МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії
програмного забезпечення

_____ Євген ДАВИДЕНКО

«___» _____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ВЕБЗАСТОСУНОК УПРАВЛІННЯ ПЕРСОНАЛЬНОЮ МУЗИЧНОЮ БІБЛІОТЕКОЮ

Спеціальність 121 Інженерія програмного забезпечення
Освітня програма «Інженерія програмного забезпечення»

Здобувач _____ **Богдан ЩЕДРОВ**

«__» _____ 20__ р.

Керівниця роботи
PhD,
доцентка (б. в. з) _____ **Катерина АНТІПОВА**

«__» _____ 20__ р.

Миколаїв – 2026

Завдання на виконання кваліфікаційної роботи

Чорноморський національний університет імені Петра Могили

Факультет	Комп’ютерних наук
Кафедра	Інженерії програмного забезпечення
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступінь	Бакалавр
Спеціальність	121 Інженерія програмного забезпечення
Освітня програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
«___» _____ 2026 р.

ЗАВДАННЯ

на кваліфікаційну бакалаврську роботу здобувача

Щедрова Богдана

1. Тема кваліфікаційної роботи Вебзастосунок управління персональною музичною бібліотекою затверджена наказом ректора ЧНУ ім. Петра Могили № _349_ від «26» __грудня__ 2025 р.
2. Строк представлення кваліфікаційної роботи «» ____ 2026 р.
3. Очікуваний результат роботи та початкові дані якщо такі потрібні.

– Очікуваний результат: розроблений та протестований вебзастосунок для персонального менеджменту музичної бібліотеки, що включає клієнтську частину на React, серверну частину на Django та інтегровану базу даних SQLite.

– Початкові дані: архітектурна модель 3-tier web application; технічне завдання, що включає вимоги до автоматизації обробки ID3-тегів; перелік необхідних бібліотек для React та Django.

4. Перелік питань, що підлягають розробці:

– аналіз предметної області та існуючих сервісів управління медіаконтентом;

– проєктування та моделювання системи (UML-діаграми використання, класів, послідовності);

– розробка архітектури бази даних для зберігання пісень, плейлистів та профілів користувачів;

– програмна реалізація серверної логіки (API, обробка завантаження файлів, парсинг метаданих);

– створення інтерфейсу користувача та реалізація медіаплеєра;

– комплексне тестування роботи всієї системи та верифікація результатів.

5. Перелік графічних матеріалів:

– діаграма прецедентів (Use Case diagram);

– діаграма класів (Class diagram);

– схеми алгоритмів обробки метаданих та фільтрації бібліотеки;

– архітектурна схема системи (3-tier architecture);

– скріншоти реалізованого інтерфейсу застосунку та візуалізації статистики;

– презентація.

6. Консультанти:

Консультант	**Кафедра (організація)**	**Частина роботи**

Дата видачі завдання «20» ___лютого___ 2026 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: **Вебзастосунок управління персональною музичною бібліотекою**

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КБР	01.02.2026	10.02.2026	Виконано
2.	Огляд літератури за темою роботи	11.02.2026	01.03.2026	Виконано
3.	Складання календарного плану КБР	01.03.2026	03.03.2026	Виконано
4.	Аналіз предметної області	04.03.2026	06.03.2026	Виконано
5.	Розробка проєктних рішень	07.03.2026	12.03.2026	Виконано
6.	Моделювання та конструювання ПЗ	13.03.2026	30.03.2026	Виконано
7.	Кодування, тестування та апробація розробленого ПЗ, аналіз результатів тестування, розробка керівництва користувача	31.03.2026	19.04.2026	Виконано
8.	Відгук керівника КБР	21.04.2026	21.04.2026	Виконано
9.	Оформлення КБР та презентації	22.04.2026	23.05.2026	Виконано
10.	Попередній захист	25.05.2026	25.05.2026	Виконано
11.	Рецензування	08.06.2026	08.06.2026	Виконано
12.	Завершення оформлення КБР та презентації	17.06.2026	17.06.2026	Виконано
13.	Захист кваліфікаційної роботи			

Здобувач _____ **Богдан ЩЕДРОВ**

«__» _____ 2026 р.

Керівниця роботи

PhD,

доцентка (б. в. з) _____ **Катерина АНТІПОВА**

«__» _____ 2026 р.

АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи

Вебзастосунок управління персональною музичною бібліотекою

Здобувач 408 гр.: Богдан ЩЕДРОВ

Керівниця: PhD, доцентка (б. в. з.) Катерина АНТІПОВА

Актуальність зумовлена стрімким зростанням обсягів персональних цифрових аудіоархівів та потребою користувачів у зручних, незалежних від глобальних стрімінгових сервісів інструментах для управління музичними колекціями з можливістю детальної аналітики.

Об’єкт – процес організації та управління персональними цифровими музичними колекціями у вебсередовищі.

Предмет – програмні засоби автоматизованого вилучення метаданих, структурування аудіофайлів та візуалізації статистичних даних у межах триланкової вебархітектури.

Мета – розробка вебзастосунку персонального менеджменту музичної медіатеки з використанням архітектури 3-tier (React, Django, SQLite) для автоматизації процесів зберігання, структурування та аналітики аудіофайлів.

Кваліфікаційна робота складається із вступу, 4 розділів, висновків та переліку джерел посилання.

У вступі обґрунтовано актуальність теми, сформульовано мету, завдання, об’єкт та предмет дослідження.

У першому розділі проведено аналіз предметної області, розглянуто існуючі аналоги програмного забезпечення та визначено вимоги до розроблюваної системи.

У другому розділі обґрунтовано вибір інструментарію, описано загальну структуру модулів, розроблено специфікацію вимог, інформаційну модель та архітектуру програмного інтерфейсу.

У третьому розділі проведено UML-моделювання, виконано фізичне проєктування бази даних та інтерфейсу користувача.

Четвертий розділ містить опис процесу програмної реалізації вебзастосунку, результати його комплексного тестування та керівництво користувача.

У висновках підбито підсумки виконаної роботи, підтверджено виконання всіх поставлених завдань та досягнення мети дослідження.

Кваліфікаційна робота викладена на 69 сторінках машинописного тексту, складається із вступу, 4 розділів, загальних висновків, переліку джерел посилання з 16 найменувань та 2 додатків. Праця містить 11 таблиць та 27 рисунків.

Ключові слова: вебзастосунок, музична медіатека, react, django, sqlite, аналіз метаданих, uml-моделювання

ABSTRACT

to the qualifying bachelor's thesis

Web application for personal music library management

Student of 408 group: Bohdan Shchedrov

Supervisor: PhD, Associate Professor Kateryna ANTIPOVA

Relevance of the topic is justified by the rapid growth of personal digital audio archives and the users' need for convenient tools, independent of global streaming services, for managing music collections with detailed analytics capabilities.

Object is the process of organizing and managing personal digital music collections in a web environment.

Subject encompasses software tools for automated metadata extraction, audio file structuring, and statistical data visualization within a three-tier web architecture.

Purpose is to develop a web application for personal music library management using a 3-tier architecture (React, Django, SQLite) to automate the processes of storing, structuring, and analyzing audio files.

The qualification work consists of an introduction, 4 sections, conclusions and a list of references.

In the introduction, the relevance of the topic is justified, and the purpose, tasks, object, and subject of the research are formulated.

In the first section, a domain analysis is conducted, existing software analogs are considered, and requirements for the developed system are determined.

In the second section, the choice of tools is justified, the general structure of modules is described, and the requirements specification, information model, and application programming interface architecture are developed.

In the third section, UML modeling is performed, and the physical design of the database and user interface is executed.

The fourth section contains a description of the web application software implementation process, the results of its complex testing, and a user manual.

In the conclusions, the results of the performed work are summarized, confirming the completion of all assigned tasks and the achievement of the research goal.

The qualification work is presented on 69 pages of typewritten text, consists of an introduction, 4 sections, general conclusions, a list of references with 16 titles and 2 appendices. The work contains 11 tables and 27 figures.

Keywords: web application, music library, react, django, sqlite, metadata analysis, uml modeling.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

CSS	Cascading Style Sheets
FLAC	Free Lossless Audio Codec
ID3	Identify an MP3
JWT	JSON Web Token
MP3	MPEG-1 Audio Layer III
REST	Representational State Transfer
SaaS	Software as a Service
SQL	Structured Query Language
WAV	Waveform Audio File Format
БД	База даних
ПК	Персональний комп’ютер

ВСТУП

Через те що більшість музики давно споживається переважно у цифровому форматі, у більшості випадків це випливає в проблему накопичення зайвих аудіофайлів на пристроях. Але, незважаючи на існуючі рішення цієї проблеми, а саме, стрімінгові музичні платформи, багато колекціонерів все одно мають потребу у більш специфічних інструментах, які забезпечуватимуть максимальну організацію своїх колекцій. Ще одним стримуючим від використання таких платформ фактором є ризик проблем з ліцензіями, що означає що в будь-який момент вашу улюблену музику можуть видалити назавжди. Усі ці фактори змушують реалізувати забезпечення повного контролю над власними даними, що робить розробку персонального локального застосунку актуальним завданням інженерії програмного забезпечення.

Практичне значення полягає у використанні алгоритмів автоматичного зчитування метаданих, що дозволяє автоматизувати процес організації медіатеки та підвищити зручність взаємодії з великими об'ємами даних.

Об’єкт роботи: процес організації та управління персональними цифровими музичними колекціями у вебсередовищі.

Предмет роботи: методи, алгоритми та програмні засоби автоматизованого вилучення метаданих, структурування аудіофайлів та візуалізації статистичних даних у межах триланкової вебархітектури.

Мета роботи полягає в розробці вебсистеми персонального менеджменту музичної медіатеки з використанням архітектури 3-tier (React, Django, SQLite) для автоматизації процесів зберігання, структурування та аналітики аудіофайлів.

Для досягнення поставленої мети необхідно виконати **наступні завдання:**

– аналіз існуючих рішень для управління музичними колекціями та виявлення їхніх недоліків;

– проєктування архітектури системи та бази даних для зберігання пісень, плейлистів та метаданих;

– здійснення моделювання системи за допомогою UML-діаграм для деталізації логіки взаємодії компонентів та процесів;

– розробка модуля автоматичної обробки аудіофайлів для вилучення інформації про виконавця, альбом та жанр;

– реалізування інтерфейсу користувача для відтворення музики та управління плейлистами;

– створення підсистеми візуалізації статистики медіатеки;

– проведення комплексного тестування розробленої вебсистеми для перевірки коректності виконання всіх функцій та стабільності роботи під навантаженням.

1 АНАЛІЗ СУЧАСНИХ СИСТЕМ УПРАВЛІННЯ МЕДІАКОНТЕНТОМ

1.1 Характеристика процесу управління цифровим музичним контентом та структури метаданих

З переходом від фізичних носіїв, таких як всім відомі вініл, касети та CD до цифрових форматів підходи до колекціонування музики суттєво змінилися. Але при цьому виникла проблема зручного зберігання, швидкого пошуку та систематизації великих обсягів аудіоданих.

Будь-яка цифрова музична бібліотека складається з аудіофайлів, які класифікуються за типами стиснення: формати без втрат (lossless), наприклад, FLAC та формати зі стисненням із втратами (lossy), найпопулярнішим з яких безумовно є MP3, який відомий кожній людині. На жаль, в сучасному світі вже майже неможливо торкнутись вашої музики, через що втрачається один з найважливіших факторів сприйняття цілих альбомів, тому для того щоб можна було отримати візуальне задоволення критично важливою складовою є метадані – інформація, що описує вміст файлу.

У контексті музичних файлів найпоширенішим стандартом збереження метаданих є стандарт ID3 [1]. Він пройшов кілька етапів розвитку, які важливо враховувати при розробці систем управління медіатеками:

– ID3v1 – перша версія стандарту, яка була дуже обмеженою. Вона розташовувалася в кінці MP3-файлу і займала рівно 128 байт. Цей стандарт дозволяв зберігати лише базову інформацію, таку як: назва, виконавець та альбом з жорстким обмеженням у 30 символів на поле;

– ID3v2 – сучасний стандарт, який змінив підхід до метаданих. Блок даних ID3v2 розташовується на початку файлу, прямо перед аудіопотоком, що дозволяє стрімінговим системам та плеєрам миттєво зчитувати інформацію про пісню ще до початку її повного завантаження.

Структура ID3v2 базується на концепції фреймів. Кожен фрейм відповідає за окремий тип даних і не має жорстких обмежень щодо довжини тексту. Найбільш

значущими фреймами, які повинна розпізнавати сучасна система управління медіатекою, є:

– TIT2 – назва композиції;

– TPE1 – виконавець;

– TALB – назва альбому;

– TYER / TDRC – рік випуску або дата запису;

– APIC (Attached Picture) – фрейм, що містить бінарні дані зображення обкладинки альбому разом із зазначенням його MIME-типу, наприклад, image/jpeg.

Саме від того, наскільки якісно програма вміє зчитувати, впорядковувати та дозволяти редагувати ці дані, залежить комфорт роботи з медіатекою, тому вся суть управління музичним архівом зводиться до того, щоб користувач міг легко та інтуїтивно взаємодіяти з цією структурою через зручний та гарний графічний інтерфейс.

Щоб розібратися, як краще побудувати персональну медіатеку, варто спочатку поглянути на самі аудіофайли, з якими працюватиме застосунок. Тип формату – це не просто розширення файлу, тому що від нього ще залежить, як швидко можна дістати метадані, наскільки стабільно звук буде передаватись через мережу та якою буде підсумкова якість звучання. Основні характеристики найпопулярніших форматів для наочності наведені у таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз популярних форматів аудіофайлів

Формат	**Тип стиснення**	**Бітрейт (типовий)**	**Переваги**	**Недоліки**
MP3	З втратами (Lossy)	128–320 кбіт/с	Мінімальний розмір файлу	Втрата частини звукової інформації на високих частотах

Кінець таблиці 1.1

WAV	Без стиснення (Uncompressed)	1411 кбіт/с	Найвища якість звуку, відсутність затримок при декодуванні, стандарт для професійної роботи	Дуже великий обсяг файлів, обмежена підтримка вбудованих метаданих
FLAC	Без втрат (Lossless)	600–1000 кбіт/с	Збереження 100% якості оригіналу при зменшенні розміру файлу на 30–50%	Більший розмір порівняно з MP3, може потребувати додаткових кодеків у старих браузерах

Після порівняння усіх популярних форматів аудіофайлів стало зрозуміло що вони відрізняються за багатьма різними параметрами і у кожному випадку доводиться жертвувати чимось.

1.2 Огляд і аналіз аналогічних систем управління медіатекою

Загалом системи керування аудіо можна звести до трьох категорій. Йдеться про стрімінги, локальні комп'ютерні плеєри та домашні медіасервери. Дослідження популярних застосунків допоможе наочно побачити їхні архітектурні підходи та зробити відповідні висновки для подальшої розробки.

Стрімінгова платформа Spotify

Spotify – один із лідерів серед стрімінгових аудіосервісів. Сервіс функціонує за моделлю «програмне забезпечення як послуга» (SaaS) [2]. Платформа надає доступ до великої кількості пісень для користувачів з усього світу. Сервіс орієнтований на хмарне прослуховування, що дозволяє користувачам не думати про фізичне зберігання файлів на своїх пристроях. На рисунку 1.1 можна побачити інтерфейс платформи Spotify на головній сторінці.

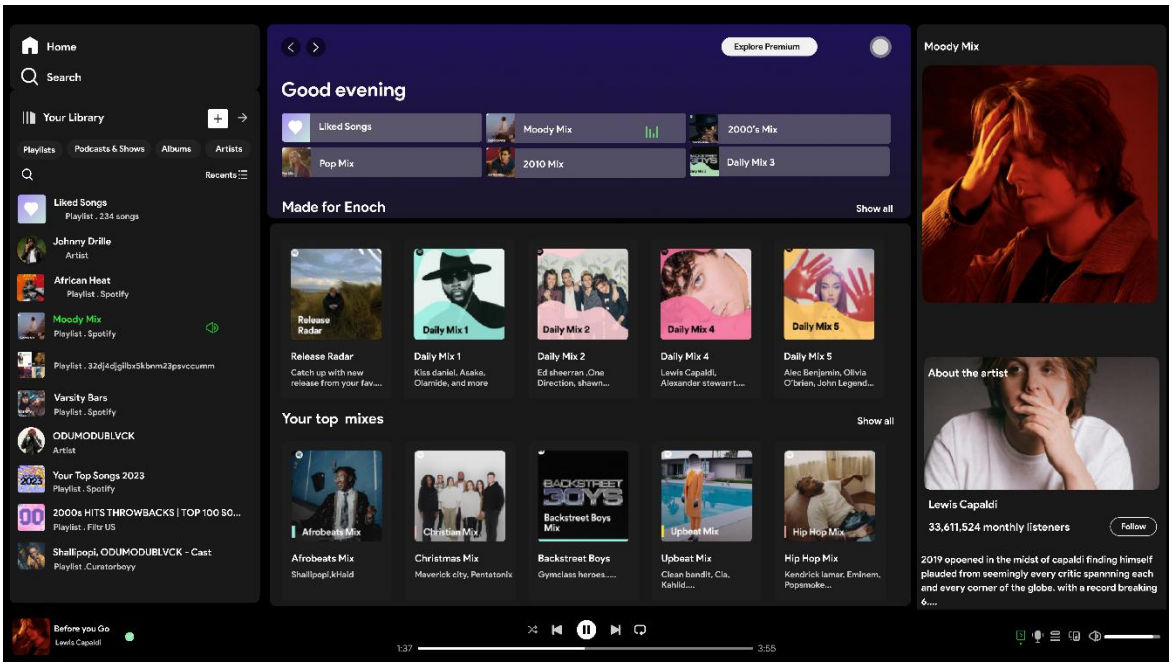

Рисунок 1.1 – Інтерфейс стрімінгової платформи Spotify

Архітектура та технологічний стек включають в себе клієнтську частину десктопного додатка, що побудована на базі Chromium Embedded Framework, а вебінтерфейс використовує React.js для забезпечення динамічного рендерингу. Серверна логіка базується на мікросервісній архітектурі, яка накопичує в собі понад 1000 сервісів, написаній переважно на Java, C++ та Python. Для зберігання даних використовуються бази даних Cassandra та PostgreSQL. Контент захищається технологіями DRM, а аудіопотік передається у зашифрованих форматах, таких як Ogg Vorbis або AAC.

Головними функціями системи є миттєвий пошук по хмарній базі, автоматична генерація персоналізованих плейлистів “Daily Mix” та “Discover Weekly” на основі вподобань користувачів, а також розвинені соціальні функції,

наприклад, є можливість стежити за активністю друзів та створювати спільні плейлисти та віднедавна можна спілкуватись там як у месенджері. Ще однією важливою особливістю є функція Spotify Connect, що дозволяє безшовно перемикати відтворення між смартфоном, ПК та смарт-колонками.

Сильною стороною Spotify є велика база контенту та інтелектуальні алгоритми рекомендацій, проте, з точки зору управління персональним архівом, система має критичні недоліки: користувач фактично орендує доступ до музики без права власності на файли, що означає що в умовах проблем з ліцензіями у власників пісень їх можуть видалити з платформи. Також у Spotify відсутня будь-яка можливість редагування метаданих та система повністю залежить від зовнішнього підключення до Інтернету. У таблиці 1.2 проведено порівняння основних критеріїв з проєктованою системою.

Таблиця 1.2 – Порівняльна таблиця застосунку Spotify

Критерій порівняння	**Spotify**	**Проєктована система**
Архітектура	Хмарні мікросервіси (Java/C++)	Триланкова вебархітектура (Django)
Власність на дані	Тимчасова оренда (SaaS)	Повне володіння файлами (Self-hosted)
Редагування тегів	Заблоковано на рівні сервера	Повний доступ через вебінтерфейс
Аналітика	Статична (раз на рік)	Динамічна (діаграми + експорт PNG)
Модель розгортання	Публічна хмара	Локальна/Корпоративна мережа

Десктопний аудіоплеєр Foobar2000

Foobar2000 є відомим локальним аудіоплеєром для ОС Windows, розроблений Пітером Павловскі [3]. Плеєр є одним з найпопулярніших серед колекціонерів аудіо завдяки своєму мінімалістичному дизайну та низькому споживанню системних ресурсів. Також цей застосунок має свою унікальність – модульну архітектуру, а ця особливість дозволяє розширювати функціонал за допомогою сторонніх компонентів. На рисунку 1.2 можна побачити інтерфейс Foobar2000.

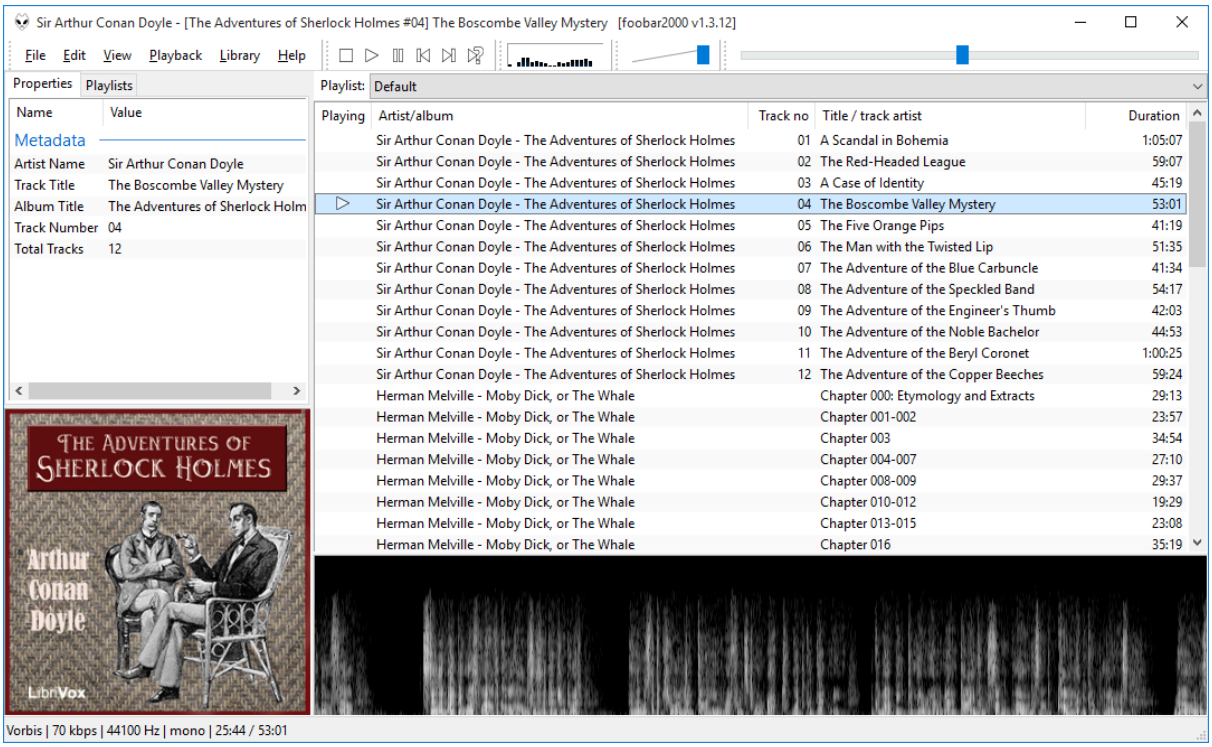

Рисунок 1.2 – Інтерфейс десктопного програвача Foobar2000

Особливості архітектури полягають у тому що система побудована як монолітна десктопна архітектура та написана на мові C++ із використанням Windows API. Завдяки використанню протоколів ASIO у програми є прямий доступ до апаратних засобів. Ця система не використовує бази даних для індексації медіатеки, але замість цього застосовує власні двійкові формати зберігання кешу та конфігурації.

Система підтримує практично всі існуючі формати аудіофайлів включаючи навіть FLAC, DSD і ALAC, які є доволі рідкісними, і також застосунок має один із

найпотужніших вбудованих редакторів метаданих. До функцій також входять вбудований конвертер форматів, інструменти для ріппінгу аудіо-дисків та можливість повної кастомізації графічного інтерфейсу.

Сильними та слабкими сторонами для користувача є найвища швидкість обробки бібліотек на сотні тисяч пісень та повна автономність. Слабкі сторони полягають у відсутності вебінтерфейсу, що обмежує користувача у використанні лише на ПК, а також застосунок має застарілий дизайн та високий поріг входження для налаштування системи під власні потреби. У таблиці 1.3 проведено порівняння основних критеріїв з проєктованою системою.

Таблиця 1.3 – Порівняльна таблиця застосунку Foobar2000

Критерій порівняння	**Foobar2000**	**Проєктована система**
Тип застосунку	Десктопний (Windows)	Вебзастосунок (кросплатформний)
Стек розробки	C++ / Win32	Повне володіння файлами (Self-hosted)
Віддалений доступ	Відсутній	Через браузер у локальній мережі
Збереження даних	Локальні конфігураційні файли	Реляційна БД SQLite 3
Інтерфейс	Статичний GUI	Адаптивний

Персональний медіасервер Plex

Plex Media Server [4] – це потужна клієнт-серверна система, призначена для організації домашніх колекцій фільмів, музики та фотографій. Серверна частина розгортається на ПК або мережевому сховищі, а доступ до контенту здійснюється через кросплатформні клієнти, включаючи веббраузери та додатки для смарт-ТВ. На рисунку 1.3 можна побачити вебінтерфейс Plex у розділі «Музика».

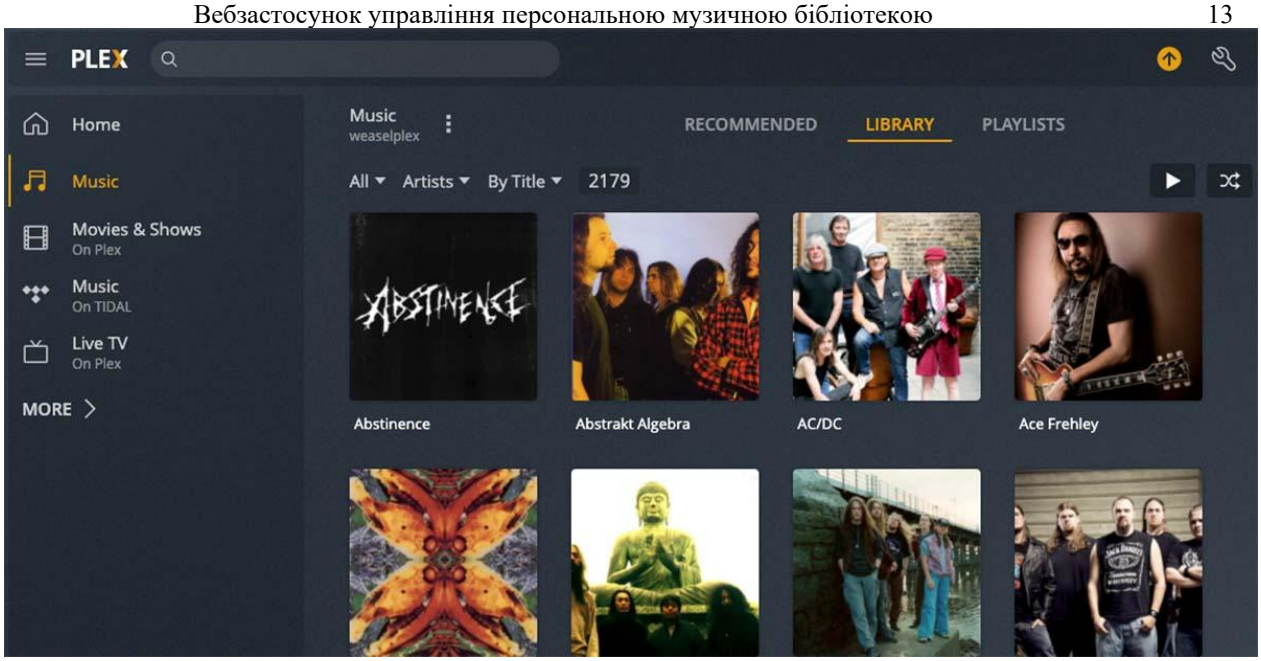

Рисунок 1.3 – Вебінтерфейс медіасервера Plex

Стек технологій включає в собі серверну частину, що написана на C++ та Python. Клієнтська вебчастина використовує React.js. Для зберігання індексів бібліотеки, історії переглядів та метаданих застосовується база даних SQLite 3. Система транскодування на базі FFmpeg є родзинкою системи, яка дозволяє перетворювати аудіо- та відеоформати під вимоги клієнтського пристрою.

Ключовими функціями Plex є автоматичне завантаження метаданих та обкладинок із зовнішніх баз даних, а також надання віддаленого доступу до бібліотеки через мережу Інтернет і можливість створювати різні профілі для членів сім'ї, а також система підтримує трансляцію на пристрої Chromecast та AirPlay.

Сильною стороною застосунку є універсальність та приємний інтерфейс, а слабкою ж стороною для суто музичних завдань є перевантаження функціями для відео, складність ручного редагування ID3-тегів безпосередньо у файлах та наявність платних обмежень для розширеної аналітики. У таблиці 1.4 проведено порівняння основних критеріїв з проєктованою системою.

Таблиця 1.4– Порівняльна таблиця застосунку Plex

Критерій порівняння	**Plex**	**Проєктована система**
Фокус контенту	Універсальний (кіно / фото / музика)	Спеціалізований (аудіо)
Джерело метаданих	Зовнішні бази даних	Вбудовані ID3-теги
Складність розгортання	Висока (налаштування портів/мережі)	Низька (запуск Django-сервера)
Платні функції	Присутні (Plex Pass для аналітики)	Відсутні
Експорт даних	Обмежений	JSON плейлисти + статистика у форматі PNG

Після порівняння аналогів, можна зробити висновок що жоден з існуючих застосунків не може виконати роль вебзастосунку, який фокусуватиметься лише на музиці, але при цьому надавав би користувачам доступ до редагування метаданих через браузер і пропонував би аналітику у вигляді графіків, а онлайн сервіси взагалі лишають користувача права власності на дані та десктопні плеєри обмежують мобільність. Усі ці недоліки якраз-таки зумовлюють необхідність розробки власного вебзастосунку, який матиме в собі поєднання зручності хмарних систем із потужністю локальних інструментів управління метаданими.

1.3 Порівняльний аналіз та обґрунтування створення власного рішення

Для того щоб детальніше обґрунтувати необхідність розробки нового вебзастосунку було проведено порівняльний аналіз розглянутих застосунків із проєктованою системою. Найголовнішими критеріями для порівняння стали архітектура системи, ступінь контролю над файлами та наявність можливості редагування метаданих.

Після аналізу можна зробити висновок що розглянуті аналоги не можуть повністю задовольнити певні умови, таким чином, Spotify, маючи хмарну архітектуру та універсальний доступ, позбавляє користувача права власності на контент та не має можливості редагувати метадані, а Foobar2000, попри можливість редагування, не має вебінтерфейсу та, відповідно, віддаленого доступу. Plex же має доступ через браузер, але його функціонал занадто сфокусований на відео.

Враховуючі перераховані недоліки, виникає потреба у розробці спеціалізованого вебзастосунку, який має бути легким, використовуючи компактну БД SQLite [5] замість важких серверних СУБД та фокусуватись виключно на музичному контенті та крім цього система має поєднувати інтуїтивний вебінтерфейс, при цьому маючи повний контроль над файлами та їх метаданими.

Розроблювана система крім всього, матиме аналітичні інструменти, які, на відміну від аналогів, дадуть змогу користувачеві експортувати плейлисти та статистику у форматах JSON та PNG, що дозволяє обмінюватись контентом або здійснювати резервне копіювання. Тому система крім простого плеєра стає ще й інструментом для менеджменту персональної музичної колекції.

Висновки до розділу 1

Виконавши аналіз предметної області, можна зробити висновок що навіть попри те що наведені аналоги домінують на ринку серед користувачів, персональні системи для роботи з аудіоархівами залишаються затребуваними. Бо поки існують користувачі, які хочуть повністю володіти своєю медіатекою та не бути залежними від ліцензійних змін у хмарних сервісах, їх потреби мають задовольнятись.

Більшість плеєрів також не мають можливості для змінення їх метаданих, а навіть якщо мають, то ці можливості дуже обмежені, а, зробивши розгляд структури цифрових аудіофайлів можна побачити їх більший потенціал.

Огляд та порівняння інших застосунків продемонстрували незадовільнення окремих умов у кожному. Ці сервіси буквально забирають у користувача право власності на файли, часто виглядають застарілими, або занадто складні в налаштуванні, тому саме ці висновки стали головними у розробці власного вебзастосунку.

2 ПРОЄКТУВАННЯ ТА СПЕЦИФІКАЦІЯ ВИМОГ ДО ВЕБЗАСТОСУНКУ

2.1 Обґрунтування вибору сучасного інструментарію розробки

Вибір стеку технологій для реалізації вебзастосунку базується на аналізі актуальних методів веброзробки та вимогах до продуктивності, безпеки і масштабованості системи. Архітектурний патерн розділення на клієнтську та серверну частини через RESTful API є стандартом, який забезпечує гнучкість розробки та полегшує подальше масштабування вебзастосунку [6].

Для реалізації серверної частини було обрано мову програмування Python та її фреймворк Django [7]. Цей фреймворк демонструє високу ефективність у розробці застосунків, які орієнтовані на роботу з базами даних, завдяки вбудованій системі об'єктно-реляційного відображення [8]. Використання бібліотеки Django REST Framework дозволяє швидко та безпечно проєктувати API-ендпоінти,а механізми безпеки Django надійно захищають систему від поширених вразливостей, таких як SQL-ін'єкції та підробка запитів між сайтами, що є критично важливим для систем з авторизацією користувачів [9].

Для роботи з метаданими аудіофайлів обрано бібліотеку Mutagen [10], яка є одним із найбільш стабільних інструментів для мови Python, який підтримує читання та запис розширених тегів ID3v2.4, включаючи маніпуляції з бінарними даними, наприклад, вшивання зображень обкладинок альбомів [11].

У якості реляційної бази даних обрано SQLite, яка не вимагає конфігурації окремого сервера і зберігає всі дані в єдиному файлі, що ідеально відповідає концепції легкої, переносимої персональної бібліотеки. Для розробки клієнтської частини обрано бібліотеку React [12]. Single-Page Applications, побудовані на базі React, забезпечують високу швидкість відтворення динамічного контенту без перезавантаження сторінки завдяки використанню технології Virtual DOM, яка мінімізує кількість прямих звернень до реального DOM браузера та значно підвищує продуктивність рендерингу [13].

При проєктуванні серверної частини розглядалися три найпопулярніші сучасні технології: Node.js Express, Python Flask та Python Django. Хоча Node.js і забезпечує високу швидкість обробки асинхронних запитів, робота з бінарними аудіофайлами та парсинг складних метаданих краще реалізовувати використовуючи Python завдяки спеціалізованим бібліотекам. Порівняння фреймворків Flask та Django наведено у таблиці 2.1.

Таблиця 2.1 – Порівняльний аналіз серверних фреймворків Python

Критерій	**Flask**	**Django**
Архітектура	Мікрофреймворк, потребує збирання з модулів	MVT
Робота з БД (ORM)	Потребує зовнішньої SQLAlchemy	Вбудована потужна Django ORM
Панель адміністратора	Відсутня (потребує розробки з нуля)	Вбудована, генерується автоматично
Безпека (CSRF, XSS)	Потребує ручного налаштування	Вбудований захист на рівні middleware
Швидкість розробки	Низька (для складних систем)	Висока (швидке прототипування)

На основі таблиці 2.1 очевидно, що для системи управління медіатекою, яка вимагає надійної роботи з реляційною базою даних та наявності ролі адміністратора, використання Django є найбільш доцільним.

Для реалізації інтерактивного користувацького інтерфейсу розглядалися бібліотека React та фреймворки Vue.js і Angular, їх порівняння виконано у таблиці 2.2.

Таблиця 2.2 – Порівняльний аналіз frontend-технологій

Критерій	**Angular**	**Vue.js**	**React**
Тип	Повний MVC фреймворк	Прогресивний фреймворк	UI бібліотека
Складність	Висока (вимагає TypeScript)	Легка	Середня (JSX)
Рендеринг	Real DOM	Virtual DOM	Virtual DOM
Компонентна база	Жорстко структурована	Гнучка	Максимально гнучка

React було обрано завдяки використання бібліотекою односпрямованого потоку даних і Virtual DOM. Це особливо важливо для застосунку, де під час відтворення музики стан плеєра оновлюється щосекунди, наприклад, для збереження поточного часу пісні. React дозволяє оновлювати лише смугу прогресу без впливу на список плейлистів чи інші ресурсоємні елементи DOM, що забезпечує високу продуктивність.

2.2 Призначення системи та загальна структура модулів

Основна ідея застосунку полягає у тому щоб створити персональну цифрову музичну бібліотеку, де користувач повністю володіє своєю медіатекою, так як на відміну від аналогічних музичних платформ, ця система зосереджена на локальному зберіганні та розширеному управлінні музичною бібліотекою. Застосунок дозволяє не лише слухати музику, а ще й напряму працювати зі структурою бібліотеки.

Щоб забезпечити стабільну роботу та зручність подальшої підтримки, систему було вирішено розбити на кілька ключових функціональних модулів:

– модуль авторизації та роботи з профілями – цей блок відповідає за все, що стосується безпеки та персоналізації,а саме він забезпечує реєстрацію, створення сесій та захищений вхід, щоб доступ до медіатеки мав лише її власник;

– модуль керування медіатекою – мабуть, найважливіша частина системи. Тут реалізовано логіку завантаження файлів через зручний механізм Drag-and-drop, автоматичне зчитування ID3-тегів за допомогою зовнішніх бібліотек та можливість вручну виправляти інформацію про треки;

– модуль відтворення та керування плеєром – модуль, який дозволяє безпосередньо взаємодіяти з музикою та включає у себе сам аудіоплеєр, регулювання гучності, чергу відтворення та стандартні режими випадкового порядку або повтору треків;

– модуль плейлистів та організації – відповідає за групування пісень. Користувач може створювати необмежену кількість списків відтворення, редагувати їх на свій розсуд та швидко знаходити потрібні композиції, використовуючи фільтри;

– модуль аналітики та експорту – блок, що збирає дані про те, що і як часто слухає користувач. Результати візуалізуються у вигляді діаграм, які можна зберегти як картинку. І також тут реалізовано можливість експорту структури бібліотеки у JSON-формат для резервного копіювання.

2.3 Специфікація вимог до програмного забезпечення

1) ***Призначення та межі проєкту:***

1.1) призначення системи: вебзастосунок призначений для надання користувачеві інструментарію для персонального менеджменту медіатеки, що включає завантаження, структурування, редагування метаданих та відтворення аудіофайлів у межах локальної мережі;

1.2) специфікація розроблена на основі стандартів веброзробки та триланкової архітектури. Обрано використання REST API для передачі даних між фронтендом та бекендом у форматі JSON;

1.3) межі проєкту ПЗ: проєкт охоплює розробку вебінтерфейсу на основі React та серверної інфраструктури на основі Django. Система розрахована на роботу з локальними аудіофайлами і не включає інтеграцію з зовнішніми платними стрімінговими сервісами.

2) ***Загальний опис:***

2.1) сфера застосування: система призначена для приватного використання у домашніх аудіоархівах або для розгортання в обмежених локальних мережах, наприклад, у невеликих офісах, закладах освіти чи музичних студіях за моделлю self-hosted медіасервера;

2.2) характеристики користувачів: адміністратор: володіє повним доступом до завантаження файлів, видалення контенту та редагування ID3-тегів; зареєстрований користувач: може створювати власні профілі та плейлисти, лайкати пісні та переглядати історію прослуховувань; гість: має право лише на прослуховування публічних розділів медіатеки без можливості внесення змін;

2.3) загальна структура і склад системи: клієнтська частина побудована на React з використанням CSS-змінних для Dark Mode. Серверна частина реалізована на Python (Django), що взаємодіє з бібліотекою Mutagen для обробки метаданих. База даних – SQLite;

2.4) загальні обмеження: необхідність підтримки HTML5 Audio API браузером, стабільна робота гарантується при одночасному доступі до 50 користувачів на одну локальну ноду сервера.

3) ***Функції системи:***

3.1) функція авторизації та управління профілем:

3.1.1) опис функції: забезпечує реєстрацію нових користувачів, вхід до системи та можливість персоналізації профілю;

3.1.2) вхідна і вихідна інформація: логін, пароль, зображення профілю; вихід – сесійний доступ та авторизований стан інтерфейсу;

3.1.3) функціональні вимоги: паролі зберігаються у вигляді хешу bcrypt, автентифікація реалізована на базі вбудованих механізмів безпеки Django.

3.2) функція управління медіатекою:

3.2.1) опис функції: дозволяє адміністратору додавати нові пісні, редагувати їх метадані (виконавець, альбом, жанр) та змінювати обкладинки;

3.2.2) вхідна і вихідна інформація: файл .mp3/.wav, нові текстові теги, зображення обкладинки; вихід – оновлені записи в БД SQLite;

3.2.3) функціональні вимоги: автоматичний парсинг метаданих бібліотекою Mutagen повинен відбуватися за менше 1 секунди після завантаження.

3.3) функція плейлистів та соціальних взаємодій:

3.3.1) формування персональних списків відтворення, можливість «вподобати» пісні та імпортувати чужі плейлисти;

3.3.2) вхідна і вихідна інформація: вибір ID пісень, назва плейлиста; вихід – сформована структура плейлиста у профілі користувача;

3.3.3) система повинна вести історію останніх 50 прослуханих пісень для кожного користувача.

3.4) функція аналітики та експорту:

3.4.1) опис функції: візуалізація статистики медіатеки та експорт даних у форматах JSON/PNG;

3.4.2) вхідна і вихідна інформація: вхідна – дані про прослуховування; вихіна – діаграма розподілу жанрів у форматі зображення або JSON-файл структури плейлиста;

3.4.3) функціональні вимоги: візуалізація статистики має базуватися на актуальних метаданих БД і бути доступною для експорту як графічний файл.

4) ***Вимоги до інформаційного забезпечення:***

4.1) джерела і зміст вхідної інформації: аудіофайли користувача, зображення обкладинок, введені вручну теги;

4.2) нормативно-довідкова інформація: класифікатор музичних жанрів згідно зі стандартом ID3v2;

4.3) вимоги до збереження інформації: дані зберігаються у реляційній БД SQLite 3, забезпечується цілісність зв’язків між таблицями пісень, виконавців та плейлистів.

5) ***Вимоги до технічного забезпечення***: сервер на базі ПК користувача з встановленним Python 3.10+, клієнтські пристрої з сучасними браузерами та можливістю відтворення звуку.

6) ***Вимоги до програмного забезпечення:***

6.1) архітектура програмної системи: триланкова (клієнт-сервер-БД);

6.2) системне програмне забезпечення: ОС Windows/Linux, інтерпретатор Python 3.10+, Node.js для розробки фронтенду;

6.3) мережне програмне забезпечення: протокол HTTP/HTTPS для взаємодії в локальній мережі;

6.4) програмне забезпечення ведення інформаційної бази: SQLite 3 через Django ORM;

6.5) мова і технологія розробки: Python (Django), JavaScript (React), CSS3.

7) ***Вимоги до зовнішніх інтерфейсів:***

7.1) інтерфейс користувача: адаптивний Dark Mode дизайн, підтримка Drag-and-drop;

7.2) апаратний інтерфейс: стандартні аудіовиходи ПК та мобільних пристроїв;

7.3) RESTful API для обміну даними у форматі JSON;

7.4) комунікаційний протокол: HTTP/HTTPS.

8) ***Властивості програмного забезпечення:***

8.1) доступність: система має бути доступною 24/7 в межах локальної мережі розгортання;

8.2) модульна структура додатків Django та компонентів React;

8.3) переносимість: можливість швидкого перенесення бази даних як файл .sqlite3 між пристроями;

8.4) продуктивність: час пошуку пісні в бібліотеці – менше 0.5 с;

8.5) надійність: автоматична обробка помилок при завантаженні пошкоджених MP3-файлів;

8.6) безпека: захист від SQL-ін’єкцій за допомогою Django ORM, використання CSRF-захисту для форм.

9) ***Інші вимоги:***

Відповідність законодавству України «Про захист персональних даних» користувачів при реєстрації в системі.

2.4 Інформаційна модель системи та структура бази даних

Для того щоб забезпечити надійне зберігання та швидкий доступ до контенту було розроблено реляційну інформаційну модель, яка реалізується засобами СУБД SQLite через систему об'єктно-реляційного відображення фреймворку Django. Ця інформаційна модель базується на виділенні ключових сутностей предметної області та встановленні зв'язків між ними.

Головною сутністю системи є «Користувач», яка успадковує базову модель аутентифікації Django та відповідно ця сутність є центральною, оскільки система реалізує концепцію персональної медіатеки: всі інші дані прив'язані до конкретного профілю.

Сутність «Пісня» відповідає за зберігання інформації про кожен завантажений аудіофайл та включає наступні поля:

– id – первинний ключ;

– title та artist – текстові поля, що заповнюються автоматично під час парсингу ID3-тегів за допомогою бібліотеки Mutagen;

– album та genre – опціональні поля для подальшої фільтрації;

– duration – цілочисельне поле, що зберігає довжину пісні в секундах для коректного відображення у плеєрі;

– audio_file та cover_image – поля типу FileField/ImageField, які містять відносні шляхи до фізичних файлів у сховищі сервера;

– user_id – зовнішній ключ, що встановлює зв'язок «один-до-багатьох» із сутністю «Користувач».

Для реалізації функціоналу списків відтворення створено сутність «Плейлист» з ідентифікатором, назвою, датою створення та посиланням на власника. Оскільки один плейлист може містити багато пісень, а одна пісня може входити до різних плейлистів, між цими сутностями встановлено зв'язок Many-to-

Many. У базі даних цей зв'язок реалізовано через асоціативну таблицю "Playlist_Track", яка містить пари playlist_id, track_id та час додавання пісні. Така структура запобігає дублюванню даних і забезпечує ефективне виконання SQL-запитів під час фільтрації контенту.

2.5 Архітектура програмного інтерфейсу

Оскільки вебзастосунок розроблено за триланковою архітектурою, клієнтська частина у вигляді Single-Page Application [14] на базі React та серверна частина на Django існують як незалежні модулі, що взаємодіють між собою через мережевий протокол HTTP. Для забезпечення стандартизованого обміну даними було спроєктовано програмний інтерфейс за принципами REST [15].

Обмін даними між клієнтом та сервером здійснюється у форматі JSON, що гарантує легкість серіалізації та десеріалізації об'єктів. Архітектура API передбачає використання стандартних HTTP-методів для виконання CRUD-операцій:

– метод GET використовується для отримання даних без зміни їх стану на сервері. Наприклад, запит на ендпоінт /api/tracks/ повертає масив JSON-об'єктів з інформацією про всі пісні авторизованого користувача. Запит на /api/statistics/ повертає агреговані дані про жанри для подальшої побудови кругової діаграми на клієнті;

– метод POST застосовується для створення нових ресурсів. Зокрема, ендпоінт /api/tracks/upload/ приймає бінарні дані аудіофайлу у форматі multipart/form-data та ініціює алгоритм збереження та парсингу метаданих. Також цей метод використовується для реєстрації користувачів та створення нових плейлистів;

– методи PUT та PATCH використовуються для оновлення існуючих записів, наприклад, при ручному редагуванні назви пісні або зміні обкладинки альбому;

– метод DELETE забезпечує безпечне видалення файлів із серверного сховища та відповідних записів із бази даних SQLite.

Для забезпечення безпеки всі API-ендпоінти, окрім реєстрації та авторизації, захищені за допомогою механізму сесій, що означає що будь-який запит до

захищених ресурсів від неавторизованого клієнта перехоплюється на рівні middleware фреймворку Django та повертає статус-код 401 Unauthorized. Завдяки такій архітектурі, фронтенд отримує можливість миттєво реагувати на дії користувача, в тому числі оновлювати списки, малювати графіки, керувати плеєром без необхідності повного перезавантаження вебсторінки, що створює плавний користувацький досвід.

Для забезпечення цілісності специфікації вимог та деталізації взаємодії системних компонентів розроблено перелік ендпоінтів, що наведено у таблиці 2.3.

Таблиця 2.3 – Специфікація програмних інтерфейсів вебзастосунку

Ендпоінт	**Метод**	**Призначення та опис дії**	**Вхідні дані (JSON / Form-data)**	**Вихідні дані (JSON)**
/api/auth/register/	POST	Реєстрація нового користувача у системі	username, email, password	Об'єкт створеного користувача, статус 201.
/api/auth/login/	POST	Автентифікація користувача та отримання токена сесії	username, password	Токен доступу, дані профілю, статус 200.
/api/tracks/	GET	Отримання повного списку пісень авторизованого користувача	Відсутні	Масив об'єктів Track з метаданими
/api/tracks/upload/	POST	Завантаження аудіофайлу та ініціалізація парсингу тегів	audio_file	Метадані вилученої пісні, статус 201

Продовження таблиці 2.3

/api/tracks/{id}/	GET	Отримання детальної інформації про конкретну композицію	id	Об'єкт Track
/api/tracks/{id}/	PATCH	Редагування метаданих пісні	title, artist, genre	Оновлений об'єкт пісні, статус 200
/api/tracks/{id}/	DELETE	Видалення пісні з медіатеки та фізичне видалення файлу	id	Повідомлення про успішне видалення
/api/playlists/	GET	Перегляд усіх створених списків відтворення користувача	Відсутні	Список об'єктів Playlist
/api/playlists/	POST	Створення нового порожнього плейлиста	name, description	Об'єкт Playlist з ID, статус 201
/api/playlists/{id}/add/	POST	Додавання обраних пісень до конкретного плейлиста	track_ids (масив ID)	Оновлений список пісень у плейлисті

Кінець таблиці 2.3

/api/export/json/	GET	Формування та завантаження структурованого файлу медіатеки	playlist_id	JSON-файл для завантаження на диск

Кожен запит до API проходить через шар посередників, або як його прийнято називати, Middleware для перевірки прав доступу, це означає, що якщо запит ініційовано неавторизованим клієнтом, система повертатиме статус-код 401 Unauthorized. Для операцій завантаження великих файлів було встановлено обмеження розміру тіла запиту, що гарантує захист від перевантаження серверних ресурсів. Використання такої структурованої системи API дозволяє реалізувати чуйний інтерфейс плеєра, який миттєво реагує на дії користувача без перезавантаження сторінки застосунку.

Висновки до розділу 2

У ході роботи над другим розділом було повністю сформовано архітектурний вигляд майбутньої системи. Вибрано стек із Django та React через необхідність створити швидкий та безпечний застосунок, який би легко розгортався на звичайному ПК.

Також створено детальну специфікацію вимог. Було визначено функціональні можливості та технічні обмеження щодо безпеки та продуктивності. Спроєктовано базу даних на основі SQLite, що в свою чергу дозволило створити легку та цілісну модель зберігання інформації, яку також за необхідністю можна швидко перенести.

Використаний REST API поєднує між собою сервер та інтерфейс користувача. Завдяки використанню формату JSON для обміну даними, досягнено високої швидкості роботи плеєра. Завершення етапу моделювання та проєктування дозволяє переходити до програмної реалізації вебзастосунку.

3 МОДЕЛЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Обґрунтування архітектури системи та вибір патернів проєктування

Для розробки системи було обрано триланкову архітектуру, яка складається з клієнта, сервера та бази даних. На серверній частині використано архітектурний патерн MVT. Через специфіку фреймворку Django, замість звичного Template, який притаманний патерну, працюють серіалізатори, так як фреймворк адаптований під REST API.

В другому розділі було обґрунтовано вибір стеку із рівня представлення на React, рівня логіки на Django REST Framework, а рівень даних – на SQLite. REST API має величезну перевагу через свою можливість реалізації незалежної роботи фронтенду та бекенду. Для безпеки використані JWT-токени, які широко використовуються для аутентифікації, при цьому дозволяючи серверу здійснювати перевірку користувачів без збереження сесії.

Якщо розібрати систему детальніше, виходить наступне:

– рівень представлення реалізований на бібліотеці React та відповідає за інтерфейс користувача, обробку подій та візуалізацію аудіоплеєра. Використання Single-Page Application дозволяє оновлювати стан плеєра, тобто, час відтворення, прогрес-бар без перезавантаження всієї сторінки що є дуже важливим для комфортним користуванням застосунком;

– для побудови рівню логіки використано фреймворк Django REST Framework, який тримає в собі основну бізнес-логіку, а саме: автоматичне зчитування метаданих ID3v2 за допомогою бібліотеки Mutagen, обробка завантажених файлів та фільтрація медіатеки за жанрами чи артистами;

– рівень даних використовує реляційну СУБД SQLite. Це рішення обрано через портативність, бо вся база даних зберігається в одному файлі, що дозволяє легко переносити персональну медіатеку між різними пристроями.

3.2 UML-моделювання структури та поведінки системи

Для візуалізації логіки розроблено UML-діаграми щоб детально показати структуру та поведінку системи. Для візуалізації взаємодії користувачів із системою розроблено діаграму варіантів використання. Рисунок 3.1 зображає відповідну діаграму, що дозволяє визначити межі системи та основні сервіси, які вона надає різним категоріям користувачів.

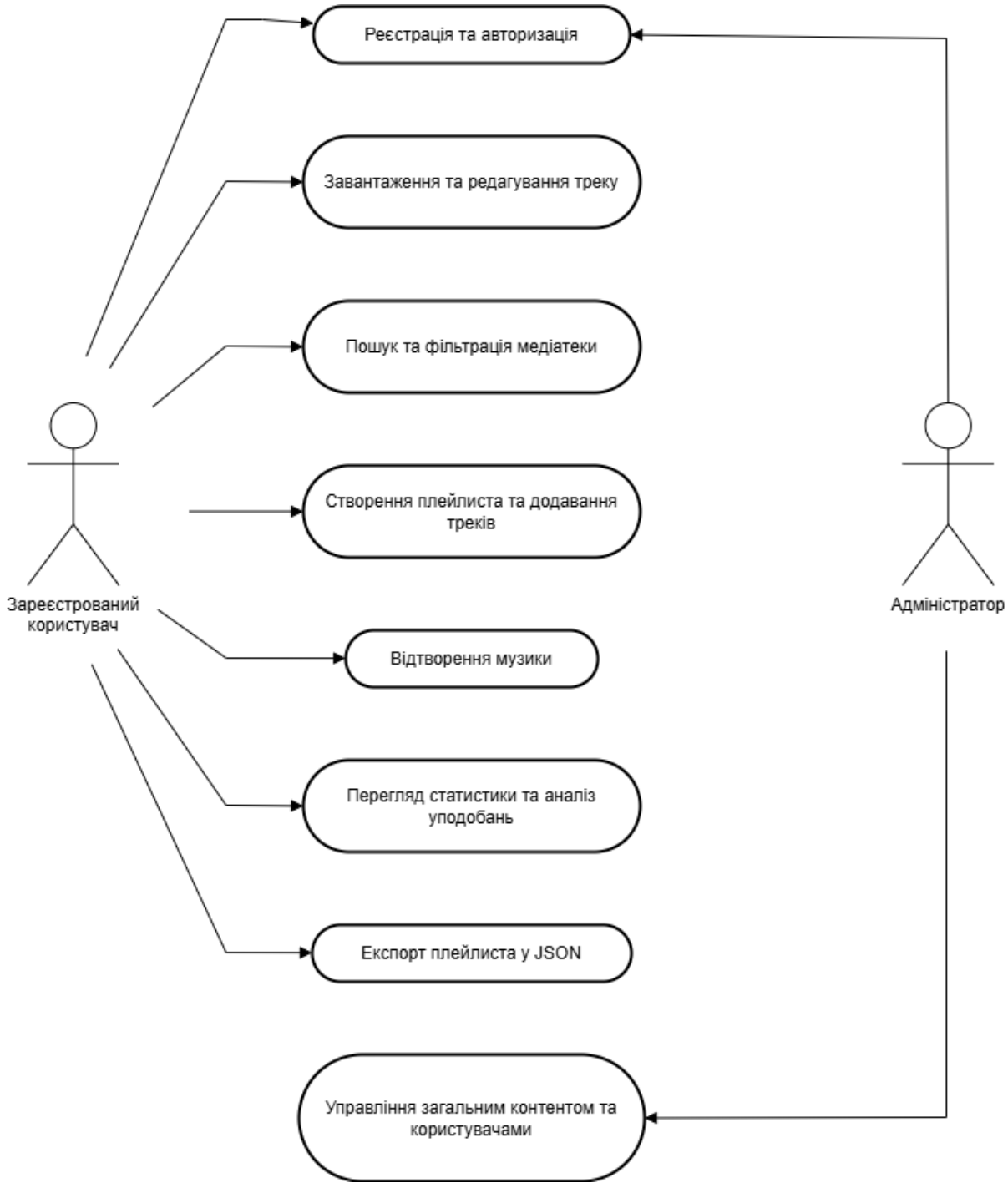

Рисунок 3.1 – Діаграма варіантів використання

Згідно з побудованою діаграмою, у системі виділено два типи дійових осіб, а саме: зареєстрований користувач, який є основним суб’єктом взаємодії, для якого реалізовано персональний робочий простір та адміністратор, є користувачем із розширеними правами доступу, який крім того відповідає за модерацію.

Для ролі зареєстрованого користувача визначено наступний набір прецедентів:

– реєстрація та авторизація – початкова точка входу, яка забезпечує безпечний доступ до персональних даних за допомогою JWT-токенів;

– завантаження та редагування треку – важлива функція додавання аудіофайлів до бази даних, яка включає в себе можливість редагування метаданих;

– пошук та фільтрація медіатеки – це інструменти для знаходження композицій за різними атрибутами;

– створення плейлиста та додавання треків – групування контенту за персональними вподобаннями користувача;

– відтворення музики – основний функціонал медіаплеєра;

– перегляд статистики та аналіз уподобань – забезпечує доступ до інформації про активність прослуховування та розподіл жанрів користувача;

– експорт плейлиста у JSON – функція вивантаження даних.

Користувачі із правами адміністратора мають доступ до функцій авторизації, а також володіють правом на управління загальним контентом та користувачами, що дозволяє їм видаляти облікові записи або некоректно завантажений медіаконтент для підтримання стабільної роботи вебзастосунку.

На рисунку 3.2 зображено діаграму послідовності, яка описує процес завантаження аудіофайлу та демонструє взаємодію між клієнтським додатком, сервером, парсером Mutagen та базою даних під час вилучення метаданих.

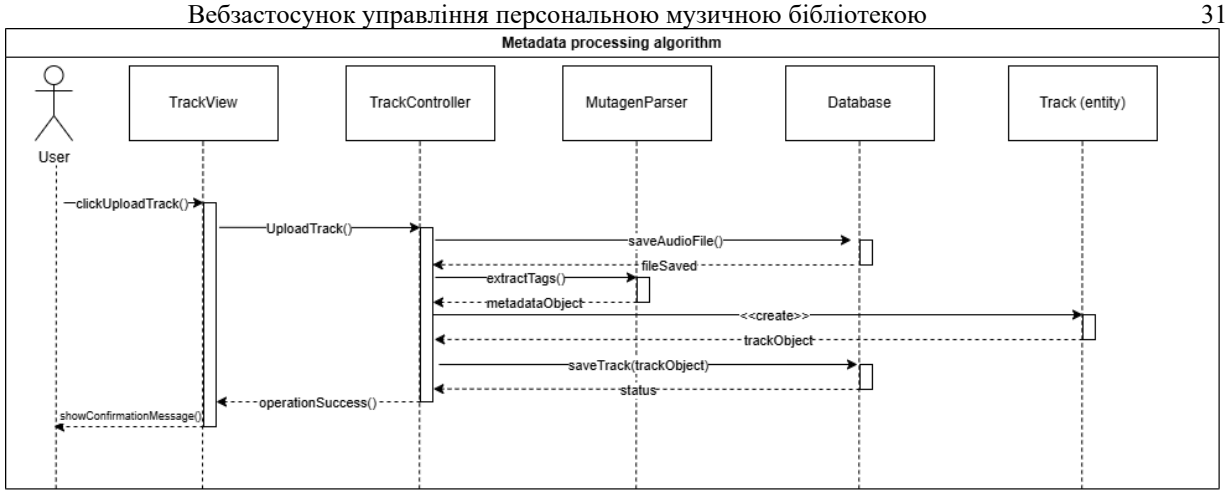

Рисунок 3.2 – Діаграма послідовності алгоритму обробки метаданих

Процес починається з того, що користувач викликає подію clickUploadTrack() у клієнтському компоненті представлення TrackView та далі логіка розгортається за наступним алгоритмом: спочатку TrackView делегує виконання бізнес-логіки серверному контролеру TrackController через метод UploadTrack(), потім контролер ініціює збереження фізичного аудіофайлу у сховищі через об'єкт Database викликом saveAudioFile(). Після підтвердження збереження файлу, TrackController звертається до спеціалізованого модуля MutagenParser викликом extractTags() для зчитування ID3-тегів безпосередньо з бінарного потоку завантаженого файлу, а на основі повернутого об'єкта метаданих контролер створює нову сутність – екземпляр класу Track. Сформований об'єкт треку передається до бази даних для фіксації запису функцією saveTrack(trackObject).

Після отримання статусу про успішне завершення транзакції, TrackController повертає сигнал operationSuccess() до інтерфейсу, що дозволяє компоненту TrackView вивести інформаційне повідомлення для кінцевого користувача.

На рисунку 3.3 зображено діаграму класів, яка відображає статичну структуру даних системи та логічні зв'язки між її основними сутностями. До ключових класів належать: User, який відповідає за керування профілем та авторизацію, Track, функціями якого є зберігання метаданих аудіофайлу та методи відтворення і редагування та Playlist, який містить у собі сукупність треків із функцією експорту. Діаграма також ілюструє кардинальність відношень, тобто,

один користувач може створювати безліч плейлистів, а один плейлист може містити безліч треків.

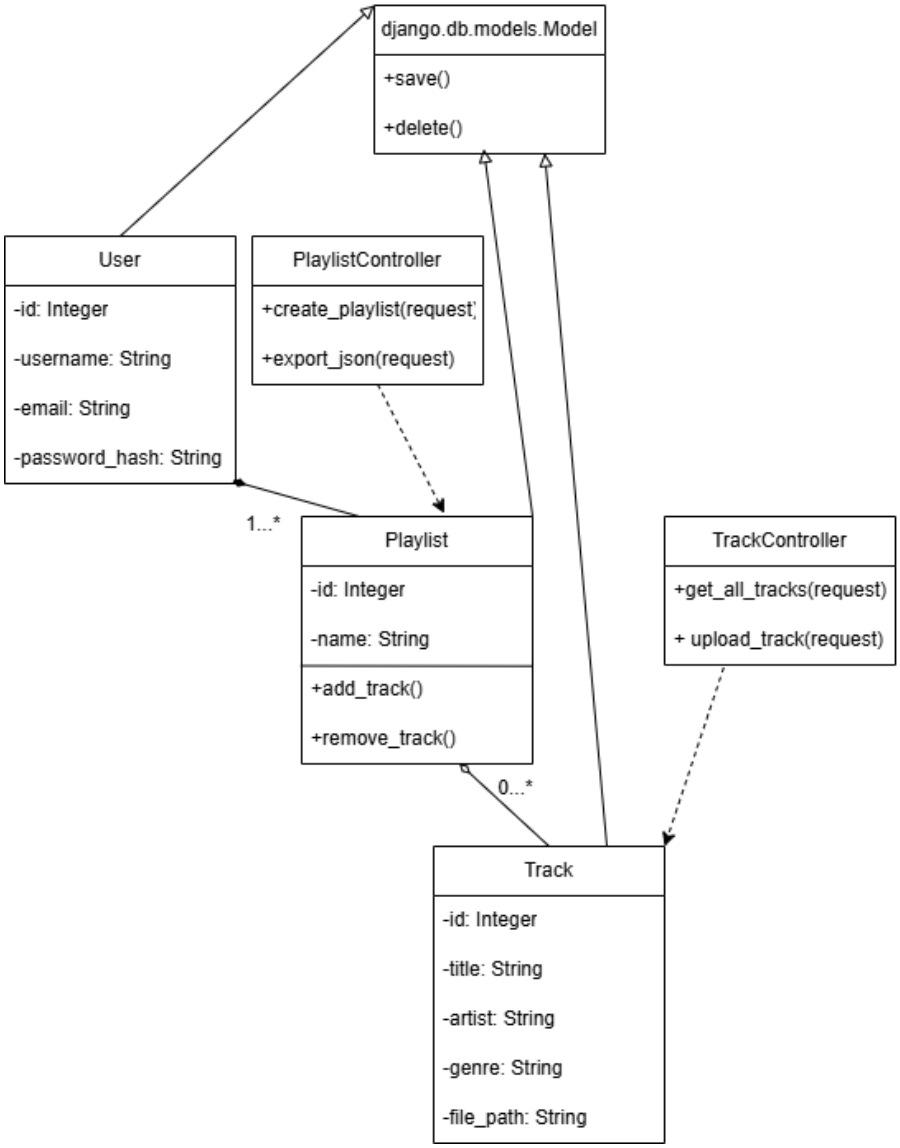

Рисунок 3.3 – Діаграма класів

На рисунку 3.4 подано діаграму розгортання, що ілюструє фізичну трирівневу архітектуру вебзастосунку. Вона складається з трьох основних вузлів:

– Client Device: клієнтський пристрій користувача, де у веббраузері виконується Frontend-компонент React SPA;

– Application Server: серверна частина, що містить Backend-компонент Django REST API, який обробляє бізнес-логіку;

– Database Server: рівень збереження даних, представлений файлом бази даних SQLite.

Зв'язок між клієнтом та сервером здійснюється за протоколом HTTP/HTTPS, а сервер застосунку взаємодіє з базою даних через SQL/ORM.

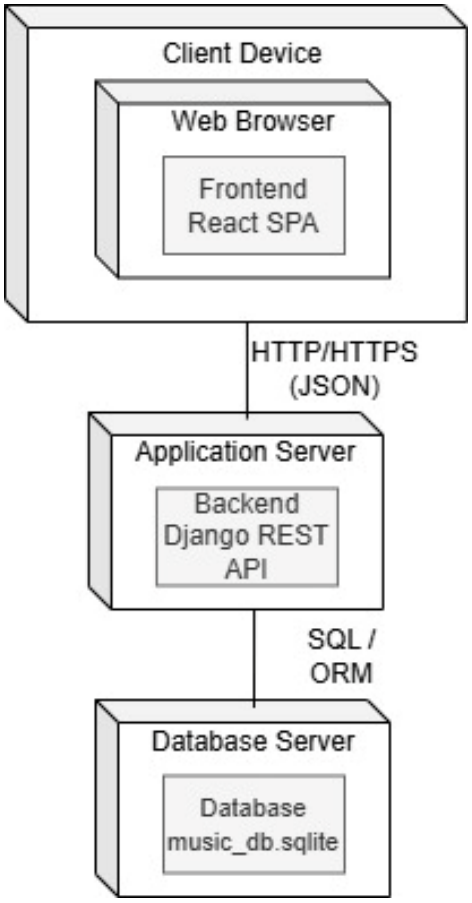

Рисунок 3.4 – Діаграма розгортання

Таким чином, проведене UML-моделювання дозволило наочно представити архітектурну логіку, статичну структуру даних та динаміку взаємодії компонентів розроблюваного вебзастосунку перед етапом його безпосередньої програмної реалізації.

3.3 Фізичне проєктування бази даних

У Django база даних нормалізується через моделі, які далі мігруються до самої БД SQLite. Були використані різні відношення для зв’язків. Many-to-Many використано для зв’язку пісень та плейлистів, а для історії прослуховувань використано One-to-Many.

Головною сутністю для збереження музичного контенту є таблиця Track. Вона містить не лише інформацію про сам файл, а й автоматично вилучені метадані

та зовнішні ключі для зв'язку з виконавцями та альбомами. Структуру таблиці Track наведено у таблиці 3.1.

Таблиця 3.1 – Структура таблиці бази даних Track

Назва поля	**Тип даних (Django ORM)**	**Призначення та опис**
id	AutoField (Integer)	Первинний ключ, унікальний ідентифікатор
title	CharField (Varchar)	Назва аудіотреку
artist_id	ForeignKey (Integer)	Зовнішній ключ до таблиці Artist зі зв'язком Many-to-One
album_id	ForeignKey (Integer)	Зовнішній ключ до таблиці Album зі зв'язком Many-to-One
duration	PositiveIntegerField	Тривалість треку в секундах, обчислюється через Mutagen
file_path	FileField (Varchar)	Відносний шлях до фізичного файлу у сховищі медіаданих
genre	CharField (Varchar)	Музичний жанр композиції
uploaded_at	DateTimeField	Точний час та дата завантаження файлу на сервер

Для забезпечення логічного групування контенту користувачами розроблено таблицю Playlist, а зв'язок між плейлистами та треками реалізовано через відношення Many-to-Many, що на фізичному рівні створює приховану транзитну таблицю перетинів. Структуру основної таблиці Playlist наведено у таблиці 3.2.

Таблиця 3.2 – Структура таблиці бази даних Playlist

Назва поля	**Тип даних (Django ORM)**	**Призначення та опис**
id	AutoField (Integer)	Первинний ключ, унікальний ідентифікатор
name	CharField (Varchar)	Назва списку відтворення
description	TextField (Text)	Розширений текстовий опис плейлиста
user_id	ForeignKey (Integer)	Зовнішній ключ до базової таблиці User
cover_image	ImageField (Varchar)	Шлях до зображення обкладинки плейлиста
created_at	DateTimeField	Дата створення списку відтворення

Для того щоб збирати аналітику та для роботи системи досягнень було створено таблицю ListeningHistory, яка фіксує кожну взаємодію користувача з аудіоконтентом. Структуру цієї таблиці наведено у таблиці 3.3.

Таблиця 3.3 – Структура таблиці бази даних ListeningHistory

Назва поля	**Тип даних (Django ORM)**	**Призначення та опис**
id	AutoField (Integer)	Первинний ключ
user_id	ForeignKey (Integer)	Ідентифікатор користувача, який прослухав трек

Кінець таблиці 3.3

track_id	ForeignKey (Integer)	Ідентифікатор прослуханої композиції
listened_at	DateTimeField	Автоматична фіксація часу

Спроєктована схема даних орієнтована на ефективне сортування об’єктів у медіатеці та забезпечує швидке виконання фільтрації за атрибутами та дозволяє формувати складну статистику прослуховувань без сильного навантаження на SQLite.

3.4 Проєктування інтерфейсу користувача

Для проєктування інтерфейсу користувача враховувались такі аспекти як: адаптивність, інтуїтивність у використанні, естетичне задоволення та можливість налаштування теми. При створенні інтерфейсу було вирішено розділити його на декілька частин: бічна панель з переходом на різні частини сторінок, нижня частина з керуванням піснею та верхня панель з налаштуванням, пошуком та сторінкою користувача.

Застосунок створено у сучасному мінімалістичному стилі, але з деякими відсиланнями на максимализм минулого, в тому числі, візуалізатор звуку який рухається під такт музиці та освітленні частини, які можуть визвати відчуття ностальгії у користувачів, які хочуть відчути те що відчували у нульових керуючи своєю музичною бібліотекою.

За допомогою спеціалізованого продукту було створено mockup інтерфейсу користувача застосунку. Рисунок 3.5 зображує базовий шаблон інтерфейсу, що є спільним для всіх сторінок застосунку та включає в себе верхню панель із пошуком та профілем, бічну панель навігації, яка містить Home, Library, Statistic та Playlists та нижню панель медіаплеєра з елементами керування відтворенням, прогрес-баром та регулятором гучності.

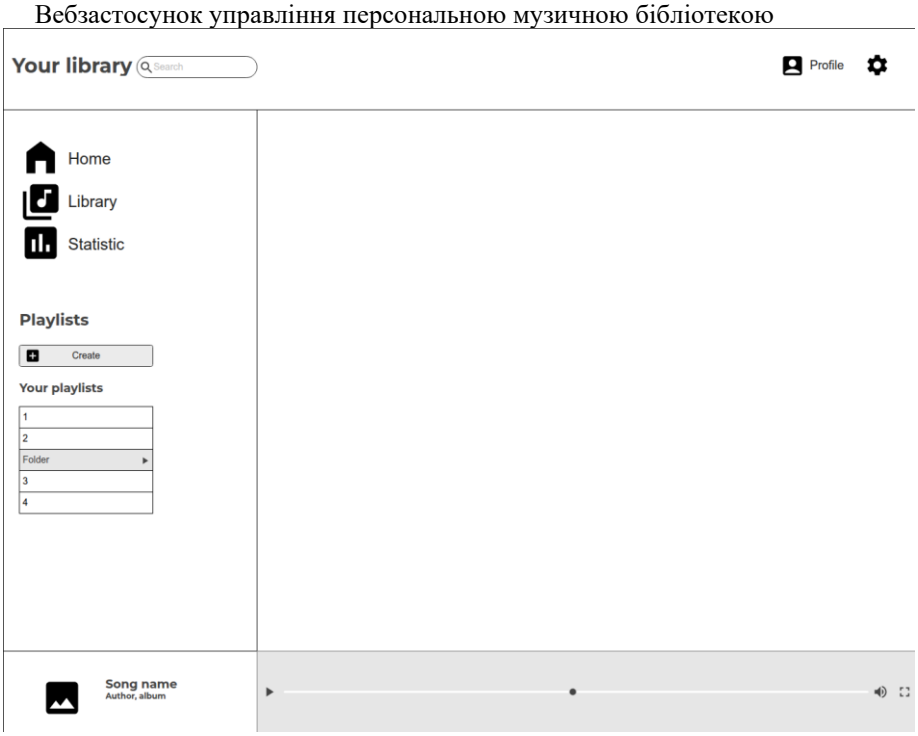

Рисунок 3.5 – Головна розмітка

На рисунку 3.6 зображено сценарій пошуку та фільтрації музики. Центральна область містить таблицю зі списком усіх завантажених треків користувача, де відображаються назва, виконавець, альбом та тривалість. Колонка “Actions” дозволяє додавати треки до обраного або викликати меню додаткових дій.

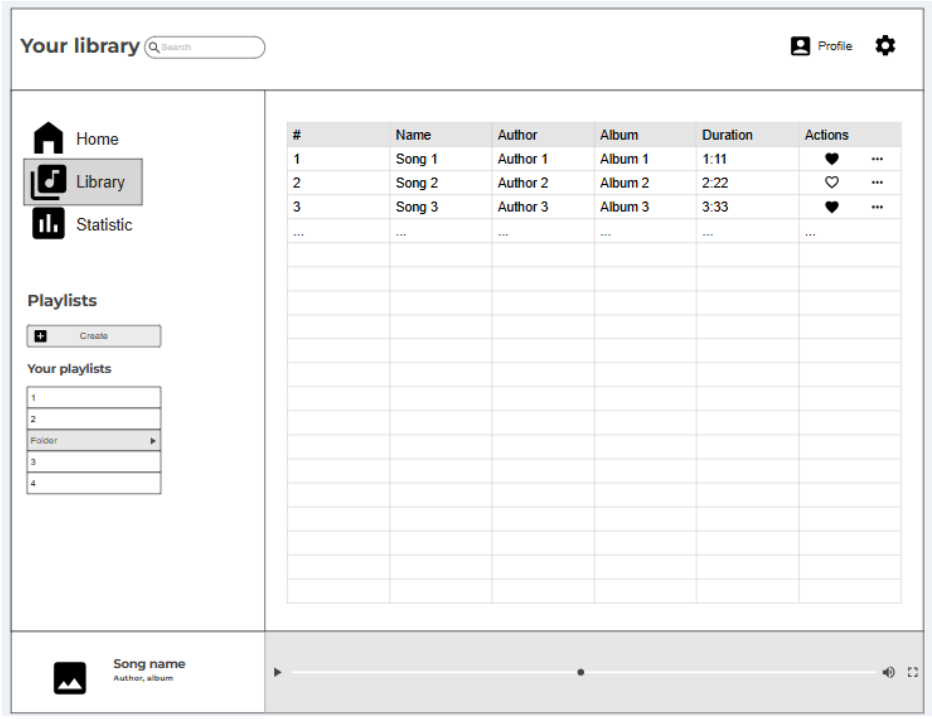

Рисунок 3.6 – Бібліотека пісень

Рисунок 3.7 зображає сценарій завантаження нового треку та редагування його метаданих. Макет містить зону для перетягування файлів та форму для введення або коригування інформації про пісню, автора та альбом, а також можливість зміни обкладинки треку перед збереженням у базі даних.

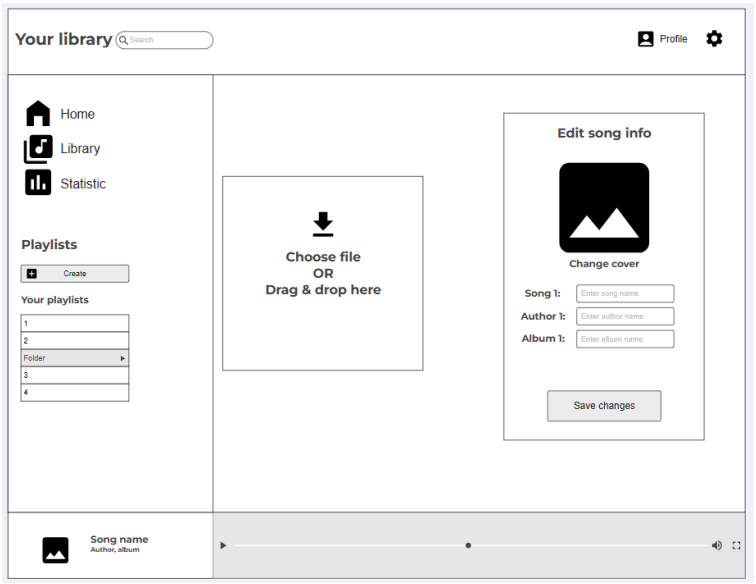

Рисунок 3.7 – Завантаження та редагування пісень

На рисунку 3.8 зображено інтерфейс керування персональними колекціями. На сторінці розміщені картки плейлистів із короткою інформацією про кількість треків та описом. Кожна картка містить кнопку “Export”, що реалізує сценарій експорту даних плейлиста у формат JSON для обміну чи резервного копіювання.

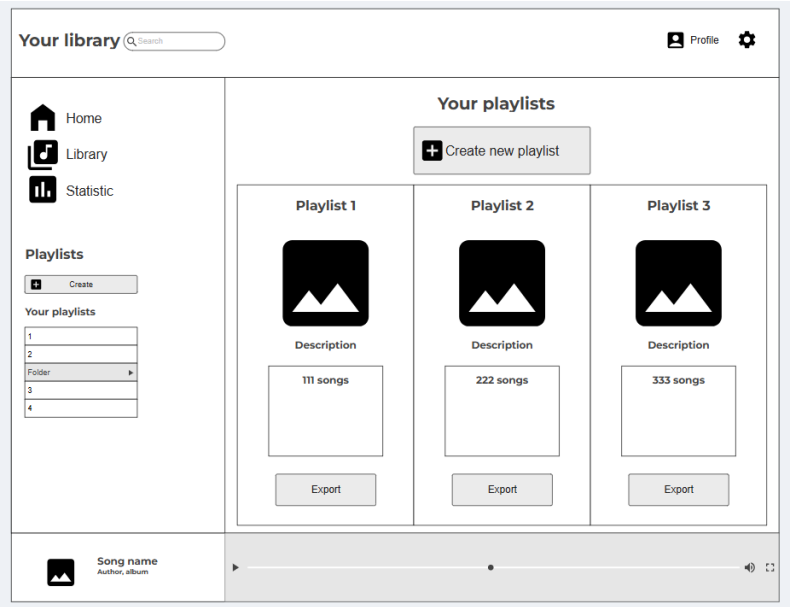

Рисунок 3.8 – Розділ «Плейлисти»

Рисунок 3.9 ілюструє аналіз музичних уподобань користувача. Для отримання інформації про свої музичні вподобання, реалізована кругова діаграма, яка візуалізує розподіл за жанрами та гістограму активності прослуховування.

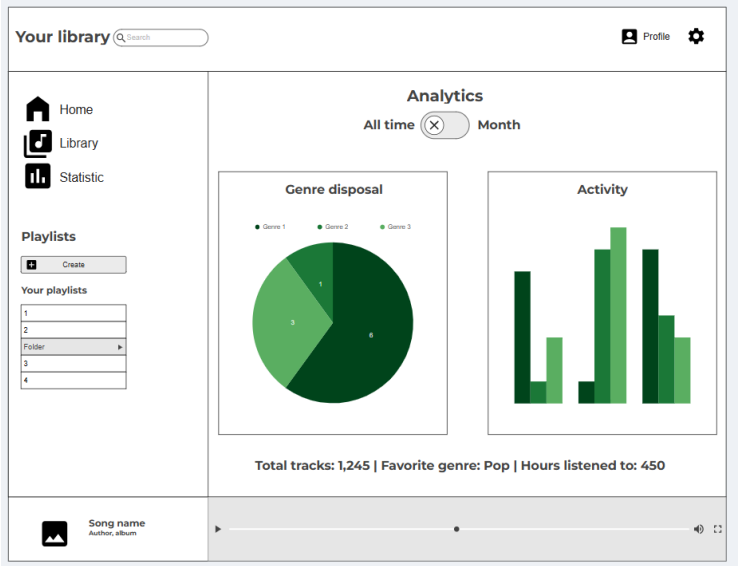

Рисунок 3.9 – Сторінка статистики

Загалом, хоч і кінцевий продукт повністю не відповідає мокапам, але основні принципи зберіглись та складають основу вигляду кожного компонента.

Висновки до розділу 3

У третьому розділі було виконано обґрунтування вибору триланкової архітектури системи та використання REST API. Було виконано UML-моделювання та усі створені діаграми описують структуру і динаміку роботи вебзастосунку.

Вимоги розроблюваної системи були виконані. Застосовано алгоритми для автоматичного вилучення ID3-тегів та впроваджено механізми безпечної аутентифікації на базі JWT-токенів. Це все забезпечує гнучкість для подальшого масштабування застосунку, а також гарантує стабільну роботу системи.

Також спроєктовано графічний інтерфейс користувача із поєднанням візуальної складової максималізму минулого та сучасного мінімалізму. Інтерфейс спроєктовано так, щоб він був інтуїтивним для використання та зручним що підтверджується наведеними макетами. Виконане проєктування створює повний і надійний фундамент для безпосередньої програмної реалізації системи.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ, ТЕСТУВАННЯ ТА КЕРІВНИЦТВО КОРИСТУВАЧА

4.1 Реалізація серверної логіки та обробки медіаконтенту

Серверну частину вебзастосунку реалізовано мовою Python з використанням фреймворку Django та архітектурної надбудови Django REST Framework для побудови API.

Інформаційна модель бази даних є основою серверної логіки та описана у файлі models.py. Головною сутністю яка зберігає аудіоконтент є клас Track. Для зберігання зв'язків між треками та виконавцями чи альбомами використано тип даних ForeignKey зі зв'язком One-to-Many, а для збереження вподобань користувачів – Many-to-Many.

Найважливішим елементом серверної обробки є контролер track_controller у views.py, який відповідає за прийом файлів від клієнта. Під час завантаження файлу завдяки методу POST сервер ініціює вилучення метаданих, що допомагає автоматизувати структурування бібліотеки, а для цього інтегровано бібліотеку mutagen.

Лістинг фрагменту коду автоматичного зчитування тривалості файлу з views.py:

```
try:
    path = track.file_path.path
    audio = mutagen.File(path)
    if audio is not None and audio.info:
        track.duration = int(audio.info.length)
        track.save()
except Exception as e:
    print(f"Помилка визначення тривалості: {e}")
```

У цьому фрагменті коду продемонстровано як у системі відбувається завантаження файлу. Система зчитує властивості аудіофайлу та зберігає тривалість до бази даних.

Також серверна логіка є особливу систему досягнень, що додає елемент геймифікації до користувацького досвіду. Логіка досягнень реалізована на рівні

серіалізатора UserSerializer у файлі serializers.py за допомогою методу get_achievements. Це працює так, що система динамічно обчислює кількість прослуханих треків у history_count та збережених пісень у fav_count і генерує масив з об'єктами відповідних досягнень.

Лістинг фрагменту коду логіки системи досягнень з serializers.py:

```
def get_achievements(self, obj):
    achievements = []
    history_count = obj.history.count()
    if history_count > 0:
        achievements.append({"title": "Перший крок", "desc": "Послухав свій перший трек", "icon": "headphones", "color": "#3498db"})
    if history_count >= 10:
        achievements.append({"title": "Меломан", "desc": "Відтворив 10 треків", "icon": "fire", "color": "#e67e22"})
    if history_count >= 50:
        achievements.append({"title": "Ді-джей", "desc": "Відтворив 50 треків", "icon": "disc", "color": "#9b59b6"})
    return achievements
```

Важливим архітектурним рішенням у серверній частині стало використання механізму сигналів фреймворку Django. Для того щоб кожен новий зареєстрований користувач автоматично отримував розширений профіль, наприклад, для збереження аватара, було застосовано декоратор @receiver на подію post_save моделі User. Це дозволяє розділити логіку реєстрації та створення профілю, уникаючи перевантаження контролера авторизації зайвим кодом.

Захист даних реалізовано через безпеку API-ендпоінтів, так як усі контролери, які працюють з персональними даними користувача, захищені за допомогою декораторів @permission_classes([IsAuthenticated]), що дає гарантію на те, що жоден неавторизований клієнт не зможе отримати доступ до ендпоінтів або виконати ін'єкцію даних, оскільки сервер перевіряє валідність JWT-токена ще до моменту виконання основної функції.

Також у контролері track_detail_controller реалізовано інтелектуальний алгоритм каскадного очищення. Це працює так, що при видаленні користувачем пісні, система перевіряє, чи залишилися в базі даних інші пісні цього виконавця або

з цього альбому. Якщо видалений трек був останнім, система автоматично видаляє порожніх виконавців та альбоми, запобігаючи засміченню бази даних SQLite неактуальними записами.

4.2 Розробка клієнтської частини

Клієнтська частина побудована завдяки використання бібліотеки React. Взаємодія з сервером відбувається через асинхронні HTTP-запити, а так як бекенд захищений JWT-токенами, кожен запит містить заголовок Authorization: Bearer <token>. На рисунку 4.1 зображено структуру проєкту у редакторі.

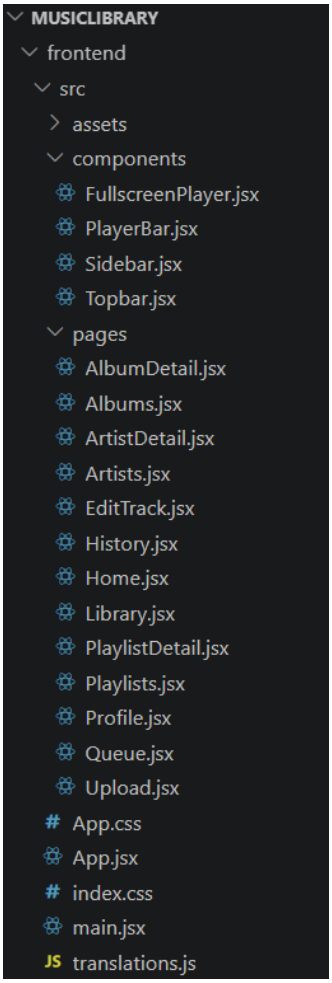

Рисунок 4.1 – Структура папок React-проєкту

Згідно з наведеною структурою, систему розділено на два основні блоки: компоненти спільного використання, такі як Sidebar.jsx, PlayerBar.jsx, Topbar.jsx та маршрутизовані сторінки, такі як Home.jsx, Library.jsx, Profile.jsx. Взаємодія

фронтенду з Django-сервером відбувається через асинхронні HTTP-запити з обов'язковою передачею JWT-токена у заголовку Authorization.

Найбільш технічно складною частиною фронтенд-розробки стала реалізація глобального компонента медіаплеєра. Оскільки плеєр повинен працювати безперервно під час навігації користувача між різними сторінками, його стан винесено на найвищий рівень ієрархії компонентів (див. додаток А).

Для відтворення музики та синхронізації інтерфейсу з подіями браузера використано набір хуків React, а замість прямого посилання на аудіофайл, застосунок попередньо завантажує його. Це дозволяє коректно відображати стан завантаження та уникати затримок під час відтворення.

Лістинг фрагменту коду логіки безпечного завантаження аудіофайлу через Blob у PlayerBar.jsx:

```
  useEffect(() => {
    if (!currentTrack || !currentTrack.file_path) {
      setBlobUrl(null);
      return;
    }
    let objectUrl = null;
    setIsLoadingAudio(true);
    const url = currentTrack.file_path.startsWith('http') ? currentTrack.file_path : `http://127.0.0.1:8000${currentTrack.file_path}`;
    fetch(url)
      .then(res => res.blob())
      .then(blob => {
        objectUrl = URL.createObjectURL(blob);
        setBlobUrl(objectUrl);
        setIsLoadingAudio(false);
      })
      .catch(err => { console.error(err); setIsLoadingAudio(false); });
    return () => { if (objectUrl) URL.revokeObjectURL(objectUrl); };
  }, [currentTrack]);
```

Цей фрагмент коду демонструє використання хука useEffect для відстеження зміни поточного треку. При зміні композиції система робить запит на сервер, отримує аудіофайл, перетворює його на локальний ObjectURL та передає у

прихований HTML5-елемент <audio>. Для того щоб запобігти витокам оперативної пам'яті створена функція очищення, бо при довгому використанню у браузері користувача без обробки подібні витоки є частим явищем.

Через побудову застосунку за використання принципу Single-Page Application, стандартний механізм переходу між HTML-сторінками мав бути замінений на динамічну маршрутизацію на стороні клієнта за допомогою бібліотеки React Router. Це дало змогу реалізувати меню навігації, яке перемикає відображувані сторінки, наприклад, з Library.jsx на Statistic.jsx миттєво, не звертаючись до сервера за новими HTML-документами. Такий підхід обрано для економії мережевого трафіку та є критично необхідним для безперервної роботи компонента медіаплеєра.

Для того щоб забезпечити правильний обмін даними з бекендом створено модуль API-клієнта. Замість формування HTTP-заголовків в кожному компоненті по окремості, логіка прикріплення токенів JWT була винесена на рівень глобальних налаштувань, а коли термін дії цього токена спливає, клієнтською частиною одразу автоматично перехоплюється помилка 401 Unauthorized від сервера та ініціюється фоновий запит на оновлення сесії, що забезпечує безперебійний користувацький досвід.

4.3 Тестування програмного продукту

За допомогою набору для автоматизації тестів Django APITestCase, який імітує реальні запити клієнтської частини до сервера, розроблено набір тестів, що дозволяє верифікувати працездатність системи [16]. Ці тести дозволили перевірити права доступу та цілісність даних при виконанні критичних операцій. З результатів на таблиці 4.1 видно що усі розроблені тестові сценарії завершилися успішно, що беззаперечно підтверджує високу надійність серверної архітектури та її готовність до експлуатації.

Таблиця 4.1 – Результати тестування основних модулів

Модуль	**Тест-кейс**	**Очікуваний результат**	**Статус**
Авторизація	Запит токена	Помилка 401 Unauthorized	Пройдено
Плейлисти	Видалення чужого списку	Помилка 403 Forbidden	Пройдено
Медіа	Завантаження MP3	Створення запису та парсинг тегів	Пройдено
Профіль	Реєстрація	Автоматичне створення UserProfile	Пройдено

Результати проведеного тестування зображено на рисунку 4.2.

```
C:\Users\bogda\OneDrive\Документи\MusicLibrary>color f0

C:\Users\bogda\OneDrive\Документи\MusicLibrary>.\venv\Scripts\activate

(venv) C:\Users\bogda\OneDrive\Документи\MusicLibrary>python manage.py test
Found 5 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....
----------------------------------------------------------------------
Ran 5 tests in 11.811s

OK
Destroying test database for alias 'default'...
```

Рисунок 4.2 – Результати тестування

Усі тести проведені успішно, без помилок, що свідчить про цілісність даних під час здійснення критичних операцій (див. додаток Б).

4.4 Керівництво користувача

Коли користувач заходить на головну сторінку, він бачить перед собою привітання та запрошення до прослуховування пісень. Також користувач може

увійти щоб відкрити набагато більше функцій. На рисунку 4.3 зображено головну сторінку для неавторизованого користувача.

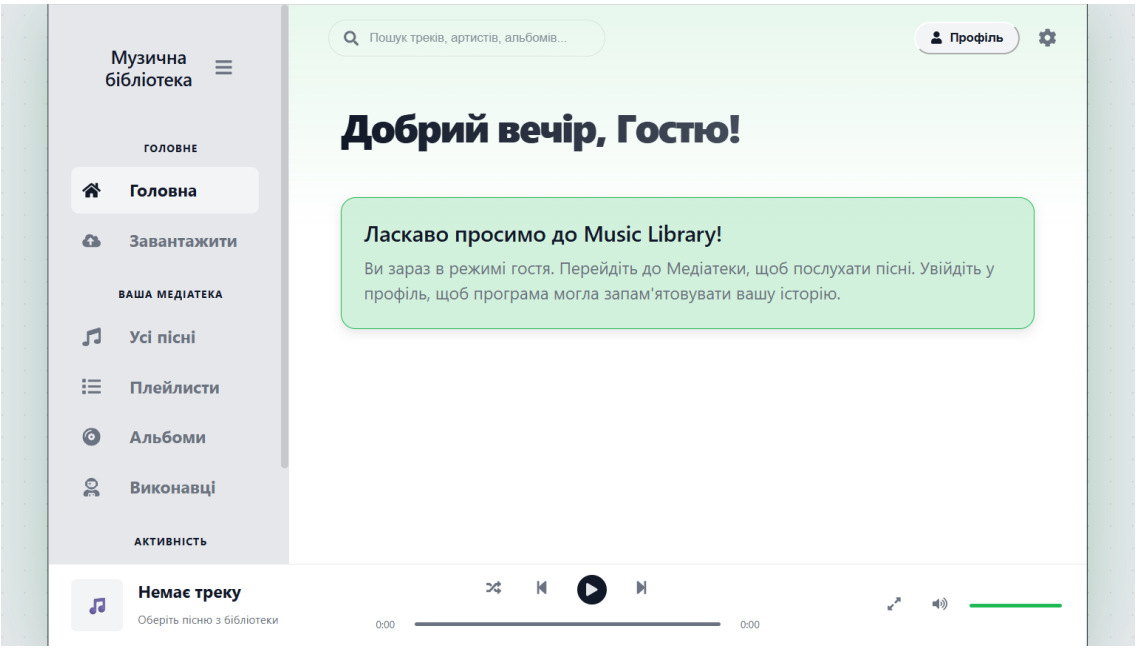

Рисунок 4.3 – Головна сторінка музичної бібліотеки для неавторизованого користувача

Для того щоб зареєструватись або аутентифікуватись, користувачу потрібно натиснути на кнопку «Профіль» зверху та перейти на сторінку користувача. На рисунку 4.4 зображений вигляд цієї сторінки до авторизації.

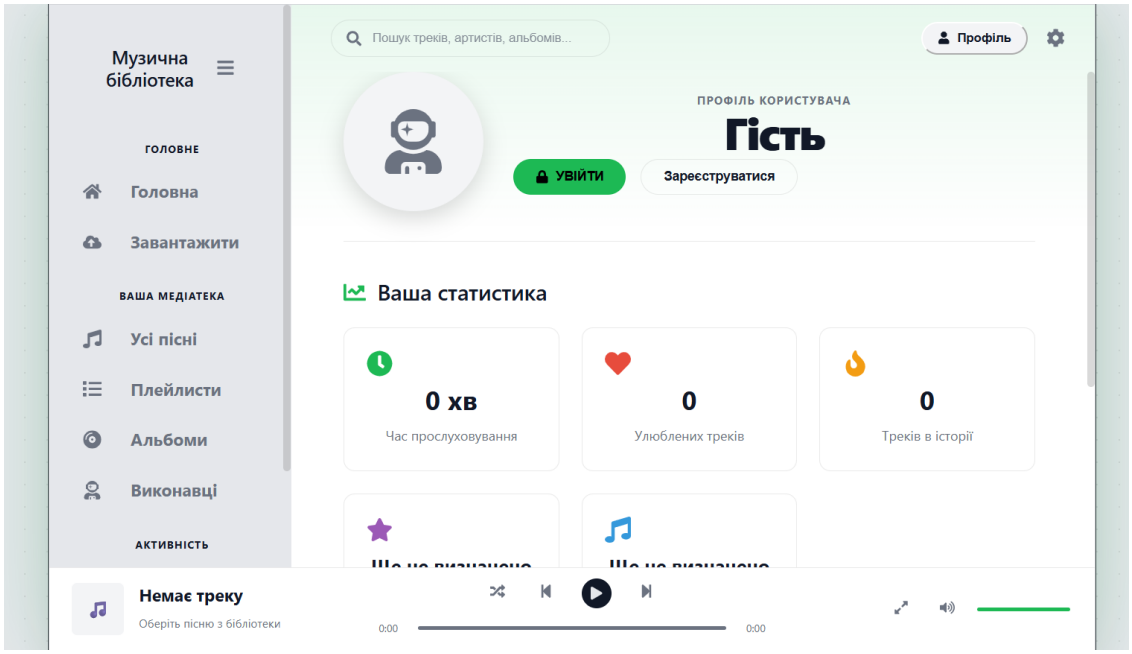

Рисунок 4.4 – Сторінка неавторизованого користувача

На рисунку 4.5 зображений вигляд сторінки авторизованого користувача, на якій можна побачити певну статистику і також змінити іконку, ім’я та пароль акаунта. Також є можливість вийти з акаунту, подивитись свої досягнення та очистити історію.

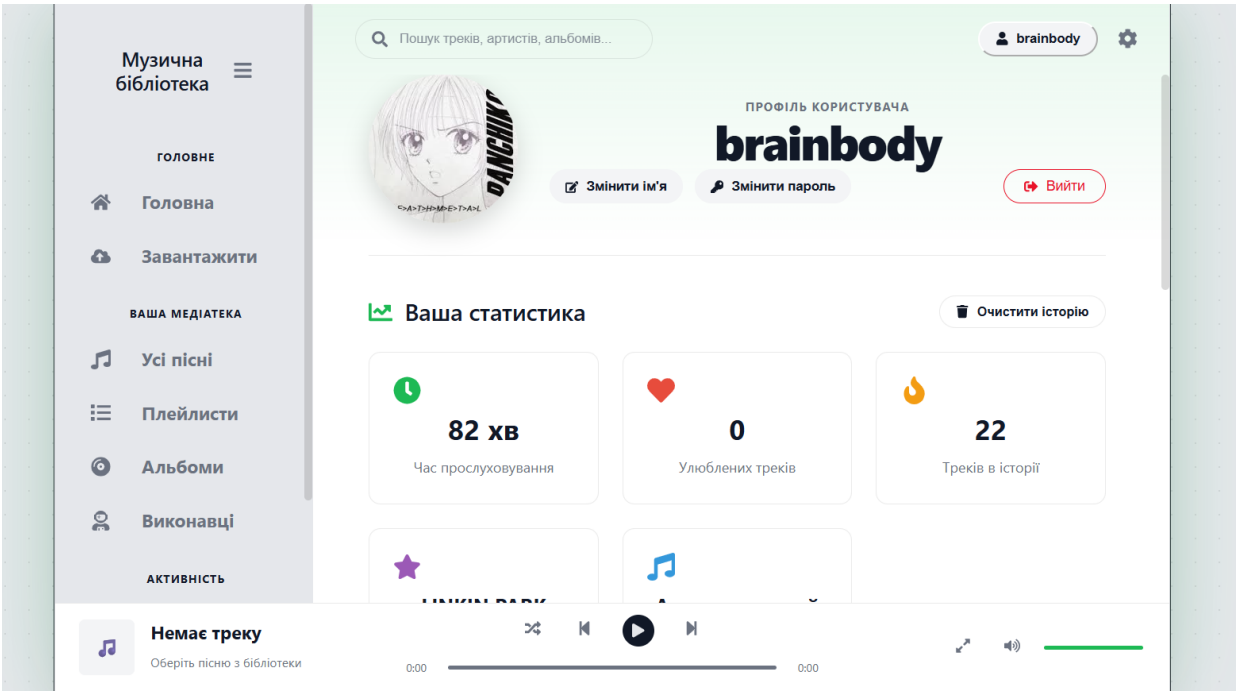

Рисунок 4.5 – Сторінка авторизованого користувача

Для того щоб користувач міг завантажити нову пісню до медіатеки, йому треба перейти на сторінку «Завантажити». Там є можливість завантажити одну чи декілька пісень та одразу змінити або додати інформацію про них. На рисунку 4.6 зображена ця сторінка.

При масовому завантаженні, наприклад, цілого альбому, компонент React генерує покрокову форму, пропонуючи користувачу послідовно перевірити або відредагувати теги, назви та обкладинки для кожної композиції окремо. Це забезпечує високу якість каталогізації контенту, оскільки кінцевий користувач повністю контролює правильність заповнення полів перед формуванням підсумкового об'єкта FormData для передачі через REST API.

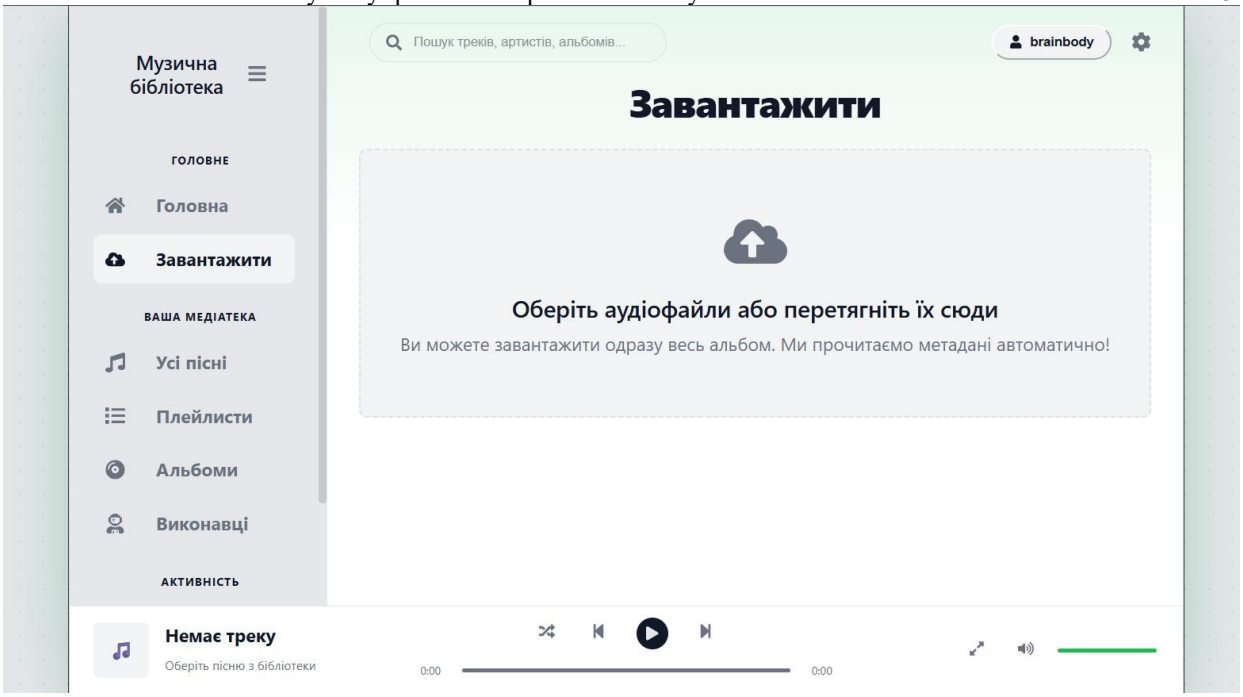

Рисунок 4.6 – Сторінка завантаження

На рисунку 4.7 зображена сторінка з редагуванням та додаванням пісень. На ній інтуїтивно можна змінити усі поля, додати обкладинку та текст, або видалити пісню.

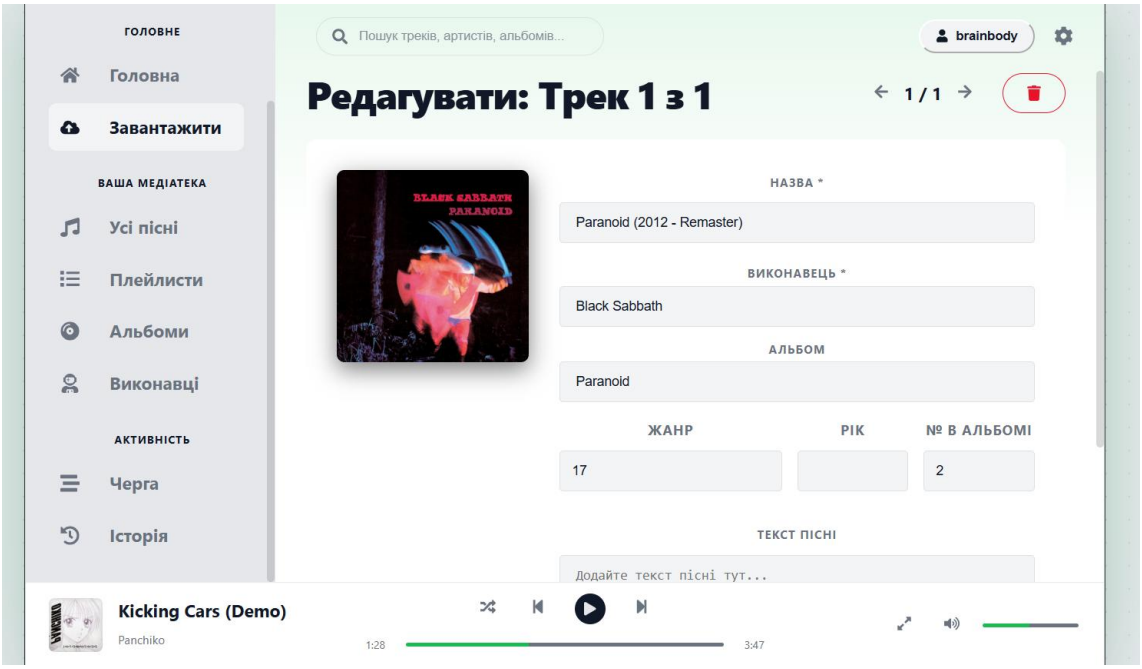

Рисунок 4.7 – Додавання та редагування пісні

Застосунок також має об’ємний функціонал з налаштування візуальної складової. Є можливість змінити мову, головний колір сайту, який впливає на усі

кнопки та інші елементи і також присутній вибір між світлою та темною темами сайту. На рисунку 4.8 зображений вигляд налаштувань.

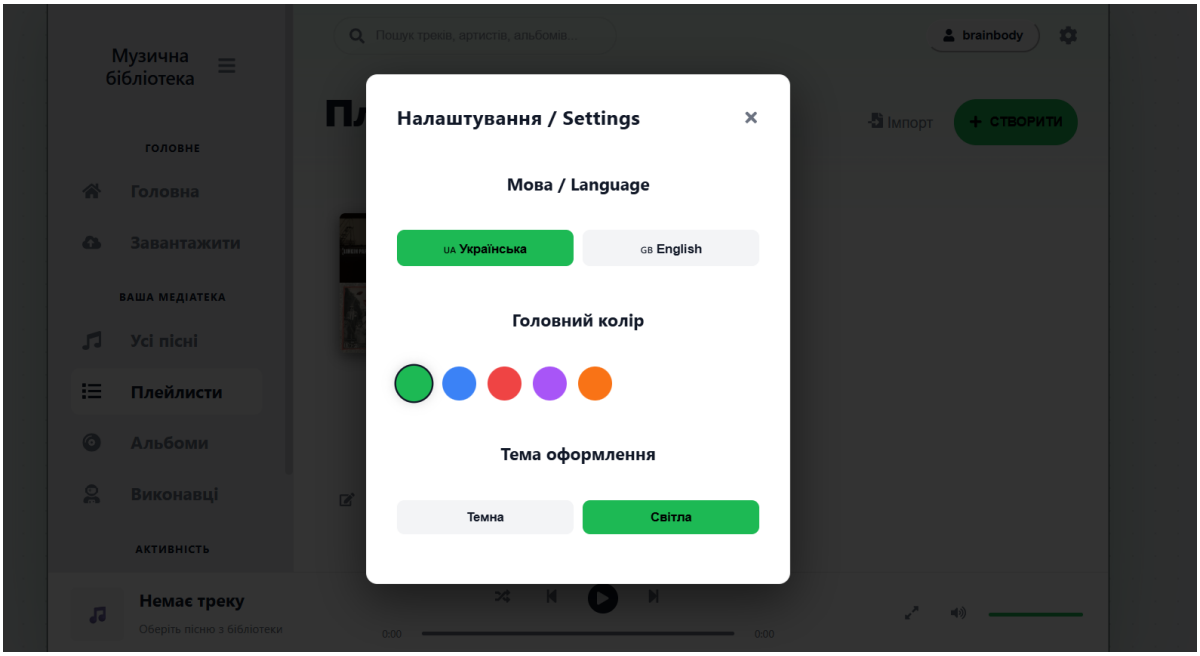

Рисунок 4.8 – Налаштування застосунку

Усі прослухані пісні користувачів зберігаються на вкладці «Історія». Там показані відповідні пісні та є можливість їх видалити з історії що видно на рисунку 4.9.

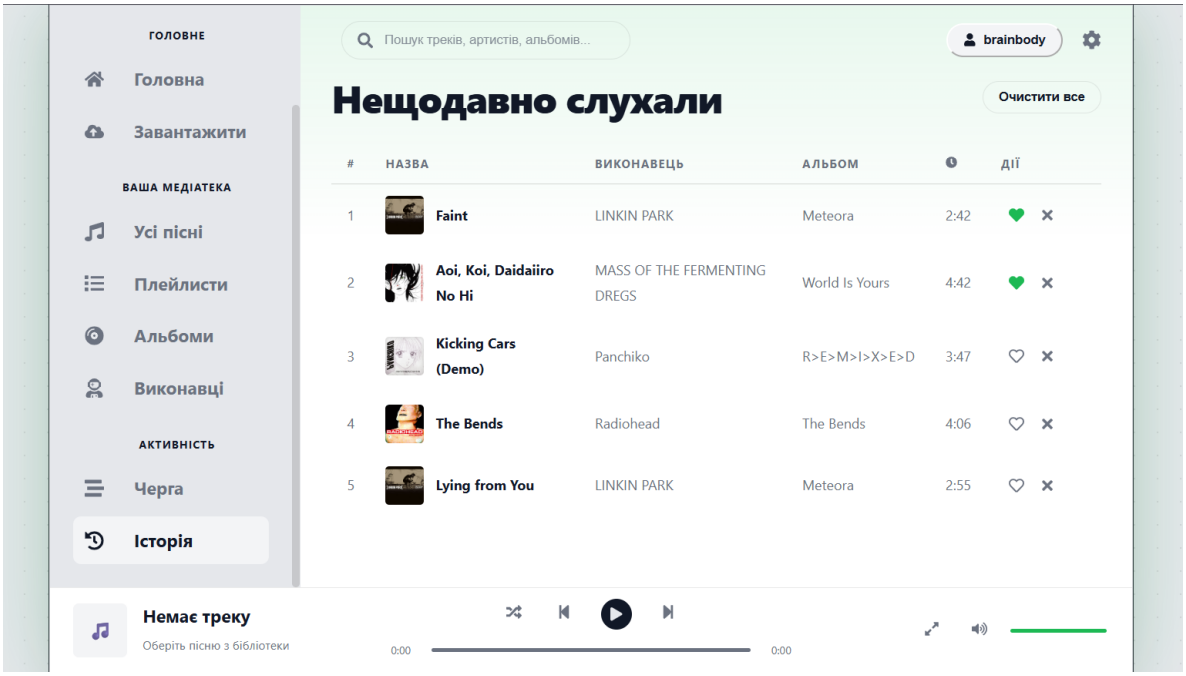

Рисунок 4.9 – Вкладка «Історія»

На рисунку 4.10 зображений головний елемент музичної бібліотеки – медіатека. Тут представлені усі завантажені пісні, також є можливість звідси перейти на їх виконавців або альбом натиснувши на відповідні назви. Також можна з кожною піснею зробити певні дії: додати до улюблених, редагувати інформацію про них, додати до плейлиста та додати до черги. На скріншоті також видно поточну пісню яка грає зараз у плеєрі та можливість керування як в будь-якому плеєрі. Усю медіатеку користувача можна відсортувати за різними полями.

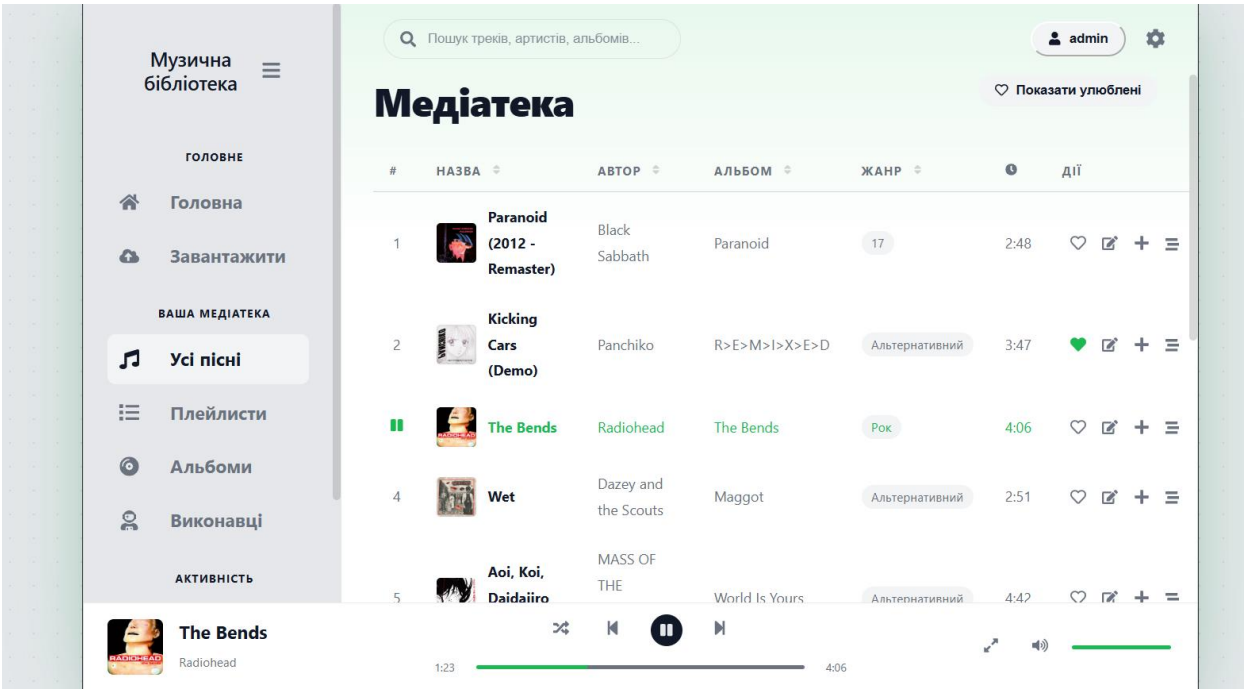

Рисунок 4.10 – Головна медіатека

На рисунку 4.11 представлено режим повноекранного плеєра, розроблений для максимального занурення користувача у процес прослуховування. Цей спеціалізований інтерфейс не лише виводить обкладинку поточного релізу у високій роздільній здатності, але й надає зручний доступ до перегляду тексту пісні. Окрім розширених елементів керування відтворенням, сторінка оснащена динамічним візуалізатором. Графічні елементи візуалізації плавно реагують на зміни стану відтворення, створюючи ефект пульсації в такт музиці, що значно підвищує інтерактивність та покращує загальний користувацький досвід.

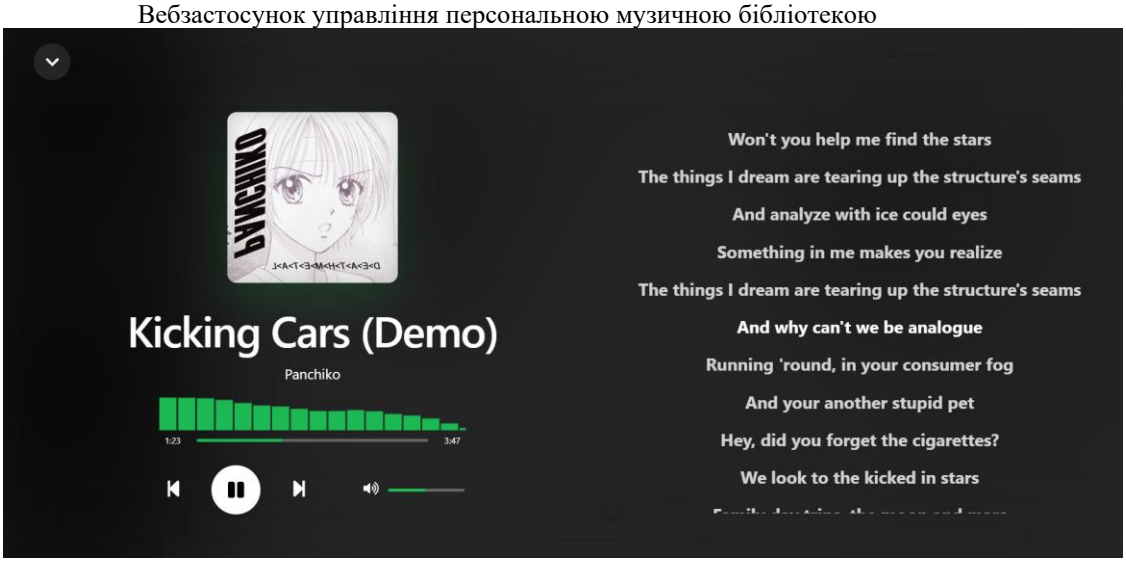

Рисунок 4.11 – Великий плеєр

На рисунку 4.12 зображено одночасно змінену тему на темну та основний колір застосунку на помаранчевий. Це показує наскільки обширні можливості кастомізації є у користувачів.

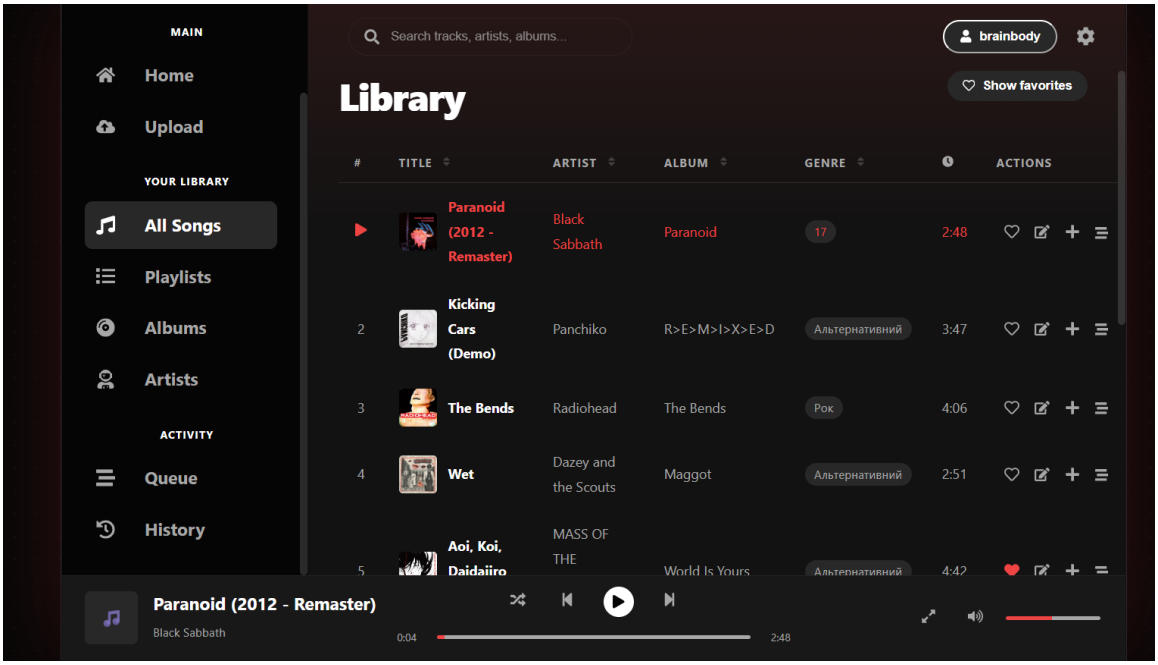

Рисунок 4.12 – Темна тема та змінений колір

На сторінці користувача також є розподіл по жанрах який заповнюється відповідно до кількості прослуханих пісень відповідного жанру та досягнення які користувач заробляє у ході використання вебзастосунку. Це зображено на рисунку 4.13.

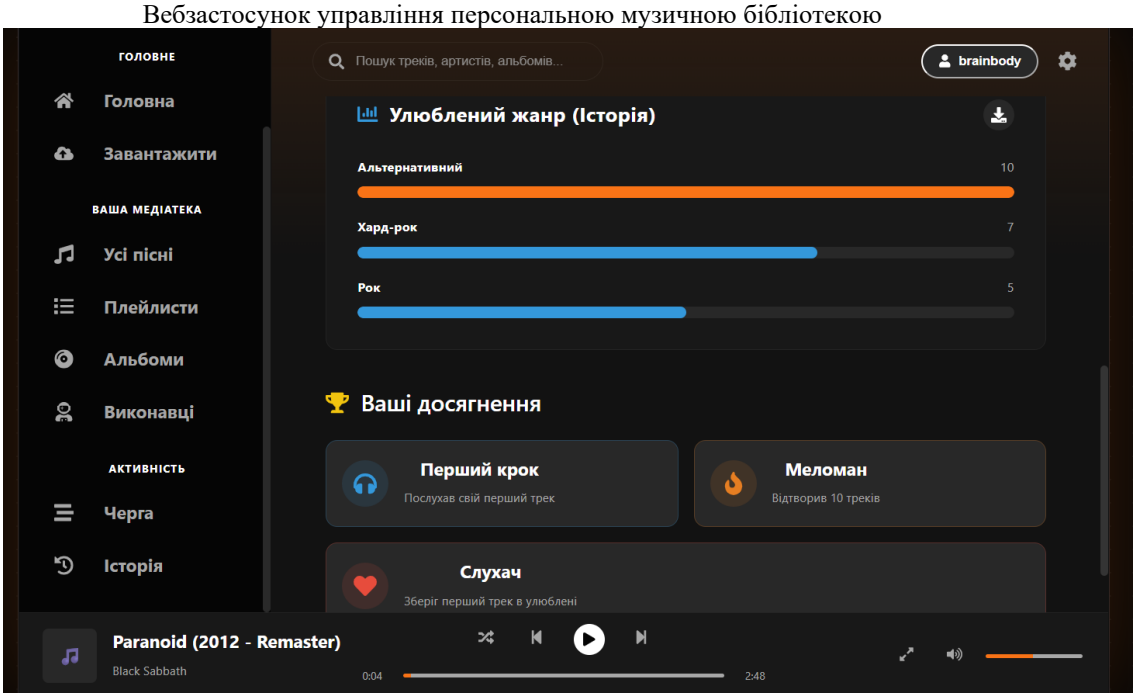

Рисунок 4.13 – Досягнення та розподіл жанрів на сторінці користувача

На рисунку 4.14 зображено панель адміна, яка доступна лише користувачам із відповідною роллю, що дозволяє йому переглянути усіх зареєстрованих користувачів та у разі чого видалити їх.

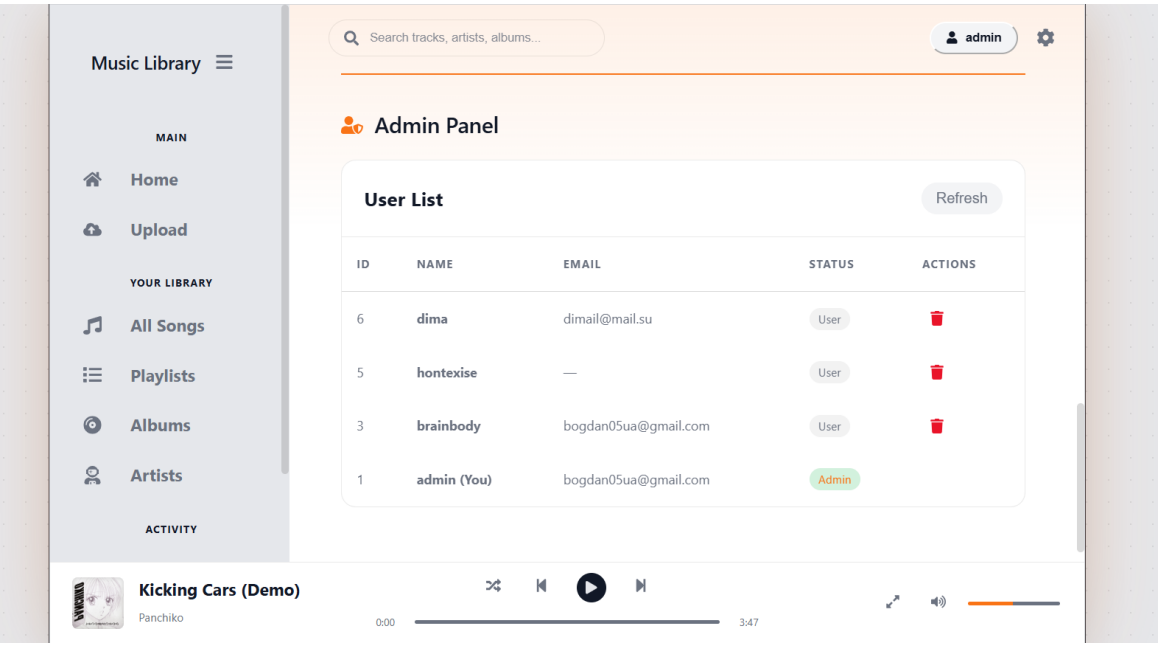

Рисунок 4.14 – Вигляд адмін панелі

На рисунку 4.15 зображено вихідний графічний матеріал – розподіл жанрів, який користувачі можуть завантажити як зображення безпосередньо із сайту та ділитись з іншими.

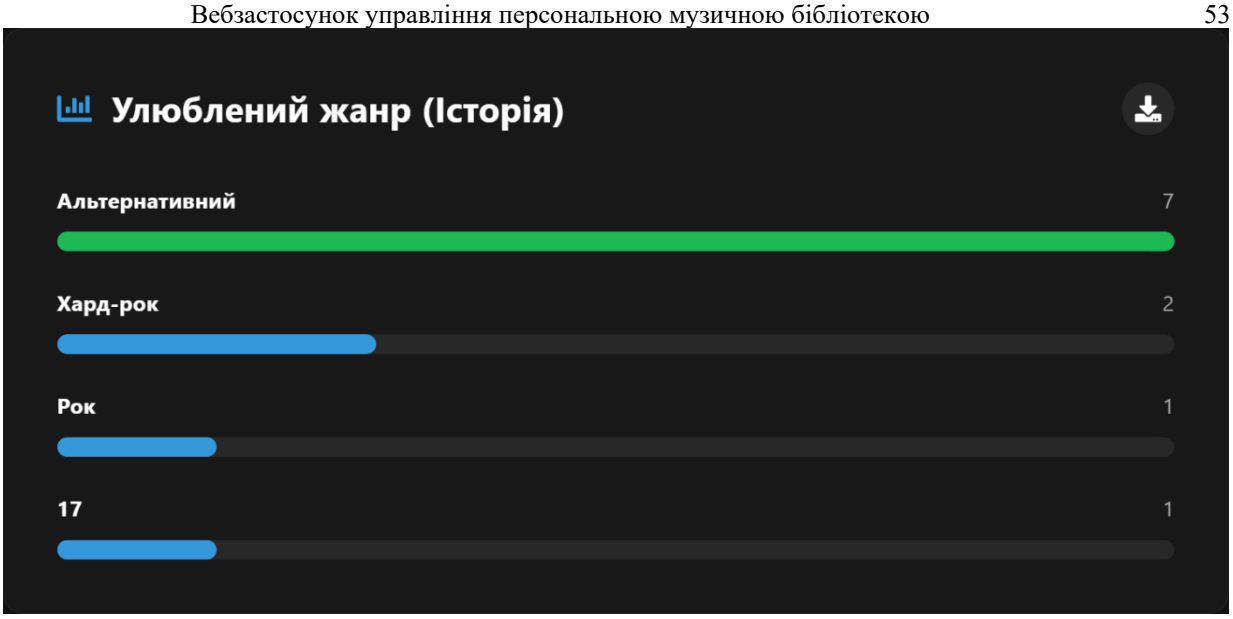

Рисунок 4.15 – Завантажена зі сторінки статистика

Застосунок вийшов приємним візуально та зручним у користуванні, що має забезпечити усім якісний досвід у користуванні кінцевим продуктом.

Висновки до розділу 4

У четвертому розділі було здійснено програмну реалізацію розробленої системи. Серверну частину успішно побудовано на базі фреймворку Django з використанням REST API, що дозволило ефективно реалізувати бізнес-логіку обробки аудіофайлів, зокрема автоматичне вилучення ID3-тегів за допомогою бібліотеки Mutagen.

Використання бібліотеки React дало можливість створити клієнтську частину у вигляді Single-Page Application, що дало змогу створити інтерфейс, який є одночасно адаптивним та інтуїтивно зрозумілим для користування. Було також проведене автоматизоване тестування, яке показало що система розроблена без критичних помилок і здатна на витримування певних навантажень. Керівництво користувача повністю демонструє різні сценарії роботи із застосунком що додатково підтверджує його готовність до використання.

ВИСНОВКИ

У результаті виконання кваліфікаційної бакалаврської роботи було отримано робочу версію вебзастосунку управління персональною музичною бібліотекою. Виконано основну мету, яка заключалась у розробці вебсистеми персонального менеджменту музичної медіатеки для автоматизації процесів зберігання, структурування та аналітики аудіофайлів.

Для реалізації серверної частини було використано фреймворк на Python Django, який надає зручні інструменти для створення REST API, обробки запитів та взаємодії з базою даних, а клієнтську частину розроблено за допомогою бібліотеки React.

У ході реалізації проєкту було досягнуто наступного:

– проаналізовано існуючі рішення для управління музичними колекціями та виявлення їхніх недоліків;

– спроєктовано архітектуру системи та бази даних для зберігання пісень, плейлистів та метаданих;

– здійснено моделювання системи за допомогою UML-діаграм для деталізації логіки взаємодії компонентів та процесів;

– розроблено модуль автоматичної обробки аудіофайлів для вилучення інформації про виконавця, альбом та жанр;

– реалізувано інтерфейс користувача для відтворення музики та управління плейлистами;

– створено підсистему візуалізації статистики медіатеки;

– проведено комплексне тестування розробленої вебсистеми для перевірки коректності виконання всіх функцій та стабільності роботи під навантаженням.

Отриманий програмний продукт є готовим до розгортання як персональний медіасервер у локальних мережах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ID3 tag version 2.4.0 – Main Structure. ID3.org. URL: https://id3.org/id3v2.4.0-structure (Accessed: 20.04.2026).

2. Spotify Official Website. Spotify. URL: https://open.spotify.com/ (Accessed: 20.04.2026).

3. Foobar2000 Official Website. Foobar2000. URL: https://www.foobar2000.org/ (Accessed: 20.04.2026).

4. Plex Media Server. Plex. URL: https://www.plex.tv/ (Accessed: 20.04.2026).

5. SQLite official documentation. SQLite Consortium. URL: https://www.sqlite.org/docs.html (Accessed: 19.04.2026).

6. Толмач М. С., Попов А. В. Деякі аспекти використання сучасних технологій під час створення вебзастосунків. Інформаційні технології в культурі, мистецтві, освіті, науці, економіці та бізнесі. 2023. С. 78–83. URL: https://infotech-soccult.knukim.edu.ua/article/download/283958/278298 (дата звернення: 20.04.2026).

7. Django official documentation. Django Software Foundation. URL: https://docs.djangoproject.com/ (Accessed: 19.04.2026).

8. Swacha J., Kulpa A. Evolution of Popularity and Multiaspectual Comparison of Widely Used Web Development Frameworks. Electronics. 2023. Vol. 12, no. 17. doi: 10.3390/electronics12173563.

9. Zhang T., Hao F., Wang Y. et al. VeriFlow: A Framework for the Static Verification of Web Application Access Control via Policy-Graph Consistency. Electronics. 2024. Vol. 13, no. 18. doi: 10.3390/electronics14183742.

10. Mutagen official documentation. Read the Docs. URL: https://mutagen.readthedocs.io/en/latest/ (Accessed: 21.04.2026).

11. Singh J. pyAudioProcessing: Audio Processing, Feature Extraction, and Machine Learning Modeling. Proceedings of the 21st Python in Science Conference. 2022. URL: https://proceedings.scipy.org/articles/majora-212e5952-017 (Accessed: 23.04.2026).

12. React official documentation. Meta Platforms, Inc. URL: https://react.dev/ (Accessed: 20.04.2026).

13. Document Object Model (DOM). MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model (Accessed: 24.04.2026).

14. Здрок В. В. Розробка Single Page Application (односторінкового застосунку) з використанням React. Матеріали науково-технічної конференції ТНТУ. Тернопіль, 2018. С. 95. URL: https://elartu.tntu.edu.ua/handle/lib/27288 (дата звернення: 20.04.2026).

15. Повний огляд REST: нюанси, поради, приклади. DOU: Спільнота програмістів. URL: https://dou.ua/forums/topic/50364/ (дата звернення: 10.05.2026).

16. Антоненко А. В., Востріков С. О., Бурачинський А. Ю., Твердохліб А. О., Балвак А. А., Слободян О. А. Особливості автоматизованого тестування з використанням фреймворків. 2024. Таврійський науковий вісник. Серія: Технічні науки. № 4. С. 3-14. doi: https://doi.org/10.32782/tnv-tech.2024.4.1.

ДОДАТОК А

Лістинг коду компонента медіаплеєра PlayerBar.jsx

```
import React, { useState, useEffect, useRef } from 'react';
import { FaPlay, FaPause, FaStepForward, FaStepBackward, FaVolumeUp, FaVolumeMute, FaSpinner, FaRandom, FaExpandAlt } from 'react-icons/fa';
import { t } from '../translations';

const PlayerBar = ({ currentTrack, audioRef, isPlaying, setIsPlaying, onNextTrack, onPrevTrack, isShuffle, toggleShuffle, onOpenFullscreen, lang = 'ua' }) => {
  const [progress, setProgress] = useState(0);
  const [currentTime, setCurrentTime] = useState('0:00');
  const [duration, setDuration] = useState('0:00');
  const [blobUrl, setBlobUrl] = useState(null);
  const [isLoadingAudio, setIsLoadingAudio] = useState(false);
  const isDragging = useRef(false);

  const [volume, setVolume] = useState(1);
  const [isMuted, setIsMuted] = useState(false);
  const [prevVolume, setPrevVolume] = useState(1);

  useEffect(() => {
    const handleMuteKey = (e) => {
      if (['INPUT', 'TEXTAREA'].includes(e.target.tagName)) return;
      if (e.code === 'KeyM') toggleMute();
    };
    window.addEventListener('keydown', handleMuteKey);
    return () => window.removeEventListener('keydown', handleMuteKey);
  }, [isMuted, volume, prevVolume]);

  const formatTime = (time) => {
    if (isNaN(time) || time === Infinity) return '0:00';
    const mins = Math.floor(time / 60);
    const secs = Math.floor(time % 60);
    return `${mins}:${secs < 10 ? '0' : ''}${secs}`;
  };
```

```
        const getCoverSrc = () => {
          if (!currentTrack || !currentTrack.cover_image) return null;
          return currentTrack.cover_image.startsWith('http')
            ? currentTrack.cover_image
            : `http://127.0.0.1:8000${currentTrack.cover_image}`;
        };

        useEffect(() => {
          if (!currentTrack || !currentTrack.file_path) {
            setBlobUrl(null);
            return;
          }
          let objectUrl = null;
          setIsLoadingAudio(true);
          const url = currentTrack.file_path.startsWith('http') ?
currentTrack.file_path : `http://127.0.0.1:8000${currentTrack.file_path}`;

          fetch(url)
            .then(res => res.blob())
            .then(blob => {
              objectUrl = URL.createObjectURL(blob);
              setBlobUrl(objectUrl);
              setIsLoadingAudio(false);
            })
            .catch(err => { console.error(err); setIsLoadingAudio(false); });

          return () => { if (objectUrl) URL.revokeObjectURL(objectUrl); };
        }, [currentTrack]);

        useEffect(() => {
          if (audioRef.current && blobUrl) {
            if (isPlaying) {
              const playPromise = audioRef.current.play();
              if (playPromise !== undefined) playPromise.catch(() => { });
            } else { audioRef.current.pause(); }
          }
```

```
  }, [blobUrl, isPlaying, audioRef]);

  const togglePlay = () => {
    if (!currentTrack || isLoadingAudio) return;
    setIsPlaying(!isPlaying);
  };

  const handleTimeUpdate = () => {
    if (!audioRef.current || isDragging.current) return;
    const current = audioRef.current.currentTime;
    const total = audioRef.current.duration;
    if (total) {
      setCurrentTime(formatTime(current));
      setProgress((current / total) * 100);
    }
  };

  const handleLoadedMetadata = () => {
    if (!audioRef.current) return;
    setDuration(formatTime(audioRef.current.duration));
    audioRef.current.volume = isMuted ? 0 : volume;
  };

  const handleSeekChange = (e) => {
    isDragging.current = true;
    const seekValue = Number(e.target.value);
    setProgress(seekValue);
    if (audioRef.current && audioRef.current.duration) {
      setCurrentTime(formatTime((seekValue / 100) *
audioRef.current.duration));
    }
  };

  const handleSeekEnd = (e) => {
    const seekValue = Number(e.target.value);
    if (audioRef.current && audioRef.current.duration) {
```

```
        audioRef.current.currentTime = (seekValue / 100) * audioRef.current.duration;
      }
      setTimeout(() => { isDragging.current = false; }, 150);
    };

    const handleVolume = (e) => {
      const val = e.target.value / 100;
      setVolume(val);
      if (val > 0) setIsMuted(false);
      else setIsMuted(true);
      if (audioRef.current) audioRef.current.volume = val;
    };

    const toggleMute = () => {
      if (isMuted) {
        setIsMuted(false);
        setVolume(prevVolume > 0 ? prevVolume : 1);
        if (audioRef.current) audioRef.current.volume = prevVolume > 0 ? prevVolume : 1;
      } else {
        setPrevVolume(volume);
        setIsMuted(true);
        setVolume(0);
        if (audioRef.current) audioRef.current.volume = 0;
      }
    };

    const progressStyle = { background: `linear-gradient(to right, var(--primary-color) ${progress}%, var(--text-muted) ${progress}%)` };
    const volValue = isMuted ? 0 : volume * 100;
    const volumeStyle = { background: `linear-gradient(to right, var(--primary-color) ${volValue}%, var(--text-muted) ${volValue}%)` };

    return (
      <div className="player-bar">
```

```
            <audio ref={audioRef} src={blobUrl || ""}
onTimeUpdate={handleTimeUpdate} onLoadedMetadata={handleLoadedMetadata}
onEnded={() => onNextTrack ? onNextTrack() : setIsPlaying(false)} preload="auto" />

            <div className="track-info">
              <div     className="cover-placeholder">{getCoverSrc()     ?     <img
src={getCoverSrc()} alt="cover" /> : <div className="empty-cover">🎵</div>}</div>
              <div className="track-text">
                <h4>{currentTrack ? currentTrack.title : t[lang].no_track}</h4>
                <p>{currentTrack       ?       (currentTrack.artist_name       ||
t[lang].unknown_artist) : t[lang].choose_song}</p>
              </div>
            </div>

            <div className="player-center">
              <div className="player-controls">
                <button  className={`control-btn ${isShuffle ? 'active-control' :
''}`} onClick={toggleShuffle} title={t[lang].shuffle}><FaRandom /></button>
                <button       className="control-btn"       onClick={onPrevTrack}
disabled={!currentTrack}><FaStepBackward /></button>
                <button   onClick={togglePlay}   className="play-btn   main-play"
disabled={!currentTrack}>{isLoadingAudio  ?  <FaSpinner  className="spin-icon"/>  :
(isPlaying ? <FaPause /> : <FaPlay />)}</button>
                <button       className="control-btn"       onClick={onNextTrack}
disabled={!currentTrack}><FaStepForward /></button>
              </div>
              <div className="progress-container">
                <span className="time">{currentTime}</span>
                <input   type="range"   className="progress-bar"   value={progress}
onChange={handleSeekChange}  onMouseUp={handleSeekEnd}  onTouchEnd={handleSeekEnd}
step="0.1" disabled={isLoadingAudio || !currentTrack} />
                <span className="time">{duration}</span>
              </div>
            </div>

            <div className="volume-controls">
```

```
              <button      className="control-btn"      onClick={onOpenFullscreen}
title={t[lang].fullscreen} disabled={!currentTrack} <FaExpandAlt /> </button>
              <button         className="control-btn"         onClick={toggleMute}
title={t[lang].mute} >{isMuted || volume === 0 ? <FaVolumeMute color="#e74c3c" />
: <FaVolumeUp />}</button>
              <input    type="range"    className="volume-bar"    value={volValue}
onChange={handleVolume} />
            </div>
          </div>
        );
      };

      export default PlayerBar;
```

ДОДАТОК Б

Лістинг коду тестування tests.py

```
from rest_framework.test import APITestCase
from rest_framework import status
from django.contrib.auth.models import User
from django.core.files.uploadedfile import SimpleUploadedFile
from .models import Track, Playlist, Artist, UserProfile, ListeningHistory

class MusicAppTests(APITestCase):
    def setUp(self):
        self.user1 = User.objects.create_user(username='user1', password='password123')
        self.user2 = User.objects.create_user(username='user2', password='password123')

        self.admin_user = User.objects.create_superuser(username='admin', password='adminpassword', email='admin@test.com')

        self.artist = Artist.objects.create(name='Test Artist')
        fake_audio = SimpleUploadedFile("song.mp3", b"file_content", content_type="audio/mpeg")

        self.track = Track.objects.create(
            title='Test Song',
            artist=self.artist,
            file_path=fake_audio
        )

    # ========================================
    # Тест 1: Створення профілю
    # ========================================
    def test_user_profile_created_automatically(self):
        """Перевіряємо, чи створюється UserProfile автоматично завдяки @receiver"""
        new_user = User.objects.create_user(username='new_guy', password='123')
```

```
        profile_exists = UserProfile.objects.filter(user=new_user).exists()
        self.assertTrue(profile_exists)

    # ========================================
    # Тест 2: плейлисти та права доступу
    # ========================================
    def test_playlist_permissions_and_import(self):
        self.client.force_authenticate(user=self.user1)
        playlist = Playlist.objects.create(name='Rock Hits', user=self.user1)
        playlist.tracks.add(self.track)

        self.client.force_authenticate(user=self.user2)

        delete_response = self.client.delete(f'/api/playlists/{playlist.id}/')
        self.assertEqual(delete_response.status_code, status.HTTP_403_FORBIDDEN)

        update_response = self.client.patch(f'/api/playlists/{playlist.id}/', {'name': 'Hacked Name'})
        self.assertEqual(update_response.status_code, status.HTTP_403_FORBIDDEN)

        import_response = self.client.post(f'/api/playlists/{playlist.id}/import/')
        self.assertEqual(import_response.status_code, status.HTTP_201_CREATED)

        cloned_playlist = Playlist.objects.filter(user=self.user2).first()
        self.assertIsNotNone(cloned_playlist)
        self.assertIn('(Копія)', cloned_playlist.name)
        self.assertEqual(cloned_playlist.tracks.count(), 1)

    # ========================================
    # Тест 3: вподобання
    # ========================================
    def test_toggle_like(self):
```

```
            self.client.force_authenticate(user=self.user1)

            response1 = self.client.post(f'/api/tracks/{self.track.id}/like/')
            self.assertEqual(response1.status_code, status.HTTP_200_OK)
            self.assertTrue(self.track.likes.filter(id=self.user1.id).exists())

            response2 = self.client.post(f'/api/tracks/{self.track.id}/like/')
            self.assertEqual(response2.status_code, status.HTTP_200_OK)
            self.assertFalse(self.track.likes.filter(id=self.user1.id).exists())

        # ========================================
        # Тест 4: історія прослуховувань
        # ========================================
        def test_history_logic(self):
            self.client.force_authenticate(user=self.user1)

            post_response   =   self.client.post('/api/history/',   {'track_id':
self.track.id})
            self.assertEqual(post_response.status_code, status.HTTP_201_CREATED)

self.assertEqual(ListeningHistory.objects.filter(user=self.user1).count(), 1)

            delete_response = self.client.delete('/api/history/')
            self.assertEqual(delete_response.status_code,
status.HTTP_204_NO_CONTENT)

self.assertEqual(ListeningHistory.objects.filter(user=self.user1).count(), 0)

        # ========================================
        # Тест 5: адмін-панель
        # ========================================
        def test_admin_permissions(self):
            self.client.force_authenticate(user=self.user1)
            response_user = self.client.get('/api/users/')
            self.assertEqual(response_user.status_code,
status.HTTP_403_FORBIDDEN)
```

```
self.client.force_authenticate(user=self.admin_user)
response_admin = self.client.get('/api/users/')
self.assertEqual(response_admin.status_code, status.HTTP_200_OK)
self.assertEqual(len(response_admin.data), 3) # user1, user2, admin
```