МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ГЛУХЕНЬКИЙ ВЛАС ВАДИМОВИЧ

Допускається до захисту:
в.о. завідувача кафедри інформаційних технологій
д-р. техн. наук, професор
_____Наталія ВЕСЕЛОВСЬКА
«____»_____ 2026 р.

РОЗРОБКА ВЕБ-ПЛАТФОРМИ МЕНЕДЖМЕНТУ ТА СТРИМІНГУ АУДІО КОНТЕНТУ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Юрій ПОРЕМСЬКИЙ, старший
викладач кафедри
інформаційних технологій,
канд. техн. наук
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____

Вінниця – 2026

АНОТАЦІЯ

Глухенький В. В. Розробка веб-платформи менеджменту та стрімінгу аудіо контенту. Спеціальність 122 «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця, 2026.

У кваліфікаційній (бакалаврській) роботі досліджено принципи розробки веб-платформ для керування, зберігання та потокового відтворення аудіоконтенту. У межах роботи розроблено веб-платформу SonicFlow, що забезпечує перегляд музичного контенту, пошук аудіозаписів, роботу з бібліотекою, перегляд нещодавно прослуханих треків, створення плейлістів, а також взаємодію з аудіоконтентом через зручний веб-інтерфейс.

Клієнтська частина реалізована з використанням React, Vite, JavaScript, Tailwind CSS та Radix UI, що забезпечує компонентну побудову інтерфейсу, адаптивність сторінок і зручну взаємодію користувача з веб-платформою. Серверна частина розроблена з використанням Python Flask і Flask-CORS, що забезпечує обробку запитів, взаємодію з клієнтською частиною та передавання даних через API. Для збереження даних про користувачів, виконавців, жанри, альбоми, аудіозаписи, плейлісти, вподобані треки та історію прослуховування використано реляційну базу даних PostgreSQL, а взаємодію сервера з базою даних реалізовано за допомогою бібліотеки psycopg2.

Ключові слова: веб-платформа, аудіоконтент, стрімінг, музичний сайт, React, Vite, JavaScript, Tailwind CSS, Python Flask, PostgreSQL.

ABSTRACT

Hlukhenkyi V. V. Development of a web platform for audio content management and streaming. Specialty 122 "Computer Science". Vasyl' Stus Donetsk National University, Vinnytsia, 2026.

The bachelor's qualification work studies the principles of developing web platforms for managing, storing and streaming audio content. Within the framework of the work, the SonicFlow web platform was developed. It provides music content browsing, audio track search, library management, viewing recently played tracks,

playlist creation, and interaction with audio content through a convenient web interface.

The client side is implemented using React, Vite, JavaScript, Tailwind CSS and Radix UI, which ensures a component-based interface structure, page responsiveness and convenient user interaction with the web platform. The server side is developed using Python Flask and Flask-CORS, which provide request processing, interaction with the client side and data transfer through the API. A PostgreSQL relational database is used to store data about users, artists, genres, albums, audio tracks, playlists, liked tracks and listening history. The interaction between the server and the database is implemented using the psycopg2 library.

Keywords: web platform, audio content, streaming, music website, React, Vite, JavaScript, Tailwind CSS, Python Flask, PostgreSQL.

ЗМІСТ

ВСТУП 5

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ВЕБ-ПЛАТФОРМ МЕНЕДЖМЕНТУ ТА СТРІМІНГУ АУДІОКОНТЕНТУ 9

1.1. 10

1.2. Аналіз сучасних підходів до організації аудіострімінгових платформ 11

1.3. Огляд існуючих веб-платформ для прослуховування та керування аудіоконтентом 14

1.4. Порівняльна характеристика існуючих рішень 19

1.5. Моделі монетизації аудіоконтенту на веб-платформах 23

1.6. Визначення функціональних та нефункціональних вимог до веб-платформи 27

Висновки до розділу 1 32

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-ПЛАТФОРМИ МЕНЕДЖМЕНТУ ТА СТРІМІНГУ АУДІОКОНТЕНТУ 34

2.1. Загальна архітектура веб-платформи 34

2.2. Обґрунтування вибору програмних засобів розробки 38

2.3. Проєктування клієнтської частини веб-платформи 41

2.4. Проєктування серверної частини та API 45

2.5. Проєктування бази даних веб-платформи 49

2.6. Розробка модуля керування аудіоконтентом 52

2.7. Розробка модуля потокового відтворення аудіо 56

2.8. Розробка користувацького інтерфейсу веб-платформи 59

Висновки до розділу 2 63

РОЗДІЛ 3. ТЕСТУВАННЯ ТА ОЦІНКА РОБОТИ ВЕБ-ПЛАТФОРМИ МЕНЕДЖМЕНТУ ТА СТРІМІНГУ АУДІОКОНТЕНТУ 65

3.1. Перевірка працездатності основних функцій веб-платформи 65

3.2. Тестування реєстрації, авторизації та особистого кабінету користувача 68

3.3. Тестування пошуку, фільтрації та категоризації аудіофайлів 72

3.4. Тестування модуля потокового відтворення аудіо 76

3.5. Тестування створення та керування плейлістами 77

3.6. Аналіз безпеки, стабільності та продуктивності веб-платформи 80

3.7. Оцінка результатів розробки 83

Висновки до розділу 3 85

ВИСНОВКИ 87

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ 90

ВСТУП

Актуальність теми дослідження. У сучасному інформаційному суспільстві цифрові технології відіграють важливу роль у сфері створення, поширення та споживання мультимедійного контенту. Особливо швидкими темпами розвивається напрям аудіоконтенту, до якого належать музичні композиції, подкасти, аудіозаписи, навчальні матеріали, авторські добірки та інші цифрові звукові файли. Користувачі дедалі частіше надають перевагу веб-платформам, які дають змогу швидко знаходити, прослуховувати, зберігати та впорядковувати аудіоматеріали без необхідності завантаження окремого програмного забезпечення.

Розробка веб-платформ менеджменту та стрімінгу аудіоконтенту є актуальним напрямом розвитку сучасних інформаційних систем. Такі платформи забезпечують зручну взаємодію користувача з аудіофайлами, дають змогу організовувати власну бібліотеку, створювати плейлісти, здійснювати пошук треків, переглядати доступний контент та виконувати потокове відтворення аудіо без попереднього повного завантаження файлу на пристрій. Завдяки цьому підвищується зручність використання цифрового сервісу, скорочується час доступу до потрібного аудіоматеріалу та створюються умови для подальшого розвитку персоналізованих музичних і аудіосервісів.

Особливе значення має не лише відтворення аудіофайлів, а й ефективне керування ними. Для цього веб-платформа повинна забезпечувати збереження даних про аудіозаписи, формування списків відтворення, організацію користувацької бібліотеки, облік історії прослуховування, роботу з альбомами, виконавцями та плейлістами. Наявність таких функцій дозволяє розглядати платформу не як звичайний аудіоплеєр, а як повноцінну інформаційну систему для роботи з аудіоконтентом.

З технічного боку, розробка веб-платформи менеджменту та стрімінгу аудіоконтенту потребує застосування сучасних засобів веб-програмування, серверних технологій і систем управління базами даних. Використання React,

Vite, JavaScript і Tailwind CSS дає змогу створити зручний, адаптивний і динамічний користувацький інтерфейс. Серверна частина, реалізована за допомогою Python Flask і Flask-CORS, забезпечує обробку запитів, взаємодію з клієнтською частиною, отримання інформації про аудіоконтент та передавання даних через API. Для збереження відомостей про користувачів, виконавців, жанри, альбоми, аудіозаписи, плейлісти, вподобані треки та історію прослуховування використовується реляційна система управління базами даних PostgreSQL, а взаємодію серверної частини з базою даних реалізовано за допомогою бібліотеки psycopg2.

Метою дослідження є проєктування та розробка веб-платформи менеджменту та стрімінгу аудіоконтенту, яка забезпечує перегляд і пошук аудіозаписів, роботу з виконавцями, альбомами та плейлістами, збереження даних у базі PostgreSQL та потокове відтворення аудіофайлів через веб-інтерфейс.

Завдання дослідження:

1. Проаналізувати сучасні веб-платформи для прослуховування, зберігання та керування аудіоконтентом, визначити їхні основні функціональні можливості, переваги та недоліки.
2. Дослідити особливості організації потокового відтворення аудіоконтенту у веб-середовищі та визначити вимоги до платформи, що розробляється.
3. Обґрунтувати вибір технологічного набору для створення веб-платформи, зокрема використання React, Vite, JavaScript, Tailwind CSS, Python Flask, Flask-CORS, PostgreSQL та psycopg2.
4. Спроєктувати загальну архітектуру веб-платформи, визначити основні модулі системи, зокрема модуль керування аудіоконтентом, модуль пошуку, модуль плейлістів, модуль взаємодії з базою даних та модуль потокового відтворення аудіо.
5. Розробити структуру реляційної бази даних PostgreSQL для зберігання відомостей про користувачів, виконавців, жанри, альбоми, аудіозаписи, плейлісти, вподобані треки та історію прослуховування.

6. Реалізувати клієнтську частину веб-платформи з використанням React, Vite, JavaScript і Tailwind CSS та забезпечити зручну взаємодію користувача з аудіоконтентом.
7. Реалізувати серверну частину веб-платформи за допомогою Python Flask для обробки запитів, взаємодії з базою даних PostgreSQL і передавання даних клієнтській частині через API.
8. Реалізувати функції пошуку, перегляду, прослуховування та впорядкування аудіоконтенту, а також роботу з альбомами, виконавцями, плейлістами та нещодавно прослуханими треками.
9. Провести тестування розробленої веб-платформи, перевірити працездатність основних функцій системи та оцінити зручність використання користувацького інтерфейсу.

Об’єкт дослідження - процес організації, керування та потокового відтворення аудіоконтенту з використанням сучасних веб-систем.

Предмет дослідження - методи, моделі та технології створення веб-платформи для менеджменту й стрімінгу аудіоконтенту, а також засоби реалізації користувацького інтерфейсу, серверної логіки та збереження даних у реляційній базі даних PostgreSQL.

Теоретичне значення роботи полягає у вивченні сучасних підходів до розробки веб-платформ для роботи з аудіоконтентом, аналізі принципів організації потокового відтворення аудіофайлів, дослідженні способів побудови клієнт-серверної архітектури та використання реляційних баз даних для зберігання інформації про користувачів і мультимедійні об’єкти.

Практична цінність роботи полягає у створенні веб-платформи SonicFlow, яка забезпечує користувачам можливість переглядати аудіоконтент, виконувати пошук аудіозаписів, прослуховувати треки, працювати з альбомами, виконавцями, плейлістами та нещодавно прослуханими композиціями через зручний веб-інтерфейс. Розроблений програмний продукт може бути використаний як основа для побудови повноцінного музичного сервісу,

платформи для подкастів, навчального аудіоресурсу або внутрішньої системи керування аудіоматеріалами.

Апробація результатів дослідження. Результати кваліфікаційної роботи не були предметом обговорення на наукових конференціях та не публікувалися у наукових виданнях.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ВЕБ-ПЛАТФОРМ МЕНЕДЖМЕНТУ ТА СТРІМІНГУ АУДІОКОНТЕНТУ

1.1. Поняття та особливості цифрового аудіоконтенту

У сучасному цифровому середовищі онлайн-контент є одним із основних видів інформаційних ресурсів, який використовується для навчання, розваг, спілкування, просування продуктів та надання послуг у мережі Інтернет. Одним із найпоширеніших різновидів такого контенту є аудіоконтент, до якого належать музичні композиції, подкасти, аудіозаписи, аудіокниги, навчальні матеріали та інші звукові файли, які можна зберігати, передавати, копіювати й відтворювати за допомогою цифрових технологій [1, 2].

Цифровий аудіоконтент - це звукова інформація, подана у цифровій формі, яку можна записувати, обробляти, зберігати, передавати через Інтернет і відтворювати на різних пристроях. На відміну від аналогових аудіозаписів, цифровий формат дає змогу швидко копіювати, редагувати, стискати та поширювати звукові матеріали з мінімальною втратою якості. Саме тому аудіоконтент став важливою частиною сучасних веб-сервісів, мобільних застосунків, освітніх платформ, музичних сервісів та інформаційних систем [1, 3].

Однією з головних переваг цифрового аудіоконтенту є його доступність. Користувачі можуть прослуховувати аудіоматеріали на комп’ютері, смартфоні, планшеті або іншому пристрої, підключеному до мережі Інтернет. Це дає можливість швидко знаходити, слухати та зберігати звукові файли незалежно від місця перебування користувача [1]. Саме тому веб-платформи для роботи з аудіоконтентом набули значного поширення, адже вони дозволяють поєднати в одному середовищі зберігання, пошук, керування та потокове відтворення аудіофайлів.

Важливою особливістю цифрового аудіоконтенту є можливість його структурованого опису за допомогою метаданих. Кожен аудіофайл може містити

не лише сам звуковий запис, а й додаткову інформацію: назву, автора, виконавця, жанр, альбом, тривалість, дату додавання, обкладинку, опис або інші характеристики. Такі дані полегшують пошук і впорядкування аудіоматеріалів, а також спрощують створення особистих бібліотек, тематичних добірок і плейлістів.

Для веб-платформи, що працює з аудіоконтентом, важливим є не лише збереження файлів, а й забезпечення зручної взаємодії користувача з ними. Користувач повинен мати змогу швидко знайти потрібний аудіозапис, переглянути інформацію про нього, додати його до бібліотеки, створити список відтворення або розпочати прослуховування без зайвих дій. Це визначає потребу у зрозумілому інтерфейсі, ефективній навігації, пошуку та категоризації аудіоматеріалів [3].

Важливою функцією сучасних аудіоплатформ є потокове відтворення аудіо. Його суть полягає в тому, що звуковий файл не потрібно повністю завантажувати на пристрій користувача перед початком прослуховування. Дані передаються поступово, тому відтворення може починатися майже одразу після вибору композиції. Такий підхід значно підвищує зручність використання аудіосервісів, оскільки користувач отримує швидкий доступ до контенту, а система може ефективніше керувати передаванням файлів [2].

Цифровий аудіоконтент також має низку технічних характеристик, які впливають на якість відтворення та ефективність його зберігання. До таких характеристик належать формат файлу, бітрейт, частота дискретизації, розмір файлу та тривалість запису. Найпоширенішими форматами аудіофайлів є MP3, WAV, AAC, OGG і FLAC. Для веб-платформ зазвичай використовуються формати, які підтримуються більшістю браузерів і забезпечують оптимальний баланс між якістю звуку та розміром файлу. Наприклад, формат MP3 є одним із найпопулярніших, оскільки має порівняно невеликий розмір і підтримується більшістю пристроїв.

Ще однією важливою можливістю цифрового аудіоконтенту є персоналізація. Веб-платформа може зберігати дії користувача: прослухані треки, обрані жанри,

додані до бібліотеки аудіозаписи або створені плейлісти. Цю інформацію можна використовувати для покращення користувацького досвіду, формування рекомендацій, відображення нещодавно прослуханих матеріалів і швидкого доступу до улюбленого контенту [4, 5].

У межах розробки веб-платформи менеджменту та стрімінгу аудіоконтенту звукові файли розглядаються не лише як окремі мультимедійні об’єкти, а як частина єдиної інформаційної системи. Така система повинна забезпечувати збереження аудіофайлів, облік метаданих, керування користувачами, створення плейлістів, пошук аудіозаписів, організацію потокового відтворення та взаємодію з базою даних. Поєднання цих можливостей дозволяє створити повноцінну веб-платформу, яка не лише відтворює аудіо, а й забезпечує комплексне керування аудіоконтентом.

Отже, цифровий аудіоконтент є важливим видом мультимедійних даних, який широко використовується в сучасних веб-сервісах. Його основними особливостями є доступність, можливість зберігання у різних форматах, використання метаданих, підтримка потокового відтворення, зручність поширення та персоналізація. Урахування цих особливостей є необхідною умовою для розробки ефективної веб-платформи менеджменту та стрімінгу аудіоконтенту.

1.2. Аналіз сучасних підходів до організації аудіострімінгових платформ

Сучасні аудіострімінгові платформи є складними веб-системами, які поєднують зберігання, пошук, керування та потокове відтворення аудіоконтенту. Їх розвиток пов’язаний із цифровізацією музичної індустрії та зміною способів споживання музики, оскільки користувачі дедалі частіше обирають онлайн-доступ до аудіоматеріалів замість локального зберігання файлів на пристроях [6]. У таких умовах аудіострімінг розглядається не лише як спосіб прослуховування музики, а як окрема модель цифрового сервісу, що забезпечує постійний доступ до великої кількості аудіоконтенту через мережу Інтернет.

Основним підходом до організації аудіострімінгової платформи є використання клієнт-серверної архітектури. Клієнтська частина відповідає за відображення інтерфейсу, взаємодію з користувачем, пошук аудіоконтенту та відтворення аудіофайлів. Серверна частина забезпечує обробку запитів, авторизацію користувачів, збереження даних, передавання аудіоматеріалів та взаємодію з базою даних. Такий підхід дозволяє розділити систему на окремі логічні компоненти, що спрощує її розробку, підтримку та подальше розширення [7].

Важливою особливістю сучасних аудіострімінгових систем є швидкий доступ до контенту. Користувач очікує, що після вибору треку або іншого аудіофайлу відтворення почнеться майже одразу, без необхідності повного завантаження файлу на пристрій. Саме тому у веб-платформах використовується потоковий підхід до передавання аудіоданих, за якого аудіофайл поступово передається із сервера до клієнта та може відтворюватися під час завантаження. Це підвищує зручність користування сервісом і робить взаємодію з аудіоконтентом більш динамічною [6, 7].

Окрему роль в організації аудіострімінгових платформ відіграє система керування аудіоконтентом. На відміну від простого аудіоплеєра, повноцінна платформа повинна забезпечувати не тільки відтворення файлів, а й їх структуроване зберігання, редагування, сортування та пошук. Для цього кожен аудіозапис має супроводжуватися метаданими: назвою, автором або виконавцем, жанром, альбомом, тривалістю, обкладинкою та іншими характеристиками. Наявність таких даних дозволяє реалізувати каталог аудіозаписів, тематичні добірки, плейлісти та зручну навігацію між матеріалами [8].

Сучасні веб-застосунки для стрімінгового прослуховування музичного контенту зазвичай містять набір базових функцій: реєстрацію й авторизацію користувачів, перегляд списку треків, пошук композицій, відтворення аудіо, створення плейлістів і збереження обраних аудіозаписів [7]. Такі функції є основою для більшості музичних онлайн-сервісів, оскільки вони забезпечують

не лише доступ до контенту, а й персоналізовану взаємодію користувача з платформою.

Одним із поширених підходів є персоналізація користувацького досвіду. Вона передбачає збереження інформації про дії користувача: прослухані треки, створені плейлісти, улюблені жанри, додані до бібліотеки композиції або нещодавно відкриті аудіоматеріали. Завдяки цьому платформа може швидше надавати доступ до потрібного контенту, показувати користувачу історію прослуховування та формувати зручне середовище для повторної взаємодії з аудіофайлами [8, 9].

Ще одним важливим підходом є організація пошуку та фільтрації аудіоконтенту. У великих аудіобібліотеках користувачу складно швидко знайти потрібний матеріал без відповідних інструментів. Тому аудіострімінгові платформи мають передбачати пошук за назвою треку, автором, жанром, альбомом або іншими параметрами. У межах розробки власної веб-платформи це означає необхідність правильного проєктування бази даних і створення серверної логіки, яка буде обробляти пошукові запити та повертати користувачу релевантні результати [7, 8].

Важливим елементом сучасних аудіоплатформ є підтримка плейлістів. Плейліст дозволяє користувачу самостійно формувати набір аудіозаписів відповідно до власних уподобань, настрою або певної тематики. Завдяки цьому аудіоконтент стає не просто набором окремих файлів, а частиною персональної бібліотеки користувача. У програмній реалізації це потребує створення зв’язків між користувачами, плейлістами та аудіозаписами в базі даних, а також реалізації функцій додавання, видалення та перегляду треків у межах конкретного плейліста [8].

Також важливим підходом є забезпечення зручного та зрозумілого інтерфейсу. Для аудіострімінгової платформи інтерфейс має бути орієнтований на швидку взаємодію з контентом: користувач повинен легко бачити список аудіозаписів, керувати відтворенням, переходити до плейлістів, виконувати пошук і переглядати власну бібліотеку. Надмірно складний або перевантажений

інтерфейс може ускладнити користування платформою, тому структура сторінок має бути логічною, а основні функції - доступними без зайвих дій [7, 9].

З технічного погляду сучасна аудіострімінгова платформа повинна мати чітко визначені компоненти: клієнтську частину, серверну частину, базу даних, сховище аудіофайлів і модуль відтворення. Клієнтська частина забезпечує відображення сторінок і роботу аудіоплеєра. Серверна частина приймає запити, працює з користувачами та аудіоконтентом. База даних зберігає структуровані відомості про користувачів, треки, плейлісти та історію прослуховування. Сховище аудіофайлів відповідає за фізичне збереження аудіоматеріалів, а модуль відтворення забезпечує доступ до файлів через веб-інтерфейс [7, 8].

Окрему увагу слід приділити масштабованості аудіострімінгових платформ. Зі збільшенням кількості користувачів, аудіозаписів і запитів до сервера система повинна зберігати стабільність роботи. Для цього ще на етапі проєктування необхідно передбачити правильну структуру бази даних, поділ функціональності на окремі модулі, оптимізацію запитів і можливість подальшого розширення функцій. Такий підхід дозволяє розглядати веб-платформу не лише як навчальний проєкт, а як основу для подальшого розвитку повноцінного музичного або аудіосервісу [8].

Отже, сучасні підходи до організації аудіострімінгових платформ базуються на клієнт-серверній архітектурі, потоковому передаванні аудіоданих, структурованому зберіганні метаданих, персоналізації користувацького досвіду, підтримці пошуку, плейлістів та адаптованого інтерфейсу. Урахування цих підходів є необхідним для розробки веб-платформи менеджменту та стрімінгу аудіоконтенту, яка має забезпечувати не тільки прослуховування аудіофайлів, а й повноцінне керування ними в межах єдиної інформаційної системи.

1.3. Огляд існуючих веб-платформ для прослуховування та керування аудіоконтентом

На сучасному ринку цифрового аудіоконтенту представлено багато веб-платформ і онлайн-сервісів, які надають користувачам можливість слухати музику, подкасти, аудіозаписи, формувати плейлісти та користуватися персоналізованими добірками. Такі сервіси відрізняються між собою набором функцій, способом організації аудіоконтенту, моделлю монетизації, дизайном інтерфейсу та рівнем персоналізації. Тому під час розробки власної веб-платформи для менеджменту та стрімінгу аудіоконтенту доцільно проаналізувати найбільш відомі рішення цього напряму, зокрема Spotify, SoundCloud, YouTube Music, Apple Music і Deezer.

Однією з найпопулярніших платформ у сфері аудіостримінгу є Spotify. Сервіс забезпечує доступ до музичних композицій, подкастів і відеоматеріалів від авторів із різних країн. Основні можливості Spotify пов’язані з прослуховуванням аудіоконтенту, створенням власних плейлістів, формуванням персональної бібліотеки та отриманням рекомендацій відповідно до музичних вподобань користувача [10]. Загальний вигляд інтерфейсу платформи Spotify доцільно представити на рисунку 1.1.

Рисунок 1.1 - Загальний вигляд інтерфейсу платформи Spotify

Характерною рисою Spotify є поєднання великого каталогу аудіоматеріалів із системою персоналізованих рекомендацій. Платформа враховує історію прослуховування користувача та на її основі пропонує добірки, які можуть відповідати його музичним смакам. Крім того, Spotify дозволяє створювати власні плейлісти, додавати композиції до улюблених, переглядати добірки за жанрами, настроєм або іншими категоріями. Для розроблюваної веб-платформи корисним є підхід Spotify до побудови особистої бібліотеки, організації плейлістів і швидкого доступу до нещодавно прослуханого контенту [10].

Ще одним відомим сервісом є SoundCloud. На відміну від багатьох інших аудіоплатформ, він орієнтований не тільки на слухачів, а й на авторів контенту: музикантів, діджеїв, подкастерів та незалежних виконавців. SoundCloud функціонує як онлайн-спільнота для творців і слухачів, де користувачі можуть відкривати нові треки, слухати їх і поширювати власні аудіозаписи [11]. Загальний вигляд інтерфейсу платформи SoundCloud можна подати на рисунку 1.2.

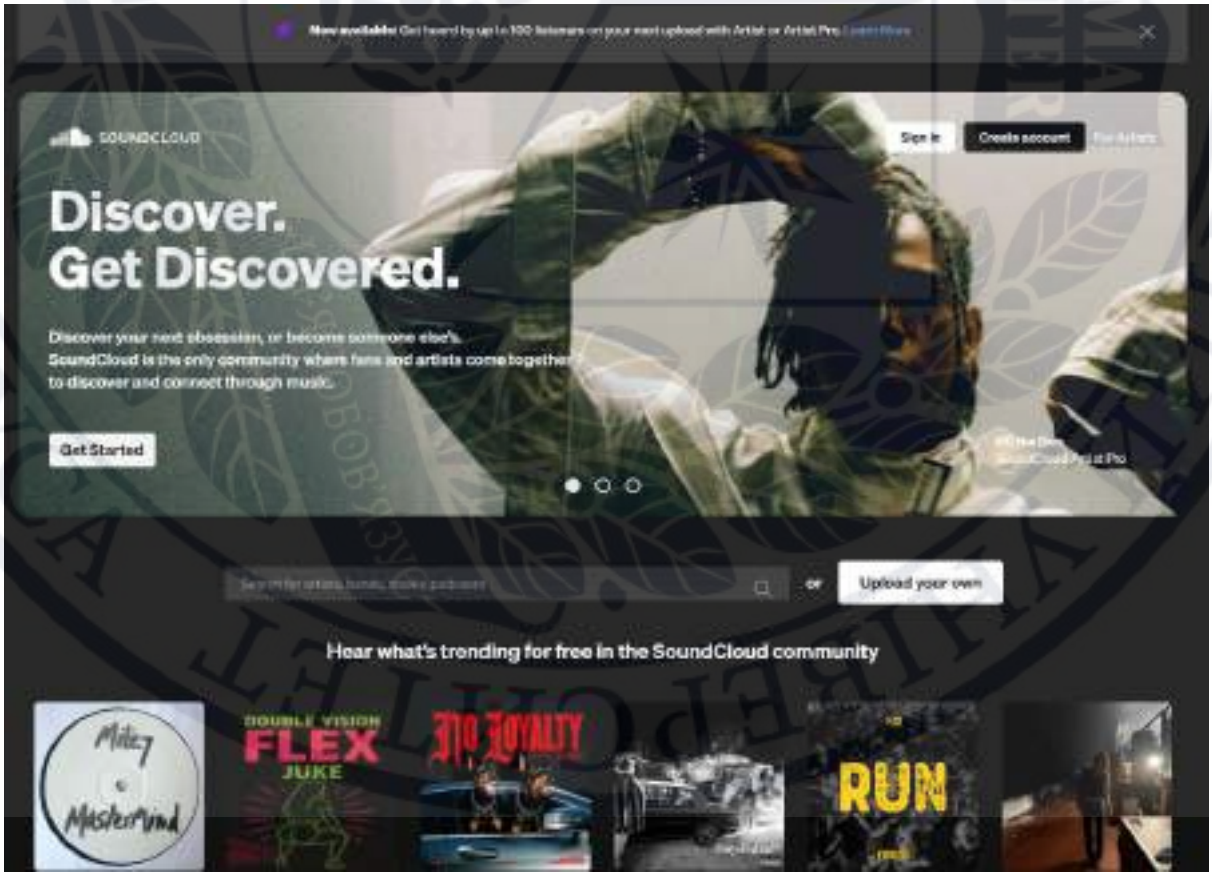

Рисунок 1.2 - Загальний вигляд інтерфейсу платформи SoundCloud

Основною відмінністю SoundCloud є можливість активного наповнення платформи авторським аудіоконтентом. Якщо Spotify більше зосереджується на масовому прослуховуванні музики, плейлістах і рекомендаціях, то SoundCloud робить акцент на публікації власних аудіозаписів, поширенні треків і взаємодії між авторами та аудиторією. У контексті цієї бакалаврської роботи такий підхід є важливим, оскільки розроблювана платформа також може розглядатися не лише як сервіс для прослуховування, а як система керування аудіоконтентом, що передбачає додавання, зберігання та впорядкування аудіофайлів [11].

YouTube Music також є прикладом сучасної платформи для прослуховування музичного контенту. Її особливість полягає у тісній інтеграції з екосистемою Google та сервісом YouTube. Завдяки цьому платформа поєднує музичні треки, відеокліпи, персональні добірки та рекомендації. YouTube Music використовує інформацію про музичні інтереси користувача для формування персональних і користувацьких міксів, які можуть відповідати певному настрою, жанру або обраним виконавцям [12]. Приклад інтерфейсу YouTube Music доцільно розмістити на рисунку 1.3.

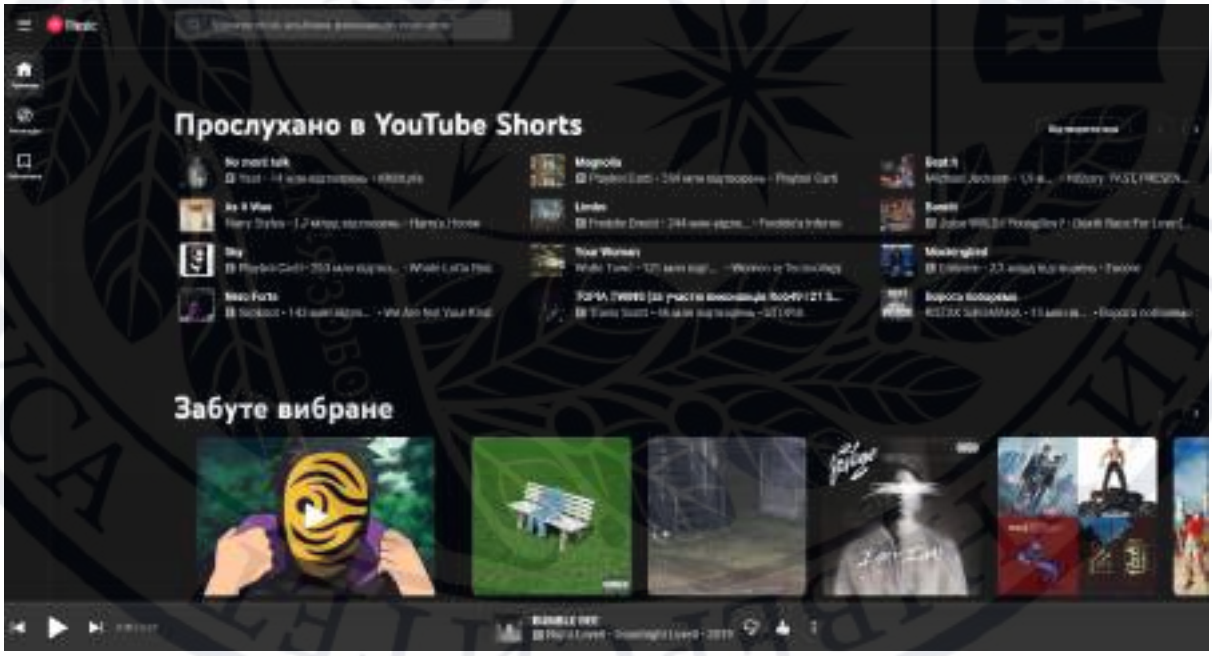

Рисунок 1.3 - Загальний вигляд інтерфейсу платформи YouTube Music

Перевагою YouTube Music є поєднання аудіоконтенту з відеоматеріалами та рекомендаційними алгоритмами YouTube. Для користувача це забезпечує зручний доступ не лише до музики, а й до кліпів, концертних виступів, реміксів та індивідуальних добірок. З погляду проєктування власної аудіоплатформи корисним є підхід YouTube Music до персоналізації, коли система враховує не тільки окремі прослухані треки, а й загальні музичні інтереси користувача [12].

Apple Music є платним музичним сервісом, який надає користувачам доступ до великої кількості композицій, плейлістів, радіостанцій, музичних відео та інших матеріалів. Згідно з офіційною інформацією Apple, користувачі можуть слухати музику на різних пристроях, користуватися ексклюзивними плейлістами, радіо та музичними відео [13]. Приклад інтерфейсу Apple Music можна представити на рисунку 1.4.

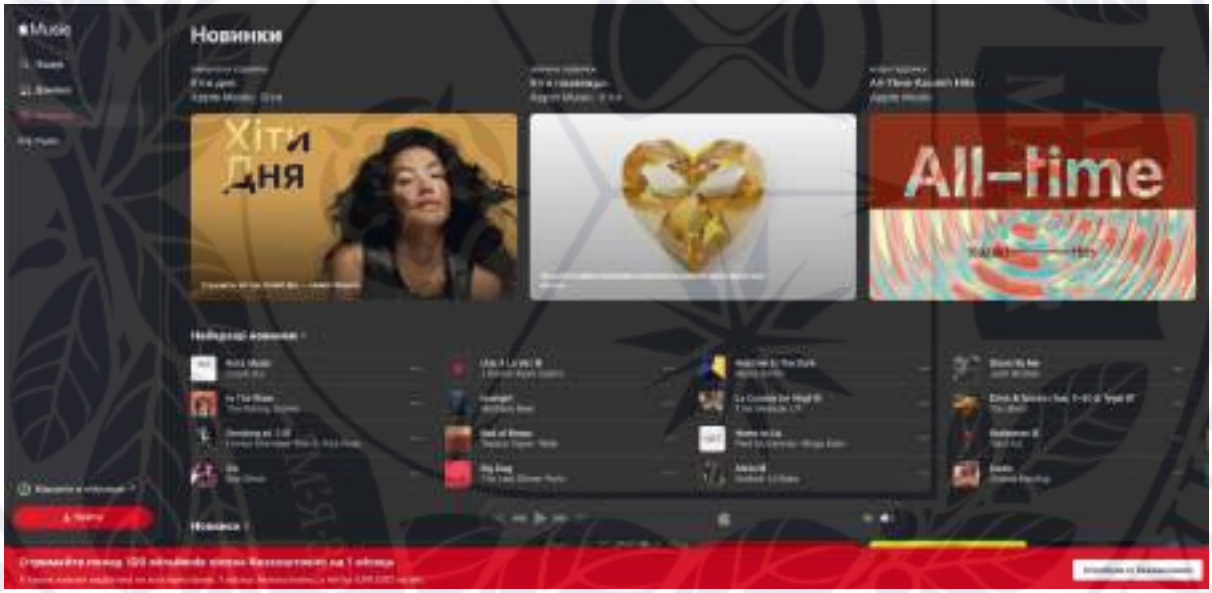

Рисунок 1.4 - Загальний вигляд інтерфейсу платформи Apple Music

Особливістю Apple Music є глибока інтеграція з екосистемою Apple, можливість синхронізації музичної бібліотеки між різними пристроями та доступ до власної музичної колекції. Такий підхід підкреслює важливість збереження персональної бібліотеки користувача, підтримки плейлістів і стабільної роботи сервісу на різних типах пристроїв. Для розроблюваної платформи це підтверджує необхідність реалізації особистого кабінету, історії

прослуховування та можливості збереження аудіозаписів у бібліотеці користувача [13, 14].

Окремо варто розглянути Deezer, який також належить до відомих аудіостримінгових сервісів. Платформа надає доступ до музики, плейлістів і подкастів, а також має персоналізовану функцію Flow, яка формує музичні рекомендації з урахуванням смаків користувача [15]. Інтерфейс платформи Deezer можна подати на рисунку 1.5.

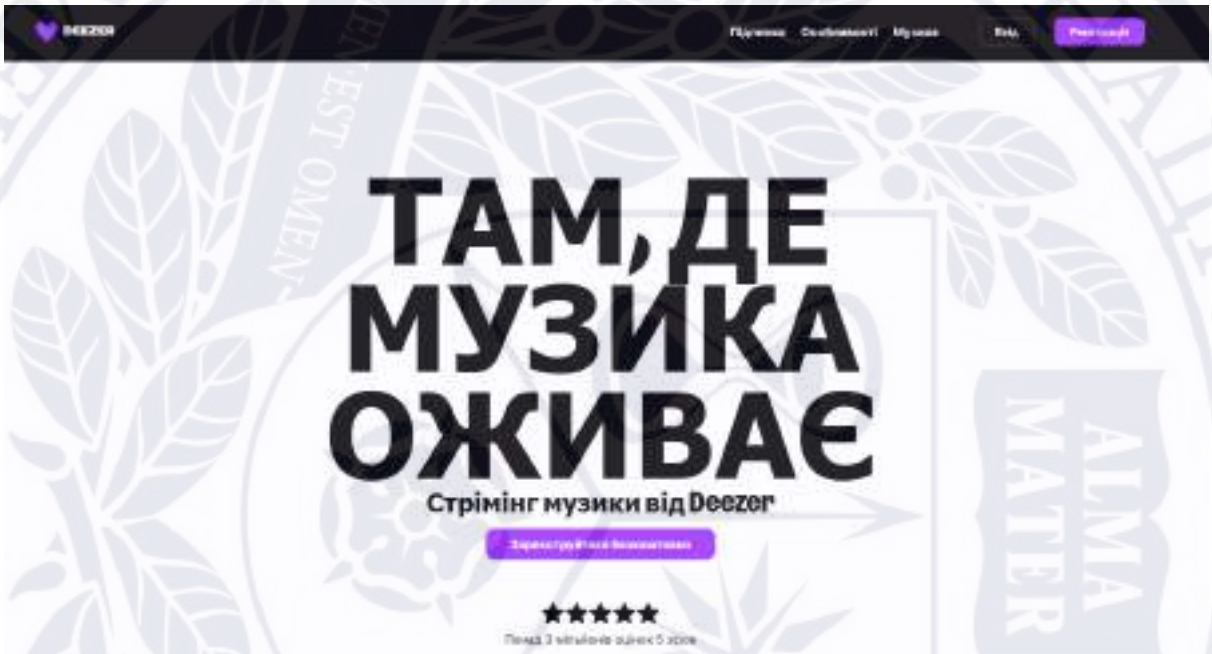

Рисунок 1.5 - Загальний вигляд інтерфейсу платформи Deezer

Сервіс Deezer демонструє значення персоналізованих музичних добірок у сучасних аудіоплатформах. Функція Flow спрямована на створення індивідуального музичного потоку, що враховує улюблені композиції, жанри та виконавців користувача [16]. Для власної веб-платформи такий підхід може бути використаний як перспективний напрям розвитку. Після реалізації базових функцій до системи можна додати рекомендаційний модуль або добірки, сформовані на основі історії прослуховування.

Проведений аналіз показує, що більшість сучасних аудіостримінгових платформ мають спільний набір функціональних можливостей. До них належать реєстрація користувачів, пошук аудіоконтенту, створення плейлістів,

збереження улюблених треків, персоналізовані рекомендації та підтримка прослуховування на різних пристроях. Водночас кожен сервіс має власний акцент: Spotify зосереджується на рекомендаціях і плейлістах, SoundCloud - на авторському контенті та взаємодії спільноти, YouTube Music - на поєднанні аудіо з відеоконтентом, Apple Music - на інтеграції з екосистемою пристроїв, а Deezer - на персоналізованих музичних добірках.

Під час розробки веб-платформи менеджменту та стрімінгу аудіоконтенту важливо врахувати функції, які є характерними для більшості аналогічних сервісів. До них можна віднести зручний інтерфейс, аудіоплеєр, пошук аудіозаписів, керування бібліотекою, створення плейлістів, авторизацію користувачів і збереження даних у базі даних. Водночас платформа, що створюється в межах бакалаврської роботи, не має повністю повторювати функціонал великих комерційних сервісів. Її основне завдання полягає у реалізації базової логіки роботи з аудіоконтентом та створенні практичної моделі керування і потокового відтворення аудіофайлів.

Отже, аналіз наявних аудіострімінгових платформ дає змогу визначити основні функціональні орієнтири для створення власного програмного продукту. На основі розгляду Spotify, SoundCloud, YouTube Music, Apple Music і Deezer можна зробити висновок, що сучасна веб-платформа для роботи з аудіоконтентом повинна поєднувати прослуховування, пошук, збереження, створення плейлістів, персоналізацію та керування користувацькими даними. Саме ці можливості доцільно врахувати під час подальшого проєктування та програмної реалізації веб-платформи менеджменту та стрімінгу аудіоконтенту.

1.4. Порівняльна характеристика існуючих рішень

Після розгляду сучасних веб-платформ, призначених для прослуховування, зберігання та керування аудіоконтентом, доцільно провести їх порівняльний аналіз. Такий аналіз дає змогу визначити функції, які найчастіше використовуються в аудіострімінгових сервісах, а також виділити підходи, які

можуть бути корисними під час створення власної веб-платформи. Для порівняння було обрано Spotify, SoundCloud, YouTube Music, Apple Music та Deezer, оскільки ці сервіси є відомими прикладами платформ для потокового відтворення аудіо та організації персональних музичних бібліотек [10–16].

Для аналізу існуючих рішень було визначено такі критерії: підтримка потокового відтворення аудіо, можливість пошуку контенту, створення плейлістів, наявність персоналізованих рекомендацій, завантаження авторських матеріалів, підтримка особистої бібліотеки, адаптація до мобільних пристроїв та модель монетизації. Обрані критерії дають змогу оцінити не лише зручність прослуховування аудіо, а й можливості керування контентом, взаємодії користувача з платформою та перспективи подальшого розвитку сервісу. Відповідно до стандарту ISO/IEC 25010, під час оцінювання програмних систем важливо враховувати функціональну придатність, зручність використання, продуктивність, безпеку, супроводжуваність і переносимість [17].

Spotify характеризується розвиненою системою потокового відтворення, персоналізованих рекомендацій і роботи з плейлістами. Платформа забезпечує доступ до музики, подкастів та інших аудіоматеріалів, а також дає змогу користувачам створювати власні колекції й отримувати рекомендації відповідно до їхніх музичних вподобань [10]. Водночас Spotify є закритим комерційним сервісом, у якому користувач не має повного контролю над структурою системи та її внутрішніми механізмами. Для розроблюваної веб-платформи корисним є підхід Spotify до формування особистої бібліотеки, плейлістів і персоналізованого доступу до аудіоконтенту.

SoundCloud вирізняється серед інших платформ тим, що значну увагу приділяє саме авторам аудіоконтенту. Сервіс дозволяє користувачам завантажувати власні треки, поширювати їх серед слухачів і взаємодіяти з аудиторією [11]. Завдяки цьому SoundCloud можна розглядати як більш відкриту платформу для незалежних виконавців, подкастерів і творців звукових матеріалів. Для власної веб-платформи такий підхід є важливим, оскільки

система менеджменту аудіоконтенту має передбачати не лише прослуховування, а й додавання, збереження, редагування та впорядкування аудіофайлів.

YouTube Music поєднує можливості аудіостримінгу з функціями відеоплатформи YouTube. Сервіс підтримує створення персональних і користувацьких міксів, які формуються з урахуванням музичних інтересів користувача [12]. Його перевагами є великий обсяг музичного та відеоконтенту, а також інтеграція з іншими сервісами Google. Проте така інтеграція одночасно робить платформу залежною від ширшої екосистеми, що не завжди є необхідним для невеликої або навчальної веб-системи. У межах розроблюваного проєкту можна врахувати підхід YouTube Music до формування рекомендацій і добірок, але основну увагу доцільно зосередити саме на аудіоконтенті.

Apple Music є прикладом сервісу, який орієнтований на тісну інтеграцію з пристроями та обліковим записом користувача. Однією з важливих можливостей платформи є синхронізація музичної бібліотеки між різними пристроями, що дозволяє користувачеві отримувати доступ до власної музики з кількох середовищ [13, 14]. Для власної платформи це підкреслює важливість реалізації особистого кабінету, збереження бібліотеки, плейлістів та історії прослуховування. Водночас Apple Music, як і Spotify, є закритим комерційним сервісом, тому його функціональність не може вільно змінюватися користувачем відповідно до індивідуальних потреб.

Deezer також підтримує потокове відтворення аудіо, пошук, створення плейлістів і персоналізовані добірки. Особливе місце в сервісі займає функція Flow, яка формує індивідуальний музичний потік на основі улюблених композицій, жанрів і виконавців користувача [15, 16]. Такий підхід демонструє важливість рекомендаційних механізмів у сучасних аудіосервісах. Для розроблюваної веб-платформи рекомендації можна розглядати як перспективний напрям подальшого розвитку після реалізації базових можливостей менеджменту та стримінгу аудіоконтенту.

Порівняльну характеристику існуючих платформ доцільно представити у вигляді таблиці 1.1.

Таблиця 1.1 — Порівняльна характеристика аудіострімінгових платформ

Критерій	Spotify	SoundCloud	YouTube Music	Apple Music	Deezer
Потокове відтворення аудіо	+	+	+	+	+
Пошук аудіоконтенту	+	+	+	+	+
Створення плейлістів	+	+	+	+	+
Персоналізовані рекомендації	+	+/-	+	+	+
Завантаження авторського контенту	-	+	+/-	-	-
Особиста бібліотека користувача	+	+	+	+	+
Підтримка подкастів	+	+	+	+/-	+
Інтеграція з відеоконтентом	+/-	-	+	+/-	-
Мобільна адаптація	+	+	+	+	+
Безкоштовний доступ	+	+	+	-	+
Платна підписка	+	+	+	+	+

Аналіз таблиці 1.1 свідчить про те, що всі розглянуті сервіси підтримують базові функції, необхідні для сучасної аудіоплатформи. До них належать потокове відтворення аудіо, пошук, створення плейлістів і наявність особистої бібліотеки користувача. Водночас між платформами є помітні відмінності. Spotify і Deezer більше орієнтовані на персоналізовані музичні добірки, SoundCloud - на завантаження та поширення авторського контенту, YouTube Music - на поєднання аудіо з відеоматеріалами, а Apple Music - на роботу в межах власної екосистеми пристроїв і сервісів [10–16].

Для розроблюваної веб-платформи менеджменту та стрімінгу аудіоконтенту найбільш важливими є реєстрація та авторизація користувачів, перегляд списку аудіозаписів, пошук контенту, створення плейлістів, збереження даних у базі, потокове відтворення аудіофайлів і зручний користувацький інтерфейс. Саме ці функції є базовими для більшості розглянутих сервісів і можуть бути реалізовані в межах бакалаврської роботи. Натомість складніші можливості, зокрема повноцінна рекомендаційна система, інтеграція з відеоконтентом або комерційна система підписок, можуть розглядатися як напрями подальшого розвитку платформи.

Окремо варто зазначити, що ефективність аудіострімінгової платформи залежить не лише від кількості реалізованих функцій, а й від зручності їх використання. Відповідно до принципів юзабіліті, інтерфейс має бути зрозумілим для користувача, відображати поточний стан системи, забезпечувати послідовність дій і зменшувати ймовірність помилок [18]. Тому під час створення власної платформи важливо не перевантажувати інтерфейс зайвими елементами, а зосередитися на основних сценаріях роботи: пошуку аудіозаписів, запуску відтворення, створенні плейлістів і керуванні особистою бібліотекою.

Отже, порівняльний аналіз існуючих сервісів дає змогу сформувати основні функціональні вимоги до власної веб-платформи. Розроблювана система повинна поєднувати потокове відтворення аудіо, керування аудіозаписами, пошук, створення плейлістів, збереження користувацьких даних і зручну

взаємодію з інтерфейсом. Саме ці можливості становлять практичну основу веб-платформи менеджменту та стрімінгу аудіоконтенту.

1.5. Моделі монетизації аудіоконтенту на веб-платформах

Монетизація аудіоконтенту є однією з важливих складових роботи сучасних аудіострімінгових платформ. Якщо на ранніх етапах розвитку цифрових сервісів основне завдання полягало у наданні користувачам доступу до музичних файлів, то сьогодні аудіоплатформи функціонують як повноцінні цифрові бізнес-моделі. Вони об'єднують прослуховування музики, роботу з авторським контентом, рекламні інструменти, платні підписки, підтримку виконавців та аналіз активності користувачів [19, 20].

Розвиток аудіостримінгу значно змінив музичну індустрію, оскільки вплинув на традиційні способи отримання прибутку від музичного контенту. Раніше основними джерелами доходу були продаж фізичних носіїв, концертна діяльність, а також реалізація окремих цифрових альбомів або треків. Натомість сьогодні важливу роль відіграють стрімінгові сервіси, платні підписки, рекламні покази та цифрові платформи для поширення музики [19]. За даними IFPI, у 2024 році світові доходи від записаної музики продовжували зростати, а стрімінг залишався одним із провідних сегментів цього ринку [21].

Однією з найбільш поширених моделей монетизації аудіоконтенту є підпискова модель. Її суть полягає в тому, що користувач сплачує щомісячну або річну плату за доступ до розширених можливостей сервісу. Зазвичай платна підписка передбачає прослуховування музики без реклами, покращену якість звуку, офлайн-доступ або додаткові інструменти для керування музичною бібліотекою. Такий підхід активно застосовують великі аудіосервіси, зокрема Spotify, Apple Music, Deezer та YouTube Music [10, 13, 15]. Для розроблюваної веб-платформи підпискова модель може розглядатися як перспективний напрям розвитку після впровадження базових функцій прослуховування, пошуку та керування аудіофайлами.

Ще одним поширеним способом отримання прибутку є рекламна монетизація. Вона використовується тоді, коли користувач має безкоштовний доступ до контенту, але під час роботи із сервісом переглядає або прослуховує рекламні матеріали. В аудіострімінгових платформах це можуть бути банери в інтерфейсі, аудіореклама між треками або рекламні вставки в подкастах. Така модель дозволяє сервісу залишатися доступним для широкого кола користувачів і водночас отримувати дохід від рекламодавців. На практиці рекламна модель часто поєднується з підписковою: безкоштовні користувачі слухають рекламу, а користувачі з платною підпискою отримують доступ до контенту без рекламних обмежень.

Важливим елементом монетизації аудіоконтенту є розподіл роялті між правовласниками, виконавцями та авторами музики. Наприклад, Spotify зазначає, що виплати не здійснюються за сталою фіксованою ставкою за одне прослуховування. Вони залежать від частки прослуховувань у загальному обсязі стрімів, а кошти спочатку надходять правовласникам, які надалі розподіляють їх між артистами відповідно до укладених угод [22]. Для невеликої навчальної платформи така модель є доволі складною, однак її варто враховувати під час аналізу комерційних сервісів, оскільки вона показує зв’язок між активністю користувачів, кількістю прослуховувань і доходами авторів.

Окрему увагу варто приділити підтримці авторів через фанатське фінансування або модель, орієнтовану на конкретного слухача. Наприклад, SoundCloud застосовує підхід fan-powered royalties, за якого дохід автора більшою мірою залежить від реальних прослуховувань його постійною аудиторією, а не лише від загальної кількості стрімів на платформі [23]. Така модель є особливо важливою для незалежних виконавців, оскільки дає змогу краще пов’язати підтримку слухачів із доходом автора. Для розроблюваної системи цей підхід може бути врахований у майбутньому, наприклад через донати, підтримку автора або внутрішню статистику прослуховувань.

Ще одним варіантом монетизації є продаж цифрового контенту. На відміну від підписковых сервісів, де користувач оплачує доступ до всієї бібліотеки,

модель прямого продажу передбачає купівлю окремих треків, альбомів або інших аудіоматеріалів. Прикладом такого підходу є Bandcamp, де користувачі можуть купувати музику безпосередньо, а значна частина коштів надходить авторам або лейблам [24]. Ця модель є корисною для платформ, які орієнтуються на незалежних музикантів, подкастерів або авторів навчального аудіоконтенту.

Для аудіоплатформ також актуальною є модель платного просування матеріалів. Вона передбачає, що автор може оплатити додаткове розміщення свого треку в рекомендаціях, тематичних добірках або на головній сторінці сервісу. Такий підхід часто застосовується в цифровому маркетингу, оскільки допомагає новим авторам швидше знаходити свою аудиторію. Водночас під час впровадження такої моделі важливо дотримуватися балансу між комерційним просуванням і якістю рекомендацій, щоб не знижувати рівень довіри користувачів до платформи.

Окремим напрямом є монетизація подкастів та авторських аудіопрограм. Подкасти можуть приносити дохід завдяки рекламі, спонсорським інтеграціям, платному доступу до окремих випусків або добровільній підтримці слухачів. Такий підхід є важливим, оскільки аудіоконтент не обмежується лише музичними треками. Веб-платформа для менеджменту та стрімінгу аудіоконтенту може використовуватися не тільки для музики, а й для лекцій, подкастів, навчальних матеріалів, аудіооглядів та інших видів звукового контенту.

У межах бакалаврської роботи головною метою розроблюваної веб-платформи є реалізація базових функцій менеджменту та стрімінгу аудіоконтенту. Тому повноцінна платіжна система або складний механізм розподілу роялті не є обов’язковими для першої версії системи. Проте архітектуру платформи доцільно проєктувати так, щоб у майбутньому її можна було доповнити фінансовими можливостями. Наприклад, у структурі бази даних можна передбачити подальше розширення для тарифних планів користувачів, статусу підписки, статистики прослуховувань, платного доступу до окремих матеріалів або фінансової підтримки авторів.

З практичної точки зору для навчальної веб-платформи можна виділити кілька перспективних напрямів монетизації. Першим напрямом є преміум-доступ, за якого користувач отримує додаткові можливості, наприклад більший обсяг для завантаження аудіофайлів, відсутність реклами або розширену бібліотеку. Другим напрямом може бути підтримка авторів через добровільні платежі або донати. Третім варіантом є платне просування аудіозаписів у пошуку чи тематичних добірках. Четвертим напрямом може виступати рекламна модель, орієнтована на безкоштовних користувачів.

У процесі проєктування власної платформи монетизацію варто розглядати не як основну функцію першої версії системи, а як один із можливих напрямів її подальшого масштабування. На початковому етапі система повинна забезпечувати стабільну роботу з користувачами, аудіофайлами, плейлістами, пошуком і потоковим відтворенням. Після цього можуть бути реалізовані додаткові модулі, зокрема облік кількості прослуховувань, поділ функцій на безкоштовні та платні, панель автора, статистика аудиторії та інтеграція з платіжними сервісами.

На рисунку 1.7 доцільно представити узагальнену схему моделей монетизації аудіоконтенту на веб-платформах.

Рисунок 1.7 - Основні моделі монетизації аудіоконтенту на веб-платформах

Отже, монетизація аудіоконтенту на веб-платформах може здійснюватися за допомогою підписок, реклами, роялті, донатів, продажу цифрових матеріалів, платного просування та спонсорських інтеграцій. Для розроблюваної веб-платформи ці моделі мають значення насамперед як напрям подальшого розвитку, оскільки на першому етапі основна увага приділяється створенню функціональної основи системи: керуванню аудіоконтентом, збереженню даних у базі, створенню плейлістів і потоковому відтворенню аудіо.

1.6. Визначення функціональних та нефункціональних вимог до веб-платформи

Після розгляду особливостей цифрового аудіоконтенту, сучасних способів організації аудіостримінгових платформ, наявних сервісів та моделей монетизації доцільно перейти до формування вимог до веб-платформи, що розробляється.

Визначення вимог є важливим етапом створення програмного продукту, оскільки саме вони окреслюють основні функції системи, її структуру, принципи взаємодії користувача з інтерфейсом, а також технічні характеристики, які необхідно забезпечити під час реалізації.

Для веб-платформи менеджменту та стрімінгу аудіоконтенту вимоги доцільно поділити на дві основні групи: функціональні та нефункціональні. Функціональні вимоги визначають конкретні можливості системи, тобто дії, які користувач або адміністратор може виконувати на платформі. Нефункціональні вимоги, у свою чергу, описують якісні характеристики системи: зручність використання, продуктивність, безпеку, стабільність, масштабованість, супроводжуваність і коректну роботу на різних пристроях. Такий підхід відповідає загальним принципам оцінювання якості програмного забезпечення, зокрема стандарту ISO/IEC 25010, у якому серед характеристик якості програмних систем виділяють функціональну придатність, продуктивність, сумісність, зручність використання, надійність, безпеку, супроводжуваність і переносимість [17].

Однією з основних функціональних вимог до розроблюваної веб-платформи є можливість перегляду та впорядкування аудіоконтенту. Користувач повинен мати змогу переглядати список доступних треків, альбомів, виконавців і плейлістів, відкривати окремі сторінки аудіоматеріалів та запускати їх відтворення через веб-інтерфейс. Реєстрація та авторизація користувачів можуть розглядатися як важливий напрям подальшого розвитку платформи, оскільки вони дозволяють персоналізувати бібліотеку, зберігати плейлісти, вподобані треки та історію прослуховування.

Наступною важливою функціональною вимогою є можливість перегляду аудіоконтенту. Користувач повинен бачити список доступних аудіозаписів із короткою інформацією про кожен файл: назвою, виконавцем або автором, жанром, тривалістю, обкладинкою та іншими метаданими. Такий підхід дає змогу не просто відображати аудіофайли, а представляти їх як структуровані об’єкти інформаційної системи. У межах розроблюваної платформи

аудіоконтент має бути поданий у зручному вигляді, щоб користувач міг швидко ознайомитися з матеріалом і перейти до його прослуховування.

Окремою функціональною вимогою є реалізація пошуку аудіозаписів. У разі збільшення кількості аудіофайлів користувачеві буде складно знаходити потрібні матеріали без пошукового механізму. Тому система повинна забезпечувати пошук за назвою треку, автором, жанром або іншими параметрами. Реалізація пошуку безпосередньо пов’язана з проєктуванням бази даних, оскільки аудіофайли мають містити структуровані метадані. Для збереження таких даних доцільно використовувати PostgreSQL, оскільки ця система управління базами даних підтримує реляційну модель, SQL-запити та роботу зі структурованими наборами даних [27].

Важливою функцією веб-платформи є потокове відтворення аудіо. Користувач повинен мати можливість запускати аудіозапис без попереднього повного завантаження файлу на свій пристрій. Для цього на клієнтській частині може використовуватися веб-аудіоплеєр, який забезпечує запуск, паузу, перемотування та відображення поточного стану відтворення. Ця функція є центральною для платформи стрімінгу аудіоконтенту, оскільки саме вона реалізує основний сценарій взаємодії користувача із сервісом.

До функціональних вимог також належить створення та керування плейлістами. Користувач повинен мати змогу створювати власні списки відтворення, додавати до них аудіозаписи, переглядати вміст плейліста та видаляти окремі треки. Плейлісти є важливим елементом персоналізації, оскільки дають користувачеві можливість самостійно впорядковувати аудіоконтент відповідно до власних уподобань. На рівні бази даних це потребує створення зв’язків між користувачами, плейлістами та аудіозаписами.

Ще однією функціональною вимогою є керування аудіоконтентом. Платформа повинна передбачати можливість додавання, редагування та видалення аудіофайлів. У межах навчального проєкту ця функція може бути реалізована для адміністратора або зареєстрованого користувача залежно від обраної логіки роботи системи. Під час додавання аудіофайлу необхідно

зберігати не тільки сам файл, а й пов’язані з ним дані: назву, автора, жанр, опис, обкладинку або шлях до файлу. Це забезпечує подальший пошук, відображення та відтворення аудіоконтенту.

До функціональних вимог доцільно також віднести наявність особистого кабінету користувача. У ньому користувач може переглядати власні дані, збережені аудіозаписи, створені плейлісти та історію прослуховування. Такий модуль підвищує зручність роботи з платформою, оскільки користувач отримує персоналізований простір для взаємодії з контентом. Крім того, особистий кабінет може стати основою для подальшого розвитку системи, наприклад додавання підписки, статистики прослуховувань або фінансової підтримки авторів.

Нефункціональні вимоги визначають якість роботи веб-платформи. Однією з ключових вимог цієї групи є зручність використання. Інтерфейс має бути зрозумілим, логічно побудованим і не перевантаженим зайвими елементами. Користувач повинен швидко знаходити основні функції: пошук, запуск аудіо, перегляд плейлістів, вхід до особистого кабінету та керування бібліотекою. Відповідно до евристик юзабіліті, інтерфейс повинен відображати стан системи, бути послідовним, допомагати користувачу уникати помилок і забезпечувати просту взаємодію з основними елементами [18].

Другою важливою нефункціональною вимогою є продуктивність. Веб-платформа повинна швидко завантажувати сторінки, коректно обробляти запити користувача, забезпечувати стабільне відтворення аудіо та не створювати зайвих затримок під час пошуку або відкриття плейлістів. На серверній частині продуктивність залежить від правильної організації маршрутів, обробки запитів і взаємодії з базою даних. Flask надає засоби для створення серверної частини веб-додатків і API, а також дозволяє описувати маршрути для обробки HTTP-запитів. У межах розроблюваної платформи Flask використовується для отримання запитів від клієнтської частини, виконання SQL-запитів до PostgreSQL та повернення результатів у форматі JSON [28, 29].

Важливою вимогою є безпека. Оскільки платформа працює з обліковими записами користувачів, паролями та персональними діями в системі, необхідно забезпечити захист доступу до даних. Паролі не повинні зберігатися у відкритому вигляді, а доступ до особистої інформації має надаватися лише відповідному користувачу. Також необхідно враховувати ризики, пов'язані з некоректною обробкою запитів, помилками авторизації, неправильним налаштуванням сервера та можливістю несанкціонованого доступу. OWASP Top 10 визначає найбільш критичні ризики безпеки веб-додатків і може використовуватися як орієнтир під час створення захищеної системи [26].

Ще однією нефункціональною вимогою є адаптивність інтерфейсу. Користувачі можуть відкривати платформу з різних пристроїв: комп'ютера, ноутбука, планшета або смартфона. Тому веб-сторінки повинні коректно відображатися на екранах різного розміру. Особливо важливо, щоб аудіоплеєр, список треків, меню, пошук і плейлісти залишалися зручними у мобільній версії. Адаптивність підвищує доступність платформи та робить її зручною для щоденного використання.

До нефункціональних вимог також належить надійність. Система повинна стабільно працювати під час виконання основних операцій: входу користувача, пошуку аудіозапису, запуску відтворення, створення плейліста або збереження даних у базі. У разі виникнення помилки користувач повинен отримати зрозуміле повідомлення, а система не повинна втрачати дані. Це особливо важливо для платформи, яка працює з медіафайлами та персональними даними користувачів.

Окремо варто виділити масштабованість і супроводжуваність. Платформа повинна мати таку структуру, щоб у майбутньому до неї можна було додати нові функції: рекомендації, монетизацію, донати, статистику прослуховувань, адміністративну панель або розширений модуль завантаження аудіофайлів. Для цього систему доцільно будувати за модульним принципом, розділяючи клієнтську частину, серверну логіку, базу даних і файлове сховище. Такий підхід спрощує подальше оновлення, підтримку та розвиток програмного продукту.

Основні функціональні вимоги до веб-платформи доцільно узагальнити у таблиці 1.2.

Таблиця 1.2 — Функціональні вимоги до веб-платформи менеджменту та стрімінгу аудіоконтенту

№	Функціональна вимога	Опис вимоги
1	Реєстрація користувача	Створення нового облікового запису в системі
2	Авторизація користувача	Вхід до системи за допомогою логіна або електронної пошти та пароля
3	Перегляд аудіоконтенту	Відображення списку доступних аудіозаписів із метаданими
4	Пошук аудіозаписів	Пошук треків за назвою, автором, жанром або іншими параметрами
5	Потокове відтворення аудіо	Прослуховування аудіофайлів через веб-інтерфейс
6	Створення плейлістів	Формування користувачем власних списків відтворення
7	Керування аудіофайлами	Додавання, редагування та видалення аудіоконтенту
8	Особистий кабінет	Перегляд користувацьких даних, бібліотеки, плейлістів та історії прослуховування

Нефункціональні вимоги до веб-платформи подано у таблиці 1.3.

Таблиця 1.3 — Нефункціональні вимоги до веб-платформи менеджменту та стрімінгу аудіоконтенту

№	Нефункціональна вимога	Опис вимоги
1	Зручність використання	Інтерфейс має бути зрозумілим, логічним і простим у використанні
2	Продуктивність	Сторінки, пошук і відтворення аудіо повинні працювати без значних затримок
3	Безпека	Система повинна захищати облікові записи та користувацькі дані
4	Адаптивність	Інтерфейс має коректно працювати на різних пристроях і розмірах екрана
5	Надійність	Платформа повинна стабільно виконувати основні функції
6	Маштабованість	Система має підтримувати можливість подальшого розширення функціоналу
7	Супроводжуваність	Код і структура системи мають бути зручними для підтримки та оновлення

Отже, сформовані функціональні та нефункціональні вимоги визначають основу для подальшого проєктування веб-платформи. Функціональні вимоги описують основні можливості системи: реєстрацію, авторизацію, перегляд, пошук, відтворення аудіо, створення плейлістів і керування аудіоконтентом. Нефункціональні вимоги визначають якість роботи платформи, зокрема її зручність, продуктивність, безпеку, адаптивність, надійність і можливість подальшого розвитку. Саме ці вимоги будуть використані під час проєктування архітектури, бази даних і програмної реалізації системи.

Висновки до розділу 1

У першому розділі було досліджено предметну область веб-платформ, призначених для менеджменту та стрімінгу аудіоконтенту. Було встановлено, що цифровий аудіоконтент є важливим різновидом мультимедійної інформації, який може зберігатися, оброблятися, передаватися через мережу Інтернет і відтворюватися на різних типах пристроїв. До цієї категорії належать музичні треки, подкасти, аудіокниги, навчальні аудіоматеріали, записи лекцій, інтерв'ю та інші звукові файли. Основними характеристиками цифрового аудіоконтенту є доступність, підтримка різних форматів аудіо, використання метаданих, можливість потокового відтворення та персоналізована взаємодія користувача з платформою [1–5].

Також було розглянуто сучасні підходи до побудови аудіострімінгових платформ. Визначено, що такі системи переважно реалізуються на основі клієнт-серверної архітектури. У такій структурі клієнтська частина відповідає за відображення інтерфейсу та взаємодію з користувачем, тоді як серверна частина забезпечує обробку запитів, авторизацію, роботу з базою даних і передавання аудіоконтенту. Крім того, встановлено, що важливими компонентами аудіоплатформ є модуль потокового відтворення, пошукова система, інструменти керування плейлістами, особиста бібліотека користувача та збереження історії прослуховування [6–9].

У процесі аналізу наявних рішень було розглянуто популярні аудіострімінгові сервіси, зокрема Spotify, SoundCloud, YouTube Music, Apple Music та Deezer. Проведений огляд показав, що ці платформи мають низку спільних функцій: пошук аудіоконтенту, потокове відтворення, створення плейлістів, формування персональних бібліотек і використання рекомендаційних механізмів. Водночас кожен сервіс має власні особливості. Spotify і Deezer зосереджуються на персоналізованих музичних добірках, SoundCloud - на авторському контенті та взаємодії між авторами й слухачами, YouTube Music - на поєднанні аудіо з відеоматеріалами, а Apple Music - на інтеграції з екосистемою пристроїв і сервісів Apple [10–16].

Крім того, було виконано порівняльну характеристику існуючих платформ за основними критеріями. До них належать підтримка потокового відтворення, пошуку, плейлістів, персоналізованих рекомендацій, завантаження авторського контенту, особистої бібліотеки, подкастів, інтеграції з відеоконтентом, мобільної адаптації та моделей доступу. На основі цього було визначено, що для розроблюваної веб-платформи найбільш важливими є реєстрація та авторизація користувачів, перегляд і пошук аудіозаписів, створення плейлістів, збереження користувацьких даних і потокове відтворення аудіофайлів [17, 18].

Окрему увагу було приділено моделям монетизації аудіоконтенту на веб-платформах. Було встановлено, що сучасні аудіосервіси можуть отримувати дохід за рахунок підписок, реклами, роялті, донатів, продажу цифрового контенту, платного просування та спонсорських інтеграцій. Для платформи, що розробляється, монетизація розглядається не як основна функція першої версії системи, а як можливий напрям подальшого розвитку. На початковому етапі головна увага зосереджується на створенні базової функціональності: керуванні аудіоконтентом, збереженні даних, створенні плейлістів і потоковому відтворенні аудіо [19–25].

Також у розділі було сформовано функціональні та нефункціональні вимоги до веб-платформи менеджменту та стрімінгу аудіоконтенту. До функціональних вимог віднесено реєстрацію користувачів, авторизацію, перегляд аудіоконтенту, пошук аудіозаписів, потокове відтворення аудіо, створення плейлістів, керування аудіофайлами та наявність особистого кабінету. Нефункціональні вимоги охоплюють зручність використання, продуктивність, безпеку, адаптивність, надійність, масштабованість і супроводжуваність системи [26–30].

Отже, у першому розділі було сформовано теоретичну й практичну основу для подальшого проєктування веб-платформи менеджменту та стрімінгу аудіоконтенту. Проведений аналіз підтвердив актуальність розробки такої системи та дав змогу визначити основні вимоги до її функціональних можливостей. Отримані результати будуть використані в наступному розділі під час проєктування архітектури, бази даних, клієнтської та серверної частини веб-платформи.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-ПЛАТФОРМИ МЕНЕДЖМЕНТУ ТА СТРІМІНГУ АУДІОКОНТЕНТУ

2.1. Загальна архітектура веб-платформи

Веб-платформа SonicFlow була спроєктована як клієнт-серверна модель для керування аудіоматеріалами та їхнім потоковим відтворенням. Її структура передбачає розчленування програмного рішення на кілька логічно відокремлених блоків: інтерфейс користувача, бекенд-частину, сховище даних, механізм відтворення аудіо та зовнішні інтеграції для доступу до музики. Такий підхід дає змогу чітко розподілити сферу відповідальності між складовими, спрощує процедури супроводу, полегшує подальше нарощування функціоналу та підвищує надійність роботи сервісу.

Фронтенд веб-платформи втілено у формі односторінкового застосунку, збудованого на базі технологій React та Vite. React використовується для формування візуального оформлення з окремих, незалежних елементів, що дозволяє розділити на самостійні модулі сторінки, картки треків, плеєр, бічне меню, верхню панель та всі спливаючі вікна [31]. Vite залучається як інструмент для розробки та пакування клієнтської частини, оскільки він гарантує швидке підняття локального оточення, підтримку актуальних JS-модулів та формування оптимізованого білду для розгортання [32].

Бекенд веб-системи реалізовано із застосуванням Python Flask. Flask являє собою мінімалістичний WSGI-фреймворк для створення веб-додатків, що забезпечує швидку розробку серверної логіки та програмних інтерфейсів (API) [33]. У рамках проєкту серверна частина відповідає за з'єднання з базою даних PostgreSQL, створення необхідних таблиць, виконання SQL-запитів та передачу оброблених даних клієнту через ендпоінти **/db-api**. Зокрема, бекенд надає API для отримання інформації про аудіозаписи, виконавців, альбоми та створені плейлісти.

Як сховище даних використовується PostgreSQL, що призначений для зберігання структурованої інформації про користувачів, митців, музичні жанри, альбоми, треки, списки відтворення, обрані композиції та історію прослуховування. PostgreSQL - це реляційна система керування базами даних, яка підтримує мову SQL, реалізацію зв'язків між сутностями, індексацію та механізми забезпечення цілісності даних [27]. У даному проєкті база містить ключові таблиці: `users`, `artists`, `genres`, `albums`, `tracks`, `track_artists`, `playlists`, `playlist_tracks`, `liked_tracks`, `recent_plays`. Така модель дозволяє зберігати медіаконтент не як набір файлів, а як цілісну інформаційну модель із взаємопов'язаними об'єктами.

Загалом, архітектурну схему веб-платформи SonicFlow доречно проілюструвати на рисунку 2.1. На цій діаграмі слід відобразити ключові складові системи: кінцевого користувача, клієнтський інтерфейс (React/Vite), серверний API (Flask), сховище PostgreSQL, модуль відтворення та зовнішні музичні джерела.

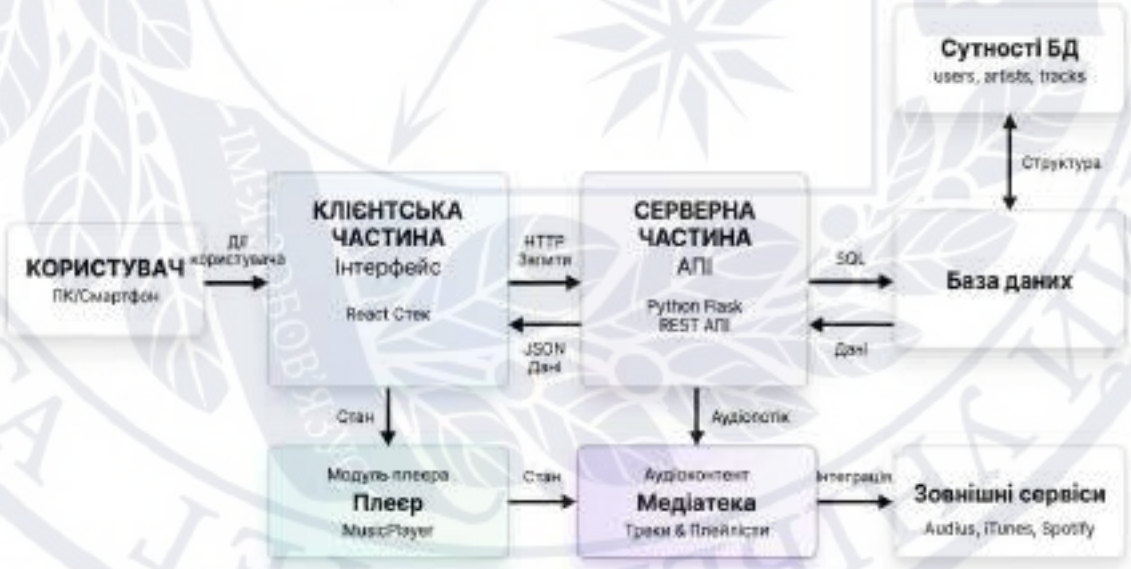

Рисунок 2.1 - Загальна архітектура веб-платформи SonicFlow

Взаємодія користувача з системою відбувається через веб-оболонку. Після завантаження ресурсу користувач потрапляє на стартову сторінку, де йому доступний огляд популярних треків, тематичних добірок, виконавців, альбомів та рекомендованих плейлістів. Інтерфейс збудовано з використанням компонентного принципу: окремі елементи відповідають за відображення карток треків, рядків у списках, обкладинок альбомів, профілів виконавців, бічного навігаційного меню, верхньої панелі керування та нижнього аудіоплеєра. Така компонування значно полегшує повторне застосування цих візуальних блоків на різних ділянках платформи.

Клієнтська частина ініціює комунікацію з сервером за допомогою API-запитів. Наприклад, для отримання переліку треків використовується маршрут **/db-api/tracks**, для виконавців - **/db-api/artists**, для альбомів - **/db-api/albums**, а для списків відтворення - **/db-api/playlists**. Сервер обробляє запит, виконує відповідний SQL-запит до PostgreSQL, формує результат у форматі JSON та надсилає його назад клієнту. Після цього React-компоненти візуалізують отримані відомості у вигляді списків, карток чи окремих сторінок.

Діаграму взаємодії клієнтської логіки з бекендом та базою даних доцільно представити на рисунок 2.2. На цій схемі слід простежити ланцюжок подій: користувач здійснює дію в інтерфейсі, React/Vite відправляє HTTP-запит до Flask API, Flask звертається до PostgreSQL, отримує дані та повертає JSON-відповідь назад на клієнт.

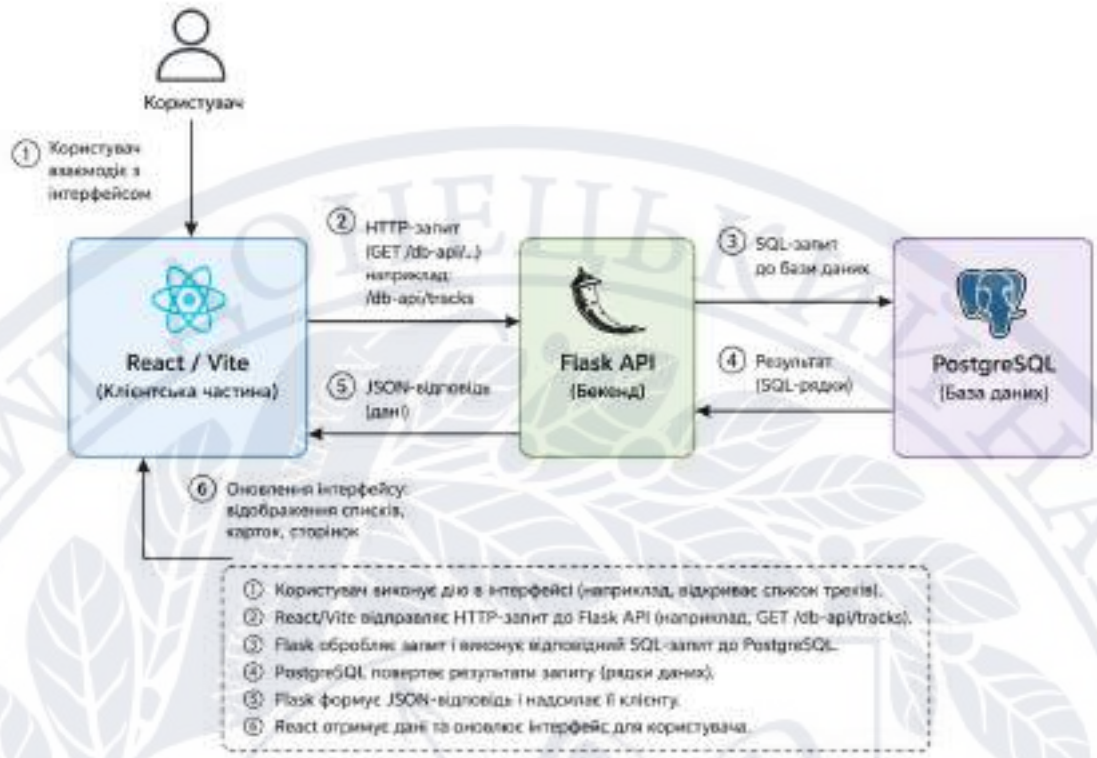

Рисунок 2.2 - Схема взаємодії клієнтської частини, Flask API та бази даних PostgreSQL

Окремим структурним елементом є модуль аудіоплеєра. Він забезпечує керування відтворенням поточної композиції, зміну статусу (пауза/відтворення), перемикання на наступний чи попередній трек, регулювання гучності, перемотування, а також відображення метаданих активного аудіофайлу. У проєкті плеєр закріплено в нижній частині інтерфейсу та зберігає свою активність при навігації між розділами. Це забезпечує високу зручність користування, оскільки користувач може досліджувати інші частини сайту без припинення прослуховування.

Для організації доступу до аудіоконтенту система використовує як дані із власного сховища PostgreSQL, так і зовнішні джерела. У розробці передбачено інтеграцію з Audius API, iTunes API та Spotify. Audius може слугувати для отримання музики незалежних артистів, iTunes API - для пошуку та попереднього прослуховування треків, а Spotify Web Playback SDK - як альтернативний режим відтворення (за умови відповідного налаштування

облікового запису). Spotify Web Playback SDK є клієнтською JS-бібліотекою, призначеною для створення Spotify Connect-пристрою у браузері та управління відтворенням контенту Spotify, проте для повного функціонування необхідна наявність Premium-підписки [34].

Загальна логіка функціонування системи полягає в тому, що користувач взаємодіє з візуальною оболонкою, обирає потрібний розділ, шукає чи ініціює відтворення треку. Якщо інформація зберігається у внутрішній базі, клієнт отримує її через Flask API. Якщо залучається зовнішнє джерело, клієнт контактує безпосередньо з відповідним API чи модулем інтеграції. Після вибору аудіозапису дані про нього передаються до централізованого модуля плеєра, який коригує статус активного треку та запускає стрімінг.

З погляду архітектурного поділу, веб-система SonicFlow структурується на таких ключових рівнях:

Рівень інтерфейсу користувача - відповідає за рендеринг сторінок, навігацію, відображення списків треків, карток альбомів, виконавців, плейлістів та панелі плеєра.

Рівень клієнтської логіки - керує маршрутизацією в застосунку, обробляє взаємодію з користувачем, підтримує стан плеєра, реалізує пошук, фільтрацію та отримання даних з API.

Рівень серверної логіки - реалізований на Flask, він обробляє вхідні HTTP-запити, взаємодіє з PostgreSQL та транслює дані клієнту.

Рівень зберігання даних - представлений PostgreSQL, який зберігає записи про користувачів, митців, жанри, альбоми, аудіотреки, плейлісти та історію прослуховування.

Рівень зовнішніх музичних сервісів - використовується для розширення бази контенту та додавання функціональних можливостей платформи.

Таке розмежування архітектурних блоків дозволяє розглядати SonicFlow як модульну веб-платформу, де кожна частина виконує свою специфічну функцію. Фронтенд займається візуалізацією та взаємодією, бекенд - обробкою даних, база даних - їхнім накопиченням, а модуль плеєра - основною задачею

потокової передачі аудіо. Завдяки цьому можна послідовно розвивати систему: додавати нові сторінки, впроваджувати механізми авторизації, вдосконалювати пошук, інтегрувати рекомендаційні системи, моделі монетизації чи адміністративну частину.

Таким чином, загальна архітектура веб-платформи SonicFlow збудована за клієнт-серверним принципом і об'єднує фронтенд на React/Vite, бекенд на Flask, базу даних PostgreSQL, модуль відтворення аудіо та інтеграції із зовнішніми сервісами. Така конфігурація повністю відповідає поставленим завданням розробки системи для керування та потокового стрімінгу аудіоконтенту, оскільки забезпечує перегляд, пошук, категоризацію, зберігання та відтворення медіаматеріалів у рамках єдиного інформаційного середовища.

2.2. Обґрунтування вибору програмних засобів розробки

Для програмної реалізації веб-платформи SonicFlow було підібрано технологічний стек, що відповідає поставленим завданням: створення інтуїтивно зрозумілого музичного інтерфейсу, показу аудіофайлів, комунікації з бекендом, зберігання даних у PostgreSQL та забезпечення безперервного аудіострімінгу. Використані інструменти сприяють логічному розділенню фронтенду, бекенд-логіки та бази даних, що значно спрощує подальше супроводження та масштабування веб-сервісу.

Фронтенд реалізовано за допомогою бібліотеки React. Цей вибір зумовлений тим, що інтерфейс аудіоплатформи містить багато елементів, що повторюються: картки треків, списки композицій, блоки альбомів, сторінки виконавців, бічна панель навігації та нижній аудіопрогравач. React дає можливість будувати інтерфейс як сукупність незалежних компонентів, застосовувати їх повторно на різних екранах та оперативно оновлювати контент без потреби перезавантажувати сторінку [31]. У даному проєкті цей підхід застосовано для таких компонентів, як `TrackCard`, `TrackRow`, `AlbumCard`, `ArtistCard`, `Sidebar`, `TopBar` та `MusicPlayer`.

Для збирання та запуску фронтенд-частини використовувався Vite. Цей інструмент забезпечує швидке ініціювання середовища розробки, підтримує актуальні JavaScript-модулі та спрощує формування фінальної збірки проєкту [32]. На практиці це означає, що клієнтську частину можна запустити командою `npm run dev`, а для підготовки продакшен-версії - `npm run build`.

Для візуального оформлення застосовано Tailwind CSS у поєднанні з готовими UI-компонентами на базі Radix UI. Tailwind CSS зручний тим, що дозволяє швидко створювати адаптивний дизайн, мінімізуючи необхідність писати великі обсяги чистого CSS. У SonicFlow його застосовано для формування темного оформлення музичної платформи, дизайну контентних карток, кнопок керування плеєром, бічної навігації та сторінок із композиціями. Radix UI слугує базовою архітектурою для елементів інтерфейсу, які вимагають доступності, зокрема модальних вікон, меню, вкладок, перемикачів та повзунків [35].

Бекенд розроблено на базі Python Flask. Flask був обраний завдяки легкості конфігурації, мінімальній кількості boilerplate-коду та зручності мапування API-маршрутів [33]. У цьому проєкті серверний файл `app.py` відповідає за встановлення зв'язку з PostgreSQL, ініціалізацію роботи з базою даних, виконання SQL-запитів та повернення результатів у форматі JSON. Такий вибір є доцільним для дипломної роботи, оскільки наочно демонструє взаємозв'язок між клієнтською частиною, сервером та сховищем даних.

Для керування даними обрано PostgreSQL. Ця система управління базами даних ідеально пасує для архівації структурованої інформації про користувачів, виконавців, жанри, альбоми, треки, списки відтворення, "уподобання" та історію прослуховування. PostgreSQL підтримує реляційну модель, SQL-запити, механізми зовнішніх ключів, індексацію та забезпечення цілісності даних [27]. У проєкті структура бази даних деталізована у файлі `database/schema.sql`, а початкові дані для наповнення таблиць містяться у `database/seed.sql`.

Для інтеграції Flask із PostgreSQL використано бібліотеку `psycopg2`. Вона надає можливість виконувати SQL-команди безпосередньо з коду Python,

отримувати відповіді від бази й передавати їх у логіку сервера. У роботі це задіяно на маршрутах `/db-api/tracks`, `/db-api/artists`, `/db-api/albums` та `/db-api/playlists`. Наприклад, при завантаженні сторінки з треками, фронтенд надсилає запит до API, сервер отримує масив записів із PostgreSQL та повертає його у JSON-форматі.

Відтворення аудіо реалізовано через клієнтський модуль `MusicPlayer` та контекстний менеджер `PlayerContext`. Така архітектура дозволяє зберігати дані про поточний трек у загальному стані застосунку, запобігаючи його зникненню при навігації між сторінками. Це критично важливо для музичного веб-сервісу, оскільки дає змогу користувачеві одночасно слухати музику і переглядати інші розділи сайту: альбоми, виконавців, плейлісти чи нещодавно прослухані треки.

Крім того, у проєкті передбачена функціональність роботи із зовнішніми музичними джерелами. Для цього створено ізольовані API-модулі: `audiusMusic.js`, `itunesMusic.js` та `spotify.js`. Такий підхід відокремлює логіку отримання контенту ззовні від основної логіки клієнта. У майбутньому це відкриває можливості для розширення платформи: інтеграції пошуку треків через сторонні сервіси, реалізації прев'ю композицій або підключення до інших музичних API.

Таким чином, для розробки SonicFlow був обраний раціональний та зручний технологічний стек: React та Vite для клієнтської частини, Tailwind CSS та Radix UI для візуального оформлення, Flask для серверної логіки, PostgreSQL як сховище даних та `psycopg2` для забезпечення зв'язку між Python-сервером та базою даних. Цей набір програмних рішень повністю відповідає цілям веб-платформи для керування та стрімінгу аудіоконтенту, оскільки гарантує створення динамічного інтерфейсу, ефективну роботу з аудіоданими, обробку запитів та структуроване збереження інформації.

2.3. Проєктування клієнтської частини веб-платформи

Фронтенд-складова веб-майданчика SonicFlow покликана забезпечити користувачам дружню взаємодію з аудіоматеріалами. Він відповідає за

презентацію сторінок, маневрування між секціями, показ переліків треків, музичних збірок, виконавців, плейлистів, а також за управління програвачем аудіо. З огляду на те, що ключова взаємодія з платформою відбувається саме через її візуальне оформлення, при проектуванні клієнтської частини було важливо досягти чіткої організації сторінок, оперативного доступу до головних функцій та візуальної відповідності актуальним стрімінговим сервісам.

Реалізація клієнтської частини базується на фреймворках React та Vite. React залучається для побудови інтерфейсу з окремих, незалежних модулів, що дає змогу багаторазово використовувати ті самі елементи на різних візуальних просторах веб-платформи [31]. Vite слугує інструментом для ініціалізації середовища розробки та фінальної збірки проекту, що суттєво спрощує роботу над фронтендом та гарантує швидке оновлення вигляду сторінок під час кодування [32]. Такий підхід є вельми доречним для аудіоресурсу, оскільки його інтерфейс містить значну кількість повторюваних блоків: картки композицій, списки пісень, секції альбомів, профілі артистів, плейлисти та стаціонарний нижній програвач.

Фундаментом клієнтської частини є її модульна архітектура. Кожна складова візуального оформлення винесена в окрему одиницю (компонент), що спрощує супровід коду та дозволяє коригувати окремі частини системи без необхідності повної реструктуризації всього екрану. Приміром, для відображення аудіозаписів можуть бути використані компоненти типу "картка треку" або "рядок композиції", для збірок- відповідні картки, для виконавців - їхні візитівки, а для навігації - бічне меню та верхня панель. Завдяки цьому візуальне подання майданчика має узгоджену структуру, а ідентичні графічні елементи зберігають спільний дизайн усюди.

В процесі розробки дизайну були визначені кілька основних областей. Перша область - це бічна панель навігації, яка забезпечує миттєве переміщення по основних секціях сервісу. Друга область - центральна частина екрану, де відображається первинний контент: добірки пісень, альбоми, артисти, плейлисти або результати пошуку. Третя область - нижній програвач аудіо, який

залишається активним під час зміни сторінок. Цей розподіл є зручним для музичних платформ, оскільки дозволяє користувачеві паралельно переглядати матеріали та контролювати відтворення.

Важливу функцію у клієнтській частині виконує механізм маршрутизації між візуальними екранами. Веб-майданчик володіє низкою логічних розділів: домашня сторінка, пошуковий розділ, бібліотека користувача, сторінки окремих альбомів, артистів, плейлистів та нещодавно прослуханого матеріалу. Переходи між цими розділами відбуваються без повного перезавантаження веб-сторінки, що типово для так званих односторінкових застосунків (SPA). Це значно підвищує швидкодію інтерфейсу та робить взаємодію користувача з платформою більш динамічною.

Для презентації музичного контенту використовуються дві формати - картки та списки. Картки ідеально підходять для представлення збірок, виконавців або тематичних добірок, бо дають можливість об'єднати візуальний образ, назву та стислу інформацію. Списки краще підходять для відображення окремих композицій, де критично важливо показати назву треку, його виконавця, альбом, часові рамки чи інші характеристики. Такий підхід дозволяє користувачеві оперативно оглянути аудіоматеріали та ініціювати відтворення потрібної доріжки.

Невід'ємним елементом клієнтської частини є аудіопрогравач. Він відповідає за старт та паузу відтворення, зміну композицій, регулювання гучності, швидке переміщення по треку та інформування про поточну композицію. У SonicFlow плеєр закріплений у нижній частині екрану, що дозволяє користувачеві керувати музикою незалежно від того, на якому саме розділі він перебуває. Така реалізація значно підвищує зручність користування сервісом, адже прослуховування не припиняється під час переходу між секціями.

Для збереження стану того, що саме прослуховується, задіяна спеціальна клієнтська логічна схема. Вона забезпечує передачу інформації про обраний аудіозапис до модуля плеєра та оновлення візуального оформлення згідно з діями користувача. Наприклад, після вибору композиції системі в плеєр

надсилається її назва, виконавець, обкладинка та посилання на сам аудіофайл чи джерело. Після цього плеєр актуалізує свій стан і починає трансляцію.

Крім того, клієнтська частина здійснює обмін даними з серверною через API-запити. Для отримання інформації про треки, збірки, артистів та плейлісти фронтенд відправляє запити типу HTTP до API, створеного на Flask. Сервер повертає відповідь у форматі JSON, після чого React-компоненти візуалізують отримані відомості на сторінці. Наприклад, список доріжок може формуватися на основі відповіді від ендпоінта /db-api/tracks, артисти - з /db-api/artists, альбоми - з /db-api/albums, а плейлісти - з /db-api/playlists. Такий підхід забезпечує логічне розділення інтерфейсу від механізмів зберігання вмісту, роблячи архітектуру майданчика більш прозорою.

Для стилістичного оформлення інтерфейсу застосовано Tailwind CSS, який дозволяє стрімко конструювати гнучкі (адаптивні) стилі, мінімізуючи необхідність у великій кількості окремих файлів CSS [35]. Завдяки цьому сторінки сервісу мають єдиний візуальний почерк, а елементи інтерфейсу коректно відображаються на екранах будь-якого розміру. Для деяких складних інтерактивних елементів можуть використовуватись компоненти Radix UI, які спрощують реалізацію модальних вікон, меню, вкладок, повзунків (слайдерів) та інших подібних візуальних інтеракцій [36].

Під час проектування фронтенду також було враховано вимоги до адаптивності. Оскільки користувачі можуть отримувати доступ до веб-платформи за допомогою стаціонарних ПК, ноутбуків або мобільних ґаджетів, інтерфейс мусить залишатися зручним на різних типах дисплеїв. Для цього сторінки побудовані на принципі гнучкої розкладки, а ключові складові - меню, блоки контенту, списки композицій та програвач - мають бути доступними й зрозумілими незалежно від пристрою.

Таким чином, клієнтська частина веб-платформи SonicFlow сконструйована як модульний односторінковий застосунок, що використовує React, Vite, JavaScript, Tailwind CSS та Radix UI. Вона відповідає за візуалізацію музичного контенту, навігацію між секціями, роботу зі списками треків,

збірками, виконавцями, плейлистами та функціонал програвача аудіо. Така архітектура сприяє формуванню ергономічного та сучасного інтерфейсу для управління та потокової трансляції аудіоматеріалів.

2.4. Проєктування серверної частини та API

Бекенд складова веб-платформи "SonicFlow" має завданням обробляти запити, що надходять від фронтенд-частини, взаємодіяти з реляційною базою даних PostgreSQL та надавати структуровані дані у форматі JSON. В описі загальної системи, бекенд функціонує як медіаторний рівень між інтерфейсом (створеним на React/Vite) та сховищем даних. Таким чином, клієнтська сторона не взаємодіє з базою даних напряму, а отримує необхідні дані через заздалегідь визначені API-ендпоінти.

Серверна логіка реалізована за допомогою Python Flask. Flask є фреймворком для вебу, що дає можливість конфігурувати серверні шляхи (маршрути), обробляти HTTP-запити та формувати відповіді для клієнта [28]. Для проєкту "SonicFlow" такий вибір є вдалим, оскільки система оперує чітко окресленим набором запитів для видобування інформації про треки, виконавців, альбоми та списки відтворення. Ба більше, застосування Flask дозволяє зберегти архітектуру бекенду лаконічною, що є практичною перевагою при розробці бакалаврського проєкту.

У рамках цього проєкту бекенд охоплює кілька ключових функцій. По-перше, він відповідає за встановлення з’єднання із БД PostgreSQL. По-друге, виконує SQL-запити для отримання релевантної інформації. По-третє, він трансформує отримані дані у формат, зручний для споживання фронтендом. По-четверте, він відправляє ці дані назад як JSON-відповідь, яку вже інтерпретують React-компоненти.

Загальну компоновку серверної частини доцільно візуалізувати на Рисунку 2.3. На ньому можна відобразити ключові складові бекенду: Flask-застосунок,

визначені API-маршрути, модуль для конекту до PostgreSQL, самі SQL-вибірки та формат JSON-відповідей.

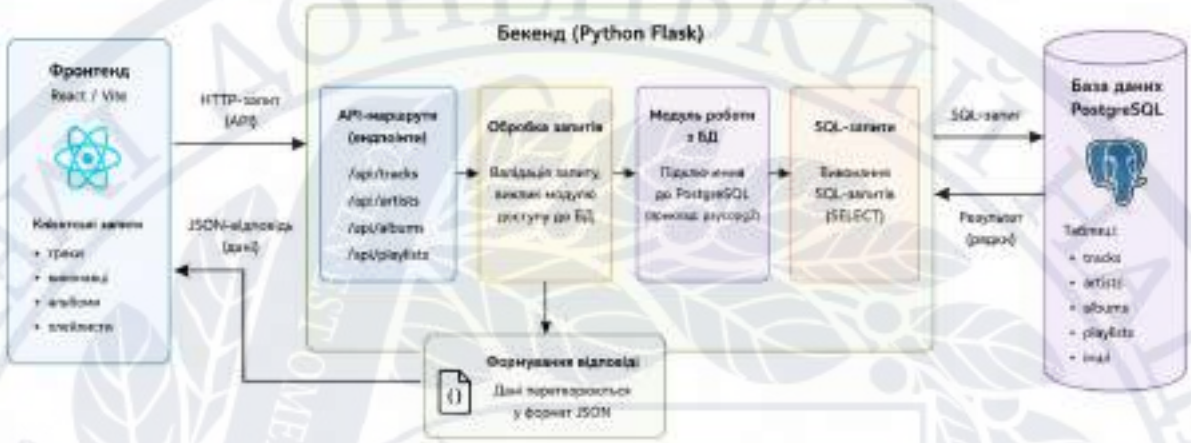

Рисунок 2.3 - Схема взаємодії серверної складової веб-платформи SonicFlow

Для коректного обміну даними між фронтендом і бекендом застосовано розширення Flask-CORS. Це є критично важливим, бо під час розробки локально клієнтська й серверна частини можуть функціонувати на різних портах (наприклад, фронтенд через Vite, а сервер окремо через Flask). Flask-CORS усуває конфлікти, пов'язані з політикою розрішення доменів (CORS) [29], забезпечуючи безперешкодну комунікацію.

Для інтеграції Python Flask із PostgreSQL задіяна бібліотека Psycopg. Вона забезпечує необхідний функціонал для підключення до БД, виконання SQL-команд та отримання результатів на сервері [30]. У практичному сенсі, це означає, що після отримання HTTP-запиту від клієнта, Flask звертається до PostgreSQL, витягує необхідні записи й передає їх відповідному API-ендпоінту.

PostgreSQL обрано як реляційну СУБД для зберігання структурованих відомостей про увесь аудіоконтент. Сюди входять метадані стосовно треків, виконавців, жанрів, альбомів, плейлістів, а також списків "уподобань" та історії прослуховувань. Використання PostgreSQL виправдане, оскільки ця СУБД

надійно підтримує виконання SQL-запитів, забезпечення зв'язків між таблицями, цілісності даних та роботи з реляційними даними [27].

Основні API-шляхи, які обслуговує веб-платформа SonicFlow, перелічені в Таблиці 2.1.

Таблиця 2.1 - Ключові API-ендпоінти серверної частини SonicFlow

№	API-маршрут	Призначення
1	/db-api/tracks	Отримання списку аудіозаписів
2	/db-api/artists	Отримання списку виконавців
3	/db-api/albums	Отримання списку альбомів
4	/db-api/playlists	Отримання списку плейлістів
5	/db-api/recent-plays	Отримання нещодавно прослуханих треків
6	/db-api/liked-tracks	Отримання вподобаних аудіозаписів

Кожен API-ендпоінт виконує вузькоспеціалізовану функцію. До прикладу, шлях `/db-api/tracks` слугує для отримання усіх треків, що мають бути показані клієнту. Після звернення до цього шляху сервер виконує SQL-запит до таблиці треків, за потреби об’єднує відомості з таблиць виконавців, альбомів чи жанрів, і повертає результат у форматі JSON. Фронтенд, своєю чергою, приймає ці дані й візуалізує їх у вигляді сіток або списків.

Маршрут `/db-api/artists` призначений для отримання інформації про виконавців. Ці дані можуть бути представлені на окремій сторінці артиста або інтегровані у картки треків. `/db-api/albums` відповідає за отримання альбомів, а `/db-api/playlists` - за відображення списків відтворення. Завдяки такому розмежуванню API має логічно структурований вигляд, де кожен шлях асоційований з певною сутністю системи.

Ілюстрацію взаємодії фронтенду з API логічно продемонструвати на Рисунку 2.4. Ця схема може показати процес, коли користувач відкриває

сторінку зі списком треків, React/Vite формує запит до Flask API, сервер отримує необхідні дані з PostgreSQL та відправляє JSON-відповідь для рендерингу в інтерфейсі.

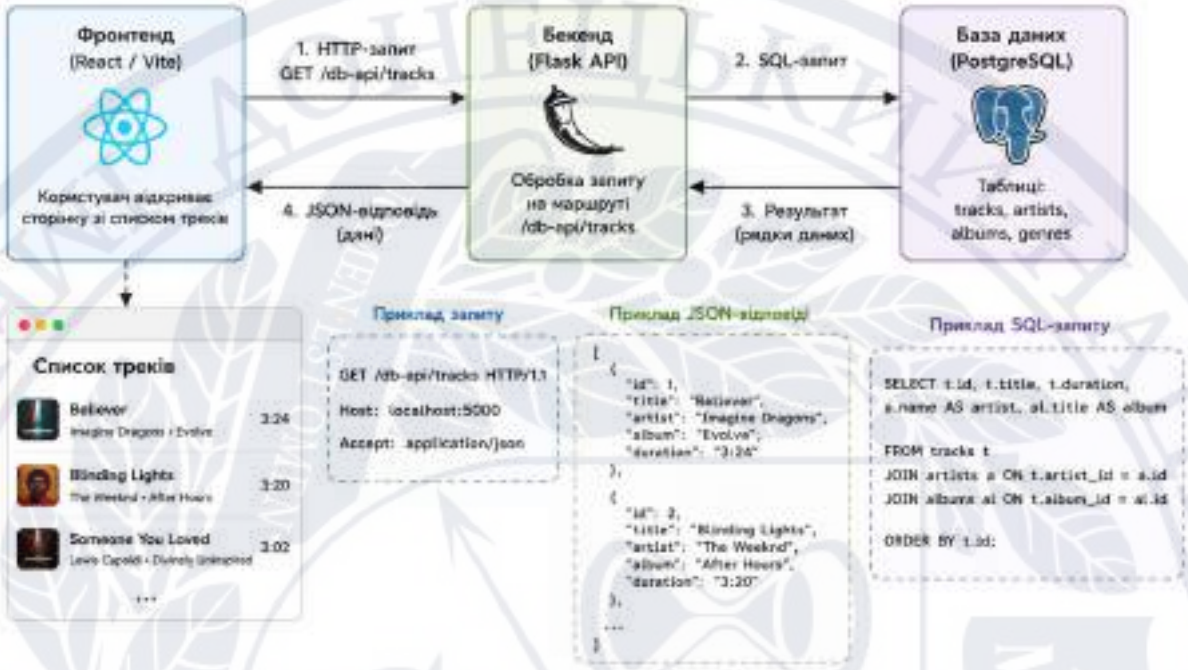

Рисунок 2.4 - Процес обробки запиту на отримання списку треків

Покрокова логіка роботи цього API виглядатиме наступним чином: користувач ініціює дію в інтерфейсі; фронтенд надсилає HTTP-запит на відповідний шлях; Flask приймає запит та активує потрібну сервер-функцію; сервер звертається до PostgreSQL; база даних повертає набір даних; Flask пакує результат у JSON-об'єкт; React-компоненти візуалізують отриману інформацію на екрані. Цей підхід забезпечує чітке розділення відповідальності між логікою фронтенду, обробкою на бекенді та рівнем персистентного зберігання даних.

Особлива увага при проєктуванні бекенду була приділена структурі передачі даних клієнту. Для аудіоплатформи життєво важливо, щоб кожен трек містив не лише назву, а й розширені відомості: ім'я виконавця, назву альбому, жанр, посилання на обкладинку, тривалість та джерело для відтворення. Саме ці дані потім використовуються у клієнтській частині для побудови елементів інтерфейсу - карток, списків треків та самого плеєра.

Серверна частина також повинна гарантувати стійкість функціонування системи. У разі невдалого звернення до БД або відсутності запитаних даних, API мусить повертати інформативну відповідь, аби клієнтська частина могла адекватно відреагувати на помилку. Це ключове для запобігання збоям у відображенні та підтримки працездатності платформи навіть за наявності технічних ускладнень.

Щодо подальших ітерацій розвитку, бекенд має потенціал для розширення новими API-ендпоінтами. Наприклад, у майбутньому можуть бути додані шляхи для реєстрації та аутентифікації користувачів, функціоналу додавання треків у "улюблені", створення користувацьких плейлистів, збереження історії прослуховування або розгортання адмін-панелі. Завдяки вибору Flask, таке нарощення функціонала можна здійснювати поетапно, додаючи нові маршрути без необхідності повної реструктуризації архітектури.

Підсумовуючи, серверний рівень веб-платформи SonicFlow розроблено як незалежний бекенд-компонент, що базується на Python Flask, додатково інтегруючи Flask-CORS, PostgreSQL та Psycopg. Він відповідає за прийом HTTP-запитів від фронтенду, виконання SQL-запитів до бази даних, отримання даних про аудіоконтент та їх подачу у форматі JSON. Така архітектура є практичною основою для створення системи управління та стрімінгу аудіоконтенту, забезпечуючи чітку взаємодію між інтерфейсом, серверною бізнес-логікою та рівнем зберігання даних.

2.5. Проєктування бази даних веб-платформи

Сховище даних є одним із наріжних каменів веб-сервісу SonicFlow, адже саме воно відповідає за фіксацію впорядкованої інформації про музичні композиції, митців, збірки, стилі, списки відтворення, а також дії, вчинені користувачами. Для архітектури цього сховища обрано PostgreSQL, оскільки ця СУБД підтримує роботу з реляційною моделлю, використання SQL-запитів,

первинних та зовнішніх ключів, накладання обмежень на цілісність даних та індексацію [27].

У рамках розробки опис структури бази даних зафіксовано у файлі database/schema.sql. Цей документ окреслює усі таблиці, взаємозв'язки між ними, правила валідації для окремих полів, а також індексації, що застосовуються для оперативного доступу до інформації. Такий підхід дозволяє зберігати музичний контент не просто як пачку окремих файлів, а як взаємопов'язану інформаційну модель, де кожен трек може мати прив'язку до виконавця, стилю, альбому, обкладинки, джерела відтворення, лічильника програвань та додаткових метаданих.

Загальний устрій сховища даних для веб-застосунку SonicFlow доцільно візуалізувати на рисунку 2.5. На цій діаграмі необхідно проілюструвати ключові таблиці бази та їхні залежності: users, artists, genres, albums, tracks, track_artists, playlists, playlist_tracks, liked_tracks, recent_plays.

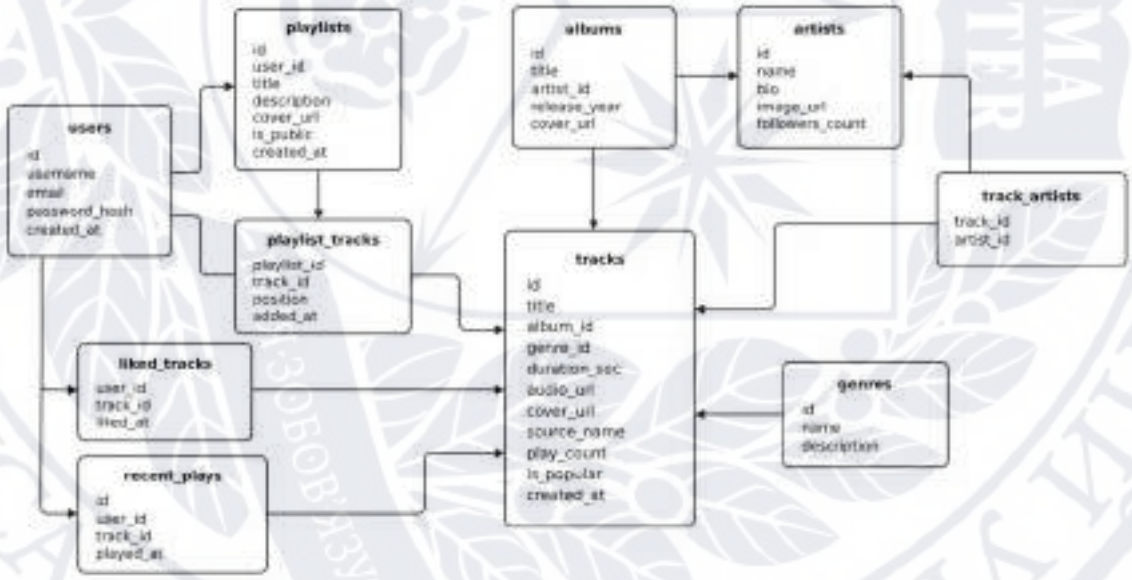

Рисунок 2.5 - Схема бази даних веб-платформи SonicFlow

Центральною для зберігання музичного контенту є таблиця tracks. Вона акумулює назву композиції, посилання на альбом та стиль, час звучання у секундах, шлях до аудіофайлу чи зовнішньої ланки, візуальне оформлення

(обкладинку), назву джерела, загальну кількість прослуховувань, маркер популярності, а також дату реєстрації запису. Саме дані з цієї таблиці використовуються клієнтською частиною для формування списків музики, карток треків та функціонування програвача.

Для обліку інформації про митців задіяна таблиця artists. Вона містить назву виконавця, його опис, прикріплене зображення та кількість підписників. Зважаючи на можливість однієї композиції мати декілька виконавців, між таблицями tracks та artists введено допоміжну таблицю track_artists. Вона імплементує зв'язок типу “багато до багатьох”, що уможливлює прив'язку однієї музичної роботи до кількох митців.

Таблиця albums слугує для збереження даних про музичні збірки. Вона містить найменування альбому, посилання на головного виконавця, рік випуску та його обкладинку. Оскільки один альбом може включати багато треків, таблиця tracks має зовнішній ідентифікатор album_id, який вказує на таблицю albums. Для категоріїзації музичних творів за напрямами використовується таблиця genres, а зв'язок із нею встановлюється через поле genre_id у таблиці tracks.

Окрема група таблиць забезпечує роботу з плейлистами. Таблиця playlists фіксує назву списку відтворення, його опис, обкладинку, статус видимості та власника-користувача. Для взаємозв'язку плейлистів із треками використовується таблиця playlist_tracks. Вона містить ідентифікатор плейлиста, ідентифікатор треку, позицію композиції у списку та дату її включення. Такий механізм дозволяє мати значну кількість треків в одному плейлисті, і навпаки - один трек може фігурувати у різних списках.

Для фіксації дій, вчинених користувачами, виділено таблиці liked_tracks і recent_plays. Таблиця liked_tracks використовується для зберігання "вподобаних" композицій, тоді як recent_plays слугує для запису історії прослуховувань. Це закладає фундамент для індивідуалізації сервісу, оскільки в перспективі на базі цих даних можна буде генерувати персоналізовані рекомендації, добірки або список нещодавно відтворених композицій.

Головні таблиці бази даних SonicFlow представлено у Таблиці 2.2.

Таблиця 2.2 - Ключові таблиці сховища даних SonicFlow

Таблиця	Призначення
Users	Збереження даних користувачів
Artists	Збереження інформації про виконавців
Genres	Збереження жанрів аудіоконтенту
Albums	Збереження альбомів
Tracks	Збереження аудіозаписів та їхніх метаданих
Track_artists	Зв’язок між треками та виконавцями
Playlists	Збереження плейлістів
Playlists_tracks	Зв’язок між плейлістами та треками
Liked_tracks	Збереження вподобаних треків
Recent_plays	Збереження історії прослуховування

У структурі бази даних також закладено механізми забезпечення цілісності даних. Наприклад, для деяких числових полів встановлені валідаційні перевірки, які унеможливлюють збереження некоректних значень: тривалість композиції мусить бути строго додатною, а кількість прослуховувань чи підписників не може мати від'ємного значення. Зовнішні ключі гарантують коректну налагодженість зв'язків між таблицями, а застосування каскадного видалення чи встановлення значення NULL допомагає контролювати реакцію системи при стиранні пов'язаних сутностей.

Для прискорення роботи системи передбачено індекси. Зокрема, індекс за лічильником прослуховувань дає можливість швидше отримувати перелік популярних треків, індекс за стилем полегшує фільтрацію музичних творів, а індекс у таблиці recent_plays забезпечує швидкий доступ до історії прослуховування конкретного користувача. Це критично важливо для музичної платформи, адже користувач очікує миттєвого завантаження списків композицій, альбомів та плейлистів.

Таким чином, сховище даних SonicFlow збудовано як реляційна система, що охоплює всі ключові сутності аудіоплатформи: користувачів, митців, стилі, збірки, треки, плейлисти, обрані композиції та історію прослуховування. Ця модель гарантує структуроване зберігання даних, підтримує логічні зв'язки між компонентами системи та слугує міцною основою для майбутнього розширення функціонального наповнення веб-сервісу.

2.6. Розробка модуля керування аудіоконтентом

Блок керування звуковим контентом являє собою один із ключових компонентів інтернет-майданчика SonicFlow. Його головна функція - приймати, структурувати, презентувати та організовувати музичні дані у візуальній частині для користувача. В рамках цього проєкту даний блок відповідає за обробку інформації щодо композицій, митців, збірок (альбомів) та переліків відтворення (плейлистів), а також забезпечує комунікацію даних між клієнтським застосунком, сервером та сховищем PostgreSQL.

На практиці управління музичними даними реалізовано шляхом інтеграції фронтенд-елементів на React, взаємодії з API-запитами до серверу на Flask та виконання SQL-запитів до PostgreSQL. Користувач оперує інтерфейсом: заходить на головну сторінку, оглядає треки, переходить до відповідних альбомів, митців чи плейлистів. Після цього клієнтська сторона відправляє запит до необхідного API-адресу, сервер видобуває дані з бази, формує відповідь у форматі JSON і реверсує її до React-компонентів.

Загальну схему роботи блоку управління звуковим контентом доречно представити на рисунку 2.6. На цій діаграмі слід відобразити послідовність взаємодії між кінцевим користувачем, клієнтською частиною, програмним інтерфейсом (API), базою даних та елементами візуалізації аудіоінформації.

Рисунок 2.6 - Схема роботи блоку управління звуковим контентом SonicFlow

Для роботи з музичними відомостями на стороні клієнта залучено окремий API-клас databaseMusic.js. Тут визначено набір функцій для отримання інформації із серверного середовища: getDatabaseTracks, getDatabaseArtists, getDatabaseAlbums та getDatabasePlaylists. Такий методологічний підхід усуває дублювання запитів у різних компонентах, дозволяючи інкапсулювати логіку отримання даних в окремому файлі. Це значно полегшує супровід програмного коду та підвищує прозорість архітектури фронтенду.

Ключовим об'єктом даного модуля є саме аудіотрек. Для вибірки переліку композицій використовується маршрут /db-api/tracks. Він підтримує опції для пошуку, фільтрації за музичним напрямом (жанром), сортування та встановлення ліміту на кількість виведених записів. Наприклад, користувач має можливість переглянути найпопулярніші треки або здійснити пошук за назвою твору, ім’ям виконавця або збірки. На сервері для цього формується SQL-запит, який об'єднує інформацію з таблиць tracks, artists, albums, genres та track_artists. У підсумку, клієнтський застосунок отримує повний набір метаданих про трек: назву, автора, альбом, жанрову належність, тривалість, зображення обкладинки, посилання на джерело та статистику прослуховувань.

Окрім треків, модуль забезпечує функціонал для роботи з митцями. Дані про них витягуються через шлях /db-api/artists. Сервер повертає унікальний код митця, його ім'я, стислу біографію, візуальний образ, кількість фоловерів та сумарну кількість пов'язаних із ним композицій. Ця інформація застосовується для конструювання сторінок виконавців та відображення їхніх міні-карток у клієнтській частині.

Для збірок (альбомів) задіяно маршрут /db-api/albums. Він надає назву збірки, рік її виходу, художнє оформлення (обкладинку), ім’я основного виконавця та кількість треків, що входять до неї. Така структура сприяє зручному представленню альбомів у вигляді графічних блоків з можливістю подальшого переходу до детального огляду відповідного звукового матеріалу.

Окремим аспектом модуля є функціонування з переліками відтворення. Для отримання списку плейлистів використовується шлях /db-api/playlists. Сервер віддає назву плейлиста, його опис, обкладинку, ім'я власника та загальну кількість композицій. У нинішній імплементації плейлисти слугують для групування аудіоконтенту та презентації тематичних добірок. У майбутньому цей функціонал може бути розширений можливостями створення користувачем власних плейлистів, а також функціями додавання та видалення композицій.

Головні операції, що виконуються блоком управління звуковим контентом, зведені у Таблиці 2.3.

Таблиця 2.3 — Основні операції блоку управління звуковим контентом

№	Функція	Опис
1	Отримання треків	Завантаження списку аудіозаписів із бази даних
2	Пошук аудіоконтенту	Пошук треків за назвою, виконавцем або альбомом
3	Фільтрація за жанром	Отримання аудіозаписів відповідно до вибраного жанру

4	Перегляд виконавців	Відображення списку виконавців і пов’язаних із ними даних
5	Перегляд альбомів	Відображення альбомів, обкладинок і кількості треків
6	Перегляд плейлістів	Відображення тематичних добірок аудіоконтенту
7	Сортування треків	Виведення популярних або нових аудіозаписів
8	Передача даних до плеєра	Передавання інформації про вибраний трек у модуль відтворення

Важливою особливістю модуля є те, що він оперує не тільки самим аудіофайлом, а й усім супутнім метаданим. Для аудіоплатформи недостатньо лише зберігати посилання на файл. Користувач повинен мати змогу бачити назву треку, його автора, альбом, жанр, оформлення, тривалість та походження. Саме з цієї причини на серверному рівні інформація з кількох таблиць консолідується в єдину структуру, яка потім передається клієнтській частині.

З точки зору інтерфейсу, музичний контент візуалізується за допомогою спеціалізованих React-елементів. Композиції можуть бути подані у вигляді мініатюрних карток або ж у вигляді спискових рядків, альбоми та митці - як окремі картки, а плейлисти - як тематичні колекції. Такий підхід забезпечує єдину стилістику подання довідкової інформації та дозволяє багаторазово використовувати ці компоненти на різних сторінках майданчика.

Блок управління звуковим контентом також перебуває у взаємозв'язку з модулем потокового відтворення. Після того як користувач обирає композицію, дані про неї пересилаються до програвача. До таких відомостей належать назва твору, його виконавець, обкладинка, хронометраж та URL-адреса аудіофайлу. Це дозволяє програвачу коректно відобразити інформацію про активний трек та ініціювати його відтворення.

Підсумовуючи, блок управління звуковим контентом у SonicFlow відповідає за збір, організацію, пошук та презентацію музичних даних. Він інтегрує клієнтські React-компоненти, запити до Flask-сервера та реляційну базу даних PostgreSQL. Така архітектурна побудова дає змогу організувати музичні дані як повноцінну інформаційну систему, де треки, митці, збірки та переліки відтворення взаємопов'язані та доступні користувачеві через зручний вебінтерфейс.

2.7. Розробка модуля потокового відтворення аудіо

Аудіострімінговий модуль є ключовою складовою веб-платформи SonicFlow, адже він надає основну функціональність системи - прослуховування аудіоматеріалів через вебінтерфейс. На відміну від звичайного завантаження файлу, стрімінг дає змогу користувачеві розпочати прослуховування запису практично миттєво після вибору, не вимагаючи попереднього збереження файлу на свій девайс. Такий метод є надзвичайно корисним для музичних веб-ресурсів, оскільки він скорочує час очікування контенту та підвищує загальну залученість користувача у роботу з сервісом.

На клієнтському рівні SonicFlow, модуль відтворення втілено у вигляді окремого елемента - аудіоплеєра. Він відповідає за відображення треку, який відтворюється зараз, ініціювання та зупинку звучання, перехід між композиціями, регулювання гучності та показ поточного статусу програвання. У дизайні проєкту плеєр розміщено в нижній частині інтерфейсу, що дозволяє користувачеві продовжувати аудіосупровід під час переміщення між різними сторінками платформи. Цей підхід є поширеним у сучасних музичних сервісах, оскільки він забезпечує можливість одночасного перегляду сайту та безперервного слухання музики.

Для фіксації даних про поточний трек використовується клієнтський стан застосунку. У SonicFlow цю логіку реалізовано через PlayerContext, який допомагає передавати відомості про активний аудіозапис між різноманітними

компонентами. Приміром, коли користувач обирає трек зі списку, до плеєра надсилаються його назва, виконавець, зображення обкладинки, тривалість та URL-адреса аудіофайлу. Після цього компонент MusicPlayer оновлює дані про обраний трек і починає його відтворення.

Алгоритм роботи модуля стрімінгового відтворення аудіо логічно зображено на діаграмі 2.7. На цій схемі можна простежити послідовність дій: вибір треку користувачем, надсилання даних до PlayerContext, отримання активного треку компонентом MusicPlayer, а відтак – виконання команд запуску, паузи, перемотування чи зміни гучності.

Рисунок 2.7 - Схема роботи модуля потокового відтворення аудіо

Первинним джерелом інформації для плеєра слугує об'єкт треку, отриманий або з бази даних, або від зовнішнього музичного сервісу. Цей об'єкт містить метадані, необхідні для візуалізації в інтерфейсі, а також поле audio_url, яке безпосередньо використовується для запуску аудіофайлу. Завдяки цьому плеєр не функціонує ізольовано, а є інтегрованою частиною загальної системи управління аудіоконтентом. Він отримує необхідні дані від модуля, що відповідає за відображення треків, і використовує їх для аудіозапуску.

Для відтворення аудіо у вебі може бути залучено стандартну технологію HTML-аудіо. Елемент HTML <audio> підтримує програвання звукових файлів безпосередньо у браузері та здатний працювати з різними типами аудіоджерел [12]. У контексті SonicFlow використання цього методу є доцільним, оскільки воно не вимагає від користувача встановлення жодного додаткового програмного забезпечення. Відтворення відбувається нативно у браузері, що повністю відповідає архітектурі веб-платформи.

Окремо у проєкті передбачено функціонал взаємодії із зовнішніми музичними джерелами. Для цього задіяні модулі інтеграції з Audius API, iTunes API та Spotify. У випадку Spotify може застосовуватися Spotify Web Playback SDK, що дозволяє дистанційно контролювати програвання їхнього контенту у браузері; однак повне використання цього інструментарію залежить від певних вимог, пов'язаних з обліковим записом користувача [34]. З огляду на це, у межах цієї бакалаврської роботи основний фокус зроблено на розробці власного модуля відтворення та обробці аудіоданих, отриманих через API самої платформи.

Ключові можливості модуля потокового відтворення аудіо перелічено у таблиці 2.4.

Таблиця 2.4 - Основні функції модуля стрімінгу аудіо

№	Функція	Опис
1	Запуск треку	Початок відтворення вибраного аудіозапису
2	Пауза відтворення	Тимчасова зупинка активного треку
3	Перемикання треків	Перехід до наступної або попередньої композиції
4	Регулювання гучності	Зміна рівня звуку під час відтворення

5	Відображення активного треку	Показ назви, виконавця та обкладинки поточної композиції
6	Передача стану плеєра	Збереження інформації про активний трек у клієнтському стані
7	Робота з аудіоджерелом	Використання посилання audio_url для запуску аудіофайлу

Важливою перевагою такої архітектури є те, що плеєр залишається активним незалежно від того, на якій сторінці перебуває користувач. Наприклад, можна запустити трек на головній сторінці, а потім перейти до каталогу альбомів або переглянути профіль виконавця, не зупиняючи при цьому музику. Це підвищує зручність користування платформою та наближає її інтерфейс до стандартів сучасних музичних сервісів.

Крім того, модуль стрімінгу закладає основу для подальшого розширення системи. У перспективі до нього можна інтегрувати відображення прогресу треку, функції повтору композиції, випадкового відтворення (шафлу), управління чергою прослуховування, налаштування еквалайзера чи зберігання історії прослуханих записів. Це дасть змогу розвинути SonicFlow із суто демонстраційної аудіоплатформи до повноцінного комерційного веб-сервісу.

Таким чином, модуль потокового відтворення аудіо в SonicFlow забезпечує запуск та контроль аудіозаписів безпосередньо у браузері. Його функціональність ґрунтується на взаємодії компонента MusicPlayer, механізму станів PlayerContext, даних про трек та його джерела audio_url. Така реалізація надає користувачу комфортний спосіб слухання контенту, дозволяючи вільно переходити між розділами платформи та керувати поточним відтворенням у межах одного вебінтерфейсу.

2.8. Розробка користувацького інтерфейсу веб-платформи

Користувацький інтерфейс веб-платформи SonicFlow розроблено з урахуванням основного призначення системи — зручного перегляду, пошуку,

впорядкування та потокового відтворення аудіоконтенту. Оскільки саме через інтерфейс користувач взаємодіє з треками, альбомами, виконавцями, плейлістами та аудіоплеєром, під час його розробки основну увагу було приділено простоті навігації, зрозумілому розміщенню елементів і швидкому доступу до основних функцій.

Інтерфейс SonicFlow реалізовано на основі компонентного підходу React. Це дозволяє розділити сторінки та окремі елементи інтерфейсу на самостійні компоненти, наприклад бокове меню, верхню панель, картки треків, картки альбомів, картки виконавців, списки композицій і нижній аудіоплеєр [31]. Такий підхід є практичним, оскільки однакові елементи можна повторно використовувати на різних сторінках платформи без дублювання коду.

Візуальне оформлення платформи побудовано в темному стилі, характерному для сучасних музичних сервісів. Темний фон зменшує візуальне навантаження під час тривалого перегляду контенту, а контрастні елементи дозволяють швидко виділяти активні кнопки, картки треків і панель програвача. Для стилізації використано Tailwind CSS, що дає змогу швидко налаштовувати відступи, сітки, кольори, розміри елементів і адаптивну поведінку сторінок [35].

Основна структура інтерфейсу складається з кількох функціональних зон. Ліва частина використовується для навігації між основними розділами платформи. У центральній області відображається основний контент: треки, альбоми, виконавці, плейлісти або результати пошуку. Нижня частина сторінки містить аудіоплеєр, який залишається доступним незалежно від того, у якому розділі перебуває користувач. Така структура дозволяє одночасно переглядати контент і керувати відтворенням аудіо.

Для відображення аудіоконтенту використовуються картки та списки. Картки застосовуються для альбомів, виконавців і добірок, оскільки вони дають змогу компактно поєднати зображення, назву й коротку інформацію. Списки доцільні для треків, де важливо показати назву композиції, виконавця, альбом або тривалість. Такий підхід робить інтерфейс зручним для швидкого перегляду великої кількості аудіоматеріалів.

Окрему роль в інтерфейсі відіграє сторінка пошуку. Вона дозволяє користувачу швидко знаходити потрібні аудіозаписи, альбоми або виконавців. Пошук є важливою функцією для аудіоплатформи, оскільки зі збільшенням кількості контенту користувачу стає складніше знаходити потрібні матеріали вручну. У SonicFlow пошук пов'язаний із клієнтською логікою та API-запитами, які повертають релевантні результати для подальшого відображення в інтерфейсі.

Важливою частиною інтерфейсу є нижній аудіоплеєр. Він містить інформацію про активний трек, кнопки запуску й паузи, елементи перемикання композицій, регулювання гучності та відображення стану відтворення. Розміщення плеєра в нижній частині сторінки є зручним, оскільки користувач може переходити між розділами сайту, не втрачаючи доступу до керування аудіо.

У процесі розробки інтерфейсу також враховано адаптивність. Користувач може відкривати платформу з різних пристроїв, тому елементи інтерфейсу повинні коректно відображатися на екранах різного розміру. Для цього використовуються гнучкі сітки, адаптивні класи Tailwind CSS і компонентна структура сторінок. Особливо важливо, щоб у мобільній версії залишалися доступними основні функції: перегляд треків, пошук, навігація та керування відтворенням.

Для окремих інтерактивних елементів можуть використовуватися компоненти Radix UI. Вони спрощують створення діалогових вікон, меню, вкладок, слайдерів та інших елементів, які мають бути зручними й доступними для користувача [36]. Це особливо корисно для аудіоплатформи, де важливо забезпечити швидку взаємодію з кнопками, меню та елементами керування плеєром.

Основні елементи користувацького інтерфейсу SonicFlow наведено в таблиці 2.5.

Таблиця 2.5 - Основні елементи користувацького інтерфейсу SonicFlow

Елемент інтерфейсу	Призначення
Бокове меню	Перехід між основними розділами платформи
Верхня панель	Відображення службових елементів і швидких дій
Картка треку	Відображення короткої інформації про аудіозапис
Картка альбому	Відображення альбому, обкладинки та пов’язаних даних
Картка виконавця	Відображення інформації про автора або виконавця
Список треків	Перегляд аудіозаписів у структурованому вигляді
Сторінка пошуку	Пошук аудіоконтенту за назвою або іншими параметрами
Нижній аудіоплеєр	Керування відтворенням активного треку

З практичного погляду розроблений інтерфейс забезпечує основні сценарії роботи користувача з платформою. Користувач може перейти до потрібного розділу, переглянути доступний аудіоконтент, знайти потрібний трек, відкрити альбом або виконавця, запустити відтворення та продовжити навігацію сайтом. Завдяки нижньому плеєру керування музикою залишається доступним під час роботи з іншими сторінками.

Отже, користувацький інтерфейс SonicFlow розроблено як адаптивний, компонентний і зручний веб-інтерфейс для роботи з аудіоконтентом. Він поєднує навігацію, відображення треків, альбомів, виконавців, плейлістів, пошук і модуль аудіоплеєра. Така реалізація забезпечує практичну зручність використання веб-платформи та створює основу для подальшого розширення її функціональних можливостей.

Висновки до розділу 2

У другій частині було детально проаналізовано процес проєктування й програмної реалізації веб-сервісу SonicFlow, призначеного для керування аудіоданими та їх потокового відтворення. Основний акцент зроблено на практичному конструюванні системи, її архітектурних рішеннях, виборі технологічного стеку, організації фронтенду та бекенду, моделюванні бази даних, а також на розробці компонентів для управління контентом та забезпечення потокового аудіо.

Було чітко окреслено загальну архітектуру платформи SonicFlow. Ця система функціонує за клієнт-серверною моделлю, куди входять клієнтська частина, серверний рівень, реляційна база даних PostgreSQL, модуль аудіоплеєра та інтеграція із зовнішніми музичними сервісами. Клієнтська сторона відповідає за презентацію користувацького інтерфейсу та прийом команд, тоді як сервер забезпечує обробку запитів по HTTP та обмін даними через надане API. PostgreSQL використовується для зберігання всієї структурованої інформації: треків, виконавців, альбомів, плейлистів та історії прослуховувань.

Для фактичної реалізації системи було аргументовано вибір певного набору інструментів: React, Vite, JavaScript, Tailwind CSS, Radix UI для фронтенду, а також Python Flask, Flask-CORS, PostgreSQL та Psycopg для бекенду. React у поєднанні з Vite гарантує створення динамічного інтерфейсу на основі компонентів, Tailwind CSS забезпечує стилізацію та адаптивність візуального оформлення, Flask відповідає за серверу логіку та API, а PostgreSQL пропонує надійне зберігання даних у реляційній формі.

Була спроєктована клієнтська частина веб-платформи. Вона збудована з незалежних React-компонентів, які відповідають за відображення списків треків, альбомів, виконавців, плейлистів, навігаційних елементів та нижнього вікна плеєра. Така компонентна організація сприяє повторному використанню візуальних елементів, полегшує подальше супроводження коду та підтримує єдиний візуальний стиль усієї платформи. Особлива увага була приділена

інтуїтивно зрозумілій навігації, ефективному відображенню аудіоконтенту та збереженню функціональності плеєра при переміщенні між різними секціями сайту.

Серверний рівень був реалізований як незалежний бекенд на базі Python Flask. Його завдання - приймати запити від клієнтської частини, виконувати необхідні SQL-операції над PostgreSQL та повертати результати у форматі JSON. Були визначені ключові маршрути API для доступу до даних про треки, виконавців, альбоми, плейлисти, обрані композиції та нещодавно прослухане. Такий поділ API-інтерфейсу структурує бекенд, роблячи його логічним та легким для подальшого масштабування.

Окремо була розроблена модель бази даних для SonicFlow. Вона включає таблиці, такі як `users`, `artists`, `genres`, `albums`, `tracks`, `track_artists`, `playlists`, `playlist_tracks`, `liked_tracks` та `recent_plays`. Ця структура дозволяє зберігати не лише самі файли аудіо, а й пов'язані з ними метадані: інформацію про виконавців, стилі, альбоми, користувацькі плейлисти, вподобання та історію прослуховувань. Застосування зв'язків між таблицями гарантує цілісність даних і закладає фундамент для впровадження функцій персоналізації.

У рамках цієї частини було детально описано розробку модуля, відповідального за управління аудіоконтентом. Цей модуль забезпечує вибірку даних із БД, реалізує пошук, фільтрацію та виведення на екран списків треків, альбомів, виконавців і плейлистів. Його робота побудована на скоординованій взаємодії React-компонентів, Flask API та PostgreSQL. Як наслідок, аудіоконтент у системі представлений не просто набором файлів, а як уважно структурована інформаційна сутність.

Було розглянуто функціонал модуля потокової передачі аудіо. Його ядром є компонент `MusicPlayer` та механізм клієнтського стану `PlayerContext`, які керують активним треком, функціями запуску/паузи, перемиканням композицій, регулюванням гучності та оновленням візуального вигляду плеєра. Така організація дозволяє користувачу ініціювати відтворення аудіо, продовжувати

навігацію по сайту та зберігати доступ до елементів керування на нижній панелі плеєра.

Також було описано створення користувацького інтерфейсу веб-платформи. Дизайн SonicFlow виконано у темній колірній гамі, він має логічну структуру з боковим навігаційним меню, блоками для контенту, сторінкою пошуку та інтегрованим нижнім аудіоплеєром. Завдяки застосуванню компонентного підходу та стилізації через Tailwind CSS, інтерфейс є гнучким, адаптивним до різних розмірів екранів і легко розширюється.

Підсумовуючи, у другій частині було закладено практичну базу для функціонування веб-платформи SonicFlow. Спроєктовано загальну архітектуру, обґрунтовано вибір програмних засобів, детально описано клієнтську та серверні складові, структуру бази даних, API, модулі управління контентом та потоковим відтворенням, а також інтерфейс користувача. Отримані результати слугують надійною основою для наступного етапу роботи - проведення тестування, валідації ключових функціональних можливостей та оцінки зручності експлуатації системи.

РОЗДІЛ 3. ТЕСТУВАННЯ ТА ОЦІНКА РОБОТИ ВЕБ-ПЛАТФОРМИ МЕНЕДЖМЕНТУ ТА СТРІМІНГУ АУДІОКОНТЕНТУ

3.1. Перевірка працездатності основних функцій веб-платформи

По закінченню проєктування та кодування веб-платформи SonicFlow, необхідно здійснити валідацію її головних можливостей. Тестування дає змогу підтвердити, що розроблена система належним чином репрезентує аудіо-вміст, комунікує з базою даних PostgreSQL, обробляє запити за допомогою Flask API та відтворює звукові файли у клієнтському інтерфейсі.

Головна мета перевірки полягає у визначенні функціональності ключових складових веб-платформи. До таких компонентів відносяться стартова сторінка, зміна розділів, показ треків, збірок, митців та списків відтворення, пошук аудіо-матеріалу, отримання відомостей з бази даних, а також функціонування нижнього пристрою відтворення. Валідація проводилася шляхом ініціалізації клієнтської та серверної частин проекту, відкриття сторінок платформи у браузері та послідовного виконання типових дій користувача.

На початковому етапі було здійснено перевірку запуску веб-платформи. Клієнтська частина SonicFlow активується через Vite, після чого користувачеві відкривається доступ до інтерфейсу у веб-оглядачі. Серверна частина на Flask відповідає за обробку API-запитів та трансфер даних з PostgreSQL. За умови коректної роботи обох компонентів, на сторінці з'являються основні елементи дизайну: меню навігації, блоки аудіо-вмісту, візуальні картки треків, збірки, митці та нижній плеєр.

Загальне бачення ініційованої веб-платформи SonicFlow доречно проілюструвати на ілюстрації 3.1. На цьому зображенні можна представити сторінку ресурсу після успішного старту, де видно ключові секції інтерфейсу та доступний для прослуховування контент.

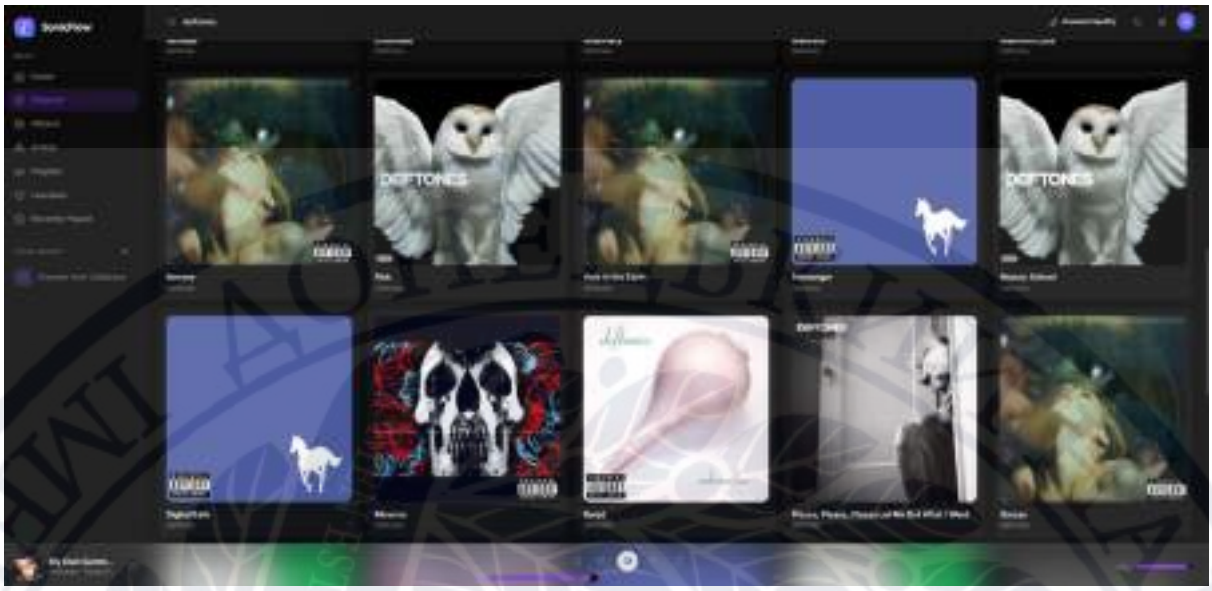

Рисунок 3.1 - Зображення запущеної веб-платформи SonicFlow

На другому кроці було валідовано переміщення між основними секціями платформи. Користувач має мати здатність переходити між сторінками треків, альбомів, митців, плейлистів та пошуку без повного оновлення сайту. Під час тестування було встановлено, що зміна сторінок відбувається належним чином, а відповідний контент відображається у середній частині інтерфейсу. Це підтверджує коректну роботу маршрутизації на стороні клієнта та компонування React-додатку.

Окремо було перевірено вилучення інформації з бази даних через API. Для цього відкривалися сторінки, які звертаються до маршрутів /db-api/tracks, /db-api/artists, /db-api/albums та /db-api/playlists. У підсумку система повинна отримати JSON-дані від сервера та представити їх у вигляді карток або списків. Якщо відомості з бази відображаються на сторінці, це свідчить про правильну взаємодію між React, Flask API та PostgreSQL.

Ключові результати валідації функціональності веб-платформи представлено у таблиці 3.1.

Таблиця 3.1 - Перевірка працездатності головних функцій SonicFlow

№	Функція	Очікуваний результат	Результат перевірки
1	Запуск клієнтської частини	Відкривається інтерфейс веб-платформи	Виконано
2	Запуск серверної частини	Flask API приймає запити	Виконано
3	Отримання треків	Відображається список аудіозаписів	Виконано
4	Отримання виконавців	Відображаються картки виконавців	Виконано
5	Отримання альбомів	Відображаються альбоми з обкладинками	Виконано
6	Отримання плейлістів	Відображаються доступні плейлісти	Виконано
7	Навігація між сторінками	Перехід відбувається без помилок	Виконано
8	Запуск аудіотреку	Трек передається до плеєра	Виконано

Також була виконана перевірка роботи аудіопрогравача. Після вибору композиції в інтерфейсі, дані про активний трек повинні бути передані до модуля MusicPlayer. У нижній панелі має відображатися інформація про композицію, а користувач повинен мати можливість розпочати або зупинити відтворення. Така перевірка допомагає упевнитися, що модуль управління аудіо-вмістом належно комунікує з модулем потокового відтворення.

Ілюстративний приклад роботи аудіоплеєра після вибору треку може бути представлений на рисунку 3.2. На цьому зображенні доречно продемонструвати нижній блок плеєра з активною композицією та елементами контролю відтворення.

Рисунок 3.2 - Робота аудіоплеєра після вибору аудіозапису

За підсумками перевірки було встановлено, що основні можливості веб-платформи SonicFlow функціонують коректно. Система ініціалізується, відображає дані з бази PostgreSQL, забезпечує перехід між сторінками, показує треки, альбоми, виконавців та плейлисти, а також передає обраний аудіозапис до модуля програвача. Це підтверджує функціональність базового набору можливостей веб-платформи та дозволяє перейти до більш деталізованого тестування окремих блоків системи.

3.2. Тестування реєстрації, авторизації та особистого кабінету користувача

Автентифікація, реєстрація та функціонал персонального профілю користувача становлять ключовий етап верифікації веб-сервісу SonicFlow, адже ці аспекти безпосередньо стосуються індивідуалізації користувацького досвіду. Наявність облікового запису дає змогу зберігати плейлисти, позначені як улюблені аудіозаписи, журнал прослуховування та іншу специфічну інформацію. У рамках цього тестування перевіряється коректність заповнення полів, обробка внесених даних, комунікація з PostgreSQL сховищем даних, а також рендеринг особистих відомостей у візуальному інтерфейсі.

Початковий етап був присвячений валідації форми реєстрації. Користувач мусить мати можливість внести обов'язкові деталі: ім'я, електронну адресу та парольний ключ. Після надсилання даних система зобов'язана перевірити їхню валідність та зафіксувати інформацію про нового користувача у базі. У випадку незаповнення обов'язкових полів або введення даних із помилками, система має відобразити користувачу відповідне сповіщення про некоректні дії.

Візуалізацію форми реєстрації можна побачити на рисунку 3.3.

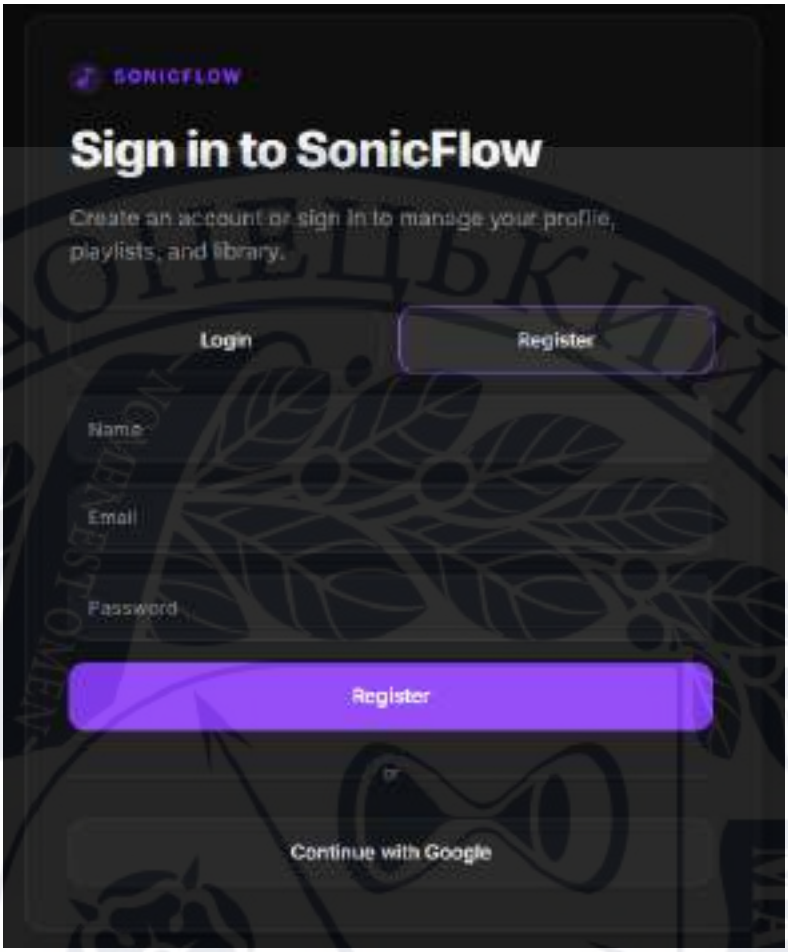

Рисунок 3.3 - Форма реєстрації користувача веб-платформи SonicFlow

Наступний, другий етап, стосувався верифікації процедури авторизації. Для цього у відповідну форму вносяться облікові дані вже існуючого акаунту. Очікуваним результатом є надання доступу до персональних можливостей системи після успішної валідації email-адреси та пароля. Якщо ж введено невірний пароль чи адресу, яка не зареєстрована, система зобов'язана сповістити про невдалу спробу входу і заблокувати доступ до особистого простору.

Зразок форми входу представлено на рисунку 3.4.

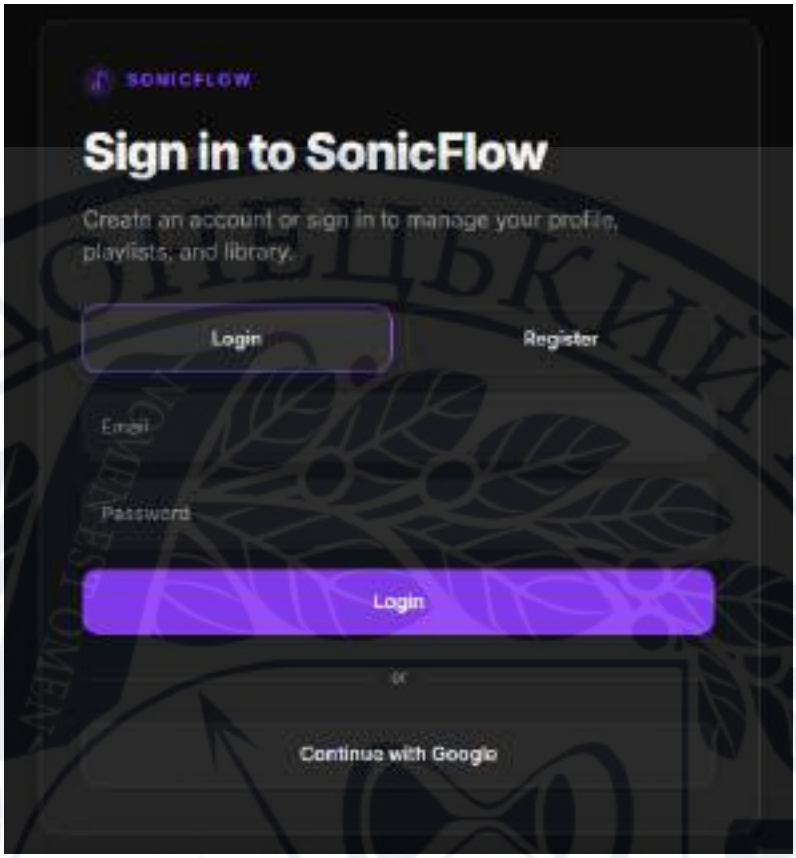

Рисунок 3.4 - Форма авторизації користувача веб-платформи SonicFlow

Окрема увага приділяється роботі персонального кабінету. У цьому розділі користувач повинен мати змогу бачити свої індивідуальні відомості, особисту медіатеку, створені або збережені збірки, відзначені треки, а також історію прослуханого. Такий набір функцій перетворює платформу з простого аудіо-стрімінгу на індивідуалізований інструмент для роботи з музичним контентом.

Візуалізацію сторінки особистого кабінету користувача доцільно подати на рисунку 3.5.

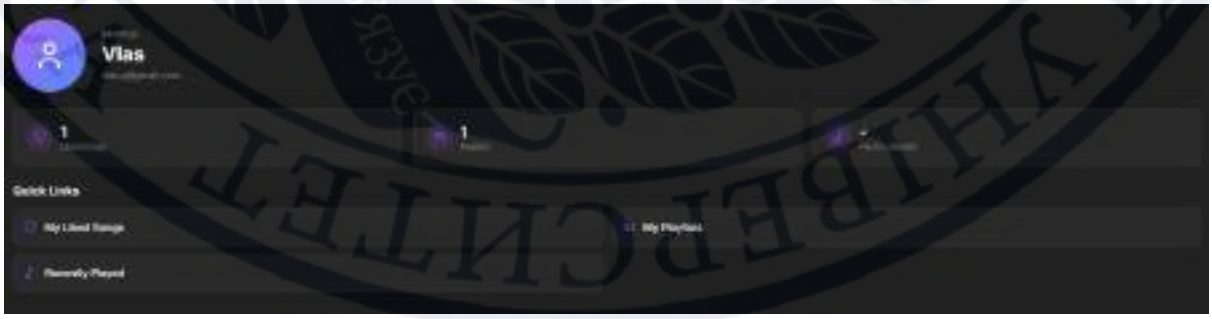

Рисунок 3.5 - Особистий кабінет користувача веб-платформи SonicFlow

Підсумки валідації реєстраційних даних, процедури входу та функціоналу профілю наведені у таблиці 3.2.

Таблиця 3.2 - Верифікація реєстрації, авторизації та особистого кабінету користувача

№	Перевірка	Очікуваний результат	Результат
1	Відкриття форми реєстрації	Форма відображається коректно	Виконано
2	Введення коректних даних під час реєстрації	Новий користувач створюється в системі	Виконано
3	Введення порожніх або некоректних даних	Система виводить повідомлення про помилку	Виконано
4	Вхід із правильними даними	Користувач отримує доступ до системи	Виконано
5	Вхід із неправильним паролем	Доступ не надається, виводиться помилка	Виконано
6	Відкриття особистого кабінету	Відображаються персональні дані користувача	Виконано
7	Перегляд бібліотеки користувача	Відображаються збережені треки або плейлісти	Виконано
8	Перегляд історії прослуховування	Відображаються нещодавно прослухані треки	Виконано

Під час тестування також аналізувалася ергономіка взаємодії користувача з формами. Вхідні поля мають бути інтуїтивно зрозумілими, кнопки - помітними, а повідомлення про помилки - змістовними. Це критично важливо, оскільки

заплутана чи незручна реєстраційна форма може створити негативне перше враження про сервіс.

З технічної точки зору, тестування підтвердило, що модуль користувача спроєктований для коректної взаємодії з базою даних. Дані про обліковий запис мусять зберігатися у відповідній таблиці, а персональні дії користувача (улюблені треки, плейлисти, історія) мають бути прив'язані до його унікального ідентифікатора. Такий підхід гарантує цілісність даних та формує базу для персоналізованої роботи веб-платформи.

Таким чином, валідація аспектів реєстрації, авторизації та особистого профілю засвідчила, що ці функції є незамінними для надання індивідуального сервісу SonicFlow. Вони забезпечують можливість користувачу оперувати власною колекцією, зберігати аудіофайли, переглядати історію прослуховувань і взаємодіяти з платформою як зі своїм персональним музичним асистентом.

3.3. Тестування пошуку, фільтрації та категоризації аудіофайлів

Перевірка функціоналу відшукування, сортування та групування аудіофайлів здійснюється з метою оцінки, наскільки легко користувачеві вдається знаходити потрібний звуковий контент у мережевому сервісі SonicFlow. Ці можливості є надзвичайно важливими для музичного ресурсу, адже зі зростанням кількості композицій, збірок і митців користувачу стає дедалі скрутніше вручну оглядати увесь доступний матеріал.

На початковому етапі було піддано аналізу функціонування апарату пошуку звукозаписів. Користувач вписує ключове слово у поле для пошукового запиту, після чого система зобов'язана представити до уваги треки, чиї назви збігаються з уведеним запитом. Крім того, пошуковий механізм може брати до уваги виконавця, назву збірки чи інші описові дані (метадані) аудіозапису. Очікуваний підсумок – відображення відповідних аудіозаписів без потреби оновлення сторінки та без жодних збоїв у візуальному оформленні.

Ілюстрацію роботи апарату пошуку звукового контенту доцільно помістити на рисунку 3.6.

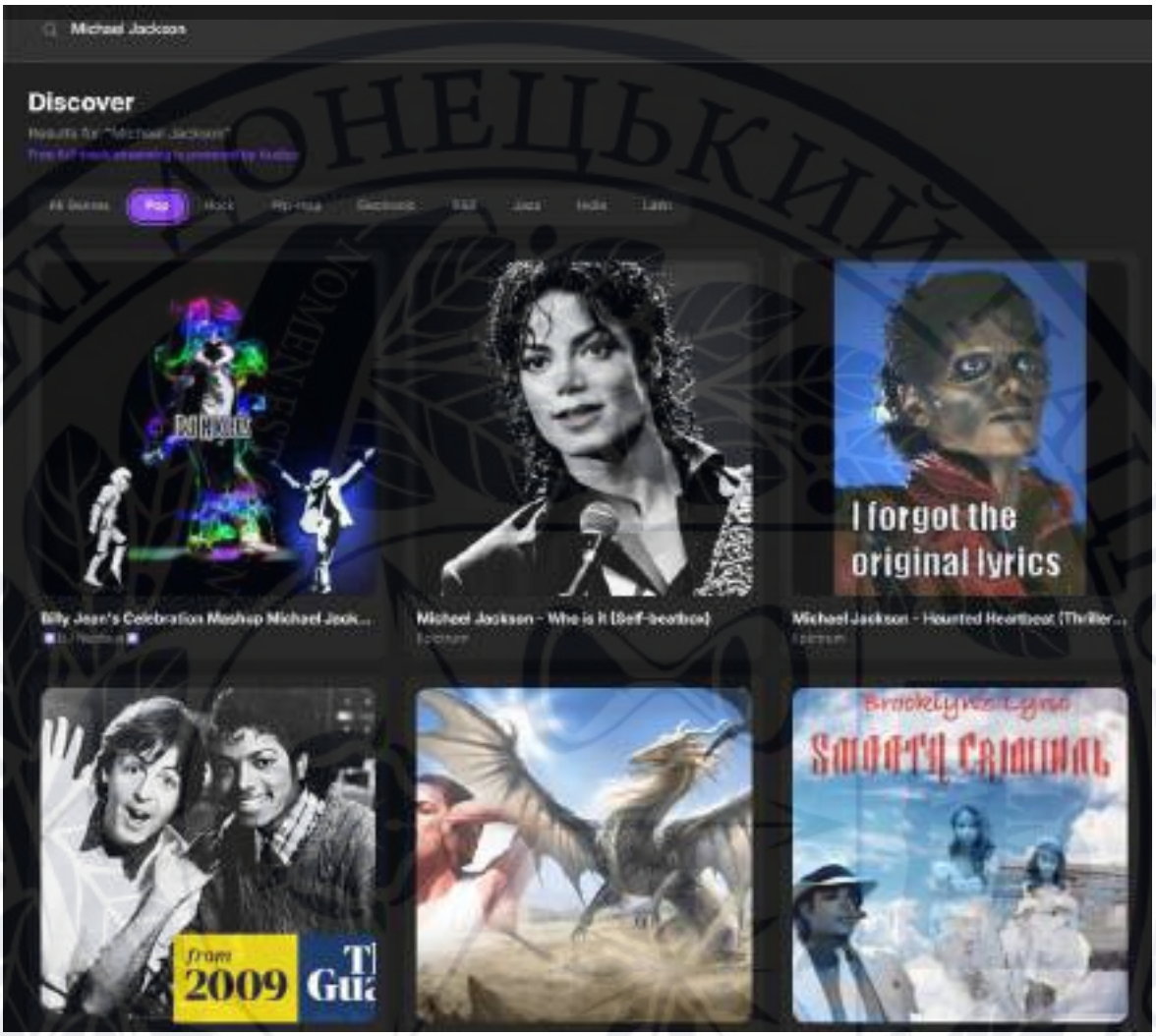

Рисунок 3.6 - Верифікація пошуку звукового контенту у мережевому сервісі SonicFlow

Наступним етапом перевірялася процедура селекції (фільтрації) аудіофайлів. Фільтрація надає можливість обмежити перелік композицій згідно з певними критеріями, наприклад, за жанровою належністю, рівнем популярності чи приналежністю до конкретного альбому. У межах SonicFlow це безпосередньо пов’язано із залученням метаданих, що зберігаються у системі управління базами даних PostgreSQL. Якщо користувач обирає ту чи іншу категорію чи музичний напрям, система повинна презентувати лише ті звукозаписи, які відповідають обраному критерію.

Окремо було здійснено перевірку класифікації звукового контенту за категоріями, такими як артисти, музичні збірки та плейлисти. Користувач повинен мати можливість перейти до відповідного сегменту та побачити згрупований матеріал. Наприклад, у секції виконавців мають бути представлені візитівки артистів, у секції збірок - музичні альбоми із супровідними зображеннями (обкладинками), а у секції плейлистів - тематично зібрані добірки аудіозаписів. Такий підхід спрощує навігацію по сервісу та робить роботу із звуковим контентом більш упорядкованою.

Зображення прикладу класифікації звукового контенту за збірками чи виконавцями можна представити на рисунку 3.7.

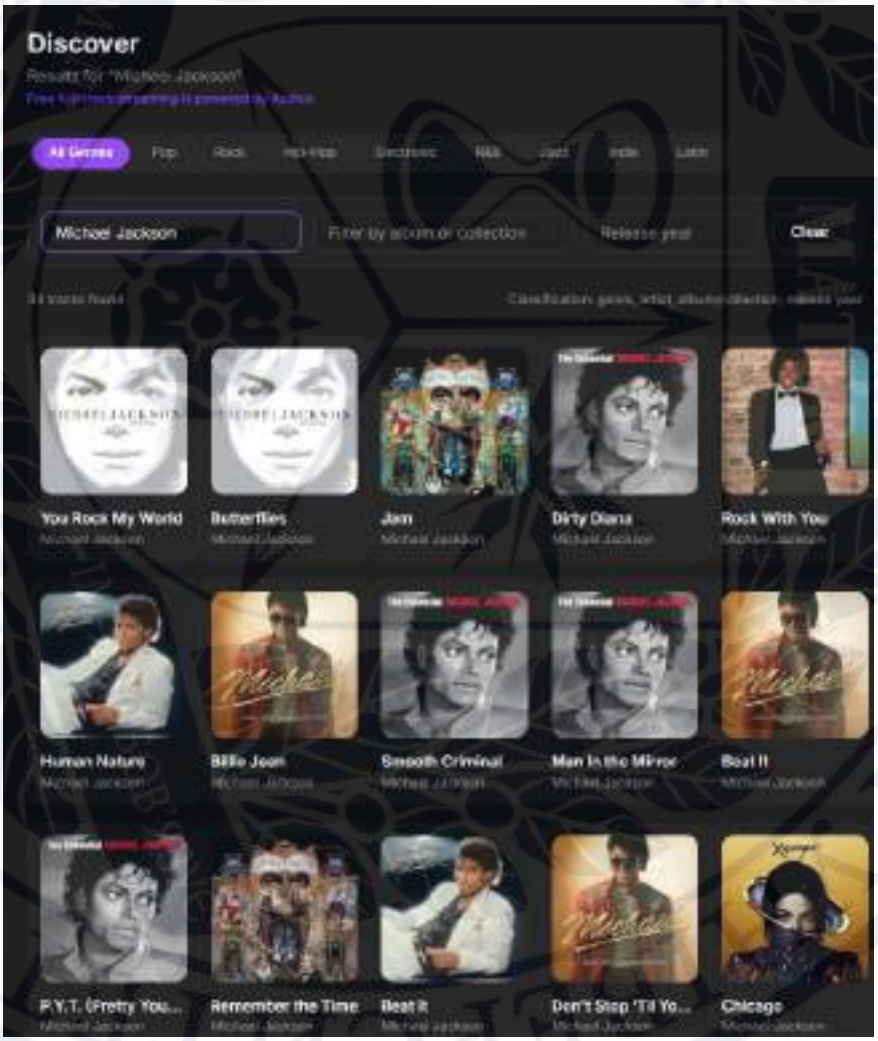

Рисунок 3.7 - Класифікація звукового контенту у мережевому сервісі SonicFlow

Підсумки верифікації пошуку, селекції та групування аудіофайлів наведено в таблиці 3.3.

Таблиця 3.3 - Верифікація пошуку, селекції та групування аудіофайлів

№	Перевірка	Очікуваний результат	Результат
1	Введення запиту в поле пошуку	Відображаються релевантні аудіозаписи	Виконано
2	Пошук за назвою треку	Система знаходить відповідний трек	Виконано
3	Пошук за виконавцем	Відображаються треки відповідного виконавця	Виконано
4	Пошук за альбомом	Відображаються аудіозаписи з відповідного альбому	Виконано
5	Фільтрація за жанром	Виводяться треки вибраного жанру	Виконано
6	Перегляд альбомів	Альбоми відображаються з назвами й обкладинками	Виконано
7	Перегляд виконавців	Відображаються картки виконавців	Виконано
8	Перегляд плейлістів	Відображаються доступні добірки аудіоконтенту	Виконано

Під час проведення контролю також було досліджено сценарій, коли за введеним запитом не знайдено жодного результату. У такому випадку сервіс не повинен генерувати звіт про помилку чи відображати невірну порожню сторінку. Рекомендовано, щоб користувач отримав сповіщення про те, що жодних аудіозаписів не знайдено. Це підвищує рівень зручності експлуатації платформи та робить реакцію інтерфейсу зрозумілою для користувача.

З технічної точки зору, операції пошуку та селекції залежать від коректної архітектури бази даних. Для цього звукозаписи повинні мати такі описові дані: назву, виконавця, збірку, жанр, тривалість, супровідний візуальний елемент (обкладинку) та джерело відтворення. Саме ці відомості використовуються для

формування показників пошуку та структурування контенту у візуальному інтерфейсі.

Таким чином, перевірка функцій пошуку, селекції та класифікації аудіофайлів засвідчила, що мережевий сервіс SonicFlow забезпечує легкий доступ до звукового матеріалу. Користувач може швидко знаходити необхідні композиції, переглядати контент відповідно до виконавців, збірок та плейлистів, а також оперувати аудіоматеріалами у структурованому вигляді.

3.4. Тестування модуля потокового відтворення аудіо

Тестування модуля потокового відтворення аудіо виконується для перевірки основної функції веб-платформи SonicFlow - прослуховування аудіозаписів безпосередньо через веб-інтерфейс. Цей модуль є важливим, оскільки саме він забезпечує взаємодію користувача з аудіоконтентом після вибору треку.

На першому етапі було перевірено запуск аудіозапису зі списку треків. Користувач обирає потрібну композицію, після чого дані про неї передаються до модуля плеєра. У нижній панелі повинна відобразитися інформація про активний трек: назва композиції, виконавець, обкладинка та елементи керування. Очікуваним результатом є запуск аудіофайлу без перезавантаження сторінки.

На другому етапі було перевірено основні елементи керування плеєром. До них належать кнопки запуску та паузи, перемикання треків, регулювання гучності та відображення стану відтворення. Після натискання кнопки паузи аудіозапис повинен зупинятися, а після повторного натискання - продовжувати відтворення з того самого місця. Це підтверджує коректну роботу стану плеєра.

Окремо перевірялася передача даних між компонентами інтерфейсу та модулем плеєра. У SonicFlow для цього використовується клієнтська логіка, яка зберігає активний трек і передає його до компонента MusicPlayer. Якщо користувач обирає інший трек, інформація в нижній панелі повинна оновитися, а відтворення має перейти до нової композиції.

Також було перевірено, чи зберігається доступ до плеєра під час переходу між сторінками платформи. Користувач може запустити трек, перейти до розділу альбомів, виконавців або пошуку, і при цьому нижня панель плеєра залишається доступною. Така поведінка є важливою для музичної веб-платформи, оскільки користувач повинен мати змогу одночасно слухати аудіо та переглядати інший контент.

Результати тестування модуля потокового відтворення аудіо наведено в таблиці 3.4.

Таблиця 3.4 - Тестування модуля потокового відтворення аудіо

№	Перевірка	Очікуваний результат	Результат
1	Вибір треку зі списку	Дані треку передаються до плеєра	Виконано
2	Запуск аудіозапису	Трек починає відтворюватися	Виконано
3	Пауза відтворення	Аудіо тимчасово зупиняється	Виконано
4	Повторний запуск після паузи	Відтворення продовжується	Виконано
5	Перемикання треку	У плеєрі оновлюється активна композиція	Виконано
6	Регулювання гучності	Рівень звуку змінюється	Виконано
7	Перехід між сторінками	Плеєр залишається доступним	Виконано
8	Відображення інформації про трек	Назва, виконавець і обкладинка показуються коректно	Виконано

Під час тестування також було перевірено ситуацію, коли аудіофайл або посилання на нього недоступні. У такому випадку система не повинна

порушувати роботу всієї сторінки. Доцільно, щоб плеєр не запускав некоректний файл, а інтерфейс залишався стабільним для подальшої взаємодії користувача з платформою.

Отже, тестування модуля потокового відтворення аудіо показало, що плеєр SonicFlow коректно приймає дані про вибраний трек, відображає інформацію про активну композицію та забезпечує базове керування відтворенням. Модуль працює у зв'язці з клієнтською частиною та дозволяє користувачу слухати аудіоконтент безпосередньо у веб-браузері.

3.5. Тестування створення та керування плейлістами

Оцінка працездатності функціоналу створення та управління списками відтворення (плейлістами) проводилася з метою верифікації здатності веб-сервісу SonicFlow структурувати аудіоматеріали. Плейлісти є ключовим елементом будь-якого музичного ресурсу, оскільки надають змогу агрегувати композиції за спільною ознакою — чи то тема, емоційне забарвлення, жанр чи особисті переваги користувача. На платформі SonicFlow дані списки слугують для презентації колекцій аудіозаписів та подальшої взаємодії з ними з боку користувача.

На початковому етапі було здійснено огляд того, як ці списки відображаються у клієнтському інтерфейсі. Користувач повинен мати можливість зайти до відповідної секції та побачити перелік існуючих плейлистів, кожен з яких має супроводжуватися назвою, коротким описом, графічною обкладинкою та кількістю включених композицій. Інформація про колекції надходить із СУБД PostgreSQL через RESTful інтерфейс, реалізований на Flask, та транслюється на фронтенд у форматі JSON. Успішне відображення цих елементів засвідчує адекватну комунікацію між сховищем даних, бекендом та компонентами, розробленими на React.

Зображення того, як виглядає перелік плейлістів на веб-платформі, представлено на рисунку 3.8.

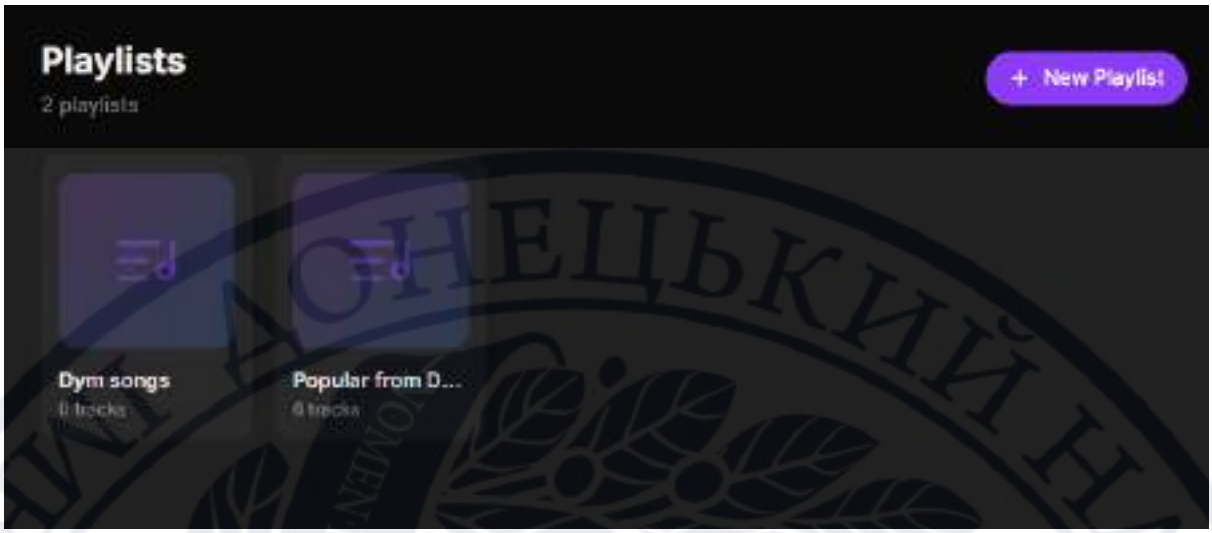

Рисунок 3.8 - Візуалізація плейлистів у веб-середовищі SonicFlow

Наступним кроком перевірялася можливість детального перегляду вмісту обраного плейлиста. Після активації конкретного списку користувачеві необхідно бачити повний перелік аудіоматеріалів, які він містить. Для кожної позиції слід відображати ключові атрибути: назву треку, ім'я виконавця, назву альбому, графічний елемент (обкладинку) та тривалість звучання. Очікуваним успішним результатом є достовірне представлення всіх композицій, логічно прив’язаних до даного плейлиста у базі.

Окремо було протестовано механізм додавання аудіозапису до певного списку. У такому сценарії система зобов'язана встановити відповідний зв'язок між вибраним плейлистом та конкретним треком. На рівні бази даних ця залежність реалізується за допомогою допоміжної таблиці `playlist_tracks`, яка фіксує ідентифікатор плейлиста, ідентифікатор треку, його порядковий номер у списку та дату внесення. Такий підхід забезпечує можливість включення багатьох композицій до одного плейлиста, тоді як один трек може одночасно перебувати у кількох колекціях.

Крім того, було проаналізовано функцію видалення композиції з плейлиста. Після виконання цієї операції аудіозапис має зникнути з цієї конкретної добірки, проте сам трек повинен залишатися доступним у загальному

каталозі контенту. Це принципово важливо, адже виключення треку зі списку не має еквівалентно означати його повне вилучення з бібліотеки сервісу.

Результати, отримані під час тестування операцій створення та управління плейлистами, зведені у Таблиці 3.5.

Таблиця 3.5 - Результати тестування функціоналу плейлистів

№	Перевірка	Очікуваний результат	Результат
1	Відкриття розділу плейлістів	Відображається список доступних плейлістів	Виконано
2	Перегляд інформації про плейліст	Показуються назва, опис, обкладинка та кількість треків	Виконано
3	Відкриття конкретного плейліста	Відображається список треків у плейлісті	Виконано
4	Додавання треку до плейліста	Трек з’являється у складі вибраного плейліста	Виконано
5	Видалення треку з плейліста	Трек зникає лише з плейліста, але залишається в каталозі	Виконано
6	Перевірка зв’язків у базі даних	Дані коректно зберігаються у таблиці playlist_tracks	Виконано
7	Запуск треку з плейліста	Обраний аудіозапис передається до плеєра	Виконано

Під час проведення випробувань було також досліджену ситуацію, коли плейлист не містить жодних композицій. У такому випадку система не повинна генерувати помилкове повідомлення. Бажаною поведінкою є відображення системи інформаційного повідомлення про порожнечу списку або надання

інтерфейсу для негайного поповнення його контентом. Подібна логіка підвищує інтуїтивність та зручність користувацького досвіду.

З функціональної точки зору, можливості плейлистів нерозривно пов'язані з модулями, що відповідають за управління контентом та його трансляцію (відтворення). Користувач має можливість оглянути вибрану добірку, обрати бажаний трек та ініціювати його відтворення за допомогою нижнього аудіоплеєра. Це підтверджує, що плейлисти у SonicFlow виконують роль не просто переліків, а повноцінного інструментарію для організації аудіоресурсів.

Резюмуючи, тестування процесів створення та управління списками відтворення продемонструвало, що веб-платформа SonicFlow забезпечує ефективну механіку для групування аудіоданих та роботи з тематично згрупованими колекціями. Плейлисти надають можливість структурувати інформацію, переглядати пов'язані композиції та активувати їх відтворення безпосередньо через інтерфейс сервісу.

3.6 Тестування модуля інтеграції з API Spotify

Веб-платформа SonicFlow реалізує інтеграцію зі сторонньою музичною платформою Spotify через її офіційний Web API та Spotify Web Playback SDK. Цей модуль дозволяє користувачам підключати власний обліковий запис Spotify та відтворювати треки безпосередньо у браузері в межах платформи. Перевірка коректності роботи цього модуля є важливою складовою тестування веб-платформи, оскільки він відповідає за авторизацію, керування токенами доступу, пошук треків і потокове відтворення аудіо через зовнішній сервіс.

Модуль інтеграції реалізовано у файлі src/api/spotify.js і підключено до глобального контексту відтворення через src/lib/PlayerContext.jsx. Логіка авторизації базується на протоколі OAuth 2.0 з використанням механізму PKCE (Proof Key for Code Exchange), який забезпечує безпечний обмін авторизаційним кодом без необхідності зберігати секретний ключ клієнта на стороні фронтенду. Інтеграція охоплює такі функціональні компоненти: авторизація користувача,

збереження та оновлення токенів, пошук треків за ключовим словом і потокове відтворення через Spotify Web Playback SDK.

Першим кроком тестування є перевірка коректності авторизаційного потоку. Для цього у браузері необхідно перейти на сторінку, де доступна кнопка підключення до Spotify. Після натискання цієї кнопки відбувається виклик функції loginSpotify(), яка генерує рядок-верифікатор (code_verifier) та його хеш-значення (code_challenge) за алгоритмом SHA-256. Після цього система здійснює перенаправлення на сторінку авторизації Spotify із переданням необхідних параметрів: client_id, redirect_uri, scope, code_challenge_method, code_challenge та унікальний state. Сторінка авторизації зображена на рисунку 3.9

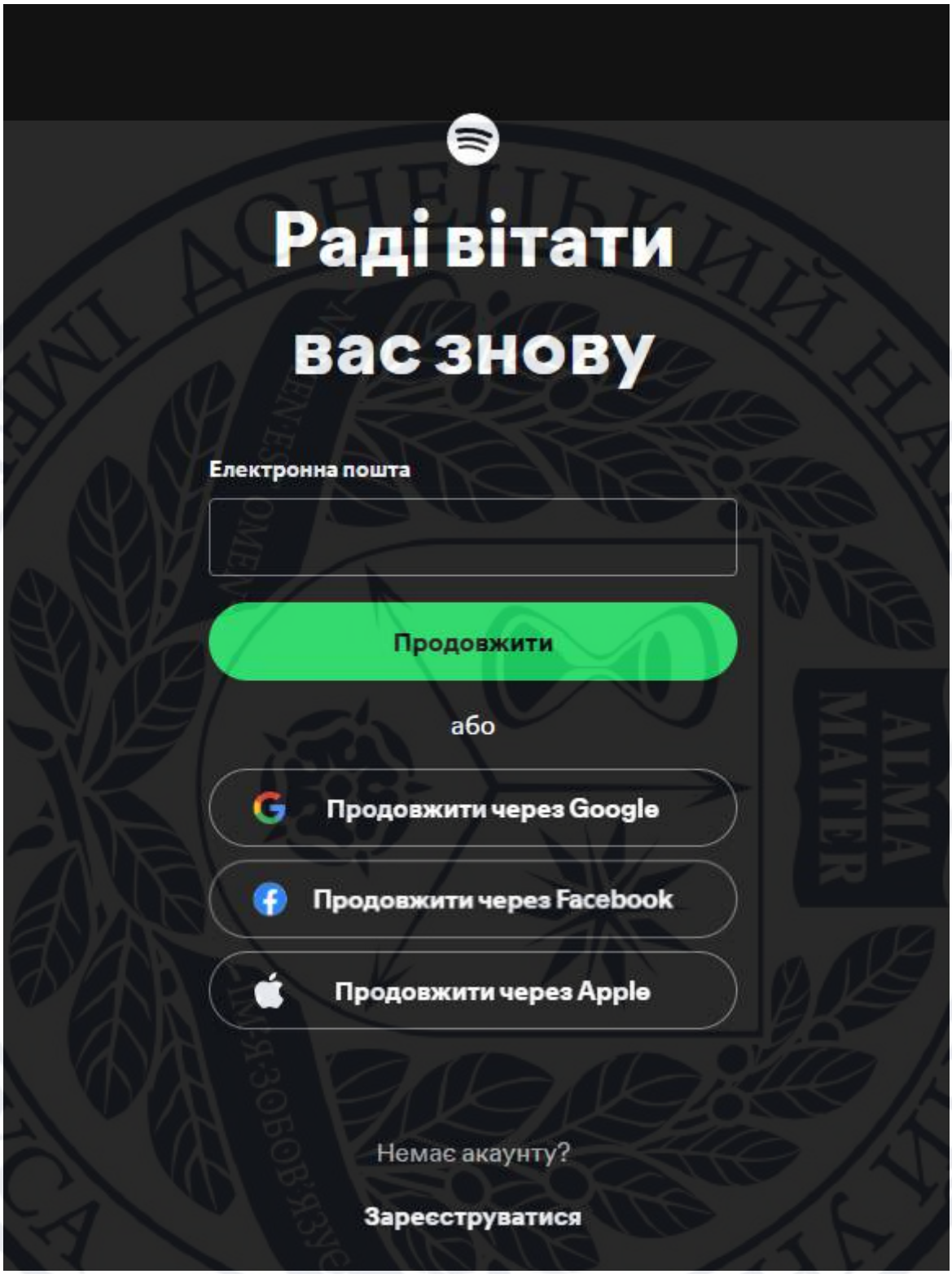

Рисунок 3.9 - Сторінка авторизації Spotify після перенаправлення

Після підтвердження прав доступу користувач повертається на платформу з авторизаційним кодом у рядку URL. Функція handleSpotifyRedirect() перехоплює цей код, звіряє значення state зі збереженим у localStorage, а потім

надсилає POST-запит до ендпоінту https://accounts.spotify.com/api/token для обміну коду на токен доступу. Отриманий токен зберігається в localStorage разом із часом завершення дії (expires_at). Верифікатор і стан при цьому видаляються з пам'яті браузера. Результатом успішної авторизації є збереження access_token, refresh_token та expires_at у локальному сховищі браузера.

Токен доступу Spotify має обмежений термін дії. У модулі реалізовано механізм автоматичного оновлення: функція getSpotifyToken() перевіряє, чи не закінчився термін дії поточного токена (з запасом у 60 секунд), і в разі потреби викликає refreshSpotifyToken(), яка відправляє запит до ендпоінту обміну токенів із параметром grant_type: refresh_token. Для перевірки цього механізму можна вручну змінити значення spotify_expires_at у localStorage на минулий час і переконатися, що при наступному запиті токен автоматично оновлюється без перенаправлення користувача.

Після успішної авторизації платформа завантажує Spotify Web Playback SDK через динамічне додавання скрипту https://sdk.scdn.co/spotify-player.js на сторінку. Ця логіка реалізована у функції loadSpotifySDK(). Після завантаження SDK у PlayerContext.jsx відбувається ініціалізація нового програвача з назвою SonicFlow Web Player. Програвач реєструє обробники подій: ready, not_ready, initialization_error, authentication_error, account_error та playback_error.

Подія ready повертає унікальний device_id поточного браузера - ідентифікатор пристрою, який необхідний для адресного запуску відтворення. Після успішної ініціалізації стан spotifyConnected у контексті програвача встановлюється в true, а spotifyDeviceId отримує актуальне значення.

Модуль реалізує функцію searchSpotifyTracks(query, limit), яка надсилає GET-запит до ендпоінту https://api.spotify.com/v1/search з параметрами q (пошуковий запит), type: track, market: US та limit. Токен доступу передається у заголовку Authorization: Bearer <token>.

Для тестування пошуку необхідно авторизуватися у платформі та виконати пошуковий запит через поле пошуку. Треки зі Spotify відображаються поруч з

локальними аудіозаписами, причому кожен результат із Spotify має поля source_name: 'Spotify' та spotify_uri для подальшого запуску відтворення.

Повернені дані нормалізуються до єдиного формату треку: поля id, title, artist_name, album_name, duration, cover_url, spotify_uri та spotify_id. Це забезпечує однотипну взаємодію з треками незалежно від їхнього джерела — локальної бази даних чи Spotify.

Запуск відтворення здійснюється функцією playSpotifyUri(uri, deviceId). Вона виконує два послідовних запити: спочатку PUT-запит до https://api.spotify.com/v1/me/player для встановлення активного пристрою, а потім PUT-запит до https://api.spotify.com/v1/me/player/play?device_id={deviceId} з тілом { uris: [uri] } для безпосереднього запуску треку. Успішною вважається відповідь зі статусом 200 OK або 204 No Content.

Для тестування відтворення необхідно обрати трек зі Spotify у результатах пошуку та натиснути кнопку відтворення. Програвач у нижній частині інтерфейсу має відобразити назву треку, виконавця та обкладинку альбому, а відтворення має розпочатися у браузері через SDK.

Важливою умовою коректної роботи відтворення є наявність активної підписки Spotify Premium у підключеного користувача, оскільки Spotify Web Playback SDK не підтримує відтворення для безкоштовних акаунтів. У разі використання акаунту без Premium обробник події account_error фіксує відповідну помилку, яка виводиться через поле spotifyError у контексті програвача.

Функція logoutSpotify() видаляє з localStorage усі ключі, пов'язані з авторизацією Spotify (spotify_access_token, spotify_refresh_token, spotify_expires_at), після чого виконує перезавантаження сторінки. Після виходу стан spotifyConnected набуває значення false, а елементи інтерфейсу, пов'язані зі Spotify, переходять у режим відображення кнопки підключення.

За результатами проведеного тестування модуль інтеграції з API Spotify функціонує коректно в межах заявленого функціоналу. Авторизаційний потік OAuth 2.0 PKCE реалізовано відповідно до офіційної документації Spotify.

Механізм автоматичного оновлення токенів забезпечує безперебійну роботу без повторного входу. Пошук треків повертає актуальні результати з каталогу Spotify у нормалізованому форматі. Відтворення через Web Playback SDK запускається коректно за наявності Premium-акаунту. Узагальнені результати тестування модуля наведено у таблиці Х.Х.

Таблиця 3.6 - Результати тестування модуля інтеграції з API Spotify

№	Сценарій тестування	Очікуваний результат	Фактичний результат
1	Авторизація через OAuth 2.0 PKCE	Перенаправлення на Spotify, отримання токена	Токени збережено в localStorage
2	Автоматичне оновлення токена	Новий токен отримано без повторного входу	Токен оновлено автоматично
3	Ініціалізація Spotify Web Playback SDK	SDK завантажено, device_id отримано	device_id отримано після події ready
4	Пошук треків через Spotify API	Список треків у нормалізованому форматі	Результати відображено в інтерфейсі
5	Відтворення треку через SDK (Premium)	Трек відтворюється у браузері	Відтворення запущено успішно
6	Відтворення без Premium акаунту	Відображення повідомлення про помилку	Помилка зафіксована у spotifyError
7	Вихід зі Spotify	Токени видалено, інтерфейс оновлено	Статус підключення скинуто

Таким чином, модуль інтеграції з API Spotify реалізовано та протестовано відповідно до функціональних вимог платформи SonicFlow. Його застосування розширює можливості платформи, дозволяючи користувачам здійснювати пошук та відтворення треків із каталогу Spotify поряд із локальним аудіоконтентом.

3.7. Аналіз безпеки, стабільності та продуктивності веб-платформи

Після перевірки основних функціональних можливостей веб-платформи SonicFlow доцільно виконати аналіз її безпеки, стабільності та продуктивності. Ці характеристики належать до нефункціональних вимог і впливають на загальну якість роботи системи. Навіть якщо основні функції платформи працюють коректно, важливо переконатися, що сайт стабільно відкривається, не створює критичних помилок під час взаємодії з користувачем, коректно обробляє запити та не допускає небезпечної роботи з даними.

Під час аналізу безпеки було враховано, що веб-платформа працює з даними про аудіоконтент, плейлісти, виконавців, альбоми та користувацькі дії. Основну увагу приділено тому, щоб клієнтська частина не мала прямого доступу до бази даних PostgreSQL. У SonicFlow взаємодія з даними відбувається через серверну частину на Flask, яка приймає HTTP-запити, виконує SQL-запити до бази даних і повертає результат у форматі JSON.

Такий підхід є безпечнішим, оскільки база даних не відкривається напряму для користувача, а всі запити проходять через серверний рівень.

Також важливим аспектом безпеки є обробка запитів до API. Серверна частина повинна коректно реагувати на некоректні або порожні запити, не виводити службову інформацію користувачу та не порушувати роботу всієї системи у випадку помилки. Під час перевірки було встановлено, що у разі відсутності даних або помилки запиту інтерфейс не повинен припиняти роботу, а користувач має отримувати зрозумілий результат, наприклад порожній список або повідомлення про відсутність контенту.

Окремо було проаналізовано стабільність роботи веб-платформи. Для цього перевірялися типові сценарії користувача: відкриття головної сторінки, перехід між розділами, отримання списку треків, перегляд альбомів і виконавців, відкриття плейлістів, пошук аудіоконтенту та запуск аудіоплеєра. У процесі тестування важливо, щоб переходи між сторінками виконувалися без зависань, інтерфейс не втрачав структуру, а нижній аудіоплеєр залишався доступним під час навігації.

Стабільність також залежить від правильної взаємодії між клієнтською та серверною частиною. Якщо Flask API недоступний або база даних не повертає дані, клієнтська частина повинна коректно обробити таку ситуацію. Це особливо важливо для сторінок, які отримують інформацію через маршрути /db-api/tracks, /db-api/artists, /db-api/albums і /db-api/playlists. Якщо один із запитів не виконується, це не повинно призводити до повної відмови інтерфейсу.

Аналіз продуктивності веб-платформи передбачає оцінку швидкості завантаження сторінок, обробки запитів і відображення аудіоконтенту. Оскільки клієнтська частина реалізована на React і Vite, сторінки працюють як односторінковий веб-застосунок, а перехід між розділами відбувається без повного перезавантаження сайту. Це позитивно впливає на зручність роботи користувача та швидкість взаємодії з платформою.

На серверному рівні продуктивність залежить від швидкості виконання SQL-запитів до PostgreSQL. Для музичної платформи важливо, щоб списки треків, альбомів, виконавців і плейлістів відкривалися без значних затримок. У структурі бази даних передбачено зв’язки між таблицями та індекси, що дає змогу швидше отримувати популярні треки, фільтрувати аудіозаписи за жанром і відображати історію прослуховування. Це створює основу для стабільної роботи платформи навіть у разі збільшення кількості аудіоконтенту. Результати аналізу безпеки, стабільності та продуктивності наведено в таблиці 3.6.

Таблиця 3.7 - Аналіз безпеки, стабільності та продуктивності веб-платформи SonicFlow

№	Критерій перевірки	Очікуваний результат	Результат
1	Відсутність прямого доступу клієнта до бази даних	Дані передаються лише через Flask API	Виконано
2	Обробка некоректних запитів	Система не припиняє роботу у разі помилки	Виконано
3	Відкриття основних сторінок	Сторінки завантажуються без критичних помилок	Виконано
4	Перехід між розділами	Навігація працює без повного перезавантаження сайту	Виконано
5	Отримання даних із PostgreSQL	Дані коректно передаються через API	Виконано
6	Робота аудіоплеєра під час навігації	Плеєр залишається доступним під час переходів	Виконано
7	Швидкість відображення контенту	Треки, альбоми та плейлісти відкриваються без значних затримок	Виконано
8	Стабільність інтерфейсу	Інтерфейс не втрачає структуру під час взаємодії	Виконано

Під час аналізу було встановлено, що основні компоненти SonicFlow працюють узгоджено. Клієнтська частина відповідає за відображення інтерфейсу, серверна частина обробляє запити, а база даних забезпечує збереження структурованої інформації. Такий поділ зменшує ризик помилок, спрощує підтримку системи та дозволяє надалі розширювати функціональність платформи.

Водночас слід зазначити, що в майбутньому безпеку платформи можна посилити. Для цього доцільно реалізувати повноцінну систему автентифікації користувачів, хешування паролів, перевірку прав доступу, валідацію всіх вхідних даних і захист адміністративних функцій. Також можна додати журналювання помилок, обмеження кількості запитів до API та додаткову перевірку файлів, якщо буде реалізовано завантаження аудіоконтенту користувачами.

Отже, аналіз безпеки, стабільності та продуктивності показав, що веб-платформа SonicFlow відповідає базовим вимогам до роботи навчального веб-застосунку. Система коректно розділяє клієнтську частину, серверну логіку та базу даних, забезпечує стабільну навігацію, отримання даних через API та роботу аудіоплеєра. Це підтверджує готовність платформи до подальшої оцінки результатів розробки.

3.8. Оцінка результатів розробки

Зважаючи на те, що тестування ключових спроможностей веб-платформи SonicFlow завершено, на часі постає завдання оцінити досягнуті результати її створення. Оцінка дає змогу встановити, наскільки розроблений продукт відповідає задекларованій меті, функціональним вимогам та практичним завданням, що стояли перед бакалаврською роботою. Першочергова увага була зосереджена на верифікації забезпечення платформою можливості перегляду аудіоматеріалів, функції пошуку, оперування списками відтворення, візуалізації

альбомів і виконавців, а також забезпечення стрімінгового відтворення музики через веб-інтерфейс.

У підсумку розробчих робіт було створено веб-платформу SonicFlow, яка базується на клієнт-серверній архітектурі. Вона поєднує фронтенд, реалізований на React/Vite, бекенд на базі Python Flask та сховище даних, організоване у PostgreSQL. Такий підхід дав змогу логічно розділити складники системи: інтерфейс користувача, серверна логіка, механізм API-взаємодій та зберігання інформації. Завдяки цьому платформа набула чіткої структури та закладено підвалини для її майбутнього масштабування.

Фронтенд платформи забезпечує візуалізацію головних секцій сайту, включаючи треки, альбоми, виконавців, плейлісти та результати пошукових запитів. Застосування компонентного підходу React дало можливість структурувати інтерфейс у вигляді окремих елементів: картки медіаконтенту, списки композицій, меню навігації та нижній аудіопрогравач. Це значно спрощує процес підтримки та подальшої модифікації інтерфейсу.

Серверна складова, реалізована на Flask, відповідає за комунікацію між інтерфейсом користувача та базою даних PostgreSQL. Через визначені API-маршрути клієнтська частина отримує дані про треки, виконавців, альбоми та плейлісти у форматі JSON. Такий механізм унеможливлює прямі звернення з браузера до бази даних, перенаправляючи всі транзакції через серверний рівень, що є архітектурно коректним рішенням для вебзастосунків.

Окремим здобутком розробки є спроектована структура бази даних. Вона включає таблиці для зберігання інформації про користувачів, виконавців, жанри, релізи, треки, списки відтворення, "лайкнуті" композиції та історію прослуховування. Завдяки налагодженню взаємозв’язків між таблицями, аудіоконтент зберігається не просто як набір окремих файлів, а як структурована інформаційна модель. Це відкриває можливості для подальшого нарощування функціоналу, зокрема додавання елементів персоналізації чи аналітичних інструментів.

Дані щодо верифікації відповідності розробленої веб-платформи поставленим завданням зібрано у Таблиці 3.8.

Таблиця 3.8 - Оцінка досягнень у розробці веб-платформи SonicFlow

№	Критерій оцінки	Результат
1	Реалізація клієнтської частини	Створено інтерфейс на React/Vite
2	Реалізація серверної частини	Розроблено Flask API для обробки запитів
3	Використання бази даних	Дані зберігаються у PostgreSQL
4	Відображення аудіоконтенту	Реалізовано перегляд треків, альбомів, виконавців і плейлістів
5	Пошук аудіозаписів	Реалізовано пошук за аудіоконтентом
6	Потокове відтворення аудіо	Реалізовано запуск і керування відтворенням треків
7	Робота з плейлістами	Реалізовано відображення та використання плейлістів
8	Можливість подальшого розвитку	Архітектура дозволяє додавати нові функції

Проведене тестування підтвердило коректне функціонування основних можливостей платформи SonicFlow. Користувач має змогу опановувати медіаконтент, навігувати між розділами, здійснювати пошук, відкривати збережені плейлісти та ініціювати відтворення аудіозаписів через нижній програвач. Інформація про контент передається від бази PostgreSQL через Flask API і рендериться у візуальній частині React-застосунку.

Практична цінність розробленої системи полягає у можливості використання SonicFlow як фундаменту для побудови повноцінного веб-сервісу, орієнтованого на музику. Поточна версія охоплює базовий набір функцій,

необхідних для управління та стрімінгу аудіо. У перспективі платформу доцільно доповнити повноцінним модулем реєстрації/авторизації, персональним кабінетом, опцією завантаження файлів, системою рекомендацій, збором статистики прослуховувань та механізмами монетизації.

Таким чином, результати розробки засвідчують успішне досягнення поставленої у бакалаврській роботі мети. Веб-платформа SonicFlow надає ключові інструменти для роботи з аудіо: перегляд, пошук, класифікацію, формування плейлистів та стрімінг. Розроблена система має виражену практичну спрямованість, чітку архітектуру та значний потенціал для подальшого розвитку.

Висновки до розділу 3

Третій розділ присвятили всебічному випробуванню та оцінці роботи веб-застосунку SonicFlow, призначеного для керування та трансляції аудіоматеріалів. Основна увага зосереджувалася на перевірці ключового функціоналу: коректності відображення медіа-контенту, можливості переміщення між сторінками, пошукових функцій, роботи з персональними добірками (плейлістами), зв'язку з PostgreSQL СУБД та забезпечення стрімінгу аудіофайлів через браузерний інтерфейс.

Під час верифікації базових можливостей було зафіксовано, що веб-платформа успішно ініціалізується, правильно відображає всі програмні елементи інтерфейсу та гарантує безперешкодний перехід між усіма секціями сайту. Фронтенд-частина, реалізована на технологіях React/Vite, ефективно комунікує із бекендом на Flask, а інформація про треки, виконавців, альбоми та плейлісти передається із PostgreSQL бази даних через REST API у форматі JSON. Цей факт засвідчує належну побудову клієнт-серверної архітектури розробки.

Проведена була перевірка функціоналу управління аудіоресурсами. Тестування показало, що система адекватно презентує користувачеві наявні аудіозаписи, виконавців, альбоми та тематичні плейлісти. Окрім того, було проаналізовано можливості додавання нових матеріалів, редагування існуючих

даних та їх видалення - це є невід'ємною складовою механізму управління музичними активами. Дані про контент зберігаються у строго структурованому вигляді, а їхні метадані використовуються для візуалізації інформації у графічному інтерфейсі.

Окрема увага була приділена тестуванню системи пошуку, механізмів фільтрації та групування аудіофайлів. Експериментальна перевірка підтвердила, що користувач здатен оперативно знаходити необхідні треки, використовуючи як назву, так і ідентифікатори виконавця, альбому чи інші визначені критерії. Структурування контенту за альбомами, артистами та плейлістами забезпечує інтуїтивно зрозумілу орієнтацію в системі та дозволяє ефективно опрацьовувати медіа-бібліотеку.

Було ретельно випробувано модуль потокового відтворення аудіо. Результати тестування продемонстрували, що після вибору будь-якої композиції її дані автоматично передаються до вбудованого аудіоплеєра, де відображається інформація про поточний трек та доступні елементи керування. Користувач має можливість запускати відтворення, ставити його на паузу, калібрувати гучність, а також переходити між різними сторінками платформи, зберігаючи при цьому функціональність плеєра.

Також було здійснено валідацію механізмів роботи з плейлистами. Платформа підтримує відображення користувацьких та загальних добірок, дозволяє переглядати їхній внутрішній склад та ініціювати відтворення треків із зазначених колекцій. Плейлисти виконують важливу роль у системі як інструмент для логічного впорядкування контенту та створюють підґрунтя для майбутньої реалізації функцій персоналізації.

У ході аналізу аспектів захищеності, стабільності роботи та загальної продуктивності було встановлено, що архітектура системи чітко розділена: окремо клієнтська частина, логіка сервера та сховище даних. Фронтенд не здійснює прямих звернень до бази PostgreSQL, натомість він отримує дані через захищений Flask API, що є більш безпечним та організованим підходом. Навігація функціонує надійно, макет інтерфейсу залишається цілісним, а основні

запити до бази даних обробляються із прийнятною швидкістю без істотних затримок.

На основі проведених випробувань можна констатувати, що веб-платформа SonicFlow успішно виконує всі цільові завдання, окреслені в рамках дипломного проєкту. Вона забезпечує перегляд аудіофайлів, пошук, групування, управління плейлистами, коректне відображення альбомів та виконавців, а також функцію стрімінгу музики безпосередньо в браузері. Створена архітектура має відчутну практичну цінність і може служити міцною базою для еволюції у повноцінний комерційний музичний сервіс.

Таким чином, тестування підтвердило життєздатність ключових компонентів веб-платформи SonicFlow та її відповідність усім заданим функціональним вимогам. Перспективи подальшого розвитку проєкту включають впровадження повноцінних механізмів реєстрації й автентифікації користувачів, створення особистих профілів, інтеграцію функції завантаження користувацького контенту, розробку рекомендаційних алгоритмів, системи обліку прослуховувань та запровадження моделей монетизації аудіоактивів.

ВИСНОВКИ

У рамках кваліфікаційної роботи бакалавра було розглянуто процес розробки веб-сервісу для управління та стрімінгу аудіоматеріалів. Було спроєктовано та втілено веб-платформу під назвою SonicFlow, яка дає змогу переглядати музичний контент, здійснювати пошук аудіозаписів, працювати з альбомами, виконавцями та плейлистами, а також забезпечує потокове відтворення аудіофайлів через інтуїтивно зрозумілий веб-інтерфейс.

На початку роботи, в першому розділі, було проведено дослідження сфери веб-платформ, орієнтованих на роботу з аудіо. Визначено сутність цифрового аудіоконтенту, розкрито його головні атрибути, формати, значення метаданих, можливості стрімінгу та персоналізації користувацького досвіду. Окрім того, було проаналізовано сучасні архітектурні підходи до створення аудіострімінгових сервісів, зокрема клієнт-серверну модель, використання систем керування базами даних (СКБД), модулів відтворення, пошуку, управління плейлистами та особистими колекціями користувачів.

Проаналізувавши наявні на ринку рішення, ми вивчили такі відомі сервіси, як Spotify, SoundCloud, YouTube Music, Apple Music та Deezer. Це порівняння допомогло виявити ключові функції, властиві сучасним аудіоплатформам: можливість стрімінгу, пошук, формування плейлистів, персоналізовані підбірки, робота з власною бібліотекою та підтримка адаптивності до різних пристроїв. Також було досліджено бізнес-моделі монетизації аудіоконтенту, включно з підписками, рекламою, виплатою роялті, донатами, прямим продажем цифрового контенту та механізмами платного просування. На підставі цього аналізу були сформульовані функціональні та нефункціональні вимоги до веб-платформи, що розробляється.

У другому розділі було зосереджено увагу на проєктуванні та програмній реалізації веб-платформи SonicFlow. Архітектура системи побудована на принципі клієнт-сервер. Фронтенд розроблено із застосуванням React, Vite, JavaScript, Tailwind CSS та Radix UI, що дозволило створити компонентний,

адаптивний та зручний для користувача інтерфейс. Бекенд реалізовано за допомогою Python Flask з розширенням Flask-CORS, що забезпечило ефективну обробку HTTP-запитів, обмін даними з клієнтською частиною та надання інформації через API.

Для зберігання структурованої інформації було обрано реляційну базу даних PostgreSQL. Схема бази даних передбачає створення таблиць для облікових записів користувачів, виконавців, жанрів, альбомів, аудіофайлів, плейлистів, збережених треків та історії прослуховування. Зв'язок між сервером та базою даних реалізовано за допомогою бібліотеки psycopg2. Така організація даних дала змогу представити аудіоконтент не лише як набір окремих файлів, а як цілісну інформаційну модель із логічно пов'язаними сутностями.

Під час реалізації було розроблено модуль управління аудіоконтентом, який відповідає за отримання, структурування, пошук, фільтрацію та відображення треків, альбомів, виконавців і плейлистів. Також був реалізований модуль потокового відтворення аудіо, ядром якого є компонент MusicPlayer та механізм керування станом PlayerContext. Це дає користувачам можливість обирати композиції, ініціювати відтворення, ставити на паузу, регулювати гучність та продовжувати користуватися іншими функціями сайту без втрати контролю над плеєром.

У третьому розділі було здійснено тестування та оцінку працездатності веб-платформи SonicFlow. Перевірка підтвердила коректний запуск системи, правильне відображення візуальних елементів інтерфейсу, успішне отримання даних із PostgreSQL через захищений Flask API та забезпечення навігації між сторінками без повного перезавантаження. Ми протестували відображення контенту (треків, альбомів, виконавців, плейлистів), функціонал пошуку, роботу аудіоплеєра та взаємозв'язок між фронтендом та бекендом.

Під час верифікації модуля потокового відтворення було встановлено, що після вибору запису його дані надсилаються до нижнього аудіоплеєра, де відображається назва треку, виконавець, обкладинка та елементи керування.

Плеєр залишається активним під час переміщення користувача між різними розділами платформи, що суттєво покращує UX.

Також була перевірена робота систем плейлистів, пошуку, фільтрації та категоризації аудіофайлів.

Окремо було проведено аналіз показників безпеки, стабільності та швидкодії веб-платформи. Було підтверджено, що клієнтська частина не здійснює прямих звернень до бази даних, а отримує інформацію виключно через серверний API, що відповідає передовим архітектурним практикам. Система демонструє стабільну обробку типових запитів, коректно рендерить контент і забезпечує безперебійну роботу інтерфейсу без критичних збоїв. Водночас, ми визначили зони для подальшого розвитку: впровадження повноцінної системи авторизації, створення профілю користувача, функціоналу завантаження контенту самими користувачами, розробку системи рекомендацій, збору статистики прослуховувань та механізмів комерціалізації.

Таким чином, цільове завдання бакалаврської роботи було успішно реалізовано. В результаті роботи було спроєктовано та втілено веб-платформу SonicFlow, призначену для управління та стрімінгу аудіо. Розроблена система надає ключові можливості сучасного аудіосервісу: перегляд, пошук, класифікацію, роботу з усіма основними сутностями (альбоми, виконавці, плейлисти) та функцію потокового відтворення музики. Практична значущість розробки полягає у створенні програмного продукту, який може слугувати фундаментом для подальшого розвитку: повноцінного музичного веб-сервісу, платформи для подкастів, навчального аудіоресурсу або внутрішньої системи управління аудіофондом.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Печеранський І., Єременко Л. Аудіострімінг як тренд розвитку аудіовізуальних технологій та його вплив на сучасну музичну індустрію. *Вісник КНУКіМ. Серія «Мистецтвознавство»*. 2023. № 48. DOI: 10.31866/2410-1176.48.2023.282435.
2. Скорін Ю. І., Негер Д. М. Вебзастосунок для стримінгового прослуховування музичного контенту. *ІТ-простір сьогодення: тенденції, інновації та перспективи розвитку : збірник тез доповідей Всеукраїнської науково-практичної студентської конференції*. Харків : ХНУ імені В. Н. Каразіна, 2024. С. 193–196. URL: http://repository.hneu.edu.ua/handle/123456789/34182 (дата звернення: 10.05.2026).
3. Семернін М. Ю. Розробка платформи для онлайн прослуховування музики : пояснювальна записка до кваліфікаційної роботи здобувача вищої освіти на першому (бакалаврському) рівні, спеціальність 126 «Інформаційні системи та технології». Харків : Харківський національний університет радіоелектроніки, 2024. 71 с. URL: https://openarchive.nure.ua/handle/document/30204 (дата звернення: 10.05.2026).
4. Коваль С. Аудіальний контент користувачів. *Теле- та радіожурналістика*. 2015. Вип. 14. С. 221–226. URL: https://publications.lnu.edu.ua/collections/index.php/teleradio/article/viewFile/1081/1070 (дата звернення: 10.05.2026).
5. Кирильчук М. М. Фонодокументи в цифровому інформаційному просторі. *Кваліфікаційна робота*. Донецький національний університет імені Василя Стуса, 2025. URL: https://jqmth.donnu.edu.ua/article/view/18124/18018 (дата звернення: 10.05.2026).
6. Печеранський І., Єременко Л. Аудіострімінг як тренд розвитку аудіовізуальних технологій та його вплив на сучасну музичну індустрію.

Вісник КНУКіМ. Серія «Мистецтвознавство». 2023. № 48. С. 26–32. DOI: 10.31866/2410-1176.48.2023.282435.

7. Скорін Ю. І., Негер Д. М. Вебзастосунок для стримінгового прослуховування музичного контенту. ІТ-простір сьогодення: тенденції, інновації та перспективи розвитку : збірник тез доповідей Всеукраїнської науково-практичної студентської конференції, 16 жовтня 2024 р. Харків : ХНУ імені В. Н. Каразіна, 2024. С. 193–196. URL: https://repository.hneu.edu.ua/handle/123456789/34182 (дата звернення: 10.05.2026).
8. Семернін М. Ю. Розробка платформи для онлайн прослуховування музики : пояснювальна записка до кваліфікаційної роботи здобувача вищої освіти на першому (бакалаврському) рівні, спеціальність 126 «Інформаційні системи та технології». Харків : Харківський національний університет радіоелектроніки, 2024. 71 с. URL: https://openarchive.nure.ua/entities/publication/345f2974-6858-4749-9adb-a227b4bc2572 (дата звернення: 10.05.2026).
9. Музикін А. М. Розробка цифрової платформи для спільного прослуховування та обговорення музичних творів. Радіоелектроніка та молодь у XXI столітті : матеріали 28-го Міжнар. молодіж. форуму, 16–18 квітня 2024 р. Харків : ХНУРЕ, 2024. Т. 6. С. 873–874. URL: https://openarchive.nure.ua/entities/publication/9c4c1ee2-2445-4261-86d0-37eee942b7d9 (дата звернення: 10.05.2026).
10. Spotify Support. What is Spotify? URL: https://support.spotify.com/us/article/what-is-spotify/(дата звернення: 10.05.2026).
11. SoundCloud. Stream and listen to music online for free with SoundCloud. URL: https://soundcloud.com/(дата звернення: 10.05.2026).
12. YouTube Music Help. Listen to personal or custom mix. URL: https://support.google.com/youtubemusic/answer/15165061(дата звернення: 10.05.2026).

13. Apple Support. Music - Official Apple Support. URL: https://support.apple.com/music(дата звернення: 10.05.2026).
14. Apple Support. Use Sync Library with your Apple Music subscription. URL: https://support.apple.com/en-us/118285(дата звернення: 10.05.2026).
15. Deezer. Listen to music online. Music streaming app. URL: https://www.deezer.com/en/(дата звернення: 10.05.2026).
16. Deezer. Deezer Flow: AI personalized playlists and music mixes. URL: https://www.deezer.com/explore/features/flow/ (дата звернення: 10.05.2026).
17. ISO/IEC 25010:2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. Geneva : International Organization for Standardization, 2011. URL: https://www.iso.org/standard/35733.html(дата звернення: 10.05.2026).
18. Nielsen J. 10 Usability Heuristics for User Interface Design. Nielsen Norman Group. 1994. URL: https://www.nngroup.com/articles/ten-usability-heuristics/(дата звернення: 10.05.2026).
19. Поплавський М. М., Трач Ю. В. Цифровізація музичної індустрії: тенденції і перспективи. Вісник Національної академії керівних кадрів культури і мистецтв. 2022. № 2. С. 30–39. URL: https://journals.uran.ua/visnyknakkkim/article/view/262202(дата звернення: 10.05.2026).
20. Булах Т. Б. Трансформація музичної індустрії: вплив соціальних медіа та цифрових платформ. Культурологічний альманах. 2024. № 2. URL: https://art-platforma.kmaecm.edu.ua/index.php/art1/article/view/171(дата звернення: 10.05.2026).
21. IFPI. Global recorded music revenues by segment 2024. URL: https://www.ifpi.org/our-industry/industry-data/(дата звернення: 10.05.2026).
22. Spotify for Artists. Royalties Guide. URL: https://artists.spotify.com/en/royalties-guide(дата звернення: 10.05.2026).

23. SoundCloud Help Center. Fan-powered Royalties. URL: https://help.soundcloud.com/hc/en-us/articles/1260801306810-Fan-powered-Royalties(дата звернення: 10.05.2026).
24. Bandcamp. About Bandcamp. URL: https://bandcamp.com/about(дата звернення: 10.05.2026).
25. Deezer Support. Artist-Centric Payment Model. URL: https://support.deezer.com/hc/en-gb/articles/360002471277-Artist-Centric-Payment-Model-ACPS(дата звернення: 10.05.2026).
26. OWASP Foundation. OWASP Top Ten Web Application Security Risks. URL: https://owasp.org/www-project-top-ten/(дата звернення: 10.05.2026).
27. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/(дата звернення: 10.05.2026).
28. Pallets Projects. Flask Documentation. URL: https://flask.palletsprojects.com/ (дата звернення: 10.05.2026).
29. Flask-CORS Documentation. URL: https://flask-cors.readthedocs.io/ (дата звернення: 10.05.2026).
30. Psycopg. PostgreSQL database adapter for Python. URL: https://www.psycopg.org/docs/ (дата звернення: 10.05.2026).
31. React. Create user interfaces from components. URL: https://react.dev/ (дата звернення: 10.05.2026).
32. Vite. Getting Started. URL: https://vite.dev/guide/ (дата звернення: 10.05.2026).
33. Pallets Projects. Flask Documentation. URL: https://flask.palletsprojects.com/ (дата звернення: 10.05.2026).
34. Spotify for Developers. Web Playback SDK. URL: https://developer.spotify.com/documentation/web-playback-sdk (дата звернення: 10.05.2026).
35. Tailwind CSS. Documentation. URL: https://tailwindcss.com/docs (дата звернення: 10.05.2026).
36. Radix UI. Primitives Documentation. URL: https://www.radix-ui.com/primitives/docs/overview/introduction (дата звернення: 10.05.2026).

37. Psycopg. PostgreSQL database adapter for Python. URL: https://www.psycopg.org/docs/ (дата звернення: 10.05.2026).

ДОДАТКИ

ДОДАТОК А

SQL-схема бази даних

```
CREATE TABLE IF NOT EXISTS users (
  id BIGSERIAL PRIMARY KEY,
  email VARCHAR(255) NOT NULL UNIQUE,
  display_name VARCHAR(120) NOT NULL,
  password_hash TEXT,
  google_sub VARCHAR(255) UNIQUE,
  auth_provider VARCHAR(40) NOT NULL DEFAULT 'local',
  avatar_url TEXT,
  created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

ALTER TABLE users ADD COLUMN IF NOT EXISTS password_hash TEXT;
ALTER TABLE users ADD COLUMN IF NOT EXISTS google_sub
VARCHAR(255) UNIQUE;
ALTER TABLE users ADD COLUMN IF NOT EXISTS auth_provider
VARCHAR(40) NOT NULL DEFAULT 'local';

CREATE TABLE IF NOT EXISTS artists (
  id BIGSERIAL PRIMARY KEY,
  name VARCHAR(180) NOT NULL UNIQUE,
  bio TEXT,
  image_url TEXT,
  followers_count INTEGER NOT NULL DEFAULT 0 CHECK (followers_count >=
0),
  created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

CREATE TABLE IF NOT EXISTS genres (
```

```
  id BIGSERIAL PRIMARY KEY,
  name VARCHAR(80) NOT NULL UNIQUE
);

CREATE TABLE IF NOT EXISTS albums (
  id BIGSERIAL PRIMARY KEY,
  title VARCHAR(220) NOT NULL,
  primary_artist_id BIGINT NOT NULL REFERENCES artists(id) ON DELETE
RESTRICT,
  release_year INTEGER CHECK (release_year IS NULL OR release_year >= 1900),
  cover_url TEXT,
  created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  UNIQUE (title, primary_artist_id, release_year)
);

CREATE TABLE IF NOT EXISTS tracks (
  id BIGSERIAL PRIMARY KEY,
  title VARCHAR(220) NOT NULL,
  album_id BIGINT REFERENCES albums(id) ON DELETE SET NULL,
  genre_id BIGINT REFERENCES genres(id) ON DELETE SET NULL,
  duration_seconds INTEGER NOT NULL CHECK (duration_seconds > 0),
  audio_url TEXT,
  cover_url TEXT,
  source_name VARCHAR(80),
  source_url TEXT,
  plays_count INTEGER NOT NULL DEFAULT 0 CHECK (plays_count >= 0),
  is_trending BOOLEAN NOT NULL DEFAULT FALSE,
  created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
```

```
CREATE TABLE IF NOT EXISTS track_artists (
  track_id BIGINT NOT NULL REFERENCES tracks(id) ON DELETE CASCADE,
  artist_id BIGINT NOT NULL REFERENCES artists(id) ON DELETE RESTRICT,
  artist_order INTEGER NOT NULL DEFAULT 1 CHECK (artist_order > 0),
  PRIMARY KEY (track_id, artist_id)
);

CREATE TABLE IF NOT EXISTS playlists (
  id BIGSERIAL PRIMARY KEY,
  user_id BIGINT REFERENCES users(id) ON DELETE SET NULL,
  name VARCHAR(160) NOT NULL,
  description TEXT,
  cover_url TEXT,
  is_public BOOLEAN NOT NULL DEFAULT TRUE,
  created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

CREATE TABLE IF NOT EXISTS playlist_tracks (
  playlist_id BIGINT NOT NULL REFERENCES playlists(id) ON DELETE
CASCADE,
  track_id BIGINT NOT NULL REFERENCES tracks(id) ON DELETE CASCADE,
  position INTEGER NOT NULL CHECK (position > 0),
  added_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  PRIMARY KEY (playlist_id, track_id),
  UNIQUE (playlist_id, position)
);

CREATE TABLE IF NOT EXISTS liked_tracks (
  user_id BIGINT NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  track_id BIGINT NOT NULL REFERENCES tracks(id) ON DELETE CASCADE,
```

```
  created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  PRIMARY KEY (user_id, track_id)
);

CREATE TABLE IF NOT EXISTS recent_plays (
  id BIGSERIAL PRIMARY KEY,
  user_id BIGINT REFERENCES users(id) ON DELETE CASCADE,
  track_id BIGINT NOT NULL REFERENCES tracks(id) ON DELETE CASCADE,
  played_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

CREATE INDEX IF NOT EXISTS idx_tracks_plays_count ON tracks (plays_count
DESC);
CREATE INDEX IF NOT EXISTS idx_tracks_genre_id ON tracks (genre_id);
CREATE INDEX IF NOT EXISTS idx_track_artists_artist_id ON track_artists
(artist_id);
CREATE INDEX IF NOT EXISTS idx_recent_plays_user_played_at ON
recent_plays (user_id, played_at DESC);
```

ДОДАТОК Б

Запуск проєкту

Python backend

У першому терміналі:

```bash
python -m venv .venv
.\.venv\Scripts\Activate.ps1
pip install -r requirements.txt
python app.py
```

Backend запускається за адресою:

```text
http://127.0.0.1:5000/
```

Під час запуску backend:

- підключається до PostgreSQL
- створює таблиці, якщо їх ще немає
- додає початкові дані
- запускає API `/db-api`

Frontend

У другому терміналі:

```bash
npm.cmd install
npm.cmd run dev
```

Сайт відкривається за адресою:

```text
http://127.0.0.1:5173/
```

Frontend звертається до backend через:

```text
/db-api
```