Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2025 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення API порталу доступу та обробки звукової інформації”

Виконав здобувач вищої освіти
IV курсу, групи КІ-21-1
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Барамба А.А.
«____» _____ 2025 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Дрєєв О. М.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

Міністерство освіти і науки України
Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Рівень вищої освіти *бакалавр*
Галузь знань *12 "Інформаційні технології"*
Спеціальність *123 Комп'ютерна інженерія*
Освітньо-професійна (освітньо-наукова) програма *"Комп'ютерна інженерія"*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____О.А.Смірнов
« __ » _____ 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Барамбі Андрію Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення API порталу доступу та обробки звукової інформації*

2. Керівник роботи *Дрєєв Олександр Миколайович, канд. техн. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу *№46-02 від 17.01.2025 року*

3. Строк подання студентом роботи до захисту *24.05.2025 р.*

4. Мета та завдання кваліфікаційної бакалаврської роботи*: Метою розробки є програмне забезпечення API порталу доступу та обробки звукової інформації*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Висновки

6. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

7. Дата видачі завдання « » 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної бакалаврської роботи	Строк виконання етапів кваліфікаційної бакалаврської роботи	Примітка
1.	Аналіз існуючих систем	10.03.2025 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2025 р.	
3.	Розробка моделі компонента	20.03.2025 р.	
4.	Розробка структур даних	25.03.2025 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2025 р.	
6.	Програмування алгоритмів	10.04.2025 р.	
7.	Оформлення ПЗ	17.04.2025 р.	
8.	Попередній захист роботи	24.05.2025 р.	

Студент _____ **Барамба А.А.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Дрєєв О.М.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Барамба А.А. Програмне забезпечення API порталу доступу та обробки звукової інформації. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

У даній кваліфікаційній бакалаврській роботі розроблено програмне забезпечення, яке призначено для обробки звукової інформації та доступу до неї.

Метою роботи є створення API порталу доступу та обробки звукової інформації.

Результат роботи – програмна реалізація API порталу доступу та обробки звукової інформації.

В процесі роботи над реалізацією системи виконано дослідження існуючих методів, алгоритмів та програмних засобів. Розроблено та реалізовано власне програмне забезпечення, здійснено опис всіх його компонентів.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows 10/11.

Програму розроблено на середовищах Visual Studio Code та Android Studio з використанням мов програмування Python та Dart відповідно.

Ключові слова: комп'ютерна інженерія, API, аудіофайл, обробка, клієнт, сервер

ABSTRACT

Baramba A.A. Software for API portal for access and processing of audio information. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

This qualification bachelor's thesis has developed software designed to process and access audio information.

The purpose of the work is to create an API portal for accessing and processing audio information.

The result of the work is a software implementation of the API portal for accessing and processing audio information.

In the process of working on the system implementation, a study of existing methods, algorithms and software tools was performed. Own software was developed and implemented, and all its components were described.

A convenient user interface was developed. Instructions for working with the software are provided.

The program can be used on smartphones with Android 12/13.

The program is developed in Visual Studio Code and Android Studio environments using Python and Dart programming languages, respectively.

Keywords: computer engineering, API, audio file, processing, client, server

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

API – Application Programming Interface – набір інструкцій, які дозволяють різним програмам спілкуватись між собою та обмінюватись даними.

HTTP – Hypertext Transfer Protocol – протокол передачі даних, що використовується в комп'ютерних мережах.

MP3 – формат файлу для зберігання аудіоінформації. Є форматом стиснення з втратами.

OSINT – Open-Source Intelligence – концепція, методологія й технологія добування та використання військової, політичної, економічної та іншої інформації з відкритих джерел, без порушення законів.

OGG – Ogg Vorbis – вільний формат стиснення звуку, що розроблявся компанією Xiph.Org Foundation. Використовує власну психоакустичну модель при стисненні з втратами.

REST – Representational State Transfer API – підхід до архітектури мережевих протоколів, які надають доступ до інформаційних ресурсів.

SOAP – Simple Object Access Protocol – протокол обміну структурованими повідомленнями в розподілених обчислювальних системах, базується на форматі XML.

WAV – формат аудіофайлу, розроблений компаніями Microsoft та IBM. Є форматом стиснення без втрат.

Метадані – це структуровані допоміжні дані, які описують параметри та налаштування.

ВСТУП

Актуальність теми. У сучасних умовах стрімкого розвитку цифрових технологій обробка звукової інформації відіграє важливу роль у багатьох сферах — від медіаіндустрії до систем штучного інтелекту. Зростання попиту на автоматизовані системи аналізу, обробки та доступу до аудіоданих зумовлює необхідність створення ефективного, гнучкого та масштабованого програмного забезпечення, зокрема API (Application Programming Interface), яке б забезпечувало централізований доступ до таких можливостей.

Незважаючи на наявність ряду відомих рішень, більшість із них є закритими комерційними платформами, що мають такі обмеження як обмежений або відсутній контроль над внутрішніми алгоритмами обробки аудіо, залежність від закордонної інфраструктури та сервісів, ризики, пов’язані з передачею персональних або чутливих звукових даних на сторонні сервери тощо.

Особливої актуальності ця проблема набуває в умовах повномасштабної війни в Україні, коли використання відкритих даних, зокрема аудіоінформації, може становити серйозну загрозу національній безпеці. Розвідка на основі відкритих джерел (OSINT) активно використовується противником для ідентифікації розташування військових, цивільних об’єктів, характеру переміщень та іншої критично важливої інформації.

Мета й завдання дослідження. Метою роботи є програмне забезпечення API порталу доступу та обробки звукової інформації.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд аналогів та прототипів, схожих сервісів. Виділення їх переваг та недоліків.

– Дослідження API порталу доступу та обробки звукової інформації.

– Програмна реалізація API порталу доступу та обробки звукової інформації.

Практична цінність отриманих результатів полягає в тому, що розроблене програмне забезпечення дозволяє безпечно та ефективно обробляти звукову інформацію в локальному середовищі, без передачі даних на зовнішні сервери. Це дає змогу знизити ризики витоку інформації та протидіяти використанню аудіоданих у розвідці на основі відкритих джерел, особливо в умовах воєнного часу. API може бути інтегроване в мобільні або десктопні застосунки, внутрішні системи організацій, а також використане у навчальних та дослідницьких цілях. Рішення є універсальним і придатним для масштабування під конкретні потреби користувачів.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення API порталу доступу та обробки звукової інформації, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Призначенням розробленої системи є забезпечення безпечного та ефективного доступу до функціоналу обробки звукової інформації через спеціалізований API. Вся обробка відбувається локально, без використання сторонніх хмарних сервісів, що гарантує високий рівень безпеки та захисту даних.

Особливе значення система має в контексті протидії загрозам, пов’язаним із використанням відкритих джерел розвідки (OSINT), коли звукова інформація може бути використана для збору розвідувальних даних. Забезпечення локальної обробки аудіо з мінімальним ризиком витоку дає змогу використовувати систему в умовах підвищених вимог до безпеки, наприклад, у державних установах, військових структурах або організаціях критичної інфраструктури. Це дозволяє контролювати весь ланцюг обробки звуку — від моменту отримання до збереження чи передачі результату.

Таким чином, призначення системи полягає у створенні надійного, масштабованого та адаптивного рішення для обробки звукової інформації з урахуванням актуальних вимог сучасного українського суспільства.

1.2 Область застосування

Область застосування програмного забезпечення API порталу доступу та обробки звукової інформації охоплює широкий спектр завдань, пов’язаних із безпечною та гнучкою роботою зі звуковими даними. Система може використовуватися в організаціях, де обробка аудіо повинна здійснюватися виключно локально, без залучення сторонніх сервісів — зокрема, в державних

установах, військових підрозділах, підприємствах критичної інфраструктури та аналітичних центрах. Її можливості з імітування звучання в різних умовах та фільтрації аудіо також є корисними для навчальних та тренувальних симуляторів, зокрема в оборонній сфері, де важливо відтворити акустичну ситуацію максимально наближено до реальної.

Також система може бути інтегрована в мобільні застосунки, десктопні програми та веб-інтерфейси, де потрібна обробка звукової інформації — наприклад, у журналістиці, розслідуваннях, підготовці подкастів, телекомунікаціях або дистанційній освіті. Для наукових та освітніх установ API стане корисним інструментом у дослідженнях, що стосуються акустики, цифрової обробки сигналів або мовних технологій. Програмне забезпечення має широку сферу застосування як у безпековому, так і в цивільному, науковому та комерційному контекстах.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення API порталу доступу та обробки звукової інформації, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

Krisp.ai – це платформа, яка надає інструменти на основі штучного інтелекту для підвищення якості звуку та продуктивності під час онлайн-зустрічей і дзвінків. Вона зосереджена на шумозаглушенні, транскрипції зустрічей і підвищенні чіткості голосу, орієнтована на професіоналів, віддалених працівників і команди.

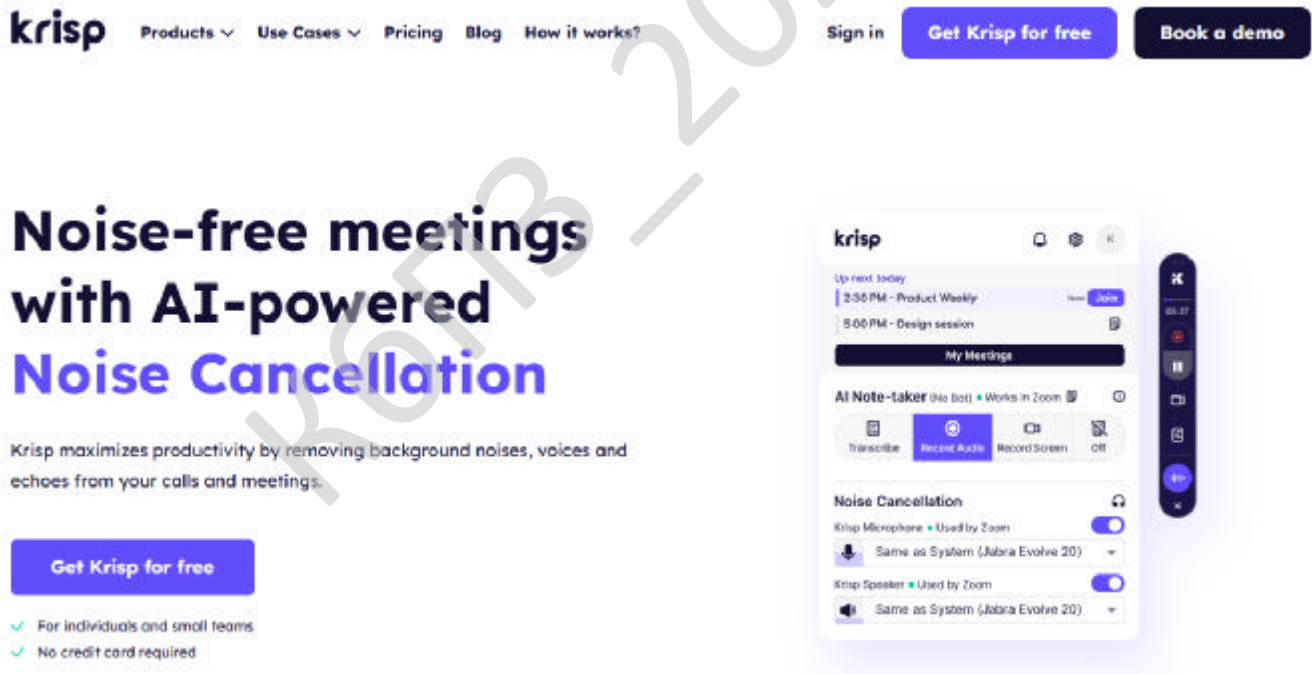

Рисунок 2.1 – Інформаційна сторінка Krisp.ai про послугу шумозаглушення

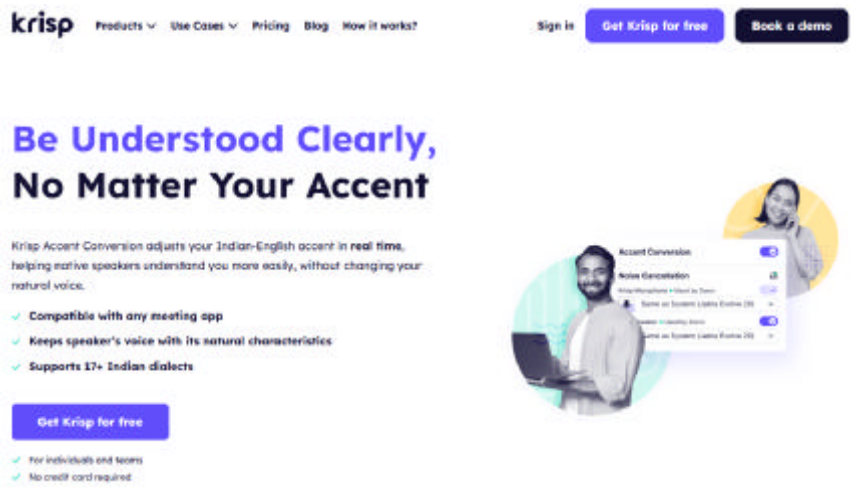

Рисунок 2.2 – Інформаційна сторінка Krisp.ai про послугу локалізації акценту

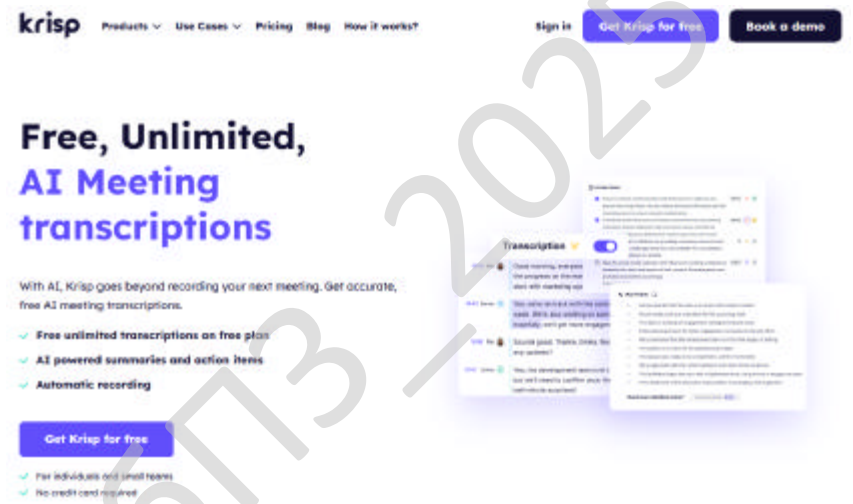

Рисунок 2.3 – Інформаційна сторінка Krisp.ai про послугу транскрипції зустрічей

Основні функції Krisp.ai:

– Шумозаглушення: Прибирає фонові шуми, такі як рух транспорту, гавкіт собак або клацання клавіатури, пропонує двонаправлене шумозаглушення, фільтруючи шум з обох кінців розмови.

– Локалізація акценту: Адаптується до різних акцентів для кращого розуміння.

– Допомога під час нарад: Забезпечує транскрипцію нарад у реальному

часі та автоматичне створення резюме, розрізняє мовців і приписує мовлення потрібним особам, створює стислі нотатки про нараду для легкого обміну та перегляду.

– Придушення відлуння: Усуває зациклення звуку між мікрофонами та динаміками.

– Покращення чіткості голосу: Покращує якість мовлення під час дзвінків для чіткого спілкування.

Adobe Podcast – це хмарна платформа зі штучним інтелектом, призначена для запису, редагування та покращення аудіо, спеціально розроблена для створення подкастів. Вона працює повністю через веб-браузер і легко інтегрується з іншими інструментами Adobe, такими як Audition і Premiere Pro. Платформа надає зручні функції для створення аудіоконтенту професійної якості, не вимагаючи складного обладнання чи досвіду.

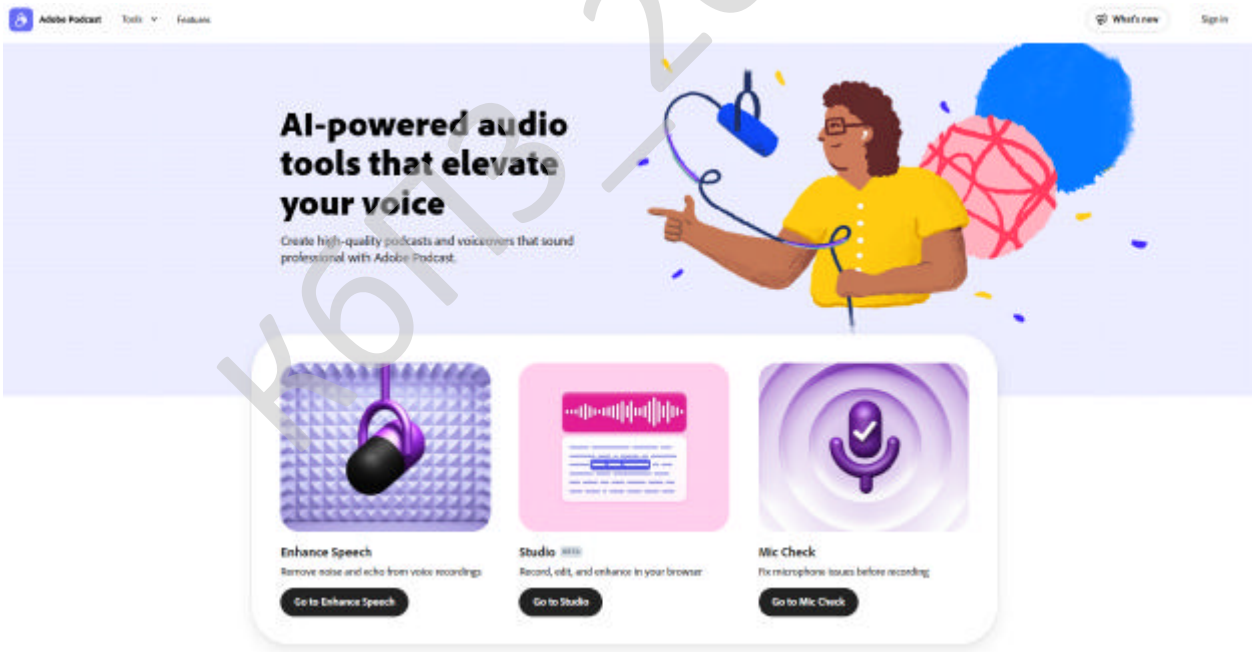

Рисунок 2.4 – Інформаційна сторінка Adobe Podcast про доступні послуги

Основні функції Adobe Podcast:

– Enhance Speech: Використовує ШІ для покращення чіткості звуку, видаляючи фоновий шум і відлуння, збалансовує нерівномірні рівні звуку та відновлює чіткість приглушених записів.

– Studio: Пропонує робочу область у браузері для запису, редагування та покращення звуку, дозволяє записувати кілька доріжок, додавати музику або звукові ефекти, а також налаштовувати рівні звуку.

– Mic Check: Аналізує налаштування мікрофона, щоб забезпечити оптимальну якість звуку перед записом, Надає зворотний зв'язок щодо відстані, рівнів підсилення, фонового шуму та відлуння.

Audio Strip – це веб-інструмент, який використовує технології штучного інтелекту та глибокого навчання для виділення вокалу та відокремлення звукових доріжок з пісень або аудіофайлів. Він призначений для музикантів, діджеїв, продюсерів та аудіо-ентузіастів, пропонуючи безкоштовні та преміум-плани з різним рівнем функціональності. Платформа спрощує завдання маніпулювання аудіо, роблячи її доступною для користувачів без просунутих технічних навичок.

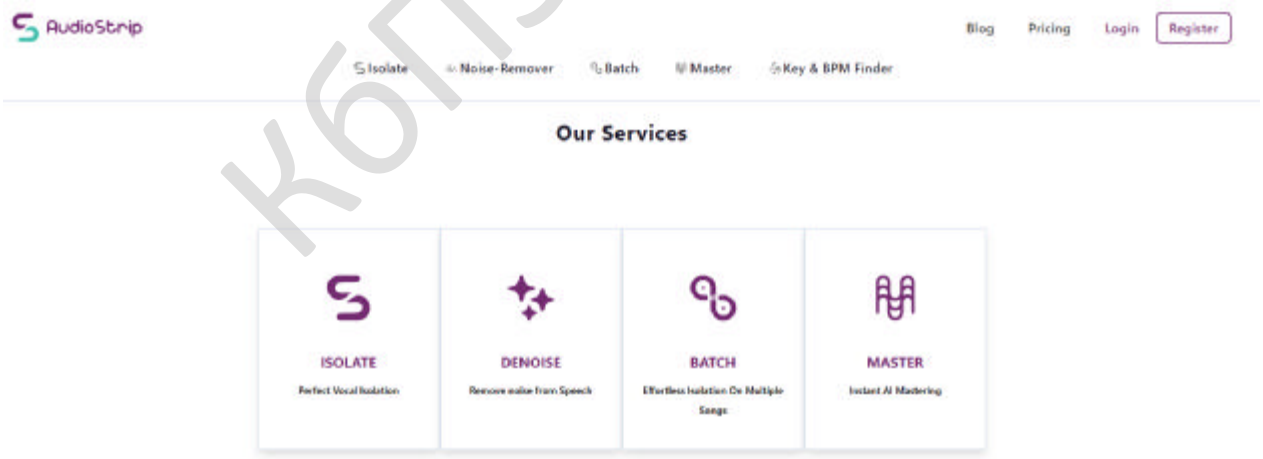

Рисунок 2.5 – Сторінка сервісів, які надає платформа Audio Strip

Основні функції Audio Strip :

– Ізоляція вокалу: Виділення вокалу з фонограми для створення

акапельної версії або ізоляції інструментальних партій для реміксування та семплювання.

– Видалення шуму: Видаляє небажані фонові шуми для покращення чіткості мовних або музичних записів.

– Пакетна обробка: Дозволяє користувачам обробляти кілька аудіофайлів одночасно, заощаджуючи час для професіоналів, які працюють з великими колекціями.

– Мастеринг зі штучним інтелектом: Використовує технологію штучного інтелекту для швидкого та професійного мастерингу оброблених аудіодоріжок.

Voicemod – це платформа для зміни голосу в реальному часі, призначена для користувачів Windows і macOS. Вона дозволяє користувачам змінювати свої голоси та додавати звукові ефекти під час живих розмов, ігор, потокового мовлення або створення контенту. Програмне забезпечення легко інтегрується з такими популярними платформами, як Discord, Twitch, Skype, Zoom та різноманітними іграми, що робить його універсальним інструментом для покращення звукового досвіду.

Основні функції Voicemod:

– Змінювач голосу в реальному часі: Пропонує понад 150 голосових фільтрів (наприклад, робота, бурундука, голоси знаменитостей) для миттєвого перетворення голосу користувача під час спілкування в реальному часі або під час запису.

– Придушення шуму та покращення голосу: Вбудовані інструменти для зменшення фонового шуму та покращення якості звуку з мікрофона для чіткої комунікації.

– Голоси зі штучним інтелектом: Дозволяє використовувати голоси, згенеровані штучним інтелектом, навчені професійними акторами для створення високоякісних ефектів.

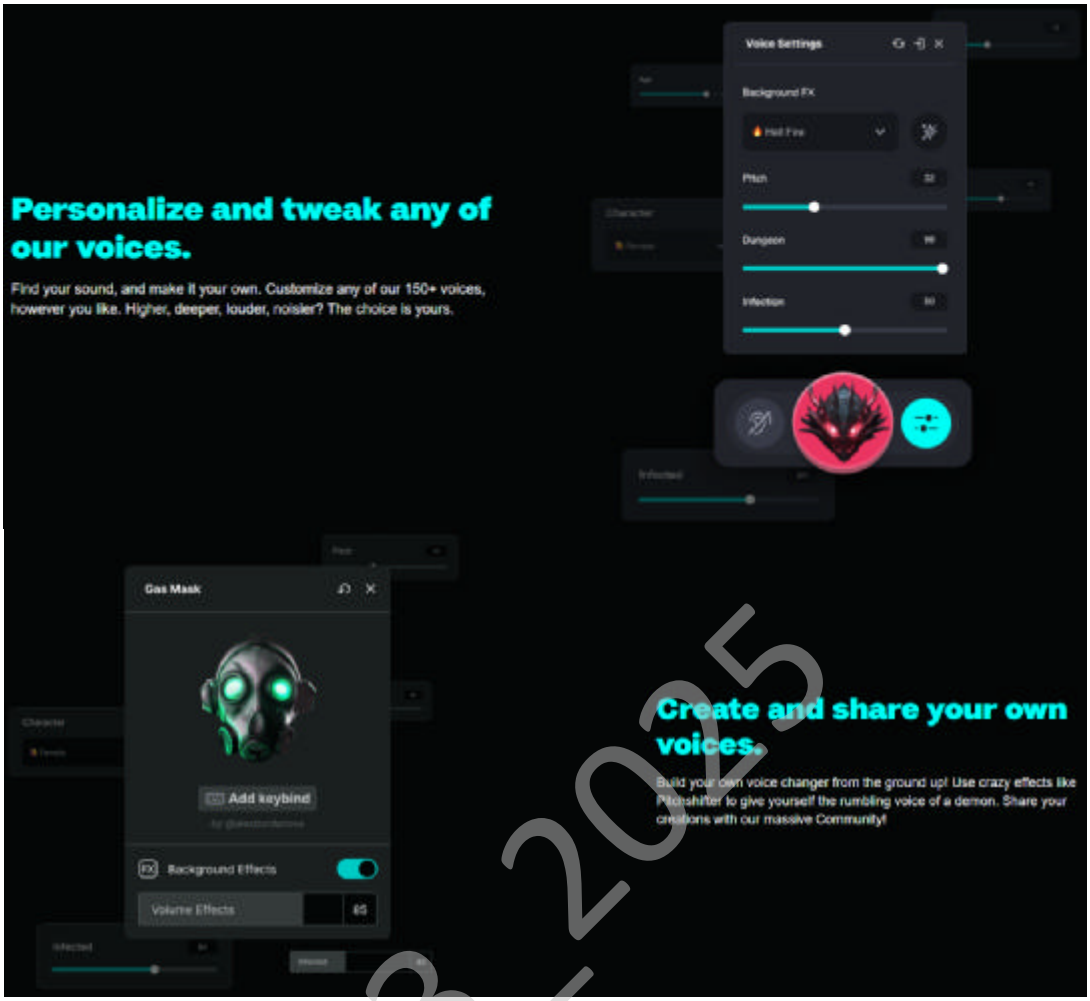

Рисунок 2.6 – Інформаційна сторінка Voicemod про деякі з доступних послуг

Cleanvoice AI – це передова платформа для редагування аудіо, призначена для автоматизації процесу очищення та покращення розмовних аудіозаписів. Вона використовує штучний інтелект для виявлення та усунення недоліків, заощаджуючи час і зусилля користувачів на постпродакшені. Платформа орієнтована насамперед на людей, що ведуть подкасти, творців контенту, бізнесменів та професіоналів, які працюють з аудіо.

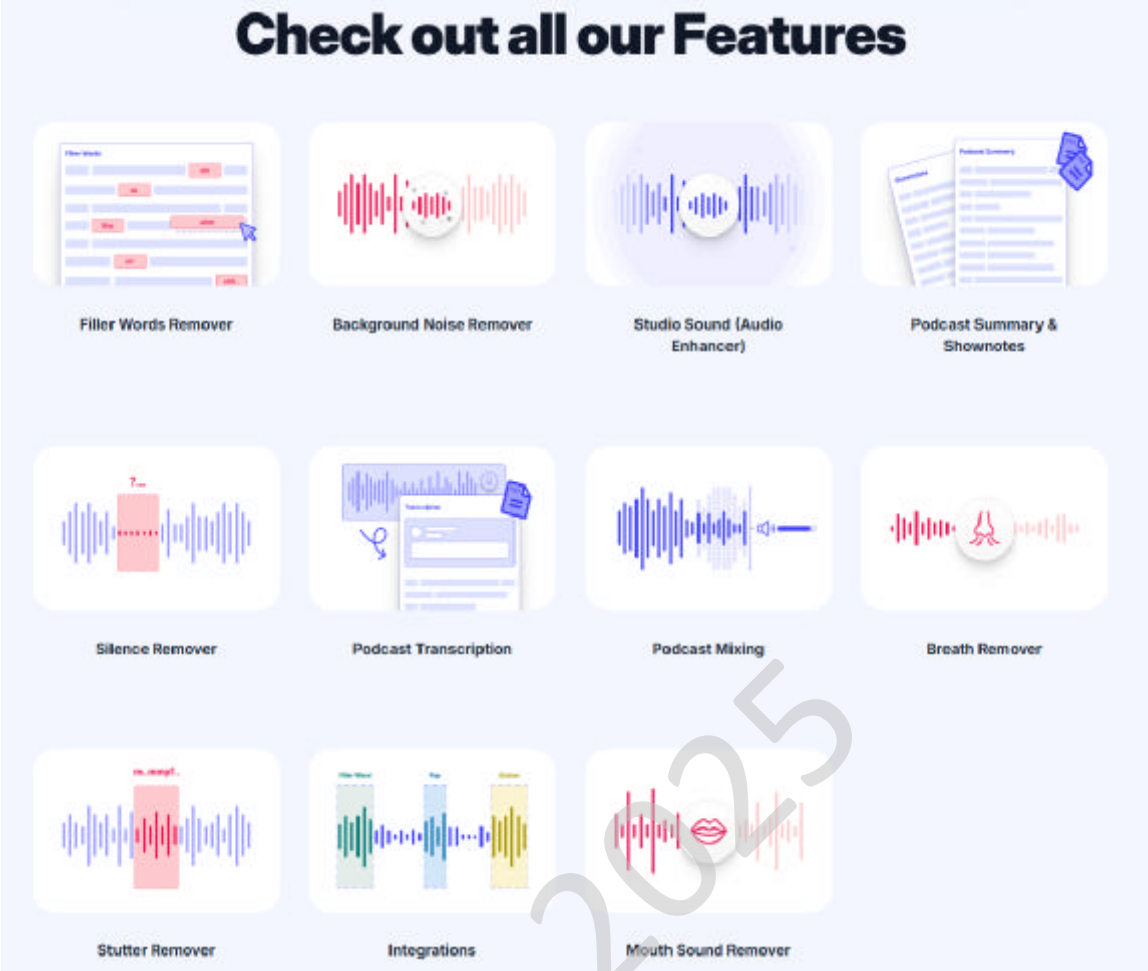

Рисунок 2.7 – Сторінка сервісів, які надає платформа Cleanvoice AI

Основні функції Cleanvoice AI:

– Видалення вставних слів: Автоматично виявляє та видаляє слова-заповнювачі (“а”, “гм”, “як”, “типу” тощо), забезпечуючи плавність мовленнєвого потоку.

– Придушення фонового шуму: Усуває небажані звуки, такі як гудіння, шипіння, шум транспорту та відлуння в приміщенні, для покращення якості звуку.

– Студійний звук (покращення аудіо): Відновлює частоти, усуває спотворення і реверберацію, а також застосовує автоматичне еквалізацію для професійної якості звуку.

– Виправлення заїкання та дублювання слів: Згладжує повторювані фрази або заїкання, щоб зробити мову більш зв’язною.

– Видалення звуків з рота: Видаляє звуки слини, плямкання губ та клацання, щоб отримати відшліфований запис.

Також Cleanvoice AI надає API, який дозволяє компаніям автоматизувати та масштабувати процеси редагування аудіо. Цей API особливо корисний для компаній, які займаються великомасштабним виробництвом подкастів або іншим розмовним контентом.

Шлях до сторінки ознайомлення з API (рис. 2.9) знаходиться у меню головної сторінки веб-сервісу (рис. 2.8).

Рисунок 2.8 – Місцезнаходження сторінки ознайомлення з функціоналом API

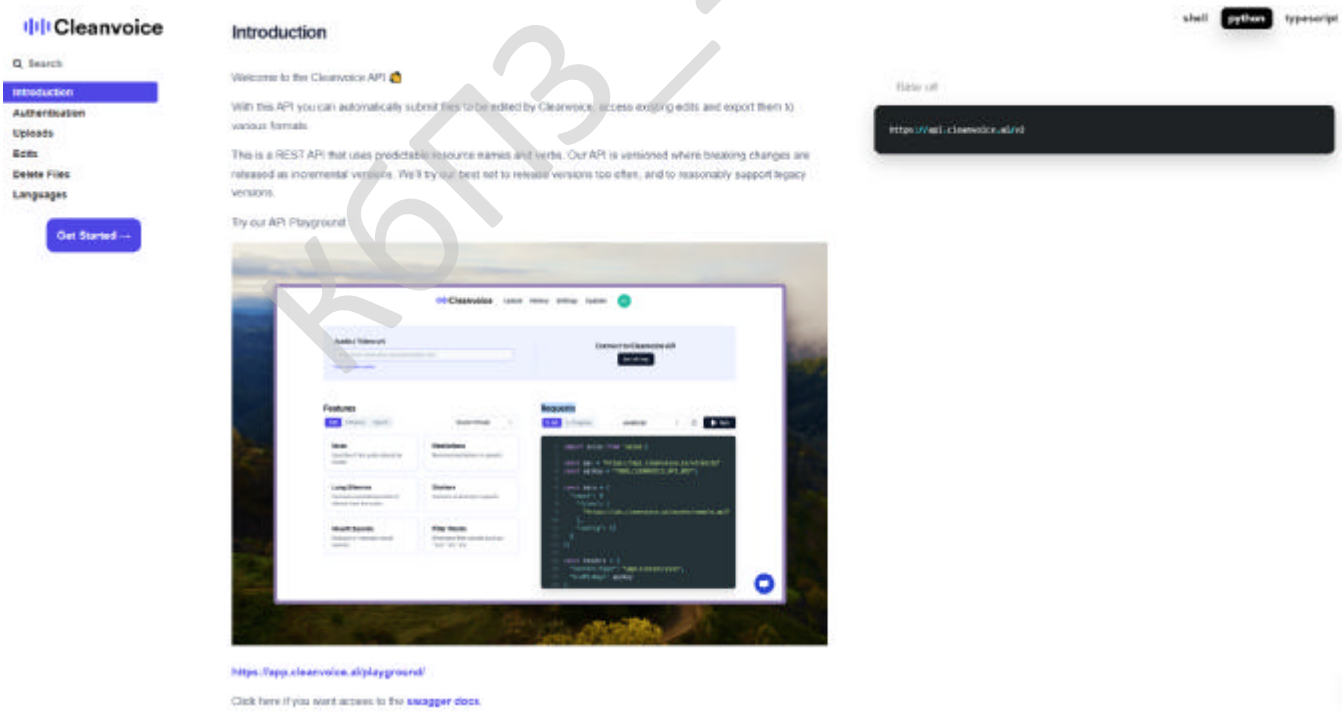

Рисунок 2.9 – Сторінка ознайомлення з API веб-сервісу Cleanvoice AI

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Серверна частина програмного забезпечення написане мовою Python. Ця мова обрана виходячи з наступних міркувань. Python – це високорівнева мова програмування загального призначення, відома своєю простотою та універсальністю. Вона підтримує декілька парадигм програмування, включаючи процедурне, об’єктно-орієнтоване та функціональне програмування та має динамічну типізацію. Всеосяжна стандартна бібліотека Python заслужила прізвисько “мова з батарейками в комплекті”.

Python широко використовується в різних галузях, таких як веб-розробка, data science, машинне навчання, автоматизація тощо. Її динамічна семантика та вбудовані структури даних роблять її ідеальною для швидкої розробки додатків.

Python вважається однією з найпростіших мов для вивчення завдяки кільком факторам:

- Читабельний синтаксис: Синтаксис мови дуже схожий на англійську, що зменшує когнітивне навантаження для початківців.
- Мінімум шаблонного коду: Завдання, які в інших мовах вимагають багатослівного коду, в Python спрощуються.
- Динамічна типізація: Розробникам не потрібно явно оголошувати типи змінних, що робить кодування швидшим і менш схильним до помилок.
- Широка документація та підтримка спільноти: Python має велику спільноту та безліч ресурсів для навчання.

Синтаксис Python відомий своєю простотою та читабельністю, що робить його доступним як для початківців, так і для досвідчених розробників. По суті, Python використовує відступи для визначення блоків коду, замість того, щоб покладатися на фігурні дужки або інші символи, які часто зустрічаються в таких мовах, як Java або C++. Такий підхід не тільки забезпечує узгоджене форматування, але й покращує візуальну чіткість коду. Кількість пробілів, що

використовуються для відступів, є гнучкою, але узгодженість всередині блоку є обов'язковою. Будь-яке відхилення призводить до синтаксичних помилок, що підкреслює прихильність Python до чистого та структурованого коду.

Змінні в Python динамічно типізуються, тобто їх тип виводиться з присвоєного їм значення, і явні оголошення не потрібні. Така динамічна типізація дозволяє розробникам писати лаконічний та адаптивний код, не турбуючись про суворі визначення типів. Простий синтаксис мови зменшує складність роботи з різноманітними структурами даних.

Використання англійських ключових слів замість символів значно покращує читабельність мови Python. Такі ключові слова, як if, else, for і while використовуються для управління потоком, роблячи намір коду зрозумілим навіть для тих, хто не знайомий з програмуванням. Функції визначаються за допомогою ключового слова def, за яким слідує ім'я функції та круглі дужки, що містять необов'язкові параметри. Крім того, синтаксис коментарів Python, що використовує хеш-символ (#), дозволяє розробникам ефективно документувати свій код, не захаращуючи логіку.

Обробка помилок у Python спрощена за допомогою винятків, які надають детальний зворотній зв'язок про проблеми, що виникають під час виконання. Мова включає в себе надійні засоби налагодження в стандартній бібліотеці, що дозволяє розробникам ефективно виявляти та вирішувати проблеми. Філософія проектування Python наголошує на "єдиному очевидному способі зробити це", що мінімізує неоднозначність та сприяє узгодженості між проектами. Цей принцип гарантує, що код на Python залишається доступним і підтримуваним, водночас задовольняючи широкий спектр парадигм програмування, таких як процедурне, об'єктно-орієнтоване та функціональне програмування.

Однією з найбільших переваг Python у розробці API є широка підтримка бібліотек. Такі фреймворки, як Flask та Django REST Framework, надають потужні інструменти для створення API з мінімальними зусиллями. Flask особливо популярний для легких модульних додатків, в той час як Django REST

Framework відмінно підходить для створення складних API з вбудованими функціями, такими як серіалізація та автентифікація. Інші фреймворки, такі як FastAPI, з'явилися як високопродуктивні варіанти, що використовують асинхронні можливості Python для ефективної обробки великомасштабних запитів до API. Ці інструменти спрощують виконання поширених завдань, таких як обробка HTTP-запитів, розбір JSON-даних та керування взаємодією з базами даних.

Python також добре підходить для роботи з сучасними веб-технологіями та форматами даних. Він має вбудовану підтримку JSON, який є найпоширенішим форматом для обміну даними в API. Це дозволяє легко створювати RESTful API, які ефективно взаємодіють між клієнтами та серверами. Крім того, сумісність Python з хмарними платформами спрощує розгортання та масштабування API. Багато хмарних провайдерів пропонують власну підтримку для додатків на Python, що полегшує інтеграцію API у масштабовані архітектури.

Активна спільнота розробників Python ще більше заохочує нову аудиторію для розробки API. Завдяки великій кількості навчальних посібників, документації та проектів з відкритим вихідним кодом, розробники можуть швидко знайти рішення для проблем, з якими вони стикаються. Така підтримка спільноти гарантує, що Python залишається в курсі останніх тенденцій у розробці API та найкращих практик безпеки.

Архітектура платформи Visual Studio Code

Програма мовою Python виконується в середовищі Visual Studio Code. Архітектура VS Code є модульною та розширюваною, розробленою на основі системи плагінів, що дозволяє легко додавати нові функції та підтримувати різні мови програмування. Ядро VS Code написано на TypeScript, типізованій підмножині JavaScript, яка забезпечує потужну перевірку типів та об'єктно-орієнтоване програмування. Це ядро відповідає за основну функціональність редактора, включаючи керування файлами, редагування тексту та користувацький інтерфейс.

Коли справа доходить до компіляції коду Python у Visual Studio Code, цей процес спрощується та інтегрується в середовище розробки. VS Code не компілює код Python безпосередньо, оскільки Python є інтерпретованою мовою. Натомість він надає зручний інтерфейс для запуску Python-скриптів та керування середовищами Python.

Щоб запускати код на Python у VS Code, користувачі зазвичай встановлюють розширення Python, яке додає комплексну підтримку мови Python. Це розширення інтегрується з інтерпретаторами Python, встановленими в системі користувача.

Після запуску коду VS Code передає файл Python обраному інтерпретатору. Інтерпретатор читає код Python, переводить його в байт-код, а потім виконує цей байт-код рядок за рядком. Під час цього процесу інтегрований термінал VS Code відображає результати роботи скрипта, включаючи будь-які оператори виводу на друк або повідомлення про помилки.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення API порталу доступу та обробки звукової інформації, яке забезпечує локальну, гнучку та безпечну обробку аудіосигналів з можливістю імітування різних акустичних умов. Система розробляється з урахуванням підвищених вимог до інформаційної безпеки, актуальних в умовах війни в Україні, зокрема задля запобігання витоку звукових даних через OSINT-інструменти.

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести критичний аналіз існуючих рішень у сфері обробки звукової інформації з відкритим або локальним API-доступом. Виявити їхні сильні

сторони та недоліки, зокрема в аспекті безпеки, універсальності та функціональності. Результати аналізу використати при формуванні технічних вимог до системи;

б) обґрунтувати вибір архітектури системи, яка включає клієнтську та серверну частини, а також визначити методи реалізації обробки аудіо з використанням сучасних бібліотек для цифрової обробки сигналів. Побудувати структурну та функціональну схеми системи;

в) реалізувати серверну частину API, що забезпечить обробку аудіо за заданими параметрами, та клієнтську частину, яка дозволить зручно взаємодіяти з користувачем і надсилати запити до сервера. Побудувати блок-схеми основних алгоритмів обробки звуку, розробити логіку взаємодії компонентів системи;

г) організувати інтерфейс користувача на клієнтській стороні, який дозволить обирати режими обробки, завантажувати та отримувати результати. Передбачити інформування користувача про помилки введення, збої під час обробки або некоректні параметри запиту;

д) розробити рекомендації щодо розгортання програмного забезпечення в захищеному середовищі, зокрема для потреб державних організацій або критичної інфраструктури. Надати методичні вказівки щодо адаптації API до інших програмних комплексів;

е) сформувати висновки щодо виконаного обсягу робіт, ефективності реалізованої системи, її придатності до впровадження в умовах реального використання, а також перспектив подальшого розвитку.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Функціонування системи “Програмне забезпечення API порталу доступу та обробки звукової інформації” базується на взаємодії клієнтської та серверної частин. Існує багато різноманітних протоколів, за допомогою яких створюють API, але в основному виділяють два найбільш популярних: RESR та SOAP.

REST (Representational State Transfer API) – це архітектурний стиль для створення вебсервісів, який базується на використанні стандартних HTTP-методів (GET, POST, PUT, DELETE) для роботи з ресурсами, представленими у вигляді URL-адрес. Основна ідея REST полягає у розділенні клієнта і сервера, де клієнт надсилає запит на певний ресурс, а сервер повертає відповідь із необхідними даними. Запити та відповіді є самодостатніми, тобто містять усю інформацію, необхідну для їх обробки, що робить REST API простим у використанні, масштабованим і гнучким для інтеграції з різними системами.

Серед основних переваг REST API – простота та інтуїтивність структури, що базується на загальновідомих HTTP-методах, а також висока масштабованість завдяки запитам без стану (stateless). REST API легко інтегруються з різними платформами й мовами програмування, підтримують роботу з різними форматами даних (наприклад, JSON, XML) і мають ефективні механізми кешування для підвищення продуктивності. Важливою перевагою є також широке розповсюдження та підтримка у сучасних фреймворках і інструментах розробки.

Проте REST API мають і певні недоліки. Однією з проблем є так зване “over-fetching” або “under-fetching” даних – коли клієнт отримує більше або менше інформації, ніж потрібно, що може призводити до зайвого навантаження на мережу чи необхідності робити кілька запитів для отримання повної інформації. Також REST API можуть бути менш гнучкими для складних

операцій, які потребують виконання багатьох взаємопов'язаних дій. Дизайн REST API для великих або складних систем може бути непростим, а зміни у структурі даних можуть впливати на сумісність із клієнтами.

SOAP (Simple Object Access Protocol API) – це протокол для обміну структурованими повідомленнями між різними програмними системами через мережу, який базується на використанні XML для форматування даних. SOAP був розроблений компанією Microsoft як стандарт для побудови надійних і сумісних вебсервісів, особливо у корпоративному середовищі. Всі запити і відповіді у SOAP API обгортаються у спеціальні "конверти" (envelopes), які містять як основні дані, так і додаткову службову інформацію, наприклад, дані автентифікації або версію протоколу.

Основними перевагами SOAP API є висока надійність і стандартизація, що робить його ідеальним для великих корпоративних систем, де потрібна чітка структура і сувора перевірка даних. SOAP підтримує розширені механізми безпеки, зокрема WS-Security для шифрування та автентифікації, що важливо для фінансових, банківських та інших чутливих застосувань. Протокол також є незалежним від мови програмування і платформи, що дозволяє інтегрувати різні системи незалежно від їх технологічної основи. Ще однією перевагою є вбудована обробка помилок, яка дозволяє швидко діагностувати проблеми у комунікації між сервісами.

До недоліків SOAP API належить складність у впровадженні та підтримці, оскільки робота з XML вимагає більше ресурсів і знань, ніж із простішими форматами на кшталт JSON. Через обов'язкове використання XML і великої кількості службової інформації SOAP-повідомлення часто мають великий розмір, що негативно впливає на швидкість передачі даних і навантаження на мережу. SOAP API не підтримує кешування запитів, що може знижувати продуктивність у порівнянні з REST. Формат також менш інтуїтивний для розробників і не підходить для безпосередньої роботи у браузерах, оскільки вимагає спеціалізованих клієнтів для обробки SOAP-повідомлень.

У підсумку, SOAP API є актуальним вибором для складних, захищених і стандартизованих інтеграцій у корпоративних та фінансових системах, однак для більшості сучасних вебсервісів і мобільних додатків часто обирають легші та гнучкіші альтернативи, такі як REST API. З цих причин серверна частина буде реалізована з допомогою REST API.

З боку користувача (клієнта) взаємодія починається з інтерфейсу, через який він має змогу завантажити звуковий файл, обрати потрібний тип обробки (наприклад, реверберація, еквалізація, імітація простору тощо) та вказати додаткові параметри, які визначають умови імітації або трансформації аудіо. Клієнтська частина може бути реалізована у вигляді мобільного застосунку, веб-інтерфейсу або десктопної програми — зручною для інтеграції в робочі процеси кінцевого користувача.

Після вибору параметрів клієнтська частина формує HTTP-запит до API, який надсилає звукові дані та відповідні метадані на сервер. Серверна частина програмного забезпечення виконує попередню перевірку запиту, верифікує формат та структуру даних, а далі передає аудіо на обробку за допомогою вбудованих аудіоалгоритмів.

Для передачі звукової інформації між клієнтом і сервером обрано стандартні аудіоформати, що забезпечують сумісність та зручність обробки. Основним є формат WAV (PCM-кодування), який використовується для обробки без втрати якості. У випадках, коли потрібна менша вага файлів, можна застосовувати формат MP3 або OGG, однак лише після завершення обробки, щоб уникнути кумулятивних втрат якості при багаторазовому стисканні.

WAV (Waveform Audio File Format) – це один із найпоширеніших аудіоформатів, розроблений компаніями Microsoft та IBM у 1991 році. Він базується на форматі RIFF (Resource Interchange File Format) і призначений для зберігання оцифрованих аудіоданих. Основною особливістю WAV є те, що зазвичай він зберігає звук у безстислому вигляді, найчастіше у форматі PCM (імпульсно-кодова модуляція), що дозволяє зберегти високу якість аудіо без

втрат. Завдяки цьому WAV-файли часто використовують у професійному звукозаписі, редагуванні аудіо та архівуванні звуку.

У WAV-файлах містяться не лише аудіодані, а й метадані, які можуть включати інформацію про частоту дискретизації, бітову глибину та кількість каналів. Частота дискретизації визначає, скільки разів за секунду відбувається вимірювання звуку (зазвичай 44,1 кГц або 48 кГц), а бітова глибина впливає на динамічний діапазон і якість відтворення (8, 16, 24 або 32 біти). Кількість каналів може бути моно, стерео або багатоканальною, що використовується у професійних аудіосистемах. Ці параметри роблять WAV гнучким форматом для різних потреб.

Головним недоліком формату WAV є великий розмір файлів, оскільки дані зберігаються без стиснення. Наприклад, аудіо у якості CD (44,1 кГц, 16 біт, стерео) займає близько 172 КБ на секунду. Через це WAV-файли менш зручні для зберігання і передачі у порівнянні зі стисненими форматами, такими як MP3 або AAC. Водночас, завдяки простоті структури і широкій підтримці, WAV залишається стандартом для обробки та редагування аудіо, а також для збереження оригінальних записів без втрат.

MP3 (MPEG-1 Audio Layer III) – це один із найпопулярніших аудіоформатів із стисненням з втратами, розроблений групою Moving Picture Experts Group (MPEG) у 1990-х роках. Формат дозволяє значно зменшити розмір аудіофайлу за рахунок видалення частини звукової інформації, яка, згідно з психоакустичною моделлю, є менш помітною для людського вуха. Це досягається шляхом усунення звуків, які знаходяться поза межами сприйняття або маскуються більш гучними сигналами, а також за рахунок спектрального стиснення. Завдяки цьому MP3-файли можуть бути у 10-11 разів меншими за оригінальні аудіозаписи у форматі CD, при цьому зберігаючи прийнятну якість для більшості користувачів.

Основним параметром, що впливає на якість звуку в MP3, є бітрейт - кількість кілобітів інформації, що передається за секунду. Він може варіюватися від 32 до 320 кбіт/с, де вищий бітрейт забезпечує кращу якість, але й більший

розмір файлу. Наприклад, 128 кбіт/с вважається стандартним для звичайного прослуховування, а 320 кбіт/с - для аудіофілів або якісного відтворення на хорошій апаратурі. При низьких бітрейтах якість звуку помітно погіршується, особливо на складних музичних композиціях або при прослуховуванні на професійному обладнанні.

Технічно MP3 розбиває звуковий сигнал на короткі часові відрізки, які потім аналізуються і кодуються окремо. Алгоритм використовує ефект маскування - коли гучніші звуки приховують тихіші, що дозволяє відкидати непомітні для слухача частоти. Після цього дані стискаються і упаковуються у фрейми, які складають MP3-файл. Такий підхід дозволяє гнучко регулювати якість і розмір файлу, а також підтримує різні частоти дискретизації (зазвичай 32 000, 44 100 або 48 000 Гц).

OGG (Формат OGG, точніше Ogg Vorbis) – це відкритий і безкоштовний аудіоформат зі стисненням з втратами, розроблений компанією Xiph.Org Foundation і офіційно випущений у 2002 році. Він призначений для зберігання аудіоданих із високою якістю при відносно невеликому розмірі файлу. На відміну від MP3, який у свій час був захищений патентами, Ogg Vorbis створювався як вільна альтернатива, що не має ліцензійних обмежень.

OGG використовує власну психоакустичну модель для стиснення звуку, що дозволяє досягати високої якості відтворення при змінному бітрейті, який може варіюватися від дуже низьких значень до понад 700 кбіт/с. Формат підтримує широкий діапазон частот дискретизації - від 2 кГц до 192 кГц, а також може кодувати до 255 аудіоканалів із розрядністю до 32 біт, що робить його придатним навіть для багатоканального звуку, наприклад, у DVD-Audio.

Хоча OGG має переваги у якості і гнучкості порівняно з MP3, він менш поширений і підтримується не всіма пристроями так широко. Проте формат активно використовується у потокових сервісах, файлообмінних мережах і серед аудіофілів, які цінують відкритість і якість звуку без ліцензійних обмежень.

У залежності від вибраного типу обробки, сервер застосовує необхідні ефекти та фільтри, використовуючи різноманітні бібліотеки (вбудовані або

зовнішні) або власні DSP-реалізації, після чого формує результат у вигляді нового аудіофайлу або масиву даних.

Додаткові параметри, що описують запит на обробку (тип ефекту, значення параметрів тощо), передаються у форматі метаданих.

Метадані – це структуровані допоміжні дані, які описують параметри та налаштування для обробки аудіосигналу. Вони не містять сам звуковий контент, але визначають, як саме цей контент має бути змінений. Наприклад, якщо користувач обирає ефект реверберації, до метаданих включаються параметри на кшталт часу затухання, типу кімнати чи рівня змішування з оригінальним сигналом.

Метадані формуються на стороні клієнта у зручному машинозчитуваному форматі. Ці дані прикріплюються до основного запиту разом із аудіофайлом і надсилаються на сервер. Сервер, отримавши запит, інтерпретує метадані для визначення алгоритму обробки: який ефект застосувати, у якій послідовності, з якими значеннями параметрів тощо. Таким чином, система забезпечує гнучке налаштування обробки без потреби ручного втручання у код.

Використання метаданих дозволяє легко серіалізувати та десеріалізувати дані на боці як клієнта, так і сервера, стандартизувати передачу налаштувань, масштабувати систему під різні типи обробки та зручно розширювати функціональність у майбутньому. Завдяки цьому підходу, API може бути універсальним інструментом, здатним працювати з широким спектром аудіообробок, забезпечуючи точне та контрольоване перетворення сигналів.

Оброблений аудіофайл повертається на клієнтську частину як відповідь сервера. Користувач отримує можливість прослухати результат, завантажити його або надіслати на подальше опрацювання.

У випадку інтеграції з іншими системами (наприклад, журналістським редактором чи внутрішнім військовим архівом), API дозволяє автоматично передавати результат у потрібне середовище або зберігати його у відповідному сховищі. Усе це відбувається без необхідності підключення до сторонніх хмарних

сервісів, що зменшує ризики витоку даних та покращує захист конфіденційної інформації.

Таким чином, користувач сприймає систему як інтуїтивно зрозумілий інструмент для швидкої та безпечної обробки звукової інформації. Завдяки чітко розмежованій архітектурі клієнт-сервер, програмне забезпечення легко масштабувати, розгортати в локальних мережах, а також адаптувати під конкретні потреби – наприклад, додати нові типи обробки або інтегрувати з інструментами аналітики. Це робить систему не лише зручною, а й універсальною платформою для роботи з аудіо в умовах сучасних інформаційних викликів.

3.2 Розробка структурної схеми

Розробка структурної схеми є ключовим етапом проєктування системи, оскільки вона дозволяє візуалізувати основні компоненти програмного забезпечення, їх функціональне призначення та взаємозв'язки між ними. У контексті програмного забезпечення API порталу доступу та обробки звукової інформації, структурна схема демонструє розподіл обов'язків між клієнтською частиною (користувацький інтерфейс), серверною частиною (обробка запитів та аудіосигналів), блоком управління ефектами та сховищем даних.

Ця схема дає змогу визначити місце кожного модуля у загальній архітектурі, оцінити логіку взаємодії, а також врахувати можливості масштабування або заміни окремих блоків без порушення роботи всієї системи. Структурна схема також є важливою для комунікації між членами команди розробки та для майбутньої технічної документації, оскільки вона фіксує логіку побудови системи на етапі проєктування.

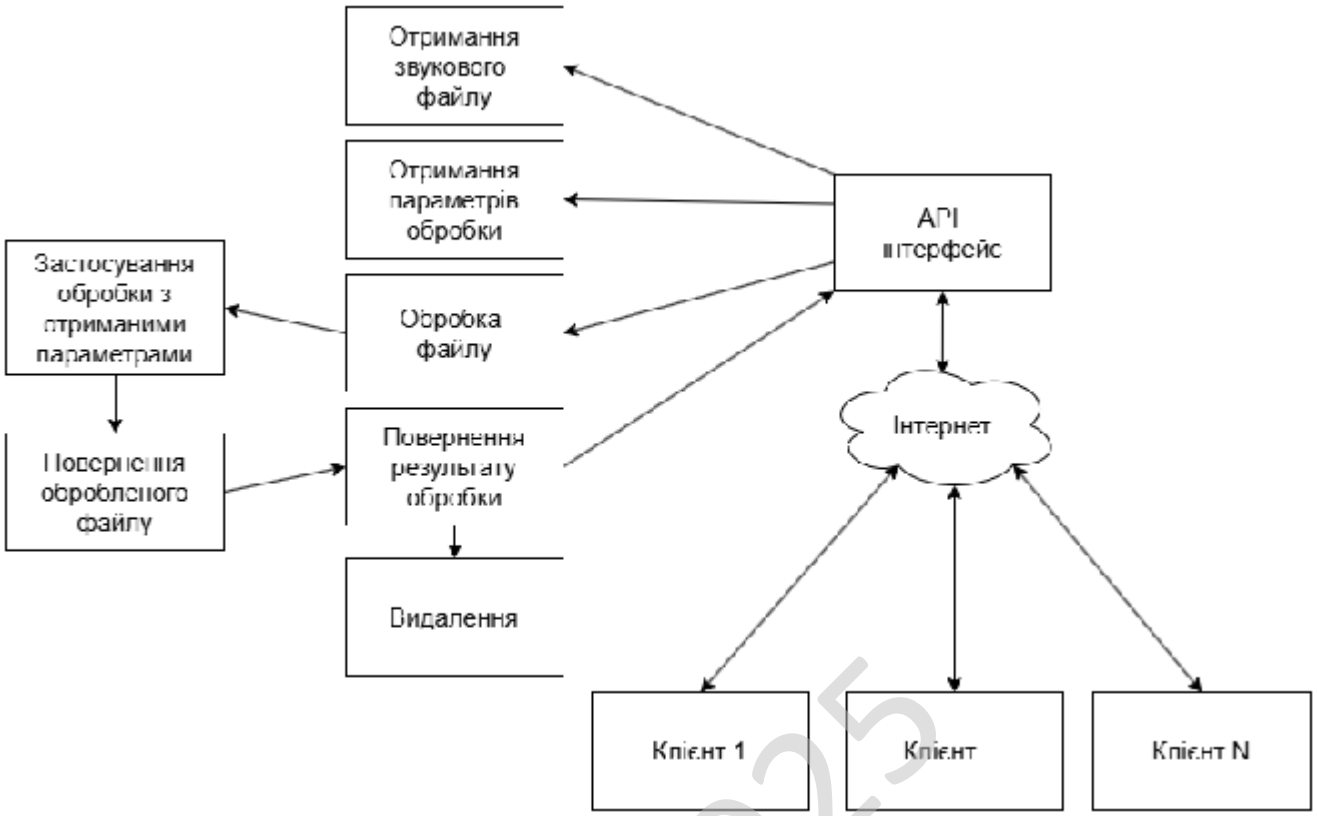

Рисунок 3.1 – Структурна схема програмного забезпечення

Клієнт 1 / Клієнт … / Клієнт N – це користувачі, які взаємодіють з системою через мобільний застосунок або веб-інтерфейс. Вони мають змогу завантажити аудіофайл, вибрати ефекти обробки, задати параметри та відправити все на сервер. Кожен клієнт може працювати незалежно, але з'єднання здійснюється централізовано через API.

Інтернет є транспортним середовищем, що забезпечує передачу даних між клієнтами та серверною частиною. Саме через інтернет надсилаються запити на обробку та повертаються результати у вигляді оброблених аудіофайлів. Надійність і швидкість цього каналу безпосередньо впливає на зручність використання системи.

API інтерфейс є центральним вузлом комунікації між клієнтами та внутрішніми компонентами обробки. Він приймає вхідні HTTP-запити, зчитує параметри, ініціює збереження аудіофайлу та передає все далі до обробки. Після

завершення обробки API повертає оброблений файл користувачеві. Архітектура API повинна бути масштабованою для підтримки високого навантаження.

Після надсилання клієнтом звукового файлу, відбувається його отримання (блок “Отримання звукового файлу”) та збереження у тимчасове сховище. Файл може бути у форматах WAV, MP3 тощо, але для обробки часто використовується WAV. Надалі файл використовується у процесі обробки.

Разом з файлом клієнт передає параметри ефекту у вигляді метаданих. Система отримує параметри, аналізує та перетворює у формат, який розуміє оброблюючий модуль. Можливість відсутності або некоректності параметрів має бути усунута починаючи з клієнтської частини. Таким чином, відбувається конфігурування обробки згідно з побажаннями користувача.

Обробка файлу є основним блоком, що відповідає за безпосередню зміну звукового сигналу згідно з параметрами, наданими користувачем. Тут важлива якість реалізації обробки, щоб не втратити якість звуку. Після завершення обробки файл передається на наступний етап.

Застосування обробки з отриманими параметрами – це перший підетап блоку обробки, на якому викликаються відповідні функції, які модифікують звукову хвилю відповідно до заданих параметрів. Важливо, щоб обробка відбувалась швидко, навіть для великих аудіофайлів. Цей етап є основним з точки зору логіки обробки аудіо.

Повернення результату обробки є другим підетапом блоку обробки, на якому після успішного застосування того чи іншого ефекту аудіодані повертаються в основну програму.

В основній програмі аудіодані зберігаються, та записуються у сформований файл, який зберігається у тимчасове сховище.

Повернення обробленого файлу – блок, що відповідає за відправку відповіді на запит клієнта у вигляді обробленого файлу. Користувач може його прослухати, зберегти або повторити обробку з іншими параметрами. Важливо забезпечити швидку доставку та правильне кодування MIME-типу файлу. Після

цього користувачеві доступні нові можливості для подальшої взаємодії з системою.

Блок “Видалення” відповідає за очищення тимчасових файлів після завершення запиту. Він зберігає стабільність і зменшує використання дискового простору. Може бути реалізований як фоновий процес або як автоматичне видалення після надсилання файлу. Наявність такого етапу допомагає уникнути переповнення сервера старими файлами.

3.3 Розробка функціональної схеми

Функціональна схема — це графічне представлення основних компонентів програмного або апаратного забезпечення, а також зв’язків між ними, яке відображає логіку роботи системи. Вона слугує засобом моделювання загальної архітектури проєкту, дозволяючи візуально простежити, як дані передаються між окремими модулями, які дії виконуються на кожному етапі, та яким чином забезпечується узгоджена взаємодія між клієнтською та серверною частинами.

На рисунку 3.2 зображено функціональну схему програмного забезпечення, яка складається з наступних двох основних блоків – “Клієнт” та “Сервер”.

Блок “Клієнт” включає в себе наступні модулі:

- модуль вибору обробки – дозволяє обрати потрібний тип обробки аудіофайлу;
- модуль налаштування обробки – забезпечує можливість користувачу вказати параметри обробки звуку;
- модуль формування HTTP-запиту – збирає дані (аудіофайл, метадані) у формат, придатний для надсилання на сервер;
- модуль надсилання HTTP-запиту – передає сформований запит через мережу до серверної частини системи;

− модуль завантаження аудіофайлу – надає інтерфейс для завантаження звукового файлу з пристрою користувача;

− модуль виведення результату – приймає оброблений аудіофайл від сервера та відображає результат користувачу;

− модуль збереження результату – завантажує оброблений аудіофайл на пристрій користувача.

Блок “Сервер” включає в себе наступні модулі:

− модуль збереження аудіофайлу – приймає аудіофайл з клієнта та тимчасово зберігає його для подальшої обробки;

− модуль збереження налаштувань обробки – зберігає передані з клієнта параметри обробки у вигляді метаданих;

− модуль обробки аудіофайлу – застосовує до звукового сигналу зазначені ефекти згідно отриманих параметрів;

− модуль надсилання результату – відправляє клієнту оброблений аудіофайл після завершення всіх обчислень;

− модуль видалення файлів – після відправки результату видаляє всі тимчасові файли (вхідні, оброблені, метадані) із сервера.

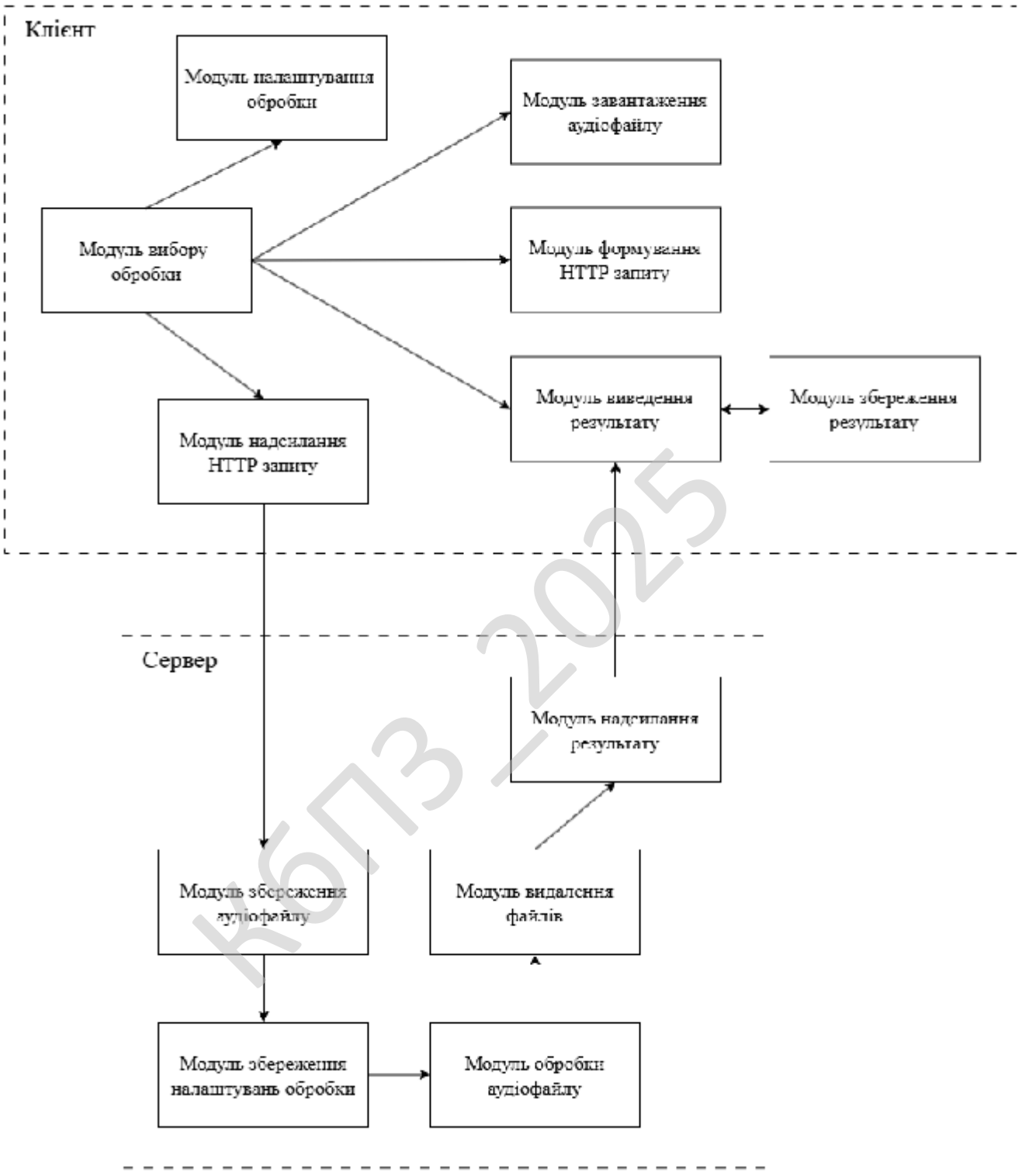

Рисунок 3.2 – Функціональна схема програмного забезпечення

3.4 Розробка діаграми процесів

Діаграма процесів — це графічне зображення послідовності дій, які виконуються в рамках певного бізнес- або технічного процесу. Вона дозволяє показати, які операції виконуються, в якій черговості, хто є виконавцем кожного етапу, та як відбувається обмін інформацією або даними між учасниками процесу.

На рисунку 3.3 зображену діаграму процесів розроблюваного програмного забезпечення, яка показує логіку системи та як в ній відбувається потік даних.

Потік даних у поданій діаграмі процесів починається з початкової точки, яка веде до головного вікна клієнтської частини застосунку. На головному вікні відображено меню обробок, доступних для вибору. Після вибору, користувач має змогу завантажити аудіофайл, який необхідно обробити.

Далі користувач переходить до налаштування параметрів обробки, після чого усі ці дані (файл і метадані налаштувань) передаються до API. API відповідає за обробку запиту, зокрема: зберігає аудіофайл, зберігає налаштування, а потім ініціює процес обробки звукового файлу відповідно до заданих параметрів.

Після завершення обробки API передає результат до модуля надсилання результату, який перенаправляє його на клієнтську частину. Дані потрапляють у вікно результату обробки, де користувач може прослухати отриманий результат. Далі користувач має змогу зберегти результат обробки або повернутись у головне вікно. Завершення взаємодії відображається переходом до кінцевої точки процесу.

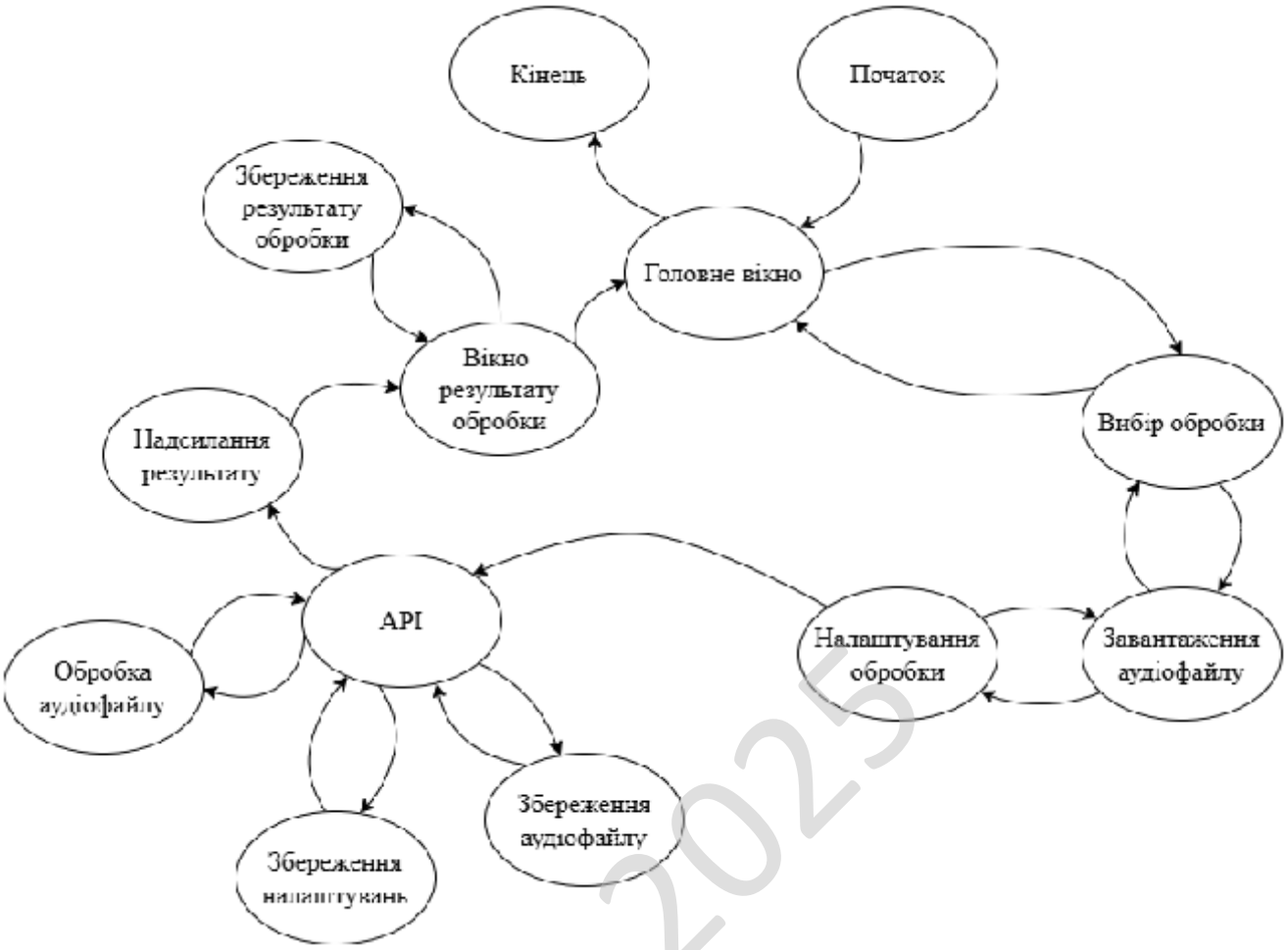

Рисунок 3.3 – Діаграма процесів

4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

4.1.1 Бібліотеки, використані при розробці серверної частини

Для реалізації серверної частини програмного забезпечення API порталу доступу та обробки звукової інформації було використано низку сучасних бібліотек, що забезпечують ефективну роботу з аудіосигналами, обробку HTTP-запитів, серіалізацію даних та забезпечення безперебійного функціонування серверу. Вибір бібліотек здійснювався на основі їх стабільності, популярності у професійному середовищі, активності спільноти розробників та сумісності з мовою програмування Python.

Клас Flask з бібліотеки flask – основний компонент однойменного мікрофреймворку для веброзробки на Python, який забезпечує легкий і гнучкий інструментарій для створення вебдодатків і API. Flask базується на бібліотеках Werkzeug (для роботи з HTTP-запитами) та Jinja2 (рушій шаблонів), що дозволяє розробникам обробляти маршрутизацію URL, формувати відповіді, працювати з шаблонами та обробляти HTTP-запити різних типів. Завдяки мінімалістичній архітектурі Flask не нав’язує жорстких правил структури проекту і не має вбудованого шару для роботи з базами даних чи формами, але підтримує розширення, які легко інтегруються для додавання потрібного функціоналу.

Основний функціонал модуля Flask включає можливість визначення маршрутів (URL) за допомогою декораторів, які зв’язують певні URL із функціями-представленнями, що обробляють запити і повертають відповіді (HTML, JSON, текст тощо). Flask підтримує обробку різних HTTP-методів (GET,

POST, PUT, DELETE), що робить його зручним для створення RESTful API. Крім того, модуль надає зручні інструменти для роботи з вхідними даними (через об'єкт request), формами, сесіями, а також для обробки помилок і винятків.

Flask також має гнучку систему контекстів додатку і запиту, яка дозволяє зберігати інформацію про поточний стан додатку і конкретного HTTP-запиту, що спрощує доступ до конфігурації, бази даних і інших ресурсів у межах обробки запиту. Завдяки цій архітектурі, а також підтримці middleware і розширень, Flask легко масштабується від невеликих проектів до складних вебсервісів. Розширення дозволяють додати функції об'єктно-реляційного відображення (ORM), аутентифікації, валідації форм, кешування та інші можливості, яких немає у базовій версії.

Приклад використання функціоналу класу Flask (Додаток Б):

```
app = Flask(__name__)
. . .
if "audio" not in request.files:
            return {"error": "File not found"}, 400
. . .
with open(file_name, "wb") as f:
            f.write(request_files["audio"].read())
. . .
room_size = float(request_form.get("room_size"))/100
            wet_level = float(request_form.get("wet_level"))/100
            dry_level = float(request_form.get("dry_level"))/100
. . .
app.run(host="0.0.0.0", port=5000, debug = True)
```

Метод send_file з бібліотеки flask – використовується для відправлення вмісту файлу клієнту у відповідь на HTTP-запит. Він автоматично обирає найефективніший спосіб передачі файлу, враховуючи можливості сервера, і підтримує різні параметри, які дозволяють контролювати тип вмісту (mimetype), спосіб завантаження (наприклад, чи має файл відкриватися у браузері або завантажуватися як вкладення), а також налаштування кешування і умовні запити.

Основний функціонал send_file() полягає у тому, що він приймає ім'я файлу або файловий об'єкт і відправляє його вміст у відповідь клієнту з відповідними HTTP-заголовками. Якщо не вказано явно, метод намагається автоматично визначити тип файлу (mimetype) за розширенням. Параметр as_attachment=True дозволяє примусово запропонувати браузеру завантажити

файл, а не відкривати його у вікні. Також можна задати власне ім’я файлу для завантаження через attachment_filename.

Метод підтримує додаткові опції, такі як cache_timeout для керування часом кешування файлу у браузері, add_etags для додавання ETag-заголовків, які допомагають оптимізувати передачу файлів, а також conditional, що дозволяє обробляти умовні запити (наприклад, для перевірки, чи змінився файл з останнього завантаження).

Приклад використання функціоналу методу send_file (Додаток Б):

```
send_file(file_data, as_attachment=True, download_name=result_audio_name, mimetype="audio/wav")
```

Клас Resource з бібліотеки flask_restful є базовим класом для створення ресурсів у RESTful API. Цей клас дозволяє розробникам організувати логіку обробки HTTP-запитів у вигляді класів, де кожен метод класу відповідає певному HTTP-методу (наприклад, get(), post(), put(), delete()). Такий підхід спрощує структуру коду, робить її більш зрозумілою і підтримуваною, особливо при створенні складних API.

Функціонал класу Resource полягає у наданні зручного інтерфейсу для визначення поведінки API при отриманні різних типів запитів. Клас автоматично інтегрується з Flask-RESTful API, дозволяючи легко додавати ресурси через метод add_resource(), який зв’язує URL-адресу з відповідним класом. Це дає змогу чітко розділяти логіку обробки різних маршрутів і підтримувати REST-архітектуру.

Крім базових методів для HTTP-запитів, Resource підтримує також різні механізми, такі як обробка аргументів запиту за допомогою reqparse, генерація помилок, а також застосування декораторів для методів класу (наприклад, для аутентифікації чи кешування). Це робить Resource гнучким інструментом для побудови як простих, так і складних RESTful сервісів.

Модуль Resource у flask_restful є ключовим елементом для організації REST API у вигляді класів із методами, що відповідають HTTP-операціям. Він забезпечить зручний і зрозумілий спосіб структурування коду та полегшить його масштабування і підтримку.

Приклад використання функціоналу класу Resource (Додаток Б):

```
class ApplyEffect(Resource):
. . .
```

Клас Api з бібліотеки flask_restful є основним класом для створення RESTful API на базі Flask. Він служить обгорткою над стандартним Flask-додатком і надає інтерфейс для реєстрації ресурсів (класів, що наслідують Resource) та маршрутизації HTTP-запитів до відповідних обробників. Клас Api спрощує організацію коду, дозволяючи чітко структурувати API, автоматично обробляти типові HTTP-методи і повертати відповіді у форматі JSON.

Функціонал модуля Api включає можливість додавання ресурсів через метод add_resource(), який зв'язує URL-шляхи з класами ресурсів. Api також підтримує обробку помилок, парсинг аргументів запитів, а також інтеграцію з різними розширеннями для аутентифікації, валідації та документування API. Крім того, Api забезпечує автоматичне формування коректних HTTP-відповідей і заголовків, що значно полегшує розробку і підтримку RESTful сервісів.

Завдяки класу Api розробникам не потрібно вручну налаштовувати маршрути і обробку HTTP-запитів, що підвищує продуктивність і знижує ймовірність помилок. Він дозволяє швидко створювати масштабовані та підтримувані вебсервіси, інтегруючи класичний Flask із принципами REST.

Приклад використання функціоналу класу Api (Додаток Б):

```
api = Api(app)
. . .
api.add_resource(ApplyEffect, '/apply_effect')
```

os – модуль стандартної бібліотеки Python, який забезпечує інтерфейс для взаємодії з операційною системою. Він надає широкий спектр функцій для роботи з файловою системою, процесами, змінними оточення та іншими системними ресурсами. Основна перевага модуля полягає у його платформонезалежності: він абстрагує відмінності між операційними системами, такими як Windows, macOS або Linux, що дозволяє писати універсальний код, який працюватиме у різних середовищах.

Однією з ключових сфер застосування модуля os є робота з файлами і каталогами. За допомогою функцій os.mkdir(), os.rmdir(), os.remove() можна

створювати, видаляти каталоги і файли. Функція os.listdir() дозволяє отримати список вмісту каталогу, а os.chdir() змінює поточний робочий каталог. Для отримання шляху до поточного каталогу використовується os.getcwd(). Крім того, модуль пропонує засоби для роботи з шляхами через підмодуль os.path, який допомагає об’єднувати шляхи, перевіряти існування файлів, отримувати їх розмір та іншу інформацію.

Модуль os також надає можливості для роботи зі змінними оточення операційної системи. За допомогою os.getenv() можна отримати значення змінної середовища, а через os.environ - встановити або змінити її. Це корисно для налаштування параметрів програми, які залежать від середовища виконання, наприклад, шляхів до ресурсів або конфігураційних файлів. Така взаємодія дозволяє робити програми більш гнучкими і адаптивними до різних умов запуску.

Приклад використання функціоналу бібліотеки os (Додаток Б):

```
os.remove(recieved_file)
os.remove(processed_file)
```

uuid – модуль, який забезпечує створення унікальних ідентифікаторів (UUID) відповідно до стандарту RFC 4122. UUID - це 128-бітні числа, які використовуються для ідентифікації об’єктів у розподілених системах без потреби у центральному реєстраторі. Модуль надає клас UUID для представлення таких ідентифікаторів та функції для їх генерації різних версій: uuid1(), uuid3(), uuid4(), uuid5(), а в новіших версіях також uuid7().

Функція uuid1() генерує UUID на основі часу та апаратного (MAC) адреса хоста, що може містити інформацію про пристрій і час створення. uuid3() і uuid5() створюють UUID на основі хешування (MD5 для uuid3() і SHA-1 для uuid5()) комбінації простору імен і імені, що дозволяє отримувати детерміновані унікальні ідентифікатори. uuid4() генерує випадковий UUID, який не містить інформації про час чи пристрій і є найбільш поширеним способом створення унікальних ідентифікаторів.

Клас UUID підтримує створення об’єктів із рядків у стандартному

шістнадцятковому форматі, байтових послідовностей або чисел, а також надає методи для конвертації UUID у рядок, байти або інші формати. Об’єкти UUID є незмінними і підтримують порівняння, хешування та інші операції, що робить їх зручними для використання як ключів у словниках або базах даних.

Модуль також містить константи для стандартних просторів імен, таких як NAMESPACE_DNS, NAMESPACE_URL, NAMESPACE_OID і NAMESPACE_X500, які використовуються з функціями uuid3() і uuid5() для створення унікальних ідентифікаторів на основі імен. Крім того, у Python 3.7+ додано атрибут is_safe для UUID версії 1, який інформує про безпеку генерації ідентифікатора у багатопроцесорних середовищах.

Приклад використання функціоналу бібліотеки uuid (Додаток Б):

```
recieved_file = "audio/" + str(uuid.uuid1()) + ".wav"
processed_file = "audio/" + str(uuid.uuid1()) + ".wav"
```

soundfile – бібліотека, яка призначена для читання і запису аудіофайлів у різних форматах, таких як WAV, FLAC, OGG та інших, з підтримкою високої якості звуку. Вона надає простий і зручний інтерфейс для роботи з аудіоданими у вигляді масивів NumPy, що робить її корисною для обробки звуку, аналізу та синтезу аудіосигналів у наукових і музичних проектах.

Основний функціонал soundfile включає можливість відкривати аудіофайли для читання і запису, отримувати інформацію про параметри аудіо (частоту дискретизації, кількість каналів, довжину), а також працювати з блоками даних у різних форматах і типах даних. Бібліотека підтримує роботу з потоками даних, що дозволяє ефективно обробляти великі аудіофайли без необхідності завантажувати їх повністю у пам’ять.

Приклад використання функціоналу бібліотеки soundfile (Додаток Б):

```
waveform, sample_rate = sf.read(recieved_file)
sf.write(processed_file, processed_waveform, sample_rate)
```

Клас BytesIO з модуля io – реалізує потік байтів, який зберігається у пам’яті, надаючи інтерфейс, схожий на роботу з файлом, але без необхідності взаємодії з файловою системою. Це означає, що замість читання або запису даних у файл на диску, всі операції відбуваються у швидкому буфері в оперативній

пам’яті. Такий підхід особливо корисний, коли потрібно тимчасово зберігати або обробляти байтові дані без створення фізичних файлів.

Основний функціонал BytesIO включає можливість читання, запису і переміщення позиції у потоці байтів. Клас підтримує стандартні методи файлових об’єктів, такі як read(), write(), seek() і tell(), що дозволяє працювати з ним так само, як із звичайним двійковим файлом. При створенні об’єкта можна передати початковий байтовий рядок, який буде використаний як початковий вміст буфера.

Клас BytesIO часто використовується у випадках, коли потрібно опрацьовувати аудіо-, відео- або інші бінарні дані, не створюючи тимчасових файлів. Наприклад, при роботі з мережевими протоколами, обробці зображень або генерації документів у пам’яті. Він також корисний для тестування коду, який працює з файлами, без необхідності створювати реальні файли на диску.

Приклад використання функціоналу класу BytesIO (Додаток Б):

```
with open(processed_file, "rb") as f:
            file_data = BytesIO(f.read())
```

pedalboard – це потужний інструмент для роботи з аудіо у Python, який дозволяє читати, записувати, обробляти звук і додавати до нього професійні аудіоефекти. Вона підтримує більшість популярних аудіоформатів (WAV, MP3, FLAC, OGG тощо) без додаткових залежностей, а також дозволяє працювати з ефектами в режимі реального часу. Pedalboard розроблена компанією Spotify і використовується як у внутрішніх проектах для покращення машинного навчання, так і для створення аудіоконтенту без необхідності застосовувати складні цифрові аудіостанції.

Основний функціонал бібліотеки полягає у створенні “педалбордів” - послідовностей аудіоефектів, які можна застосовувати до аудіосигналу. До стандартного набору входять такі ефекти, як хорус (Chorus), дисторшн (Distortion), реверберація (Reverb), компресор (Compressor), гейн (Gain), фільтри (HighpassFilter, LowpassFilter, LadderFilter), затримка (Delay), пітч-шифт (PitchShift) та інші. Крім того, pedalboard підтримує завантаження сторонніх

плагінів у форматі VST3 і Audio Units, що розширює можливості обробки звуку.

Pedalboard забезпечує високу продуктивність і ефективне використання пам'яті, звільняючи глобальний інтерпретаторний лок (GIL) Python і дозволяючи використовувати багатоядерність без складної багатопроцесорної логіки. Це робить обробку аудіо значно швидшою у порівнянні з іншими бібліотеками. Вона також підтримує потокову обробку звуку, що дозволяє застосовувати ефекти у реальному часі, наприклад, для живих виступів або інтерактивних застосунків.

Для роботи з файлами pedalboard надає зручні інструменти для читання і запису аудіо через клас AudioFile, що дозволяє легко інтегрувати бібліотеку у різні робочі процеси. Код, який застосовує ефекти, виглядає інтуїтивно: створюється об'єкт Pedalboard, до якого додаються потрібні ефекти, і потім аудіо проходить через цей ланцюжок для обробки. Після цього результат можна зберегти у файл або передати далі у програмі.

Приклад використання функціоналу бібліотеки pedalboard (Додаток Б):

```
        reverb = Reverb(room_size=room_size, wet_level=wet_level,
dry_level=dry_level, width=1.0)
        board = Pedalboard([reverb])
        return board.process(waveform, sample_rate)
```

noisereduce – бібліотека Python для зменшення шуму у часових сигналах, таких як мова, біоакустичні та фізіологічні сигнали. Вона базується на методі спектрального гейтингу (spectral gating), який полягає у побудові спектрограми сигналу, оцінці порогу шуму для кожної частотної смуги та застосуванні маски, що пригнічує шум нижче цього порогу. Такий підхід дозволяє ефективно зменшувати фоновий шум, зберігаючи при цьому якість корисного сигналу.

Noisereduce підтримує два основні алгоритми: Stationary Noise Reduction (зменшення стаціонарного шуму), де поріг шуму залишається сталим протягом усього сигналу, та Non-stationary Noise Reduction (зменшення нестаціонарного шуму), де поріг оновлюється динамічно у часі. Це дозволяє адаптуватися до змінних шумових умов і покращувати якість очищення аудіо. У новіших версіях бібліотеки реалізовано також PyTorch-імплементацію спектрального гейтингу, що значно прискорює обробку і дозволяє інтегрувати алгоритм у нейронні

мережі.

Функція reduce_noise() є основним інтерфейсом для застосування шумозаглушення: користувач передає аудіокліп і, за бажанням, фрагмент, що містить лише шум, для точнішої оцінки порогу.

Приклад використання функціоналу бібліотеки noisereduce (Додаток Б):

```
return rn.reduce_noise(y=waveform, sr=sample_rate)
```

4.1.2 Блок-схема серверної частини програмного забезпечення

На рисунку 4.1 наведено блок-схему серверної частини програмного забезпечення. Її робота складається з наступних пунктів:

– перевірка присутності файлу у надісланих файлах запиту. Якщо файл відсутній – повертається повідомлення про помилку з кодом 400. Якщо присутній – відбувається перехід до наступного блоку;

– збереження надісланого аудіофайлу. Файл, що надіслав користувач при запиті, зберігається локально;

– збереження надісланих налаштувань обробки;

– збереження частоти дискретизації надісланого аудіофайлу;

– збереження інформації про хвилю надісланого аудіофайлу;

– обробка аудіофайлу. Для запобігання використанню повторюваного коду, обробка аудіофайлу відбувається в окремому невеликому модулі;

– збереження обробленого аудіофайлу;

– конвертація обробленого аудіофайлу. Аудіофайл конвертується у послідовність бітів і копіюється в буфер оперативної пам’яті;

– видалення надісланого аудіофайлу;

– видалення обробленого аудіофайлу;

– надсилання результату обробки. У відповідь на запит користувача повертається послідовність бітів обробленого аудіофайлу з буфера оперативної пам’яті.

Рисунок 4.1 – Блок-схема серверної частини програмного забезпечення

4.1.3 Бібліотеки, використані при розробці клієнтської частини

file_picker – бібліотека, яка надає можливість інтегрувати у додаток нативний файловий провідник для вибору файлів або директорій із пристрою користувача. Вона підтримує вибір як одного, так і кількох файлів одночасно, а також дозволяє фільтрувати файли за розширеннями, що робить процес вибору більш зручним і контрольованим. File_picker працює на різних платформах – Android, iOS, Web, Windows, macOS і Linux, забезпечуючи єдиний інтерфейс для роботи з файлами незалежно від операційної системи.

Основний функціонал бібліотеки включає методи для відкриття діалогів вибору файлів (pickFiles()), вибору директорій (getDirectoryPath()), а також діалог збереження файлу (saveFile()). Після вибору файлу або файлів бібліотека повертає інформацію про них у вигляді об’єктів, які містять шлях до файлу, ім’я, розмір, розширення та навіть байтовий вміст, що дозволяє легко працювати з файлами без додаткових перетворень. Це особливо корисно для подальшої обробки, відображення або завантаження файлів, наприклад, у хмарні сервіси.

File_picker підтримує гнучке фільтрування файлів за типом або конкретними розширеннями, що дозволяє обмежити вибір користувача лише тими файлами, які потрібні додатку (наприклад, лише зображення, документи або аудіофайли). Крім того, бібліотека дозволяє обробляти ситуації, коли користувач скасовує вибір, що робить її інтеграцію більш надійною і зручною для користувача.

Завдяки своїй простоті та потужності, file_picker часто використовується для реалізації функціоналу завантаження файлів, імпорту даних, вибору документів чи мультимедіа у мобільних і вебдодатках на Flutter.

Приклад використання функціоналу бібліотеки file_picker (Додаток Б):

```
Future<void> loadSound() async {
    FilePickerResult? result = await FilePicker.platform.pickFiles(type:
FileType.any);
```

path – кросплатформена бібліотека для роботи з файловими шляхами у вигляді рядків. Вона надає набір функцій для маніпуляції шляхами, таких як

об’єднання, розділення, нормалізація, отримання розширень файлів, імен каталогів та інших операцій, пов’язаних із шляхами до файлів і папок. Основна мета бібліотеки – забезпечити однакову поведінку з файловими шляхами на різних платформах (Windows, POSIX-системи, веб), абстрагуючи від відмінностей у форматах шляхів.

Функціонал бібліотеки включає такі операції, як join() для об’єднання частин шляху з урахуванням правильного роздільника, basename() для отримання імені файлу або каталогу з повного шляху, dirname() для отримання шляху до каталогу, extension() для визначення розширення файлу, а також normalize() для приведення шляху до канонічної форми. Всі ці функції працюють з рядками і не взаємодіють безпосередньо з файловою системою, що робить їх легкими та швидкими.

Бібліотека підтримує роботу з різними стилями шляхів через клас Context, який дозволяє явно вказати, чи слід обробляти шляхи у стилі Windows, POSIX або URL. Це корисно, коли потрібно працювати з шляхами, які не залежать від платформи, на якій виконується код. Наприклад, можна створити контекст для Windows-шляхів і маніпулювати ними навіть на Linux або macOS.

Приклад використання функціоналу бібліотеки path (Додаток Б):

```
loadedSound = File(result.files.single.path!);
. . .
else if (path.extension(loadedSound!) == '.wav' ||
path.extension(loadedSound!) == '.mp3' || path.extension(loadedSound!) == '.ogg'){
. . .
}
. . .
path.basename(widget.processedSoundPath)
```

path_provider – бібліотека, яка надає зручний і кросплатформений спосіб отримання доступу до часто використовуваних директорій файлової системи на пристрої користувача. Вона підтримує основні платформи, такі як Android, iOS, Linux, macOS і Windows, і допомагає розробникам знаходити системні шляхи до тимчасових папок, папок з документами додатка, кешем, папок підтримки додатка та інших важливих директорій. Це дозволяє зберігати файли у відповідних місцях, які гарантують безпечне та ефективне зберігання даних.

Основний функціонал path_provider полягає у наданні асинхронних методів, таких як getTemporaryDirectory(), getApplicationDocumentsDirectory(), getApplicationSupportDirectory(), getDownloadsDirectory() та інших, які повертають об’єкти типу Directory з шляхами до відповідних системних папок. Наприклад, getTemporaryDirectory() повертає тимчасову директорію, вміст якої може бути видалений операційною системою, а getApplicationDocumentsDirectory() – директорію, призначену для зберігання постійних файлів додатка, доступних лише йому.

Приклад використання функціоналу бібліотеки path_provider (Додаток Б):

```
Directory tempDir = await getTemporaryDirectory();
```

dart:io – бібліотека, яка надає широкий набір API для роботи з операційною системою у не-веб-застосунках. Вона дозволяє працювати з файлами, каталогами, сокетами, процесами, HTTP-серверами та клієнтами, а також іншими операціями вводу-виводу. Бібліотека підтримує асинхронні операції, які реалізовані через об’єкти Future і Stream, що забезпечує ефективне і неблокуюче виконання коду. Важливо, що dart:io не підтримується у браузерних додатках, але доступна для серверних додатків, командних скриптів, а також мобільних і десктопних Flutter-додатків.

Основна частина функціоналу dart:io пов’язана з роботою з файловою системою. Класи File, Directory і Link дозволяють створювати, читати, записувати, видаляти і маніпулювати файлами та папками. Наприклад, можна створювати файли, перевіряти їх існування, копіювати, перейменовувати, отримувати інформацію про розмір, дату останньої модифікації та інші атрибути. Більшість цих операцій є асинхронними і повертають Future, що дозволяє не блокувати основний потік виконання.

dart:io також надає інструменти для роботи з мережевими з’єднаннями через сокети і WebSocket, а також для створення HTTP-серверів і клієнтів. Це дозволяє будувати як прості серверні додатки, так і складні мережеві сервіси без необхідності використовувати сторонні бібліотеки. Бібліотека підтримує роботу з процесами операційної системи, що дає змогу запускати зовнішні команди та

взаємодіяти з ними, керувати потоками вводу-виводу цих процесів.

Особливістю dart:io є її орієнтація на асинхронність і неблокуючий ввід-вивід, що відповідає сучасним підходам до розробки високопродуктивних додатків. Синхронні методи в бібліотеці позначені суфіксом Sync і використовуються рідше через потенційне блокування виконання. Асинхронна модель дозволяє ефективно масштабувати додатки, особливо серверні, і підтримує роботу з великими обсягами даних або численними одночасними з'єднаннями.

Приклад використання функціоналу бібліотеки dart:io (Додаток Б):

```
processedFile = File(processedSoundPath);
. . .
final downloadsDir = Directory('/storage/emulated/0/Download');
```

http – високорівнева, асинхронна та зручна у використанні бібліотека для виконання HTTP-запитів. Вона надає простий інтерфейс для відправлення GET, POST, PUT, DELETE та інших типів запитів, а також для отримання відповідей від серверів у форматі тексту, JSON чи байтів. Завдяки підтримці Future, бібліотека органічно інтегрується з асинхронною моделлю Dart, що дозволяє писати неблокуючий код.

Основний функціонал включає топ-рівневі функції, такі як http.get(), http.post(), які дозволяють швидко виконувати окремі запити без зайвих налаштувань. Для більш складних сценаріїв можна використовувати клас Client, який підтримує відкриття постійних з'єднань і повторне використання ресурсів, що підвищує продуктивність при великій кількості запитів до одного сервера. Після завершення роботи з клієнтом його слід закривати через метод close().

Бібліотека підтримує роботу з URI, автоматичне кодування параметрів, встановлення заголовків, передачу тіл запитів у вигляді рядків, байтів або формованих даних. Вона також дозволяє отримувати детальну інформацію про відповіді – статусний код, заголовки, тіло відповіді. Для складніших випадків можна створювати власні об'єкти запитів (Request, StreamedRequest) і надсилати їх через Client.send(), що дає гнучкість у налаштуванні.

Для кросплатформеності у Flutter бібліотека підтримує різні клієнти

HTTP, які можна конфігурувати залежно від платформи (Android, iOS, Web, Desktop). Наприклад, для веб-версії використовується клієнт на основі Fetch API, а для мобільних платформ – IOClient або інші, що забезпечує оптимальну роботу на кожній платформі. Це дозволяє розробникам писати універсальний код без необхідності турбуватися про специфіку платформи.

Приклад використання функціоналу бібліотеки http (Додаток Б):

```
var request = http.MultipartRequest("POST",
Uri.parse("http://192.168.0.109:5000/apply_effect"));
. . .
request.files.add(await http.MultipartFile.fromPath("audio",
loadedSound!.path));
```

audioplayers – популярна бібліотека для відтворення аудіофайлів, яка підтримує одночасне відтворення кількох звукових доріжок і асинхронне управління аудіо. Вона дозволяє легко інтегрувати аудіо у мобільні та вебдодатки, працюючи з локальними файлами, потоковим аудіо з інтернету, а також з байтовими масивами. Бібліотека підтримує основні функції керування відтворенням: програвання, паузу, зупинку, перемотування, регулювання гучності і балансування звуку.

Серед ключових особливостей Audioplayers – кросплатформеність, підтримка різних форматів аудіо, а також можливість одночасного відтворення кількох аудіоджерел. Бібліотека надає події та стріми для відстеження стану плеєра, позиції відтворення та тривалості треку, що дозволяє створювати інтерактивні аудіоінтерфейси з прогрес-барами і контролерами.

Приклад використання функціоналу бібліотеки audioplayers (Додаток Б):

```
AudioPlayer audioPlayer = AudioPlayer();
. . .
Future setAudio() async {
    audioPlayer.setSource(DeviceFileSource(widget.processedSoundPath));
  }
. . .
audioPlayer.onPlayerStateChanged.listen((state) {
      setState(() {
        isPlaying = state == PlayerState.playing;
      });
    });
    audioPlayer.onDurationChanged.listen((newDuration) {
      setState(() {
        duration = newDuration;
      });
    });
```

```
    audioPlayer.onPositionChanged.listen((newPosition) {
      setState(() {
        position = newPosition;
      });
    });

. . .
await audioPlayer.seek(position);
await audioPlayer.resume();

. . .
if (isPlaying) {
await audioPlayer.pause();
} else {
await audioPlayer.resume();
}
```

permission_handler – бібліотека для запиту та перевірки дозволів на доступ до різних ресурсів пристрою, таких як камера, мікрофон, місцезнаходження, контакти, сховище та інші. Вона підтримує основні мобільні платформи – Android та iOS – і надає єдиний API для роботи з дозволами під час виконання додатку, що відповідає сучасним вимогам безпеки та конфіденційності.

Основний функціонал permission_handler включає можливість перевірити статус дозволу (наприклад, чи він наданий, відхилений, обмежений або постійно відхилений), а також запросити дозвіл у користувача через системний діалог. При цьому, якщо дозвіл уже наданий, діалог не з'являється, і повертається відповідний статус. Бібліотека також дозволяє відкривати налаштування додатку на пристрої, щоб користувач міг змінити дозволи вручну.

Для Android плагін підтримує показ пояснень (rationale), які можна відобразити користувачу перед запитом дозволу, щоб підвищити ймовірність його надання. Крім того, permission_handler надає зручні методи для обробки різних сценаріїв, таких як відмова користувача, тимчасове обмеження або постійна відмова, що дозволяє розробникам реалізовувати гнучку логіку взаємодії з користувачем.

Приклад використання функціоналу бібліотеки permission_handler (Додаток Б):

```
var permissionStatus = await Permission.manageExternalStorage.request();
if (!permissionStatus.isGranted) {
  return 'Access denied';
}
```

4.1.4 Блок-схема клієнтської частини програмного забезпечення

На рисунку 4.2 наведено блок-схему серверної частини програмного забезпечення. Її робота складається з наступних пунктів:

- виведення головного вікна;
- вибір обробки для аудіофайлу. Користувач вибирає, яку обробку серед запропонованих застосувати до аудіофайлу;
- налаштування обробки;
- передача аудіофайлу та налаштувань до API;
- отримання результату обробки;
- виведення результату обробки;
- збереження результату. Якщо користувач обрав збереження – результат обробки зберігається на пристрої і в застосунку відкривається меню доступних обробок. Якщо ні – результат не зберігається і в застосунку відкривається меню доступних обробок.

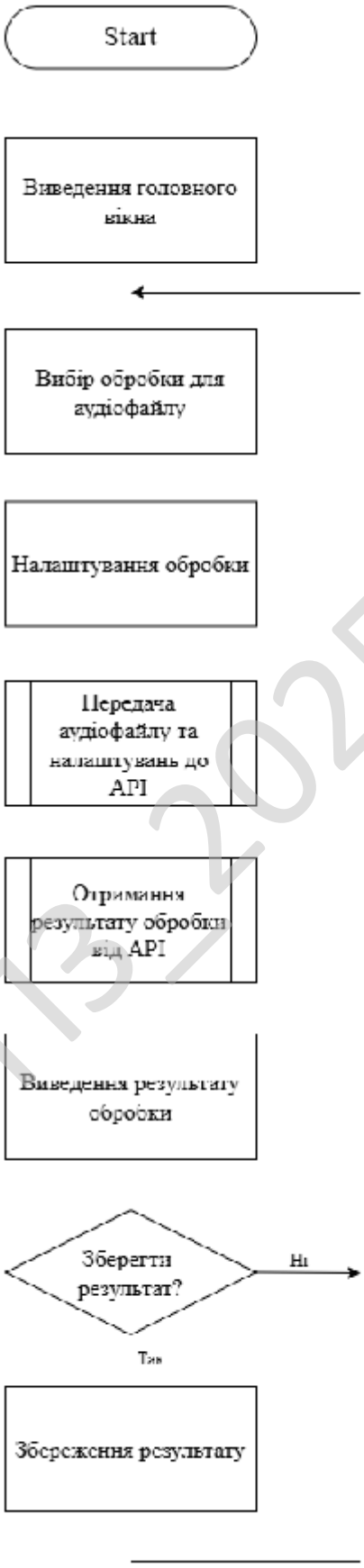

Рисунок 4.2 – Блок-схема клієнтської частини програмного забезпечення

4.2 Захист розробленого програмного забезпечення

Захист програмного забезпечення є критично важливим аспектом при роботі з персоніфікованими або конфіденційними даними, зокрема — із звуковою інформацією, яка може містити голос користувача або інші чутливі аудіодані. Тому у розробленому програмному забезпеченні API порталу доступу та обробки звукової інформації було передбачено низку заходів, спрямованих на зменшення ризику витоку інформації та запобігання сторонньому доступу до оброблених матеріалів. Один із таких підходів — використання унікальних і невпізнаваних імен для збереження тимчасових аудіофайлів на сервері.

З цією метою в системі застосовується стандартна бібліотека uuid, яка генерує унікальний ідентифікатор для кожного аудіофайлу, що завантажується на сервер. Завдяки цьому назви файлів не містять жодної персоналізованої інформації і не можуть бути вгадані або пов’язані з конкретним користувачем. Це дозволяє приховати структуру файлової системи та унеможливлює прямий несанкціонований доступ до збережених файлів через URL або інші засоби.

Після завершення обробки аудіо і передачі результату користувачеві, всі проміжні файли — як вхідні, так і оброблені — видаляються із файлової системи за допомогою бібліотеки os. Такий підхід мінімізує час зберігання файлів на сервері та значно знижує імовірність їх перехоплення у разі несанкціонованого доступу до серверного середовища. Видалення файлів реалізоване одразу після формування відповіді, що гарантує повну автоматизацію цього захисного заходу.

Такий підхід забезпечує збереження конфіденційності користувача, відповідність сучасним вимогам безпеки, а також дозволяє масштабувати систему без втрати контролю над ризиками, пов’язаними з обробкою звукової інформації.

5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 показане головне вікно програми, на якому знаходиться меню доступних ефектів, що складається з наступних елементів:

- Reverb.
- Pitch Shift.
- Reduce Background Noise.

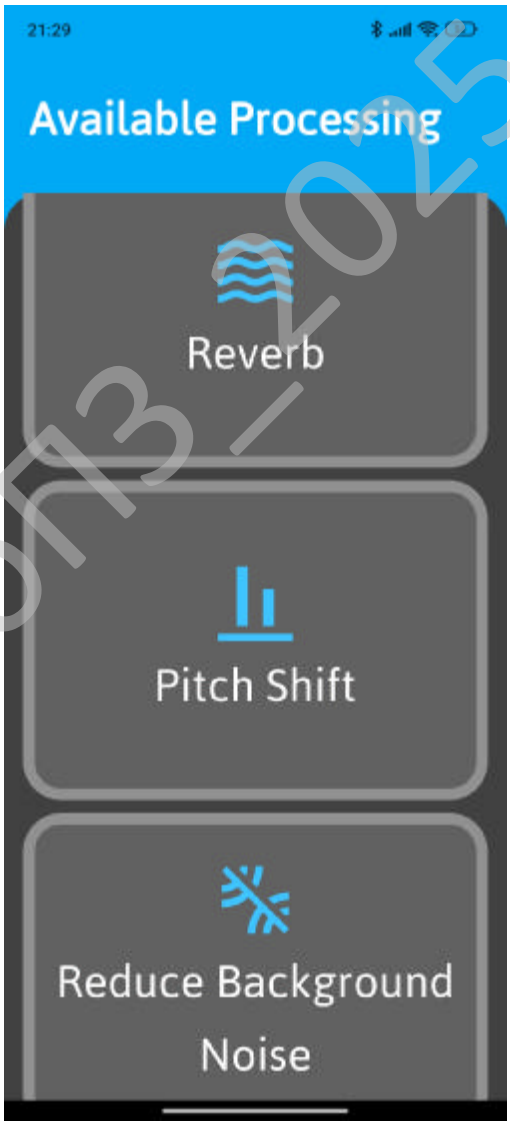

Рисунок 5.1 – Головне вікно програми

Після вибору пункту меню “Reverb”, відкривається сторінка налаштувань обраного ефекту (рис. 5.2), яка складається з наступних елементів:

- назва обраного ефекту;
- кнопка для додавання аудіофайлу;
- статус завантаження аудіофайлу;
- слайдер для налаштування розміру кімнати “Room Size”;
- слайдер для налаштування гучності обробленого сигналу “Wet Level”;
- слайдер для налаштування гучності необробленого сигналу “Dry Level”;
- кнопка відміни застосування ефекту;
- кнопка застосування ефекту.

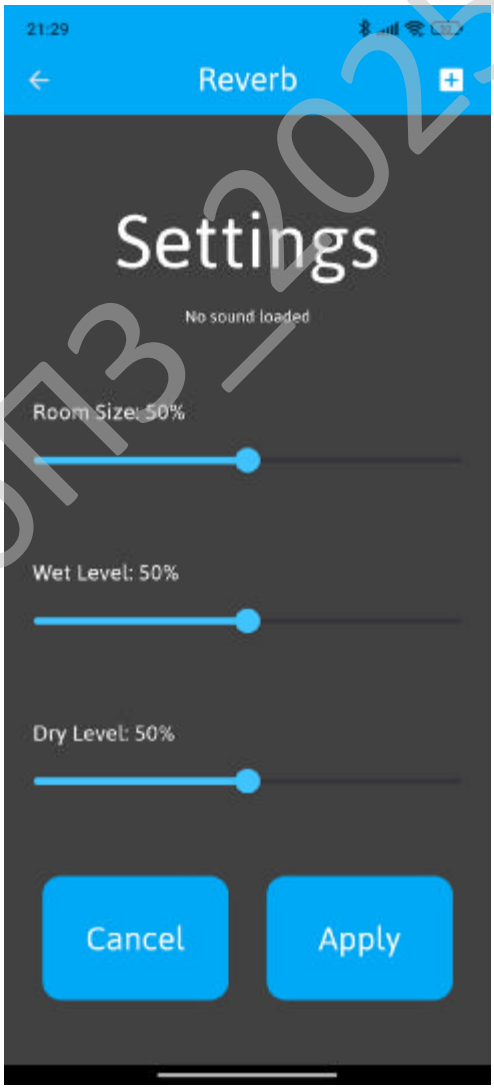

Рисунок 5.2 – Вікно налаштувань ефекту “Reverb”

Після вибору пункту меню “Pitch Shift”, відкривається сторінка налаштувань обраного ефекту (рис. 5.3), яка складається з наступних елементів:

- назва обраного ефекту;
- кнопка для додавання аудіофайлу;
- статус завантаження аудіофайлу;
- слайдер для налаштування висоти тону “Semitones”;
- кнопка відміни застосування ефекту;
- кнопка застосування ефекту.

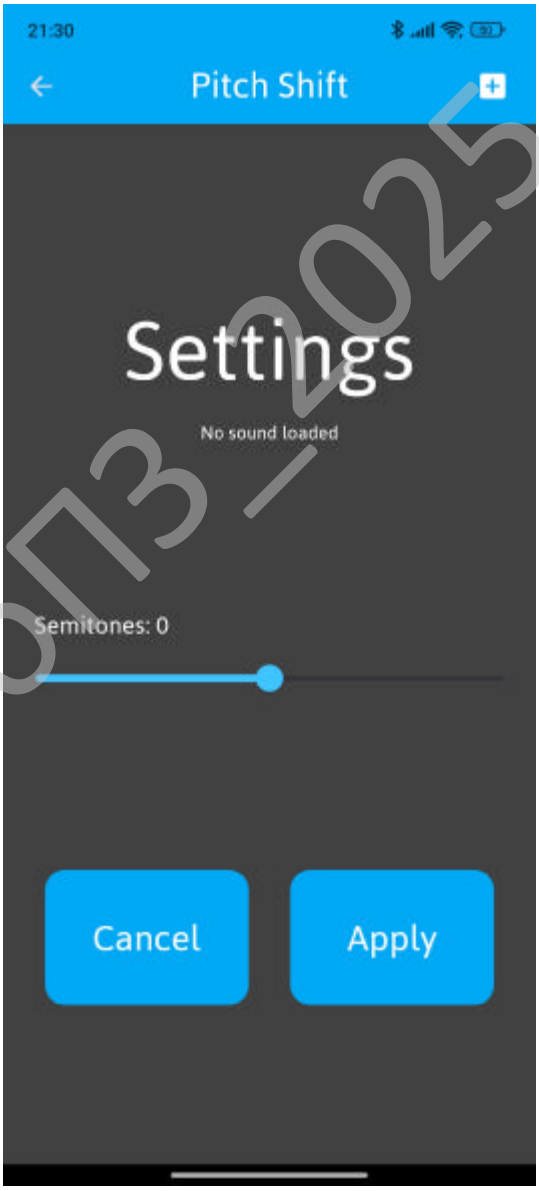

Рисунок 5.3 – Вікно налаштувань ефекту “Pitch Shift”

Після вибору пункту меню “Reduce Background Noise”, відкривається сторінка налаштувань обраного ефекту (рис. 5.4), яка складається з наступних елементів:

- назва обраного ефекту;
- кнопка для додавання аудіофайлу;
- статус завантаження аудіофайлу;
- кнопка відміни застосування ефекту;
- кнопка застосування ефекту.

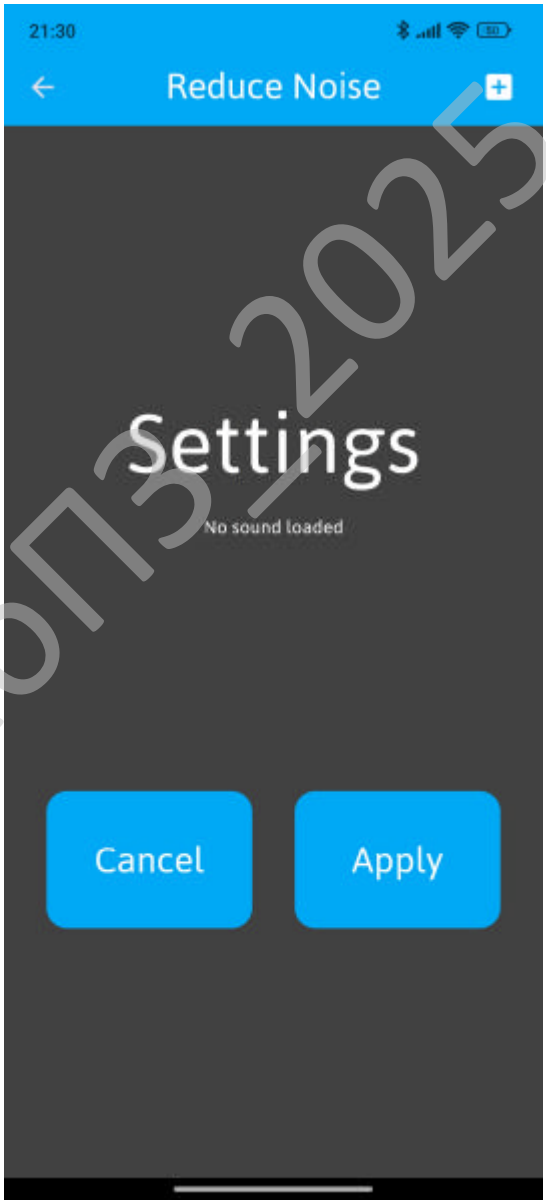

Рисунок 5.4 – Вікно налаштувань ефекту “Reduce Background Noise”

Після застосування обраного ефекту, відкривається вікно результату обробки (рис. 5.5), яке складається з наступних елементів:

– назва вікна;

– слайдер для візуального відображення програвання обробленого аудіофайлу;

– кнопка для програвання обробленого аудіофайлу;

– кнопка повернення до меню;

– кнопка завантаження обробленого аудіофайлу.

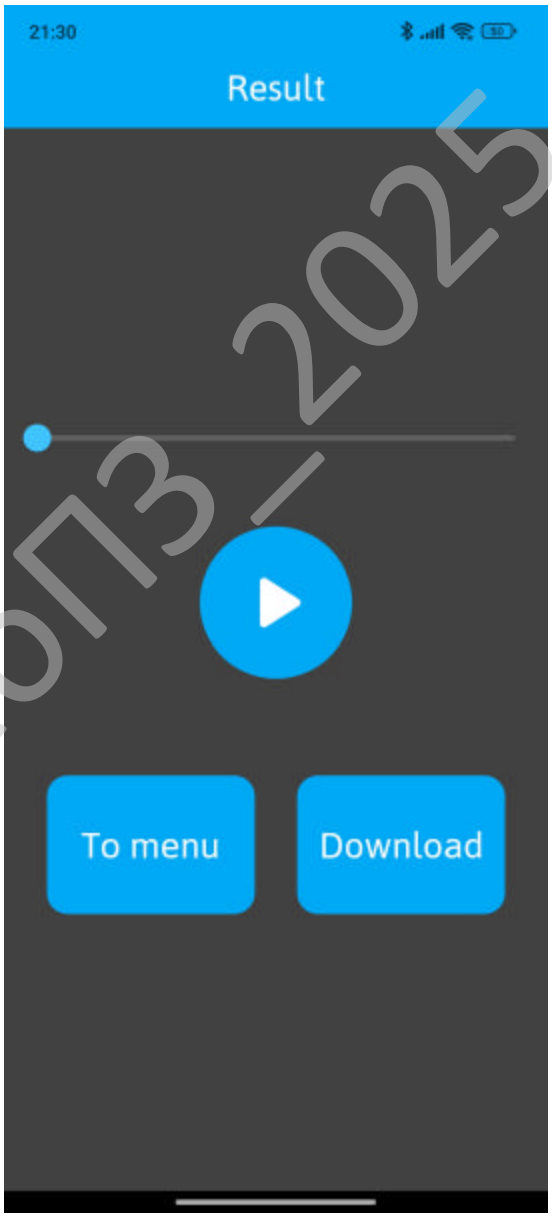

Рисунок 5.5 – Вікно результату обробки

Методика впровадження системи в промислову експлуатацію передбачає мінімальне залучення додаткових ресурсів, що забезпечує її швидке та економічно вигідне впровадження у виробниче середовище. Оскільки система побудована за принципом клієнт-серверної архітектури, її розгортання потребує лише попередньої інсталяції серверної частини на визначену машину та забезпечення доступу клієнтів через API. Уся комунікація здійснюється через стандартизовані HTTP-запити, що не потребує специфічного програмного забезпечення на стороні користувача.

Клієнтська частина системи реалізована з урахуванням принципів інтуїтивно зрозумілого дизайну, що суттєво спрощує процес взаємодії з інтерфейсом. Завдяки логічній структурі вікон та зрозумілим візуальним елементам, користувачеві не потрібен попередній інструктаж чи спеціальна технічна підготовка. Це дозволяє впроваджувати програмне забезпечення навіть у середовищах, де технічна грамотність персоналу є обмеженою.

Під час впровадження необхідно провести тестування серверної частини з урахуванням навантаження, типового для запланованого середовища експлуатації. У разі потреби можлива адаптація параметрів обробки або розширення серверної потужності для забезпечення оптимальної продуктивності. Також рекомендується налаштувати систему резервного копіювання оброблених аудіофайлів і логів обробки для контролю процесів.

Завершальним етапом впровадження є інтеграція системи в поточні бізнес-процеси або технологічний ланцюг, залежно від галузі. У разі потреби може бути реалізовано додатковий API-доступ до зовнішніх систем. Загалом, методика впровадження є гнучкою та масштабованою, що дозволяє легко інтегрувати рішення як у невеликих установах, так і в промислових або оборонних об’єктах.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання кваліфікаційної роботи, призначено для доступу та обробки звукової інформації.

Для вирішення поставленої мети було:

– Проведено критичний аналіз існуючих рішень у сфері обробки звукової інформації з відкритим або локальним API-доступом. Виявлено їхні сильні сторони та недоліки, зокрема в аспекті безпеки, універсальності та функціональності;

– Обґрунтовано вибір архітектури системи, яка включає клієнтську та серверну частини, а також визначено методи реалізації обробки аудіо з використанням сучасних бібліотек для цифрової обробки сигналів;

– Реалізовано серверну частину API, що забезпечує обробку аудіо за заданими параметрами, та клієнтську частину, яка дозволяє зручно взаємодіяти з користувачем і надсилати запити до сервера.;

– Побудовано блок-схеми основних алгоритмів обробки звуку, розроблено логіку взаємодії компонентів системи;

– Організовано інтерфейс користувача на клієнтській стороні, який дозволяє обирати режими обробки, завантажувати та отримувати результати.

– Передбачено інформування користувача про помилки введення, та некоректні параметри запиту;

– Розроблено функціонал для забезпечення захисту даних;

– Розроблено рекомендації щодо розгортання програмного забезпечення в захищеному середовищі, зокрема для потреб державних організацій або критичної інфраструктури.

Реалізоване рішення поєднує сучасні методи розробки API, обробки звуку та принципи побудови зручного користувацького інтерфейсу, що робить систему придатною до широкого кола застосувань.

Розробка супроводжувалась побудовою функціональної схеми, діаграми процесів, вибором оптимальних форматів передавання даних і обґрунтуванням технічних рішень щодо зберігання та захисту інформації. У процесі дослідження було проаналізовано бібліотеки, необхідні для реалізації серверної частини, та розроблено чіткий алгоритм функціонування клієнтської частини.

Для розробки програмного забезпечення API порталу доступу та оброки звукової інформації використовувалися мови програмування Dart та Python. Дані мови програмування дозволили ефективно вирішити поставлену мету та задачі кваліфікаційної роботи.

Практична цінність реалізованої системи полягає у її потенціалі до використання в оборонних, журналістських, дослідницьких і медіа-сферах, зокрема в контексті протидії OSINT-загрозам в умовах війни в Україні. Система може бути ефективно інтегрована в уже наявні інформаційні комплекси, забезпечуючи фільтрацію, анонімізацію та маскування звукової інформації залежно від поставлених цілей.

У результаті виконання кваліфікаційної роботи було досягнуто основної мети – створення програмного забезпечення, здатного забезпечити ефективну взаємодію з аудіоданими через API з можливістю масштабування, розширення функцій та адаптації до специфічних умов експлуатації. Отримані результати можуть слугувати базою для подальших досліджень та практичного використання у суміжних галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is Open-Source Intelligence? [Електронний ресурс]. Режим доступу – https://www.sans.org/blog/what-is-open-source-intelligence/

2. Enhancing Investigations with Audio OSINT Analysis: A Comprehensive Approach [Електронний ресурс]. Режим доступу – https://medium.com/@rabbi_security/enhancing-investigations-with-audio-osint-analysis-a-comprehensive-approach-1d1cc355025d

3. Hear No Evil: An Introduction to Audio File Analysis for OSINT [Електронний ресурс]. Режим доступу – https://medium.com/@aleksanderrr_/hear-no-evil-an-introduction-to-audio-file-analysis-for-osint-f545c8623377

4. Krisp – World's best AI Note Taker and Noise Cancelling app [Електронний ресурс]. Режим доступу – https://krisp.ai/

5. Krisp Review: Features, Pros, Cons, & Alternatives [Електронний ресурс]. Режим доступу – https://10web.io/ai-tools/krisp/

6. Adobe Podcast | AI audio recording and editing, all on the web [Електронний ресурс]. Режим доступу – https://podcast.adobe.com/en

7. Adobe Podcast AI: Honest Review & Beginner's Guide [Електронний ресурс]. Режим доступу – https://www.elegantthemes.com/blog/business/adobe-podcast-ai-review

8. AudioStrip - The Best Online Vocal Isolator for Free [Електронний ресурс]. Режим доступу – https://www.audiostrip.com/

9. Extract and Isolate Stems with Audio Strip: Pros, Cons, and Alternatives [Електронний ресурс]. Режим доступу – https://www.toolify.ai/ai-news/extract-and-isolate-stems-with-audio-strip-pros-cons-and-alternatives-1079243

10. Voicemod: Free Real Time Voice Changer for PC & Mac [Електронний ресурс]. Режим доступу – https://www.voicemod.net/

11. Is Voicemod Safe for Gamers and Streamers? 11 Pros and Cons [Електронний ресурс]. Режим доступу – https://www.softlist.io/is-voicemod-safe-for-gamers-and-streamers/

12. Cleanvoice AI | Edit Audio & Video Podcast in 10 Mins, in Clicks [Електронний ресурс]. Режим доступу – https://cleanvoice.ai/

13. Cleanvoice AI Review: Features, Use Cases, Pricing [Електронний ресурс]. Режим доступу – https://www.castmagic.io/software-review/cleanvoice-ai

14. Python (programming language) [Електронний ресурс]. Режим доступу – https://en.wikipedia.org/wiki/Python_(programming_language)#Development

15. What Is Python? (Definition, Uses, Difficulty) [Електронний ресурс]. Режим доступу – https://builtin.com/software-engineering-perspectives/python

16. Python Syntax [Електронний ресурс]. Режим доступу – https://www.geeksforgeeks.org/python-syntax/

17. What Is Python Used For? A Beginner's Guide [Електронний ресурс]. Режим доступу – https://www.coursera.org/in/articles/what-is-python-used-for-a-beginners-guide-to-using-python

18. Why Python is the Perfect Choice for API Development [Електронний ресурс]. Режим доступу – https://www.linkedin.com/pulse/why-python-perfect-choice-api-development-vibidsoft

19. Python APIs [Електронний ресурс]. Режим доступу – https://rapidapi.com/collection/list-of-python-apis

20. An Introduction to REST API with Python [Електронний ресурс]. Режим доступу – https://www.integrate.io/blog/an-introduction-to-rest-api-with-python/

21. Visual Studio Code [Електронний ресурс]. Режим доступу – http://en.wikipedia.org/wiki/Visual_Studio_Code

22. Python Development in Visual Studio Code [Електронний ресурс]. Режим доступу – https://realpython.com/python-development-visual-studio-code/

23. REST API: Key Concepts, Best Practices, and Benefits [Електронний ресурс]. Режим доступу – https://www.altexsoft.com/blog/rest-api-design/

24. REST API: what it is, how it works, advantages and disadvantages [Електронний ресурс]. Режим доступу – https://www.thepowermba.com/en/blog/rest-api-what-it-is

25. Advantages and Disadvantages of REST API [Електронний ресурс]. Режим доступу – https://dev.to/vishalpawar1010/advantages-and-disadvantages-of-rest-api-365l

26. What Is SOAP API? - Definition - Codeless Platforms [Електронний ресурс]. Режим доступу – https://www.codelessplatforms.com/docs/knowledge-base/glossary-of-terms/what-is-soap-api/

27. Exploring SOAP API: Advantages and Implementation Strategies [Електронний ресурс]. Режим доступу – https://mailchimp.com/resources/what-is-soap-api/

28. Key Differences Between REST and SOAP APIs - Postman Blog [Електронний ресурс]. Режим доступу – https://blog.postman.com/soap-vs-rest/

29. REST API vs SOAP: A Comprehensive Comparison for Developers [Електронний ресурс]. Режим доступу – https://ultahost.com/blog/soap-vs-rest-api/

30. REST, SOAP, and HTTP APIs: What's the difference? [Електронний ресурс]. Режим доступу – https://qase.io/blog/rest-vs-soap-vs-http-api/

31. WAV [Електронний ресурс]. Режим доступу – https://uk.wikipedia.org/wiki/WAV

32. MP3 [Електронний ресурс]. Режим доступу – https://uk.wikipedia.org/wiki/MP3

33. Ogg Vorbis [Електронний ресурс]. Режим доступу – https://uk.wikipedia.org/wiki/Ogg_Vorbis

34. Metadata [Електронний ресурс]. Режим доступу – https://en.wikipedia.org/wiki/Metadata

35. Metadata API Queries - World Bank Data Help Desk [Електронний ресурс]. Режим доступу – https://datahelpdesk.worldbank.org/knowledgebase/articles/1886695-metadata-api-queries

36. Flask [Електронний ресурс]. Режим доступу – https://uk.wikipedia.org/wiki/Flask

37. Створення RESTful API за допомогою Python і Flask [Електронний ресурс]. Режим доступу – https://uk.sharpcoderblog.com/blog/creating-restful-apis-with-python-and-flask

38. flask.send_file [Електронний ресурс]. Режим доступу – https://tedboy.github.io/flask/generated/flask.send_file.html

39. resources [Електронний ресурс]. Режим доступу – https://flask-restful.readthedocs.io/en/latest/quickstart.html

40. Flask-RESTful [Електронний ресурс]. Режим доступу – https://flask-restful.readthedocs.io/en/latest/

41. python-soundfile — python-soundfile 0.13.1 documentation [Електронний ресурс]. Режим доступу – https://python-soundfile.readthedocs.io/en/0.13.1/

42. os — Miscellaneous operating system interfaces [Електронний ресурс]. Режим доступу – https://docs.python.org/uk/3.9/library/os.html

43. io — Core tools for working with streams [Електронний ресурс]. Режим доступу – https://docs.python.org/uk/3.8/library/io.html

44. Introducing Pedalboard: Spotify's Audio Effects Library for Python [Електронний ресурс]. Режим доступу – https://engineering.atspotify.com/2021/09/introducing-pedalboard-spotifys-audio-effects-library-for-python

45. Noisereduce [Електронний ресурс]. Режим доступу – https://pypi.org/project/noisereduce/

46. uuid — UUID objects according to RFC 4122 [Електронний ресурс]. Режим доступу – https://docs.python.org/uk/3.13/library/uuid.html

47. file_picker | Flutter package [Електронний ресурс]. Режим доступу – https://pub.dev/packages/file_picker

48. path | Dart package [Електронний ресурс]. Режим доступу – https://pub.dev/packages/path

49. path_provider | Flutter package [Електронний ресурс]. Режим доступу – https://pub.dev/packages/path_provider

50. dart:io [Електронний ресурс]. Режим доступу – https://dart.dev/libraries/dart-io

51. http | Dart package [Електронний ресурс]. Режим доступу – https://pub.dev/packages/http

52. audioplayers | Flutter package [Електронний ресурс]. Режим доступу – https://pub.dev/packages/audioplayers

53. permission_handler | Flutter package [Електронний ресурс]. Режим доступу – https://pub.dev/packages/permission_handler

Додаток А
(обов'язковий)

Технічне завдання

Зміст

					ВКРБ-123.25.0002.00.00.ТЗ	
Вим.	Арк.	№ документа	Підпис	Дата		
Розробив		Барамба А.А.			Програмне забезпечення API порталу доступу та обробки звукової інформації	Літ. Б; Аркуш 1; Аркушів 6
Перевірив		Дрєєв О.М.				
Н. Контр.		Коваленко А.С.				ЦНТУ КІ-21-1
Затв.		Смірнов О.А.				

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку API порталу доступу та обробки звукової інформації.

2 Підстава для розробки

Підставою для розробки служить завдання на кваліфікаційну бакалаврську роботу №46-02 від 17.01.2025 року, видане на кафедрі кібербезпеки та програмного забезпечення.

3 Мета та призначення розробки

Метою кваліфікаційної бакалаврської роботи є розробка програмного забезпечення API порталу доступу та обробки звукової інформації.

4 Джерела розробки

Джерелом цієї кваліфікаційної бакалаврської роботи є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– API порталу доступу та обробки звукової інформації;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий користувацький інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп’ютер Intel Xeon E5 2650 v2/ RAM 12 ГБ / SSD 2 ТБ / NVIDIA GeForce GTX 1660 Super, 6 ГБ або сумісні з ним.

5.8.2 Мова програмування

Програму розроблено на мовах програмування Python та Dart.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД.

7 Перелік документів, що розробляються

– Структурна схема системи.
– Функціональна схема системи.
– Діаграма процесів.
– Блок-схема алгоритму роботи програми.
– Пояснювальна записка.

8 Етапи розробки

8.1 Збір і обробка інформації по темі кваліфікаційної бакалаврської роботи. Постановка задачі на виконання кваліфікаційної бакалаврської роботи (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень кваліфікаційної бакалаврської роботи.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання кваліфікаційної бакалаврської роботи на попередній захист 24.05.2025 р.

11.2 Подання кваліфікаційної бакалаврської роботи на захист 02.06.2025 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи

за першим (бакалаврським) рівнем вищої освіти

_____ О.М. Дрєєв

Програмне забезпечення API порталу доступу та обробки звукової інформації

Лістинг програми

Код документу 12

Носій: CD/DVD-диск

Загальна кількість аркушів: 21

Літера: РП

Кропивницький – 2025 року

// main.py – Основний код роботи API

```
from flask import Flask, request, send_file
from flask_restful import Resource, Api
import request_manager as reqm
import reverb
import pitch_shift
import reduce_noise
import soundfile as sf
import os
import uuid
from io import BytesIO

app = Flask(__name__)

api = Api(app)

class ApplyEffect(Resource):
    def post(self):
        if "audio" not in request.files:
            return {"error": "File not found"}, 400

        effect_type = request.form.get("effect_type")

        recieved_file = "audio/" + str(uuid.uuid1()) + ".wav"
        reqm.get_sound(request.files, recieved_file)

        processed_file = "audio/" + str(uuid.uuid1()) + ".wav"

        settings = reqm.get_settings(effect_type, request.form)

        waveform, sample_rate = sf.read(recieved_file)

        match effect_type:
            case "reverb":
                processed_waveform = reverb.apply(waveform, sample_rate,
settings["room_size"], settings["wet_level"], settings["dry_level"])
            case "pitch_shift":
                processed_waveform = pitch_shift.apply(waveform,
sample_rate, settings["semitones"])
            case "noise_reduce":
                processed_waveform = reduce_noise.apply(waveform,
sample_rate)

        sf.write(processed_file, processed_waveform, sample_rate)

        with open(processed_file, "rb") as f:
            file_data = BytesIO(f.read())

        result_audio_name = processed_file.split('/')[1]

        os.remove(recieved_file)
        os.remove(processed_file)

        return send_file(file_data, as_attachment=True,
download_name=result_audio_name, mimetype="audio/wav")

api.add_resource(ApplyEffect, '/apply_effect')

if __name__ == '__main__':

    app.run(host="0.0.0.0", port=5000, debug = True)
```

// request_manager – Скрипт, що відповідає за прийом аудіофайлу і налаштувань до нього у вигляді метаданих

```
def get_sound(request_files, file_name):

    with open(file_name, "wb") as f:
            f.write(request_files["audio"].read())


def get_settings(effect_type, request_form):

    match effect_type:

        case "reverb":
            room_size = float(request_form.get("room_size"))/100
            wet_level = float(request_form.get("wet_level"))/100
            dry_level = float(request_form.get("dry_level"))/100
            return {"room_size": room_size, "wet_level": wet_level,
"dry_level": dry_level}

        case "pitch_shift":
            semitones = float(request_form.get("semitones"))
            return {"semitones": semitones}
```

// reverb.py – Скрипт, що відповідає за застосування ефекту реверберації до аудіофайлу

```
from pedalboard import Pedalboard, Reverb

def apply(waveform, sample_rate, room_size, wet_level, dry_level):

    reverb = Reverb(room_size=room_size, wet_level=wet_level,
dry_level=dry_level, width=1.0)
    board = Pedalboard([reverb])

    return board.process(waveform, sample_rate)
```

// pitch_shift.py – Скрипт, що відповідає за застосування ефекту підняття тону до аудіофайлу

```
from pedalboard import Pedalboard, PitchShift

def apply(waveform, sample_rate, semitones):

    pitch_shift = PitchShift(semitones=semitones)
    board = Pedalboard([pitch_shift])

    return board.process(waveform, sample_rate)
```

// reduce_noise.py – Скрипт, що відповідає за зменшення фонового шуму аудіофайлу

```
import noisereduce as rn

def apply(waveform, sample_rate):

    return rn.reduce_noise(y=waveform, sr=sample_rate)
```

// main.dart – Скрипт, з якого налаштовується та запускається мобільний застосунок

```
import 'package:flutter/material.dart';
import 'package:process_your_sound/screens/effects_screen.dart';
import 'package:process_your_sound/screens/reverb_screen.dart';
import 'package:process_your_sound/screens/pitch_shift_screen.dart';
import 'package:process_your_sound/screens/reduce_noise_screen.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      theme: ThemeData.dark().copyWith(
        appBarTheme: AppBarTheme(
          titleTextStyle: TextStyle(
            color: Colors.white,
            fontFamily: 'Jaldi',
          ),
        ),
        floatingActionButtonTheme: FloatingActionButtonThemeData(
          backgroundColor: Colors.lightBlue,
          foregroundColor: Colors.white,
        ),
        textTheme: ThemeData.dark().textTheme.apply(
              bodyColor: Colors.white,
              displayColor: Colors.white,
              fontFamily: 'Jaldi',
            ),
      ),
      initialRoute: EffectsScreen.id,
      routes: {
        EffectsScreen.id: (context) => EffectsScreen(),
        ReverbScreen.id: (context) => ReverbScreen(),
        PitchShiftScreen.id: (context) => PitchShiftScreen(),
        ReduceNoiseScreen.id: (context) => ReduceNoiseScreen(),
      },
    );
  }
}
```

// components/custom_slider.dart – Кастомний віджет слайдера

```
import 'package:flutter/material.dart';

class CustomSlider extends StatelessWidget {
  const CustomSlider(
      {super.key,
      required this.minValue,
      required this.maxValue,
      required this.setting,
      required this.title,
      required this.trailingText,
      required this.onChanged});

  final double minValue;
  final double maxValue;
  final double setting;
  final String title;
  final String trailingText;
  final ValueChanged<double> onChanged;

  @override
  Widget build(BuildContext context) {
    return Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Padding(
          padding: EdgeInsets.symmetric(horizontal: 23.0),
          child: Text(
            '$title: ${setting.toInt()}$trailingText',
            style: TextStyle(fontSize: 20.0),
          ),
        ),
        Slider(
          activeColor: Colors.lightBlueAccent,
          min: minValue,
          max: maxValue,
          value: setting,
          onChanged: onChanged,
        ),
      ],
    );
  }
}
```

// components/invalid_file_AD.dart – Кастомний віджет діалогу оповіщення. Виводить на екран оповіщення про неправильно обраний файл

```
import 'package:flutter/material.dart';

class InvalidFileAD extends StatelessWidget {
  const InvalidFileAD({super.key});

  @override
  Widget build(BuildContext context) {
    return AlertDialog(
      title: Text("Invalid file!"),
      actions: [
        TextButton(
          onPressed: () => Navigator.pop(context),
          child: Text('Back to settings'),
        )
      ],
    );
  }
}
```

// components/no_sound_AD.dart – Кастомний віджет діалогу оповіщення. Виводить на екран оповіщення про відсутність завантаженого аудіофайлу

```
import 'package:flutter/material.dart';

class NoSoundAD extends StatelessWidget {
  const NoSoundAD({super.key});

  @override
  Widget build(BuildContext context) {
    return AlertDialog(
      title: Text("You haven't loaded any sound!"),
      actions: [
        TextButton(
          onPressed: () => Navigator.pop(context),
          child: Text('Back to settings'),
        )
      ],
    );
  }
}
```

// components/operation_button.dart – Кастомний віджет кнопки

```
import 'package:flutter/material.dart';

class OperationButton extends StatelessWidget {
  OperationButton({
    super.key,
    required this.title,
    required this.onPressed,
  });

  String title;
  VoidCallback onPressed;

  @override
  Widget build(BuildContext context) {
    return TextButton(
      style: TextButton.styleFrom(
        backgroundColor: Colors.lightBlue,
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(15.0),
        ),
        fixedSize: Size(
          150.0,
          100.0,
        ),
      ),
      onPressed: onPressed,
      child: Text(
        title,
        style: TextStyle(
          color: Colors.white,
          fontSize: 30.0,
        ),
        textAlign: TextAlign.center,
      ),
    );
  }
}
```

// components/select_effect_button – Кастомний віджет детектора жесту, який імітує функціонал кнопки

```
import 'package:flutter/material.dart';

class SelectEffectButton extends StatelessWidget {
  SelectEffectButton({
    super.key,
    this.icon,
    this.title,
    this.onTap,
  });

  IconData? icon;
  String? title;
  VoidCallback? onTap;

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: onTap,
      child: Container(
        margin: EdgeInsets.symmetric(vertical: 5.0),
        decoration: BoxDecoration(
          color: Colors.grey[700],
          borderRadius: BorderRadius.all(Radius.circular(30.0)),
          border: Border.all(
            width: 9.0,
            color: Colors.white30,
          ),
        ),
        height: 250.0,
        width: 300.0,
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Icon(
              icon,
              size: 70.0,
              color: Colors.lightBlueAccent,
            ),
            Text(
              title!,
              textAlign: TextAlign.center,
              style: TextStyle(
                fontSize: 40.0,
                fontWeight: FontWeight.w400,
              ),
            ),
          ],
        ),
      ),
    );
  }
}
```

// services/sound_manager.dart – Коспонент, який відповідає за завантаження аудіофайлу у застосунок, завантаження обробленого аудіофайлу у пам'ять пристрою, формування та надсилання запитів до API

```
import 'dart:io';
import 'package:file_picker/file_picker.dart';
import 'package:path/path.dart';
import 'package:http/http.dart' as http;
import 'package:path_provider/path_provider.dart';
import 'package:permission_handler/permission_handler.dart';

class SoundManager {
  File? loadedSound;
  String processedSoundPath = "empty";
  bool soundIsLoaded = false;

  var request = http.MultipartRequest(
      "POST", Uri.parse("http://192.168.0.109:5000/apply_effect"));

  Future<void> loadSound() async {
    FilePickerResult? result =
        await FilePicker.platform.pickFiles(type: FileType.any);

    if (result != null) {
      loadedSound = File(result.files.single.path!);
      soundIsLoaded = true;
    } else {
      print('Picker canceled');
    }
  }

  Future<String> downloadSound(
      String processedSoundPath, String processedSoundName) async {
    var permissionStatus = await Permission.manageExternalStorage.request();
    if (!permissionStatus.isGranted) {
      return 'Access denied';
    }

    final downloadsDir = Directory('/storage/emulated/0/Download');
    if (!downloadsDir.existsSync()) {
      return 'Downloads folder not found!';
    }

    final tempDirFile = File(processedSoundPath);
    final fileToSave = File('${downloadsDir.path}/$processedSoundName');
    await tempDirFile.copy(fileToSave.path);
    return 'File downloaded successfully!';
  }

  String loadedSoundName() {
    return basename(loadedSound!.path);
  }

  Future<void> applyReverb(
      {required double roomSize,
      required double wetLevel,
      required double dryLevel}) async {
    File? processedFile;

    request.files
        .add(await http.MultipartFile.fromPath("audio", loadedSound!.path));

    request.fields["room_size"] = roomSize.toString();
    request.fields["wet_level"] = wetLevel.toString();
    request.fields["dry_level"] = dryLevel.toString();
    request.fields["effect_type"] = "reverb";
```

```
    var response = await request.send();

    String processedSoundName =
        response.headers.values.elementAt(0).split('=')[1].replaceAll('-',
'_');

    if (response.statusCode == 200) {
      Directory tempDir = await getTemporaryDirectory();
      processedSoundPath = '${tempDir.path}/$processedSoundName';
      processedFile = File(processedSoundPath);
      await processedFile.writeAsBytes(await response.stream.toBytes());
    } else {
      print("Processing Error: ${response.statusCode}");
    }
  }

  Future<void> applyPitchShift({required double semitones}) async {
    File? processedFile;

    request.files
        .add(await http.MultipartFile.fromPath("audio", loadedSound!.path));

    request.fields["semitones"] = semitones.toString();
    request.fields["effect_type"] = "pitch_shift";

    var response = await request.send();

    if (response.statusCode == 200) {
      Directory tempDir = await getTemporaryDirectory();
      processedSoundPath = '${tempDir.path}/processed_audio.wav';
      processedFile = File(processedSoundPath);
      await processedFile.writeAsBytes(await response.stream.toBytes());
    } else {
      print("Processing Error: ${response.statusCode}");
    }
  }

  Future<void> applyNoiseReduce() async {
    File? processedFile;

    request.files
        .add(await http.MultipartFile.fromPath("audio", loadedSound!.path));

    request.fields["effect_type"] = "noise_reduce";

    var response = await request.send();

    if (response.statusCode == 200) {
      Directory tempDir = await getTemporaryDirectory();
      processedSoundPath = '${tempDir.path}/processed_audio.wav';
      processedFile = File(processedSoundPath);
      await processedFile.writeAsBytes(await response.stream.toBytes());
    } else {
      print("Processing Error: ${response.statusCode}");
    }
  }
}
```

// screens/effects_screen.dart – Вікно вибору ефекту обробки

```
import 'package:flutter/material.dart';
import 'package:process_your_sound/components/select_effect_button.dart';
import 'package:process_your_sound/screens/pitch_shift_screen.dart';
import 'package:process_your_sound/screens/reverb_screen.dart';
import 'package:process_your_sound/screens/reduce_noise_screen.dart';
import 'package:process_your_sound/services/sound_manager.dart';

class EffectsScreen extends StatelessWidget {
  EffectsScreen({super.key});

  static const String id = 'effects_screen';

  SoundManager soundManager = SoundManager();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.lightBlue,
      body: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Container(
            padding: EdgeInsets.only(
              top: 60.0,
              left: 20.0,
              right: 30.0,
              bottom: 30.0,
            ),
            child: Text(
              'Available Processing',
              style: TextStyle(
                fontSize: 40.0,
                fontWeight: FontWeight.bold,
              ),
            ),
          ),
          Expanded(
            child: Container(
              decoration: BoxDecoration(
                color: Colors.grey[800],
                borderRadius: BorderRadius.only(
                  topLeft: Radius.circular(30.0),
                  topRight: Radius.circular(30.0),
                ),
              ),
              child: Padding(
                padding: EdgeInsets.symmetric(horizontal: 15.0),
                child: ListView(
                  children: [
                    SelectEffectButton(
                      title: 'Reverb',
                      icon: Icons.waves,
                      onTap: () {
                        Navigator.pushNamed(context, ReverbScreen.id);
                      },
                    ),
                    SelectEffectButton(
                      title: 'Pitch Shift',
                      icon: Icons.align_vertical_bottom,
                      onTap: () {
                        Navigator.pushNamed(context, PitchShiftScreen.id);
                      },
                    ),
                    SelectEffectButton(
```

```
                        title: 'Reduce Background Noise',
                        icon: Icons.leak_remove,
                        onTap: () {
                          Navigator.pushNamed(context, ReduceNoiseScreen.id);
                        },
                      ),
                    ],
                  ),
                ),
              ),
            )
          ],
        ),
      );
  }
}
```

// screens/reverb_screen.dart – Вікно налаштування ефекту реверберації

```
import 'package:flutter/material.dart';
import 'package:process_your_sound/screens/result_screen.dart';
import 'package:process_your_sound/services/sound_manager.dart';
import 'package:process_your_sound/components/custom_slider.dart';
import 'package:process_your_sound/components/operation_button.dart';
import 'package:process_your_sound/components/no_sound_AD.dart';
import 'package:process_your_sound/components/invalid_file_AD.dart';
import 'package:path/path.dart' as path;

class ReverbScreen extends StatefulWidget {
  ReverbScreen({super.key});

  static const String id = 'reverb_screen';

  @override
  State<ReverbScreen> createState() => _ReverbScreenState();
}

class _ReverbScreenState extends State<ReverbScreen> {
  SoundManager soundManager = SoundManager();
  double roomSize = 50.0;
  double wetLevel = 50.0;
  double dryLevel = 50.0;
  String? loadedSound = 'No sound loaded';

  void goToResultPage() {
    Navigator.push(context, MaterialPageRoute(builder: (context) {
      return ResultScreen(processedSoundPath:
soundManager.processedSoundPath);
    }));
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(
          'Reverb',
          style: TextStyle(
            fontSize: 30.0,
          ),
        ),
        backgroundColor: Colors.lightBlue,
        centerTitle: true,
        actions: [
          TextButton(
            child: Icon(
              Icons.add_box,
              color: Colors.white,
              size: 25.0,
            ),
            onPressed: () async {
              await soundManager.loadSound();
              setState(() {
                loadedSound = 'Loaded sound:
${soundManager.loadedSoundName()}';
              });
            },
          ),
        ],
      ),
      backgroundColor: Colors.grey[800],
      body: Column(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
```

```
children: [
  Column(
    children: [
      Text(
        'Settings',
        style: TextStyle(
          fontSize: 70,
        ),
      ),
      Text(
        loadedSound!,
        style: TextStyle(fontSize: 15.0),
      ),
    ],
  ),
  CustomSlider(
    setting: roomSize,
    title: 'Room Size',
    trailingText: "%",
    onChanged: (newValue) {
      setState(() {
        roomSize = newValue;
      });
    },
    maxValue: 100,
    minValue: 0,
  ),
  CustomSlider(
    setting: wetLevel,
    title: 'Wet Level',
    trailingText: "%",
    onChanged: (newValue) {
      setState(() {
        wetLevel = newValue;
      });
    },
    maxValue: 100,
    minValue: 0,
  ),
  CustomSlider(
    setting: dryLevel,
    title: 'Dry Level',
    trailingText: "%",
    onChanged: (newValue) {
      setState(() {
        dryLevel = newValue;
      });
    },
    maxValue: 100,
    minValue: 0,
  ),
  Row(
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    children: [
      OperationButton(
        title: 'Cancel',
        onPressed: () {
          Navigator.pop(context);
        },
      ),
      OperationButton(
        title: 'Apply',
        onPressed: () async {
          if (loadedSound == 'No sound loaded') {
            showDialog(
              context: context,
```

```
                    builder: (context) {
                      return NoSoundAD();
                    },
                  );
                } else if (path.extension(loadedSound!) == '.wav' ||
                    path.extension(loadedSound!) == '.mp3' ||
                    path.extension(loadedSound!) == '.ogg') {
                  await soundManager.applyReverb(
                      roomSize: roomSize,
                      wetLevel: wetLevel,
                      dryLevel: dryLevel);

                  goToResultPage();
                } else {
                  showDialog(
                    context: context,
                    builder: (context) {
                      return InvalidFileAD();
                    },
                  );
                }
              },
            ),
          ],
        )
      ],
    ),
  );
 }
}
```

// screens/pitch_shift_screen.dart – Вікно налаштування ефекту зміни тону

```
import 'package:path/path.dart' as path;
import 'package:flutter/material.dart';
import 'package:process_your_sound/screens/result_screen.dart';
import 'package:process_your_sound/services/sound_manager.dart';
import 'package:process_your_sound/components/custom_slider.dart';
import 'package:process_your_sound/components/operation_button.dart';
import 'package:process_your_sound/components/no_sound_AD.dart';
import 'package:process_your_sound/components/invalid_file_AD.dart';

class PitchShiftScreen extends StatefulWidget {
  const PitchShiftScreen({super.key});

  static const String id = 'pitch_shift_screen';

  @override
  State<PitchShiftScreen> createState() => _PitchShiftScreenState();
}

class _PitchShiftScreenState extends State<PitchShiftScreen> {
  SoundManager soundManager = SoundManager();
  double semitones = 0.0;
  String? loadedSound = 'No sound loaded';

  void goToResultPage() {
    Navigator.push(context, MaterialPageRoute(builder: (context) {
      return ResultScreen(processedSoundPath:
soundManager.processedSoundPath);
    }));
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(
          'Pitch Shift',
          style: TextStyle(
            fontSize: 30.0,
          ),
        ),
        backgroundColor: Colors.lightBlue,
        centerTitle: true,
        actions: [
          TextButton(
            child: Icon(
              Icons.add_box,
              color: Colors.white,
              size: 25.0,
            ),
            onPressed: () async {
              await soundManager.loadSound();
              setState(() {
                loadedSound = 'Loaded sound:
${soundManager.loadedSoundName()}';
              });
            },
          ),
        ],
      ),
      backgroundColor: Colors.grey[800],
      body: Column(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        children: [
          Column(
```

```
            children: [
              Text(
                'Settings',
                style: TextStyle(
                  fontSize: 70,
                ),
              ),
              Text(
                loadedSound!,
                style: TextStyle(fontSize: 15.0),
              ),
            ],
          ),
          CustomSlider(
            setting: semitones,
            title: 'Semitones',
            trailingText: "",
            onChanged: (newValue) {
              setState(() {
                semitones = newValue;
              });
            },
            maxValue: 25,
            minValue: -25,
          ),
          Row(
            mainAxisAlignment: MainAxisAlignment.spaceEvenly,
            children: [
              OperationButton(
                title: 'Cancel',
                onPressed: () {
                  Navigator.pop(context);
                },
              ),
              OperationButton(
                title: 'Apply',
                onPressed: () async {
                  if (loadedSound == 'No sound loaded') {
                    showDialog(
                      context: context,
                      builder: (context) {
                        return NoSoundAD();
                      },
                    );
                  } else if (path.extension(loadedSound!) == '.wav' ||
                      path.extension(loadedSound!) == '.mp3' ||
                      path.extension(loadedSound!) == '.ogg') {
                    await soundManager.applyPitchShift(
                      semitones: semitones,
                    );
                    goToResultPage();
                  } else {
                    showDialog(
                      context: context,
                      builder: (context) {
                        return InvalidFileAD();
                      },
                    );
                  }
                },
              ),
            ],
          )
        ],
      ),
    );
```

```
    }
}
```

// screens/reduce_noise_screen.dart – Вікно налаштування ефекту зменшення фонового шуму

```
import 'package:flutter/material.dart';
import 'package:process_your_sound/screens/result_screen.dart';
import 'package:process_your_sound/services/sound_manager.dart';
import 'package:process_your_sound/components/operation_button.dart';
import 'package:process_your_sound/components/no_sound_AD.dart';
import 'package:process_your_sound/components/invalid_file_AD.dart';
import 'package:path/path.dart' as path;

class ReduceNoiseScreen extends StatefulWidget {
  const ReduceNoiseScreen({super.key});

  static const String id = 'reduce_noise_screen';

  @override
  State<ReduceNoiseScreen> createState() => _ReduceNoiseScreenState();
}

class _ReduceNoiseScreenState extends State<ReduceNoiseScreen> {
  SoundManager soundManager = SoundManager();
  String? loadedSound = 'No sound loaded';

  void goToResultPage() {
    Navigator.push(context, MaterialPageRoute(builder: (context) {
      return ResultScreen(processedSoundPath:
soundManager.processedSoundPath);
    }));
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(
          'Reduce Noise',
          style: TextStyle(
            fontSize: 30.0,
          ),
        ),
        backgroundColor: Colors.lightBlue,
        centerTitle: true,
        actions: [
          TextButton(
            child: Icon(
              Icons.add_box,
              color: Colors.white,
              size: 25.0,
            ),
            onPressed: () async {
              await soundManager.loadSound();
              setState(() {
                loadedSound = 'Loaded sound:
${soundManager.loadedSoundName()}';
              });
            },
          ),
        ],
      ),
      backgroundColor: Colors.grey[800],
      body: Column(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        children: [
          Column(
            children: [
```

```
            Text(
              'Settings',
              style: TextStyle(
                fontSize: 70,
              ),
            ),
            Text(
              loadedSound!,
              style: TextStyle(fontSize: 15.0),
            ),
          ],
        ),
        Row(
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
          children: [
            OperationButton(
              title: 'Cancel',
              onPressed: () {
                Navigator.pop(context);
              },
            ),
            OperationButton(
              title: 'Apply',
              onPressed: () async {
                if (loadedSound == 'No sound loaded') {
                  showDialog(
                    context: context,
                    builder: (context) {
                      return NoSoundAD();
                    },
                  );
                } else if (path.extension(loadedSound!) == '.wav' ||
                    path.extension(loadedSound!) == '.mp3' ||
                    path.extension(loadedSound!) == '.ogg') {
                  await soundManager.applyNoiseReduce();

                  goToResultPage();
                } else {
                  showDialog(
                    context: context,
                    builder: (context) {
                      return InvalidFileAD();
                    },
                  );
                }
              },
            ),
          ],
        )
      ],
    ),
  );
 }
}
```

// screens/result_screen.dart – Вікно результату обробки

```
import 'package:audioplayers/audioplayers.dart';
import 'package:flutter/material.dart';
import 'package:path/path.dart' as path;
import 'package:process_your_sound/components/operation_button.dart';
import 'package:process_your_sound/screens/effects_screen.dart';
import 'package:process_your_sound/services/sound_manager.dart';

class ResultScreen extends StatefulWidget {
  ResultScreen({super.key, required this.processedSoundPath});

  static const String id = 'result_screen';
  String processedSoundPath;

  @override
  State<ResultScreen> createState() => _ResultScreenState();
}

class _ResultScreenState extends State<ResultScreen> {
  @override
  void initState() {
    super.initState();

    setAudio();

    audioPlayer.onPlayerStateChanged.listen((state) {
      setState(() {
        isPlaying = state == PlayerState.playing;
      });
    });

    audioPlayer.onDurationChanged.listen((newDuration) {
      setState(() {
        duration = newDuration;
      });
    });

    audioPlayer.onPositionChanged.listen((newPosition) {
      setState(() {
        position = newPosition;
      });
    });
  }

  Future setAudio() async {
    audioPlayer.setSource(DeviceFileSource(widget.processedSoundPath));
  }

  SoundManager soundManager = SoundManager();

  AudioPlayer audioPlayer = AudioPlayer();

  bool isPlaying = false;

  Duration duration = Duration.zero;

  Duration position = Duration.zero;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        automaticallyImplyLeading: false,
        title: Text(
          'Result',
```

```
            style: TextStyle(
              fontSize: 30.0,
            ),
          ),
          backgroundColor: Colors.lightBlue,
          centerTitle: true,
        ),
        backgroundColor: Colors.grey[800],
        body: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Column(
              children: [
                Slider(
                  activeColor: Colors.lightBlueAccent,
                  inactiveColor: Colors.grey[700],
                  min: 0,
                  max: duration.inSeconds.toDouble(),
                  value: position.inSeconds.toDouble(),
                  onChanged: (value) async {
                    final position = Duration(seconds: value.toInt());
                    await audioPlayer.seek(position);

                    await audioPlayer.resume();
                  },
                ),
              ],
            ),
            SizedBox(height: 40.0),
            CircleAvatar(
              backgroundColor: Colors.lightBlue,
              radius: 55.0,
              child: IconButton(
                icon: Icon(
                  isPlaying ? Icons.pause : Icons.play_arrow_rounded,
                  color: Colors.white,
                  size: 70.0,
                ),
                onPressed: () async {
                  if (isPlaying) {
                    await audioPlayer.pause();
                  } else {
                    await audioPlayer.resume();
                  }
                },
              ),
            ),
            SizedBox(height: 70.0),
            Row(
              mainAxisAlignment: MainAxisAlignment.spaceEvenly,
              children: [
                OperationButton(
                  title: 'To menu',
                  onPressed: () {
                    audioPlayer.pause();
                    Navigator.push(context, MaterialPageRoute(builder:
(context) {
                      return EffectsScreen();
                    }));
                  },
                ),
                OperationButton(
                  title: 'Download',
                  onPressed: () async {
                    String downloadResult = await soundManager.downloadSound(
                        widget.processedSoundPath,
```

```
                        path.basename(widget.processedSoundPath));
                    showDialog(
                      context: context,
                      builder: (context) {
                        return AlertDialog(
                          title: Text(downloadResult),
                          actions: [
                            TextButton(
                              onPressed: () {
                                Navigator.push(context,
                                    MaterialPageRoute(builder: (context) {
                                  return EffectsScreen();
                                }));
                              },
                              child: Text('Back to menu'),
                            )
                          ],
                        );
                      },
                    );
                  },
                ),
              ],
            )
          ],
        ),
      );
    }
}
```