

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:

«Сервіс для прослуховування музики»

Виконав студент групи ІПЗ-21-2
БОНДАР Вадим Максимович

Керівник роботи:
САВІЦЬКИЙ Роман Святославович

Рецензент:
КОНЦИДАЙЛО Андрій Михайлович

Житомир – 2025

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

«26» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **БОНДАР Вадим Максимович**

Керівник роботи: **САВІЦЬКИЙ Роман Святославович**

Тема роботи: **«Сервіс для прослуховування музики»**,

затверджена Наказом закладу вищої освіти від **«14» лютого 2025 р., №61/с**

Вихідні дані для роботи: Об'єктом дослідження є використання інформаційних технологій для створення вебсервісу для прослуховування музики, що надає користувачам можливість знаходити, відтворювати, зберігати та ділитися музичними композиціями.

Предметом дослідження є функціональні та технічні аспекти створення та використання онлайн-сервісу для прослуховування музики.

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Савіцький Р.С.	27.02.2025р.	27.02.2025р.
2	Савіцький Р.С.	27.02.2025р.	27.02.2025р.
3	Савіцький Р.С.	27.02.2025р.	27.02.2025р.

РЕФЕРАТ

Кваліфікаційна робота освітнього ступеня «бакалавр» присвячена розробці вебсервісу для прослуховування музики. Метою роботи є створення інтерактивної платформи, яка дозволяє користувачам здійснювати пошук, прослуховування та управління музичними композиціями в режимі онлайн. Предметом дослідження є технології створення потокових музичних сервісів, зокрема їх архітектура, функціональність і засоби реалізації. Об'єктом дослідження виступає вебсервіс для прослуховування музики.

У процесі роботи було проведено порівняльний аналіз аналогів та визначено функціональні вимоги до системи. Обрано клієнт-серверну архітектуру з використанням REST API для взаємодії між фронтендом і бекендом. Фронтенд частину реалізовано з використанням JavaScript-фреймворку React, а серверну частину за допомогою PHP-фреймворку Laravel. Для зберігання аудіофайлів використовується хмарне сховище AWS S3, для стрімінгу використано алгоритм HLS, для системи управління базами даних застосовано MySQL. У якості засобів для створення персоналізованих плейлистів було обрано Python ML.

Пояснювальна записка до кваліфікаційної роботи містить 50 сторінок, включає 35 ілюстрацій, 22 таблиці, 8 лістингів програмного коду та 9 сторінок додатків. Загальний обсяг — 70 сторінок.

Ключові слова: МУЗИКА, ВЕБСЕРВІС, ПІСНЯ, HLS, ПЛЕЙЛИСТ.

					ІПЗ.КР.Б – 121 – 25 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Сервіс для прослуховування музики Пояснювальна записка	Літ.	Арк.	Аркушів
Розроб.		Бондар В.М.						
Керівник		Савіцький Р.С.					3	70
Рецензент		Концидайло А.М.				Житомирська політехніка Група ІПЗ-21-2		
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

The qualification work for the bachelor's degree is devoted to the development of a web service for listening to music. The purpose of the work is to create a modern interactive platform that allows users to search, listen to and manage musical compositions online. The subject of the research is the technology for creating streaming music services, in particular their architecture, functionality and means of implementation. The object of the research is a web service for listening to music.

In the process of work, existing streaming services were analyzed and functional requirements for the system were determined. A client-server architecture using REST API was chosen for interaction between the frontend and backend. The frontend part was implemented using the React JavaScript framework, and the server part using the Laravel PHP framework. AWS S3 cloud storage is used to store audio files, the HLS protocol was used for streaming, and MySQL was used for the database management system. Python ML was chosen as a tool for creating personalized playlists.

The explanatory note to the qualification work contains 50 pages, includes 35 illustrations, 22 tables, 8 code listings, and 9 pages of appendices. The total volume is 70 pages.

Keywords: MUSIC, WEBSERVICE, SONG, HLS, PLAYLIST.

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ	9
1.1 Постановка задачі.....	9
1.2 Аналіз аналогів програмного продукту	10
1.3 Вибір архітектури сервісу для прослуховування музики	15
1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	17
Висновки до першого розділу.....	21
РОЗДІЛ 2. ПРОЄКТУВАННЯ СЕРВІСУ ДЛЯ ПРОСЛУХОВУВАННЯ МУЗИКИ	22
2.1 Визначення варіантів використання та об'єктно орієнтованої структури системи.....	22
2.2 Розробка бази даних системи.....	27
2.3 Проєктування та реалізація алгоритмів роботи системи	33
2.4 Реалізація функціоналу додатку	38
Висновок до другого розділу.....	43
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ.....	44
3.1 Архітектура та план розгортання сервісу.....	44
3.2 Структура інтерфейсу	45
3.3 Тестування роботи програмного продукту.....	54
Висновки до третього розділу	56
ВИСНОВКИ.....	57
ДОДАТКИ.....	61
Додаток А	62
Додаток Б.....	68
Додаток В	69
Додаток Г.....	70
Додаток Д	70

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БД – база даних

ПЗ – програмне забезпечення

AI/ML – Artificial Intelligence/Machine Learning

REST API – Representational State Transfer Application Programming Interface

ORM – Object-Relational Mapping

HTTP – Hypertext Transfer Protocol

ABS – Adaptive Bitrate Streaming

Формат M3U8 – M3U це текстовий формат плейлиста, який містить шляхи до медіафайлів (MP3, AAC, TS). M3U8 – це його версія з кодуванням UTF-8.

KNN – K-Nearest Neighbors

CDN – Content Delivery Network

HLS – HTTP Live Streaming

ВСТУП

Актуальність теми полягає у впливі росту цифрових технологій та популярності музичних стрімінгових сервісів на попит користувачів щодо унікальності музичних платформ. Слухач очікує якісний контент разом із зручним доступом до улюблених треків та персоналізованими рекомендаціями. У зв'язку з цим виникає потреба в ефективному музичному сервісі з використанням передових технологій для стабільного прослуховування музики, широкого функціоналу і персоналізованого досвіду для кожного користувача. Дослідження спрямоване на пошук найкращих методів розробки і оптимізації сервісу з використанням передових стандартів веброботки.

Метою є створення онлайн-сервісу для прослуховування музики, який забезпечує зручний доступ до аудіо в режимі реального часу, дозволяє користувачам створювати власні плейлисти, зберігати улюблені треки та отримувати рекомендації відповідно до індивідуальних музичних уподобань. Особливий акцент робиться на реалізації функціоналу із застосуванням сучасних вебтехнологій та ефективних рішень для потокової передачі аудіо.

Мета дослідження включає наступні завдання:

- провести аналіз існуючих аналогів стрімінгових сервісів;
- визначити ключові функціональні вимоги до платформи на основі досвіду провідних конкурентів на ринку;
- обрати відповідний стек технологій для реалізації вебсервісу з урахуванням вимог до продуктивності, масштабованості та безпеки;
- спроектувати архітектуру вебзастосунку;
- реалізувати основні модулі: каталог треків, плеєр, взаємодію користувача з музичним контентом, систему рекомендацій;
- реалізувати автентифікацію та авторизацію користувача з можливістю збереження вподобаного контенту та історії прослуховувань;

- інтегрувати потокову передачу аудіо з використанням алгоритму HLS;
- реалізувати систему рекомендації на основі Python AI/ML KNN;
- розробити адміністративну панель для керування контентом платформи;
- здійснити тестування функціональних компонентів системи.

Предметом дослідження виступають методи та засоби розробки, що застосовуються для створення онлайн-сервісів потокової передачі аудіо. Особлива увага приділяється протоколам передачі даних, механізмам кешування, технологіям обробки та зберігання музичного контенту, а також методам персоналізації — зокрема, алгоритму рекомендацій на основі логів прослуховування та характеристик треків.

Об’єктом дослідження є процес розробки онлайн-платформи для стрімінгу музики як складного багаторівневого вебзастосунку, що включає як клієнтську, так і серверну частину, взаємодіє з базою даних і хмарним сховищем, а також механізми реалізації потокової передачі аудіо та дослідження алгоритмів генерації персоналізованих музичних рекомендацій.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

1.1 Постановка задачі

Основна мета роботи полягає у розробці вебсервісу для прослуховування музики, який забезпечує зручний та інтуїтивний інтерфейс для користувачів, дозволяє відтворювати аудіо-треки онлайн, створювати персональні плейлисти та отримувати рекомендації на основі індивідуальних вподобань. Реалізація даного сервісу має на меті підвищити доступність музичного контенту, забезпечити високу якість потокової передачі аудіо та створити приємний користувацький досвід.

Вебсервіс для прослуховування музики має складатися з таких ключових модулів:

1. Музичний каталог забезпечує зберігання, редагування та перегляд музичних треків, плейлистів, виконавців і альбомів, які можна фільтрувати за жанрами, артистами, альбомами, а також здійснювати пошук за ключовими словами.
2. Користувач включає механізм автентифікації, авторизації, також можливість створювати, редагувати (зокрема налаштовувати приватність), зберігати власні плейлисти та додавати до себе плейлисти інших користувачів.
3. Плеєр дозволяє відтворювати музику потоково з базовим та розширеним керуванням: пауза, перемотування, зміна гучності, перехід між треками, випадкове відтворення, зациклення, перегляд тексту пісні та додаткової інформації.
4. Рекомендації, а саме музичні добірки генеруються на основі історії прослуховування, що покращує досвід взаємодії з платформою.
5. Адміністратор включає керування контентом, модерацію повідомлень та перегляд статистики активності сервісу (Топ пісень за місяць, за весь час, загальна кількість прослуховувань, список найпопулярніших виконавців).

Основними етапами розробки платформи є:

1. Розробка логіки взаємодії користувача з інтерфейсом для забезпечення зручності та інтуїтивного користування.
2. Створення адаптивних вебсторінок, що коректно масштабуються під різні екрани – від мобільних пристроїв до великих моніторів.
3. Створення серверного інтерфейсу для обміну даними між фронтендом і бекендом у стандартизованому форматі (REST API).
4. Налаштування двостороннього зв'язку між клієнтською частиною та сервером, включаючи запити, обробку даних та помилок.
5. Підключення до зовнішніх хмарних сервісів для завантаження, збереження, трансляції аудіоконтенту.
6. Створення структури бази даних для зберігання інформації про користувачів, треки, плейлисти та інші елементи системи.
7. Розробка механізмів входу в систему: реєстрація, вхід.
8. Реалізація відтворення аудіо в реальному часі з мінімальною затримкою.
9. Можливість для користувачів додавати, редагувати, видаляти, наповнювати плейлисти та переглядати доступний контент.
10. Розробка алгоритмів рекомендацій на основі історії прослуховувань користувача.
11. Запровадження ролей і обмежень для різних типів користувачів для забезпечення безпеки.

Результатом виконання роботи має бути веборієнтований багатокористувацький сервіс для прослуховування музики, який поєднує клієнтську частину, серверну інфраструктуру, інтеграцію з хмарним сховищем для зберігання аудіофайлів, потоковим стрімінгом, рекомендаціями та керуванням базами даних.

1.2 Аналіз аналогів програмного продукту

Стрімінгові сервіси є одним із основних способів легального споживання медіа контенту. Для розробки власного музичного сервісу

					ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
						10

1. Безкоштовний рівень, який надає доступ до більшої частини каталогу з рекламними вставками, обмеженим функціоналом та нижчою якістю звуку.

2. Преміум рівень дозволяє слухати музику офлайн, без реклами, якість пісень значно вища, необмежений пропуск пісень та інші.

Spotify, як і усі інші стрімінгові платформи, застосовує ABR – адаптивний бітрейт. Це підхід, що дозволяє динамічно регулювати якість бітрейту залежно від поточної пропускної здатності мережі користувача. Щодо конкретних протоколів доставки, Spotify використовує власний, пропрієтарний протокол для потокової передачі контенту. Цей протокол був розроблений для оптимізації, буферизації та мінімізації затримки. Важливо зазначити, що достеменно невідомо, який саме протокол використовується в деталях, оскільки інформація про його внутрішню архітектуру є комерційною таємницею компанії.

Однак з концептуальної точки зору, підхід дуже схожий на HTTP Live Streaming (HLS) [4] – протокол розроблений Apple [5], який широко застосовується для доставки мультимедійного контенту.

Apple Music – стрімінговий сервіс від Apple, запущений у 2015 році. Він є ключовим компонентом екосистеми Apple, але також доступний на інших платформах, таких як Android та Windows (Рис. 1.2).

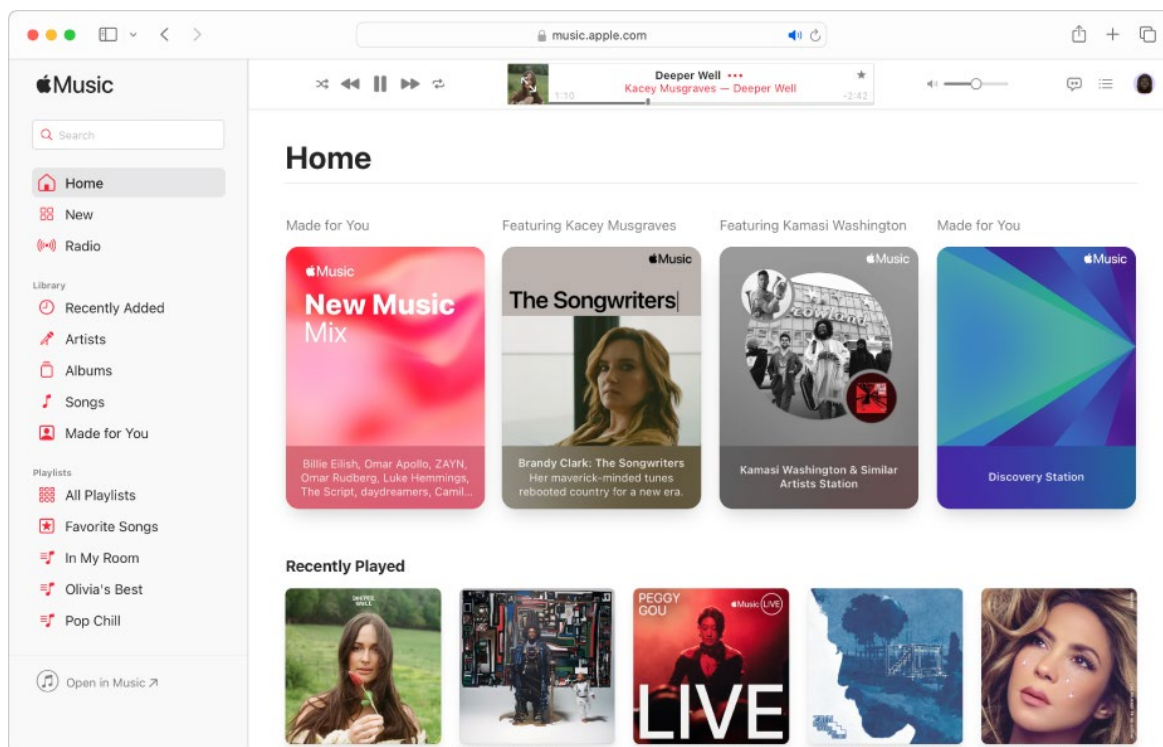


Рисунок 1.2 Apple music

Ключові функціональні особливості Apple Music включають інтеграцію з екосистемою Apple, що забезпечує безкоштовний доступ до сервісу на пристроях Apple. Це вважається однією з головних переваг для користувачів продукції компанії. Серед ключових технологічних рішень – підтримка Lossless Audio та Dolby Atmos, що надають змогу слухати музику у високоякісному форматі без втрат, а також у форматі просторового аудіо. Ці функції доступні всім підписникам без додаткової плати. Окрему увагу приділено кураторському контенту та радіо – користувачам пропонуються плейлисти, створені музичними експертами й виконавцями, а також радіостанції з діджеями. Бізнес-модель Apple Music – це тільки платна підписка (Premium), яка передбачає надання безкоштовного пробного періоду для нових користувачів.

Для доставки контенту використовується HTTP Live Streaming – HLS. Оскільки HLS є протоколом, розробленим Apple, відповідно Apple Music використовує HLS як основний протокол для доставки аудіоконтенту, що забезпечує нативну підтримку на всіх пристроях Apple та оптимізовану інтеграцію в їхню екосистему.

У Apple Music технологія HLS використовується для забезпечення адаптивного потокового відтворення аудіо. Аудіофайли попередньо кодуються у кількох варіантах бітрейту та розбиваються на короткі сегменти тривалістю приблизно 6–10 секунд. Після цього створюються спеціальні плейлисти у форматі .m3u8. Майстер-плейлист містить посилання на всі доступні варіанти якості, тоді як для кожного з них формується окремий медіа-плейлист, що включає послідовний список URL-адрес аудіосегментів конкретного бітрейту.

Клієнтський плеєр Apple Music завантажує майстер-плейлист, обирає найоптимальніший варіант якості відповідно до поточного стану мережі та починає поступово завантажувати аудіосегменти з відповідного медіа-плейлиста. У разі зміни мережевих умов плеєр може динамічно перейти на інший рівень якості, продовжуючи відтворення без помітних збоїв, просто звертаючись до іншого медіа-плейлиста.

Застосування HLS у Apple Music забезпечує високоякісне, гнучке та масштабоване потокове аудіо, яке легко розповсюджується через CDN і гарантує стабільне відтворення навіть за нестабільного інтернет-з'єднання.

Порівняльний аналіз Spotify vs Apple Music (див. табл. 1.1)

Таблиця 1.1

Порівняльний аналіз Spotify та Apple Music

N	Характеристика	Spotify	Apple Music	Розроблюваний продукт
1.	Ключова перевага	Персоналізовані рекомендації, подкасти,	Інтеграція з екосистемою Apple, Lossless Audio, кураторський контент	Персоналізовані рекомендації, кураторський контент(створений адміністратором)
2.	Доставка контенту	Власний пропрієтарний протокол	HLS	HLS

3.	Платформи	Дуже широкий вибір (веб, мобільні, десктоп, Smart TV, консолі)	Apple-пристрої, Android, Windows (через iTunes)	Веборієнтований з адаптивним інтерфейсом
4.	Взаємодія з користувачем	Більш розвинені	Менш розвинені	Середні

Аналіз Spotify та Apple Music підкреслює, що успішний музичний стрімінговий сервіс потребує надійної технічної архітектури та сильної орієнтації на користувача. Обидва сервіси використовують принципи адаптивного бітрейту для забезпечення безперебійного прослуховування, хоча Apple Music робить це, базуючись на своєму власному протоколі HLS. Використання HLS, або будь-якого подібного стандарту адаптивного стрімінгу є важливим для забезпечення масштабованості, високої якості та надійності доставки контенту у сервісі. А також стало зрозуміло, що взаємодія з користувачем вирішує, що справжня конкурентна перевага полягає у тому, як сервіс взаємодіє з користувачем. Чи це через виняткові рекомендації, унікальні соціальні функції, фокус на якості звуку, чи бездоганна інтеграція з пристроями. Формування привабливого та зручного користувацького досвіду є вирішальним чинником для залучення нових користувачів і утримання наявної аудиторії.

1.3 Вибір архітектури сервісу для прослуховування музики

Для розробки ефективного та масштабованого сервісу для прослуховування музики було обрано клієнт-серверну архітектуру (див. рис. 1.3.). Цей підхід є одним із найпоширеніших у створенні вебдодатків та забезпечує розділення відповідальностей між користувацьким інтерфейсом (клієнтом) і серверною логікою.



Рисунок 1.3. Клієнт-серверна архітектура

Архітектура клієнт-сервер базується на взаємодії двох основних компонентів:

- клієнта – програми, що запускається на пристрої користувача;
- сервера – централізованої системи, що обробляє запити, зберігає і керує даними.

Клієнтська частина відповідає за взаємодію з користувачем. Відображення інтерфейсу, обробку подій, відтворення музики та надсилання запитів до сервера.

Серверна частина забезпечує бізнес-логіку додатку: автентифікацію користувачів, управління музичною базою, обробку потокового аудіо, збереження даних у базі даних, а також інтеграцію з хмарним сховищем для зберігання музичних файлів.

До переваг клієнт-серверної архітектури для музичного сервісу можна віднести те, що клієнтська частина відповідає за інтерфейс і взаємодію з користувачем, що дозволяє розвантажити сервер від зайвих обчислень та забезпечити швидкий відгук на дії користувача. Сервер, у свою чергу, зосереджений на зберіганні, обробці даних та управлінні доступом, що сприяє оптимізації продуктивності системи загалом. Централізоване управління через сервер дозволяє ефективно контролювати автентифікацію користувачів і обмеження доступу до музичного контенту, підвищуючи рівень безпеки як для персональних даних, так і для захищеного контенту. У разі зростання навантаження архітектура дозволяє масштабувати систему,

додаючи серверні ресурси або нові вузли, що забезпечує стабільну роботу сервісу за великої кількості користувачів.

Окремою перевагою є можливість централізованого оновлення серверної логіки без потреби змінювати клієнтські застосунки, що значно спрощує технічну підтримку та впровадження нових функцій. Крім того, одна й та сама серверна логіка може бути використана для різних типів клієнтів — веб, мобільних (iOS, Android) і десктопних (Windows, macOS, Linux), що робить архітектуру зручною для масштабування та розширення функціональності в майбутньому.

Клієнт надсилає запити до сервера через REST API [6], отримуючи у відповідь необхідні дані у форматі JSON [7]. Це дозволяє ефективно реалізувати функції пошуку музики, керування плейлистами, авторизації та потокового відтворення.

1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Для реалізації сервісу прослуховування музики було обрано технології та інструменти, які забезпечують високу продуктивність, масштабованість, зручність розробки та підтримку. Основна мета вибору інструментальних засобів – створення стабільного, швидкого та зручного у використанні додатку, який легко розширювати та обслуговувати.

Для клієнтської частини було обрано React [8] – одну з найпопулярніших JavaScript [9] бібліотек для створення інтерфейсів у веброзробці. У порівнянні з Angular [10], який є громіздким фреймворком з жорсткими структурами, React дозволяє швидко почати роботу завдяки своїй гнучкості й простій компонентній моделі. Vue.js [11], хоча й привабливий через низький поріг входу, поступається React у плані екосистеми, масштабованості та підтримки великих проєктів. Крім того, React має величезну кількість готових бібліотек і плагінів, що дозволяє значно скоротити час на реалізацію типових задач – від маршрутизації до керування станом. Завдяки віртуальному DOM React забезпечує високу продуктивність

при зміні інтерфейсу, що важливо при створенні динамічних SPA-додатків із великою кількістю взаємодій.

Для маршрутизації у SPA [12] обрано React Router DOM [13]. Бібліотека дає змогу створювати багаторівневі маршрути без перезавантаження сторінки. Альтернативи, наприклад Vue Router, менш зручні в роботі з динамічними параметрами або вкладеними маршрутами, а власні реалізації маршрутизації не мають належної гнучкості й підтримки. Таким чином, React Router DOM забезпечує ідеальний баланс між простотою використання та функціональністю.

Для валідації форм на клієнтській частині було використано бібліотеку Yup [14], яка є потужним інструментом для створення схематичної перевірки даних у поєднанні з React. У порівнянні з власноруч написаною валідацією або простими умовними перевірками, Yup забезпечує декларативний і зрозумілий синтаксис, що дозволяє легко описувати складні правила перевірки.

Одним із ключових елементів будь-якого музичного сервісу є відтворення аудіо. Оскільки сучасні браузерери не завжди підтримують HLS нативно, було вирішено використати бібліотеку HLS.js [15]. Вона дозволяє програвати HLS-поток практично у всіх сучасних браузерах, що значно підвищує доступність сервісу для різних користувачів.

Для стилізації інтерфейсу було обрано TailwindCSS [16] фреймворк, який дозволяє створювати привабливий і адаптивний дизайн без необхідності писати багато власного CSS-коду [17]. На відміну від Bootstrap [18], Tailwind не нав'язує готові компоненти й стилі, а дозволяє створювати унікальний дизайн із набору утилітарних класів.

Усі HTTP-запити до серверної частини обробляються за допомогою бібліотеки Axios [19], яка забезпечує просту й гнучку роботу з API. Axios підтримує promise, легко налаштовується і добре інтегрується з React-додатком.

Серверна частина проєкту реалізована з використанням Laravel [20] – одного з найпопулярніших PHP-фреймворків [21]. Він побудований за архітектурою MVC і має безліч вбудованих функцій, таких як ORM Eloquent [22], зручна маршрутизація, підтримка аутентифікації та багато іншого.

Для автентифікації користувачів використовується Laravel Sanctum [23] – легкий та ефективний інструмент, який добре підходить для реалізації захищеної авторизації в SPA-додатках. Він дозволяє працювати як з токенами, так і з сесійною аутентифікацією. Для проєкту обрано валідацію за токеном.

У якості бази даних було обрано MySQL [24] – реляційну систему керування базами даних. Вона чудово інтегрується з Laravel, особливо через Eloquent ORM, і підходить для зберігання структурованої інформації, як-от дані користувачів, плейлисти, треки та інше. Для зручного адміністрування використовується phpMyAdmin.

Оскільки один із головних викликів музичних сервісів — це ефективне зберігання та передача великих обсягів мультимедійного контенту, для цього застосовується хмарне сховище AWS S3 [25]. Воно забезпечує високу надійність, доступність і масштабованість. У порівнянні з локальним зберіганням або менш популярними хмарними платформами, AWS надає широкі можливості й добре інтегрується з іншими компонентами інфраструктури.

Щоб забезпечити захист аудіофайлів, використовується алгоритм шифрування AES-128 [26]. Він дозволяє зашифровувати .ts-сегменти HLS-потoku, запобігаючи несанкціонованому доступу до контенту.

Для обробки та підготовки аудіо (кодування, сегментація, шифрування) використовується FFmpeg [27] – інструмент для мультимедійної обробки.

Для реалізації персоналізованих рекомендацій було використано Python [28] і алгоритм K-Nearest Neighbors [29]. KNN, або K-найближчих сусідів, є простим, але ефективним алгоритмом машинного навчання. Його основний принцип полягає у прийнятті рішень на основі схожості з найближчими

точками даних у просторі ознак. Python обраний завдяки своїй популярності у сфері аналізу даних і машинного навчання.

Аналіз основних протоколів доставки медіа-контенту наведено у таблиці рис. 1.2.

Таблиця 1.2

Порівняння Range Streaming [30] vs HTTP Live Streaming

Характеристика	Range Streaming	HTTP Live Streaming (HLS)
Принцип роботи	Клієнт запитує байтові діапазони файлу.	Файл розбивається на короткі сегменти. Плейлист (.m3u8) містить в собі посилання на сегменти.
Адаптивний бітрейт	Відсутній. Якість фіксована. Можлива буферизація при нестабільній мережі.	Підтримується. Також можлива оптимізація, яка включає зменшення тривалості сегментів.
Захист контенту	Важко реалізувати.	Легко. Кожен .ts сегмент може бути зашифрований.
Масштабованість (CDN)	Низька.	Висока.
Перемотування	Потребує розрахунків байтових зміщень. Що є тривалим процесом.	Швидко, оскільки елементи вже готові.
Трансляції	Не підтримується.	Підтримується.
Кількість HTTP запитів	Маленька.	Більша (1 запит на сегмент).
Користувацький досвід	Можливі переривання та буферизація.	Плавне відтворення.

Очевидним вибором є HLS.

Також було використано інші інструменти, такі як Postman [31] – для тестування REST API під час розробки та VS Code [32] – як основне середовище розробки.

Висновки до першого розділу

У даному розділі було виконано постановку задачі на кваліфікаційну роботу та визначено функціональні вимоги до сервісу прослуховування музики. Проведено дослідження існуючих рішень на ринку, таких як Spotify, Apple Music, завдяки чому було сформовано перелік основних функціональних можливостей системи: потокове відтворення аудіо, персоналізовані рекомендації, керування плейлистами, пошук за жанрами та виконавцями, зручний користувацький інтерфейс і адаптивність до різних типів пристроїв.

Також було продемонстровано, обґрунтовано та доведено вибір архітектури клієнт-серверного типу, яка забезпечує масштабованість, модульність і можливість подальшого розширення системи. Для розробки клієнтської частини було використано мову програмування JavaScript та фреймворк React, Tailwind для стилізації. Серверну частину реалізовано з використанням PHP фреймворку Laravel. У якості бази даних використано MySQL, а для зберігання аудіофайлів – хмарне сховище Amazon S3. Для стрімінгу – протокол HLS. Python та KNN для рекомендацій.

Обрані інструментальні засоби дозволяють ефективно реалізувати вебсервіс для прослуховування музики, забезпечуючи масштабовану архітектуру, інтеграцію з хмарними сервісами, зручну розробку та підтримку. Такий стек технологій також створює міцну основу для подальшого розширення сервісу, включно з мобільними та десктопними додатками.

РОЗДІЛ 2. ПРОЄКТУВАННЯ СЕРВІСУ ДЛЯ ПРОСЛУХОВУВАННЯ МУЗИКИ

2.1 Визначення варіантів використання та об'єктно орієнтованої структури системи

Сервіс призначений для прослуховування, створення та організації музичного контенту. Він використовує архітектуру та логіку взаємодії об'єктів для надання користувачам комплексного музичного досвіду. Система базується на стеку React та React Router DOM для клієнтської частини, що забезпечує швидкий та динамічний користувацький інтерфейс. Для ефективної доставки та відтворення потокового аудіо використовується бібліотека HLS.js, яка інтегрує протокол HTTP Live Streaming. Стилiзація інтерфейсу здійснюється за допомогою TailwindCSS, а для взаємодії з бекендом використовується Axios.

Серверна частина розроблена на фреймворку Laravel, який забезпечує надійну та масштабовану основу для обробки запитів та керування даними. Для автентифікації користувачів застосовується Laravel Sanctum. Дані зберігаються в базі даних MySQL. Зберігання медіа-файлів, зокрема аудіо-сегментів HLS, здійснюється в хмарному сервісі AWS S3, що гарантує високу доступність та масштабованість. Для шифрування аудіо-сегментів використовується стандарт AES-128, а процес створення та шифрування .ts файлів забезпечується інструментом FFmpeg. Додатково, для реалізації функціоналу персоналізованих рекомендацій, застосовується алгоритм KNN (K-Nearest Neighbors), імплементований на Python, що дозволяє формувати рекомендації на основі історії прослуховувань користувача. Основними функціями системи є відтворення музики, створення та редагування плейлистів, персональні рекомендації. У системі визначено дві основні ролі: Користувач та Адмін (див. рис. 2.1)

Продовження таблиці 2.1

«extend»	опціонально переглядати детальну інформацію про поточний трек: альбом, виконавець, дата релізу.
	Користувач може переглядати текст пісні.
	Користувач може додавати поточний трек до існуючих плейлистів або видаляти його з них.
	Користувач може додавати поточний трек як улюблений до спеціального списку.
	Користувач має можливість поділитися посиланням на пісню.
	Прецедент який розширює прослуховування музики, дозволяючи контролювати процес відтворення.

Таблиця 2.2

Опис прецеденту «Керування плеєром»

Назва	Керування плеєром.
Опис	Прецедент об'єднує всі дії, пов'язані з контролем над відтворенням музики. Він є розширенням прецеденту «Прослуховування музики».
Актор	Користувач.
Зв'язки «include»	Дозволяє перемотувати трек вперед або назад.
	Перемикання на наступну пісню в черзі.
	Перемикання на минулу пісню в черзі.
	Увімкнення/вимкнення повторного відтворення поточного треку, коли він закінчиться.
	Регулювання рівня звуку.

Таблиця 2.3

Опис прецеденту «Пошук контенту»

Назва	Пошук контенту.
Опис	Дозволяє користувачеві знаходити музичні твори, альбоми, виконавців, або плейлисти за ключовими словами або фільтрацією.
Актор	Користувач.

Таблиця 2.4

Опис прецеденту «Перегляд рекомендацій»

Назва	Перегляд рекомендацій.
-------	------------------------

Продовження таблиці 2.4

Опис	Користувач може переглядати персоналізовані рекомендації щодо музики у вигляді щоденних добірок (плейлистів).
Актор	Користувач.

Таблиця 2.5

Опис прецеденту «Підписка на виконавця»

Назва	Підписка на виконавця.
Опис	Користувач може додати до своїх підписок виконавців і переглядати їх у своєму списку.
Актор	Користувач.

Таблиця 2.6

Опис прецеденту «Зв'язок з адміністрацією»

Назва	Зв'язок з адміністрацією.
Опис	Дозволяє користувачеві надсилати скарги, пропозиції до адміністрації сервісу.
Актор	Користувач.

Таблиця 2.7

Опис прецеденту «Керування плейлистами»

Назва	Керування плейлистами.
Опис	Користувач може створювати плейлисти, додавати/видаляти з них пісні, редагувати, видаляти. Також додавати до свого списку плейлисти інших користувачів.
Актор	Користувач.

Таблиця 2.8

Опис прецеденту «Управління контентом»

Назва	Управління контентом.
Опис	Головний прецедент для адміністратора, що дозволяє йому керувати всією музичною бібліотекою сервісу.
Актор	Адмін.
Зв'язки «include»	Керування бібліотекою пісень, а саме додавання пісень, альбомів, їх редагування, видалення.
	Управління системними плейлистами.
	Додавання, редагування, видалення виконавців.

Опис прецеденту «Перегляд статистики»

Назва	Перегляд статистики.
Опис	Адміністратор може переглядати статистичні дані про використання сервісу(кількість прослуховувань, популярність треків, виконавців, активність користувачів).
Актор	Адмін.

Розглянемо діаграму класів (див. рис. 2.2). Діаграма класів описує основні сутності та взаємозв'язки сервісу для прослуховування музики. Користувач (User) може створювати плейлисти (Playlist), додавати пісні, які сподобались до списку вподобань (LikedSong), зберігати плейлисти інших користувачів (SavedPlaylist), підписуватись на улюблених виконавців, а саме додавати їх в свій список (ArtistSubscription). Виконавці можуть мати багато альбомів (Album). Пісні можуть існувати як частина альбома або як сингли. Кожна пісня може мати кілька жанрів (Genre), а також може бути доданою до багатьох плейлистів (PlaylistSong). (SongPlay) відстежує історію прослуховувань.

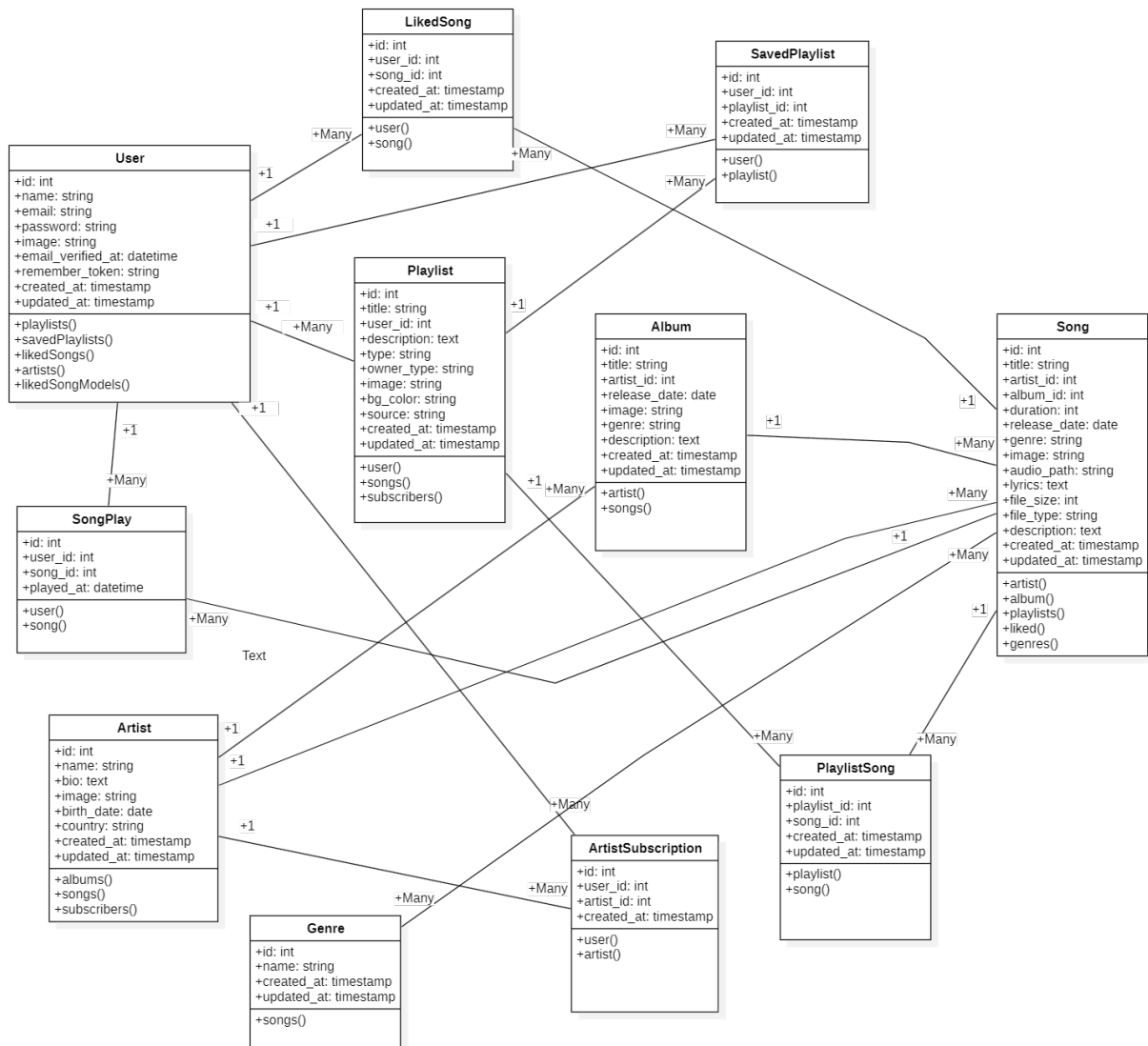


Рисунок 2.2 Діаграма класів

2.2 Розробка бази даних системи

Для зберігання даних користувачів та контенту використовується реляційна база даних MySQL. Для взаємодії з базою даних на бекенді застосовується ORM Eloquent. Це дозволяє працювати з таблицями як з об'єктами, сутностями та їх взаємозв'язками, значно спрощуючи запити. ORM забезпечує абстракцію над SQL-запитами, автоматично реалізує зв'язки між моделями, що робить роботу з базою даних більш зручною і безпечною. Крім того, Laravel надає механізми для міграцій, що полегшують ведення версій та зміну структури бази даних. Структуру бази даних проєкту наведено на рис. 2.3.

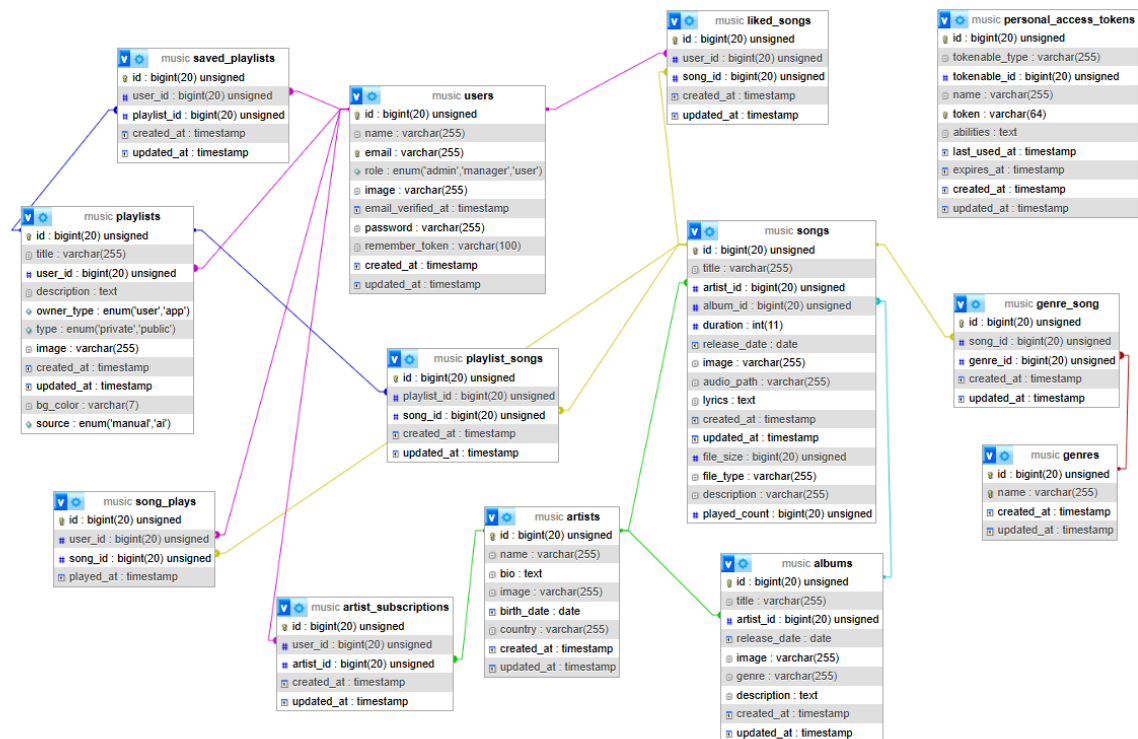


Рисунок 2.3 Структура бази даних

Опис для структури наведено в таблицях 2.10-2.20.

Таблиця «Users» призначена для збереження даних користувача (див. табл. 2.10).

Таблиця 2.10

Таблица «Users»

Назва	Тип даних	PK	FK	Опис
Id	Bigint	+	-	Ідентифікатор
Name	Varchar(255)	-	-	Ім'я
Email	Varchar(255)	-	-	Пошта@
Role	Enum(“admin”, ”user”)	-	-	Роль
Image	Varchar(255)	-	-	Зображення
Password	Varchar(255)	-	-	Пароль
Created_at	timestamp	-	-	Дата створення

Таблиця «Songs» призначена для збереження даних пісень (див. табл. 2.11).

Таблиця 2.11

Таблиця «Songs»

Назва	Тип даних	PK	FK	Опис
Id	Bigint	+	-	Ідентифікатор
Title	Varchar(255)	-	-	Назва
Artist_id	Bigint	-	Artist.id	Виконавець
Album_id	Bigint	-	Album.id	Альбом
Duration	Int	-	-	Тривалість
Release_date	Date	-	-	Дата випуску
Image	Varchar(255)	-	-	Зображення
Audio_path	Varchar(255)	-	-	Шлях файлу
Lyrics	text	-	-	Текст пісні
Created_at	timestamp	-	-	Дата створення
File_size	Bigint	-	-	Розмір файлу
File_type	Varchar(255)	-	-	Тип
Description	Varchar(255)	-	-	Опис
Played_count	Bigint	-	-	Кількість прослуховувань

Таблиця «Playlists» призначена для збереження даних плейлистів (див. табл. 2.12).

Таблиця 2.12

Таблиця «Playlists»

Назва	Тип даних	PK	FK	Опис
Id	Bigint	+	-	Ідентифікатор
Title	Varchar(255)	-	-	Назва
User_id	Bigint	-	User.id	Власник (користувач)
Description	Text	-	-	Опис
Owner_type	Enum(„user”, „app”)	-	-	Тип Власника
Type	Enum(„private”, „public”)	-	-	Приватність
Image	Varchar(255)	-	-	Зображення
Created_at	timestamp	-	-	Дата створення
Bg_color	Varchar(7)	-	-	Колір фону
source	Enum(„manual”, „ai”)	-	-	Метод створення

Таблиця «Artists» призначена для збереження даних виконавців (див. табл. 2.13).

Таблиця 2.13

Таблиця «Artists»

Назва	Тип даних	PK	FK	Опис
Id	Bigint	+	-	Ідентифікатор
Name	Varchar(255)	-	-	Ім'я
bio	Text	-	-	Опис
Image	Varchar(255)	-	-	Зображення
Birth_date	date	-	-	Дата народження
country	Varchar(255)	-	-	Країна
Created_at	timestamp	-	-	Дата створення

Таблиця «Albums» призначена для збереження даних альбомів (див. табл. 2.13).

Таблиця 2.14

Таблиця «Albums»

Назва	Тип даних	PK	FK	Опис
Id	Bigint	+	-	Ідентифікатор
Title	Varchar(255)	-	-	Назва
Artist_id	Bigint	-	Artist.id	Виконавець
Release_date	Text	-	-	Дата випуску
Image	Enum(„user”, „app”)	-	-	Зображення
Description	Varchar(255)	-	-	Опис
Created_at	timestamp	-	-	Дата створення

Таблиця «Genres» призначена для опису жанрів (див. табл. 2.15).

Таблиця 2.15

Таблиця «Genres»

Назва	Тип даних	PK	FK	Опис
Id	Bigint	+	-	Ідентифікатор
Name	Varchar(255)	-	-	Назва

Продовження таблиці 2.15

Created_at	timestamp	-	-	Дата створення
------------	-----------	---	---	----------------

Таблиця «SongPlays» призначена для збереження даних про прослухані користувачем пісні (див. табл. 2.16).

Таблиця 2.16

Таблиця «SongPlays»

Назва	Тип даних	PK	FK	Опис
Id	Bigint	+	-	Ідентифікатор
User_id	Bigint	-	User.id	Користувач
Song_id	Bigint	-	Song.id	Пісня
Played_at	timestamp	-	-	Дата програвання

Таблиця «Saved_playlists» призначена для збереження даних про збережені користувачем плейлисти інших користувачів. Також це є pivot таблицею для реалізації many-to-many зв'язку. Один користувач може зберегти багато плейлистів, один плейлист може бути збережений багатьма користувачами (див. табл. 2.17).

Таблиця 2.17

Таблиця «Saved_playlists»

Назва	Тип даних	PK	FK	Опис
Id	Bigint	+	-	Ідентифікатор
User_id	Bigint	-	User.id	Користувач
Playlist_id	Bigint	-	Playlist.id	Плейлист
Created_at	timestamp	-	-	Дата Збереження

Таблиця «Liked_songs» призначена для збереження даних про пісні, які вподобав користувач. Також це є pivot таблиця для реалізації many-to-many зв'язку. Один користувач може вподобати багато пісень, одна пісня може бути вподобана багатьма користувачами (див. табл. 2.18).

Таблиця 2.18

Таблиця «Liked_songs»

Назва	Тип даних	PK	FK	Опис
Id	Bigint	+	-	Ідентифікатор
User_id	Bigint	-	User.id	Користувач
Song_id	Bigint	-	Song.id	Пісня
Created_at	timestamp	-	-	Дата додавання

Таблиця «Playlist_songs» призначена для збереження пісень до плейлистів користувача. Також це є pivot таблиця для реалізації many-to-many зв'язку. Один плейлист може містити багато пісень, одна пісня може входити до багатьох плейлистів (див. табл. 2.19).

Таблиця 2.19

Таблиця «Playlist_songs»

Назва	Тип даних	PK	FK	Опис
Id	Bigint	+	-	Ідентифікатор
Playlist_id	Bigint	-	Playlist.id	Плейлист
Song_id	Bigint	-	Song.id	Пісня
Created_at	timestamp	-	-	Дата додвання

Таблиця «Artist_subscriptions» призначена для збереження підписок користувача на виконавців. Також це є pivot таблиця для реалізації many-to-many зв'язку. Один користувач може підписатися на багатьох виконавців, один виконавець може мати багато підписників (див. табл. 2.20).

Таблиця 2.20

Таблиця «Artist_subscriptions»

Назва	Тип даних	PK	FK	Опис
Id	Bigint	+	-	Ідентифікатор
User_id	Bigint	-	User.id	Користувач
Artist_id	Bigint	-	Artist.id	Пісня
Created_at	timestamp	-	-	Дата підписки

Для реалізації функціоналу було спроектовано логічну модель бази даних, що охоплює основні напрямки взаємодії користувача з музичним контентом.

Таблиця «User» зберігає основну інформацію про користувачів сервісу, аватар, пошту, пароль. Вона є центральною у системі та пов'язана з більшістю інших сутностей.

Таблиці «Artist, Album, Song, Genre» відповідають за зберігання музичного контенту.

Функціонал взаємодії користувачів з контентом реалізований за допомогою таких таблиць:

- liked_songs дозволяє користувачам зберігати улюблені треки;
- saved_playlists дає змогу зберігати плейлисти інших користувачів;
- artist_subscription реалізує підписку на виконавців;
- song_plays зберігає історію прослуховувань;
- playlists дозволяє користувачам створювати власні плейлисти.

2.3 Проектування та реалізація алгоритмів роботи системи

Для більшості ключових функцій сервісу, таких як прослуховування, створення та редагування плейлистів, отримання персональних рекомендацій, підписка на виконавців чи плейлисти, користувачам необхідно пройти процес автентифікації. Представлена діаграма активності (рис. 2.4) детально ілюструє послідовність дій, які виконуються користувачем у сервісі, відображаючи основні потоки від початку взаємодії до завершення.

Процес взаємодії з сервісом починається з початкової точки, що веде до активності «Реєстрація користувача». Після чого перевіряється «Чи наявний обліковий запис?». Якщо облікового запису немає – гілка ні, інакше – так. Після перевірки правильності введення даних, чи то під час реєстрації, чи автентифікації, користувач входить до облікового запису.

Після успішного проходження автентифікації користувач потрапляє на центральний пункт «Перегляд головної сторінки» на якому користувач може отримати доступ до більшості інших функціональних можливостей:

- перегляд новинок (нових пісень, плейлистів доданих адміністратором);
- перегляд рекомендацій, що формуються персонально для нього;
- редагування/перегляд особистого контенту.

Також у разі виникнення питань або проблем, користувач може скористатися формою для зв'язку з адміністрацією.

Однією з ключових активностей, до якої можливо перейти з різних точок є «Прослуховування музики». Під час або після цього процесу користувач може виконувати ряд додаткових дій для взаємодії з треком. Безпосередньо пов'язаним з минулим блоком є «Керування плеєром», який охоплює всі функції контролю відтворення.

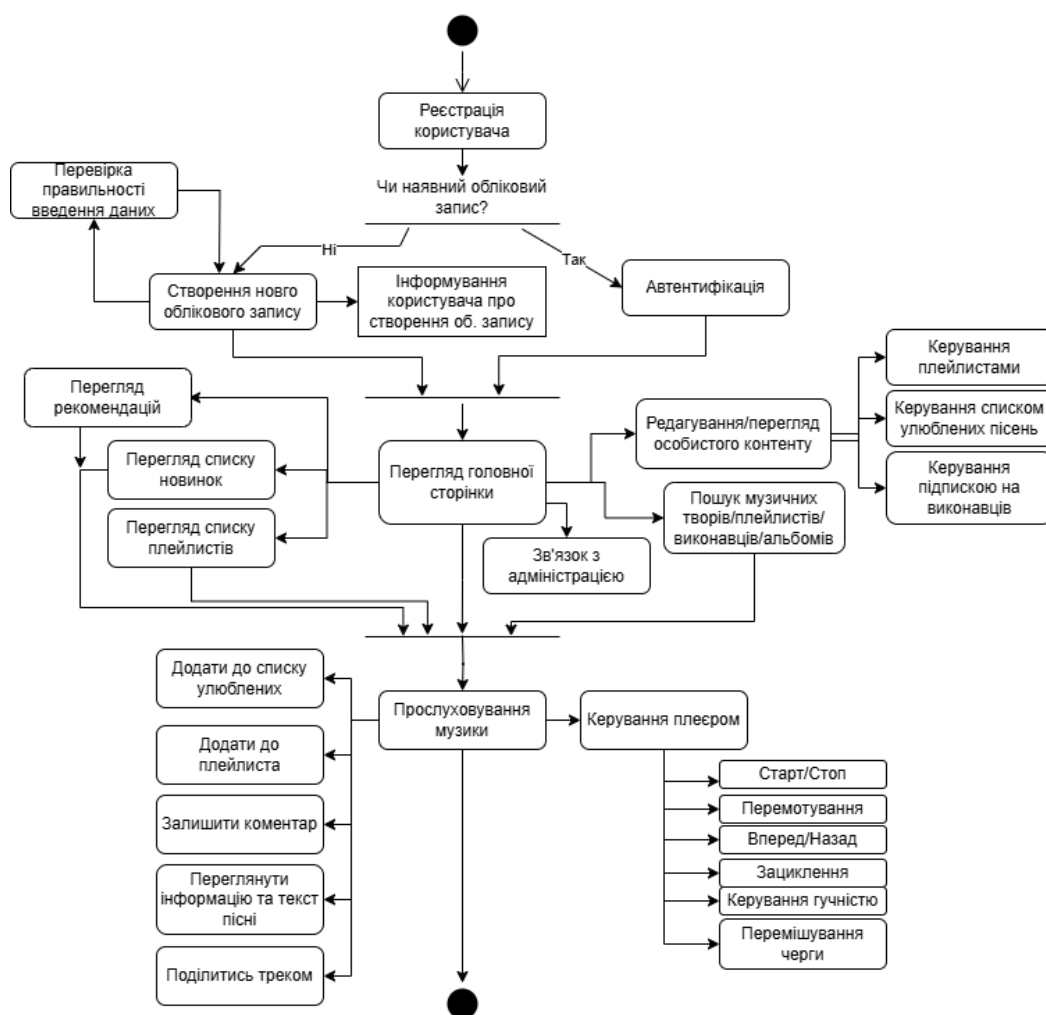


Рисунок 2.4 Діаграма активності користувача

Для детального розуміння взаємодії між компонентами системи під час потокового відтворення музики наведено діаграму послідовностей (див. рис.

```
sequenceDiagram
    participant User as Користувач (браузер)
    participant Client as Клієнтська частина (React + HLS.js)
    participant Server as Laravel-бекенд
    participant DB as База даних (MySQL)
    participant S3 as AWS S3
    participant Key as Ключ

    Note over Client: Процес прослуховування музики з HLS

    User->>Client: Обирає пісню
    Client->>Server: GET /api/music/{songId}/stream
    Note over Client: Запит на отримання URL HLS-плейлиста
    Server->>DB: SELECT song data
    DB-->>Server: Song info
    Server->>S3: GET original .m3u8 file
    S3-->>Server: Original .m3u8 content
    Note over Server: Парсинг .m3u8 файлу  
Модифікація посилань на .ts сегменти  
Додавання URL для ключів з авторизацією  
Збірка нового .m3u8 плейлиста
    Server-->>Client: Modified .m3u8 playlist
    Note over Client: Плейлист з модифікованими  
посиланнями на сегменти  
та URL для ключів
    Client->>Client: HLS.js парсить .m3u8 файл
    Client->>Key: GET /api/key?id={keyId}&token={auth_token}
    Note over Client: URL ключа з .m3u8 файлу
    Key-->>Client: Encryption key
    Client->>S3: GET first .ts segment
    S3-->>Client: Encrypted .ts segment
    Client->>Client: Дешифрування .ts сегмента
    Client->>Client: Відтворення аудіо
    Client->>User: Програвання музики
    loop [Для кожного наступного сегмента]
        Client->>S3: GET next .ts segment
        S3-->>Client: Encrypted .ts segment
        Client->>Client: Дешифрування сегмента
        Client->>Client: Продовження відтворення
    end
```

Процес починається з того, що користувач обирає пісню для прослуховування у браузері. Клієнтська частина надсилає GET-запит до

Laravel-бекенду для отримання потоку музики. Бекенд, отримавши запит, звертається до бази даних MySQL для отримання інформації про обрану композицію, після чого завантажує оригінальний .m3u8 файл плейлиста з AWS S3.

Наступним етапом є модифікація плейлиста. Система парсить отриманий .m3u8 файл та модифікує посилання на сегменти, додаючи підписані url-посилання для кожного з них. Це забезпечує контрольований доступ до контенту та запобігає несанкціонованому завантаженню. Результатом цього процесу є створення нового .m3u8 плейлиста з оновленими посиланнями, який повертається до клієнтської частини.

HLS.js бібліотека на клієнті парсить отриманий модифікований плейлист та починає процес відтворення. Для кожного аудіосегмента виконується послідовність операцій: спочатку система запитує ключ шифрування, якщо сегмент зашифрований, потім завантажує зашифрований .ts сегмент з AWS S3, виконує його дешифрування та передає для відтворення. Цей процес повторюється циклічно для кожного наступного сегмента, забезпечуючи безперервне відтворення музичного контенту.

Для створення користувачам персоналізованих добірок, система використовує автоматизований процес генерації рекомендацій. Цей процес не відбувається в реальному часі за запитом користувача, а виконується заздалегідь визначеним розкладом (раз на день), оптимізуючи навантаження.

Дана діаграма послідовності (див. рис. 2.6) детально ілюструє взаємодію між основними компонентами під час цього запланованого завдання. Вона показує, як планувальник завдань ініціює процес для кожного користувача, як Laravel взаємодіє з БД для отримання даних про прослуховування, викликає Python скрипт для обчислення рекомендацій на основі навченої моделі KNN, і зрештою, як створюються та зберігаються нові плейлисти з рекомендаціями для подальшого доступу користувачам.

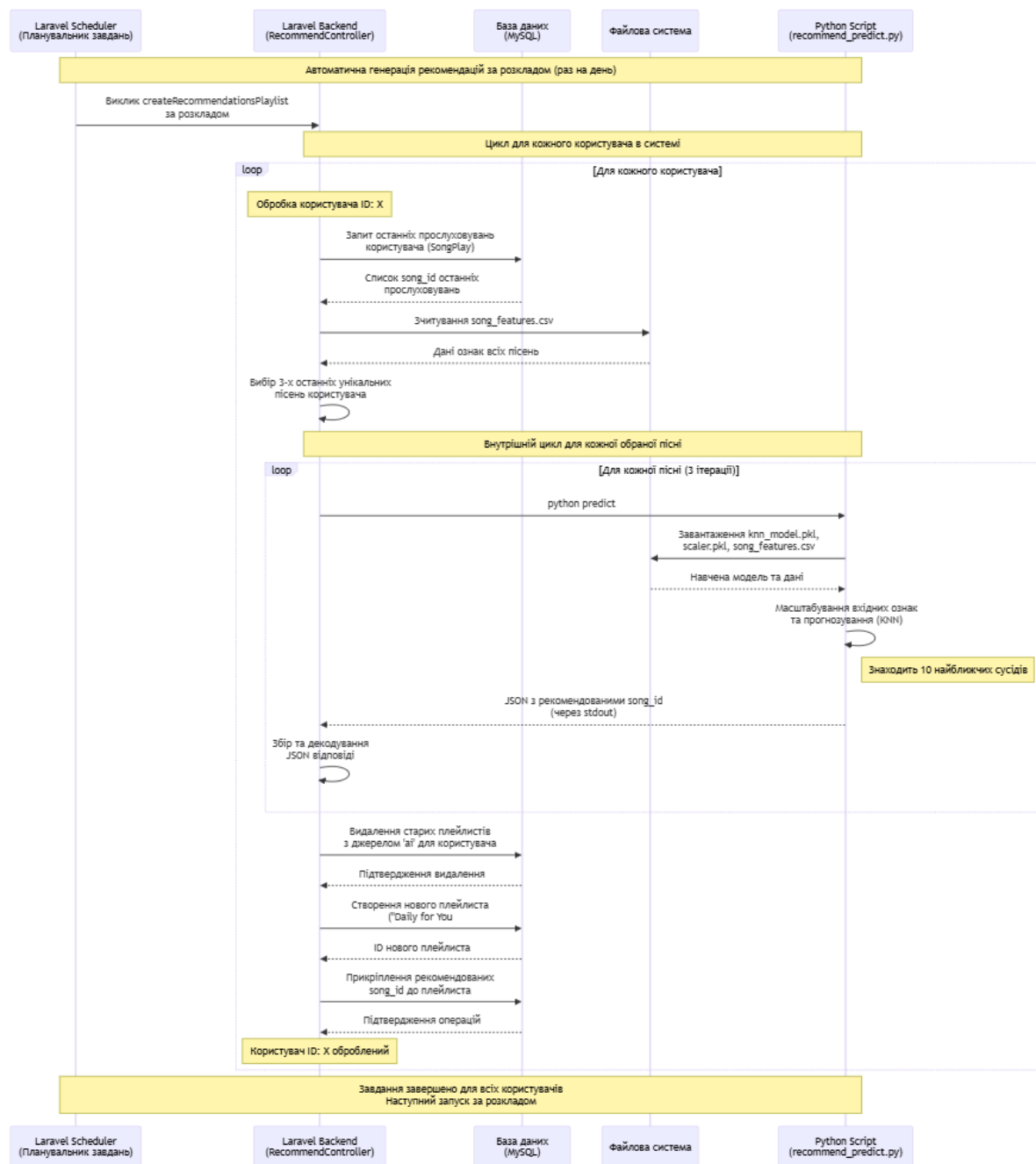


Рисунок 2.6 Діаграма послідовностей генерування рекомендацій

Процес починається з виконання запланованого завдання `createRecommendationsPlaylist` за розкладом через `Laravel Scheduler`. Система запускає цикл обробки для кожного користувача в системі. Для конкретного користувача `RecommendController` запитує останні прослуховування користувача з БД. Далі відбувається завантаження файлу `song_features.csv`, який містить дані про всі пісні, після чого система вибирає з останніх улюблених пісень користувача 3 пісні.

Наступним етапом є внутрішній цикл для кожної обраної пісні, де система виконує по три ітерації для кожної композиції. Для кожної пісні запускається Python скрипт predict, який завантажує ML модель, scaler та файл song_features.csv. Python скрипт виконує масштабування вхідних даних та прогнозування за допомогою KNN алгоритму. Система знаходить 10 найближчих сусідів та повертає JSON з рекомендованими піснями.

RecommendController збирає та декодує отримані JSON відповіді, після чого видаляє старі плейлисти з джерелом AI для користувача, створює новий плейлист типу "Мікс дня №" та додає до нього отримані рекомендації.

Вся система працює автоматично за розкладом, забезпечуючи регулярне оновлення персоналізованих рекомендацій для кожного користувача. Використання машинного навчання з алгоритмом KNN дозволяє системі аналізувати музичні характеристики та знаходити схожі композиції на основі попередніх вподобань користувача.

2.4 Реалізація функціоналу додатку

Система реалізована як розподілений вебдодаток з чітким розділенням на клієнтську та серверну частини, що взаємодіють через REST API.

Для навігації між різними сторінками додатку без повного перезавантаження сторінок у React було використано React-router-dom (див. лістинг 2.1.).

```
const router = createBrowserRouter([
  {
    path: '/',
    element: <RootLayout/>,
    children: [
      {index: true, element: <HomePage/>, loader: HomePageLoader},
      {path: '/playlist/:id', element: <PlaylistPage/>,
        loader: PlaylistPreLoader},
      {path: '/playlists', element: <PlaylistsPage/>,
        loader: AllPlaylistsLoader},
      {path: '/track/:id', element: <SongPage/>, loader: FetchSongData},
      {path: '/tracks', element: <MusicPage/>, loader: AllMusicLoader},
      {path: '/artist/:id', element: <ArtistPage/>, loader: FetchArtistData},
      {path: '/album/:id', element: <AlbumPage/>, loader: FetchAlbumData},
      {path: '/albums', element: <AlbumsPage/>, loader: AllAlbumsLoader},
      {path: '/liked', element: <LikedSongPage/>, loader: FetchLikedSongData},
      {path: '/my-artists', element: <ArtistSubscriptionList/>,
        loader: FetchArtistsData},
    ],
    errorElement: <ErrorLayout/> },
]);
```

```

{path: "/login", element: <PublicRoute><Login/></PublicRoute>},
{path: "/register", element: <PublicRoute><Register/></PublicRoute>},
{path: 'admin',
  element: <PrivateRoute roles={"admin"}><AdminLayout/></PrivateRoute>,
  children: [
    {index: true, element: <AnalyticsPage/>, loader: FetchAnalytData},
    {path: "song",
      children: [
        {index: true, element: <AdminSongsPage/>, loader: AdminMusicLoader},
        {path: "create", element: <AddSong/>, loader: FetchGenreData},
        {path: "update/:id", element: <UpdateSong/>, loader: SongPreLoader},]
      }
  ]
}

```

Лістинг 2.1 Використання React-router-dom для маршрутизації

Для аутентифікації використовується пакет Laravel Sanctum на основі API токенів. Під час логування створюється токен (ліст. 2.2), який записується в базу даних, і зберігається, доки користувач не вийде з облікового запису, або не вийде термін валідності токена.

```

public function login(Request $request)
{
    $validated = $request->validate([
        'email'=>'required|email|exists:users',
        'password'=>'required',
    ]);
    $user = User::where('email',$validated['email']->first();
    if(!$user || !Hash::check($validated['password'],$user->password)){
        return response()->json([
            'message' => 'The provided credentials are incorrect.'
        ],401);
    }
    $token = $user->createToken($user->name);
    return [
        'user'=>$user,
        'token'=>$token->plainTextToken,
    ];}

```

Лістинг 2.2 Генерація токена та відправка на клієнт

На стороні клієнта токен зберігається у localStorage. Під час використання сервісу, клієнт запитує доступ до роутів на основі раніше збереженого токена (ліст. 2.3).

```

async function getUser() {
    setIsLoading(true);
    await api.get('/user',{
        headers:{
            Authorization:`Bearer ${token}`,
        }
    }).then((response)=>{
        setUser(response.data);
        setIsLoading(false);
    });
}

```

```

    }).catch((error)=>{
        console.log(error);
        if (error.response && error.response.status === 401) {
            {
                logOut();
            }
        }
    }).finally(()=>{
        setIsLoading(false);
    })
    setIsLoading(false);
}

```

Лістинг 2.3 Get запит на роут /user з токеном в headers

На бекенді роути захищені за допомогою middleware('auth:sanctum').

Особлива увага приділена реалізації HLS доставки контенту та його захисту. Попередня обробка відбувається за допомогою бібліотеки Ffmpeg, де аудіофайли сегментуються на короткі .ts сегменти (ліст. 2.4), шифруються за допомогою AES-128 (див. ліст. 2.5.). Також створюється головний плейлист файл, що містить посилання на зашифровані сегменти та url для запиту ключа дешифрування. Усі HLS файли та оригінали пісень зберігаються в AWS S3 (див. ліст. 2.6.). Увесь код контролеру, який відповідає за функціональність та роботу з піснями, включно з створенням, збереженням HLS можна переглянути у додатку А.

```

private function createHlsFromAudio(string $inputPath, string $segmentPath,
string $outputPath, string $keyInfoPath): void
{
    $inputPathEscaped = escapeshellarg($inputPath);
    $segmentPathEscaped = $segmentPath;
    $outputPathEscaped = escapeshellarg($outputPath);
    $keyInfoPathEscaped = escapeshellarg($keyInfoPath);

    $command = "ffmpeg -i $inputPathEscaped -vn -c:a aac -b:a 320k -f hls
-hls_time 12 -hls_list_size 0 -hls_segment_filename $segmentPathEscaped -
hls_key_info_file $keyInfoPathEscaped $outputPathEscaped";
    exec($command);
}

```

Лістинг 2.4 Сегментація. Створення HLS файлів

```

private function prepareHlsEncryption(string $hlsPath, string $filename):
array
{
    mkdir($hlsPath, 0777, true);

    $keyPath = storage_path("app/hls_keys/{$filename}.key");
    $keyInfoPath = storage_path("app/temp/key info file.txt");
}

```

```

$keyUri = config('app.url2') . "/api/hls/token?id=$filename";
$segmentPath = "{$hlsPath}/audio_%03d.ts";
$outputPath = "{$hlsPath}/playlist.m3u8";

$key = random_bytes(16);
file_put_contents($keyPath, $key);

$keyInfoContent = "{$keyUri}\n{$keyPath}";
file_put_contents($keyInfoPath, $keyInfoContent);

return [
    'keyPath' => $keyPath,
    'keyInfoPath' => $keyInfoPath,
    'segmentPath' => $segmentPath,
    'outputPath' => $outputPath,
];
}

```

Лістинг 2.5 Створення ключа AES-128

```

$s3_hls_Path = "hls/{$filenameWithoutExtension}";
foreach (glob("#{localPath}/hls/*") as $file) {
    $filename = basename($file);
    Storage::disk('s3')->put("#{s3_hls_Path}/{ $filename}",
file_get_contents($file));
}

```

Лістинг 2.6 Збереження HLS в AWS S3

Для персоналізованих рекомендацій підготовка даних відбувається за допомогою консольної команди `app:generate-recommendation-dataset` (додаток Б), яка вивантажує ознаки всіх пісень з бази даних у CSV файл.

Навчання моделі виконується командою `app:generate-recommendations` (додаток В) яка запускає скрипт на виконання для завантаження даних, масштабування ознак, тренування моделі за допомогою KNN та збереження навченої моделі (див. ліст. 2.7).

```

import pandas as pd
from sklearn.neighbors import NearestNeighbors
from sklearn.preprocessing import StandardScaler
import pickle
#Load data
df = pd.read_csv('ai/recommendation_data.csv')
#Aggregate song features
features = df.groupby('song_id').agg({
    'genre_id': 'first',
    'duration': 'mean',
    'release_date': 'mean',
    'artist_id': 'first'
}).reset_index()
#Scale

```

```

scaler = StandardScaler()
X = scaler.fit_transform(features[['genre_id', 'duration', 'release_date',
'artist_id']])
#Train KNN
model = NearestNeighbors(n_neighbors=10, metric='euclidean')
model.fit(X)
with open('ai/knn_model.pkl', 'wb') as f:
    pickle.dump(model, f)
features.to_csv('ai/song_features.csv', index=False)
with open('ai/scaler.pkl', 'wb') as f:
    pickle.dump(scaler, f)

```

Лістинг 2.7 Навчання моделі

Генерація рекомендацій для користувачів відбувається за розкладом за допомогою Laravel Scheduler (додаток Г), який для кожного користувача при створенні плейлистів викликає метод recommendSongs (додаток Д), який у свою чергу отримує останні прослухані пісні користувача, з яких бере ознаки, та на основі цього за допомогою Python скрипту прогнозує рекомендовані пісні (ліст. 2.8).

```

import sys
import json
import pandas as pd
import pickle
from sklearn.neighbors import NearestNeighbors
import os
script_dir = os.path.dirname(os.path.realpath(__file__))
if len(sys.argv) < 5:
    print("[]")
    sys.exit(0)
genre_id = int(sys.argv[1])
duration = float(sys.argv[2])
release_date = float(sys.argv[3])
artist_id = int(sys.argv[4])
X_input = [[genre_id, duration, release_date, artist_id]]
#model
with open(os.path.join(script_dir, 'knn_model.pkl'), 'rb') as f:
    model = pickle.load(f)

with open(os.path.join(script_dir, 'scaler.pkl'), 'rb') as f:
    scaler = pickle.load(f)
features = pd.read_csv(os.path.join(script_dir, 'song_features.csv'))
X_scaled = scaler.transform(X_input) #scale
distances, indices = model.kneighbors(X_scaled, n_neighbors=10)
recommended_ids = features.iloc[indices[0]]['song_id'].astype(int).tolist()
print(json.dumps(recommended_ids))

```

Лістинг 2.8 Метод прогнозування

Скрипт реалізує механізм генерації рекомендацій для музичного сервісу на основі моделі k-найближчих сусідів. На вхід подаються ознаки пісні – жанр, тривалість, рік випуску та ідентифікатор виконавця. Дані масштабуються з використанням попередньо збереженого скейлера, після чого модель знаходить 10 найближчих за ознаками пісень. У результаті скрипт повертає список ідентифікаторів рекомендованих треків у форматі JSON.

Висновок до другого розділу

У другому розділі було детально представлено архітектурне та функціональне проєктування, визначаючи ключові компоненти та взаємодії системи. Було розроблено варіанти використання системи, де чітко окреслено ролі користувача, адміністратора, а також типові сценарії їхньої взаємодії з сервісом.

Класова діаграма та структура бази даних відображають об'єктно-орієнтований підхід до моделювання. Запропонована модель даних є гнучкою та масштабованою, що забезпечує ефективне збереження й обробку основної інформації платформи. Така архітектура дозволяє надійно реалізувати ключові функції взаємодії користувача та контенту.

Ключові алгоритми роботи системи, такі як, автентифікація, прослуховування музики та генерація персональних рекомендацій, були детально окреслені за допомогою діаграм активності та послідовностей. Це дозволило візуалізувати логіку взаємодії між компонентами системи та її користувачами, особливо на прикладі складних процесів HLS транслювання контенту (включаючи сегментацію, шифрування та динамічну видачу ключів) та механізму генерації рекомендацій на основі KNN-алгоритму.

Загалом, рішення представлені у цьому розділі, демонструють цілісну та продуману архітектуру сервісу, що поєднує сучасні технології, хмарні рішення та методи машинного навчання. Такий підхід забезпечує не лише високу ефективність та безпеку функціонування платформи, але й створює міцну основу для її подальшого розвитку та розширення функціоналу.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ

3.1 Архітектура та план розгортання сервісу

Сервіс для прослуховування музики передбачає використання трирівневої архітектури, що забезпечує чітке розділення відповідальності та ефективну взаємодію між компонентами системи. Діаграму розгортання можна побачити на рис. 3.1.

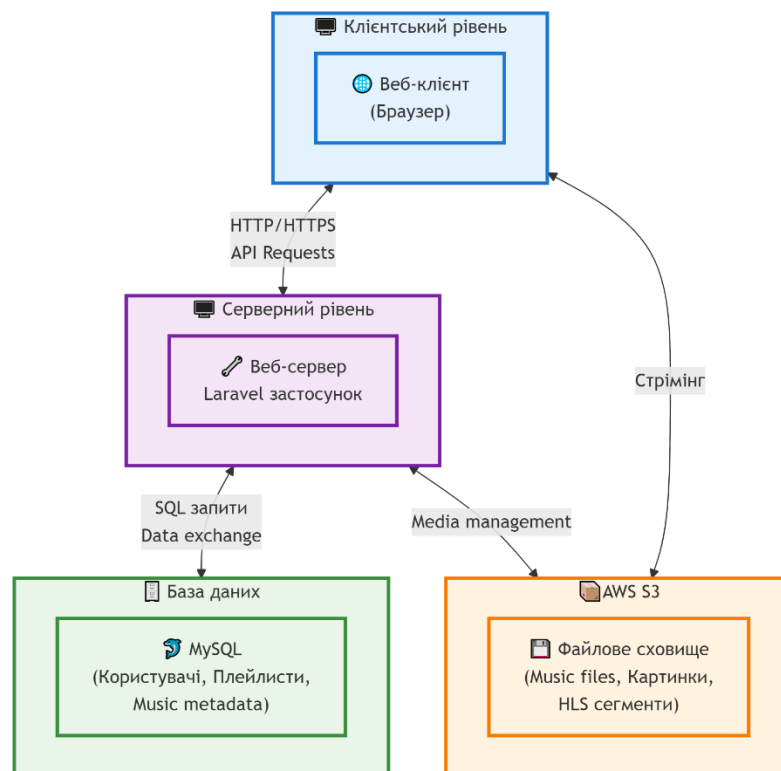


Рисунок 3.1 Діаграма розгортання

На клієнтському рівні знаходиться вебклієнт (браузер), який є безпосереднім інтерфейсом для користувачів. Всі взаємодії між вебклієнтом та серверною частиною відбуваються за допомогою HTTP/HTTPS API запитів.

Серверний рівень представлений вебсервером, на якому розгорнуто Laravel застосунок. Цей рівень є центральним компонентом системи, відповідальним за обробку бізнес-логіки та керування даними. Laravel застосунок виконує SQL запити та обмін даними для взаємодії з базою даних, яка реалізована на MySQL.

Окрім реляційної бази даних, для зберігання великих медіа-файлів використовується хмарне сховище на базі AWS S3. Тут зберігаються музичні

файли, картинки та HLS сегменти. Особливістю є те, що стрімінг HLS-контенту відбувається безпосередньо з AWS S3 до вебклієнта, оптимізуючи доставку медіа. Також, Laravel застосунок відповідає за управління медіа в AWS S3 (завантаження, видалення, організація файлів).

Така архітектура дає змогу забезпечити масштабованість, ефективне розподілення навантаження та високу доступність сервісу. Її реалізація запланована як наступний етап після завершення функціональної частини розробки.

3.2 Структура інтерфейсу

Для використання усіх функцій сервісу, користувач має бути обов'язково зареєстрований та автентифікований. Тому під час переходу на сайт, користувач відразу потрапляє на сторінку автентифікації або реєстрації (рис. 3.2).

Рисунок. 3.2 Автентифікація

Після того як користувач був автентифікований, він потрапляє на головну сторінку, де можна побачити нещодавно додані адміністратором плейлисти, новинки пісень, персоналізовані плейлисти, форму для фідбеку (рис. 3.3-3.4). Також внизу екрану можна побачити аудіоплеєр (рис. 3.6).

За допомогою плеєра користувач може програвати музику, змінювати позицію, регулювати звук, перемішувати чергу, перемикає пісні вперед/назад, ставити на паузу/програвати, зациклювати, переглядати

інформацію про пісню у додатковому вікні, переглянути текст пісні. Під час використання деяких функцій, колір кнопок змінюється на зелений.

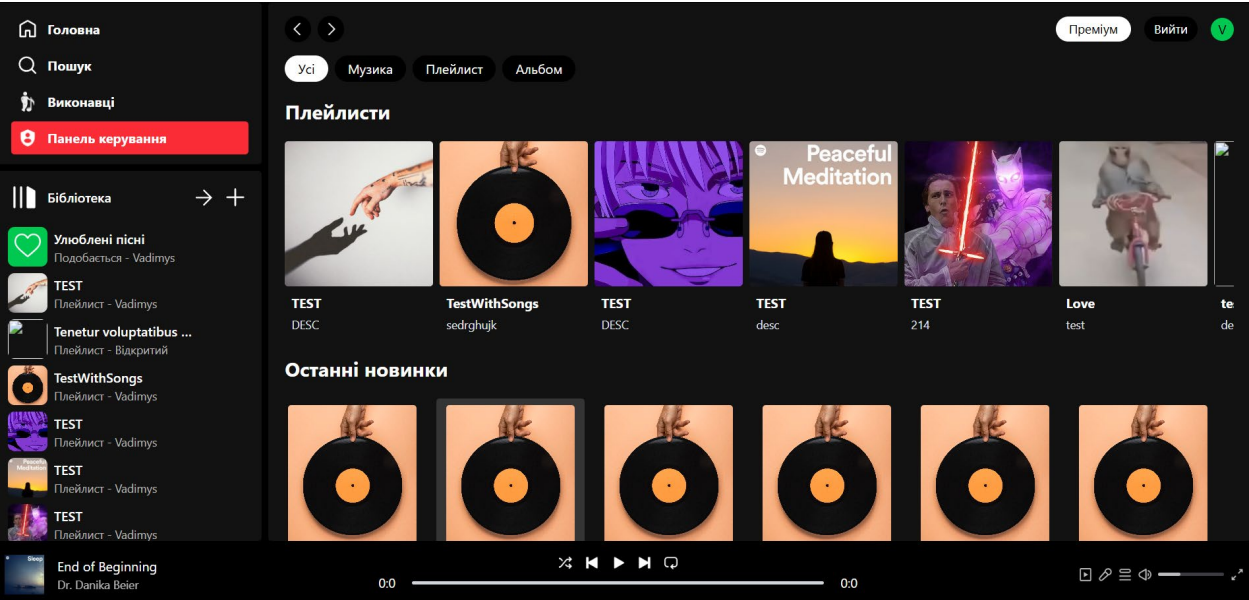


Рисунок 3.3 Головна сторінка

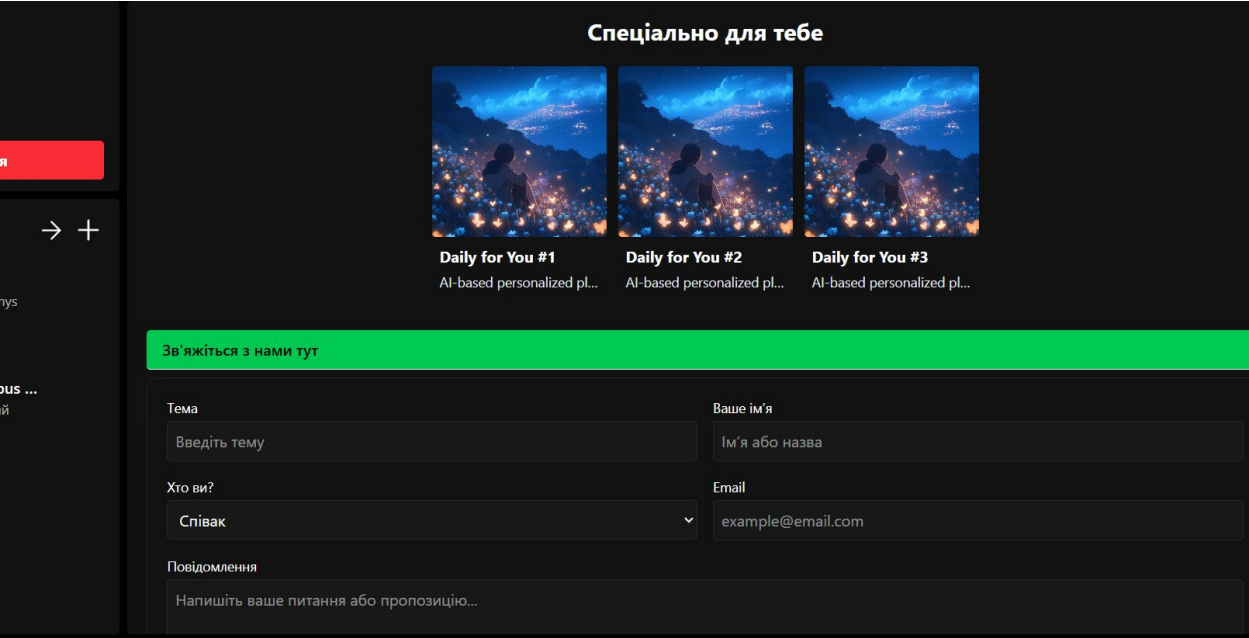


Рисунок 3.4 Головна сторінка, персоналізовані плейлисти, форма

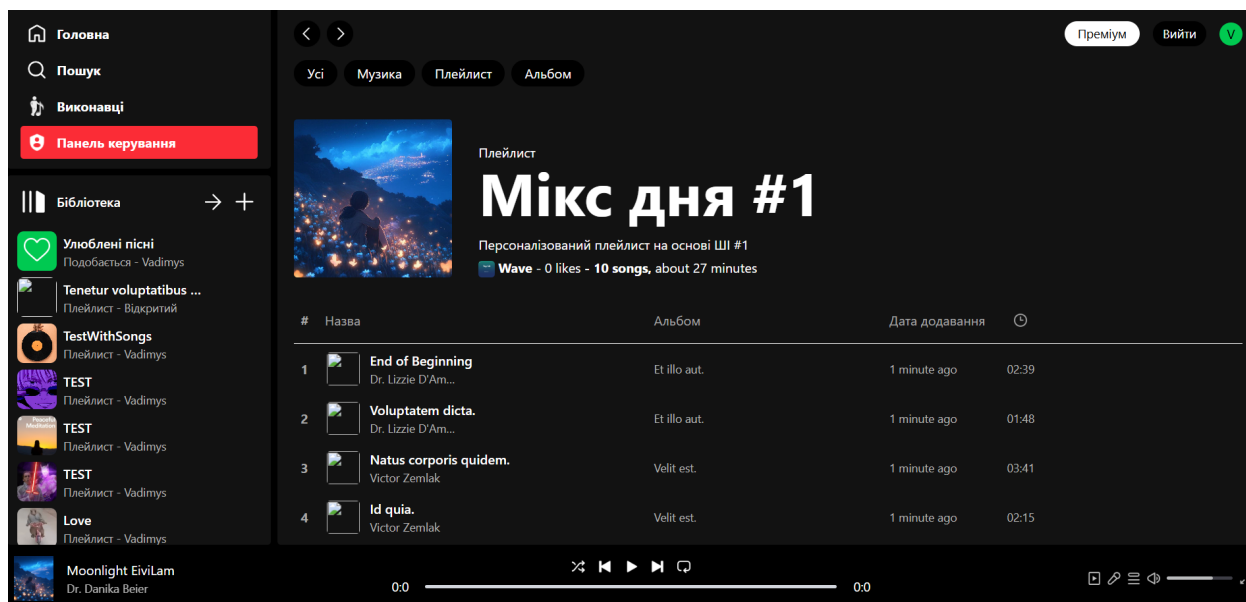


Рисунок 3.5 Перегляд персоналізованого плейлиста

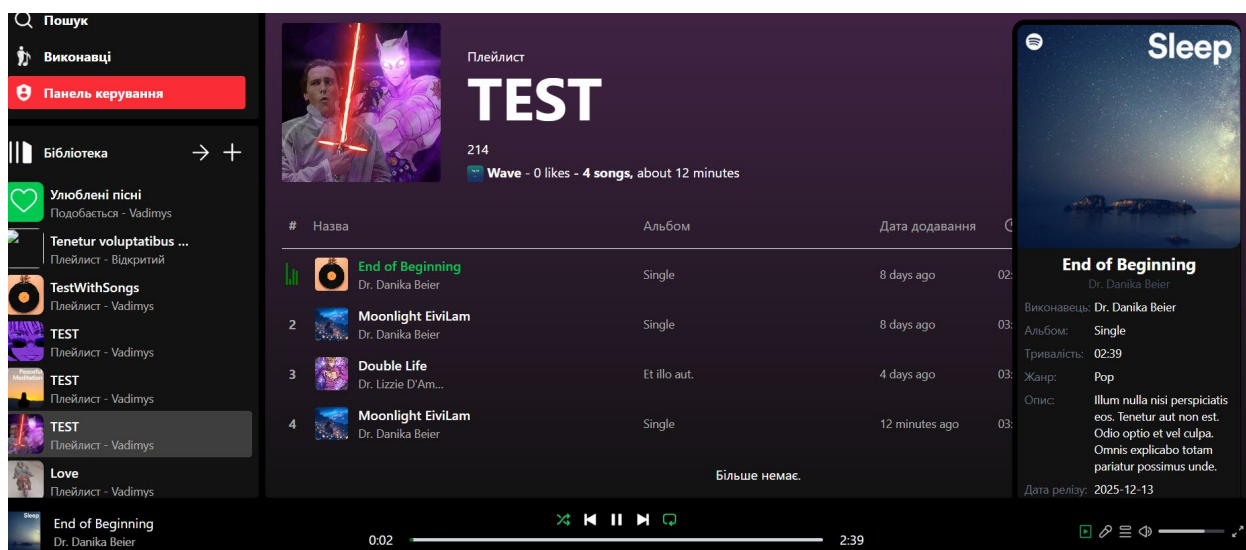


Рисунок 3.6 Аудіоплеєр

У меню зліва розташовані кнопки для швидкого доступу до головної сторінки, пошуку, списку виконавців, на які підписаний користувач. Для ролі «Адмін» доступна червона кнопка, яка веде до панелі керування. Кнопка «+» призначена для додавання особистих плейлистів. У списку нижче доступний список улюблених пісень з зеленим серцем та плейлисти користувача, створені власноруч, або додані. Сортування відбувається за датою додавання. Детальніше на рис. 3.7.

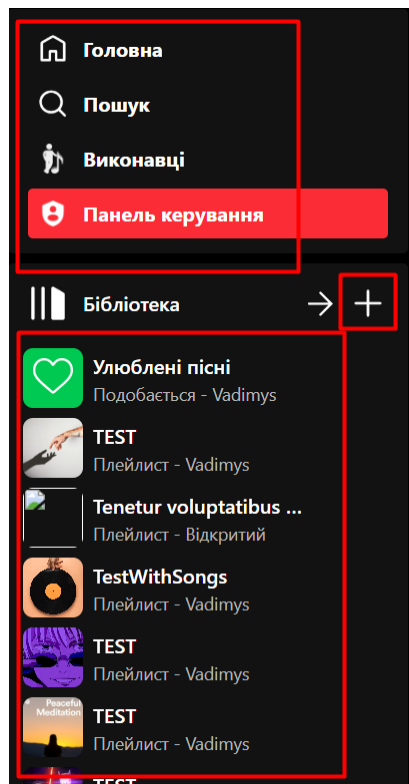


Рисунок 3.7 Ліве меню

Під час натискання кнопки «Пошук», користувач вводить в поле пошуковий запит, після чого відбувається пошук найкращого результату, а також вибірка з таблиць пісень, виконавців, альбомів, плейлистів за схожим значенням (рис. 3.8).

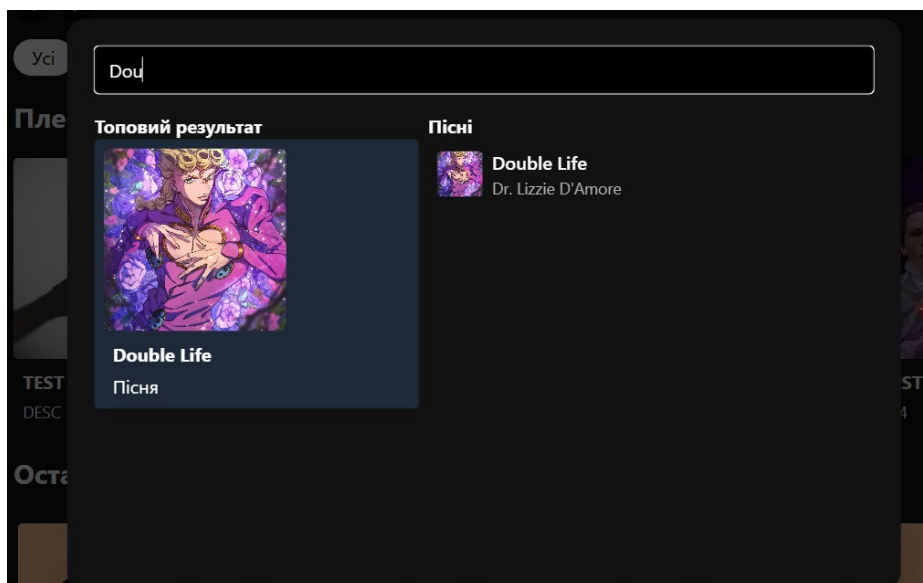


Рисунок 3.8 Загальний пошук

На сторінці виконавців користувач може бачити, видаляти доданих ним виконавців (рис. 3.9).

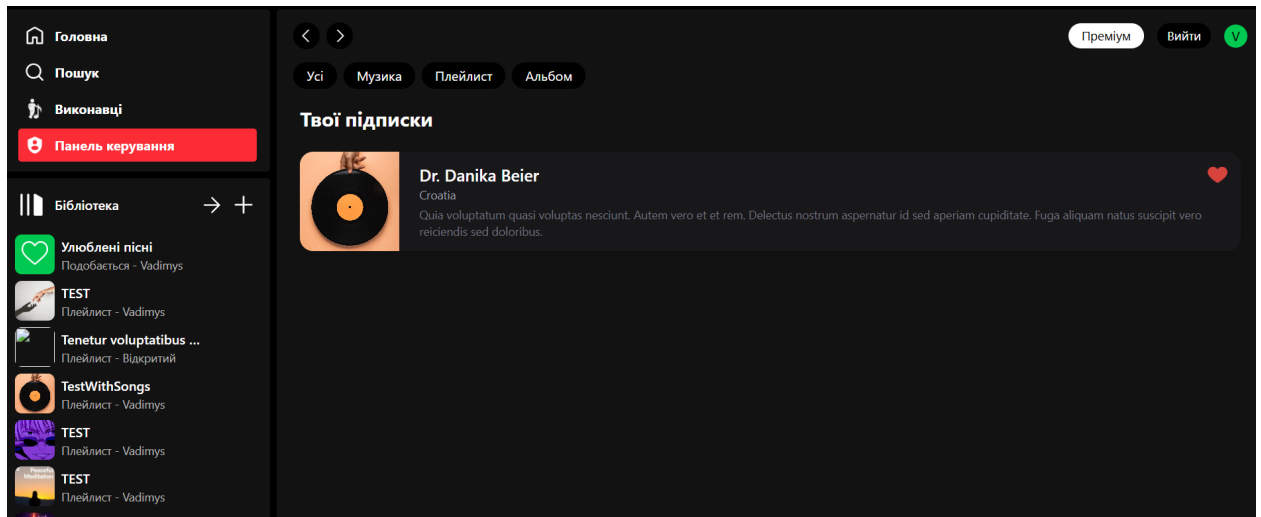


Рисунок 3.9 Список улюблених виконавців

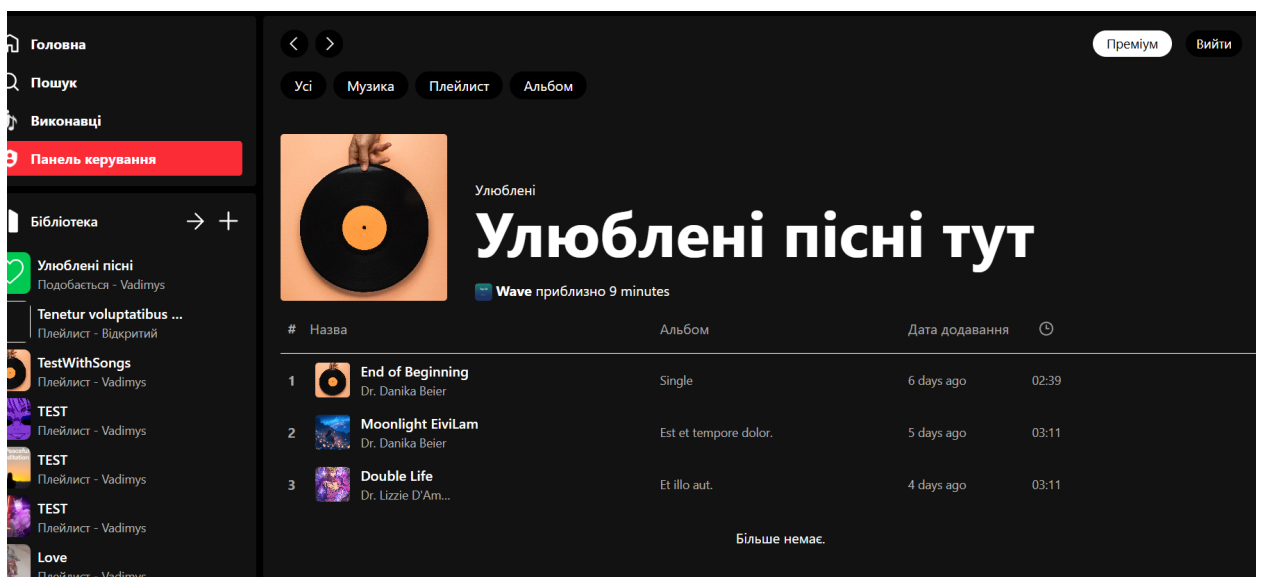


Рисунок 3.10 Список улюблених пісень

Редагування особистих плейлистів (рис. 3.11-3.14).

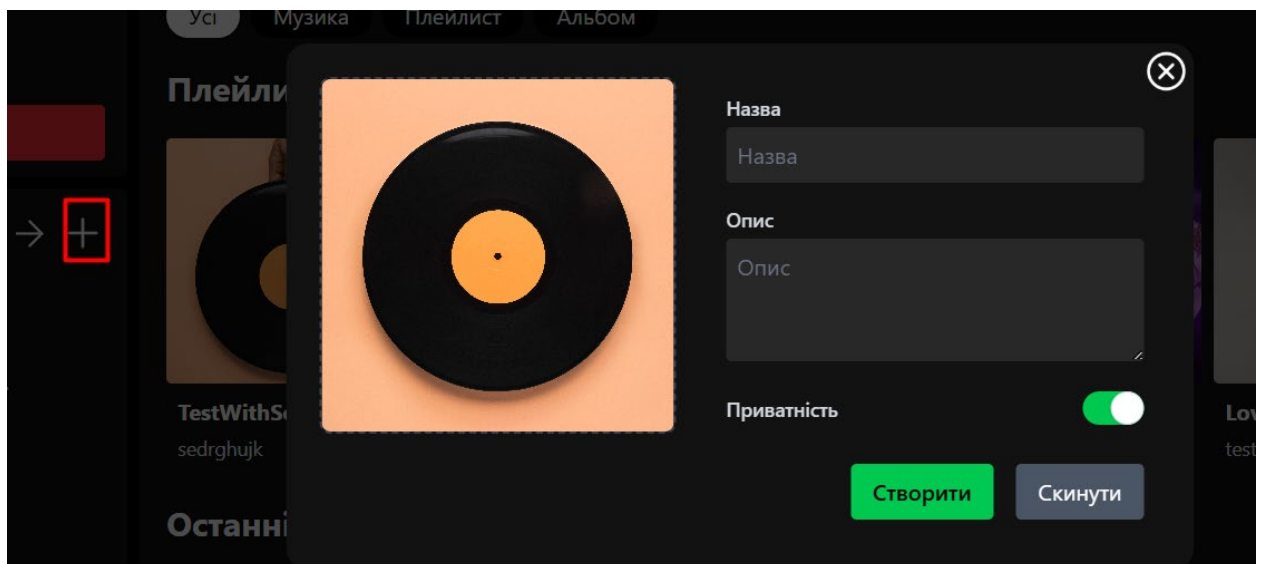


Рисунок 3.11 Створення плейлиста

					ИЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						49

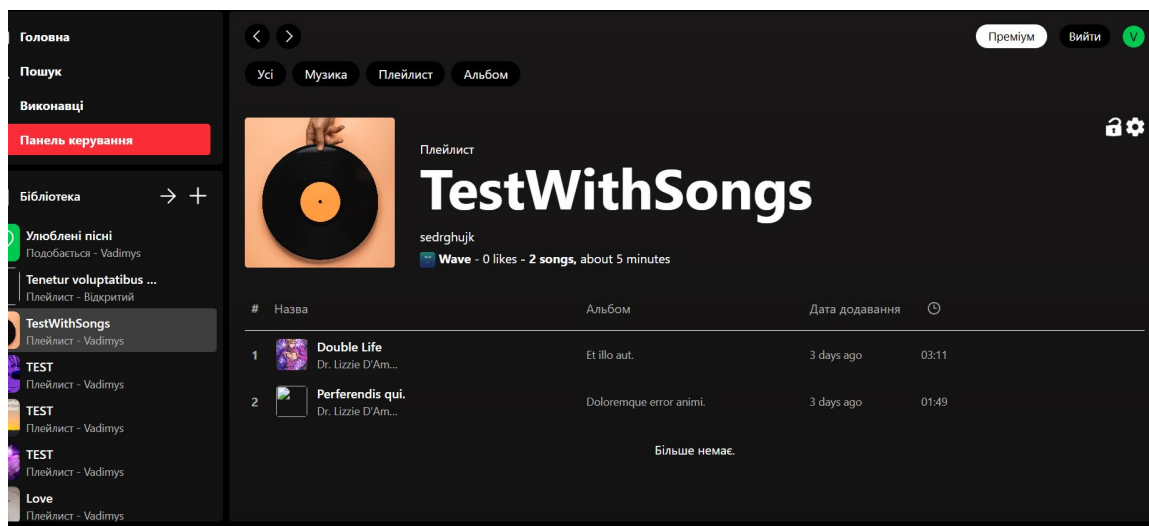


Рисунок 3.12 Перегляд плейлиста

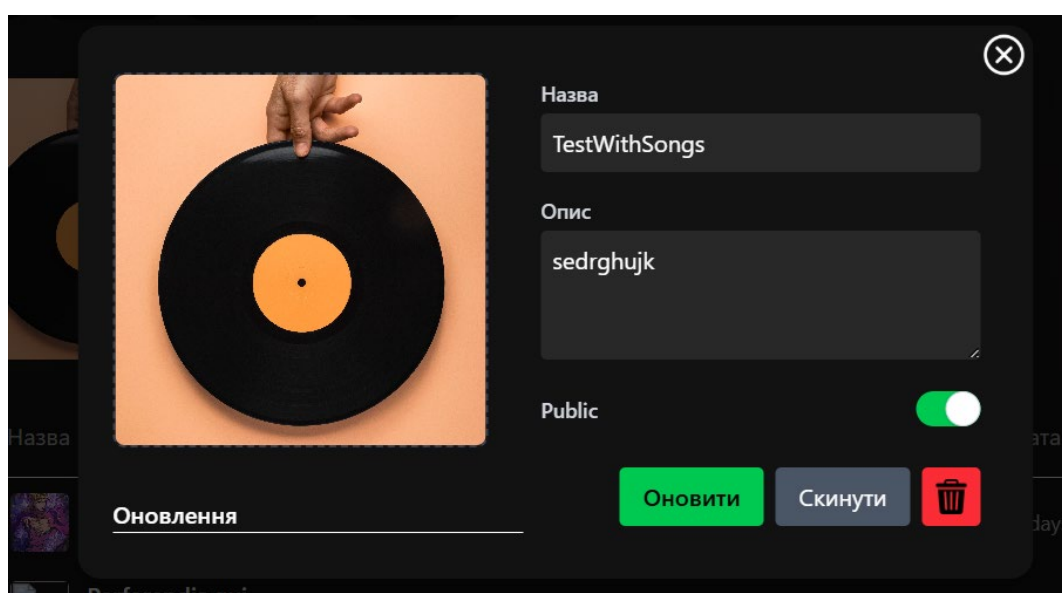


Рисунок 3.13 Оновлення плейлиста

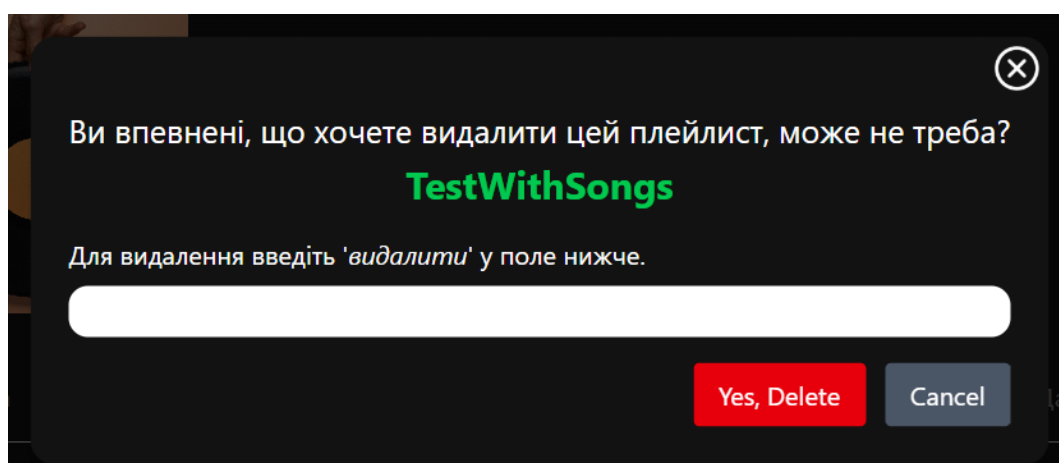


Рисунок 3.14 Підтвердження видалення плейлиста

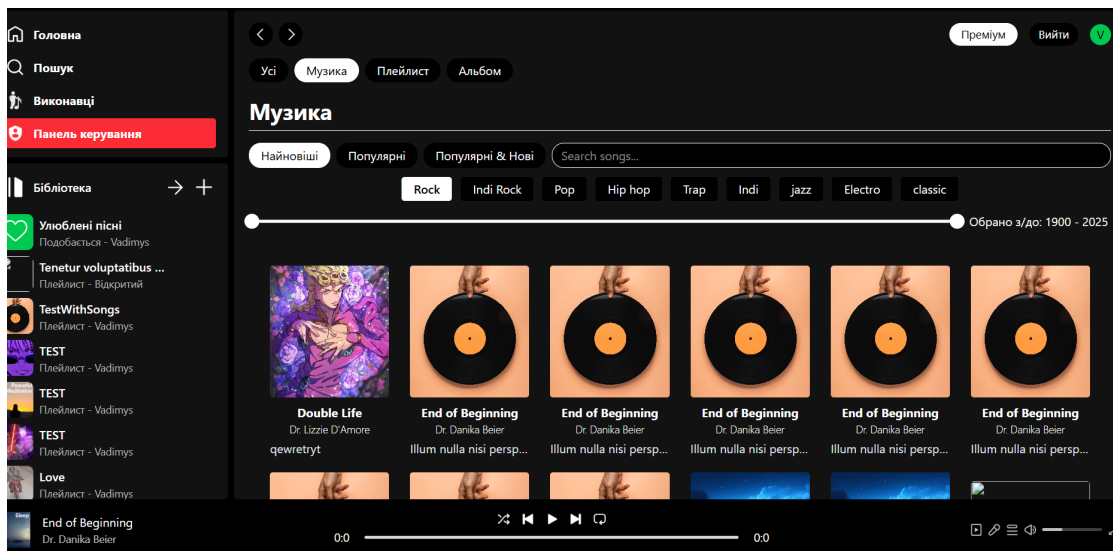


Рисунок 3.15 Окремий перегляд пісень, фільтрація пісень

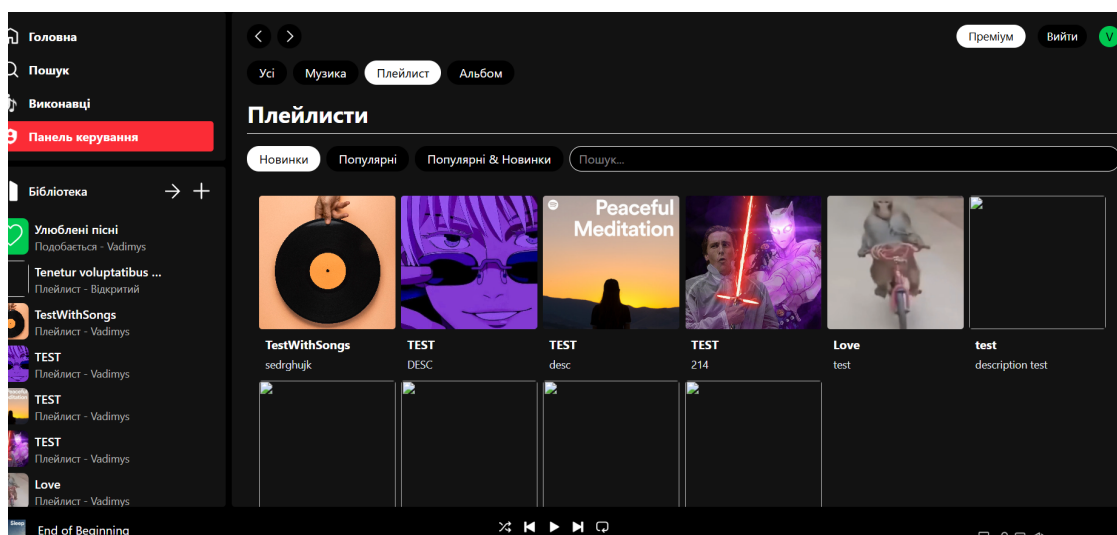


Рисунок 3.16 Окремий перегляд плейлистів, фільтрація плейлистів

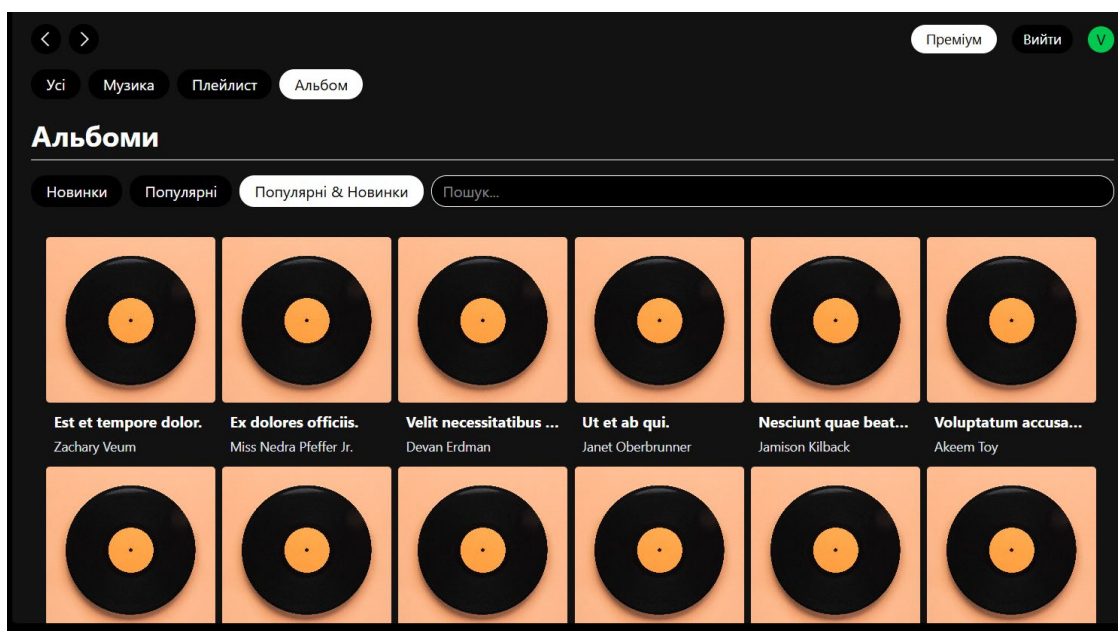


Рисунок 3.17 Окремий перегляд альбомів, фільтрація альбомів

					ИЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						51

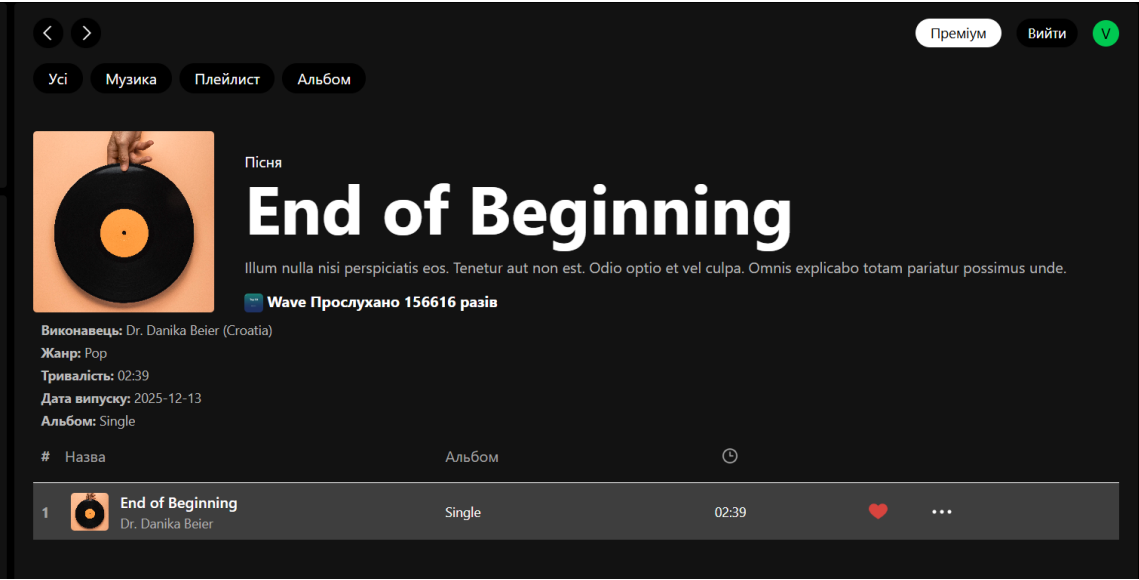


Рисунок 3.18 Перегляд детальної інформації про пісню

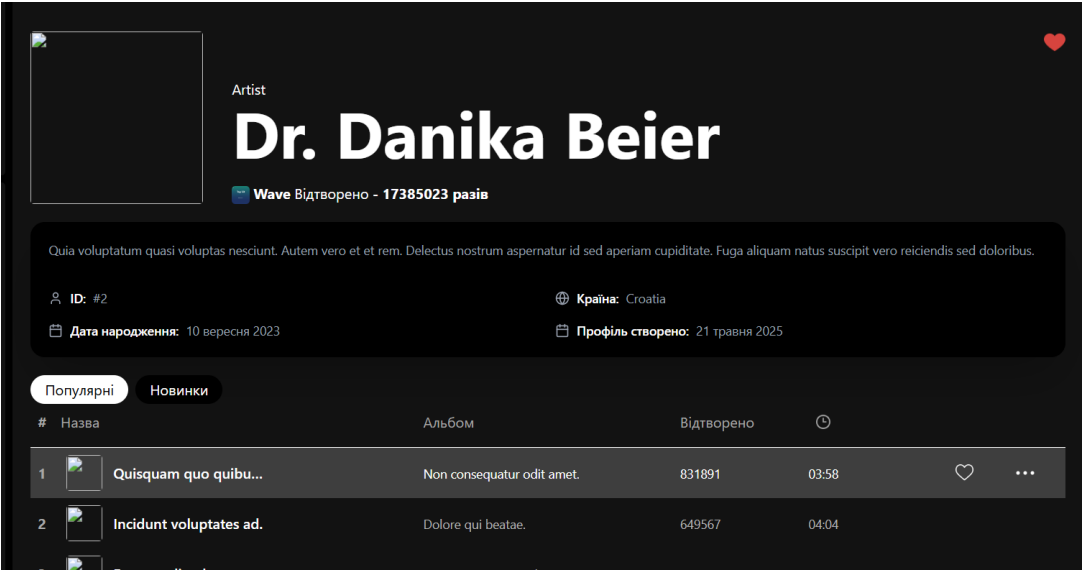


Рисунок 3.19 Перегляд сторінки виконавця

На будь-якій сторінці де розміщується списки пісень з InfiniteScroll – можливо викликати меню за допомогою кнопки, яка з’являється при наведенні на пісню. У меню доступні такі функції (рис. 3.20):

1. При наведенні на «Додати до плейлиста» відкривається ще одна панель з плейлистами користувача, по яким можна робити пошук та при натисканні на один з плейлистів, пісня додається до нього.
2. Видалити з плейлиста, якщо плейлист належить користувачу.
3. Переглянути виконавця пісні.
4. Показати деталі пісні.
5. Поділитись піснею.

6. Додати до списку улюблених.

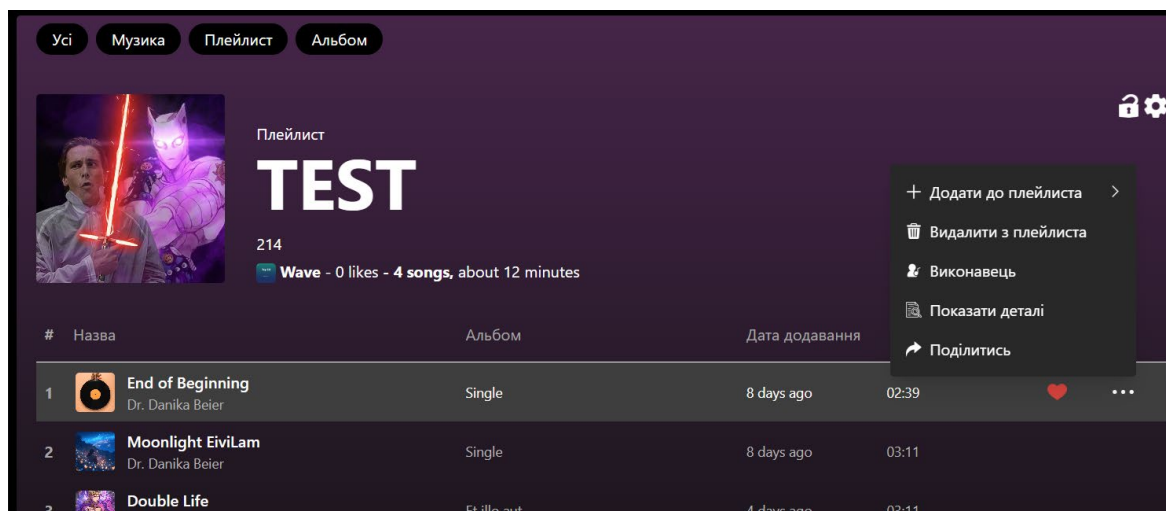


Рисунок 3.20 Меню пісні

Адміністратор має можливість редагувати, наповнювати, видаляти медіа контент сервісу, а також переглядати статистику (рис. 3.21).

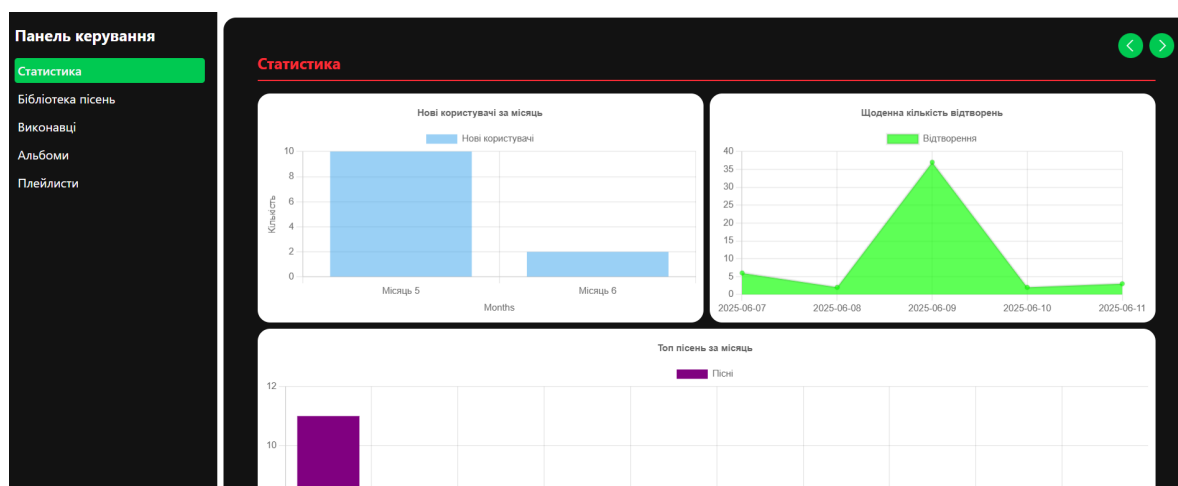


Рисунок 3.21 Адмін. Панель керування

Приклад редагування, наповнення, видалення контенту (рис. 3.22-3.24).

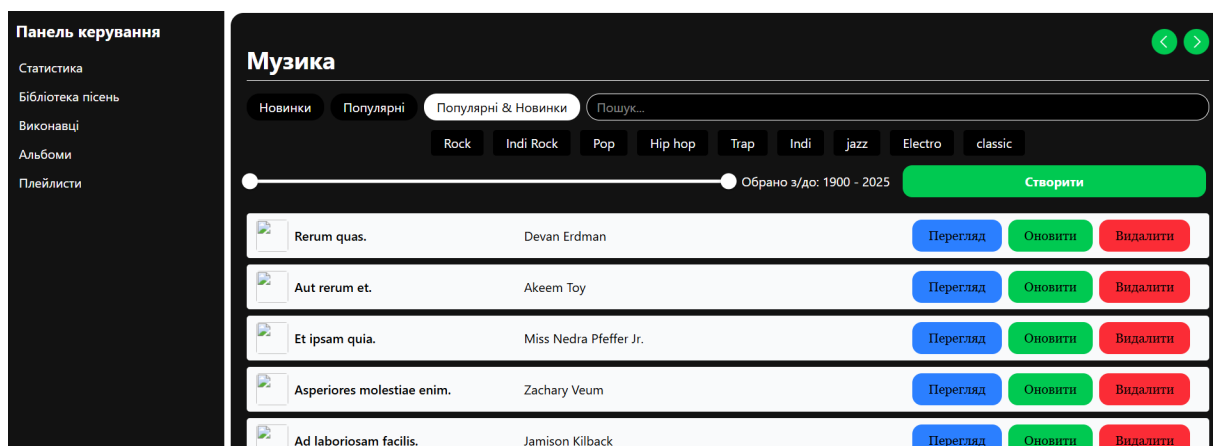


Рисунок 3.22 Адмін. Приклад перегляду, пошуку контенту

Створити пісню

Основна інформація

Назва

Назва

Пошук виконавця

Почніть вводити ім'я...

Альбом*

Виберіть альбом

Деталі

Дата випуску

13.12.2025

Жанр

Обрати жанр

Лірика

Enter song lyrics...

Медіа

Зображення

Вибрати файл

Файл не вибрано

Audio файл

Вибрати файл

Файл не вибрано

Опис

Enter song description...

Створити

Скинути

Рисунок 3.23 Адмін. Створення контенту

Музика

Новинки

Популярні

Популярні & Новинки

Пошук...

Rock

Indi Rock

Pop

Hip hop

Trap

Indi

jazz

Electro

classic

Обрано з/до: 1900 - 2025

Створити

Rerum quas.

Чи справді Ви хочете видалити пісню?

Так

Ні

Aut rerum et.

Перегляд

Оновити

Видалити

Et ipsam quia.

Miss Nedra Pfeffer Jr.

Перегляд

Оновити

Видалити

Asperiores molestiae enim.

Zachary Veum

Перегляд

Оновити

Видалити

Рисунок 3.24 Адмін. Видалення контенту

3.3 Тестування роботи програмного продукту

Тестування платформи здійснювалося із застосуванням різноманітних методів, що підтвердило її стабільність, коректність обробки запитів та відповідей, а також надійність взаємодії між клієнтською та серверною частинами. Перевірка серверної частини проводилася за допомогою менеджера запитів Postman, який забезпечив можливість надсилати запити до окремих ендпоінтів та зручно аналізувати отримані відповіді, контролюючи правильність обробки вхідних даних. Для автоматизації тестування захищених ендпоінтів було розроблено спеціальний авторизаційний скрипт. Цей скрипт виконує вхід користувача, зберігає отриманий токен у

середовище Music у Postman та автоматично додаває його до заголовків наступних запитів (рис. 3.25).

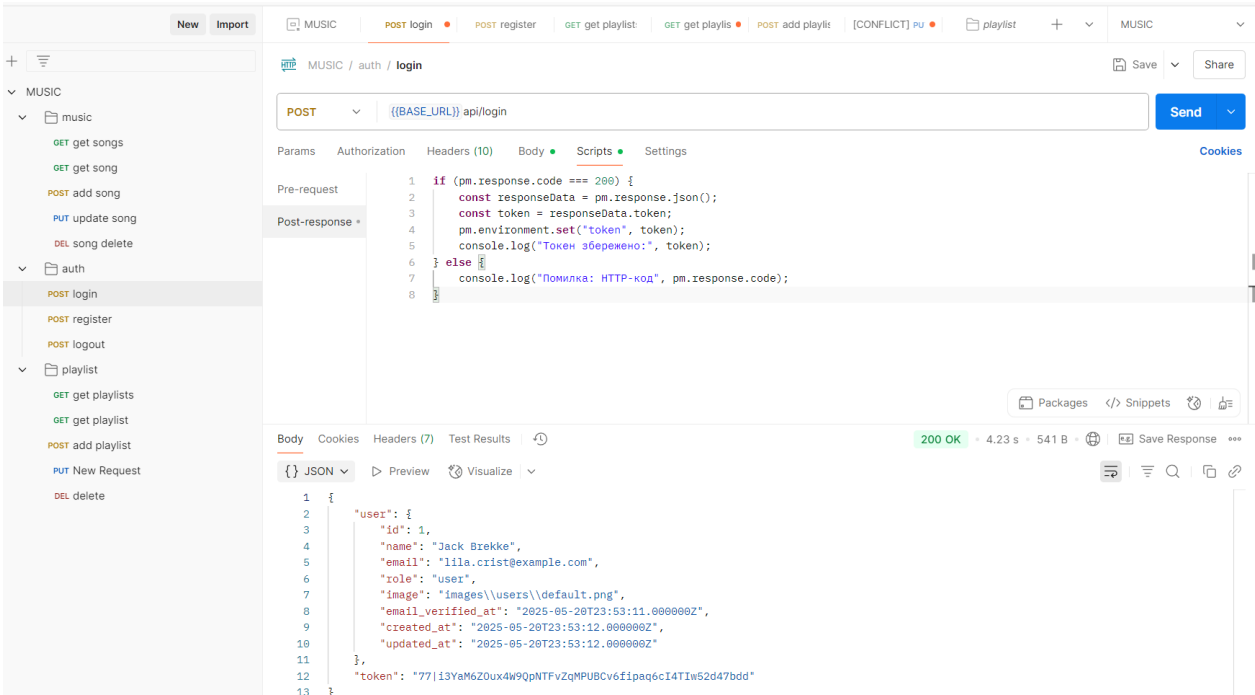


Рисунок 3.25 Скрипт для автентифікації у Postman

Для перевірки клієнтської частини платформи застосовувалося мануальне тестування, яке охоплювало всі ключові сценарії використання вебсайту. Зокрема автентифікація, прослуховування музики, створення та редагування плейлистів, пошук контенту, а також керування вподобаними треками та підписки на виконавців.

В результаті цього дослідження були виявлені та оброблені помилки, що виникали, щоб користувач міг побачити, що саме пішло не так. Для усіх форм вебсайту були написані упр-схеми, що ефективно валідують усі поля та збирають помилки для їх відображення. Приклад цих помилок при невдачі валідації можна побачити на сторінці додавання нової пісні на рисунку 3.26.

Рисунок 3.26 Валідація форми для створення пісні за допомогою уир схеми

Висновки до третього розділу

У цьому розділі було розглянуто технічну реалізацію вебсервісу, побудованого за принципами трирівневої архітектури. Детально описано особливості та план розгортання платформи, включаючи взаємодію клієнтської та серверної частини, обробку даних на стороні Laravel, а також використання MySQL та хмарного сховища AWS S3 для збереження структурованих даних і медіа-контенту. Така архітектура забезпечує надійність, масштабованість і зручність підтримки системи. Інтерфейс платформи продуманий для забезпечення інтуїтивної взаємодії з користувачем. Особливу увагу приділено взаємодії користувача з контентом та можливості гнучкого керування аудіовідтворенням. Також було реалізовано повноцінний функціонал для адміністратора, який дає змогу керувати вмістом сервісу й переглядати аналітику.

Проведене тестування як серверної, так і клієнтської частини дозволило переконатися в коректній роботі основних функцій, стійкості системи та зручності використання. Автоматизація авторизації в Postman, ручне тестування сценаріїв користувача та валідація форм за допомогою схем уир допомогли підвищити надійність платформи та покращенню користувацького досвіду. Реалізований функціонал створює надійну основу для подальшого масштабування та розвитку вебсервісу.

ВИСНОВКИ

Під час виконання випускної кваліфікаційної роботи бакалавра було розроблено вебсервіс для прослуховування музичного контенту із використанням сучасних інструментів розробки ПЗ.

У першому розділі було проведено аналіз аналогів музичних сервісів, що дало змогу виокремити ключові функціональні вимоги до майбутньої системи. Особливу увагу було приділено можливості інтеграції ШІ-інструментів для генерації персоналізованих рекомендацій та реалізації надійної системи HLS-доставки та захисту контенту. На основі порівняльного аналізу технологій було обґрунтовано вибір клієнт-серверної архітектури, а також стеку технологій React.js, Laravel, MySQL, AWS S3. Обґрунтовано вибір інструментальних засобів з урахуванням масштабованості, продуктивності та безпеки.

У другому розділі було зосереджено увагу на архітектурному та функціональному проєктуванні платформи. Розроблено діаграми, які демонструють логіку роботи системи. Ключові алгоритми роботи системи, такі як, автентифікація, прослуховування музики та генерація персональних рекомендацій, були детально окреслені за допомогою діаграм активності та послідовностей.

У третьому розділі налагоджено взаємодію між фронтендом і бекендом, Python-скриптами для рекомендацій, а також файловим сховищем AWS S3. Реалізовано повноцінний інтерфейс із можливістю реєстрації та автентифікації користувачів, прослуховування музики, створення та редагування плейлистів, та підписок на виконавців/плейлисти, що формує способи того, як користувач може взаємодіяти з сервісом. Здійснено тестування як серверної логіки, так і клієнтської частини, що дозволило виявити та усунути критичні помилки, перевірити продуктивність та відповідність очікуванням користувачів.

Результатом кваліфікаційної роботи є готовий сервіс для прослуховування музичного контенту.

ДЖЕРЕЛА

1. Офіційний сайт Spotify [Електронний ресурс] – Режим доступу до ресурсу: <https://www.spotify.com>
2. Офіційний сайт Apple Music [Електронний ресурс] – Режим доступу до ресурсу: <https://www.apple.com/music>
3. Freemium: Definition, Examples, and Pros & Cons for Business [Електронний ресурс] – Режим доступу до ресурсу: <https://www.investopedia.com/terms/f/freemium.asp>
4. HTTP Live Streaming [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.apple.com/streaming/>
5. Офіційний сайт Apple [Електронний ресурс] – Режим доступу до ресурсу: <https://www.apple.com/>
6. What Is a REST API? Examples, Uses, and Challenges / Postman Blog [Електронний ресурс] – Режим доступу до ресурсу: <https://blog.postman.com/rest-api-examples/>
7. Working with JSON [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/enUS/docs/Learn_web_development/Core/Scripting/JSON
8. Офіційна документація React [Електронний ресурс] – Режим доступу до ресурсу: <https://react.dev/learn>
9. Офіційна документація JavaScript [Електронний ресурс] – Режим доступу до ресурсу: <https://devdocs.io/javascript/>
10. Офіційна документація Angular [Електронний ресурс] – Режим доступу до ресурсу: <https://angular.dev/>
11. Офіційна документація Vue.js [Електронний ресурс] – Режим доступу до ресурсу: <https://vuejs.org/>
12. What is Single Page Application [Електронний ресурс] – Режим доступу до ресурсу: <https://startup-house.com/glossary/what-is-single-page-application-spa-routing>

13. Офіційна документація React Router DOM [Електронний ресурс] – Режим доступу до ресурсу: <https://reactrouter.com>
14. Офіційна документація YUP [Електронний ресурс] – Режим доступу до ресурсу: <https://www.npmjs.com/package/yup>
15. Офіційна документація HLS.js [Електронний ресурс] – Режим доступу до ресурсу: <https://www.npmjs.com/package/hls.js?activeTab=readme>
16. Офіційна документація Tailwind CSS [Електронний ресурс] – Режим доступу до ресурсу: <https://tailwindcss.com/docs>
17. Cascading Style Sheets CSS [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/CSS>
18. Офіційний сайт Bootstrap [Електронний ресурс] – Режим доступу до ресурсу: <https://getbootstrap.com/>
19. Офіційна документація Axios [Електронний ресурс] – Режим доступу до ресурсу: <https://axios-http.com/docs/intro>
20. Офіційна документація Laravel [Електронний ресурс] – Режим доступу до ресурсу: <https://laravel.com/>
21. Офіційна документація PHP [Електронний ресурс] – Режим доступу до ресурсу: <https://www.php.net/docs.php>
22. Офіційна документація ORM Eloquent [Електронний ресурс] – Режим доступу до ресурсу: <https://laravel.com/docs/12.x/eloquent>
23. Офіційна документація Laravel Sanctum [Електронний ресурс] – Режим доступу до ресурсу: <https://laravel.com/docs/12.x/sanctum>
24. Офіційна документація MySQL [Електронний ресурс] – Режим доступу до ресурсу: <https://www.mysql.com/>
25. Офіційний сайт Amazon S3 AWS [Електронний ресурс] – Режим доступу до ресурсу: <https://aws.amazon.com/s3/>
26. AES [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard
27. Офіційний сайт Ffmpeg [Електронний ресурс] – Режим доступу до ресурсу: <https://ffmpeg.org/>

28. Офіційний сайт Python [Електронний ресурс] – Режим доступу до ресурсу: <https://www.python.org/>
29. KNN algorithm [Електронний ресурс] – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/k-nearest-neighbours/>
30. Range Streaming [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/enUS/docs/Web/HTTP/Guides/Range_requests
31. Документація до Postman [Електронний ресурс] – Режим доступу до ресурсу: <https://learning.postman.com/docs/introduction/overview/>
32. VS Code [Електронний ресурс] – Режим доступу до ресурсу: <https://code.visualstudio.com/>

ДОДАТКИ

					ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
						61

Лістинг контролеру SongController.php

```

class SongController extends Controller
{
    public function index(Request $request)
    {
        //on playlist create
        if ($request->query("onplaylistcreate") == true) {
            $search = $request->query("query");
            $songs = Song::where('title', 'like', "%{$search}%")->get();
            return SongResource::collection($songs);
        }

        //main music page
        if ($request->query("main") == true) {
            $perPage = 15;
            $page = $request->query("page", 1);
            $filter = $request->query("filter", "both"); // latest, popular, both
            $search = $request->query("search");
            $genreId = $request->query("genre_id");

            $query = Song::with('artist');

            if ($request->has('year_range')) {
                $years = $request->input("year_range");
                if (is_array($years) && count($years) === 2) {
                    $fromYear = intval($years[0]);
                    $toYear = intval($years[1]);
                    $fromDate = $fromYear . '-01-01';
                    $toDate = $toYear . '-12-31';
                    $query->whereBetween('release_date', [$fromDate, $toDate]);
                }
            }

            if ($genreId) {
                $query->whereHas('genres', function ($q) use ($genreId) {
                    $q->where('genres.id', $genreId);
                });
            }

            if ($search) {
                $query->where(function($q) use ($search) {
                    $q->where('title', 'like', "%{$search}%")
                    ->orWhere('description', 'like', "%{$search}%")
                    ->orWhereHas('artist', function ($artistQuery) use ($search) {
                        $artistQuery->where('name', 'like', "%{$search}%");
                    });
                });
            }

            if ($filter === "latest") {
                $query->orderByDesc('created_at');
            } elseif ($filter === "popular") {

```

```

        $query->orderByDesc('played_count');
    } else {
        $query->orderByDesc('played_count')->orderByDesc('created_at');
    }

    $songs = $query->paginate($perPage, ['*'], 'page', $page);

    $songsResource = SongResource::collection($songs);//Serialization of songs with image
s3
    $songsPaginationObj = $songsResource->response()->getData(true);
    return response()->json([
        'songs' => $songsPaginationObj,
    ]);
}
//limit
if($request->query("limit")){
    return Song::with('artist')->latest()->limit(15)->get();
}
// TYT HEMAE is liked
return SongResource::collection(Song::with(['album','artist'])->get());
}
public function store(Request $request)
{
    Gate::authorize("onlyAdmin");
    $newSong = null;
    $validated = $request->validate([
        'title' => 'required|string|max:255',
        'artist_id' => 'required|exists:artists,id',
        'album_id' => 'nullable|exists:albums,id',
        'duration' => 'nullable|integer|min:1',
        'release_date' => 'nullable|date',
        'image' => 'nullable|image|mimes:jpg,jpeg,png|max:2048',
        'audio_path' => 'required|file|mimes:mp3,wav|max:20240',
        'lyrics' => 'nullable|string',
        'description' => 'nullable|string',
    ]);

    //Validated genre
    $genre = $request->validate(['genre' => 'required|exists:genres,id']);
    // image AWS S3 storage
    if ($request->hasFile('image')) {
        $validated['image'] = $request->file('image')->store('images',[
            "disk"=>'s3',
            "visibility"=>"private"
        ]);
    }
    // MP3
    try{
        if ($request->hasFile('audio_path')) {
            $newSong = $this->saveAudioFile($request->file("audio_path"),$validated);
        }
    }catch(\Throwable $e){
        File::deleteDirectory(storage_path('app/temp'));
    }
}

```

```

        return response()->json([
            'message' => 'Fiel uploading error.',
            'error' => $e->getMessage()
        ], 500);
    }
    $song = Song::create($newSong);
    $song->genres()->attach($genre);

    return response()->json([
        'message' => 'The song is successfully created.',
        'song' => $song
    ], 201);
}

public function show(Song $song, Request $request)
{
    $user = $request->user();
    if ($request->query("info") == "full") {
        $song->load(['album', 'artist', 'genres']);
    }
    $isLiked = $user ? $user->likedSongs()->where('song_id', $song->id)->exists() : false;
    $song->setAttribute('is_liked', $isLiked);
    return new SongResource($song);
}

public function update(Request $request, Song $song)
{
    Gate::authorize("onlyAdmin");

    $update = null;
    $validated = $request->validate([
        'title' => 'nullable|string|max:255',
        'artist_id' => 'nullable|exists:artists,id',
        'album_id' => 'nullable|exists:albums,id',
        'duration' => 'nullable|integer|min:1',
        'release_date' => 'nullable|date',
        'image' => 'nullable|image|mimes:jpg,jpeg,png|max:2048',
        'audio_path' => 'nullable|file|mimes:mp3,wav|max:20240',
        'lyrics' => 'nullable|string',
        'description' => 'nullable|string',
    ]);

    //Validated genre
    $genre = $request->validate(['genre' => 'nullable|exists:genres,id']);
    // image aws storage
    if ($request->hasFile('image')) {
        if ($song->image && Storage::disk('s3')->exists($song->image)) {
            Storage::disk('s3')->delete($song->image);
        }
        $validated['image'] = $request->file('image')->store('images',[
            "disk"=>'s3',
            "visibility"=>"private"
        ]);
    }
}

```

```

    }

    //audio AWS S3 storage
    try{
        if ($request->hasFile('audio_path')) {
            $path = $song->audio_path;
            if ($path && Storage::disk('s3')->exists($path)) {
                $uniqueId = explode('/', $path)[1];
                $oldFileType = $song->file_type ? $song->file_type : "mp3";
                Storage::disk('s3')->deleteDirectory("hls/{$uniqueId}");
                Storage::disk('s3')->delete("songs/{$uniqueId}.{$oldFileType}");
                Storage::delete("hls_keys/{$uniqueId}.key");
            }
            $update = $this->saveAudioFile($request->file("audio_path"),$validated);
        }
    } catch(\Throwable $e){
        File::deleteDirectory(storage_path('app/temp'));
        return response()->json([
            'message' => 'File uploading error.',
            'error' => $e->getMessage()
        ], 500);
    }
    if($update===null){
        $song->update($validated);
    } else
        $song->update($update);

    //genre update
    $song->genres()->sync($genre);

    return response()->json([
        'message' => 'The song is successfully updated.',
        'data' => $song,
    ]);
}

public function destroy(Song $song)
{
    Gate::authorize("onlyAdmin");

    if ($song->image && Storage::disk('s3')->exists($song->image)) {
        Storage::disk('s3')->delete($song->image);
    }
    $path = $song->audio_path;
    if ($path && Storage::disk('s3')->exists($path)) {
        $uniqueId = explode('/', $path)[1];
        $oldFileType = $song->file_type ?? "mp3";
        Storage::disk('s3')->deleteDirectory("hls/{$uniqueId}");
        Storage::disk('s3')->delete("songs/{$uniqueId}.{$oldFileType}");
        Storage::delete("app/hls_keys/{$uniqueId}.key");
    }
    $song->delete();
    return response()->json([

```

```

        'message' => 'The song Delete is successful.',
    ]);
}

public function stream(Song $song, Request $request)
{
    $playlistFilePath = $song->audio_path;

    try {
        $modifiedPlaylist = $this->processPlaylist($playlistFilePath);

        return response($modifiedPlaylist, 200, [
            'Content-Type' => 'application/vnd.apple.mpegurl',
        ]);
    } catch (\Exception $e) {
        return abort(404, 'Song file playlist not found.');
```

```

    }
}

protected function processPlaylist($playlistFilePath)
{
    $cacheKey = 'hls_playlist_' . md5($playlistFilePath);
    $cacheDuration = now()->addMinutes(600);

    return Cache::remember($cacheKey, $cacheDuration, function () use ($playlistFilePath) {
        $contents = Storage::disk('s3')->get($playlistFilePath);
        $lines = explode("\n", $contents);
        $output = [];

        $baseDir = dirname($playlistFilePath);
        foreach ($lines as $line) {
            $line = trim($line);
            if ($line === "" || str_starts_with($line, '#')) {
                $output[] = $line;
                continue;
            }
            $tsPath = "{$baseDir}/{$line}";
            try {
                $url = Storage::disk('s3')->temporaryUrl($tsPath, now()->addMinutes(600));
                $output[] = $url;
            } catch (\Exception $e) {
                $output[] = "HLS segment not found: {$tsPath}. Error: " . $e->getMessage();
            }
        }
        return implode("\n", $output);
    });
}

private function saveAudioFile($file,$validated) {

    $original_file = $file;
    $validated['file_size'] = $original_file->getSize();
    //Original to S3

```

```

$file_s3_path = $original_file->store('songs', [
    "disk" => 's3',
    "visibility" => "private"
]);
$filenameWithoutExtension = pathinfo($file_s3_path, PATHINFO_FILENAME);
$extension = pathinfo($file_s3_path, PATHINFO_EXTENSION);
// Temp original path
$localPath = storage_path("app/temp/creating");
mkdir($localPath, 0777, true);
$originalPath = "{$localPath}/original.mp3";
$original_file->move($localPath, 'original.mp3');
//Encryption key
$encryption = $this->
>prepareHlsEncryption("{$localPath}/hls",$filenameWithoutExtension);
// HLS
$this->createHlsFromAudio($originalPath, $encryption['segmentPath'],
$encryption['outputPath'], $encryption['keyInfoPath']);
//HLS to S3
$s3_hls_Path = "hls/{$filenameWithoutExtension}";
foreach (glob("{$localPath}/hls/*") as $file) {
    $filename = basename($file);
    Storage::disk('s3')->put("{$s3_hls_Path}/{$filename}", file_get_contents($file));
}

$validated['duration'] = $this->getSongDuration($originalPath);
$validated['audio_path'] = "{$s3_hls_Path}/playlist.m3u8";
$validated['file_type'] = $extension;
File::deleteDirectory(storage_path('app/temp'));
return $validated;
}

private function createHlsFromAudio(string $inputPath, string $segmentPath, string
$outputPath, string $keyInfoPath): void
{
    $inputPathEscaped = escapeshellarg($inputPath);
    $segmentPathEscaped = $segmentPath;
    $outputPathEscaped = escapeshellarg($outputPath);
    $keyInfoPathEscaped = escapeshellarg($keyInfoPath);

    $command = "ffmpeg -i $inputPathEscaped -vn -c:a aac -b:a 320k -f hls -hls_time 12 -
hls_list_size 0 -hls_segment_filename $segmentPathEscaped -hls_key_info_file
$keyInfoPathEscaped $outputPathEscaped";
    exec($command);
}

private function prepareHlsEncryption(string $hlsPath, string $filename): array
{
    mkdir($hlsPath, 0777, true);

    $keyPath = storage_path("app/hls_keys/{$filename}.key");
    $keyInfoPath = storage_path("app/temp/key_info_file.txt");
    $keyUri = config('app.url2') . "/api/hls/token?id={$filename}";
    $segmentPath = "{$hlsPath}/audio_%03d.ts";
    $outputPath = "{$hlsPath}/playlist.m3u8";

```

```

$key = random_bytes(16);
file_put_contents($keyPath, $key);

$keyInfoContent = "{$keyUri}\n{$keyPath}";
file_put_contents($keyInfoPath, $keyInfoContent);

return [
    'keyPath' => $keyPath,
    'keyInfoPath' => $keyInfoPath,
    'segmentPath' => $segmentPath,
    'outputPath' => $outputPath,
];
}

private function getSongDuration($filePath){
    $command = "ffprobe -i " . escapeshellarg($filePath) . " -show_entries format=duration -v
quiet -of csv=p=0";
    $duration = shell_exec($command);
    return $duration;
}

```

Додаток Б

Лістинг команди app:generate-recommendation-dataset

```

class GenerateRecommendationDataset extends Command
{
    /**
     * The name and signature of the console command.
     *
     * @var string
     */
    protected $signature = 'app:generate-recommendation-dataset';

    /**
     * The console command description.
     *
     * @var string
     */
    protected $description = 'Command description';

    /**
     * Execute the console command.
     */
    public function handle()
    {
        $fp = fopen('ai/recommendation_data.csv', 'w');

        fputcsv($fp, [
            'song_id',
            'genre_id',
            'duration',
            'release_date',
            'artist_id'
        ]);
    }
}

```

```

foreach (Song::with(['artist', 'genres'])->get() as $song) {
    $genre = $song->genres->first()?->id;
    if (!$genre || !$song->duration || !$song->release_date || !$song->artist_id) {
        continue;
    }

    fputcsv($fp, [
        $song->id,
        $genre,
        $song->duration,
        date('Y', strtotime($song->release_date)),
        $song->artist->id,
    ]);
}

fclose($fp);
$this->info("CSV created: ai/recommendation_data.csv");
}
}

```

Додаток В

Лістинг команди app:generate-recommendations

```

class GenerateRecommendations extends Command
{
    /**
     * The name and signature of the console command.
     *
     * @var string
     */
    protected $signature = 'app:generate-recommendations';

    /**
     * The console command description.
     *
     * @var string
     */
    protected $description = 'Command description';

    /**
     * Execute the console command.
     */
    public function handle()
    {
        $python = base_path('ai/recommend.py');
        exec("python $python", $output, $status);

        if ($status === 0) {
            $this->info('Model trained successfully.');
```

```

        } else {
            $this->error('Python training failed.');
```

```

        }
    }
}

```

Лістинг планувальника Laravel Scheduler

```
Schedule::command(GenerateRecommendationDataset::class)->everyMinute();
Schedule::command(GenerateRecommendations::class)->everyMinute();
Schedule::command(GenerateAllRecommendations::class)->everyMinute();
```

Лістинг методу recommendSongs

```
private function recommendSongs(User $user, $stopN = 10)
{
    $plays = SongPlay::where('user_id', $user->id)
        ->orderByDesc('played_at')->pluck('song_id')->unique();

    if ($plays->isEmpty()) return [];

    //ознаки
    $featuresCsv = base_path('ai/song_features.csv');
    $features = collect(array_map('str_getcsv', file($featuresCsv)))
        ->skip(1)
        ->map(fn($row) => [
            'song_id' => $row[0],
            'genre_id' => $row[1],
            'duration' => $row[2],
            'release_date' => $row[3],
            'artist_id' => $row[4]
        ]);

    $targetSongs = $features->whereIn('song_id', $plays->toArray())->values()->take(3);

    if ($targetSongs->isEmpty()) return [];

    $recommendedLists = [];

    foreach ($targetSongs as $song) {
        $song = json_decode(json_encode($song), true); // convert stdClass to array

        $genre_id = escapeshellarg($song['genre_id']);
        $duration = escapeshellarg($song['duration']);
        $release_date = escapeshellarg($song['release_date']);
        $artist_id = escapeshellarg($song['artist_id']);

        $scriptPath = base_path('ai/recommend_predict.py');
        $command = "python $scriptPath $genre_id $duration $release_date $artist_id";

        $output = [];
        exec($command, $output);

        $recommended = json_decode(implode(", ", $output), true);
        if (!empty($recommended)) {
            $recommendedLists[] = $recommended;
        }
    }
    return $recommendedLists;
}
```