

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

ПОЯСНЮВАЛЬНА ЗАПИСКА

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

«Музичний стримінговий сервіс»

Виконала студентка групи ІПЗ-20-2
БІЛЬКЕВИЧ Дар'я Володимирівна

Керівник роботи:
ЄФРЕМОВ Юрій Миколайович

Рецензент:
КОНЦИДАЙЛО Андрій Михайлович

Житомир – 2024

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

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна ВАКАЛЮК

«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **БІЛЬКЕВИЧ Дар'я Володимирівна**

Керівник роботи: **ЄФРЕМОВ Юрій Миколайович**

Тема роботи: **«Музичний стримінговий сервіс»,**

затверджена Наказом закладу вищої освіти від **«23» лютого 2024 р., №74с.**

Вихідні дані для роботи: **процес створення та реалізації музичного стримінгового сервісу як програмного продукту, зокрема використання веб-орієнтованих технологій для створення музичного стримінгового сервісу.**

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Єфремов Ю.М.	27.02.2024	27.02.2024
2	Єфремов Ю.М.	27.02.2024	27.02.2024
3	Єфремов Ю.М.	27.02.2024	27.02.2024

РЕФЕРАТ

Випускна кваліфікаційна робота складається з веб-додатку для музичного стримінгового сервісу та пояснювальної записки. Пояснювальна записка до випускної роботи містить текстову частину, викладену на 72 сторінках друкованого тексту. Список використаних джерел містить 25 найменувань і займає 2 сторінки. У роботі наведено 50 ілюстрацій та 9 таблиць. Також у роботі є 19 сторінок додатків. Загальний обсяг роботи – 91 сторінки.

Головною метою є створення зручного і сучасного інструменту, який надасть користувачам можливість насолоджуватися музикою в онлайн-режимі, впорядковувати її за власним смаком, взаємодіяти з музичним контентом, а також надати функціональність для ведення власного музичного простору.

У ході роботи було проведено аналіз напрямків використання інформаційних технологій, встановлено постановку задачі та проаналізовано існуючі програмні продукти відносно музичного стримінгового сервісу. В результаті було обрано архітектуру системи, що відповідає вимогам проекту. Далі було визначено варіанти використання та побудовано об'єктно-орієнтовану структуру системи. Здійснено розробку баз даних та реалізацію алгоритмів роботи системи, включаючи функціонал інтерфейсу та налаштування параметрів сервісу. Всі етапи проекту були детально протестовані для забезпечення стабільності та надійності платформи, що підтверджено висновками третього розділу з роботи.

КЛЮЧОВІ СЛОВА: СЕРВІС, СТРИМІНГ, МІКРОСЕРВІС, АУДИОФАЙЛ, ПЛЕЙЛИСТА, NATS.

					ІПЗ.КР.Б-121-24-ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Музичний стримінговий сервіс Пояснювальна записка	Літ.	Арк.	Аркушів
Розроб.		Бількевич Д.В.					3	91
Керівник		Єфремов Ю.М.				Житомирська політехніка, група ІПЗ-20-2		
Н. контр.		Концидайло А.М.						
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

The bachelor's thesis consists of a web application for a music streaming service and an explanatory note. The explanatory note for the thesis contains the textual part, presented in 72 pages of printed text. The list of references includes 25 titles and occupies 3 pages. The work contains 50 illustrations and 9 tables. There are also 19 pages of appendices in the work. The total volume of the work is 91 pages.

The main purpose is to create a convenient and modern tool that gives users the opportunity to sweeten music online, organize it according to their own taste, interact with musical content, and also provide functionality for managing their own music space.

During the work, an analysis of information technology usage trends was conducted, the problem statement was established, and existing software products related to the music streaming service were analyzed. As a result, an architecture of the system that meets project requirements was selected. Subsequently, usage scenarios were identified, and an object-oriented structure of the system was developed. The development of databases and implementation of system operation algorithms, including interface functionality and service parameter configuration, were carried out. All project stages were thoroughly tested to ensure platform stability and reliability, as confirmed by the conclusions of the third chapter of the work.

KEYWORDS: SERVICE, STREAMING, MICROSERVICE, AUDIO FILE, PLAYLIST, NATS.

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ІІЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ.....	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1 АНАЛІЗ НАПРЯМІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ	7
1.1 Постановка задачі	9
1.2 Аналіз аналогів.....	10
1.3 Вибір архітектури сервісу	17
1.4 Обґрунтування вибору засобів реалізації та вимоги до апаратного забезпечення.....	19
Висновки до першого розділу.....	20
РОЗДІЛ 2 ПРОЄКТУВАННЯ СЕРВІСУ	22
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи	22
2.2 Розробка бази даних системи.....	28
2.3 Проєктування та реалізація алгоритмів системи	34
2.4 Реалізація функціоналу системи	38
Висновки до другого розділу.....	48
РОЗДІЛ 3 ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СЕРВІСОМ.....	49
3.1 Порядок встановлення та налаштування параметрів сервісу.....	49
3.2 Структура інтерфейсу сервісу	50
3.3 Тестування сервісу	63
Висновки до третього розділу	66
ВИСНОВКИ	68
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	70
ДОДАТКИ.....	72

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних

ПЗ – програмне забезпечення

СУБД – система управління базами даних

API – Application Programming Interface (Прикладний програмний інтерфейс)

URL – Uniform Resource Locator (Уніфікований локатор ресурсу)

HTTPS – HyperText Transfer Protocol Secure (Захищений протокол передачі гіпертексту)

JSON – JavaScript Object Notation (Запис об'єктів JavaScript)

NATS – No Automatic Transport System (Система без автоматичного транспортування)

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

Актуальність теми. У сучасному світі, де технології стрімко розвиваються, створення музичного стримінгового сервісу виявляється актуальним і стратегічно важливим завданням. Споживачі сьогодні цінують негайний та миттєвий доступ до різноманітного музичного контенту, а музичні стримінгові платформи є відповіддю на це зростаюче прагнення. Розвиток пристроїв і підвищення їхньої популярності роблять акцент на можливості прослуховування в онлайн-режимі з будь-якого місця надзвичайно важливим для користувачів.

Крім того, музичні стримінгові сервіси є платформою для співпраці музикантів, що сприяє їхньому просуванню, розвитку і залученню нових шарів аудиторії. Така платформа стимулюватиме підтримку та співпрацю між людьми, незалежно від їхнього місця перебування, що буде спрощувати життя та допомагатиме зберегти важливі соціальні зв'язки, навіть коли вони фізично неможливі.

Мета даної роботи полягає в розробці та впровадженні програмного продукту – музичного стримінгового сервісу. Основний акцент буде зроблено на створенні зручної та ефективної платформи для користувачів, забезпечуючи їм швидкий та персоналізований доступ до різноманітного музичного контенту. Також буде враховано впровадження функціональних можливостей для можливості популяризації власних файлів музикантами.

Встановлене мета обумовлює наступні **завдання**:

- аналіз ринку та споживчого попиту;
- вибір технологічного стеку та архітектури сервісу;
- проектування інтерфейсу користувача;
- проектування структурних складових та алгоритмів роботи додатку;
- розробка та впровадження баз даних для зберігання інформації про користувачів, аудіофайли, плейлисти та статистику;
- тестування та оптимізація продукту.

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

Об’єктом дослідження є процес створення та реалізації музичного стримінгового сервісу як програмного продукту. Дослідження охоплюватиме технічні аспекти розробки, взаємодію користувачів із платформою, а також можливості оптимізації та вдосконалення функціональності для задоволення різноманітних потреб аудиторії.

Предметом дослідження є використання веб-орієнтованих технологій для створення музичного стримінгового сервісу.

В роботі використано такі **методи дослідження**:

- системний аналіз: вивчення та аналіз вимог, моделювання взаємодії;
- об’єктно-орієнтоване проектування: визначення об’єктів та класів, розробка архітектури;
- веб-програмування: реалізація серверної та клієнтської частин, забезпечення масштабованості;
- дизайн: створення користувацького інтерфейсу, розробка дизайну за сучасними тенденціями.

РОЗДІЛ 1 АНАЛІЗ НАПРЯМІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

1.1 Постановка задачі

Назва системи. Музичний стримінговий сервіс.

Призначення системи. Система розроблена для надання користувачам можливості отримувати доступ до широкого асортименту музичного контенту через глобальну мережу Інтернет.

Мета. Надання можливості стримінгового прослуховування музики в режимі реально часу. Сервіс дозволяє користувачам створювати власні плейлисти та редагувати їх, отримувати персональну аналітку по прослуховуванням та шукати музичний контент за власними вподобаннями. Також наявна можливість завантажувати власні файли на стримінгову платформу для популяризації власного контенту.

Вхідні дані. Користувацькі, музичні та технічні дані.

Задачі, які потрібно вирішити в процесі опрацювання роботи та реалізації програмного комплексу.

1. Аналіз вимог:

1.1. Провести детальний аналіз вимог користувачів до функціоналу музичного стримінгового сервісу.

1.2. Визначити ключові елементи інтерфейсу та функції, які забезпечать зручне та привабливе користувацьке взаємодію.

2. Вибір технічного стеку:

2.1. Обрати оптимальний стек технологій для розробки веб-додатку з урахуванням швидкодії, масштабованості та підтримки реактивного підходу.

2.2. Визначити та обґрунтувати використання React для фронтенду, Next.js для серверного рендерінгу та Node.js для розробки серверної логіки.

3. Створення серверної логіки та API:

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

- 3.1. Реалізувати серверну частину додатку на базі Node.js та Express.js для забезпечення надійного та швидкого API.
- 3.2. Взаємодія з базою даних PostgreSQL для ефективного зберігання метаданих про музичний контент.
4. Розробка інтерфейсу користувача:
 - 4.1. Створити динамічний та зручний інтерфейс веб-додатку, використовуючи бібліотеку React.
 - 4.2. Забезпечити користувачам можливість легкого навігування, пошуку та відтворення музичних композицій.
5. Імплементація функцій рекомендацій:
 - 5.1. Реалізувати алгоритми рекомендацій, що базуються на історії прослуховування користувачів та аналізі їхніх уподобань.
 - 5.2. Забезпечити персоналізовані рекомендації та можливість відкриття нових творів.
6. Тестування та оптимізація продукту:
 - 6.1. Провести ретельне тестування всіх функціональних компонентів та взаємодій.
 - 6.2. Здійснити оптимізацію для забезпечення високої продуктивності та ефективності роботи додатку.

Результатом виконання плану дій є створення розширеного інтерактивного музичного стримінгового сервісу, який відповідає потребам користувачів та надає їм зручний і привабливий інтерфейс для насолоди музикою. Завдяки вибору оптимального технічного стеку та розробці ефективної серверної логіки та API, сервіс забезпечує надійну та швидку роботу усіх його компонентів.

1.2 Аналіз аналогів

Музичні стримінгові сервіси в сучасному світі є невід’ємною частиною культури споживання музики. З метою створення високоякісного та конкурентоздатного продукту в сфері музичного стримінгу, необхідно

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		10

провести докладний аналіз аналогів. У цьому розділі роботи розглядається три провідних сервіси, кожен з них має свої особливості, переваги та недоліки, які визначатимуть стратегію розвитку нового програмного продукту.

Аналіз буде виявляти важливі аспекти, такі як функціональність, інтерфейс, кількість музичного контенту, якість відтворення аудіо, а також можливості користувачів слідкувати за статистикою своїх прослуховувань. Розгляд аналогів допоможе визначити ключові пункти для створення власного продукту.

Spotify

Spotify [1] – це шведський музичний стримінговий сервіс, запущений у 2008 році (рис. 1.1). Сервіс надає доступ до мільйонів треків в різних жанрах, включаючи музичні альбоми, підкасти та аудіокниги. Spotify пропонує два типи підписок: безкоштовний, з рекламними перервами, та платний, який включає безрекламний доступ та інші функції. Сервіс також славиться своєю активною спільнотою.

Артисти можуть завантажити свою музику за допомогою платформи “Spotify for Artists”. Після входу в даний акаунт, артистам надається можливість додавання нових треків, альбомів чи плейлистів.

Переваги:

- широкий вибір контенту, який надає доступ до мільйонів треків, підкастів, аудіокниг;
- зручний інтерфейс та можливість створення та поширення плейлистами;
- кросплатформенність – можливість використання на різних пристроях;
- наявність особистого звіту «Wrapped» з річною статистикою прослуховувань;
- швидке оновлення музичної бібліотеки.

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		11

Недоліки:

- обмеження для безкоштовних користувачів, такі як обов’язкова наявність реклами та обмеження пропускної здатності треків;
- деякі треки та альбоми можуть бути недоступні в певних регіонах;
- зростаюча конкуренція з боку інших стримінгових сервісів.

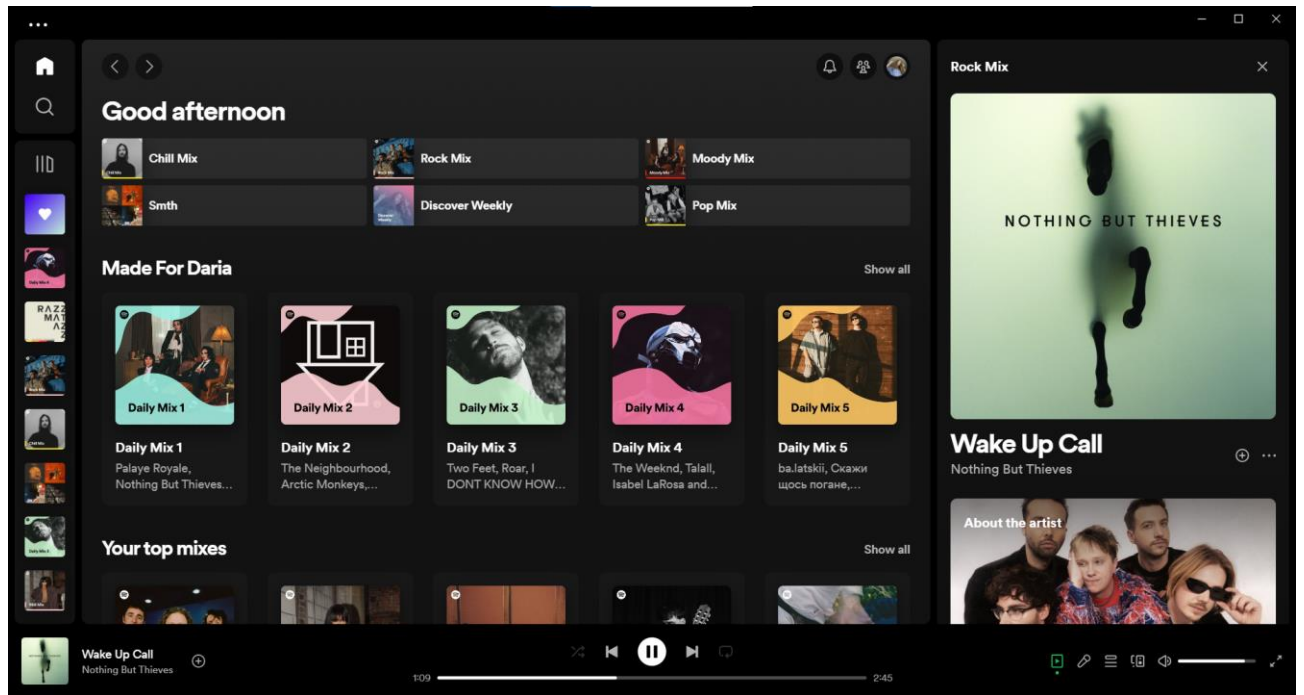


Рис. 1.1 – Інтерфейс Spotify

Apple Music

Apple Music [2] – це музичний сервіс, розроблений компанією Apple, який був запущений у 2015 році (рис. 1.2). Сервіс інтегрується з екосистемою Apple, що дозволяє користувачам слухати музику, включаючи ексклюзивний контент, та зберігати власну музичну бібліотеку. Apple Music відзначається ексклюзивним контентом та якісною звуковою підтримкою, має доступ до музичної бібліотеки iTunes.

Apple Music також має інтерфейс для артистів під назвою “Apple Music for Artists”. Тут артисти можуть додавати нові альбоми та треки, а також відстежувати відгуки та аналіз слухань своєї музики.

Переваги:

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		12

- ексклюзивний контент та можливість раннього доступу до нових релізів;
- інтеграція з іншими сервісами та пристроями Apple, такими як Siri та HomePod;
- синхронізований доступ до музики на різних пристроях Apple;
- особистий звіт “Replay” з інформацією про найбільш прослуховані треки протягом року.

Недоліки:

- інтерфейс може виглядати менш інтуїтивним для новачків порівняно з іншими стримінговими сервісами;
- обмежена кількість безкоштовного контенту для користувачів;
- обмежена сумісність з пристроями від інших виробників;
- вища вартість підписки порівняно з іншими сервісами.

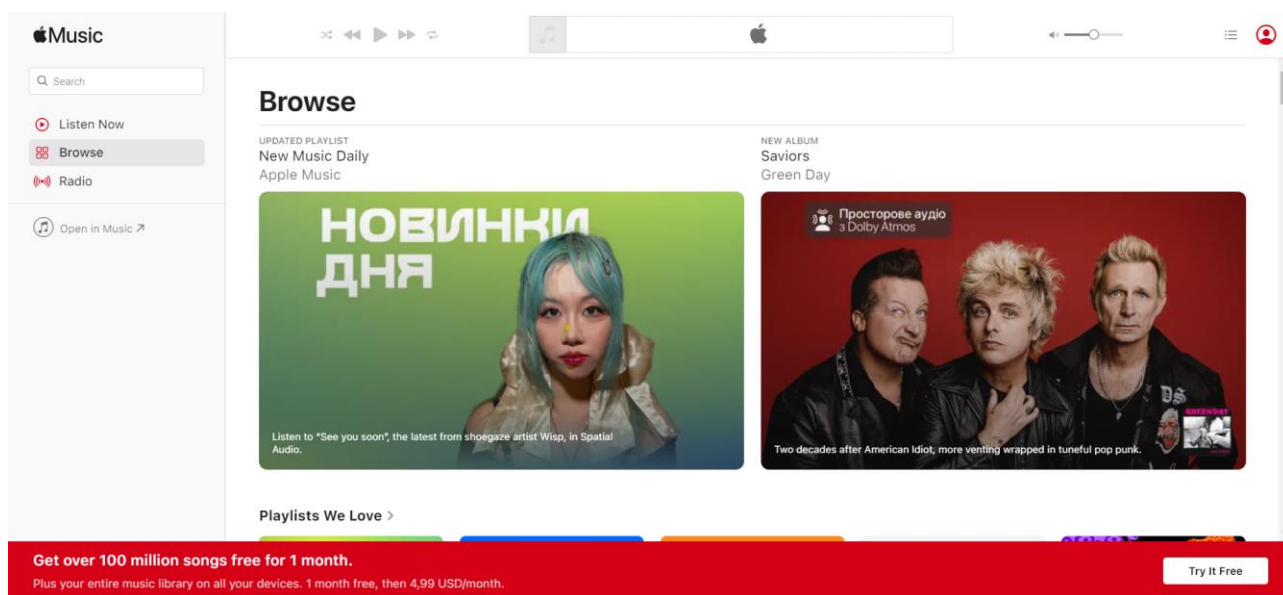


Рис. 1.2 – Інтерфейс Apple Music

YouTube Music

YouTube Music [3] – це музичний стримінговий сервіс, що базується на контенті, що розміщений на YouTube (рис. 1.3). Він пропонує можливість слухати музику та переглядати відеокліпи, інтегруючись з великою базою відео на YouTube.

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

Артисти можуть завантажити свою музику через YouTube Studio або Music Manager. YouTube Music також має інструмент "YouTube for Artists", де артисти можуть аналізувати статистику та додавати свою музику на платформу.

Переваги:

- доступ до великої кількості відеокліпів та музичного контенту, який розміщений на платформі YouTube;
- інтеграція з YouTube дозволяє легко переключатися між аудіо та відео версіями пісень;
- інтеграція з іншими сервісами Google, забезпечуючи зручний доступ;
- річний звіт «Your Data» з інформацією про найбільш прослуховувані треки та виконавців користувача протягом року.

Недоліки:

- можливість відсутності рідкісного або ексклюзивного музичного контенту;
- інтерфейс може виглядати перевантаженим через комбінацію відео та аудіо контенту.

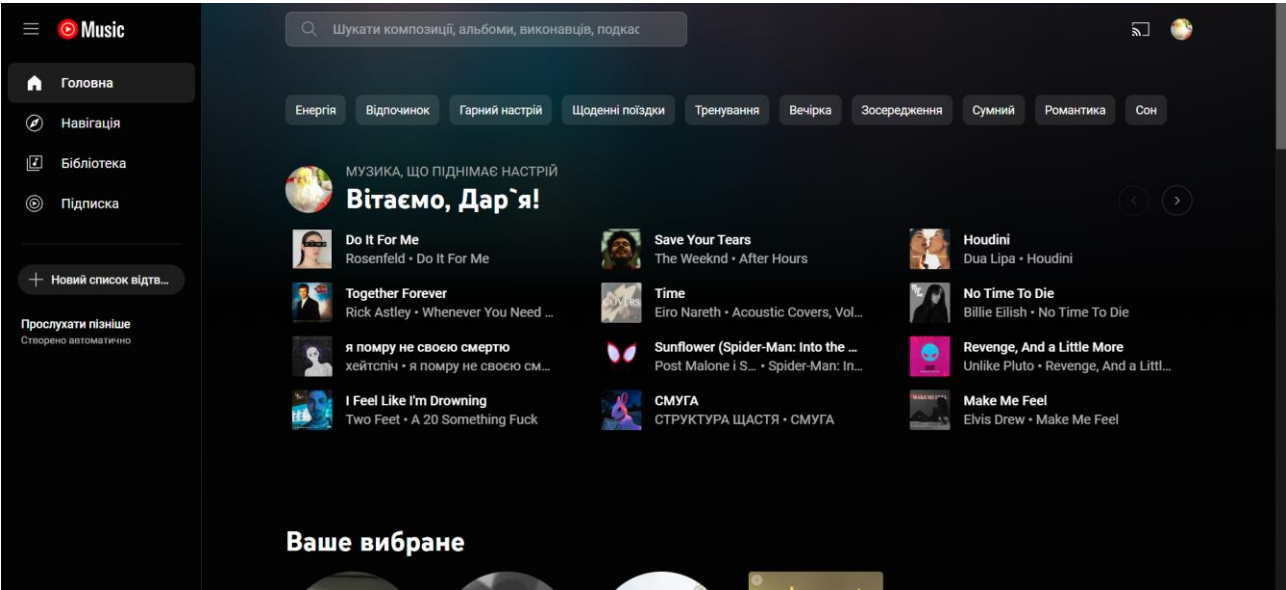


Рис. 1.3 – Інтерфейс YouTube Music

SoundCloud

SoundCloud [4] – це музичний стримінговий сервіс, що орієнтований на незалежних музикантів та користувачів, які хочуть ділитися своєю музикою та відкривати нових артистів (рис. 1.4). Він пропонує можливість завантажувати, прослуховувати, коментувати та ділитися музичними треками.

Артисти можуть завантажувати свою музику безпосередньо на платформу SoundCloud, де вони можуть створювати плейлисти, публікувати треки та взаємодіяти зі своєю аудиторією. SoundCloud також надає статистику прослуховувань, яка допомагає артистам аналізувати свої треки та розуміти вподобання слухачів.

Переваги:

- SoundCloud є популярним серед незалежних музикантів, діджеїв та продюсерів, які хочуть поширювати свою музику безпосередньо до слухачів;
- користувачі можуть коментувати треки в реальному часі, що сприяє активній взаємодії між слухачами та артистами;
- на платформі можна знайти багато ексклюзивних треків, реміксів та міксів, які не завжди доступні на інших сервісах;
- артисти можуть отримувати детальну статистику про прослуховування, що допомагає їм краще розуміти свою аудиторію та покращувати контент.

Недоліки:

- безкоштовний план має обмеження на кількість завантажень та прослуховувань, а також включає рекламу;
- інколи якість звуку може бути нижчою порівняно з іншими музичними стримінговими сервісами;
- SoundCloud менше використовується відомими артистами, що може обмежувати доступ до популярного контенту;

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		15

- інтерфейс може бути неінтуїтивним для нових користувачів через велику кількість функцій та можливостей.

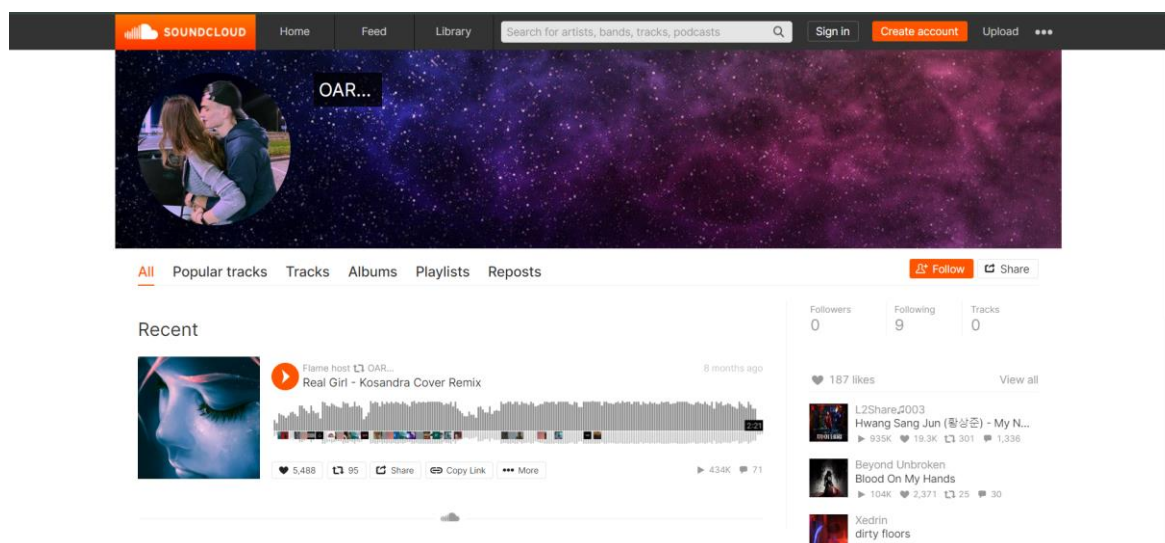


Рис. 1.4 – Інтерфейс SoundCloud

Аналізуючи наявні переваг та недоліків, складемо порівняльну таблицю програмних аналогів (табл. 1.1).

Таблиця 1.1

Порівняння аналогів

Характеристика	Spotify	Apple Music	YouTube Music	SoundCloud	Мій
Широкий вибір контенту	+	+	+	+	+
Ексклюзивний контент	-	+	-	+	+
Можливість створення плейлистів	+	+	+	+	+
Можливість завантажувати музику	+	+	+	+	+
Особистий звіт зі статистикою прослуховувань	+	+	+	-	-
Рекламні обмеження	+	+	-	+	-
Зручний інтерфейс	+	-	+	-	+
Популярність серед відомих артистів	+	+	-	-	-

1.3 Вибір архітектури сервісу

У світлі швидкого розвитку технологій та зростаючого попиту на музичні стримінгові сервіси, вибір правильної архітектури стає критичним аспектом успішної розробки та функціонування продукту. В даному розділі розглядається важливість вибору архітектурного патерну та конкретної архітектури для музичного стримінгового сервісу.

Зазначені три архітектурні патерни – Monolithic Architecture, Microservices Architecture [5] та Serverless Architecture [6] , які представляють різні підходи до розбудови систем.

Monolithic Architecture (монолітна архітектура): цей патерн визначає використання єдиного блоку для розгортання та виконання всіх компонентів додатку. Хоча такий підхід є простим для розробки та тестування, він може обмежувати гнучкість та масштабованість.

Microservices Architecture (мікросервісна архітектура): мікросервіси розділяють додаток на невеликі, автономні сервіси, що забезпечує гнучкість та масштабованість. Проте це вимагає більшого управління та може викликати високі витрати на зв'язання між сервісами.

Serverless Architecture (безсерверна архітектура): використання функцій, що викликаються подіями, замість традиційних серверів. Це зменшує витрати на інфраструктуру, але може призводити до затримок через холодний старт функцій.

З урахуванням особливостей музичного стримінгового сервісу було обрано мікросервісну архітектуру, спираючись на такі причини:

1. Гнучкість та швидкість розробки: мікросервісна архітектура дозволяє розробникам працювати над конкретними сервісами незалежно, що сприяє швидкості розробки та внесенню змін;
2. Масштабованість: використання окремих мікросервісів дозволяє масштабувати лише необхідні компоненти системи, забезпечуючи оптимальне використання ресурсів та високу продуктивність;

3. Надійність та тривалість роботи: розділення на мікросервіси підвищує надійність системи. Відмова в одному сервісі не вплине на роботу інших, що забезпечить тривалу та стабільну роботу;
4. Модульність: кожен мікросервіс може розглядатись як окремий модуль, що дозволяє ефективно вдосконалювати та розвивати систему без необхідності великих переписувань коду;
5. Підтримка найновіших технологій: використання мікросервісів дозволяє впроваджувати та використовувати найновіші технології в кожному сервісі окремо, що полегшує підтримку та розвиток.

Розглянемо основні складові мікросервісної архітектури:

1. Мікросервіси – кожен мікросервіс відповідає за конкретну функціональність, таку як управління користувачами, керування контентом, статистика тощо. Кожен мікросервіс може бути незалежним і розвиватися окремо від інших.
2. Бази даних – кожен мікросервіс може мати свою власну базу даних або використовувати спільну базу даних з іншими мікросервісами.
3. Шина повідомлень – забезпечує асинхронний зв'язок між мікросервісами. Мікросервіси можуть публікувати повідомлення в шину, а інші мікросервіси можуть підписуватися на ці повідомлення. Це дозволяє мікросервісам спілкуватися один з одним публікуючи та отримуючи повідомлення, не знаючи деталей реалізації один одного.

Враховуючи особливості стримінгового сервісу, який має велику кількість користувачів, потребує швидкого реагування та оптимального використання ресурсів, мікросервісна архітектура є оптимальним вибором. Цей підхід дозволить ефективно втілювати та розвивати різноманітні функції продукту, забезпечуючи високий рівень користувацького досвіду.

1.4 Обґрунтування вибору засобів реалізації та вимоги до апаратного забезпечення

При розробці музичного стримінгового сервісу, необхідно обрати технології, які не лише відповідають вимогам проекту, а й забезпечують ефективність, масштабованість та надійність.

У зв'язку з цим, ми вирішили обрати мову програмування TypeScript [7] для розробки на серверній та клієнтській сторонах. TypeScript надає переваги статичного типу даних, що дозволяє зменшити кількість помилок та полегшити розуміння коду для команди розробників. Крім того, TypeScript є популярним вибором для розробки великих проектів, таких як наш, завдяки його масштабованості та підтримці сучасних функцій програмування.

Для розробки серверної частини нашого додатку ми обрали фреймворк Express.js [8]. Він є одним з найпопулярніших та надійних веб-фреймворків для Node.js та TypeScript. Його мінімалістична структура дозволяє розробникам швидко створювати потужні API та обробляти запити з високою ефективністю. Крім того, Express має широку спільноту користувачів та багато розширень, що полегшують розробку та підтримку додатку.

На клієнтській частині ми використовуємо Next.js [9] для створення інтерфейсу користувача. Даний React фреймворк дозволяє швидко та ефективно розробляти високопродуктивні клієнтські додатки. Він підтримує SSR (server-side rendering) та SSG (static site generation), що забезпечує швидке завантаження сторінок. Крім того, Next.js має багато вбудованих функцій, таких як маршрутизація та оптимізація зображень, що полегшує розробку та підтримку додатку.

Для збереження даних ми обрали MongoDB [10], оскільки ця NoSQL база даних є гнучкою та масштабованою, що ідеально підходить для мікросервісної архітектури. MongoDB дозволяє нам легко розширювати нашу базу даних з ростом обсягу даних та кількості користувачів, що відповідає потребам нашого проекту.

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		19

Для керування інфраструктурою та автоматизації процесів розгортання ми обрали Kubernetes [11], Docker [12] та Skaffold [13]. Kubernetes використовується для оркестрації контейнерів та забезпечення масштабованості нашого додатку, Docker – для контейнеризації та ізоляції наших додатків, а Skaffold допомагає автоматизувати розгортання та тестування додатків в середовищі Kubernetes. Використання цих інструментів забезпечує нам швидкість розгортання, ефективне використання ресурсів та надійність нашого додатку.

Для забезпечення ефективного та надійного розподілу трафіку між нашими мікросервісами в межах кластера Kubernetes, ми обрали Nginx Ingress Controller [14]. Він використовується для балансування навантаження та маршрутизації запитів від користувачів до відповідних мікросервісів, що дозволяє нам ефективно керувати ресурсами та забезпечити стабільну роботу нашого додатку.

Для забезпечення асинхронної та масштабованої комунікації між нашими мікросервісами ми обрали NATS Streaming Server [15], [16]. Цей сервіс органічно доповнює наш стек інструментів, що складається з Kubernetes, Docker та Skaffold, і робить нашу мікросервісну архітектуру ще більш гнучкою, масштабованою та надійною.

Висновки до першого розділу

У першому розділі було визначено основні завдання для розробки музичного стримінгового сервісу та детально розглянуто конкурентне оточення, проаналізовано характеристики інших продуктів на ринку. При порівнянні з аналогами, такими як Spotify, Apple Music, YouTube Music, SoundCloud ми визначили ключові особливості, які має мати наш продукт.

Розглянуто переваги мікросервісної архітектури, обґрунтовуючи її вибір для нашого музичного стримінгового сервісу. Також було надано загальне планування реалізацій наших серверних та клієнтських частин.

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		20

У контексті інструментів було вибрано TypeScript та Express.js для серверної частини та Next.js для клієнтської, що забезпечить зручний та ефективний інтерфейс. Обираючи MongoDB для збереження даних, ми врахували його гнучкість та масштабованість.

РОЗДІЛ 2 ПРОЄКТУВАННЯ СЕРВІСУ

2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи

У цьому розділі ми зосередимося на проектуванні музичного стримінгового сервісу, що є одним із найпопулярніших та широко використовуваних типів онлайн-розваг. З метою створення ефективного та зручного для користувачів сервісу, необхідно ретельно продумати його функціональні та нефункціональні вимоги, а також визначити варіанти його використання та структуру системи.

Функціональні вимоги

Функціональні вимоги визначають основні функції та можливості системи, які повинні бути реалізовані для досягнення її цілей та задоволення потреб користувачів.

Для початку, зазначимо, що існують наступні ролі у сервісі:

1. **Гість** – фізична особа, один з видів користувачів, яка не має особистого акаунту у сервісі. У даному ресурсі гість має обмежені можливості у користуванні, але достатні, для задоволення базових потреб.
2. **Користувач** – фізична особа, один з видів користувачів, що має доступ до загальнодоступної частини сервісу, а також до інформації, функціоналу та розділів, що відкриваються після авторизації..
3. **Адміністратор** – це спеціальна роль у системі, яка має розширені повноваження щодо управління та контролю над ресурсом

Тож розглянемо функціональні вимоги для кожної з них.

Гість:

- перегляд треків;
- прослуховування треків;
- пошук треків;
- можливість первинної реєстрації та подальша авторизація.

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		22

Користувач:

- перегляд треків;
- прослуховування треків;
- пошук треків;
- створення власних плейлистів;
- редагування власних плейлистів (редагування назви, додавання або видалення треків);
- редагування плейлиста «Улюблене»;
- видалення власних плейлистів;
- редагування персональної інформації;
- завантаження власних треків на сервіс;
- редагування власних треків;
- видалення власних треків;
- перегляд персоналізованої статистики прослуховувань;
- перегляд повідомлень від адміністратора (якщо трек було видалено).

Адміністратор:

- перегляд загальної статистики сервісу;
- перегляд детальної статистики сервісу;
- видалення треків користувач та надсилання їм повідомлення з причиною видалення.

Нефункціональні вимоги

Нефункціональні вимоги визначають характеристики та критерії якості системи, які забезпечують її ефективність, надійність та зручність для користувачів. Вони включають наступні аспекти:

1. **Продуктивність:** система повинна забезпечувати швидке завантаження треків та мінімальні затримки при прослуховуванні музики.
2. **Масштабованість:** система повинна бути здатною обробляти великий обсяг користувачів та даних без погіршення продуктивності.

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		23

3. Безпека: система повинна забезпечувати захист персональних даних користувачів та конфіденційність інформації. Всі дані повинні передаватися через захищені канали зв'язку, такі як HTTPS.
4. Сумісність: система повинна бути сумісною з різними веб-браузерами та операційними системами. Вона повинна коректно працювати на сучасних версіях популярних браузерів (Chrome, Firefox, Safari, Edge).

Вимоги до часових характеристик реакції системи на дії користувача

Часові характеристики реакції системи є важливим показником її продуктивності та якості обслуговування. Вони включають наступні вимоги:

1. Час завантаження головної сторінки: головна сторінка сервісу повинна завантажуватися за час, не більший ніж 2 секунди, при стандартному інтернет-з'єднанні.
2. Час відповіді на запити пошуку: відповідь на запити пошуку треків повинна надаватися не пізніше ніж через 1 секунду після відправлення запиту користувачем.
3. Час старту відтворення треку: від моменту натискання на кнопку відтворення до фактичного початку програвання треку повинно проходити не більше ніж 3 секунди.
4. Час реєстрації та авторизації: процес реєстрації нового користувача або авторизації вже існуючого повинен займати не більше 3 секунд.
5. Час створення та редагування плейлистів: створення нового плейлиста або внесення змін до існуючого має відбуватися протягом 2 секунд.
6. Час завантаження користувацьких треків: завантаження користувацьких треків на сервер повинно займати не більше 10 секунд для треків стандартної довжини (3-5 хвилин) при стандартному інтернет-з'єднанні.

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		24

7. Час реагування на дії адміністратора: адміністраторські дії, такі як перегляд статистики чи видалення треків, повинні виконуватися протягом 2 секунд.
8. Час відображення повідомлень: повідомлення від адміністратора, зокрема про видалення треків, повинні відображатися користувачу негайно, але не пізніше ніж через 2 секунди після їх відправлення.

Специфікація варіантів використання системи

Специфікація варіантів використання системи є важливим аспектом проектування, оскільки вона визначає різні сценарії взаємодії користувачів з сервісом. Вона допомагає ідентифікувати та документувати всі можливі дії, які користувачі можуть виконувати, а також способи, якими ці дії підтримуються системою. Це дозволяє зрозуміти потреби користувачів, визначити ключові функції системи та забезпечити, що всі функціональні вимоги будуть враховані під час розробки. Діаграма використання [17] допомагає візуалізувати ці сценарії, що сприяє кращому розумінню структури та функціоналу системи для всіх учасників проекту і продемонстрована нижче (рис. 2.1).

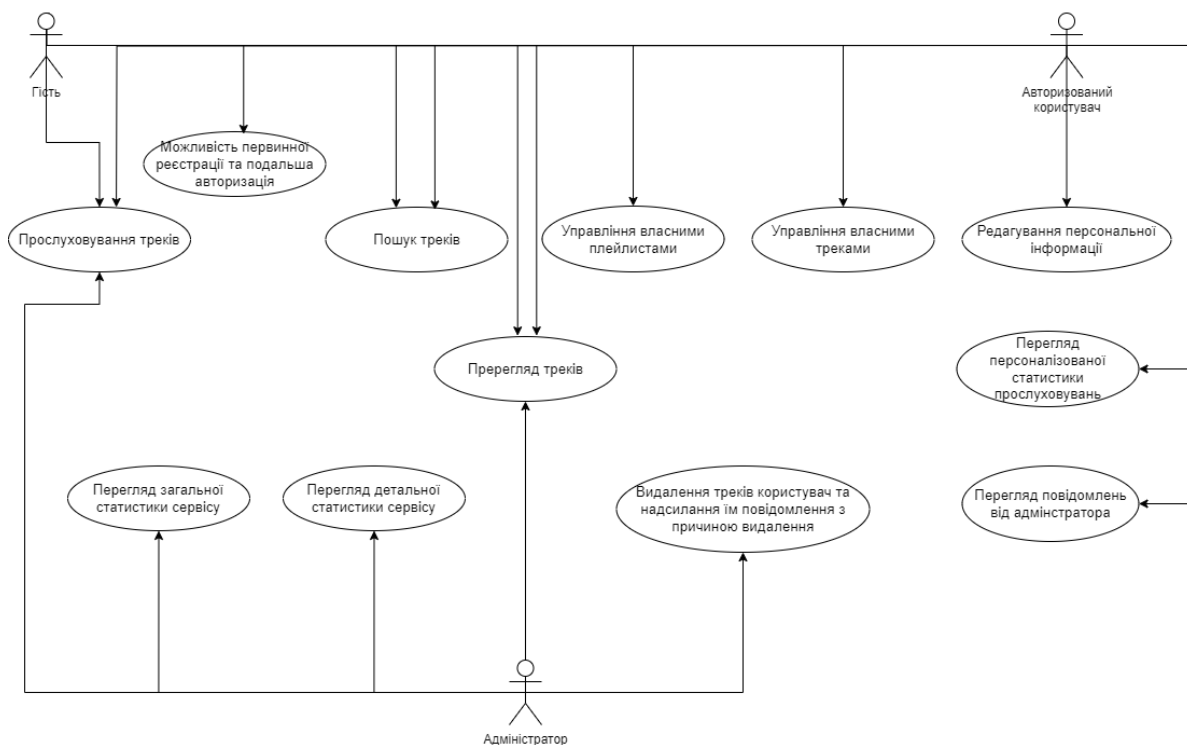


Рис. 2.1 – Діаграма використання

Об'єктно-орієнтована структура системи

Об'єктно-орієнтоване проектування (ООП) є підходом до розробки програмного забезпечення, який базується на концепції об'єктів. Вони є абстракціями реальних сутностей та мають властивості (поля) і поведінку (методи). У контексті мікросервісної архітектури, кожен мікросервіс можна розглядати як окремий об'єкт з власними моделями даних, методами та специфічною логікою.

Розглянемо об'єктно-орієнтовану структуру системи, побудованої за принципами мікросервісної архітектури. Для кращого розуміння структурних та функціональних аспектів системи, використовується комбінована діаграма, яка включає в себе елементи класичної діаграми класів, а також показує взаємодію між мікросервісами за допомогою подій та слухачів.

Комбінована діаграма класів та взаємодій мікросервісної системи є потужним інструментом для візуалізації структури та взаємодії компонентів системи. Вона включає:

1. Класи моделей: відображають структуру даних у кожному мікросервісі. Наприклад, у мікросервісі аутентифікації є моделі User з відповідними полями (ім'я, електронна адреса, пароль тощо) та методами.
2. Методи класів: Описують функціональні можливості, які надаються мікросервісами. Наприклад, методи реєстрації, аутентифікації, отримання та редагування персональної інформації.
3. Події та слухачі: відображають асинхронну взаємодію між мікросервісами. Події використовуються для повідомлення інших сервісів про зміни стану або інші важливі події. Клас Publisher відповідає за публікацію подій, а Listener отримує ці події та виконує відповідні дії. Наприклад, після реєстрації нового користувача мікросервіс аутентифікації використовує UserSignedUpPublisher для публікації події UserSignedUp. Ця подія буде отримана мікросервісом плейлистів, який за допомогою UserSignedUpListener

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		26

створить новий обліковий запис користувача та додасть для нього список відтворення «Улюблене». Такий підхід забезпечує ефективну та гнучку інтеграцію між мікросервісами, дозволяючи їм реагувати на зміни стану системи в реальному часі.

Комбінована діаграма класів та взаємодій дозволяє легко зрозуміти, як різні частини системи взаємодіють одна з одною. Вона забезпечує візуальне представлення як статичних аспектів (структура даних), так і динамічних аспектів (взаємодія через події). Використання об'єктно-орієнтованого підходу в кожному мікросервісі сприяє розділенню відповідальності та зменшує зв'язність між компонентами. Це робить систему більш гнучкою та легкою для підтримки і розширення.

Дана діаграма знаходиться в додатку А.

Основні елементи діаграми:

- 1. Сервіс Authentication: відповідає за аутентифікацію та авторизацію користувачів у системі музичного стрімінгу. Він надає ендпоінти для реєстрації, входу та виходу користувача, а також для отримання та зміни інформації про користувача. Сервіс реагує, якщо адміністратор видалив файл користувача та неодмінно надсилає повідомлення з причиною до нього.
- 2. Сервіс Content: відповідає за управління аудіофайлами, які доступні для прослуховування в системі. Він надає можливість завантаження, редагування та видалення файлів, а також публікує події про їх створення, редагування та видалення.
- 3. Сервіс Playlists: відповідає за управління плейлистами користувачів. Він надає можливість створення, редагування та видалення плейлистів, а також додавання та видалення файлів з плейлистів. Сервіс також реагує на зміни в файлах та автоматично оновлює плейлисти відповідно до змін, а також створює плейлист «Улюблене», коли користувач зареєструвався.

4. Сервіс Statistics: відповідає за збір та аналіз статистичних даних про використання музичного стримінгового сервісу. Він моніторить активність користувачів, таку як кількість відтворень треків, популярність артистів, пісень, жанрів тощо. Сервіс, як і минулий, реагує на зміни в файлах та автоматично оновлює статистику відповідно до змін, а також реагує на реєстрацію користувачів, заносючи ці дані до статистики.

2.2 Розробка бази даних системи

Для збереження всієї інформації, що обробляється у сервісі, використовується MongoDB. Ця система управління базами даних дозволяє зберігати дані у вигляді JSON-подібних документів, що забезпечує зручну та ефективну роботу зі складними структурами. Кожен мікросервіс має свою власну базу даних MongoDB, що дозволяє кожному сервісу незалежно керувати своїми даними та забезпечує гнучкість та масштабованість системи, тож розглянемо кожну базу даних.

База даних **authentication** має наступний вигляд (рис. 2.2):

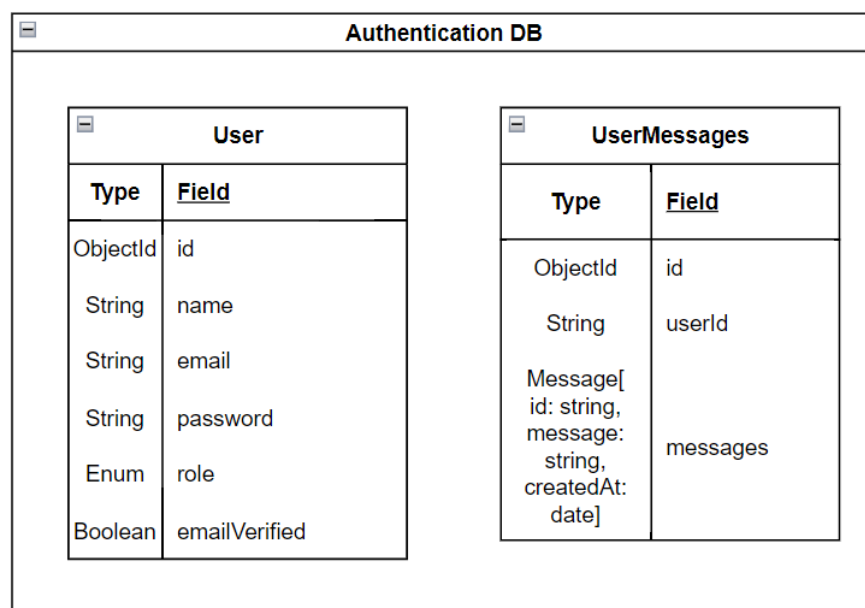


Рис. 2.2 – Структура бази даних authentication

Містить наступні таблиці:

1. Users – таблиця, що зберігає дані користувача (табл. 2.1).
2. UsersMessage – таблиця, що зберігає повідомлення для конкретного користувача від адміністратора (табл. 2.2).

Таблиця 2.1

Опис таблиці Users

Поле	Тип	Призначення
id	ObjectId	Ідентифікатор
name	String	Ім'я користувача
email	String	Пошта користувача
password	String	Пароль користувача
role	Enum	Роль користувача
emailVerified	Boolean	Позначка того чи була верифікована пошта

Таблиця 2.2

Опис таблиці UserMessages

Поле	Тип	Призначення
id	ObjectId	Ідентифікатор
userId	String	Ідентифікатор користувача, якому належить повідомлення
messages	Message [message: String, createdAt: date, read: Boolean]	Об'єкт повідомлення, що містить текст повідомлення, дату, коли було створене повідомлення та позначку чи було прочитано повідомлення.

База даних **content** має наступний вигляд (рис. 2.3):

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		29

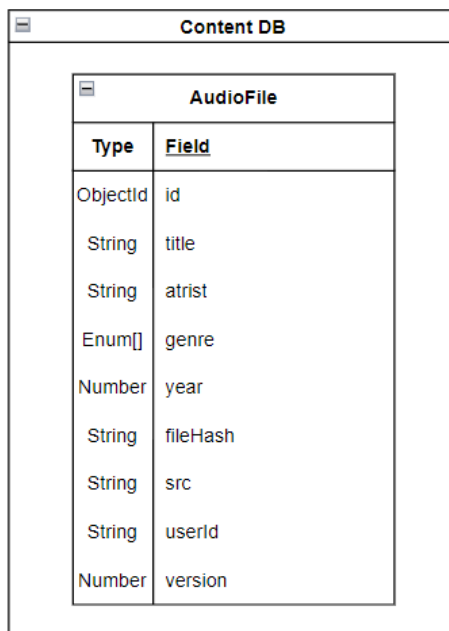


Рис. 2.3 – Структура бази даних content

Містить наступну таблицю:

1. AudioFile – таблиця, що зберігає дані про файли, які користувач завантажує на сервіс (табл. 2.3).

Таблиця 2.3

Опис таблиці AudioFiles

Поле	Тип	Призначення
id	ObjectId	Ідентифікатор
title	String	Назва треку
artist	String	Виконавець треку
genre	Enum[]	Масив жанрів треку
year	Number	Рік релізу треку
duration	Number	Тривалість треку
fileHash	String	Хешовані дані треку, щоб перевіряти файл на ідентичність
src	String	Посилання на трек у Cloudinary
userId	String	Ідентифікатор користувача, який завантажив цей трек

База даних **playlists** має наступний вигляд (рис. 2.4):

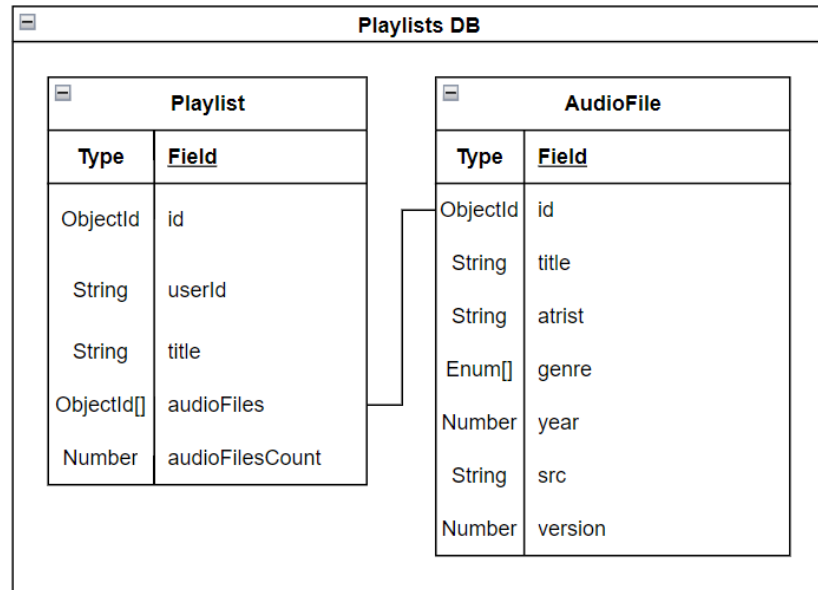


Рис. 2.4 – Структура бази даних playlists

Містить наступні таблиці:

1. Playlist – таблиця, що зберігає інформацію про плейлисти користувача (табл. 2.4).
2. AudioFile – таблиця, що зберігає файли, які будуть міститись у плейлистах (табл. 2.5).

Таблиця 2.4

Опис таблиці Playlist

Поле	Тип	Призначення
id	ObjectId	Ідентифікатор
userId	String	Ідентифікатор користувача, який створив плейлист
title	String	Назва плейлиста
audioFiles	ObjectId[]	Файли, що містяться в плейлисті
audioFilesCount	Number	Загальна кількість файлів у плейлисті

Опис таблиці AudioFile

Поле	Тип	Призначення
id	ObjectId	Ідентифікатор
title	String	Назва треку
artist	String	Виконавець треку
genre	Enum[]	Масив жанрів треку
year	Number	Рік релізу треку
duration	Number	Тривалість треку
src	String	Посилання на трек у Cloudinary

Дана таблиця схожа на таблицю з бази даних content, але не містить полів userId та fileHash у зв’язку з тим, що при оновленні даних або їх видаленні ці поля ніяк не впливають.

База даних **statistics** має наступний вигляд (рис. 2.5):

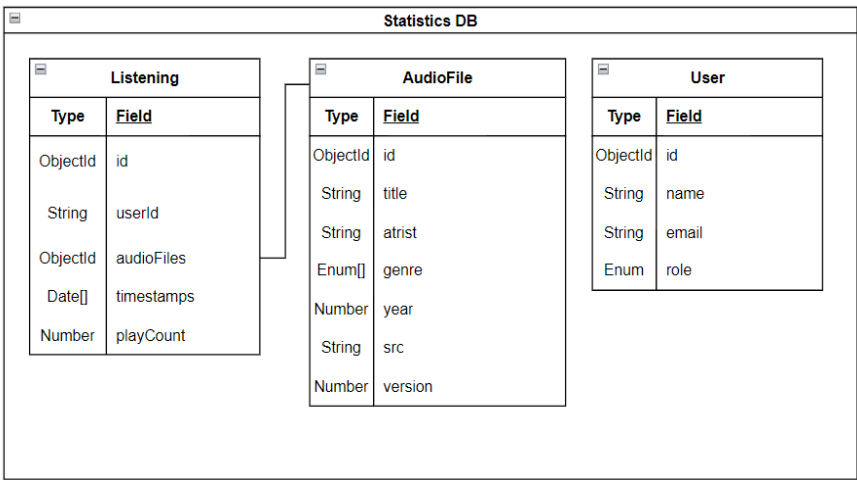


Рис. 2.5 – Структура бази даних statistics

Містить наступні таблиці:

- 1. Listening – таблиця, що зберігає інформацію про прослуховування конкретного користувача (табл. 2.6).
- 2. AudioFile – таблиця, що зберігає файли, які будуть зберігатись у прослуховуваннях (табл. 2.7).

3. User – таблиця, що зберігає інформацію про користувачів, які прослуховують треки (табл. 2.8).

Таблиця 2.6

Опис таблиці Listening

Поле	Тип	Призначення
id	ObjectId	Ідентифікатор
userId	String	Ідентифікатор користувача, який прослуховує треки
audioFileId	ObjectId	Ідентифікатор треку, який прослуховується
timestamps	Date[]	Масив дат, коли був прослуханий трек
playCount	Number	Загальна кількість прослуховування треку

Таблиця 2.7

Опис таблиці AudioFiles

Поле	Тип	Призначення
id	ObjectId	Ідентифікатор
title	String	Назва треку
artist	String	Виконавець треку
genre	Enum[]	Масив жанрів треку
year	Number	Рік релізу треку
duration	Number	Тривалість треку
src	String	Посилання на трек у Cloudinary
userId	String	Ідентифікатор користувача, який завантажив цей трек

Дана таблиця схожа на таблицю з бази даних content, але не містить поля fileHash у зв'язку з тим, що при оновленні даних або їх видаленні це поле ніяк не впливає на статистику сервісу.

Таблиця 2.8

Опис таблиці Users

Поле	Тип	Призначення
id	ObjectId	Ідентифікатор
name	String	Ім'я користувача
email	String	Пошта користувача
role	Enum	Роль користувача

Дана таблиця не містить полів password та emailVerified, на відмінну від таблиці у базі даних authentication, бо ці поля ніяк не впливають на статистику сервісу.

2.3 Проєктування та реалізація алгоритмів системи

Алгоритми відіграють важливу роль у забезпеченні ефективності та правильності функціонування системи, допомагаючи у вирішенні різноманітних завдань, від обробки запитів користувачів до управління потоками даних та їхньої обробки.

У цьому контексті ми детально розглянемо різні аспекти роботи системи, використовуючи діаграми активностей [18] для відображення послідовностей дій та процесів. Ці діаграми дозволять нам краще зрозуміти, як система взаємодіє з користувачами та внутрішніми компонентами, які керуються певними алгоритмами.

Основні елементи діаграми активностей:

- початковий вузол діяльності (круг) – вказує на початок виконання дій при запуску діяльності;
- гонтьоль потоку (стрілка) – показує послідовність виконання дій;

- дія (заокруглений прямокутник) – представляє собою конкретне завдання або дію, яку слід виконати;
- вузол-рішення (ромб з розгалудженням) – елемент, після якого потік обирає один із можливих шляхів в залежності від умов рішення;
- вузол-злиття (ромб після рішення) – об'єднує потоки, які були розділені на етапі рішення;
- вузол-розгалуження (зафарбований прямокутник зі стрілками) – розділяє на паралельні потоки без прийняття рішення;
- об'єднуючий вузол (зафарбований прямокутник зі стрілками) – об'єднує декілька паралельних потоків, розділених вузлом-розгалуженням;
- кінцевий вузол діяльності (круг з контуром) – кінцева точка діаграми, де збігаються всі потоки;
- доріжка – спосіб групування дій, виконуваних одним актором або групою.

Розглянемо діаграму «Додавання треку до плейлиста», яка продемонстрована нижче (рис. 2.6). Вона відображає послідовність дій Гостя та Користувача при спробі додати новий файл.

Пояснення:

- Початковий вузол діяльності: момент, коли Гість увійшов на сервіс у пошуках треку.
- Дія: пошук треку.
- Вузол-рішення: чи дав пошук результати?
 - Так:
 - Дія: відображення результатів пошуку.
 - Дія: вибір треку на спроба додати його у плейлист.
 - Вузол-рішення: Гість попередньо реєструвався?
 - Так:
 - Вузол-розгалуження (виконує Користувач):

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		35

- Дія: додавання треку до «Улюбленого».
- Дія: додавання треку до існуючого плейлиста.
- Дія: створення нового плейлиста.
 - Дія: автоматичне додавання треку в новий плейлист.
- Об'єднуючий вузол.
- Кінцевий вузол діяльності.
- Ні:
 - Реєстрація/авторизація гостя.
 - Вузол-розгалуження.
- Ні:
 - Дія: повідомлення про відсутність результатів пошуку.
 - Кінцевий вузол діяльності.

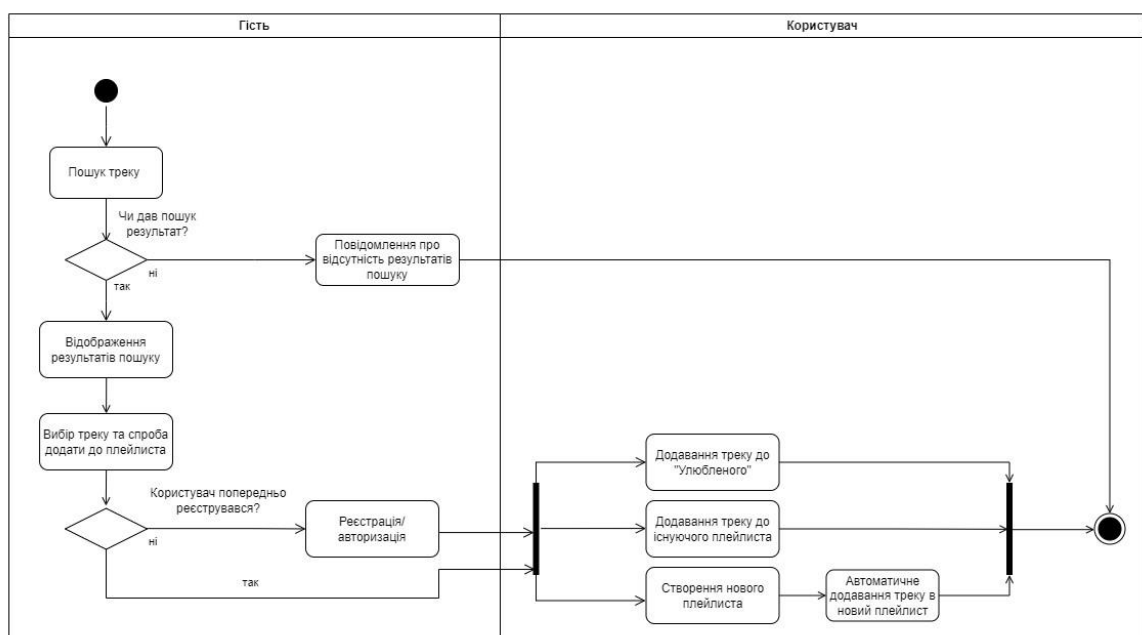


Рис. 2.6 – Діаграма активностей «Додавання треку до плейлиста»

Розглянемо наступну діаграму активностей «Видалення треку Адміністратором», яка продемонстрована нижче (рис. 2.7). Вона відображає послідовність дій Адміністратора та користувача при видаленні файлу.

Пояснення:

- Початковий вузол діяльності: момент, коли Адміністратор намагається потрапити до адмін-панелі.
- Дія: авторизація на сервісі як адміністратор.
- Вузол-рішення: користувач має роль адміністратора?
 - Так:
 - Дія: перехід на головну сторінку.
 - Дія: перехід на сторінку видалення треків користувача.
 - Дія: перегляд базової інформації про обраний трек та його прослуховування.
 - Вузол-рішення: трек задовольняє всі вимоги сервісу?
 - Так:
 - Кінцевий вузол діяльності.
 - Ні:
 - Видалення треку Адміністратором з зазначеною причиною.
 - Надсилання повідомлення про видалення треку Користувачу.
 - Отримання Користувачем повідомлення про видалення треку.
 - Перегляд повідомлення від Адміністратора.
 - Вузол-рішення: бажаєте завантажити трек повторно?
 - Так:
 - Дія: повторне завантаження треку на сервіс з урахуванням зауважень Адміністратора.
 - Ні:
 - Кінцевий вузол діяльності.

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		37

- Ні:
 - Дія: повідомлення про заборону доступу.
 - Кінцевий вузол діяльності.

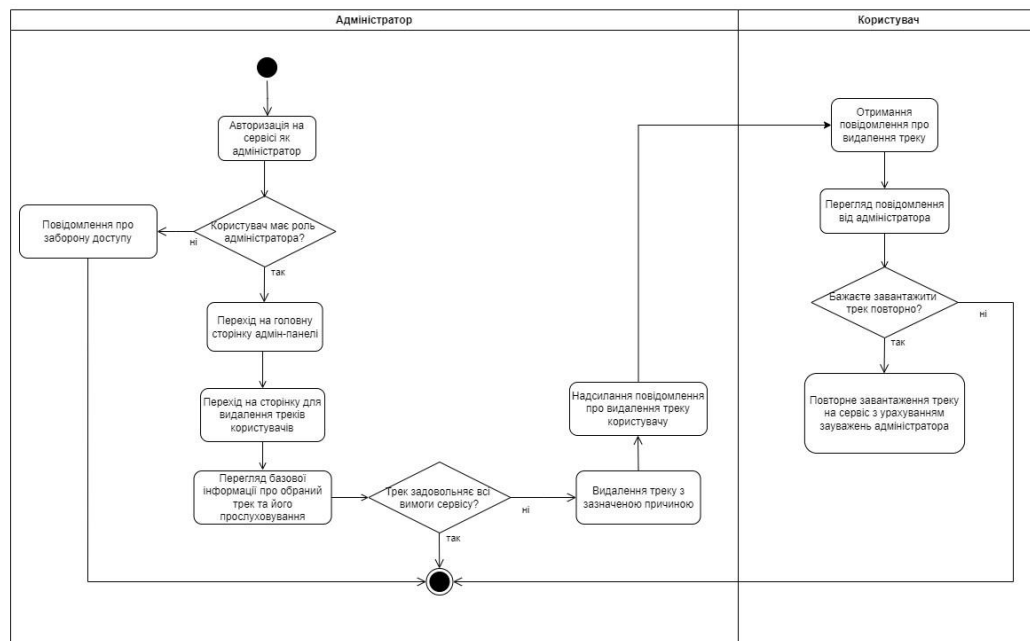


Рис. 2.7 – Діаграма активностей «Видалення треку Адміністратором»

2.4 Реалізація функціоналу системи

Для побудови мікросервісного додатку з ефективною обробкою подій із гарантією доставки, ключовим аспектом є використання NATS Streaming. Цей сервер дозволяє створювати надійні з'єднання між мікросервісами та забезпечує масштабовану обробку подій.

Для взаємодії з NATS ми використовуємо клас NatsWrapper, який надає зручний інтерфейс для підключення та використання NATS Streaming клієнта.

Реалізація класу NatsWrapper:

```

import nats, { Stan } from 'node-nats-streaming';

class NatsWrapper {
  private _client?: Stan;

  get client() {
    if (!this._client) {

```

```

        throw new Error('Cannot access NATS client before connecting');
    }

    return this._client;
}

connect(clusterId: string, clientId: string, url: string) {
    this._client = nats.connect(clusterId, clientId, { url });

    return new Promise<void>((resolve, reject) => {
        this.client.on('connect', () => {
            console.log('Connected to NATS!');
            resolve();
        });

        this.client.on('error', (err) => {
            reject(err);
        });
    });
}

export const natsWrapper = new NatsWrapper();

```

Даний клас використовується для підключення до сервера NATS Streaming. Він надає метод `connect()`, що дозволяє встановити зв'язок з сервером NATS за допомогою вказаного `clusterId`, `clientId` і `url`. Після успішного підключення об'єкт `_client` отримує NATS клієнт (Stan), через який здійснюється подальша взаємодія з сервером.

Далі розглянемо виклик вище розглянутого об'єкту

Підключення до NATS:

```

await natsWrapper.connect(
    process.env.NATS_CLUSTER_ID,
    process.env.NATS_CLIENT_ID,
    process.env.NATS_URL
);

```

Обробка закриття з'єднання:

```

natsWrapper.client.on('close', () => {
    console.log('NATS connection closed');
    process.exit();
});

```

Обробка сигналів завершення процесу:

```

process.on('SIGINT', () => natsWrapper.client.close());
process.on('SIGTERM', () => natsWrapper.client.close());

```

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		39

Далі для ефективної і розширюваної системи розглянемо такі компоненти, як події (events), підписники (listeners) та публікатори (publishers). Давайте детальніше розглянемо кожен з цих компонентів та їх впровадження у нашому сервісі.

Було створено базовий клас для публікаторів, який розміщений у власно створеній бібліотеці для доступності його використання у різних мікросервісах.

Основні елементи класу Publisher:

Інтерфейс Event:

```
interface Event {
  subject: Subjects;
  data: any;
}
```

Інтерфейс Event визначає структуру події, яка включає тему (subject) та дані (data). Це допомагає забезпечити типізацію та консистентність формату даних при публікації подій.

Абстрактний клас Publisher:

```
export abstract class Publisher<T extends Event> {
  abstract subject: T['subject'];
  protected client: Stan;

  constructor(client: Stan) {
    this.client = client;
  }

  publish(data: T['data']): Promise<void> {
    return new Promise((resolve, reject) => {
      this.client.publish(this.subject, JSON.stringify(data), (err) => {
        if (err) {
          return reject(err);
        }

        console.log('Event published to subject', this.subject);
        resolve();
      });
    });
  }
}
```

Клас Publisher має наступні властивості та методи:

- `subject`: абстрактне поле, яке визначає тему події, що буде публікуватися. Кожен підклас, який наслідується від `Publisher`, повинен реалізувати це поле згідно конкретної теми.
- `client`: екземпляр клієнта NATS Streaming, через який відбувається публікація подій.
- `publish(data: T['data'])`: метод, який приймає дані події (`data`) і публікує їх у NATS Streaming. Дані перетворюються в JSON-рядок перед публікацією. Метод повертає обіцянку (`Promise`), яка вирішується після успішної публікації або відхилення у разі помилки.

Поданий клас `Publisher` дозволяє іншим сервісам публікувати події в NATS Streaming, що використовується для асинхронної комунікації між сервісами. Наприклад, при створенні нового аудіофайлу, сервіс може використовувати підклас `Publisher` для публікації події про створення аудіофайлу, яку інші сервіси (наприклад, сервіс `Playlists`) можуть підписатися та обробити цю подію.

Приклад використання:

Створення конкретного підкаласу `Publisher`: (у мікросервісі `Content`)

```
import { Publisher, Subjects, ContentCreatedEvent } from '@dbmusicapp/common';

export class ContentCreatedPublisher extends Publisher<ContentCreatedEvent> {
  readonly subject = Subjects.ContentCreated;
}
```

Публікація події: (у мікросервісі `Content`)

```
await new ContentCreatedPublisher(natsWrapper.client).publish({
  id: audioFile.id,
  title: audioFile.title,
  artist: audioFile.artist,
  genre: audioFile.genre,
  year: audioFile.year,
  duration: audioFile.duration,
  src: audioFile.src,
  userId: audioFile.userId,
  version: audioFile.version,
});
```

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ІЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

Також було створено базовий клас для слухачів, який розміщений у тій самій бібліотеці, що і клас Publisher.

Основні елементи класу Listener:

Інтерфейс Event:

```
interface Event {
  subject: Subjects;
  data: any;
}
```

Абстрактний клас Listener:

```
export abstract class Listener<T extends Event> {
  abstract subject: T['subject'];
  abstract queueGroupName: string;
  abstract onMessage(data: T['data'], msg: Message): void;
  private client: Stan;
  protected ackWait = 5 * 1000;

  constructor(client: Stan) {
    this.client = client;
  }

  subscriptionOptions() {
    return this.client
      .subscriptionOptions()
      .setDeliverAllAvailable()
      .setManualAckMode(true)
      .setAckWait(this.ackWait)
      .setDurableName(this.queueGroupName);
  }

  listen() {
    const subscription = this.client.subscribe(
      this.subject,
      this.queueGroupName,
      this.subscriptionOptions()
    );

    subscription.on('message', (msg: Message) => {
      console.log(`Message received: ${this.subject} / ${this.queueGroupName}`);

      const parsedData = this.parseMessage(msg);
      this.onMessage(parsedData, msg);
    });
  }

  parseMessage(msg: Message) {
    const data = msg.getData();
    return typeof data === 'string'
```

```

    ? JSON.parse(data)
    : JSON.parse(data.toString('utf8'));
  }
}

```

Клас Listener має кілька ключових методів та властивостей:

- subject: абстрактне поле, яке визначає тему події. Кожен конкретний підписник повинен визначити, на яку тему він підписується.
- queueGroupName: абстрактне поле, яке визначає групу черг для обробки подій. Це корисно для балансування навантаження між кількома інстанціями сервісу.
- onMessage: абстрактний метод, який повинен бути реалізований в підкласах для обробки отриманих даних події.
- client: екземпляр клієнта NATS Streaming, який використовується для підписки на події.
- ackWait: час очікування підтвердження обробки повідомлення, встановлений за замовчуванням на 5 секунд.
- subscriptionOptions: метод, який налаштовує параметри підписки, такі як режим підтвердження вручну та час очікування.
- listen(): метод, який здійснює підписку на події та налаштовує обробку отриманих повідомлень.
- parseMessage: метод для розбору отриманих повідомлень, що конвертує їх з JSON-формату.

Поданий клас використовується для створення конкретних підписників, які будуть обробляти різні події, що генеруються іншими сервісами. Наприклад, сервіс Playlists може підписуватися на події, які генерує сервіс Content, такі як додавання або видалення аудіофайлів, і оновлювати відповідні плейлисти.

Приклад використання:

Створення конкретного підписника: (мікросервіс Playlist)

```

import { Message } from 'node-nats-streaming';
import { Subjects, Listener, ContentCreatedEvent } from '@dbmusicapp/common';
import { AudioFile } from '../models/audio-file';

```

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		43

```
import { queueGroupName } from './queue-group-name';

export class ContentCreatedListener extends Listener<ContentCreatedEvent> {
  readonly subject = Subjects.ContentCreated;
  queueGroupName = queueGroupName;

  async onMessage(data: ContentCreatedEvent['data'], msg: Message) {
    const { id, title, artist, genre, year, duration, src } = data;
    const audioFile = AudioFile.build({
      id,
      title,
      artist,
      genre,
      year,
      duration,
      src,
    });

    await audioFile.save();
    msg.ack();
  }
}
```

Слід зазначити, що ми використовуємо тут queueGroupName. Приклад використання:

```
export const queueGroupName = 'playlists-service';
```

Цей приклад показує визначення чергової групи для сервісу Playlist. Всі екземпляри сервісу, які підписані на одну тему з використанням цієї чергової групи, будуть отримувати повідомлення тільки для власної обробки. Це забезпечує їх коректну і синхронізовану роботу. Загалом queueGroupName допомагає забезпечити правильну та ефективну роботу повідомлень в сервісах, зменшуючи подвійну обробку та забезпечуючи консистентність взаємодії між компонентами системи.

Ініціалізація підписника: (мікросервіс Playlist)

```
import { natsWrapper } from './nats-wrapper';
import { ContentCreatedListener } from './events/listeners/content-created-listener';

new ContentCreatedListener(natsWrapper.client).listen();
```

З основних функцій системи одією з найважливіших є створення контенту – завантаження треку на сервіс, тож розглянемо його детальніше.

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		44

Налаштування *multer* [18] для зберігання файлів:

```
const storage: multer.StorageEngine = multer.diskStorage({
  destination: (req, file, cb) => {
    const uploadsDir = './src/uploads';
    if (!fs.existsSync(uploadsDir)) {
      fs.mkdirSync(uploadsDir);
    }
    cb(null, uploadsDir);
  },
  filename: (req, file, cb) => {
    cb(null, file.originalname);
  },
});

const upload = multer({ storage });
```

Ми використовуємо *multer* для збереження файлів у директорії *./src/uploads*. Це необхідно для тимчасового зберігання файлів до їхнього завантаження на Cloudinary.

Завантаження на Cloudinary:

```
await cloudinary.uploader.upload(
  req.file!.path,
  {
    folder: 'musicapp',
    use_filename: true,
    resource_type: 'raw',
  },
  async (err, result) => {
    if (err) {
      console.error('Cloudinary upload error:', err);
      res.status(500).json({ message: 'Cloudinary upload failed!' });
    } else {
      const { title, artist, genre, year, duration } = req.body;

      const audioFile =
        result &&
        AudioFile.build({
          title,
          artist,
          genre,
          year,
          duration,
          fileHash,
          src: result.url,
          userId: req.currentUser!.id,
        });
    }
  });
```

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		45

```

        if (audioFile) {
            await audioFile.save();
            res.status(201).send(audioFile);
        } else {
            res.status(500).json({ message: 'Result is undefined' });
        }
    }
}
);

```

У даному коді виконується процес завантаження аудіофайлу на платформу Cloudinary [19]. Після того як файл успішно завантажений, обробляється результат цього завантаження. Перш ніж відправити файл на Cloudinary, відбувається валідація даних форми, включаючи назву, виконавця та інші поля. Якщо вони відповідають вимогам, файл зберігається в папці «musicapp» на Cloudinary, і його URL зберігається в базі даних, разом з іншою відомостями про файл та користувача, який його завантажив. В разі успішного збереження, клієнту повертається відповідь зі статусом 201 та об'єктом audioFile, який містить всі деталі про завантажений файл. Також слід зазначити, що саме тут відбувається публікація події про створення контенту, фрагмент коду якої був наведений вище.

На клієнтській частині створення контенту відбувається наступним чином:

Фрагмент з конфігурацією запиту (useRequest):

```

const { doRequest, errors } = useRequest({
  url: '/api/content',
  method: 'post',
  headers: { 'Content-Type': 'multipart/form-data' },
  onSuccess: () => {
    setLoading(false);
    toast.success('Файл успішно завантажено!');
    router.push('/');
  },
});

```

Слід зазначити, що useRequest є індивідуальним хуком [20], написаним для потреб управління запитами до сервера в середовищі React. Даний фрагмент ініціалізує змінні doRequest і errors, які дозволяють взаємодіяти з HTTP-запитами до API. Ключовими параметрами конфігурації є url, method, і

headers, де url вказує на точку входу /api/content, method визначає HTTP-метод POST для відправлення даних, а headers встановлює тип контенту як multipart/form-data, що потрібно для відправлення файлів.

Фрагмент з обробником подання форми (onSubmit):

```
const onSubmit = async (event) => {
  event.preventDefault();
  if (!acceptedTerms) {
    toast.error('Ви повинні прийняти умови використання, щоб продовжити.');
```

return;

```
  }
  setLoading(true);

  const formData = new FormData();
  formData.append('title', song.title);
  formData.append('artist', song.artist);
  formData.append('year', song.year);
  formData.append('duration', song.duration);

  song.genre.forEach((genre, index) => {
    formData.append(`genre[${index}]`, genre);
  });

  if (song.file) {
    formData.append('file', song.file);
  }

  doRequest(formData);
};
```

Даний фрагмент коду включає функцію onSubmit, яка обробляє подання форми при натисканні кнопки «Завантажити». Вона початково перевіряє, чи користувач погодився з умовами використання. Якщо умови не погоджені, виводиться відповідне повідомлення помилки. Далі створюється об'єкт FormData [21], в який додаються всі введені користувачем дані про пісню: назва, виконавець, рік випуску, тривалість та жанри. Коли користувач вибрав аудіофайл, він також додається до FormData. Після підготовки всіх даних викликається doRequest(formData), що ініціює відправку цих даних на сервер для збереження нового аудіофайлу.

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		47

Висновки до другого розділу

У другому розділі було проведено детальний аналіз функціональних та нефункціональних вимог до музичного стримінгового сервісу. Об'єктно-орієнтована структура системи, заснована на мікросервісній архітектурі, гарантує гнучкість та легкість інтеграції компонентів, що є критичним для успішного створення сучасного стримінгового сервісу.

Розроблено комплексну структуру баз даних для музичного стримінгового сервісу. Кожна з баз (authentication, content, playlists, statistics) має свою унікальну функціональність та включає в себе відповідні таблиці з чітко визначеними полями та зв'язками між ними.

Створено та реалізовано комплекс алгоритмів для ефективної роботи музичного стримінгового сервісу, що охоплюють різноманітні аспекти, від обробки запитів користувачів до управління даними та їхньої адміністрації. Використання діаграм активностей дозволило наочно відобразити послідовності дій та процесів, які відбуваються в системі,

Також продемонстровано опис асинхронної комунікації між сервісами та розглянуто завантаження файлів через бібліотеку `multer` з подальшим збереженням на платформі `Cloudinary`, що дозволяє користувачам безпечно та ефективно керувати файлами у додатку.

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		48

РОЗДІЛ 3 ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СЕРВІСОМ

3.1 Порядок встановлення та налаштування параметрів сервісу

Для ефективного впровадження та налаштування музичного стрімінгового сервісу, що базується на сучасних технологіях контейнеризації та оркестрації [22], необхідно розуміти основні компоненти та їх взаємодію. Система розгорнута з використанням Docker, Kubernetes та Skaffold, що забезпечує високу гнучкість та масштабованість інфраструктури.

На першому етапі встановлення необхідно підготувати середовище для розгортання мікросервісів та клієнтської складової. Кожен мікросервіс і клієнтська частина розгортаються у власному контейнері, що спрощує процес керування та масштабування. Для цього використовується Docker для контейнеризації кожного компонента системи. Контейнери для мікросервісів authentication, content, playlists, statistics, а також клієнтської частини на основі Next.js, створюються на базі відповідних Docker образів.

Далі, за допомогою Kubernetes і Skaffold встановлюються та оркеструються ці контейнери. Kubernetes забезпечує автоматизацію управління контейнерами, ресурсами та моніторингом, що є важливим для забезпечення стабільної роботи системи при змінному навантаженні.

Після розгортання контейнерів і оркестрації мікросервісів, необхідно налаштувати їх взаємодію та інтеграцію. Кожен мікросервіс підключається до власної бази даних MongoDB, що забезпечує збереження та маніпулювання відповідними даними.

Сервіс content взаємодіє з хмарним сховищем Cloudinary через відповідні API для завантаження та зберігання медіафайлів, що є ключовим аспектом для надання користувачам доступу до контенту.

Інтерфейс для взаємодії з сервісом забезпечується за допомогою Ingress, який маршрутизує зовнішні та внутрішні запити до відповідних мікросервісів та забезпечує захист трафіку.

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		49

Цей підхід до розгортання та налаштування сервісу забезпечує масштабованість, надійність та ефективність в роботі музичного стримінгового сервісу, що відповідає сучасним вимогам до інфраструктури веб-програмних додатків.

3.2 Структура інтерфейсу сервісу

Інтерфейс є ключовим елементом будь-якого проекту, оскільки він визначає зручність користування та взаємодію користувачів з платформою. У цьому розділі розглянуто структуру інтерфейсу нашого сервісу, яка базується на сучасних дизайнерських та UI/UX практиках [23].

Структура інтерфейсу включає в себе елементи навігації, візуальне оформлення, управління контентом та інші функціональні аспекти, що впливають на враження користувача від використання нашим сервісу. Тож детально розглянемо його.

Початково користувач потрапляє на головну сторінку (рис. 3.1):

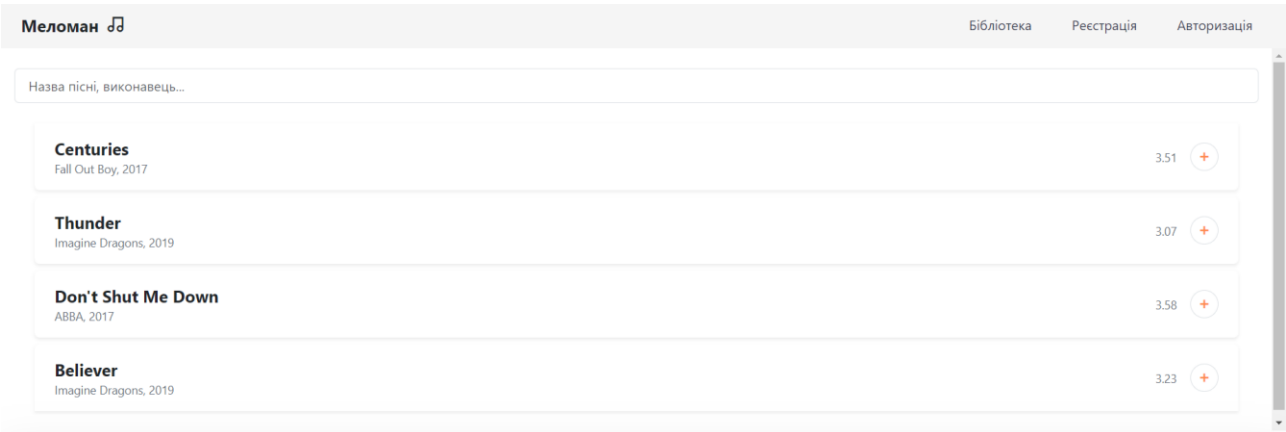


Рис. 3.1 – Головна сторінка

Тут він може прослуховувати музику, перемикається між нею та змінювати налаштування звуку. Також, користувач може скористатись пошуком (рис. 3.2), якщо хоче знайти якийсь конкретний трек за назвою або виконавцем:

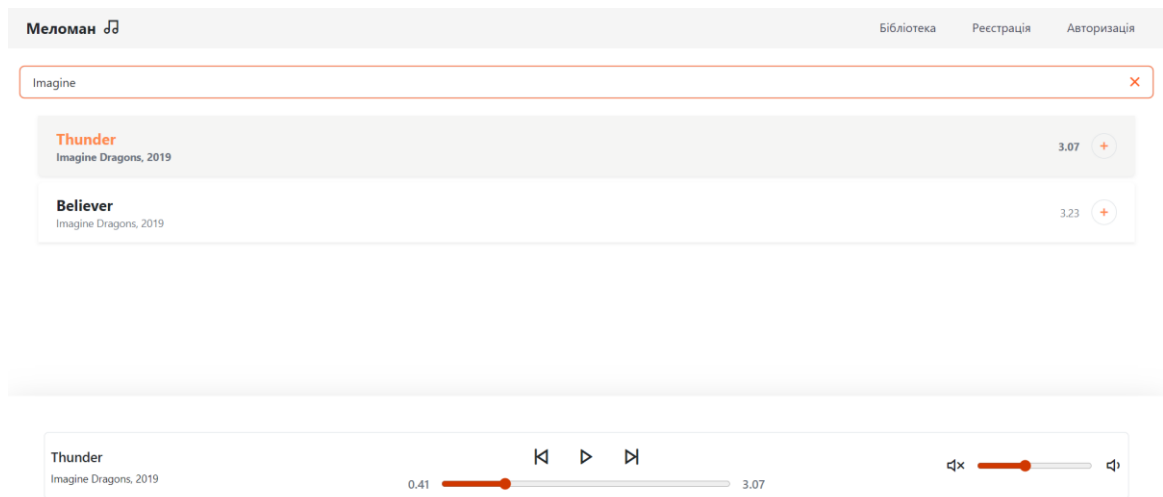


Рис. 3.2 – Результат пошуку

Також користувач може перейти на сторінку бібліотеки (рис. 3.3), де переглянути певні треки за доступними жанрами сервісу (рис. 3.4):

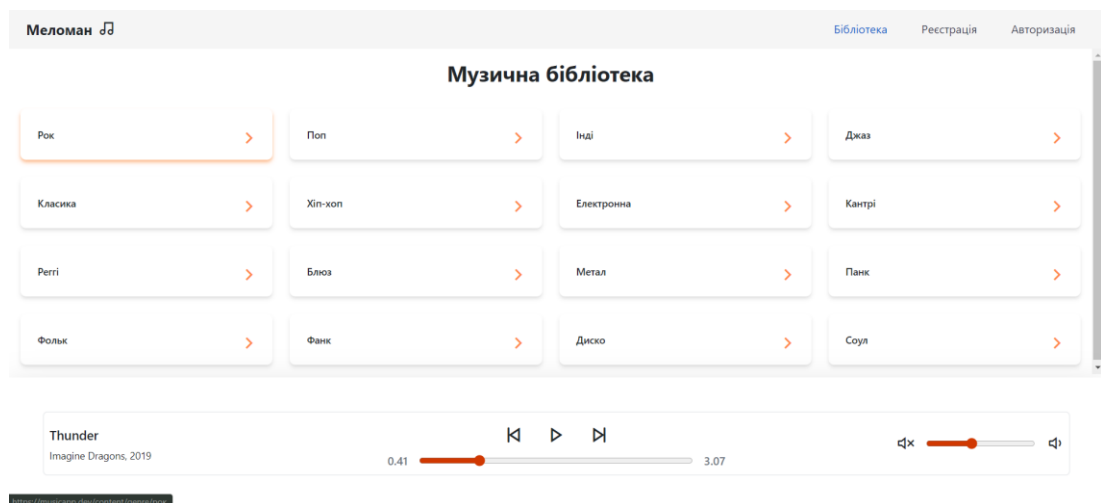


Рис. 3.3 – Перегляд сторінки бібліотеки

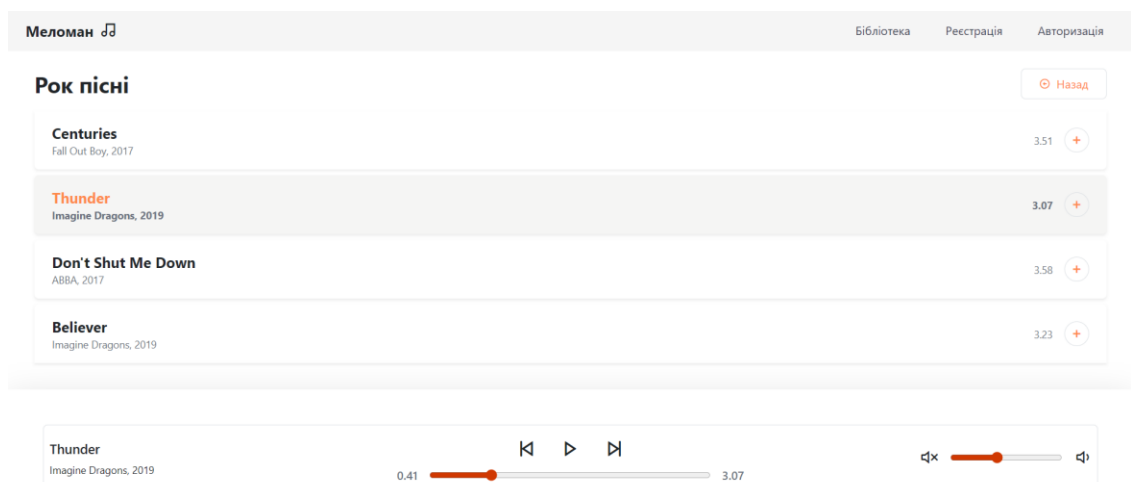


Рис. 3.4 – Перегляд сторінки пісень конкретного жанру

Біля кожною пісня ми бачимо знак «+». Натиснувши на нього ми отримуємо наступне (рис. 3.5):

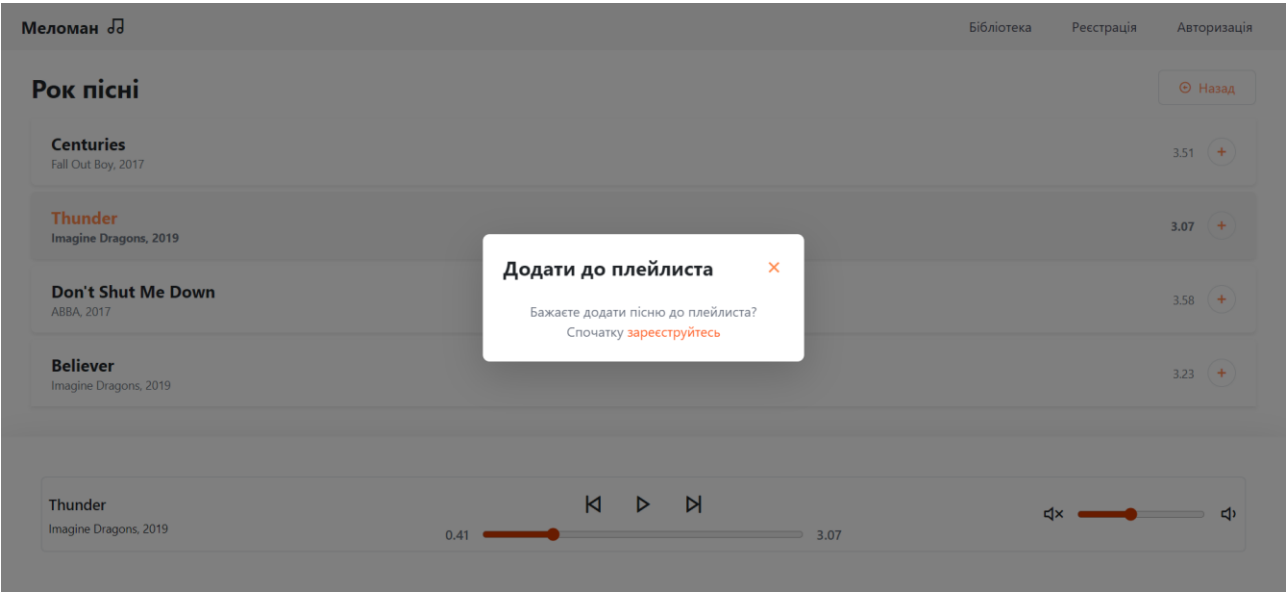


Рис. 3.5 – Модальне вікно для додавання треку до плейлиста

Як бачимо, звичайний користувач не може створювати власних плейлистів. Початково йому варто зареєструватись або авторизуватись на сервісі. Ми можемо це зробити або натиснувши на дане посилання, або перейшовши за посилання у верхньому меню (рис. 3.6):

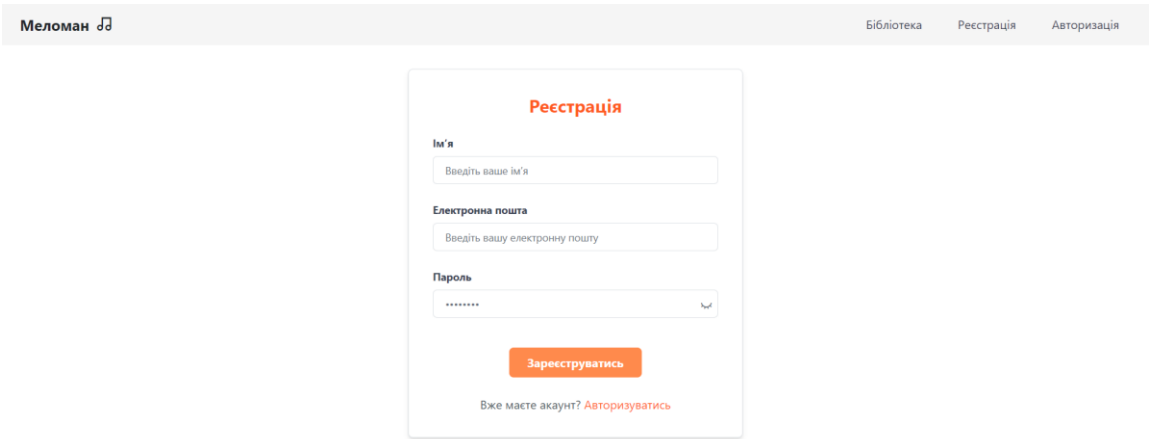


Рис. 3.6 – Сторінка реєстрації

При реєстрації користувач має ввести своє ім'я, пошту, яка в подальшому може бути верифікована, та пароль. Ми вже зареєстровані на сервісі, тому авторизуємось, перейшовши за посиланням і заповнимо дані (рис. 3.7):

Меломан

Бібліотека

Реєстрація

Авторизація

Авторизація

Електронна пошта

jabko0137@gmail.com

Пароль

.....

Авторизуватись

Ще не маєте акаунту?
Зареєструватись

Рис. 3.7 – Сторінка авторизації

При успішній авторизації переходимо на головну сторінку користувача (рис. 3.8). Як бачимо, у нас змінилась верхня панель, розширивши свої функції, а також з'явилась додаткова можливість додавати пісню до плейлиста «Улюблене»:

Меломан

Бібліотека

Мої плейлисти

Особистий кабінет

Вихід

Назва пісні, виконавець...

Centuries

Fall Out Boy, 2017

3.51

+

Thunder

Imagine Dragons, 2019

3.07

+

Don't Shut Me Down

ABBA, 2017

3.58

+

Believer

Imagine Dragons, 2019

3.23

+

Thunder

Imagine Dragons, 2019

0.16

3.07

0.16

3.07

Рис. 3.8 – Головна сторінка авторизованого користувача

Додавання треку до «Улюбленого» виконується через натискання іконки сердечка, якщо ми хочемо його туди додати (рис. 3.9). Після цього його також можна видалити звідти, зробивши таку саму операцію. Слід зазначити, що цей плейлист автоматично створюється у кожного зареєстрованого користувача:

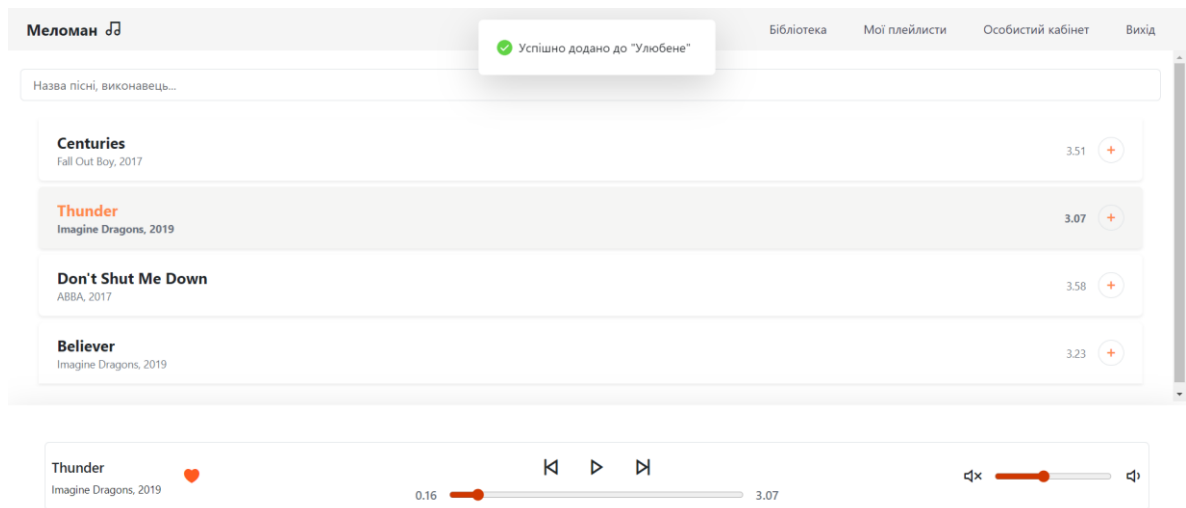


Рис. 3.9 – Додавання до «Улюблене»

Ми також можемо додати треки до своїх власно створених плейлистів, натиснувши плюс біля обраного з них (рис. 3.10):

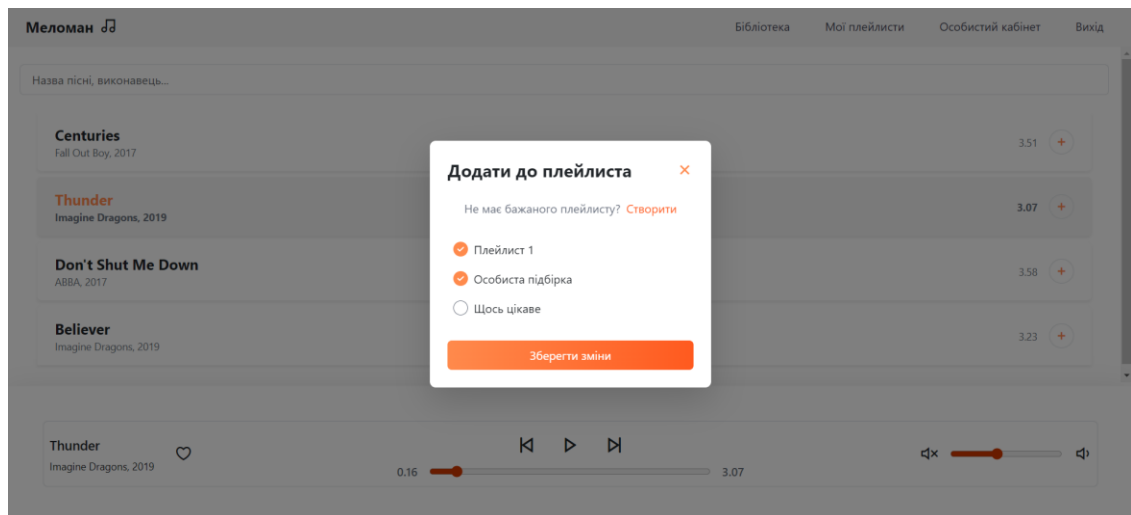


Рис. 3.10 – Модальне вікно для додавання треку до плейлиста

Якщо ми хочемо створити новий плейлист і додати туди даний трек, то у нас є доступне для цього посилання. Якщо хочемо додати або видалити з існуючих – користуємось чек-боксами для цього.

При додаванні треку до існуючого плейлиста або плейлистів, натискаємо біля нього чек-бокс і далі кнопку «Зберегти зміни» (рис. 3.11). Далі нас буде перенаправлено на сторінку «Мої плейлисти» (рис. 3.12):

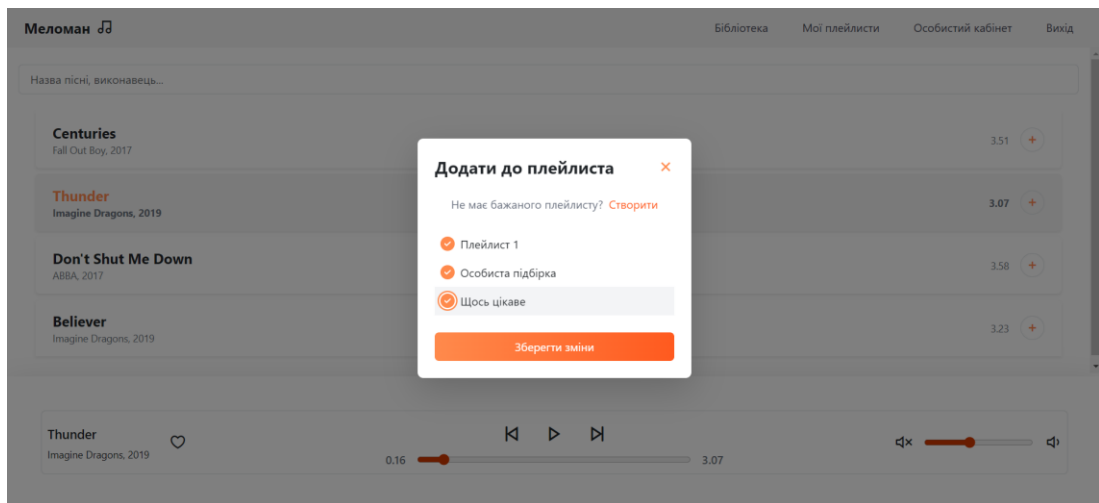


Рис. 3.11 – Вибір до якого плейлиста додати

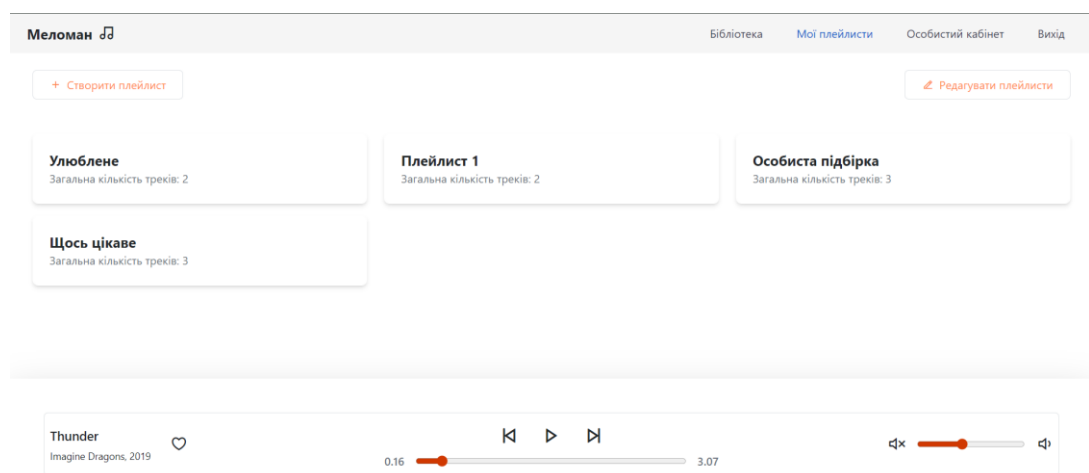


Рис. 3.12 – Сторінка «Мої плейлисти»

На сторінці «Мої плейлисти» ми можемо створити новий плейлист, натиснувши відповідну кнопку, коректно заповнивши дані і натиснувши кнопку «Створити» (рис. 3.13):

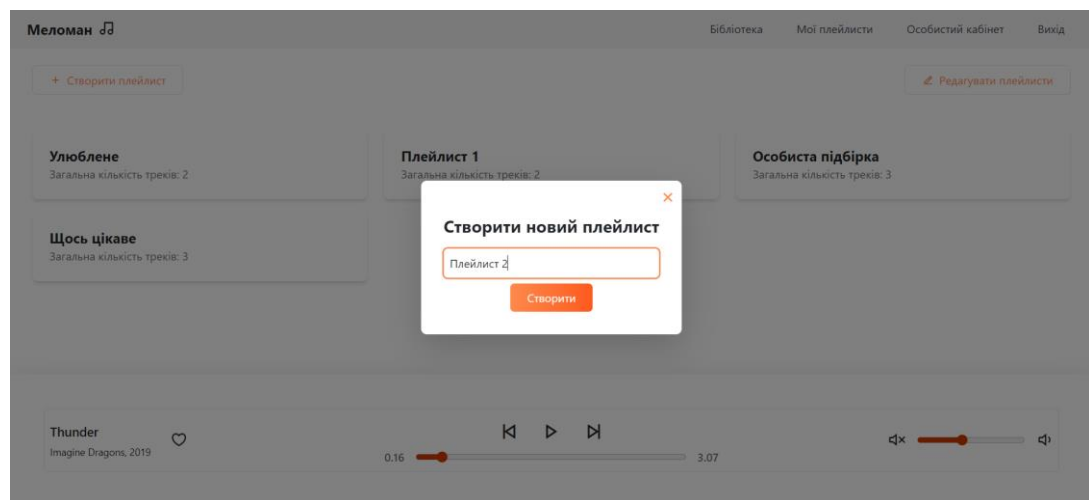


Рис. 3.13 – Модальне вікно для створення плейлиста

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		55

На цій сторінці ми також можемо відредагувати плейлисти, а саме видалити їх, натиснувши відповідну кнопку. Коли ми видалили всі плейлисти, що нам треба, використовуючи відповідні іконки, натискаємо кнопку «Зберегти» і отримуємо оновлений список плейлистів (рис. 3.14):

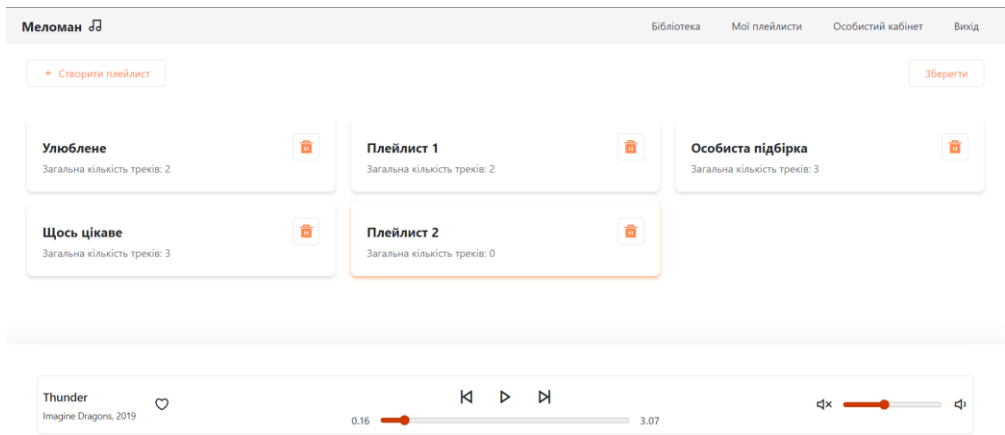


Рис. 3.14 – Режим редагування плейлистів, обираємо видалити вище створений плейлист

Ми можемо також перейти на сторінку обраного плейлиста і прослухати пісні в ньому, або ж відредагувати його, натиснувши відповідну кнопку. Змінимо назву та видалимо один з треків з плейлиста (рис. 3.15):

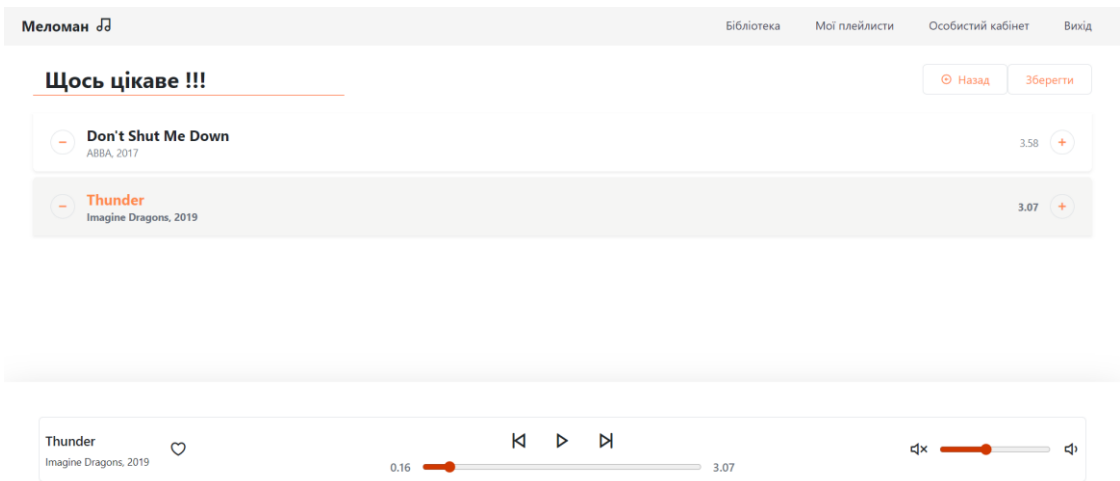


Рис. 3.15 – Режим редагування плейлиста, зміна назва і видалення другого треку зі списку

Тепер перейдемо на сторінку «Особистий кабінет» (рис. 3.16):

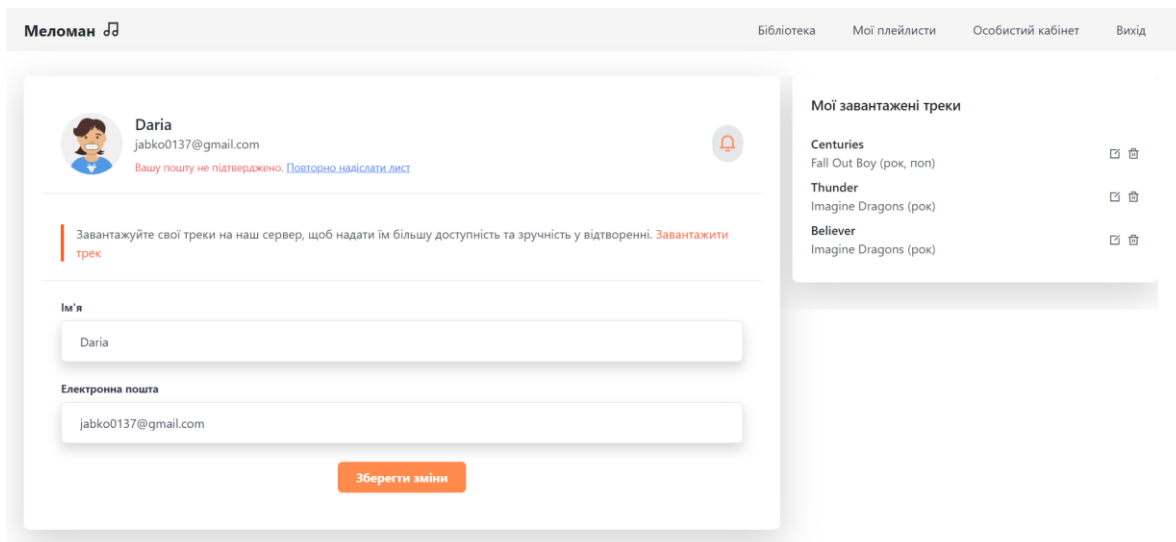


Рис. 3.16 – Особистий профіль користувача

На даній сторінці ми бачимо особисті відомості користувача, які можемо змінити. Вводимо нове ім'я і натискаємо відповідну кнопку для збереження змін (рис. 3.17):

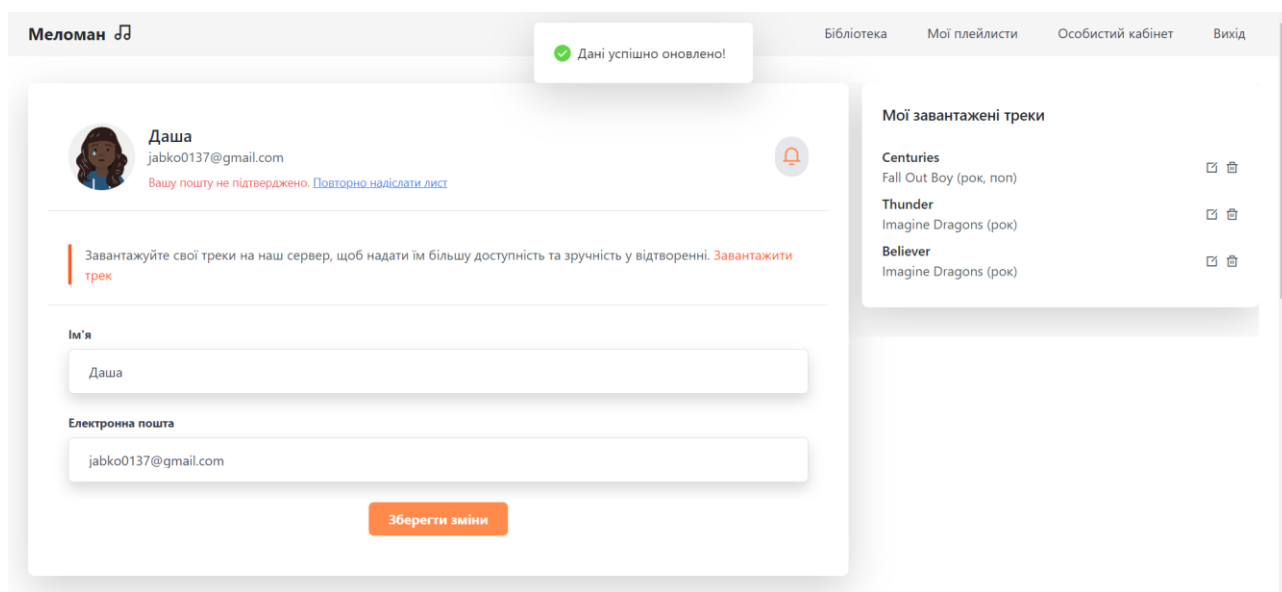


Рис. 3.17 – Оновлення даних користувача

Також ми бачимо, що у користувача не підтверджена пошта. Початково це відбувається при реєстрації, але якщо з якихось причин це не було зроблено, то тепер це можливо ще раз у особистому кабінеті. Натискаємо відповідне посилання та отримуємо лист для підтвердження за вказаною поштою (рис. 3.18). Переходимо за посиланням, авторизуємось повторно (рис. 3.19) і дивимось на зміни у особистому кабінеті (рис. 3.20):

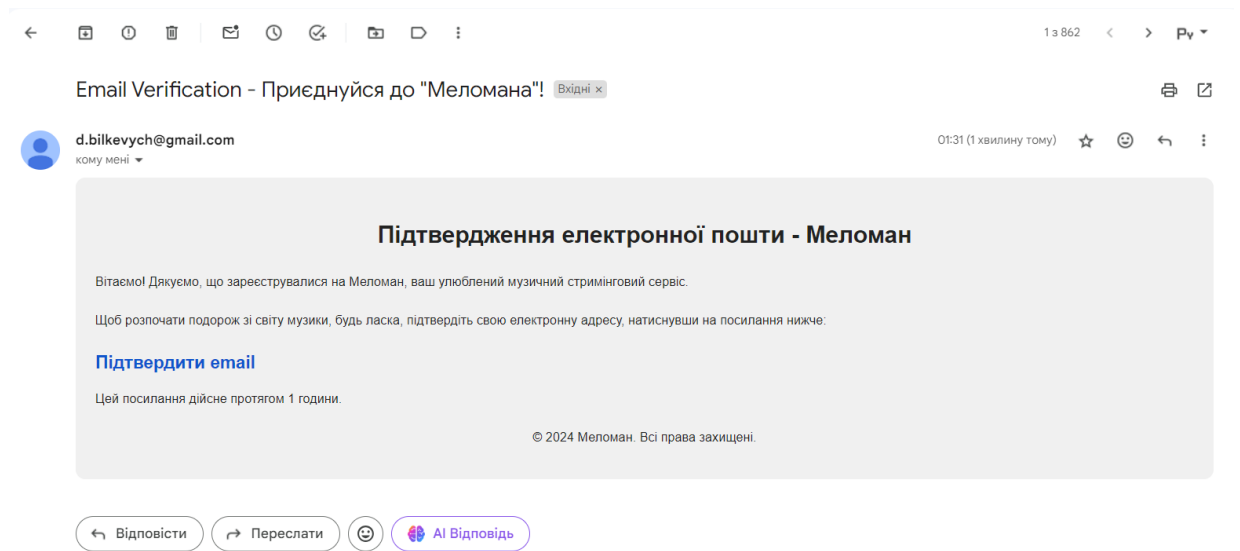


Рис. 3.18 – Отримуємо відповідного листа на пошті і переходимо за посиланням

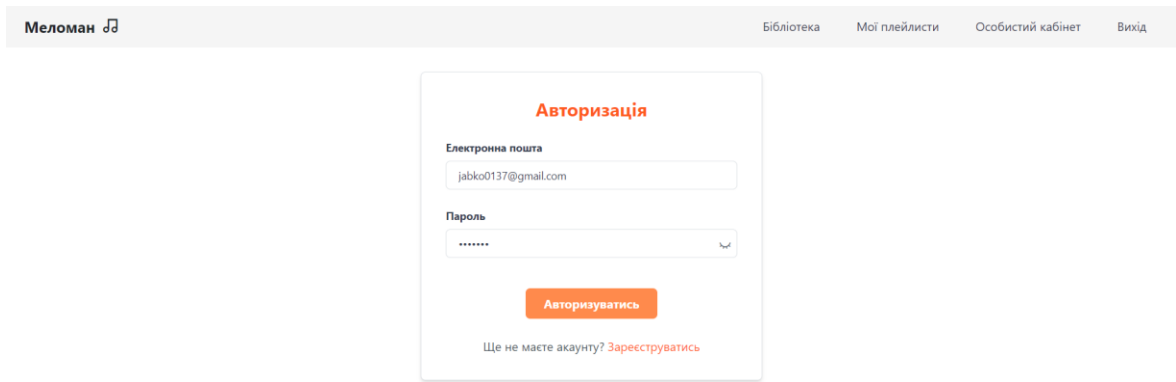


Рис. 3.19 – Авторизуємось на сервісі за вказаною поштою

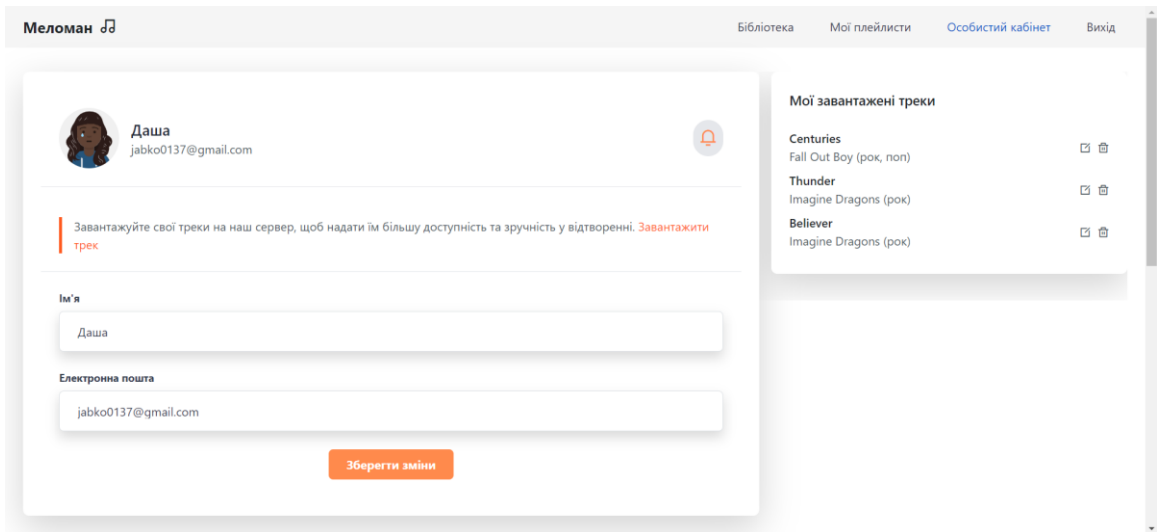


Рис. 3.20 – Оновлення профілю користувача (повідомлення про непідтверджену пошту зникло)

Також ми можемо завантажити трек (рис. 3.21), натиснувши відповідне посилання:

The screenshot shows a web form titled "Завантажити власний трек" (Upload your own track). The form includes the following fields and elements:

- Header: "Меломан" with a music note icon, and navigation links: "Бібліотека", "Мої плейлисти", "Особистий кабінет", "Вихід".
- Title: "Завантажити власний трек".
- Form fields:
 - "Назва пісні" (Song name): Text input with "Zeck".
 - "Виконавець" (Artist): Text input with "Echo".
 - "Жанр" (Genre): Dropdown menu with options: "рок", "поп", "інді", "джаз". "інді" is selected.
 - "Рік випуску" (Release year): Text input with "2024".
 - "Файл (MP3)": File upload area showing a successful upload of "Zeck - Echo.mp3" (2.38 MB). A button "Upload another?" is visible.
 - "Тривалість (у форматі 0.00)" (Duration): Text input with "2.38".

Рис. 3.21 – Сторінка завантаження треку, заповнення відповідних даних

Є можливість відредагувати трек, обравши відповідний трек у блоці «Мої завантажені треки» і натиснувши обрану іконку. Додамо ще один жанр до треку (рис. 3.22):

The screenshot shows a web form titled "Редагувати трек" (Edit track). The form includes the following fields and elements:

- Header: "Меломан" with a music note icon, and navigation links: "Бібліотека", "Мої плейлисти", "Особистий кабінет", "Вихід".
- Title: "Редагувати трек".
- Form fields:
 - "Назва пісні" (Song name): Text input with "Zeck".
 - "Виконавець" (Artist): Text input with "Echo".
 - "Жанр" (Genre): Dropdown menu with options: "рок", "поп", "інді", "джаз". "інді" is selected.
 - "Рік випуску" (Release year): Text input with "2024".
 - "Тривалість (у форматі 0.00)" (Duration): Text input with "2.38".
 - "Посилання на файл" (File link): Text input with "http://res.cloudinary.com/duyajhryn/raw/upload/v1718319144/musicapp/Zeck_-_Echo_me0h0p.mp3".

Рис. 3.22 – Сторінка редагування конкретного треку. Обираємо ще один жанр

Нижче у особистому кабінеті ми можемо переглянути статистику по найбільш прослуховуваним виконавцях за певний період (рис. 3.23), також статистику прослуховувань власних завантажених файлів (рис. 3.24):

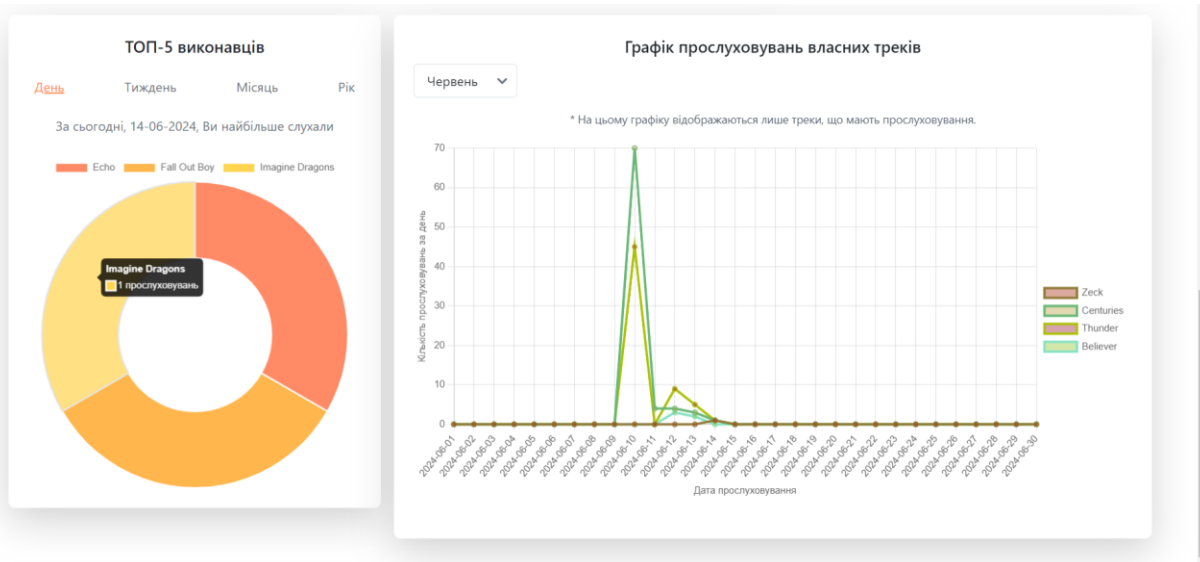


Рис. 3.23 – Статистичні дані по прослуховуванням користувача

Тепер спробуємо зайти в адмін-панель. Для цього, авторизуємось як адміністратор:

Меломан

Бібліотека Мої плейлисти Особистий кабінет Вихід

Авторизація адміністратора

Електронна пошта

ipz202_bdv@student.ztu.edu.ua

Пароль

Авторизуватись

Рис. 3.24 – Авторизація адміністратора сервісу

При успішній авторизації переходимо у адмін-панель, де бачимо загальну (рис. 3.25) та детальну (рис. 3.26), (рис. 3.27) статистики, а також можливість видалення файлів користувачів (рис. 3.28):

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		60

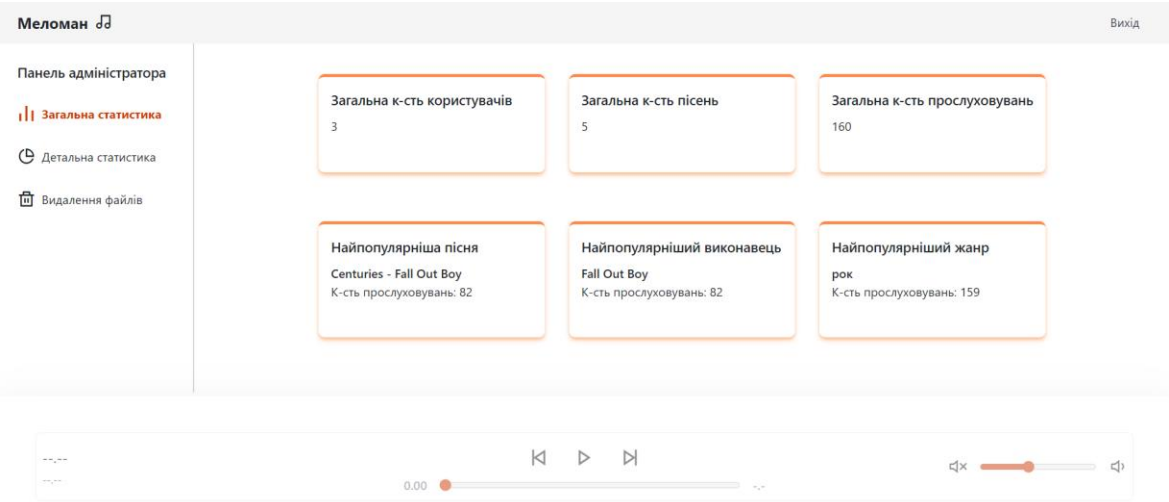


Рис. 3.25 – Загальна статистика сервісу

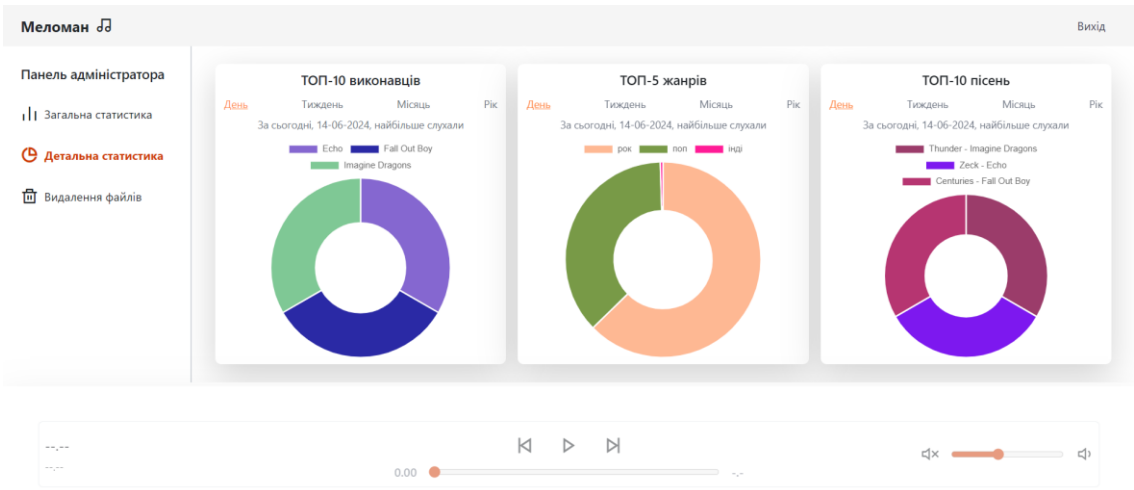


Рис. 3.26 – Детальна статистика сервісу, початок

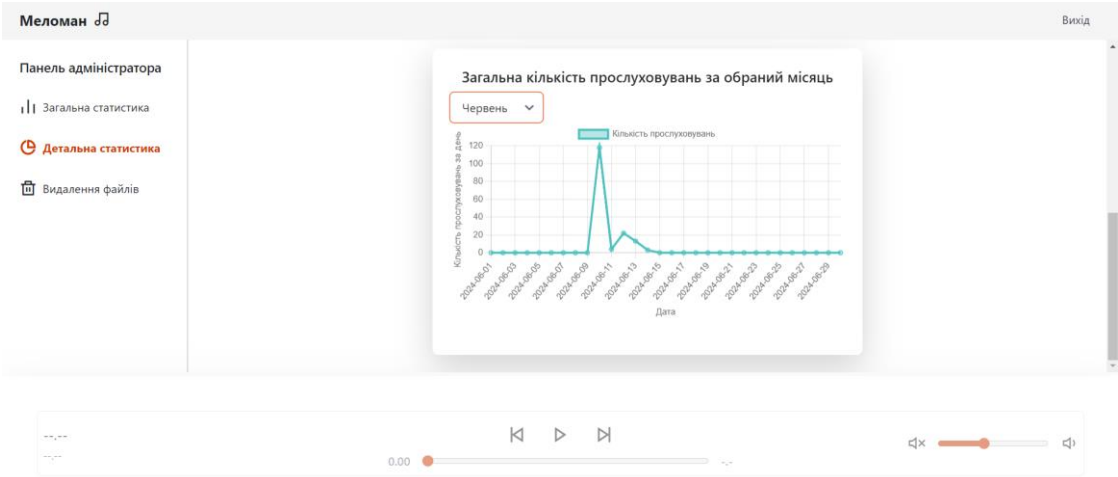


Рис. 3.27 – Детальна статистика сервісу, продовження

Далі оберемо на панелі «Видалення файлів». Адміністратор може тут прослуховувати треки і видалити трек з сервісу, пояснивши причину (рис. 3.28):

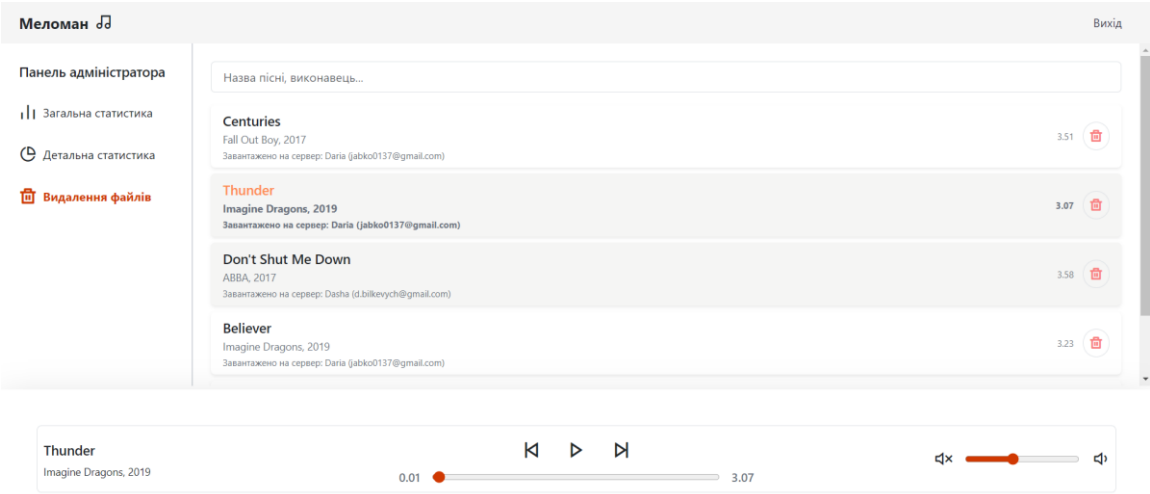


Рис. 3.28 – Видалення файлів

Обираємо трек і натискаємо на відповідну іконку, заповнюємо причину (рис. 3.29):

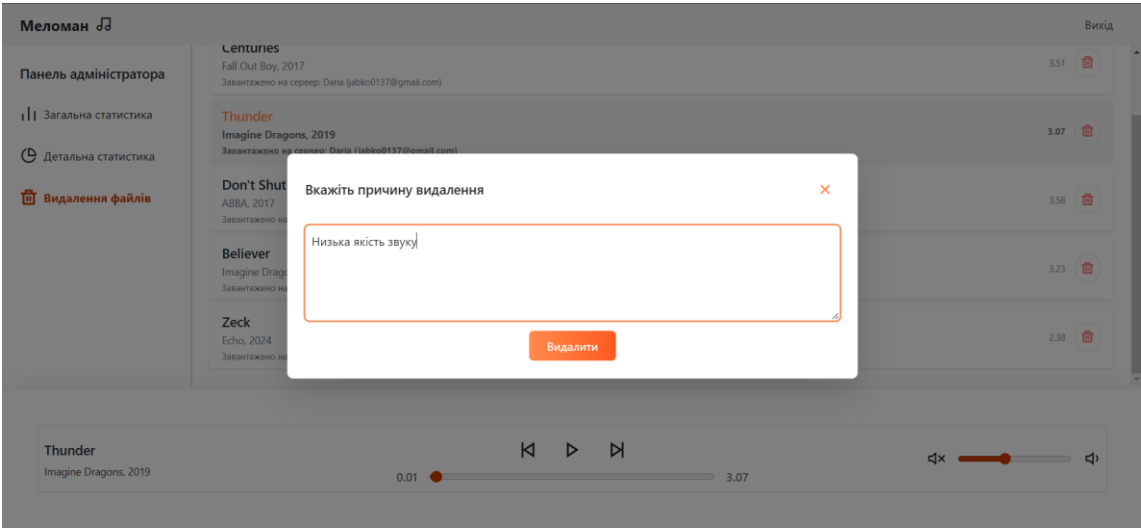


Рис. 3.29 – Модальне вікно для причини видалення файлу

При успішному видаленні файлу у користувача в особистому кабінеті з’явиться повідомлення (рис. 3.30):

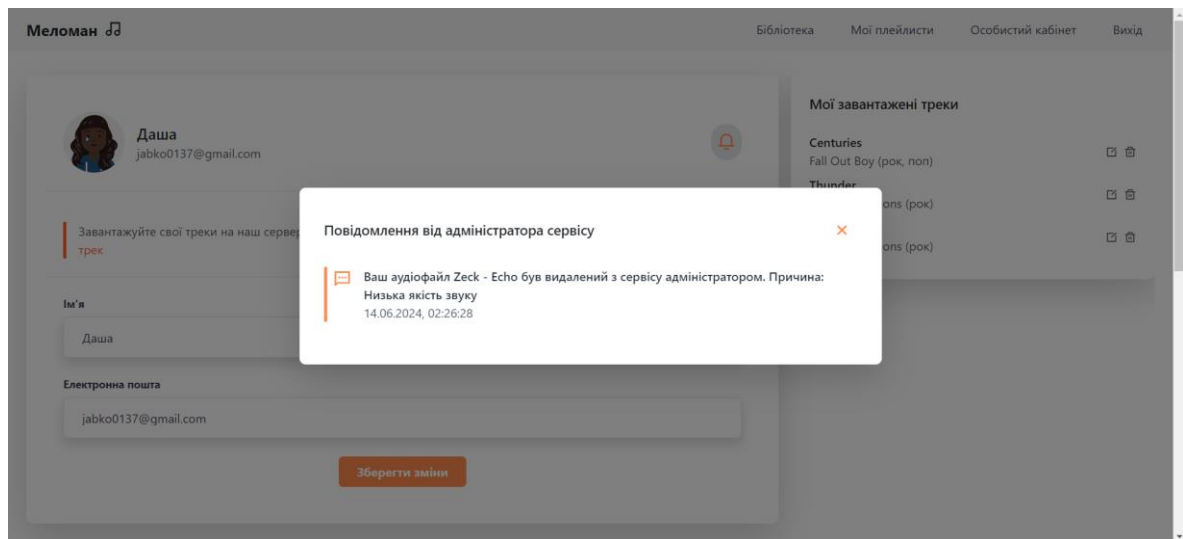


Рис. 3.30 – Отримання повідомлення про видалення файлу адміністратором

3.3 Тестування сервісу

Розглянемо процес тестування ключових аспектів нашої платформи, що включає в себе валідацію даних під час різних операцій та використання інструменту Postman [25] для тестування API.

Валідація даних є критично важливим аспектом тестування для будь-якої платформи або додатку. Цей процес спрямований на перевірку правильності введених користувачем даних перед їх обробкою і зберіганням у системі. Правильна валідація дозволяє запобігти можливим помилкам та забезпечити консистентність та надійність платформи.

Зупинимось на деяких типових помилках, які може допустити користувач: валідація полів (рис. 3.31), некоректні дані при реєстрація (рис. 3.32), неунікальна назва плейлиста (рис. 3.33), некоректні дані при завантаженні файлу (рис. 3.34), (рис. 3.35):

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		63

Меломан

Бібліотека

Реєстрація

Авторизація

Авторизація

Електронна пошта

Введіть вашу електронну пошту

Пароль

Заповніть це поле.

Авторизуватись

Ще не маєте акаунту? [Зареєструватись](#)

Рис. 3.31 – Валідація полів при авторизації

Меломан

Ця електронна адреса вже використовується! Авторизуйтеся, будь ласка!

Бібліотека

Реєстрація

Авторизація

Реєстрація

Ім'я

Test

Електронна пошта

jabko0137@gmail.com

Пароль

Зареєструватись

Вже маєте акаунт? [Авторизуватись](#)

Рис. 3.32 – При реєстрації вказали пошту, яка вже є у використанні

Меломан

Бібліотека

Міі плеєлисти

Особистий кабінет

Вихід

Створити плеєлист

Улюблене

Загальна кількість треків: 2

Щось цікаве !!!

Загальна кількість треків: 2

Плеєлист 1

Загальна кількість треків: 3

Особиста підбірка

Загальна кількість треків: 3

Створити новий плеєлист

Плеєлист 1

Плеєлист з такою назвою вже існує

Створити

0:00

⏮

⏪

⏩

⏭

⏮

⏪

⏩

⏭

Рис. 3.34 – Спроба створити плеєлист з неунікальною назвою

Меломан

Назва є обов'язковою та повинна містити від 1 до 20 символів

Бібліотека

Міі плеєлисти

Особистий кабінет

Вихід

Завантажити власний трек

Назва пісні

Виконавець

Echo

Жанр (можна обрати декілька)

рок

поп

інді

джаз

Рис. 3.34 – При завантаженні файлу певне поле було незаповнене

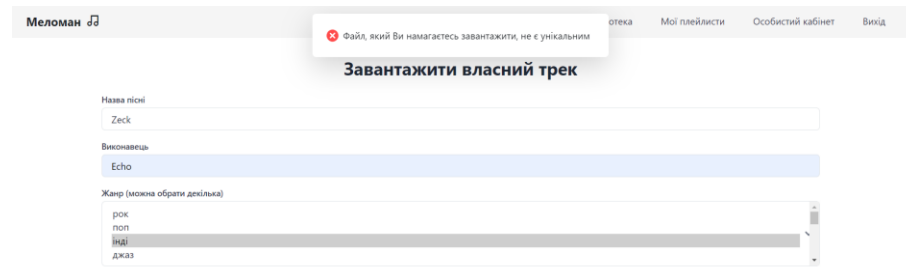


Рис. 3.35 – Спроба завантажити файл, який ідентичний тому, що вже є на сервісі, навіть якщо змінити всі інші параметри треку

Тестування платформи через використання інструменту Postman відіграє критичну роль у забезпеченні надійності та функціональності її API. Postman дозволяє розробникам створювати, виконувати та автоматизувати тестові сценарії для перевірки правильності роботи веб-сервісів перед впровадженням. Основні аспекти тестування включають функціональне тестування, під час якого перевіряється, чи повертають запити до API очікувані результати. Це охоплює перевірку відповідей, обробку помилок і стабільність API під навантаженням. Важливим елементом є також перевірка валідації даних, щоб забезпечити, що API обробляє різні типи даних належним чином, включаючи коректність форматування, обов'язкові поля та валідацію введених значень.

Під час розробки була створена колекція запитів, але зупинимось на даному прикладі (рис. 3.36), (рис. 3.37), (рис. 3.38):

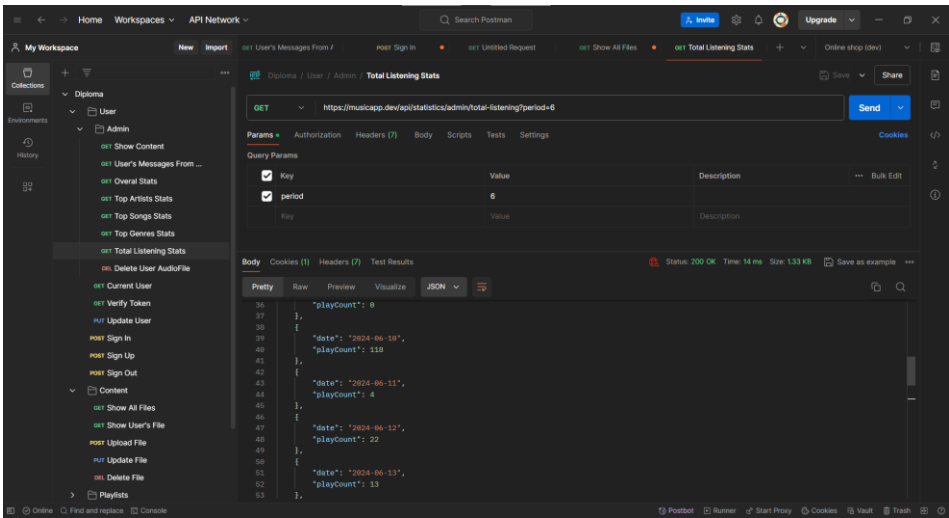


Рис. 3.36 – Виконання запиту на отримання статистичних даних адміністратора

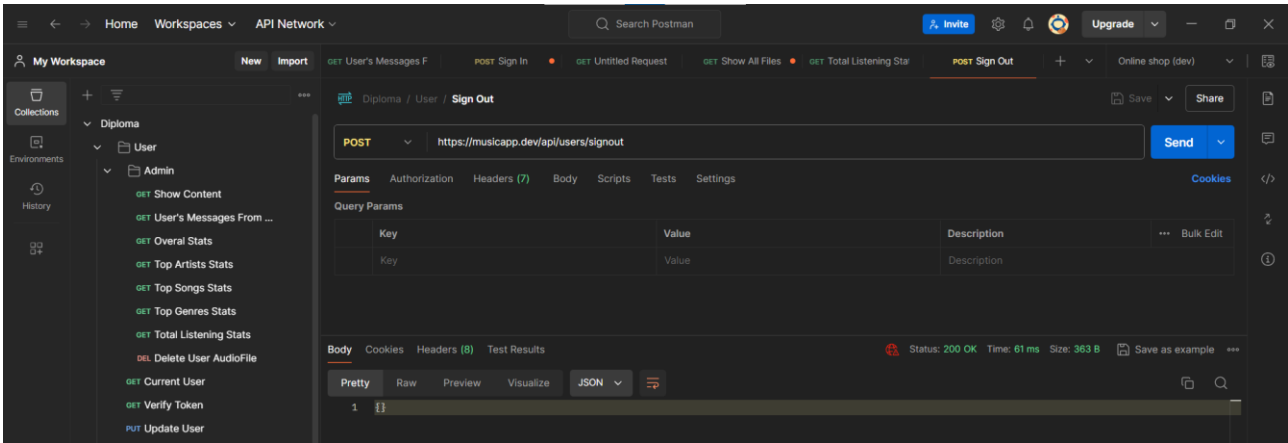


Рис. 3.37 – Виконання запиту на вихід з акаунту

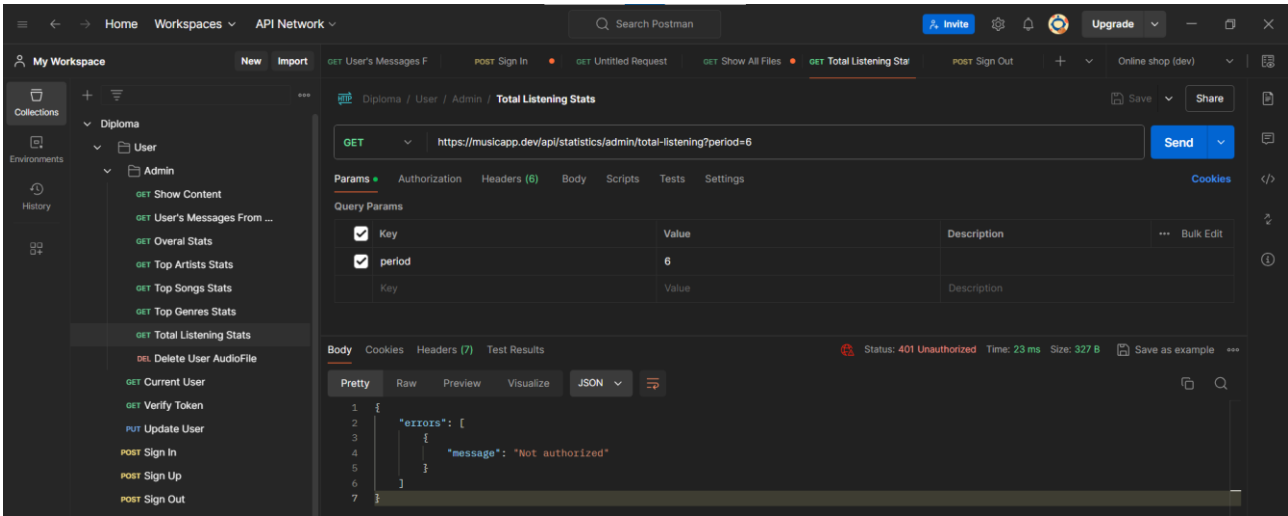


Рис. 3.38 – Повторна спроба для отримання статистичних даних адміністратора. Як наслідок – виведення помилки

Висновки до третього розділу

У третьому розділі було розглянуто процес встановлення та налаштування музичного стримінгового сервісу з використанням сучасних технологій Docker, Kubernetes та Skaffold. Цей підхід дозволяє забезпечити високу масштабованість, надійність і ефективність системи, забезпечуючи оптимальну інфраструктуру для модернізації веб-додатків.

Розглянуто структуру інтерфейсу музичного стримінгового сервісу, що включає гнучку навігацію, зручне керування контентом та інтеграцію із соціальними функціями, такими як створення плейлистів та персоналізовані налаштування. Різні ролі користувачів, включаючи звичайних користувачів

та адміністратора, мають доступ до різноманітних функцій, що сприяє високому рівню персоналізації та забезпечує відповідність їх потребам та очікуванням.

Проведено тестування ключових аспектів платформи, зокрема валідації даних та API через інструмент Postman. Цей процес підтвердив надійність та стабільність платформи, забезпечивши коректну обробку введених даних і правильність виконання API-запитів під різними умовами використання.

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		67

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено музичний стримінговий сервіс, який відповідає сучасним вимогам до веб-додатків, забезпечуючи високу масштабованість, надійність та ефективність. Поставлені завдання включали аналіз аналогів програмного продукту, вибір архітектури, проектування системи, реалізацію функціоналу та тестування платформи.

У першому розділі було розглянуто основні завдання для розробки музичного стримінгового сервісу та аналіз існуючих рішень на ринку, таких як Spotify, Apple Music, YouTube Music та SoundCloud. Було досліджено переваги мікросервісної архітектури та вибрано відповідні технології для реалізації серверної та клієнтської частин сервісу. Вибір TypeScript, Express.js та Next.js забезпечив зручний та ефективний інтерфейс користувача, а MongoDB гарантує гнучкість та масштабованість збереження даних.

У другому розділі було визначено функціональні та нефункціональні вимоги до сервісу, спроектовано об'єктно-орієнтовану структуру системи на основі мікросервісної архітектури. Було розроблено комплексну структуру баз даних. Реалізація алгоритмів роботи сервісу та використання асинхронної комунікації між сервісами забезпечили ефективну обробку запитів користувачів та управління даними.

У третьому розділі було розглянуто порядок встановлення та налаштування сервісу з використанням Docker, Kubernetes та Skaffold, що забезпечує оптимальну інфраструктуру для сучасних веб-додатків. Було описано структуру інтерфейсу сервісу, який включає гнучку навігацію, зручне керування контентом та інтеграцію із соціальними функціями. Особлива увага була приділена різним ролям користувачів, включаючи звичайних користувачів та адміністратора, що забезпечує високий рівень персоналізації та відповідність їхнім потребам. Тестування платформи, включаючи валідацію даних та API через Postman, підтвердило її надійність та стабільність.

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		68

В загальному підсумку, було успішно розроблено музичний стримінговий сервіс, що відповідає сучасним вимогам, забезпечуючи високу якість обслуговування користувачів та ефективне управління контентом.

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		69

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Spotify [Електронний ресурс] – Режим доступу до ресурсу: <https://www.spotify.com/ua-uk/free/>
2. Apple Music [Електронний ресурс] – Режим доступу до ресурсу: <https://music.apple.com/ua/browse?l=uk>
3. YouTube Music [Електронний ресурс] – Режим доступу до ресурсу: <https://music.youtube.com/>
4. SoundCloud [Електронний ресурс] – Режим доступу до ресурсу: <https://soundcloud.com/>
5. Microservices vs. Monolithic Architecture [Електронний ресурс] – Режим доступу до ресурсу: <https://www.atlassian.com/microservices/microservices-architecture>
6. Serverless Framework [Електронний ресурс] – Режим доступу до ресурсу: <https://www.serverless.com/framework/docs>
7. TypeScript [Електронний ресурс] – Режим доступу до ресурсу: <https://www.typescriptlang.org/>
8. Express [Електронний ресурс] – Режим доступу до ресурсу: <https://expressjs.com/>
9. Next.js [Електронний ресурс] – Режим доступу до ресурсу: <https://nextjs.org/docs>
10. MongoDB [Електронний ресурс] - Режим доступу до ресурсу: <https://www.mongodb.com/docs/>
11. Kubernetes [Електронний ресурс] - Режим доступу до ресурсу: <https://kubernetes.io/docs/home/>
12. Docker [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.docker.com/>
13. Skaffold [Електронний ресурс] – Режим доступу до ресурсу: <https://skaffold.dev/docs/>

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		70

14. NGINX Ingress Controller [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.nginx.com/nginx-ingress-controller/>
15. NATS Streaming Server [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.nats.io/>
16. Building microservice architecture with NATS [Електронний ресурс] - Режим доступу до ресурсу: <https://nats.io/blog/building-scalable-microservices-with-nats/>
17. UML Use Case Diagram [Електронний ресурс] - Режим доступу до ресурсу: <https://www.uml-diagrams.org/use-case-diagrams.html>
18. UML Activity Diagram [Електронний ресурс] - Режим доступу до ресурсу: <https://www.uml-diagrams.org/activity-diagrams.html>
19. Multer [Електронний ресурс] - Режим доступу до ресурсу: <https://expressjs.com/en/resources/middleware/multer.html>
20. Cloudinary for Node.js [Електронний ресурс] - Режим доступу до ресурсу: https://cloudinary.com/documentation/node_integration
21. Building your own hooks [Електронний ресурс] - Режим доступу до ресурсу: <https://legacy.reactjs.org/docs/hooks-custom.html>
22. FormData [Електронний ресурс] - Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/API/FormData>
23. What is container orchestration? [Електронний ресурс] - Режим доступу до ресурсу: <https://www.redhat.com/en/topics/containers/what-is-container-orchestration>
24. 10 Usability Heuristics for User Interface Design - [Електронний ресурс] - Режим доступу до ресурсу: <https://www.nngroup.com/articles/ten-usability-heuristics/>
25. Postman [Електронний ресурс] - Режим доступу до ресурсу: <https://learning.postman.com/docs/introduction/overview/>

ДОДАТКИ

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		72

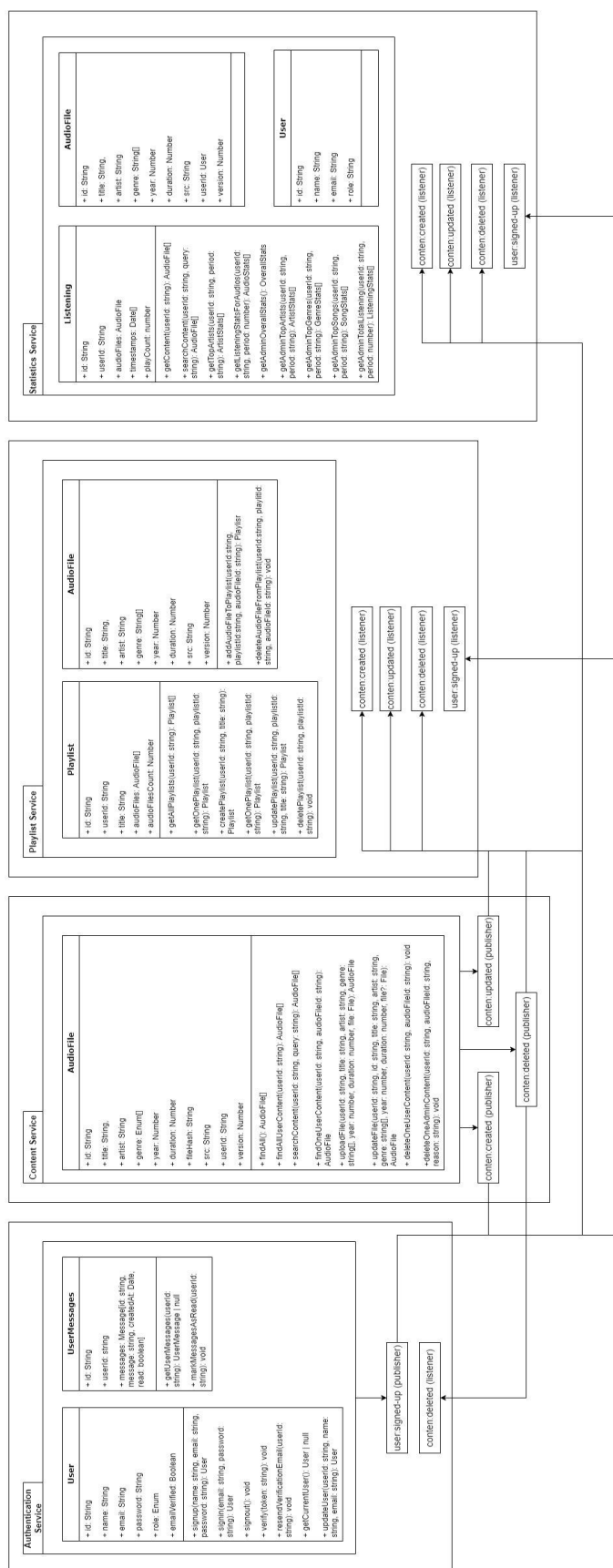


Рис. А.1 – Діаграма класів та взаємодій мікросервісної системи

Scaffold конфігурація для розгортання мікросервісів у Kubernetes

```
apiVersion: skaffold/v4beta9
kind: Config
build:
  artifacts:
    - image: dariabilkevych/authentication
      context: authentication
      sync:
        manual:
          - src: src/**/*.ts
            dest: .
      docker:
        dockerfile: Dockerfile
    - image: dariabilkevych/client
      context: client
      sync:
        manual:
          - src: '**/*.js'
            dest: .
      docker:
        dockerfile: Dockerfile
    - image: dariabilkevych/content
      context: content
      sync:
        manual:
          - src: src/**/*.ts
            dest: .
      docker:
        dockerfile: Dockerfile
    - image: dariabilkevych/playlists
      context: playlists
      sync:
        manual:
          - src: src/**/*.ts
            dest: .
      docker:
        dockerfile: Dockerfile
    - image: dariabilkevych/statistics
      context: statistics
      sync:
        manual:
          - src: src/**/*.ts
            dest: .
      docker:
        dockerfile: Dockerfile
  local:
    push: false
manifests:
```

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ІЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		74

```
rawYaml:
  - ./infra/k8s/*
deploy:
  kubect1: {}
```

Конфігураційний файл Kubernetes Ingress для маршрутизації

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-service
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/use-regex: 'true'
    nginx.ingress.kubernetes.io/proxy-body-size: '1024m'
spec:
  rules:
    - host: musicapp.dev
      http:
        paths:
          - path: /api/users/(?.*)
            pathType: ImplementationSpecific
            backend:
              service:
                name: authentication-srv
                port:
                  number: 3000
          - path: /api/content/(?.*)
            pathType: ImplementationSpecific
            backend:
              service:
                name: content-srv
                port:
                  number: 3000
          - path: /api/playlists/(?.*)
            pathType: ImplementationSpecific
            backend:
              service:
                name: playlists-srv
                port:
                  number: 3000
          - path: /api/statistics/(?.*)
            pathType: ImplementationSpecific
            backend:
              service:
                name: statistics-srv
                port:
                  number: 3000
          - path: /?(.*)
            pathType: ImplementationSpecific
```

```

backend:
  service:
    name: client-srv
    port:
      number: 3000

```

Модель користувача

```

import mongoose from 'mongoose';
import { Password } from '../services/password';
import { UserRoles } from '@dbmusicapp/common';

interface UserAttrs {
  name: string;
  email: string;
  password: string;
  role: UserRoles;
  emailVerified: boolean;
}

interface UserModel extends mongoose.Model<UserDoc> {
  build(attrs: UserAttrs): UserDoc;
}

interface UserDoc extends mongoose.Document {
  name: string;
  email: string;
  password: string;
  role: UserRoles;
  emailVerified: boolean;
}

const userSchema = new mongoose.Schema(
  {
    name: {
      type: String,
      required: true,
    },
    email: {
      type: String,
      required: true,
    },
    password: {
      type: String,
      required: true,
    },
    role: {
      type: String,
      required: true,
      enum: Object.values(UserRoles),
    },
  }
);

```

```

        default: UserRoles.User,
      },
      emailVerified: {
        type: Boolean,
        default: false,
      },
    },
  },
  {
    toJSON: {
      transform(doc, ret) {
        ret.id = ret._id;
        delete ret._id;
        delete ret.password;
        delete ret.__v;
      },
    },
  },
}
);

userSchema.pre('save', async function (done) {
  if (this.isModified('password')) {
    const hashed = await Password.toHash(this.get('password'));
    this.set('password', hashed);
  }
  done();
});

userSchema.statics.build = (attrs: UserAttrs) => {
  return new User(attrs);
};

const User = mongoose.model<UserDoc, UserModel>('User', userSchema);

export { User };

```

Модель аудіофайлу

```

import mongoose from 'mongoose';
import { updateIfCurrentPlugin } from 'mongoose-update-if-current';
import { Genre } from '@dbmusicapp/common';

interface AudioFileAttrs {
  title: string;
  artist: string;
  genre: Genre[];
  year: number;
  duration: number;
  fileHash: string;
  src: string;
  userId: string;
}

```

```

}

interface AudioFileDoc extends mongoose.Document {
  title: string;
  artist: string;
  genre: Genre[];
  year: number;
  duration: number;
  fileHash: string;
  src: string;
  userId: string;
  version: number;
}

interface AudioFileModel extends mongoose.Model<AudioFileDoc> {
  build(attrs: AudioFileAttrs): AudioFileDoc;
}

const audioFileSchema = new mongoose.Schema(
{
  title: {
    type: String,
    required: true,
  },
  artist: {
    type: String,
    required: true,
  },
  genre: {
    type: [
      {
        type: String,
        enum: Object.values(Genre),
        required: true,
      },
    ],
    required: true,
  },
  year: {
    type: Number,
    required: true,
  },
  duration: {
    type: Number,
    required: true,
  },
  fileHash: {
    type: String,
  },
  src: {

```

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ІЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		78

```

        type: String,
        required: true,
      },
      userId: {
        type: String,
        required: true,
      },
    },
  },
  {
    toJSON: {
      transform(doc, ret) {
        ret.id = ret._id;
        delete ret._id;
      },
    },
  },
}
);

audioFileSchema.set('versionKey', 'version');
audioFileSchema.plugin(updateIfCurrentPlugin);

audioFileSchema.statics.build = (attrs: AudioFileAttrs) => {
  return new AudioFile(attrs);
};

const AudioFile = mongoose.model<AudioFileDoc, AudioFileModel>(
  'AudioFile',
  audioFileSchema
);

export { AudioFile };

```

Модель плейлиста

```

import mongoose from 'mongoose';
import { AudioFileDoc } from './audio-file';

interface PlaylistAttrs {
  userId: string;
  title: string;
  audioFiles: AudioFileDoc[] | any;
}

interface PlaylistDoc extends mongoose.Document {
  userId: string;
  title: string;
  audioFiles: AudioFileDoc[] | any;
  audioFilesCount: number;
}

```

```

interface PlaylistModel extends mongoose.Model<PlaylistDoc> {
  build(attrs: PlaylistAttrs): PlaylistDoc;
}

const playlistSchema = new mongoose.Schema(
  {
    userId: {
      type: String,
      required: true,
    },
    title: {
      type: String,
      required: true,
    },
    audioFiles: {
      type: [mongoose.Schema.Types.ObjectId],
      default: [],
      ref: 'AudioFile',
    },
    audioFilesCount: {
      type: Number,
      default: 0,
    },
  },
  {
    toJSON: {
      transform(doc, ret) {
        ret.id = ret._id;
        delete ret._id;
      },
    },
  }
);

playlistSchema.statics.build = (attr: PlaylistAttrs) => {
  return new Playlist(attr);
};

const Playlist = mongoose.model<PlaylistDoc, PlaylistModel>(
  'Playlist',
  playlistSchema
);

export { Playlist };

```

Модель прослуховувань

```

import mongoose from 'mongoose';
import { AudioFileDoc } from './audio-file';

```

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		80

```

interface ListeningAttrs {
  userId: string;
  audioFileId: AudioFileDoc;
  timestamps: Date[];
  playCount: number;
}

interface ListeningDoc extends mongoose.Document {
  _id: mongoose.Types.ObjectId;
  userId: string;
  audioFileId: AudioFileDoc;
  timestamps: Date[];
  playCount: number;
}

interface ListeningModel extends mongoose.Model<ListeningDoc> {
  build(attrs: ListeningAttrs): ListeningDoc;
}

const listeningSchema = new mongoose.Schema({
  userId: {
    type: String,
    required: true,
  },
  audioFileId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'AudioFile',
  },
  timestamps: [
    {
      type: Date,
    },
  ],
  playCount: {
    type: Number,
    default: 0,
  },
});

listeningSchema.statics.build = (attr: ListeningAttrs) => {
  return new Listening(attr);
};

const Listening = mongoose.model<ListeningDoc, ListeningModel>(
  'Listening',
  listeningSchema
);

export { Listening };

```

Функція для отримання поточного користувача

```
import { Request, Response, NextFunction } from 'express';
import jwt from 'jsonwebtoken';

interface UserPayload {
  id: string;
  name: string;
  email: string;
  role: string;
  emailVerified: boolean;
}

declare global {
  namespace Express {
    interface Request {
      currentUser?: UserPayload;
    }
  }
}

export const currentUser = (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  if (!req.session?.jwt) {
    return next();
  }

  try {
    const payload = jwt.verify(
      req.session.jwt,
      process.env.JWT_KEY!
    ) as UserPayload;
    req.currentUser = payload;
  } catch (err) {}

  next();
};
```

Файл з основним кодом Express додатку у мікросервісі Authentication

```
import express from 'express';
import 'express-async-errors';
import { json } from 'body-parser';
import cookieSession from 'cookie-session';
import dotenv from 'dotenv';
import { errorHandler, NotFoundError } from '@dbmusicapp/common';
```

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		82

```

import { signinRouter } from './routes/auth/signin';
import { signoutRouter } from './routes/auth/signout';
import { signupRouter } from './routes/auth/signup';

import { currentUserRouter } from './routes/user/current-user';
import { updateUserRouter } from './routes/user/update';
import { getUserMessagesRouter } from './routes/user/show-messages';
import { markMessagesAsReadRouter } from './routes/user/read-messages';
import { getAllUsersRouter } from './routes/user/show-all';

import { emailVerificationRouter } from './routes/auth/verify-email';
import { resendVerificationEmailRouter } from './routes/auth/resend-verification-
email';

const app = express();
app.set('trust proxy', true);
app.use(json());
app.use(
  cookieSession({
    signed: false,
    secure: process.env.NODE_ENV !== 'test',
  })
);

dotenv.config();

app.use(signinRouter);
app.use(signoutRouter);
app.use(signupRouter);

app.use(currentUserRouter);
app.use(updateUserRouter);
app.use(getUserMessagesRouter);
app.use(markMessagesAsReadRouter);
app.use(getAllUsersRouter);

app.use(emailVerificationRouter);
app.use(resendVerificationEmailRouter);

app.all('*', async (req, res) => {
  throw new NotFoundError();
});

app.use(errorHandler);

export { app };

```

Файл для запуску сервера та підключення до MongoDB і NATS у мікросервісі

Authentication

```
import mongoose from 'mongoose';
import { app } from './app';
import { natsWrapper } from './nats-wrapper';
import { ContentDeletedListener } from './events/listeners/content-deleted-listener';

const start = async () => {
  if (!process.env.JWT_KEY) {
    throw new Error('JWT_KEY must be defined');
  }

  if (!process.env.MONGO_URI) {
    throw new Error('MONGO_URI must be defined');
  }

  if (!process.env.NATS_CLIENT_ID) {
    throw new Error('NATS_CLIENT_ID must be defined');
  }

  if (!process.env.NATS_URL) {
    throw new Error('NATS_URL must be defined');
  }

  if (!process.env.NATS_CLUSTER_ID) {
    throw new Error('NATS_CLUSTER_ID must be defined');
  }

  try {
    await natsWrapper.connect(
      process.env.NATS_CLUSTER_ID,
      process.env.NATS_CLIENT_ID,
      process.env.NATS_URL
    );
    natsWrapper.client.on('close', () => {
      console.log('NATS connection closed');
      process.exit();
    });
    process.on('SIGINT', () => natsWrapper.client.close());
    process.on('SIGTERM', () => natsWrapper.client.close());

    new ContentDeletedListener(natsWrapper.client).listen();

    await mongoose.connect(process.env.MONGO_URI);
    console.log('Connected to MongoDB!');
  } catch (err) {
    console.error(err);
  }
}
```

		Бількевич Д.В.			ІПЗ.КР.Б-121-24-ІЗ	Арк.
		Єфремов Ю.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		84

```

app.listen(3000, () => {
  console.log('Listening on port 3000!');
});

start();

```

Головний компонент додатку, який відповідає за рендеринг інтерфейсу

```

import 'bootstrap/dist/css/bootstrap.css';
import 'remixicon/fonts/remixicon.css';
import '../styles/globals.css';
import { Transition } from '@headlessui/react';
import { Toaster, ToastIcon, resolveValue } from 'react-hot-toast';
import { PlayerProvider } from '../components/player-context';
import buildClient from '../api/build-client';
import Header from '../components/header';
import Player from '../components/player';
import AdminSidebar from '../components/admin-sidebar';

const AppComponent = ({ Component, pageProps, currentUser }) => {
  const excludePlayer = Component.excludePlayer || false;
  const isAdmin = currentUser && currentUser.role === 'admin';

  return (
    <div className="flex flex-col h-screen">
      <Toaster position="top-center" reverseOrder={true}>
        {(t) => (
          <Transition
            as="div"
            appear
            show={t.visible}
            className="transform p-4 flex bg-white rounded shadow-lg"
            enter="transition-all duration-150"
            enterFrom="opacity-0 scale-50"
            enterTo="opacity-100 scale-100"
            leave="transition-all duration-150"
            leaveFrom="opacity-100 scale-100"
            leaveTo="opacity-0 scale-75"
          >
            <ToastIcon toast={t} />
            <p className="px-2">{resolveValue(t.message)}</p>
          </Transition>
        )}
      </Toaster>
      <Header currentUser={currentUser} />
      <div className="flex flex-grow">
        {isAdmin && <AdminSidebar />}
        <div className="flex-1">

```

```

        <PlayerProvider>
          <Component currentUser={currentUser} {...pageProps} />
          <Player isVisible={!excludePlayer} currentUser={currentUser} />
        </PlayerProvider>
      </div>
    </div>
  </div>
);
};

AppComponent.getInitialProps = async (appContext) => {
  const client = buildClient(appContext.ctx);
  const { data } = await client.get('/api/users/currentuser');

  let pageProps = {};
  if (appContext.Component.getInitialProps) {
    pageProps = await appContext.Component.getInitialProps(
      appContext.ctx,
      client,
      data.currentUser
    );
  }

  return {
    pageProps,
    ...data,
  };
};

export default AppComponent;

```

Реалізація плеєру

```

import React, { useEffect, useRef, useContext, useState } from 'react';
import { PlayerContext } from './player-context';
import axios from 'axios';
import { toast } from 'react-hot-toast';
import { UserRoles } from '@dbmusicapp/common';

const Player = ({ isVisible, currentUser }) => {
  const audioRef = useRef();
  const {
    currentSong,
    setCurrentSong,
    isPlaying,
    setIsPlaying,
    currentTime,
    setCurrentTime,
    volume,
    setVolume,
  } = useContext(PlayerContext);

```

```

    content,
    setContent,
  } = useContext(PlayerContext);
  const [isFavorite, setIsFavorite] = useState(false);
  const [favoritePlaylistId, setFavoritePlaylistId] = useState(null);

  const recordPlayback = async (audioFileId) => {
    try {
      await axios.post('/api/statistics/playback', { audioFileId });
    } catch (error) {
      console.error('Error recording playback:', error);
    }
  };

  useEffect(() => {
    if (currentSong && audioRef.current) {
      setIsPlaying(true);
      audioRef.current.play();
      recordPlayback(currentSong.id); // Record playback when a new song starts
    }
  }, [currentSong, setIsPlaying]);

  const handlePlayPause = () => {
    if (audioRef.current.paused) {
      setIsPlaying(true);
      audioRef.current.play();
    } else {
      setIsPlaying(false);
      audioRef.current.pause();
    }
  };

  const handleSkip = (direction) => {
    const currentIndex = content.findIndex(
      (song) => song.id === currentSong?.id
    );
    const nextIndex =
      direction === 'prev'
        ? currentIndex === 0
          ? content.length - 1
          : currentIndex - 1
        : currentIndex === content.length - 1
          ? 0
          : currentIndex + 1;

    setCurrentSong(content[nextIndex]);
  };

  const handleEnded = () => {
    const currentIndex = content.findIndex(

```

```

        (song) => song.id === currentSong?.id
    );
    const nextIndex = (currentIndex + 1) % content.length;
    setCurrentSong(content[nextIndex]);
};

useEffect(() => {
    const fetchAndCheckFavoritePlaylist = async () => {
        if (!currentSong || !currentUser) {
            setIsFavorite(false);
            return;
        }

        try {
            const response = await axios.get('/api/playlists');
            if (response.data) {
                const favoritePlaylist = response.data.find(
                    (playlist) => playlist.title === 'Улюблене'
                );
                if (favoritePlaylist) {
                    setFavoritePlaylistId(favoritePlaylist.id);

                    const favoritePlaylistAudioFiles = favoritePlaylist.audioFiles;
                    const isCurrentSongInFavorites = favoritePlaylistAudioFiles.some(
                        (audioFile) => audioFile.id === currentSong.id
                    );
                    setIsFavorite(isCurrentSongInFavorites);
                }
            }
        } catch (error) {
            console.error('Error fetching playlists:', error);
        }
    };

    fetchAndCheckFavoritePlaylist();
}, [currentSong, currentUser]);

const handleFavoriteToggle = async () => {
    if (isFavorite) {
        await axios.delete(
            `/api/playlists/${favoritePlaylistId}/${currentSong.id}`
        );
        setIsFavorite(false);
        toast.success('Успішно видалено з "Улюблене"');
        setContent((prevContent) =>
            prevContent.filter((song) => song.id !== currentSong.id)
        );
    } else {
        await axios.post(`/api/playlists/${favoritePlaylistId}/add-audio`, {
            audioFileId: currentSong.id,

```

```

    });
    setIsFavorite(true);
    toast.success('Успішно додано до "Улюбене"');
    setContent((prevContent) => [...prevContent, currentSong]);
  }
};

useEffect(() => {
  if (audioRef.current) {
    audioRef.current.volume = volume;
  }
}, [volume]);

return (
  <div
    className={`absolute bottom-0 left-0 right-0 p-5 shadow-lg bg-white ${
      !currentSong && 'pointer-events-none opacity-50'
    } ${!isVisible ? 'hidden' : ''}`}
  >
    <div className="flex items-center border rounded-md p-2">
      <div className="flex items-center w-1/3">
        <div>
          <h5 className="text-lg font-medium mb-1">
            {currentSong ? currentSong.title : '---'}
          </h5>
          <p className="text-sm">
            {currentSong
              ? `${currentSong.artist}, ${currentSong.year}`
              : '---'}
          </p>
        </div>
        {currentUser && currentUser.role === UserRoles.User && (
          <div className="ml-7">
            {isFavorite ? (
              <i
                className="ri-heart-3-fill text-2xl text-orange-500 cursor-
pointer"
                onClick={handleFavoriteToggle}
              />
            ) : (
              <i
                className="ri-heart-3-line text-2xl cursor-pointer hover:text-
gray-500"
                onClick={handleFavoriteToggle}
              />
            )}
          </div>
        )}
      </div>
    </div>
  </div>
);

```

```

<div className="flex flex-col items-center w-1/3">
  <audio
    src={currentSong?.src}
    ref={audioRef}
    onTimeUpdate={(e) => setCurrentTime(e.target.currentTime)}
    onEnded={handleEnded}
  />
  <div className="flex justify-center gap-7">
    <i
      className="ri-skip-back-line ri-2x cursor-pointer hover:text-gray-
500"
      onClick={() => handleSkip('prev')}
    />
    <i
      className={`ri-${
        isPlaying ? 'pause' : 'play'
      }-line ri-2x cursor-pointer hover:text-gray-500`}
      onClick={handlePlayPause}
    />
    <i
      className="ri-skip-forward-line ri-2x cursor-pointer hover:text-
gray-500"
      onClick={() => handleSkip('next')}
    />
  </div>
  <div className="flex justify-between items-center w-full gap-3">
    <div className="flex items-center">
      <p className="text-base font-medium text-gray-500">
        {Math.floor(currentTime / 60)}.
        {Math.floor(currentTime % 60)
          .toString()
          .padStart(2, '0')}
      </p>
    </div>
    <input
      type="range"
      className="w-full range pr-6 accent-orange-600"
      min={0}
      max={currentSong?.duration * 60 || 0}
      value={currentTime}
      onChange={(e) => {
        audioRef.current.currentTime = e.target.value;
        setCurrentTime(e.target.value);
      }}
    />
    <div className="flex items-center">
      <p className="text-base font-medium text-gray-500">
        {currentSong ? currentSong.duration : '-.-'}
      </p>
    </div>
  </div>

```

```

        </div>
    </div>

    <div className="flex gap-3 items-center justify-end w-1/3">
        <i
            className="ri-volume-mute-line text-2xl cursor-pointer hover:text-
gray-500"
            onClick={() => {
                setVolume(0);
                audioRef.current.volume = 0;
            }}
        />
        <input
            type="range"
            className="pr-6 accent-orange-600"
            min={0}
            max={1}
            step={0.1}
            value={volume}
            onChange={(e) => {
                audioRef.current.volume = e.target.value;
                setVolume(e.target.value);
            }}
        />
        <i
            className="ri-volume-down-line text-2xl cursor-pointer hover:text-
gray-500"
            onClick={() => {
                setVolume(1);
                audioRef.current.volume = 1;
            }}
        />
    </div>
</div>
</div>
);
};

export default Player;

```