

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

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:

**«Інтеграція аналітичного функціоналу до сервісу
музичного контенту»**

Виконав студент групи ЗПЗм-22-1
ПАВЛІШИН Марк Сергійович

Керівник роботи:
ЄФРЕМОВ Юрій Миколайович

Рецензент:
ВЛАСЕНКО Олег Васильович

Житомир – 2023

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

«ЗАТВЕРДЖУЮ»

зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«18» вересня 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: ПАВЛІШИН Марк Сергійович

Керівник роботи: ЄФРЕМОВ Юрій Миколайович

Тема роботи: **«Інтеграція аналітичного функціоналу до сервісу музичного контенту»**, затверджена Наказом закладу вищої освіти від **«18» вересня 2023 р. № 481/с**

Вихідні дані для роботи: **інтеграція аналітичного функціоналу до сервісу музичного контенту.**

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Єфремов Ю.М.	22.09.2023р.	22.09.2023р.
2	Єфремов Ю.М.	22.09.2023р.	22.09.2023р.
3	Єфремов Ю.М.	22.09.2023р.	22.09.2023р.

РЕФЕРАТ

Кваліфікаційна робота орієнтується на розвиток інтеграції аналітичного функціоналу до сервісу музичного контенту з акцентом на статистику прослуховування. Робота включає в себе як сервіс, так і розгорнуту пояснювальну записку, в якій обрано статистику прослуховування як головний предмет дослідження та інновації. Пояснювальна записка складається з 56 сторінок, містить 34 ілюстрації, 1 таблицю, 1 додаток та 22 літературних джерела.

Метою даної роботи є створення інтеграція аналітичного функціоналу до сервісу музичного контенту, який надає користувачам можливість збереження і аналізу даних щодо їхнього прослуховування музичного контенту. Сервіс орієнтований на ринки України та сусідніх країн і спроектований для надання користувачам засобів для відображення та аналізу статистики своїх прослуховувань.

Досліджено існуючі аналоги та конкурентні платформи, з фокусом на їхніх можливостях збору та відображення статистики прослуховування. На основі цього аналізу було сформульовано технічне завдання та обрано технології для реалізації проекту, включаючи React, TypeScript для фронтенду, ASP.NET, C# для бекенду та SQLServer для бази даних.

Завершенням даної магістерської роботи стала інтеграція аналітичного функціоналу до сервісу музичного контенту, яка зосереджена на забезпеченні користувачів інструментами для відображення та аналізу статистики їхнього музичного прослуховування. Розроблений сервіс є конкурентоспроможним на ринку музичних стрімінгових платформ і відзначається легкістю користування, інтуїтивним інтерфейсом та широкими можливостями для взаємодії з контентом. **КЛЮЧОВІ СЛОВА:** СЕРВІС, СТАТИСТИКА, МУЗИКА, ДОДАТОК

ABSTRACT

This qualification work focuses on the development of integrating analytical functionality into a music content service, with an emphasis on listening statistics. The work includes both a service and an extensive explanatory note, which selects listening statistics as the main subject of research and innovation. The explanatory note comprises 56 pages, contains 34 illustrations, 1 table, 1 appendix, and 22 literary sources.

The aim of this work is to create an integration of analytical functionality into a music content service, which provides users with the ability to save and analyze data about their music listening habits. The service is targeted at the markets of Ukraine and neighboring countries, and is designed to provide users with tools for displaying and analyzing their listening statistics.

Existing analogs and competitive platforms have been researched, with a focus on their capabilities for collecting and displaying listening statistics. Based on this analysis, a technical task was formulated, and technologies were chosen for the project implementation, including React and TypeScript for the frontend, ASP.NET, C# for the backend, and SQLServer for the database.

The completion of this master's thesis was the integration of analytical functionality into the music content service, focused on providing users with tools for displaying and analyzing their music listening statistics. The developed service is competitive in the market of music streaming platforms and is characterized by ease of use, an intuitive interface, and extensive possibilities for interacting with content. **KEYWORDS:** SERVICE, STATISTICS, MUSIC, APPLICATION.

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	5
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	7
ВСТУП	8
РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ ЗАСОБІВ І ТЕХНОЛОГІЙ ВИКОРИСТАННЯ СЕРВІСУ МУЗИЧНОГО КОНТЕНТУ	10
1.1. Постановка задачі	10
1.2. Аналіз аналогів програмного продукту.....	10
1.3. Планування етапів проектування системи.....	18
1.4.Планування алгоритму збору та аналізу даних.....	20
Висновки до розділу 1.....	21
РОЗДІЛ 2. ПРОЕКТУВАННЯ ІНТЕГРАЦІЇ АНАЛІТИЧНОГО ФУНКЦІОНАЛУ ДО СЕРВІСУ МУЗИЧНОГО КОНТЕНТУ.....	22
2.1. ВИБІР АРХІТЕКТУРИ СЕРВІСУ.....	22
2.2. ОБГРУНТУВАННЯ ВИБОРУ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ТА ВИМОГИ ДО АПАРАТНОГО ЗАБЕЗПЕЧЕННЯ.....	24
2.3. Визначення варіантів використання та взаємодії з сервісом музичного контенту та аналітичним функціоналом.....	30
2.4. РЕАЛІЗАЦІЯ ФУНКЦІОНАЛУ ІНТЕГРАЦІЇ АНАЛІТИЧНОГО ФУНКЦІОНАЛУ ДО СЕРВІСУ МУЗИЧНОГО КОНТЕНТУ	33
Висновки до розділу 2.....	41
РОЗДІЛ 3. ПОРЯДОК ВЗАЄМОДІЇ З МОДУЛЕМ АНАЛІТИЧНОГО ФУНКЦІОНАЛУ	42
3.1. Порядок взаємодії з інтерфейсом модуля аналітичного функціоналу..	42
3.2. Порядок взаємодії з додатковими елементами системи.	45
Висновки до розділу 3.....	53

ВИСНОВКИ.....	54
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	55
ДОДАТКИ.....	57

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

React	- відкрита JavaScript бібліотека для створення інтерфейсів користувача
SCSS	- Syntactically Cascading Style Sheets (з <i>англ.</i> – Синтаксично каскадні таблиці стилів)
ASP.NET Core	- технологія створення вебзастосунків і вебсервісів від компанії Майкрософт.
API	- Application Programming Interface (з <i>англ.</i> – прикладний програмний інтерфейс) контракт серверного застосунку з інтерфейсом користувача

ВСТУП

Актуальність обраної теми. Життя в 21-му сторіччі неможливо уявити без музики: вона є супутником на дорозі до роботи, фоновим звуком в кафе або торгівельних центрах інструментом для релаксації чи навіть засобом самовираження. За останні десятиліття технологічний прогрес суттєво змінив способи доступу до музичного контенту. Якщо раніше переважною формою були фізичні носії — касети, CD, вінілові платівки, то тепер більша частина аудиторії переходить на стрімінгові платформи.

Український ринок музичних сервісів, як і ринки сусідніх країн, проходить період активних змін. З'являється все більше локальних та іноземних платформ, що намагаються зайняти свою нішу. Однак, не всі з них можуть похвалитися зручним інтерфейсом, широким вибором музики та можливістю ефективної взаємодії між користувачами.

Все це робить актуальним інтеграцію аналітичного функціоналу до сервісу музичного контенту, функціонал буде враховувати специфіку цільового ринку та надавати користувачам широкі можливості для взаємодії з контентом.

Основна мета роботи: створення комплексного рішення для аналізу статистичних показників програвання музики та веб сервісу, яке буде конкурентоспроможним на ринку стрімінгових сервісів музики.

Основні завдання, які обумовлює мета роботи:

- Проведення аналізу існуючих аналогів та конкурентних платформ.
- Розробка технічного завдання для програмного рішення.
- Вибір технологій для фронтендної, бекендної частини та бази даних.
- Підбір алгоритмів для збору статистики.
- Тестування розробленого сервісу.

Об'єктом дослідження є взаємодія користувача статистичними даними сервісу музичного контенту, який реалізується за допомогою сучасних технологій, таких як React, TypeScript, ASP.NET Core, C# та SQLServer.

У роботі буде використано методи аналізу та синтезу, тестування програмного забезпечення, а також методи об'єктно-орієнтованого програмування.

Таким чином, дана магістерська робота має на меті створити комплексне рішення, яке зможе виконувати збір статистики та задовольнити базові потреби широкого кола користувачів, а також стати конкурентоспроможним на ринку.

Апробація дослідження – участь у конференції “Комп’ютерні технології: інновації, проблеми, рішення” (м. Житомир), 29-30 листопада 2023 р. із тезами “АНАЛІЗ ТЕНДЕНЦІЙ РОЗВИТКУ РИНКУ МУЗИЧНИХ СТІМІНГОВИХ СЕРВІСІВ” [1].

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ ЗАСОБІВ І ТЕХНОЛОГІЙ ВИКОРИСТАННЯ СЕРВІСУ МУЗИЧНОГО КОНТЕНТУ

1.1. Постановка задачі

Головною метою цієї кваліфікаційної роботи магістра є інтеграція аналітичного функціоналу щодо сервісу музичного контенту. Проект передбачає створення платформи, яка не лише надасть користувачам доступ до різноманітних музичних треків та альбомів, але й забезпечить їх засобами для глибокого аналізу їхніх музичних переваг. Важливим елементом буде візуалізація даних про популярність композицій, частоту прослуховувань та вибір користувачів, що дозволить їм краще зрозуміти свої музичні вподобання.

У планах проекту також входить реалізація персоналізованих функцій, таких як рекомендаційні системи для адаптації до індивідуальних музичних вподобань користувачів. Це сприятиме не тільки зростанню задоволення користувачів від сервісу, але й допоможе відкрити нові музичні напрямки та артистів.

Нарешті, особливий акцент планується зробити на розробці інтуїтивно-зрозумілого та легкого у використанні інтерфейсу. Це забезпечить комфортне та просте користування сервісом для людей різного віку та з різним рівнем технічної обізнаності, що є ключовим для успішного залучення широкої аудиторії користувачів.

1.2. Аналіз аналогів програмного продукту

Сфера музичного контенту в інтернеті стає все більш насиченою і конкурентною. Це незаперечний факт, який вимагає від розробників програмного забезпечення глибокого і всеосяжного розуміння потреб користувачів, а також актуальних тенденцій в галузі розробки. Існує безліч платформ для прослуховування музики, і кожна має свою унікальну пропозицію, що охоплює різні жанри, особливості та формати контенту. З урахуванням такої великої конкуренції, ключем до успіху є здатність

розробити продукт, який не просто відповідає основним потребам користувачів, але і пропонує щось унікальне, що відрізняє його від інших, у даній роботі як нововведення було обрано функціонал відображення статистики, що не надають конкурентні додатки.

По-перше, серед відомих музичних сервісів є **YouTube Music** [6], який є у власності американської транснаціональної корпорації Google (Рис. 1.1).

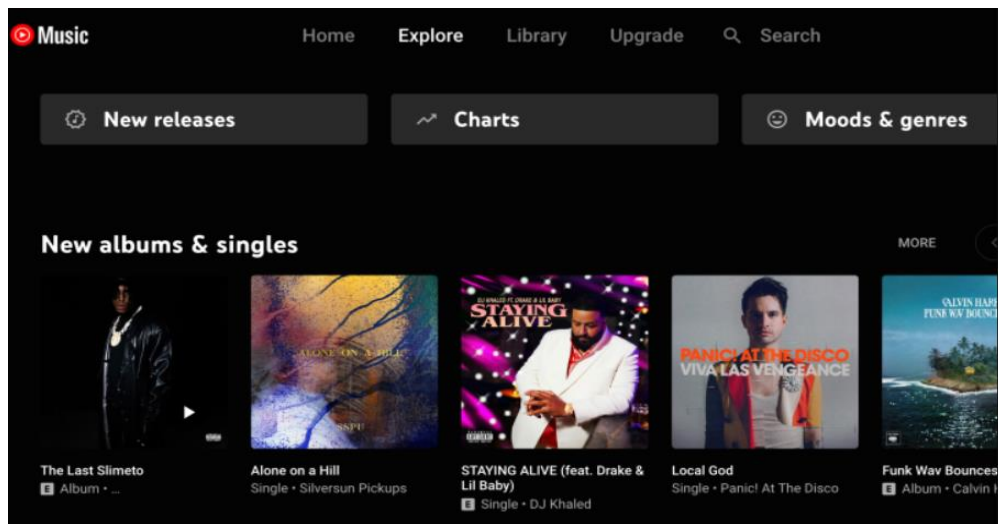


Рисунок 1.1 – Застосунок ресурсу «YouTube Music».

Однією з найбільших переваг є тісна інтеграція з YouTube. По-перше це дає доступ до величезної кількості контенту, який може бути недоступний на інших платформах, по-друге, крім композицій, користувачі можуть переглядати відео, що робить досвід більш мультимедійним (Рис 1.2).



Рисунок 1.2 – перегляд відео з музикою.

YouTube Music здійснює глибоку персоналізацію вибору музики на основі історії перегляду на YouTube, що дозволяє пропонувати рекомендації з більшою якістю, також серед переваг слід відмітити те що підписка на основну версію сервісу YouTube також видаляє рекламу з музичного застосунку YouTube Music.

Серед недоліків потрібно згадати, що у безкоштовній версії YouTube Music присутня нав'язлива реклама, яка може зіпсувати користувацький досвід. Якість звуку, хоча і добра, може бути не на рівні з конкурентами, що спеціалізуються лише на аудіо-контенті. Інтерфейс може виявитися не інтуїтивним для нових користувачів, і, оскільки платформа первинно орієнтована на відео, вона може не надавати оптимального досвіду для тих, хто хоче зосередитися на прослуховуванні цілих альбомів чи дискографій (Рис. 1.3).

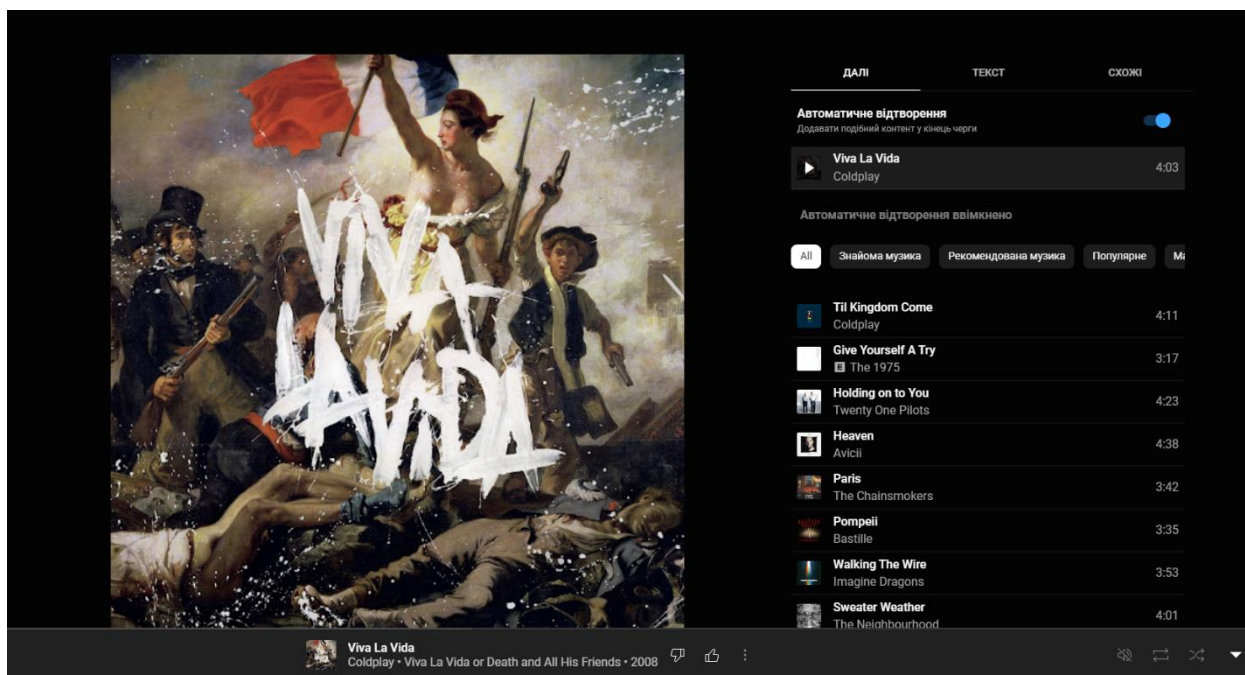


Рисунок 1.3 – Плеєр орієнтований на програвання відео контенту.

Сервіс не пропонує функціоналу для перегляду статистики, хоча слід відмітити що він пропонує плейлист найбільш прослуховуваної музики щорічно.

Наступною компанією, яка має вагому частину користувачів на ринку світових музичних сервісів є шведська компанія **Spotify** [18](Рис. 1.4).

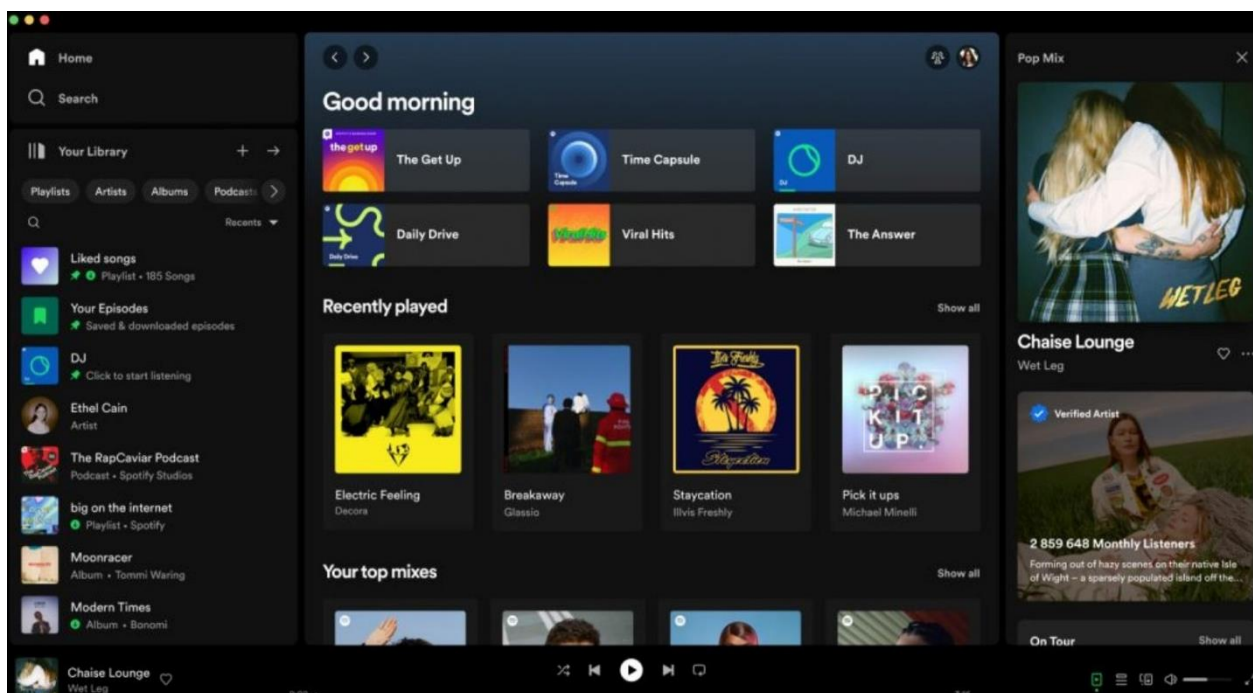


Рисунок 1.4 – Застосунок компанії «Spotify».

Spotify є одним з найпопулярніших музичних стрімінгових сервісів в світі, і не без причини. По-перше, платформа вражає своєю екстенсивною музичною бібліотекою, яка містить мільйони треків від відомих виконавців до незалежних артистів. Система рекомендацій Spotify, яка базується на штучному інтелекті, є однією з найкращих на ринку, що дозволяє обирати композиції відповідно користувацьким вподобанням.

Крім того слід відмітити функцію плавної та швидкої зміни між дивайсами, що дає змогу безперервно слухати музику всюди (Рис 1.5).

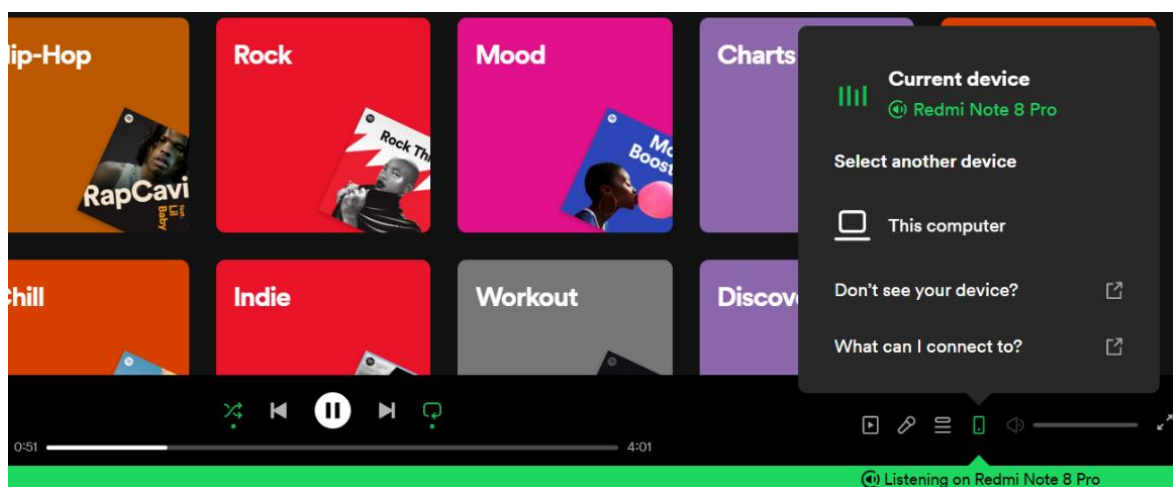


Рисунок 1.5 – Швидкий вибір дивайсу для відтворення.

Ще однією важливою перевагою є гнучкість платіжних планів. Від безкоштовної версії до різних платних планів, Spotify дає можливість користувачам вибрати опцію, яка найкраще відповідає їхнім потребам та бюджету.

У Spotify, не зважаючи на його багатий функціонал, існують певні недоліки. До них можна віднести обмежений доступ до деяких музичних треків і альбомів у залежності від регіону користувача. Також слід відмітити, що безкоштовна версія сервісу майже повністю обмежує користувацький досвід надаючи можливість слухати лише випадкові композиції в плейлистах або альбомах (Рис 1.6).

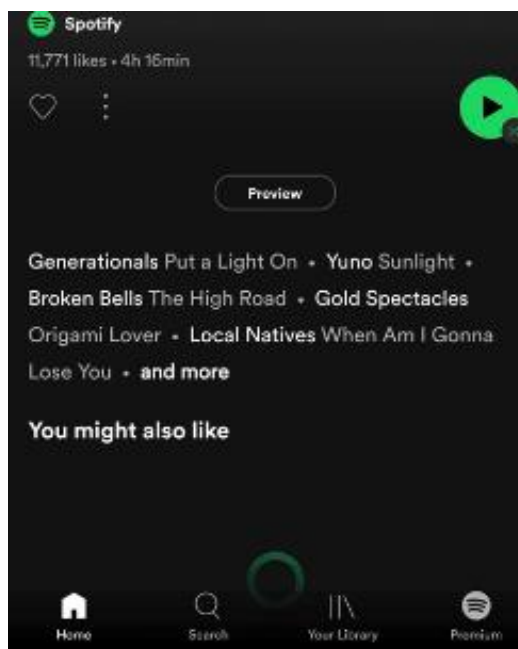


Рисунок 1.6 – Обмежене прослуховування в безкоштовній версії.

Крім того, інтерфейс, хоча і здебільшого зручний, може виявитися перевантаженим для новачків, які тільки ознайомлюються з системою, особливо це стосується навігаційної панелі (Рис 1.7).

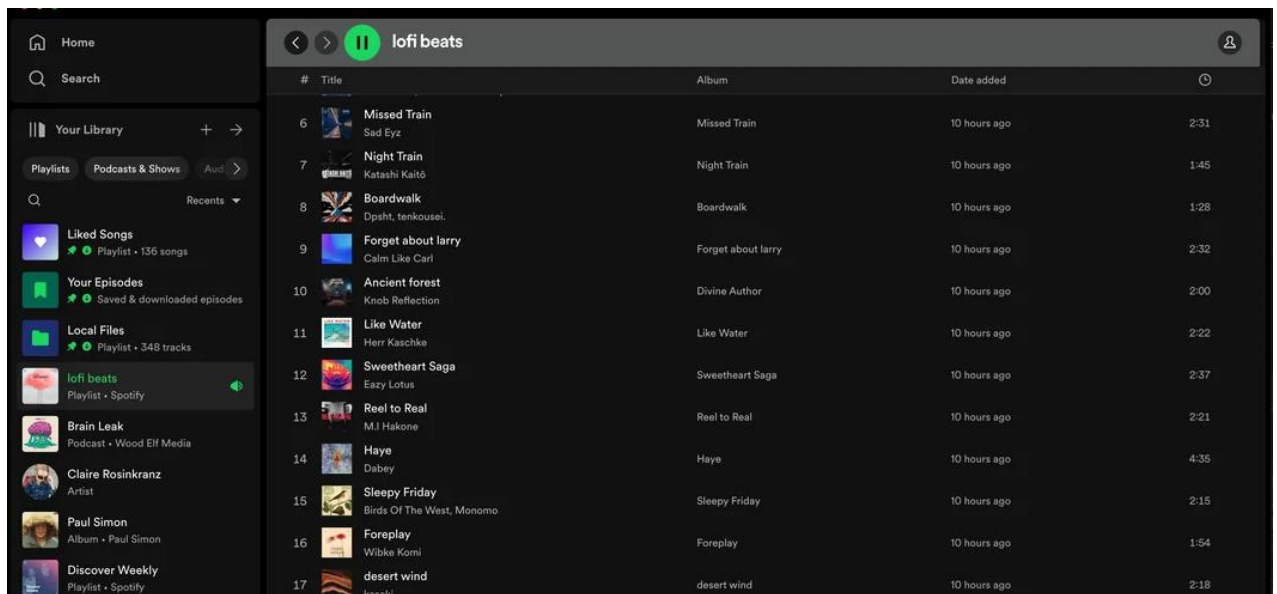


Рисунок 1.7 – Навантажений інтерфейс застосунку.

Ще одним недоліком є відсутність інтегрованого перегляду відеокліпів, що є у вищенаведеного конкурента є можливим.

Як і минулий сервіс, Spotify в якості статистики пропонує плейлист з найбільш прослуховуваними треками щорічно “Spotify Wrapped”, слід відмітити, що серед фанаців цієї платформи він користується неабиякою популярністю.

Наступним з гігантів цього ринку є **Apple Music** [9] — сервіс, який не просто пропонує доступ до великої музичної бібліотеки, а створює справді цілісну аудіо-екосистему, яка інтегрується з рядом інших продуктів американської транснаціональної корпорації Apple, що надає доступ до прослуховування музики всюди, а також плавну і швидку зміну дивайсів прослуховування в режимі реального часу. Нижче наведено головний екран застосунку (Рис. 1.8).

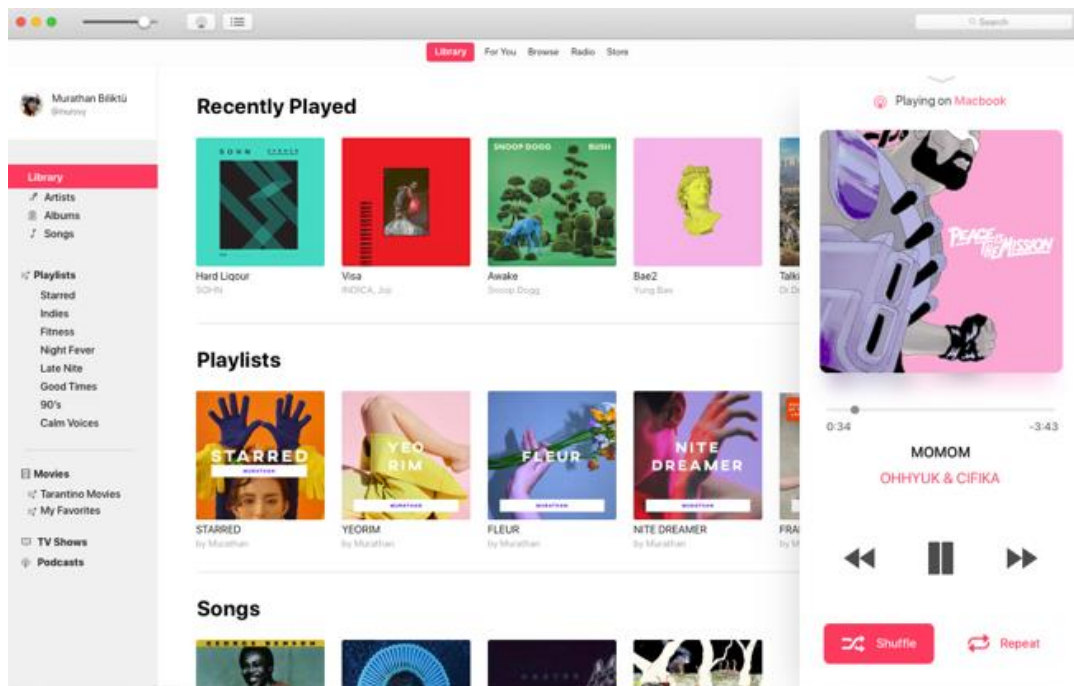


Рисунок 1.8 – Застосунок «Apple Music».

Apple Music має дуже велику музичну бібліотеку, яка містить понад 75 мільйонів композицій. Сервіс пропонує велику кількість жанрів, а також ексклюзивний контент, такий як інтерв'ю з артистами, ексклюзивні релізи, що дозволяють платформі мати унікальний та недоступний на жодній іншій платформі музичний контент. Особливої уваги заслуговує інтеграція з Apple ecosystem, що забезпечує гладкий користувацький досвід через різні пристрої, включаючи iPhone, iPad, Mac і Apple Watch. Сервіс пропонує низку персональних плейлистів, базуючись на вподобаннях користувача, що забезпечує більш індивідуалізований досвід. Також Apple Music пропонує окремий додаток для прослуховування класичної музики (Рис 1.9).

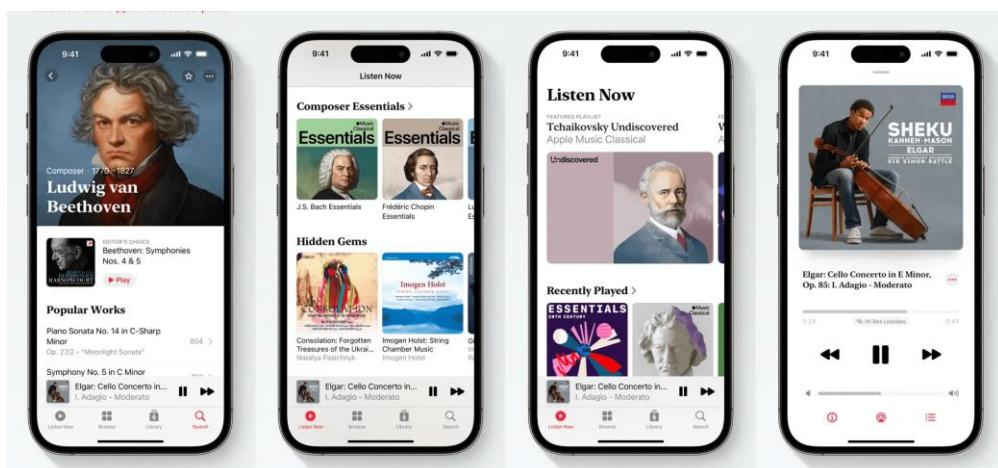


Рисунок 1.9 – Застосунок класичної музики

Однак, Apple Music не є без своїх недоліків. Сервіс пропонує обмежений функціонал у безкоштовній версії, що робить його менш доступним для широкого кола користувачів. Хоча музична бібліотека є великою, деякі виконавці або треки все ж можуть бути відсутніми, особливо якщо вони належать до незалежних лейблів або є менш популярними. Крім того, порівняно з іншими сервісами, Apple Music може виявитися дорожчим для користувачів, які не використовують інші продукти Apple, враховуючи те що доступ до даного сервісу можна отримати тільки маючи один з дивайсів компанії Apple.

Також Apple Music пропонує дуже обмежені можливості для персоналізації інтерфейсу. Це може викликати незручності для користувачів, які хочуть адаптувати додаток під свої індивідуальні потреби, наприклад відсутня адаптивність сітки відображення іконок списком або плитками на сторінках плейлистів, артистів або альбомів (Рис. 1.10).

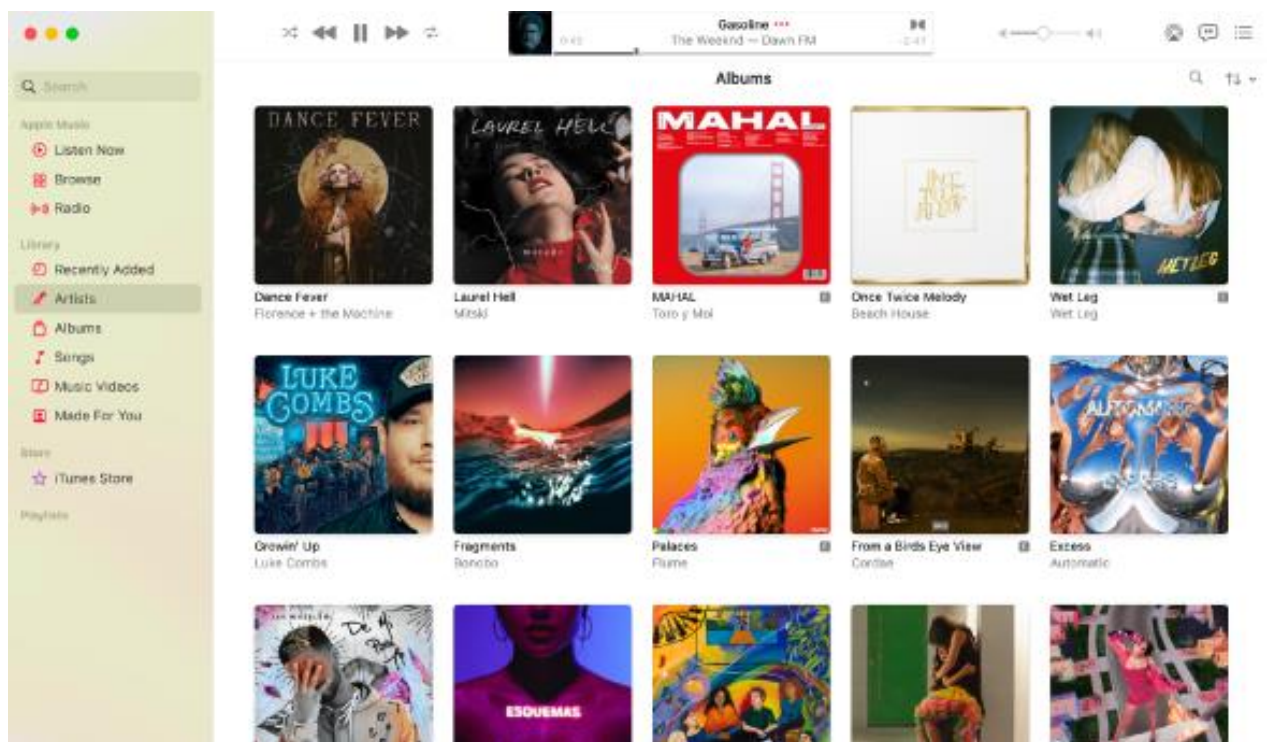


Рисунок 1.10 – Сторінка альбомів у застосунку.

Навідміну від попередніх сервісів, AppleMusic не пропонує функціоналу статистики ні в якому форматі.

У ході аналізу конкурентів було порівняно 3 кандидати, кожен з яких пропонує досить вдале та комплексне рішення. З урахуванням проаналізованої інформації, розроблюване рішення має шанс зайняти свою нішу на ринку. Для цього можна запропонувати імплементований функціонал детальнішого збору та візуалізації статистики прослуховування ніж у конкурентних платформ.

Таблиця 1.1.

Аналіз веб-додатків лідерів ринку

Компанія	Оформлення інтерфейсу	Швидка зміна дивайсів прослуховування	Візуалізація статистики прослуховувань	Обмеженість безкоштовної версії
YouTube Music	Зручний інтерфейс, який адаптований здебільшого для відео	Немає перемикання між дивайсами	Відсутня	Доступне прослуховування будь-якої музики з рекламою
Spotify	Сучасний та інтуїтивний інтерфейс	Зручне та швидке перемикання між дивайсами	Відсутня	Неможливе прослуховування музики за вибором користувача а лише у випадковому порядку з рекламними вставками
Apple Music	Зручний та простий дизайн	Наявне лише перемикання всередині дивайсів екосистеми Apple	Відсутня	Неможливе прослуховування будь-якої музики і перемикання пісень

1.3. Планування етапів проектування системи

Сервіс має надавати користувачам можливість прослуховування треків, гнучкого створення плейлистів, рекомендаційної системи, аналіз статистики прослуховувань, детальний пошук.

Основні етапи у виконанні завдання:

- а) Створення макетів сайту
 - Створення плану дій.
 - Використання дизайнерської програми Figma для створення макетів.
- б) Вибір найкращого макету з представлених.

- Сторінка авторизації та реєстрації.
- Головна сторінка з рекомендованими треками.
- Сторінка пошуку музичних композицій.
- Каталог альбомів, жанрів, і виконавців.
- Сторінка користувача з його плейлистами та плеєром.

в) Розробка програмного продукту

- Реалізація сторінки авторизації та реєстрації з використанням React, Typescript, SCSS.

г) Розробка головної сторінки.

- Розділення макета на блоки.
- Адаптивна верстка з використанням компонентного підходу фреймворку React та стилізація за допомогою SCSS.

г) Розробка каталогу музичного контенту.

- Створення анімацій для динамічної взаємодії з контентом.
- Створення блоку швидкого доступу з найбільш прослуховуваними плейлистами.
- Розробка блоків для відображення рекомендованих та нещодавно прослуханих плейлистів, артистів, альбомів.

д) Функціонал програвача

- Реалізація вбудованого аудіо-програвача.
- Інтеграція з API для підвантаження інформації про музику.
- Керування типом програвання набору музичних композицій.

е) Додаткові функції

- Реалізація рекомендаційної системи.
- Створення функціоналу для пошуку музики та зберігання у користувацькій бібліотеці.
- Реалізація можливості швидкої та інтуїтивної навігації сервісом.
- Реалізація функціоналу збору статистики по відтворюванню музичних композицій.

В кінці планується отримати інтеграцію аналітичного функціоналу до сервісу музичного контенту, яка відповідатиме сучасним стандартам, користувацьким вимогам та пропонуватиме функціонал, який відсутній у конкурентів.

1.4. Планування алгоритму збору та аналізу даних

Для ефективного збору та аналізу даних про прослуховування користувачів, необхідно ретельно розробити формат моделей даних, які будуть використовуватися на наступних етапах та представлені у інтерфейсі системи. Моделі повинні бути створені таким чином, щоб вони включали в себе всю необхідну інформацію для подальшого аналізу. Вони повинні містити інформацію про користувача, який здійснив прослуховування, дату прослуховування композиції і посилання на конкретну пісню, яку було прослухано.

Після успішного збору необхідних даних, необхідно визначити оптимальні алгоритми для обробки та візуалізації накопиченої статистичної інформації. Наприклад, можливо реалізувати візуалізацію даних за певними періодами часу, щоб надати користувачам можливість спостерігати за тим, скільки пісень було прослухано в конкретний період часу. Також важливо буде представити інформацію про найбільш популярних артистів і пісні з докладним розглядом кількості прослуховувань для кожного з них.

Також надзвичайно важливим елементом є самі методи візуалізації статистики. У цьому випадку, ми можемо використовувати графіки, таблиці та діаграми, які повинні бути створені у привабливому дизайні, спрямованому на комфорт та зацікавлення користувачів. Візуалізація даних у вигляді графіків та діаграм допоможе зрозуміти тенденції та патерни в прослуховуванні музики, а таблиці дозволять легко порівнювати різні аспекти даних.

Висновки до розділу 1.

За результатами аналізу існуючих аналогів програмного продукту у сфері музичного контенту було вивчено ключові особливості, переваги та недоліки популярних сервісів. Це дозволило скласти порівняльну таблицю та визначити найефективніші підходи до реалізації поставленої задачі. Після цього було розроблено план реалізації системи та визначено алгоритм для збору та аналізу даних.

По-перше, рішення не повинне повторювати помилки конкурентів, виявлені під час аналізу.

По-друге, інтерфейс сервісу має бути не лише функціональним, але і естетично привабливим, щоб забезпечити комфортне та приємне користування.

Третє це те, що в якості переважаючого функціоналу можна обрати збір статистики, який відсутній у платформ, які зараз найбільш популярні.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ІНТЕГРАЦІЇ АНАЛІТИЧНОГО ФУНКЦІОНАЛУ ДО СЕРВІСУ МУЗИЧНОГО КОНТЕНТУ

2.1 Вибір архітектури сервісу

Для успішної інтеграції аналітичного функціоналу до сервісу музичного контенту важливо вибрати правильну архітектуру, яка забезпечить швидкість, надійність та масштабованість. Для побудови архітектурного рішення було обрано наступні патерни і підходи проектування:

- DDD
- CQRS
- Mediator
- Web API
- Repositories

Domain-Driven Design (DDD) (з англ. предметно-орієнтоване проектування) є методологією розробки програмного забезпечення, яка зосереджується на створенні яскраво вираженої доменної моделі. Ця модель є представленням ключових бізнес-процесів, правил та функціональності системи. У DDD особливу увагу приділяють створенню мови, яка надає можливість ефективного спілкування між розробниками і замовниками, допомагаючи формувати складні бізнес-завдання у вигляді коду [10]. Основні елементи предметно-орієнтованого проектування:

- Сутності (Entities): Описують основні елементи домену з унікальними ідентифікаторами.
- Вартісні Об'єкти (Value Objects): Описують характеристики або атрибути об'єктів, але не мають унікального ідентифікатора.
- Агрегати (Aggregates): Група сутностей та вартісних об'єктів, які можуть бути розглянуті як цільна одиниця.
- Доменні Події (Domain Events): Відображають зміни у доменній моделі.

- Сервіси (Services): Описують дії, які не можна природньо підібрати до сутності чи вартісних об'єктів.
- Репозиторії (Repositories): Забезпечують доступ до агрегатів та сутностей.
- Фабрики (Factories): Відповідають за створення складних об'єктів та агрегатів.

CQRS (Command Query Responsibility Segregation) (з англ. відокремлення відповідальності за команди та запити) — це архітектурний паттерн в розробці програмного забезпечення, який відділяє операції зчитування від операцій модифікації даних. Таке розділення дозволяє оптимізувати обидва види операцій незалежно одна від одної, забезпечуючи вищу продуктивність, масштабованість та безпеку даних [15].

В традиційних архітектурах, особливо в CRUD (Create, Read, Update, Delete) системах, зазвичай використовується одна база даних або представлення моделі для обробки і команд, і запитів. Це може призводити до ряду проблем, наприклад, до необхідності компромісів між оптимізацією запитів для читання та запису. CQRS ефективно вирішує цю проблему, розділяючи логіку читання та запису на дві різні частини системи. У такій архітектурі, моделі для читання можуть бути оптимізовані для швидких запитів, в то час як моделі для запису можуть бути сконцентровані на валідації бізнес-правил. Додатково, розділеність дозволяє легше масштабувати кожен частину системи незалежно від іншої. Спочатку може здатися, що CQRS ускладнює архітектуру, але на практиці цей підхід може призвести до більш чистого коду, спрощуючи розробку та підтримку. Це зокрема актуально для великих, складних систем, де різні частини системи мають різні вимоги до масштабованості та продуктивності.

Mediator (з англ. посередник), є паттерном проектування, який спрямований на зменшення зв'язаності між класами або компонентами системи. Ідея полягає в тому, щоб ізолювати взаємодії між різними частинами системи, переносячи їх на спеціалізований об'єкт — посередника. Тобто, замість того, щоб компоненти взаємодіяли один з одним напряму,

вони взаємодіють через посередника, який знає, як правильно спрямувати запит або команду [3].

Використання Mediator може бути особливо корисним у великих та складних системах, де безпосередня взаємодія між компонентами може призвести до великої зв'язаності та збільшити складність управління системою. В таких випадках Mediator допомагає створити більш модульну, легко розширювану та легко тестовану систему.

Web API (Web Application Programming Interface) (з англ. програмний інтерфейс веб-додатків) — це набір протоколів та інструментів для побудови веб-сервісів. Зазвичай, Web API використовується для обміну даними між різними веб-сервісами або для взаємодії між сервером та клієнтської стороною додатку. На відміну від веб-серверів, які служать для надання веб-сторінок для перегляду в браузері, Web API фокусується на тому, щоб надавати програмний інтерфейс для взаємодії з додатками [14].

Паттерн "Repository" (з англ. репозиторій) — це шар між додатком та базою даних, який забезпечує абстракцію для доступу до даних. У загальному випадку, репозиторій виконує роль моста, який дозволяє вашому коду взаємодіяти з базою даних на більш високому рівні абстракції.

Додаток взаємодіє з репозиторієм, як з колекцією об'єктів. Запити до бази даних, створення, оновлення або видалення записів перетворюються на операції над об'єктами цієї колекції. Це робить код менш залежним від конкретних технологій баз даних, що полегшує тестування та підтримку [8].

2.2 Обґрунтування вибору інструментальних засобів та вимог до апаратного забезпечення

Для виконання технічного завдання обрані такі технології та інструменти: JetBrains Rider, ASP.NET Core, C#, Visual Studio Code, Figma React і TypeScript, SCSS. Кожен з цих засобів має свої специфічні переваги, які обґрунтовують їх вибір. Розглянемо обрані програми, які були використані для створення кінцевого продукту:

JetBrains Rider — це інтегроване середовище розробки (IDE), яке пропонує широкий спектр можливостей для розробки програм на мовах C# і будь-якої іншої мови з сімейства мов у .NET. Це продукт відомої компанії JetBrains, яка спеціалізується на створенні високоякісних інструментів для програмістів. Rider базується на двох ключових технологіях: ReSharper і IntelliJ IDEA. ReSharper надає багатий набір функцій для розробки на C#, а IntelliJ IDEA забезпечує стійкість та зручність інтерфейсу, за які цінується ця платформа. Однією з найбільших переваг Rider є його швидкість і ефективність. IDE підтримує широкий спектр функцій для рефакторингу коду, автоматичної генерації коду, швидкого доступу до документації та інтеграції з системами контролю версій, такими як Git. Також Rider має підтримку різних видів проектів і фреймворків, включаючи .NET Core, Mono і Unity. Це робить його гнучким рішенням для розробників різних напрямків: від веб-розробки до розробки ігор або ж мобільних додатків. Ще однією важливою особливістю є підтримка кросплатформенності. Rider доступний для Windows, macOS та Linux, що дозволяє розробникам працювати в зручному для них середовищі. Інтеграція з іншими інструментами і сервісами також є сильною стороною Rider. Він може легко інтегруватися з базами даних, Docker, системами контролю завдань і багато іншим. Це забезпечує централізоване середовище для усіх аспектів розробки програмного забезпечення [12].

ASP.NET Core представляє собою перебудовану версію ASP.NET, яка була розроблена з нуля для підтримки сучасних веб-технологій і кросплатформенної розробки. Він є частиною ширшого екосистеми .NET Core і забезпечує підтримку мов програмування, таких як C# і F#. Однією з найбільших переваг ASP.NET Core є його кросплатформенність. Ви можете розробляти та розгортати ваші веб-додатки на різних операційних системах, включаючи Windows, macOS і Linux. Це може бути вельми корисно для команд, які використовують різні робочі станції або для розгортання в різних середовищах. ASP.NET Core також характеризується високою

продуктивністю, завдяки оптимізаціям, таким як in-memory кешування, асинхронне програмування та ефективна обробка запитів. Фреймворк підтримує різні підходи до розробки веб-додатків, включаючи MVC (Model-View-Controller), Web API для рестфул-сервісів, та Blazor для створення інтерактивних веб-додатків на C# замість JavaScript. З питань безпеки, ASP.NET Core надає засоби для аутентифікації, авторизації, захисту від CSRF атак, SQL ін'єкцій, та інших типів веб-загроз. Щодо розширюваності, фреймворк має модульну архітектуру, яка дозволяє легко додавати нові функціональні можливості через NuGet пакети або власні бібліотеки. Додатково, ASP.NET Core інтегрується з багатьма сучасними інструментами розробки, такими як Docker для контейнеризації, Kubernetes для оркестрації контейнерів, та CI/CD інструментами для автоматизації розгортання [2].

C# є мовою програмування, розробленою Microsoft як частина платформи .NET. Ця мова була представлена в 2000 році і з того часу постійно розвивається. C# є об'єктно-орієнтованою мовою, хоча вона також підтримує концепції з функціонального, компонентного та процедурного програмування. Однією з ключових особливостей C# є її сильна типізація, що допомагає у попередженні багатьох помилок на етапі компіляції. Мова має широкий набір стандартних бібліотек, які допомагають у виконанні різноманітних завдань, від роботи з файлами і регулярними виразами до створення веб-сервісів та графічних інтерфейсів. C# широко використовується для розробки веб-додатків (зокрема, з використанням ASP.NET), настільних додатків, мобільних додатків (через Xamarin), графічних ігор (за допомогою Unity), а також для розробки системного та вбудованого програмного забезпечення. Мова забезпечує підтримку різних парадигм програмування, включаючи асинхронне програмування, LINQ для роботи з колекціями даних, та атрибути та анотації для декларативного програмування. C# постійно оновлюється, і в нових версіях додаються сучасні функції, такі як кортежі, шаблони записів, та шаблони з виведенням типів, що робить його сучасною та потужною мовою програмування. Це

також одна з найбільш популярних мов програмування у світі, завдяки своїй гнучкості, виразності та сильній підтримці спільноти [2].

Visual Studio Code, часто скорочено як VS Code, є безкоштовним редактором коду, розробленим Microsoft. Він є одним з найпопулярніших редакторів коду завдяки своїй гнучкості, високій продуктивності та широкому набору функцій. Цей редактор підтримує множину мов програмування та технологій "з коробки" і може бути далі налаштований або розширений за допомогою плагінів. VS Code пропонує великий набір можливостей, включаючи підсвітку синтаксису, автозавершення коду з інтелектуальними підказками, вбудований термінал, підтримку систем контролю версій, зокрема Git, а також дебагінг [21].

React — це JavaScript-бібліотека для побудови користувацьких інтерфейсів, розроблена і підтримувана компанією Facebook. Одна з основних характеристик React — це використання компонентної архітектури, яка дозволяє розбивати інтерфейс на незалежні, повторно використовувані компоненти. Компоненти можуть містити власну логіку, стани і розмітку, і їх можна легко інтегрувати один в одного [4].

В React компоненти — це основні будівельні блоки вашого застосунку. Вони дозволяють вам створити повторно використовувані, самостійні частини інтерфейсу, що можна легко інтегрувати в інші частини застосунку. Компоненти можуть бути функціональними або класовими, але в сучасному React частіше використовують функціональні компоненти з хуками для управління станом [17].

Розглянемо приклад елементарного функціонального компонента в React, який виводить текст "Hello, World!" на екран:

```
import React from 'react';
```

```
// Створення функціонального компонента
const HelloWorld = () => {
  return (
    <div>
```

```
    Hello, World!
  </div>
);
};

export default HelloWorld;
```

Цей компонент можна потім використовувати в інших частинах вашого застосунку просто шляхом імпорту й вставки в JSX-розмітці. Наприклад:

```
import React from 'react';
import HelloWorld from './HelloWorld';

const App = () => {
  return (
    <div>
      <HelloWorld />
    </div>
  );
};

export default App;
```

React використовує віртуальний DOM (Document Object Model), який дає змогу оптимізувати рендеринг і покращує продуктивність. Завдяки цьому, React здатний швидко оновлювати візуальний інтерфейс, вносячи лише необхідні зміни в реальний DOM. React також дозволяє працювати з різними платформами та технологіями. Наприклад, з React Native можна створювати мобільні додатки, а за допомогою таких рішень, як Next.js або Gatsby, можна побудувати серверно-рендерені веб-додатки. React має активну спільноту та велику кількість доступних розширень та бібліотек, що значно спрощує розробку. Його API досить стабільний, але в той же час постійно розвивається, включаючи введення нових можливостей, таких як хуки, які забезпечують більш ефективне управління станами та життєвими циклами компонентів. React підтримує JSX (JavaScript XML), синтаксичний розширювач для JavaScript, який дозволяє вам писати HTML-подібні теги та

компоненти прямо у JavaScript-кодi. Це робить код більш читабельним та підтримуваним.

Figma — це хмарна платформа для дизайну інтерфейсів, що забезпечує зручні інструменти для створення, прототипування, та співпраці між дизайнерами, розробниками, і замовниками. Однією з ключових особливостей Figma є можливість реального часу співпраці: кілька користувачів може одночасно працювати над одним проектом, бачити зміни одне одного і коментувати їх в реальному часі. Це істотно прискорює процес розробки і дозволяє отримати швидкий зворотний зв'язок. Figma підтримує векторну графіку, що дозволяє створювати дизайни з високою деталізацією і масштабованістю. Також є багатий набір інструментів для створення UI/UX дизайну, включаючи різноманітні шаблони, компоненти, та бібліотеки. Крім того, Figma має інтеграцію з іншими популярними інструментами та платформами, такими як Slack, Zeplin і деякі системи управління версіями, що робить його ще більш зручним для командної роботи [7].

TypeScript — це надмножина JavaScript, яка додає статичні типи до цієї мови. Розроблена Microsoft, ця технологія забезпечує розширену підтримку типізації, інтерфейсів, загальних типів та інших засобів для написання надійного та масштабованого коду. TypeScript компілюється в чистий JavaScript, тому його можна використовувати в будь-якому браузері, на будь-якому хості, на будь-якій операційній системі. Однією з основних переваг TypeScript є статична типізація. Це дозволяє розробникам виявляти помилки на етапі написання коду, замість того, щоб виявляти їх під час виконання в JavaScript. Визначення типів покращують читабельність коду і спрощують процес рефакторингу, оскільки редактор коду може точно вказувати, де що потрібно змінити [5].

SCSS (Syntactically Cascading Awesome Stylesheets) — це препроцесор для CSS, який значно розширює можливості базового CSS, додаючи функції, такі як змінні, вкладені правила, міксіни, спадкування та інші. Змінні дозволяють вам використовувати одні й ті ж значення в різних частинах

вашого стилевого коду, що полегшує його підтримку. Вкладеність забезпечує більш інтуїтивний та читабельний спосіб організації CSS. Міксіни — це фактично функції стилів, які можна повторно використовувати по всьому коду. SCSS також підтримує математичні операції, умовні оператори та цикли, які дають вам можливість писати дуже гнучкий та динамічний код. І після всього цього, SCSS код компілюється в стандартний CSS код, який можна використовувати в будь-якому веб-проекті [13].

Вибір таких технологій створює потужну екосистему для розробки сучасних веб-застосунків. JetBrains Rider і Visual Studio Code надають високопродуктивне середовище для розробки, ASP.NET Core і C# забезпечують надійний бекенд, Figma допомагає спроектувати початковий дизайн системи, а React дозволяє створювати динамічні користувацькі інтерфейси, TypeScript додає строгу типізацію для збільшення надійності коду, а SCSS полегшує роботу з стилями. Разом ці технології формують збалансований стек, який є вдалим для розробки високоякісних, масштабованих рішень.

2.3. Визначення варіантів використання та взаємодії з сервісом музичного контенту та аналітичним функціоналом.

Сервіс музичного контенту надає користувачам можливість отримувати доступ до широкої бібліотеки музики для прослуховування та збір статистики. Додаток також повинен надавати можливість прослуховувати музику, створювати та кастомізувати плейлисти, отримувати релевантні рекомендації щодо нових артистів та пісень. Система має демонструвати усі переваги використання сервісу музичного контенту та переконувати користувачів у тому, що саме цей сервіс надає найкращий музичний досвід і відмінно підходить для їхніх потреб.

Користувачі можуть виконувати такі дії за допомогою сервісу:

- Зареєструватись в системі.
- Редагувати та налаштовувати власний профіль.

- Створювати, кастомізувати плейлисти та створювати копії плейлистів.
- Виконувати пошук музики за жанрами або використовуючи пошукову стрічку.
- Додавати артистів, альбоми, плейлисти до бібліотеки.
- Переглядати сторінки та дискографію артистів.
- Прослуховувати музику використовуючи плеєр та змінювану чергу пісень.
- Переглядати щомісячну статистику прослуховувань.
- Лайкати пісні додаючи її у стандартний плейлист улюблених пісень.

Можливості використання системи більш детально візуалізовані на Use case[11] діаграмі (Рис. 2.1) та на FDD [20] діаграмі (Рис. 2.2).

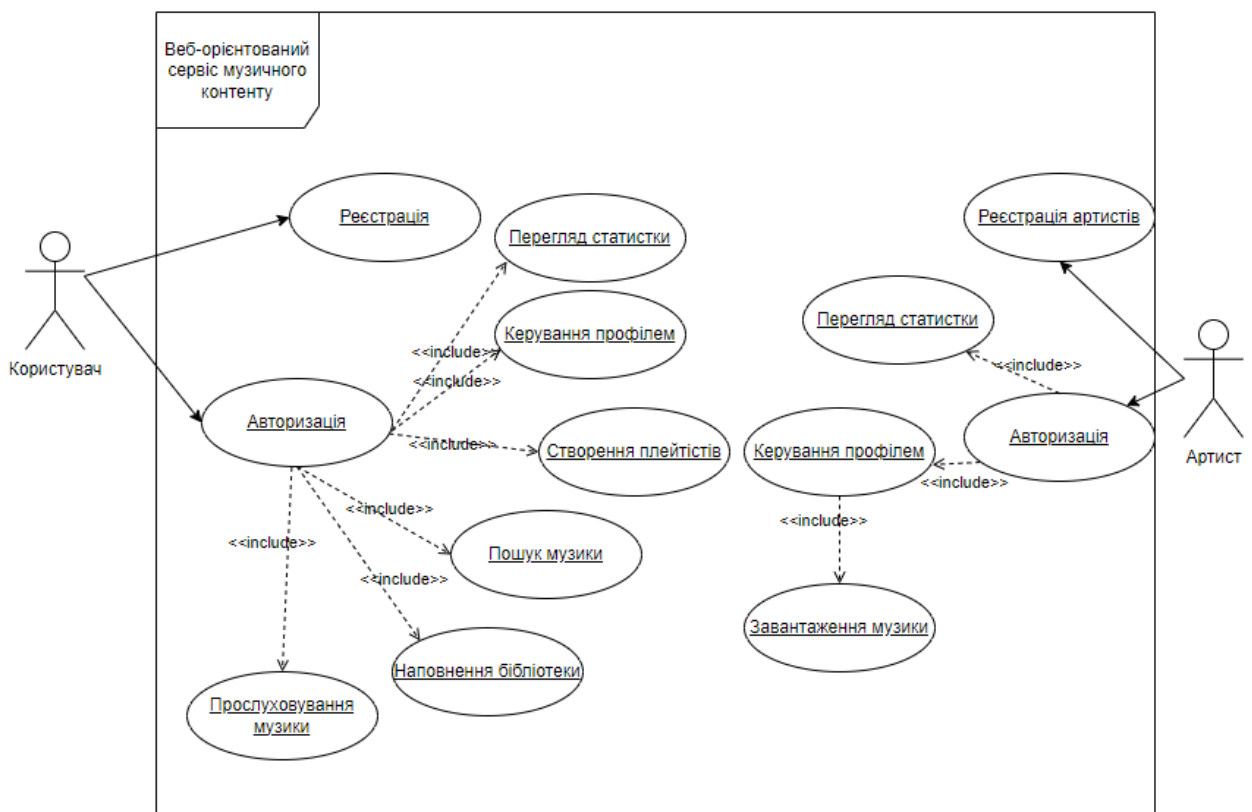


Рисунок 2.1 – Use case діаграма системи.

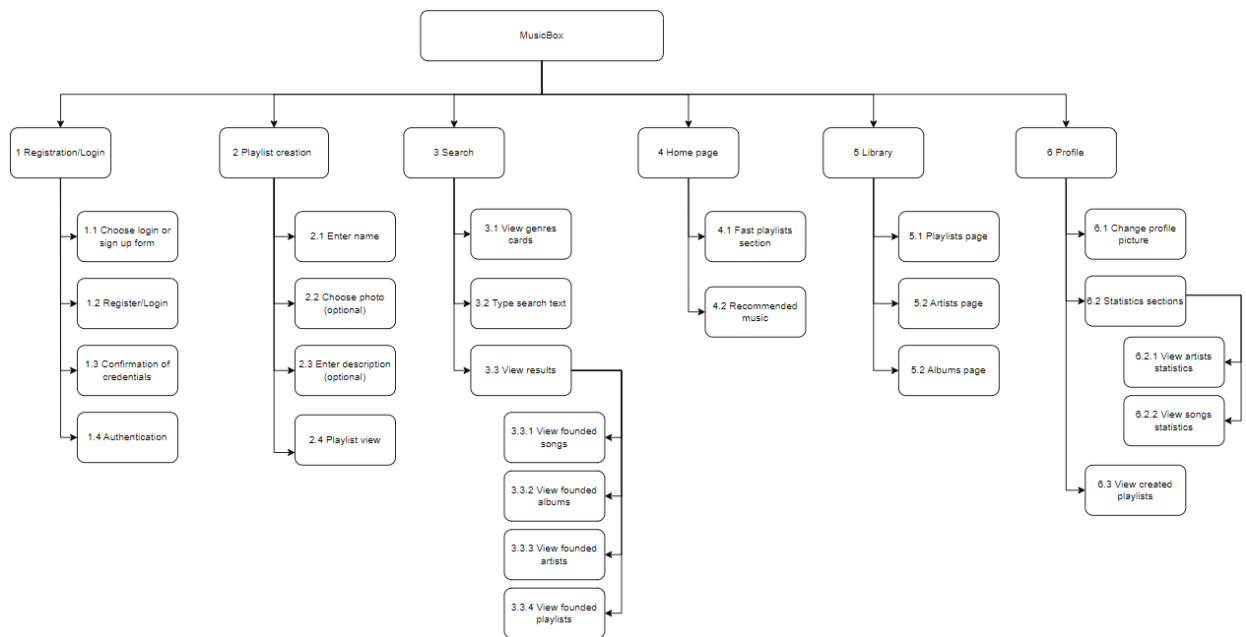


Рисунок 2.2 – FDD діаграма системи.

Функціональні правила сервісу:

- Лише зареєстровані користувачі можуть використовувати систему.
- Користувачі можуть створювати свої персональні облікові записи та налаштовувати їхній профіль, включаючи зміну фотографії профілю, ніку, пароллю тощо.
- Сервіс надає можливість пошуку музичних треків за назвою, виконавцем, жанром, альбомом тощо.
- Сервіс надає можливість прослуховування музики онлайн.
- Сервіс відслідковує та зберігає прослуховування користувачів задля збору статистики.
- Сервіс може надавати рекомендації користувачам на основі їхніх музичних вподобань та історії прослуховування.

Нефункціональні вимоги сервісу:

Інтуїтивність і зручність користування:

- Користувачам має бути легко розуміти і використовувати інтерфейс сервісу без необхідності додаткових інструкцій або навчання.
- Сервіс повинен бути придатним для користувачів з різними рівнями технічної грамотності, включаючи тих, хто не є досвідченими

користувачами комп'ютерів або мобільних пристроїв.

Швидкодія:

- Сервіс повинен завантажувати аудіо контент і відтворювати його без помітних затримок, щоб забезпечити плавне та безперервне прослуховування для користувачів.

2.4 Реалізація функціоналу інтеграції аналітичного функціоналу до сервісу музичного контенту

Головною метою технічного завдання є розробка та реалізація системного рішення, яке дозволить виконувати прослуховування, після чого зберігати їх аналізу та візуалізації статистичних показників. Спочатку розглянемо, програвач, який дозволить збирати інформацію про прослуховування. Даний компонент має забезпечувати можливість переходу між композиціями в порядку черги, ставлення музики на паузу, відтворення треків у режимі випадкового відтворення або активацію постійного повторення пісні. Крім того, важливою є можливість налаштування гучності.

Функціонал був реалізований за допомогою фреймворку React, в поєднанні з мовою програмування TypeScript. Для створення можливості програвання музики та обробки подій, пов'язаних з програванням, були використані стандартні конструкції фреймворку - хуки. Зокрема, хук `useEffect` [16] був використаний для забезпечення безперервного програвання аудіо та обробки подій, таких як вимкнення звуку, коли пісня ставиться на паузу, вимкнення звуку без зупинки програвання при натисканні на кнопку вимкнення, а також для зміни гучності під час програвання. Для того щоб скористатися функціоналом плеєра потрібен перелік пісень. Створюємо масив пісень для програвання:

```
const songs = [  
  {  
    id: 1,  
    order: 1,  
    name: "Ho Hay",  
    explicit: false,
```

```

    artistName: "The Lumineers",
    albumId: 1,
    albumName: "The Lumineers",
    durationSec: 162,
    albumCoverPath: "artists/TheLumineers/TheLumineers/TheLumineers.jpg",
    audioPath: "artists/TheLumineers/TheLumineers/1HoHay.mp3",
    dateAdded: "2012-04-03 00:00:00",
    numberOfListens: 24213012,
    isLiked: false,
  },
  {
    id: 2,
    order: 2,
    name: "Flowers In Your Hair",
    explicit: false,
    artistName: "The Lumineers",
    albumId: 1,
    albumName: "The Lumineers",
    durationSec: 110,
    albumCoverPath: "artists/TheLumineers/TheLumineers/TheLumineers.jpg",
    audioPath: "artists/TheLumineers/TheLumineers/2FlowersInYourHair.mp3",
    dateAdded: "2012-04-03 00:00:00",
    numberOfListens: 12301,
    isLiked: false,
  }
];

```

Для того щоб розуміти яку пісню зараз потрібно програвати використаємо концепцію контекстів в React. Контекст являє собою звичайний компонент, який по рівні ієрархії знаходиться вище, тому може передавати певну інформацію дочірнім компонентам, концептуально це виглядає наступним чином:

```

const App = (): JSX.Element => {
  return (
    <div className={styles.App}>
      <MusicContextProvider>
        <Player />
      </MusicContextProvider>
    </div>
  )
}

```

```
);
};
export default App;
```

Компонент `MusicContextProvider` містить в собі список пісень, які вибрав для відтворення користувач, інформацію про пісню, яка зараз має програватись і інформацію про те чи показаний зараз плеєр, це знадобиться для конфігурації стилів у інших компонентах системи та всі необхідні методи для того щоб мати змогу змінювати значення вищезазначених полів.

Після отримання списку пісень та поточної пісні з будь-якої сторінки, плеєр буде відображено на екрані і почне відтворення вибраної пісні. Для відтворення аудіо використовувався стандартний хук `useRef` [19]. Основна логіка плеєра була реалізована за допомогою головного хука `useEffect` у компоненті плеєра:

```
useEffect(() => {
  if (currentSong.order !== 0) {
    if (isPlaying) {
      audioTag.current.play();
      animationRef.current = requestAnimationFrame(whilePlaying);
      audioTag.current.volume = volume;
      if (isMuted) {
        audioTag.current.muted = true;
      } else audioTag.current.muted = false;
      const interval = setInterval(() => {
        const seconds = Math.floor(audioTag.current.duration);
        setDuration(seconds);
        progressBar.current.max = seconds;
      }, 1000);
      setInterval(() => {
        if (duration > 0 || duration !== undefined) {
          clearInterval(interval);
          if (audioTag.current.currentTime === audioTag.current.duration) {
            if (isListenCounted === false && currentSong.order !== 0 &&
              audioTag.current.currentTime === audioTag.current.duration) {
              setIsListenCounted(true);
              void dispatch(
```

```

        createUserListen(currentSong.id)
    );
    setIsListenCounted(false);
}
if (songs.length === 1 || isOnRepeat) {
    repeatSong();
} else {
    isRandom ? skipRandom() : skipForward();
}
}
}
}, 1100);
} else {
    audioTag.current.pause();
    audioTag.current.volume = volume;
    cancelAnimationFrame(animationRef.current);
}
}

}, [songs, currentSong, duration, isMuted, isPlaying, isListenCounted, isRandom, isOnRepeat, volume,
whilePlaying, skipBack, skipForward, skipRandom, repeatSong, dispatch]);

```

Цей хук керує різними аспектами програвання музики, такими як перемотування прогрес-бару пісні під час прослуховування, перемикання між піснями, зміну гучності, вимкнення звуку, перемотування пісні, зупинку програвання та навіть відправку запиту на сервер при повному прослуховуванні пісні. Більшість функцій, які були перелічені в кінці цього хука в масиві залежностей, були реалізовані як функції зворотнього виклику, які виконують прості операції з піснями або аудіофайлом. Наприклад, розглянемо функцію, яка випадково перемикає пісню:

```

const skipRandom = useCallback(() => {
    if (songs.length !== 1) {
        const randomNum = Math.floor(Math.random() * songs.length) + 1;
        if (randomNum === currentSong.order) {
            skipRandom();
        } else {
            setCurrentSong(songs.find((x) => x.order === randomNum)!);
        }
    }
}

```

```
}, [currentSong, songs, setCurrentSong]);
```

Ця функція використовує функцію `Math.random()` в мові TypeScript для відтворення наступної пісні випадковим чином. Також варто відзначити, що випадкове відтворення залежить від кількості пісень у контексті. Наприклад, якщо користувачем був обраний альбом, в якому є лише одна пісня, то ця функція очевидно не буде працювати, оскільки немає інших пісень для вибору.

Також важливо розглянути частину JSX компонента плеєра. У ній використані стандартні HTML-компоненти, такі як `<input>`, який відображає прогрес-бар аудіофайлу і надає можливість перемотувати пісню, `<button>`, який керує програванням (пауза або перемикання пісні), `<img>`, який відображає зображення альбому, де знаходиться поточна пісня. Також використані компоненти, які належать до бібліотеки React, такі як `<Link>`, який спрощує навігацію користувача до артистів, пісні яких відтворюються в даний момент при натисканні на елемент [22]. Реалізація JSX частини плеєру:

```
if (songs.length !== 0) {
  setPlayerVisibility(true);
  return (
    <div className={styles.Container}>
      <div className={styles.musicDiv}>
        {songs.map((music) =>
          currentSong.order === music.order ? (
            <> { <> <img src={"/" + music.albumCoverPath} alt={music.albumName} />
              <div className={styles.SongInfo}>
                <h1>{music.name}</h1>
                <Link className={styles.artistLink} to={`/artist/${music.artistId}`}
                  <h3>{music.artistName}</h3> </Link>
              </div> </> } <audio src={"/" + music.audioPath} ref={audioTag} />
            </> : null )}
        </div>
      <div className={styles.player}>
        <div className={styles.inputButtons}>
```

```

{ <div className={styles.progressBar}>
  <p className={styles.PcurrentTime}> calculateDuration(currentTime) </p>
  <input type="range" className={styles.currentProgress} defaultValue="0" ref={progressBar}
    onChange={changeRange} /> <p className={styles.Pduration}>calculateDuration(duration) </p>
</div> }
<div className={styles.buttons}>
  <button onClick={() => setIsRandom(!isRandom)} className={styles.randomMusicsButton}>
    {isRandom ? <RandomMusicsTrue /> : <RandomMusicsFalse />} </button>
  <button onClick={skipBack}> <SkipBack /> </button>
  <button className={styles.playPause} onClick={() => setIsPlaying(!isPlaying)} >
    {isPlaying ? <Pause /> : <Play />} </button>
  <button onClick={skipForward}> <SkipForward /> </button>
  <button onClick={() => {setIsOnRepeat(!isOnRepeat)}} className={styles.onRepeatButton}>
    {isOnRepeat ? <RepeatOn /> : <RepeatOff />} </button>
</div>
</div>
</div>
{
  <div className={styles.test}>
    <button className={styles.volumeButton} onClick={() => setIsMuted(!isMuted)} >
      {isMuted ? <VolumeOff /> : <VolumeOn />} </button>
    <input type="range" step="0.01" onChange={(e) => setVolume(e.target.value)} value={volume}
      max="1" min="0" />
  </div> } </div> );
} else {
  return <></>;
}

```

Розглянемо детальніше рядки, які дозволяють нам зберігати дані для збереження даних, які в подальшому будуть використовуватись для аналізу та візуалізації статистики.

```

if (isListenCounted === false && currentSong.order !== 0 &&
  audioTag.current.currentTime === audioTag.current.duration) {
  setIsListenCounted(true);
  void dispatch(
    createUserListen(currentSong.id)
  );
  setIsListenCounted(false);
}

```

У цих рядках першочерговим є умова чи була дана композиція вже прослуханою за це програвання, для реалізації якої використовується стандартний хук `useState` у змінній `isListenCounted`. Після цього проводиться порівняння часу відтворення аудіофайлу з поточним часом прослуховування. Якщо всі умови виконані, ми відправляємо запит на сервер для збереження інформації про те, що пісню було прослухано, і встановлюємо значення `isListenCounted` на `false` для фіксації прослуховування наступних пісень.

Після цього наступна інформація буде збережена:

а) Зберігання інформації про користувача, який здійснив прослуховування:

- Користувач_ID: унікальний ідентифікатор користувача.
- Ім'я користувача: ім'я або псевдонім користувача (буде попульовано з токена, який фронтенд надсилає разом з запитом).
- Email: адреса електронної пошти користувача (якщо доступно).

б) Зберігання інформації про прослуховану композицію:

- ID композиції: унікальний ідентифікатор пісні.
- Назва композиції: назва пісні (буде попульовано з об'єкту пісні).
- Артист: виконавець пісні (буде попульовано з батьківського об'єкту пісні).
- Дата прослуховування: точна дата та час прослуховування

Після виконання цих кроків, дані стануть доступними для подальшого аналізу та візуалізації. Давайте розглянемо приклад того, як було реалізовано отримання загальної статистики щодо прослуховування користувача за весь час на серверній стороні.

Обробник цього запиту виглядає наступним чином:

```
public class GetGlobalListeningDataDtoQueryHandler :  
    LoggableRequestHandlerBase<GetGlobalListeningDataQuery, OutgoingGetGlobalListeningDataDto>  
{  
    private readonly IUnitOfWork _unitOfWork;  
  
    public GetGlobalListeningDataDtoQueryHandler(ILogger logger, IHttpContextAccessor httpContextAccessor,  
        IUnitOfWork unitOfWork)  
        : base(logger, httpContextAccessor)
```

```

{
    _unitOfWork = unitOfWork;
}

public override async Task<IHandlerResult<OutgoingGetGlobalListeningDataDto>>
Handle(GetGlobalListeningDataQuery query, CancellationToken cancellationToken)
{
    var userName = HttpContextAccessor.HttpContext?.User.Claims.Single(x => x.Type is
nameof(IdentityUserModel.Name)).Value;

    var listens = await _unitOfWork.UserListenRepository.GetGlobalListeningData ByUsernameAsync(userName,
cancellationToken);

    return Data(new OutgoingGetGlobalListeningDataDto(listens.Count, GetTotalHours(listens)));
}

private int GetTotalHours(IEnumerable<UserListen> listens)
{
    return (int)Math.Floor(listens.Select(x => x.Song.DurationSec).Sum() / 3600.0);
}
}

```

В класі обробника існує поле `_unitOfWork`, яке відповідає за зв'язок з репозиторіями. У цьому конкретному випадку, це репозиторій для зберігання даних про прослуховування користувача. На початку методу `Handle`, ми отримуємо ім'я користувача з токєну, який був розглянутий раніше. Потім виконується асинхронний запит до бази даних, щоб отримати всі дані про прослуховування цього користувача. Після цього ми викликаємо метод `GetTotalHours`, який обчислює загальну кількість годин прослуховування усіх пісень, які були отримані з бази даних. На завершення, результат обробки обробника повертається у вигляді вихідної моделі даних.

Вихідний об'єкт має наступний вигляд:

```
public record OutgoingGetGlobalListeningDataDto(int NumberOfSongs, int Hours) : IOutgoingControllerDto;
```

Висновки до розділу 2

У другому розділі була розглянута реалізація основних функцій програми, включаючи створення комплексного рішення для збору статистики по прослуховуванням та обробника серверної сторони одного з запитів на отримання статистичних даних. Були надані фрагменти коду з відповідними поясненнями, які розкривають, які функції програвача будуть реалізовані і як можна керувати відтворенням музики, її зберіганням та як статистика по прослуховуванням в подальшому може бути візуалізована.

Вибір архітектури програмного рішення було зроблено на основі дослідження. Враховуючи потреби цільової аудиторії та технічні особливості задачі, було вирішено використовувати комбінацію сучасних технологій, таких як ASP.NETCore/C# для бекенду, React/TypeScript для фронтенду, Figma для дизайну. Ці засоби дозволять створити високопродуктивне, безпечне та легко масштабоване рішення, яке відповідатиме всім сучасним стандартам та вимогам користувачів

РОЗДІЛ 3. ПОРЯДОК ВЗАЄМОДІЇ З МОДУЛЕМ АНАЛІТИЧНОГО ФУНКЦІОНАЛУ

3.1. Порядок взаємодії з інтерфейсом модуля аналітичного функціоналу.

У головному меню сервісу користувач може побачити наступні опції (Рис 3.1).

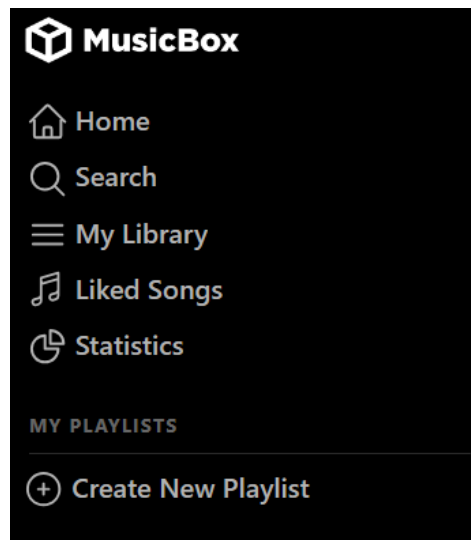


Рисунок 3.1 - Сторінки доступні з меню

Для переходу до модуля аналітичного функціоналу потрібно обрати опцію Statistics. На сторінці статистики першою частиною проаналізованих статистичних даних користувача є загальні відомості про кількість прослуханих пісень та час прослуховування (Рис. 3.2).

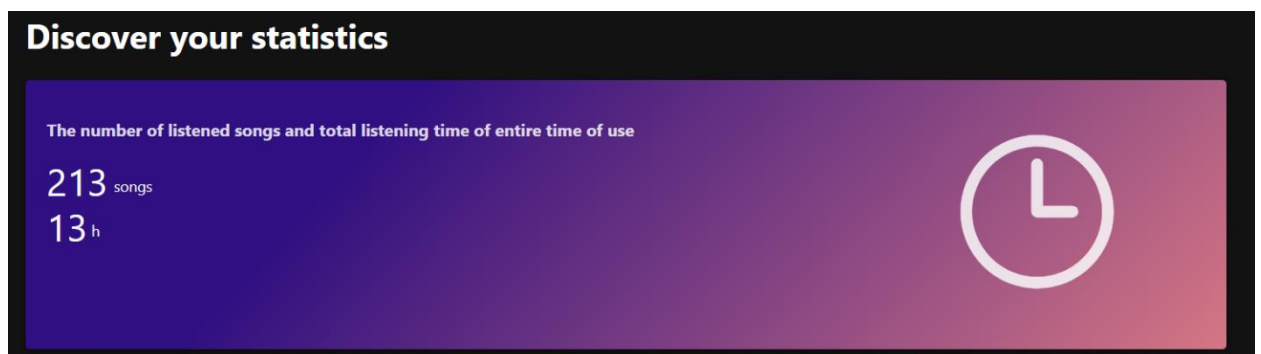


Рисунок 3.2 - Відомості про прослухані пісні та години

Наступним статистичним показником є відомість про кількість прослуховування за останній тиждень, відображений у графіку. Також у

окремому блоці користувач може побачити який день був з найбільшою кількістю прослуховувань (Рис. 3.3).

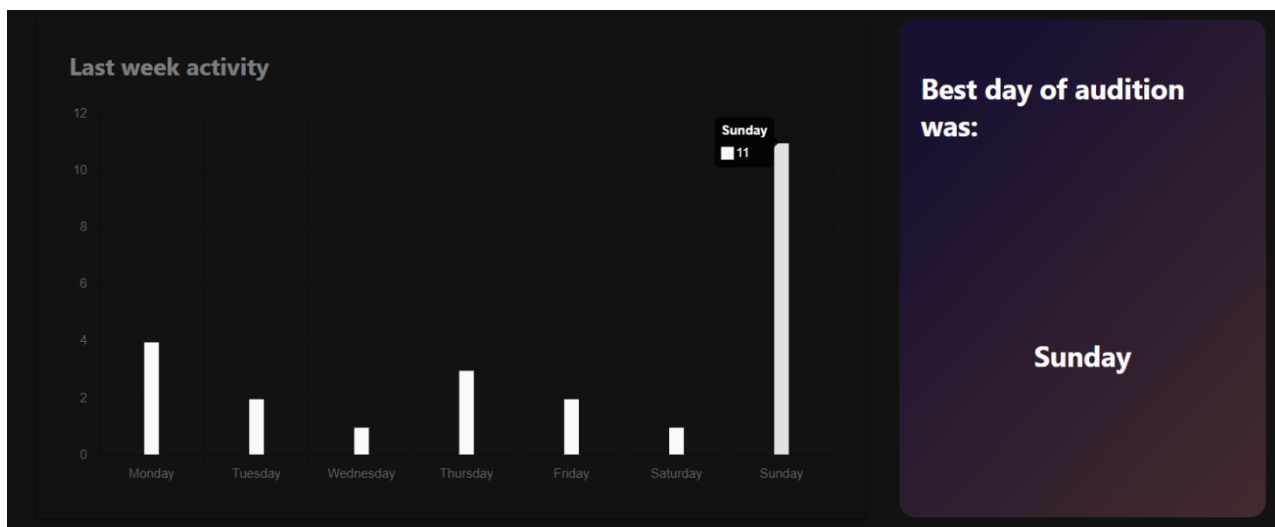


Рисунок 3.3 - Графік прослуханих пісень за тиждень

Нижче на сторінці користувач може побачити інформацію про пісні які він найбільше прослуховував у цьому місяці з відображеною точною кількістю прослуховувань (Рис. 3.4). Далі можемо побачити подібну секцію для артистів (Рис. 3.5).

Top songs this month				
1		don't forget to breathe today	10 listens	4:07
2		Guys	9 listens	4:30
3		The Mother We Share	9 listens	3:11
4		Be My Mistake	8 listens	4:16
5		My Cell	7 listens	3:16
6		Salt And The Sea	7 listens	4:30
7		Formidable	6 listens	2:56
8		Blossom	6 listens	3:46
9		idk anything (demo)	5 listens	3:12
10		All Up In My Head	4 listens	3:51

Рисунок 3.4 - Найкращі пісні за місяць

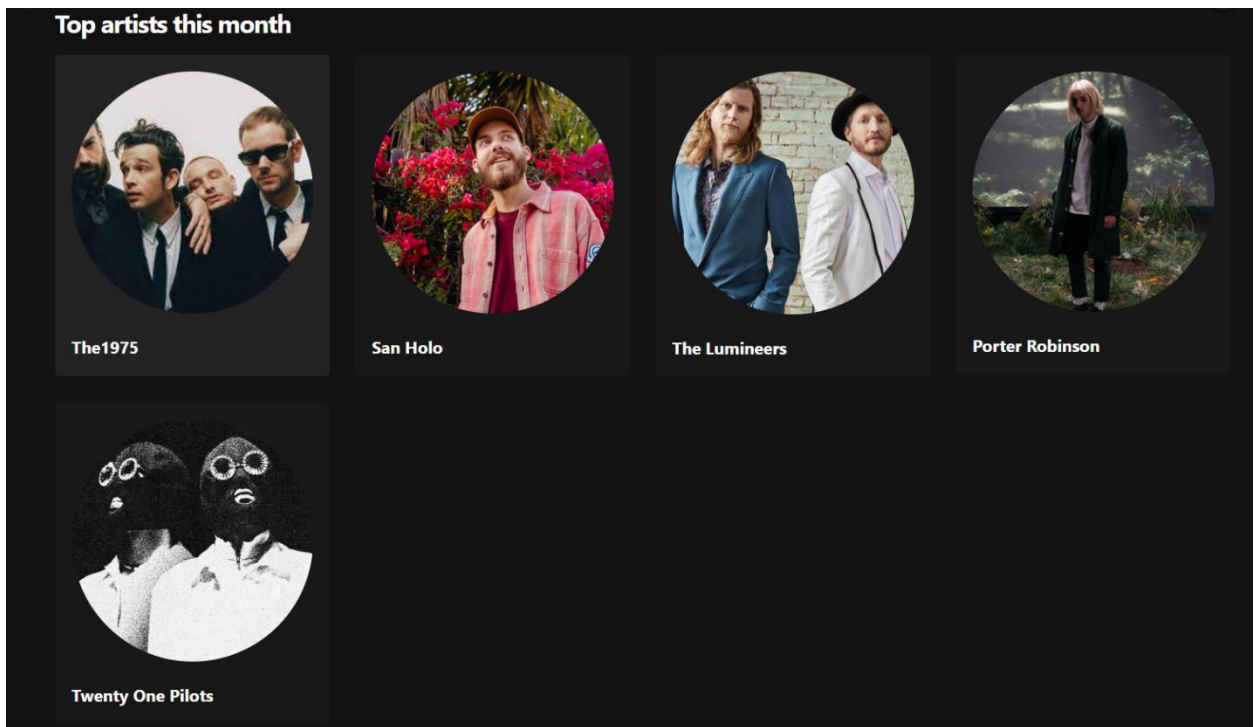


Рисунок 3.5 - Найкращі артисти за місяць

Останнім розглянемо графік, який відображує кількість прослуховувань користувача за рік з помісячними показниками. Також користувач одразу може побачити у якому місяці була найбільш кількість прослуховувань в блоці справа (Рис. 3.6).



Рисунок 3.6 - Графік прослуховувань за рік

3.2. Порядок взаємодії з додатковими елементами системи.

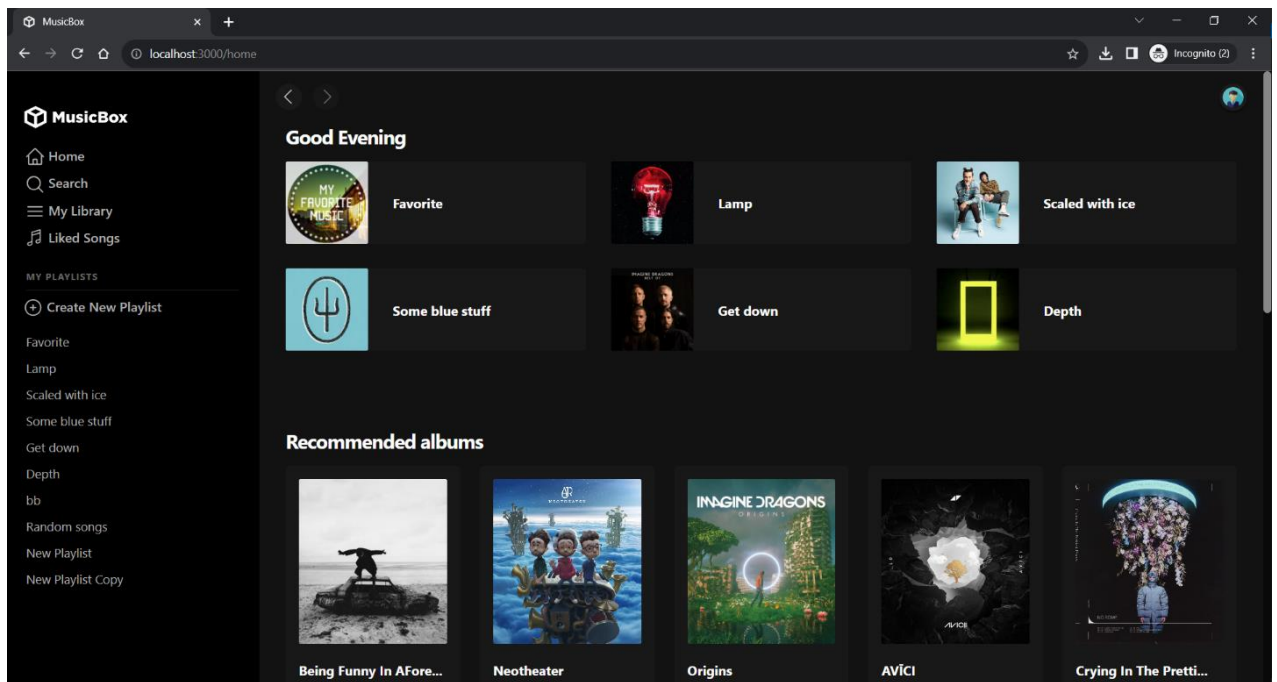


Рисунок 3.7 - Головна сторінка

На головній сторінці сервісу користувач зможе знайти рекомендовану музику, представлену у формі швидких плейлистів, що розташовані у верхній частині сторінки. Крім того, на цій сторінці є секції з рекомендованими плейлистами, доступ до яких можна знайти далі на сторінці. Для зручності користувача, існує навігаційна панель, розташована з лівого боку, яка дозволяє швидко переходити між різними сторінками сервісу. Наприклад, ми розглянемо другий пункт меню - пошук (Рис. 3.8).

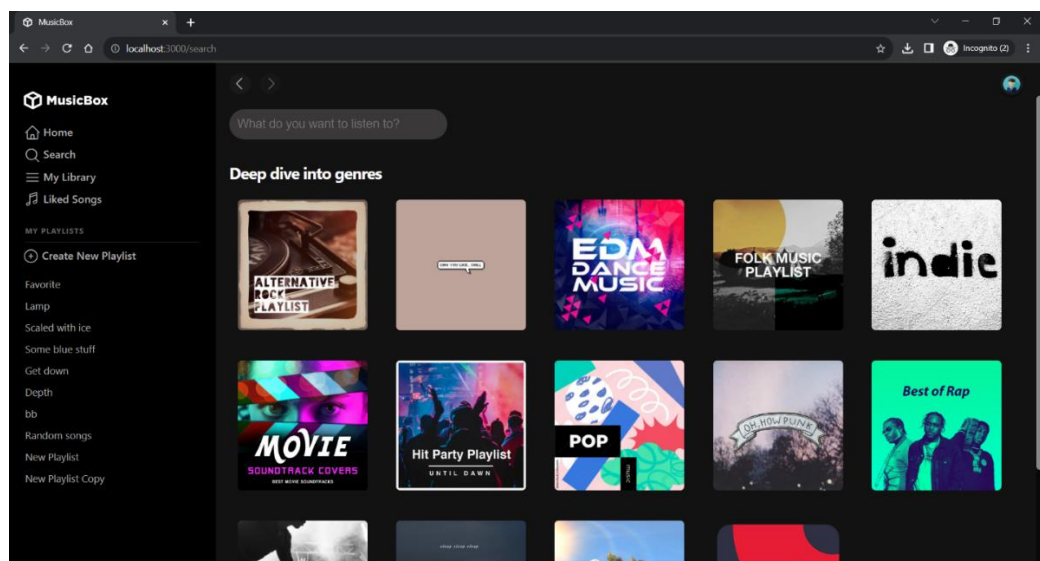


Рисунок 3.8 – Сторінка пошуку

На цій сторінці сервісу користувач може виконувати пошук за допомогою карток жанрів або, при бажанні, вводити запит у пошукову стрічку. Якщо потрібно, результати можна очистити. Після цього результати відображаються у вигляді окремих списків для пісень, альбомів, виконавців та публічних плейлистів інших користувачів, які доступні для загального використання (див. Рис. 3.9).

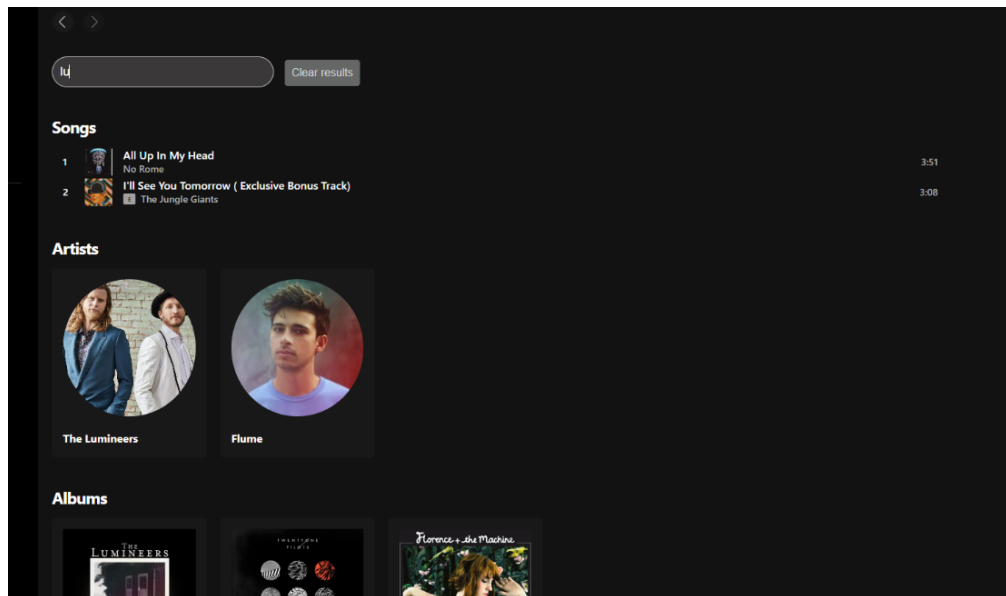


Рисунок 3.9 – Результати пошуку

Далі розглянемо сторінку бібліотеки користувача. Ця сторінка дозволяє користувачу переглядати свої збережені плейлисти, альбоми та артистів. За замовчуванням, першою сторінкою, яка буде відображена, є сторінка плейлистів (див. Рис. 3.10).

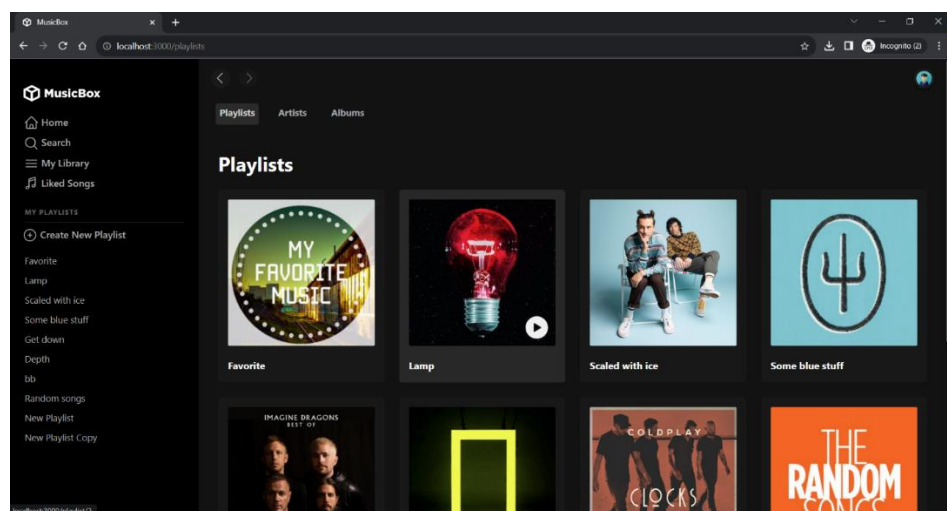


Рисунок 3.10 – Вкладка плейлистів у бібліотеці

Детальніше розглянемо сторінку плейлиста та функції, які можна використовувати на цій сторінці (Рис. 3.11).

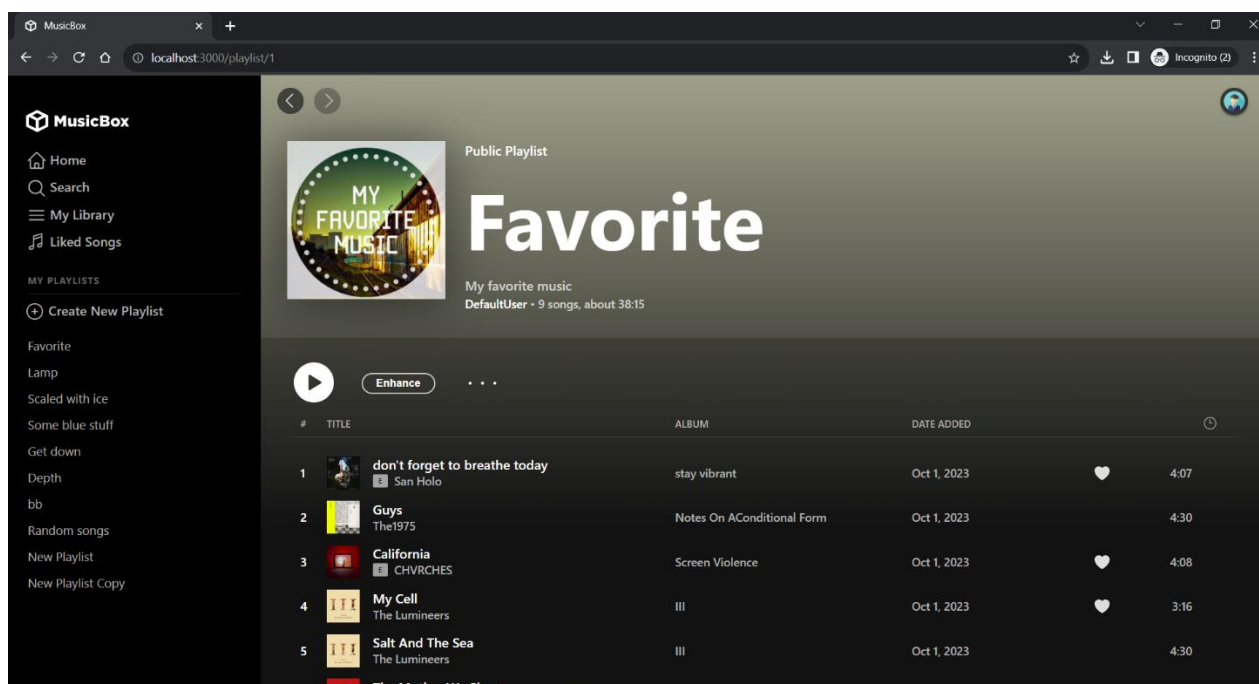


Рисунок 3.11 – Сторінка плейлиста

На даній сторінці користувач може розпочати прослуховування музики, натиснувши кнопку "Грати" під обгорткою плейлисту. У режимі програвання пісні будуть відтворюватися в порядку, починаючи з першої пісні, або користувач може вибрати конкретну пісню з плейлисту, з якої він хоче почати прослуховування. Після увімкнення музики відобразиться плеєр, і пісня, яка програватиметься, буде підсвічена (Рис. 3.12).

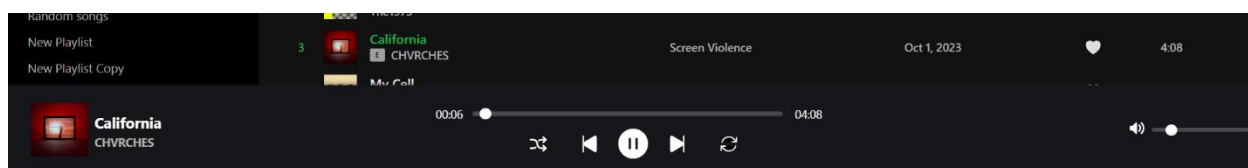


Рисунок 3.12 – Програвання пісні

Також можна виконувати редагування плейлисту за допомогою контекстних меню. Для користувача на цій сторінці доступне контекстне меню плейлиста (Рис. 3.13) або контекстне меню пісні (Рис. 3.14).

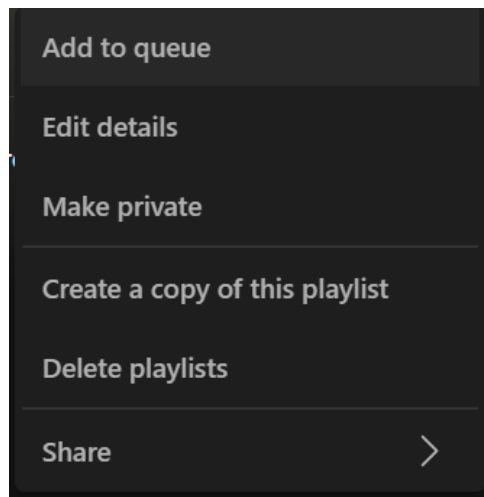


Рисунок 3.13 – Контекстне меню плейлиста

Використовуючи контекстне меню плейлиста, користувач може виконувати наступні дії: додавати плейлист в чергу для програвання, редагувати плейлист (ця можливість також доступна при наведенні на обгортку плейлисту), змінювати рівень доступу до плейлисту, створювати копію плейлисту, видаляти його та ділитися ним. Важливо зауважити, що деякі з цих опцій доступні тільки для автора плейлисту.

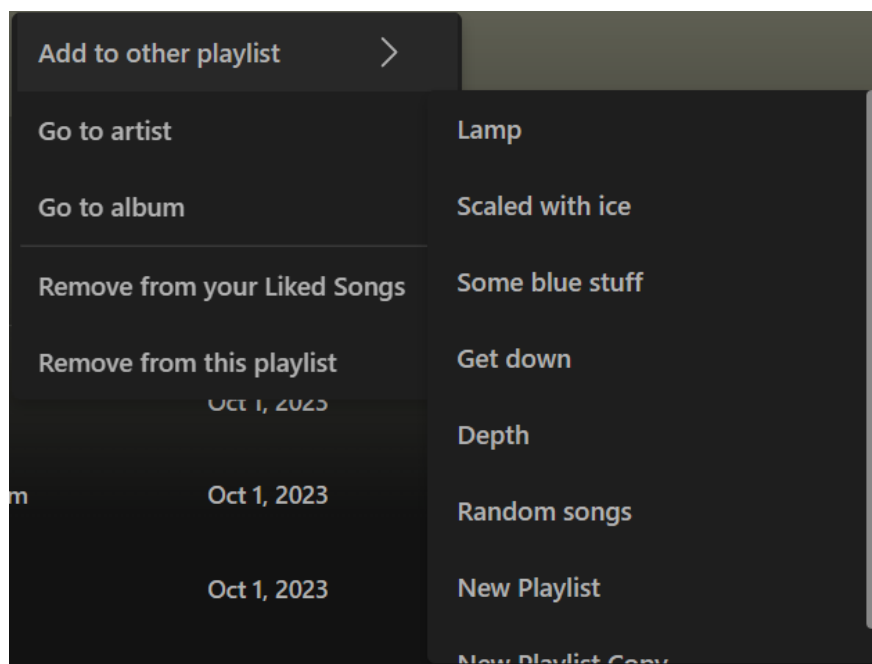


Рисунок 3.14 – Контекстне меню пісні.

У цьому контекстному меню користувач може виконувати дії щодо пісні, такі як додавання цієї пісні до іншого власного плейлисту, перехід на сторінку артиста, перехід до альбому, в якому знаходиться пісня, видалення

пісні з плейлисту, а також, якщо ця пісня раніше була лайкнута, її можна видалити із списку улюблених пісень. Аналогічно до контекстного меню плейлиста, деякі з цих опцій доступні лише автору пісні.

Перейдемо до вкладинки артистів бібліотеки (Рис. 3.15).

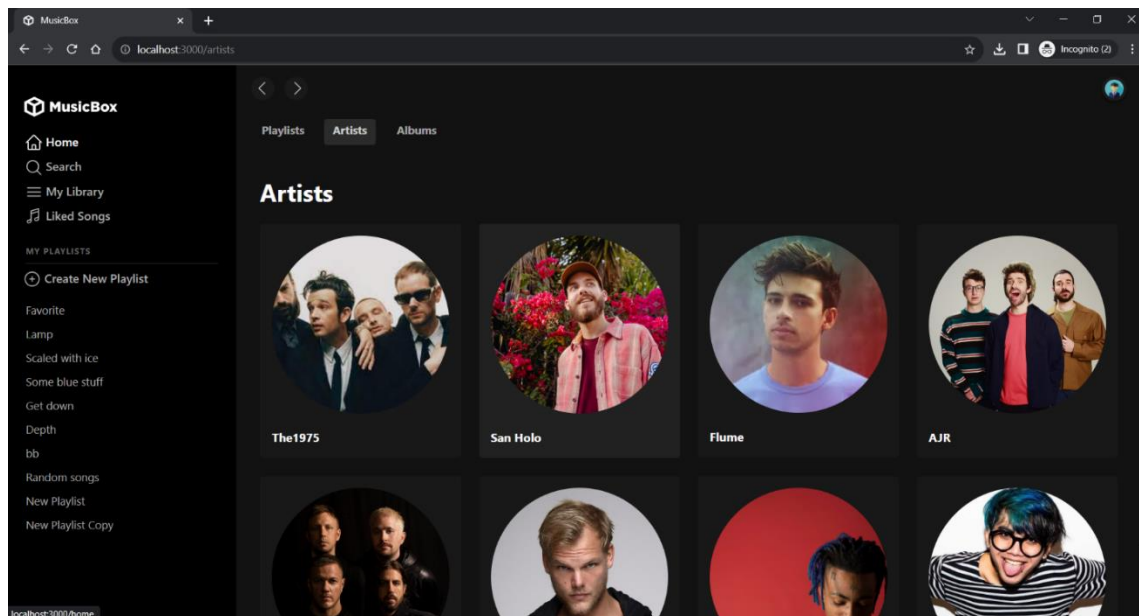


Рисунок 3.15 – Вкладка артистів у бібліотеці

На цій вкладці артисти будуть представлені у порядку, в якому користувач останнього разу підписувався на них. З цієї вкладки можна навігуватися на сторінки артистів, яких користувач хоче передивитися.

Розглянемо детальніше сторінку артиста (Рис. 3.16).

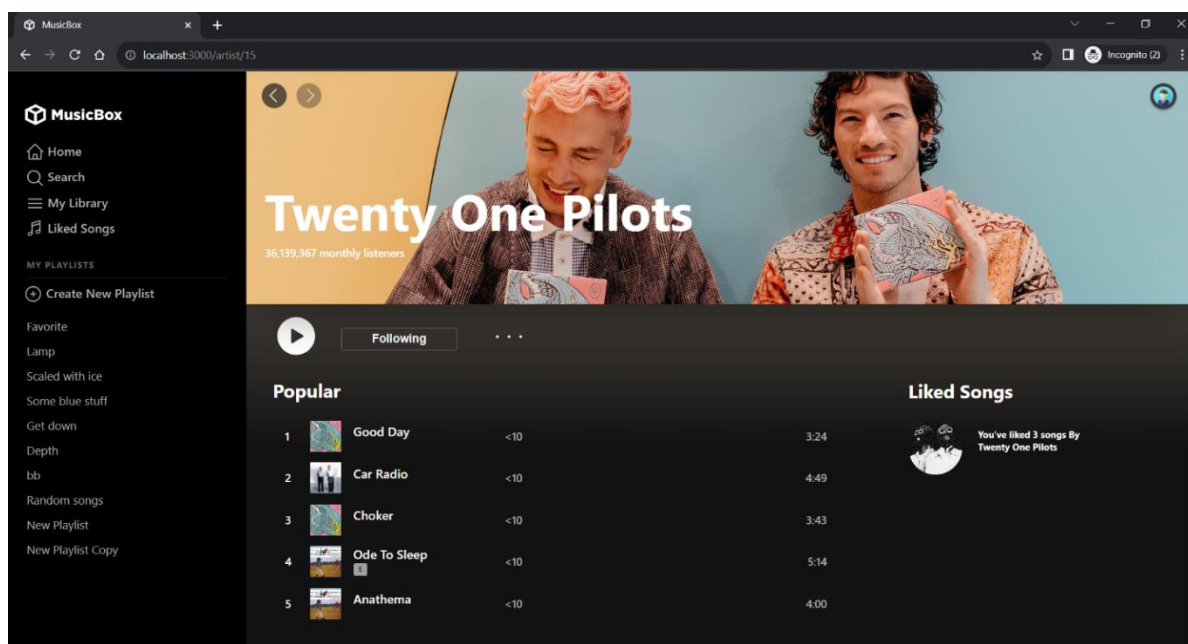


Рисунок 3.16 – Сторінка артиста

На сторінці артиста користувач може переглянути статистику місячних прослуховувань артиста, чи його сторінку верифіковано, найпопулярніші пісні, підписатися на артиста або відписатися від нього

Після цього користувач перейти на вкладинку збережених альбомів в бібліотеці (Рис. 3.17).

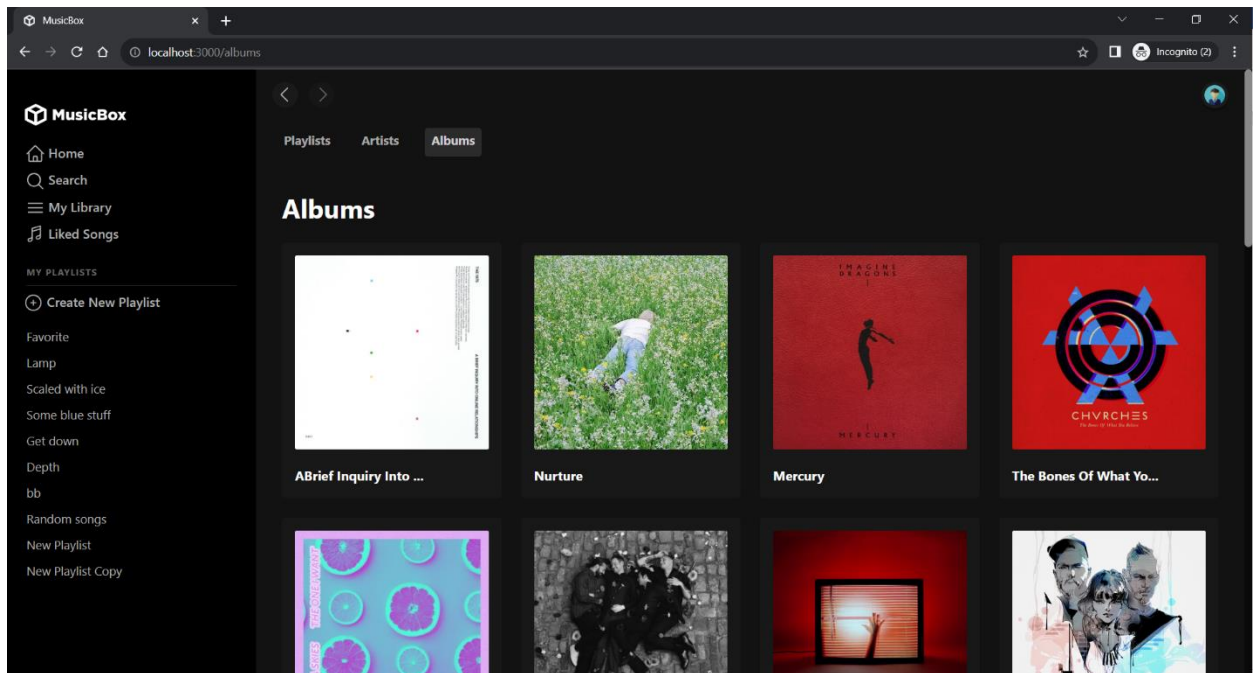


Рисунок 3.17 – Вкладка альбомів в бібліотеці

З цієї сторінки користувач може навігуватися на улюблені альбоми. Розглянемо детальніше сторінку альбому (Рис. 3.18).

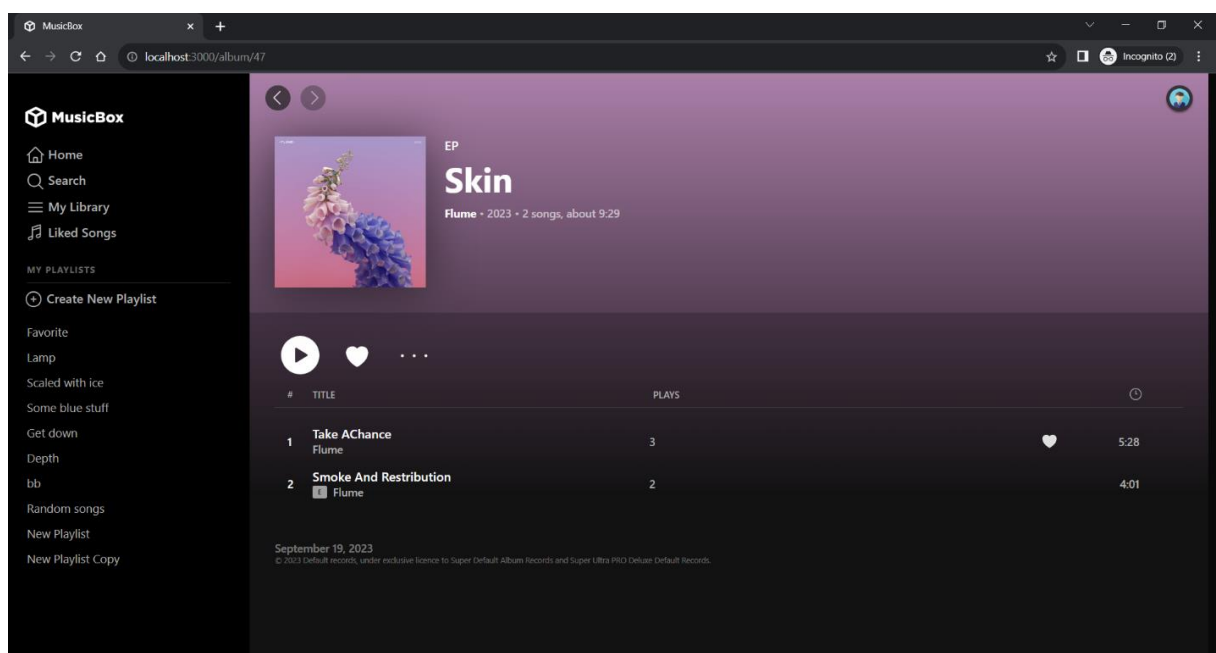


Рисунок 3.18 – Сторінка альбома

На цій сторінці користувач може ознайомитись з альбомом артиста, функціонал щодо програвання дещо схожий на плейлист, тобто можна розпочати спочатку або обрати конкретну. З приводу керування можна розглянути контекстне меню альбому (Рис. 3.19).

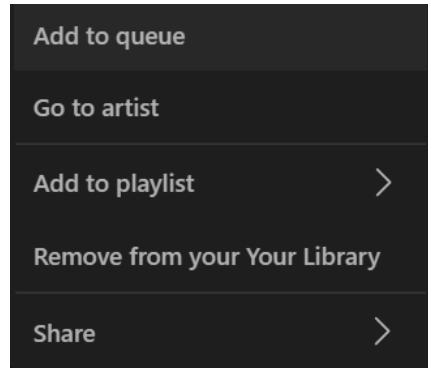


Рисунок 3.19 – Контекстне меню альбома.

Це контекстне меню надає користувачу можливість додати всі пісні з альбому до черги програвання, перейти на сторінку артиста, включити всі пісні з альбому в один з власних плейлистів, видалити альбом з бібліотеки, якщо він вже там є, або додати його, якщо він ще не був доданий. Крім того, можна натиснути на опцію "поділитися", після чого посилання на альбом буде скопійоване в буфер обміну.

Наступною розглянемо функцію створення плейлисту, яка доступна на боковій панелі(Рис. 3.20).

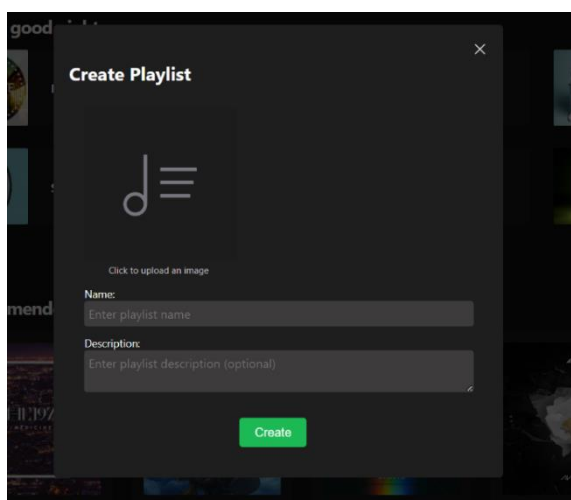
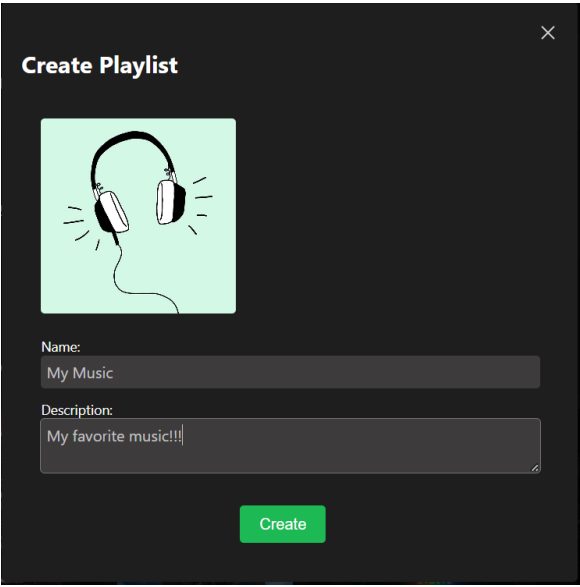


Рисунок 3.20 – Форма створення плейлисту

Для створення плейлисту, користувач може вибрати зображення, заповнити поля з назвою та описом, але також є можливість створити

плейлист, вказавши лише його назву. Плейлист буде створено з приватним рівнем доступу за замовчуванням. Заповнимо всі поля і створимо новий плейлист (Рис 3.21).

A dark-themed modal dialog box titled "Create Playlist" with a close button (X) in the top right corner. It features a light green square placeholder for a playlist image showing a pair of headphones. Below the image are two text input fields: "Name:" with the value "My Music" and "Description:" with the value "My favorite music!!!". A green "Create" button is positioned at the bottom center.

Create Playlist

My Music

My favorite music!!!

Create

Рисунок 3.21 – Створення плейлисту

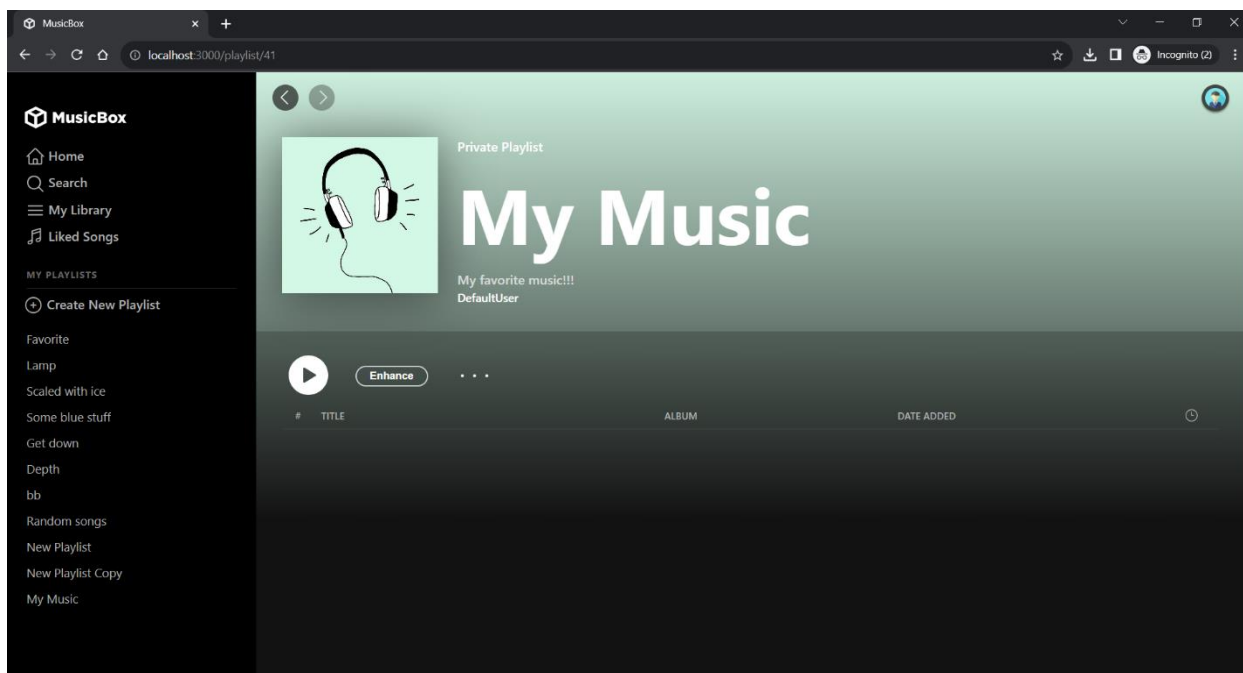


Рисунок 3.22 – Створений плейлист

Висновки до розділу 3

У третьому розділі магістерської кваліфікаційної роботи було детально розглянуто інтерфейс взаємодії з модулем інтеграції аналітичного функціоналу. Також були презентовані сторінки артистів, рекомендації, статистика та можливості пошуку.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи магістра була реалізована інтеграція аналітичного функціоналу до сервісу музичного контенту. На етапі початкового дослідження були проаналізовані конкуренти та їхні сервіси, і виявлені переваги та недоліки кожного з них. Основний функціонал системи був визначений, а технічне завдання та макет системи були імплементовані у програмі Figma.

У роботі був здійснений вибір оптимальних інструментів для розробки сервісу з метою забезпечення вищої ефективності порівняно з конкурентами. Друга частина кваліфікаційної роботи включала в себе представлення основного функціоналу кінцевого рішення.

У третьому розділі роботи був продемонстрований інтерфейс сервісу музичного контенту, а також надано детальний опис процесу користування додатком, та взаємодію з інтегрованим модулем візуалізації статистики. Було продемонстровано структуру сервісу, де користувачі можуть шукати музику, додавати її до своїх плейлистів та слухати в режимі онлайн.

Отже, завершивши виконання кваліфікаційної роботи, було створено інтеграцією аналітичного модулю відображення статистики для сервісу музичного контенту з, інтуїтивним та адаптивним дизайном, що робить користування ним простим, комфортним та інтерактивним. Розроблене рішення відповідає сучасним стандартам та критеріям, та містить іноваційний функціонал що робить його конкурентоспроможним на ринку музичних сервісів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Павлішин М. С. АНАЛІЗ ТЕНДЕНЦІЙ РОЗВИТКУ РИНКУ МУЗИЧНИХ СТРІМІНГОВИХ СЕРВІСІВ – апробація дослідження на конференції “Комп’ютерні технології: інновації, проблеми, рішення” (м. Житомир), 29-30 листопада 2023 р.
2. Ендрю Троелсен. Мова програмування C# 7 і платформи .NET Core (видання номер 8) 2019. – 1146 ст, 98 ст.
3. C# Corner [Електронне джерело] – Режим доступу до ресурсу <https://www.c-sharpcorner.com/article/command-mediator-pattern-in-asp-net-core-using-mediator2/>
4. Code academy [Електронне джерело] – Режим доступу до ресурсу <https://www.codecademy.com/resources/blog/what-is-react/>
5. Dev T [Електронне джерело] – Режим доступу до ресурсу <https://dev.to/brianschnee/how-to-learn-and-use-typescript-a-comprehensive-beginners-guide-5l>
6. Digital trends [Електронне джерело] – Режим доступу до ресурсу <https://www.digitaltrends.com/home-theater/what-is-youtube-music/>
7. Free code camp [Електронне джерело] – Режим доступу до ресурсу <https://www.freecodecamp.org/news/figma-crash-course/>
8. Geeks for geeks [Електронне джерело] – Режим доступу до ресурсу <https://www.geeksforgeeks.org/repository-design-pattern/>
9. Imore [Електронне джерело] – Режим доступу до ресурсу <https://www.imore.com/apple-music>
10. Imore [Електронне джерело] – Режим доступу до ресурсу <https://www.infoq.com/news/2021/10/ddd-vas-gateway-soundcloud/>
11. Java T point [Електронне джерело] – Режим доступу до ресурсу <https://www.javatpoint.com/uml-use-case-diagram>
12. JetBrains Blog [Електронне джерело] – Режим доступу до ресурсу <https://blog.jetbrains.com/blog/2023/12/06/introducing-jetbrains-ai-and-the-in-ide-ai-assistant/>

13. Log Rocket [Електронне джерело] – Режим доступу до ресурсу
<https://blog.logrocket.com/the-definitive-guide-to-scss/>
14. Mdn Web Docs [Електронне джерело] – Режим доступу до ресурсу
https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction
15. Microsoft Learn [Електронне джерело] – Режим доступу до ресурсу
<https://learn.microsoft.com/en-us/archive/msdn-magazine/2015/august/cutting-edge-cqrs-and-events-a-powerful-duo>
16. React dev [Електронне джерело] – Режим доступу до ресурсу
<https://react.dev/reference/react/useState>
17. React legacy [Електронне джерело] – Режим доступу до ресурсу
<https://uk.legacy.reactjs.org/docs/components-and-props.html>
18. Researchgate [Електронне джерело] – Режим доступу до ресурсу
https://www.researchgate.net/publication/366295843_Analysis_of_The_Trend_of_Spotify
19. Upmostlly [Електронне джерело] – Режим доступу до ресурсу
<https://upmostlly.com/tutorials/how-to-react-userref-hook>
20. Visual Paradigm [Електронне джерело] – Режим доступу до ресурсу
<https://online.visual-paradigm.com/diagrams/features/functional-decomposition-diagram-tool/>
21. Visual Studio Code [Електронне джерело] – Режим доступу до ресурсу
<https://code.visualstudio.com/Docs#:~:text=Visual%20Studio%20Code%20is%20a%20lightweight%20but%20powerful,%28such%20as%20C%2B%2B%2C%20C%23%2C%20Java%2C%20Python%2C%20PHP%2C%20Go%2C.NET%29>
22. W3 schools [Електронне джерело] – Режим доступу до ресурсу
https://www.w3schools.com/react/react_jsx.asp

ДОДАТКИ

Лістинг програмного коду

```

namespace MusicBox.Application.CQS.Playlist.Commands.Handlers;

public class CreatePlaylistCommandHandler :
    LoggableRequestHandlerBase<CreatePlaylistCommand,
    OutgoingCreatePlaylistDto>
{
    private readonly IUnitOfWork _unitOfWork;
    private readonly List<IDomainModelBusinessRule<PlaylistDomainModel>>
    _playlistDomainModelRules;
    private readonly
    List<IDomainModelBusinessRule<UserPlaylistDomainModel>>
    _userPlaylistDomainModelRules;

    public CreatePlaylistCommandHandler(
        IUnitOfWork unitOfWork,
        ILogger logger,
        IHttpContextAccessor httpContextAccessor,
        IEnumerable<Tuple<DomainModelOperations,
        IDomainModelBusinessRule<PlaylistDomainModel>>>
        playlistDomainModelRules,
        IEnumerable<Tuple<DomainModelOperations,
        IDomainModelBusinessRule<UserPlaylistDomainModel>>>
        userPlaylistDomainModelBusinessRules)
        : base(logger, httpContextAccessor)
    {
        _unitOfWork = unitOfWork;
        _playlistDomainModelRules = playlistDomainModelRules.Where(x =>
        x.Item1 == DomainModelOperations.Create)
            .Select(x => x.Item2)
            .ToList();
        _userPlaylistDomainModelRules =
        userPlaylistDomainModelBusinessRules.Where(x => x.Item1 ==
        DomainModelOperations.Create)
            .Select(x => x.Item2)
            .ToList();
    }

    public override async Task<IHandlerResult<OutgoingCreatePlaylistDto>>
    Handle(CreatePlaylistCommand command, CancellationToken cancellation_token)
    {
        var username = HttpContextAccessor.HttpContext?.User.Claims.Single(x =>
        x.Type is nameof(IdentityUserModel.Name)).Value;
    }
}

```

```

        var user = await _unitOfWork.UserRepository.GetByNameAsync(username,
cancellationTokens);

        var userPlaylistWithSameName = await
_unitOfWork.PlaylistRepository.ExistByAuthorNameAndPlaylistName(user.Nam
e, command.Description, cancellationTokens);

        if (userPlaylistWithSameName)
        {
            Logger.Error($"User {user.Name} already have playlist with name
{command.Name}.", HttpContextAccessor.HttpContext!.TraceIdentifier);
            return BadRequest<OutgoingCreatePlaylistDto>(new Error($"You already
have playlist with name {command.Name}, please create new name."));
        }

        string? newFileName = null;

        if (command.TitleImage is not null)
        {
            newFileName = await CreateUniqueFileName(user.Name,
command.Name, Path.GetExtension(command.TitleImage.FileName),
cancellationTokens);
        }

        var playlistCreationExecutionResult = await PlaylistDomainModel.Create(
            command.Name,
            command.Description,
            command.TitleImage is not null ? Pathes.PlaylistsImagesPath +
newFileName : Defaults.DefaultPlaylistImagePath,
            user.Name,
            _playlistDomainModelRules);

        if (!playlistCreationExecutionResult.Success)
        {
            Logger.Error("Error occurred during Playlist creation. " +
playlistCreationExecutionResult.Error.Message,
HttpContextAccessor.HttpContext!.TraceIdentifier);
            return
BadRequest<OutgoingCreatePlaylistDto>(playlistCreationExecutionResult.Error);
        }

        int? createdPlaylistId = null;
        Guid? userPlaylistId = null;

        try

```

```

    {
        var outgoingCreatePlaylistDto = await
_unitOfWork.PlaylistRepository.Create(playlistCreationExecutionResult.Payload,
cancellationTokens);

        var userPlaylistDomainModel = await UserPlaylistDomainModel.Create(
            user.Id,
            outgoingCreatePlaylistDto.Id,
            _userPlaylistDomainModelRules);

        userPlaylistId = await
_unitOfWork.UserPlaylistRepository.CreateAsync(userPlaylistDomainModel.Payload, cancellationTokens);
        createdPlaylistId = outgoingCreatePlaylistDto.Id;

        if (command.TitleImage is not null)
        {
            await SavePlaylistImage(command.TitleImage, newFileName!,
cancellationTokens);
        }

        Logger.Info($"Playlist with id {outgoingCreatePlaylistDto.Id} was
created.", HttpContextAccessor.HttpContext!.TraceIdentifier);
        return Data(outgoingCreatePlaylistDto);
    }
    catch (Exception exception)
    {
        Logger.Error(exception,
HttpContextAccessor.HttpContext!.TraceIdentifier);

        if (createdPlaylistId is not null && userPlaylistId is not null)
        {
            await
_unitOfWork.PlaylistRepository.DeleteAsync(createdPlaylistId.Value);
            await
_unitOfWork.UserPlaylistRepository.DeleteAsync(userPlaylistId.Value);
            await _unitOfWork.SaveAsync(cancellationTokens);
        }

        if (command.TitleImage is not null)
        {
            DeletePlaylistImage(newFileName!);
        }

        throw;
    }

```

```

    }
}
private async Task SavePlaylistImage(IFormFile playlistImage, string fileName,
CancellationTokens cancellationTokens)
{
    var baseDirectory = RemoveAfterSrc(Environment.CurrentDirectory);
    var fileStream = new FileStream(baseDirectory +
Pathes.PlaylistsImagesPathForSavingImage + fileName, FileMode.Create);
    await playlistImage.CopyToAsync(fileStream, cancellationTokens);
    await fileStream.DisposeAsync();
}

public static void DeletePlaylistImage(string fileName)
{
    var baseDirectory = RemoveAfterSrc(Environment.CurrentDirectory);
    File.Delete(baseDirectory + Pathes.PlaylistsImagesPathForSavingImage +
fileName);
}

private static string RemoveAfterSrc(string input)
{
    var index = input.IndexOf("src\\", StringComparison.Ordinal);

    if (index >= 0)
    {
        input = input[..(index + 4)];
    }

    return input;
}

private async Task<string> CreateUniqueFileName(string username, string
playlistName, string fileExtension, CancellationTokens cancellationTokens)
{
    var existingPlaylists = await
_unitOfWork.PlaylistRepository.GetPlaylistsByUsername(username,
cancellationTokens);
    var newPlaylistCount = existingPlaylists == null ? 0 : existingPlaylists.Count
+ 1;
    return newPlaylistCount + username + Regex.Replace(playlistName, @"[^A-
Za-z0-9\s]", "") + fileExtension;
}
}

```

```

using AutoMapper;
using Microsoft.AspNetCore.Http;
using MusicBox.Abstractions.Authorization;
using MusicBox.Abstractions.Domain.DomainModels;
using MusicBox.Abstractions.HandlerResult;
using MusicBox.Application.Constants;
using MusicBox.Application.Contract.CQS.Playlist.Commands;
using MusicBox.Application.Contract.DTO.Playlist.CreatePlaylistCopy;
using MusicBox.Core.Domain.DomainModels;
using MusicBox.Core.Handlers;
using MusicBox.Domain.DomainModels.Playlist;
using MusicBox.Domain.DomainModels.UserPlaylist;
using MusicBox.Infrastructure.UnitOfWork.Interfaces;
using ILogger = MusicBox.Abstractions.Logging.ILogger;

namespace MusicBox.Application.CQS.Playlist.Commands.Handlers;

public class CreatePlaylistCopyCommandHandler : LoggableRequestHandler-
Base<CreatePlaylistCopyCommand, OutgoingCreatePlaylistCopyDto>
{
    private readonly IUnitOfWork _unitOfWork;
    private readonly IMapper _mapper;
    private readonly List<IDomainModelBusinessRule<PlaylistDomainModel>>
_playlistDomainModelRules;
    private                                readonly
List<IDomainModelBusinessRule<UserPlaylistDomainModel>>
_userPlaylistDomainModelRules;

    public CreatePlaylistCopyCommandHandler(

```

```

        IUnitOfWork unitOfWork,
        ILogger logger,
        IHttpContextAccessor httpContextAccessor,
        IEnumerable<Tuple<DomainModelOperations, IDomainModelBusinessRule<PlaylistDomainModel>>> playlistDomainModelRules,
        IEnumerable<Tuple<DomainModelOperations, IDomainModelBusinessRule<UserPlaylistDomainModel>>> userPlaylistDomainModelBusinessRules,
        IMapper mapper)
        : base(logger, httpContextAccessor)
    {
        _unitOfWork = unitOfWork;
        _mapper = mapper;
        _playlistDomainModelRules = playlistDomainModelRules.Where(x =>
x.Item1 == DomainModelOperations.Create)
            .Select(x => x.Item2)
            .ToList();
        _userPlaylistDomainModelRules = userPlaylistDomainModelBusinessRules.Where(x => x.Item1 == DomainModelOperations.Create)
            .Select(x => x.Item2)
            .ToList();
    }

    public override async Task<IHandlerResult<OutgoingCreatePlaylistCopyDto>>
Handle(CreatePlaylistCopyCommand command, CancellationToken cancellationToken)
    {
        const string copy = " Copy";
        var playlist = await _unitOfWork.PlaylistRepository.GetAsync(command.Id,

```

```
cancellationToken);
```

```
    if (playlist is null)
```

```
    {
```

```
        Logger.Warning($"Playlist with identifier {command.Id} was not found.",  
HttpContextAccessor.HttpContext!.TraceIdentifier);
```

```
        return BadRequest<OutgoingCreatePlaylistCopyDto>();
```

```
    }
```

```
    var username = HttpContextAccessor.HttpContext?.User.Claims.Single(x =>  
x.Type is nameof(IdentityUserModel.Name)).Value;
```

```
    var user = await _unitOfWork.UserRepository.GetByNameAsync(username,  
cancellationToken);
```

```
    string? newFileName = null;
```

```
    if (playlist.TitlePath is not null)
```

```
    {
```

```
        newFileName = await CreateUniqueFileName(user.Name, playlist.Name +  
" Copy", GetSymbolsAfterLastDot(playlist.TitlePath), cancellationToken);
```

```
    }
```

```
    var playlistCreationExecutionResult = await PlaylistDomainModel.Create(  
        playlist.Name + copy,
```

```
        playlist.Description,
```

```
        newFileName is not null && playlist.TitlePath is not De-  
faults.DefaultPlaylistImagePath ? Pathes.PlaylistsImagesPath + newFileName :  
Defaults.DefaultPlaylistImagePath,
```

```
        user.Name,
```

```

        _playlistDomainModelRules);

    if (!playlistCreationExecutionResult.Success)
    {
        Logger.Error("Error occurred during Playlist creation. " + playlistCreation-
ExecutionResult.Error.Message,                                HttpContextAcces-
sor.HttpContext!.TraceIdentifier);

        return                                                    BadRe-
quest<OutgoingCreatePlaylistCopyDto>(playlistCreationExecutionResult.Error);
    }

    var                                outgoingCreatePlaylistDto                                =                                await
_unitOfWork.PlaylistRepository.Create(playlistCreationExecutionResult.Payload,
cancellationToken);

    var userPlaylistDomainModel = await UserPlaylistDomainModel.Create(
        user.Id,
        outgoingCreatePlaylistDto.Id,
        _userPlaylistDomainModelRules);

    await
_unitOfWork.UserPlaylistRepository.CreateAsync(userPlaylistDomainModel.Payl
oad, cancellationToken);

    await
_unitOfWork.PlaylistSongRepository.CopyToAnotherPlaylistAsync(playlist.Id,
outgoingCreatePlaylistDto.Id, cancellationToken);

    try
    {

```

```

        if (newFileName is not null)
        {
            var baseDirectory = RemoveAfterSrc(Environment.CurrentDirectory);
            var sourceFilePath = baseDirectory +
Pathes.PlaylistsImagesPathForSavingImage +
            GetLastSegment(playlist.TitlePath!);
            var destinationFilePath = Path.Combine(baseDirectory +
Pathes.PlaylistsImagesPathForSavingImage,newFileName);

            File.Copy(sourceFilePath, destinationFilePath);
        }
    }
    catch (Exception exception)
    {
        Logger.Error(exception, HttpContextAccessor.HttpContext!.TraceIdentifier);
        DeletePlaylistImage(newFileName);
        throw;
    }

    Logger.Info($"Playlist copy of playlist with id {command.Id} with new id
{outgoingCreatePlaylistDto.Id} was created.", HttpContextAccessor.HttpContext!.TraceIdentifier);

    return DataMapper.Map<OutgoingCreatePlaylistCopyDto>(outgoingCreatePlaylistDto));
}

private static void DeletePlaylistImage(string fileName)
{

```

```

        var baseDirectory = RemoveAfterSrc(Environment.CurrentDirectory);
        File.Delete(baseDirectory + Pathes.PlaylistsImagesPathForSavingImage +
fileName);
    }

private static string RemoveAfterSrc(string input)
{
    var index = input.IndexOf("src\\", StringComparison.Ordinal);

    if (index >= 0)
    {
        input = input[..(index + 4)];
    }

    return input;
}

private async Task<string> CreateUniqueFileName(string username, string
playlistName, string fileExtension, CancellationToken cancellationTok
en)
{
    var existingPlaylists = await
_unitOfWork.PlaylistRepository.GetPlaylistsByUsername(username, cancella
tionToken);

    var newPlaylistCount = existingPlaylists is null ? 0 : existingPlaylists.Count +
1;

    return newPlaylistCount + username + playlistName + fileExtension;
}

private string GetLastSegment(string input)

```

```

    {
        var segments = input.Split('/');
        var lastSegment = segments[^1];
        return lastSegment;
    }

    static string GetSymbolsAfterLastDot(string input)
    {
        var lastDotIndex = input.LastIndexOf('.');
        return input[lastDotIndex..];
    }
}

using Microsoft.AspNetCore.Http;
using MusicBox.Abstractions.Domain.DomainModels;
using MusicBox.Abstractions.HandlerResult;
using MusicBox.Abstractions.Logging;
using MusicBox.Application.Contract.CQS.Playlist.Commands;
using MusicBox.Application.Contract.DTO.Playlist.CreatePlaylistSong;
using MusicBox.Core.Domain.DomainModels;
using MusicBox.Core.Errors;
using MusicBox.Core.Handlers;
using MusicBox.Domain.DomainModels.PlaylistSong;
using MusicBox.Infrastructure.UnitOfWork.Interfaces;

namespace MusicBox.Application.CQS.Playlist.Commands.Handlers;

public class CreatePlaylistSongCommandHandler : LoggableRequestHandler-

```

```

Base<CreatePlaylistSongCommand, OutgoingCreatePlaylistSongDto>
{
    private readonly IUnitOfWork _unitOfWork;
    private readonly IEnumerable<Tuple<DomainModelOperations, IDomainModelBusinessRule<PlaylistSongDomainModel>>>
        _playlistSongDomainModelRules;

    public CreatePlaylistSongCommandHandler(
        IUnitOfWork unitOfWork,
        ILogger logger,
        IHttpContextAccessor httpContextAccessor,
        IEnumerable<Tuple<DomainModelOperations, IDomainModelBusinessRule<PlaylistSongDomainModel>>> playlistUserDomainModelRules)
        : base(logger, httpContextAccessor)
    {
        _unitOfWork = unitOfWork;
        _playlistSongDomainModelRules = playlistUserDomainModelRules;
    }

    public override async Task<IHandlerResult<OutgoingCreatePlaylistSongDto>>
        Handle(CreatePlaylistSongCommand command, CancellationToken cancellation)
    {
        var doesPlaylistSongAlreadyExist = await
            _unitOfWork.PlaylistSongRepository.ExistByPlaylistAndSongIdAsync(command.
                PlaylistId, command.SongId, cancellation);

        if (doesPlaylistSongAlreadyExist)
        {

```

```

        Logger.Error($"Playlist with id {command.PlaylistId} already contain song
with id {command.SongId}.", HttpContextAccessor.HttpContext!.TraceIdentifier);
        return BadRequest<OutgoingCreatePlaylistSongDto>(new Error($"Playlist
with id {command.PlaylistId} already contain song with id {com-
mand.SongId}."));
    }

```

```

        var playlistSongs = await
_unitOfWork.SongRepository.GetPlaylistSongsByPlaylistId(command.PlaylistId,
cancellationToken);

```

```

var playlistSongDomainModel = await PlaylistSongDomainModel.Create(
    command.PlaylistId,
    command.SongId,
    playlistSongs.Count + 1,
    _playlistSongDomainModelRules
    .Where(x => x.Item1 == DomainModelOperations.Create)
    .Select(x => x.Item2)
    .ToList());

```

```

if (!playlistSongDomainModel.Success)
{
    Logger.Error("Error occurred during UserSong creation. " + playlistSong-
DomainModel.Error.Message, HttpContextAcces-
sor.HttpContext!.TraceIdentifier);
    return BadRequest<OutgoingCreatePlaylistSongDto>(playlistSongDomainModel.Error);
}

```

```

        var playlistSongDto = await
_unitOfWork.PlaylistSongRepository.CreateAsync(playlistSongDomainModel.Pa
yload, cancellationToken);

```

```

        Logger.Info($"UserSong record with id {playlistSongDto.Id} was created.",
HttpContextAccessor.HttpContext!.TraceIdentifier);
        return Data(playlistSongDto);
    }
}

```

```

using AutoMapper;
using Microsoft.AspNetCore.Http;
using MusicBox.Abstractions.HandlerResult;
using MusicBox.Abstractions.Logging;
using MusicBox.Application.Contract.CQS.Playlist.Queries;
using MusicBox.Application.Contract.DTO.Playlist.GetAllPlaylists;
using MusicBox.Core.Handlers;
using MusicBox.Infrastructure.UnitOfWork.Interfaces;

```

```

namespace MusicBox.Application.CQS.Playlist.Queries.Handlers;

```

```

public class GetAllPlaylistsQueryHandler : LoggableRequestHandler-
Base<GetAllPlaylistsQuery, OutgoingGetAllPlaylistsDto>
{
    private readonly IUnitOfWork _unitOfWork;
    private readonly IMapper _mapper;

    public GetAllPlaylistsQueryHandler(
        IUnitOfWork unitOfWork,

```

```

    IMapper mapper,
    ILogger logger,
    IHttpContextAccessor httpContextAccessor) : base(logger, httpContextAccess-
sor)
    {
        _unitOfWork = unitOfWork;
        _mapper = mapper;
    }

    public override async Task<IHandlerResult<OutgoingGetAllPlaylistsDto>>
Handle(GetAllPlaylistsQuery query, CancellationToken cancellationToken)
    {
        var playlists = await
        _unitOfWork.PlaylistRepository.GetAllAsync(cancellationToken);

        if (playlists.Any())
        {
            Logger.Info("Playlists with ids " + $"{string.Join(", ", playlists.Select(m
=> m.Id).ToList())} were received.", HttpContextAcces-
sor.HttpContext!.TraceIdentifier);
        }

        return Data(new OutgoingGetAllPlaylistsDto(playlists.Select(x =>
        _mapper.Map<OutgoingGetAllPlaylistsSinglePlaylistDto>(x)).ToList()));
    }
}

using Microsoft.AspNetCore.Http;
using MusicBox.Abstractions.HandlerResult;
using MusicBox.Abstractions.Logging;

```

```

using MusicBox.Application.Contract.CQS.Playlist.Queries;
using MusicBox.Application.Contract.DTO.Playlist.GetPlaylistById;
using MusicBox.Core.Handlers;
using MusicBox.Infrastructure.UnitOfWork.Interfaces;

namespace MusicBox.Application.CQS.Playlist.Queries.Handlers;

public class GetPlaylistByIdQueryHandler : LoggableRequestHandler-
Base<GetPlaylistByIdQuery, OutgoingGetPlaylistByIdDto>
{
    private readonly IUnitOfWork _unitOfWork;

    public GetPlaylistByIdQueryHandler(
        IUnitOfWork unitOfWork,
        ILogger logger,
        IHttpContextAccessor httpContextAccessor) : base(logger, httpContextAcces-
sor)
    {
        _unitOfWork = unitOfWork;
    }

    public override async Task<IHandlerResult<OutgoingGetPlaylistByIdDto>>
Handle(GetPlaylistByIdQuery query, CancellationToken cancellationToken)
    {
        var playlist = await
_unitOfWork.PlaylistRepository.GetPlaylistWithSongsById(query.Id, cancella-
tionToken);

        if (playlist == null)

```

```

    {
        Logger.Info($"Playlist with id {query.Id} not found.", HttpContextAccessor.HttpContext!.TraceIdentifier);
        return NotFound<OutgoingGetPlaylistByIdDto>(new Errors.Playlist.NotFoundError());
    }

```

```

    Logger.Info($"Playlist with id {playlist.Id} was received.", HttpContextAccessor.HttpContext!.TraceIdentifier);
    return Data(new OutgoingGetPlaylistByIdDto(
        playlist.Id,
        playlist.PlaylistSongs.Count,
        playlist.PlaylistSongs?.Select(x => x.Song.DurationSec).Sum(),
        playlist.TitlePath,
        playlist.Name,
        playlist.Description,
        playlist.Author,
        playlist.IsPublic));
}
}

```

```

import { FastAverageColor } from 'fast-average-color';
import { useEffect, useRef, useState } from 'react';
import { Link, useParams } from 'react-router-dom';
import { ElipsisHorisontal, PlayArtistPage, Time } from '../assets';
import { useAppContext } from '../contexts/MusicContext';
import { useAppDispatch, useAppSelector } from '../hooks/hooks';
import { fetchAlbumById } from '../hooks/reducers/albumsActions/albumDetail.slice';

```

```

import { formatDateWithFullNames, getYearFromDate } from
'../../utils/formatDate';

import { fetchAlbumSongsById } from
'../../hooks/reducers/songActions/albumSongs.slice';

import { fetchUserPlaylists } from
'../../hooks/reducers/playlistActions/userPlaylists.slice';

import { getUserAlbumDetails } from
'../../hooks/reducers/userActions/getUserAlbumDetails.slice';

import { deleteUserAlbum } from
'../../hooks/reducers/userActions/deleteUserAlbum.slice';

import { createUserAlbum } from
'../../hooks/reducers/userActions/createUserAlbum.slice';

import { getSongsDuration, getSortedSongsByOrder } from
'../../utils/songsFormatter';

import { Song } from '../../models/Song';
import PressedLike from '../../assets/Icons/PressedLike';
import Like from '../../assets/Icons/Like';
import styles from './AlbumDetail.module.scss';
import SongListItem from '../../components/SongListItem/SongListItem';
import SongItemContextMenu from
'../../components/ContextMenus/SongItemContextMenu/SongItemContextMenuPr
ops';
import UnitContextMenu from
'../../components/ContextMenus/UnitContextMenu/UnitContextMenu';

const AlbumDetail = () : JSX.Element => {
  const { setSongs, setCurrentSong, currentSong, isPlayerShown } = useAppCon-
text();
  const { id } = useParams<{ id: undefined }>();

```

```

const coverRef = useRef<HTMLImageElement | null>(null);
const [windowDimensions, setWindowDimensions] = use-
eState(getWindowDimensions());
const [contextMenuSongInfo, setContextMenuSongInfo] = useState({
  name: "",
  songId: 0,
  artistId: 0,
  albumId: 0,
  isLiked: false,
  albumTitlePath: "",
  audioPath: "",
  artistName: ""
});
const [contextMenuPosition, setContextMenuPosition] = useState({ x: 0, y: 0 });
const [isContextMenuShown, setIsContextMenuShown] = useState(false);

const [isAlbumContextMenuShown, setIsAlbumContextMenuShown] = us-
eState(false);
const [albumContextMenuPosition, setAlbumContextMenuPosition] = useState({
x: 0, y: 0 });

const dispatch = useAppDispatch();
var {
  userPlaylists: { userPlaylists },
  albumDetail: { album, loading },
  albumSongs: { songs },
  getUserAlbumDetails: { userAlbumDetails },
} = useAppSelector((state) => state);

```

```

const showContextMenu = (event: React.MouseEvent<HTMLDivElement>) => {
  event.preventDefault();

  setIsContextMenuShown(false);
  const newPosition = {
    x: event.pageX,
    y: event.clientY,
  };

  if (windowDimensions.x - newPosition.x < 254) {
    newPosition.x = newPosition.x - 240;
  }
  if (windowDimensions.y - newPosition.y < 160) {
    newPosition.y = newPosition.y - 150;
  }

  setContextMenuPosition(newPosition);
  setIsContextMenuShown(true);
};

const showAlbumContextMenu = () => {
  const contextMenu = document.getElementById("AlbumOptions");

  setIsAlbumContextMenuShown(false);
  const newPosition = {
    x: contextMenu!.getBoundingClientRect().x! + 25,
    y: contextMenu!.getBoundingClientRect().y! + 45,
  };
};

```

```
setAlbumContextMenuPosition(newPosition!);  
setIsAlbumContextMenuShown(!isAlbumContextMenuShown);  
};
```

```
const hideContextMenu = () => {  
  setIsContextMenuShown(false);  
};
```

```
const hideAlbumContextMenu = () => {  
  setIsAlbumContextMenuShown(false);  
};
```

```
const handleLikeButtonClick = async () => {  
  if(userAlbumDetails?.isUserLiked != null){  
    if(userAlbumDetails.isUserLiked === true) {  
      await dispatch(deleteUserAlbum(id!));  
      handleAlbumDetailsUpdate();  
    }  
    else {  
      await dispatch(createUserAlbum(id!));  
      handleAlbumDetailsUpdate();  
    }  
  }  
};
```

```
const handleSongListUpdate = async () => {  
  await dispatch(fetchAlbumSongsById(id!));  
};
```

```
}
```

```
const handleAlbumDetailsUpdate = async () => {  
  await dispatch(getUserAlbumDetails(id!));  
}
```

```
function getWindowDimensions() {  
  const { innerWidth: x, innerHeight: y } = window;  
  return {  
    x,  
    y  
  };  
}
```

```
useEffect(() => {  
  function handleResize() {  
    setWindowDimensions(getWindowDimensions());  
  }  

```

```
  window.addEventListener('resize', handleResize);  
  return () => window.removeEventListener('resize', handleResize);  
}, []);
```

```
useEffect(() => {  
  if (coverRef.current !== null) {  
    coverRef.current.crossOrigin = 'Anonymous';  
    new FastAverageColor()  
      .getColorAsync(coverRef.current)
```

```

.then((color) => {
  const background = document.getElementById('Background');
  const playlistBackground =
    document.getElementById('PlaylistBackground');
  if (background !== null) {
    background.style.backgroundColor = color.rgb;
  }
  if (playlistBackground !== null) {
    playlistBackground.style.backgroundColor = color.rgb;
  }
})
.catch((err) => {
  console.log(err);
}));
}, [album]);

```

```

useEffect(() => {
  if (id !== null) {
    dispatch(fetchAlbumById(id));
    dispatch(fetchAlbumSongsById(id));
    dispatch(fetchUserAlbumDetails(id));
    dispatch(fetchUserPlaylists());
  }
}, [dispatch, id]);

```

```

return (

```

```

<>

```

```

{!loading && (
  <>
  {album != null && (
    <div className={styles.PlaylistDetail}>
      <div className={styles.Cover} onClick={() => {
        hideContextMenu();
        hideAlbumContextMenu();
      }}>
        <div className={styles.Background} id="Background" />
        <div className={styles.Gradient} />
        <div className={styles Img}>
          <img
            src={"/" + album.coverPath}
            alt="Album img"
            ref={coverRef}
          />
        </div>
        <div className={styles.Infos} onClick={() => {
          hideContextMenu();
          hideAlbumContextMenu();
        }}>
          <div className={styles.Playlist}>{album.albumType}</div>
          <div className={styles.Title}>
            <h1>{album.name}</h1>
          </div>
          <div className={styles.Details}>
            <span className={styles.Text_Bold}>
              <Link to={` /artist/${album.artistId}`}>

```

```

        {album.artistName}
      </Link>
    </span>
    <span className={styles.Text_Light}>
      {getYearFromDate(album.releaseDate)}
    </span>
    <span className={styles.Text_Light}>
      {songs?.length === undefined ? "" : songs.length} songs, {" "}
      {getSongsDuration(songs)}
    </span>
  </div>
</div>
</div>
</div>

<div className={styles.List_Background} id="PlaylistBackground" />
<div className={styles.Buttons_Section} onClick={() => {
  hideContextMenu();
}}>

<UnitContextMenu
  contextMenuItemsOptions={{
    isAddToPlaylistIncluded: true,
    isAddToQueueIncluded: true,
  }}
  album={{
    albumId: album.id,
    artistId: album.artistId,
    isLiked: userAlbumDetails?.isUserLiked!,
  }}

```

```

        songs : songs
    }}
    playlist={undefined}
    artist={undefined}
    isShown={isAlbumContextMenuShown}
    position={albumContextMenuPosition}
    playlists={userPlaylists}
    hideContextMenu={hideAlbumContextMenu}
    handleUpdate={handleAlbumDetailsUpdate}
/>

<div
    className={styles.PlayAlbum_Button}
    onClick={() => {
        setSongs(songs!);
        setCurrentSong(songs!.find((x) => x.order === 1)!);
    }}
>
    <PlayArtistPage />
</div>

<div
    className={styles.Like_Button}
    onClick={handleLikeButtonClick}
>
    {userAlbumDetails?.isUserLiked ? <PressedLike /> : <Like />}
</div>

<div className={styles.Elipses} onClick={() => {
    showAlbumContextMenu();

```

```

        hideContextMenu();
    }} onContextMenu={(event: React.MouseEvent<HTMLDivElement>)
=> event.preventDefault()} id="AlbumOptions">
        <ElipsisHorizontal />
    </div>
</div>
<div className={styles.List} onClick={hideAlbumContextMenu}>
    <div className={styles.Heading_Sticky}>
        <div className={styles.Heading}>
            <div onClick={() => {
                hideContextMenu();
                hideAlbumContextMenu();
            }}>#</div>
            <div onClick={() => {
                hideContextMenu();
                hideAlbumContextMenu();
            }}>Title</div>
            <div onClick={() => {
                hideContextMenu();
                hideAlbumContextMenu();
            }}>Plays</div>
            <div className={styles.Length} onClick={() => {
                hideContextMenu();
                hideAlbumContextMenu();
            }}>
                <Time />
            </div>
        </div>
    </div>

```

```
</div>
```

```
<SongItemContextMenu  
  contextMenuItemsOptions={{  
    isRemoveFromPlaylistIncluded: true,  
    isGoToArtistIncluded: true,  
    isGoToAlbumIncluded: false  
  }}  
  song={contextMenuSongInfo}  
  isShown={isContextMenuShown}  
  position={contextMenuPosition}  
  playlists={userPlaylists}  
  hideContextMenu={hideContextMenu}  
  handleSongsListUpdate={handleSongListUpdate}  
>
```

```
{getSortedSongsByOrder(songs).map((item: Song, index: number) => (  
  <SongListItem  
    key={item.order}  
    song={item}  
    index={index}  
    current={item.id === currentSong.id}  
    songListColumnOptions={{  
      isAlbumCoverIncluded: false,  
      isDateAddedIncluded: false,  
      isNumberOfPlaysIncluded: true,  
      isArtistNameIncluded: true,  
      isAlbumNameIncluded: false,  

```

```

        includeMoreThenThreeColumns: false
      }}
      setSongs={() => setSongs(songs!)}
      setCurrentSongId={() => setCurrentSong(item)}
      setContextMenuSongIndex={() =>
        setContextMenuSongInfo({
          name: item.name,
          songId: item.id,
          artistId: item.artistId,
          albumId: item.albumId,
          isLiked: item.isLiked,
          albumTitlePath: item.albumCoverPath,
          audioPath: item.audioPath,
          artistName: item.artistName
        })
      }
      hideContextMenu={hideContextMenu}
      showContextMenu={showContextMenu}
      handleSongsListUpdate={handleSongListUpdate}
    />
  )))}
</div>

<div className={styles.AlbumCredits}>
  {formatDateWithFullNames(album.releaseDate)}
  <br />
  <div className={styles.CredsText}>{album.credits}</div>
</div>

{isPlayerShown === true ? (

```

```

        <div className={styles.PlayerArea} />
      ) : (
        null
      )}
    </div>
  )}
</>
)}
</>
);
};

```

```
export default AlbumDetail;
```

```
import styles from "../Player.module.scss";
```

```
import {
```

```
  Play,
```

```
  Pause,
```

```
  SkipBack,
```

```
  SkipForward,
```

```
  RandomMusicsTrue,
```

```
  RandomMusicsFalse,
```

```
  VolumeOff,
```

```
  VolumeOn,
```

```
  RepeatOn,
```

```
  RepeatOff,
```

```
} from "../assets";
```

```
import { useCallback, useEffect, useRef, useState } from "react";
```

```
import { useAppContext } from "../../contexts/MusicContext";
import { useAppDispatch } from "../../hooks/hooks";
import { createUserListen } from
"../../hooks/reducers/userActions/createUserListen.slice";
import { Link } from "react-router-dom";
```

```
const Player = (): JSX.Element => {
  const dispatch = useAppDispatch();
  const [isPlaying, setIsPlaying] = useState<boolean>(true);
  const [isListenCounted, setIsListenCounted] = useState<boolean>(false);
  const [volume, setVolume] = useState<string>("0.5");
  const [duration, setDuration] = useState<number>(0);
  const [isRandom, setIsRandom] = useState<boolean>(false);
  const [isOnRepeat, setIsOnRepeat] = useState<boolean>(false);
  const [currentTime, setCurrentTime] = useState<number>(0);
  const [isMuted, setIsMuted] = useState<boolean>(false);
  var { songs, currentSong, setCurrentSong, setPlayerVisibility } =
    useAppContext();
```

```
const audioTag = useRef<any>(null);
const progressBar = useRef<any>(null);
const animationRef = useRef<any>(null);
```

```
const resetAudioToStart = useCallback(() => {
  if(audioTag.current.currentTime){
    setIsPlaying(false);
    setCurrentTime(0);
    audioTag.current.currentTime = 0;
```

```

    progressBar.current.value = 0;
    setIsPlaying(true);
    audioTag.current.play();
  }
}, []);

```

```

const calculateDuration = (sec: number) => {
  const minutes = Math.floor(sec / 60);
  const newMinutes = minutes < 10 ? `0${minutes}` : `${minutes}`;
  const seconds = Math.floor(sec % 60);
  const newSeconds = seconds < 10 ? `0${seconds}` : `${seconds}`;

  return `${newMinutes}:${newSeconds}`;
};

```

```

const skipRandom = useCallback(() => {
  if (songs.length !== 1) {
    const randomNum = Math.floor(Math.random() * songs.length) + 1;
    if (randomNum === currentSong.order) {
      skipRandom();
    } else {
      setCurrentSong(songs.find((x) => x.order === randomNum)!);
    }
  }
}, [currentSong, songs, setCurrentSong]);

```

```

const skipForward = useCallback(() => {
  if (songs.length !== 1) {

```

```

    if (isRandom) {
      skipRandom();
    } else {
      const idNum = currentSong.order;
      const newId = idNum + 1;
      if (newId > songs.length) {
        setCurrentSong(songs.find((x) => x.order === 1!));
      } else {
        setCurrentSong(songs.find((x) => x.order === newId!));
      }
    }
  }
}, [currentSong, isRandom, songs, skipRandom, setCurrentSong]);

const repeatSong = useCallback(() => {
  if (songs.length === 1 || isOnRepeat) {
    setCurrentSong(currentSong);
    resetAudioToStart();
  }
}, [isOnRepeat, songs, currentSong, setCurrentSong, resetAudioToStart]);

const skipBack = useCallback(() => {
  if (songs.length !== 1) {
    if (!isRandom) {
      const idNum = currentSong.order;
      const newId = idNum - 1;
      if (newId === 0) {
        setCurrentSong(songs.find((x) => x.order === songs.length!));
      }
    }
  }
}, [currentSong, isRandom, songs, skipRandom, setCurrentSong]);

```

```

    } else {
      setCurrentSong(songs.find((x) => x.order === newId!));
    }
  } else {
    skipRandom();
  }
}
}, [currentSong, isRandom, songs, setCurrentSong, skipRandom]);

```

```

const whilePlaying = useCallback(() => {
  progressBar.current.value = audioTag?.current?.currentTime;
  animationRef.current = requestAnimationFrame(whilePlaying);
  changeCurrentTime();
}, []);

```

```

const changeRange = () => {
  audioTag.current.currentTime = progressBar.current.value;
  changeCurrentTime();
};

```

```

const changeCurrentTime = () => {
  setCurrentTime(progressBar.current.value);
};

```

```

useEffect(() => {
  if (currentSong.order !== 0) {
    if (isPlaying) {
      audioTag.current.play();
    }
  }
}, [currentSong, isPlaying]);

```

```

animationRef.current = requestAnimationFrame(whilePlaying);
audioTag.current.volume = volume;

if (isMuted) {
  audioTag.current.muted = true;
} else audioTag.current.muted = false;

const interval = setInterval(() => {
  const seconds = Math.floor(audioTag.current.duration);
  setDuration(seconds);
  progressBar.current.max = seconds;
}, 1000);

setInterval(() => {
  if (duration > 0 || duration !== undefined) {
    clearInterval(interval);
    if (audioTag.current.currentTime === audioTag.current.duration) {
      if (
        isListenCounted === false &&
        currentSong.order !== 0 &&
        audioTag.current.currentTime === audioTag.current.duration
      ) {
        setIsListenCounted(true);
        void dispatch(
          createUserListen(currentSong.id)
        );
        setIsListenCounted(false);
      }
    }
  }
}

```

```

        if (songs.length === 1 || isOnRepeat) {
            repeatSong();
        } else {
            isRandom ? skipRandom() : skipForward();
        }
    }
}
}, 1100);
} else {
    audioTag.current.pause();
    audioTag.current.volume = volume;
    cancelAnimationFrame(animationRef.current);
}
}
}, [
    songs,
    currentSong,
    duration,
    isMuted,
    isPlaying,
    isListenCounted,
    isRandom,
    isOnRepeat,
    volume,
    whilePlaying,
    skipBack,
    skipForward,
    skipRandom,

```

```

    repeatSong,
    dispatch,
  ]);

  if (songs.length !== 0) {
    setPlayerVisibility(true);
    return (
      <div className={styles.Container}>
        <div className={styles.musicDiv}>
          {songs.map((music) =>
            currentSong.order === music.order ? (
              <>
                {
                  <>
                    <img
                      src={"/" + music.albumCoverPath}
                      alt={music.albumName}
                    />
                    <div className={styles.SongInfo}>
                      <h1>{music.name}</h1>
                      <Link
                        className={styles.artistLink}
                        to={` /artist/${music.artistId}`}
                      >
                        <h3>{music.artistName}</h3>
                      </Link>
                    </div>
                  </>
                }
              </>
            ) : null
          )}
        </div>
      </div>
    );
  }

```

```

    }
    <audio src={"/" + music.audioPath} ref={audioTag} />
  </>
) : null
)}
</div>
<div className={styles.player}>
  <div className={styles.inputButtons}>
    {
      <div className={styles.progressBar}>
        <p className={styles.PcurrentTime}>
          {calculateDuration(currentTime)}
        </p>
        <input
          type="range"
          className={styles.currentProgress}
          defaultValue="0"
          ref={progressBar}
          onChange={changeRange}
        />

        <p className={styles.Pduration}>
          {calculateDuration(duration)}
        </p>
      </div>
    }
  <div className={styles.buttons}>
    <button

```

```

        onClick={() => setIsRandom(!isRandom)}
        className={styles.randomMusicsButton}
    >
        {isRandom ? <RandomMusicsTrue /> : <RandomMusicsFalse />}
    </button>
    <button onClick={skipBack}>
        <SkipBack />
    </button>
    <button
        className={styles.playPause}
        onClick={() => setIsPlaying(!isPlaying)}
    >
        {isPlaying ? <Pause /> : <Play />}
    </button>
    <button onClick={skipForward}>
        <SkipForward />
    </button>
    <button
        onClick={() => {setIsOnRepeat(!isOnRepeat)}}
        className={styles.onRepeatButton}
    >
        {isOnRepeat ? <RepeatOn /> : <RepeatOff />}
    </button>
</div>
</div>
</div>

{

```

```

    <div className={styles.test}>
      <button
        className={styles.volumeButton}
        onClick={() => setIsMuted(!isMuted)}
      >
        {isMuted ? <VolumeOff /> : <VolumeOn />}
      </button>
      <input
        type="range"
        step="0.01"
        onChange={(e) => setVolume(e.target.value)}
        value={volume}
        max="1"
        min="0"
      />
    </div>
  }
</div>

);
} else {
  return <></>;
}
};

export default Player;

```