує необхідну обробку сигналу і передає результати.

Інтеграція фронтенду і бекенду повинна враховувати також питання безпеки, зокрема аутентифікацію та авторизацію користувачів. Для цього можуть бути використані токени доступу (наприклад, JWT), що дозволяють обмежити доступ до певних функцій додатка і захистити персональні дані користувачів [16].

Для зберігання інформації доцільно використовувати реляційну базу даних, наприклад, PostgreSQL, яка вирізняється високою надійністю та здатністю обробляти великі обсяги даних. У цій базі даних можна зберігати результати аналізу, історію взаємодій користувачів із системою, а також конфігураційні параметри, необхідні для функціонування системи. [17].

Важливим елементом архітектури є модуль обробки аудіо сигналів, який може бути реалізований як окремий сервіс у межах мікросервісної архітектури. Цей сервіс обробляє аудіо сигнали, виділяє з них ознаки та передає результати розпізнавання до бекенду. Для підвищення швидкодії та масштабованості такий модуль може бути винесений на окремий сервер або у хмару.

Комунікація між компонентами системи відбувається за допомогою HTTP-запитів або вебсокетів. Для інтерактивних сценаріїв, таких як обробка аудіо в реальному часі, вебсокети є кращим вибором, оскільки вони дозволяють передавати дані в режимі потоків із мінімальними затримками. У випадку

стандартної взаємодії, наприклад, надсилання аудіофайлу та отримання результату, використовуються RESTful API [18].

Захист системи передбачає впровадження протоколу шифрування HTTPS для передачі даних, а також токенів для аутентифікації (JWT). Це дозволяє захистити конфіденційність користувачів та забезпечити контроль доступу до функціональності системи.

У таблиці 1.8 наведено опис компонентів системи розпізнавання аудіо сигналів, включаючи технології їх реалізації, функціональність та особливості взаємодії.

Таблиця 1.8 - Характеристика компонентів системи розпізнавання аудіо

| Компонент            | Технологія  | Функціональність                                                          | Взаємодія                                                          | Особливості безпеки                                                    |
|----------------------|-------------|---------------------------------------------------------------------------|--------------------------------------------------------------------|------------------------------------------------------------------------|
| Frontend             | React.js    | - інтерфейс користувача;<br>- запис аудіо;<br>- відображення результатів. | - REST API запити;<br>- WebSocket з'єднання;<br>- JWT авторизація. | - XSS захист;<br>- HTTPS;<br>- валідація введення.                     |
| Backend              | Django      | - бізнес-логіка;<br>- обробка запитів;<br>- управління даними.            | - REST API endpoint;<br>- ORM;<br>- WebSocket server.              | - JWT токени;<br>- CORS політики;<br>- захист від SQL-ін'єкцій.        |
| База даних           | PostgreSQL  | - зберігання результатів;<br>- користувацькі дані;<br>- налаштування.     | - ORM Django;<br>- SQL запити;<br>- реплікація.                    | - шифрування даних;<br>- Backup;<br>- контроль доступу.                |
| Модуль обробки аудіо | Python + ML | - аналіз аудіо;<br>- виділення ознак;<br>- розпізнавання.                 | - API інтеграція;<br>- асинхронна обробка;<br>- кешування.         | - валідація файлів;<br>- обмеження розміру;<br>- сканування на віруси. |

Frontend частина, реалізована на React.js, забезпечує інтерактивний користувацький інтерфейс та ефективну обробку аудіо на стороні клієнта. Backend на Django надає надійну платформу для реалізації бізнес-логіки та інтеграції з системами розпізнавання. PostgreSQL забезпечує надійне зберігання даних з можливістю масштабування. Особлива увага приділяється безпеці системи, що реалізується на всіх рівнях архітектури. Frontend включає захист від XSS-атак та валідацію введення користувача. Backend забезпечує автентифікацію через JWT токени та захист від SQL-ін'єкцій. База даних використовує шифрування та контроль доступу для захисту користувацьких даних.

Модуль обробки аудіо реалізований як окремий компонент, що дозволяє гнучко масштабувати обчислювальні ресурси залежно від навантаження. Взаємодія між компонентами здійснюється через чітко визначені API інтерфейси, що забезпечує слабе зв'язування та можливість незалежного розвитку кожного компонента.

Таким чином, архітектурний підхід до побудови системи розпізнавання аудіо сигналів поєднує SPA-архітектуру для фронтенду, потужний бекенд на основі Django, а також модуль обробки аудіо з використанням машинного навчання. Ця структура забезпечує ефективність, масштабованість і зручність підтримки системи, адаптуючи її до змінних потреб користувачів і бізнесу.

## **1.5 Висновки за розділом 1**

Визначено, що розпізнавання аудіо сигналів є однією з ключових складових сучасних інформаційних технологій, яка знаходить застосування у різноманітних галузях. Основою сучасних систем є поєднання класичних методів обробки сигналів і передових технологій машинного навчання, таких як глибокі нейронні мережі, включаючи згорткові, рекурентні та трансформерні архітектури.

Показано наявність двох основних підходів до реалізації систем розпізнавання: хмарні сервіси, що забезпечують високу точність але потребують постійного підключення до мережі, та локальні рішення, що гарантують конфіденційність даних при нижчій точності розпізнавання.

Визначено ключові методи та алгоритми обробки аудіо сигналів, включаючи попередню обробку, виділення акустичних ознак та використання різних архітектур нейронних мереж для кінцевого розпізнавання.

Показано, що архітектурний підхід до побудови систем розпізнавання аудіо сигналів, який включає SPA-додатки на основі React.js, бекенд на Django і окремий модуль обробки аудіо, забезпечує модульність, масштабованість і ефективну інтеграцію.

Встановлено необхідність розробки системи, що забезпечує баланс між точністю розпізнавання та практичністю використання, з урахуванням сучасних вимог до обробки аудіо сигналів.

## 2 МАТЕРІАЛИ ТА МЕТОДИ

Технічне завдання описує розробку вебзастосунку для розпізнавання аудіо сигналів на базі фреймворків Django та React.js. Система призначена для надання користувачам зручного інтерфейсу завантаження та обробки аудіофайлів з можливістю вибору методу розпізнавання та формату виведення результатів. Основними завданнями системи є забезпечення точного розпізнавання аудіо, зручності використання та надійності роботи. Система має підтримувати популярні формати аудіофайлів та забезпечувати збереження результатів розпізнавання. На рисунку 2.1 представлено структуру технічного завдання, яка включає чотири основні блоки: терміни та визначення, функціональні вимоги, нефункціональні вимоги та опис системи. Функціональні вимоги деталізовані трьома основними компонентами: керування аудіофайлами, вибір системи розпізнавання та робота з результатами. Нефункціональні вимоги охоплюють аспекти інтерфейсу користувача, підтримки браузерів, продуктивності та безпеки. Така структура забезпечує охоплення всіх аспектів розроблюваної системи. Текст технічного завдання наведено в Додатку А.



Рисунок 2.1 - Структура технічного завдання

## 2.1 Терміни та визначення

Для кращого розуміння специфіки системи та її компонентів, всі використані терміни згруповано за категоріями в таблиці 2.1. Така класифікація дозволяє структурувати термінологічну базу проєкту та забезпечує чітке розуміння використаних понять.

Таблиця 2.1 - Класифікація термінів та визначень

| Категорія               | Терміни       | Визначення                            |
|-------------------------|---------------|---------------------------------------|
| Архітектурні компоненти | Система       | Вебзастосунок для розпізнавання аудіо |
|                         | API           | Інтерфейс програмування               |
|                         | MVC           | Архітектурний патерн                  |
|                         | DB            | База даних                            |
| Технології розробки     | Django        | Python-фреймворк                      |
|                         | React.js      | JavaScript бібліотека                 |
|                         | REST API      | Архітектурний стиль                   |
|                         | WebSocket     | Протокол комунікації                  |
| Користувацькі ролі      | Користувач    | Особа, що працює з системою           |
|                         | Адміністратор | Особа, що керує системою              |
| Бізнес-терміни          | Аудіо сигнал  | Звукові дані                          |
|                         | Розпізнавання | Процес обробки                        |
|                         | Транскрибація | Перетворення в текст                  |

Представлена класифікація термінів охоплює всі ключові аспекти системи: від архітектурних компонентів до бізнес-термінів, що забезпечує єдине розуміння термінології всіма учасниками проєкту.

## 2.2 Функціональні вимоги

У таблиці 2.2 наведено пріоритезацію функціональних вимог системи з визначенням критеріїв прийняття та залежностей між компонентами. Така структуризація допомагає планувати розробку та впровадження функціоналу системи.

Таблиця 2.2 - Пріоритезація функціональних вимог

| Функціональна вимога        | Пріоритет | Критерії прийняття                                                                                         | Залежності             |
|-----------------------------|-----------|------------------------------------------------------------------------------------------------------------|------------------------|
| Керування аудіофайлами      | Високий   | - підтримка MP3/WAV;<br>- перевірка розміру;<br>- валідація формату;<br>- зміна/видалення поточного файлу. | -                      |
| Вибір системи розпізнавання | Високий   | - наявність вибору;<br>- збереження налаштувань.                                                           | Керування файлом       |
| Генерація результату        | Високий   | - точність розпізнавання;<br>- обробка помилок.                                                            | Вибір системи          |
| Збереження результатів      | Середній  | - різні формати;<br>- історія операцій.                                                                    | Генерація результату   |
| Управління історією         | Низький   | - перегляд попередніх результатів;<br>- фільтрація.                                                        | Збереження результатів |

Визначені пріоритети та критерії прийняття забезпечують чітке розуміння послідовності розробки та впровадження функціоналу, а також встановлюють чіткі критерії оцінки якості реалізації кожної вимоги.

## 2.3 Нефункціональні вимоги

Інтерфейс користувача:

- адаптивний дизайн для різних пристроїв;
- інтуїтивно зрозуміле розташування елементів;
- миттєвий відгук на дії користувача.

Підтримка браузерів:

- Mozilla Firefox (останні версії);
- Google Chrome (останні версії);
- Safari (останні версії);
- Opera (останні версії).

Таблиця 2.3 містить метрики для оцінки відповідності системи нефункціональним вимогам. Кожна метрика має визначену критичність, що відображає її важливість для успішного функціонування системи.

Таблиця 2.3 - Метрики нефункціональних вимог

| Категорія       | Вимога                | Метрика | Критичність |
|-----------------|-----------------------|---------|-------------|
| Продуктивність  | Час відгуку           | < 2 сек | Висока      |
| Швидкодія       | Обробка запитів       | < 5 сек | Висока      |
| Навантаження    | Одночасні користувачі | > 100   | Середня     |
| Доступність     | Час роботи            | 99.9%   | Висока      |
| Безпека         | Шифрування даних      | HTTPS   | Висока      |
| Масштабованість | Зростання даних       | 1TB/рік | Середня     |

Визначені метрики забезпечують об'єктивну оцінку якості системи та встановлюють конкретні цільові показники для розробки. Особлива увага приділяється критичним параметрам, таким як час відгуку системи та безпека даних.

## **2.4 Характеристика досліджуваної сфери (предметної області)**

Предметна область охоплює розробку системи розпізнавання аудіо сигналів через веб-інтерфейс. Система призначена для автоматизованої обробки аудіофайлів та їх перетворення в текстовий формат. Особливістю даної системи є можливість вибору різних методів розпізнавання, що дозволяє користувачам обрати оптимальний варіант залежно від потреб та якості вхідного аудіо.

Процес взаємодії з системою реалізований через веб-браузер, де користувач має можливість завантажити аудіофайл у форматах MP3 або WAV. Система проводить валідацію файлу, перевіряючи його розмір та формат. Після успішного завантаження користувач обирає метод розпізнавання та запускає процес обробки. Результати розпізнавання відображаються на екрані та можуть бути збережені в різних форматах.

Важливим є управління помилками та надання користувачу зворотного зв'язку щодо статусу виконуваних операцій. При виникненні проблем система надає зрозумілі повідомлення та можливість повторної спроби. Всі результати розпізнавання записуються у базу даних, що дозволяє користувачам повертатися до попередніх результатів та завантажувати їх у потрібному форматі.

Подібна система може стати корисним інструментом для різноманітної аудиторії користувачів: від студентів та викладачів, які працюють з аудіо лекціями, до професіоналів, які потребують швидкого та якісного перетворення аудіо в текст. Особлива увага приділяється простоті використання та інтуїтивності інтерфейсу, що робить систему доступною навіть для користувачів без спеціальних технічних знань.

## **2.5 Опис системи**

У сучасних умовах швидкого зростання кількості цифрових даних особливо актуальною стає потреба в автоматизованих системах для обробки та аналізу аудіоінформації. Розпізнавання аудіо сигналів стає критично важливим

для багатьох галузей - від медичної діагностики до систем безпеки та розваг. Така технологія дозволяє не лише перетворювати звук у текст, але й аналізувати емоційний стан мовця, виявляти аномалії у звукових сигналах та класифікувати різні типи звуків.

Створення ефективної системи розпізнавання аудіо вимагає вирішення низки технічних викликів. Серед них - забезпечення точності розпізнавання в умовах шуму, обробка різноманітних форматів аудіо, оптимізація швидкодії та захист конфіденційних даних. Особлива увага приділяється зручності використання та інтеграції з існуючими інформаційними системами.

Розроблена система надає користувачам повний цикл обробки аудіо сигналів - від завантаження файлу до отримання готового результату. Реалізовано можливість вибору методу розпізнавання, що дозволяє оптимізувати процес під конкретні потреби користувача. Система забезпечує надійне збереження результатів та можливість їх подальшого використання.

Важливою особливістю є модульна архітектура, що забезпечує гнучкість у розширенні функціоналу та адаптації до нових вимог. Система може бути використана як самостійний продукт або інтегрована в більші інформаційні комплекси, що робить її універсальним інструментом для різних сфер застосування.

## **2.6 Огляд інструментів розробки**

Для реалізації системи розпізнавання аудіо сигналів обрано набір сучасних інструментів, кожен із яких корелює певний аспект функціонування системи. Django забезпечує надійну серверну частину та обробку даних, React.js відповідає за інтерактивний користувацький інтерфейс, а SQLite надає ефективне зберігання даних. Для реалізації функціональності розпізнавання та обробки аудіосигналів використовуються спеціалізовані бібліотеки: Librosa для аналізу аудіосигналів, TensorFlow та Keras для реалізації моделей машинного

навчання, Scikit-learn для попередньої обробки даних. Візуалізація результатів аналізу забезпечується за допомогою бібліотек Matplotlib та Seaborn.

Обрані інструменти формують цілісну екосистему для розробки сучасного веб-застосунку, де кожен компонент оптимально виконує свої функції, забезпечуючи надійність, безпеку та ефективність системи в цілому.

### 2.6.1 Огляд Django

Django є високорівневим веб-фреймворком, написаним на мові Python, який забезпечує швидку розробку безпечних та підтримуваних веб-застосунків. Створений у 2003 році для новинного сайту, Django швидко став одним з найпопулярніших інструментів веб-розробки завдяки своїй філософії "batteries included" (все включено), що передбачає наявність всіх необхідних компонентів у складі базового фреймворку.

Архітектурно Django базується на патерні MTV (Model-Template-View), що є адаптацією класичного MVC (Model-View-Controller). У цій архітектурі Model відповідає за організацію структури даних і реалізацію бізнес-логіки, Template забезпечує представлення даних користувачу, а View виступає в ролі контролера, обробляючи запити та координуючи взаємодію між моделлю та шаблоном. Така архітектура гарантує чітке розмежування функцій між різними компонентами програми, що полегшує розробку та підтримку коду [19].

Система ORM (Object-Relational Mapping) є одним з найпотужніших компонентів Django. Вона надає абстрактний шар для взаємодії з реляційними базами даних через Python-об'єкти, що значно спрощує процес розробки. ORM підтримує широкий спектр операцій: від простих CRUD-операцій (Create, Read, Update, Delete) до складних запитів з використанням Q-об'єктів для складної фільтрації, агрегаційних функцій та транзакцій. Важливою особливістю є підтримка різних систем керування базами даних (PostgreSQL, MySQL, SQLite, Oracle) з можливістю легкого переключення між ними [19].

Безпека в Django реалізована на декількох рівнях. Фреймворк автоматично захищає від найпоширеніших веб-вразливостей: SQL-ін'єкцій через використання параметризованих запитів ORM, міжсайтового скриптингу (XSS) завдяки автоматичному екрануванню даних у шаблонах, підробки міжсайтових запитів (CSRF) через систему токенів. Вбудована система автентифікації пропонує готові рішення для управління користувачами, сесіями та дозволами, включаючи підтримку груп користувачів та деталізованих прав доступу.

Маршрутизація URL в Django реалізована через систему URLconf, яка забезпечує гнучке зіставлення URL-адрес з функціями-обробниками. Підтримка регулярних виразів та конвертерів шляхів дозволяє створювати складні схеми маршрутизації. Важливими особливостями є можливість реверсивного розв'язання URL за їх іменами, що робить код більш підтримуваним, та підтримка вкладених URL-конфігурацій через `include()`, що дозволяє модульно організовувати маршрути у великих проектах.

Система шаблонів Django надає потужний інструментарій для створення динамічних веб-сторінок. Вона підтримує наслідування шаблонів через механізм розширення базових шаблонів, включення фрагментів через теги `include`, користувацькі теги та фільтри для розширення функціональності. Важливою особливістю є контекстні процесори, які дозволяють автоматично додавати дані до контексту всіх шаблонів, наприклад, для відображення даних поточного користувача або глобальних налаштувань сайту.

Форми в Django представляють собою потужний механізм для обробки користувацького введення. Вони забезпечують автоматичну валідацію даних, генерацію HTML-розмітки та обробку файлових завантажень. `ModelForm` дозволяє автоматично створювати форми на основі моделей, що значно прискорює розробку. Підтримка власних віджетів та валідаторів надає можливість розширювати стандартну функціональність форм.

Django REST Framework (DRF) розширює можливості Django для створення API. Він надає потужні інструменти для серіалізації даних, автоматичної генерації API-документації через `swagger/OpenAPI`, управління

автентифікацією та авторизацією. Підтримка різних форматів даних (JSON, XML), версіонування API, обмеження швидкості запитів та кешування роблять DRF незамінним інструментом для розробки сучасних веб-сервісів [20].

Адміністративний інтерфейс Django (admin site) є унікальною особливістю фреймворку, що автоматично генерує інтерфейс для управління даними на основі моделей. Він підтримує налаштування відображення полів, фільтрацію, пошук та групові операції. Можливість розширення та кастомізації адмін-інтерфейсу дозволяє створювати повноцінні системи управління контентом.

Система міграцій Django забезпечує версіонування схеми бази даних та її синхронізацію з моделями. Міграції автоматично генеруються на основі змін у моделях та можуть бути застосовані або відкочені, що спрощує розгортання та оновлення застосунків. Підтримка залежностей між міграціями забезпечує коректну послідовність їх застосування.

Кешування в Django реалізовано на декількох рівнях: від кешування окремих звернень до бази даних до кешування цілих сторінок. Підтримуються різні бекенди кешування (Memcached, Redis, файлова система), що дозволяє обрати оптимальне рішення залежно від вимог проекту. Система сигналів забезпечує автоматичне оновлення кешу при зміні даних.

Тестування є невід'ємною частиною розробки на Django. Фреймворк надає вбудований модуль тестування, що базується на unittest, але розширює його специфічними для веб-розробки можливостями. Підтримка фікстур, тестових клієнтів та можливість ізоляції тестової бази даних забезпечують ефективне тестування застосунків.

Інтернаціоналізація та локалізація в Django реалізовані через систему перекладів gettext та підтримку часових поясів. Фреймворк дозволяє легко створювати багатомовні сайти з автоматичним перемиканням мов та форматуванням дат, чисел та валют відповідно до локальних стандартів.

Django також надає широкі можливості для розширення через систему додатків (apps). Модульна архітектура дозволяє легко інтегрувати сторонні

компоненти та повторно використовувати код між проектами. Велика екосистема сторонніх пакетів розширює можливості фреймворку для вирішення специфічних задач.

Підсумовуючи, Django є комплексним рішенням для веб-розробки, що поєднує потужність, гнучкість та безпеку. Фреймворк постійно розвивається, зберігаючи при цьому зворотну сумісність та стабільність API. Це робить його відмінним вибором як для невеликих проектів, так і для великих корпоративних застосунків, особливо в контексті розробки систем з високими вимогами до безпеки та масштабованості.

### 2.6.2 Огляд React.js

React.js, створений компанією Facebook (тепер Meta), представляє одну з найпоширеніших бібліотек JavaScript для розробки інтерфейсів користувача. З моменту свого публічного релізу в 2013 році React революціонізував підхід до розробки веб-застосунків, запровадивши декларативну парадигму та ефективні механізми оновлення DOM [21].

Фундаментальною концепцією React є компонентний підхід до розробки інтерфейсів. Компоненти є автономними та багаторазово використовуваними елементами інтерфейсу користувача, які можуть включати власну логіку, стан і методи життєвого циклу. React підтримує два типи компонентів: класові та функціональні. Класові компоненти, які історично з'явилися першими, базуються на ES6 класах і надають повний доступ до життєвого циклу компонента. Функціональні компоненти, які стали стандартом з появою хуків у React 16.8, пропонують більш лаконічний та зрозумілий спосіб написання компонентів [16].

Віртуальний DOM є ключовою інновацією React, що забезпечує високу продуктивність при оновленні інтерфейсу. Це легковажна копія реального DOM браузера, що зберігається в пам'яті. Коли стан застосунку змінюється, React спочатку оновлює віртуальний DOM, а потім використовує алгоритм узгодження

(reconciliation) для визначення мінімально необхідних змін у реальному DOM. Цей процес, відомий як "diffing", значно оптимізує продуктивність, Оскільки взаємодія з реальним DOM вимагає значних ресурсів і може впливати на продуктивність.

JSX (JavaScript XML) є невід'ємною частиною екосистеми React. Це розширення синтаксису JavaScript, що дозволяє писати HTML-подібну розмітку безпосередньо в JavaScript коді. JSX транспілюється в звичайний JavaScript за допомогою інструментів на кшталт Babel. Важливими особливостями JSX є:

- можливість вбудовування JavaScript виразів у фігурних дужках;
- автоматичне екранування значень для запобігання XSS-атак;
- підтримка атрибутів у camelCase нотації;
- можливість створення користувацьких атрибутів через data-атрибути.

Управління станом є критичним аспектом React-застосунків. React пропонує кілька механізмів для роботи зі станом:

- локальний стан компонентів через useState хук;
- контекст (Context API) для передачі даних через дерево компонентів без явної передачі пропсів;
- редюсери (useReducer) для управління складним станом та логікою;
- зовнішні бібліотеки управління станом, такі як Redux або MobX.

Хуки, представлені в React 16.8, радикально змінили підхід до розробки компонентів. Основні хуки включають:

- useState для управління локальним станом;
- useEffect для роботи з побічними ефектами;
- useContext для доступу до контексту;
- useRef для створення мутабельних референсів;
- useMemo та useCallback для оптимізації продуктивності;
- useEffect для синхронних побічних ефектів.

Продуктивність React-застосунків забезпечується кількома механізмами:

- мемоізація компонентів через React.memo;
- відкладене завантаження компонентів через React.lazy та Suspense;

- фрагменти (Fragments) для уникнення зайвих DOM-вузлів;
- рендеринг за пріоритетами через Concurrent Mode;
- профілювання та оптимізація через React Developer Tools.

React надає потужні інструменти для роботи з формами:

- контрольовані компоненти з двостороннім зв'язуванням даних;
- неконтрольовані компоненти з використанням референсів;
- валідація форм та обробка помилок;
- інтеграція з бібліотеками форм, такими як Formik або React Hook Form.

Обробка подій в React реалізована через синтетичні події (Synthetic Events), які забезпечують:

- крос-браузерну сумісність;
- делегування подій для підвищення продуктивності;
- доступ до нативних подій браузера;
- можливість зупинки поширення подій.

Життєвий цикл компонентів в React включає кілька фаз:

- монтування (mounting);
- оновлення (updating);
- розмонтування (unmounting);
- обробка помилок (error handling).

React підтримує кілька важливих паттернів розробки:

- компоненти вищого порядку (HOC);
- рендер-пропси (Render Props);
- композиція компонентів;
- інверсія контролю.

Екосистема React включає багато додаткових інструментів та бібліотек:

- React Router для маршрутизації;
- Redux та MobX для управління станом;
- Styled Components та Emotion для стилізації;
- React Query та SWR для управління серверним станом;
- Testing Library для тестування;

- Next.js та Gatsby для серверного рендерингу.

React також надає потужні можливості для оптимізації:

- Code Splitting для розділення коду;
- Tree Shaking для видалення невикористаного коду;
- кешування компонентів та результатів обчислень;
- ледаче завантаження зображень та інших ресурсів.

Розробка на React підтримується потужними інструментами:

- Create React App для швидкого створення проектів;
- React Developer Tools для відлагодження;
- ESLint та Prettier для форматування коду;
- Jest та React Testing Library для тестування.

Підсумовуючи, React.js представляє собою потужну та гнучку бібліотеку для розробки користувацьких інтерфейсів, що поєднує високу продуктивність, зручність розробки та великий набір інструментів. Постійний розвиток бібліотеки та активна спільнота забезпечують її актуальність та ефективність у сучасній веб-розробці.

### 2.6.3 Огляд SQLite

SQLite є компактною, вбудованою реляційною базою даних, що реалізує більшість стандарту SQL. Її унікальність полягає у відсутності окремого серверного процесу - вся база даних зберігається в єдиному файлі на диску. Це робить SQLite ідеальним вибором для локальних застосунків та вбудованих систем, де важлива простота розгортання та мінімальне споживання ресурсів [22].

Архітектура SQLite забезпечує повну підтримку ACID-транзакцій (Atomicity, Consistency, Isolation, Durability), що гарантує цілісність даних навіть у випадку збоїв системи або відключення живлення. База даних використовує механізм блокування на рівні файлу, що забезпечує безпечний конкурентний доступ до даних з різних процесів та потоків.

Особливістю SQLite є її динамічна система типів, яка дозволяє зберігати значення будь-якого типу в будь-якому стовпці незалежно від оголошеного типу даних. Це надає гнучкість при розробці, хоча вимагає додаткової уваги до валідації даних на рівні застосунку. База даних підтримує індекси, включаючи складені та унікальні, що дозволяє оптимізувати швидкість пошуку та вибірки даних.

SQLite включає потужний оптимізатор запитів, який аналізує структуру запиту та доступні індекси для вибору найефективнішого плану виконання. Підтримуються складні SQL-конструкції, включаючи підзапити, об'єднання таблиць, групування та агрегатні функції. Вбудований механізм повнотекстового пошуку дозволяє ефективно шукати текстову інформацію.

Важливою перевагою SQLite є її надійність та стабільність, підтверджена великою кількістю автоматизованих тестів. База даних має вбудовані механізми відновлення після збоїв та захист від пошкодження даних. Формат файлу бази даних є стабільним та зворотно сумісним, що гарантує довготривале зберігання інформації.

Інтеграція SQLite з Django здійснюється через вбудований драйвер, що забезпечує повну підтримку всіх можливостей ORM. Це надає можливість взаємодіяти з базою даних через високорівневий інтерфейс, використовуючи об'єкти Python, при цьому зберігаючи можливість прямого виконання SQL-запитів коли це необхідно для оптимізації продуктивності.

#### **2.6.4 Розпізнавання та обробка аудіо сигналів**

Сучасні системи розпізнавання та обробки аудіосигналів базуються на комплексному використанні спеціалізованих програмних засобів, що забезпечують повний цикл обробки звукової інформації. При розробці таких систем важливим є вибір інструментів, що дозволяють ефективно працювати з різними аспектами аудіоданих: від базової обробки сигналів до складних алгоритмів розпізнавання.

Основу інструментального забезпечення складають бібліотеки для обробки аудіо, які реалізують базові операції з аудіосигналами. Такі бібліотеки як Librosa забезпечують широкий спектр функцій для аналізу звуку, включаючи завантаження аудіофайлів, маніпуляції з сигналами та екстракцію характеристик. PyDub та SoundFile пропонують низькорівневий доступ до аудіоданих, що особливо важливо при роботі з різними форматами файлів та потоковою передачею звуку [8].

Особливе місце в інструментарії обробки аудіосигналів займають засоби аналізу часових та частотних характеристик. Сучасні методи дозволяють досліджувати амплітудні, енергетичні та спектральні властивості сигналів. Важливим аспектом є можливість обчислення мел-частотних кепстральних коефіцієнтів (MFCC), які враховують особливості людського сприйняття звуку та широко використовуються в задачах розпізнавання мовлення [11].

Значний прогрес у розвитку систем розпізнавання аудіо пов'язаний із впровадженням методів машинного навчання. Сучасні фреймворки, такі як TensorFlow та PyTorch, надають потужний інструментарій для розробки та навчання нейронних мереж різних архітектур.

Базові нейронні мережі (NN) використовуються як фундаментальний інструмент для класифікації ознак, отриманих після попередньої обробки сигналу. Їх архітектура, що складається з послідовності повнозв'язних шарів, дозволяє виявляти складні нелінійні залежності в даних та формувати високорівневі абстракції з простих характеристик сигналу [12].

Згорткові нейронні мережі (CNN) демонструють особливу ефективність при роботі з часово-частотними представленнями аудіосигналів, такими як спектрограми та мел-спектрограми. Їх архітектура, що базується на операціях згортки, дозволяє автоматично виявляти локальні патерни та просторові особливості в даних. CNN здатні ефективно обробляти двовимірні представлення аудіосигналів, виділяючи характерні частотні компоненти та їх зміну в часі. Застосування операцій агрегації (pooling) в CNN забезпечує інваріантність до незначних зсувів та масштабування у вхідних даних [12].

Рекурентні нейронні мережі (RNN) спеціально розроблені для обробки послідовних даних, що робить їх особливо цінними при аналізі аудіосигналів. Завдяки наявності зворотних зв'язків, RNN здатні зберігати інформацію про попередні стани системи та враховувати часову динаміку сигналу. Це дозволяє ефективно моделювати довготривалі залежності в аудіоданих та враховувати контекст при розпізнаванні. Особливо ефективними є варіанти RNN з механізмами LSTM (Long Short-Term Memory) та GRU (Gated Recurrent Unit), які вирішують проблему зникаючого градієнта та здатні запам'ятовувати релевантну інформацію протягом довгих послідовностей [12].

Архітектури на основі трансформерів представляють новітній підхід до обробки послідовних даних. Завдяки механізму самоуваги (self-attention), трансформери здатні ефективно моделювати взаємозв'язки між різними частинами сигналу незалежно від їх взаємного розташування в послідовності. Це забезпечує можливість врахування як локальних, так і глобальних залежностей у даних. Багатошарова архітектура трансформерів, що включає механізми кодування позицій та багатоголову увагу, дозволяє досягати високої точності при розпізнаванні складних аудіосигналів [13].

Сучасні системи розпізнавання аудіо часто використовують гібридні підходи, комбінуючи різні типи нейронних мереж для досягнення оптимальних результатів. Наприклад, CNN можуть використовуватися для початкової обробки спектрограм, після чого отримані ознаки подаються на вхід RNN або трансформерів для аналізу часової динаміки. Такі комбіновані архітектури дозволяють ефективно використовувати переваги кожного типу нейронних мереж та досягати високої якості розпізнавання в різних умовах застосування.

Для попередньої обробки даних та оцінки результатів застосовуються універсальні інструменти машинного навчання. Бібліотека Scikit-learn надає засоби для масштабування даних, кодування категоріальних змінних та оцінки якості моделей. Важливим компонентом є засоби візуалізації, представлені бібліотеками Matplotlib та Seaborn, які дозволяють аналізувати характеристики сигналів та результати навчання моделей.

Важливим аспектом є попередня обробка сигналів, яка включає фільтрацію шумів, нормалізацію амплітуди та сегментацію на фрейми. Ці операції є критичними для забезпечення якості вхідних даних та, відповідно, точності розпізнавання. Сучасні інструменти надають різноманітні методи фільтрації та покращення якості сигналу, що дозволяє адаптувати систему до різних умов запису та типів шумів [11].

При виборі інструментальних засобів для конкретної системи розпізнавання аудіо необхідно враховувати ряд факторів: вимоги до точності розпізнавання, доступні обчислювальні ресурси, необхідність роботи в реальному часі, можливості інтеграції з іншими компонентами системи. Важливими критеріями також є наявність документації, активність спільноти розробників та стабільність роботи інструментів.

Таким чином, сучасний інструментарій для розпізнавання та обробки аудіосигналів представляє собою комплексну екосистему засобів, що дозволяють вирішувати широкий спектр задач: від базової обробки звуку до складних систем розпізнавання з використанням методів глибокого навчання. Правильний вибір та комбінація цих інструментів є ключовим фактором успішної реалізації систем розпізнавання аудіо.

## **2.7 Висновки за розділом 2**

Виявлено основні вимоги до розроблюваної системи розпізнавання аудіо сигналів, включаючи функціональні можливості завантаження та обробки аудіофайлів, вибору методу розпізнавання та роботи з результатами, а також нефункціональні вимоги щодо адаптивності інтерфейсу, продуктивності та безпеки.

Показано, що для реалізації системи оптимальним є використання комбінації технологій: Django для серверної частини, React.js для клієнтської частини та SQLite для зберігання даних. Для обробки аудіоданих обрано спеціалізовані бібліотеки: Librosa для аналізу аудіосигналів, TensorFlow і Keras

для реалізації та навчання моделей, Matplotlib і Seaborn для візуалізації результатів. Кожен з цих інструментів забезпечує необхідний функціонал та відповідає встановленим вимогам до системи.

Визначено архітектурні особливості обраних інструментів розробки: Django забезпечує надійну основу для серверної частини з вбудованими механізмами безпеки та ORM, React.js надає гнучкий інструментарій для створення інтерактивного користувацького інтерфейсу, а SQLite забезпечує надійне зберігання даних без необхідності окремого серверного процесу.

Встановлено, що обрана комбінація технологій формує цілісну екосистему для розробки сучасного веб-застосунку, де кожен компонент оптимально виконує свої функції, забезпечуючи надійність, безпеку та ефективність системи в цілому.

## **3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ**

### **3.1 Моделювання архітектури системи засобами UML**

Уніфікована мова моделювання (UML) надає стандартизований набір інструментів для візуалізації архітектури системи, що особливо важливо при розробці складних програмних рішень [23].

Використання UML при проєктуванні системи надає ряд суттєвих переваг на всіх етапах розробки. Насамперед, UML забезпечує стандартизований підхід до візуалізації архітектури системи, що значно спрощує комунікацію між членами команди розробки. Це особливо важливо при роботі над складними компонентами системи, такими як модулі розпізнавання аудіо та обробки результатів.

Важливою перевагою є можливість раннього виявлення потенційних проблем та архітектурних обмежень. Детальне моделювання системи на етапі проєктування дозволяє ідентифікувати можливі вузькі місця в продуктивності та потенційні проблеми з масштабуванням ще до початку реалізації. Наприклад, при проєктуванні системи розпізнавання аудіо це допомогло оптимізувати процес обробки файлів та взаємодію між клієнтською та серверною частинами.

UML також сприяє повторному використанню коду та компонентів. Чітка візуалізація структури системи допомагає виявити можливості для створення універсальних компонентів, які можуть бути використані в різних частинах системи. В нашому випадку це дозволило створити модульну архітектуру, де компоненти обробки аудіо можуть бути легко модифіковані або замінені без впливу на інші частини системи.

Документування системи за допомогою UML значно полегшує процес підтримки та подальшого розвитку програмного забезпечення. Діаграми служать наочним посібником для нових розробників, які приєднуються до проєкту, та допомагають швидко зрозуміти структуру та принципи роботи системи. Це

особливо цінно при довгостроковій підтримці проєкту, коли потрібно вносити зміни або додавати нову функціональність.

Використання UML також сприяє підвищенню якості кінцевого продукту, оскільки дозволяє ретельно продумати та спланувати всі аспекти системи перед початком реалізації. Це зменшує кількість помилок проєктування та знижує витрати на їх виправлення на пізніх етапах розробки.

Для проєктування системи було обрано п'ять основних типів UML-діаграм, кожна з яких відображає різні аспекти архітектури та функціонування системи.

Діаграма сценаріїв використання дозволяє визначити основні сценарії взаємодії користувачів з системою. В контексті розроблюваної системи розпізнавання аудіо, ця діаграма відображає такі ключові функції як завантаження файлів, вибір методу розпізнавання та отримання результатів. Особлива увага приділяється відображенню взаємозв'язків між різними варіантами використання та їх залежностями.

Діаграма активності застосовується для моделювання бізнес-процесів у системі. зокрема процесу обробки аудіофайлу від його завантаження до отримання результату розпізнавання. Вона дозволяє чітко визначити послідовність дій, умови прийняття рішень та можливі альтернативні потоки виконання.

Діаграма послідовності демонструє взаємодію між компонентами системи в часі, що особливо важливо для розуміння процесу обміну повідомленнями між клієнтською та серверною частинами застосунку. Ця діаграма допомагає оптимізувати процеси комунікації та виявити потенційні проблеми в архітектурі системи.

Діаграма класів ілюструє статичну архітектуру системи, показуючи класи, їх атрибути, методи та взаємозв'язки. У контексті нашої системи, ця діаграма є особливо важливою для розуміння взаємодії між різними компонентами Django та React.js.

Діаграма розгортання описує фізичну архітектуру системи, відображаючи розміщення програмних компонентів на апаратних ресурсах. Вона допомагає визначити оптимальну конфігурацію розгортання системи та забезпечити ефективну взаємодію між її компонентами.

Використання UML при проєктуванні системи розпізнавання аудіо дозволило створити чітку та зрозумілу документацію, що полегшує процес розробки та подальшої підтримки системи. Кожна з обраних діаграм доповнює одна одну, створюючи повне представлення про структуру та функціонування системи на різних рівнях абстракції.

### **3.2 Діаграма варіантів використання**

Діаграма варіантів використання (Use Case Diagram) слугує інструментом для моделювання поведінки системи з позиції кінцевого користувача. Вона відображає функціональні можливості системи та взаємодію між системою та її користувачами. Основними структурними елементами діаграми варіантів використання є актори, варіанти використання та зв'язки між ними [23].

Актори (Actors) являють собою ролі, які приймають зовнішні користувачі або системи при взаємодії з розроблюваною системою. На діаграмі актори зображуються у вигляді стилізованих людських фігур. Важливо розуміти, що актор не обов'язково представляє конкретну фізичну особу - це може бути роль користувача, зовнішня система або пристрій.

Варіанти використання (Use Cases) відображають функціональні можливості системи з точки зору акторів. Вони представляються на діаграмі у вигляді овалів з назвою функціональності всередині. Кожен варіант використання описує певний сценарій взаємодії між актором та системою, спрямований на досягнення конкретної мети. Варіанти використання повинні бути незалежними один від одного та описувати завершені дії.

Взаємозв'язки (Associations) на діаграмі варіантів використання можуть бути трьох основних типів. Асоціації зображуються у вигляді суцільних ліній і

показують прямий зв'язок між актором та варіантом використання, вказуючи, що актор безпосередньо взаємодіє з цією функціональністю системи. Зв'язок включення (include) відображається пунктирною стрілкою з позначкою «include» і вказує, що один варіант використання обов'язково включає функціональність іншого варіанту використання. Наприклад, для завантаження результату обов'язково потрібно вибрати тип файлу для збереження. Зв'язок розширення (extend) також зображується пунктирною стрілкою, але з позначкою «extend». Цей тип зв'язку показує, що один варіант використання може бути розширений функціональністю іншого варіанту використання за певних умов. Таке розширення є необов'язковим і може відбуватися лише при виконанні певних умов. Правильне використання цих елементів та зв'язків дозволяє створити чітку та зрозумілу модель функціональності системи, яка служить основою для подальшої розробки та документації. На рисунку 3.1 представлена діаграма варіантів використання розробленої системи.

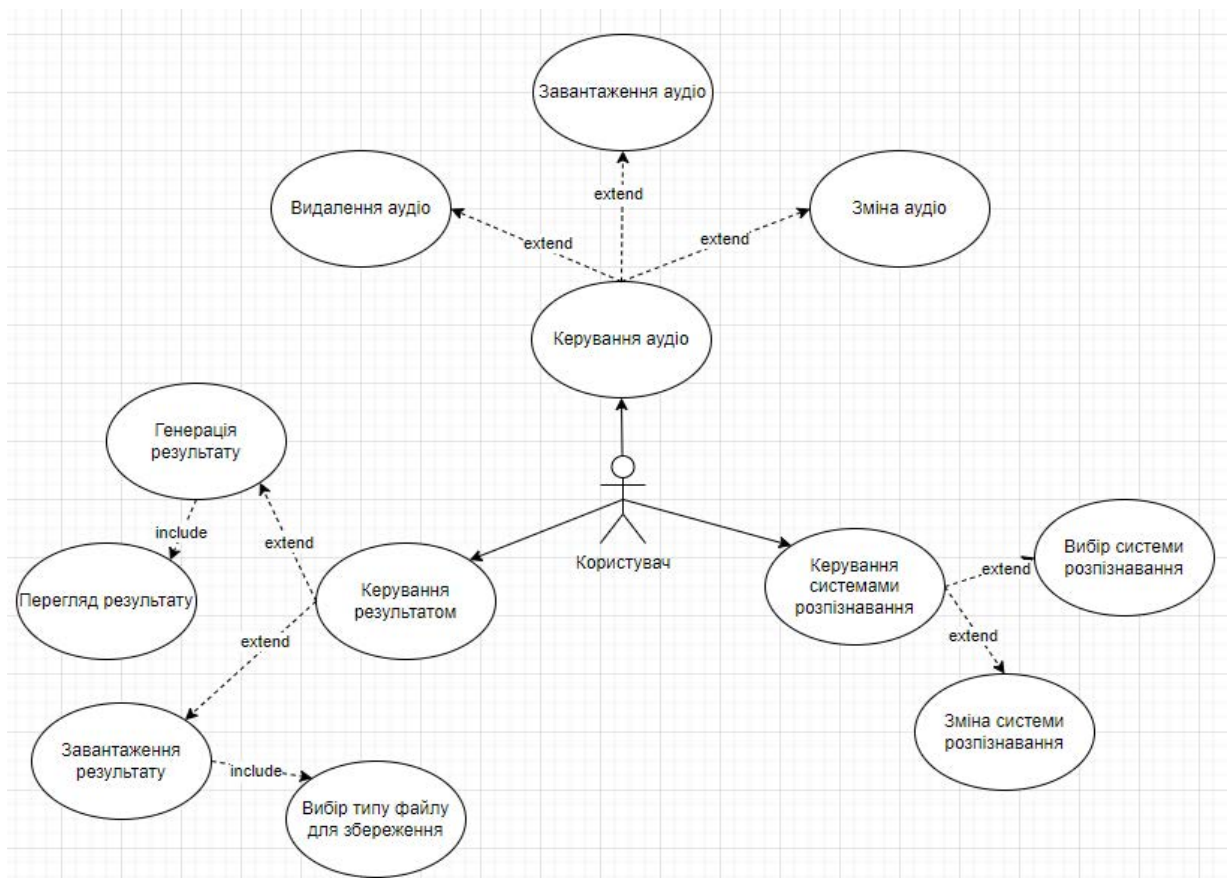


Рисунок 3.1 – Діаграма варіантів використання

В розробленій системі центральним актором є «Користувач», який взаємодіє з трьома основними групами функціональності: керування аудіо, керування системами розпізнавання та керування результатом.

Від основного варіанту використання «Керування аудіо» через зв'язок `extend` розходяться більш специфічні варіанти: завантаження, видалення та зміна аудіо. Це відображає можливість користувача виконувати різні операції з аудіофайлами. «Керування системами розпізнавання» розширюється варіантами вибору та зміни системи розпізнавання, що дозволяє користувачу налаштовувати процес обробки аудіо.

«Керування результатом» є основним елементом для роботи з результатами розпізнавання та має зв'язки з генерацією, переглядом та завантаженням результату. Процес генерації результату починається з моменту, коли користувач ініціює розпізнавання через інтерфейс системи, натискаючи кнопку «Розпізнати». Система виконує послідовність дій: спочатку аудіофайл передається на сервер через захищене з'єднання, де відбувається його валідація та попередня обробка. Далі, залежно від обраного методу розпізнавання, система застосовує відповідні алгоритми для аналізу аудіо сигналу. Результати обробки зберігаються в локальній базі даних, що забезпечує можливість подальшого доступу до них та ведення історії розпізнавань.

Після успішного завершення процесу розпізнавання, система формує відповідь, яка містить текстовий результат, та відображає його користувачу через інтерфейс. У випадку виникнення будь-яких помилок на будь-якому етапі обробки (некоректний формат файлу, помилка сервера, невдале розпізнавання), система генерує відповідне повідомлення про помилку. Такий підхід забезпечує надійний зворотний зв'язок з користувачем та підвищує зручність використання системи.

Важливим елементом є варіант використання «Вибір типу файлу для збереження», який включається у процес завантаження результату через зв'язок `include`.

Така структура діаграми чітко відображає всі можливі шляхи взаємодії користувача з системою та демонструє логічні зв'язки між різними функціональними можливостями. Це дозволяє краще зрозуміти потреби користувачів та забезпечити повноту реалізації необхідного функціоналу.

Розглянемо детально варіанти використання розробленої системи.

Варіант використання «Керування аудіо» забезпечує основну взаємодію користувача з аудіофайлами. Цей варіант активується коли користувач знаходиться на головній сторінці системи та виконує операції з файлами. При виникненні проблем з розміром файлу або невідповідністю формату система інформує користувача та пропонує альтернативні дії.

Процес «Завантаження аудіо» ініціюється користувачем через відповідну кнопку інтерфейсу. Система відкриває доступ до файлової системи користувача для вибору аудіофайлу. Після успішного вибору, інформація про файл відображається в інтерфейсі. Система контролює розмір та формат файлу, надаючи відповідні повідомлення при порушенні обмежень.

«Зміна аудіо» дозволяє користувачу оновити поточний аудіофайл. Цей варіант доступний тільки після початкового завантаження файлу. При натисканні на назву поточного файлу відкривається можливість вибору нового, з такими ж перевірками розміру та формату.

«Видалення аудіо» дозволяє користувачу видалити поточний аудіофайл через відповідну кнопку інтерфейсу. Цей варіант доступний після початкового завантаження файлу.

«Керування системами розпізнавання» надає функціонал для вибору та налаштування методу розпізнавання. «Вибір системи розпізнавання» надає можливість користувачу обрати між доступними системами через спеціальний селектор в інтерфейсі. «Зміна системи розпізнавання» дозволяє перемикатися між різними методами в процесі роботи.

«Керування результатом» є комплексним варіантом використання, що включає генерацію, перегляд та завантаження результатів розпізнавання. Для

роботи з результатами необхідна наявність завантаженого файлу та обраної системи розпізнавання.

«Генерація результату» запускається натисканням кнопки «Розпізнати». Система обробляє файл на сервері та зберігає результат в базі даних. При виникненні проблем з сервером або процесом розпізнавання, користувач отримує інформативне повідомлення про помилку з можливістю повторної спроби.

«Перегляд результату» та «Завантаження результату» стають доступними після успішної генерації. Система відображає текст розпізнавання в інтерфейсі та надає можливість зберегти результат на пристрій користувача. «Вибір типу файлу для збереження» дозволяє вибрати формат файлу для завантаження результату.

Кожен варіант використання має чітко визначені передумови та потік подій, що забезпечує логічну послідовність дій користувача та передбачувану поведінку системи.

### **3.3 Діаграма діяльності**

Діаграма діяльності (Activity Diagram) є одним з інструментів UML для візуального моделювання, що дозволяє детально представити послідовність дій та логіку роботи системи [23].

В контексті розробки системи розпізнавання аудіо, діаграма діяльності допомагає зрозуміти, як відбувається процес від моменту завантаження файлу до отримання результату розпізнавання. Вона відображає не лише послідовність дій, але й розподіл відповідальності між різними компонентами системи, можливі варіанти розвитку подій та обробку помилкових ситуацій.

Особливістю діаграми діяльності є її здатність наочно демонструвати паралельні процеси та умовні переходи, що особливо важливо при розробці веб-застосунків, де користувацький інтерфейс має взаємодіяти з серверною

частиною. Діаграма показує, як система обробляє запити користувача, як відбувається валідація даних, та які відповіді формуються в різних ситуаціях.

Використання діаграми діяльності на етапі проєктування дозволяє виявити потенційні проблеми в логіці роботи системи, оптимізувати процеси та забезпечити коректну обробку всіх можливих сценаріїв використання. Це робить її незамінним інструментом як для розробників, так і для комунікації з замовниками та іншими зацікавленими сторонами проєкту.

Діаграма діяльності використовує ряд стандартизованих елементів для моделювання процесів. Ключовим елементом є позначення станів системи: початковий стан відображається суцільним чорним колом, що символізує точку входу в процес, тоді як кінцевий стан позначається чорним колом з білою окантовкою, вказуючи на завершення всіх дій процесу.

Основний функціонал системи відображається через дії, які графічно представлені прямокутниками із заокругленими кутами. Кожна така дія описує конкретний крок процесу. Для показу послідовності виконання дій використовуються направлені стрілки, які з'єднують прямокутники дій між собою, чітко демонструючи потік управління в системі.

Важливим елементом є вузли прийняття рішень, які зображуються у вигляді ромбів. Ці елементи використовуються для моделювання розгалужень процесу, коли подальший хід виконання залежить від певної умови. Наприклад, перевірка успішності розпізнавання може призвести до різних гілок виконання. Такі ромби також використовуються для об'єднання різних потоків виконання назад в єдиний процес.

Для структурування діаграми та відображення відповідальності різних компонентів системи використовуються доріжки - вертикальні смуги, що розділяють діаграму на логічні зони. Кожна доріжка представляє певний компонент системи (наприклад, користувацький інтерфейс, серверну частину) або учасника процесу, показуючи, хто відповідає за виконання конкретних дій. Це допомагає краще зрозуміти розподіл обов'язків та взаємодію між різними частинами системи.

На представлений на рисунку 3.2 діаграмі відображено процес генерації результату розпізнавання аудіо з розподілом дій між трьома компонентами системи: Користувач, React (клієнтська частина) та Django (серверна частина).

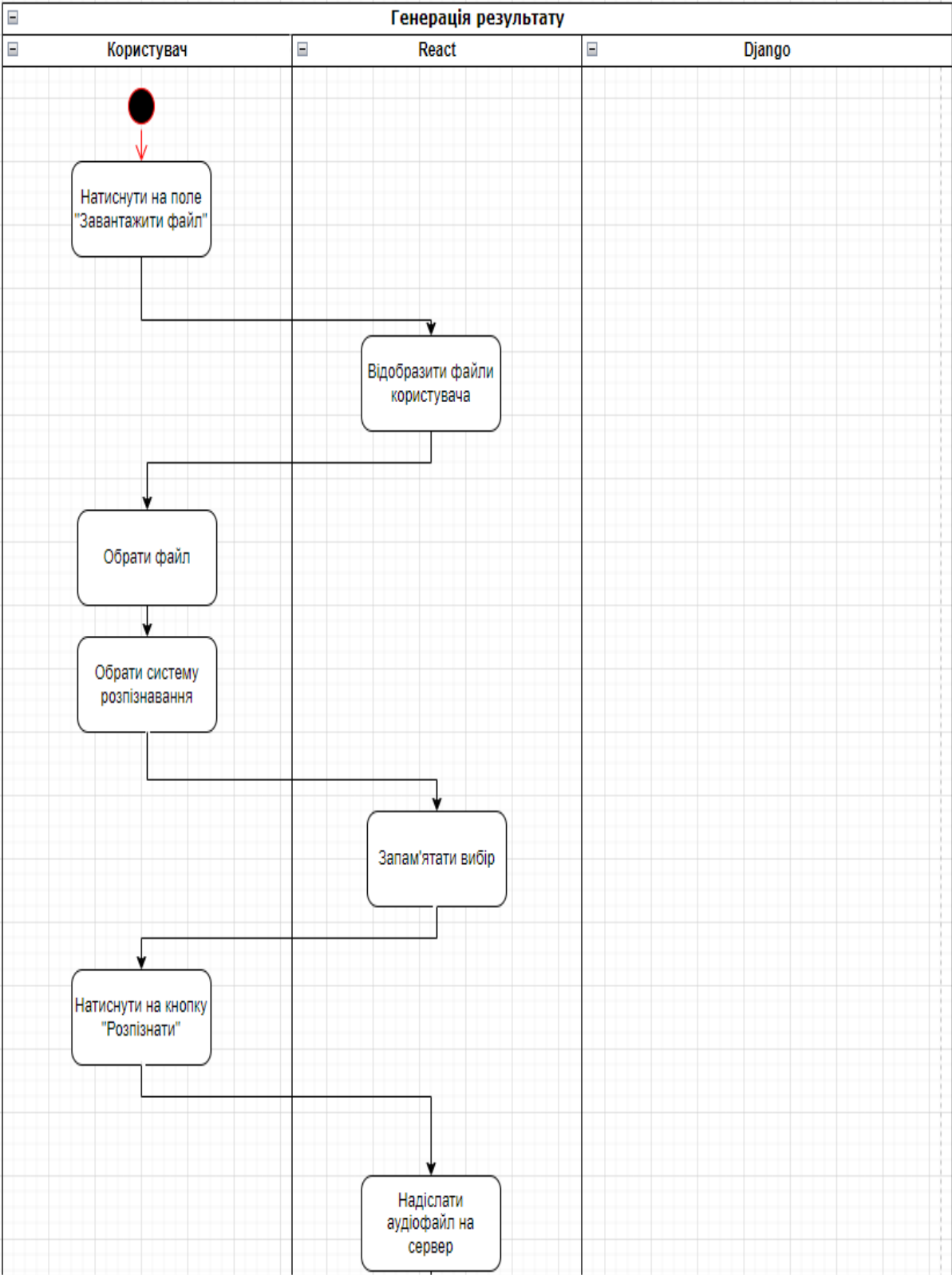


Рисунок 3.2 – Діаграма діяльності

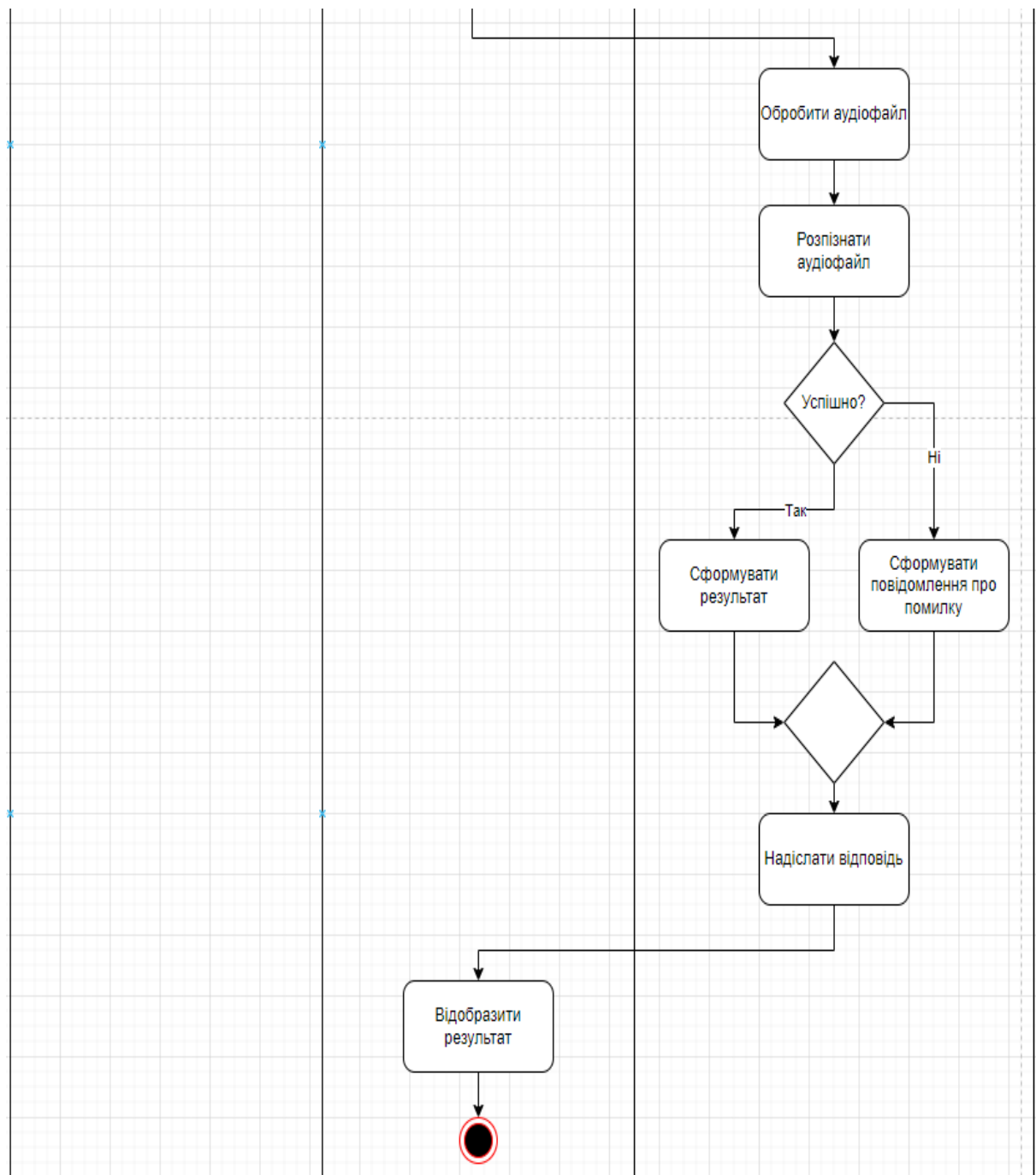


Рисунок 3.2, аркуш 2

Процес починається з дії користувача - натискання на поле завантаження файлу. React-компонент відображає файлову систему користувача для вибору файлу. Після вибору файлу користувач обирає систему розпізнавання, що зберігається React-компонентом. При натисканні кнопки "Розпізнати" файл передається на сервер.

Серверна частина (Django) виконує послідовність дій з обробки та розпізнавання аудіофайлу. У разі успішного розпізнавання формується результат, в протилежному випадку - повідомлення про помилку. Незалежно від результату, відповідь надсилається назад до клієнтської частини, яка відображає її користувачу.

Використання доріжок (Користувач, React, Django) чітко демонструє розподіл відповідальності між різними частинами системи та їх взаємодію. Умовні переходи (ромби) показують обробку різних сценаріїв виконання, забезпечуючи коректну обробку як успішних, так і помилкових ситуацій.

Така діаграма допомагає зрозуміти послідовність дій, необхідних для генерації результату, та візуалізує взаємодію між різними компонентами системи в процесі обробки запиту користувача.

### **3.4 Діаграма послідовності**

Діаграма послідовності (Sequence Diagram) є інструментом UML для моделювання взаємодії об'єктів системи у хронологічній послідовності. Вона відображає обмін повідомленнями між компонентами системи, показуючи як відбувається взаємодія для досягнення певної функціональності [23].

Діаграма послідовності використовує спеціалізований набір елементів для відображення взаємодії об'єктів системи. Ключовим елементом є лінія життя об'єкта (Lifeline), яка представлена у вигляді вертикальної пунктирної лінії. Вона показує час існування об'єкта в рамках взаємодії та містить прямокутник з назвою об'єкта у верхній частині. Лінія життя відображає період активності об'єкта в системі.

Повідомлення між об'єктами зображуються за допомогою горизонтальних стрілок різних типів. Синхронні повідомлення, які вимагають негайної відповіді, позначаються суцільною стрілкою з заповненим наконечником. Асинхронні повідомлення, що не блокують виконання

відправника, використовують відкритий наконечник стрілки. Відповіді на повідомлення зазвичай показуються пунктирною стрілкою.

Активація об'єкта зображується вузькими прямокутниками на лінії життя, які ілюструють проміжок часу, протягом якого об'єкт виконує визначену операцію. Вкладені активації виникають, коли об'єкт викликає методи інших об'єктів під час своєї активності.

Фрейми взаємодії використовуються для групування послідовності повідомлень за певною логікою. Вони можуть показувати умовні конструкції (alt), цикли (loop), паралельне виконання (par) та інші специфічні випадки взаємодії. Кожен фрейм має свій оператор та умови виконання.

Створення та знищення об'єктів також може бути показано на діаграмі. Створення позначається спеціальним повідомленням до об'єкта, а знищення - символом X в кінці лінії життя, коли об'єкт перестає існувати.

Рівні (шари) на діаграмі можуть бути використані для групування об'єктів за їх логічною належністю до певних компонентів системи (наприклад, інтерфейс користувача, бізнес-логіка, доступ до даних), що покращує розуміння архітектури системи.

На наведеній на рисунку 3.3 діаграмі представлено процес генерації результату розпізнавання аудіо. Діаграма включає шість основних компонентів: Client (користувач), AudioUploader (React компонент), Router (маршрутизатор Django), AudioFileViewSet (контролер Django), AudioFile (модель Django) та DB (база даних).

Процес починається з дії користувача - натискання на поле завантаження файлу. AudioUploader компонент реагує відображенням файлів користувача. Після вибору файлу відбувається його встановлення через метод `setFile()`. Аналогічно, при виборі системи розпізнавання викликається метод `setRecognition()`.

Коли користувач натискає кнопку «Розпізнати», викликається метод `handleUpload()` в AudioUploader, який ініціює ланцюжок викликів через серверні

компоненти. Запит проходить через Router до AudioFileViewSet, де створюється новий запис в моделі AudioFile, який потім зберігається в базі даних.

Результат обробки (або повідомлення про помилку) повертається тим же шляхом назад до клієнта через ланцюжок компонентів. На клієнтській стороні результат встановлюється через метод setTranscript() для відображення користувачу.

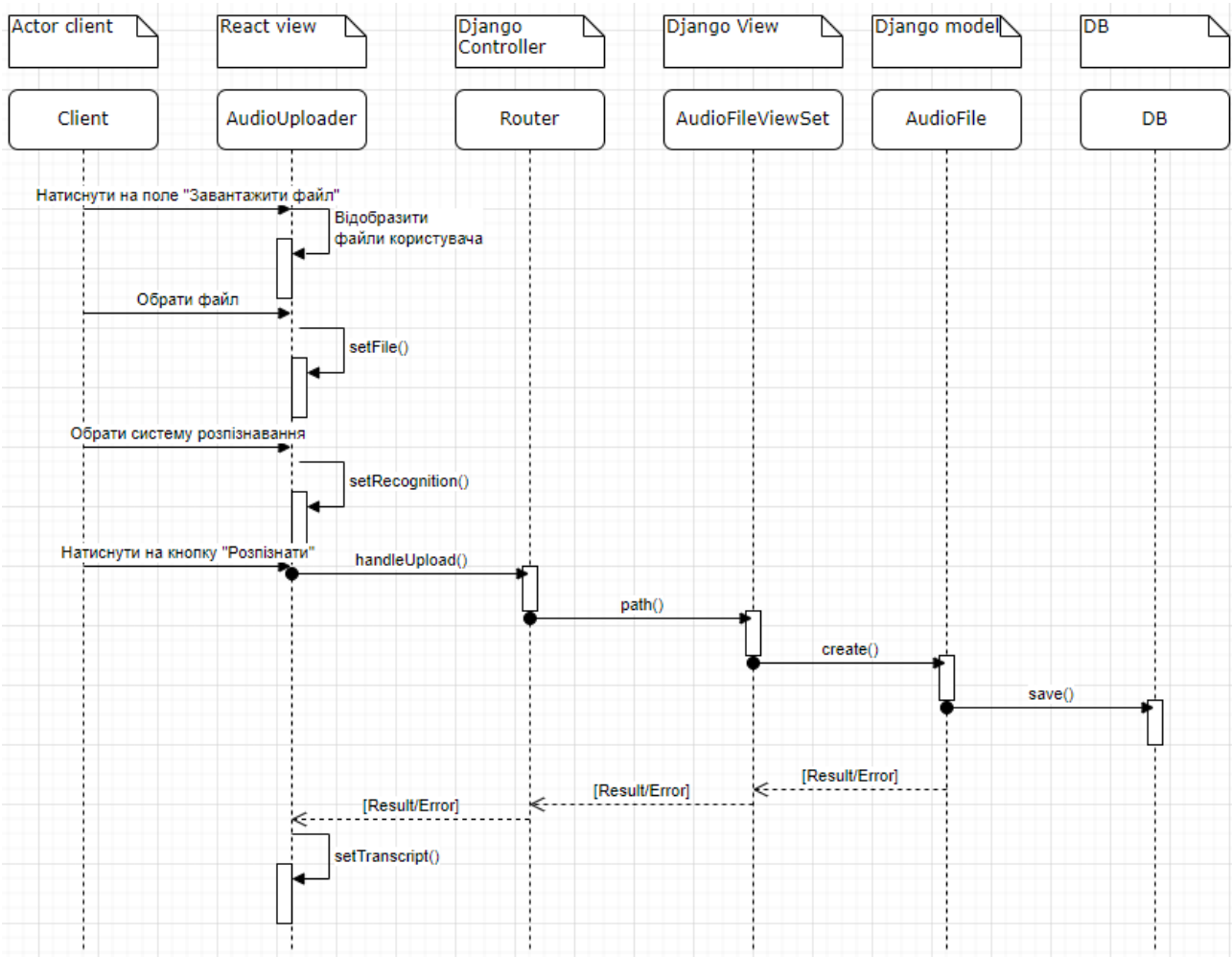


Рисунок 3.3 – Діаграма послідовності

Діаграма чітко показує взаємодію між клієнтською та серверною частинами застосунку, демонструючи як дані передаються між різними шарами системи та як обробляються результати запитів.

### 3.5 Діаграма класів

Діаграма класів (Class Diagram) є структурним елементом UML, що відображає статичну структуру системи через класи, їх атрибути, методи та взаємозв'язки між ними. Вона надає детальне представлення про архітектуру системи на рівні класів та їх взаємодії [23].

Діаграма класів використовує спеціальну нотацію для відображення різних аспектів об'єктно-орієнтованої системи. Прямокутник класу розділяється на три окремі секції, що містять назву класу, його атрибути та методи.

Верхня секція прямокутника призначена для назви класу. Тут можуть розміщуватися додаткові позначення, такі як стереотипи чи ключові слова. Це допомагає визначити специфічну роль або тип класу в системі.

Середня секція містить опис атрибутів класу. Кожен атрибут характеризується своєю видимістю. Після позначення видимості вказується назва атрибуту, його тип даних, а за необхідності - початкове значення.

Нижня секція відведена для методів класу. Опис кожного методу включає позначення видимості, назву методу, список параметрів та тип значення, що повертається. Параметри методів можуть мати власні типи даних та значення за замовчуванням.

Взаємозв'язки між класами відображаються різними типами ліній та стрілок, кожен з яких має своє значення. Важливим аспектом діаграми є позначення множинності зв'язків між класами. Ці позначення розміщуються біля кінців ліній зв'язку та вказують, скільки об'єктів одного класу може бути пов'язано з об'єктом іншого класу.

На представленій на рисунку 3.4 діаграмі відображено основні класи системи розпізнавання аудіо, що реалізують серверну частину застосунку на базі Django. Центральним елементом є клас `AudioFileViewSet`, який успадковує функціональність від базового класу `ViewSet` фреймворку Django REST Framework. Цей клас відповідає за обробку HTTP-запитів та керування аудіофайлами.

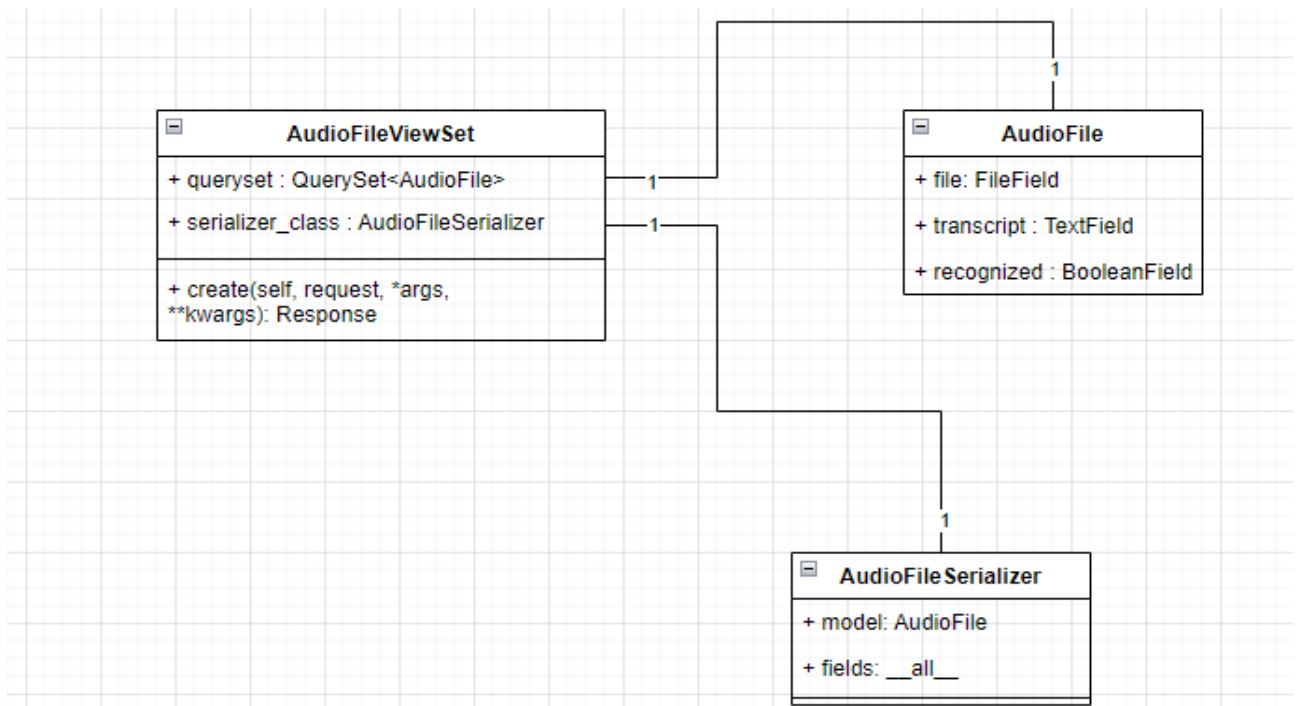


Рисунок 3.4 – Діаграма класів

**AudioFileViewSet** має наступні ключові атрибути:

- `queryset` типу `QuerySet<AudioFile>` для роботи з колекцією аудіофайлів;
- `serializer_class`, що вказує на `AudioFileSerializer` для серіалізації даних;
- метод `create` для обробки запитів на створення нових записів.

Клас **AudioFile** представляє модель даних та містить поля:

- `file` типу `FileField` для збереження аудіофайлу;
- `transcript` типу `TextField` для зберігання результату розпізнавання;
- `recognized` типу `BooleanField` для відстеження статусу розпізнавання.

**AudioFileSerializer** забезпечує конвертацію між об'єктами **AudioFile** та JSON-представленням, маючи:

- посилання на модель **AudioFile**;
- налаштування полів для серіалізації через атрибут `fields`.

Між класами встановлені зв'язки типу «один до одного» (1:1), що відображають їх взаємодію в системі. **AudioFileViewSet** пов'язаний з **AudioFile** через `queryset` та з **AudioFileSerializer** через `serializer_class`, а **AudioFileSerializer** безпосередньо пов'язаний з моделлю **AudioFile**.

Така структура забезпечує чітке розмежування функціональних обов'язків між компонентами системи: модель відповідає за управління даними, а серіалізатор – за їхнє перетворення, а `viewset` – за обробку запитів та бізнес-логіку.

### 3.6 Діаграма розгортання

Діаграма розгортання (Deployment Diagram) є інструментом UML для моделювання фізичної архітектури системи. Вона показує розташування та взаємозв'язки між апаратними та програмними компонентами системи в реальному середовищі функціонування [23].

Основним структурним елементом діаграми розгортання є вузол (node), який представляє фізичні обчислювальні ресурси, такі як сервери, робочі станції або мобільні пристрої. Всередині вузла можуть бути вказані характеристики середовища виконання, операційної системи та встановленого програмного забезпечення.

Артефакти є іншим елементом діаграми, що представляють конкретні програмні компоненти, які розгортаються на вузлах. Вони зображуються у вигляді прямокутників зі спеціальним значком у правому верхньому куті. Артефакти можуть бути виконуваними файлами, бібліотеками, базами даних або іншими програмними ресурсами.

З'єднання між вузлами відображаються за допомогою ліній, які показують комунікаційні шляхи. На цих лініях можуть бути вказані протоколи зв'язку (наприклад, HTTP, HTTPS) та інші характеристики взаємодії. Стереотипи використовуються для уточнення природи з'єднань та вузлів.

Середовище виконання (execution environment) може бути показано як вкладений елемент всередині вузла. Воно представляє програмну інфраструктуру, необхідну для виконання артефактів, наприклад, веб-сервер або контейнер сервлетів.

Інтерфейси на діаграмі розгортання показують точки взаємодії між різними компонентами системи. Вони можуть бути представлені як надані або необхідні інтерфейси, що дозволяє чітко визначити залежності між компонентами системи.

Специфікації розгортання можуть включати додаткові властивості та параметри, які вказуються всередині вузлів або поруч з артефактами. Це можуть бути версії програмного забезпечення, системні вимоги або конфігураційні параметри.

На представлений на рисунку 3.5 діаграмі зображено розгортання системи розпізнавання аудіо сигналів. Центральним елементом є Web Server, який функціонує під керуванням операційної системи Linux Ubuntu версії 18 або вище. На сервері розгорнуто необхідне програмне забезпечення:

- Python версії 3.7 або вище для виконання серверної частини;
- веб-сервер Apache2 або nginx для обробки HTTP-запитів;
- реляційна база даних SQLite3 для збереження інформації.

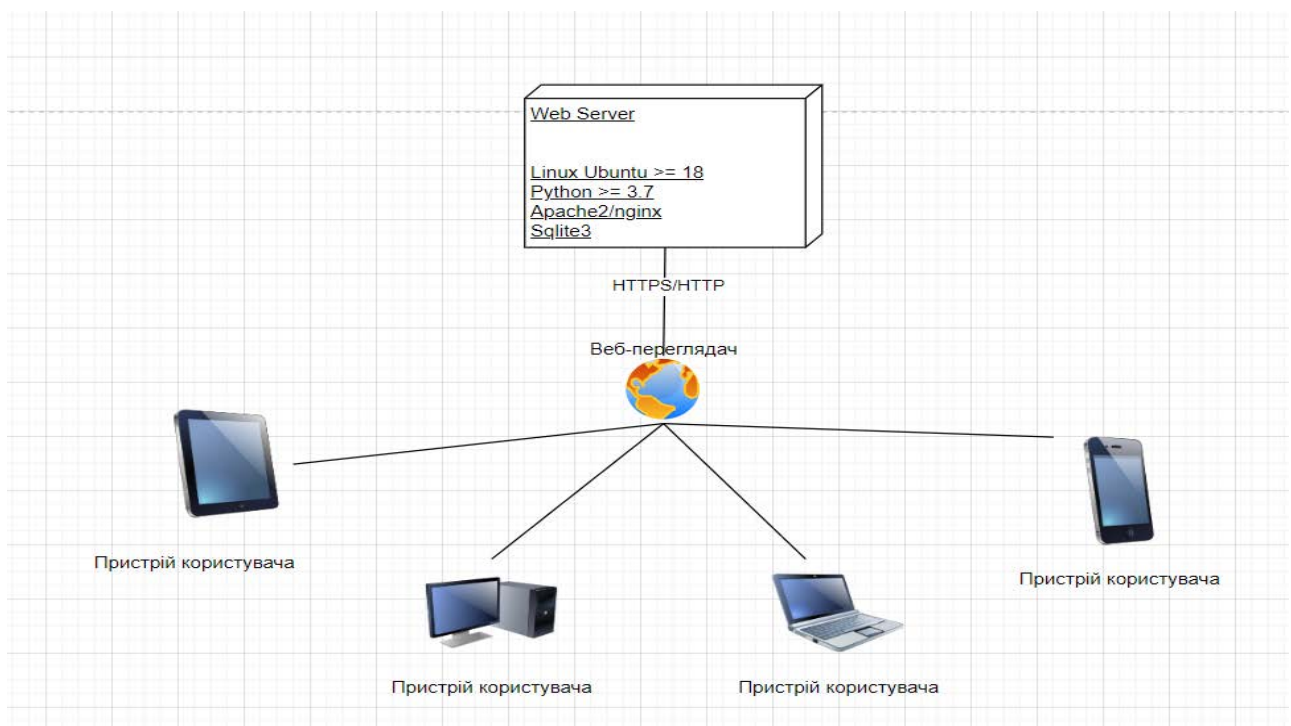


Рисунок 3.5 – Діаграма розгортання

Сервер взаємодіє з клієнтськими пристроями через веб-браузер за протоколами HTTPS/HTTP. На діаграмі показано різні типи клієнтських пристроїв, що можуть підключатися до системи: планшети, смартфони, настільні комп'ютери та ноутбуки. Це підкреслює кросплатформенність та доступність системи з різних типів пристроїв.

Веб-браузер виступає в ролі клієнтського середовища виконання, де працює React-застосунок. Він забезпечує єдиний інтерфейс для всіх типів пристроїв, адаптуючись під різні розміри екранів та особливості взаємодії.

Така архітектура розгортання забезпечує:

- централізоване зберігання та обробку даних на сервері;
- легкий доступ з різних пристроїв через веб-браузер;
- масштабованість системи;
- можливість оновлення серверної частини без змін на клієнтських пристроях.

### **3.7 Висновки за розділом 3**

Виділено основні варіанти використання, такі як керування аудіо, керування системами розпізнавання та керування результатом, що дозволило структурувати вимоги до системи та визначити необхідний функціонал.

Визначено детальну послідовність дій при обробці аудіофайлів, включаючи розподіл відповідальності між клієнтською (React) та серверною (Django) частинами системи.

Встановлено часові аспекти взаємодії між компонентами системи та механізми обміну повідомленнями.

Встановлено структуру класів системи, яка відображає взаємозв'язки між основними компонентами: `AudioFileViewSet`, `AudioFile` та `AudioFileSerializer`.

Визначено фізичну архітектуру системи, включаючи необхідне програмне забезпечення та взаємодію між серверною частиною та клієнтськими пристроями.

Спроектowana архітектура забезпечує гнучкість, масштабованість та доступність до системи для різних категорій користувачів, незалежно від їх технічного оснащення.

## 4 ТЕСТУВАННЯ РЕАЛІЗАЦІЇ ТА РЕЗУЛЬТАТИ

### 4.1 Вибір та обґрунтування засобів реалізації

Для реалізації системи розпізнавання аудіо сигналів було обрано сучасний стек технологій, що включає Django для серверної частини, React.js використовується для реалізації клієнтської частини системи та SQLite для управління даними. Такий вибір зумовлений специфікою задачі та вимогами до системи [16].

Django, як фреймворк високого рівня, забезпечує стабільну та ефективну основу для роботи серверної частини системи. Його архітектура MTV (Model-Template-View) дозволяє ефективно організувати бізнес-логіку та управління даними. Інтегрована система ORM значно полегшує взаємодію з базою даних, тоді як Django REST Framework забезпечує ефективні засоби для розробки API. Особливо важливим є наявність у Django вбудованих механізмів безпеки та можливостей для оптимізації продуктивності, що критично для системи, яка працює з файлами [15].

React.js обрано для розробки користувацького інтерфейсу завдяки його компонентному підходу та ефективному управлінню станом застосунку. Віртуальний DOM забезпечує високу продуктивність при оновленні інтерфейсу, що важливо при роботі з динамічними даними. Використання JSX робить код більш читабельним та спрощує розробку, а багата екосистема бібліотек дозволяє швидко впроваджувати необхідний функціонал [20].

SQLite як система управління базами даних ідеально підходить для даного проєкту через свою легкість, відсутність необхідності в окремому сервері та надійність при роботі з локальними даними. Важливою перевагою є вбудована підтримка SQLite в Django, що спрощує розгортання системи. База даних забезпечує атомарність транзакцій та ефективне зберігання метаданих аудіофайлів та результатів розпізнавання [22].

Взаємодія між компонентами системи організована через REST API, що забезпечує чітке розділення відповідальності та можливість незалежного

масштабування частин системи. Клієнтська частина комунікує з сервером через HTTP-запити, отримуючи дані у форматі JSON. Це дозволяє досягти високої гнучкості та можливості подальшого розширення функціоналу [20].

Така архітектура дозволяє ефективно вирішувати основні завдання системи: завантаження та обробку аудіофайлів, їх розпізнавання та зберігання результатів. При цьому забезпечується висока продуктивність, надійність та зручність використання для кінцевого користувача.

Для обробки та аналізу аудіосигналів обрано бібліотеку Librosa, яка надає широкий спектр функцій для роботи з аудіоданими. Вибір Librosa обґрунтований її можливостями для:

- завантаження та попередньої обробки аудіофайлів;
- екстракції часових та частотних характеристик сигналів;
- обчислення мел-частотних кепстральних коефіцієнтів (MFCC);
- візуалізації спектрограм та інших характеристик сигналу.

Для реалізації методів машинного навчання обрано фреймворк TensorFlow у поєднанні з Keras. Такий вибір зумовлений наступними перевагами:

- ефективна реалізація різних архітектур нейронних мереж;
- зручний високорівневий API через Keras;
- потужні засоби оптимізації та навчання моделей;
- можливість візуалізації процесу навчання.

Для візуалізації результатів аналізу використовуються бібліотеки Matplotlib та Seaborn, які забезпечують:

- створення інформативних графіків та діаграм;
- візуалізацію характеристик аудіосигналів;
- представлення результатів класифікації.

В системі реалізовано комбінований підхід до розпізнавання аудіо, що використовує переваги різних архітектур нейронних мереж:

- згорткові нейронні мережі (CNN) для обробки спектрограм;
- рекурентні нейронні мережі (RNN) для аналізу часової динаміки.

Така комбінація дозволяє досягти високої точності розпізнавання при роботі з часово-частотними характеристиками аудіосигналів.

## 4.2 Розробка клієнтської частини

Клієнтська частина системи розпізнавання аудіо сигналів реалізована з використанням бібліотеки React.js.

React.js використовується для створення сучасних користувацьких інтерфейсів завдяки своєму компонентному підходу. В основі лежить концепція компонентів – незалежних та повторно використовуваних блоків інтерфейсу, кожен з яких має власну логіку та відображення [21].

Розглянемо основні характеристики React-компонентів.

Внутрішній стан компонента (State) – це об'єкт, що зберігає дані, необхідні для роботи компонента. При зміні стану відбувається автоматичне оновлення інтерфейсу.

Властивості компонента (Props) – параметри, що передаються від батьківського компонента до дочірнього, забезпечуючи гнучку конфігурацію.

Життєвий цикл компонента – набір методів, що викликаються на різних етапах існування компонента (створення, оновлення, видалення).

JSX синтаксис, що дозволяє писати HTML-подібну розмітку безпосередньо в JavaScript коді.

На рисунку 4.1 представлено базову структуру React-компонента, що демонструє основні елементи його побудови. Лістинг коду компоненту розробленої системи наведено в Додатку Б.

Компонент починається з імпорту необхідних залежностей з бібліотеки React. Далі йде оголошення класу, що наслідується від базового компонента React. У конструкторі відбувається ініціалізація стану та обробка властивостей. Метод render() визначає, що саме буде відображено на сторінці, використовуючи JSX синтаксис.

```

import React, { Component } from 'react';

class MyComponent extends Component {
  constructor(props) {
    super(props);
    this.state = {
      // стан компонента
    };
  }

  componentDidMount() {
    // логіка, яка виконується після монтування компонента
  }

  render() {
    return (
      <div>
        { /* JSX, що відображається компонентом */ }
        <h1>{this.props.title}</h1>
        <p>{this.state.someData}</p>
      </div>
    );
  }
}

export default MyComponent;

```

Рисунок 4.1 – Приклад компоненту

В контексті системи розпізнавання аудіо сигналів було розроблено ряд спеціалізованих компонентів:

- компонент завантаження аудіофайлів;
- компонент вибору методу розпізнавання;
- компонент відображення результатів.

### 4.3 Розробка серверної частини

Серверна частина системи розпізнавання аудіо реалізована на базі фреймворку Django, використовуючи архітектурний патерн MTV (Model-Template-View). Для забезпечення ефективної взаємодії з клієнтською частиною

реалізовано REST API за допомогою Django REST Framework [20].

Django-модель виступає основним компонентом для управління даними. На рисунку 4.2 представлено реалізацію моделі `AudioFile`, яка відповідає за зберігання інформації про аудіофайли та результати їх розпізнавання. Лістинг коду моделей розробленої системи наведено в Додатку Б.

```
class AudioFile(models.Model):  
    file = models.FileField(upload_to='audio/')  
    transcript = models.TextField(blank=True)  
    recognized = models.BooleanField(default=False)
```

Рисунок 4.2 – Модель `AudioFile`

Модель `AudioFile` наслідується від базового класу `models.Model` та містить три основних поля:

- `file`: поле типу `FileField` для зберігання аудіофайлів, які завантажуються у директорію `'audio/'`;
- `transcript`: текстове поле для збереження результату розпізнавання з можливістю залишатися пустим;
- `recognized`: булеве поле зі значенням за замовчуванням `False`, яке відмічає статус розпізнавання файлу.

Для обробки запитів від клієнтської частини розроблено представлення (View) на базі `ViewSet`, код якого показано на рисунку 4.3. Лістинг коду представлень розробленої системи наведено в Додатку Б.

`AudioFileViewSet` наслідується від `viewsets.ModelViewSet` та реалізує наступну функціональність:

- отримання всіх аудіо файлів через `queryset`;
- серіалізацію даних за допомогою `AudioFileSerializer`;
- конвертацію аудіо файлів у формат WAV через метод `convert_to_wav`;
- створення нових записів через метод `create`, який:
- отримує файл з запиту;
- перевіряє наявність файлу;

- визначає метод розпізнавання;
- конвертує файл у WAV формат;
- виконує розпізнавання;
- зберігає результат.

```
class AudioFileViewSet(viewsets.ModelViewSet):
    queryset = AudioFile.objects.all()
    serializer_class = AudioFileSerializer

    def convert_to_wav(self, file_path):
        audio = AudioSegment.from_file(file_path)
        return audio.export(format="wav")

    def create(self, request, *args, **kwargs):
        file = request.data.get('file', None)
        recognizer = None

        if file is None:
            return Response({'file': 'This field is required.'}, status=status.HTTP_400_BAD_REQUEST)

        recognition_method = request.data.get('recognition_method', 'google') # Метод розпізнавання

        try:
            # Конвертація аудіофайлу у формат WAV
            wav_file = self.convert_to_wav(file)
        except Exception as e:
            return Response({'error': f'Error converting audio to WAV: {e}'}, status=status.HTTP_500_INTERNAL_SERVER_ERROR)
```

Рисунок 4.3 – Представлення AudioFileViewSet

Особлива увага приділена обробці помилок. У випадку відсутності файлу повертається HTTP 400 BAD REQUEST, при помилках конвертації - HTTP 500 INTERNAL SERVER ERROR. Це забезпечує надійне інформування клієнтської частини про статус обробки запиту.

Така архітектура забезпечує чітке розділення відповідальності: модель відповідає за структуру даних, представлення відповідає за обробку запитів і реалізацію бізнес-логіки, а серіалізатор - за перетворення даних між форматами Python та JSON.

#### 4.4 Основні класи та функції системи

Центральним елементом розробленої системи є функціональність генерації результату розпізнавання аудіо. Для реалізації цього процесу розроблено два ключових класи, які забезпечують повний цикл обробки аудіофайлів.

Клас `AudioUploader` є сполучною ланкою між користувацьким інтерфейсом та серверним API. Він реалізує наступні функції:

- взаємодію з Django REST API через HTTP-запити;
- управління процесом завантаження файлів;
- відображення результатів розпізнавання в інтерфейсі;
- обробку помилок та відображення повідомлень користувачу.

`AudioFileViewSet` виступає основним обробником запитів на серверній стороні та забезпечує:

- прийом та валідацію вхідних файлів;
- визначення та застосування обраного методу розпізнавання;
- конвертацію аудіо файлів у необхідний формат;
- збереження результатів у базу даних;
- підготовку відповіді для клієнтської частини.

Взаємодія між цими класами відбувається за допомогою REST API, де `AudioUploader` формує та надсилає запити, а `AudioFileViewSet` їх обробляє та повертає результати. Така архітектура забезпечує чітке розділення відповідальності та гнучкість у розширенні функціоналу системи.

Додатково класи підтримуються допоміжними компонентами:

- серіалізаторами для перетворення даних;
- системами валідації вхідних даних;
- механізмами обробки помилок;
- утилітами для взаємодії з файлами.

Така структура дозволяє ефективно реалізувати основний бізнес-процес системи, забезпечуючи надійність роботи та зручність подальшої підтримки.

#### **4.5 Результати навчання та оцінка якості моделі**

Навчання моделі розпізнавання аудіосигналів проводилося з використанням бібліотек TensorFlow і Keras на основі даних набору ESC-50, який містить 50 класів звуків (рис. 4.4).

На рисунку 4.5 наведено розподіл аудіофайлів за категоріями.

Попередній перегляд даних вибірки:

|      | filename          | fold | target | category       | esc10 | src_file | take |
|------|-------------------|------|--------|----------------|-------|----------|------|
| 1995 | 5-263831-B-6.wav  | 5    | 6      | hen            | False | 263831   | B    |
| 1996 | 5-263902-A-36.wav | 5    | 36     | vacuum_cleaner | False | 263902   | A    |
| 1997 | 5-51149-A-25.wav  | 5    | 25     | footsteps      | False | 51149    | A    |
| 1998 | 5-61635-A-8.wav   | 5    | 8      | sheep          | False | 61635    | A    |
| 1999 | 5-9032-A-0.wav    | 5    | 0      | dog            | True  | 9032     | A    |

Рисунок 4.4 - Фрагмент структури набору даних ESC-50

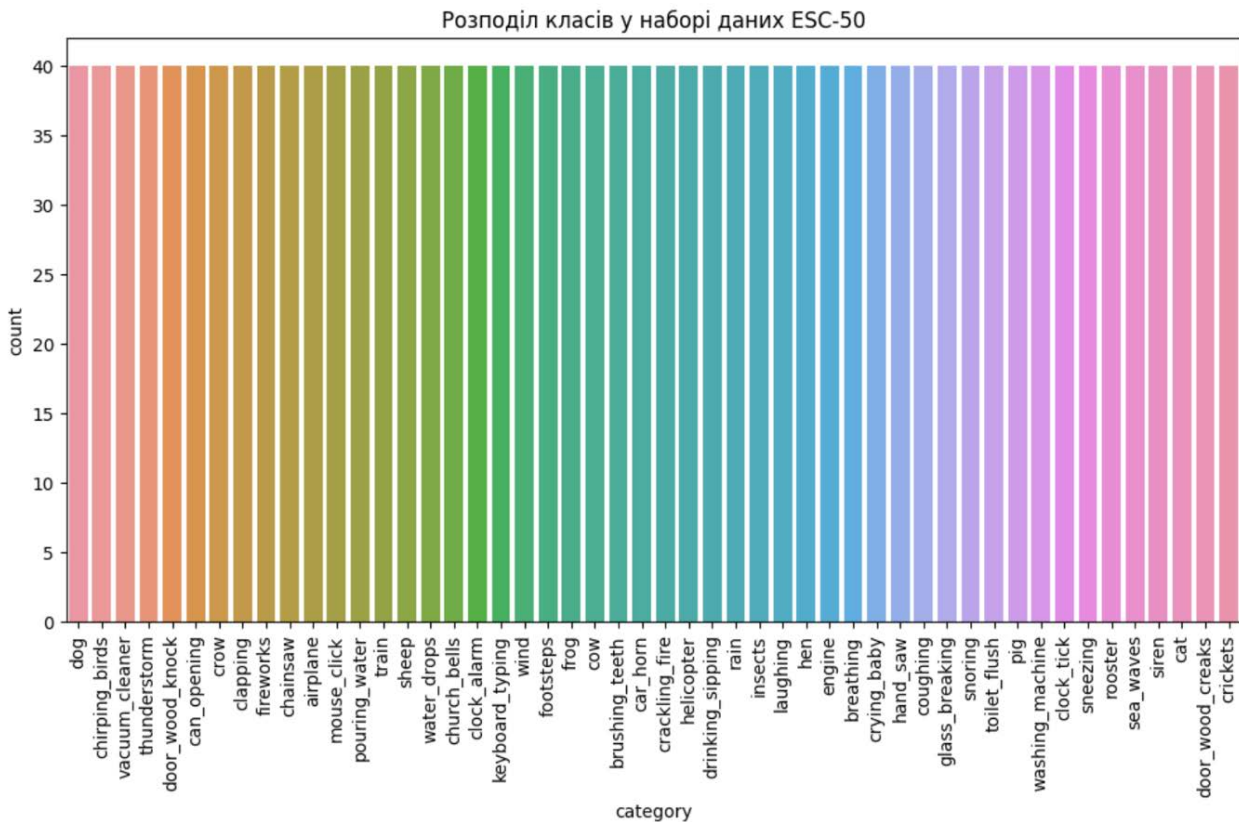


Рисунок 4.5 - Розподіл аудіофайлів за категоріями у наборі даних ESC-50

На рисунку 4.6 наведена матриця помилок, яка демонструє кількість правильних і неправильних класифікацій для кожного класу звуків у наборі даних. По діагоналі матриці розташовані значення правильних класифікацій, які показують, скільки разів модель коректно розпізнала кожен клас. Інші значення в рядках і стовпцях відповідають помилковим класифікаціям, коли звуки одного класу були віднесені до іншого.

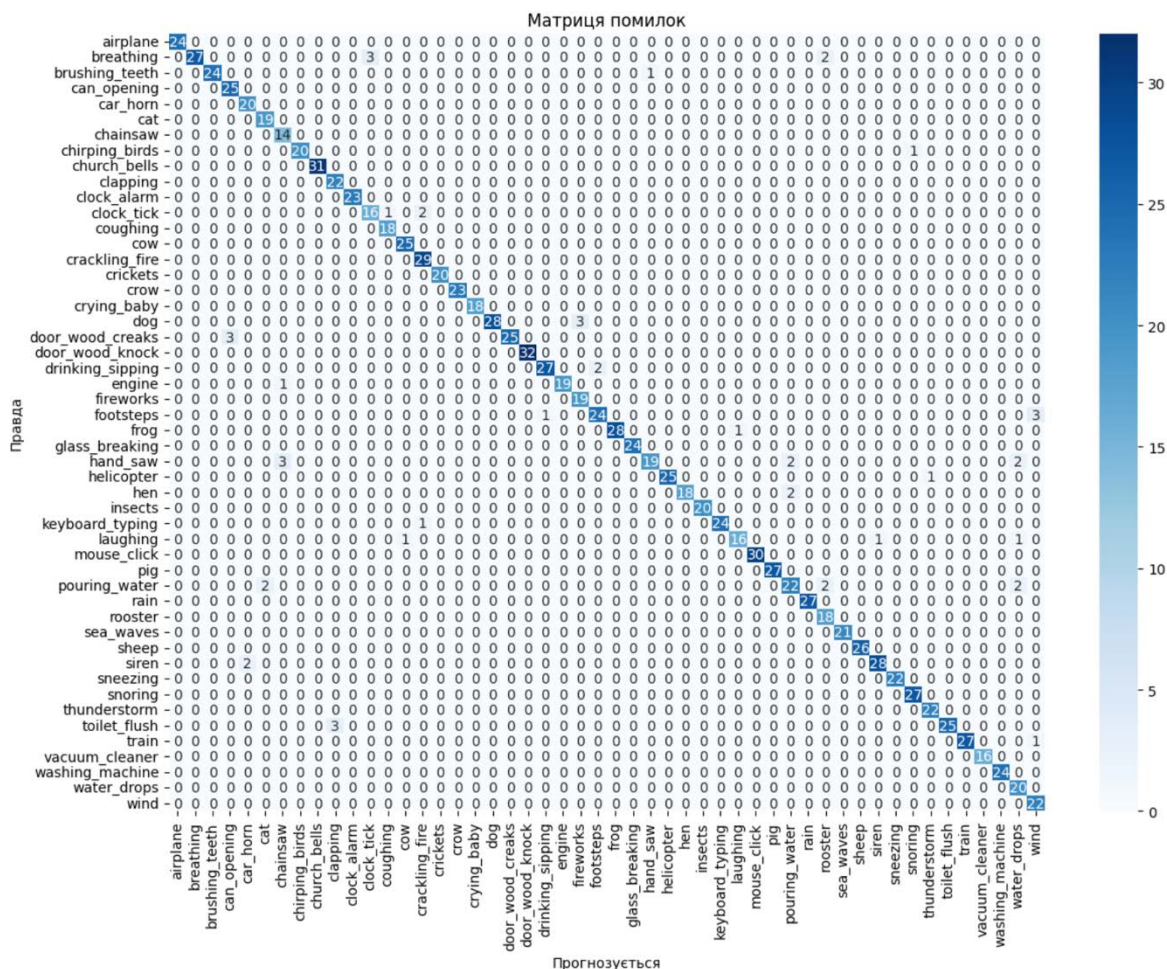


Рисунок 4.6 - Матриця помилок класифікації аудіосигналів набору ESC-50

З аналізу матриці можна зробити висновок, що модель демонструє високу точність у класифікації багатьох класів, що підтверджується високими значеннями на діагоналі.

Ці результати підкреслюють важливість подальшої оптимізації моделі, зокрема через збільшення обсягу навчальних даних для складних класів або використання додаткових ознак, які краще описують акустичні характеристики.

Основним завданням було забезпечити високу точність класифікації звуків різних класів. На основі класифікаційного звіту (рис. 4.7) проведено детальний аналіз ефективності розробленої моделі для задачі розпізнавання звуків із набору. Основними метриками для оцінки якості моделі стали точність (precision), повнота (recall) та F1-міра.

Загальна точність моделі становить 96%, що підтверджує її високу здатність до класифікації аудіоданих. Середні значення F1-міри, точності та повноти для всіх класів дорівнюють 0.96, що свідчить про збалансованість моделі для різних категорій звуків. Модель демонструє високі показники точності (1.00) для більшості класів. Ці результати вказують на надійність системи в ідентифікації цих звуків. Однак, для деяких класів спостерігається зниження точності, наприклад, для chainsaw (0.78) та pouring\_water (0.85). Це може бути пов'язано зі схожістю цих звуків із іншими класами або недостатньою кількістю навчальних даних.

| Класифікаційний звіт: |           |        |          |         |
|-----------------------|-----------|--------|----------|---------|
|                       | precision | recall | f1-score | support |
| airplane              | 1.00      | 1.00   | 1.00     | 24      |
| breathing             | 1.00      | 0.84   | 0.92     | 32      |
| brushing_teeth        | 1.00      | 0.96   | 0.98     | 25      |
| can_opening           | 0.89      | 1.00   | 0.94     | 25      |
| car_horn              | 0.91      | 1.00   | 0.95     | 20      |
| cat                   | 0.90      | 1.00   | 0.95     | 19      |
| chainsaw              | 0.78      | 1.00   | 0.88     | 14      |
| chirping_birds        | 1.00      | 0.95   | 0.98     | 21      |
| church_bells          | 1.00      | 1.00   | 1.00     | 31      |
| clapping              | 0.88      | 1.00   | 0.94     | 22      |
| clock_alarm           | 1.00      | 1.00   | 1.00     | 23      |
| clock_tick            | 0.84      | 0.84   | 0.84     | 19      |
| coughing              | 0.95      | 1.00   | 0.97     | 18      |
| cow                   | 0.96      | 1.00   | 0.98     | 25      |
| crackling_fire        | 0.91      | 1.00   | 0.95     | 29      |
| crickets              | 1.00      | 1.00   | 1.00     | 20      |
| crow                  | 1.00      | 1.00   | 1.00     | 23      |
| crying_baby           | 1.00      | 1.00   | 1.00     | 18      |
| dog                   | 1.00      | 0.90   | 0.95     | 31      |
| door_wood_creaks      | 1.00      | 0.89   | 0.94     | 28      |
| door_wood_knock       | 1.00      | 1.00   | 1.00     | 32      |
| drinking_sipping      | 0.96      | 0.93   | 0.95     | 29      |
| engine                | 1.00      | 0.95   | 0.97     | 20      |
| fireworks             | 0.86      | 1.00   | 0.93     | 19      |
| footsteps             | 0.92      | 0.86   | 0.89     | 28      |
| frog                  | 1.00      | 0.97   | 0.98     | 29      |
| glass_breaking        | 1.00      | 1.00   | 1.00     | 24      |
| hand_saw              | 0.95      | 0.73   | 0.83     | 26      |
| helicopter            | 1.00      | 0.96   | 0.98     | 26      |
| hen                   | 1.00      | 0.90   | 0.95     | 20      |

Рисунок 4.7 - Метрики якості розпізнавання для окремих класів аудіосигналів

|                 |      |      |      |      |
|-----------------|------|------|------|------|
| insects         | 1.00 | 1.00 | 1.00 | 20   |
| keyboard_typing | 1.00 | 0.96 | 0.98 | 25   |
| laughing        | 0.94 | 0.84 | 0.89 | 19   |
| mouse_click     | 1.00 | 1.00 | 1.00 | 30   |
| pig             | 1.00 | 1.00 | 1.00 | 27   |
| pouring_water   | 0.85 | 0.79 | 0.81 | 28   |
| rain            | 1.00 | 1.00 | 1.00 | 27   |
| rooster         | 0.82 | 1.00 | 0.90 | 18   |
| sea_waves       | 1.00 | 1.00 | 1.00 | 21   |
| sheep           | 1.00 | 1.00 | 1.00 | 26   |
| siren           | 0.97 | 0.93 | 0.95 | 30   |
| sneezing        | 1.00 | 1.00 | 1.00 | 22   |
| snoring         | 0.96 | 1.00 | 0.98 | 27   |
| thunderstorm    | 0.96 | 1.00 | 0.98 | 22   |
| toilet_flush    | 1.00 | 0.89 | 0.94 | 28   |
| train           | 1.00 | 0.96 | 0.98 | 28   |
| vacuum_cleaner  | 1.00 | 1.00 | 1.00 | 16   |
| washing_machine | 1.00 | 1.00 | 1.00 | 24   |
| water_drops     | 0.80 | 1.00 | 0.89 | 20   |
| wind            | 0.85 | 1.00 | 0.92 | 22   |
| accuracy        |      |      | 0.96 | 1200 |
| macro avg       | 0.96 | 0.96 | 0.96 | 1200 |
| weighted avg    | 0.96 | 0.96 | 0.96 | 1200 |

Рисунок 4.7, аркуш 2

Повнота моделі також залишається високою для більшості класів, однак клас pouring\_water має значення 0.79, а laughing 0.84, що свідчить про часткові пропуски при ідентифікації цих звуків. Найвищі значення повноти (1.00) досягнуто для більшості класів, таких як dog, siren, train, glass\_breaking, що вказує на повне охоплення відповідних зразків.

F1-міра, яка є гармонійним середнім між точністю та повнотою, виявила найнижчі значення для класу chainsaw (0.83), що вказує на необхідність подальшого вдосконалення алгоритмів для цього класу.

Аналіз результатів показав, що розроблена модель є ефективною для класифікації звуків і демонструє високі показники якості для більшості класів. Водночас окремі класи, потребують оптимізації.

На рисунку 4.8 відображено зміну точності моделі (accuracy) в процесі навчання та валідації протягом 50 епох. З графіка видно, що модель швидко навчається на початкових епохах, досягаючи високих значень точності (понад

90%) вже після перших 10 епох. Після цього спостерігається поступова стабілізація, і до кінця навчання тренувальна точність досягає  $\approx 99\%$ , тоді як валідаційна точність  $\approx 96\%$ . Це вказує на високу здатність моделі до узагальнення та мінімальні ознаки перенавчання.

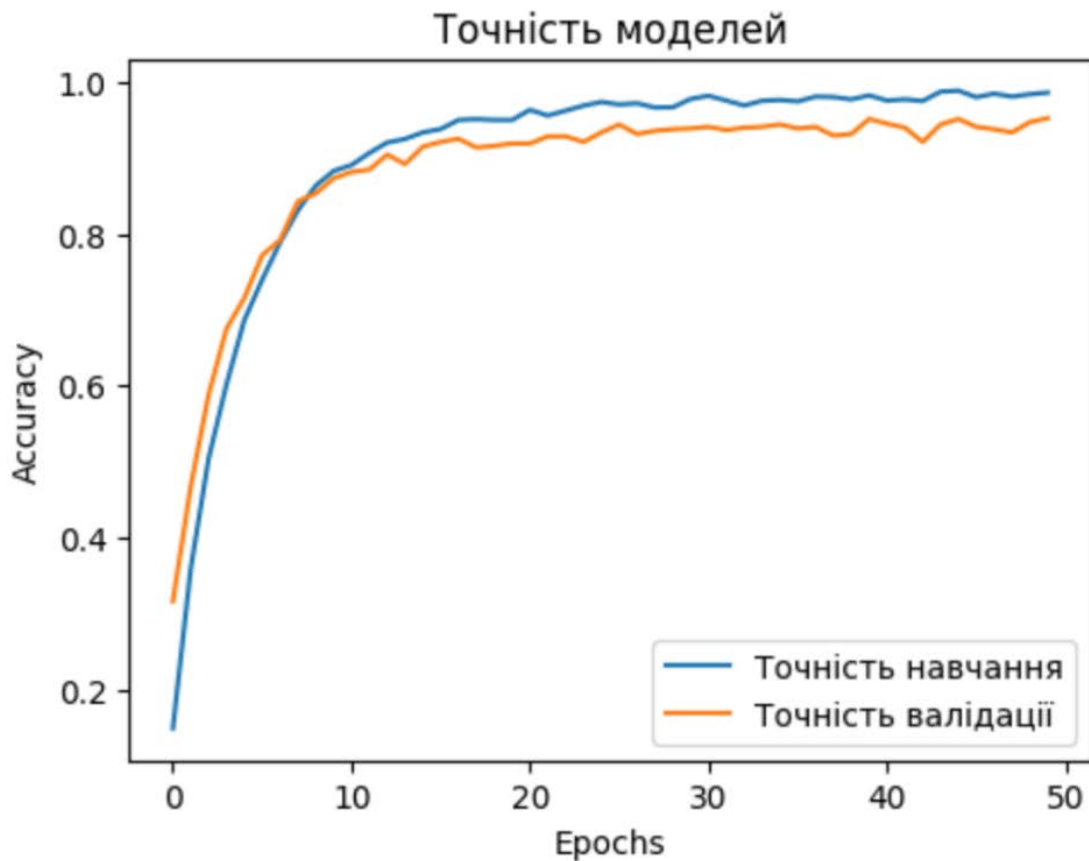


Рисунок 4.8 - Динаміка точності моделі на навчальній та валідаційній вибірках

На рисунку 4.9 представлено зміну втрат (loss) для тренувальної та валідаційної вибірок. Спостерігається різке зниження втрат на початкових епохах, що свідчить про ефективне навчання моделі. Тренувальні втрати досягають мінімальних значень ( $\approx 0.1$ ) до кінця навчання. Втрати на валідаційній вибірці також знижуються, проте стабілізуються на рівні  $\approx 0.4$ . Це свідчить про узгодженість між навчанням і валідацією, а також про відсутність суттєвого перенавчання.

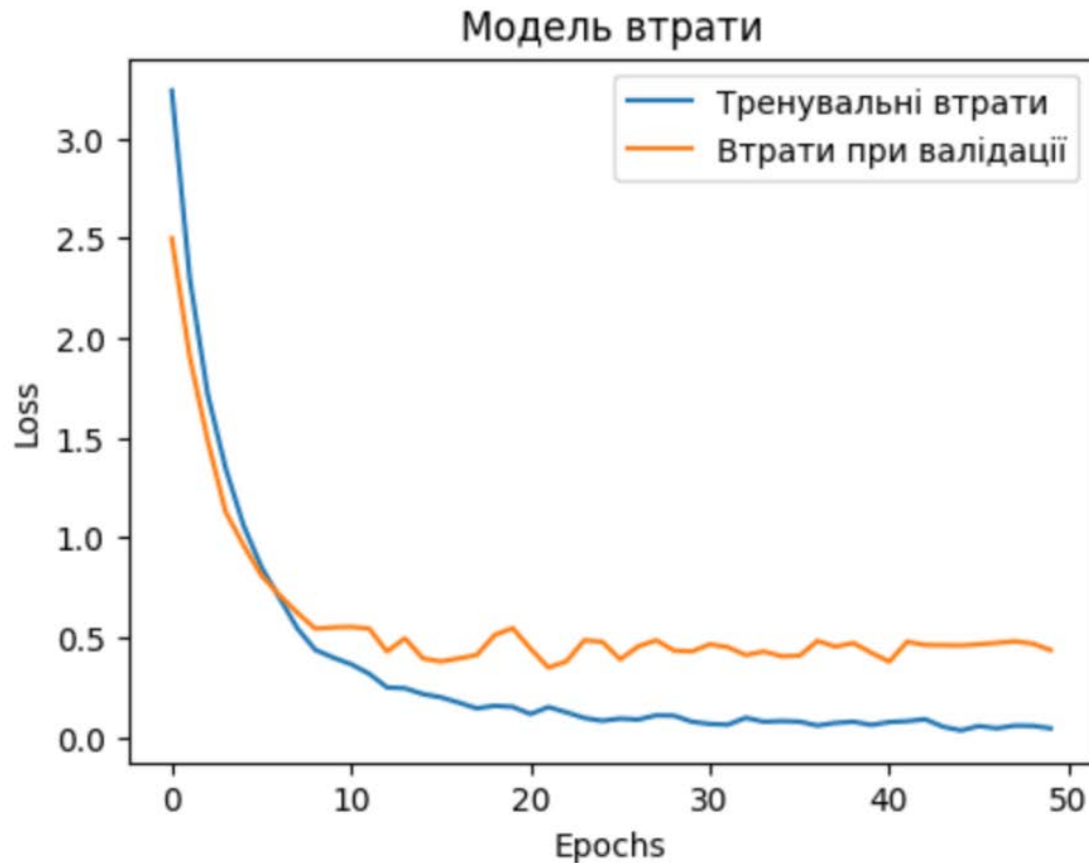


Рисунок 4.9 - Динаміка функції втрат на навчальній та валідаційній вибірках

Незважаючи на незначний розрив між тренувальними та валідаційними показниками, результати свідчать про гарну узагальнюючу здатність моделі. Подальше вдосконалення можна досягти шляхом:

- збільшення обсягу навчальних даних;
- застосування методів регуляризації для зменшення розриву між тренувальними та валідаційними втратами;
- тонкої оптимізації гіперпараметрів моделі.

Отримані результати демонструють готовність моделі до використання в задачах автоматичного розпізнавання звуків із високими вимогами до точності класифікації.

## 4.6 Тестування проєкту

Тестування є важливим етапом розробки системи розпізнавання аудіо. Для забезпечення якості та надійності роботи системи реалізовано два основні типи тестів: модульні (unit) та інтеграційні (integration).

Модульне тестування спрямоване на забезпечення правильності та надійності роботи окремих компонентів системи, ізольовано від інших частин. Для React-компонентів використовується бібліотека Jest разом з Testing Library, що дозволяє тестувати компоненти максимально наближено до того, як їх використовує реальний користувач. На рисунку 4.10 представлено приклад модульного тестування React-компонента AudioUploader.

```
const mockAxios = new MockAdapter(axios);

describe('AudioUploader Component', () => {
  beforeEach(() => {
    mockAxios.reset();
  });

  it('renders without crashing', () => {
    render(<AudioUploader />);
  });

  it('handles file change correctly', async () => {
    const { container } = render(<AudioUploader />);
    const fileInput = container.querySelector('input[type="file"]');
    const testFile = new File(['test content'], 'test.mp3', { type: 'audio/mp3' });

    fireEvent.change(fileInput, { target: { files: [testFile] } });

    await waitFor(() => {
      expect(screen.getByText('Файл: test.mp3')).toBeInTheDocument();
    });
  });
});
```

Рисунок 4.10 – Модульні тести компонента AudioUploader

У модульних тестах перевіряється:

- базовий рендеринг компонента без помилок (renders without crashing);
- коректна обробка завантаження файлів (handles file change correctly);

- правильність оновлення стану компонента;
- відображення інтерфейсу завантаження.

Для імітації зовнішніх залежностей використовується механізм мокування. Наприклад, `mockAxios` імітує HTTP-запити, що дозволяє тестувати компонент без реальної взаємодії з сервером.

Інтеграційне тестування фокусується на перевірці правильної взаємодії між різними модулями системи. Для Django використовується вбудований фреймворк тестування разом з `APITestCase` для тестування REST API. На рисунку 4.11 показано реалізацію інтеграційних тестів для серверної частини.

```
class AudioFileIntegrationTest(APITestCase):
    def setUp(self):
        self.file = SimpleUploadedFile("test.mp3", b"file_content", content_type="audio/mp3")

    def test_create_audio_file_with_google_recognition(self):
        with patch('recognition.views.sr.Recognizer') as mock_recognizer:
            mock_recognizer.return_value.record.return_value = "mock_transcript"

            data = {
                'file': self.file,
                'recognition_method': 'google',
            }

            response = self.client.post('/api/audiofiles/', data, format='multipart')

            self.assertEqual(response.status_code, status.HTTP_201_CREATED)
            self.assertEqual(AudioFile.objects.count(), 1)
            self.assertEqual(AudioFile.objects.first().transcript, 'mock_transcript')
```

Рисунок 4.11 – Інтеграційні тести AudioFile

Інтеграційні тести перевіряють:

- повний цикл створення та обробки аудіофайлу;
- взаємодію з різними системами розпізнавання;
- коректність роботи API endpoints;
- обробку помилок та граничних випадків.

Особлива увага приділяється тестуванню взаємодії з системами розпізнавання. Використовується патчинг (`patch`) для імітації роботи розпізнавачів, що дозволяє проводити тести незалежно від доступності зовнішніх сервісів.

В обох типах тестів використовуються assert-перевірки для валідації очікуваних результатів:

- assertEquals для перевірки точних співпадінь;
- assertTrue/assertFalse для логічних перевірок;
- assertEquals для перевірки входження значень.

Для забезпечення надійного тестування використовуються спеціальні тестові файли та імітація роботи систем розпізнавання через механізм мокування. Це дозволяє проводити тестування без залежності від зовнішніх сервісів та забезпечити стабільність тестів.

Тестування проводиться автоматично при кожному оновленні коду, що дає змогу оперативно виявляти та усувати потенційні помилки. Обраний підхід забезпечує високу якість та надійність системи.

## **4.7 Керівництво користувача**

### **4.7.1 Підготовка до роботи**

Для успішного використання системи розпізнавання аудіо необхідно забезпечити відповідність технічним вимогам. Система потребує використання сучасних версій веб-браузерів. Найкраща продуктивність досягається при використанні Google Chrome версії 90 або новішої, Mozilla Firefox версії 88 або новішої. Також підтримуються Safari та Opera останніх версій. Важливо мати увімкнену підтримку JavaScript у браузері, оскільки це важливо для роботи користувацького інтерфейсу.

Для доступу до системи необхідно відкрити веб-браузер та ввести адресу <https://audio.loc>. При першому відвідуванні рекомендується дозволити браузеру доступ до локальних файлів, що необхідно для завантаження аудіофайлів. У випадку проблем з доступом до системи слід перевірити стабільність інтернет-з'єднання та спробувати очистити кеш браузера.

При підготовці аудіофайлів для розпізнавання варто врахувати певні обмеження та рекомендації. Система працює з файлами форматів MP3 та WAV,

при цьому розмір файлу не повинен перевищувати 10 МБ. Для досягнення найкращих результатів розпізнавання рекомендується використовувати аудіозаписи з мінімальним рівнем фонового шуму, чітким мовленням та якісним записом звуку.

Система має певні технічні обмеження, про які варто пам'ятати. При використанні Google Web Speech API необхідне стабільне підключення до інтернету, тоді як PocketSphinx може працювати в офлайн-режимі. Час обробки файлу може варіюватися залежно від його розміру та обраного методу розпізнавання. Для великих файлів рекомендується розділяти їх на менші частини для більш ефективної обробки.

У разі виникнення проблем з роботою системи рекомендується спочатку перевірити актуальність версії браузера та стабільність інтернет-з'єднання. Якщо проблема зберігається, можна спробувати альтернативний браузер або очистити кеш і файли cookie. При постійних проблемах рекомендується звернутися до адміністратора системи.

#### 4.7.2 Загальний опис інтерфейсу

Базова сторінка системи, яку бачить користувач при першому вході, відображена на рисунку 4.12. Початковий стан містить інформаційне повідомлення про відсутність розпізнаних файлів. При розробці інтерфейсу особлива увага приділялася інтуїтивності та простоті використання.

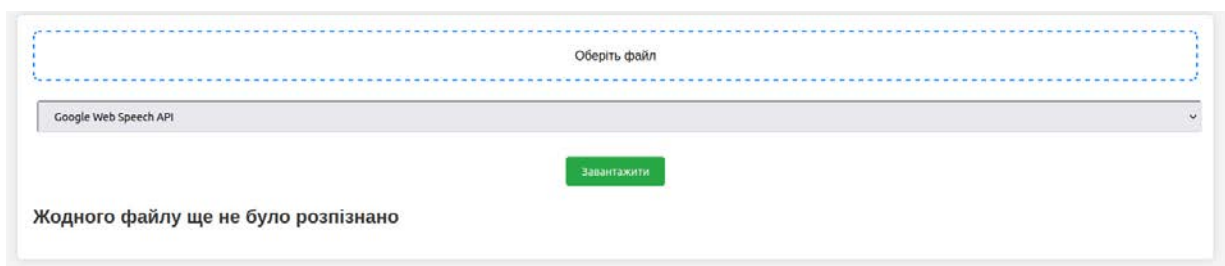


Рисунок 4.12 – Початковий стан головної сторінки системи

Структурно інтерфейс поділено на чотири основні блоки, розташовані у логічній послідовності:

- секція для завантаження аудіофайлу;
- панель вибору методу розпізнавання;
- елемент запуску розпізнавання;
- область виведення результатів.

Важливою особливістю інтерфейсу є його адаптивність – він автоматично оптимізується під різні розміри екранів, забезпечуючи зручність використання як на комп'ютерах, так і на мобільних пристроях.

### 4.7.3 Функціональні елементи інтерфейсу

Завантаження файлу. Секція завантаження файлів (рис. 4.13) є початковою точкою взаємодії з системою. Натискання на неї активує стандартний діалог вибору файлів (рисунок 4.14). Після вибору файлу його назва з'являється в інтерфейсі, як показано на рисунку 4.15.

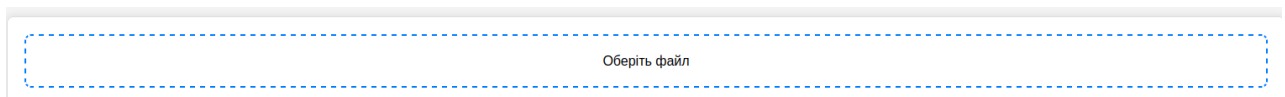


Рисунок 4.13 – Компонент завантаження аудіофайлів

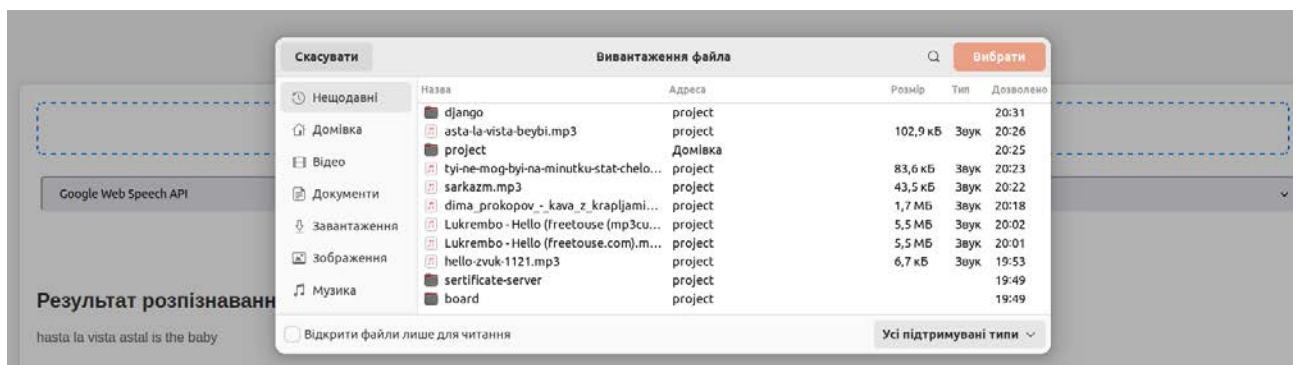


Рисунок 4.14 – Діалогове вікно вибору файлу



Рисунок 4.15 – Інтерфейс системи після успішного завантаження файлу

Вибір системи розпізнавання: за замовчуванням використовується Google Web Speech API. (рисунок 4.16). Користувач має можливість змінити метод розпізнавання через спеціальний селектор (рисунок 4.17), обираючи між онлайн-розпізнаванням через Google Web Speech API та локальною обробкою за допомогою PocketSphinx.



Рисунок 4.16 – Панель вибору методу розпізнавання



Рисунок 4.17 – Селектор методів розпізнавання

Відображення результатів. Фінальні результати розпізнавання відображаються у нижній частині інтерфейсу. На рисунках 4.18 та 4.19 представлені приклади виведення результатів для обох методів розпізнавання. Система надає можливість зберегти розпізнаний текст у різних форматах.

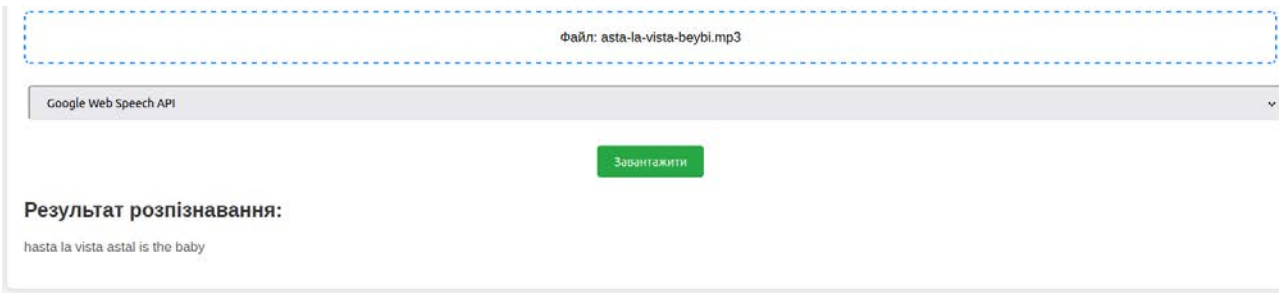


Рисунок 4.18 – Виведення результатів онлайн-розпізнавання

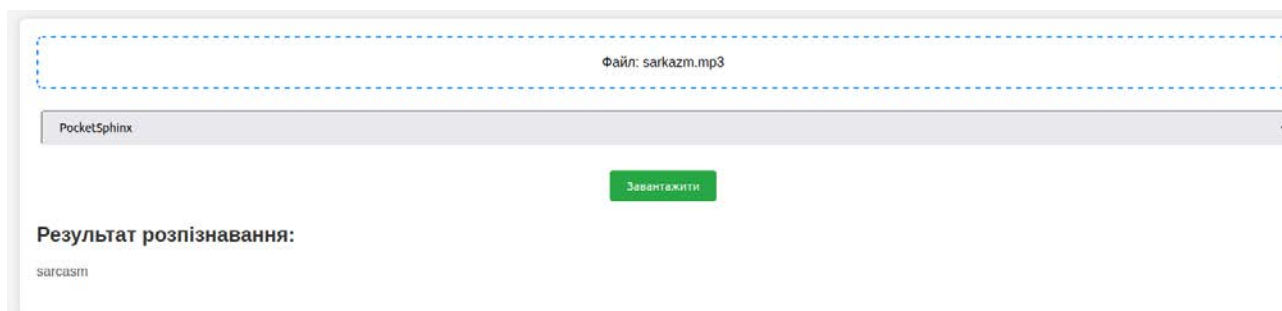


Рисунок 4.19 – Виведення результатів локального розпізнавання

Всі елементи керування супроводжуються візуальними індикаторами та чіткими позначеннями їх функцій. Система забезпечує постійний зворотний зв'язок через індикатори процесу та інформаційні сповіщення про стан операцій.

#### 4.8 Висновки за розділом 4

Показано архітектурні особливості реалізації системи через взаємодію основних компонентів: AudioUploader відповідає за реалізацію клієнтської частини, забезпечуючи завантаження аудіофайлів, тоді як AudioFileViewSet на серверній стороні обробляє запити, пов'язані з управлінням аудіофайлами. Розроблена архітектура забезпечує чітке розділення відповідальності та гнучкість у розширенні функціоналу системи.

Визначено ефективність системи тестування через впровадження модульних та інтеграційних тестів. Модульні тести забезпечують надійність окремих компонентів, а інтеграційні - коректність їх взаємодії. Використання механізмів мокування дозволяє проводити тестування незалежно від зовнішніх сервісів.

Встановлено, що розроблений користувацький інтерфейс відповідає вимогам зручності використання та інтуїтивної зрозумілості. Система надає користувачам повний набір функцій пов'язаних з управлінням та обробкою аудіофайлів: від завантаження до отримання результатів розпізнавання, при цьому забезпечуючи адаптивність під різні пристрої та надійний зворотний зв'язок.

## ВИСНОВКИ

У результаті виконання магістерської роботи була створена автоматизована система для розпізнавання аудіосигналів, яка базується на сучасних веб-технологіях, забезпечуючи зручність використання, високу точність і ефективність обробки даних..

Виявлено актуальність та перспективність розробки систем розпізнавання аудіо через їх широке застосування в різних галузях: від медичної діагностики до систем безпеки. Аналіз існуючих рішень показав необхідність створення системи, що поєднує простоту використання з надійністю роботи.

Показано ефективність використання комбінації технологій Django та React.js для реалізації системи. Django забезпечує надійну серверну частину з вбудованими механізмами безпеки та управління даними, тоді як React.js надає гнучкий інструментарій для створення інтерактивного користувацького інтерфейсу.

Визначено оптимальну архітектуру системи через використання паттерну MVC та REST API для взаємодії між компонентами. Спроектowana структура забезпечує чітке розділення відповідальності, що підтверджено розробленими UML-діаграмами: варіантів використання, діяльності, послідовності, класів та розгортання.

Встановлено надійність розробленої системи завдяки впровадженню комплексного тестування, що включає модульні та інтеграційні тести. Тестування охоплює всі ключові компоненти системи та забезпечує стабільність її роботи при різних сценаріях використання.

Результатом роботи є повнофункціональна система розпізнавання аудіо, що надає користувачам:

- зручний веб-інтерфейс для завантаження та обробки аудіофайлів;
- можливість вибору методу розпізнавання;
- надійне збереження та доступ до результатів;
- адаптивність під різні пристрої та браузері.

Розроблена система може бути використана як самостійний продукт або інтегрована в більші інформаційні системи, що робить її універсальним інструментом для вирішення задач розпізнавання аудіо сигналів.

## ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Войтко, В. В. Аналіз сучасних засобів створення та обробки аудіоконтенту / В. В. Войтко, С. В. Бевз, С. М. Бурбело та ін. // Вчені записки ТНУ імені В. І. Вернадського. Серія: Технічні науки. – 2020. – Том 31 (70), № 1. – С. 55–59.
2. Одінцов, О. В. Комп'ютерна система розпізнавання джерела акустичних сигналів / О. В. Одінцов, О. В. Ткаченко // Вісник Вінницького політехнічного інституту. – 2017. – № 6. – С. 7–12.
3. Kheddar, H. Himeur Automatic Speech Recognition using Advanced Deep Learning Approaches: A survey [Electronic resource] / H. Kheddar, M. Hemis, Y. Himeur // Information Fusion, Elsevier. – 2024. – Access mode: <https://arxiv.org/pdf/2403.01255>.
4. An End-to-End Audio Classification System based on Raw Waveforms and Mix-Training Strategy / J. Chen, J. Hao, K. Chen et al. // INTERSPEECH 2019: 20th Annual Conference of the International Speech Communication Association, Graz, 15–19 September 2019: proceedings. – Red Hook: ISCA, 2019. – P. 3644 – 3648.
5. Tacotron: Towards End-to-End Speech Synthesis / Y. Wang, R.J. Skerry-Ryan, D. Stanton et al. // INTERSPEECH 2017: 18th Annual Conference of the International Speech Communication Association, Stockholm, 20–24 August 2017: proceedings. – Red Hook: ISCA, 2017. – P.4006 – 4010.
6. Юдін, О. К. Аналіз сучасних систем та методів розпізнавання аудіосигналів у задачах ідентифікації та верифікації / О. К. Юдін, Р. В. Зюбіна // Проблеми інформатизації та управління. – 2017. – № 3 (59). – С. 75–79.
7. Technology of sound source localization based on incoming sound intensity / A. Sieriebriakov, O. Hospodarchuk, D. Lakhtyr et al. // 2023 IEEE: 18th International Conference on Computer Science and Information Technologies (CSIT), Lviv, 19–21 October 2023: proceedings. – Los Alamitos: IEEE, 2023. – P. 1–5.
8. Barry, P. Head First Python: A Brain-Friendly Guide / Paul Barry. – Sebastopol: O'Reilly Media, 2023. – 560 p.

9. Оцінка ефективності засобів розпізнавання ключових слів у голосовому сигналі / І. А. Дичка, І. А. Терейковський, А. В. Дідус та ін. // Вчені записки ТНУ імені В. І. Вернадського. Серія: Технічні науки. – 2023. – Том 34 (73), № 3. – С. 123–129.
10. Huang, X. Spoken Language Processing. Guide to Algorithms and System Developmen / X. Huang, A. Acero, H.-W. Hon. – Upper Saddle River: Prentice Hall PTR, 2001. – 980 p.
11. Mohan, A. Cross-lingual context sharing and parameter-tying for multi-lingual speech recognition / A. Mohan, R. Rose // 2013 IEEE: Workshop on Automatic Speech Recognition and Understanding, Olomouc, 8-12 December 2013: proceedings. – Los Alamitos: IEEE, 2013. – P. 126–131.
12. Нейромережеві моделі та методи розпізнавання фонем в голосовому сигналі в системі дистанційного навчання / В. М. Михайленко, Л. О. Терейковська, І. А. Терейковський та ін. – Київ: ЦП «Компринт», 2017. – 252 с.
13. Deep Learning for Audio Signal Processing / H. Purwins; B. Li, T. Virtanen et al. // IEEE Journal of Selected Topics in Signal Processing. – 2019. – Vol. 13(2). – P. 206-219.
14. Booch, G. The Unified Modeling Language User Guide / G. Booch, J. Rumbaugh, I. Jacobson. – Boston: Addison-Wesley, 2005. – 496 p.
15. Django documentation [Electronic resource]. – Access mode: <https://docs.djangoproject.com/>.
16. Django and React: A Full Stack Tutorial [Electronic resource]. – Access mode: <https://www.fullstackpython.com/django-react-tutorial.html>.
17. Using PostgreSQL with Django [Electronic resource]. – Access mode: <https://docs.djangoproject.com/en/stable/ref/databases/>.
18. WebSockets in Django [Electronic resource]. – Access mode: <https://channels.readthedocs.io/>.
19. Automate the Boring Stuff with Python [Electronic resource]. – Access mode: <https://automatetheboringstuff.com/>.

20. Integrating Django with Reactjs using Django REST Framework [Electronic resource]. – Access mode: [https://www.geeksforgeeks.org/integrating-django-with-reactjs-using-django-rest-framework/?utm\\_source=chatgpt.com](https://www.geeksforgeeks.org/integrating-django-with-reactjs-using-django-rest-framework/?utm_source=chatgpt.com) /
21. React – A JavaScript library for building user interfaces [Electronic resource]. – Access mode: <https://reactjs.org/>.
22. SQL Tutorial [Electronic resource]. – Access mode: <https://www.w3schools.com/sql/>.
23. Fowler, M. UML Distilled: A Brief Guide to the Standard Object Modeling Language / M. Fowler. – Boston: Addison-Wesley, 2004. – 208 p.

**ДОДАТОК А**  
**Технічне завдання**

## **Вступ**

Технічне завдання описує розробку вебзастосунку для розпізнавання аудіо сигналів. Система призначена для надання користувачам зручного інтерфейсу завантаження та обробки аудіофайлів з можливістю вибору методу розпізнавання та формату виведення результатів. Основними завданнями системи є забезпечення точного розпізнавання аудіо, зручності використання та надійності роботи. Система має підтримувати популярні формати аудіофайлів та забезпечувати збереження результатів розпізнавання.

### **А.1 Підстава для розробки**

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Розробка автоматизованої системи розпізнавання аудіо сигналів», затверджене наказом Національного університету «Запорізька політехніка» № 398 від 07 жовтня 2024 р.

### **А.2 Призначення розробки**

Система призначена для автоматизованого розпізнавання аудіо сигналів через веб-інтерфейс. Основними завданнями системи є:

- надання зручного інтерфейсу для завантаження аудіофайлів;
- забезпечення вибору методу розпізнавання;
- обробка та розпізнавання аудіо сигналів;
- збереження та надання доступу до результатів розпізнавання.

### **А.3 Основні вимоги до програми, що розробляється**

#### **А.3.1 Вимоги до функціональних характеристик**

Головні функціональні вимоги можна представити наступним чином:

а) керування аудіофайлами:

- 1) завантаження файлів форматів MP3 та WAV;
- 2) перевірка розміру та формату файлів;
- 3) можливість зміни та видалення поточного файлу;

б) керування системами розпізнавання:

- 1) вибір між Google Web Speech API та PocketSphinx;
- 2) збереження налаштувань розпізнавання;

в) керування результатами:

- 1) генерація результатів розпізнавання;
- 2) перегляд результатів;
- 3) збереження результатів у різних форматах;
- 4) ведення історії розпізнавань.

### **A.3.2 Вимоги до інтерфейсу програми**

Інтерфейс програми повинен бути розроблений з урахуванням принципів UI/UX дизайну та забезпечувати максимальну зручність використання для кінцевого користувача. Вимогами є:

- адаптивний дизайн для різних пристроїв;
- інтуїтивно зрозуміле розташування елементів;
- миттєвий відгук на дії користувача;
- чітке відображення статусу операцій;
- зрозумілі повідомлення про помилки;
- підтримка різних форматів виведення результатів.

### **A.3.3 Вимоги до надійності**

Система повинна забезпечувати стабільну роботу та цілісність даних при різних сценаріях використання та можливих збоях, вимогами є:

- валідація вхідних файлів;
- обробка помилок завантаження та розпізнавання;
- збереження проміжних результатів;
- можливість повторної спроби при помилках;
- захист від втрати даних при збоях;
- журналювання критичних подій.

#### **A.3.4 Умови експлуатації**

Для експлуатації програмного забезпечення для розпізнавання аудіо сигналів необхідна наявність:

- сучасного веб-браузера (Google Chrome, Mozilla Firefox, Safari, Opera);
- стабільного підключення до інтернету;
- увімкненої підтримки JavaScript;
- дозволу на доступ до локальних файлів у браузері.

#### **A.3.5 Вимоги до складу та параметрів технічний засобів**

Для експлуатації програмного забезпечення для розпізнавання аудіо сигналів необхідно таке апаратне та програмне забезпечення:

а) серверна частина:

- 1) ОС Linux Ubuntu 18+;
- 2) Python 3.7+;
- 3) веб-сервер Apache2/nginx;
- 4) SQLite3;

б) клієнтська частина:

- 1) сучасний веб-браузер;
- 2) підтримка JavaScript;

- 3) мінімум 2 ГБ оперативної пам'яті;
- 4) вільний дисковий простір від 500 МБ.

### **A.3.6 Вимоги до маркування і пакування**

Система постачається у вигляді веб-застосунку, що не потребує спеціального маркування чи пакування. Всі компоненти системи та документація надаються в електронному вигляді через систему контролю версій Git.

**ДОДАТОК Б**  
**Програмний код**

## Б.1 Код компоненты AudioUploader()

# КОМПОНЕНТ

```
import React, { useState } from 'react';
import axios from 'axios';
import './App.css';

function SoundRecognizer() {
  const [audioFile, setAudioFile] = useState(null);
  const [recognitionResult, setRecognitionResult] = useState("");
  const [processingMethod, setProcessingMethod] = useState('google');
  const [uploadedFileName, setUploadedFileName] = useState("");

  const processFileSelection = (event) => {
    const chosenFile = event.target.files[0];
    setAudioFile(chosenFile);
    setUploadedFileName(chosenFile.name);
  };

  const updateProcessingMethod = (event) => {
    setProcessingMethod(event.target.value);
  };

  const processRecognition = async () => {
    const requestData = new FormData();
    requestData.append('file', audioFile);
    requestData.append('recognition_method', processingMethod);

    try {
      const result = await axios.post('http://localhost:8000/api/audiofiles/', requestData, {
        headers: {
          'Content-Type': 'multipart/form-data',
        },
      });

      setRecognitionResult(result.data.transcript);
    } catch (error) {
      console.error('Recognition processing error:', error);
    }
  };

  return (
    <div className="container">
```

```

<div className="section input-section" onClick={() =>
document.getElementById('audioInput').click()}>
    {!uploadedFileName && <label htmlFor="audioInput">Оберіть файл</label>}
    <input type="file" id="audioInput" accept=".mp3, .wav"
onChange={processFileSelection} />
    {uploadedFileName && <span>Файл: {uploadedFileName}</span>}
</div>

<div className="section">
    <select value={processingMethod} onChange={updateProcessingMethod}>
        <option value="google">Google Web Speech API</option>
        <option value="pocketsphinx">PocketSphinx</option>
    </select>
</div>

<div className="section">
    <button onClick={processRecognition}>Обробити</button>
</div>

{recognitionResult && (
    <div className="section result">
        <h2>Результат розпізнавання:</h2>
        <p>{recognitionResult}</p>
    </div>
)}

{!recognitionResult && (
    <div className="section result">
        <h2>Жодного файлу ще не було розпізнано</h2>
    </div>
)}
</div>
);
}

```

```
export default SoundRecognizer;
```

## Б.2 Код моделі

```

# Модель
from django.db import models

class SoundRecord(models.Model):

```

```
audio_data = models.FileField(upload_to='audio/')
recognition_text = models.TextField(blank=True)
is_processed = models.BooleanField(default=False)
```

```
def save(self, *args, **kwargs):
    if self.audio_data:
        self.audio_data._committed = True
    super().save(*args, **kwargs)
```

### Б.3 Код представления

```
# Представления
from rest_framework import viewsets, status
from rest_framework.response import Response
from .models import SoundRecord
from .serializers import SoundRecordSerializer
import speech_recognition as sr
from pocketsphinx import LiveSpeech
from pydub import AudioSegment

class SoundProcessorViewSet(viewsets.ModelViewSet):
    queryset = SoundRecord.objects.all()
    serializer_class = SoundRecordSerializer

    def transform_audio_format(self, input_path):
        audio_content = AudioSegment.from_file(input_path)
        return audio_content.export(format="wav")

    def create(self, request, *args, **kwargs):
        audio_content = request.data.get('file', None)
        processor = None

        if audio_content is None:
            return Response({'file': 'This field is required.'},
                status=status.HTTP_400_BAD_REQUEST)

        processing_type = request.data.get('recognition_method', 'google')

        try:
            wav_content = self.transform_audio_format(audio_content)
        except Exception as e:
            return Response({'error': f'Error transforming audio to WAV: {e}'},
                status=status.HTTP_500_INTERNAL_SERVER_ERROR)
```

```

sound_record = SoundRecord(audio_data=wav_content)

if processing_type == 'google':
    processor = sr.Recognizer()
elif processing_type == 'pocketsphinx':
    processor = LiveSpeech()
else:
    return Response({'recognition_method': 'Invalid processing type.'},
status=status.HTTP_400_BAD_REQUEST)

try:
    with sr.AudioFile(wav_content) as source:
        if processing_type == 'google':
            audio_content = processor.record(source)
            recognition_result = processor.recognize_google(audio_content)
        elif processing_type == 'pocketsphinx':
            recognition_result = ""
            for segment in processor:
                recognition_result += str(segment) + " "

        sound_record.recognition_text = recognition_result
        sound_record.is_processed = True
        sound_record.save()
        serializer = SoundRecordSerializer(sound_record)
        return Response(serializer.data, status=status.HTTP_201_CREATED)
except sr.UnknownValueError:
    return Response({'error': 'Audio content recognition failed.'},
status=status.HTTP_400_BAD_REQUEST)
except sr.RequestError as e:
    return Response({'error': f'Recognition processing error: {e}'},
status=status.HTTP_500_INTERNAL_SERVER_ERROR)

```

**ДОДАТОК В**  
**Слайди презентації**

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
Національний університет «Запорізька політехніка»  
Кафедра програмних засобів

# РОЗРОБКА АВТОМАТИЗОВАНОЇ СИСТЕМИ РОЗПІЗНАВАННЯ АУДІО СИГНАЛІВ

---

- Виконав: студент групи КНТ-123м
- Керівник: д-р.філософ.

Денис МОРОЗОВ

Сергій ЛЕОЩЕНКО

Рисунок В.1 – Слайд 1

- Об'єкт дослідження – процес розробки автоматизованої системи розпізнавання аудіо сигналів із використанням сучасних веб-технологій.
- Предмет дослідження – методи та засоби реалізації веб-систем розпізнавання аудіо з використанням сучасних веб-технологій.
- Мета роботи – розробка системи розпізнавання аудіо сигналів з використанням сучасних веб-технологій для забезпечення високої точності розпізнавання та зручного користувацького інтерфейсу.
- Актуальність дослідження. Актуальність даної теми зумовлена необхідністю створення ефективних та гнучких інструментів для обробки аудіо сигналів. Така розробка має важливе значення для впровадження інноваційних рішень у сфері автоматизації процесів, підвищення якості взаємодії між користувачами та інформаційними системами, а також розширення можливостей аналізу великих обсягів аудіоданих.

Рисунок В.2 – Слайд 2

## ЗАВДАННЯ ДОСЛІДЖЕННЯ:

- провести аналіз існуючих рішень та методів розпізнавання аудіо сигналів, визначити їх переваги та обмеження;
- сформулювати вимоги до системи розпізнавання аудіо сигналів та обрати оптимальні технології для її реалізації;
- спроектувати архітектуру системи, визначити основні компоненти та взаємозв'язки між ними;
- розробити серверну частину системи з використанням [Django](#), реалізувати необхідні API та функції обробки аудіо;
- створити клієнтську частину на базі [React.js](#) з інтуїтивно зрозумілим інтерфейсом користувача;
- провести тестування розробленої системи.

Рисунок В.3 – Слайд 3

## Аналіз методів та алгоритмів розпізнавання аудіо

| Метод/ Алгоритм                             | Принципи роботи                                                | Переваги                                                                                                   | Недоліки                                                                                      | Ефективність застосування                                                                            |
|---------------------------------------------|----------------------------------------------------------------|------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| Приховані марковські моделі (HMM)           | Моделювання послідовності звуків як ланцюжка прихованих станів | - врахування часової динаміки;<br>- моделювання варіативності вимови;<br>- використання алгоритму Вітербі. | - складність навчання;<br>- потреба у великій кількості даних;<br>- обмежена точність.        | - базові системи розпізнавання;<br>- класифікація послідовностей;<br>- інтеграція з іншими методами. |
| Динамічна трансформація часової шкали (DTW) | Порівняння часових послідовностей різної довжини               | - простота реалізації;<br>- робота з різною тривалістю;<br>- незалежність від диктора.                     | - висока обчислювальна складність;<br>- чутливість до шуму;<br>- обмежена масштабованість.    | - розпізнавання окремих слів;<br>- порівняння патернів;<br>- прототипні системи.                     |
| Згорткові нейронні мережі (CNN)             | Обробка спектрограм та виявлення частотно-часових патернів     | - ефективна обробка локальних патернів;<br>- стійкість до зсувів;<br>- паралельна обробка.                 | - потреба у великих датасетах;<br>- ресурсоемісність навчання;<br>- складність інтерпретації. | - аналіз спектрограм;<br>- класифікація звуків;<br>- виділення ознак.                                |
| Рекурентні нейронні мережі (RNN/LSTM)       | Моделювання довготривалих залежностей в послідовностях         | - обробка послідовностей довільної довжини;<br>- врахування контексту;<br>- адаптивне навчання.            | - проблема зникаючого градієнта;<br>- складність навчання;<br>- послідовна обробка.           | - розпізнавання мовлення;<br>- моделювання контексту;<br>- реальночасові системи.                    |
| Transformer                                 | Паралельна обробка послідовностей з механізмом самоуваги       | - ефективна паралельна обробка;<br>- глобальний контекст;<br>- висока точність.                            | - квадратична складність;<br>- великі вимоги до пам'яті;<br>- складна архітектура.            | - сучасні системи розпізнавання;<br>- багатомовні системи;<br>- складні завдання.                    |
| End-to-end системи                          | Пряме перетворення аудіо в текст                               | - оптимізація всього процесу;<br>- спрощена архітектура;<br>- краща продуктивність.                        | - потреба у великих датасетах;<br>- складність навчання;<br>- менша гнучкість.                | - промислові системи;<br>- інтегровані рішення;<br>- спеціалізовані застосування.                    |

Рисунок В.4 – Слайд 4

## Діаграма варіантів використання

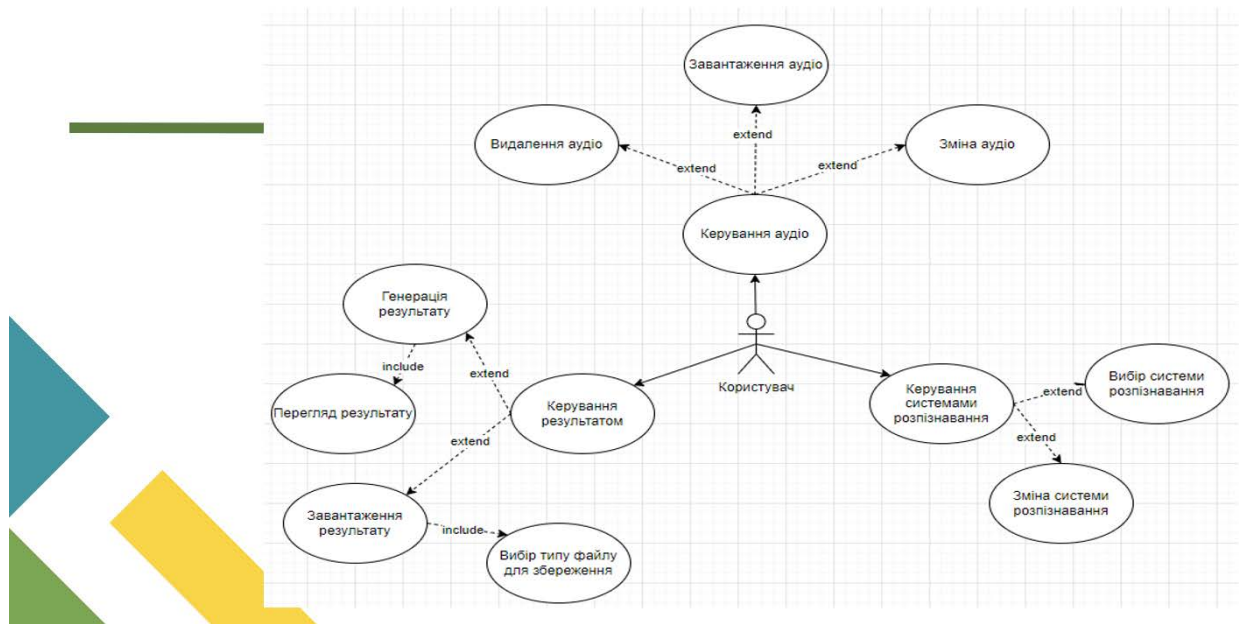


Рисунок В.5 – Слайд 5

## Діаграма діяльності

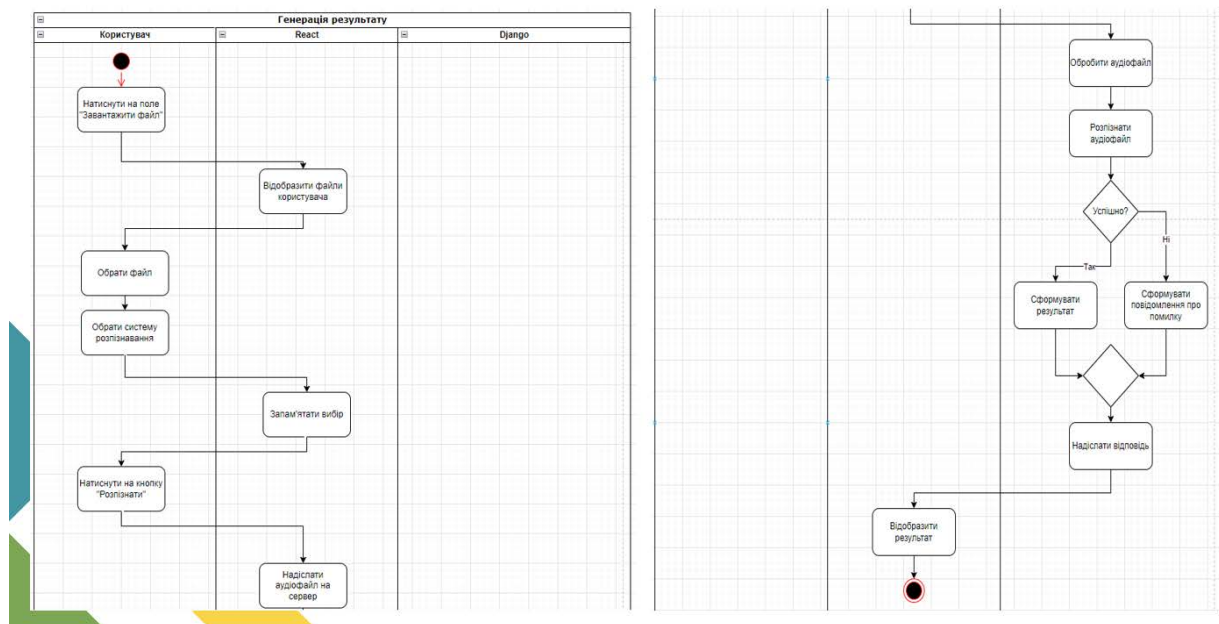


Рисунок В.6 – Слайд 6

## Діаграма послідовності

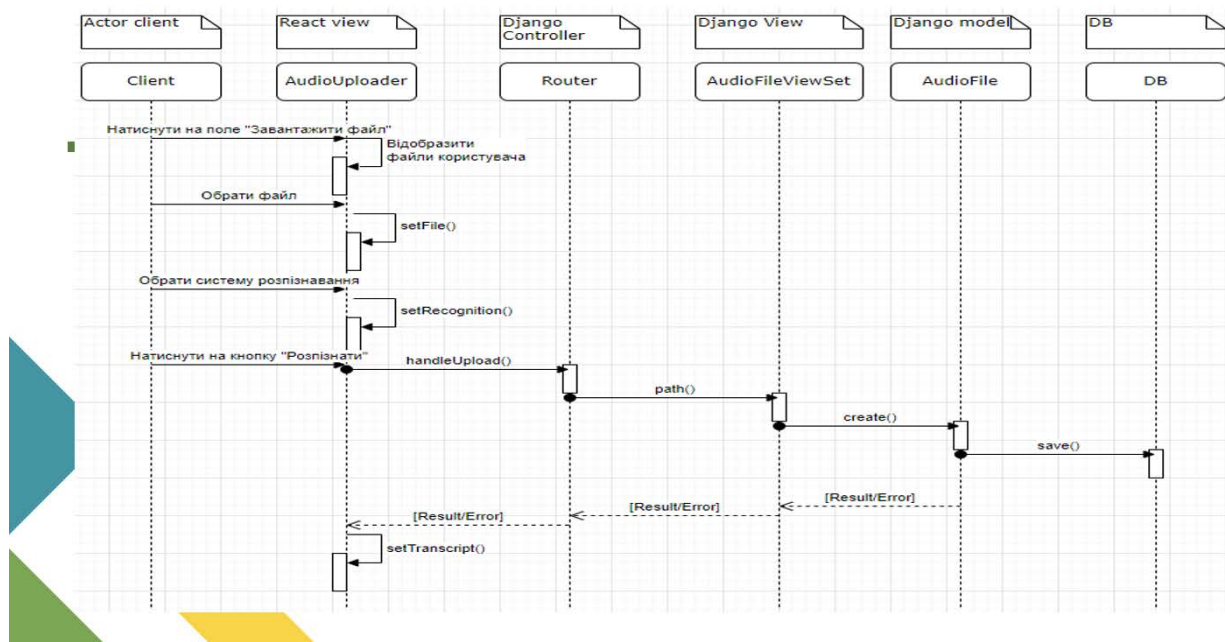


Рисунок В.7 – Слайд 7

## Діаграма класів

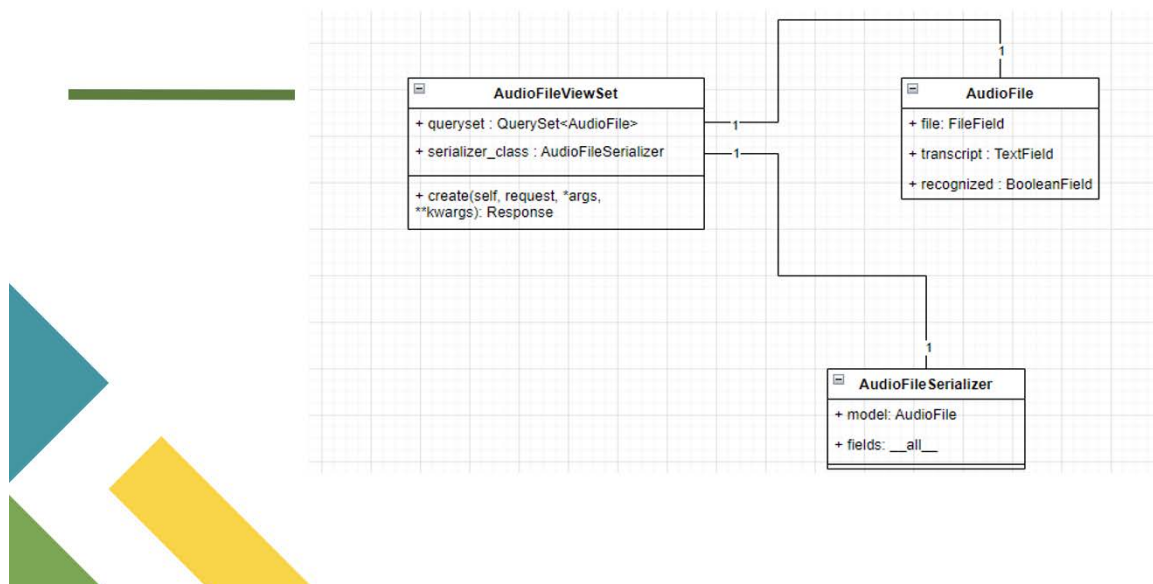


Рисунок В.8 – Слайд 8

## Діаграма розгортання

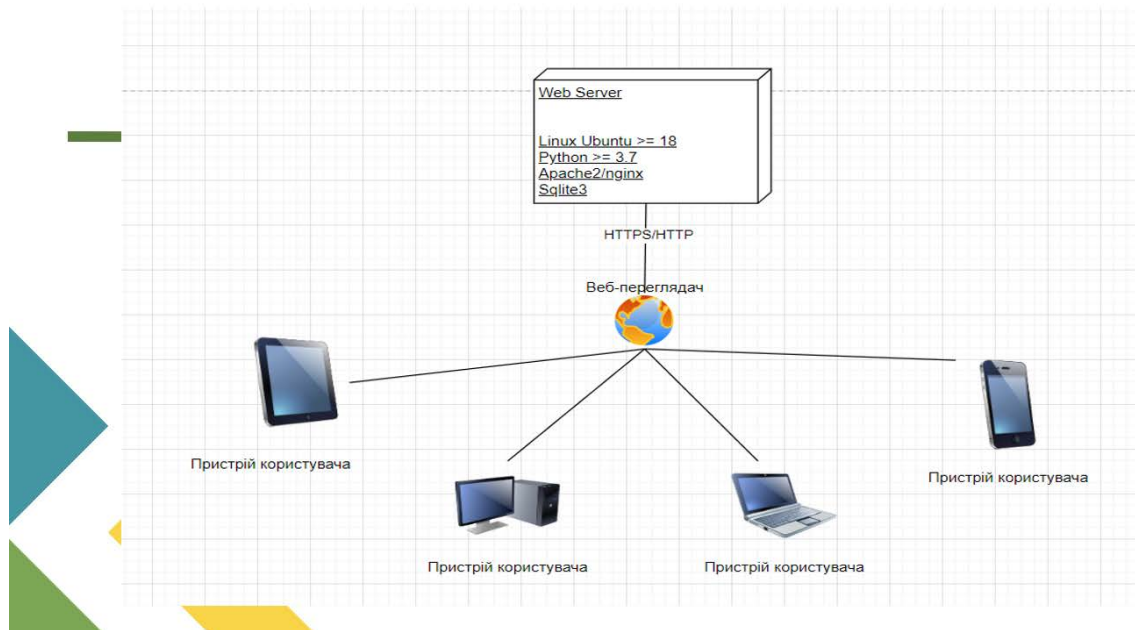


Рисунок В.9 – Слайд 9

## Структура навчального набору даних ESC-50

Попередній перегляд даних вибірки:

|      | filename          | fold | target | category       | esc10 | src_file | take |
|------|-------------------|------|--------|----------------|-------|----------|------|
| 1995 | 5-263831-B-6.wav  | 5    | 6      | hen            | False | 263831   | B    |
| 1996 | 5-263902-A-36.wav | 5    | 36     | vacuum_cleaner | False | 263902   | A    |
| 1997 | 5-51149-A-25.wav  | 5    | 25     | footsteps      | False | 51149    | A    |
| 1998 | 5-61635-A-8.wav   | 5    | 8      | sheep          | False | 61635    | A    |
| 1999 | 5-9032-A-0.wav    | 5    | 0      | dog            | True  | 9032     | A    |

Розподіл класів у наборі даних ESC-50

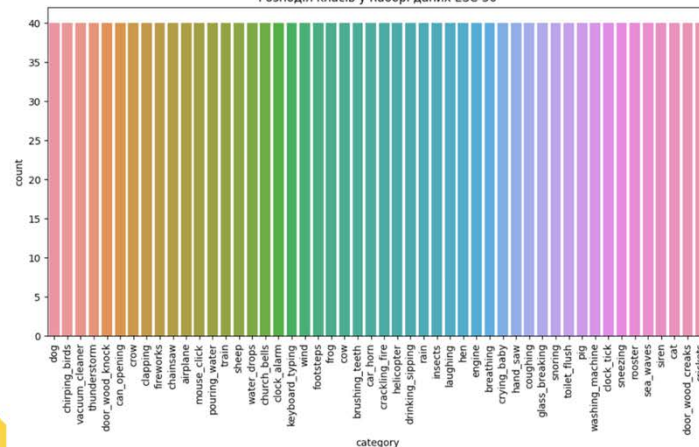


Рисунок В.10 – Слайд 10

## Матриця помилок класифікації

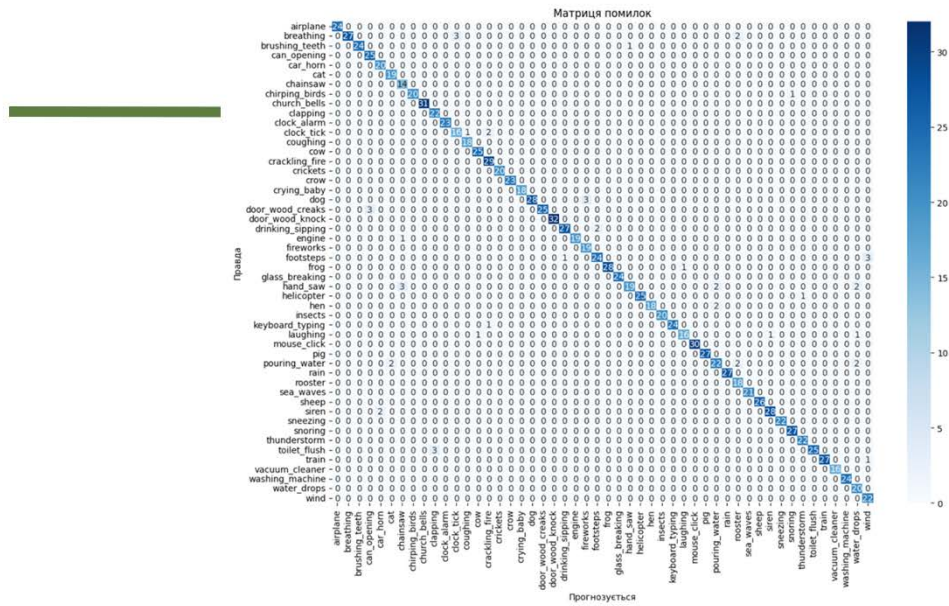


Рисунок В.11 – Слайд 11

## Метрики якості розпізнавання

Класифікаційний звіт:

|                  | precision | recall | f1-score | support |
|------------------|-----------|--------|----------|---------|
| airplane         | 1.00      | 1.00   | 1.00     | 24      |
| breathing        | 1.00      | 0.84   | 0.92     | 32      |
| brushing_teeth   | 1.00      | 0.96   | 0.98     | 25      |
| can_opening      | 0.89      | 1.00   | 0.94     | 25      |
| car_horn         | 0.91      | 1.00   | 0.95     | 20      |
| cat              | 0.90      | 1.00   | 0.95     | 19      |
| chainsaw         | 0.78      | 1.00   | 0.88     | 14      |
| chirping_birds   | 1.00      | 0.95   | 0.98     | 21      |
| church_bells     | 1.00      | 1.00   | 1.00     | 31      |
| clapping         | 0.88      | 1.00   | 0.94     | 22      |
| clock_alarm      | 1.00      | 1.00   | 1.00     | 23      |
| clock_tick       | 0.84      | 0.84   | 0.84     | 19      |
| coughing         | 0.95      | 1.00   | 0.97     | 18      |
| cow              | 0.96      | 1.00   | 0.98     | 25      |
| crackling_fire   | 0.91      | 1.00   | 0.95     | 29      |
| crickets         | 1.00      | 1.00   | 1.00     | 20      |
| crow             | 1.00      | 1.00   | 1.00     | 23      |
| crying_baby      | 1.00      | 1.00   | 1.00     | 18      |
| dog              | 1.00      | 0.90   | 0.95     | 31      |
| door_wood_creaks | 1.00      | 0.89   | 0.94     | 28      |
| door_wood_knock  | 1.00      | 1.00   | 1.00     | 32      |
| drinking_sipping | 0.96      | 0.93   | 0.95     | 29      |
| engine           | 1.00      | 0.95   | 0.97     | 20      |
| fireworks        | 0.86      | 1.00   | 0.93     | 19      |
| footsteps        | 0.92      | 0.86   | 0.89     | 28      |
| frog             | 1.00      | 0.97   | 0.98     | 29      |
| glass_breaking   | 1.00      | 1.00   | 1.00     | 24      |
| hand_saw         | 0.95      | 0.73   | 0.83     | 26      |
| helicopter       | 1.00      | 0.96   | 0.98     | 26      |
| hen              | 1.00      | 0.90   | 0.95     | 20      |
| insects          | 1.00      | 1.00   | 1.00     | 20      |
| keyboard_typing  | 1.00      | 0.96   | 0.98     | 25      |
| laughing         | 0.94      | 0.84   | 0.89     | 19      |
| mouse_click      | 1.00      | 1.00   | 1.00     | 30      |
| pig              | 1.00      | 1.00   | 1.00     | 27      |
| pouring_water    | 0.85      | 0.79   | 0.81     | 28      |
| rain             | 1.00      | 1.00   | 1.00     | 27      |
| rooster          | 0.82      | 1.00   | 0.90     | 18      |
| sea_waves        | 1.00      | 1.00   | 1.00     | 21      |
| sheep            | 1.00      | 1.00   | 1.00     | 26      |
| siren            | 0.97      | 0.93   | 0.95     | 30      |
| sneezing         | 1.00      | 1.00   | 1.00     | 22      |
| snoring          | 0.96      | 1.00   | 0.98     | 27      |
| thunderstorm     | 0.96      | 1.00   | 0.98     | 22      |
| toilet_flush     | 1.00      | 0.89   | 0.94     | 28      |
| train            | 1.00      | 0.96   | 0.98     | 28      |
| vacuum_cleaner   | 1.00      | 1.00   | 1.00     | 16      |
| washing_machine  | 1.00      | 1.00   | 1.00     | 24      |
| water_drops      | 0.80      | 1.00   | 0.89     | 20      |
| wind             | 0.85      | 1.00   | 0.92     | 22      |
| accuracy         |           |        | 0.96     | 1200    |
| macro avg        | 0.96      | 0.96   | 0.96     | 1200    |
| weighted avg     | 0.96      | 0.96   | 0.96     | 1200    |

Рисунок В.12 – Слайд 12

## Динаміка точності та втрат моделі в процесі навчання та валідації

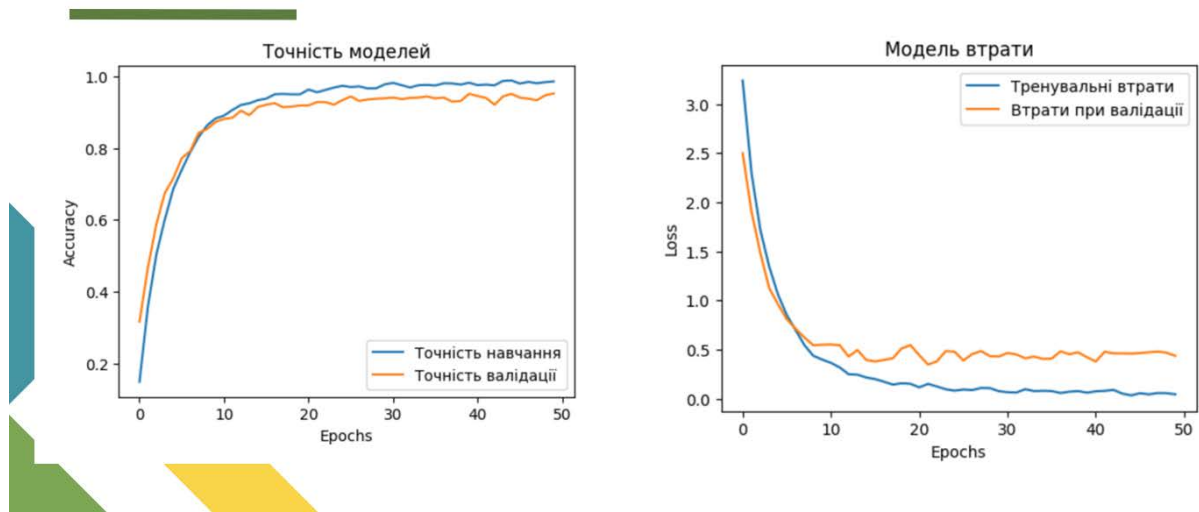


Рисунок В.13 – Слайд 13

## Інтерфейс програми

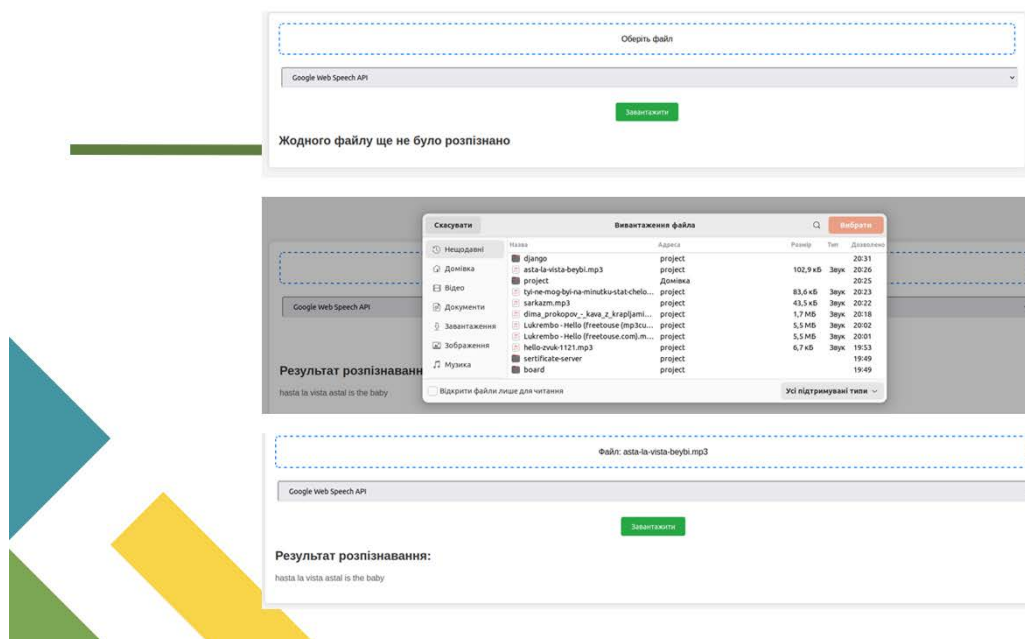


Рисунок В.14 – Слайд 14

## ВИСНОВКИ

У результаті виконання магістерської роботи була створена автоматизована система для розпізнавання аудіосигналів, яка базується на сучасних веб-технологіях, забезпечуючи зручність використання, високу точність і ефективність обробки даних..

Виявлено актуальність та перспективність розробки систем розпізнавання аудіо через їх широке застосування в різних галузях: від медичної діагностики до систем безпеки. Аналіз існуючих рішень показав необхідність створення системи, що поєднує простоту використання з надійністю роботи.

Показано ефективність використання комбінації технологій Django та React.js для реалізації системи. Django забезпечує надійну серверну частину з вбудованими механізмами безпеки та управління даними, тоді як React.js надає гнучкий інструментарій для створення інтерактивного користувацького інтерфейсу.

Визначено оптимальну архітектуру системи через використання паттерну MVC та REST API для взаємодії між компонентами. Спроектована структура забезпечує чітке розділення відповідальності, що підтверджено розробленими UML-діаграмами: варіантів використання, діяльності, послідовності, класів та розгортання.

Встановлено надійність розробленої системи завдяки впровадженню комплексного тестування, що включає модульні та інтеграційні тести. Тестування охоплює всі ключові компоненти системи та забезпечує стабільність її роботи при різних сценаріях використання.

Рисунок В.15 – Слайд 15