

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:

**«Інформаційна система організації прослуховування  
аудіоконтенту»**

Виконав студент групи ПЗ-22-3  
ВАСЯНОВИЧ Олександр Андрійович

Керівник роботи:  
МОРОЗОВ Андрій Васильович

Рецензент:  
ЛОКТИКОВА Тамара Миколаївна

Житомир – 2026

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

**«ЗАТВЕРДЖУЮ»**

Зав. кафедри інженерії  
програмного забезпечення  
Тетяна Вакалюк

**«28» лютого 2026 р.**

**ЗАВДАННЯ**  
**на кваліфікаційну роботу**

Здобувач вищої освіти: **ВАСЯНОВИЧ Олександр Андрійович**

Керівник роботи: **МОРОЗОВ Андрій Васильович**

Тема роботи: «Інформаційна система організації прослуховування аудіоконтенту»,

затверджена Наказом закладу вищої освіти **№ 92/с** від **«27» лютого 2026 р.**,

Вихідні дані для роботи: процес розповсюдження та споживання цифрового аудіоконтенту в мережевому середовищі: методи, моделі та програмні засоби побудови інформаційної системи для стрімінгу аудіоконтенту.

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

| Розділ | Консультант  | Завдання видав | Завдання прийняв |
|--------|--------------|----------------|------------------|
| 1      | Морозов А.В. | 01.03.2026р.   | 01.03.2026р      |
| 2      | Морозов А.В. | 01.03.2026р    | 01.03.2026р      |
| 3      | Морозов А.В. | 01.03.2026р    | 01.03.2026р      |

## РЕФЕРАТ

Пояснювальна записка до випускної кваліфікаційної роботи бакалавра містить 119 сторінок, 38 ілюстрацій, 14 таблиць, 8 додатків та 36 використаних літературних джерел.

Метою роботи є розробка інформаційної системи для організації прослуховування аудіоконтенту з підтримкою обробки мультитрекового аудіо (stems), збору статистики телеметрії та векторною системою рекомендацій.

У ході дослідження було проаналізовано сучасні підходи до побудови стримінгових платформ та виявлено недоліки існуючих рішень. Для реалізації системи рекомендацій застосовано алгоритми розрахунку центроїдів векторів на основі експоненційної ковзної середньої та просторовий пошук із використанням HNSW-індексів. Обробку та доставку медіаконтенту реалізовано за допомогою протоколу HTTP Live Streaming (HLS). Обробка і збір статистики забезпечується лямбда-архітектурою з використанням патерну CQRS.

Результатом роботи є вебплатформа «IndieStream», що забезпечує доставку медіаданих, отримання статистики телеметрії, персоналізованих рекомендацій, безпечне зберігання контенту в S3-сумісному сховищі та синхронізоване відтворення аудіодоріжок на стороні клієнта. Практична цінність роботи полягає у розробці програмного продукту, який задовольняє потреби незалежних артистів у розширеному представленні медіаконтенту, що не забезпечується традиційними централізованими платформами. Запропоновані архітектурні рішення можуть бути використані як основа для масштабування медіа-платформ.

**КЛЮЧОВІ СЛОВА:** АУДІОКОНТЕНТ, ВЕКТОРНА СХОЖІСТЬ, ІНФОРМАЦІЙНА СИСТЕМА, МОДУЛЬНИЙ МОНОЛІТ, СТРИМІНГ.

|            |      |                |        |      |                                                                                             |                                            |      |         |
|------------|------|----------------|--------|------|---------------------------------------------------------------------------------------------|--------------------------------------------|------|---------|
|            |      |                |        |      | ІПЗ.КР.Б-121-26-ПЗ                                                                          |                                            |      |         |
| Змн.       | Арк. | № докум.       | Підпис | Дата | Інформаційна система організації прослуховування аудіоконтенту.<br><br>Пояснювальна записка | Літ.                                       | Арк. | Аркушів |
| Розроб.    |      | Васянович О.А. |        |      |                                                                                             |                                            |      |         |
| Керівник   |      | Морозов А.В.   |        |      |                                                                                             |                                            | 3    | 119     |
| Рецензент. |      | Локтікова Т.М. |        |      |                                                                                             | Житомирська політехніка,<br>група ІПЗ-22-3 |      |         |
| Зав. каф.  |      | Вакалюк Т.А.   |        |      |                                                                                             |                                            |      |         |

## ABSTRACT

The bachelor's thesis explanatory note contains 119 pages, 38 illustrations, 14 tables, 8 appendices, and 36 literary sources.

The aim of the work is to develop an information system for organizing audio content streaming with support for multitrack audio (stems) processing, telemetry statistics collection, and a vector-based recommendation system.

During the research, modern approaches to building streaming platforms were analyzed, and the shortcomings of existing solutions were identified. To implement the recommendation system, vector centroid calculation algorithms based on the exponential moving average and spatial search using HNSW indexes were applied. Media content processing and delivery are implemented using the HTTP Live Streaming (HLS) protocol. Statistics collection and processing are ensured by the Lambda architecture using the CQRS pattern.

The result of the work is the "IndieStream" web platform, which provides media data delivery, telemetry statistics retrieval, personalized recommendations, secure content storage in an S3-compatible repository, and synchronized audio track playback on the client side. The practical value of the work lies in the development of a software product that meets the needs of independent artists in an advanced presentation of media content, which is not provided by traditional centralized platforms. The proposed architectural solutions can be used as a basis for scaling media platforms.

**KEYWORDS:** AUDIO CONTENT, INFORMATION SYSTEM, MODULAR MONOLITH, STREAMING, VECTOR SIMILARITY.

## ЗМІСТ

|                                                                                                                           |     |
|---------------------------------------------------------------------------------------------------------------------------|-----|
| РЕФЕРАТ.....                                                                                                              | 3   |
| ABSTRACT.....                                                                                                             | 4   |
| ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....                                                                                            | 6   |
| ВСТУП.....                                                                                                                | 7   |
| РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ІНФОРМАЦІЙНИХ<br>ТЕХНОЛОГІЙ ДЛЯ ОРГАНІЗАЦІЇ ПРОСЛУХОВУВАННЯ<br>АУДІОКОНТЕНТУ..... | 10  |
| 1.1 Постановка задачі.....                                                                                                | 10  |
| 1.2 Аналіз аналогів програмного продукту.....                                                                             | 12  |
| 1.3 Вибір архітектури веборієнтованої системи прослуховування<br>аудіоконтенту.....                                       | 16  |
| 1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного<br>забезпечення.....                            | 20  |
| Висновки до першого розділу.....                                                                                          | 23  |
| РОЗДІЛ 2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОРГАНІЗАЦІЇ<br>ПРОСЛУХОВУВАННЯ АУДІОКОНТЕНТУ.....                            | 24  |
| 2.1 Визначення варіантів використання.....                                                                                | 24  |
| 2.2 Розробка бази даних системи.....                                                                                      | 45  |
| 2.3 Проєктування та реалізація алгоритмів роботи системи.....                                                             | 57  |
| 2.4 Реалізація додатку інформаційної системи організації прослуховування<br>аудіоконтенту.....                            | 68  |
| Висновки до другого розділу.....                                                                                          | 71  |
| РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ІНФОРМАЦІЙНОЮ<br>СИСТЕМОЮ ОРГАНІЗАЦІЇ ПРОСЛУХОВУВАННЯ АУДІОКОНТЕНТУ...73          | 73  |
| 3.1 Порядок встановлення та налаштування параметрів системи.....                                                          | 73  |
| 3.2 Структура інтерфейсу інформаційної системи організації прослуховування<br>аудіоконтенту.....                          | 78  |
| 3.3. Тестування роботи програмного комплексу.....                                                                         | 91  |
| Висновки до третього розділу.....                                                                                         | 96  |
| ВИСНОВКИ.....                                                                                                             | 97  |
| СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....                                                                                           | 99  |
| ДОДАТКИ.....                                                                                                              | 103 |

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних

СУБД – система управління базами даних

ШІ – штучний інтелект

API (Application Programming Interface) – прикладний програмний інтерфейс

CI (Continuous Integration) – безперервна інтеграція

DTO (Data Transfer Object) – об'єкт передачі даних між шарами архітектури.

HLS (HTTP Live Streaming) – протокол потокової передачі медіаданих через HTTP

ORM (Object-Relational Mapping) – технологія програмування, яка зв'язує бази даних з об'єктно-орієнтованим кодом

S3 (Simple Storage Service) – протокол та інтерфейс об'єктного зберігання даних

SQL (Structured Query Language) – мова структурованих запитів до реляційних баз даних

UI (User Interface) – інтерфейс користувач

HNSW (Hierarchical Navigable Small World) – алгоритм ефективного наближеного векторного пошуку.

## ВСТУП

Сервіси цифрового розповсюдження аудіоконтенту орієнтовані на масового слухача та стандартні формати відтворення. Це обмежує можливості незалежних виконавців щодо гнучкого представлення власного медіаконтенту. Існує необхідність розробки інструментів для презентації контенту у багатокомпонентних форматах, зокрема через використання технології мультитрекового аудіо (stems) та потокової передачі (HLS). Вимоги до персоналізації потребують впровадження механізмів рекомендацій, побудованих на просторовому аналізі та обробці телеметрії взаємодій користувачів у реальному часі. Розробка платформи «IndieStream» вирішує завдання побудови системи, де процеси медіадоставки, керування stem-компонентами, збирання та обробка телеметрії, формування векторних рекомендацій реалізовані на базі інкапсульованої модульної архітектури, що дозволяє уникнути недоліків традиційних централізованих платформ.

Метою випускної кваліфікаційної роботи бакалавра є розробка інформаційної системи для організації прослуховування аудіоконтенту з підтримкою обробки мультитрекового аудіо (stems), збору статистики телеметрії та векторною системою рекомендацій.

Для досягнення мети визначено наступні завдання:

- проведення аналізу предметної галузі, виявлення недоліків існуючих стрімінгових платформ та формування вимог до цільової системи;
- проєктування архітектурної моделі бекенду;
- реалізація підсистеми збереження медіаданих та їх доставки через HTTP Live Streaming з підтримкою часткового завантаження;
- розробка модуля інтелектуальних рекомендацій та системи збору телеметрії;
- проєктування клієнтської частини із впровадженням динамічного мікшування звуку;
- реалізація програмного комплексу та автоматизоване тестування системи.

Об'єктом дослідження є автоматизація процесів доставки, обробки та споживання багатокомпонентного цифрового аудіоконтенту. Предметом дослідження є використання архітектурних патернів, просторових баз даних та вебтехнологій для розробки системи стрімінгу мультитрекового аудіоконтенту.

Для вирішення поставлених завдань у роботі застосовано: методи системного аналізу для формування вимог; концепції предметно-орієнтованого проєктування (DDD) для побудови архітектури бекенду та UML-моделювання - при побудові моделі архітектури застосунку; математичні методи - алгоритми просторового пошуку для рекомендаційної системи; концепції лямбда-архітектури для обробки часових рядів телеметрії; методи автоматизованого статичного аналізу коду для контролю цілісності архітектури. Інформаційною базою дослідження є технічна документація використаних технологій, офіційні специфікації стандартів та матеріали аналізу програмних аналогів.

Результатом роботи є готова до розгортання вебплатформа «IndieStream». Програмний продукт дозволяє незалежним виконавцям презентувати аудіоконтент у мультитрековому форматі з можливістю динамічного керування аудіодоріжками. Впроваджені архітектурні рішення (мікробатчинг аналітики, просторове індексування) вирішують типові проблеми високонавантажених систем, усуваючи вузькі місця на рівні бази даних. Розроблена система може використовуватися незалежними артистами для організації дистрибуції контенту, а реалізовані модулі телеметрії і рекомендацій придатні для ізолюваного інтегрування у сторонні медіа-сервіси.

За темою випускної кваліфікаційної роботи бакалавра було опубліковано тези: Васянович О. А., Морозов А. В. АВТОМАТИЗОВАНА ГЕНЕРАЦІЯ МОСК-ДАНИХ У РОЗРОБЦІ СТРІМІНГОВИХ СЕРВІСІВ. Тези доповідей XI Міжнародної науково-технічної конференції «Інформаційно-комп'ютерні технології», 02-03 квітня 2026 року. Житомир: «Житомирська політехніка», 2026. С.87-88.

Васянович О. А., Морозов А. В. АРХІТЕКТУРА РЕКОМЕНДАЦІЙНИХ СИСТЕМ В УМОВАХ ВІДСУТНОСТІ СТАТИСТИКИ ВЗАЄМОДІЙ. Тези

доповідей XI Міжнародної науково-технічної конференції «Інформаційно-комп'ютерні технології», 02-03 квітня 2026 року. Житомир: «Житомирська політехніка», 2026. .С.510-511.

Васянович О. А., Концидайло А. М. АВТОМАТИЗОВАНИЙ КОНТРОЛЬ ЯКОСТІ АРХІТЕКТУРНИХ ОБМЕЖЕНЬ У МОДУЛЬНИХ МОНОЛІТНИХ СИСТЕМАХ. Матеріали XXVI Всеукраїнської науково-технічної конференції молодих вчених, аспірантів та студентів «Стан, досягнення та перспективи інформаційних систем і технологій», 16-17 квітня 2026 року. Одеса: ОНТУ, 2026. .С.158-159.

Васянович О. А., Нерода С. І. АВТОМАТИЗОВАНИЙ КОНТРОЛЬ ЯКОСТІ АРХІТЕКТУРНИХ ОБМЕЖЕНЬ У МОДУЛЬНИХ МОНОЛІТНИХ СИСТЕМАХ. Матеріали XXVI Всеукраїнської науково-технічної конференції молодих вчених, аспірантів та студентів «Стан, досягнення та перспективи інформаційних систем і технологій», 16-17 квітня 2026 року. Одеса: ОНТУ, 2026. .С.237-239.

Структура та обсяг проєкту (роботи). Випускна робота складається з вступу, 3-х розділів, висновків, списку використаних джерел із 36 найменувань (3 стор.), 8 додатків (16 стор.), 14 таблиць та 38 ілюстрацій по тексту. Повний обсяг роботи становить 119 сторінок, з них 92 сторінки основного тексту.

# РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ ОРГАНІЗАЦІЇ ПРОСЛУХОВУВАННЯ АУДІОКОНТЕНТУ

## 1.1 Постановка задачі

Основна проблема традиційної цифрової дистрибуції полягає в обмеженості стандартних форматів відтворення, де кінцевий користувач взаємодіє виключно з фінальним зведеним аудіофайлом. Для задоволення потреб незалежних виконавців система повинна підтримувати технологію мультитрекового аудіо (stems) - роздільного зберігання та синхронного відтворення окремих звукових доріжок, що складають цілісну композицію. Реалізація такого функціоналу вимагає автоматизації процесів асинхронної обробки медіафайлів, їх конвертації у потокові формати та забезпечення мікшування сигналів на стороні клієнта.

Інформаційна система «IndieStream» розробляється як платформа, що фокусується на підтримці незалежних виконавців, надаючи їм інструменти для презентації музики у розширеному форматі. Об'єктом автоматизації є процеси збереження, доставки, агрегації метаданих та персоналізованої рекомендації аудіоконтенту в мережевому середовищі.

Для досягнення поставленої мети цільова система проєктується за принципами предметно-орієнтованого проєктування і повинна включати ізольовані програмні модулі.

Модуль управління медіаконтентом забезпечує завантаження мультитрекових записів, валідацію метаданих, зберігання файлів у S3-сумісному сховищі, модерацію треків. Модуль керує життєвим циклом треку, де невід'ємним етапом є асинхронний конвеєр обробки. Конвеєр включає автоматичну модерацію контенту та генерацію семантичних метаданих із застосуванням штучного інтелекту, а також передбачає підсистему ручного контролю адміністратором (Human-in-the-Loop) для вирішення конфліктних станів.

Модуль ідентифікації та безпеки відповідає за авторизацію користувачів, розмежування прав доступу та захист маршрутів. Забезпечує негайне відкликання сесій та керує профілями користувачів.

Модуль агрегації та управління бібліотекою забезпечує створення системних та користувацьких плейлистів, керування приватністю та соціальною взаємодією. Модуль телеметрії відповідає за збір та обробку подій взаємодії з плеєром у реальному часі, для подальшого збереження і відображення аналітики прослуховувань авторам контенту.

Модуль рекомендацій забезпечує персоналізовану видачу контенту, обробляє події взаємодії з треками, розраховує вектор смаку користувача і виконує просторовий пошук сумісних треків у базі даних.

Підсистема клієнтського відтворення (Frontend) забезпечує взаємодію користувача з платформою. Повинна реалізувати маршрутизацію аудіосигналів для безперервного та синхронізованого мікшування аудіо-доріжок, а також керувати чергою відтворення і станом інтерфейсу.

Розробка системи передбачає послідовне виконання етапів проєктування та реалізації. На початковому етапі формується архітектура реляційної бази даних із застосуванням скриптів суворої міграції та створюється структура серверної частини з жорстким дотриманням меж інкапсуляції доменів. На етапі реалізації бекенду розробляються API-шлюзи, впроваджуються механізми взаємодії з хмарним сховищем та створюється конвеєр обробки аудіо. Паралельно налаштовується інфраструктура збору метрик і розгортається векторний рушій рекомендацій. Розробка клієнтського інтерфейсу фокусується на створенні інтерактивного медіаплеєра, панелей списків відтворення, візуалізації статистики для артистів та механізмів оптимістичного оновлення інтерфейсу (Optimistic UI) для усунення мережових затримок. Завершальним етапом є автоматизоване тестування архітектурних правил системи, інтеграційне тестування з використанням ізольованих контейнерних середовищ та навантажувальне тестування.

Результатом виконання зазначених вище етапів є створення цілісної платформи, яка надає незалежним виконавцям можливості розповсюдження багатокomпонентного аудіоконтенту, а користувачам - сучасний інструментарій для взаємодії з музикою, що виходить за межі традиційних стандартів стримінгу.

## 1.2 Аналіз аналогів програмного продукту

Розробка інформаційної системи «IndieStream» потребує детального дослідження ринку існуючих стримінгових сервісів та платформ дистрибуції медіаконтенту. Проведений аналіз дозволяє виявити переваги та недоліки сучасних рішень, а також визначити технологічні прогалини, які може заповнити розроблюваний програмний продукт. У межах дослідження було розглянуто системи різних класів: від масових сервісів до вузькоспеціалізованих апаратних та програмних рішень.

Найбільшими представниками ринку є платформи «Spotify» [1] (рис. 1.1) та «Apple Music» [2]. Дані сервіси використовують модель масового стримінгу (Mass Market Streaming) та володіють найбільшими бібліотеками ліцензованого контенту. Основною технічною перевагою цих систем є висока доступність, стійкість до високих навантажень та наявність складних систем рекомендацій, що базуються на алгоритмах колаборативної фільтрації та машинного навчання. Проте в контексті завдань розроблюваної системи вони мають суттєві обмеження. Платформи орієнтовані виключно на доставку зведених стереотреків (Master record), що повністю виключає можливість інтерактивної взаємодії слухача зі структурою композиції. Крім того, замкнена екосистема ускладнює розповсюдження контенту для незалежних виконавців, які не мають підтримки великих лейблів. Рекомендаційна система фокусує увагу слухачів на вже популярному контенті, чим ускладнює просування для незалежних виконавців.

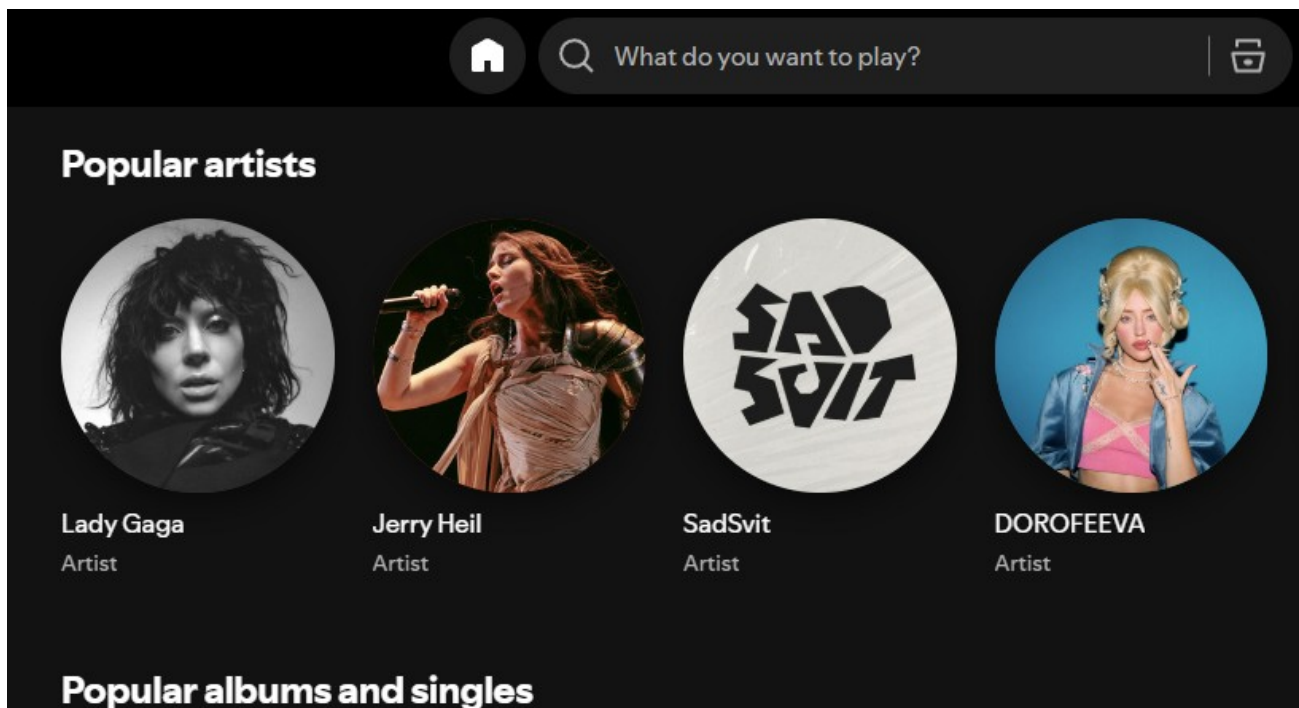


Рис. 1.1 - Інтерфейс платформи «Spotify»

Платформа «SoundCloud» [3] вважається найбільш орієнтованою на незалежних авторів (рис. 1.2). Сервіс фокусується на соціальній взаємодії через систему коментування конкретних фрагментів аудіотреку на часовій шкалі та швидкому рендерингу хвильових форм аудіосигналу.

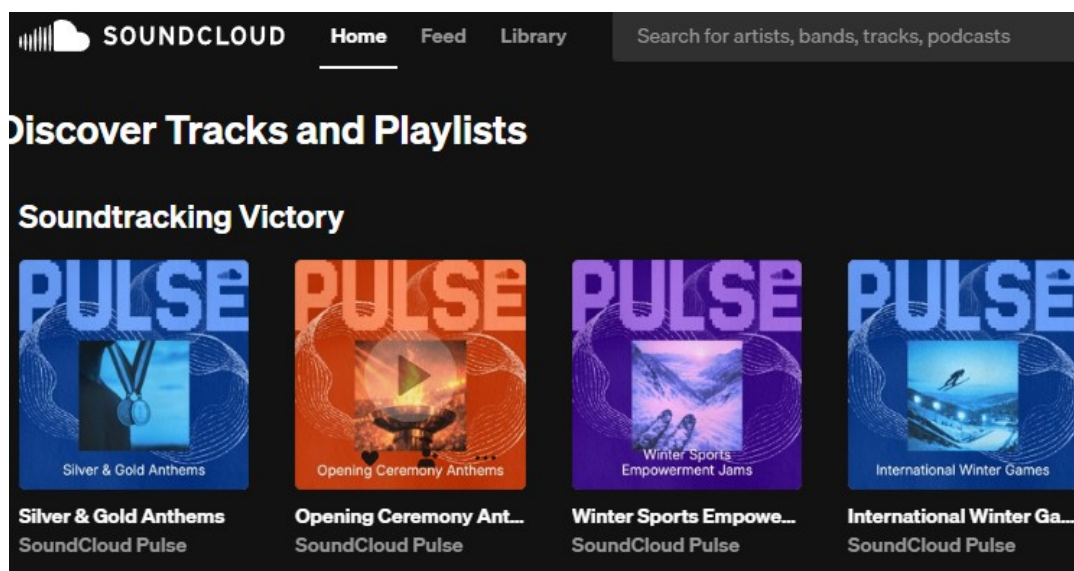


Рис. 1.2 - Платформа «SoundCloud»

Перевагою «SoundCloud» є низький поріг входу для публікації контенту. До недоліків належить відсутність підтримки багатокомпонентного аудіо (стемів). Система не надає користувачеві інструментів для динамічного керування окремими доріжками. Вбудована система виявлення нового контенту

значною мірою залежить від соціального графа користувача та базового пошуку за текстовими тегамі. Це знижує релевантність алгоритмічної видачі для нових артистів, треки яких ще не накопичили достатньої кількості прослуховувань для спрацьовування колаборативної фільтрації.

Сервіс «Bandcamp» [4] представляє іншу модель - статичного цифрового каталогу (рис. 1.3), побудованого за принципом «Direct-to-Fan» (прямий зв'язок між автором та слухачем). Платформа надає пріоритет продажам цифрових та фізичних копій альбомів.

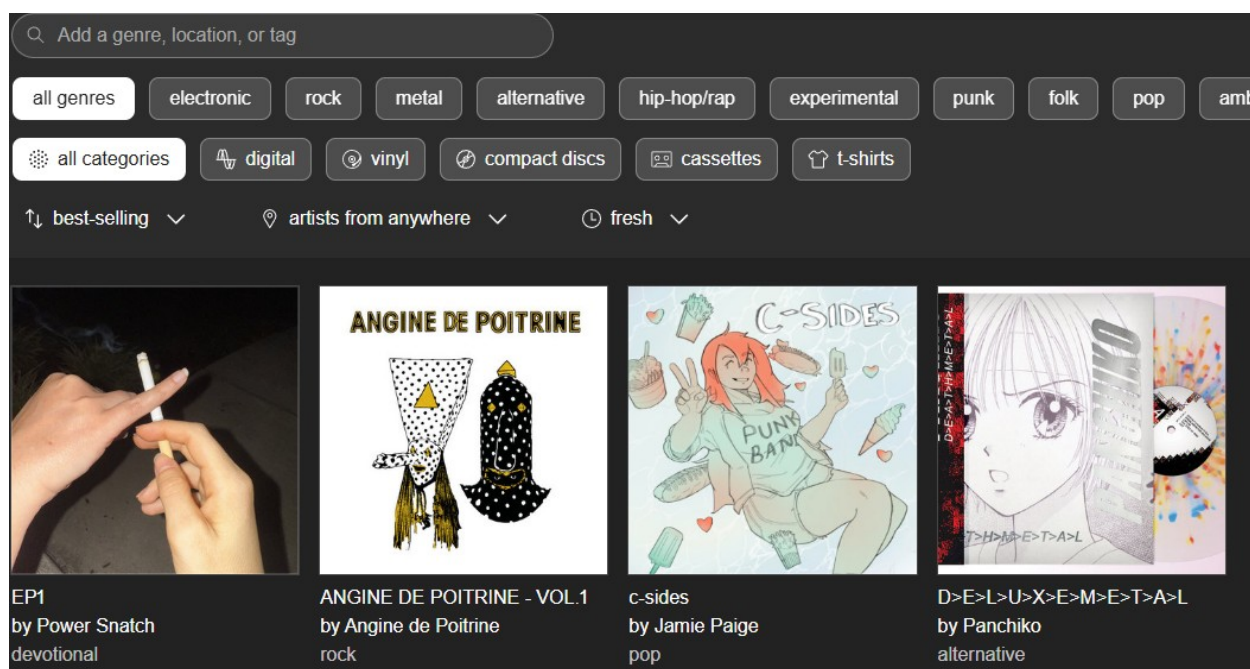


Рис. 1.3 - Вебінтерфейс сервісу «Bandcamp»

Хоча «Bandcamp» дозволяє завантажувати аудіо у форматах без втрат якості (FLAC/WAV), він залишається насамперед вітриною цифрового магазину, а не інтерактивним плеєром. У системі фактично відсутні алгоритми персоналізованого алгоритмічного підбору контенту (Discovery Feed), а вебплеєр виконує виключно базові функції послідовного відтворення попередньо завантажених стереофайлів.

Окремим класом рішень є професійні інструменти диджеїнгу та продакшену, такі як формат «Native Instruments Stems» або спеціалізований апаратний пристрій «Stem Player» [5] (рис. 1.4).



Рис. 1.4 - Апаратний пристрій «Stem Player»

Головною перевагою цих систем є можливість ізольованого керування окремими складовими композиції (вокал, ударні, бас) у реальному часі. Проте дані рішення вимагають попередньої підготовки аудіофайлів у закритих форматах та є апаратно залежними. Вони потребують використання професійного програмного забезпечення або купівлі окремих пристроїв, що робить їх недоступними для масового слухача. Крім того, вони є виключно інструментами відтворення і позбавлені мережевої інфраструктури для дистрибуції контенту, обчислення аналітики чи алгоритмічного просування.

На основі проведеного аналізу сформовано порівняльну таблицю функціональних характеристик розглянутих рішень (табл. 1.1).

Таблиця 1.1

Порівняльна характеристика систем прослуховування аудіоконтенту

| Характеристика                                              | Spotify | SoundCloud | Bandcamp | Stem Player | IndieStream (планується) |
|-------------------------------------------------------------|---------|------------|----------|-------------|--------------------------|
| Інтерактивне відтворення багатокомпонентного аудіо (стемів) | -       | -          | -        | +           | +                        |
| Фокус на незалежних авторів                                 | -       | +          | +        | -           | +                        |
| Доступність через вебінтерфейс                              | +       | +          | +        | -           | +                        |
| Система рекомендацій                                        | +       | +          | -        | -           | +                        |

|  |  |  |  |  |
|--|--|--|--|--|
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

На основі виявлених недоліків існуючих систем, розробка платформи «IndieStream» спрямована на впровадження унікальної можливості динамічного керування аудіодоріжками безпосередньо у браузері кінцевого користувача, що замінює необхідність використання професійного апаратного забезпечення. Для усунення проблеми «холодного старту», притаманної масовим платформам, проєктується векторна система рекомендацій, яка обчислює геометричний центроїд інтересів користувача та виконує просторовий пошук схожих треків, гарантуючи високу релевантність для маловідомих виконавців. Клієнтська частина системи передбачає інтуїтивно зрозумілий інтерфейс для потокового мікшування аудіо, що не потребує спеціальних технічних знань. Таким чином, створення системи «IndieStream» є технологічно обґрунтованим рішенням, яке заповнює вільну нішу інтерактивного вебстримінгу та надає незалежному сегменту музичної індустрії повноцінний інструмент для дистрибуції та просування складних медіаформатів.

### **1.3 Вибір архітектури веборієнтованої системи прослуховування аудіоконтенту**

Вибір архітектури інформаційної системи «IndieStream» є етапом, що визначає стабільність роботи під навантаженням, можливості масштабування та ізоляцію відмов під час обробки багатокомпонентного аудіоконтенту. Враховуючи необхідність інтеграції інтелектуальних підсистем, високонавантаженої телеметрії та потокового передавання медіа, було проведено порівняльний аналіз трьох основних архітектурних патернів: традиційного моноліту [6], мікросервісної архітектури та модульного моноліту [7]. Монолітна архітектура характеризується високою швидкістю розробки на початкових етапах та простотою інфраструктурного розгортання. Проте зі збільшенням функціоналу цей підхід призводить до формування надмірно зв'язаної бази коду. Відсутність строгих меж між компонентами генерує приховані залежності, ускладнює автоматизоване тестування та виключає можливість незалежного масштабування обчислювально важких процесів (наприклад, обробки медіа файлів чи розрахунку векторів).

Мікросервісна архітектура забезпечує максимальну ізоляцію доменів та незалежне горизонтальне масштабування компонентів. Однак для цільової системи даний підхід генерує надмірні накладні витрати на управління мережевою взаємодією та оркестрацію інфраструктури. Розподілені транзакції між мікросервісами ускладнюють забезпечення узгодженості даних (Data Consistency).

Для реалізації платформи «IndieStream» обрано архітектурний патерн «модульний моноліт» [7] (Modular Monolith) на базі принципів предметно-орієнтованого проєктування (Domain-Driven Design). Даний підхід поєднує переваги простоти розгортання моноліту із чіткою логічною ізоляцією, властивою мікросервісам. Система проєктується як сукупність функціональних модулів [8] (Bounded Contexts), кожен з яких відповідає за конкретну доменну область: управління медіатекою, стримінг, рекомендації та користувацькі профілі.

Ключовим правилом обраної архітектури є строге дотримання меж модулів. Компоненти (внутрішні сервіси, репозиторії) одного модуля приховуються за допомогою модифікаторів доступу на рівні пакетів (package-private) і є недоступними для зовнішнього виклику. Взаємодія між модулями реалізується виключно через явно визначені публічні фасади (Public API Facades) та асинхронну взаємодію через механізм подій домену (Domain Events). Це дозволяє у майбутньому, при масштабуванні легко винести будь-який модуль в окремий мікросервіс без глобальної переробки коду. Узагальнена структура архітектурної взаємодії в межах системи представлена на рис. 1.5.

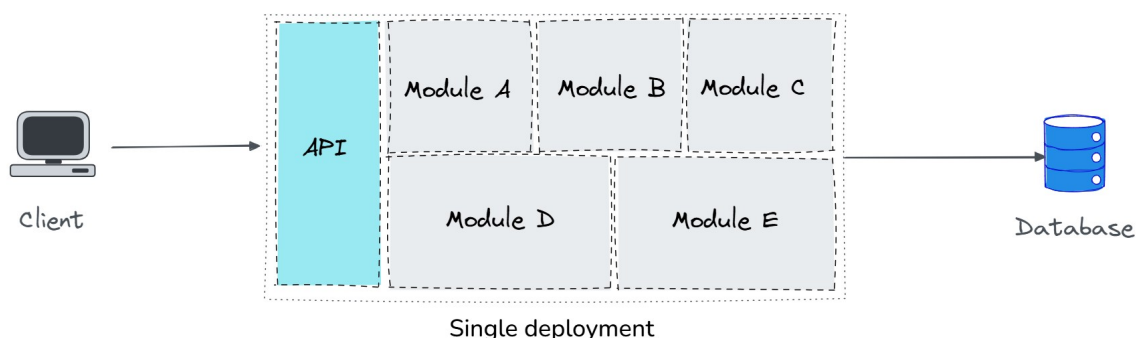


Рис. 1.5 - Узагальнена діаграма архітектурної взаємодії системи

Контроль за дотриманням цих архітектурних обмежень виконується автоматизовано на етапі безперервної інтеграції (CI) за допомогою статичного аналізатора коду (ArchUnit [9]).

Логічна структура серверної частини базується на принципах багат шарової архітектури [10] (Layered Architecture) з суворим дотриманням потоку даних: Controller => Service => Repository. У межах модульного моноліту контролери взаємодіють лише із сервісами, а доступ до бази даних напрямую заборонено.

Доменний шар залишається незалежним від зовнішніх форматів представлення даних. Трансформація внутрішніх сутностей (Entities) у незмінні структури даних (DTO) виконується на рівні сервісів перед поверненням результату до контролера. Такий підхід гарантує цілісність бізнес-логіки та запобігає ненавмисним змінам стану об'єктів у шарі представлення.

Нижче наведено діаграму внутрішньої архітектури модуля (рис. 1.6.), що ілюструє дотримання принципу ізоляції та правила трансформації даних.

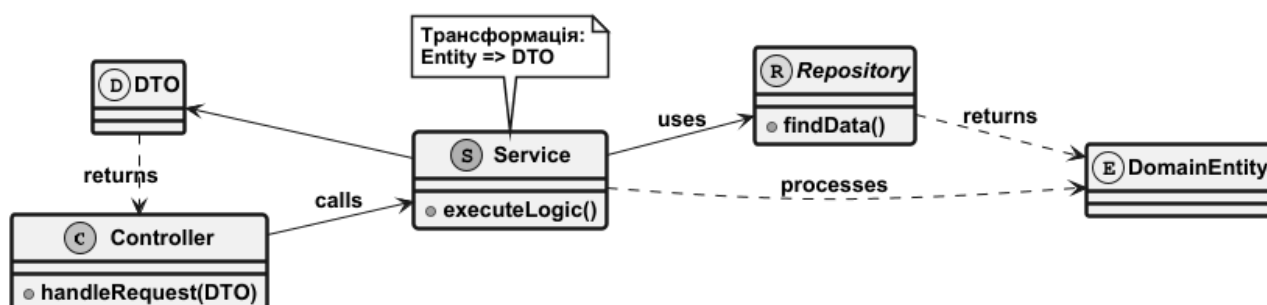


Рис. 1.6 - Діаграма внутрішньої архітектури модуля

Для оптимізації клієнт-серверної взаємодії впроваджено архітектурний патерн Backend-For-Frontend (BFF) [11]. Агрегація даних (наприклад, об'єднання метаданих треку, інформації про автора та статусу лайку) виконується на стороні сервера за допомогою паралельних кросмодульних проєкцій. Це усуває проблему надлишкових запитів (N+1) та каскадних мережових запитів (Network Waterfalls) на стороні клієнта.

Замість автоматичної генерації схем (ORM DDL) , у системі впроваджено підхід «schema-first with migrations» із суворим версіонуванням міграцій бази даних . Це забезпечує повний контроль над структурою таблиць, індексами та

типами даних, що є критичним при роботі з векторними даними для системи рекомендацій.

Для обробки високонавантаженої статистики прослуховувань спроектовано підсистему телеметрії на базі лямбда-архітектури [12] (Lambda Architecture) та принципу поділу відповідальності на команди і запити (CQRS). Швидкий шлях (Fast Path) використовує потоки даних у пам'яті для відмовостійкої пакетної обробки подій, після чого фонові процеси виконують агрегацію у реляційні таблиці (Slow Path).

Для клієнтської частини обрано архітектуру односторінкового застосунку (SPA), побудовану за методологією Feature-Sliced Design (FSD) [13]. Даний підхід забезпечує жорстку типізацію та логічне групування коду за функціональними особливостями (features), усуваючи проблему неструктурованих директорій.

Управління станом клієнтського застосунку розділено на два незалежні рівні. Рівень глобального клієнтського стану (Client State Engine) відповідає за детерміновану роботу аудіоплеєра, управління чергою відтворення, логіку мікшування стемів та стан інтерфейсу. Реалізується за допомогою патерну State Machine. Рівень серверного стану (Server State) забезпечує асинхронне отримання даних, кешування, фонову синхронізацію та управління мутаціями. Використовує механізм оптимістичного оновлення інтерфейсу (Optimistic UI) для миттєвого відображення змін без очікування відповіді від сервера (наприклад, при зміні стану публікації чи додаванні до плейлиста).

Такий розподіл дозволяє уникнути надмірної складності у коді фронтенду та забезпечує високу швидкість відгуку інтерфейсу, що є необхідним для комфортного прослуховування музики.

Безпека базується на архітектурі безстанової авторизації (Stateless JWT) з інжекцією ідентифікаторів безпосередньо у контекст безпеки (SecurityContext Principal) для запобігання підміні суб'єкта (IDOR). Маршрутизація помилок уніфікована згідно зі стандартом RFC 7807, що гарантує передбачуваний формат відповідей для клієнтського застосунку.

Підсумовуючи, модульний моноліт у поєднанні з FSD на клієнті та лямбда-архітектурою телеметрії є оптимальним і технологічно обґрунтованим рішенням для системи «IndieStream». Впроваджена структура забезпечує виконання специфічних вимог до обробки медіа та векторних обчислень, усуває недоліки монолітної зв'язності та створює готовий фундамент для прозорої екстракції окремих доменів у мікросервіси в умовах масштабування навантаження.

#### **1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення**

Реалізація інформаційної системи «IndieStream» базується на клієнт-серверній архітектурі з використанням сучасного стеку технологій, що забезпечує високу продуктивність при обробці поточкових медіаданих та масштабованість підсистем інтелектуального аналізу.

Для розробки серверної частини обрано мову програмування Java 21 [14] та фреймворк Spring Boot 3.2 [15]. Вибір Java 21 зумовлений наявністю довгострокової підтримки (LTS) та впровадженням віртуальних потоків, що суттєво оптимізує роботу з блокуючими операціями введення-виведення, які є характерними для стримінгових сервісів. Фреймворк Spring Boot забезпечує інструментарій для суворої ізоляції модулів через бібліотеку Spring Modulith та інтеграцію з векторними базами даних за допомогою фреймворку Spring AI [16], що спрощує роботу з моделями ШІ без залучення складних сторонніх бібліотек.

Як основне інформаційне забезпечення системи використовується СУБД PostgreSQL 16 [17]. Важливою характеристикою обраної версії є повна сумісність із розширенням «pgvector» [18], яке дозволяє зберігати вектори вподобань користувачів та виконувати пошук схожого контенту безпосередньо засобами бази даних за допомогою нативних SQL-запитів. Для управління міграціями схеми даних обрано інструмент Flyway, що гарантує цілісність структури таблиць у різних середовищах розгортання.

Для збереження великих бінарних об'єктів (аудіостемів та графічних ресурсів) обрано S3-сумісне сховище MinIO [19]. Це дозволяє абстрагуватися від файлової системи сервера та забезпечує горизонтальне масштабування медіатеки. Redis використовується для тимчасового збереження стану, що критично для високонавантажених процесів. Його основні задачі включають підтримку безстанової авторизації шляхом збереження «чорного списку» відкликаних JWT-токенів та функціонування як відмовостійкого буфера для системи телеметрії (Redis [20] Streams). Це дозволяє акумулювати потік подій прослуховування та взаємодії перед їх мікробатчингом у PostgreSQL, знижуючи навантаження на основну базу даних.

Клієнтська частина системи реалізується за допомогою бібліотеки React 19 [21] та мови програмування TypeScript [22]. Використання суворої типізації є обов'язковою умовою для запобігання помилкам при роботі зі складними структурами даних. Інструментарій фронтенд-розробки включає:

- Web Audio API - для маршрутизації та синхронізованого мікшування мультитрекових потоків (stems) на стороні клієнта у реальному часі без надмірного споживання оперативної пам'яті.

- Vite - засіб збірки проєкту, що забезпечує високу швидкість розробки та оптимізацію модулів.

- Zustand [23] - для детермінованого управління глобальним клієнтським станом (логіка аудіо програвача, керування чергою відтворення );

- TanStack Query [24] - забезпечує синхронізацію серверного стану, кешування та механізми оптимістичного оновлення інтерфейсу (Optimistic UI)

- Tailwind CSS та Radix UI - для побудови адаптивного інтерфейсу та імплементації доступних (accessible) компонентів керування.

Інфраструктура системи контейнеризується за допомогою Docker [25] та Docker Compose. Таке рішення дозволяє інкапсулювати компоненти платформи (серверний додаток, СУБД, кеш-сервер та об'єктне сховище) у ізольовані контейнери, що гарантує ідентичність їхньої роботи незалежно від конфігурації хост-системи. Процеси збірки, автоматизованого тестування та перевірки якості

коду інтегровані у конвеєр безперервної розробки (CI/CD), реалізований засобами GitHub Actions [26], що мінімізує ризик виникнення дефектів при розгортанні нових версій модулів.

Для розгортання системи у робочому середовищі (Production) обрано розподілену хмарну інфраструктуру. Серверна частина, модульний моноліт, базу даних та об'єктне сховище, розгортається на віртуальному виділеному сервері (VPS) від провайдера Hetzner Cloud [27] під управлінням операційної системи Ubuntu 24.04. Як зворотний проксі-сервер (Reverse Proxy) використовується Nginx, який відповідає за термінацію SSL-з'єднань та маршрутизацію HTTP-запитів до відповідних внутрішніх служб. Клієнтський вебзастосунок розгортається на хмарній платформі Vercel [28], яка забезпечує автоматичну глобальну доставку статичних файлів та інтеграцію з конвеєром CI/CD.

Апаратні вимоги для функціонування системи розділяються на вимоги до середовища розробки та цільового сервера розгортання. Для локальної робочої станції розробника (Development середовище), з урахуванням одночасного запуску інтегрованого середовища розробки (IDE), процесів збірки фронтенду та повної локальної інфраструктури у Docker-контейнерах, визначено наступні параметри:

- процесор: 4-ядерний із частотою від 2.5 ГГц;
- оперативна пам'ять: не менше 16 ГБ;
- дисковий простір: від 256 ГБ на SSD-носії.

Завдяки оптимізації контейнеризації та відсутності графічного інтерфейсу, вимоги для ізольованого серверного середовища розгортання (Production) є більш стриманими. Для забезпечення стабільного стрімінгу, обробки телеметрії та обчислення векторних рекомендацій цільовий сервер повинен відповідати наступним мінімальним характеристикам:

- процесор: 4 віртуальні ядра (vCPU);
- оперативна пам'ять: 8 ГБ (мінімально необхідний обсяг для паралельної роботи JVM, PostgreSQL з pgvector, MinIO та Redis);

- дисковий простір: 80 ГБ NVMe SSD.

Вимоги до клієнтського обладнання передбачають наявність сучасного веббраузера (Chrome 110+, Firefox 105+) з підтримкою Web Audio API для коректної роботи інтерактивного мікшування аудіодоріжок.

### Висновки до першого розділу

У результаті проведеного дослідження було детально проаналізовано предметну галузь організації розповсюдження та прослуховування аудіоконтенту. Виявлено, що існуючі рішення оптимізовані виключно під доставку зведених стереотреків, що створює суттєвий функціональний розрив у сегменті інтерактивної взаємодії. Це підтверджує актуальність розробки інформаційної системи «IndieStream», яка надає незалежним виконавцям спеціалізовані інструменти для роботи з багатокомпонентним аудіо (stems).

На основі порівняльного аналізу аналогів сформовано перелік функціональних вимог, ключовими з яких є: розробка конвеєра обробки та мікшування стемів через браузер, впровадження системи збору телеметрії та реалізація просторових векторних рекомендацій для усунення проблеми «холодного старту» нових виконавців. Для виконання цих завдань обґрунтовано вибір архітектури «модульний моноліт» (Domain-Driven Design). Даний підхід гарантує сувору інкапсуляцію доменів (Медіа, Ідентифікація, Телеметрія, Рекомендації) та усуває надмірну складність розподілених транзакцій на етапі прототипування.

Визначений технологічний стек, що включає Java 21, Spring Boot, React 19 (за методологією Feature-Sliced Design), СУБД PostgreSQL з розширенням pgvector, кеш Redis та S3-сховище MinIO, повністю відповідає сучасним вимогам до продуктивності потокових медіа-платформ. Використання хмарної інфраструктури на базі Hetzner Cloud та Vercel у поєднанні з автоматизованим CI/CD конвеєром GitHub Actions і суворим контролем архітектурних меж (ArchUnit) гарантує стабільність розробленої системи, стійкість до мережевих навантажень та її готовність до повноцінної програмної реалізації на наступних етапах дослідження.

## РОЗДІЛ 2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОРГАНІЗАЦІЇ ПРОСЛУХОВУВАННЯ АУДІОКОНТЕНТУ

### 2.1 Визначення варіантів використання

Проектування системи «IndieStream» ґрунтується на детальній декомпозиції доменних областей, функціональних обов'язків, якісних характеристик та архітектурних обмежень. Перехід від концептуального аналізу до проектування вимагає чіткої фіксації можливостей, які система надає різним категоріям користувачів, а також визначення технічних обмежень, що гарантують стабільність роботи в умовах інтенсивного стримінгу аудіо та обробки векторних даних.

#### Функціональні вимоги

Управління багатокомпонентним медіаконтентом передбачає створення середовища для публікації творів, де базовим агрегатом є трек, що містить масив синхронізованих аудіодоріжок (стемів), графічну обкладинку та семантичні метадані. Система повинна забезпечувати асинхронний конвеєр обробки вхідних медіаданих. Цей процес включає автоматичну екстракцію аудіо у формати HTTP Live Streaming (HLS) для створення маніфестів потокової передачі та валідацію цілісності файлів. Життєвий цикл контенту керується через жорстку матрицю переходів машини станів (Finite State Machine). Завантажений контент проходить обов'язкову ШІ-модерацію (мультимодальний аналіз) для генерації векторних тегів та виявлення порушень, після чого переходить у стан готовності до публікації, чекає перевірки від адміністратора або блокується.

Інтерактивне відтворення та стримінг - основний функціональний вузол, відповідальний за доставку синхронізованого аудіосигналу слухачу в реальному часі. Система має реалізовувати механізми точного позиціонування всередині треку та забезпечувати роботу віртуального мікшера, який дозволяє користувачу незалежно змінювати гучність або вимикати окремі компоненти композиції (вокал, інструменти) без втрати синхронізації між потоками. Важливою вимогою є підтримка безперервності відтворення при навігації

|  |  |  |  |  |                    |      |
|--|--|--|--|--|--------------------|------|
|  |  |  |  |  | ІПЗ.КР.Б-121-26-ПЗ | Арк. |
|  |  |  |  |  |                    |      |
|  |  |  |  |  |                    |      |
|  |  |  |  |  |                    | 24   |

користувача по сторінках платформи, що забезпечується збереженням стану плеєра в межах єдиної сесії застосунку.

Персоналізація та аналітика базуються на постійному зборі даних про взаємодію користувача з медіаконтентом: факти повного прослуховування, додавання до плейлистів, уподобання та пропуски треків. Для розв'язання проблеми «холодного старту» нових користувачів передбачено процедуру калібрування, яка примусово ініціалізує початковий вектор смаку. Програмний комплекс повинен автоматично обробляти телеметричні дані шляхом асинхронної мікропакетної обробки, розраховуючи експоненційну ковзну середню (ЕМА) для формування точних рекомендацій. Автори контенту отримують доступ до агрегованих аналітичних звітів (кількість унікальних слухачів, графіки прослуховувань, географічний розподіл), згенерованих шляхом маршрутизації запитів до часових рядів (Time-Series) бази даних.

Адміністрування та безпека охоплюють розмежування прав доступу, управління користувачами та модерацію контенту. Підсистема безпеки забезпечує миттєву інвалідацію скомпрометованих або заблокованих сесій користувачів через механізм чорних списків токенів (Token Blacklist). Адміністративний модуль містить інструменти для роботи з глобальним реєстром контенту та розв'язання конфліктів III-модерації за принципом Human-in-the-Loop. Обробка зовнішніх скарг щодо порушення авторських прав або неприйнятного контенту ініціюється користувачами через спеціалізовану електронну адресу. Відповідні звернення опрацьовуються адміністратором вручну через інтерфейс платформи шляхом ініціації глобальних подій домену: блокування облікового запису або примусового переведення об'єкта у статус архіву/видалення з формуванням незмінного аудиторського сліду.

Окрім модерації, до зони відповідальності адміністратора належить інфраструктурне обслуговування платформи. Реалізовано функціонал для примусового перерахунку історичних показників телеметрії та ініціації повної реіндексації векторних представлень треків (embeddings) у разі оновлення моделі або пошкодження даних. Адміністратор має доступ до інструментів

перезапуску обрахунку телеметрії за певні проміжки часу та реіндексації векторів усіх схвалених треків.

#### Нефункціональні вимоги

Якість функціонування «IndieStream» визначається параметрами надійності, безпеки та масштабованості. Показник доступності системи має становити не менше 99.9% часу, система повинна бути стійкою до короткочасних втрат з'єднання під час стримінгу, завдяки механізмам буферизації даних на стороні клієнта. Система має коректно обробляти виняткові ситуації на серверному рівні, повертаючи стандартизовані повідомлення про помилки, що не порушують роботу інтерфейсу. Інфраструктура телеметрії вимагає використання відмовостійкого буфера у пам'яті (Redis Streams) для запобігання втраті даних під час пікових навантажень на основну реляційну базу.

Безпека даних та конфіденційність реалізуються за допомогою безстанової авторизації (Stateless JWT). Ідентифікація суб'єкта виконується виключно через вилучення ідентифікаторів з криптографічно підписаного токена, що унеможлиблює атаки типу IDOR (Insecure Direct Object Reference). Забороняється зберігання паролів у відкритому вигляді; використовується алгоритм гешування BCrypt. Причому ідентифікатори сесії повинні зберігатися у захищених файлах cookie, недоступних для шкідливих скриптів. Взаємодія між сервером та клієнтом у мережі захищається налаштуваннями міждоменного доступу (CORS) та обмеженнями на завантаження виконуваних файлів. Механізми бази даних автоматично застосовують каскадне приховування контенту (плейлистів, треків) при блокуванні облікового запису автора.

Масштабованість закладається на рівні модульної структури системи, де кожен функціональний блок має чітко визначені кордони та взаємодіє з іншими виключно через публічні інтерфейси або систему подій. Такий підхід дозволяє незалежно розвивати та розгортати компоненти системи у контейнеризованих середовищах, що гарантує ідентичну роботу на різних хост-системах. Програмний комплекс повинен бути сумісним із сучасними браузерами.

Часові характеристики реакції системи є критичними для забезпечення безперервного користувацького досвіду. Встановлено ліміти швидкодії:

- застосування візуальних змін при мутації даних (наприклад, уподобання треку, зміна приватності плейлиста) відбувається миттєво (до 50 мс) за рахунок механізмів оптимістичного оновлення інтерфейсу (Optimistic UI), ще до завершення мережевого запиту до сервера;

- будь-яка візуальна зміна або перехід між розділами (за виключенням завантаження важкого медіаконтенту) має відбуватися протягом не більше 200 мс. Це необхідно для створення відчуття миттєвого відгуку системи на дії користувача;

- час від моменту ініціації відтворення до появи звукового сигналу не повинен перевищувати 2 секунди. Використання формату HLS забезпечує завантаження лише початкових фрагментів маніфесту замість всього об'єму медіафайлу;

- при керуванні стем-плеєром затримка між маніпуляцією елементами інтерфейсу та фактичною зміною відтворення звуку має бути непомітною для людського слуху (до 50-100 мс);

- формування списку рекомендованого контенту на основі векторної схожості має виконуватися в межах 500 мс.

Для формалізації наведених вимог та трансформації їх у план проєктування розроблено модель взаємодії акторів. У межах платформи виділено три основні категорії акторів: слухач (споживає контент, формує алгоритмічний профіль смаку, взаємодіє зі стем-мікшером), незалежний виконавець (наповнює медіатеку, керує життєвим циклом треків, аналізує телеметрію) та адміністратор (забезпечує вирішення конфліктів модерації, керує реєстром та блокує порушників). Функціонали адміністратора та артиста є похідними від функціоналу слухача. Опис акторів наведено у таблиці 2.1.

Таблиця 2.1.

## Виявлені актори

| Актор         | Короткий опис                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Слухач        | Користувач, який взаємодіє з системою для пошуку та потокового споживання аудіоконтенту. Слухач формує власний алгоритмічний профіль смаку (вектор інтересів), може керувати віртуальним стем-мікшером (незалежна зміна гучності доріжок), створювати плейлисти (приватні, публічні), а також взаємодіяти з іншими користувачами (підписки, колаборативні плейлисти).                                                                                                                              |
| Артист        | Користувач, який поширює власний аудіоконтент. Артист ініціює конвеєр обробки мультитрекових релізів, керує життєвим циклом публікацій через матрицю станів, налаштовує семантичні метадані та працює з аналітичними панелями для моніторингу телеметрії (часові ряди прослуховувань, географія аудиторії).                                                                                                                                                                                        |
| Адміністратор | Спеціаліст, відповідальний за підтримку коректної роботи платформи. Виконує процеси модерації (Human-in-the-Loop) для вирішення конфліктів автоматичної ШІ-модерації, здійснює нагляд за дотриманням правил, реагує на зовнішні скарги шляхом блокування/розблокування медіаконтенту, облікових записів, переглядає телеметрію системи, а також виконує сервісні інфраструктурні задачі: реіндексацію векторної бази, перерахунок телеметрії та управління модулем симуляції даних для тестування. |

Виявлені варіанти використання наведено у таблиці 2.2.

Таблиця 2.2

## Виявлені варіанти використання

| Актор  | Найменування                             | Короткий опис                                                                                                                                                                            |
|--------|------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Слухач | Реєстрація та ініціалізація профілю      | Створення облікового запису із проходженням двокрокового інтерактивного онбордингу для розв'язання проблеми «холодного старту» (вибір жанрів та треків для ініціалізації вектора смаку). |
| Слухач | Пошук контенту та отримання рекомендацій | Здійснення пошуку медіаконтенту, а також отримання персоналізованих добірок на основі актуального вектора смаку.                                                                         |
| Слухач | Відтворення потокового аудіо             | Прослуховування медіаконтенту та управління глобальною чергою із забезпеченням безперервного відтворення (Autoplay).                                                                     |
| Слухач | Керування стем-мікшером                  | Динамічне регулювання гучності, ізоляція або повне вимкнення окремих складових композиції (вокал, бас, інструментал тощо) у реальному часі без розриву синхронізації потоків.            |

| Актор         | Найменування                                 | Короткий опис                                                                                                                                                                                                    |
|---------------|----------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Слухач        | Управління плейлистами                       | Створення, глибоке копіювання та редагування плейлистів, налаштування рівнів доступу (публічний/приватний) та додавання співавторів для колаборації.                                                             |
|               | Соціальна взаємодія та формування смаку      | Формування соціального графа (підписки на профілі та плейлисти), уподобання треків, а також застосування негативного зворотного зв'язку («Не цікаво») для поповнення блок-листа та корекції видачі рекомендацій. |
|               | Керування обліковим записом                  | Редагування метаданих профілю (аватар, опис), налаштування параметрів приватності та, за потреби, повне скидання вектора смаку до початкового стану.                                                             |
| Артист        | Завантаження треку                           | Ініціалізація конвеєра завантаження метаданих, аудіофайлів та графічних обкладинок з візуалізацією поточного стану обробки треку.                                                                                |
| Артист        | Налаштування семантичних метаданих           | Визначення назви, опису, жанрової приналежності, додавання користувацьких тегів та встановлення маркера відвертого контенту (Explicit).                                                                          |
| Артист        | Керування життєвим циклом треку              | Зміна стану завантажених матеріалів (публікація, приховування, архівування, відновлення) для управління доступом та їхньою видимістю в системі.                                                                  |
|               | Аналіз телеметрії                            | Перегляд агрегованих показників взаємодії з контентом у реальному часі та аналіз історичних часових рядів (динаміка унікальних слухачів, індекси популярності).                                                  |
|               | Взаємодія з системою модерації               | Подання формальних апеляцій на зняття блокування з контенту.                                                                                                                                                     |
| Адміністратор | Модерація контенту (Human-in-the-Loop)       | Розгляд результатів обробки медіаконтенту, вирішення конфліктів III-модерації та примусове архівування медіаданих, що порушують правила платформи.                                                               |
| Адміністратор | Управління глобальним реєстром та аналітикою | Пошук, фільтрація всього завантаженого контенту, перегляд глобальної системної телеметрії та телеметрії конкретних треків.                                                                                       |

| Актор         | Найменування                                     | Короткий опис                                                                                                                                                   |
|---------------|--------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Адміністратор | Модерація та опрацювання скарг                   | Реакція на зовнішні звернення користувачів через інтерфейс платформи: блокування облікових записів порушників, або контенту із формуванням аудиторського сліду. |
| Адміністратор | Обслуговування інфраструктури та симуляція даних | Ініціалізація перерахунку телеметрії, запуск перерахунку векторних представлень контенту (embeddings).                                                          |

Для візуалізації сценаріїв взаємодії акторів із програмним комплексом розроблено діаграму варіантів використання (рис. 2.1, рис. 2.2).

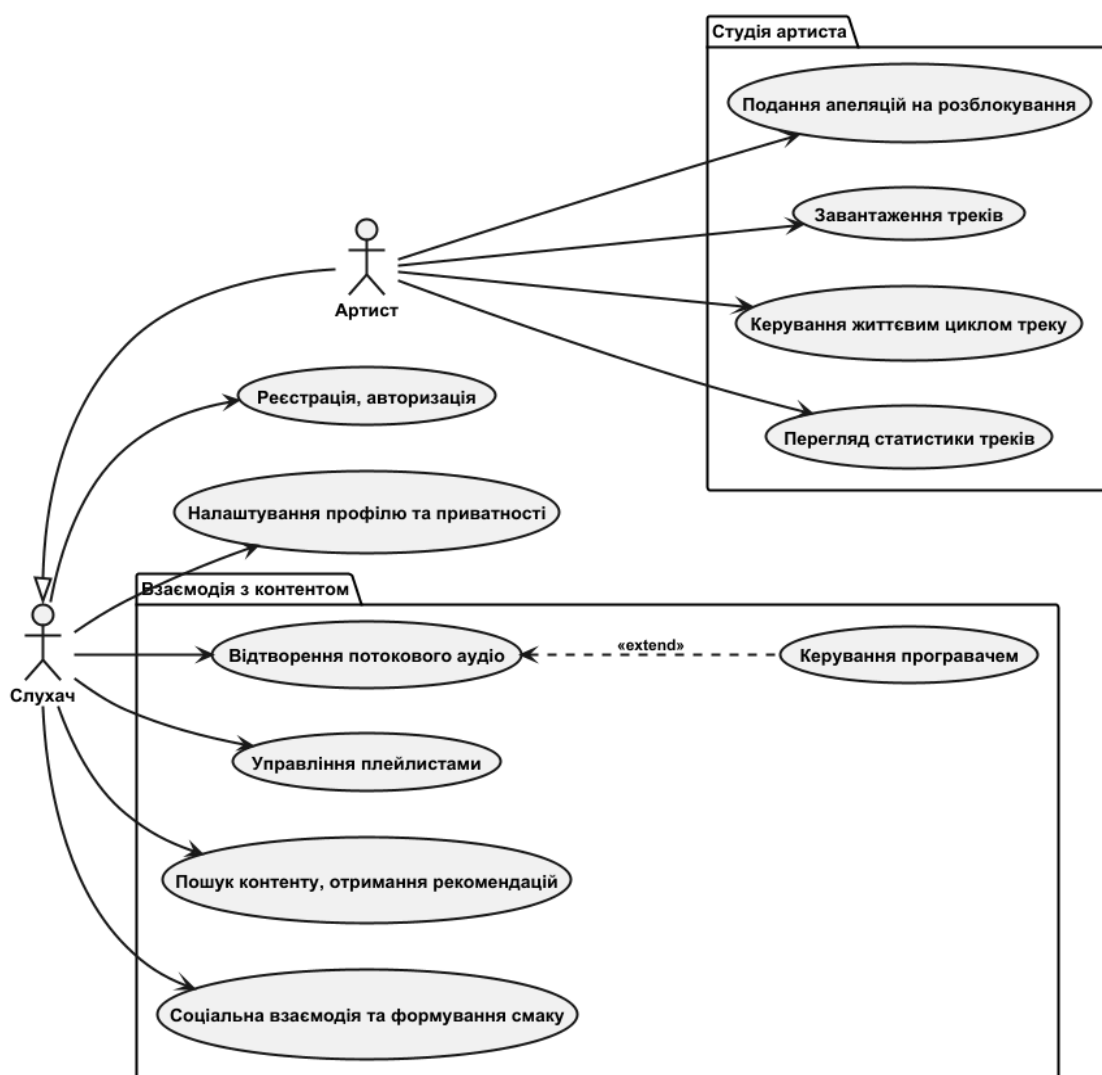


Рис. 2.1 - Діаграма варіантів використання системи слухачем та артистом

|  |  |  |  |  |
|--|--|--|--|--|
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

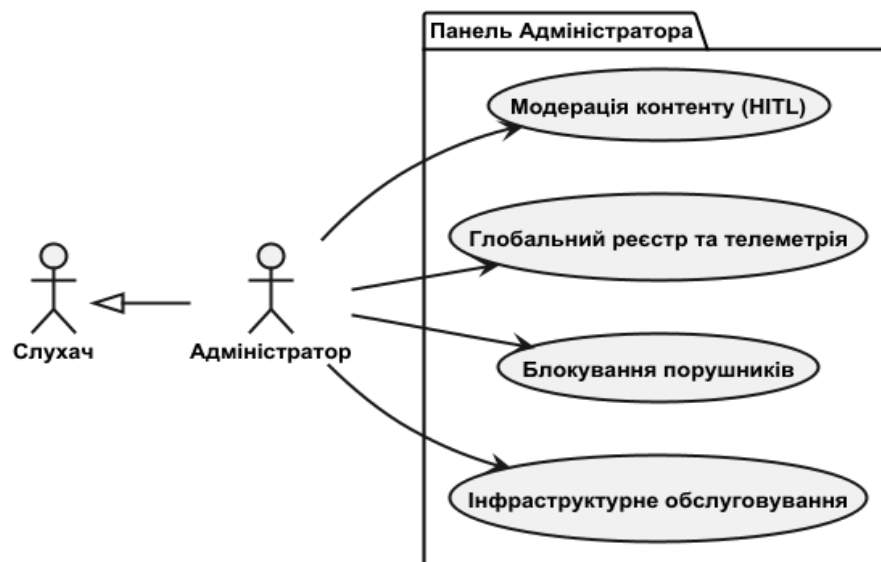


Рис. 2.2 - Діаграма варіантів використання системи адміністратором

Специфікація ключових прецедентів використання наведено в таблицях 2.3 - 2.5. Прецедент керування програвачем (табл. 2.3) визначає механізм інтерактивної взаємодії зі структурою аудіокомпозиції. Враховуючи архітектурну вимогу опціональності мультитрекового формату, система динамічно конструює граф обробки сигналу (Web Audio API) залежно від наявності стемів у метаданих конкретного релізу.

Таблиця 2.3

#### Специфікація прецеденту керування програвачем

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Найменування         | Керування програвачем                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Актор                | Слухач                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Мета                 | Налаштування індивідуального звучання композиції шляхом маніпуляції гучністю її складових компонентів (за їх наявності).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| Передумови           | Користувач авторизований, обрано реліз із підтримкою багатокомпонентного відтворення, ініціалізовано завантаження аудіо                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Основний потік подій | <ol style="list-style-type: none"> <li>1. Система аналізує метадані треку для визначення наявності та конфігурації опціональних аудіостемів.</li> <li>2. Клієнтський застосунок ініціює завантаження аудіо-чанків (HTTP Range) через захищений проксі-маршрут.</li> <li>3. Формується граф Web Audio API, де кожен потік маршрутизується через незалежний вузол підсилення (GainNode).</li> <li>4. Відбувається синхронне відтворення доступних доріжок.</li> <li>5. Слухач взаємодіє з інтерфейсом мікшера (регулювання гучності, ізоляція, вимкнення).</li> <li>6. Система миттєво застосовує зміни до відповідних вузлів підсилення. фазову синхронізацію всіх потоків.</li> </ol> |

|            |                                                                                                                                                                                                    |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Післяумови | Зміни застосовано без розриву фазової синхронізації потоків. Поточний стан відтворення зафіксовано локальним клієнтським рушієм управління станом. Факт прослуховування зафіксовано у буфері подій |
| Специфіка  | Обробка дій слухача виконується локально апаратними засобами браузера із затримкою до 50 мс. У разі відсутності стемів відтворюється єдиний майстер-трек без відображення елементів мікшера.       |

Прецедент завантаження треку (табл. 2.4) описує асинхронний конвеєр наповнення медіатеки. Процес вимагає строгої ізоляції етапів обробки файлів (ETL-конвеєр), генерації оптимізованих форматів та обов'язкового проходження автоматизованої модерації під контролем машини станів (FSM).

Таблиця 2.4

## Специфікація прецеденту завантаження треку

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Найменування         | Завантаження треку                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Актор                | Артист                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| Мета                 | Публікація нового музичного твору.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Передумови           | Користувач автентифікований. Підготовлено графічну обкладинку, базовий аудіофайл (майстер-трек) та опціональний набір стемів (до 8 доріжок) однакової тривалості, які при накладанні один на одного формують акустичну картину, що відповідає базовому аудіофайлу.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Основний потік подій | <ol style="list-style-type: none"> <li>1. Артист заповнює форму метаданих (назва, жанр, маркер вікових обмежень) та ініціює передачу файлів.</li> <li>2. Система зберігає бінарні дані у S3-сумісному сховищі під унікальними ідентифікаторами та переводить об'єкт у статус «PROCESSING».</li> <li>3. Асинхронний конвеєр виконує акустичну валідацію: перевіряє відповідність форматів, тривалості та дельти гучності (LUFS) між стемами і майстер-треком. Також генеруються оптимізовані зображення обкладинок (WebP).</li> <li>4. Система перетворює аудіодані у формат HTTP Live Streaming (генерація HLS-маніфестів та чанків) для забезпечення гарантованої доступності потоку.</li> <li>5. Стан об'єкта змінюється на «AI_ANALYSIS». Ініціюється мультимодальна III-модерація метаданих та контенту.</li> <li>6. За відсутності порушень III генерує семантичні теги, а об'єкт переводиться у статус «APPROVED».</li> <li>7. На основі призначених тегів система формує 768-вимірне векторне представлення треку (embedding).</li> <li>8. Артист отримує можливість перевести трек у статус «PUBLISHED».</li> </ol> |
| Альтернативні потоки | За. Невідповідність аудіоданих: Якщо перевірка тривалості, формату або відхилення LUFS не пройдена, процес переривається. Трек переходить у статус «FAILED» із фіксацією причини помилки.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

|                      |                                                                                                                                                                                                                                                                      |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Альтернативні потоки | ба. Неоднозначний контент (Human-in-the-Loop): Якщо ІІІ виявляє дискусійні або незначні порушення, трек переводиться у статус «IN_REVIEW». Адміністратор платформи здійснює ручну перевірку та виносить рішення про схвалення або відхилення із зазначенням причини. |
| Альтернативні потоки | бб. Критичні порушення: Якщо ІІІ виявляє серйозні порушення правил, трек миттєво переводиться у статус «REJECTED» (або «BANNED»), процес публікації блокується.                                                                                                      |
| Післяумови           | Фізичні файли інкапсульовано у сховищі. Створено векторне представлення треку. Артист отримує доступ до керування станом треку.                                                                                                                                      |
| Специфіка            | Конвеєр обробляє довільну кількість стемів (не більше 8). Клієнтський застосунок використовує механізм опитування (polling) для відображення реального стану обробки через API машини станів без очікування розриву HTTP-з'єднання.                                  |

Прецедент формування вектора смаку (табл. 2.5) деталізує процес збору телеметрії та її трансляції у математичну модель вподобань слухача. Обчислення виконуються асинхронно для запобігання блокування медіа маршрутів.

Таблиця 2.5.

## Специфікація прецеденту формування вектора смаку

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Найменування         | Формування вектора смаку                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Актор                | Слухач                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| Мета                 | Обчислення геометричного центроїда інтересів користувача у багатовимірному просторі для коригування алгоритмічної видачі рекомендацій.                                                                                                                                                                                                                                                                                                   |
| Передумови           | Наявність цільового об'єкта (треку), що має власне векторне представлення (embedding) у СУБД.                                                                                                                                                                                                                                                                                                                                            |
| Основний потік подій | 1. Слухач виконує цільову дію щодо треку (повне відтворення, оцінка, додавання до плейлиста або маркування «Не цікаво»).<br>2. Клієнтський застосунок генерує подію взаємодії з треком.<br>3. Модуль рекомендацій зчитує координати вектора цільового треку і застосовує ваговий коефіцієнт залежно від типу події (наприклад, уподобання має вищу вагу за пасивне прослуховування).<br>4. Алгоритм розраховує зсув вектора користувача. |
| Післяумови           | Оновлені координати користувацького вектора збережено у реляційній БД. Подальші просторові запити на пошук контенту виконуються відносно нового вектора.                                                                                                                                                                                                                                                                                 |
| Специфіка            | При реєстрації негативного зворотного зв'язку (дія «Не цікаво») система ініціює негативний векторний зсув та записує ідентифікатор треку у тимчасовий блок-лист (7 діб) для гарантованого виключення з HNSW-індексу.                                                                                                                                                                                                                     |

Наступним етапом проєктування є визначення логічної структури системи, вираженої через сукупність об'єктно-орієнтованих сутностей та доменних сервісів. В межах архітектури «модульний моноліт» кожна сутність інкапсулює стан та базову логіку конкретної предметної області, забезпечуючи цілісність даних та чітке розділення відповідальності. Взаємодія між ізольованими компонентами відбувається виключно через публічні API-фасади та механізм асинхронних подій. Зважаючи на масштаб платформи «IndieStream», загальну схему декомпозировано на окремі предметні області.

#### Об'єктна структура та опис сутностей системи

Логічна модель системи побудована за принципами предметно-орієнтованого проєктування (DDD), де кожна сутність є частиною відповідного обмеженого контексту.

Домен автентифікації (рис. 2.3) відповідає за ідентифікацію користувачів, збереження їхніх персональних метаданих, керування доступом та алгоритмічним станом профілю. Ядром інформаційної моделі модуля є сутності User та UserProfile, пов'язаних відношенням «один-до-одного». Такий поділ виконано для оптимізації роботи з базою даних: сутність облікових записів містить виключно критичні для авторизації поля (email, username, alias passwordHash, role), що пришвидшує вибірку даних під час перевірки облікових даних фільтрами Spring Security. Поле isBanned слугує первинним системним маркером для оперативного блокування підозрілих або скомпрометованих облікових записів.

Розширена інформація, що не вимагається при кожній транзакції автентифікації, винесена до сутності UserProfile. Вона містить шляхи до медіафайлів обкладинок (avatarPath, bannerPath), текстовий опис (bio) та конфігураційні прапорці приватності (isPrivate, hideSubscriptions). Для усунення проблеми N+1 запитів під час формування публічних сторінок, сутність зберігає денормалізовані лічильники соціального графа (followersCount та followingCount). Поле tasteVector представляє собою 768-вимірний масив чисел з рухомою комою. Цей масив є математичним представленням музичних

вподобань слухача для системи алгоритмічних рекомендацій. Якщо значення цього поля дорівнює нулю (NULL), система ідентифікує стан "холодного старту" та ініціює процедуру первинного калібрування смаку при наступній авторизації.

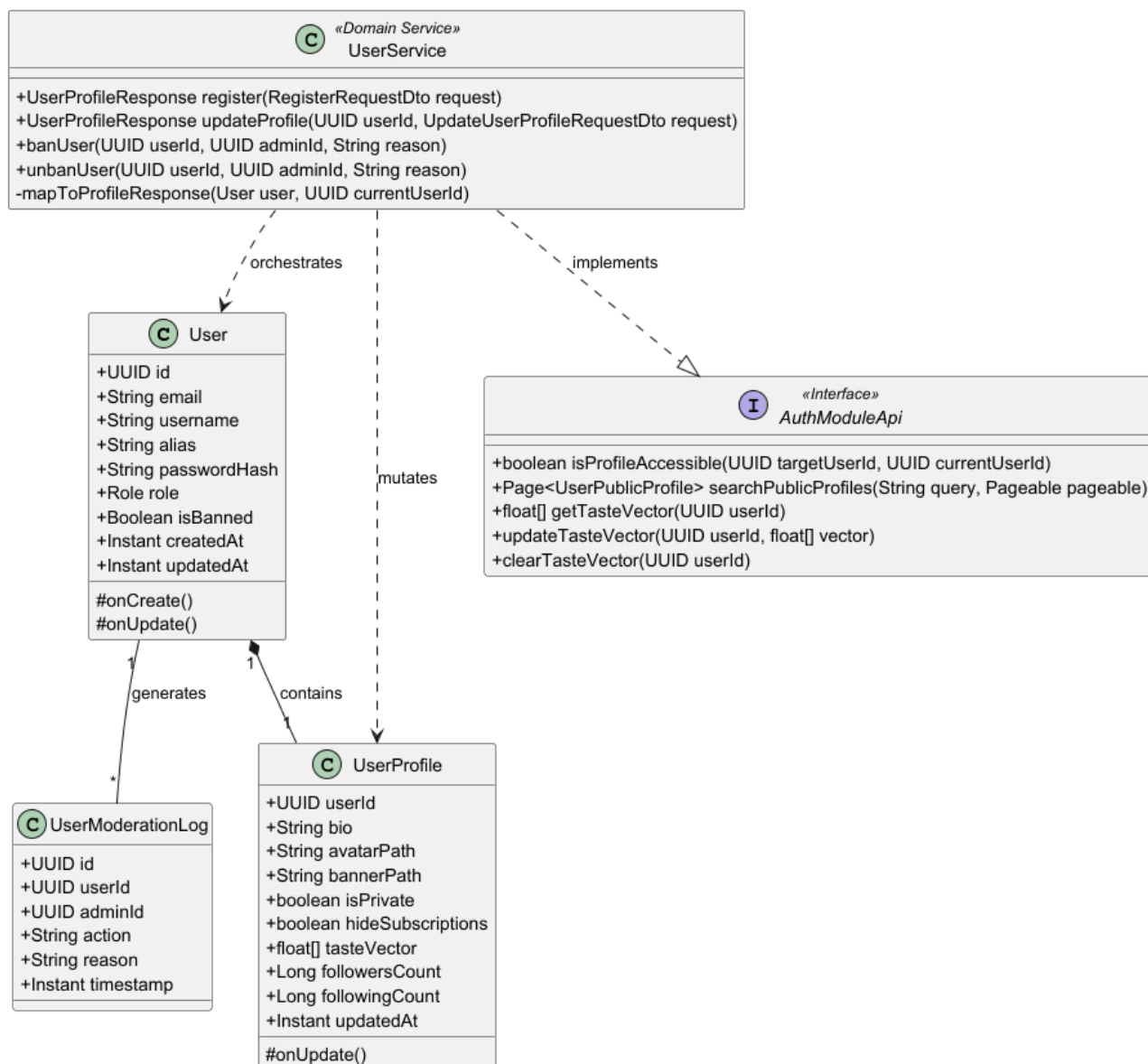


Рис. 2.3 - Структура сутностей, інтерфейсів та методів домену автентифікації

Бізнес-логіка домену інкапсульована у класі `UserService`, який безпосередньо імплементує міжмодульний контракт `AuthModuleApi`. Транзакційний метод `register()` забезпечує атомарне створення об'єктів `User` та `UserProfile` з хешуванням пароля алгоритмом `BCrypt`. Після збереження сутності у базі даних, метод генерує асинхронну подію `UserRegisteredEvent`, яка

сповіщає інші модулі про необхідність виділення ресурсів для нового профілю. Оновлення текстових і медіаданих здійснюється через методи `updateProfile()`, `updateAvatar()` та `updateBanner()`, які виконують перевірку активності профілю перед внесенням змін та делегують фізичне завантаження файлів інфраструктурним сервісам сховища.

Окремим блоком реалізовано методи адміністративного контролю. Метод `banUser()` змінює стан прапорця `isBanned` на істинний, автоматично формує незмінний аудиторський запис через створення об'єкта `UserModerationLog` (із фіксацією ідентифікатора адміністратора та причини), після чого звертається до сервісу `TokenBlacklistService` для миттєвої інвалідації всіх активних JWT-сесій користувача у відмовостійкому буфері `Redis`. Метод `unbanUser()` відновлює доступ.

Для забезпечення міжмодульної взаємодії без порушення інкапсуляції, сервіс реалізує інтерфейс `AuthModuleApi`. Наприклад, домен плейлистів використовує метод `isProfileAccessible()` для верифікації того, чи не заблоковано автора контенту і чи не приховано його профіль налаштуваннями приватності. Метод `searchPublicProfiles()` відповідає за роботу глобального пошуку, гарантуючи приховування заблокованих акаунтів та виконуючи лексикографічне сортування результатів за статусом ролі (пріоритезуючи профілі артистів над звичайними слухачами), а потім за кількістю підписок на нього. Для потреб рекомендаційного рушія фасад надає швидкий доступ до модифікації векторних представлень через методи `getTasteVector()` та `updateTasteVector()`. Метод `clearTasteVector()` дозволяє користувачам самостійно скидати видачу рекомендацій до початкового стану.

Наступним логічним кроком є проектування структури домену медіаконтенту (`Media Module`), який є центральним функціональним ядром платформи «IndieStream». Цей модуль інкапсулює комплексну бізнес-логіку обробки багатокомпонентного аудіо, управління життєвим циклом об'єктів через машину станів (FSM), генерації медіаформатів для потокової передачі та

взаємодії зі штучним інтелектом. Діаграму класів ключових сутностей та сервісів домену наведено на рис. А.1.

Головною сутністю предметної області виступає клас Track. Він зберігає фізичні ідентифікатори файлів у сховищі MinIO: minioBucketPath для майстер-треку, coverMinioPath для обкладинки та hlsManifestPath для оптимізованого потокового відтворення. Для реалізації функціоналу віртуального стем-мікшера на клієнті використовується поле stemsMetadata формату JSONB, яке динамічно зберігає конфігурацію доступних аудіодоріжок (вокал, бас, інструментал тощо). Атрибут status типу TrackStatus визначає поточну фазу обробки та доступності контенту (PROCESSING, AI\_ANALYSIS, PUBLISHED, ARCHIVED та ін.), виступаючи єдиним джерелом істини (Single Source of Truth) для систем контролю доступу. Інтеграція з рекомендаційним рушієм забезпечується полем vector, що містить 768-вимірний масив (векторне представлення треку, обчислене на основі TrackTags та жанру). Агреговані лічильники, такі як playCount та popularityScore, винесені безпосередньо в таблицю треків для оптимізації сортування без виконання ресурсомістких SQL-запитів до модуля телеметрії.

Життєвий цикл кожного треку суворо регламентується машиною станів. Доменний сервіс TrackTransitionEngine відповідає за верифікацію математичної матриці переходів. Його основний метод transitionTrack() валідує легітимність зміни статусу (наприклад, забороняє перехід з ARCHIVED назад у PUBLISHED) і паралельно генерує незмінний запис у сутність TrackAuditLog. Це забезпечує збереження повного аудиторського сліду: ідентифікатора ініціатора зміни, попереднього і нового статусів та причини переходу, що є критично важливим для процесів модерації (Human-in-the-Loop).

Процес обробки «сирих» файлів інкапсульовано у класі AudioPipelineOrchestrator, який функціонує в асинхронному режимі. Метод handleTrackUploadedEvent() перехоплює доменну подію після коміту транзакції завантаження і запускає метод executePipeline(). Алгоритм оркестратора лінійно виконує послідовність операцій: спочатку ініціюється валідація цілісності

(StemAntiSpoofingService) для перевірки акустичної відповідності стемів майстер-треку. У разі успіху викликається синхронна екстракція HLS-сегментів, що усуває стан гонки (race condition) та гарантує наявність потокового маніфесту до моменту передачі об'єкта адміністраторам або III. На завершальному етапі генеруються оптимізовані WebP та OGG асети, після чого об'єкт переводиться у стан AI\_ANALYSIS, а метод executeAiModeration() делегує виклик мультимодальній моделі Gemini.

Міжмодульна взаємодія поділена на спеціалізовані фасади за принципом єдиної відповідальності (SRP). Інтерфейс ArtistModuleApi (реалізований через ArtistTrackManagementService) надає авторам безпечні контракти для мутації життєвого циклу публікацій: publishTrack(), toggleTrackVisibility() та archiveTrack(). Ці методи діють як делегати для TrackTransitionEngine, попередньо перевіряючи право власності через вилучення UUID з контексту безпеки JWT.

Інтерфейс MediaModuleApi (реалізований через TrackService) відповідає за читання та агрегацію даних. Метод getHlsResource() забезпечує прямий захищений доступ до сегментів потокового аудіо через InputStreamResource, ізолюючи реальні шляхи MinIO від клієнта. Методи агрегації та пошуку (getTrackMetadata, searchPublicTracks) виконують перевірку стану видимості (відфільтровуючи все, окрім PUBLISHED для звичайних слухачів) та усувають проблему N+1 запитів шляхом використання O(1) зіставлення авторів та їх публічних профілів (UserProfile), отримані з домену автентифікації.

Третім ключовим компонентом логічної структури системи є домен плейлистів, який відповідає за агрегацію аудіоконтенту в користувацькі колекції, керування параметрами спільного доступу та інтеграцію соціальних графів плейлистів (підписки, співавторство). Діаграму класів ключових сутностей та сервісів домену наведено на рис. 2.4.

Основним агрегатом предметної області виступає сутність Playlist. Для забезпечення високої продуктивності при формуванні списків, сутність містить поля агрегації: trackCount (загальна кількість треків), totalDurationSeconds

(сумарна тривалість аудіо) та followersCount (кількість підписників). Це усуває необхідність виконання ресурсомістких операцій COUNT() та SUM() при кожному завантаженні сторінки профілю. Контроль доступу та поведінки об'єкта визначається трьома логічними прапорцями: isPublic (видимість у глобальному пошуку), isCollaborative (можливість додавання треків сторонніми співавторами) та isSystem (незмінність системних колекцій, таких як «Liked Tracks»). Унікальним атрибутом сутності є поле centroidVector - 768-вимірний масив даних, що зберігає математичний центр інтересів (координати у векторному просторі) всіх доданих до плейлиста композицій для роботи алгоритмів рекомендацій.

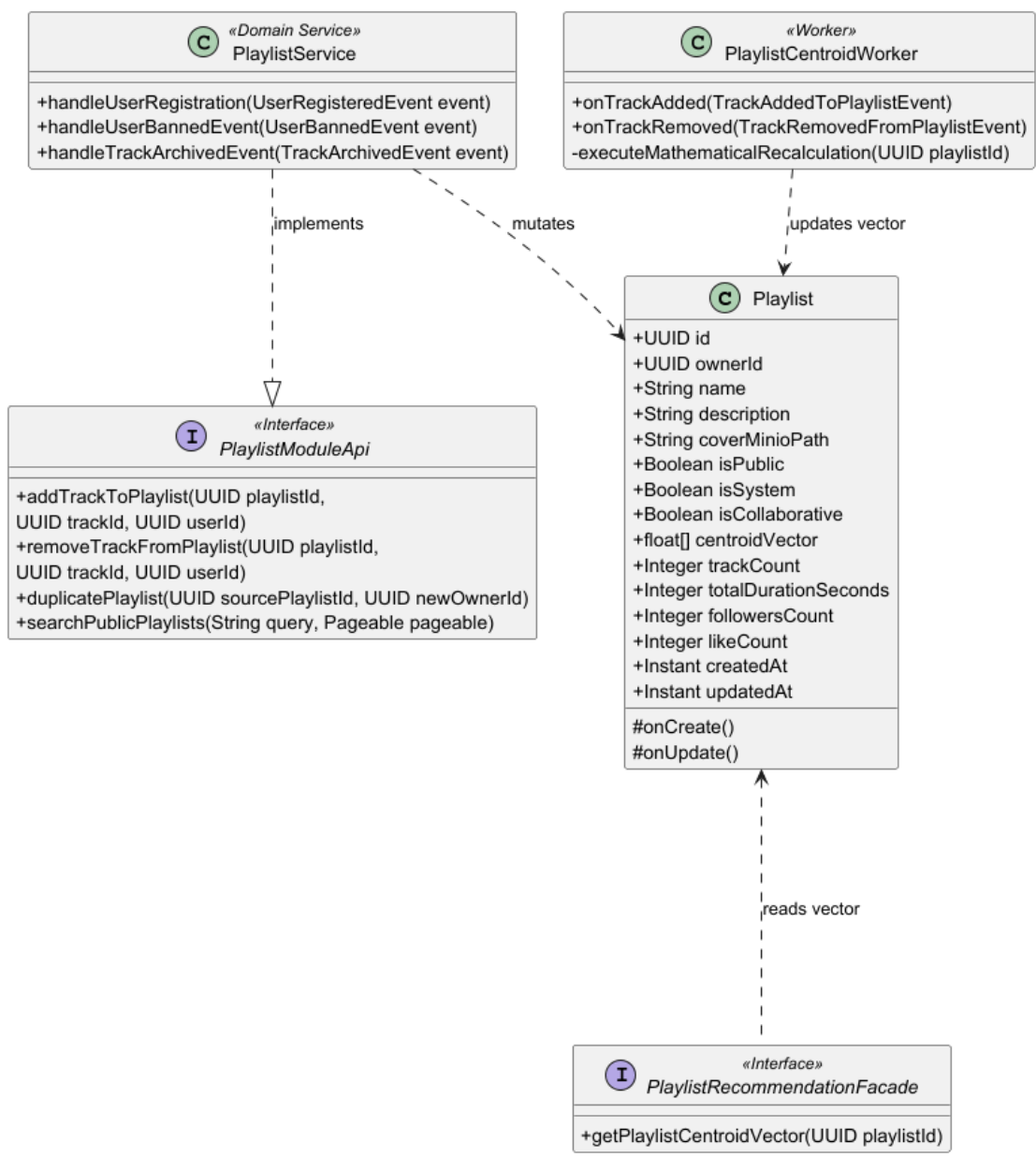


Рис. 2.4 - Структура сутностей, інтерфейсів та методів домену плейлистів

Керування бізнес-логікою та міжмодульними контрактами інкапсульовано у класі `PlaylistService`, що реалізує інтерфейс `PlaylistModuleApi`. Додавання та видалення треків через методи `addTrackToPlaylist()` і `removeTrackFromPlaylist()` виконується із застосуванням песимістичного блокування бази даних (`findByIdWithPessimisticWrite`). Це рішення гарантує математичну точність перерахунку полів `trackCount` та `totalDurationSeconds` навіть за умови одночасних запитів від кількох співавторів колекції. Метод `duplicatePlaylist()` реалізує функціонал глибокого копіювання; для уникнення вичерпання оперативної пам'яті сервера та перевищення лімітів часу з'єднань на великих наборах даних, метод використовує нативний SQL-запит масового вставлення (`bulk insert`) складністю  $O(1)$ , замість стандартної ORM-ітерації (ітеративного обходу колекції сутностей).

Окрема увага в архітектурі сервісу приділена реактивності та каскадній відповідності даних через механізм доменних подій. Асинхронний метод `handleUserRegistration()` автоматично створює системний плейлист «Liked Tracks» при реєстрації нового слухача. Метод `handleUserBannedEvent()` перехоплює подію блокування облікового запису та ініціює каскадне приховування всіх пов'язаних колекцій з публічного доступу. Аналогічно, `handleTrackArchivedEvent()` реагує на архівування або блокування конкретного треку в домені медіа, ініціюючи безпечне масове очищення посилань на цей трек у таблиці зв'язків та перерахунок загальної тривалості постраждалих плейлистів.

Взаємодія з рекомендаційним рушієм винесена у відокремлені компоненти для дотримання принципів SOLID. Асинхронний обробник `PlaylistCentroidWorker` реагує на події додавання (`TrackAddedToPlaylistEvent`) або видалення треків. Обробник налаштований на фазу `AFTER_COMMIT`, він гарантовано запускає метод `executeMathematicalRecalculation()` лише після успішного збереження нових зв'язків у базі даних. Цей метод виконує нативний запит до розширення `pgvector` для перерахунку середнього значення всіх векторів треків колекції. Результат обчислень експонується для інших модулів

платформи через `PlaylistRecommendationFacade` та його метод `getPlaylistCentroidVector()`, який дозволяє виконувати пошук семантично схожого контенту у багатовимірному просторі векторів.

Домен телеметрії відповідає за збір даних про взаємодію користувачів із платформою, захист від штучного відтворення та агрегацію метрик аналітики (CQRS). Специфікою даного модуля є повна відмова від використання «важких» ORM-сутностей (JPA Entities) на користь архітектури Lambda. Модуль взаємодіє з базою даних виключно через оптимізовані запити репозиторіїв та «легкі» структури даних (Records), що дозволяє витримувати інтенсивні потоки подій. Узагальнену структуру ключових класів та сервісів домену наведено на рис. Б.1.

Оскільки логіку домену оптимізовано для масового запису, базовими моделями даних виступають незмінні об'єкти (Records), такі як `RawPlaybackRecord` та `RawInteractionRecord`. Вони містять ключові атрибути подій: ідентифікатори сесії, треку, користувача, тривалість відтворення (`playbackDurationMs`), статус (`playbackStatus`), а також телеметричні метадані (`clientId`, `userAgent`). Ці структури функціонують як об'єкти передачі даних (DTO) і використовуються для групового збереження записів (пакетної обробки) засобами нативного JDBC [29]. Таке архітектурне рішення дозволяє оптимізувати запис великих масивів інформації в базу даних та уникнути додаткових обчислювальних витрат, властивих ORM.

Точкою входу потоку подій виступає сервіс `TelemetryIngestionGateway`. Його методи `ingestPlayback()` та `ingestInteraction()` функціонують в асинхронному режимі без очікування підтвердження виконання, забезпечуючи миттєву серіалізацію вхідних об'єктів у структуру `Map<String, String>` та їх подальше перенесення до відмовостійкого буфера `Redis Streams`. Інтеграція шаблону стійкості `Circuit Breaker` гарантує, що у разі тимчасової недоступності буфера проміжних даних активуються резервні методи (зокрема `fallbackPlaybackIngestion()`). Це дозволяє ізолювати збій інфраструктури та

локально обмежити обробку телеметрії без порушення процесу відтворення аудіоконтенту на стороні клієнтського застосунку.

Асинхронна обробка та стиснення накопичених даних інкапсульовані в розподіленому планувальнику `TimeWindowRollupWorker`. Метод `executeMicroBatchRollupCron()` запускається кожні 5 хвилин для мікробатчінгу: він витягує сирі дані, виконує агрегацію для надання авторам аналітики в режимі близькому до реального часу, та паралельно публікує події `PassiveTasteShiftsAggregatedEvent` для модуля рекомендацій. Методи `executeHourlyEventBroadcastCron()` та `executeDailyRollupCron()` виконують фіналізацію часових вікон, агрегуючі відповідні записи. Усі операції з базою даних сервіс делегує безпосередньо репозиторіям, таким як `AnalyticsQueryRepository`.

Аналітична частина системи побудована за патерном CQRS [30] (Command Query Responsibility Segregation), де операції читання повністю відокремлені від запису. Сервіси на кшталт `ArtistAnalyticsService`, `CuratorAnalyticsService` та `AdminAnalyticsService` відповідають виключно за формування DTO для статистики (наприклад, `getTrackAnalyticsWithRealTime()`). Вони звертаються до `AnalyticsQueryRepository` для виконання складних SQL-проєкцій та узагальнених табличних виразів (CTE) без завантаження повних графів об'єктів у пам'ять. Міжмодульна взаємодія формалізована через фасад `TelemetryModuleApi`, метод якого `getRecentTrackIds()` дозволяє іншим доменам отримувати історію дій слухача.

Домен рекомендацій інкапсулює алгоритми семантичного аналізу, керування векторними просторовими даними та математичне моделювання вподобань користувачів. Модуль базується на розширенні PostgreSQL `pgvector` з використанням HNSW-індексів для виконання просторових запитів. Робота модуля відбувається в асинхронному режимі, повністю ізолюючи обчислення від критичного шляху потокової передачі медіа. Діаграму класів сервісів та обробників домену наведено на рис. 2.5.

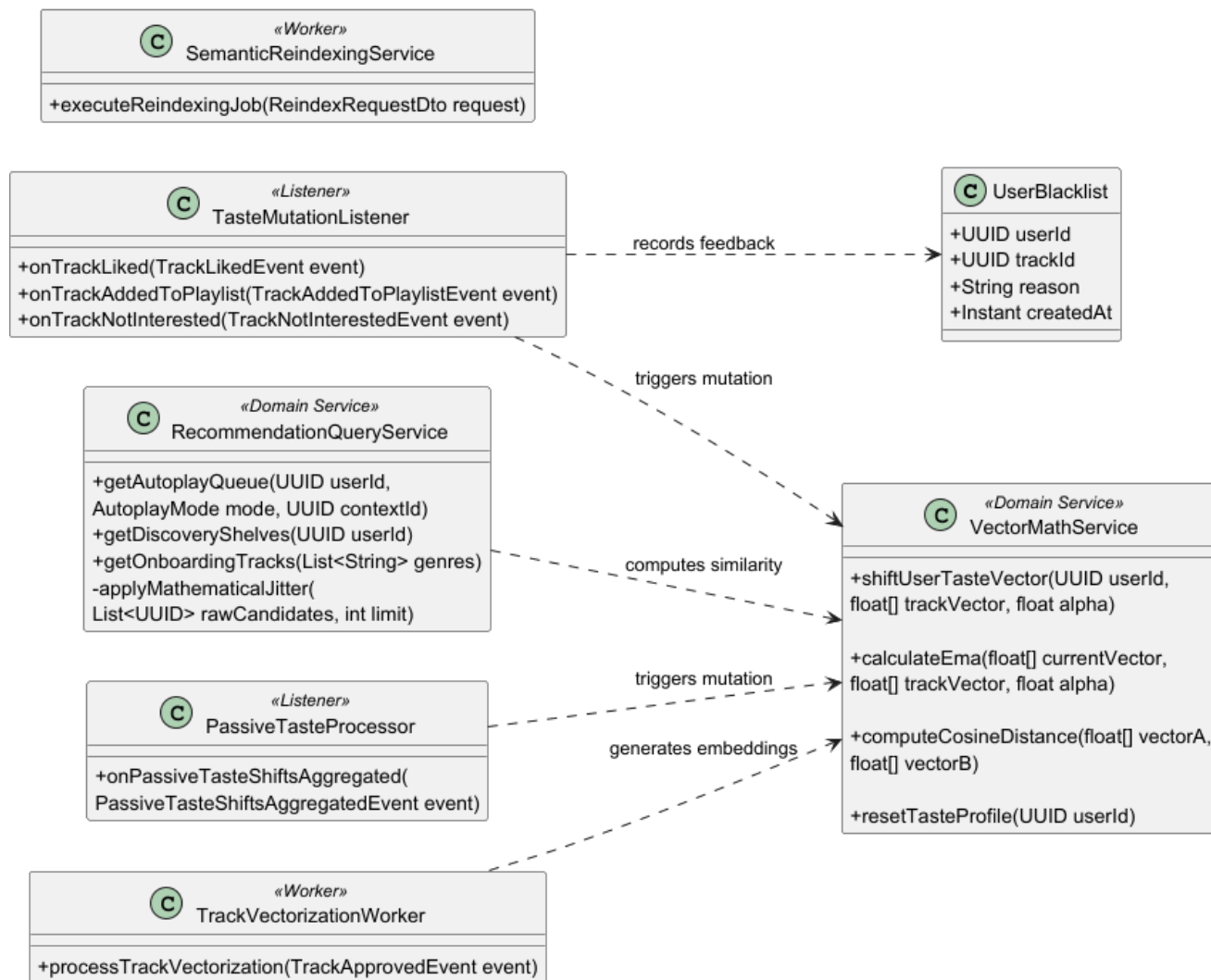


Рис. 2.5 - Структура сутностей, інтерфейсів та методів домену рекомендацій

Стан користувачів та медіаконтенту зберігається у відповідних доменах автентифікації та медіа, тому єдиною локальною сутністю модуля є клас UserBlacklist. Він фіксує негативний зворотний зв'язок (подія «Не цікаво») та формує динамічний блок-лист. На рівні бази даних це забезпечує гарантоване виключення небажаних об'єктів із простору пошуку алгоритмічної видачі на визначений термін у 7 діб.

Математичне моделювання та розрахунки у багатовимірному векторному просторі забезпечує доменний сервіс VectorMathService. Метод calculateEma() реалізує алгоритм експоненційного ковзного середнього для модифікації 768-вимірному вектора музичних вподобань користувача. Розрахунки використовують фіксовані вагові коефіцієнти, що відповідають семантиці дій

|  |  |  |  |  |
|--|--|--|--|--|
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

слухача: позитивні оцінки та додавання творів до колекцій формують позитивне зміщення вектора, тоді як пропуск композиції зумовлює негативний зсув. Зіставлення просторових координат та визначення ступеня схожості об'єктів виконується у методі `computeCosineDistance()` за допомогою нативного оператора косинусної відстані ( $\leq$ ).

Корекція координат вектора смаку ініціюється системою асинхронних обробників подій. Явні та однозначні дії користувача, такі як уподобання або додавання треку до плейлиста, опрацьовуються сервісом `TasteMutationListener` миттєво після фіксації відповідних доменних подій. Для захисту математичної моделі від навмисного спотворення через багаторазове повторення однакових дій застосовується механізм перевірки ідемпотентності на базі сховища `Redis` із терміном дії блокування у сім діб. Це унеможливорює повторні векторні зсуви при багаторазових однотипних взаємодіях з одним і тим самим треком. Натомість пасивні взаємодії, зокрема факти відтворення аудіосигналу, передаються до класу `PassiveTasteProcessor` у формі періодичних мікропакетів телеметрії, оскільки такі події відбуваються значно частіше та мають менший ваговий вплив на підсумкове положення вектора інтересів.

Розрахунок початкових просторових координат для нового медіаконтенту здійснюється асинхронним компонентом `TrackVectorizationWorker`. Метод `processTrackVectorization()` перехоплює подію успішного завершення модерації треку - стану схвалено, на одному з попередніх етапів якої модель `Google Gemini` проаналізувала мультимодальний вміст та сформувала семантичні теги. На основі сформованих текстових метаданих воркер звертається до інфраструктурного сервісу `OpenAI` для генерації 768-вимірною векторного представлення за допомогою моделі класу `Embedding Small`. Для масового оновлення координат у межах усього каталогу застосовується `SemanticReindexingService`. Його алгоритм передбачає фіксовану часову паузу між ітераціями звернень до зовнішнього інтерфейсу для дотримання встановлених лімітів частоти запитів та запобігання виникненню помилок із кодом `HTTP 429`.

Формування персоналізованої видачі для клієнтського інтерфейсу координується класом RecommendationQueryService. Сервіс забезпечує побудову багатокомпонентних добірок у методі getDiscoveryShelves(), первинне налаштування алгоритму шляхом видачі треків в методі getOnboardingTracks() користувачу без ініціалізованого вектора смаку, а також генерацію черги відтворення для безперервного відтворення треків в методі getAutoplayQueue(). Для запобігання повторенням контенту реалізовано підсистему протидії втомі алгоритму, яка виключає з пошукової вибірки композиції, зафіксовані у журналах прослуховувань користувача за останні сім діб. З метою уникнення зациклення рекомендацій та ізоляції користувача в межах однорідного контенту, метод applyMathematicalJitter() вносить випадковий математичний шум в просторові оцінки. Що забезпечує варіативність видачі та стабільне оновлення пропонованого матеріалу під час кожного нового запиту.

## 2.2 Розробка бази даних системи

Зберігання великих бінарних об'єктів (BLOB) організовано в єдиному контейнері сумісного з S3 сховища MinIO. Для дотримання архітектурних обмежень модульного моноліту (Spring Modulith), взаємодія зі сховищем строго ізольована. Відповідальність за управління контентом розподілено між відповідними модулями системи (Auth, Playlist, Media). Кожен модуль використовує власний ізольований екземпляр клієнта сховища, що виключає звернення до внутрішніх структур даних інших доменів.

Внутрішня ієрархія файлів у контейнері формується за принципом логічної ізоляції об'єктів з використанням унікальних ідентифікаторів (UUID) для запобігання колізіям. Маршрутизація даних відбувається за ідентифікаторами відповідних сутностей. Для медіафайлів (майстер-треки, обкладинки, аудіо-стеми) базовим рівнем ізоляції є ідентифікатор виконавця (artistId). Структура збереження стемів передбачає додаткову жорстку санітизацію назв директорій: усі спеціальні символи видаляються регулярними виразами, а символи переводяться в нижній регістр для блокування вразливостей типу Path Traversal. Дані плейлистів та профілів користувачів

ізолюються у власних директоріях за ідентифікаторами відповідних сутностей (playlistId та userId відповідно).

При оновленні користувацького медіа (аватарів, банерів або обкладинок) система виконує запит на видалення попереднього об'єкта зі сховища перед фіксацією нового шляху в базі даних PostgreSQL. Операція видалення налаштована за принципом «тихої відмови» відсутність файлу у сховищі не призводить до переривання транзакції в основній базі даних, що гарантує цілісність реляційних записів.

Збереження згенерованого потокового контенту формату HTTP Live Streaming (HLS) реалізовано через завантаження директорій, сформованих інструментами конвертації. Під час завантаження система динамічно призначає відповідні MIME-типи для коректної маршрутизації: маніфести (.m3u8) маркуються як application/vnd.apple.mpegurl, сегменти (.ts) - як video/MP2T [31], а нетипізовані файли зберігаються як бінарні масиви. Результатом цієї операції є збереження в базі даних абсолютного шляху до головного маніфесту.

Архітектура системи повністю виключає прямий доступ клієнтських додатків до об'єктного сховища. MinIO функціонує виключно у закритій приватній мережі, а система бекенду виконує роль авторизаційного проксі-шару. Отримання даних зі сховища здійснюється у вигляді цілісних вхідних потоків без застосування обмежень на діапазон байтів, що забезпечує коректне завантаження маніфестів HLS та окремих сегментів. Зчитаний потік транслюється клієнту через обгортку потокової передачі на рівні контролера. Передача медіаданих супроводжується жорстким контролем доступу на основі JWT та перевіркою стану сутності за допомогою машини станів (FSM). Конвертація отриманих бінарних відповідей у віртуальні посилання в оперативній пам'яті (Blob) на клієнтській стороні виключає можливість витоку або підробки оригінальних URL-адрес об'єктного сховища.

Для оптимізації високоінтенсивних операцій вводу/виводу, ізоляції бази даних від пікових навантажень та управління тимчасовими станами системи використовується in-memory сховище Redis. Архітектурно взаємодію з Redis

поділено на кілька функціональних доменів: буферизацію телеметрії, механізми захисту, обчислення метрик реального часу, кешування та керування доступом.

Підсистема телеметрії використовує структуру даних Redis Streams. Ініціалізація потоків та групи обробників (`group:telemetry-workers`) відбувається автоматично на етапі запуску додатка. Задля контролю споживання оперативної пам'яті довжина кожного потоку обмежена параметром `maxlen(10000)`, а порожні значення примусово відфільтровуються перед серіалізацією записів. Первинні дані фактів прослуховування фіксуються у потоці `stream:playback`. Структура такого запису містить ідентифікатор події, ідентифікатор треку, тривалість відтворення, метадані клієнта (ідентифікатор користувача або IP-адресу для анонімних сесій, User-Agent, ISO-код країни) та часову мітку надходження запиту. Цільові дії користувачів (наприклад, вподобання чи додавання до плейлиста) записуються у потік `stream:interactions` зі збереженням ідентифікатора користувача, типу взаємодії та ідентифікатора цільової сутності. Повідомлення, серіалізація або обробка яких завершилась помилкою, перенаправляються до черги повідомлень `stream:dlq`, що фіксує ідентифікатор події, тип операції, текст виключення та час виникнення збою.

Системне кешування (Spring Cache) побудоване на ізоляції просторів імен через обов'язкове додавання префіксу `cacheName::` до ключів. Перетворення структур даних здійснюється за допомогою `StringRedisSerializer` для ключів та `GenericJackson2JsonRedisSerializer` для значень. Підключення модуля `JavaTimeModule` забезпечує коректну підтримку типів `OffsetDateTime` та збереження поліморфної інформації про об'єкти. Кеш логічно розподілено на три регіони сегменти від інтенсивності оновлення даних: `analytics:historical` з часом життя (TTL) 10 хвилин для ресурсоємних аналітичних CQRS-запитів, `analytics:live` з TTL 30 секунд для високочастотних метрик реального часу, та базовий кеш із TTL 15 хвилин для загальних даних. Збереження порожніх значень у кеші заборонено на рівні конфігурації.

Модуль контролю лімітів та запобігання штучному наповненню статистики (Fraud Guards) спирається на атомарне виконання Lua-скриптів. Що

|  |  |  |  |  |                    |      |
|--|--|--|--|--|--------------------|------|
|  |  |  |  |  | ІПЗ.КР.Б-121-26-ПЗ | Арк. |
|  |  |  |  |  |                    | 47   |
|  |  |  |  |  |                    |      |

дозволяє об'єднати операції інкременту та встановлення часу життя ключа в єдину транзакцію. Захист від масової генерації прослуховувань з однієї IP-адреси реалізовано через ключ `telemetry:farm:ip:<clientId>`. Запис функціонує як накопичувальний лічильник (INCRBY) загального часу відтвореного аудіо. Перевищення порогу в 16 годин протягом 24-годинного вікна призводить до маркування трафіку як автоматизованого. Захист від циклічного прослуховування одного треку фіксується ключем `telemetry:loop:<userId>:<trackId>`, який обмежує зарахування метрик до 10 разів на добу. Обидва ключі безпеки отримують суворий TTL у 24 години.

Відстеження активності користувачів у реальному часі побудовано на базі структури впорядкованих множин (Sorted Sets), що забезпечує логарифмічну складність операцій  $O(\log N)$ . Система формує список активних слухачів кожного треку у ключі `pulse:track:<trackId>`. Елементом множини виступає ідентифікатор користувача, а метрикою сортування (Score) - поточна часова мітка у мілісекундах. Активні слухачі рахуються за 5-хвилинне вікно: при кожному новому оновленні статусу застарілі записи синхронно видаляються операцією `removeRangeByScore`, після чого актуальна кількість слухачів розраховується за допомогою команди `ZCOUNT`.

Для забезпечення ідемпотентності мутацій векторів рекомендацій впроваджено механізм блокування. Він запобігає багаторазовим зсувам математичного центроїда профілю через повторювані дії користувача (наприклад, циклічне додавання та видалення треку). При виконанні дії система генерує ключ `recommendation:taste_shift:<userId>:<trackId>:<action>` зі значенням «LOCKED». Запис створюється через метод `setIfAbsent` із встановленим TTL 7 днів. Наявність цього ключа блокує подальші спроби перерахунку вектора за вказаним треком та дією.

Анульовані JWT-токени сесій зберігаються у ключі `jwt:blacklist:<token>` зі значенням «revoked». Час життя кожного ключа встановлюється динамічно і точно дорівнює залишковому часу дії самого токена на момент його анулювання. Ключ `jwt:banned_user:<userId>` встановлюється примусово на 7

днів для заблокованих користувачів, зумовлює миттєву відмову в доступі на рівні фільтрів безпеки ще до звернення до основної бази даних.

Для забезпечення збереження структурованої інформації, підтримки транзакційної цілісності (ACID) та реалізації складної реляційної моделі предметної області в системі «IndieStream» застосовано СУБД PostgreSQL 16. Вибір зумовлений наявною підтримкою розширення pgvector, що забезпечує зберігання 768-вимірних векторів та високопродуктивне виконання просторових запитів на основі косинусної відстані (оператор  $\leq$ ) з використанням HNSW-індексів безпосередньо на рівні бази даних. Це виключає необхідність транспортування великих масивів даних до оперативної пам'яті додатку для розрахунку математичних центроїдів рекомендаційної системи. Керування схемою бази даних здійснюється виключно через версіоновані міграції Flyway (виключаючи використання механізмів автоматичного оновлення DDL), що гарантує детермінованість структури у середовищах розробки та розгортання. Опис схеми даних розділено на детальний розгляд ключових таблиць, що формують ядро предметної області, та огляд допоміжних структур.

Таблиця tracks є центральною сутністю бази даних, що відповідає за збереження метаданих музичного контенту та інтеграцію із зовнішніми підсистемами. Вона керує станами обробки файлів та зберігає шляхи маршрутизації до об'єктного сховища. Для підтримки гнучкості метаданих від AI-модуля використовується тип jsonb, запити до якого оптимізовані за допомогою GIN-індексів (idx\_tracks\_stems\_metadata, idx\_tracks\_tags). Для розрахунків рекомендацій таблиця містить векторне представлення семантичних тегів аудіо-характеристик (vector(768)), швидкість пошуку за яким забезпечується HNSW-індексом. Опис структури наведений у табл. 2.6.

## Структура таблиці tracks

| Поле                                  | Тип даних        | Опис                                                      |
|---------------------------------------|------------------|-----------------------------------------------------------|
| id                                    | uuid             | Первинний ключ запису.                                    |
| artist_id                             | uuid             | Зовнішній ключ (із ON DELETE CASCADE) на виконавця.       |
| title                                 | varchar(255)     | Назва аудіотреку.                                         |
| minio_bucket_path                     | varchar(512)     | Абсолютний шлях до оригінального майстер-треку у MinIO.   |
| cover_minio_path                      | varchar(512)     | Шлях до обкладинки треку у сховищі.                       |
| hls_manifest_path                     | varchar(255)     | Шлях до згенерованого .m3u8 маніфесту.                    |
| stems_metadata                        | jsonb            | Словник метаданих розщеплених аудіодоріжок (стемів).      |
| duration_seconds                      | integer          | Тривалість треку в секундах.                              |
| genre                                 | varchar(100)     | Жанр композиції.                                          |
| is_explicit                           | boolean          | Прапор наявності ненормативної лексики.                   |
| tags                                  | jsonb            | AI-згенеровані теги та теги, запропоновані виконавцем.    |
| status                                | varchar(50)      | Поточний стан життєвого циклу треку (FSM).                |
| has_appealed                          | boolean          | Прапор подачі апеляції на модерацію.                      |
| play_count,<br>skip_count, like_count | integer          | Кешовані агреговані лічильники взаємодій.                 |
| popularity_score                      | double precision | Розрахункова метрика популярності з урахуванням згасання. |
| vector                                | vector(768)      | Масив чисел з плаваючою комою для векторного пошуку.      |

Дані користувачів розділені на дві незалежні таблиці: users та user\_profiles. Таблиця users (табл. 2.7) зберігає виключно критичні та конфіденційні дані для перевірки ідентичності та керування доступом.

Структура таблиці users

| Поле          | Тип даних    | Опис                                    |
|---------------|--------------|-----------------------------------------|
| id            | uuid         | Первинний ключ користувача.             |
| email         | varchar(255) | Унікальна електронна адреса.            |
| username      | varchar(50)  | Унікальне системне ім'я.                |
| alias         | varchar(100) | Відображуване ім'я користувача (UTF-8). |
| password_hash | varchar(255) | Хеш пароля для автентифікації.          |
| role          | varchar(50)  | Системна роль користувача.              |
| is_banned     | boolean      | Прапор глобального блокування.          |

Таблиця user\_profiles (табл. 2.8) зв'язана з users відношенням один-до-одного та зберігає публічну інформацію, соціальні метрики, а також алгоритмічний вектор вподобань taste\_vector, що використовується для математичної видачі рекомендаційної стрічки.

Таблиця 2.8

Структура таблиці user\_profiles

| Поле                             | Тип даних    | Опис                                                     |
|----------------------------------|--------------|----------------------------------------------------------|
| user_id                          | uuid         | Первинний ключ та зовнішній зв'язок (ON DELETE CASCADE). |
| bio                              | varchar(500) | Текстовий опис профілю.                                  |
| avatar_path, banner_path         | varchar(255) | Шляхи до графічних ресурсів профілю у MinIO.             |
| is_private                       | boolean      | Прапор загального приховання профілю.                    |
| hide_subscriptions               | boolean      | Прапор приховання списку підписок.                       |
| followers_count, following_count | bigint       | Денормалізовані лічильники соціального графа.            |
| taste_vector                     | vector(768)  | Вектор представлення музичних смаків користувача.        |

Таблиця playlists (табл. 2.9) відповідає за логічне групування контенту. Для уникнення N+1 проблем та оптимізації пагінації у BFF-шарі таблиця

містить денормалізовані сумарні метрики тривалості та кількості треків. Поле `centroid_vector` є результатом асинхронних обчислень фонових обробників і містить геометричний центр векторів усіх доданих треків, на основі чого виконується косинусний пошук подібних плейлистів до вектора смаку користувача.

Таблиця 2.9

#### Структура таблиці `playlists`

| Поле                                             | Тип даних                  | Опис                                                  |
|--------------------------------------------------|----------------------------|-------------------------------------------------------|
| <code>id</code>                                  | <code>uuid</code>          | Первинний ключ плейлиста.                             |
| <code>owner_id</code>                            | <code>uuid</code>          | Ідентифікатор користувача-власника.                   |
| <code>name, description</code>                   | <code>varchar, text</code> | Назва та розширений текстовий опис плейлиста.         |
| <code>cover_minio_path</code>                    | <code>varchar(512)</code>  | Шлях до обкладинки плейлиста.                         |
| <code>is_public</code>                           | <code>boolean</code>       | Прапор видимості для звичайного користувача.          |
| <code>is_system</code>                           | <code>boolean</code>       | Маркер системних плейлистів (наприклад, "Вподобані"). |
| <code>is_collaborative</code>                    | <code>boolean</code>       | Прапор дозволу на спільне редагування.                |
| <code>track_count, total_duration_seconds</code> | <code>integer</code>       | Попередньо агреговані обсяги контенту.                |
| <code>followers_count, like_count</code>         | <code>integer</code>       | Денормалізовані метрики взаємодій користувачів.       |
| <code>centroid_vector</code>                     | <code>vector(768)</code>   | Математичний центр усіх векторів треків плейлиста.    |

Збереження даних прослуховувань реалізовано за допомогою секціонованої таблиці `raw_playback_logs` (табл. 2.10). Розбиття таблиці за часовим діапазоном місяця (`partition by RANGE (created_at)`) забезпечує оптимізацію дискового вводу-виводу при масових вставках та спрощує переміщення і зчитування історичних даних. Таблиця фіксує всі контекстні метрики програвання для роботи системи телеметрії та рекомендацій.

Структура таблиці raw\_playback\_logs

| Поле                                  | Тип даних            | Опис                                                            |
|---------------------------------------|----------------------|-----------------------------------------------------------------|
| event_id                              | uuid                 | Унікальний ідентифікатор події телеметрії.                      |
| user_id                               | uuid                 | Ідентифікатор користувача                                       |
| track_id                              | uuid                 | Ідентифікатор відтвореного треку.                               |
| session_id                            | uuid                 | Ідентифікатор поточної сесії відтворення.                       |
| start_position_ms,<br>end_position_ms | integer              | Фактичні мітки початку та завершення прослуховування.           |
| playback_duration_ms                  | integer              | Загальна тривалість активного відтворення.                      |
| client_ip, user_agent                 | varchar, text        | Метрики клієнтського середовища.                                |
| is_suspected_bot                      | boolean              | Прапор ізоляції трафіку за правилами Farm Guards.               |
| playback_status                       | varchar(20)          | Розрахунковий статус: SKIP, PARTIAL, або FULL.                  |
| source_type, source_id                | varchar(50),<br>uuid | Джерело відтворення (наприклад, конкретний плейлист або пошук). |
| client_country                        | varchar(2)           | Двосимвольний ISO-код країни слухача.                           |
| created_at                            | timestamp            | Час події, ключ секціонування таблиці.                          |

Таблиця raw\_interaction\_logs (табл. 2.11) виконує роль журналу для реєстрації первинних дій користувачів. Вона фіксує факти вподобань, додавання аудіотреків до плейлистів та підписки на інші профілі. Для цієї структури впроваджено щомісячне секціонування за часом. Такий підхід оптимізує дискові операції під час збереження великих масивів подій.

Таблиця 2.11

Структура таблиці raw\_interaction\_logs

| Поле     | Тип даних | Опис                                                |
|----------|-----------|-----------------------------------------------------|
| event_id | uuid      | Унікальний системний ідентифікатор події взаємодії. |
| user_id  | uuid      | Ідентифікатор користувача, що виконав дію           |

| Поле             | Тип даних   | Опис                                                        |
|------------------|-------------|-------------------------------------------------------------|
| target_id        | uuid        | Ідентифікатор цільового об'єкта взаємодії.                  |
| interaction_type | varchar(50) | Тип зареєстрованої дії.                                     |
| source_type      | varchar(50) | Контекст виклику події в інтерфейсі.                        |
| ui_surface       | varchar(50) | Графічний елемент, де зафіксовано клік.                     |
| created_at       | timestamp   | Точний час події, що виступає ключем секціонування таблиці. |

Для реалізації архітектури CQRS у частині аналітики створено агрегаційний шар, базовим елементом якого є таблиця track\_hourly\_stats (табл. 2.12). Дані записуються та оновлюються фоновими обробниками та містять попередньо розраховані абсолютні показники активності треку в межах певної години.

Таблиця 2.12

## Структура таблиці track\_hourly\_stats

| Поле             | Тип даних | Опис                                                  |
|------------------|-----------|-------------------------------------------------------|
| track_id         | uuid      | Ідентифікатор треку.                                  |
| hour_timestamp   | timestamp | Мітка початку годинного вікна агрегації.              |
| plays            | integer   | Сумарна кількість якісних відтворень треку за годину. |
| skips            | integer   | Кількість відтворень, що не подолали поріг якості.    |
| unique_listeners | integer   | Різькість унікальних користувачів за період.          |
| likes            | integer   | Кількість вподобань за вказаний період.               |

Дані з журналів прослуховувань та взаємодій обробляються асинхронно. Залежно від типу зафіксованої дії, фонові процеси агрегують ці записи. Результати обчислень вносяться до таблиць погодинної та добової статистики треків і плейлистів.

Для нормалізації відношень типу «багато-до-багатьох» розроблено групу таблиць із композитними первинними ключами. Усі зовнішні ключі містять правило ON DELETE CASCADE. Це автоматично видаляє пов'язані записи при

видалені основних сутностей і запобігає появі помилок неіснуючих посилань. Кожна таблиця має чітко визначену доменну відповідальність:

- `playlist_tracks` пов'язує аудіотреки зі списками відтворення, містить поле `position_index` для збереження точного порядку композицій та поле `added_by_id` для ідентифікації користувача, який додав аудіотрек. Індекс `idx_playlist_tracks_position` прискорює упорядковане читання списку;
- `playlist_collaborators` визначає перелік співавторів списку відтворення;
- `playlist_followers` зберігає підписки користувачів на списки відтворення;
- `user_followers` пов'язує ідентифікатори підписника `follower_id` та цільового користувача `followed_id`. Для швидкого підрахунку взаємних підписок створено індекси за обома напрямками зв'язку.

Аналітичний шар системи містить групу таблиць, які працюють за аналогічною логікою до раніше описаної `track_hourly_stats`. Накопичують кількість відтворень, пропусків, унікальних слухачів та вподобань. Оновлення цих таблиць відбувається через регулярні проміжки часу, що знижує навантаження на систему під час перегляду аналітичних панелей. Відмінності між ними полягають у цільових сутностях, часових інтервалах та напрямках статистики:

- `track_daily_stats` функціонує за повною аналогією до годинної статистики треків. Вона відрізняється лише часовим кроком агрегації, який становить одну добу (`date_timestamp`). Дані сюди переносяться нічними процесами з годинних таблиць для довготривалого зберігання;
- `playlist_hourly_stats` зберігає аналітичні показники з кроком в одну годину, але фокусується на іншій сутності списках відтворення (`playlist_id`);
- `playlist_daily_stats` працює за аналогією до годинної статистики списків відтворення. Узагальнює дані про взаємодію з плейлистами до добових інтервалів.

Для побудови деталізованих графіків розгорнуто дві спеціалізовані таблиці добової аналітики:

- `track_daily_demographics` розбиває кількість унікальних слухачів треку за географічною ознакою. Використовує двосимвольні коди країн `client_country`, які зберігаються в журналах прослуховувань;
- `track_daily_sources` фіксує розподіл за джерелами відтворення контенту `source_type` (наприклад, відтворення із пошуку, плейлисту чи головної сторінки).

Таблиця `track_audit_logs` забезпечує ведення безперервного аудиту життєвого циклу музичних творів. Вона фіксує кожну зміну треку в межах машини станів. Запис містить унікальний ключ `id`, посилання на цільовий аудіотрек `track_id` із підтримкою каскадного видалення та ідентифікатор ініціатора зміни `actor_id`. Для відстеження історії переходів зберігаються попередній (`previous_status`) та новий (`new_status`) статуси, а також текстове обґрунтування операції у полі `reason`. Структуровані результати мультимодального аналізу контенту моделлю штучного інтелекту фіксуються у полі `ai_payload` типу `jsonb`. Час виконання операції зберігається у полі `created_at`.

Таблиця `user_moderation_logs` зберігає записи про факти застосування або скасування обмежень доступу. Кожен запис ідентифікується полем `id` та містить зовнішній ключ користувача `user_id`. Таблиця фіксує ідентифікатор адміністратора `admin_id`, тип виконаної дії `action` та детальне текстове обґрунтування прийнятого рішення у полі `reason`, точний час операції `created_at`.

Службовий рівень бази даних представлений двома таблицями, які забезпечують стабільність розгортання та виконання задач у розподіленому середовищі:

- `shedlock` координує роботу планувальника завдань при масштабуванні бекенду, зберігає назву процесу, час початку блокування та ім'я вузла, який виконує задачу;

- flyway\_schema\_history фіксує послідовність застосованих SQL-міграцій, контрольні суми файлів, час виконання та статус успішності операцій, забезпечує ідентичність структури даних у середовищі розробки та на продакшн-сервері.

Фізичну модель даних основного бізнес-домену та аналітики треків наведено на рис. В.1. Для збереження читабельності архітектурної схеми на діаграмі не відображено інфраструктурні таблиці (контроль міграцій бази даних та координація фонових завдань), партиції таблиць телеметрії, журнали аудиту й модерації, статистичні агрегації плейлистів, а також проміжні таблиці зв'язків «багато-до-багатьох» (треки у плейлистах, співавтори плейлистів, підписки користувачів). Їх детальний опис та структура наведені у тексті підрозділу.

## 2.3 Проектування та реалізація алгоритмів роботи системи

Клієнтський додаток платформи реалізовано як односторінковий додаток (SPA) з використанням бібліотеки React та суворої типізації TypeScript. Для забезпечення масштабованості кодової бази, інженерної керованості та ізоляції бізнес-логіки архітектура фронтенду побудована за методологією Feature-Sliced Design (FSD). Цей підхід гарантує суворий односпрямований потік залежностей, де верхні шари архітектури можуть використовувати модулі з нижніх, але не навпаки.

Структура клієнтського додатку поділяється на чотири основні архітектурні шари: app, pages, features та shared.

Шар app (Ядро) є точкою входу в програму. Він містить глобальні стилі, конфігурацію маршрутизації та базові провайдери контексту (зокрема TelemetryProvider), що ініціалізують роботу системи.

Шар pages (Сторінки) відповідає за компонування користувацьких інтерфейсів відповідно до визначених маршрутів. Він об'єднує функціональні блоки з нижніх шарів у готові представлення. До цього шару належать сторінки панелі керування (dashboard), профілю користувача (profile), адміністративні розділи (admin) та робочий простір артиста (artist).

Шар features (Бізнес-модулі) містить основну інкапсульовану бізнес-логіку платформи. Кожен модуль відповідає за конкретну предметну область і містить власні API-клієнти, типи, хуки та UI-компоненти. Основні модулі включають:

- media відповідає за управління аудіоконтентом, поділяється на підмодулі відтворення (playback), завантаження (upload), адміністрування та модерації треків;

- telemetry забезпечує фоновий збір даних про взаємодію користувача з платформою та стан відтворення медіа через спеціалізовані хуки (usePlaybackTelemetry, useInteractionTracker).

- recommendations та search відповідають за формування стрічок рекомендацій та обробки пошукових запитів.

- auth, playlist, profile, library: Модулі керування ідентифікацією, профілем, соціальними зв'язками та користувацькими плейлистами.

Шар shared (Спільний простір) акумулює абстракції та інструменти, що використовуються всіма іншими шарами. Тут розміщено глобальні UI-компоненти побудовані на базі примітивів Radix UI, конфігурацію HTTP-клієнта (Axios), механізми локалізації (i18n) та інструментарій керування станом додатку.

Управління станом розділено на два незалежні контури для уникнення конфліктів даних. Серверний стан, кешування запитів та синхронізація з бекендом реалізується за допомогою TanStack Query. Це дозволяє використовувати механізми оптимістичного оновлення інтерфейсу (Optimistic UI) під час виконання мутацій (наприклад, додавання треку в плейлист). Клієнтський стан, такий як робота глобального аудіоплеєра, черга відтворення та налаштування інтерфейсу бібліотеки, керується детермінованими кінцевими автоматами на базі глобальних сховищ Zustand (playerStore, libraryStore). Відтворення аудіо реалізовано через гібридний рушій, що об'єднує hls.js для роботи з мережевим потоком частин аудіофайлів та Web Audio API для

синхронізованого мікшування аудіоданих на клієнті. Організація взаємодії компонентів клієнтської частини наведена на рис. 2.6.

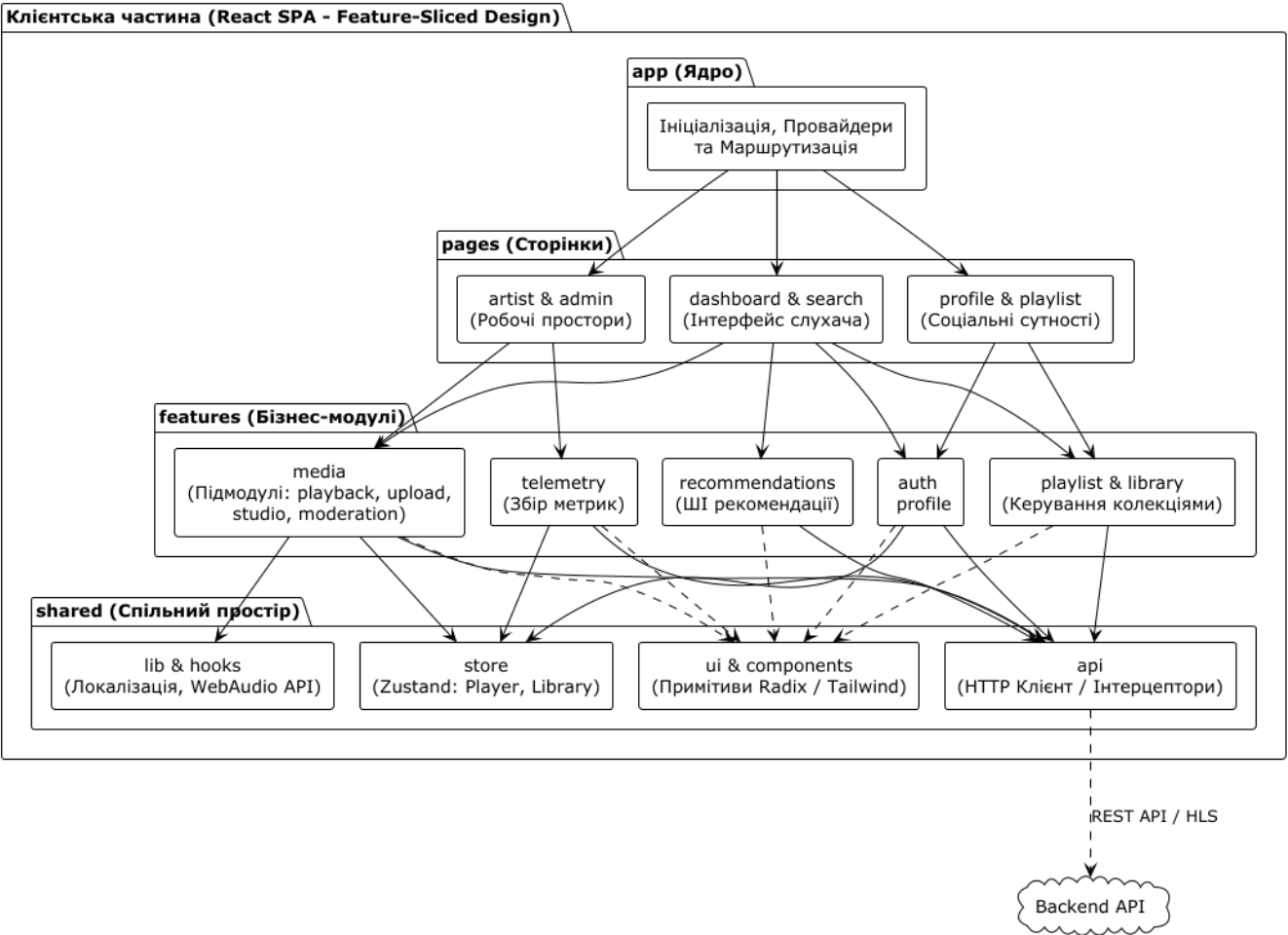


Рис. 2.6 - Діаграма компонентів клієнтської частини

Взаємодія між структурними модулями бекенд-частини платформи «IndieStream» базується на принципах модульної монолітної архітектури. Для забезпечення автономії кодової бази та суворого дотримання міжмодульних меж взаємодія між компонентами реалізована виключно через публічні інтерфейси-фасади або за допомогою асинхронної обробки доменних подій. Пряме впровадження внутрішніх сервісів одного модуля в інший на рівні контейнера Spring повністю заблоковано, а інкапсуляція бізнес-логіки забезпечується використанням пакетного (package-private) рівня доступу для внутрішніх компонентів. Організація взаємодії компонентів рівня контейнера бекенду наведена на рис. 2.7.

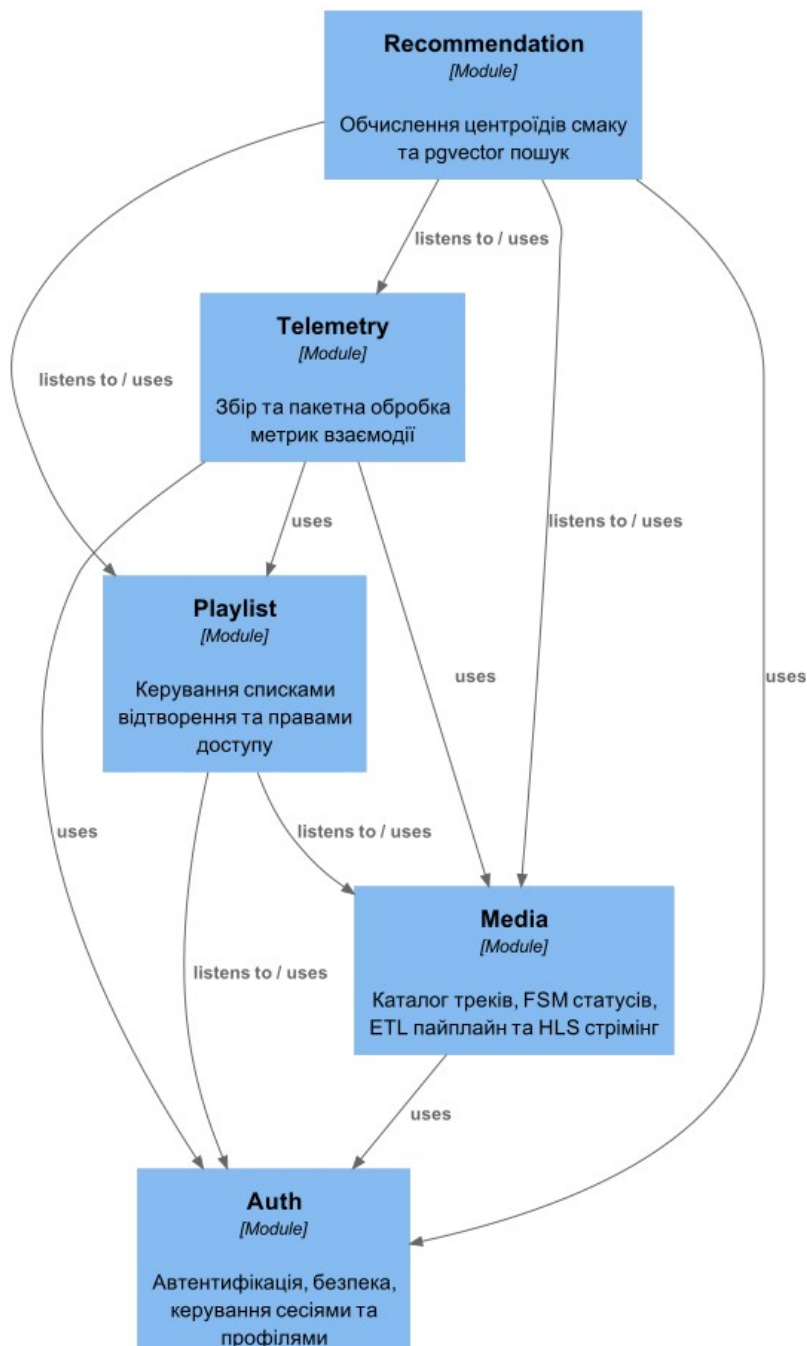


Рис. 2.7 - Діаграма компонентів бекенду

Детальна внутрішня об'єктна структура кожного з наведених бізнес-модулів, специфікація їхніх внутрішніх класів, методів, взаємозв'язків та доменних сутностей повністю спроектована й реалізована на основі відповідних доменних областей, детально описаних у підрозділі 2.1. Що усуває потребу в повторному декомпонуванні внутрішніх ізольованих компонентів у межах даного параграфа. З метою збереження високого рівня абстракції та читабельності архітектурного графу на схемі не відображено допоміжні та інфраструктурні модулі системи. До них належать:

|  |  |  |  |  |
|--|--|--|--|--|
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

- shared акумулює спільні для всіх доменів об'єкти передачі даних (DTO) та події міжмодульної агрегації лічильників;
- config містить загальносистемні конфігураційні класи для керування асинхронними пулами потоків (AsyncConfig) та перехоплення і стандартизації помилок у формат RFC 7807 (GlobalExceptionHandler);
- library реалізує патерн Backend-for-Frontend (BFF), виконує паралельну агрегацію даних через сервіси LibraryAggregationService та SearchAggregationService для оптимізації завантаження інтерфейсу користувача та усунення каскадних мережових запитів;
- seeder ізольований інфраструктурний компонент, заблокований за допомогою профільного середовища @Profile("seeder"), призначений для поетапного наповнення об'єктного сховища та бази даних синтетичними бізнес-сутностями, часовими рядами телеметрії та подальшого каскадного очищення середовища розгортання за потреби.

Потокова передача аудіоконтенту, доставка HLS-сегментів та посемплове мікшування багатоканальних аудіоактивів (Stems) забезпечується взаємодією клієнтського гібридного аудіо-рушія та серверної частини. Архітектурно прямий доступ користувачів до об'єктного сховища MinIO закрито. Безпека дистрибуції контенту підтримується механізмом перевірки прав доступу в реальному часі без додаткових звернень до реляційної бази даних.

Процес ініціюється користувачем через графічний інтерфейс, що викликає зміну стану відтворення у глобальному сховищі клієнта usePlayerStore. Після оновлення стану гібридний аудіо-рушій WebAudioEngine ініціалізує паралельні екземпляри бібліотеки hls.js. Вони відповідають за одночасне завантаження маніфесту та сегментів майстер-треку разом із окремими аудіодоріжками. Для проходження автентифікації мережовий клієнт аудіо-рушія додає поточний JWT-токен до відповідних запитів.

Сформовані запити на отримання .m3u8 маніфестів та .ts аудіосегментів проходять через вебсервер Nginx, який маршрутизує їх до захищеного ендпоінту бекенду. На стороні сервера контролер TrackController вилучає об'єкт

автентифікації безпосередньо з контексту безпеки Spring Security. Далі виконується перевірка відповідності запиту правилам видимості кінцевого автомата станів (FSM) треку для запобігання несанкціонованому доступу до ізольованих активів.

Після успішної валідації сервіс MinioStorageService звертається до внутрішнього об'єктного сховища та отримує необхідний бінарний потік за унікальним ідентифікатором файлу. Зчитані дані повертаються клієнту у вигляді ресурсу InputStreamResource. Сервер обробляє HTTP Range запити і віддає контент зі статусом HTTP 206 Partial Content, що дозволяє клієнту виконувати точне позиціонування (seeking) під час відтворення.

На фінальному етапі клієнтський рушій направляє отримані поточкові дані через вузли MediaElementAudioSourceNode у загальний граф AudioContext. Це забезпечує точну синхронізацію доріжок і дозволяє системі динамічно керувати рівнями гучності окремих фрагментів без вичерпання лімітів оперативної пам'яті браузера. На рис. 2.8 зображено діаграму послідовності процесу потокової передачі аудіоконтенту.

Процес публікації аудіотреку охоплює транзакційне збереження метаданих, медіа-файлів, перевірку їхньої цілісності, модерацію та асинхронну генерацію семантичних метаданих. Клієнтський додаток ініціює операцію після виконання локальної валідації файлів у формі завантаження треку.

Серверна частина приймає запит та розміщує оригінальні бінарні файли (майстер-трек, стемі, зображення обкладинки) в об'єктному сховищі MinIO за унікальними ідентифікаторами. У реляційній базі даних створюється запис із початковим станом обробки, публікується доменна подія завантаження TrackUploadedEvent, після чого клієнтський додаток починає циклічне опитування статусу треку.

Окремий асинхронний оркестратор перехоплює подію та виконує послідовний конвеєр обробки. Спочатку сервіс StemAntiSpoofingService перевіряє відповідність сукупності стемів майстер-треку, обчислює акустичні дельти LUFS та порівнює тривалість. Після успішної валідації здійснюється

```
sequenceDiagram
    actor User as Користувач
    participant Frontend as Frontend Layer
    participant Backend as Backend Layer
    participant MinIO as MinIO

    Note over Frontend: Player UI, Audio Engine (WebAudio + hls.js)
    Note over Backend: TrackController, MinioService

    User->>Frontend: 1 Натискання "Play"
    activate Frontend
    Frontend->>Frontend: 2 Запуск відтворення
    Frontend->>Backend: 3 GET HLS сегменти (JWT)
    activate Backend
    Backend->>Backend: 4 Валідація прав (FSM)
    Backend->>Backend: 5 Запит маніфесту / сегментів
    Backend->>MinIO: 6 S3 API Request
    MinIO-->>Backend: 7 Binary Stream
    Backend-->>Backend: 8 Media Stream Data
    Backend-->>Frontend: 9 HTTP 206 Partial Content
    deactivate Backend
    Note over Frontend: 10 Мікшування доріжок (AudioContext)
    Frontend-->>Frontend: 11 Синхронізований звук
    Frontend-->>User: 12 Відтворення аудіо
    deactivate Frontend
```

Подію схвалення треку перехоплює обробник модуля рекомендацій `TrackVectorizationWorker`. Він асинхронно звертається до `OpenAI API` для створення 768-вимірної математичної вектора на основі семантичних тегів. Згенерований вектор зберігається у `PostgreSQL` з використанням просторового індексування `HNSW` розширення `pgvector`. Публікація треку відбувається виключно за ініціативою автора: відповідний запит переводить сутність у статус `PUBLISHED`.

Наведена діаграма послідовності (рис. 2.9) відображає успішний сценарій, також оброляються і альтернативні потоки. У випадку помилок трансформації аудіо-доріжок (стемів) або невідповідності інтегрального рівня гучності (LUFS) майстер треку конвеєр переривається, а трек переходить у стан FAILED. Якщо ШІ-модуль виявляє критичні порушення політик платформи, актив автоматично отримує термінальний статус REJECTED або BANNED із наданням автору права на апеляцію. При виявленні ШІ неоднозначного контенту конвеєр надає статус IN\_REVIEW для очікування ручної перевірки адміністратором, який або схвалює, або остаточно блокує трек.

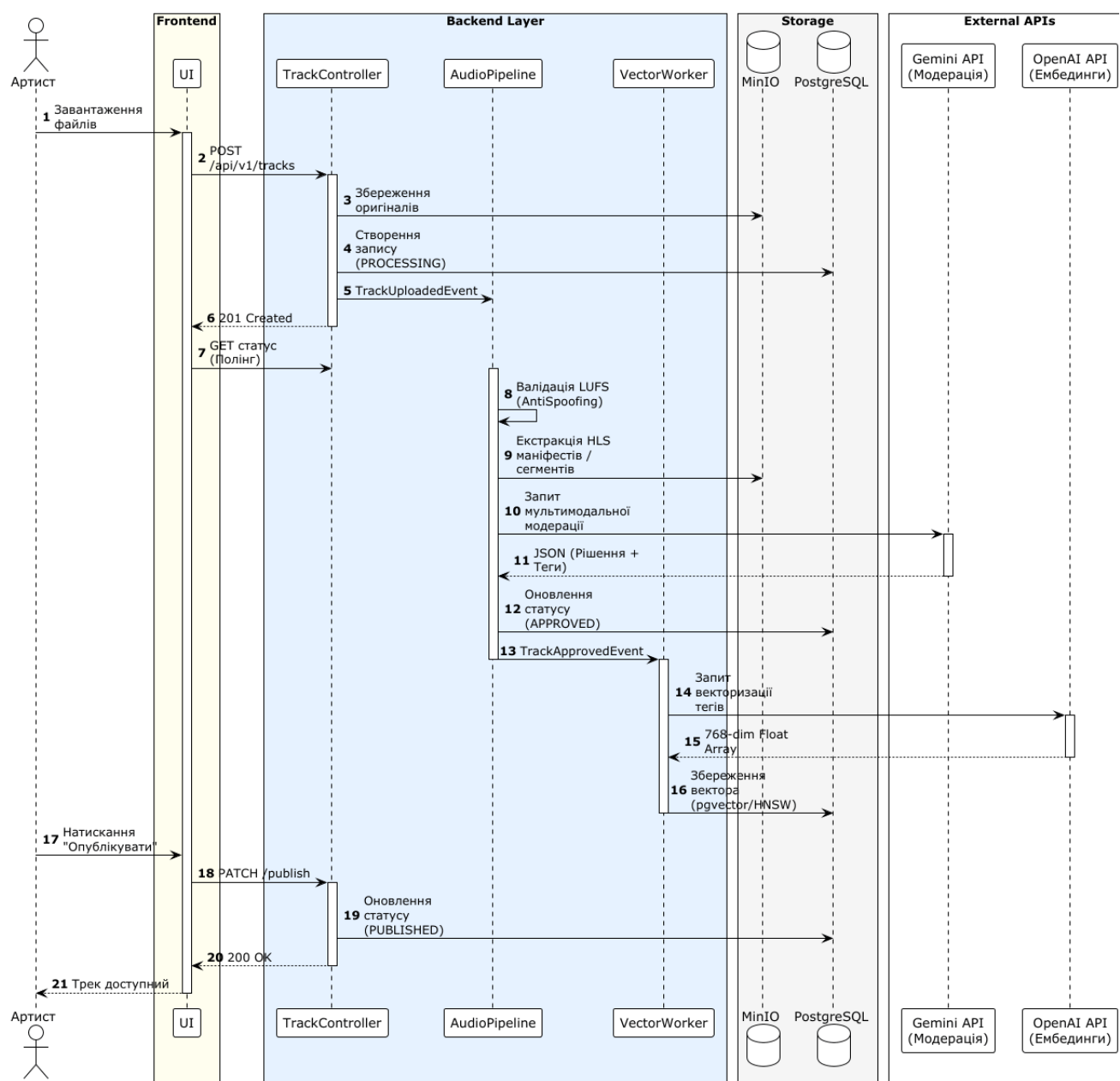


Рис. 2.9 - Діаграма послідовності завантаження та обробки аудіотреку

Процес збору телеметрії відтворень та коригування рекомендаційного вектора користувача побудований за принципом асинхронної пакетної обробки. Глобальне сховище клієнтського додатка відстежує стан відтворення, розраховує точні часові інтервали та відкидає випадкові перемикання тривалістю менш ніж дві секунди. Щоб мережеві запити телеметрії не блокували головний потік виконання і не знижували швидкодію інтерфейсу, додаток передає зібрані дані до ізольованого фоновому процесу (Web Worker). Цей процес надсилає телеметрію на сервер, який одразу записує її до швидкого сховища в оперативній пам'яті Redis Streams.

Кожні п'ять хвилин планувальник TimeWindowRollupWorker зчитує накопичені події з Redis і пакетно зберігає їх у PostgreSQL, що зменшує навантаження на реляційну базу даних. Коли система успішно зберігає новий пакет відтворень, модуль телеметрії публікує доменну подію. Сервіс модуля рекомендацій реагує на неї та запускає перерахунок вектора смаку профілю. Математична модель експоненційного ковзного середнього (ЕМА) обчислює нові координати: зсуває 768-вимірний вектор користувача в напрямку центроїдів щойно прослуханих треків. База даних оновлює цей вектор, завдяки чому наступний пошук косинусної подібності pgvector враховує актуальні смаки користувача. На рис. 2.10 зображено діаграму послідовності процесу збору телеметрії відтворень та оновлення вектора смаку користувача.

Модуль телеметрії пакетно обробляє історію взаємодій виключно для збору статистики, не публікуючи подій для системи рекомендацій. Прямі взаємодії користувача змінюють рекомендаційний вектор смаку користувача миттєво, оскільки мають меншу частоту і більшу вагу для рекомендацій. Для підтримання швидкодії клієнтський додаток застосовує оптимістичне оновлення інтерфейсу (Optimistic UI), відображаючи зміни ще до завершення мережевого запиту. На сервері транзакційне збереження цих операцій одразу публікує доменні події, які ініціюють перерахунок векторів.

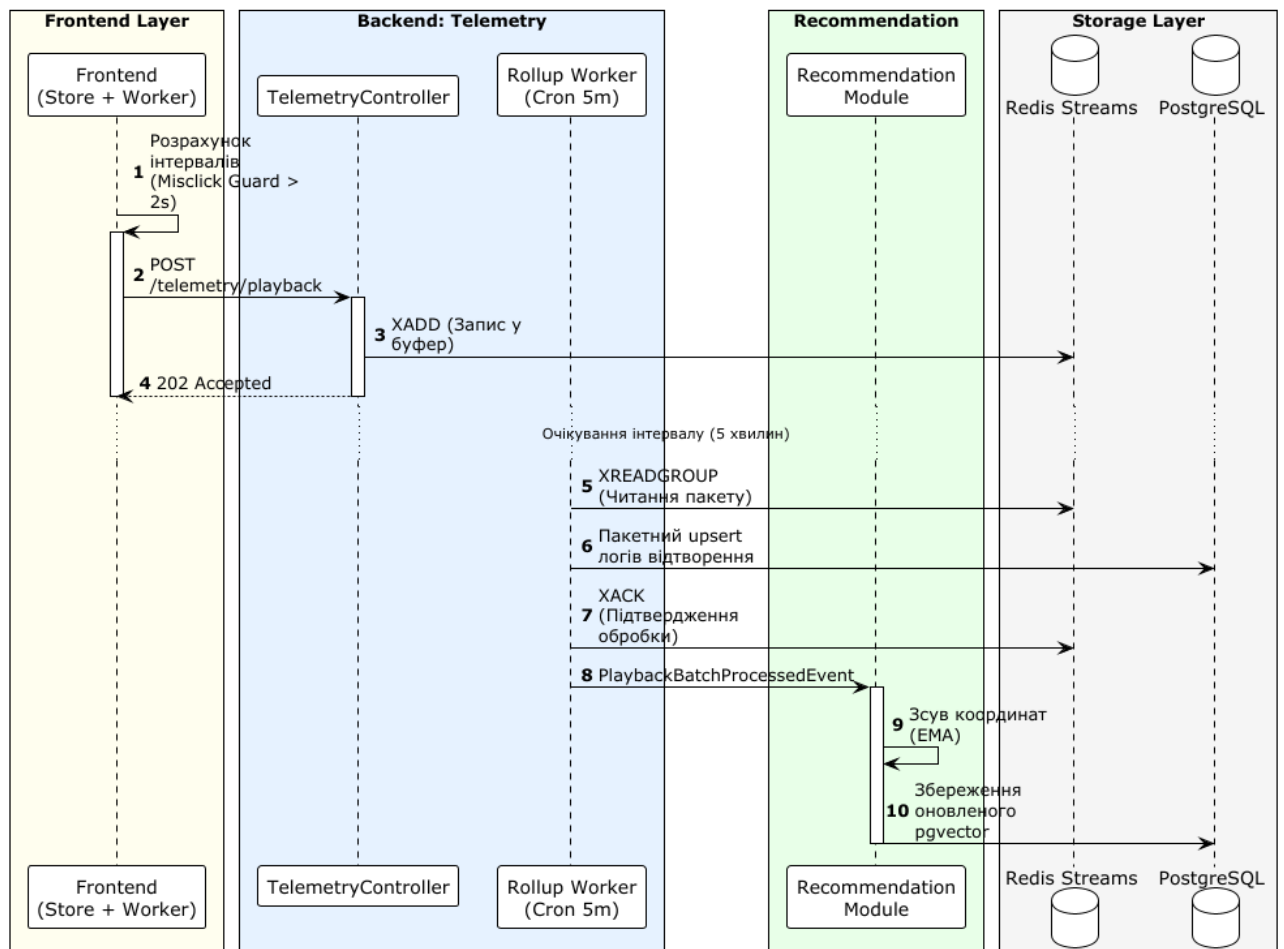


Рис. 2.10 - Діаграма послідовності процесу обробки телеметрії відтворення треку

Додавання треку до плейлиста паралельно запускає два асинхронні процеси: обчислення нового геометричного центроїда списку відтворення та зсув вектора смаку користувача. Встановлення позначки «подобається» генерує подібну подію, яка оновлює виключно профіль слухача, оскільки для системного списку улюблених треків окремий вектор не формується.

Щоб запобігти зміщенню вектора через багаторазові повторювані дії користувача (наприклад, циклічне додавання та видалення одного треку), модуль рекомендацій перевіряє унікальність кожної операції за допомогою ключів ідемпотентності в Redis із семиденним терміном дії, вектор оновлюється виключно для нових взаємодій. Аналогічно обробляється взаємодія «не цікавить»: після підтвердження унікальності запиту система відштовхує (просторово віддаляє від вектора треку) вектор слухача від небажаного треку та записує його ідентифікатор до таблиці user\_blacklists,

виключаючи трек із подальшого пошуку рекомендацій для користувача. Діаграма послідовності процесу обробки прямих взаємодій користувача зображена на рис. Г.1.

Математичною основою системи рекомендацій є просторова векторна модель, де ключові сутності (треки, плейлисти, користувачі) представлені у вигляді 768-вимірних координат. Алгоритмічне забезпечення платформи спирається на три базові математичні моделі. Система визначає релевантність між слухачем та контентом шляхом обчислення косинусної подібності (2.1). Ця метрика розраховує косинус кута між двома векторами у багатовимірному просторі, чим ближче значення до одиниці, тим вища семантична схожість об'єктів.

$$\text{similarity}(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}, \quad (2.1)$$

де  $A$  - просторовий вектор профілю користувача;

$B$  - вектор цільового треку або плейлиста;

$n = 768$  (розмірність простору).

Для динамічної адаптації профілю слухача до його поточних музичних вподобань модуль рекомендацій застосовує алгоритм експоненційного ковзного середнього ЕМА (2.2). Він забезпечує зсув координат вектору користувача:

$$V_t = \alpha \cdot T_t + (1 - \alpha) \cdot V_{t-1}, \quad (2.2)$$

де  $V_t$  - оновлений вектор користувача;

$V_{t-1}$  - його попередній стан;

$T_t$  - координати вектору треку, з яким відбулася взаємодія;

$\alpha$  - ваговий коефіцієнт, що визначає ступінь впливу дії користувача (наприклад, вподобання треку має вищий вплив, ніж повне прослуховування).

Для просторового представлення плейлиста система обчислює його геометричний центроїд (2.3). Показник розраховується як середнє арифметичне векторів усіх включених до нього аудіотреків, по цьому значенню, рушій рекомендацій може знаходити музику, релевантну загальній тематиці всього плейлиста.

$$C = \frac{1}{k} \sum_{i=1}^k T_i, \quad (2.3)$$

де  $C$  - результуючий координатний центр плейлиста;

$T_i$  - вектор  $i$ -го аудіотреку;

$k$  - загальна кількість треків у списку.

Для внесення математичної пертурбації (Jitter) та динамічного перевпорядкування елементів вибірки обчислюється модифікована позиційна оцінка (2.4). Ця функція додає контрольовану дисперсію до початкового рейтингу, зберігаючи загальну релевантність масиву.

$$S_i = (N - i) \cdot (0.8 + 0.4 \cdot r), \quad (2.4)$$

де  $S_i$  - фінальна оцінка  $i$ -го елемента після пертурбації;

$N$  - загальний розмір вибірки;

$i$  - індекс елемента у початковому впорядкованому списку ( $0 \leq i < N$ );

$r$  - псевдовипадкова величина з рівномірним розподілом у діапазоні  $[0, 1)$ .

## **2.4 Реалізація додатку інформаційної системи організації прослуховування аудіоконтенту**

Реалізація рекомендаційного ядра та підсистеми агрегації телеметрії базується на суровій ізоляції доменів та використанні просторових індексів бази даних. Центральним компонентом системи рекомендацій є сервіс RecommendationQueryService (додаток Д), який агрегує дані з різних модулів системи через публічні API-фасади, у класі реалізовано формування персоналізованих черг відтворення та підбірок контенту.

Процес генерації безперервного потоку аудіотреків ініціюється шляхом визначення цільового просторового вектора. Залежно від контексту відтворення, ним може виступати як профіль смаків користувача, так і геометричний центроїд поточного плейлиста. Для запобігання ефекту ехокамери алгоритм застосовує механізм уникнення перенасичення (Anti-Fatigue). Сервіс отримує від модуля телеметрії список ідентифікаторів треків, з якими користувач взаємодіяв протягом останніх семи днів, і виключає їх із результатів пошуку.

Пошук релевантних композицій виконується на рівні PostgreSQL за допомогою розширення pgvector, яке обчислює косинусну подібність між цільовим вектором та векторами аудіозаписів. Алгоритм реалізує стратегію надлишкової вибірки: база даних повертає 50 найближчих сусідів (пул кандидатів), які математично є найбільш точним результатом. Однак ідеальне математичне впорядкування призводить до стагнації результатів при повторних запитах. Для вирішення цієї проблеми застосовується алгоритм математичної пертурбації (2.4). Після перерахунку масив сортується за спаданням і обрізається до цільового розміру. Такий підхід зберігає загальну просторову релевантність, але гарантує динамічність контенту. Фінальним етапом є трансформація ідентифікаторів у повноцінні DTO через MediaModuleApi із суворим дотриманням правил видимості машини станів (Finite State Machine).

За відсутності розрахованого вектора смаків система виконує запит до каталогу, вибираючи найпопулярніші треки за вказаними користувачем жанрами. Щоб уникнути домінування одного жанру застосовується алгоритмічне перемішування колекції перед відправкою клієнтському додатку.

Асинхронна зміна векторів смаку реалізована в класі TasteMutationListener (додаток Ж). Архітектура компонента побудована на перехопленні доменних подій модуля плейлистів та модуля телеметрії. Кожна дія (вподобання, додавання до плейлиста, негативний зворотний зв'язок, повне прослуховування, пропуск) має власну математичну вагу зсуву.

Для захисту математичного рушія від багаторазових повторюваних дій користувача реалізовано механізм ідемпотентності на базі Redis. Перед виконанням будь-яких обчислень алгоритм намагається атомарно створити ключ, який складається з ідентифікатора користувача, ідентифікатора треку та типу дії, із часом життя (TTL) 7 днів. Операція setIfAbsent гарантує, що лише перша взаємодія ініціює обчислення, а всі наступні ідентичні дії протягом цього періоду ігноруються.

Обробка подій відбувається з ізоляцією транзакцій на рівні БД. Анотація @TransactionalEventListener ( TransactionPhase.AFTER\_COMMIT) гарантує, що

зміщення вектора ініціюється лише після успішного збереження основної дії користувача. Застосування рівня поширення `Propagation.REQUIRES_NEW` створює незалежну транзакцію для математичних обчислень, що запобігає відкату основної бізнес-логіки при вичерпанні ліміту часу на відповідь або помилок під час оновлення просторових векторів у PostgreSQL.

З метою оптимізації споживання оперативної пам'яті сервіс уникає повного завантаження структури треків. Замість вилучення об'єкта `Track`, який містить великі об'єми JSONB-метаданих, сервіс використовує `MediaVectorApi` для читання виключно ідентифікатора та 768-вимірному вектора. У випадку отримання події негативного зворотного зв'язку система додатково виконує синхронний запис у таблицю `user_blacklists`, формуючи список фільтрації для майбутніх запитів просторового пошуку.

Агрегація аналітичних даних та управління часовими рядами реалізовані у компоненті `TimeWindowRollupWorker` (додаток І), який оперує в рамках Lambda-архітектури. Його відповідальність розділена на пакетну обробку для аналітики в режимі близькому до реального часу та фонову агрегацію історичних даних.

Пакетна обробка виконується кожні п'ять хвилин. Алгоритм визначає часові вікна, яке охоплює поточну та дві попередні години, що дозволяє коректно обробляти запізнілі події з черги Redis. Завдяки розподіленому блокуванню бібліотеки `ShedLock (@SchedulerLock)` система унеможливорює стан гонитви (`Race condition`) під час горизонтального масштабування, тобто що агрегацію конкретного часового вікна виконує лише один екземпляр застосунку в кластері.

У процесі агрегації записів журналу телеметрії алгоритм відокремлює зміни смаків користувачів (повні прослуховування або пропуски) та формує події `PassiveTasteShiftsAggregatedEvent`. Логіка публікації цих подій інкапсульована в конструкцію `try-catch`, що забезпечує відмовостійкість: збої в модулі рекомендацій не переривають процес розрахунку статистичної телеметрії для адміністратора або авторів контенту.

Остаточна фіксація статистики завершених часових вікон відбувається за розкладом на п'ятій хвилині кожної години, після чого система розсилає доменні події з фінальними лічильниками. Процес стиснення даних (Daily Rollup) запускається щодобово. Він агрегує погодинні записи у денні таблиці зі зміщенням вікна на три дні назад, та перераховує оцінки популярності аудіотреків на основі зваженого алгоритмічного загасання. Таке перекриття забезпечує автоматичне відновлення цілісності агрегатів у випадку збоїв інфраструктури. Системна маршрутизація запитів гарантує, що аналітичні вибірки автоматично перемикаються між погодинними та денними таблицями залежно від запитаного діапазону.

### Висновки до другого розділу

У другому розділі виконано комплексне проєктування програмної архітектури системи організації прослуховування аудіоконтенту «IndieStream». Процес проєктування охопив етапи від формалізації концептуальних вимог до глибокої деталізації алгоритмів обробки медіаконтенту, векторних обчислень рекомендацій та взаємодії клієнтського застосунку із серверною інфраструктурою. Було формалізовано варіанти використання системи, визначено ключові ролі акторів (Слухач, Артист, Адміністратор) та описано специфікації для основних прецедентів.

Для реалізації бізнес-вимог розроблено об'єктно-орієнтовану та компонентну структуру системи. На рівні серверної частини застосовано принципи предметно-орієнтованого проєктування (DDD) у межах модульного моноліту, де міжмодульна взаємодія інкапсулюється через програмні фасади та асинхронні доменні події, а для оптимізації складних операцій читання (зокрема, у модулі аналітики) точково застосовано підхід CQRS. Для клієнтської частини розроблено архітектуру за методологією Feature-Sliced Design (FSD).

Спроектовано фізичну модель даних, де кожен інструмент виконує ізольовану функцію: СУБД PostgreSQL відповідає за збереження бізнес-сутностей та просторових векторів (через розширення pgvector), In-Memory

сховище Redis використовується для пакетної обробки й контролю ідемпотентності, а S3-сумісне сховище MinIO - для безпосереднього зберігання аудіофайлів та обкладинок.

За допомогою графічних моделей стандарту UML формалізовано динамічну поведінку системи. Діаграми послідовності деталізують взаємодії компонентів під час виконання критичних процесів: потокової передачі аудіоконтенту; завантаження та обробки аудіотреку; обробки телеметрії відтворення треку.

Окремо формалізовано математичне забезпечення платформи, зокрема застосування векторної алгебри просторового пошуку. Також наведено детальний опис програмного коду ключових алгоритмів платформи: механізмів обробки рекомендаційних запитів, транзакційної обробки доменних подій взаємодії та фоновій агрегації аналітики в рамках Lambda-архітектури.



Vercel. Це забезпечує мінімальну затримку при завантаженні статичних ресурсів програми (HTML, CSS, JavaScript) кінцевими користувачами.

Серверна інфраструктура розгорнута на віртуальному виділеному сервері (VPS) від Hetzner. Для ізоляції процесів, керування залежностями та забезпечення відтворюваності середовища використовується технологія контейнеризації Docker. Всі серверні компоненти об'єднані у внутрішню віртуальну мережу Docker Compose, що унеможливорює несанкціонований доступ до баз даних та сховищ ззовні.

Єдиною точкою входу для зовнішніх запитів до серверної частини є вебсервер Nginx, який виконує роль зворотного проксі-сервера та термінатора SSL-з'єднань. Nginx налаштовано на обробку великих тіл запитів (client\_max\_body\_size) для підтримки завантаження багатоканальних аудіофайлів (Stems) та передачу оригінальних метаданих клієнта (X-Real-IP) для коректної роботи модулів телеметрії.

У середовищі віртуального сервера розгортаються ізольовані контейнери бекенд-додатка, реляційної бази даних PostgreSQL, системи кешування Redis та об'єктного сховища MinIO. Усі зазначені компоненти функціонують у межах закритої внутрішньої віртуальної мережі Docker Compose, що повністю блокує прямий доступ до інфраструктури збереження даних із зовнішніх мереж.

Отримання медіа-контенту реалізоване шляхом маршрутизації клієнтських запитів виключно через захищені проксі-ендпоінти бекенд-додатка. Пряма взаємодія користувачів із об'єктним сховищем MinIO повністю виключена. Доступ до HLS-сегментів та медіа-файлів надається лише після успішної автентифікації запиту на рівні бекенду та перевірки прав доступу до ресурсу, що запобігає несанкціонованому отриманню бінарних файлів.. Передача аудіопотоку клієнту здійснюється за допомогою механізму обробки часткових запитів (HTTP 206 Partial Content), що оптимізує навантаження на канали зв'язку.

Бекенд-додаток взаємодіє із зовнішніми хмарними сервісами штучного інтелекту по протоколу HTTPS. Інтеграція охоплює два напрямки: виконання

|  |  |  |  |  |                    |      |
|--|--|--|--|--|--------------------|------|
|  |  |  |  |  | ІПЗ.КР.Б-121-26-ПЗ | Арк. |
|  |  |  |  |  |                    | 74   |
|  |  |  |  |  |                    |      |

запитів до Google Gemini API з метою мультимодального аналізу та модерації завантажуваного аудіоконтенту і запити до OpenAI API для генерації математичних векторів на основі сформованих семантичних тегів треку.

Автоматизація процесів збирання, тестування та розгортання програмного комплексу (CI/CD) побудовано через засоби платформи GitHub Actions. Тригером для запуску конвеєра є створення запиту на злиття (pull request) або безпосереднє внесення змін (push) у робочі гілки main та dev.

Для створення оптимізованого образу серверної частини проводиться багатоетапна збірка у середовищі Docker. На першому етапі базовий образ eclipse-temurin:21-jdk-alpine компілює вихідний код, завантажує залежності та збирає виконуваний артефакт за допомогою системи Gradle. На другому етапі готовий JAR-файл переноситься у мінімальне середовище виконання eclipse-temurin:21-jre-alpine. На цьому кроці конвеєр додатково встановлює пакет ffmpeg на рівні операційної системи, що є критично необхідним для роботи внутрішнього конвеєра обробки аудіоданих та формування HLS-сегментів.

Інтеграційне тестування серверної частини відбувається в ізольованому середовищі. Для повноцінної перевірки конвеєр розгортає допоміжні інфраструктурні контейнери: PostgreSQL із розширенням pgvector для операцій з просторовими даними, Redis із конфігурацією витіснення пам'яті allkeys-lru та сервіс об'єктного сховища MinIO. Після успішного проходження тестів система генерує та зберігає актуальні архітектурні діаграми модулів (Spring Modulith) як артефакти конвеєру. Перевірка клієнтської частини включає встановлення залежностей, статичний аналіз типів TypeScript та тестове збирання проєкту. Фактичне розгортання фронтенду не входить до цього пайплайну, оскільки воно повністю делеговане Edge Network хмарної платформи Vercel.

Конвеєр ініціює процес безперервного розгортання (CD) на робочий сервер виключно після оновлення гілки main. Зібраний Docker-образ бекенду отримує тег latest і публікується у реєстрі GitHub Container Registry (GHCR) [32]. Після цього пайплайн встановлює SSH-з'єднання з віддаленим віртуальним сервером (VPS), завантажує нову версію образу та перезапускає

контейнер серверної частини без переривання роботи суміжних сервісів. Завершальним етапом розгортання є автоматичне очищення дискового простору сервера від застарілих та невикористовуваних образів.

Для дотримання вимог щодо централізованого зберігання вихідного коду розроблено завдання автоматичної синхронізації. За умови успішного збирання обох частин проєкту, конвеєр виконує примусове дзеркалювання репозиторію до університетської системи керування версіями GitLab. Результат успішного виконання конвеєра CI/CD наведено на рис. 3.2.

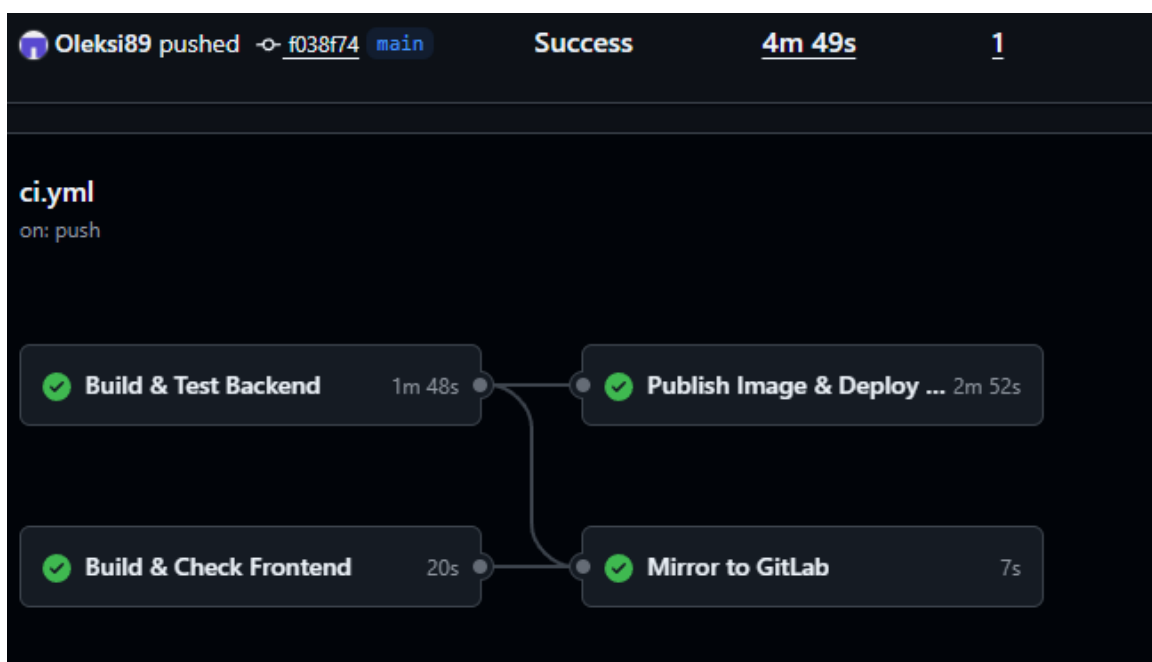


Рис. 3.2 - Візуалізація успішного виконання етапів CI/CD конвеєра

Конфігурація робочого середовища на VPS визначається декларативним файлом `docker-compose.yml`. Інфраструктурні компоненти бази даних та об'єктного сховища використовують іменовані томи (volumes) для незалежного збереження даних на диску сервера. Контейнер серверної частини конфігурується з урахуванням політики залежностей (`depends_on`), тому запуск бекенду ініціюється виключно після підтвердження працездатності (`healthcheck`) інших контейнерів (`postgres`, `minio`, `redis`), що запобігає помилкам підключення під час старту платформи. Комунікація між компонентами всередині віртуального сервера відбувається у закритій мережі Docker з використанням внутрішніх доменних імен контейнерів.

Розділення середовищ розробки, тестування та експлуатації реалізовано через механізм профілів конфігурації Spring Boot (dev, prod, seeder). Це унеможлиблює випадкове виконання інфраструктурних операцій або запис тестових даних у робочу базу. Профіль dev налаштовано на детальне логування SQL-запитів для локальної розробки, тоді як промисловий профіль prod мінімізує обсяг логів для економії ресурсів. Спеціалізований профіль seeder ізолює процеси генерації синтетичного контенту у власному просторі імен.

Базові параметри системи визначені у декларативному файлі application.yml. Конфігурація включає виділення окремої групи потоків (із префіксом media-worker-) для асинхронної обробки медіафайлів та встановлення лімітів на розмір завантажень (до 50 МБ на файл). Для забезпечення відмовостійкості інтеграції з OpenAI API застосовано патерни Circuit Breaker та Retry на базі бібліотеки Resilience4j, щоб в разі перевищення ліміту відмов тимчасово блокувались виклики зовнішнього сервісу.

Управління конфіденційними даними та інфраструктурними ключами відбувається виключно через змінні середовища. Локальне середовище та віртуальний сервер використовують файл .env, який динамічно передає дані доступу до бази даних PostgreSQL, MinIO, секрет JWT та API-ключі хмарних сервісів (Gemini, OpenAI) у контейнери через конфігурацію docker-compose.yml. Такий підхід гарантує неможливість потрапляння паролів до системи контролю версій.

Для безпеки CI/CD конвеєра GitHub Actions використовуються захищенні змінні (GitHub Secrets). Авторизаційні ключі (VPS\_HOST, VPS\_USER, VPS\_SSH\_KEY) забезпечують доступ до віддаленого сервера для безперервної доставки (CD), а GITLAB\_TOKEN зберігає ключ доступу до університетського репозиторію. Решта інфраструктурних змінних (облікові дані PostgreSQL, MinIO та JWT) використовується конвеєром виключно на етапі автоматизованого збирання для ініціалізації допоміжних сервісів під час виконання інтеграційних тестів.

### 3.2 Структура інтерфейсу інформаційної системи організації прослуховування аудіоконтенту

Робота з веборієнтованим застосунком розпочинається з екранів автентифікації та реєстрації. Інтерфейс містить екранні форми для введення облікових даних із вбудованою клієнтською валідацією, що попереджає користувача про помилки форматування чи невідповідність паролів ще до відправлення запиту. Після успішної авторизації система зберігає сесійні токени для безпечного доступу до функціоналу платформи.

Для нових користувачів (без сформованого вектора смаку, необхідного для генерації персоналізованих рекомендацій) інтерфейс передбачає автоматичне перенаправлення на сторінку калібрування. Процедура калібрування (налаштування вектора смаку) поділена на два інтерактивні кроки. Перший етап (рис. 3.3) реалізовано у вигляді візуальної сітки музичних жанрів. Інтерфейс пропонує користувачу обрати декілька категорій, які візуально виділяються при взаємодії.

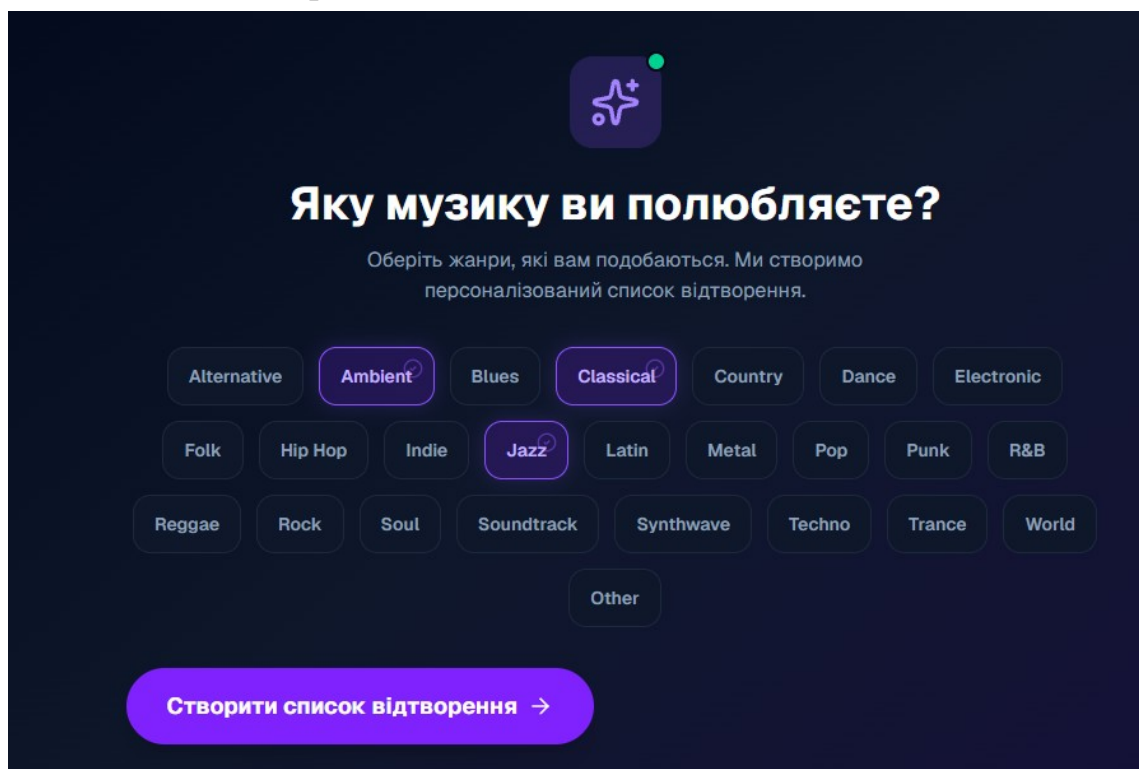


Рис. 3.3 - Екран вибору музичних жанрів

На другому етапі (рис. 3.4) відображається сформований список популярних аудіотреків з обраних жанрів.

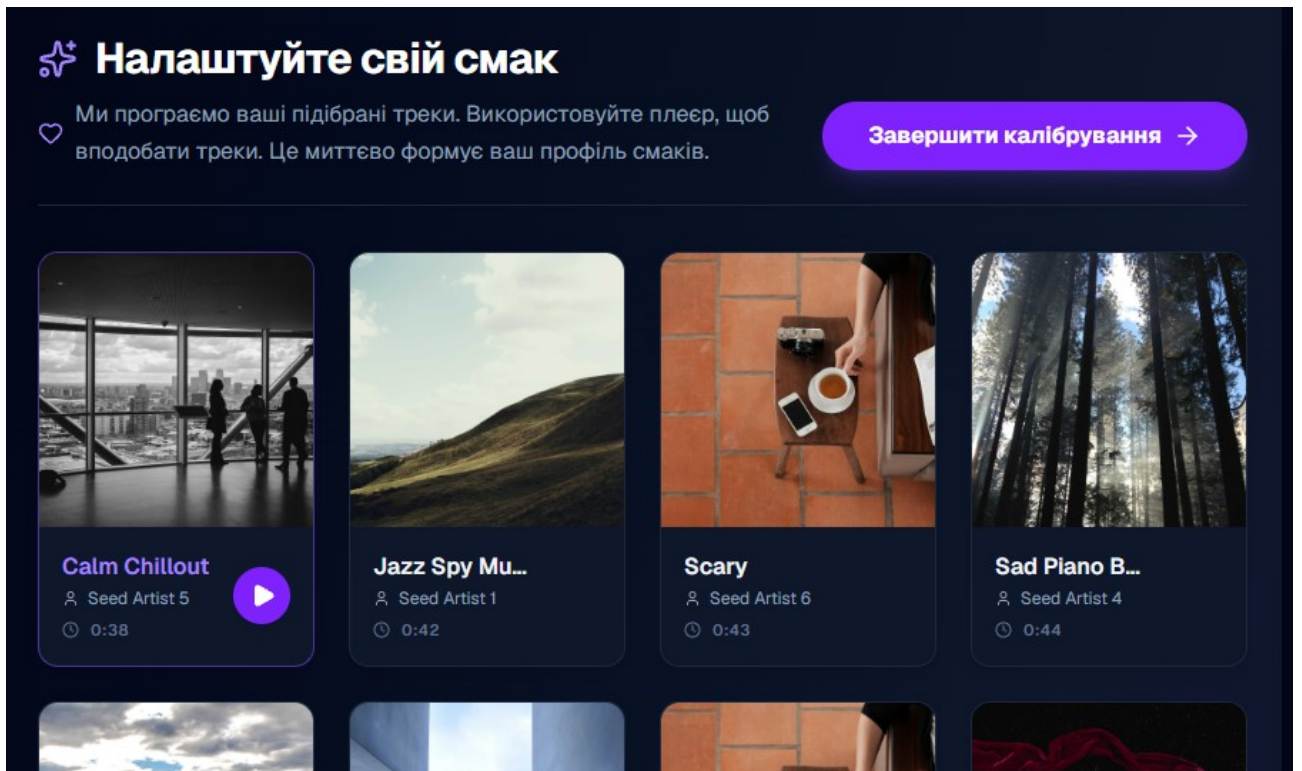


Рис. 3.4 - Сформований список композицій

Користувач має всі інструменти для початкової взаємодії з медіаконтентом: елементи управління відтворенням, кнопки вподобання та додавання треків до плейлистів. Інтерфейс обробляє ці дії за двома різними сценаріями. Натискання кнопок уподобання або збереження в плейлист миттєво генерує події для оновлення профілю смаків користувача. Натомість інформація про відтворення потокового аудіо накопичується клієнтським застосунком і відправляється після закінчення прослуховування на бекенд, вплив цих даних на вектор користувача відбувається у межах 5 хвилин.

Після ознайомлення із запропонованим контентом користувач натискає кнопку завершення калібрування. У цей момент застосунок ініціює перевірку актуального стану рекомендаційної моделі. Якщо зібраних даних достатньо для ініціалізації профілю, система знімає обмеження маршрутизації та перенаправляє слухача на головну інформаційну панель. У випадку, якщо алгоритм не отримав необхідних взаємодій, щоб ініціалізувати вектор смаку, інтерфейс повертає користувача на початковий екран вибору жанрів.

Після успішної авторизації або завершення процедури калібрування користувач переходить до головної сторінки платформи. Інтерфейс (рис. 3.5)

побудовано за принципом фіксованих зон: бічна панель бібліотеки та нижній блок аудіоплеєра залишаються статичними, тоді як центральна область забезпечує незалежне прокручування контенту. Головний екран містить блоки динамічних рекомендацій (треків і плейлистів) та сітку найновіших аудіотреків. Рекомендаційні полиці реалізовано у вигляді адаптивних горизонтальних списків із підтримкою прокручування.

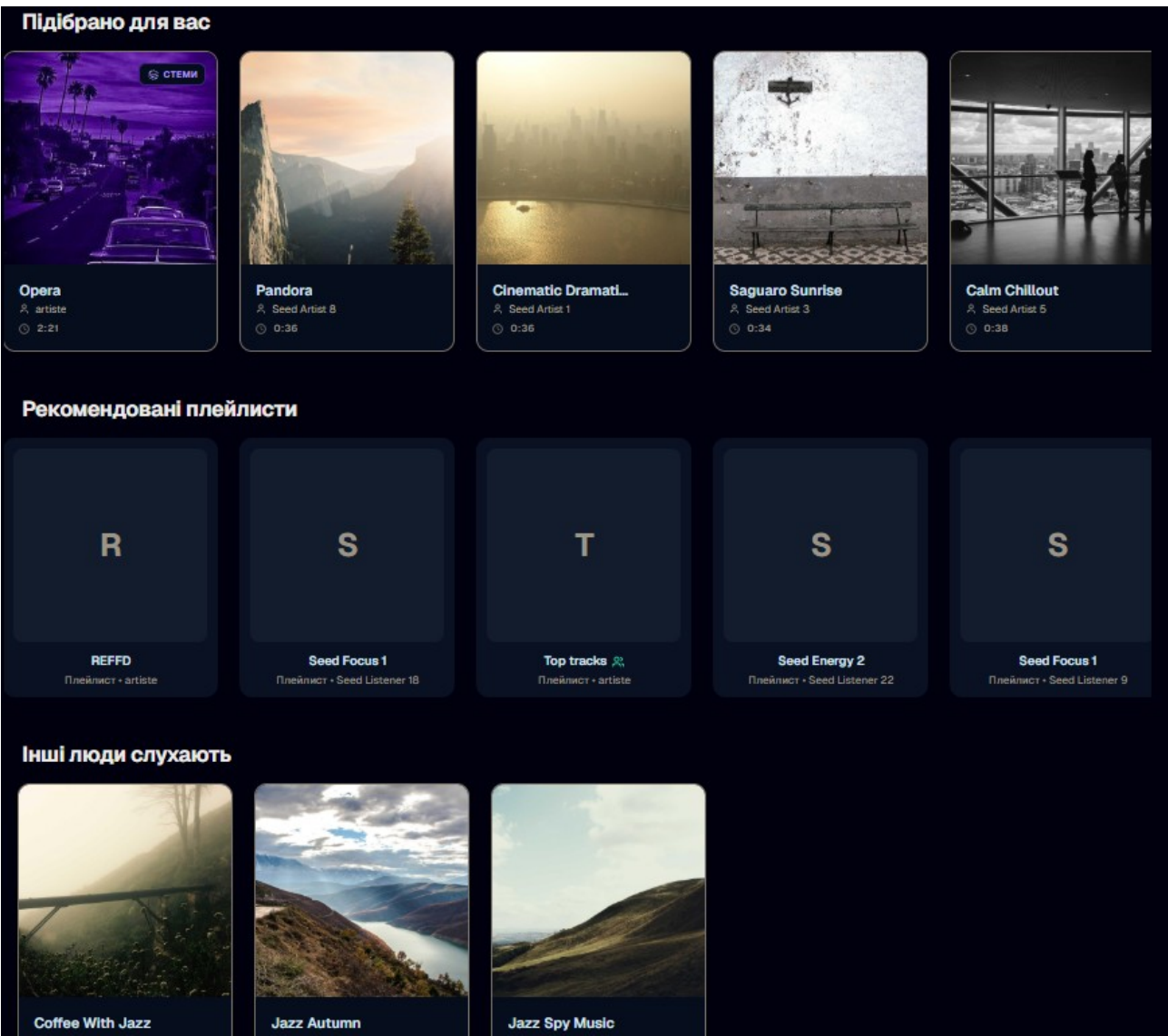


Рис. 3.5 - Головна сторінка платформи

У нижній частині екрана закріплено універсальну панель керування аудіопотоком (рис. 3.6). Її візуальна структура об'єднує блок відображення обкладинки, метаданих поточного треку, кнопки уподобання, додавання в плейлист, кнопку відтворення, елементи навігації по черзі відтворення, інтерактивний таймлайн для точного позиціонування, перемикач режиму

відтворення, повзунок регулювання гучності, а також кнопку виклику черги відтворення. При її натисканні у правій частині інтерфейсу відкривається додаткова бічна панель для управління поточним списком (попередні, активний трек, наступні композиції).

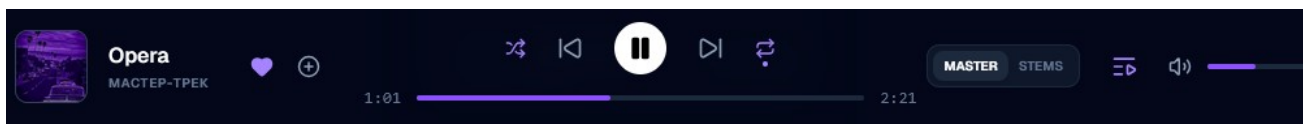


Рис. 3.6 - Інтерфейс аудіоплеєра

Інтерфейс програвача підтримує послідовне перемикання композицій, активацію циклічного відтворення (окремого треку або всієї черги) та алгоритмічне перемішування списку із можливістю повернення до початкового порядку. Окремим функціональним вузлом є інтерфейс керування відтворенням багатокompонентного аудіо (Stems), який реалізовано у вигляді інтерактивного спливаючого вікна мікшера (рис.3.7), яке містить набір лінійних регуляторів для незалежної зміни гучності звукових компонентів (наприклад, вокал, інструменти). Зміна положення повзунка користувачем ініціює миттєве коригування відповідного вузла у графі Web Audio API, завдяки цьому можна налаштовувати гучність окремого компоненту без розривів або пауз синхронного відтворення паралельних аудіопотоків.

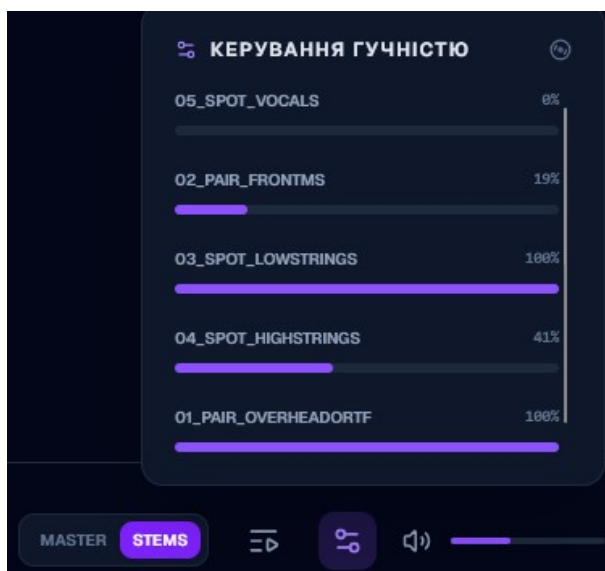


Рис. 3.7 - Вікно керування аудіодоріжками

Ініціалізація відтворення з будь-якого контейнера (наприклад, зі сторінки плейлиста) автоматично встановлює контекст програвання. Цей контекст

безпосередньо впливає на інтерфейс: якщо джерелом є плейлист, його назва з відповідним підписом фіксується у верхній частині бічної панелі черги. Для підтримки безперервного звукового потоку застосунок використовує функцію автоматичного доповнення списку (Autoplay), якщо не ввімкнено циклічне відтворення. При вичерпанні черги в залежності від встановленого контексту програвання: для плейлиста черга доповнюється треками, релевантними його загальній тематиці, а за відсутності контексту контент підбирається виключно на основі персонального вектора смаку слухача. Паралельно з відтворенням на клієнтському рівні функціонує підсистема збору телеметрії.

Основним інструментом взаємодії з контентом є контекстні меню, вміст яких динамічно адаптується залежно від типу об'єкта та прав доступу користувача. Контекстне меню треку (рис. 3.8) дозволяє керувати чергою відтворення, переходити до профілю автора, копіювати посилання на трек, та додавати трек до списків відтворення. Вибір пункту «Додати до плейлиста» розгортає додатковий список плейлистів доступних для редагування. Опція «Не цікаво» вносить трек до списку небажаних для виключення з майбутніх рекомендацій.

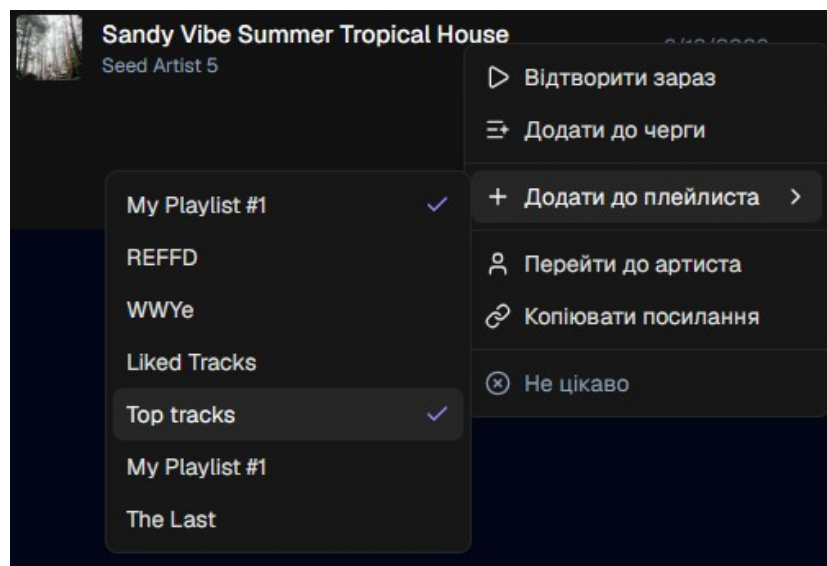


Рис. 3.8 - Контекстне меню взаємодії з треком

Пункти контекстного меню плейлиста залежать від статусу власності. Для створених користувачем плейлистів (рис. 3.9) доступний розширений набір інструментів: редагування деталей, зміна рівня приватності («Зробити

публічним»), увімкнення режиму співпраці. клонування списку та видалення з бібліотеки. Меню стороннього публічного плейлиста, збереженого до власної бібліотеки, містить базові функції: додавання всього списку треків колекції до черги відтворення, клонування, додавання всього вмісту до іншого плейлиста, та копіювання посилань для поширення, видалення з бібліотеки.

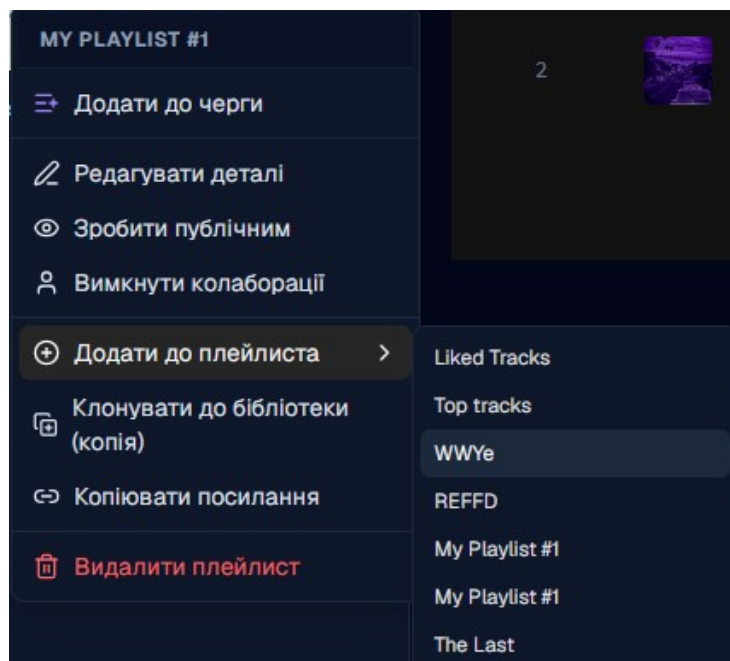


Рис. 3.9 - Контекстне меню власного плейлиста

Бічна панель бібліотеки (рис. 3.10) відображає власні плейлисти користувача (включно зі списком уподобань), колекції інших авторів, де слухач є співавтором (незалежно від їхнього статусу приватності), а також збережені публічні плейлисти та профілі підписок. Для навігації передбачено локальний текстовий пошук та систему вкладок фільтрації для відображення усіх елементів, виключно плейлистів або тільки профілів.

Створення нового плейлиста ініціюється окремою кнопкою «+», яка відкриває модальне вікно з полями для введення назви, опису та вибору рівня приватності. Сторінка самого плейлиста (див. рис. 3.10) містить назву, опис, інформаційний блок із загальною кількістю треків та сумарною тривалістю звучання. Перелік композицій організовано у вигляді таблиці з колонками: порядковий номер, назва, дата додавання та тривалість відтворення. Глобальні дії з колекцією виконуються через кнопку контекстного меню на сторінці, функціонал якого повністю дублює можливості меню плейлиста у списку

бібліотеки. Для власника колекції в інформаційному блоці додатково доступна кнопка виклику вікна статистики.

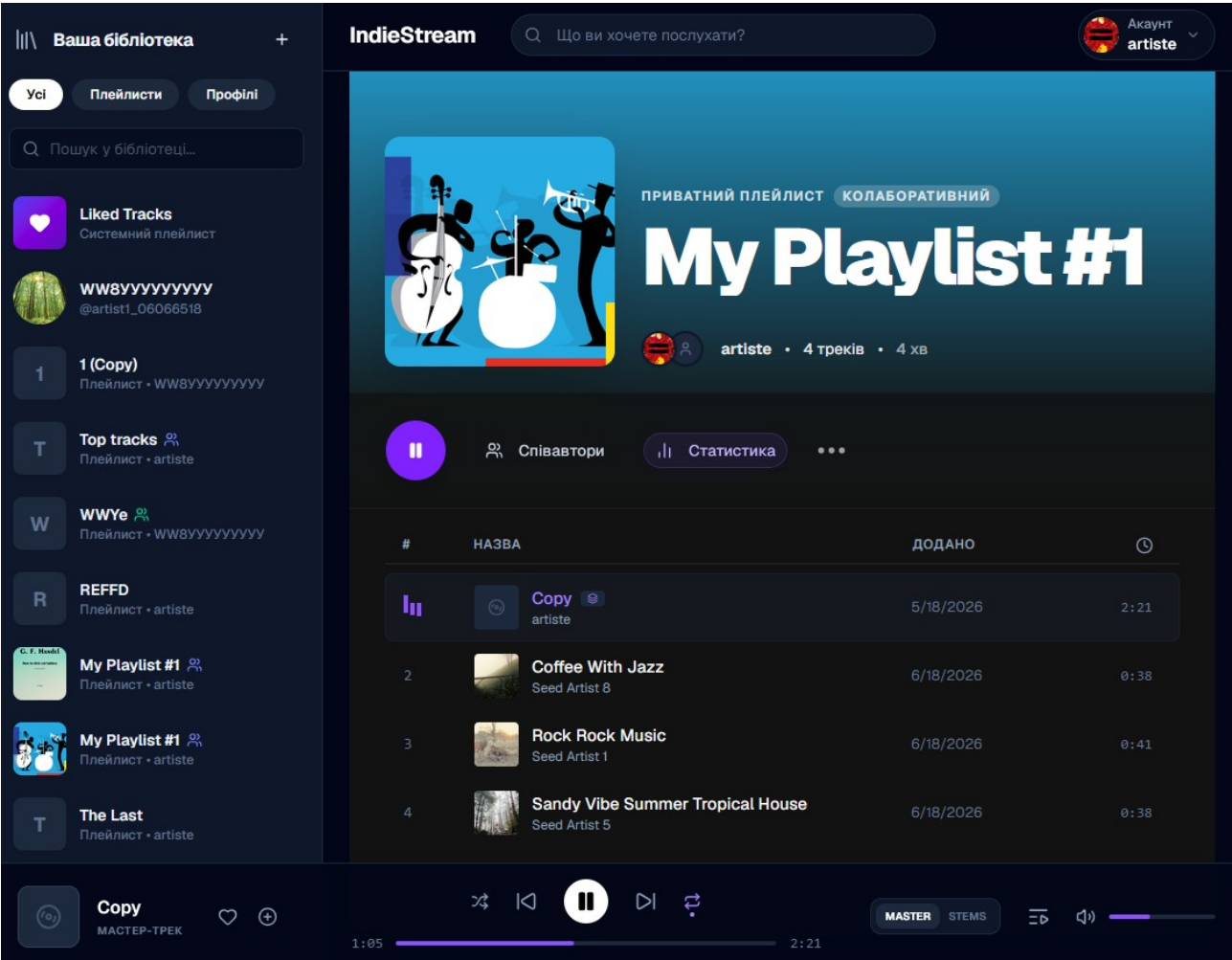


Рис. 3.10 - Інтерфейс панелі бібліотеки та сторінки плейлиста

Власник має право відкрити розширене вікно редагування деталей плейлиста для встановлення графічної обкладинки, зміни опису та керування перемикачами доступу: «Публічний плейлист» та «Колаборативний» (для дозволу іншим співавторам додавати треки

У разі активації режиму співпраці власник має доступ до панелі управління співавторами, де через пошук можна додавати нових користувачів. Інші співавтори мають можливість лише залишити колекцію. Загальний перелік співавторів відображається для всіх відвідувачів сторінки, а сам статус співавторства гарантує користувачу повний доступ до перегляду та редагування вмісту плейлиста навіть за умови його приватності.

Пошуковий інтерфейс платформи розділено на два рівні. У верхній навігаційній панелі закріплено поле швидкого пошуку, з випадаючим списком перших результатів. Цей компонент оновлюється безпосередньо під час введення тексту з алгоритмічною затримкою (debounce), забезпечуючи попередній перегляд знайдених об'єктів.

Для детального дослідження каталогу передбачена окрема сторінка розширеного пошуку. Її інтерфейс використовує систему вкладок для категоризації глобальної видачі на відповідні розділи: загальні результати, профілі користувачів, треки та плейлисти. У межах вкладки треків додатково реалізовано панель розширеної фільтрації з конкретними музичними жанрами та семантичними тегами.

Інтерфейс сторінки профілю користувача (рис. 3.11) відображає списки загальнодоступних треків, створених публічних плейлистів, а також перелік підписників та підписок. На сторінці розміщено синхронізовану кнопку підписки, стан якої миттєво оновлюється при натисканні без перезавантаження сторінки. Для забезпечення безпеки даних та захисту від несанкціонованого доступу інтерфейс профілю взаємодіє з серверними фільтрами приватності: якщо обліковий запис заблоковано або обмежено в правах, клієнтські графічні компоненти автоматично приховують вміст сторінки та відображають ізольований екран помилки або обмеженого доступу.

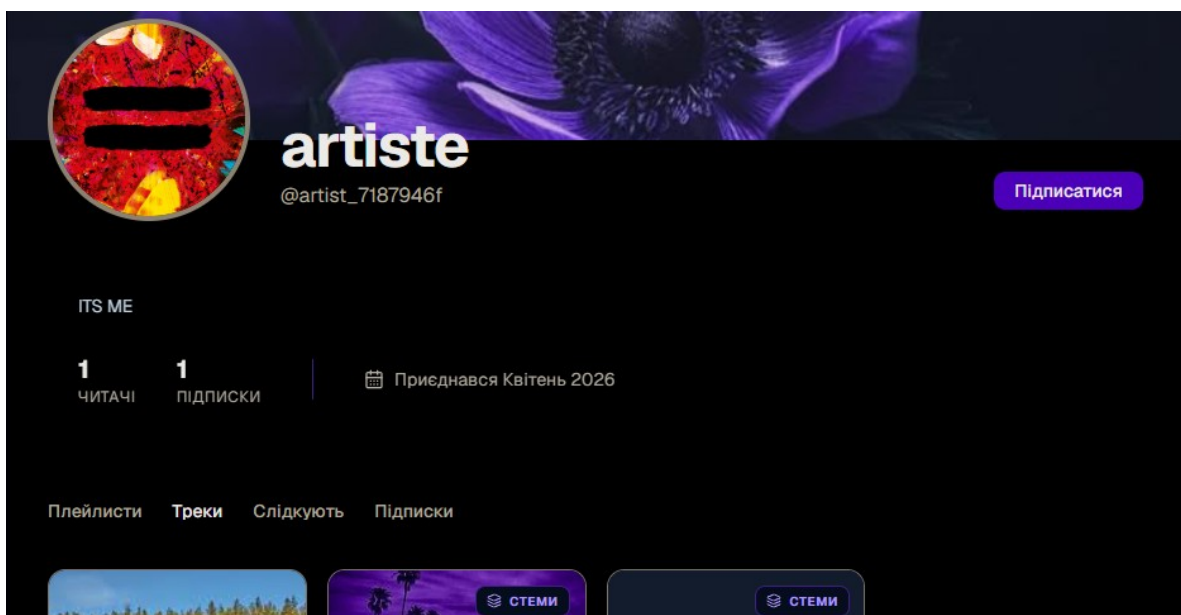


Рис. 3.11 - Інтерфейс сторінки профілю користувача

Сторінка власного профілю користувача додатково має кнопку редагування профілю (з налаштуванням псевдоніму, опису, приватності), та секцію історії прослуховувань, яка відображає список відтворених композицій за останні сім днів.

Сторінка налаштувань облікового запису призначена для відображення базових ідентифікаційних даних користувача: зареєстровану електронну адресу та унікальне системне ім'я. Інтерфейс містить селектор вибору мовної локалізації, що за замовчуванням встановлює українську мову відображення. У нижній частині панелі виділено відокремлену зону для повного скидання профілю смаків слухача. Натискання цієї кнопки відображає діалогове вікно підтвердження дії, після отримання згоди система очищає вектор смаку та список нецікавих треків, після чого користувач повертається до початкового стану «холодного старту» для повторного проходження калібрування смаку.

Панель керування автора містить графічний модуль для публікації медіаконтенту, реалізований у вигляді багатокрокового майстра завантаження з підтримкою прямого перетягування файлів (drag-and-drop) у робочу область застосунку. На першому етапі (рис. 3.13) форми користувач заповнює усі метадані аудіотреку, включаючи назву, жанр, семантичні теги та маркер вікових обмежень.

Рис. 3.13 - Інтерфейс заповнення метаданих аудіотреку

На другому етапі інтерфейс (рис. 3.14) надає елементи для завантаження графічної обкладинки та обов'язкового зведеного аудіофайлу (майстер-треку).

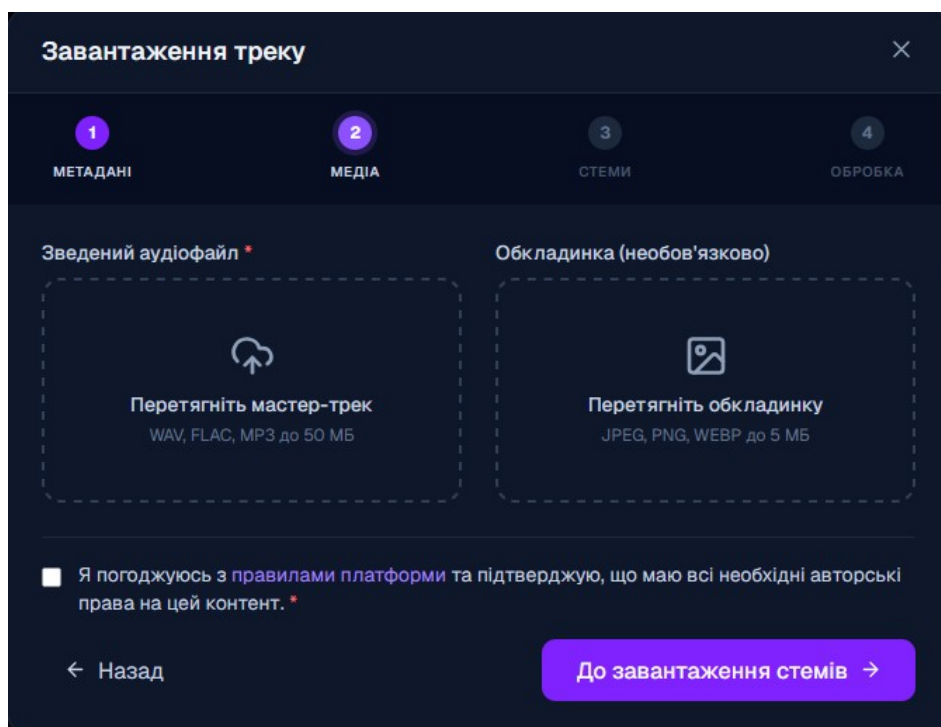


Рис. 3.14 - Інтерфейс завантаження обкладинки та зведеного аудіофайлу

Наступний етап (рис. 3.15) передбачає завантаження окремих аудіо файлів композиції. Крок є опціональним і безпосередньо визначає, чи буде для даного треку доступний режим багатодоріжкового відтворення (стем-мікшер).

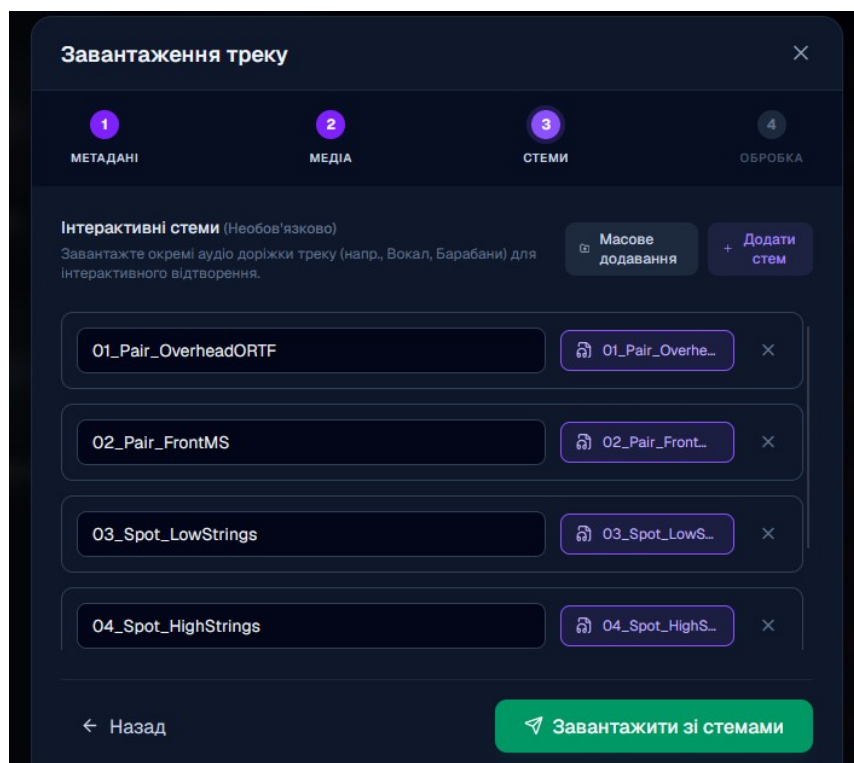


Рис. 3.15 - Інтерфейс завантаження файлів аудіодоріжок

|  |  |  |  |  |
|--|--|--|--|--|
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Після підтвердження відправки даних та файлів інтерфейс автоматично перемикається на екран (рис. 3.16) опитування стану обробки треку.

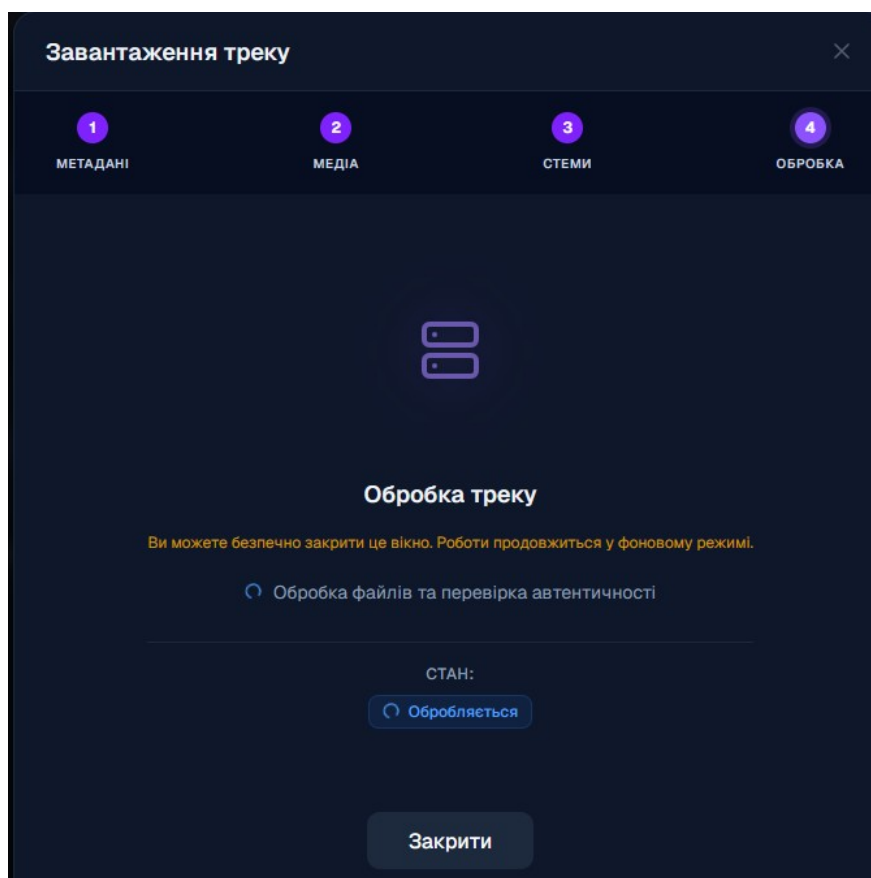


Рис. 3.16 - Інтерфейс опитування стану обробки треку

Робочий простір автора контенту (артиста) містить вкладку «Мій каталог», яка реалізована у вигляді табличного реєстру завантажених композицій. Для кожного треку в реєстрі відображається поточний стан обробки та дата завантаження. Взаємодія з елементами таблиці здійснюється через контекстне меню, набір керуючих функцій якого залежить від статусу треку. Інтерфейс надає автору інструменти для публікації, приховування, архівування композиції або подання формальної апеляції на результати модерації. Аналіз взаємодії аудиторії з контентом виконується через вкладку аналітики. Інтерфейс містить панель вибору звітного періоду, на основі якого система агрегує сумарні показники: загальну кількість прослуховувань, унікальних слухачів та збережень (уподобань) для всього каталогу артиста. Під загальними метриками розташовано перелік опублікованих композицій із

колонками деталізації: назва треку, прослуховування, слухачі, частка пропусків та алгоритмічний індекс популярності за обраний проміжок часу.

Вибір конкретної композиції з переліку ініціює перехід до вкладки розширеної статистики треку (рис. К.1). Верхній блок інтерфейсу відображає метрики реального часу («Зараз слухають» та поточний індекс популярності), основна область містить агреговані метрики за обраний період (загальні прослуховування, слухачі, частка дослухувань та пропусків). Візуалізація часових рядів реалізована через динамічний графік, який демонструє криві прослуховувань, унікальних слухачів із розбиттям по днях або годинах залежно від масштабу обраного періоду. Додатковий аналітичний блок містить діаграми, які візуалізують розподіл джерел трафіку та географію аудиторії за країнами.

Адміністративний інтерфейс платформи має сторінку глобального реєстру усіх завантажених композицій (рис. 3.17). Підтримується пошук за назвою треку, обліковим записом або псевдонімом артиста, а також сортування записів за датою та фільтрацію за поточним статусом обробки. Кожен запис у таблиці відображає ідентифікатор, назву, виконавця, статус та дату завантаження.

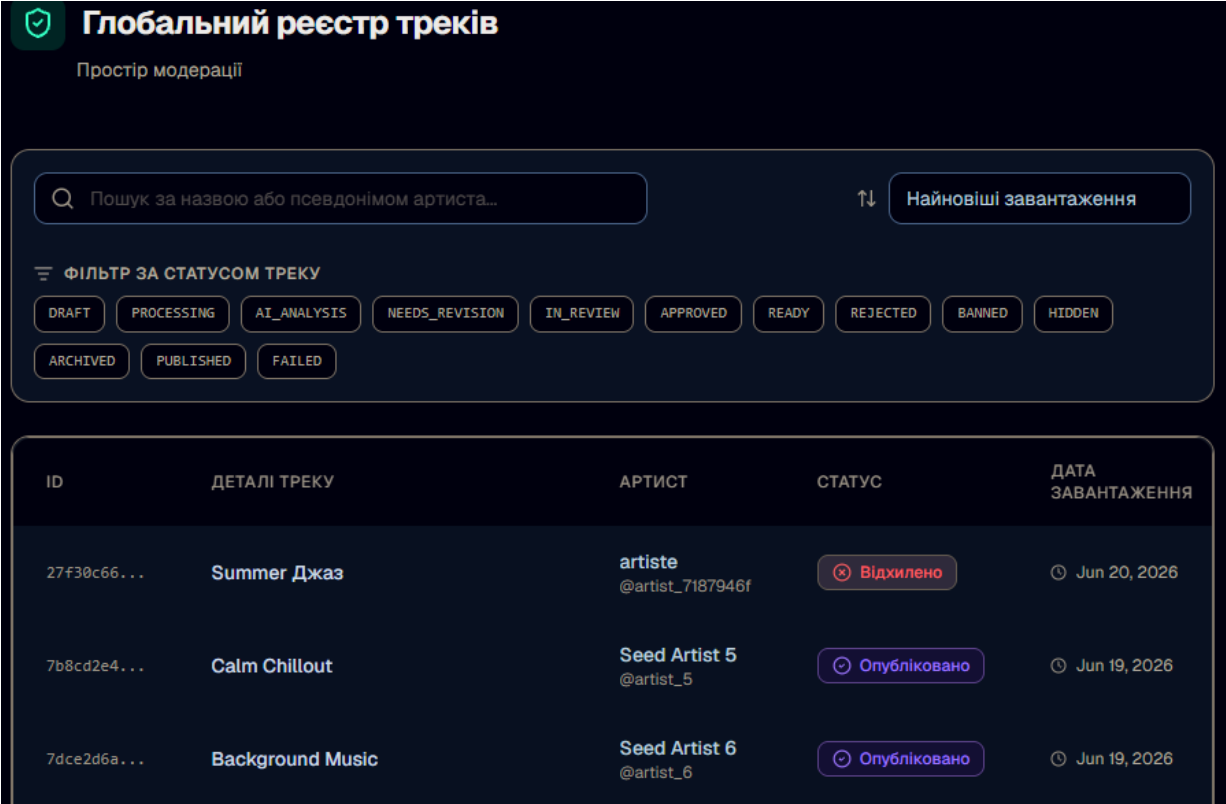


Рис. 3.17 - Глобальний реєстр треків в інтерфейсі адміністратора

Вибір конкретного треку в реєстрі відкриває сторінку керування композицією, структура якої розділена на дві вкладки. Основна вкладка модерації (рис. 3.18) забезпечує прослуховування аудіо, перегляд даних автора (логін, псевдонім) та статус розгляду апеляції. Залежно від поточного стану треку, доступні відповідні елементи керування: кнопки для схвалення, відхилення, блокування або архівування, всі дії вимагають текстового зазначення причини. Адміністратор може безпосередньо з цієї сторінки заблокувати або розблокувати обліковий запис артиста, вказавши причину такого рішення. У правій частині вкладки розміщено блок хронології, який відображає історію змін стану, дії автора та адміністратора, а також результати автоматичної перевірки ШІ.

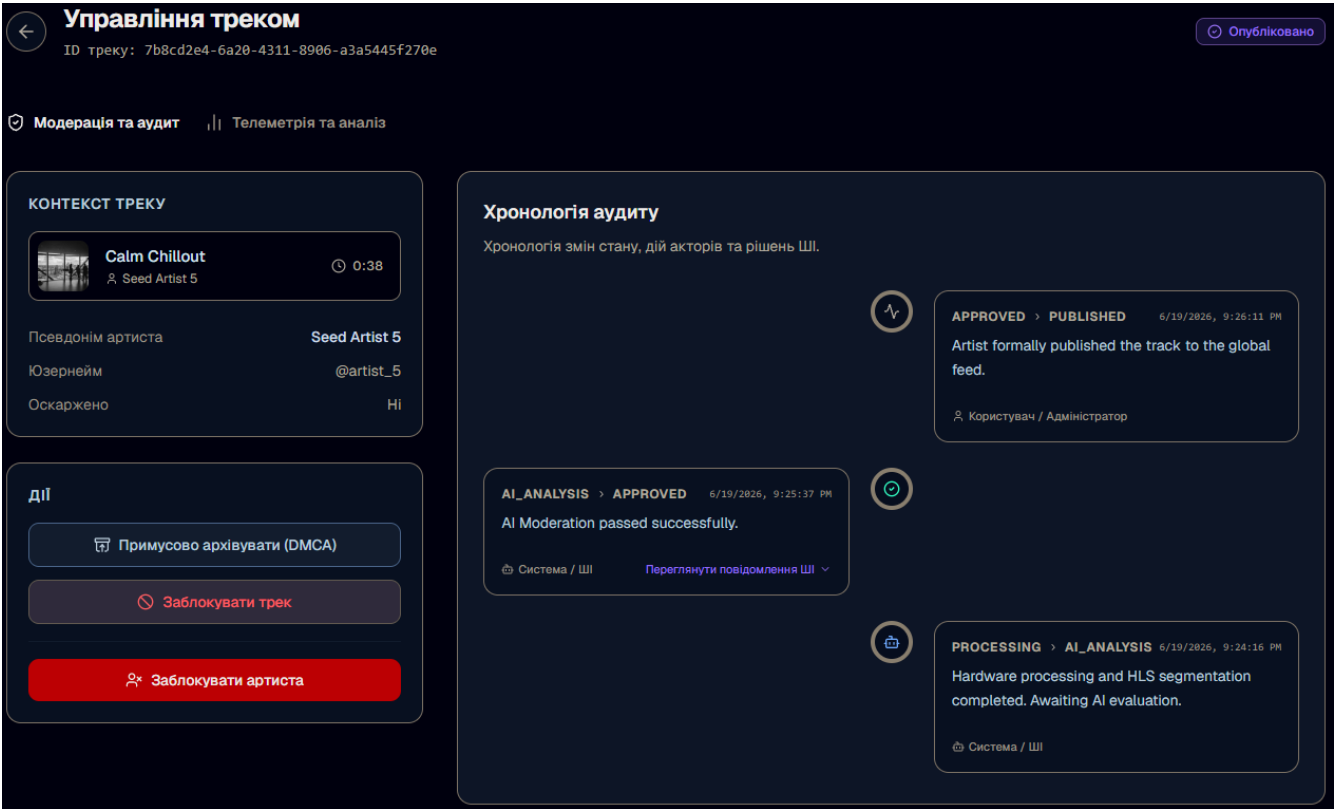


Рис. 3.18 - Вкладка модерації треку

Вкладка статистики треку для адміністратора містить ідентичний набір метрик, що й вкладка статистики для артистів (див. рис. К.1): від загальних лічильників до деталізованих графіків динаміки за обраний період. Для загального моніторингу активності платформи передбачено сторінку глобальної телеметрії. Вона має мінімалістичну структуру та відображає виключно

агреговані показники за обраний період: загальну кількість прослуховувань, унікальних слухачів, глобальну частку дослухувань та вподобань. Ця сторінка не містить графічних діаграм і призначена для швидкого огляду загальної статистики системи. Додатково тут розміщено інструменти інфраструктурного обслуговування платформи: керуючі елементи для ініціалізації примусового перерахунку погодинних та добових агрегацій телеметрії за вказаний часовий проміжок, а також функцію запуску глобальної реіндексації векторних представлень для всіх опублікованих і схвалених треків.

### 3.3. Тестування роботи програмного комплексу

Для перевірки коректності функціонування програмного комплексу реалізовано стратегію автоматизованого інтеграційного та архітектурного тестування. Тести інтегровані у життєвий цикл збірки проєкту. У разі помилки хоча б одного сценарію процес компіляції та розгортання автоматично блокується.

Тестове середовище побудовано з використанням фреймворків JUnit 5 [33], Spring Boot Test [34] та бібліотеки MockMvc [35] для симуляції HTTP-запитів. Для уникнення компромісів у вигляді in-memory баз даних застосовано технологію Testcontainers. Вона автоматично розгортає ізольовані Docker-контейнери з PostgreSQL (включно з розширенням pgvector), Redis та сховищем MinIO на час виконання тестів. Це забезпечує повну ідентичність тестового та середовища розгортання, дозволяє перевіряти реальні SQL-запити, міграції бази даних та взаємодію з об'єктним сховищем. Зовнішні асинхронні інтеграції, такі як звернення до моделей штучного інтелекту або утиліт транскодування, ізольовані за допомогою механізмів підміни об'єктів (Mockito) для гарантування швидкості та стабільності виконання.

Контроль архітектурних обмежень реалізовано за допомогою бібліотеки ArchUnit та інструментарію Spring Modulith. Відповідний набір тестів аналізує граф залежностей проєкту, блокує циклічні залежності між доменами, перевіряє дотримання інкапсуляції на рівні пакетів та гарантує дотримання правильного

напрямку викликів між контролерами, сервісами і репозиторіями згідно з принципами чистої архітектури.

Основний обсяг перевірок складає набір із 37 комплексних інтеграційних сценаріїв, які охоплюють вузли бізнес-логіки та перевірки станів системи:

У модулі авторизації та безпеки протестовано механізми перевірки JWT-токенів. Тести підтверджують відхилення запитів без валідного токена (статус 401 Unauthorized), обмеження доступу за ролями (RBAC, статус 403 Forbidden) та обробку невалідних вхідних даних із поверненням стандартизованих відповідей згідно зі специфікацією RFC 7807 (Problem Details).

Для модуля роботи з медіаконтентом реалізовано сценарії завантаження файлів формату Multipart, валідації метаданих і тегів. Окремо перевіряється логіка машини станів (FSM), яка керує життєвим циклом треку. Тести перевіряють механізми захисту від несанкціонованого доступу (IDOR): треки у статусі обробки доступні лише їхнім власникам, а опубліковані - всім користувачам. Також покрита логіка потокового передавання даних, де перевіряється здатність сервера обробляти Range-запити та повертати частини медіафайлу з HTTP-статусом 206 Partial Content.

У модулі плейлистів перевіряється ізоляція даних між користувачами. Сценарії тестування підтверджують, що приватні плейлисти недоступні для читання чи модифікації сторонніми особами. Додатково тестується коректність оновлення агрегованих даних плейлистів, таких як кількість підписників і загальна тривалість, при паралельних змінах стану об'єктів та підписах.

Тестування модуля телеметрії та аналітики спрямоване на перевірку пропускнуої здатності та захищеності системи. Сценарії підтверджують миттєве прийняття телеметричних пакетів (статус 202 Accepted) без блокування основного потоку виконання. Тестується робота алгоритму Token Bucket (Rate Limiter), який блокує запити від IP-адрес при перевищенні ліміту у 120 запитів на хвилину, повертаючи статус 429 Too Many Requests для захисту бази даних від DDoS-атак та штучного збільшення прослуховувань. Окремо перевірено

ізоляцію аналітичних дашбордів (CQRS read models) та генерацію експортних файлів телеметрії формату CSV.

Реалізоване покриття гарантує збереження цілісності бізнес-правил та інфраструктурної взаємодії модулів програмного комплексу. Структуру розроблених тестових сценаріїв за модулями наведено у табл. 3.1.

Таблиця 3.1

### Розроблені тестові сценарії

| Назва модуля                   | Тип тестів   | Кількість тестів | Напрямки перевірки                                                                                                                                               |
|--------------------------------|--------------|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Всі модулі                     | Архітектурні | 4                | Контроль інкапсуляції, відсутність циклічних залежностей, дотримання напрямку викликів (Controller => Service => Repository).                                    |
| Модуль авторизації та безпеки  | Інтеграційні | 18               | Валідація JWT-токенів, обмеження RBAC (401/403), обробка некоректних DTO за стандартом RFC 7807, пошук та оновлення профілів користувачів.                       |
| Модуль медіаконтенту           | Інтеграційні | 6                | Завантаження Multipart-файлів, контроль станів FSM, IDOR-захист ізольованих ресурсів, HTTP 206 стрімінг (Range-запити), управління життєвим циклом треків.       |
| Модуль плейлистів              | Інтеграційні | 4                | CRUD-операції, контроль доступу до приватних плейлистів, перевірка недоторканності системних списків, розрахунок агрегованих даних при підписах.                 |
| Модуль телеметрії та аналітики | Інтеграційні | 7                | Пропускна здатність Ingestion API (статус 202), робота алгоритму Token Bucket (Rate Limiter), фільтрація RBAC для аналітичних вітрин даних, генерація CSV-файлу. |
| Модуль рекомендацій            | Інтеграційні | 2                | Фіксація негативних взаємодій (Not Interested), скидання математичного вектора смаку.                                                                            |
| Разом                          |              | 41               |                                                                                                                                                                  |

Загальні результати успішного проходження всього набору тестів у локальному середовищі розробки наведено на рис. 3.19.

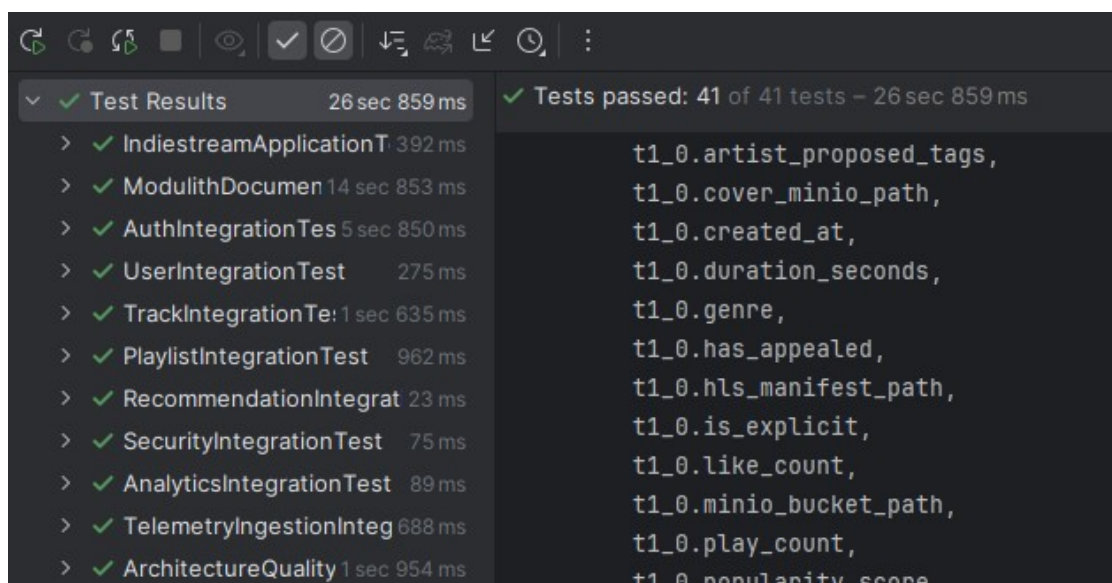


Рис. 3.19 - Успішне виконання тестів

Для визначення максимальної пропускної здатності програмного комплексу та оцінки ефективності використання обчислювальних ресурсів проведено стрес-тестування API бекенду VPS сервера за допомогою інструментарію Apache Jmeter [36]. Об'єктом тестування виступив маршрут глобальної стрічки треків (GET /api/v1/tracks?page=0&size=20). Вибір цього маршруту зумовлений необхідністю комплексної перевірки бізнес-логіки: валідації JWT-токенів, виконання багатотабличних SQL-запитів із пагінацією та серіалізації структур DTO. Такий підхід дозволив ізолювати навантаження безпосередньо на серверну частину (Spring Boot та PostgreSQL), виключивши вплив обмежень клієнтського мережевого каналу.

Тестовий сценарій імітував паралельне підключення 500 користувачів із сумарним виконанням 10 000 запитів. Результати навантажувального тестування (рис. 3.20) підтверджують стійкість архітектури. Система стабільно обробляла понад 305 запитів на секунду (Throughput), середній час обробки запиту склав 1167 мс, при цьому відсоток відхилених або неуспішних запитів (Error Rate) дорівнював нулю. Графік зміни часу відповіді (рис. 3.21) демонструє агреговані показники затримки протягом усього циклу тестування,

відображаючи лінійне та передбачуване зростання часу відповіді внаслідок утворення черги обробки під екстремальним навантаженням.

| Label ↑        | # Samples | Average | Min | Max  | Std. Dev. | Error % | Throughput | Received KB... | Sent KB/sec |
|----------------|-----------|---------|-----|------|-----------|---------|------------|----------------|-------------|
| Tracks Request | 10000     | 1167    | 64  | 3860 | 557.90    | 0.00%   | 305.1/sec  | 5169.25        | 114.10      |
| TOTAL          | 10000     | 1167    | 64  | 3860 | 557.90    | 0.00%   | 305.1/sec  | 5169.25        | 114.10      |

Рис. 3.20 - Результати стрес-тестування API

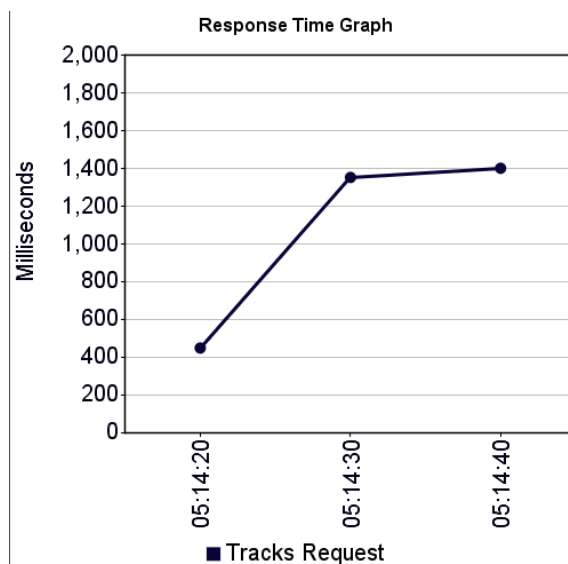


Рис. 3.21 - Графік зміни часу відповіді під час стрес-тестування API

Важливим показником ефективності є повне використання апаратних ресурсів хмарного сервера. Під час експерименту зафіксовано стовідсоткове завантаження всіх чотирьох доступних ядер процесора (400% утилізації vCPU, рис. 3.22). Цей факт доводить відсутність внутрішніх програмних блокувань (deadlocks) у пулі з'єднань з базою даних HikariCP та ефективну роботу багатопотокового середовища вебсервера Tomcat. Загалом результати тестування засвідчують високу відмовостійкість програмного комплексу та його здатність максимально використовувати доступні апаратні потужності для обслуговування інтенсивного конкурентного трафіку без переривання роботи.

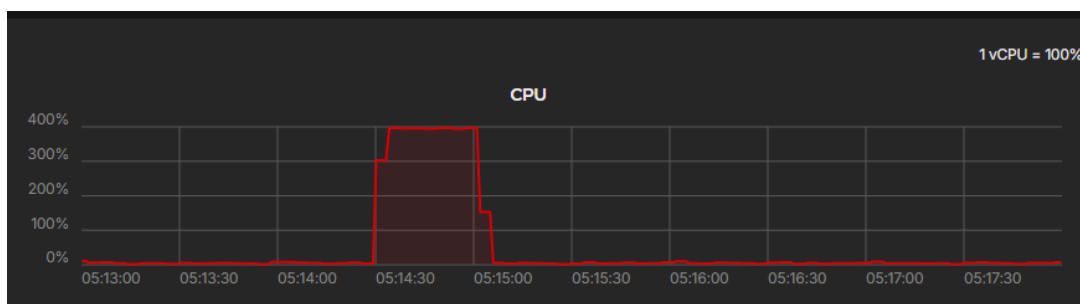


Рис. 3.22 - Використання ресурсів центрального процесора (CPU) сервера

|  |  |  |  |  |
|--|--|--|--|--|
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

## Висновки до третього розділу

У третьому розділі описано процеси розгортання, налаштування, функціонування та тестування розробленого програмного комплексу. Архітектура розгортання базується на розподіленій хмарній інфраструктурі: клієнтську частину розміщено на платформі Vercel для забезпечення оптимізованої доставки статичних файлів, тоді як серверні компоненти контейнеризовано засобами Docker та розгорнуто на віртуальному виділеному сервері (VPS) із налаштуванням зворотного проксі-сервера Nginx. Автоматизація збирання, тестування і доставки коду реалізована через CI/CD конвеєр GitHub Actions, що гарантує цілісність оновлень платформи без переривання роботи системи.

Розроблений клієнтський інтерфейс забезпечує повний цикл взаємодії з платформою з урахуванням розмежування прав доступу. Реалізовано відповідний інтерфейс для слухачів, авторів контенту (артистів) та адміністраторів. Механізми клієнтської маршрутизації та інтеграція Web Audio API дозволяють здійснювати синхронізоване відтворення кількох аудіо-доріжок без затримок. Системи контекстних меню, панелі аналітики та інструменти модерації оптимізують управління контентом та даними телеметрії.

Точність та коректність роботи бізнес-логіки підтверджена набором архітектурних та інтеграційних тестів. Застосування фреймворку Testcontainers забезпечило ідентичність середовища тестування та експлуатації для перевірки складних SQL-запитів і транзакцій. Проведене стрес-тестування зафіксувало відмовостійкість архітектури: система здатна обробляти інтенсивні паралельні запити із повною утилізацією апаратних ресурсів сервера без вичерпання пулу з'єднань чи виникнення внутрішніх блокувань. Результати тестування доводять готовність програмного комплексу до експлуатації та масштабування.

## ВИСНОВКИ

У результаті виконання випускної кваліфікаційної роботи розроблено інформаційну систему «IndieStream», яка забезпечує прослуховування багатокомпонентного аудіоконтенту (stems), збір і обробку статистики телеметрії, формування персоналізованих рекомендацій за допомогою алгоритмів просторового пошуку.

Створений програмний комплекс виконує поставлену мету, пропонує незалежним виконавцям зручну платформу для публікації аудіоконтенту у розширеному форматі та отримання статистики, а слухачам - інтерактивний інтерфейс для гнучкого відтворення аудіодоріжок та отримання персоналізованих рекомендацій.

Відповідно до визначених завдань, розроблено серверну частину за архітектурою «модульний моноліт» із дотриманням принципів предметно-орієнтованого проєктування (DDD). У системі реалізовано асинхронний конвеєр обробки медіафайлів у формат HTTP Live Streaming (HLS) із безпечним збереженням бінарних об'єктів у MinIO. Відтворення багатокомпонентних треків на клієнті забезпечується через інтеграцію Web Audio API, що дозволяє слухачу синхронізовано керувати гучністю окремих стемів без розривів медіапотoku.

Платформа застосовує виклики до ШІ моделей для автоматичної модерації, генерації семантичних метаданих і генерації векторних представлень аудіотреків. Застосування розширення pgvector дозволило динамічно обчислювати центроїд смаків користувача для формування алгоритмічної видачі, ефективно вирішуючи проблему «холодного старту». Агрегацію телеметрії побудовано на базі Lambda-архітектури з використанням Redis Streams та CQRS, що дозволяє обробляти високоінтенсивні потоки подій без деградації продуктивності бази даних. Клієнтський застосунок спроектовано за методологією Feature-Sliced Design (FSD), управління станом у якому реалізовано через Zustand і TanStack Query.

Проведене архітектурне, інтеграційне та стрес-тестування підтвердило надійність системи: комплекс стабільно обробляє інтенсивні паралельні запити з повною утилізацією ресурсів сервера та нульовим відсотком відхилених транзакцій. Практичне значення результатів полягає у створенні готової до розгортання вебплатформи і придатності модулів телеметрії і рекомендацій для ізольованого інтегрування у сторонні медіа-сервіси. Застосовані архітектурні підходи (ізоляція доменних областей, фонові агрегація телеметрії та просторове індексування) ефективно покривають основні вузькі місця високонавантажених систем.

Напрямки подальшого вдосконалення програмного комплексу включають винесення високонавантажених модулів (конвеєра обробки медіа та підсистеми аналітики) у окремі мікросервіси для масштабування. Також планується розширення переліку метрик та відслідковуваних дій телеметрії щоб покращити точність алгоритмічних рекомендацій. Окрім цього, закладений архітектурний фундамент обробки телеметрії і статистики дозволяє органічно інтегрувати систему підписок, інструменти монетизації для авторів контенту та механізми реклами.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Spotify – Web Player. URL:<https://open.spotify.com/> (дата звернення: 21.04.2026).
2. Apple Music. URL:<https://www.apple.com/apple-music/> (дата звернення: 21.04.2026).
3. SoundCloud – Listen to free music and podcasts. URL:<https://soundcloud.com> (дата звернення: 21.04.2026).
4. Bandcamp. URL:<https://bandcamp.com/> (дата звернення: 21.04.2026).
5. Stem Player Official Site. URL:<https://www.stemplayer.com/> (дата звернення: 21.04.2026).
6. Fowler M. MonolithFirst / Martin Fowler. URL:  
<https://martinfowler.com/bliki/MonolithFirst.html> (дата звернення: 21.04.2026).
7. What Is a Modular Monolith? Milan Jovanović URL:  
<https://www.milanjovanovic.tech/blog/what-is-a-modular-monolith> (дата звернення: 21.04.2026).
8. Fowler M. BoundedContext. MartinFowler.com. URL:  
<https://martinfowler.com/bliki/BoundedContext.html> (дата звернення: 21.04.2026).
9. ArchUnit User Guide. ArchUnit. URL:  
[https://www.archunit.org/userguide/html/000\\_Index.html](https://www.archunit.org/userguide/html/000_Index.html) (дата звернення: 15.06.2026).
10. Layered Architecture. O'Reilly Online Learning. URL:  
<https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html> (дата звернення: 21.04.2026).
11. Newman S. Pattern: Backends For Frontends / Sam Newman. URL:  
<https://samnewman.io/patterns/architectural/bff/> (дата звернення: 15.06.2026).

12. Lambda Architecture. Databricks Glossary. URL: <https://www.databricks.com/glossary/lambda-architecture> (дата звернення: 15.06.2026).
13. Feature-Sliced Design: Architectural methodology for frontend projects. URL: <https://feature-sliced.design/docs> (дата звернення: 15.06.2026).
14. Java 21 Documentation — JDK 21. Oracle. URL: <https://docs.oracle.com/en/java/javase/21/> (дата звернення: 23.04.2026).
15. Spring Boot Reference Documentation. Spring.io. URL: <https://docs.spring.io/spring-boot/docs/current/reference/html/> (дата звернення: 23.04.2026).
16. Spring AI Reference. Spring. URL: <https://docs.spring.io/spring-ai/reference/> (дата звернення: 15.06.2026).
17. PostgreSQL 16 Documentation. PostgreSQL. URL: <https://www.postgresql.org/docs/16/index.html> (дата звернення: 15.06.2026).
18. pgvector: Open-source vector similarity search for Postgres. GitHub. URL: <https://github.com/pgvector/pgvector> (дата звернення: 23.04.2026).
19. MinIO Object Storage Documentation. MinIO. URL: <https://min.io/docs/minio/linux/index.html> (дата звернення: 15.06.2026).
20. Redis Documentation. Redis. URL: <https://redis.io/docs/latest/> (дата звернення: 15.06.2026).
21. React 19 Documentation. React.dev. URL: <https://react.dev/blog/2024/04/25/react-19> (дата звернення: 23.04.2026).
22. TypeScript Documentation. TypeScript. URL: <https://www.typescriptlang.org/docs/> (дата звернення: 15.06.2026).
23. Zustand Documentation. Poimandres. URL: <https://docs.pmnd.rs/zustand/getting-started/introduction> (дата звернення: 15.06.2026).
24. TanStack Query Overview. TanStack. URL: <https://tanstack.com/query/latest/docs/framework/react/overview> (дата звернення: 23.04.2026).

25. Docker Documentation. Docker. URL:<https://docs.docker.com> (дата звернення: 23.04.2026).
26. GitHub Actions Documentation. GitHub Docs. URL:<https://docs.github.com/en/actions> (дата звернення: 15.06.2026).
27. Hetzner Cloud Documentation. Hetzner. URL:<https://docs.hetzner.com/cloud/> (дата звернення: 15.06.2026).
28. Vercel Documentation. Vercel. URL:<https://vercel.com/docs> (дата звернення: 15.06.2026).
29. Java Database Connectivity (JDBC) API. Oracle. URL:<https://docs.oracle.com/en/java/javase/21/docs/api/java.sql/java/sql/package-summary.html> (дата звернення: 15.06.2026).
30. Fowler M. CQRS / Martin Fowler. URL:<https://martinfowler.com/bliki/CQRS.html> (дата звернення: 15.06.2026).
31. HLS Streaming Explained: TS, M3U8, and M3U files. ECWEB. URL:[https://ecweb.ecer.com/topic/en/detail-285897-hls\\_streaming\\_explained\\_ts\\_m3u8\\_and\\_m3u\\_files.html](https://ecweb.ecer.com/topic/en/detail-285897-hls_streaming_explained_ts_m3u8_and_m3u_files.html) (дата звернення: 15.06.2026).
32. Working with the Container registry. GitHub Docs. URL:<https://docs.github.com/en/packages/working-with-a-github-packages-registry/working-with-the-container-registry> (дата звернення: 15.06.2026).
33. JUnit 5 User Guide. JUnit. URL:<https://junit.org/junit5/docs/current/user-guide/> (дата звернення: 15.06.2026).
34. Spring Boot Reference: Testing. Spring. URL:<https://docs.spring.io/spring-boot/reference/testing/index.html> (дата звернення: 15.06.2026).
35. MockMvc (Spring Framework Testing). Spring. URL:<https://docs.spring.io/spring-framework/reference/testing/spring-mvc-test-framework.html> (дата звернення: 15.06.2026).

36. Apache JMeter User's Manual. Apache. URL:  
<https://jmeter.apache.org/usermanual/index.html> (дата  
 15.06.2026). звернення:

# ДОДАТКИ

|  |  |  |  |  |                     |      |
|--|--|--|--|--|---------------------|------|
|  |  |  |  |  | ІІЗ.КР.Б-121-26-ІІЗ | Арк. |
|  |  |  |  |  |                     | 103  |
|  |  |  |  |  |                     |      |
|  |  |  |  |  |                     |      |

Структура сутностей, інтерфейсів та методів домену медіаконтенту

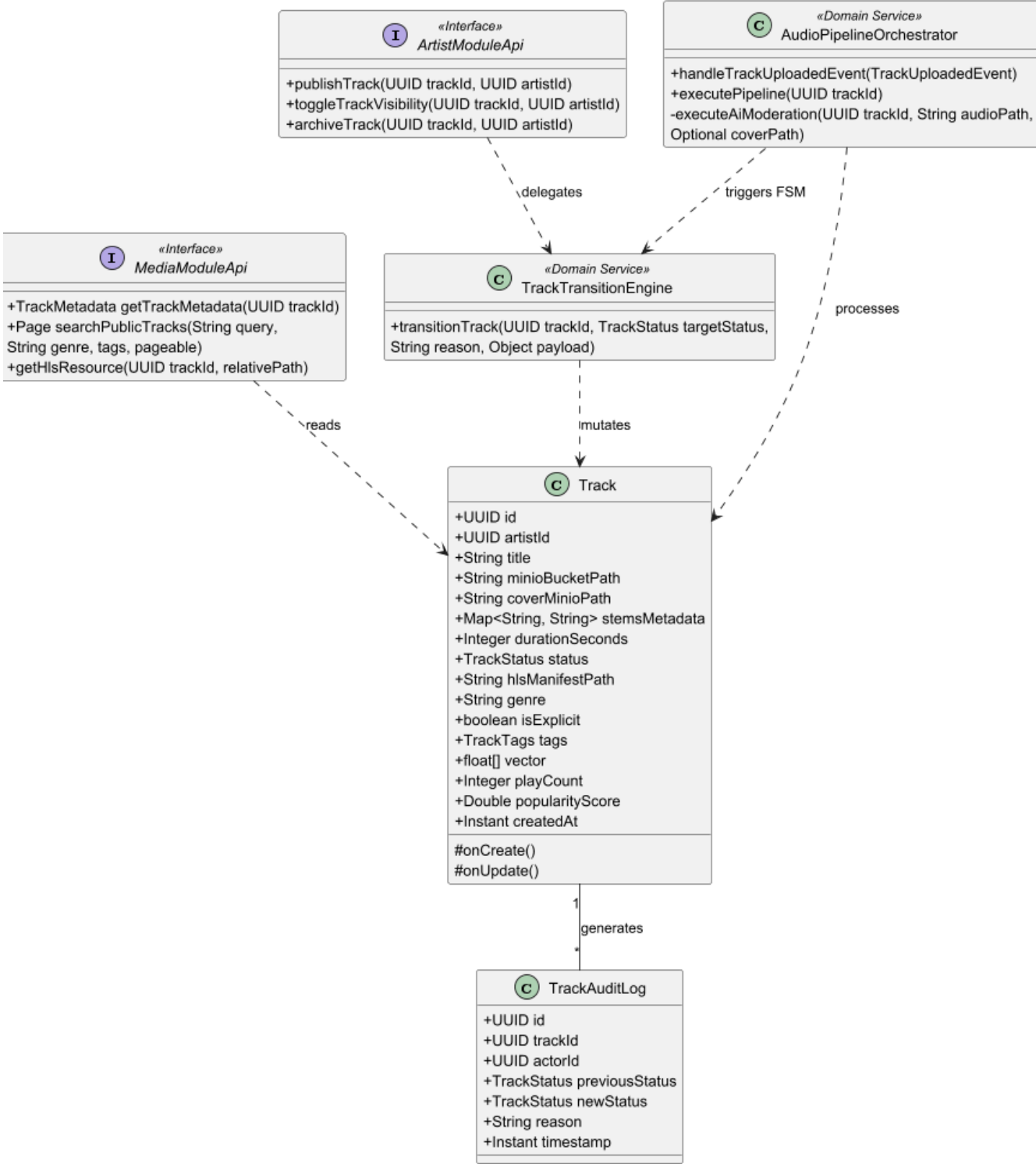


Рис. А.1 - Діаграма сутностей домену медіа

## Структура сутностей, інтерфейсів та методів домену телеметрії

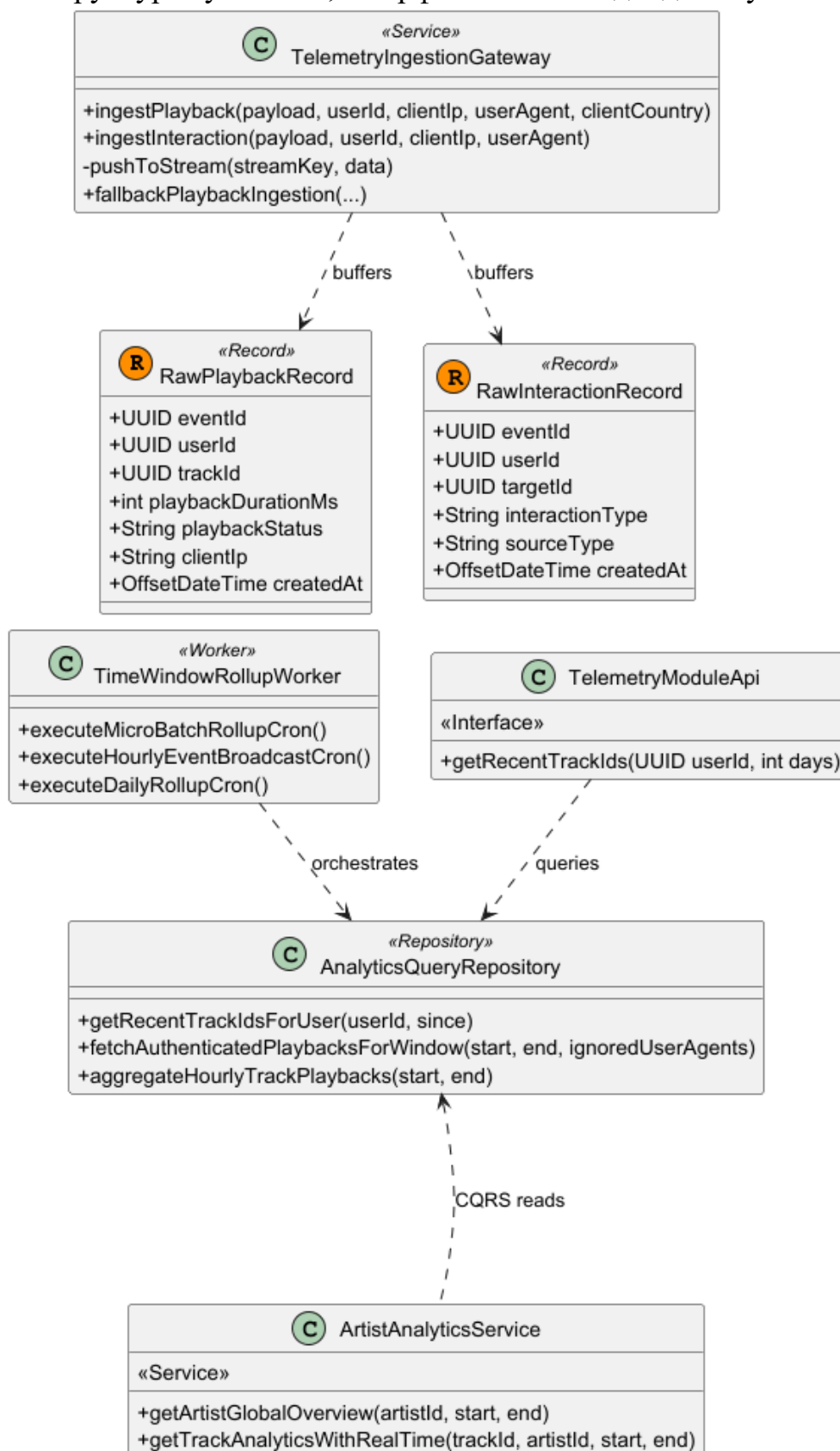


Рис. Б.1 - Діаграма сутностей домену телеметрії

Діаграма фізичної моделі даних основного бізнес-домену системи та статистики

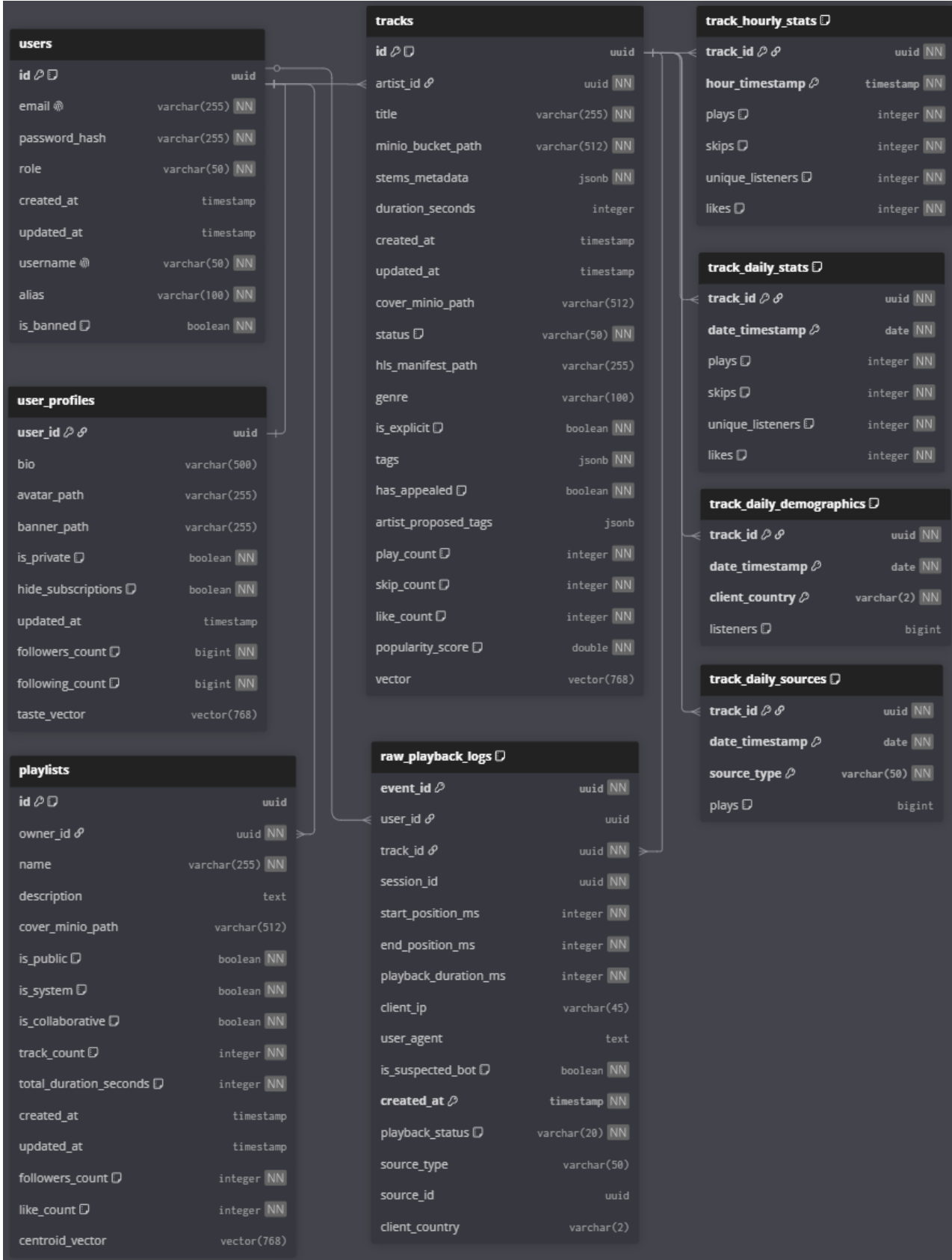


Рис. В.1 - Діаграма фізичної моделі даних основних сутностей

## Діаграма послідовності обробки прямих взаємодій користувача з треком

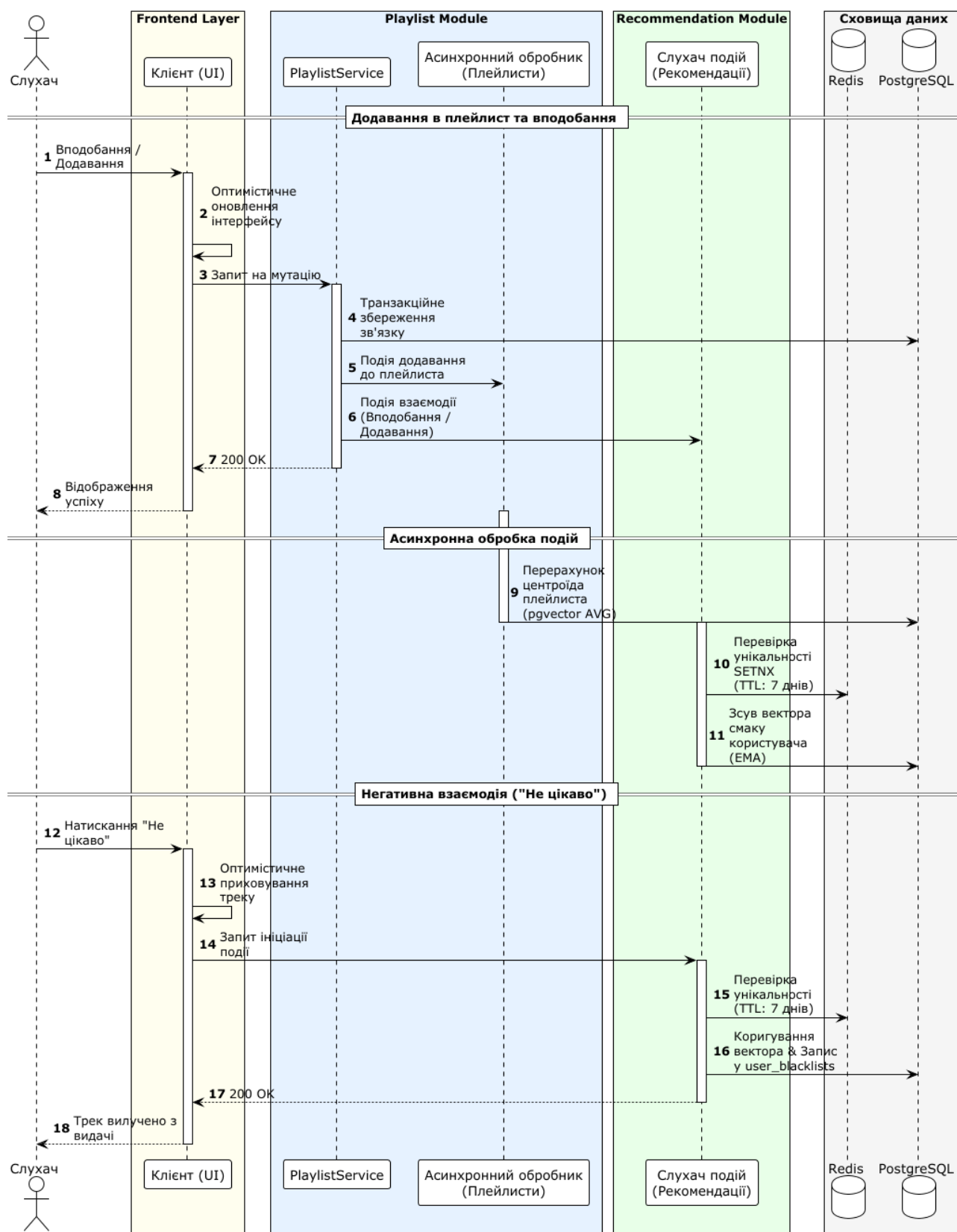


Рис. Г.1 - Діаграма послідовності обробки прямих взаємодій користувача з треком

## Лістинг коду класу RecommendationQueryService

```

package com.indiestream.recommendation.service;

import com.indiestream.auth.AuthModuleApi;
import com.indiestream.media.api.MediaModuleApi;
import com.indiestream.media.api.TrackMetadata;
import com.indiestream.playlist.PlaylistDto;
import com.indiestream.playlist.api.PlaylistModuleApi;
import com.indiestream.playlist.api.PlaylistRecommendationFacade;
import com.indiestream.recommendation.dto.AutoplayMode;
import com.indiestream.recommendation.dto.DiscoveryShelvesDto;
import com.indiestream.recommendation.repository.RecommendationQueryEngine;
import com.indiestream.telemetry.api.TelemetryModuleApi;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.data.domain.PageRequest;
import org.springframework.stereotype.Service;

import java.util.*;

/**
 * Orchestrates cross-module AI recommendations.
 * Fetches optimal UUIDs via pgvector, applies anti-fatigue and mathematical jitter,
 * and hydrates the results utilizing strict public API boundaries.
 */
@Slf4j
@Service
@RequiredArgsConstructor
public class RecommendationQueryService {

    private final RecommendationQueryEngine queryEngine;
    private final AuthModuleApi authModuleApi;
    private final MediaModuleApi mediaModuleApi;
    private final PlaylistModuleApi playlistModuleApi;
    private final TelemetryModuleApi telemetryModuleApi;
    private final PlaylistRecommendationFacade playlistRecommendationFacade;

    private static final int ANTI_FATIGUE_DAYS = 7;
    private static final int OVERSAMPLE_POOL_SIZE = 50;
    private static final int TARGET_RESULT_SIZE = 20;
    private static final int SHELF_SIZE = 10;

    /**
     * Continuous Playback Engine.
     * Generates a fresh queue of highly relevant tracks based on context.
     */
    public List<TrackMetadata> getAutoplayQueue(UUID userId, AutoplayMode mode, UUID
contextId) {
        float[] targetVector = determineTargetVector(userId, mode, contextId);

```

```

    if (targetVector == null) {
        log.warn("Target vector is null. Falling back to generic popular tracks.");
        return getColdStartFallback();
    }

    // 1. Fetch recently listened tracks to exclude (Anti-Fatigue)
    List<UUID> fatiguedIds = telemetryModuleApi.getRecentTrackIds(userId,
ANTI_FATIGUE_DAYS);

    // 2. Execute mathematical pgvector search (Oversampled)
    List<UUID> candidateIds = queryEngine.findClosestTracks(userId, targetVector, fatiguedIds,
OVERSAMPLE_POOL_SIZE);

    // 3. Apply Jitter and slice down to target size
    List<UUID> finalIds = applyMathematicalJitter(candidateIds, TARGET_RESULT_SIZE);

    // 4. Resolve UUIDs to rich DTOs preserving the modified AI ranking order
    return mediaModuleApi.getPublicTracksMetadata(finalIds);
}

/**
 * Aggregates the multi-lane dashboard interface.
 */
public DiscoveryShelvesDto getDiscoveryShelves(UUID userId) {
    float[] tasteVector = authModuleApi.getTasteVector(userId);

    if (tasteVector == null) {
        throw new IllegalStateException("User requires Onboarding. Taste Profile is NULL.");
    }

    List<UUID> fatiguedIds = telemetryModuleApi.getRecentTrackIds(userId,
ANTI_FATIGUE_DAYS);

    // Shelf 1: Made For You (Tracks)
    List<UUID> madeForYouIds = applyMathematicalJitter(
        queryEngine.findClosestTracks(userId, tasteVector, fatiguedIds,
OVERSAMPLE_POOL_SIZE),
        SHELF_SIZE
    );
    List<TrackMetadata> madeForYou =
mediaModuleApi.getPublicTracksMetadata(madeForYouIds);

    // Shelf 2: Discover Playlists
    List<UUID> playlistIds = queryEngine.findClosestPlaylists(userId, tasteVector,
SHELF_SIZE);
    List<PlaylistDto> discoverPlaylists = playlistModuleApi.getPlaylistsByIds(playlistIds);

    // Shelf 3: Listeners Like You (Collaborative Filtering)
    List<UUID> collabIds = queryEngine.findTracksFromSimilarUsers(userId, tasteVector,
SHELF_SIZE);
    List<TrackMetadata> listenersLikeYou =
mediaModuleApi.getPublicTracksMetadata(collabIds);

```

```

        return new DiscoveryShelvesDto(madeForYou, discoverPlaylists, listenersLikeYou);
    }

    /**
     * Cold Start Onboarding Resolver.
     * Searches for popular tracks matching the user's initially selected genres.
     * Relies on the Media module's native sorting (popularity_score DESC).
     */
    public List<TrackMetadata> getOnboardingTracks(List<String> genres) {
        if (genres == null || genres.isEmpty()) {
            return getColdStartFallback();
        }

        List<TrackMetadata> aggregated = new ArrayList<>();
        // Fetch top 10 from each requested genre to form a diverse initial seed
        for (String genre : genres) {
            aggregated.addAll(mediaModuleApi.searchPublicTracks(null, genre, null,
                PageRequest.of(0, 10)).getContent());
        }

        // Shuffle to prevent one genre from completely dominating the top of the list visually
        Collections.shuffle(aggregated);
        return aggregated.stream().limit(30).toList();
    }

    private float[] determineTargetVector(UUID userId, AutoplayMode mode, UUID contextId) {
        if (mode == AutoplayMode.PLAYLIST && contextId != null) {
            return playlistRecommendationFacade.getPlaylistCentroidVector(contextId).orElse(null);
        }
        return authModuleApi.getTasteVector(userId);
    }

    private List<TrackMetadata> getColdStartFallback() {
        return mediaModuleApi.searchPublicTracks(null, null, null, PageRequest.of(0,
            TARGET_RESULT_SIZE)).getContent();
    }

    /**
     * Prevents recommendation stagnation.
     * Applies a randomized mathematical perturbation (Jitter) to the perfectly ordered results.
     * Retains general proximity accuracy but ensures consecutive reloads feel "fresh".
     */
    private List<UUID> applyMathematicalJitter(List<UUID> rawCandidates, int limit) {
        if (rawCandidates == null || rawCandidates.isEmpty()) return Collections.emptyList();

        Random random = new Random();
        int maxBaseScore = rawCandidates.size();

        return rawCandidates.stream()
            .map(id -> {
                int originalIndex = rawCandidates.indexOf(id);
                // Base score: Higher is better (index 0 gets score 50)
                double baseScore = maxBaseScore - originalIndex;
            })
    }

```

```

        // Jitter modifier: Randomize between 0.8x and 1.2x (+/- 20% variance)
        double jitterModifier = 0.8 + (0.4 * random.nextDouble());
        double finalScore = baseScore * jitterModifier;
        return new AbstractMap.SimpleEntry<>(id, finalScore);
    })
    .sorted((a, b) -> Double.compare(b.getValue(), a.getValue())) // Sort DESC by final
jittered score
    .map(AbstractMap.SimpleEntry::getKey)
    .limit(limit)
    .toList();
}
}

```

## Лістинг коду класу TasteMutationListener

```

package com.indiestream.recommendation.service;

import com.indiestream.media.api.MediaVectorApi;
import com.indiestream.playlist.event.TrackAddedToPlaylistEvent;
import com.indiestream.playlist.event.TrackLikedEvent;
import com.indiestream.recommendation.domain.UserBlacklist;
import com.indiestream.recommendation.domain.UserBlacklistId;
import com.indiestream.recommendation.event.TrackNotInterestedEvent;
import com.indiestream.recommendation.repository.UserBlacklistRepository;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.data.redis.core.StringRedisTemplate;
import org.springframework.scheduling.annotation.Async;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.transaction.event.TransactionPhase;
import org.springframework.transaction.event.TransactionalEventListener;

import java.time.Duration;
import java.time.Instant;
import java.util.UUID;

/**
 * Asynchronous orchestrator handling explicit user actions.
 * Protects the VectorMathService from malicious "Toggle Spam" using a Redis Idempotency
 * Guard.
 */
@Slf4j
@Component
@RequiredArgsConstructor
public class TasteMutationListener {

    private final VectorMathService vectorMathService;
    private final MediaVectorApi mediaVectorApi;
    private final UserBlacklistRepository userBlacklistRepository;
    private final StringRedisTemplate redisTemplate;

    private static final Duration IDEMPOTENCY_TTL = Duration.ofDays(7);

    @Async
    @TransactionalEventListener(phase = TransactionPhase.AFTER_COMMIT)
    @Transactional(propagation = Propagation.REQUIRES_NEW)
    public void onTrackLiked(TrackLikedEvent event) {
        if (acquireIdempotencyLock(event.userId(), event.trackId(), "LIKE")) {
            processShift(event.userId(), event.trackId(), VectorMathService.ALPHA_LIKE);
        }
    }
}

```

```

@Async
@TransactionalEventListener(phase = TransactionPhase.AFTER_COMMIT)
@Transactional(propagation = Propagation.REQUIRES_NEW)
public void onTrackAddedToPlaylist(TrackAddedToPlaylistEvent event) {
    if (acquireIdempotencyLock(event.userId(), event.trackId(), "PLAYLIST_ADD")) {
        processShift(event.userId(), event.trackId(), VectorMathService.ALPHA_PLAYLIST_ADD);
    }
}

@Async
@TransactionalEventListener(phase = TransactionPhase.AFTER_COMMIT, fallbackExecution =
true)
@Transactional(propagation = Propagation.REQUIRES_NEW)
public void onTrackNotInterested(TrackNotInterestedEvent event) {
    if (acquireIdempotencyLock(event.userId(), event.trackId(), "NOT_INTERESTED")) {
        processShift(event.userId(), event.trackId(),
VectorMathService.ALPHA_NOT_INTERESTED);

        // Persist the hard blocklist
        userBlacklistRepository.save(
            new UserBlacklist(
                new UserBlacklistId(event.userId(), event.trackId()),
                "User explicitly marked as not interested",
                Instant.now()
            )
        );
        log.info("Track {} added to blocklist for User {}", event.trackId(), event.userId());
    }
}

/**
 * Executes the mathematical shift if the track exists and has a calculated vector.
 */
private void processShift(UUID userId, UUID trackId, float alpha) {
    // Fetch strictly the needed fields (id and vector) to prevent full entity hydration
    mediaVectorApi.getTrackVector(trackId).ifPresentOrElse(
        vector -> vectorMathService.shiftUserTasteVector(userId, vector, alpha),
        () -> log.warn("Cannot shift taste: Track {} has no vector space or does not exist.",
trackId)
    );
}

/**
 * Redis Idempotency Guard.
 * Prevents a user from spamming Like/Unlike to maliciously push their vector towards a specific
track.
 *
 * @return true if the action is novel and should proceed, false if it's a blocked duplicate.
 */
private boolean acquireIdempotencyLock(UUID userId, UUID trackId, String action) {
    String key = String.format("recommendation:taste_shift:%s:%s:%s", userId, trackId, action);
    Boolean isNovel = redisTemplate.opsForValue().setIfAbsent(key, "LOCKED",
IDEMPOTENCY_TTL);

```

```
if (Boolean.FALSE.equals(isNovel)) {  
    log.debug("Idempotency guard triggered: Blocked duplicate taste shift for key {}", key);  
    return false;  
}  
return true;  
}
```

## Лістинг коду класу TimeWindowRollupWorker

```

package com.indiestream.telemetry.worker;

import com.indiestream.shared.event.PlaylistCountersAggregatedEvent;
import com.indiestream.shared.event.TrackCountersAggregatedEvent;
import com.indiestream.telemetry.api.event.PassiveTasteShiftsAggregatedEvent;
import com.indiestream.telemetry.repository.RawPlaybackRecord;
import com.indiestream.telemetry.repository.TelemetryRollupRepository;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import net.javacrumbs.shedlock.spring.annotation.SchedulerLock;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.ApplicationEventPublisher;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Service;

import java.time.OffsetDateTime;
import java.time.ZoneOffset;
import java.time.temporal.ChronoUnit;
import java.util.List;
import java.util.Map;

/**
 * Distributed orchestrator for time-series aggregation.
 * Implements Near Real-Time (NRT) Micro-Batching for dashboards
 * and exact-once Modulith event broadcasting for closed windows.
 */
@Slf4j
@Service
@RequiredArgsConstructor
public class TimeWindowRollupWorker {

    private final TelemetryRollupRepository rollupRepository;
    private final ApplicationEventPublisher eventPublisher;

    @Value("${telemetry.ignored-user-agents:}")
    private List<String> ignoredUserAgents;

    /**
     * MICRO-BATCH: Executes every 5 minutes.
     * Continuously upserts the current active hour to provide Near Real-Time analytics to Artists.
     */
    @Scheduled(fixedRate = 300000)
    @SchedulerLock(name = "micro_batch_telemetry_rollup", lockAtLeastFor = "1m",
        lockAtMostFor = "4m")
    public void executeMicroBatchRollupCron() {
        OffsetDateTime now = OffsetDateTime.now(ZoneOffset.UTC);
        // Look back 2 hours to capture late-arriving events, spanning up to the current minute
        OffsetDateTime start = now.minusHours(2).truncatedTo(ChronoUnit.HOURS);
        OffsetDateTime end = now.plusHours(1).truncatedTo(ChronoUnit.HOURS);
    }

```

```

log.debug("[MICRO-BATCH] Aggregating telemetry from {} to {}", start, end);

try {
    List<RawPlaybackRecord> rawRecords =
rollupRepository.fetchAuthenticatedPlaybacksForWindow(start, end, ignoredUserAgents);

    // Map raw telemetry to cross-module DTOs
    List<PassiveTasteShiftsAggregatedEvent.PassiveTasteShiftRecord> shifts =
rawRecords.stream()
    .filter(r -> "FULL".equals(r.playbackStatus()) || "SKIP".equals(r.playbackStatus()))
    .map(r -> new PassiveTasteShiftsAggregatedEvent.PassiveTasteShiftRecord(r.userId(),
r.trackId(), r.playbackStatus()))
    .toList();

    if (!shifts.isEmpty()) {
        log.debug("[AI CORE] Broadcasting {} passive taste shifts.", shifts.size());
        eventPublisher.publishEvent(new PassiveTasteShiftsAggregatedEvent(shifts));
    }
} catch (Exception e) {
    // Catch to prevent analytical rollups from failing if the AI engine stalls
    log.error("[AI CORE] Failed to process passive taste shifts during micro-batch: {}",
e.getMessage(), e);
}

executeHourlyRollup(start, end, false);
}

/**
 * CRON: Executes exactly once at minute 5 of every hour.
 * Broadcasts the finalized statistics of the previously closed hour.
 */
@Scheduled(cron = "0 5 * * *")
@SchedulerLock(name = "hourly_event_broadcast", lockAtLeastFor = "2m", lockAtMostFor =
"10m")
public void executeHourlyEventBroadcastCron() {
    OffsetDateTime closedHour =
OffsetDateTime.now(ZoneOffset.UTC).minusHours(1).truncatedTo(ChronoUnit.HOURS);

    log.info("[CRON] Broadcasting aggregated events for closed hour: {}", closedHour);

    List<TrackCountersAggregatedEvent> trackEvents =
rollupRepository.fetchTrackStatsForHour(closedHour);
    trackEvents.forEach(eventPublisher::publishEvent);

    List<PlaylistCountersAggregatedEvent> playlistEvents =
rollupRepository.fetchPlaylistStatsForHour(closedHour);
    playlistEvents.forEach(eventPublisher::publishEvent);
}

/**
 * CRON: Executes every day at 02:00 AM UTC.
 */

```

```

    @Scheduled(cron = "0 0 2 * * *")
    @SchedulerLock(name = "daily_telemetry_rollup", lockAtLeastFor = "5m", lockAtMostFor =
"30m")
    public void executeDailyRollupCron() {
        OffsetDateTime targetDayEnd =
OffsetDateTime.now(ZoneOffset.UTC).truncatedTo(ChronoUnit.DAYS);
        OffsetDateTime targetDayStart = targetDayEnd.minusDays(3); // Auto-Heal Strategy

        log.info("[CRON] Starting Daily Telemetry Rollup for window: {} to {}", targetDayStart,
targetDayEnd);
        int rows = executeDailyRollup(targetDayStart, targetDayEnd, false);
        log.info("[CRON] Daily aggregation complete. {} tracks preserved.", rows);
    }

    /**
     * Executes the hourly database aggregation and dispatches Modulith events.
     *
     * @return A map containing operation statistics for the caller.
     */
    public Map<String, Integer> executeHourlyRollup(OffsetDateTime start, OffsetDateTime end,
boolean isForceRecalculation) {
        log.info("Executing Hourly Rollup from {} to {}", start, end);

        if (isForceRecalculation) {
            rollupRepository.clearHourlyAggregations(start, end);
        }

        int trackPlayRows = rollupRepository.aggregateHourlyTrackPlaybacks(start, end);
        int trackInteractionRows = rollupRepository.aggregateHourlyTrackInteractions(start, end);
        int playlistPlayRows = rollupRepository.aggregateHourlyPlaylistPlaybacks(start, end);
        int playlistInteractionRows = rollupRepository.aggregateHourlyPlaylistInteractions(start, end);

        return Map.of(
            "trackPlaybacksAggregated", trackPlayRows,
            "trackInteractionsAggregated", trackInteractionRows,
            "playlistPlaybacksAggregated", playlistPlayRows,
            "playlistInteractionsAggregated", playlistInteractionRows
        );
    }

    /**
     * Executes the daily database compression logic.
     */
    public int executeDailyRollup(OffsetDateTime start, OffsetDateTime end, boolean
isForceRecalculation) {
        log.info("Executing Daily Compression Rollup from {} to {} (Force: {})", start, end,
isForceRecalculation);

        if (isForceRecalculation) {
            rollupRepository.clearDailyAggregations(start, end);
        }
    }

```

```

        int compressedTracksCount = rollupRepository.aggregateDailyTrackStats(start, end);

        rollupRepository.aggregateDailyPlaylistStats(start, end);
        rollupRepository.aggregateDailyTrackDemographics(start, end);
        rollupRepository.aggregateDailyTrackSources(start, end);
        rollupRepository.refreshTrackPopularityScores();
        return compressedTracksCount;
    }
}

```

Інтерфейс вкладки статистики треку

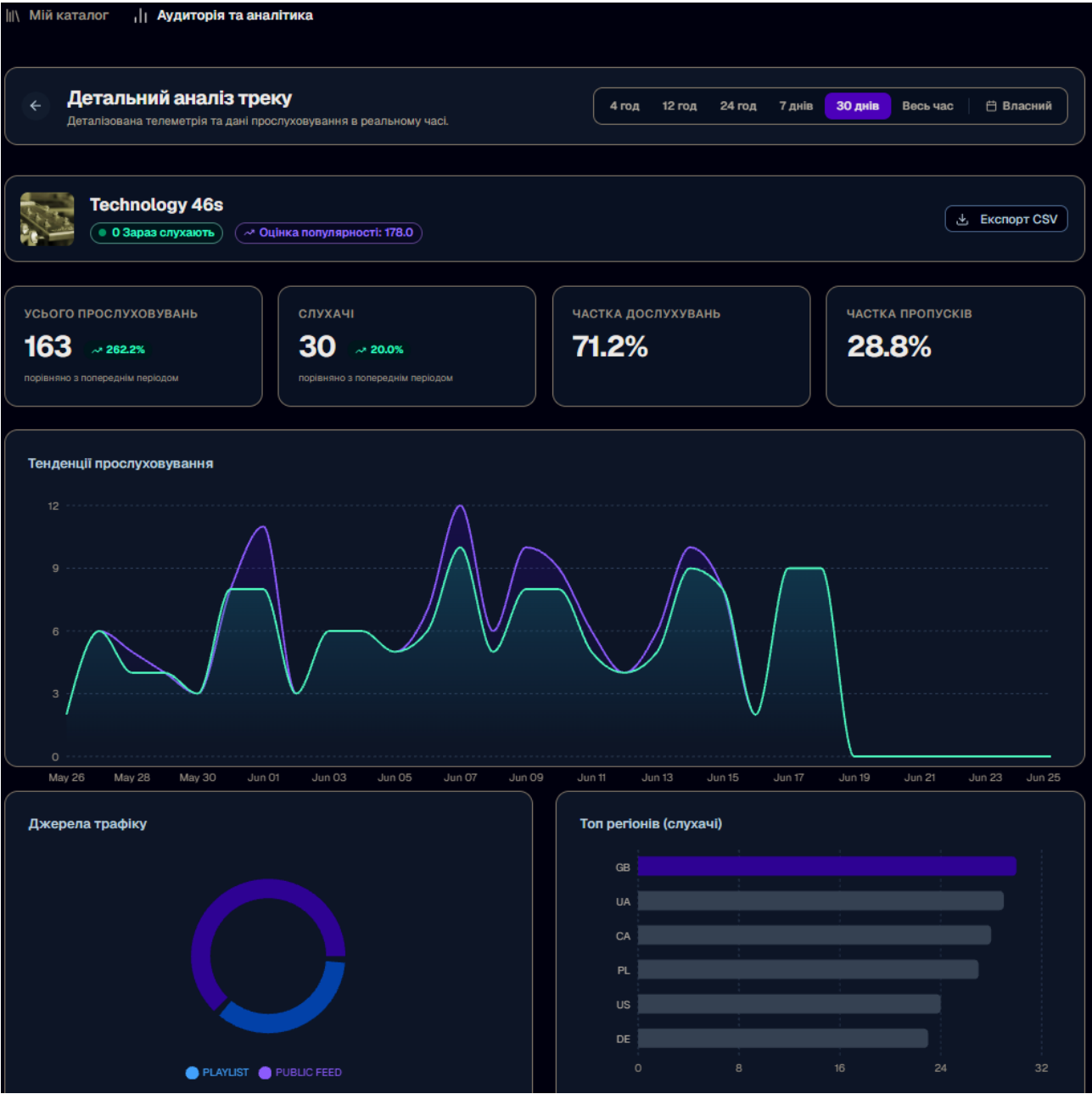


Рис. К.1 - Інтерфейс вкладки статистики треку