

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Вебплатформа для прослуховування музики»

Виконав студент групи ЗПЗ-22-1
ХАЄЦЬКИЙ Андрій Віталійович

Керівник роботи:
ДУНЄВ Сергій Сергійович

Рецензент:
ЛОБАНЧИКОВА Надія Миколаївна

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ХАЄЦЬКИЙ Андрій Віталійович**

Керівник роботи: : **ДУНЄВ Сергій Сергійович**

Тема роботи: **«Вебплатформа для прослуховування музики»,**
затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №93/с**
Вихідні дані для роботи: **об'єктом дослідження є інформаційні технології та програмні засоби для створення та забезпечення стабільного функціонування вебплатформи прослуховування музики, а предметом дослідження є сучасні методи, підходи та технологічні рішення, які застосовуються під час розробки вебзастосунків для роботи з мультимедійним контентом.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	ДУНЄВ С.С.	02.03.2026 р.	02.03.2026 р.
2	ДУНЄВ С.С.	02.03.2026 р.	02.03.2026 р.
3	ДУНЄВ С.С.	02.03.2026 р.	02.03.2026 р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з вебплатформи для прослуховування музики та пояснювальної записки. Пояснювальна записка до випускної роботи містить текстову частину, викладену на 70 сторінках друкованого тексту. Список використаних джерел містить 30 найменувань і займає 3 сторінки. У роботі представлено 38 рисунків та 12 таблиць. Додатки займають 7 сторінок. Загальний обсяг пояснювальної записки становить 77 сторінки.

Метою роботи є створення зручної, адаптивної та функціональної вебплатформи для прослуховування музики, яка забезпечуватиме користувачам швидкий доступ до аудіо контенту, інтуїтивно зрозумілий інтерфейс та взаємодію з сервісом на різних типах пристроїв. Під час розробки вебплатформи використовувалися сучасні вебтехнології орієнтовані для створення зручного інтерфейсу, забезпечення високої продуктивності, безпеки, обробки та передачі даних, гнучкості для подальшого масштабування функціоналу платформи.

У ході реалізації вебплатформи було використано сучасний стек веб-технологій. Для розробки клієнтської частини використано фреймворк Next.js для побудови динамічного інтерфейсу користувача з акцентом на оптимізацію та продуктивність, для серверної частини використано Nest.js для побудови масштабованої та структурованої архітектури серверного застосунку з використанням REST API. Для зберігання текстової інформації використано систему керування базами даних PostgreSQL, яка надає надійне сховище даних, обробку та керування даними.

Перевірка вебплатформи відбувалася через модульне тестування, додатково оцінювалось швидкість та зручність використання вебзастосунку.

КЛЮЧОВІ СЛОВА: ВЕБПЛАТФОРМА, КЛІЄНТ-СЕРВЕР, ПРОСЛУХОВУВАННЯ, POSTGRESQL, NESTJS, NEXTJS.

					ІПЗ.КР.Б-121-26-ПЗ			
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дат.</i>	Вебплатформа для прослуховування музики. Пояснювальна записка	<i>Літ.</i>	<i>Арк.</i>	<i>Аркуців</i>
<i>Розроб.</i>		Хасцький А.В.						
<i>Керівник.</i>		Дунев С.С.					3	77
<i>Н. контр.</i>		Тичина О.П.				Житомирська політехніка, група ЗІПЗ-22-1		
<i>Зав. каф.</i>		Вакалюк Т.А.						

ABSTRACT

The bachelor's thesis consists of a web platform for listening to music and an explanatory note. The explanatory note accompanying the thesis comprises a text section spanning 70 pages of printed text. The list of references contains 30 entries and spans 3 pages. The thesis includes 38 figures and 12 tables. The appendices span 7 pages. The total length of the explanatory note is 77 pages.

The goal of this project is to create a user-friendly, responsive, and functional web platform for listening to music that will provide users with quick access to audio content, an intuitive interface, and the ability to interact with the service across various types of devices. During the development of the web platform, modern web technologies are used to create a user-friendly interface, ensure high performance, security, data processing and transmission, and flexibility for future scaling of the platform's functionality.

A modern stack of web technologies was used during the development of the web platform. For the client-side development, the Next.js framework was used to build a dynamic user interface with a focus on optimization and performance; for the server-side, Nest.js was used to build a scalable and structured server application architecture using REST API. The PostgreSQL database management system was used to store text information, providing reliable data storage, processing, and management.

The web platform was tested using modular testing, and the speed and usability of the web application were also evaluated.

WEB PLATFORM, CLIENT-SERVER, LISTENING, POSTGRESQL, NESTJS, NEXTJS.

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1 АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ВЕБПЛАТФОРМИ	9
1.1 Постановка задачі.....	9
1.2 Аналіз аналогів програмного продукту	11
1.3 Вибір архітектури для веборієнтованої системи	16
1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	19
Висновки до першого розділу.....	22
РОЗДІЛ 2 ПРОЕКТУВАННЯ ВЕБПЛАТФОРМИ.....	23
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури	23
2.2 Розробка бази даних системи.....	30
2.3 Проектування та реалізація алгоритмів роботи системи	36
2.4 Реалізація вебплатформи для прослуховування музики.....	39
Висновки до другого розділу	45
РОЗДІЛ 3 ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБПЛАТФОРМОЮ...	46
3.1 Порядок встановлення та налаштування параметрів вебплатформи	46
3.2 Структура інтерфейсу вебплатформи	48
3.3 Тестування вебплатформи.....	62
Висновки до третього розділу	65
ВИСНОВКИ	66
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	67
ДОДАТКИ.....	70

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс (англ. Application Programming Interface)

CRUD – основні функції для управління даними (англ. Create Read Update Delete)

HLS – мережевий протокол для потокової передачі відео та аудіо через інтернет (англ. HTTP Live Streaming)

REST – передача репрезентативного стану через які взаємодіють веб-застосунки (англ. Representational State Transfer)

SPA – односторінковий застосунок (англ. Single Page Application)

БД – база даних

ПЗ – програмне забезпечення

СКБД – система керування базами даних

ВСТУП

Актуальність теми пояснюється розвитком мультимедійних сервісів та постійним зростанням споживання музичного контенту в мережі Інтернет. У сучасних умовах розвитку інформаційних технологій вебплатформи для прослуховування музики є величезною складовою цифрового середовища, які надають зручний доступ для прослуховування аудіо контенту, персоналізацію на основі прослуховування користувача та можливість використання на різних типах пристроїв.

Метою випускної кваліфікаційної роботи бакалавра є створення вебплатформи для прослуховування музики. Розроблений вебзастосунок повинен забезпечувати стабільне та безперервне відтворення аудіоконтенту, бути адаптивним для використання на різних типах пристроїв, підтримувати масштабованість та відповідати вимогам щодо продуктивності, надійності й зручності користування.

Сформульована мета обумовлює необхідність виконання наступних **завдань**:

1. Дослідження різних підходів до розробки вебплатформ для прослуховування музики.

2. Аналіз існуючих музичних платформ з метою визначення їх функціональних можливостей, переваг та недоліків для подальшого формування вимог власного програмного продукту.

3. Обґрунтування вибору сучасних технологій та інструментів для розробки вебзастосунку.

4. Моделювання структури бази даних для зберігання, обробки та керування мультимедійним контентом та даними користувача.

5. Проектування механізмів взаємодії користувача із системою, як-от аутентифікація, функціонал пошуку, відтворення музики, створення плейлистів та інше.

6. Обґрунтування та вибір архітектури програмного забезпечення із застосуванням сучасних принципів проєктування та об'єктно-орієнтованого підходу.

7. Створення адаптивного користувацького інтерфейсу для забезпечення коректної роботи на різних типах пристроїв.

8. Проведення тестування програмного забезпечення для перевірки стабільності роботи, продуктивності та коректності функціонування основних компонентів системи.

Об'єктом дослідження є інформаційні технології та програмні засоби для створення та забезпечення стабільного функціонування вебплатформи прослуховування музики.

Предметом дослідження є сучасні методи, підходи та технологічні рішення, які застосовуються під час розробки веб-застосунків для роботи з мультимедійним контентом.

РОЗДІЛ 1 АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ВЕБПЛАТФОРМИ

1.1 Постановка задачі

Сучасний розвиток цифрових технологій активно змінює сферу розваг та музики. Вебплатформи для прослуховування музики дозволяють користувачам швидко отримувати доступ до великої кількості бібліотеки пісень та альбомів різних жанрів і виконавців. Зазвичай вони включають для кожного користувача персоналізацію на основі їх прослуховувань, мають зручний пошук пісень.

Переважно такі платформи є посередниками між музичними виконавцями, лейблами та слухачами. Вони об'єднують контент у єдину бібліотеку, надаючи можливість користувачам легко знаходити нові пісні, створювати власні плейлисти та ділитися ними з іншими. Система рекомендацій допомагає підбирати музику, яка відповідає вподобанням користувача.

Окрім основного функціоналу, вебплатформи пропонують додаткові можливості, як от: оцінки та відгуки на пісні, збереження треків для офлайн-прослуховування, інструменти для моніторингу популярності контенту і т.д. Кожен такий функціонал надає користувачам зручність та швидкість у використанні.

Метою розробки є створення вебплатформи для прослуховування музики, яка надаватиме зручний доступ до пісень, створення власних плейлистів та пошуку.

Для досягнення поставленої мети потрібно виконати наступні завдання.

Перш за все потрібно розробити структурну архітектуру платформи, обґрунтувати вибір технологій, які гарантуватимуть стабільну роботу та захищеність даних користувачів; визначити фреймворки та бібліотеки для швидкої та ефективного розробки.

Далі розумним рішенням є проєктування та реалізація структури бази даних, де потрібно створити таблиці для зберігання інформації про користувачів, пісні, альбоми, виконавців, плейлисти і т.д. Для цього можна використати СУБД PostgreSQL, яка гарантує збереження цілісності даних за допомогою зв'язків між

таблицями та дозволяє виконувати складні запити для ефективного доступу до інформації.

Наступним етапом потрібно реалізувати серверну частину вебплатформи, яка відповідатиме за обробку запитів від користувачів, взаємодію з базою даних, авторизацію та автентифікацію, логіку збереження музики та зображень, управління плейлистами. Серверна частина повинна стабільно працювати навіть при неочікуваних помилках, відповідати вимогам безпеки та ефективно взаємодіяти з клієнтським інтерфейсом.

Після реалізації серверної частини логічним етапом є створення зручного та зрозумілого у використанні клієнтського інтерфейсу з яким буде взаємодіяти користувач. Потрібно реалізувати основні функції: реєстрація та вхід; пошук треків, альбомів, виконавців; перегляд інформації про пісні, альбоми, виконавців, жанри; CRUD плейлистів, додавання треків у улюблене та ін. Клієнтська частина повинна бути адаптивною на різних пристроях, підтримуватися різними браузерами та відповідати сучасним вимогам дизайну.

Останнім кроком є проведення тестування функціональності та стійкості розробленої вебплатформи. Найкращим варіантом є модульне та функціональне тестування. Модульне тестування сприятиме виявленню помилок у програмному коді з метою їх виправлення, а функціональне для перевірки правильності роботи платформи щодо визначеного функціоналу. Додатково, потрібно виконати перевірки стабільності роботи системи та відповідність її очікуваній поведінці при різних сценаріях.

Після дотримання та виконання усіх етапів буде створено функціональну вебплатформу для прослуховування музики, яка надаватиме користувачам зручний доступ до пісень, альбомів та виконавців, можливість створювати власні плейлисти, додавати улюблені пісні та інше. Платформа дозволить швидко знаходити музику за альбомами або виконавцями, взаємодіяти з системою через зрозумілий та адаптивний інтерфейс. Вона сприятиме покращенню досвіду користувачів та задовольнятиме потреби як слухачів, так і адміністраторів самої платформи.

1.2 Аналіз аналогів програмного продукту

Аналіз аналогів програмного продукту проводиться для оцінки існуючих рішень на ринку та визначення сильних і слабких сторін конкурентів. Дане дослідження дозволяє зрозуміти, які функції користувачі вважають корисними, а які потребують покращення. Після цього аналізу потрібно обрати оптимальні підходи до реалізації платформи та сформулювати вимоги щодо функціоналу.

Для аналізу було обрано наступні найпопулярніші платформи для прослуховування музики: Spotify, YouTube Music, Apple Music та Deezer.

Spotify [1] є відомою та популярною платформою для прослуховування музики, яка надає доступ до мільйонів треків та подкастів. Spotify було створено у 2006 році у Швеції, а офіційно запущено у 2008 році. Платформа надає користувачам можливість знаходити улюблені пісні, слухати альбоми та формувати власні плейлисти.

Spotify має систему персоналізованих рекомендацій, яка підбирає треки на основі смаків користувача та історії прослуховувань. Платформа дозволяє користувачам ділитися музикою з друзями, підписуватися на плейлисти інших та зберігати пісні для офлайн-прослуховування.

Spotify надає зручний інтерфейс для користування на комп'ютерах, смартфонах та веб-браузері. Користувачі можуть легко шукати музику за жанрами, виконавцями або альбомами. Додатково, має відтворення подкастів та інтеграцію з іншими додатками, що розширює можливості платформи.

Серед переваг Spotify виділяють велику бібліотеку пісень, ефективну систему рекомендацій, зручний інтерфейс та доступність на різних пристроях. До недоліків належить обмеження безкоштовної версії, яка включає рекламу та не дозволяє пропускати деякі треки, а також залежність від інтернет-з'єднання для онлайн-прослуховування.

Інтерфейс платформи Spotify наведено на рис. 1.1.

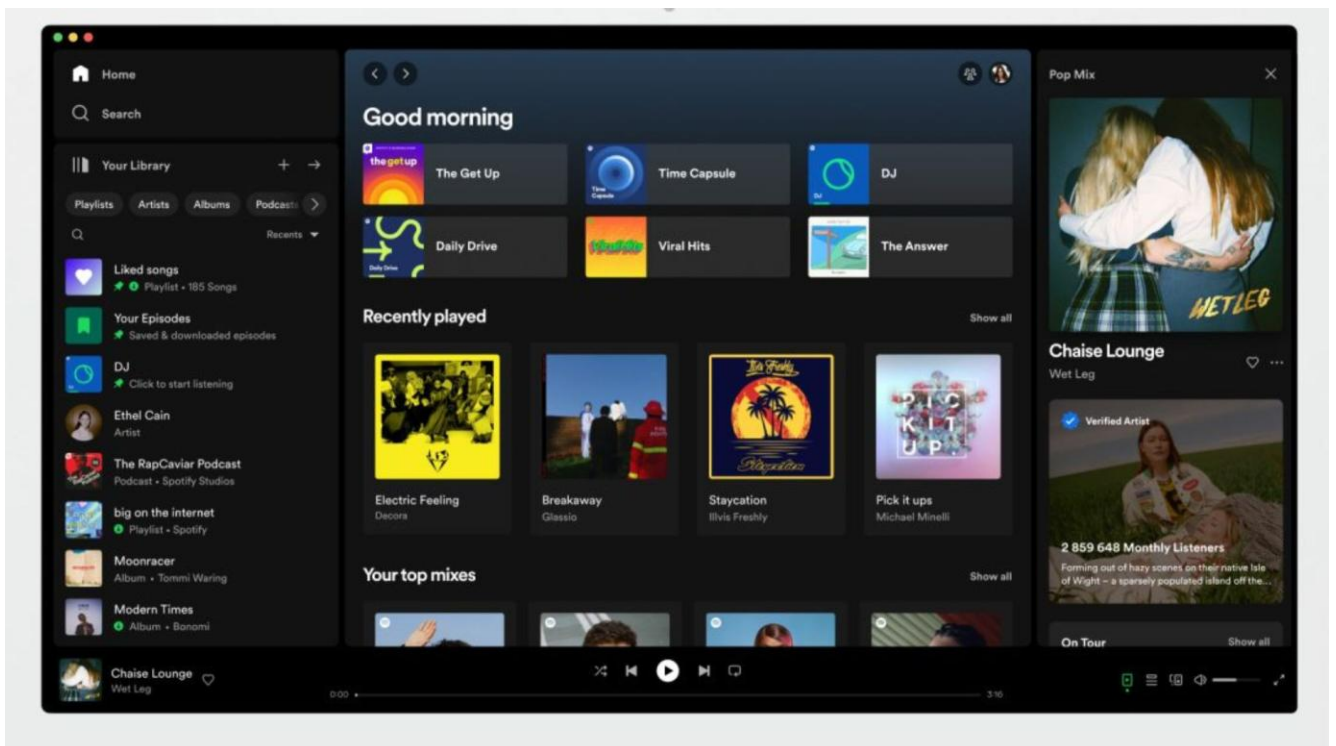


Рис. 1.1. Інтерфейс Spotify

YouTube Music [2] – платформа для прослуховування музики та перегляду музичних відео, яка була запущена у 2015 році компанією Google. Вона пропонує користувачам функціонал прослуховування пісень, альбомів та створення власних плейлистів. Крім музики, платформа надає доступ до офіційних відеокліпів та концертних записів.

Платформа має персоналізовані рекомендації, підбираючи музику на основі історії прослуховувань та вподобань користувача. Доступні функції пошуку за жанрами, виконавцями та плейлистами інших користувачів. Платформа також інтегрується з іншими продуктами Google для зручності у використанні за її межами.

Інтерфейс YouTube Music реалізовано для комп'ютерів, мобільних пристроїв та веб-браузера. Користувачі можуть легко переходити між аудіо та відео, зберігати треки для офлайн-прослуховування та налаштовувати власні плейлисти. Додатково реалізовано функціонал автоматичного відтворення та рекомендації нових пісень.

Серед переваг платформи є пряма інтеграція з YouTube, широкий каталог музичних відео, персоналізовані рекомендації та зручність використання на різних пристроях. До недоліків належать обмеження безкоштовної версії, наявність

реклами та потреба у стабільному інтернет-з'єднанні для перегляду відео та потокового відтворення музики.

Інтерфейс платформи YouTube Music представлений на рис. 1.2.

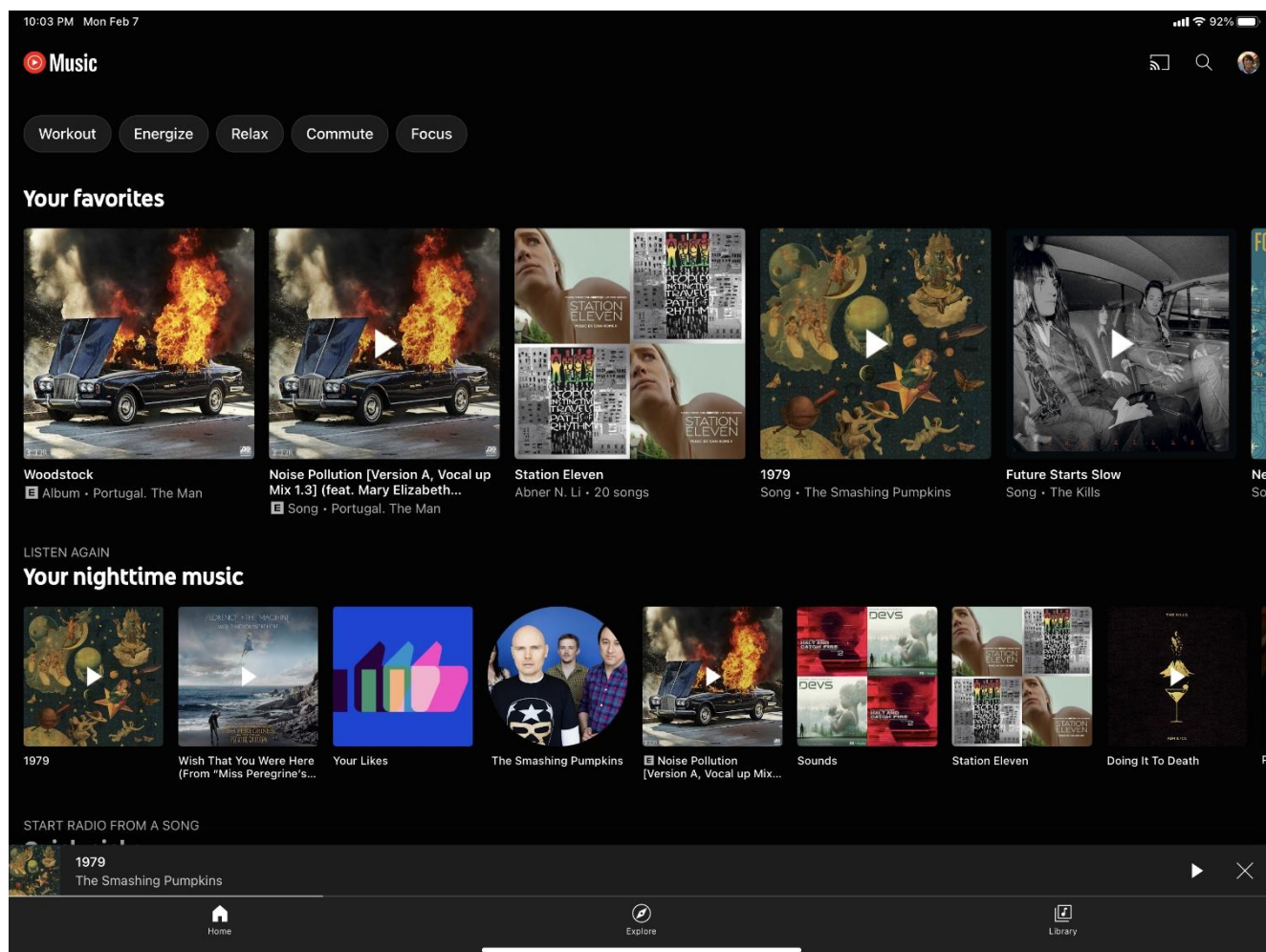


Рис. 1.2. Інтерфейс YouTube Music

Apple Music [3] є платформою для прослуховування музики, яка була запущена компанією Apple у 2015 році. Вона надає доступ до мільйонів треків, альбомів та плейлистів різних жанрів. Користувачі можуть створювати власні плейлисти та додавати улюблені пісні для швидкого доступу.

Платформа також має персоналізовані рекомендації на основі смаків користувача та історії прослуховувань. Користувачам доступні функції пошуку за жанрами, виконавцями та альбомами. Apple Music дозволяє слухати нові релізи, офіційні альбоми та подкасти, а також слідкувати за улюбленими артистами.

Інтерфейс платформи підтримується на комп'ютерах, мобільних пристроях та веб-браузерах. Користувачі можуть зберігати треки для офлайн-

прослуховування, перемикається між плейлистами та користуватися інтеграцією з іншими сервісами Apple. Додатково надається доступ до ексклюзивного контенту та радіостанцій.

Серед переваг Apple Music виділяють велику бібліотеку музики, інтеграцію з екосистемою Apple, персоналізовані рекомендації та можливість офлайн-прослуховування. До недоліків можна віднести обмеження безкоштовної версії, відсутність повної кросплатформної підтримки на деяких пристроях та необхідність платної підписки для повного доступу до всіх функцій.

Інтерфейс Apple Music наведено на рис. 1.3.

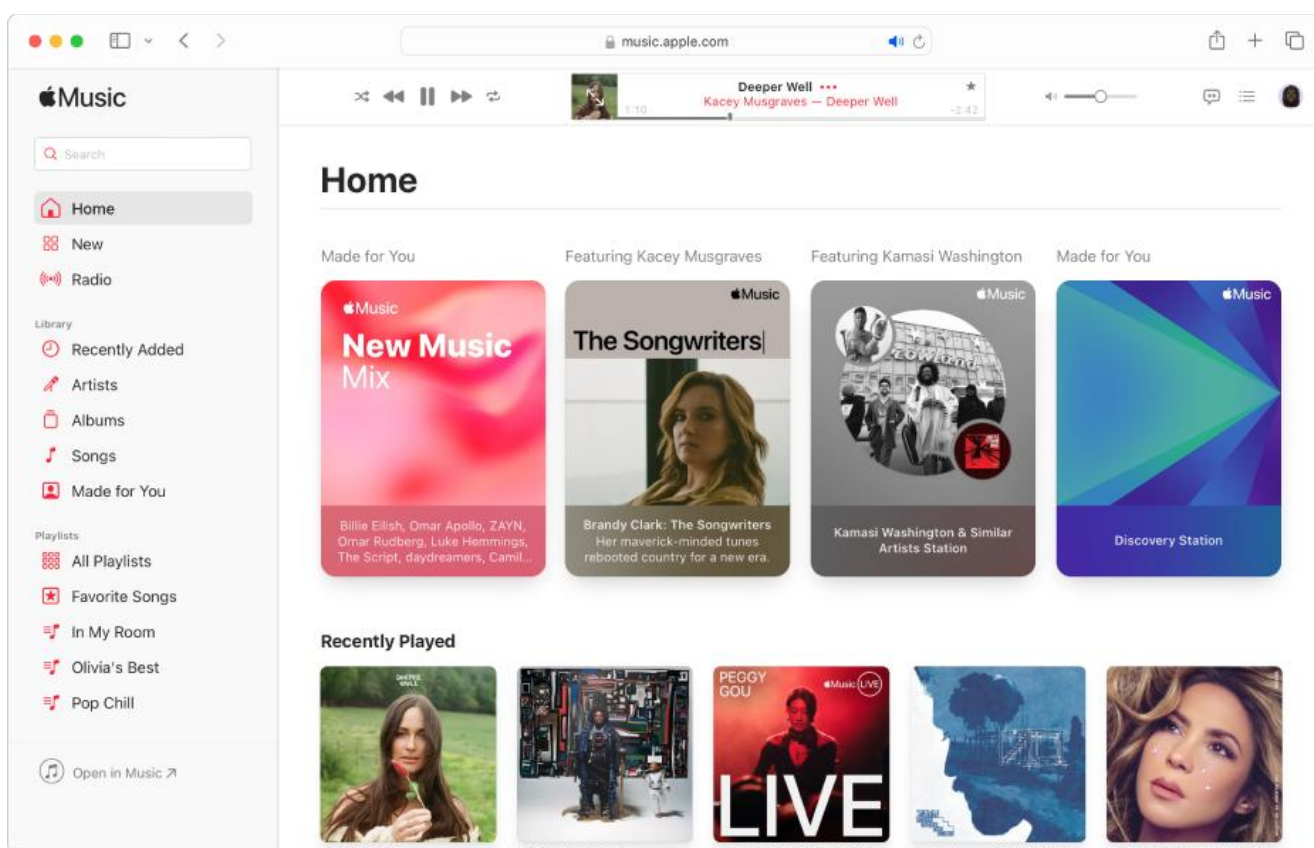


Рис. 1.3. Інтерфейс Apple Music

Deezer [4] – платформа для прослуховування музики, яка була створена у 2007 році у Франції. Платформа надає доступ до мільйонів треків, альбомів та плейлистів різних жанрів. Користувачі можуть формувати власні плейлисти, додавати улюблені пісні та відкривати нову музику за допомогою рекомендацій.

Платформа має власну систему персоналізованих рекомендацій на основі смаків користувача та історії прослуховувань. Користувачам доступний пошук

музики за жанрами, виконавцями та популярними плейлистами. Deezer також надає функції для прослуховування подкастів та радіостанцій.

Інтерфейс Deezer підтримується на різних типах пристроїв. Користувачі можуть зберігати пісні для офлайн-прослуховування, переглядати детальну інформацію про альбоми та виконавців, а також ділитися плейлистами з іншими користувачами.

Перевагою Deezer є широка бібліотека музики, персоналізовані рекомендації, зручний інтерфейс та підтримка різних пристроїв. До недоліків можна віднести: наявність реклами у безкоштовній версії, обмежені можливості прослуховування без підписки та залежність від інтернет-з'єднання для онлайн-прослуховування.

Інтерфейс платформи Deezer представлений на рис. 1.4.

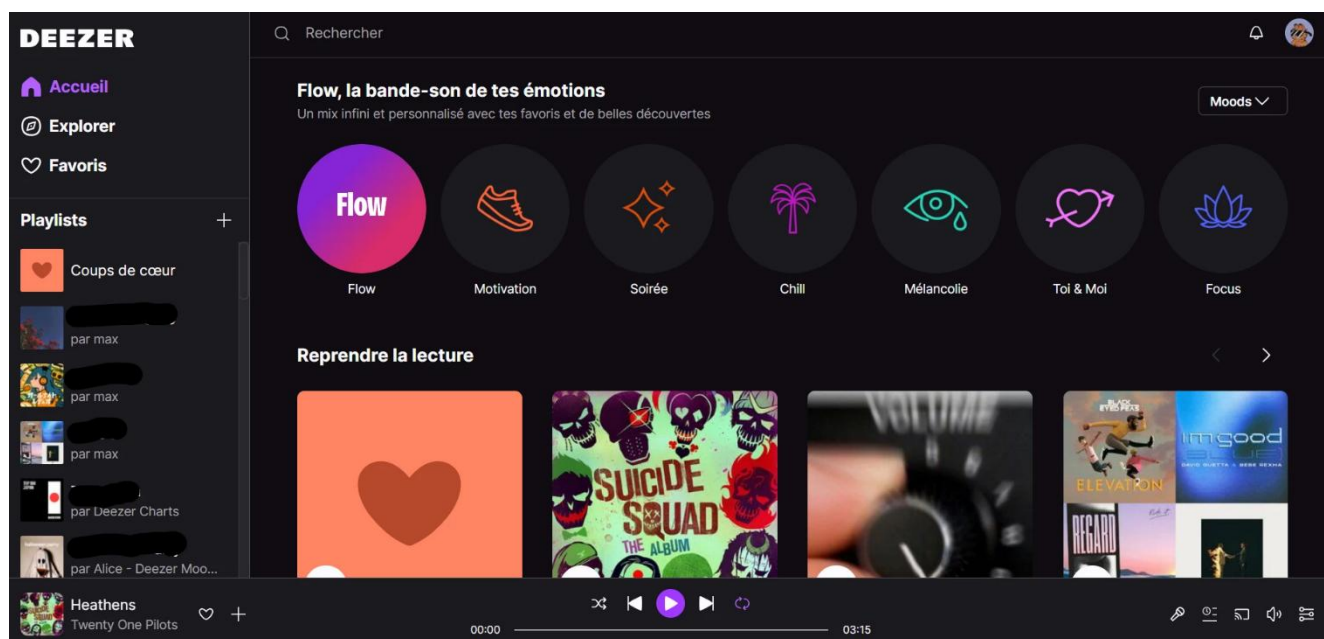


Рис. 1.4. Інтерфейс Deezer

Для наочності порівняльного аналізу аналогів програмного продукту результати наведено в табл. 1.1.

Таблиця 1.1

Порівняння аналогів платформ

Назва платформи	Зручність інтерфейсу	Персоналізовані рекомендації	Доступність на пристроях	Наявність реклами/обмежень
Spotify	+	+	+	+/-
YouTube Music	+/-	+	+	-

Apple Music	+	+	+/-	-
Deezer	+/-	+/-	+	-

Після проведеного аналізу та дослідження функціональних можливостей існуючих платформ для прослуховування музики можна сформулювати базові функції до створюваної платформи, як-от: можливість створювати пісні, альбоми, виконавців, жанри; створювати плейлисти; зберігати улюблені пісні; формувати історію прослуховувань; пошуку і фільтрації музики за жанрами, альбомами та виконавцями; редагування профілю; отримати та змінити підписку на платформі.

1.3 Вибір архітектури для веборієнтованої системи

Вибір архітектури для веборієнтованої системи є важливим етапом життєвого циклу розробки програмного продукту. Обравши правильну архітектуру можна створити стабільну робочу платформу, яка надаватиме зручну взаємодію користувачів з системою та ефективно управління даними.

Перед вибором конкретного архітектурного рішення необхідно розглянути найбільш поширені підходи, які використовуються у сучасній веброботі.

Мікросервісна архітектура [5] є архітектурою, яка передбачає розподіл вебсистеми на окремі невеликі сервіси, кожен з яких виконує конкретну функцію. Кожен сервіс працює незалежно та взаємодіє з іншими через чітко визначені інтерфейси. Використовуючи цю архітектуру можна спростити розробку, тестування та підтримку системи. На відміну від інших архітектур, мікросервісна дозволяє масштабувати окремі частини платформи без потреби змінювати всю систему.

До переваг архітектури виділяють гнучкість у розвитку функцій, можливість використання різних технологій для різних сервісів та простішу локалізацію помилок. Кожен сервіс можна оновлювати окремо, завдяки чому швидше впроваджувати нові функції. Надійність системи є вищою, оскільки збій одного сервісу не зупиняє роботу всієї платформи.

Недоліки архітектури складаються з складності налаштування взаємодії між сервісами та управління даними, які розподілені між ними. Потрібно забезпечувати стабільну комунікацію та синхронізацію, що може вимагати додаткових ресурсів. Також є ризик підвищення вимоги до моніторингу та тестування, оскільки зростає кількість окремих компонентів системи.

Клієнт-серверна архітектура [6] є архітектурою, яка розділяє систему на два основні рівні: клієнтську частину та серверну. Клієнт надсилає запити до сервера, а сервер обробляє їх і повертає необхідні дані. Така з'єднання дозволяє централізовано зберігати та обробляти інформацію, тим самим зменшити навантаження на пристрої користувачів та забезпечити узгодженість та правильність даних.

Перевагою клієнт-серверної архітектури є її простота реалізації та зрозумілість організації взаємодії між компонентами, де дані зберігаються на сервері, що підвищує їхню безпеку та полегшує резервне копіювання. Дані також можна безпечно резервувати так як вони зберігаються на сервері. Архітектура має централізовану обробку для ефективного керування ресурсами та контролю доступу користувачів до системи.

Недоліком архітектури є залежність від роботи сервера та його ресурсів. Якщо сервер виходить з ладу, клієнти не можуть отримати доступ до даних або користуватися функціями системи. На сервер зростає навантаження при великій кількості користувачів, що вимагає додаткових ресурсів та оптимізації продуктивності.

Монолітна архітектура [7] – архітектура, де створена вебсистема є єдиним великим застосунком, у якому всі функції об'єднані в одному кодовому середовищі. Серверна та клієнтська логіка, обробка даних і взаємодія з базою даних реалізовані всередині одного додатку. Архітектура спрощує початкову розробку та дозволяє швидко запускати платформу на відміну від інших архітектур.

Серед переваг монолітної архітектури є простота розгортання та налагодження, одна система виходу і централізоване управління кодом. Вона зручна для невеликих або середніх проєктів, де не потрібна масштабована

інфраструктура. При використанні архітектури не виникає складнощів з комунікацією між окремими компонентами системи, так як всі вони працюють в межах одного застосунку.

Недоліком архітектури є складність масштабування та оновлення окремих функцій. З розширенням системи збільшується ризик появи помилок при внесенні змін, зростає навантаження на сервер. Також, великі монолітні архітектури важко адаптувати до нових технологій та розподілених середовищ.

Багаторівнева архітектура [8] є архітектурою, де вебсистема розділена на окремі рівні, які виконують різні функції. Зазвичай виділяють три основні рівні: презентаційний (клієнтський інтерфейс), логіки застосунку (серверна частина) та доступу до даних (база даних). Завдяки різним рівням системи, розділяються обов'язки між компонентами та підвищується гнучкість системи.

Серед переваг багаторівневої архітектури є покращена масштабованість, порівняно з іншими архітектурами, можливість окремого оновлення рівнів та спрощене управління безпекою. Кожен рівень може працювати на окремих серверах або бути реалізований окремим сервісом, щоби підвищити надійність та стійкість системи. Архітектура спрощує інтеграцію з іншими сервісами та дозволяє розширити функціонал.

Недоліками архітектури є проектування та налаштування взаємодії між рівнями. Для забезпечення стабільної роботи потрібна додаткова координація компонентів і моніторинг продуктивності. Зростає обсяг адміністративної роботи та необхідність оптимізації для швидкого обміну даними між рівнями.

Остаточний вибір архітектурного підходу здійснено на основі аналізу вимог до розроблюваної платформи, серед яких були визначені масштабованість, гнучкість, зручність підтримки та ефективна взаємодія компонентів системи. Для організації взаємодії між клієнтською та серверною частинами обрано клієнт-серверну архітектуру, оскільки вона забезпечує централізовану обробку запитів та узгоджене зберігання даних. Для побудови серверної частини з можливістю масштабування та гнучкого розвитку функціоналу доцільно використати багаторівневу архітектуру для розділення системи на три логічні рівні.

1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Враховуючи функціонал майбутньої платформи та обраних архітектурних моделей потрібно обрати оптимальний набір технологій для реалізації клієнтської та серверної частин. Оскільки розробка орієнтована на створення вебзастосунку, для її реалізації обрано сучасні програмні засоби та інструменти.

Для реалізації серверної та клієнтської частин платформи було використано середовище виконання Node.js [9].

Node.js – середовище виконання JavaScript на сервері, яке використовується для обробки запитів користувачів та управління даними. Воно забезпечує стабільну роботу серверної частини вебзастосунків, дозволяє створювати API, керувати базою даних та інтегруватися з іншими сервісами. Node.js застосовується для побудови швидких і масштабованих платформ з однією мовою програмування.

Серверна та клієнтська частини застосунку були реалізовані з використанням мови програмування TypeScript [10].

TypeScript – мова програмування, яка розширює JavaScript і додає статичну типізацію. Вона використовується для створення більш надійного та зрозумілого коду у вебзастосунках. TypeScript дозволяє уникати помилок під час розробки, спрощує підтримку великих проєктів та надає зручну інтеграцію з сучасними фреймворками.

Для розробки серверної частини вебзастосунку було використано фреймворк Nest.js [11], який підтримує багаторівневу архітектуру.

Nest.js є JavaScript-фреймворком, який використовується для побудови серверних вебзастосунків. Структура проєкту є організованою завдяки розділенню логіки на модулі (рівні), що спрощує створення API. Nest.js надає можливість ефективно обробляти запити користувачів, працювати з базами даних та інтегрувати різні сервіси для стабільної та масштабованої роботи серверної частини.

Для зберігання даних вебплатформи було використано систему керування базами даних PostgreSQL [12].

PostgreSQL – система керування реляційними базами даних, яка використовується для збереження та обробки інформації у вебзастосунках. Вона надає безпечне сховище для зберігання даних та підтримку складних запитів.

Для реалізації функціоналу оплати підписок на платформі було використано платіжну систему Stripe [13].

Stripe є сервісом для обробки онлайн-платежів, який використовується для прийому та управління оплатою у вебзастосунках. Він надає безпечну взаємодію з банківськими картками та іншими методами оплати. Stripe має функціонал здійснення транзакції та відстеження історії платежів, що є зручним для відображення даної інформації користувачеві.

Для потокової передачі аудіо з базовим рівнем захисту контенту було використано технологію HLS [14] із підтримкою шифрування медіа сегментів.

Протокол HLS використовується для трансляції аудіо та відео через інтернет у режимі потокового відтворення. Він передає контент на різні пристрої з адаптивною якістю залежно від швидкості з'єднання. HLS застосовується для забезпечення стабільного та безперервного відтворення музики або відео у вебзастосунках з спрощенням інтеграції мультимедійного контенту з клієнтським інтерфейсом.

Для розробки клієнтської частини застосунку було обрано фреймворк Next.js [15] з підтримкою React [16] та TypeScript.

Next.js – фреймворк для React, який використовується для створення вебзастосунків з серверним рендерингом. Він використовує швидке завантаження сторінок з покращенням роботи SEO та дозволяє організувати маршрутизацію у додатку. Next.js застосовується для побудови клієнтської частини платформи, інтеграції з серверними API та забезпечення стабільної та ефективної взаємодії користувача з вебзастосунком.

Для стилізації користувацького інтерфейсу було використано CSS-фреймворк Tailwind CSS [17].

Tailwind CSS є фреймворком для стилізації вебзастосунків, який використовується для створення адаптивного та сучасного інтерфейсу без написання великої кількості власного CSS-коду. Він надає швидке оформлення компонентів, уніфіковане використання стилів та легке налаштування дизайну.

Для реалізації анімацій у користувацькому інтерфейсі було використано бібліотеку Motion for React (Framer Motion) [18].

Бібліотека Motion використовується для створення анімацій та плавних переходів у вебзастосунках. Вона надає зручне управління рухом компонентів, інтеграцію з існуючими елементами інтерфейсу та підвищує привабливість користувацького досвіду.

Мінімальні системні вимоги для роботи розробленої вебплатформи є такі: 6200 МБ оперативної пам'яті; 400 МБ вільного дискового простору; процесор з тактовою частотою 2,4 ГГц; операційна система Windows, Linux.

У результаті аналізу сучасних технологій та засобів розробки було обрано оптимальний програмний стек для реалізації музичної вебплатформи. Використання середовища виконання Node.js, мови програмування TypeScript, фреймворків Nest.js і Next.js, системи керування базами даних PostgreSQL та додаткових інструментів Tailwind CSS, Motion for React, Stripe та HLS забезпечує створення сучасної, масштабованої та надійної інформаційної системи.

Обрані технології дозволяють ефективно реалізувати серверну та клієнтську частини вебплатформи, забезпечити швидку обробку запитів, безпечне зберігання даних, потокове відтворення аудіоконтенту, інтеграцію із зовнішніми сервісами та зручний користувацький інтерфейс. Крім того, використання статичної типізації, модульної архітектури та сучасних засобів розробки сприяє підвищенню якості програмного коду, спрощує його супровід і створює передумови для подальшого розвитку та масштабування вебплатформи. Визначені мінімальні системні вимоги підтверджують можливість розгортання та експлуатації системи на сучасних програмно-апаратних платформах без необхідності використання спеціалізованого обладнання.

Висновки до першого розділу

Було проведено аналіз предметної області та визначено основні вимоги до вебплатформи для прослуховування музики. Розглянуто сучасні тенденції розвитку цифрових музичних платформ, що обґрунтувало актуальність створення власної платформи.

Проведено порівняльний аналіз існуючих аналогів, а саме Spotify, YouTube Music, Apple Music та Deezer. Визначено базовий функціонал для подальшої розробки платформи для прослуховування музики.

Обрано клієнт-серверну та багаторівневу архітектуру на основі аналізу їх переваг та недоліків, інструментальні засоби та визначено апаратні вимоги для роботи системи.

РОЗДІЛ 2 ПРОЕКТУВАННЯ ВЕБПЛАТФОРМИ

2.1 Визначення варіантів використання та об'єктно-орієнтованої структури

Варіанти використання системи відображають способи взаємодії користувачів із вебплатформою. Вони дозволяють визначити основні сценарії роботи та об'єкти, які беруть участь у цих процесах. На основі таких сценаріїв формуються вимоги користувачів і визначаються ключові функції системи.

На основі проведеного порівняльного аналізу функціоналу різних вебплатформ можна сформулювати наступні вимоги: можливість прослуховувати, шукати, переглядати пісні в бібліотеці; можливість створювати публічні та приватні плейлисти; можливість зберігати пісні до улюблених; можливість додавати та видаляти пісні в плейлистах; можливість перегляду списку пісень по альбомам, артистах та жанрах; можливість редагування особистих даних в профілі; можливість оформлення підписки на платформі; можливість перегляду, створення, редагування та видалення пісень, виконавців, альбомів та жанрів.

Серед функціональних вимог є основні дії користувачів. Перш за все, передбачено авторизацію, яка включає створення облікового запису та надання доступу до функцій платформи після реєстрації. Додатково система повинна забезпечувати роботу з інформацією, дозволяючи її зберігати, переглядати, додавати, редагувати та видаляти відповідно до прав користувача.

Нефункціональні вимоги до системи складаються з наступних напрямів: сприйняття, надійність, продуктивність та можливість експлуатації.

Сприйняття. Час навчання користувачів повинен складати 20–30 хвилин для нових користувачів та 10–15 хвилин для досвідчених. Час відгуку системи на типові завдання не повинен перевищувати 10 секунд, а на складні завдання – 20 секунд. Інтерфейс повинен бути простим та зрозумілим, із достатньою кількістю контекстних підказок.

Надійність. Час, витрачений на обслуговування системи, не повинен перевищувати 4% від загального часу роботи. Середній час безвідмовної роботи

становить 10 робочих днів. Максимальна допустима норма помилок та дефектів – 1 помилка на 10 000 рядків коду.

Продуктивність. Система повинна підтримувати мінімум 1000 одночасно активних користувачів, які працюють із загальною базою даних.

Можливість експлуатації. Система повинна дозволяти збільшення потужності при зростанні кількості користувачів без погіршення швидкодії та працездатності.

Аналіз функціональних вимог і очікувань користувачів дозволив визначити основні сценарії роботи вебплатформи для прослуховування музики. На основі зібраної інформації побудовано діаграму варіантів використання (рис. 2.1), яка демонструє взаємодію між системою та основними ролями: користувач та адміністратор. Кожна роль має доступ до функціоналу відповідно до покладених обов’язків у межах платформи.

Користувач має можливість реєструватися та авторизуватися у системі, змінювати особисті дані, переглядати треки, слухати їх, шукати та зберігати улюблені пісні, оформлення підписки, історія прослуховувань та створення власних плейлистів із можливістю їх редагування та видалення.

Адміністратор має доступ до управління контентом платформи, як-от додавання треків, альбомів, виконавців та жанрів.

При аналізі функціональних можливостей вебплатформи були визначені ключові сутності, встановлені логічні зв’язки між ними та визначені атрибути, необхідні для підтримки основних процесів. На основі цієї структури створену діаграму класів (рис. 2.2 – 2.3), яка відображає об’єктно-орієнтовану модель системи з деталізованою ієрархією компонентів.

Модель складається з окремих класів, які взаємодіють через API. Класи відповідають за обробку запитів від користувачів, авторизацію та автентифікацію, логіку створення та редагування даних.

Побудована модель є основою для реалізації програмної логіки та взаємодії модулів системи. Вона розмежовує функціональність системи, щоби спростити

тестування та підтримку коду, забезпечити масштабованість і гнучкість платформи для подальшого розвитку функціоналу.

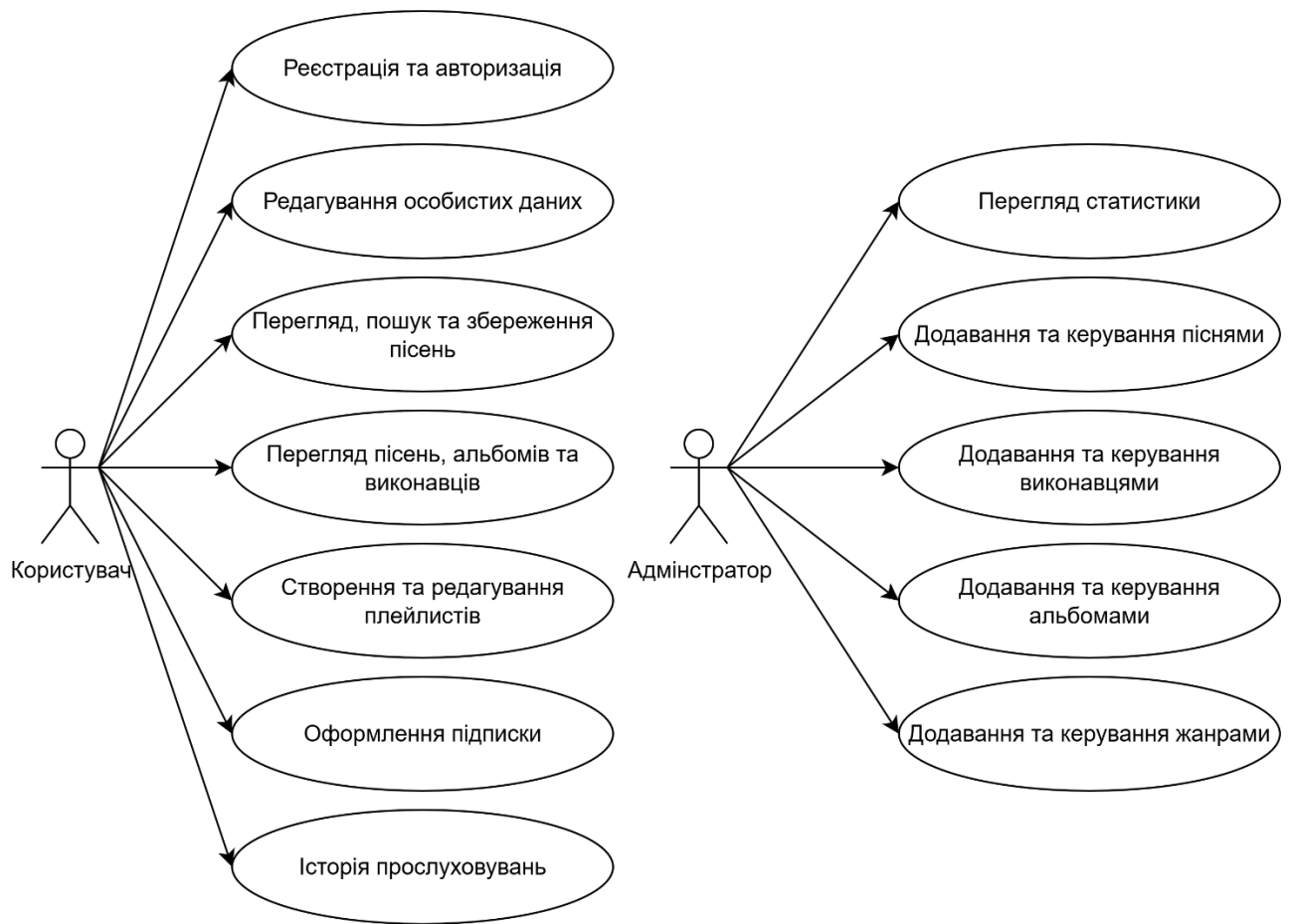


Рис. 2.1. Діаграма варіантів використання платформи

Опишемо діаграму класів.

Subscription – це клас підписки, який має наступні методи: `getSubscriptionPlans()` відповідає за отримання планів підписки; `getMySubscription()` відповідає за отримання підписки поточного користувача; `createCheckout()` відповідає за створення оформлення підписки; `createPortal()` відповідає за створення порталу підписки; `getSubscriptionHistory()` відповідає за отримання історії підписки; `cancelSubscription()` відповідає за скасування підписки.

User – це клас користувача, який має наступні методи: `getCurrentUser()` відповідає за отримання поточного користувача; `updateUser()` відповідає за оновлення користувача.



Рис. 2.2. Діаграма класів

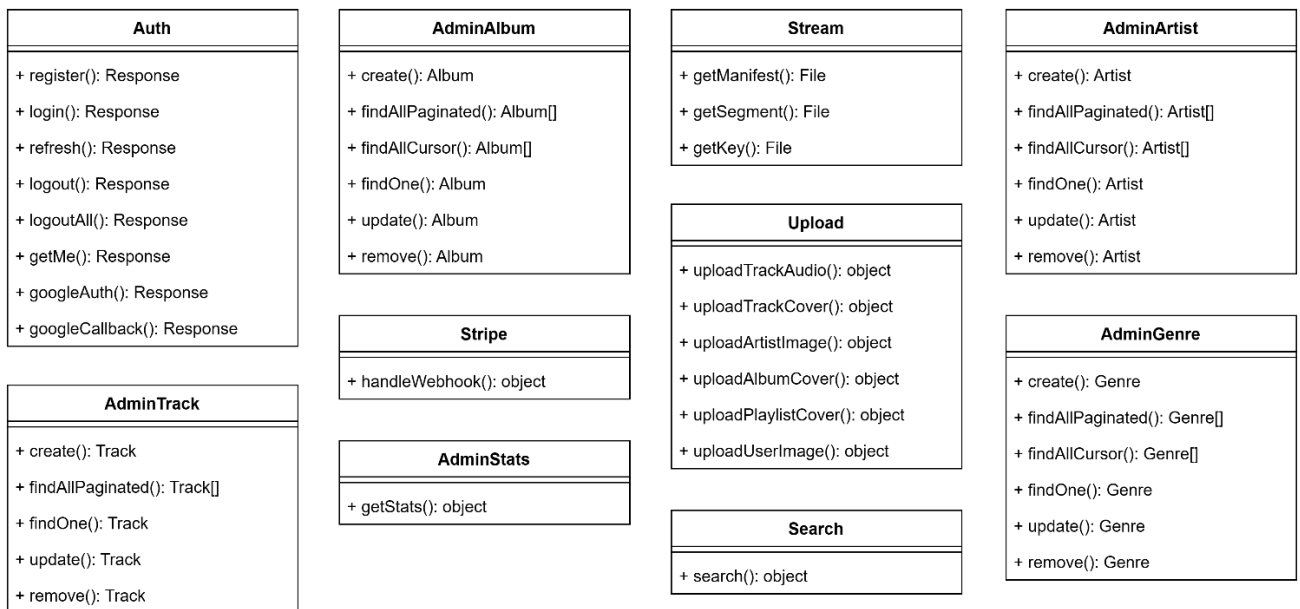


Рис. 2.3. Діаграма класів (продовження)

Artist – це клас виконавця, який має наступні методи: `findAll()` відповідає за отримання виконавців з пагінацією; `findAllCursor()` відповідає за отримання виконавців з курсором; `findBySlug()` відповідає за отримання виконавця по сленгу; `findOne()` відповідає за отримання виконавця; `getTracksByArtist()` відповідає за отримання пісень з курсором конкретного виконавця.

Playlist – це клас плейлиста, який має наступні методи: `create()` відповідає за створення плейлиста; `findMyPlaylists()` відповідає за отримання плейлистів з пагінацією; `findMyPlaylistsCursor()` відповідає за отримання плейлистів з курсором; `findOne()` відповідає за отримання плейлиста; `update()` відповідає за оновлення плейлиста; `remove()` відповідає за видалення плейлиста; `addTrack()` відповідає за додавання пісні до плейлиста; `removeTrack()` відповідає за видалення пісні зі плейлиста.

ListingHistory – це клас історії прослуховувань, який має наступні методи: `findMyHistory()` відповідає за отримання історії прослуховувань поточного користувача; `addTrackToHistory()` відповідає за додавання пісні до історії прослуховувань.

FavoriteTrack – це клас улюблених пісень, який має наступні методи: `findMyFavorites()` відповідає за отримання улюблених пісень поточного користувача; `findMyFavoriteIds()` відповідає за отримання ідентифікаторів

улюблених пісень поточного користувача; `addToFavorite()` відповідає за додавання пісні до списку улюблених; `removeFromFavorite()` відповідає за видалення пісні зі списку улюблених.

`RefreshToken` – це клас токена для оновлення токена доступу, який має наступні методи: `createToken()` відповідає за створення токена; `update()` відповідає за оновлення токена; `updateMany()` відповідає за оновлення токенів.

`Album` – це клас альбому, який має наступні методи: `findAll()` відповідає за отримання альбомів з пагінацією; `findAllCursor()` відповідає за отримання альбомів з курсором; `findBySlug()` відповідає за отримання альбому по сленгу; `findOne()` відповідає за отримання альбому; `getTracksByAlbum()` відповідає за отримання пісень з курсором конкретного альбому.

`Genre` – це клас жанру, який має наступні методи: `findAll()` відповідає за отримання жанрів з пагінацією; `findAllCursor()` відповідає за отримання жанрів з курсором; `findBySlug()` відповідає за отримання жанру по сленгу; `findOne()` відповідає за отримання жанру; `getTracksByGenre()` відповідає за отримання пісень з курсором конкретного жанру.

`Track` – це клас пісні, який має наступні методи: `findAll()` відповідає за отримання пісень з пагінацією; `findAllCursor()` відповідає за отримання пісень з курсором; `findBySlug()` відповідає за отримання пісні по сленгу; `findOne()` відповідає за отримання пісні.

`Auth` – це клас авторизації, який має наступні методи: `register()` відповідає за реєстрацію користувача; `login()` відповідає за вхід користувача; `refresh()` відповідає за оновлення оновлюючого та токена доступу; `logout()` відповідає за вихід користувача; `logoutAll()` відповідає за завершення всіх сесій користувача; `getMe()` відповідає за отримання поточного користувача; `googleAuth()` відповідає за вхід через Google; `googleCallback()` відповідає за отримання відповіді від Google.

`AdminAlbum` – це клас альбомів для керування адміністратором, які має наступні методи: `create()` відповідає за створення альбому; `findAllPaginated()` відповідає за отримання альбомів з пагінацією; `findAllCursor()` відповідає за отримання альбомів з курсором; `findOne()` відповідає за отримання альбому;

update() відповідає за оновлення альбому; remove() відповідає за видалення альбому.

Stream – це клас потоку пісень, який має наступні методи: getManifest() відповідає за отримання файлу маніфесту пісень; getSegment() відповідає за отримання конкретного сегменту пісні; getKey() відповідає за отримання ключа для правильного відтворення пісні.

AdminArtist – це клас виконавців для керування адміністратором, який має наступні методи: create() відповідає за додавання виконавця; findAllPaginated() відповідає за отримання виконавців з пагінацією; findAllCursor() відповідає за отримання виконавців з курсором; findOne() відповідає за отримання виконавця; update() відповідає за оновлення виконавця; remove() відповідає за видалення виконавця.

AdminTrack – це клас пісень для керування адміністратором, який має наступні методи: create() відповідає за створення пісні; findAllPaginated() відповідає за отримання альбомів з пагінацією; findOne() відповідає за отримання пісні; update() відповідає за оновлення пісні; remove() відповідає за видалення пісні.

Stripe – це клас платіжної системи Stripe, який має наступні методи: handleWebhook() відповідає за отримання подій від Stripe.

Upload – це клас завантаження файлів, який має наступні методи: uploadTrackAudio() відповідає за завантаження аудіо пісні; uploadTrackCover() відповідає за завантаження обкладинки пісні; uploadArtistImage() відповідає за завантаження зображення виконавця; uploadAlbumCover() відповідає за завантаження обкладинки альбому; uploadPlaylistCover() відповідає за завантаження обкладинки плейлиста; uploadUserImage() відповідає за завантаження зображення користувача.

AdminGenre – це клас жанрів для керування адміністратором, який має наступні методи: create() відповідає за додавання жанру; findAllPaginated() відповідає за отримання жанрів з пагінацією; findAllCursor() відповідає за отримання жанрів з курсором; findOne() відповідає за отримання жанру; update() відповідає за оновлення жанру; remove() відповідає за видалення жанру.

AdminStats – це клас статистики для керування адміністратором, який має наступні методи: `getStats()` відповідає за отримання статистики платформи.

Search – це клас пошуку контенту, який має наступні методи: `search()` відповідає за пошук та отримання даних з платформи.

2.2 Розробка бази даних системи

Для зберігання інформації вебплатформи для прослуховування музики використовується PostgreSQL – реляційна система керування базами даних. Вона оперує структурованими таблицями та зв'язками між ними для надійного зберігання даних.

Структура бази даних платформи складається з одинадцяти таблиць. Розглянемо призначення кожної з них.

User – таблиця, що зберігає інформацію про користувачів.

RefreshToken – таблиця, що зберігає інформацію про оновлюючі токени.

Artist – таблиця, що зберігає інформацію про виконавців.

Genre – таблиця, що зберігає інформацію про жанри.

Album – таблиця, що зберігає інформацію про альбоми.

Track – таблиця, що зберігає інформацію про пісні.

FavoriteTrack – таблиця, що зберігає інформацію про улюблені пісні користувачів.

Playlist – таблиця, що зберігає інформацію про плейлисти користувачів.

PlaylistTrack – таблиця, що зберігає інформацію про зв'язок між плейлистом та піснею.

ListeningHistory – таблиця, що зберігає інформацію про історію прослуховувань користувача.

Subscription – таблиця, що зберігає інформацію про підписки користувачів.

Опис кожної з таблиць бази даних представлений у табл. 2.1 - 2.11.

Табл. 2.1 містить опис структури таблиці User, у якій зберігаються основні дані користувача, зокрема ідентифікатори, ім'я та адреса електронної пошти.

Таблиця 2.1

Опис таблиці User

	Тип даних	Опис
id	String	Ідентифікатор користувача
googleId	String	Ідентифікатор Google
username	String	Ім'я користувача
email	String	Пошта користувача
passwordHash	String	Хеш пароль
avatarKey	String	Ключ зображення
role	String	Роль користувача
createdAt	DateTime	Дата створення
updatedAt	DateTime	Дата оновлення

Табл. 2.2 містить опис структури таблиці RefreshToken, у якій зберігаються токени оновлення користувачів, дані про сесії та параметри пристроїв.

Таблиця 2.2

Опис таблиці RefreshToken

Назва поля	Тип даних	Опис
id	String	Ідентифікатор
userId	String	Ідентифікатор користувача
tokenHash	String	Хеш токену
expiresAt	DateTime	Дата придатності
createdAt	DateTime	Дата створення
revokedAt	DateTime	Дата відкликання
userAgent	String	Рядок ідентифікації користувача
ipAddress	String	IP-адреса
deviceType	String	Тип пристрою
browser	String	Браузер
os	String	Операційна система

sessionName	String	Назва сесії
replacedByTokenId	String	Ідентифікатор минулого токена

Табл. 2.3 містить опис структури таблиці Artist, у якій зберігаються основні дані про виконавців, зокрема ім'я, біографія та зображення.

Таблиця 2.3

Опис таблиці Artist

Назва поля	Тип даних	Опис
id	String	Ідентифікатор
name	String	Ім'я виконавця
slug	String	Псевдонім виконавця
bio	String	Біографія виконавця
imageKey	String	Ключ зображення
createdAt	DateTime	Дата створення
updatedAt	DateTime	Дата оновлення

Табл. 2.4 містить опис структури таблиці Genre, у якій зберігаються назви музичних жанрів та їх унікальні ідентифікатори.

Таблиця 2.4

Опис таблиці Genre

Назва поля	Тип даних	Опис
id	String	Ідентифікатор
name	String	Назва жанру
slug	String	Сленг жанру
createdAt	DateTime	Дата створення

Табл. 2.5 містить опис структури таблиці Album, у якій зберігаються дані про музичні альбоми, зокрема їх назви, описи, обкладинки та дати випуску.

Опис таблиці Album

Назва поля	Тип даних	Опис
id	String	Ідентифікатор
artistId	String	Ідентифікатор виконавця
title	String	Назва альбому
slug	String	Сленг альбому
description	String	Опис альбому
coverImageKey	String	Ключ обкладинки
releaseDate	DateTime	Дата виходу альбому
isPublished	Boolean	Опублікований альбом
createdAt	DateTime	Дата створення
updatedAt	DateTime	Дата оновлення

Табл. 2.6 містить опис структури таблиці Track, у якій зберігаються дані про музичні композиції, їх характеристики та зв'язки з виконавцями, альбомами і жанрами.

Опис таблиці Track

Назва поля	Тип даних	Опис
id	String	Ідентифікатор
artistId	String	Ідентифікатор виконавця
albumId	String	Ідентифікатор альбому
genreId	String	Ідентифікатор жанру
title	String	Назва пісні
slug	String	Сленг пісні
description	String	Опис пісні
duration	Integer	Тривалість пісні
hlsManifestKey	String	Ключ маніфесту пісні

hlsDirectoryKey	String	Ключ розташування маніфесту
coverImageKey	String	Ключ обкладинки пісні
trackNumber	Integer	Номер пісні
playsCount	Integer	Кількість відтворень пісні
isPublished	Boolean	Опублікована пісня
createdAt	DateTime	Дата створення
updatedAt	DateTime	Дата оновлення

Табл. 2.7 містить опис структури таблиці FavoriteTrack, у якій зберігаються відомості про музичні композиції, додані користувачами до улюблених.

Таблиця 2.7

Опис таблиці FavoriteTrack

Назва поля	Тип даних	Опис
userId	String	Ідентифікатор користувача
trackId	String	Ідентифікатор пісні
createdAt	DateTime	Дата створення

Табл. 2.8 містить опис структури таблиці Playlist, у якій зберігаються дані про плейлисти користувачів, їх назви, описи та налаштування видимості.

Таблиця 2.8

Опис таблиці Playlist

Назва поля	Тип даних	Опис
id	String	Ідентифікатор
userId	String	Ідентифікатор користувача
name	String	Назва плейлиста
description	String	Опис плейлиста
coverImageKey	String	Ключ обкладинки плейлиста
visibility	String	Видимість плейлиста

createdAt	DateTime	Дата створення
updatedAt	DateTime	Дата оновлення

Табл. 2.9 містить опис структури таблиці PlaylistTrack, у якій зберігаються дані про композиції, що входять до плейлистів, та порядок їх розташування.

Таблиця 2.9

Опис таблиці PlaylistTrack

Поле	Тип даних	Опис
id	String	Ідентифікатор
playlistId	String	Ідентифікатор плейлиста
trackId	String	Ідентифікатор пісні
order	Integer	Порядок розташування пісні
addedAt	DateTime	Дата додавання

Табл. 2.10 містить опис структури таблиці ListeningHistory, у якій зберігається історія прослуховування музичних композицій користувачами.

Таблиця 2.10

Опис таблиці ListeningHistory

Поле	Тип даних	Опис
id	String	Ідентифікатор
userId	String	Ідентифікатор користувача
trackId	String	Ідентифікатор пісні
listenedAt	DateTime	Дата прослуховування

Табл. 2.11 містить опис структури таблиці Subscription, у якій зберігаються дані про підписки користувачів, їх тип, статус та термін дії.

Таблиця 2.11

Опис таблиці Subscription

Поле	Тип даних	Опис
id	String	Ідентифікатор
userId	String	Ідентифікатор користувача
stripeCustomerId	String	Ідентифікатор клієнта Stripe
stripeSubscriptionId	String	Ідентифікатор підписки Stripe
stripePriceId	String	Ідентифікатор ціни Stripe
status	String	Статус підписки
type	String	Тип підписки
trialEndsAt	DateTime	Дата завершення пробного періоду
currentPeriodEndsAt	DateTime	Дата поточного завершення періоду
cancelAtPeriodEnd	Boolean	Скасовано в кінці періоду
createdAt	DateTime	Дата створення
updatedAt	DateTime	Дата оновлення

2.3 Проектування та реалізація алгоритмів роботи системи

З урахуванням функціонального призначення розроблюваної платформи вебзастосунок спроектовано таким чином, щоби забезпечити зручну взаємодію користувача з інтерфейсом, швидкий доступ до основних функціональних можливостей системи та інтуїтивно зрозумілий процес виконання всіх передбачених операцій. Архітектура програмного забезпечення орієнтована на мінімізацію кількості дій, необхідних для виконання типових сценаріїв роботи, що підвищує ефективність використання платформи та покращує користувацький досвід.

Для наочного відображення логіки взаємодії різних категорій користувачів із системою було побудовано діаграми активностей, які демонструють послідовність виконання основних функцій та можливі сценарії використання вебзастосунку.

На рис. 2.4 представлено діаграму активності користувача, яка відображає послідовність основних дій, що можуть виконуватися авторизованим користувачем музичної платформи. Після успішної автентифікації користувач отримує доступ до функцій пошуку та прослуховування музичних композицій, перегляду інформації про альбоми, виконавців і музичні жанри, а також створення, редагування та видалення власних плейлистів. Крім цього, передбачено можливість додавання композицій до списку обраного, перегляду історії прослуховувань, оформлення преміум-підписки, а також перегляду та редагування персональних даних користувача. Після виконання необхідних операцій сценарій взаємодії із системою завершується.

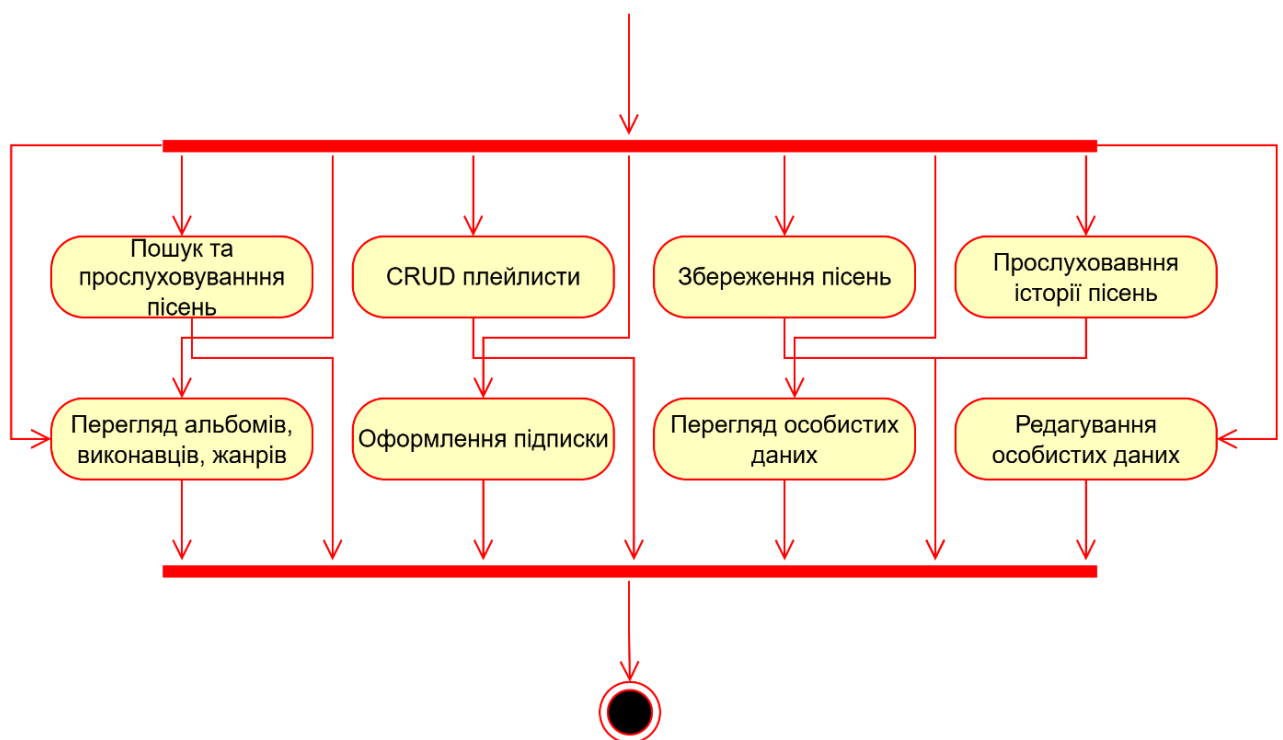


Рис. 2.4. Діаграма активності користувача

На рис. 2.5 наведено діаграму активності адміністратора, яка ілюструє процес виконання адміністративних функцій у межах музичної платформи. Після авторизації адміністратор отримує доступ до інструментів керування музичним контентом, що включають створення, редагування та видалення музичних композицій, альбомів, виконавців і жанрів. Додатково адміністратор має

можливість переглядати статистичні показники роботи платформи, здійснювати керування власними персональними даними та забезпечувати коректне функціонування інформаційної системи. Наведена діаграма демонструє основні сценарії взаємодії адміністратора із системою та відображає логічну послідовність виконання адміністративних операцій. Після завершення необхідних дій робота адміністратора із системою завершується.

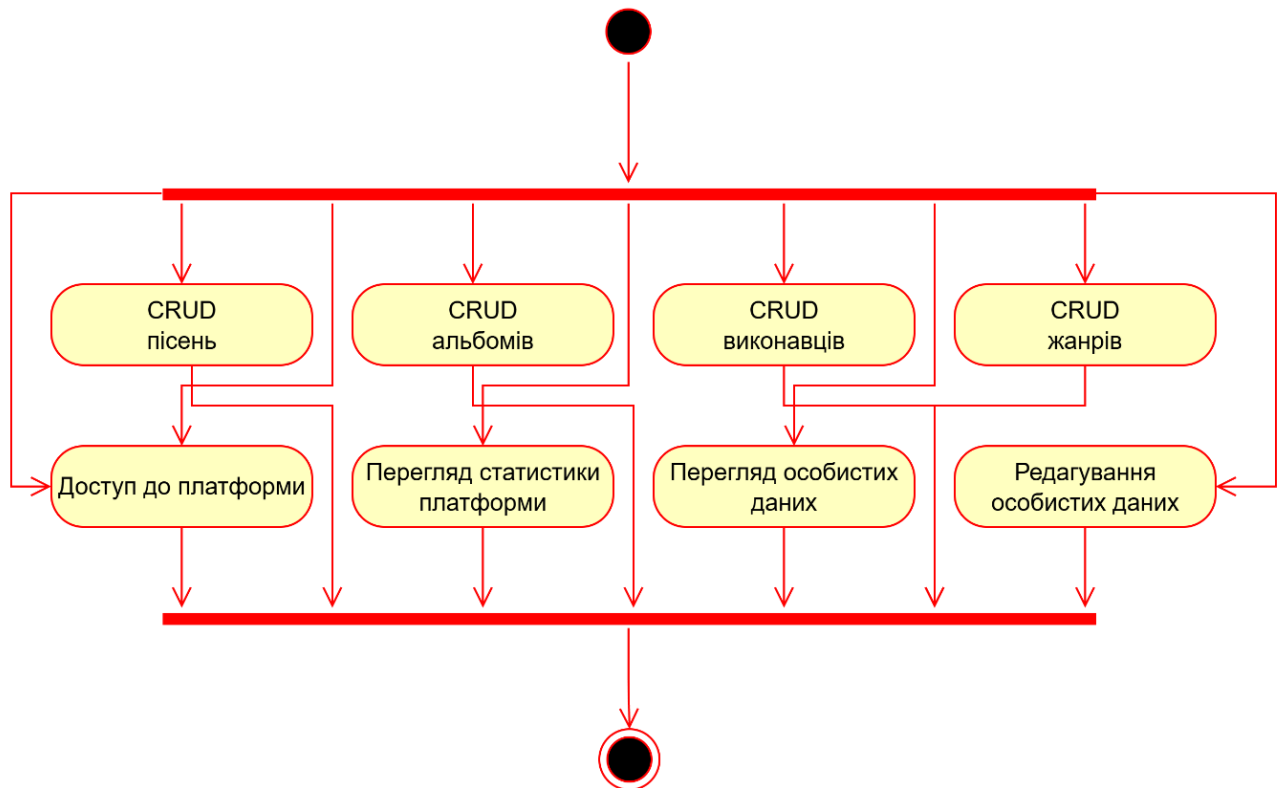


Рис. 2.5. Діаграма активності адміністратора

Таким чином, побудовано діаграми активностей для користувача та адміністратора, які відображають основні сценарії взаємодії із системою та демонструють функціональні можливості кожної ролі. Використання діаграм дозволило формалізувати бізнес-процеси, визначити послідовність виконання операцій та забезпечити цілісне уявлення про логіку роботи вебзастосунку. Отримані результати створюють основу для подальшої реалізації програмної частини системи та її практичного впровадження.

2.4 Реалізація вебплатформи для прослуховування музики

Розглянемо реалізацію компонентів із функціоналу платформи, а саме механізм завантаження, обробки та відтворення музичного контенту.

Все починається з завантаження адміністратором аудіофайлу на стороні клієнта.

На початковому етапі створення пісні адміністратор обирає аудіофайл для завантаження його на сервер. Після вибору файлу відбувається відправка його на локальний маршрут API. Нижче представлено фрагмент коду відправки файлу на локальний сервер клієнта.

```
export async function uploadTrackAudio(
  file: File,
  onProgress?: (progress: number) => void,
): Promise<UploadAudioResponseDto> {
  return await uploadFile(
    file,
    localApiEndpoints.upload.track.audio,
    onProgress,
  );
}

export async function uploadFile<T>(
  file: File,
  endpoint: string,
  onProgress?: (progress: number) => void,
): Promise<T> {
  const formData = new FormData();
  formData.append("file", file);
  return new Promise<T>((resolve, reject) => {
    const xhr = new XMLHttpRequest();
    xhr.open("POST", endpoint);
    xhr.upload.onprogress = (event) => {
      if (!event.lengthComputable) return;
      const progress = Math.round((event.loaded / event.total) * 100);
      onProgress?.(progress);
    };
    xhr.onload = () => {
      try {
```

```

    const data = JSON.parse(xhr.responseText) as T;
    if (xhr.status >= 200 && xhr.status < 300) {
        resolve(data);
    } else {
        reject(new ApiError("Upload failed", xhr.status));
    }
} catch {
    reject(new ApiError("Upload failed", xhr.status));
}
});
xhr.onerror = () => {
    reject(new ApiError("Network error during upload", xhr.status));
};
xhr.send(formData);
});
}

```

У даному коді реалізовано механізм завантаження різних файлів на локальні маршрути API із використанням об'єкта XMLHttpRequest. Передавання файлу здійснюється у форматі FormData для коректної відправки мультимедійного контенту через HTTP-запит. Додатково реалізовано відстеження прогресу завантаження для відображення його в інтерфейсі користувача, обробку успішної відповіді сервера та механізми обробки помилок мережі й некоректних відповідей.

Після відправки аудіофайлу клієнтською частиною застосунку, запит надходить на локальний API-маршрут, реалізований у Next.js. Цей маршрут виконує роль проміжного серверного шару, який приймає POST-запит від клієнта, обробляє отриманий файл та надалі перенаправляє його на основний сервер для подальшого збереження й обробки. Нижче наведено код отримання аудіофайлу на серверному маршруті.

```

export async function POST(request: NextRequest) {
    try {
        const accessToken = await getAccessToken();
        if (!accessToken) {
            return NextResponse.json({ message: "Unauthorized" }, { status: 401 });
        }
        const formData = await request.formData();
        const data = await serverApiClient<UploadAudioResponseDto>(

```

```

        backendApiEndpoints.upload.track.audio,
        {
            method: "POST",
            token: accessToken,
            body: formData,
        },
    );
    return NextResponse.json(data);
} catch (error) {
    return getNextErrorResponse(error);
}
}

```

Після відправки файлу з локального маршруту він потрапляє на серверну частину вебплатформи. Нижче наведено фрагмент коду, який приймає аудіофайл, тимчасово локального його зберігає, перевіряє тип та розмір файлу, які допустимі для збереження музичного контенту.

```

@Post('tracks/audio')
@Roles(UserRole.ADMIN)
@UseInterceptors(
    FileInterceptor('file', {
        storage: createDiskStorage('audio'),
        fileFilter: createFileFilter(audioFileType),
        limits: {
            fileSize: audioMaxFileSize,
        },
    }),
)
async uploadTrackAudio(
    @UploadedFile(
        new ParseFilePipe({
            validators: [new MaxFileSizeValidator({ maxSize: audioMaxFileSize })],
        }),
    )
    file: Express.Multer.File,
) {
    return this.uploadService.uploadTrackAudio(file);
}

```

Після валідації та збереження файлу виконується виклик методу `uploadTrackAudio`, який відповідає за подальшу обробку аудіо контенту. У цьому

методі здійснюється конвертація завантаженого аудіофайлу у формат HLS за допомогою сервісу HlsService. У результаті обробки отримуємо структуру потокового аудіо з генерацією ключів доступу до HLS-маніфесту та директорії із сегментами, які будуть збережені разом з створеною піснею. У випадку виникнення будь-якої помилки буде повернено відповідне повідомлення про помилку. Після завершення обробки тимчасовий файл автоматично видаляється з файлової системи сервера.

```
async uploadTrackAudio(
  file: Express.Multer.File,
): Promise<UploadAudioResponseDto> {
  try {
    const uploadResult = await this.hlsService.convertAudioToHls(file.path);
    return {
      hlsManifestKey: uploadResult.hlsManifestKey,
      hlsDirectoryKey: uploadResult.hlsDirectoryKey,
    };
  } catch (error) {
    throw new BadRequestException('Failed to convert audio to HLS');
  } finally {
    await unlink(file.path).catch(() => undefined);
  }
}
```

Для конвертації аудіофайлу в формат HLS використовується програмний інструмент ffmpeg [19], який призначений для обробки, конвертації, кодування та потокового передавання аудіо- і відеоданих у різних мультимедійних форматах.

Під час конвертації аудіофайл розбивається на невеликі аудіосегменти, які об'єднуються в єдиний маніфест файл для відтворення аудіо. Для захисту та доступу аудіосегментів використовується механізм шифрування з генерацією ключа без якого не можливо відтворювати аудіо. В сучасних вебплатформах використовується технологія DRM для захисту цифрового контенту, але вона потребує додаткових витрат в плані використання стороннього сервера та аудіо плеєра для програвання аудіо контенту, тому було реалізовано спрощений механізм з реалізацією захисту у вигляді використання ключа шифрування. Детальний код повного HlsService сервісу представлено у Додатку А.

Після успішної конвертації адміністратор може додати пісню до бібліотеки, яку вже можна відтворювати.

Для відтворення мультимедійного контенту у форматі HLS на стороні клієнта використовується бібліотека hls.js [20], яка надає підтримку потокового відтворення аудіо в браузерях, що не мають вбудованої підтримки HLS. У наведеному нижче фрагменті коду реалізовано ініціалізацію HLS-плеєра, підключення аудіопотоку до HTML-аудіоелемента та очищення ресурсів при зміні поточного треку або завершенні роботи компонента.

```
useEffect(() => {
  const audio = audioRef.current;
  if (!audio || !currentTrack) return;
  hlsRef.current?.destroy();
  hlsRef.current = null;
  audio.pause();
  audio.removeAttribute("src");
  audio.load();
  if (Hls.isSupported()) {
    const hls = new Hls({
      enableWorker: true,
    });
    hlsRef.current = hls;
    hls.loadSource(currentTrack.audioUrl);
    hls.attachMedia(audio);
    return () => {
      hls.destroy();
      hlsRef.current = null;
    };
  }
  if (audio.canPlayType("application/vnd.apple.mpegurl")) {
    audio.src = currentTrack.audioUrl;
    audio.load();
  }
}, [currentTrack]);
```

Під час відтворення аудіофайлу відбувається отримання маніфест файлу для відтворення аудіо. Отримання відбувається через локальний API-маршрут по trackId, код якого наведений нижче.

```
export async function GET(_: NextRequest, { params }: RouteParams) {
```

```

const { trackId } = await params;
const accessToken = await getAccessToken();
if (!accessToken) {
  return new Response("Unauthorized", { status: 401 });
}
const response = await fetch(
  `${env.apiUrl}${backendApiEndpoints.stream.trackManifest(trackId)}`,
  {
    headers: {
      Authorization: `Bearer ${accessToken}`,
    },
    cache: "no-store",
  },
);
const body = await response.text();
return new Response(body, {
  status: response.status,
  headers: {
    "Content-Type": "application/vnd.apple.mpegurl",
    "Cache-Control": "no-store",
  },
});
}

```

Після успішного отримання маніфесту, бібліотека `hls.js` зчитує файл та отримує інформацію про правильність відтворення аудіоконтенту. Після читання бібліотека автоматично зробить ще один додатковий запит для отримання ключа доступу, який потрібен для відтворення аудіосегментів. Нижче представлено код локального API-маршруту за допомогою якого отримується ключ доступу по `trackId`.

```

export async function GET(_: NextRequest, { params }: RouteParams) {
  const { trackId } = await params;
  const accessToken = await getAccessToken();
  if (!accessToken) {
    return new Response("Unauthorized", { status: 401 });
  }
  const response = await fetch(
    `${env.apiUrl}${backendApiEndpoints.stream.trackKey(trackId)}`,
    {
      headers: {

```

```

        Authorization: `Bearer ${accessToken}`,
    },
    cache: "no-store",
  },
);
return new Response(response.body, {
  status: response.status,
  headers: {
    "Content-Type": "application/octet-stream",
    "Cache-Control": "no-store",
  },
});
}

```

Після отримання ключа бібліотека hls.js починає отримувати аудіосегменти які будуть відтворені в порядку зазначені в файлі маніфесту. Отримання сегментів також відбувається через локальний API-маршрут.

На стороні сервера під час отримання маніфесту відбувається пошук пісні та відправка його на клієнт. За схожим принципом отримується ключ доступу та аудіосегменти. Детальний код роботи перевірки та відправки HLS-файлів наведено в Додатку Б відповідно.

Висновки до другого розділу

Під час проектування вебплатформи для прослуховування музики було проведено аналіз основних сценаріїв взаємодії користувачів із системою та визначено функціональні та нефункціональні вимоги до ПЗ. На основі аналізу створено об'єктно-орієнтовану модель для реалізації функціональних компонентів серверної частини вебплатформи.

Було реалізовано та описано структуру бази даних для зберігання інформації про користувачів, пісні, альбоми, виконавці та інше з використанням реляційної бази даних PostgreSQL.

Описано основні сценарії взаємодії користувачів із платформою, а саме основні операції, які виконуються в межах типових процесів функціонування вебзастосунку.

РОЗДІЛ 3 ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБПЛАТФОРМОЮ

3.1 Порядок встановлення та налаштування параметрів вебплатформи

Для забезпечення стабільної та безперебійної роботи вебплатформи важливим етапом є правильне проєктування процесу її розгортання. Архітектура розгортання демонструє взаємодію між окремими компонентами системи, способи обміну даними, порядок використання зовнішніх сервісів. Під час розробки платформи процес розгортання передбачає поетапне налаштування клієнтської та серверної частин, бази даних, системи зберігання файлів, інтеграцію із зовнішніми сервісами, необхідними для реалізації додаткового функціоналу. Такий підхід забезпечує масштабованість, спрощує супровід програмного забезпечення та підвищує надійність його роботи.

На рис. 3.1 наведено діаграму розгортання, яка відображає структуру музичної платформи, взаємозв'язки між її програмними компонентами та зовнішніми сервісами. Діаграма демонструє процес взаємодії клієнтської частини із сервером застосунку, базою даних і сервісом обробки платежів, що забезпечує коректне функціонування всіх можливостей вебплатформи.

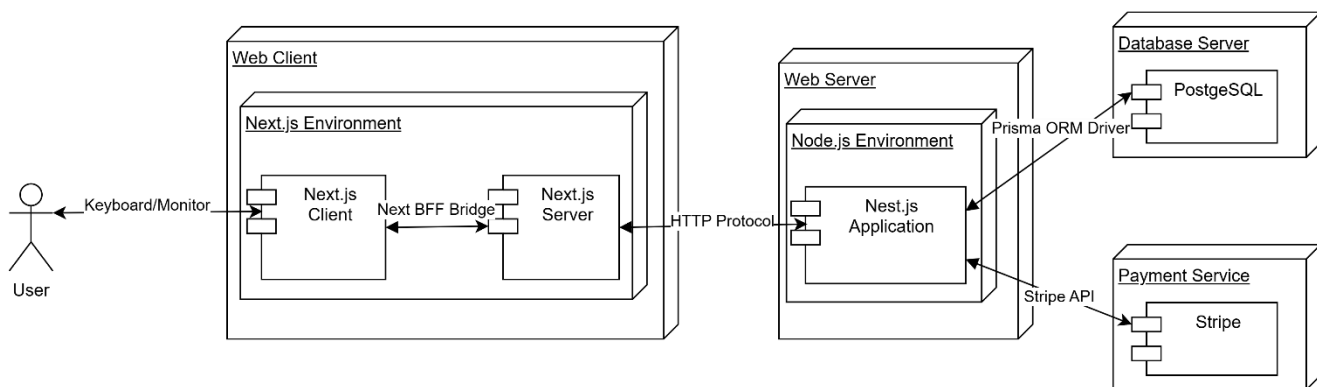


Рис. 3.1. Діаграма розгортання платформи

Опишемо основні компоненти діаграми розгортання.

Web Client – клієнтська частина вебплатформи, реалізована у вигляді вебзастосунку на основі фреймворку Next.js. Вона відповідає за формування користувацького інтерфейсу, відображення інформації та взаємодію користувача із

системою. Під час першого завантаження сторінок застосунок виконує HTTP-запити безпосередньо до серверної частини для отримання необхідних даних. Для виконання подальших дій користувача, наприклад створення плейлистів, додавання композицій до улюблених, оформлення підписки або редагування персональних даних, використовується механізм Next BFF (Backend for Frontend) Bridge. Спочатку запит надходить до локального сервера Next.js, який виконує додаткову обробку, після чого перенаправляє його до серверної частини платформи з метою приховання внутрішньої структури API від клієнта.

Web Server – серверна частина вебплатформи, реалізована за допомогою фреймворку Nest.js, який працює в середовищі виконання Node.js. Сервер містить бізнес-логіку системи, обробку HTTP-запитів, автентифікацію та авторизацію користувачів, управління музичним каталогом, плейлистами, історією прослуховувань і підписками. Окрім роботи з текстовими даними, сервер керує медіафайлами, включаючи їх завантаження, зберігання та передачу клієнтській частині. Для взаємодії з реляційною базою даних використовується ORM Prisma, для зручного доступу до даних, що спрощує виконання операцій створення, читання, оновлення й видалення інформації. Для реалізації платних підписок сервер інтегрований із зовнішньою платіжною системою Stripe, взаємодія з якою здійснюється через Stripe API.

Database Server – сервер бази даних, призначений для централізованого зберігання структурованої інформації, необхідної для роботи вебплатформи. На ньому розгорнуто систему керування базами даних PostgreSQL, у якій зберігаються дані про користувачів, музичні композиції, альбоми, виконавців, жанри, плейлисти, історію прослуховувань, улюблені композиції, токени авторизації і т.д. Усі операції з базою даних виконуються тільки серверною частиною, з метою захисту інформації та дотримання принципів безпеки.

Payment Service – зовнішній сервіс обробки платіжних операцій, представлений платформою Stripe. Він використовується для оформлення та керування платними підписками користувачів, виконання безпечної обробки платежів, зберігання інформації про статус підписок і отримання повідомлень про

зміну стану платіжних операцій. Через використання Stripe, фінансові дані користувачів не зберігаються безпосередньо на сервері вебплатформи відповідно до сучасних вимог інформаційної безпеки з метою зменшення ризиків витоку конфіденційної інформації.

Таким чином, визначено основні програмні компоненти вебплатформи та принципи їх взаємодії, побудовано діаграму розгортання, яка наочно відображає структуру системи, включаючи клієнтську та серверну частини, сервер бази даних і зовнішній платіжний сервіс.

3.2 Структура інтерфейсу вебплатформи

При переході на головну сторінку вебплатформи неавторизованому користувачеві відображається сторінка з детальною інформацією про вебплатформу. Користувач може переглянути про переваги та можливості, які надає вебплатформа. На рис. 3.2 наведено головну сторінку вебплатформи.

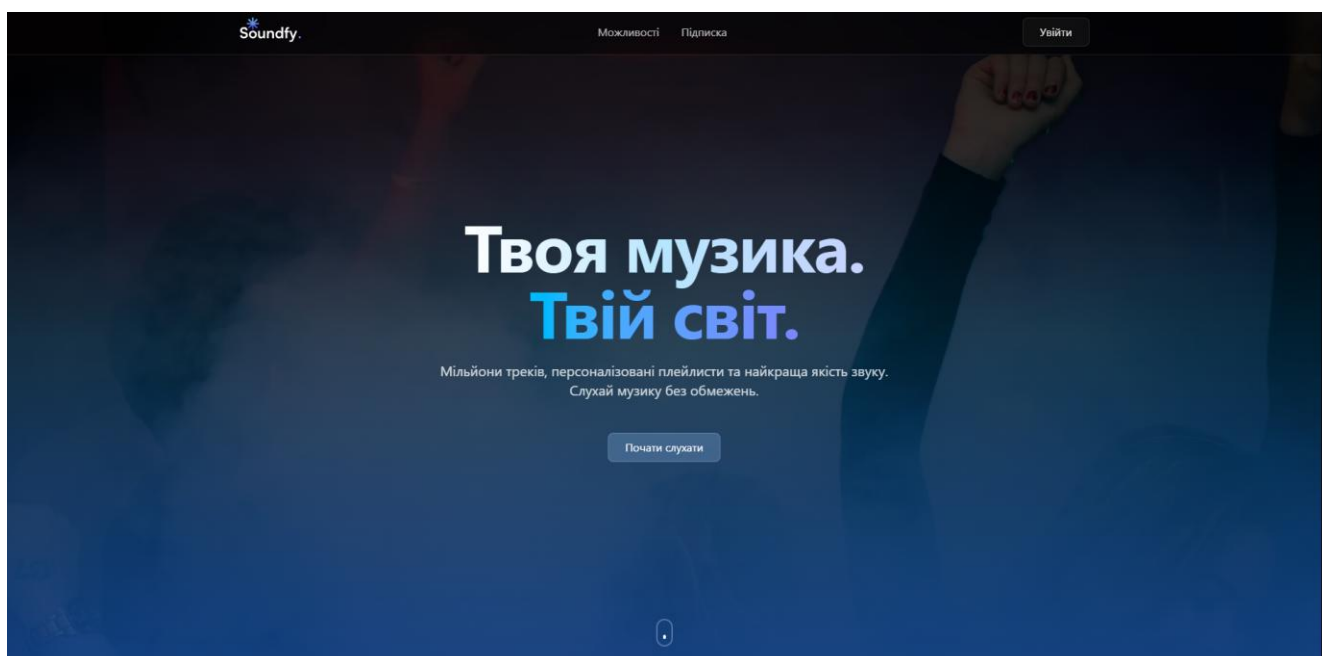


Рис. 3.2. Головна сторінка вебплатформи

На головній сторінці присутня внутрішня навігація та між сторінками. При натисканні на посилання підписки відкриється сторінка підписки, де користувач може ознайомитися з цінами та перевагами на різні версії підписок. На рис. 3.3 представлено сторінку підписки вебплатформи.

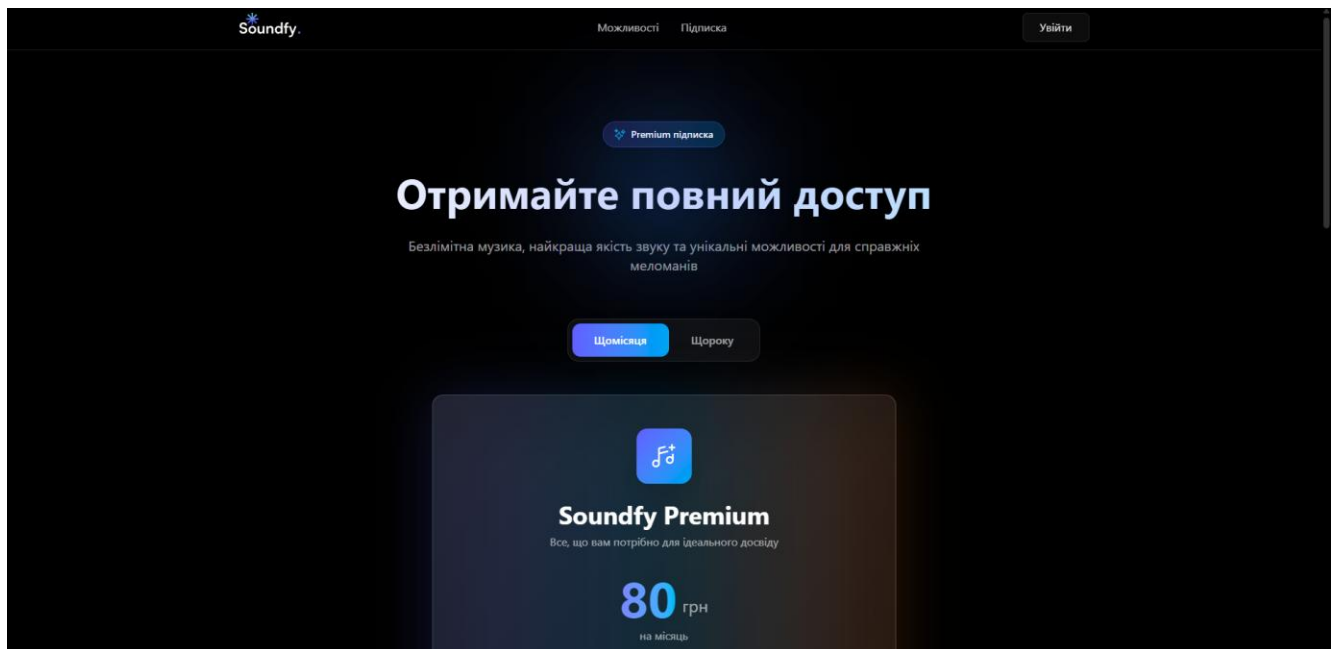


Рис. 3.3. Сторінка підписки вебплатформи

При натисканні кнопки увійти відкриється вікно входу, де потрібно ввести електронну пошту та пароль, щоби увійти на платформу. Для створення нового облікового запису потрібно натиснути на зареєструватися, після чого відкриється нове вікно з формою, яку необхідно заповнити: ім'я користувача, електронна пошта та пароль. За бажанням, користувач може увійти або зареєструватися через Google. На рис. 3.4 та 3.5 зображено вікна входу та реєстрації відповідно.

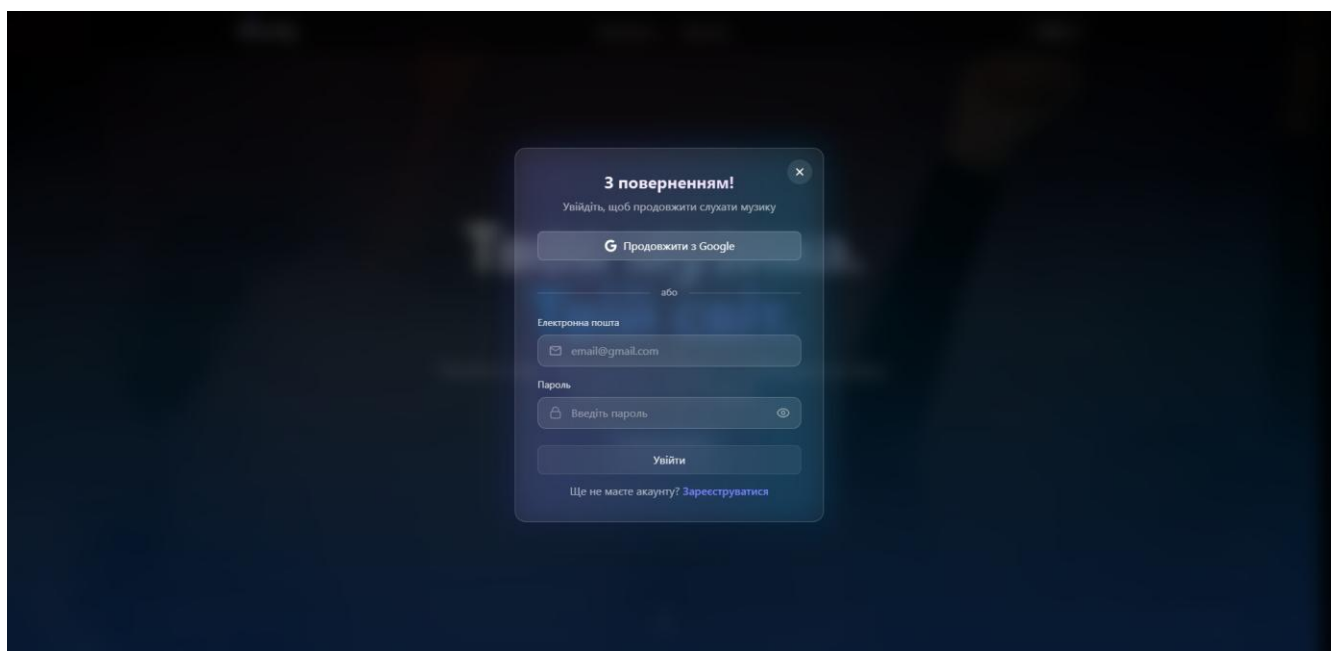


Рис. 3.4. Вікно входу

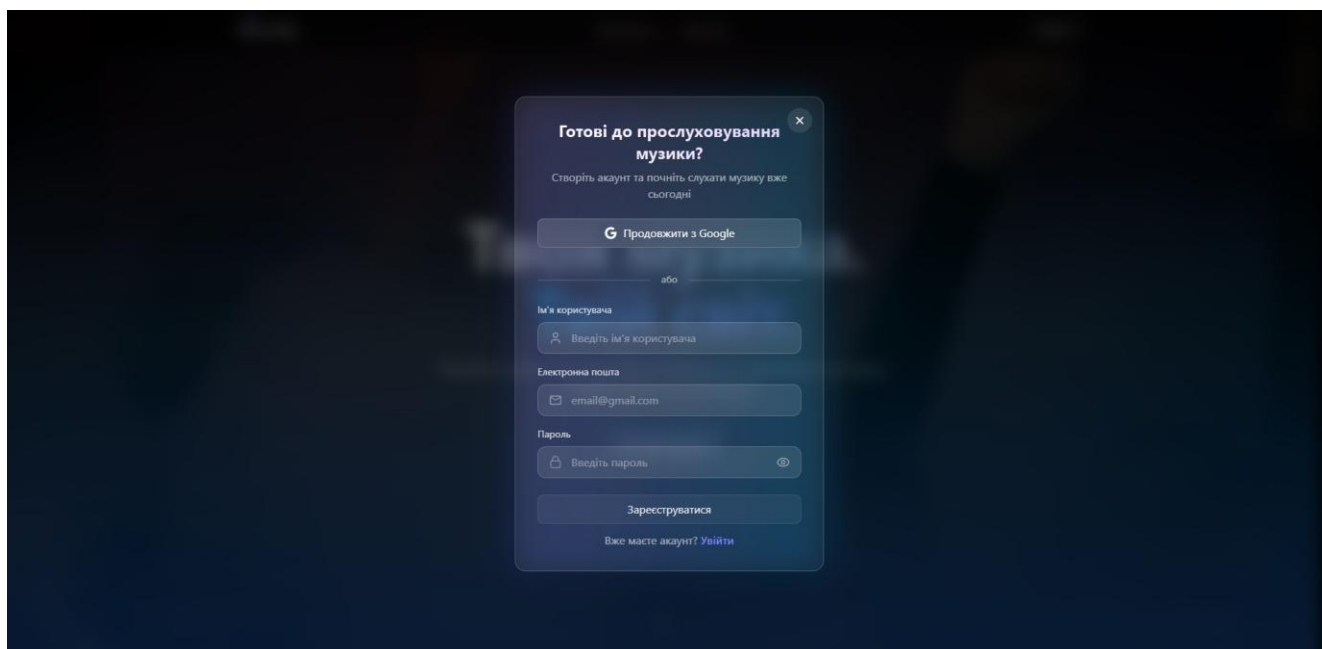


Рис. 3.5. Вікно реєстрації

Після успішного входу користувача буде переадресовано на сторінку бібліотеки платформи. На сторінці бібліотеки користувач може використати швидкий доступ для прослуховування своїх улюблених пісень, власних плейлистів та історії прослуховувань. Окрім швидкого доступу є доступ до прослуховування останніх випущених альбомів, пісень, а також виконавців та жанрів. На рис. 3.6 представлено бібліотеку платформи.

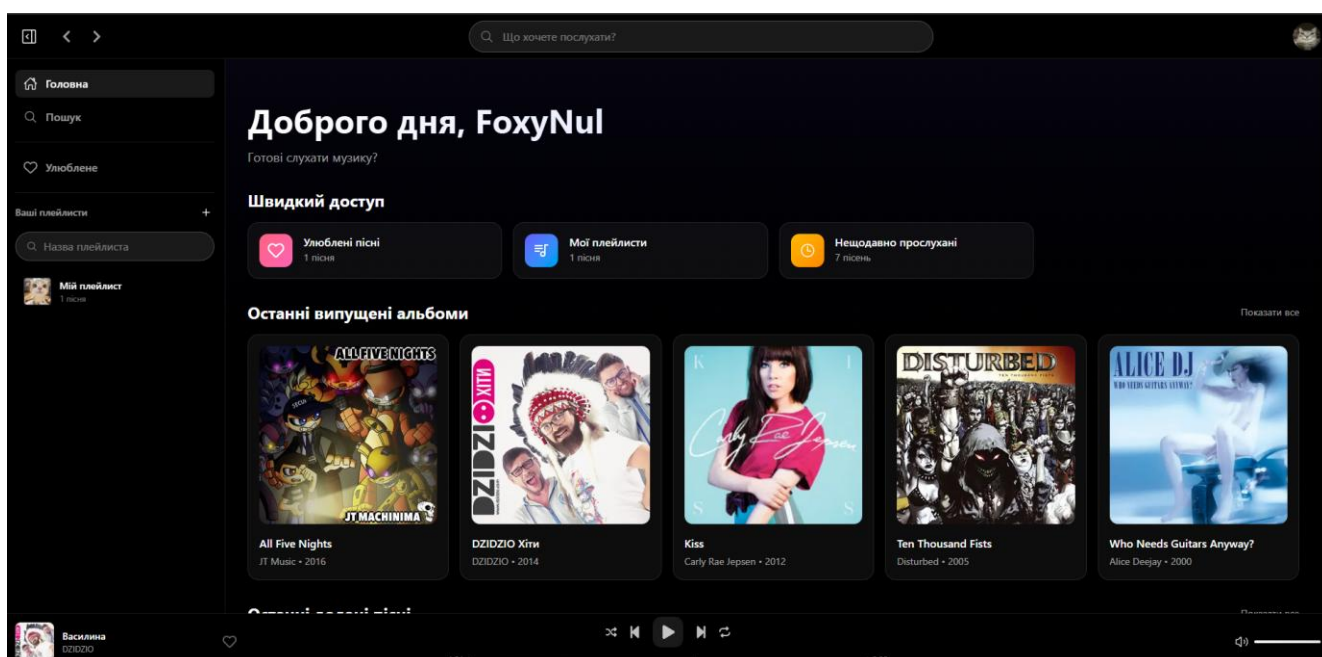


Рис. 3.6. Бібліотека платформи

Верхня частина бібліотеки містить зручну навігацію між сторінками, пошук та аватар профілю. Зліва знаходиться бічна панель, яка є зручною для навігацією між сторінками (елементами) та список плейлистів користувача з функціоналом додавання та пошуку, нижче відображається плеєр для відтворення пісень.

Для створення нового плейлиста потрібно натиснути на іконку плюса, яка відкриває вікно створення плейлиста. Вікно містить форму та має наступні поля для заповнення: назва, опис та видимість, яка надає доступ до цього плейлиста іншим користувачам. На рис. 3.7 зображено вікно створення плейлиста.

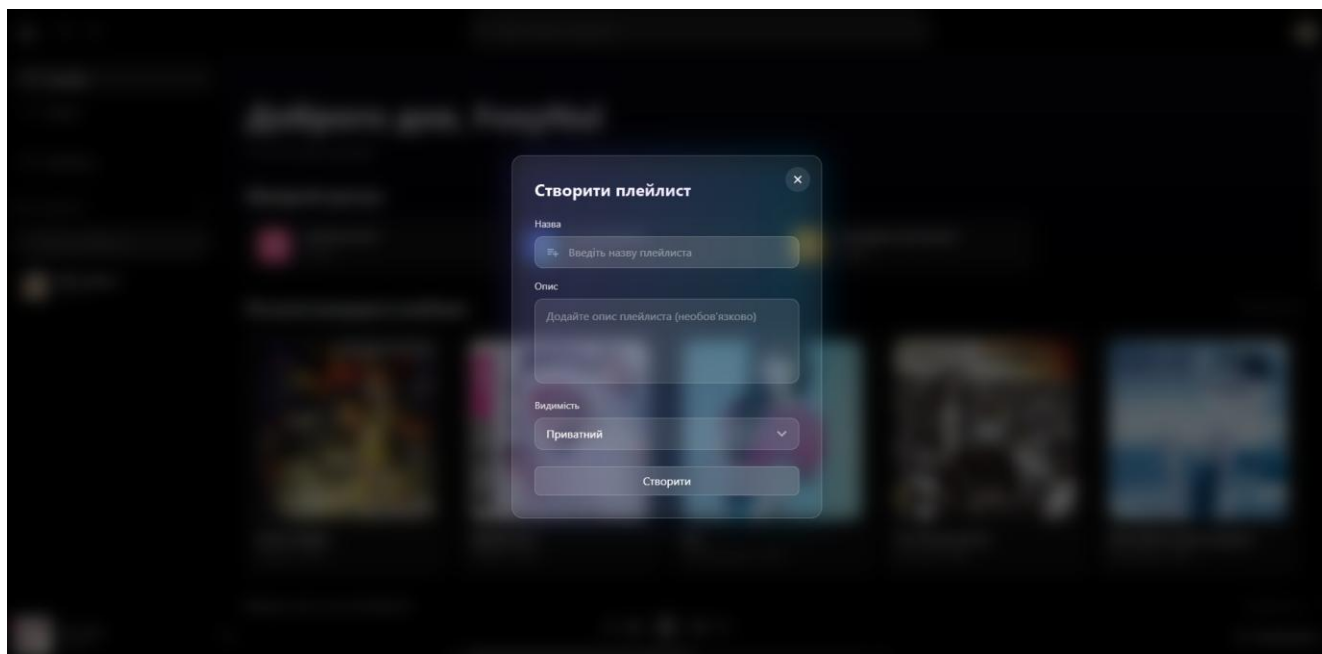


Рис. 3.7. Вікно створення плейлиста

Після заповнення форми та натискання кнопки створення користувачу буде відображене повідомлення про успішне створення плейлиста, а список з плейлистами буде оновлено. На рис. 3.8 наведено повідомлення про успішне створення плейлиста та оновлений список.

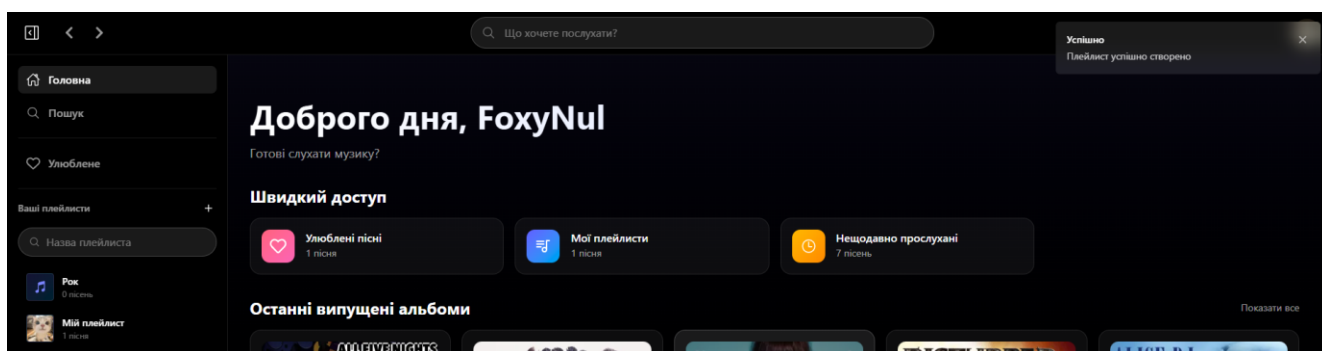


Рис. 3.8. Успішно створений плейлист

Для додавання пісень до плейлиста користувач може використати меню дій на карті пісні або сторінці альбому, пісні, виконавця або жанру. На рис. 3.9 представлено меню дій на картці пісні.

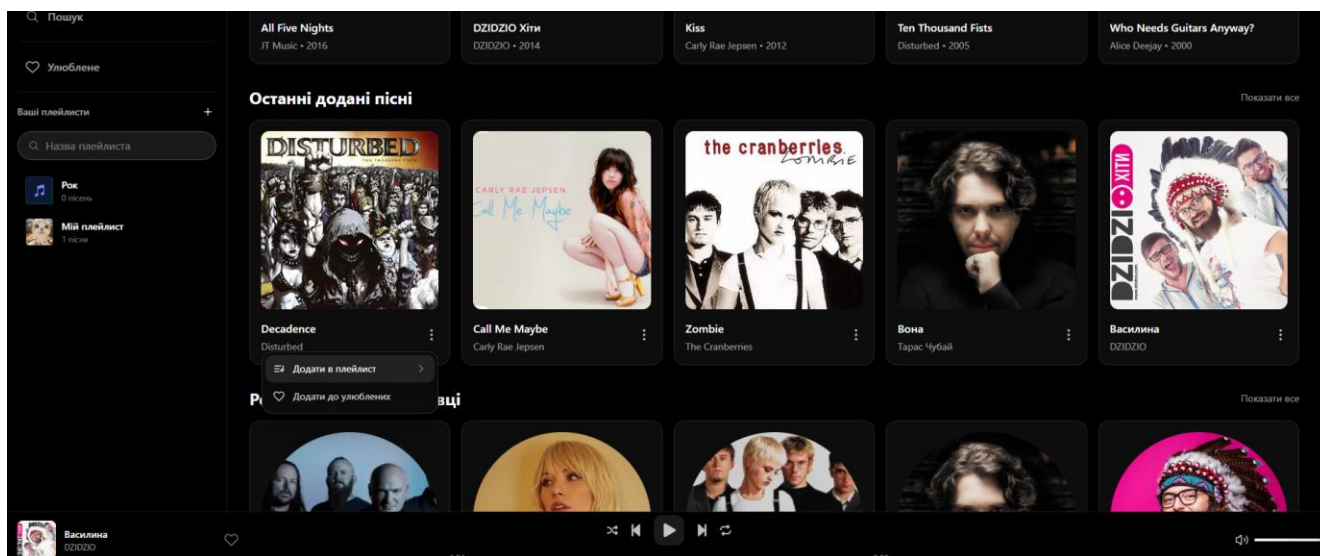


Рис. 3.9. Меню дій (картка пісні)

Після відкриття меню та натискання на елемент меню, який відповідає певному плейлисту, пісню буде додано до вибраного плейлиста. При натисканні на плейлист в списку бічної панелі відкриється сторінка обраного плейлиста. На рис. 3.10 зображено сторінку плейлиста. Для повного перегляду плейлистів у вигляді карток можна скористатися швидким доступом на головній сторінці бібліотеки платформи.

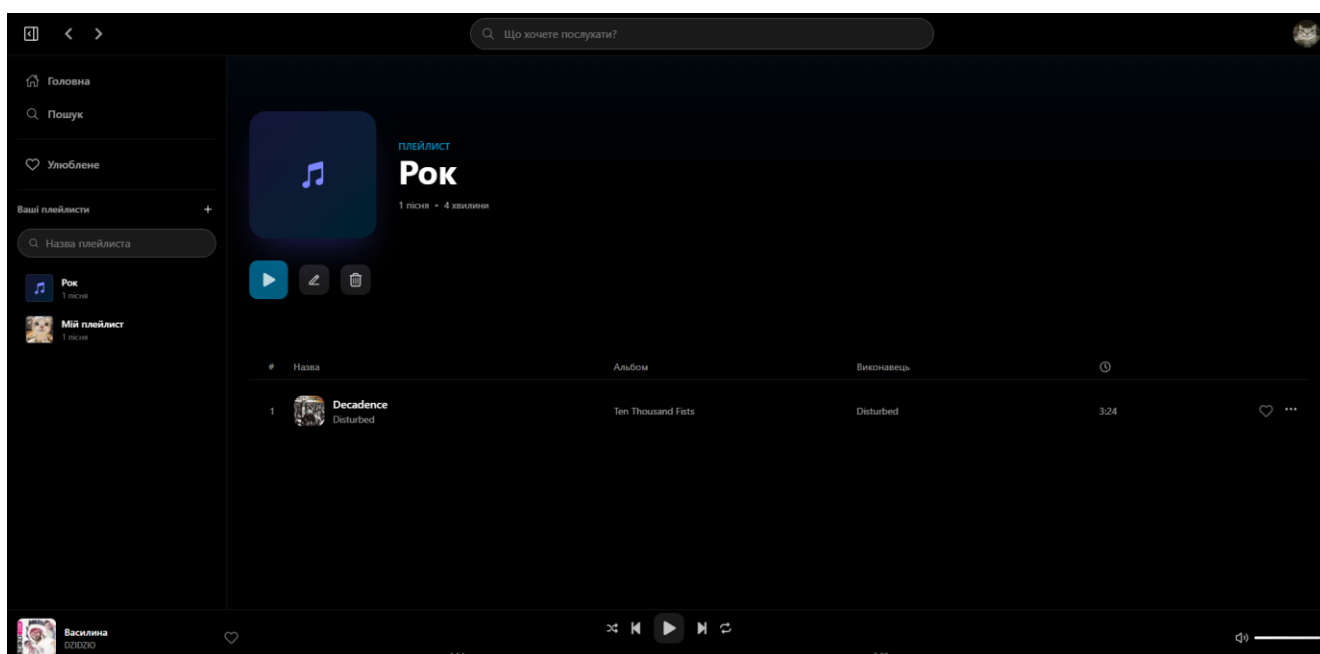


Рис. 3.10. Сторінка плейлиста

На сторінці плейлиста користувач може прослуховувати додані їм пісні, прибирати пісні з плейлиста, редагувати та видаляти плейлист. Під час редагування є можливість завантажити обкладинку плейлиста для швидкої орієнтації по зображенню. На рис. 3.11 представлено завантажену обкладинку плейлиста. Перед видаленням плейлиста користувачу буде відображено діалогове вікно, де потрібно підтвердити або скасувати видалення.

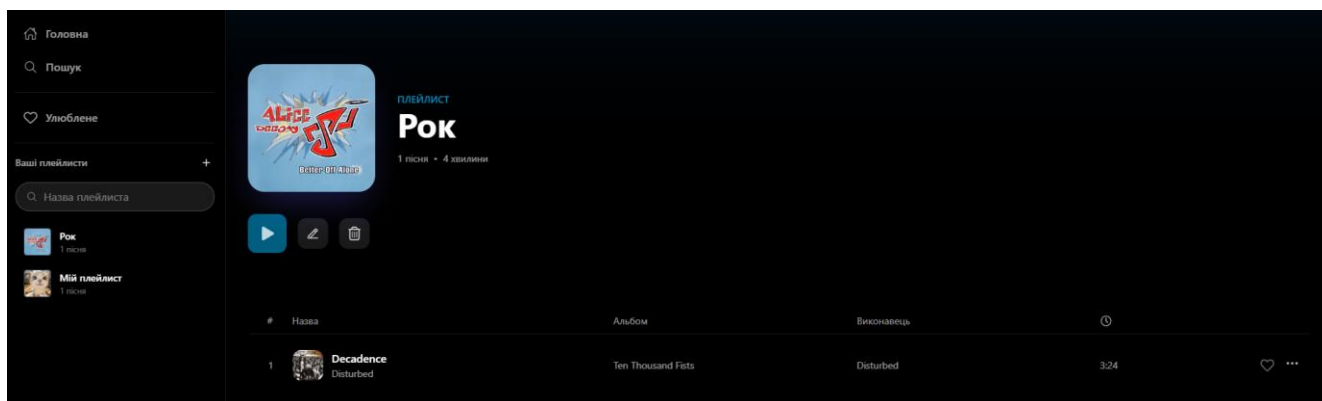


Рис. 3.11. Оновлена обкладинка плейлиста

На рис. 3.12 наведено діалогове вікно видалення плейлиста.

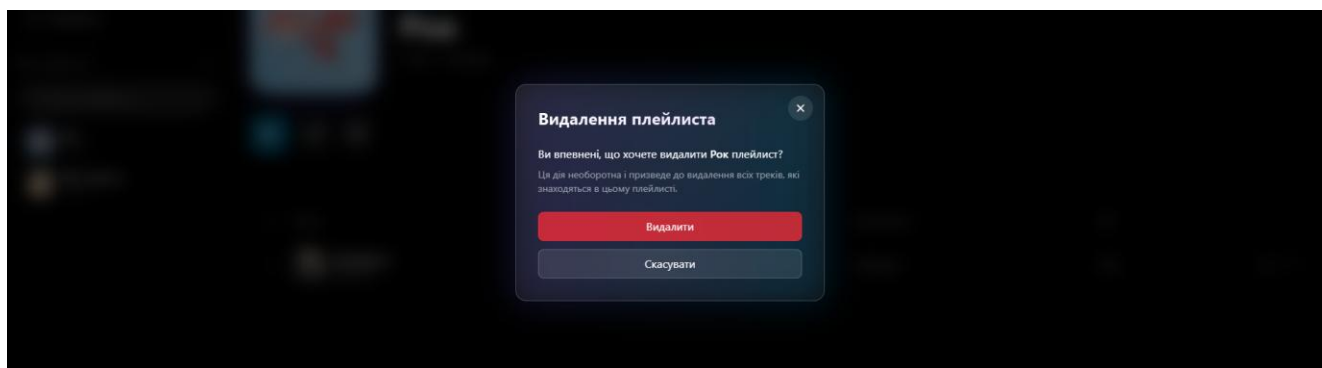


Рис. 3.12. Діалогове вікно видалення плейлиста

Після видалення плейлиста, список на бічній панелі оновиться, користувача буде переадресовано на головну сторінку бібліотеки платформи з відображенням повідомлення про успішне видалення. На рис. 3.13 зображено успішне видалення плейлиста.

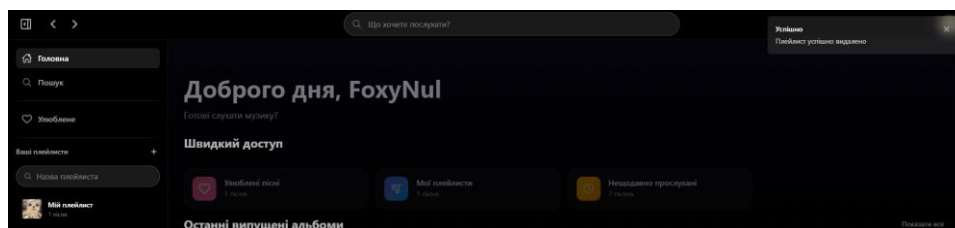


Рис. 3.13. Успішно видалений плейлист

Для відкриття сторінки улюблених пісень використовується навігаційна панель в бічній панелі або кнопка швидкого доступу. На рис. 3.14 представлено сторінку улюблених пісень. Сторінка підтримує тільки видалення пісень доданих до улюблених. Щоби додати пісню до улюблених можна використати меню дій як і для плейлиста або натиснути на кнопку сердечка, яка розташована на панелі плеєра чи в елементі списку пісень. Після натискання пісня буде додана до улюблених, а сердечко зміниться на зафарбоване відповідно. Для видалення також можна натиснути ще раз на сердечко або використати меню дій.

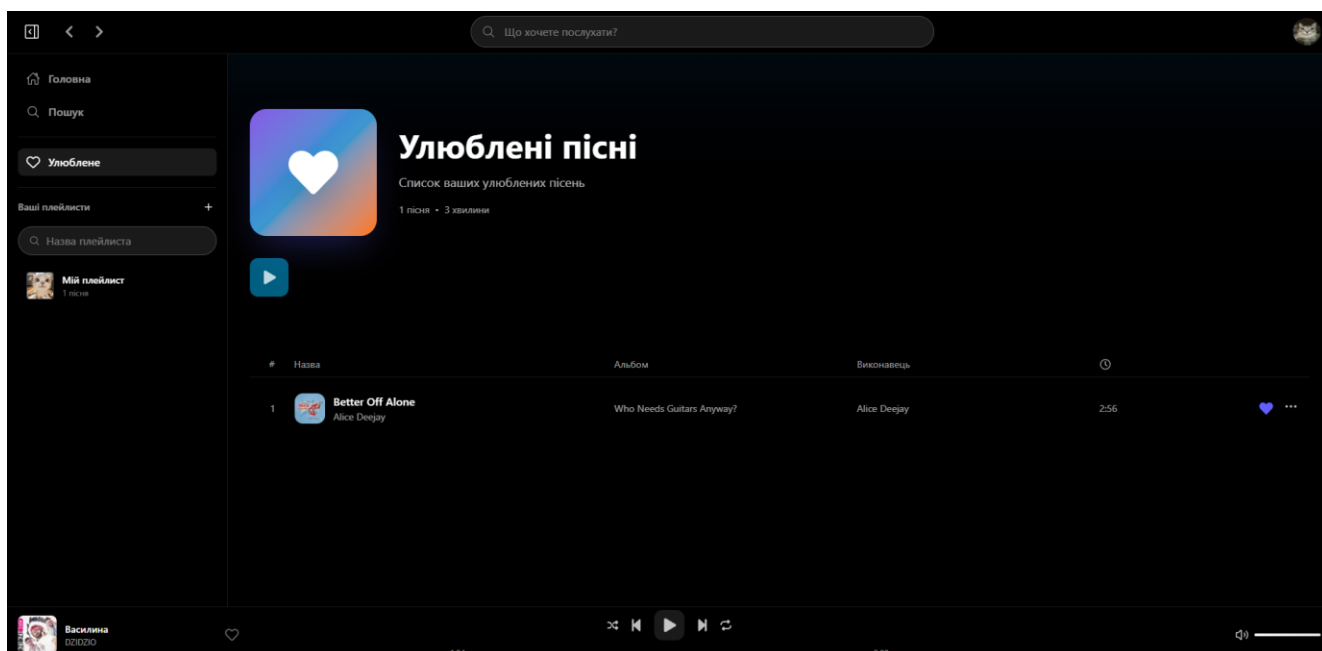


Рис. 3.14. Сторінка улюблених пісень

Для перегляду та прослуховування історії прослуховувань можна скористатися швидким доступ при натисканні кнопки якого відкриється сторінка.

Сторінка містить список всіх прослуханих користувачем пісень протягом всього часу. Кожна пісня додається на початок списку, якщо вона була прослухана протягом 30 секунд. На рис. 3.15 представлено сторінку історії прослуховувань.

Платформа підтримує пошук пісень, альбомів та артистів. Для пошуку чогось конкретного можна скористатися полем вводу у верхній частині панелі бібліотеки платформи, навігацією бокового меню або кнопкою пошуку, яка доступна тільки на мобільній версії. Після введення як мінімум 3 символів почнеться пошук за введеним запитом. Знайдені пісні та альбоми підтримують пряме відтворення з пошуку.

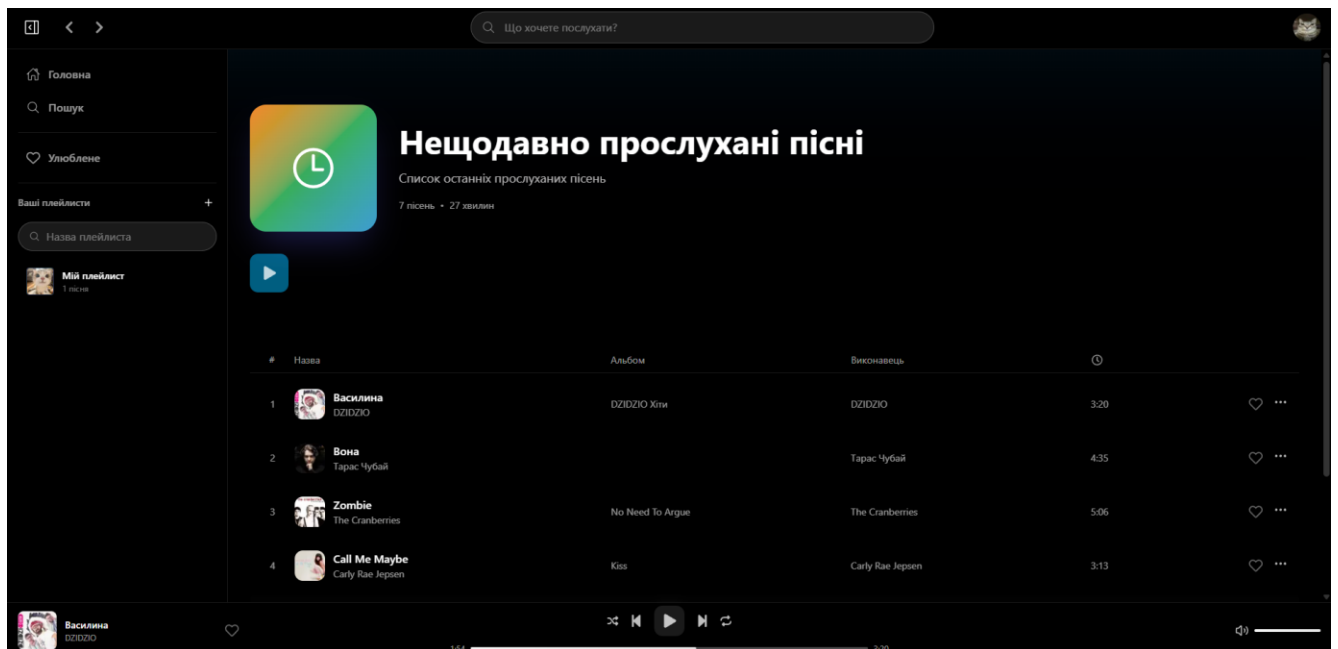


Рис. 3.15. Сторінка історії прослуховувань

На рис. 3.16 наведено приклад пошуку та відтворення пісні.

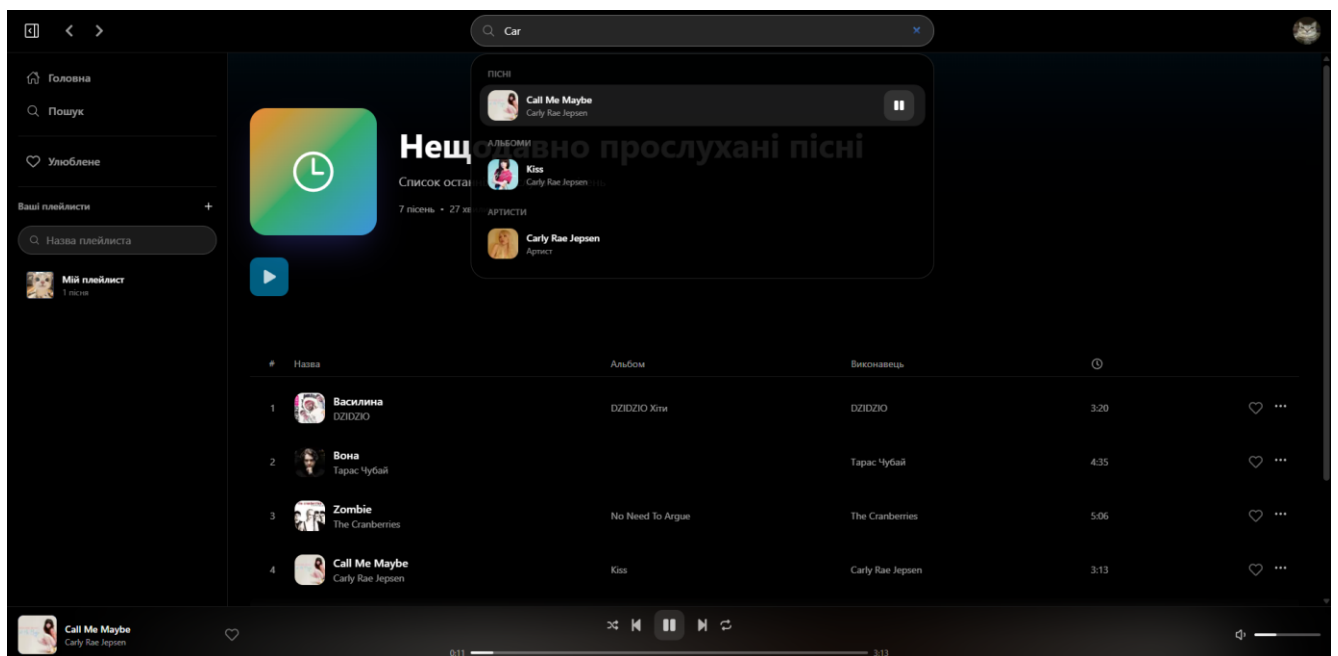


Рис. 3.16. Пошук виконавця

Для відображення більшої кількості альбомів, останніх доданих пісень, виконавців та жанрів потрібно натиснути кнопку показу всього, яка відкриє окрему сторінку під обраний музикальний контент. Сторінки майже ідентичні між собою, але мають різні елементи відображення. Кожна з цих сторінок підтримує нескінченне завантаження наявного контенту за допомогою курсору. На рис. 3.17 зображено сторінку останніх доданих пісень.

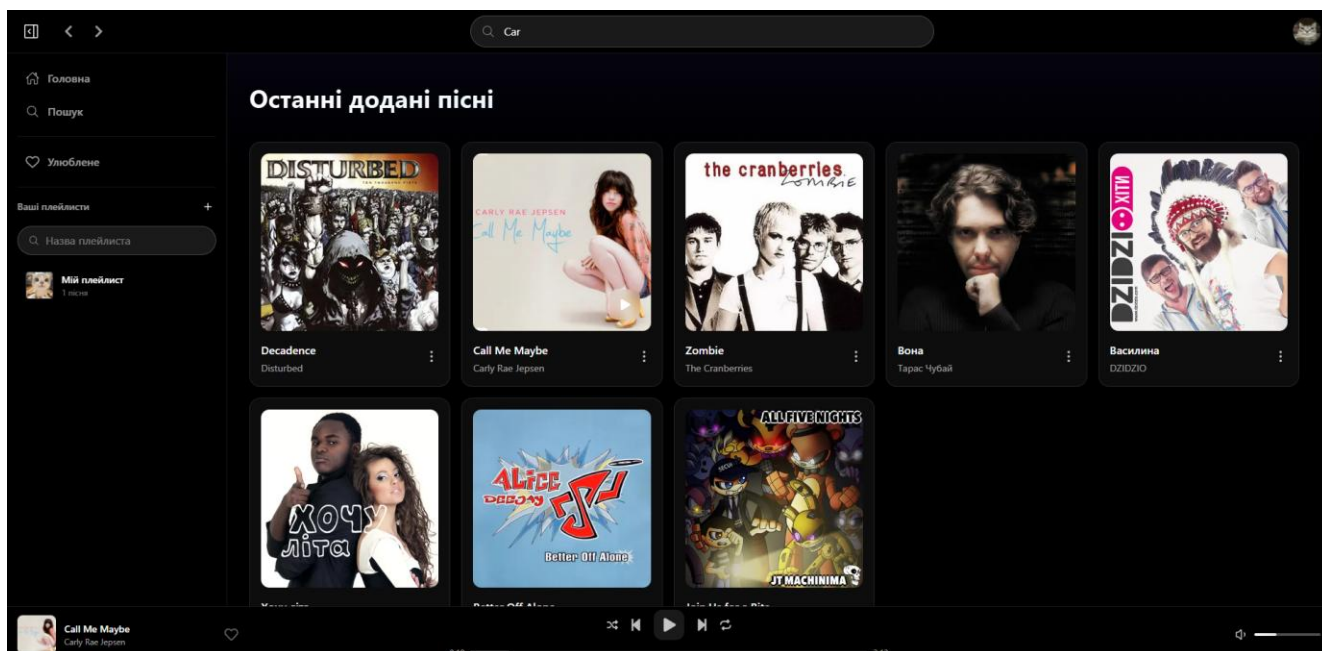


Рис. 3.17. Сторінка останніх доданих пісень

Для перегляду повної інформації про альбом, пісню, виконавця або жанр потрібно натиснути на відповідну картку, яка відкриває сторінку обраного контенту. Кожна сторінка містить список пісень, які також підвантажуються за допомогою курсору. На рис. 3.18 представлено сторінку альбому.

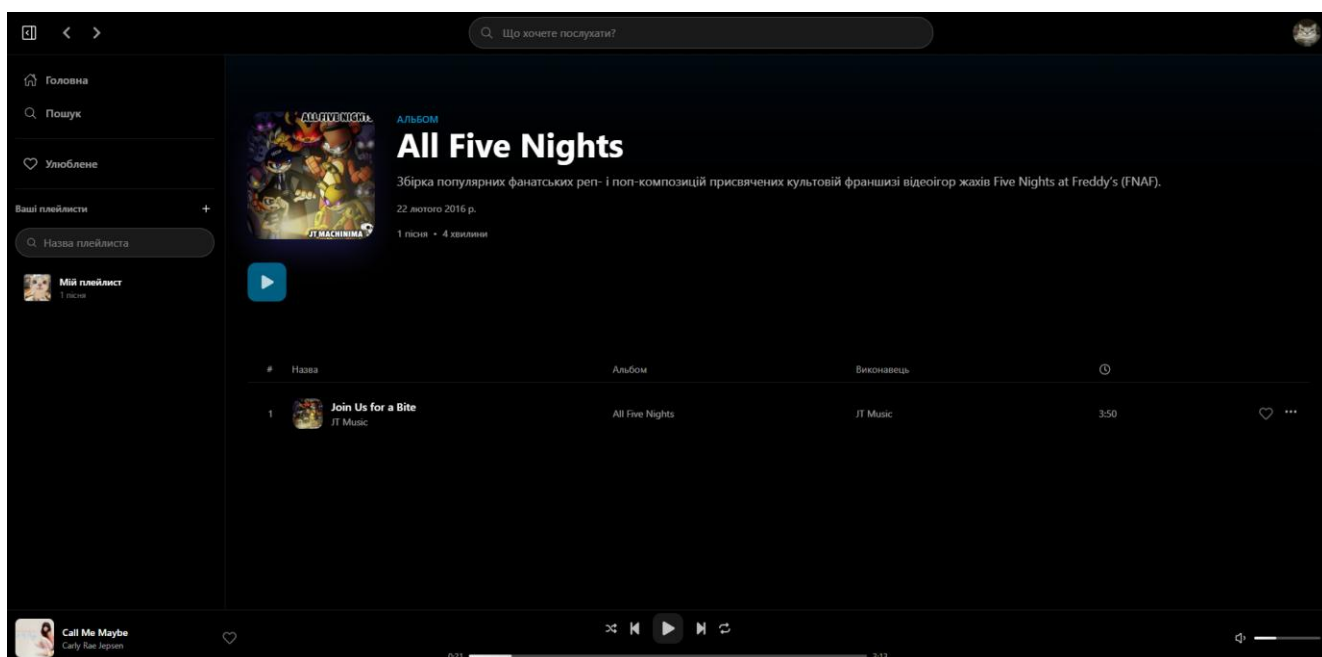


Рис. 3.18. Сторінка альбому

Плеєр платформи підтримує мета відтворення з можливістю синхронізації, наявний функціонал перемішування та повтору пісень. Під час програвання пісні на фоні плеєра використовується ефект програвання музики, який можна вимкнути

за потреби у налаштуваннях доступних у профілі користувача. На рис. 3.19 наведено приклад ефекту під час програвання пісні.

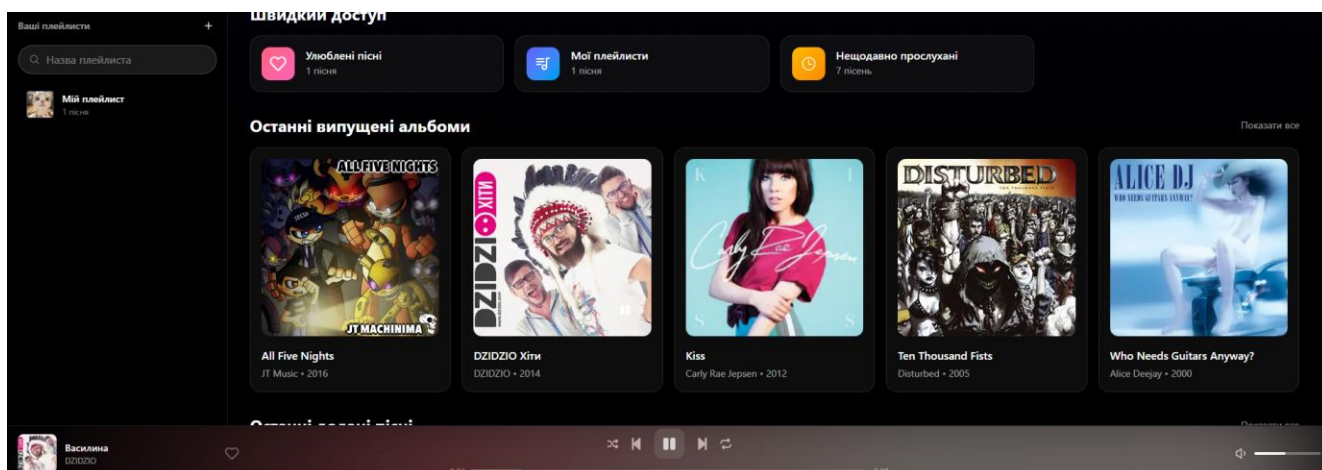


Рис. 3.19. Приклад ефекту під час програвання пісні

Для редагування персональної інформації можна скористатися профілем користувача, який можна відкрити за допомогою натискання аватара профіля розташованого в правій верхній частині бібліотеки платформи. Для редагування потрібно натиснути кнопку користувача з відміткою олівця для того, щоби увімкнути зміну даних. Після повторного натискання дані будуть автоматично оновлені. На рис. 3.20 зображено профіль користувача у вигляді вікна.

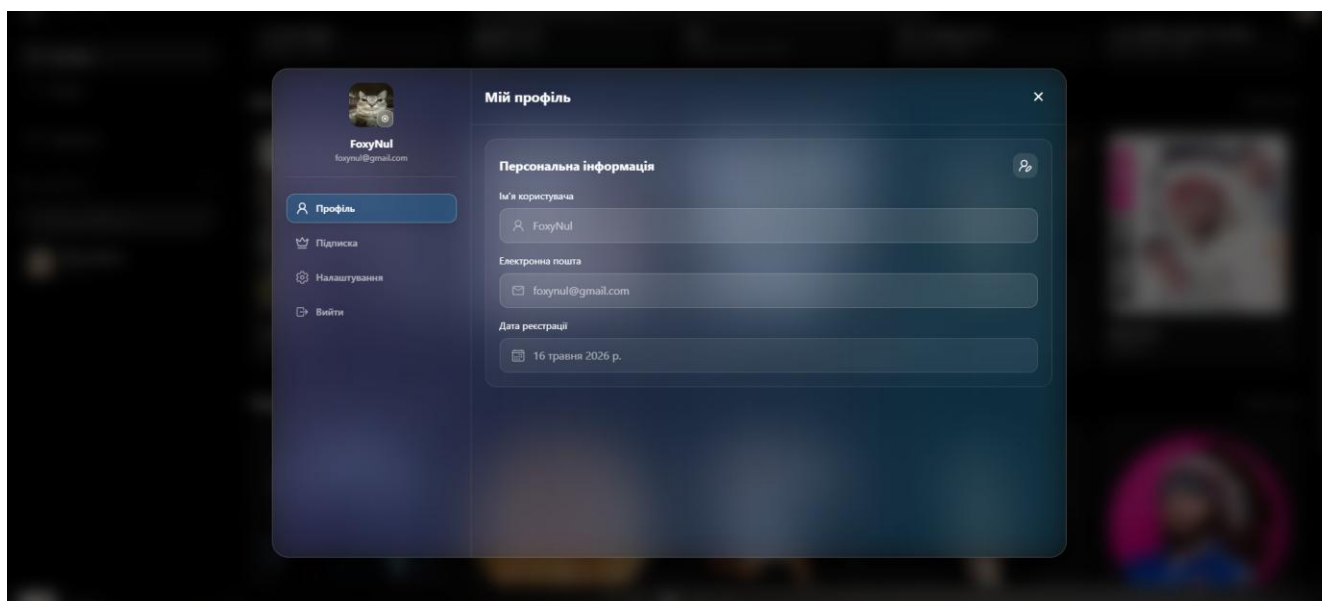


Рис. 3.20. Профіль користувача

Для прослуховування пісень без перешкоджень користувачу потрібно оформити підписку на платформу. Платформа підтримує два плани підписок:

місячна та річна. На рис. 3.21 представлено приклад пропозиції оформлення підписки після прослуховування пісні.

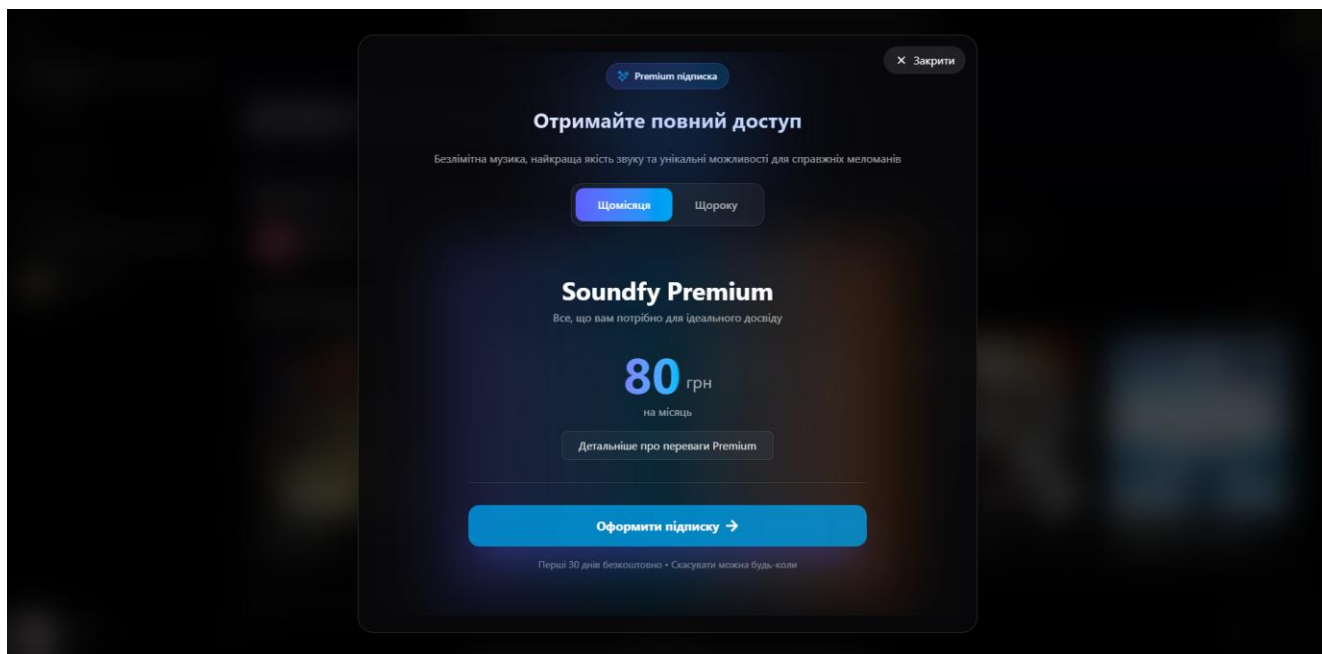


Рис. 3.21. Пропозиція оформлення підписки

Після вибору плану підписки користувач переходить до оформлення, де його перекидає на платіжну систему Stripe. Заповнивши всі дані правильно та оплативши підписку користувача буде знову направлено до бібліотеки платформи. На рис 3.22 наведено приклад оформлення підписки.

Рис. 3.22. Приклад оформлення підписки

Для зміни плану чи скасування підписки користувачу потрібно відкрити профіль та перейти до підписки через навігацію. В цьому вікні відображається

поточна підписка користувача та історія платежів здійснених ним протягом усього періоду. На рис 3.23 зображено вікно активної підписки.

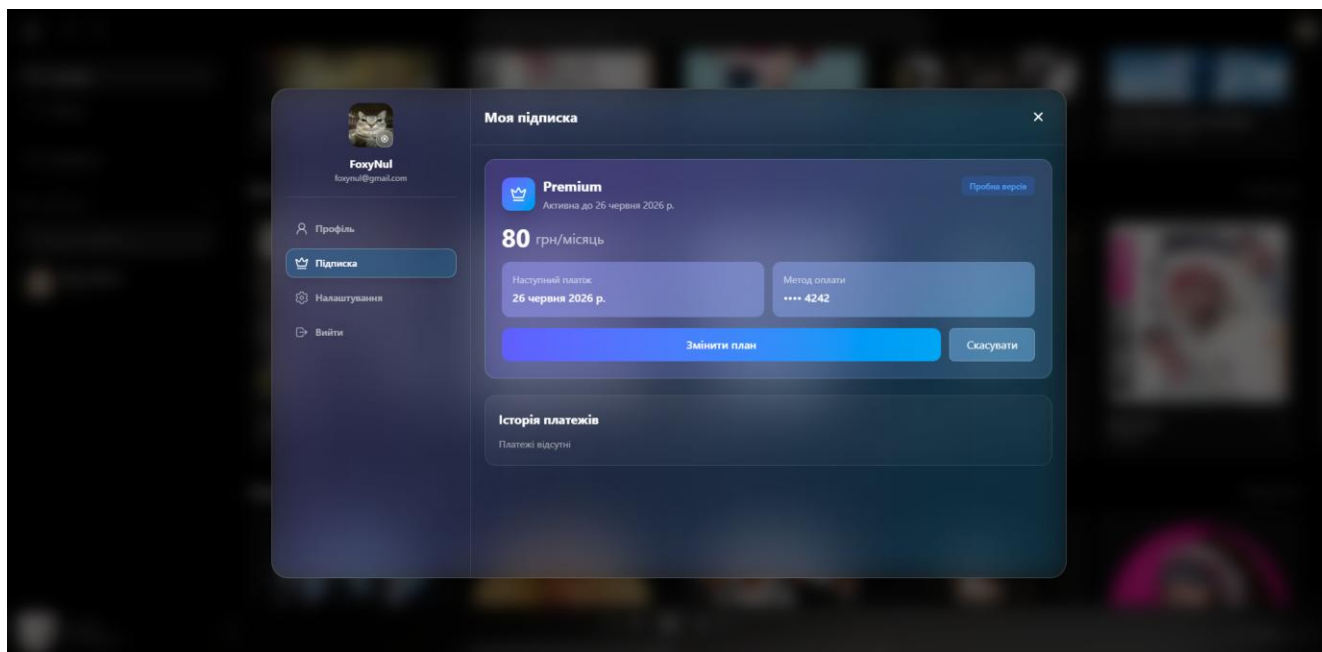


Рис. 3.23. Вікно активної підписки

Для зручності в налаштуваннях користувач може увімкнути збереження стану відтворення та гучності, налаштувати ефекти доступні на платформі. На рис 3.24 представлено вікно налаштувань.

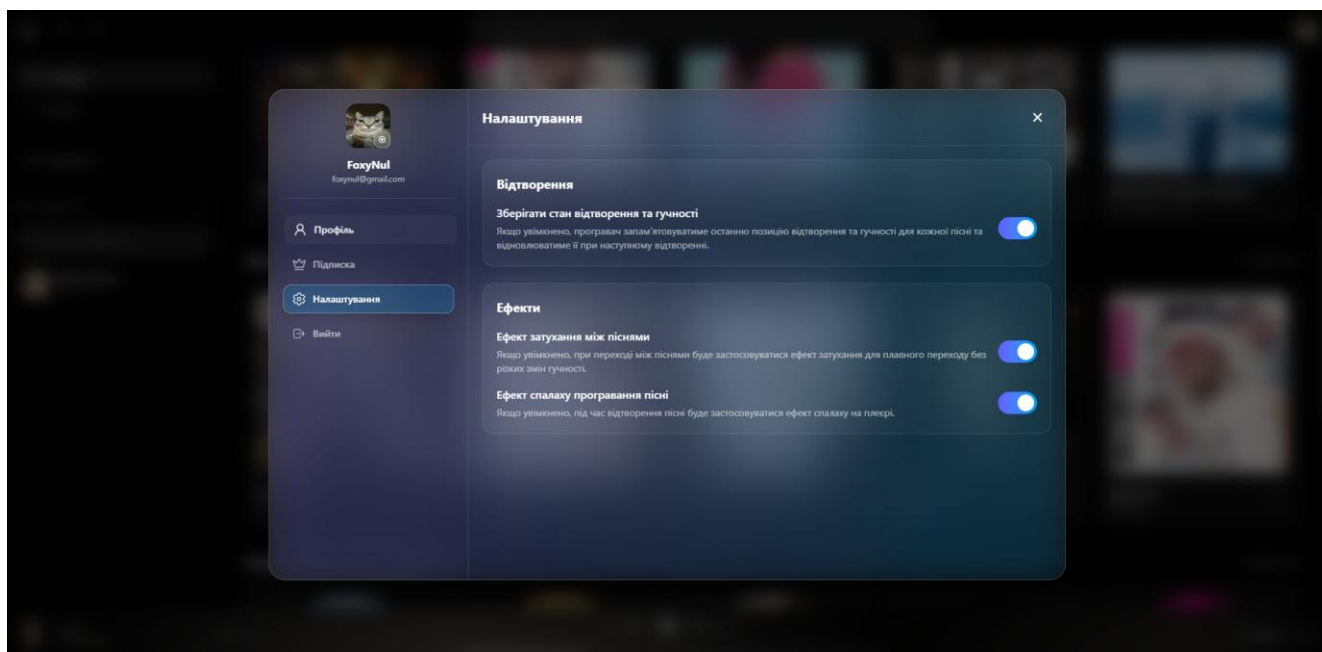


Рис. 3.24. Вікно налаштувань

Для наповнення контенту вебплатформи використовується панель адміністратора. На головній панелі адміністратора можна повернутися назад до

бібліотеки платформи, переглянути кількість наявних даних про альбоми, виконавців, пісень та жанрів. На рис 3.25 наведено сторінку головної панелі адміністратора.

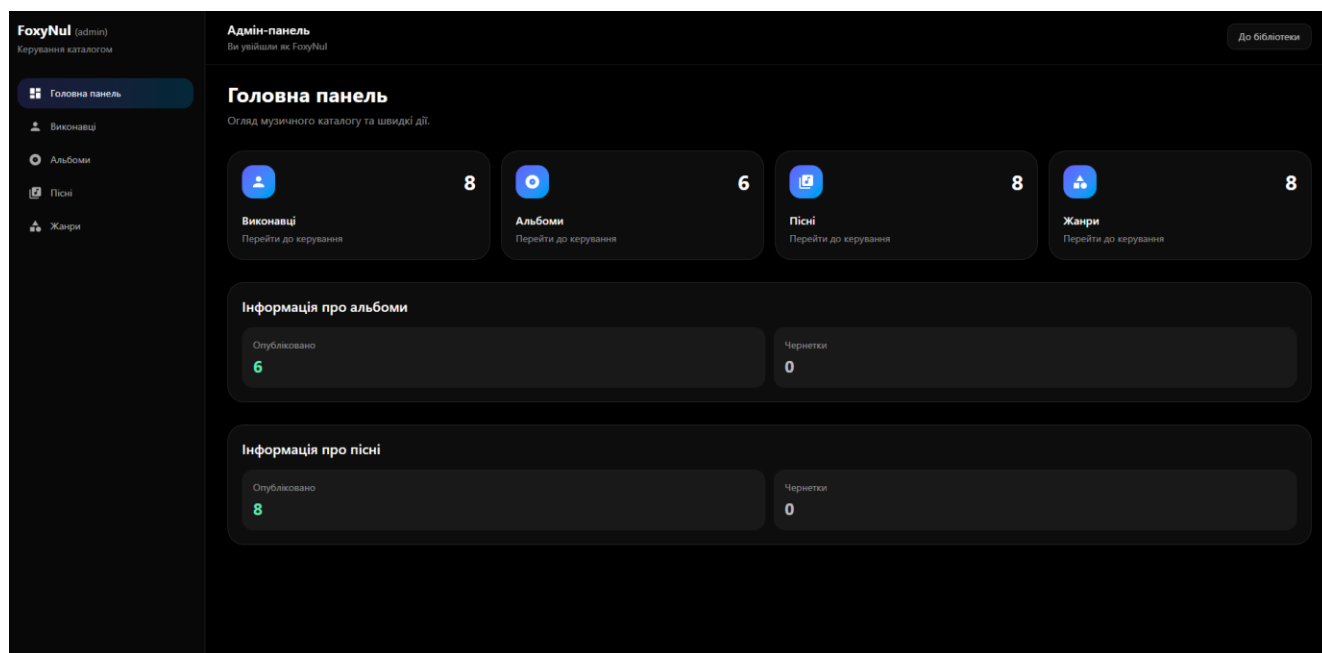


Рис. 3.25. Головна панель адміністратора

Для додавання контенту на платформу потрібно скористатися навігаційним меню яке розташоване зліва або натиснути на кнопку додавання при наведенні на картку контенту. При переході на сторінку певного контенту буде відображено інформацію про контент у вигляді списку. Кожен елемент списку підтримує дію редагування та видалення. У верхній частині сторінки відображається назва секції, пошук та кнопка додавання. На рис. 3.26 зображено сторінку пісень.

Під час додавання, редагування та видалення контенту використовуються схожі вікна з різними полями як для плейлиста, який був описаний вище.

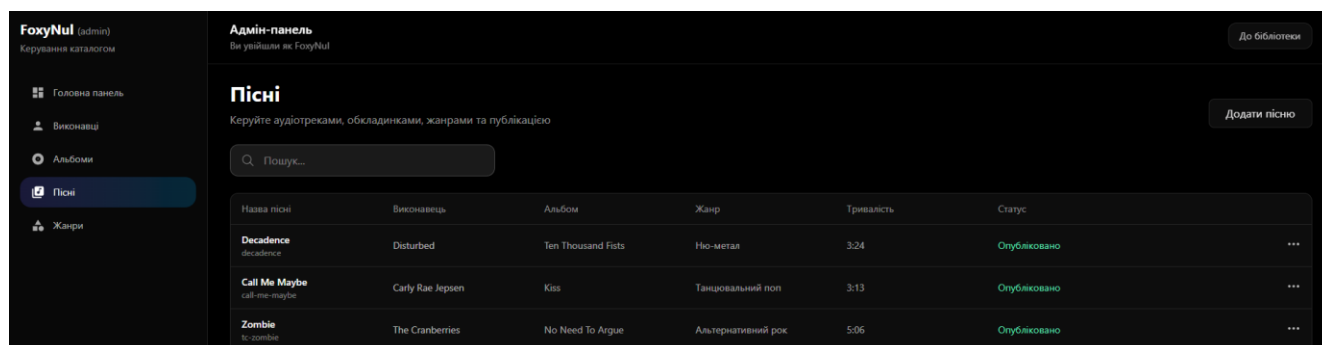


Рис. 3.26. Сторінка пісень

Вебплатформа підтримує адаптивність на різних пристроях. Для мобільних пристроїв були створені деякі окремі компоненти з метою зручності використання, наприклад, аудіо плеєр та деяке меню в профілі користувача. На рис. 3.27 представлено головну сторінку, бібліотеку платформи, сторінку улюблених пісень, сторінку останніх випущених альбомів, профіль користувача та музичний плеєр.

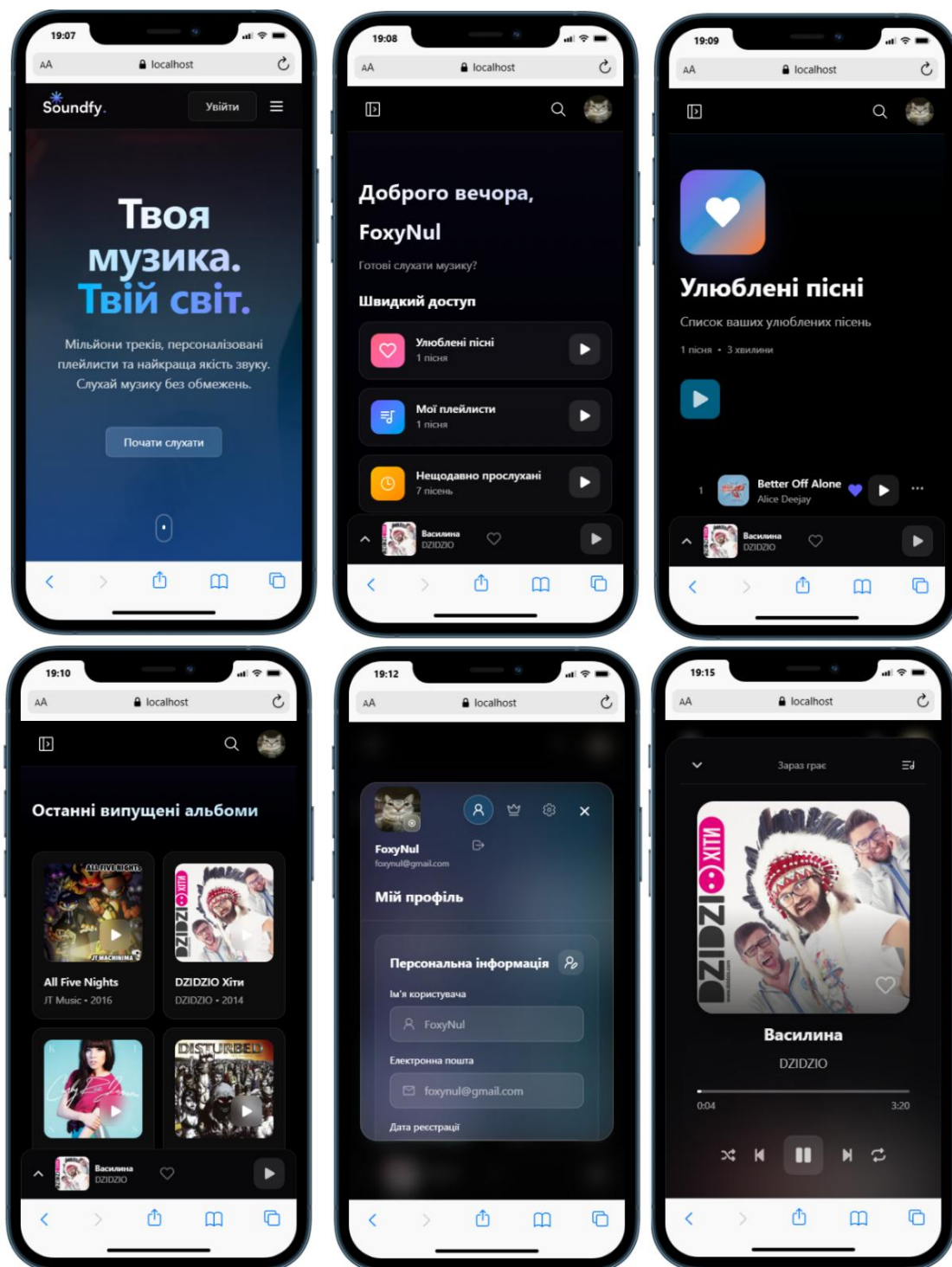


Рис. 3.27. Адаптивний дизайн сторінок

3.3 Тестування вебплатформи

Тестування вебплатформи для прослуховування музики є складовою процесу розробки програмного забезпечення. Під час тестування перевіряється робота основних функцій системи, сумісність з різними пристроями та стабільність відтворення аудіо. Додатково оцінюється коректність роботи інтерфейсу користувача та серверної частини платформи.

Для перевірки роботи серверної частини вебплатформи було використано модульне тестування [21].

Модульне тестування є методом перевірки якості програмного забезпечення під час якого модулі системи перевіряються незалежно від інших компонентів. Додатково через тестування можна визначити правильність роботи функцій, сервісів і методів обробки даних на ранніх етапах розробки. Основною ціллю є перевірка логіки роботи окремих частин системи та коректності отриманих результатів.

Для реалізації модульного тестування серверної частини вебплатформи було використано вбудовані засоби фреймворку NestJS із використанням тестового середовища Jest, яке входить до стандартного набору інструментів фреймворку. Під час тестування перевірялася робота сервісів, контролерів і методів обробки даних, оцінювалася правильність виконання бізнес-логіки, обробки виняткових ситуацій, взаємодії між окремими компонентами системи та коректність повернення результатів API-запитів. Для забезпечення незалежності тестів використовувалися механізми ізоляції компонентів та імітації (mock) зовнішніх залежностей для перевірки логіки роботи кожного модуля окремо без необхідності взаємодії з реальною базою даних або сторонніми сервісами.

Під час проведення тестування було перевірено коректність роботи механізмів автентифікації та авторизації користувачів, обробки пісень, роботи з плейлистами, історією прослуховувань, підписками та інших сервісів, які реалізують основний функціонал вебплатформи. Особлива увага приділялася

перевірці граничних випадків, обробці некоректних вхідних даних та правильному формуванню повідомлень про помилки відповідно до реалізованої бізнес-логіки.

```
PASS src/modules/genres/genres.controller.spec.ts
PASS src/modules/users/users.service.spec.ts
PASS src/modules/refresh-tokens/refresh-tokens.service.spec.ts
PASS src/modules/playlists/playlists.controller.spec.ts
PASS src/modules/upload/upload.service.spec.ts
PASS src/modules/admin-stats/admin-stats.controller.spec.ts
PASS src/modules/tracks/tracks.controller.spec.ts
PASS src/modules/health/health.controller.spec.ts
PASS src/modules/admin-albums/admin-albums.service.spec.ts
PASS src/modules/users/users.controller.spec.ts
PASS src/modules/admin-stats/admin-stats.service.spec.ts
PASS src/modules/history/history.controller.spec.ts
PASS src/modules/playlists/playlists.service.spec.ts
PASS src/modules/admin-artists/admin-artists.service.spec.ts

Test Suites: 41 passed, 41 total
Tests: 225 passed, 225 total
Snapshots: 0 total
Time: 17.453 s, estimated 19 s
Ran all test suites.
```

Рис. 3.28. Результати модульного тестування

За результатами виконання всіх модульних тестів можна зробити висновок, що основні програмні модулі системи функціонують коректно та відповідають встановленим функціональним вимогам. Під час виконання тестів критичних помилок або збоїв у роботі окремих компонентів виявлено не було. Отримані результати підтверджують правильність реалізації бізнес-логіки, стабільність роботи серверної частини вебплатформи та готовність системи до подальшого інтеграційного тестування й експлуатації. Використання автоматизованого модульного тестування є основою для безпечного внесення змін до програмного коду в процесі подальшого розвитку інформаційної системи.

Для перевірки візуального вигляду та продуктивності реалізованої вебплатформи було використано інструмент PageSpeed Insights [22].

PageSpeed Insights є інструментом, розробленим компанією Google, який призначений для аналізу швидкості завантаження вебсторінок, оцінювання стабільності інтерфейсу та визначення рівня оптимізації клієнтської частини застосунку. Аналіз проводиться на основі методології Lighthouse, яка дозволяє оцінити не лише продуктивність вебресурсу, а й доступність, відповідність сучасним вебстандартам та ефективність використання ресурсів браузера.

Під час тестування інструмент аналізує ключові показники продуктивності вебзастосунку, серед яких швидкість завантаження сторінки, час відображення основного вмісту (Largest Contentful Paint), швидкість першого відображення

контенту (First Contentful Paint), стабільність розташування елементів інтерфейсу (Cumulative Layout Shift) та час готовності сторінки до взаємодії з користувачем (Interaction to Next Paint). Крім цього, виконується перевірка ефективності оптимізації зображень, використання вебшрифтів, JavaScript-коду, CSS-ресурсів, механізмів кешування та інших компонентів, що безпосередньо впливають на швидкодію клієнтської частини вебплатформи.

Тестування мобільної та комп'ютерної версій вебплатформи виконувалося у локальному середовищі розробки з метою оцінки продуктивності застосунку без впливу зовнішніх факторів, пов'язаних із розгортанням на віддаленому сервері або особливостями мережевої інфраструктури. На рис. 3.29 наведено результати тестування для мобільної та комп'ютерної версій вебплатформи.

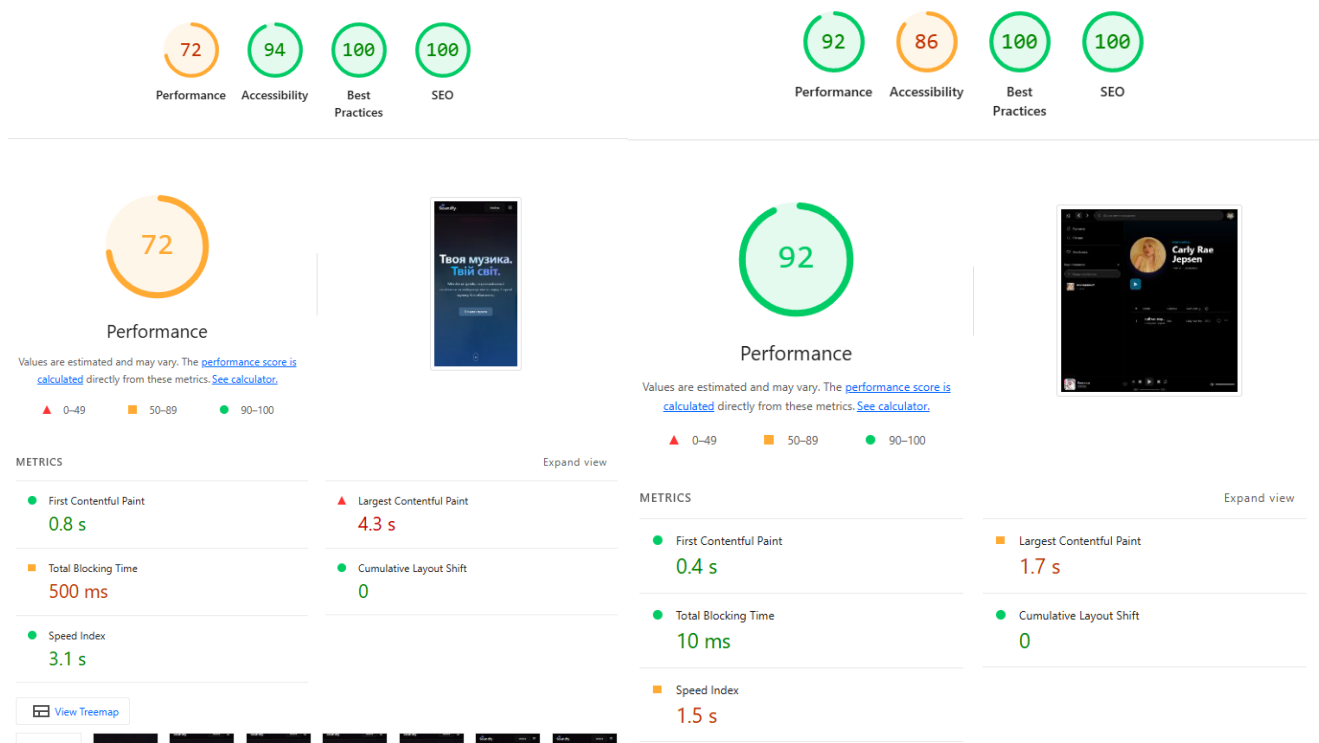


Рис. 3.29. Результати тестування

Отримані результати показали достатній рівень оптимізації клієнтської частини застосунку, стабільне відображення інтерфейсу користувача та коректну роботу основних функцій вебплатформи на різних типах пристроїв. Оптимізація на

мобільних пристроях не найгірша, але виправляється шляхом доставки зображень на клієнт в меншому розмірі, які потребують браузерні елементи.

Висновки до третього розділу

При проектуванні вебплатформи було поетапно розгорнуто веб платформу. Було розгорнуто клієнтську і серверну частину, налаштовано базу даних для взаємодії з сервером, інтегровано платіжний сервіс. Після конфігурації та збірки усіх частин системи було отримано стабільну та безперебійну робочу вебплатформу.

Було описано структуру інтерфейсу користувача з основними функціональними можливостями та сценаріями взаємодії користувачів, проаналізовано доступність вебплатформи на різних типах пристроїв.

Проведене модульне тестування з перевіркою на правильність роботи функціоналу вебплатформи, протестовано продуктивність системи. Отримані результати підтвердили що застосунок відповідає усім сучасними вимогам щодо швидкодії, оптимізації та продуктивності.

ВИСНОВКИ

У процесі виконання випускної кваліфікаційної роботи бакалавра було розроблено функціональну вебплатформу для прослуховування музики протягом етапу дослідження предметної області, проєктування архітектури системи та технічної реалізації. Проведено аналіз потреб основних груп користувачів, досліджено сучасні музичні вебплатформи та визначено основні функціональні можливості майбутньої платформи. Сформовані вимоги стали основою для вибору архітектури системи, технологічного стеку та інструментальних засобів розробки з урахуванням потреб у надійності, масштабованості та зручності взаємодії з платформою.

Під час архітектурного проєктування вебплатформи було сформовано логічну структуру системи беручи до уваги взаємодію користувачів відповідно до їхніх ролей, особливостей поведінки та типових сценаріїв використання. Сформовано об'єктно-орієнтовану модель застосунку з основними сутностями системи, їхніми атрибутами та взаємозв'язками між ними. Спроектовано та реалізовано базу даних у середовищі PostgreSQL для зберігання основної інформації та відображення її на платформі, проведено опис реалізації функціональних можливостей вебплатформи.

Використано клієнт-серверну та багатошарову архітектури, які гарантують стабільність, швидкодію та розширюваність в плані нового функціоналу. Використано шифрування аудіо через технологію HLS та інтегровано модель підписки на платформі за допомогою платіжної системи Stripe. Для оцінювання надійності та ефективності вебплатформи було проведено модульне та продуктивне тестування, результати якого підтвердили правильність та стабільність функціонування системи через тести, а також раціональне використання ресурсів і відповідність встановленим вимогам щодо продуктивності.

У результаті було розроблено сучасну, масштабовану вебплатформу для прослуховування музики, яка відповідає потребам користувачів та надає зручний доступ до аудіоконтенту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Spotify. URL: <https://open.spotify.com/> (дата звернення: 26.06.2026).
2. YouTube Music. URL: <https://music.youtube.com/> (дата звернення: 26.06.2026).
3. Apple Music. URL: <https://music.apple.com/> (дата звернення: 26.06.2026).
4. Deezer. URL: <https://www.deezer.com/> (дата звернення: 26.06.2026).
5. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol : O'Reilly Media, 2021. 616 p.
6. Client-Server Architecture – System Design. GeeksforGeeks. URL: <https://www.geeksforgeeks.org/system-design/client-server-architecture-system-design/> (дата звернення: 26.06.2026).
7. Monolithic Architecture – System Design. GeeksforGeeks. URL: <https://www.geeksforgeeks.org/system-design/monolithic-architecture-system-design/> (дата звернення: 26.06.2026).
8. Multitiered Architectures in Distributed System. GeeksforGeeks. URL: <https://www.geeksforgeeks.org/system-design/multitiered-architectures-in-distributed-system/> (дата звернення: 26.06.2026).
9. Node.js. URL: <https://nodejs.org/> (дата звернення: 26.06.2026).
10. TypeScript: JavaScript With Syntax For Types. URL: <https://www.typescriptlang.org/> (дата звернення: 26.06.2026).
11. NestJS. A progressive Node.js framework. URL: <https://nestjs.com/> (дата звернення: 26.06.2026).
12. PostgreSQL. The World's Most Advanced Open Source Relational Database. URL: <https://www.postgresql.org/> (дата звернення: 26.06.2026).
13. Stripe. URL: <https://stripe.com/> (дата звернення: 26.06.2026).
14. HTTP Live Streaming. Apple Developer Documentation. URL: <https://developer.apple.com/documentation/http-live-streaming> (дата звернення: 26.06.2026).
15. Riva M. Real-World Next.js: Build Scalable, High-Performance, and Modern Web Applications Using Next.js, the React Framework for Production. Birmingham : Packt Publishing, 2022. 366 p.

16. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. 2nd ed. Sebastopol : O'Reilly Media, 2020. 310 p.
17. Tailwind CSS. Rapidly Build Modern Websites Without Ever Leaving Your HTML. URL: <https://tailwindcss.com/> (дата звернення: 26.06.2026).
18. Motion. JavaScript Animation Library for React and JavaScript. URL: <https://motion.dev/> (дата звернення: 26.06.2026).
19. FFmpeg. URL: <https://ffmpeg.org/> (дата звернення: 26.06.2026).
20. HLS.js. GitHub Repository. URL: <https://github.com/video-dev/hls.js> (дата звернення: 26.06.2026).
21. Module Testing. GeeksforGeeks. URL: <https://www.geeksforgeeks.org/software-testing/module-testing/> (дата звернення: 26.06.2026).
22. PageSpeed Insights. URL: <https://pagespeed.web.dev/> (дата звернення: 26.06.2026).
23. Agnihotri K., Dahale P. React Development Using TypeScript: Modern Web App Development Using Advanced React Techniques. New Delhi : BPB Publications, 2024. 332 p.
24. Baumgartner S. TypeScript Cookbook: Real-World Type-Level Programming. Sebastopol : O'Reilly Media, 2023. 419 p.
25. Booz R. Introduction to PostgreSQL for the Data Professional. Cambridge : Redgate Software, 2025. 347 p.
26. Linjanja P. Scalable Application Development with NestJS: Leverage REST, GraphQL, Microservices, Testing, and Deployment for Seamless Growth. Birmingham : Packt Publishing, 2025. 612 p.
27. Cherny B. Programming TypeScript: Making Your JavaScript Applications Scale. Sebastopol : O'Reilly Media, 2019. 324 p.
28. Staiano F. Designing and Prototyping Interfaces with Figma. 3rd ed. Birmingham : Packt Publishing, 2026. 452 p.
29. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. Sebastopol : O'Reilly Media, 2020. 432 p.

30. McLean J. Quick Start Guide to FFmpeg: Learn to Use the Open Source Multimedia-Processing Tool Like a Pro. Berkeley : Apress, 2023. 222 p.

					ИПЗ.КР.Б-121-26-ИПЗ	
						69

ДОДАТКИ

					ІПЗ.КР.Б-121-26-ІПЗ	
						70

```

import { Injectable } from '@nestjs/common';
import { ConfigService } from '@nestjs/config';
import { HlsConversionResult } from './hls.types';
import { randomBytes, randomUUID } from 'crypto';
import { join } from 'path';
import { spawn } from 'child_process';
import { mkdir, writeFile, unlink } from 'fs/promises';
import { existsSync } from 'fs';

@Injectable()
export class HlsService {
  constructor(private readonly configService: ConfigService) {}

  async convertAudioToHls(path: string): Promise<HlsConversionResult> {
    const hlsId = randomUUID();

    const rootDir = this.configService.get<string>('ST_ROOT_DIR') || 'storage';
    const privateDir = `${rootDir}/private`;
    const relativeDir = `tracks/hls/${hlsId}`;
    const outputDir = join(process.cwd(), privateDir, relativeDir);

    await mkdir(outputDir, { recursive: true });

    const manifestPath = join(outputDir, 'master.m3u8');
    const segmentPattern = join(outputDir, 'segment_%03d.ts');

    const keyPath = join(outputDir, 'enc.key');
    const keyInfoPath = join(outputDir, 'enc.keyinfo');

    const encryptionKey = randomBytes(16);

    await writeFile(keyPath, encryptionKey);

    await writeFile(keyInfoPath, ['enc.key', keyPath].join('\n'));

    await this.runFfmpeg([
      '-i',
      path,

```

```

        '-vn',
        '-c:a',
        'aac',
        '-b:a',
        '128k',

        '-hls_time',
        '6',
        '-hls_playlist_type',
        'vod',
        '-hls_key_info_file',
        keyInfoPath,
        '-hls_segment_filename',
        segmentPattern,

        manifestPath,
    ]);

    await unlink(keyInfoPath).catch(() => undefined);

    return {
        hlsDirectoryKey: `private/${relativeDir}`,
        hlsManifestKey: `private/${relativeDir}/master.m3u8`,
    };
}

private runFfmpeg(args: string[]) {
    return new Promise<void>((resolve, reject) => {
        const ffmpegPath = this.getFfmpegPath();

        const ffmpeg = spawn(ffmpegPath, ['-y', ...args]);

        ffmpeg.stderr.on('data', () => {});

        ffmpeg.on('close', (code) => {
            if (code === 0) return resolve();
            reject(new Error(`FFmpeg failed with code ${code}`));
        });
    });
}

```

```

private getFfmpegPath(): string {
    const customPath = this.configService.get<string>('FFMPEG_PATH');

    if (customPath) {
        return customPath;
    }

    const localFfmpegPath = join(process.cwd(), 'bin', 'ffmpeg.exe');

    if (existsSync(localFfmpegPath)) {
        return localFfmpegPath;
    }

    return 'ffmpeg';
}
}

```

```

import { Controller, Get, Param, Res, UseGuards } from '@nestjs/common';
import { JwtAuthGuard } from '../auth/guards/jwt-auth.guard';
import { StreamService } from './stream.service';
import type { Response } from 'express';

@Controller('stream/tracks')
@UseGuards(JwtAuthGuard)
export class StreamController {
  constructor(private readonly streamService: StreamService) {}

  @Get('/:trackId/master.m3u8')
  getManifest(@Param('trackId') trackId: string, @Res() res: Response) {
    return this.streamService.sendManifest(trackId, res);
  }

  @Get('/:trackId/segments/:segment')
  getSegment(
    @Param('trackId') trackId: string,
    @Param('segment') segment: string,
    @Res() res: Response,
  ) {
    return this.streamService.sendSegment(trackId, segment, res);
  }

  @Get('/:trackId/key')
  getKey(@Param('trackId') trackId: string, @Res() res: Response) {
    return this.streamService.sendKey(trackId, res);
  }
}

import {
  BadRequestException,
  Injectable,
  NotFoundException,
} from '@nestjs/common';
import { ConfigService } from '@nestjs/config';
import { PrismaService } from '../prisma/prisma.service';
import { readFile } from 'fs/promises';
import { join, normalize } from 'path';

```

```

import { Response } from 'express';

@Injectable()
export class StreamService {
  constructor(
    private readonly prisma: PrismaService,
    private readonly configService: ConfigService,
  ) {}

  async sendManifest(trackId: string, res: Response) {
    const track = await this.findPublishedTrack(trackId);

    if (!track.hlsManifestKey || !track.hlsDirectoryKey) {
      throw new NotFoundException('Stream not found');
    }

    const manifestPath = this.resolvePrivatePath(track.hlsManifestKey);
    let manifest = await readFile(manifestPath, 'utf-8');

    manifest = manifest.replace(
      /URI="enc\.key"/g,
      `URI="/api/stream/tracks/${track.id}/key"`,
    );

    manifest = manifest.replace(
      /segment_(\d+)\.ts/g,
      `/api/stream/tracks/${track.id}/segments/segment_${1}.ts`,
    );

    res.setHeader('Content-Type', 'application/vnd.apple.mpegurl');
    res.setHeader('Cache-Control', 'no-store');

    return res.send(manifest);
  }

  async sendSegment(trackId: string, segment: string, res: Response) {
    if (!/^segment_\d+\.ts$/.test(segment)) {
      throw new BadRequestException('Invalid segment name');
    }
  }
}

```

```

const track = await this.findPublishedTrack(trackId);

if (!track.hlsDirectoryKey) {
    throw new NotFoundException('Stream not found');
}

const segmentPath = this.resolvePrivatePath(
    `${track.hlsDirectoryKey}/${segment}`,
);

res.setHeader('Content-Type', 'video/mp2t');
res.setHeader('Cache-Control', 'private, max-age=300');

return res.sendFile(segmentPath);
}

async sendKey(trackId: string, res: Response) {
    const track = await this.findPublishedTrack(trackId);

    if (!track.hlsDirectoryKey) {
        throw new NotFoundException('Stream not found');
    }

    const keyPath = this.resolvePrivatePath(`${track.hlsDirectoryKey}/enc.key`);

    res.setHeader('Content-Type', 'application/octet-stream');
    res.setHeader('Cache-Control', 'no-store');

    return res.sendFile(keyPath);
}

private async findPublishedTrack(trackId: string) {
    const track = await this.prisma.track.findFirst({
        where: {
            id: trackId,
            isPublished: true,
        },
    });

    if (!track) {

```

```

        throw new NotFoundException('Track not found');
    }

    return track;
}

private resolvePrivatePath(key: string) {
    const rootDir = this.configService.get<string>('ST_ROOT_DIR') || 'storage';

    const privateRoot = join(process.cwd(), rootDir, 'private');

    const normalizedKey = key.replace(/^private\/\//, '');

    const resolvedPath = normalize(join(privateRoot, normalizedKey));

    if (!resolvedPath.startsWith(privateRoot)) {
        throw new BadRequestException('Invalid file path');
    }

    return resolvedPath;
}
}

```