

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістра

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ
ВИЗНАЧЕННЯ ПОПУЛЯРНОСТІ АЛЬБОМІВ МУЗИЧНИХ ВИКОНАВЦІВ
RESEARCH AND SOFTWARE IMPLEMENTATION OF METHODS FOR
EVALUATING THE POPULARITY OF MUSIC ALBUMS

Виконав: студент(ка) 2 курсу, групи КНТ-121м

Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Галімов О.А.

(прізвище та ініціали)

Керівник Скрупський С.Ю.

(прізвище та ініціали)

Рецензент Голуб Т.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 2022 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Галімова Олексія Андрійовича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів визначення популярності альбомів музичних виконавців
Research and Software Implementation of Methods for Evaluating the Popularity of Music Albums

керівник проєкту (роботи) Скрупський Степан Юрійович, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “4” листопада 2022 року № 363

2. Строк подання студентом проєкту (роботи) грудень 2022 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Проєктування програмного забезпечення. 4. Основні рішення щодо реалізації компонентів системи. 5. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Скрупський С.Ю., доцент		
Нормоконтролер	Дейнега Л.Ю., ст. викладач		

7. Дата видачі завдання “1” вересня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз та дослідження предметної області	2-5 тижні	Розділи 1, 2
3	Розробка архітектури програми	6 тиждень	Розділ 3
4	Розробка програми	7-9 тижні	Розділ 4
5	Тестування та експериментальне дослідження програми	10-11 тижні	Розділ 5
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	12 тиждень	Додатки
7	Захист роботи	13 тиждень	

Студент(ка)

(підпис)

Галімов О.А.

(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Скрупський С.Ю.

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
100 с., 25 рис., 3 табл., 3 дод., 16 джерел.

МУЗИЧНИЙ АЛЬБОМ, МУЗИЧНИЙ ГУРТ, ВИКОНАВЕЦЬ,
ПОПУЛЯРНІСТЬ АЛЬБОМУ, КЛАСИФІКАЦІЯ.

Об'єкт роботи – процес впорядкування прослуховування музичних композицій і їх альбомів. Предмет роботи – методи визначення популярності альбомів музичних виконавців.

Мета роботи – розробити програмне забезпечення прослуховування музичних альбомів і композицій для впорядкування даного процесу зі сторони слухачів і підтримки менеджерів з авторами альбомів.

Матеріали, методи та технічні засоби: мова Python, бібліотека scikit-learn, фреймворк веброзробки Django, система керування базами даних MySQL.

Результати. Наукова новизна роботи сформована в методі визначення популярності альбомів музичних виконавців, який виконує класифікацію музичних альбомів на популярні і непопулярні, застосовуючи алгоритм ID3 на основі групи ознак, що характеризують музичні композиції в альбомі за різними критеріями, що дозволяє приймати рішення стосовно керування маркетинговою компанією альбому до його видання.

Висновки. Представлене за результатами роботи програмне забезпечення дозволяє надавати слухачам інформацію про альбоми, композиції, музичні гурти, об'єднувати їх у власні переліки, підписуватися на переліки інших слухачів, ділитися персональними порадами, висловлювати власні уподобання композиціям і альбомам, визначати популярність альбомів.

Галузь використання – музичні сервіси, менеджмент музичних гуртів.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 100 pages, 25 figures, 3 tables, 3 appendixes, 16 sources.

MUSIC ALBUM, MUSIC BAND, ARTIST, ALBUM POPULARITY, CLASSIFICATION.

The object of the work is the process of arranging the listening of musical compositions and its albums. The subject of the work is methods of determining the popularity of musical artists' albums.

The goal of the work is to develop software for listening to music albums and compositions to arrange this process on the part of listeners and to support managers together with album authors.

Materials, methods and technical means: Python language, scikit-learn library, Django web development framework, MySQL database management system.

The results. The scientific novelty of the work is formed in the method of determining the popularity of music albums, which performs the classification of music albums into popular and unpopular classes, applying the ID3 algorithm based on a group of features characterizing the musical compositions in an album according to various criteria. The method allows decisions to be made regarding the management of the marketing company of an album before its release.

Conclusions. The software, presented based on the results of the work, allows users to provide listeners with information about albums, compositions, musical groups, combine them into own lists, subscribe to the lists of other listeners, share personal advices, express user own preferences for compositions and albums, and determine the popularity of albums.

Field of usage is music services, management of music bands.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз проблеми та постановка завдань дослідження	10
1.1 Основні характеристики музичних композицій та альбомів.....	10
1.2 Огляд програмного забезпечення прослуховування музичних альбомів	12
1.3 Висновки за розділом 1	15
2 Матеріали і методи.....	16
2.1 Задача визначення популярності альбомів музичних виконавців	16
2.2 Дерева рішень	17
2.3 Метод визначення популярності альбомів музичних виконавців	18
2.4 Висновки за розділом 2	20
3 Проєктування програмного забезпечення	22
3.1 Вибір рішень для розробки програмного забезпечення.....	22
3.2 Визначення функціональних вимог до програми	23
3.3 Визначення структури бази даних	28
3.4 Висновки за розділом 3	43
4 Основні рішення щодо реалізації компонентів системи.....	44
4.1 Опис структури розробленої програмної системи	44
4.2 Основні реалізовані класи	45
4.3 Висновки за розділом 4	51
5 Експлуатація, тестування та експериментальне дослідження програми	52
5.1 Опис роботи з програмою	52
5.2 Експериментальне дослідження	57
5.3 Висновки за розділом 5	60
Висновки	61
Перелік джерел посилання	62
Додаток А Технічне завдання	64

Додаток Б Текст програми	70
Додаток В Слайди презентації.....	93

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

ID3 – Iterative Dichotomiser 3;

БД – база даних.

ВСТУП

Застосування інформаційних технологій для прослуховування музики свого часу змінило дану галузь і сприяло її подальшому активному розвитку. Але одним з логічних етапів цього розвитку є збільшення аудіо-даних, серед яких важко знайти ту музику, яка зацікавить кожного користувача особисто, а окрім того важко визначити ще й, на скільки буде популярним той чи інший альбом виконавця [1]-[3]. З одного боку слухачі потребують знаходження тієї музики, які справді може їх зацікавити. З іншого боку автори альбомів, представники сервісів з прослуховування музики мають потребу в тому, щоб зрозуміти, які альбоми можуть бути більш або менш цікавими аудиторії. У цій роботі основна увага приділена розв'язанню проблеми як раз з другого боку, тобто зі сторони музичних виконавців та сервісів прослуховування музики.

У сучасному світі прослуховування музики відбувається декілька фактично протилежних тенденцій прослуховування. З одного боку люди все ж рідше слухають музику саме у вигляді концепції повного альбому, частіше ніж раніше віддаючи перевагу окремим композиціям або перелікам композицій [4]-[5]. І це призводить до того, що авторам, які створюють альбоми, ще важливішим стає розуміння успішності результату їх тривалої роботи [6]. З іншого боку відбувається поширення вінілових дисків, а вони представляють музику саме у вигляді альбомів. Це сприяє тому, щоб концепція альбомів розвивалась далі.

Тому в роботі потрібно розробити програму, яка буде використовуватися для прослуховування і впорядкування прослуховування музики, враховуючи концепцію альбомів та виконуючи визначення їх популярності після публікації.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Основні характеристики музичних композицій та альбомів

До основних характеристик музичних композицій, які використовуються одним з найвідоміших сервісів прослуховування музики, сервісом Spotify, і які дозволяють перевести музичну композицію в цифрове її представлення на основі набору значень характеристик, належать:

- тональність музичної композиції, що представляється цілими числами;
- модальність: музична композиція може бути мажорною (значення 1) чи мінорною (значення 0);
- позитивність: міра від 0 до 1, що репрезентує ступінь позитивності, яку виражає музична композиція: високе значення відповідає музичним композиціям, які звучать більш позитивно, а музичні композиції з низьким значенням цього показника звучать більш негативно;
- танцювальність – це показник, який визначає те, в якій мірі слід вважати, що композиція пасує під танець, для обчислення цього показника поєднуються різні музичні елементи, при цьому нульове значення відповідає композиції, для якої танцювальність не характерна, а значення 1 – для найбільш танцювальної;
- енергія – це показник, який вказує на активність і інтенсивність музичної композиції, знаходиться в інтервалі від 0,0 до 1,0;
- акустичність – це показник, який визначає те, чи є музична композиція акустичною: для акустичної композиції дорівнює 1, а нульове значення визначає протилежний стан;
- речитативність – це показник, який визначає, чи є в музичній композиції вимовлений текст: більше значення (ближче до 1) представляє більшу частку мови в композиції, при цьому за значення більше 0,66 композиції можуть майже повністю або повністю складатися з тексту, за значень від 0,33 до 0,66 композиції більш ймовірно містять як музику, так і

текст, значення менше 0,33 відповідає випадкам, коли вимовлений текст швидше за все відсутній;

– інструментальність – це показник, який визначає, чи є ця музична композиція інструментальною, тобто чи відсутній у ній вокал: у випадку значення 1 музична композиція не містить вокальної складової або якщо проводиться оцінка, то можна говорити, що ймовірність наявності вокалу в композиції є дуже низькою, при цьому якщо значення перевищує 0,5, то це визначає, що музична композиція більш ймовірно є інструментальною;

– виконання наживо – це показник, який визначає, чи виконана ця музична композиція наживо, чи вона була записана в студії звукозапису: музична композиція вважається записаною наживо, коли слухачі присутні на запису, значення вище 0,8 вказують на високу вірогідність живого виконання;

– темп – це показник, який вимірюється кількістю ударів на хвилину;

– тактовий розмір – це показник, який визначає кількість ударів у кожному такті музичної композиції: може бути, наприклад, «3/4» [7]-[8].

На даний момент сервіс Spotify, є певним рушієм, який визначає тренди музичного ринку, надаючи важливу для кожного виконавця інформацію, включаючи кількість підписників, кількість прослуховувань, а додатковим інструментом, який зашитий в роботу сервісу загалом, є індекс популярності Spotify, який характеризується наступним:

– сервіс Spotify відстежує популярність всіх музичних композицій відносно всіх інших композицій: сервіс має значення, яке ранжує всіх виконавців та їхні композиції за шкалою від 0 до 100, що і є індексом популярності, який показує, на скільки популярна композиція на платформі, підсумовуючи популярність усіх пісень з кожного альбому і з кожного виконавця;

– високе значення індексу популярності покращує видимість музичної композиції на платформі;

- індекс популярності Spotify не відображається на сторінці виконавця, на відміну від інших показників, як наприклад, кількість слухачів і підписників за місяць, це значення доступне тільки через прикладний програмний інтерфейс для розробників Spotify;
- індекс популярності визначає, як алгоритм Spotify рекомендує композицію новим слухачам, додає її до переліків відтворення та полегшує її пошук;
- індекс популярності Spotify розраховується на основі загальної кількості потоків музичної композиції, частоти відтворення композиції та часу, який минув з останнього прослуховування [9]-[12].

1.2 Огляд програмного забезпечення прослуховування музичних альбомів

Виходячи з описаних фактів у попередньому підрозділі, звичайно, що першим програмним засобом, який має бути розглянуто в розрізі програм прослуховування музичних альбомів, є Spotify (рис. 1.1) [13].

Spotify надає велику кількість функціональних можливостей, пов'язаних з прослуховуванням як окремих музичних композицій, так і їх альбомів, переліків. Переліки складаються як редакторами, так і автоматично сервісом, а також і користувачами. Користувачі можуть ділитися власними переліками. Вони можуть також зберігати собі переліки інших користувачів, уподобати композиції, альбоми, музичні гурти, що призводить до їх внесення у відповідні переліки користувача. Окрім того, як було розглянуто в попередньому підрозділі, Spotify через прикладний інтерфейс надає також додаткову інформацію про музичні композиції. Хоча слід відзначити, що в самому інтерфейсі сервісу ці можливості недоступні. Будь-який користувач не може отримати саме цю інформацію таким чином. Усі користувачі можуть дізнатися про кількість прослуховувань композиції, з цього складаються рейтинги найкращих композицій гурту.

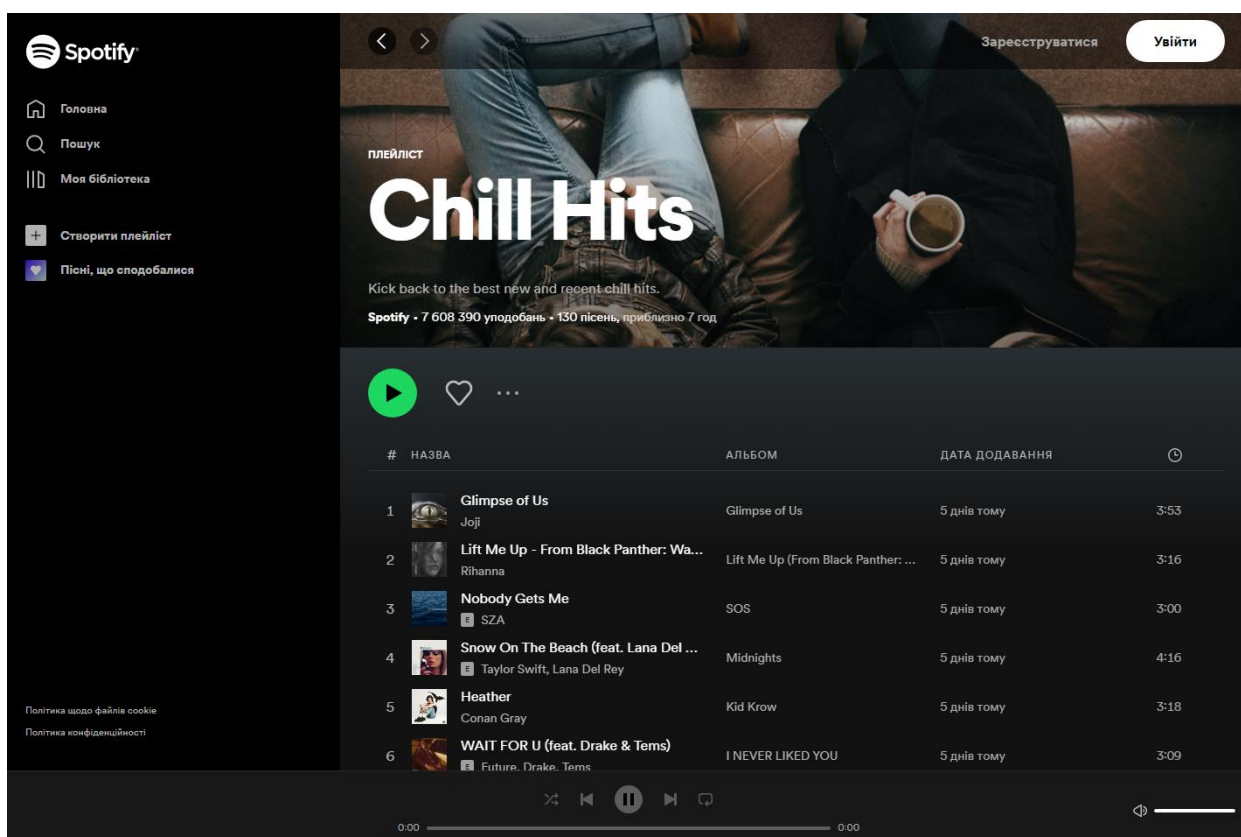


Рисунок 1.1 – Інтерфейс Spotify

Основою роботи є сторінка альбому або переліку, звідки можна отримати доступ до композицій. Слід відзначити, що це сервіс, який має платні підписки, і для того щоб слухати музику без реклами, потрібно оформити платну підписку.

Подібним сервісом є Soundcloud (рис. 1.2) [14]. Він на відміну від попереднього надає значно більше можливостей безкоштовного прослуховування. Сервіс також дозволяє прослуховувати музику композиціями, альбомами, можна підписуватися на інших користувачів і слід відзначити, що можливості соціальної мережі набагато ширші як раз у Soundcloud і практично не розвинені у Spotify.

За результатами огляду виділено основні функціональні можливості цих програм, визначено можливості, які будуть додатково реалізовані в програмі, яка розробляється, і які охоплюють створення порад для прослуховування та визначення популярності альбомів у майбутньому (табл. 1.1).

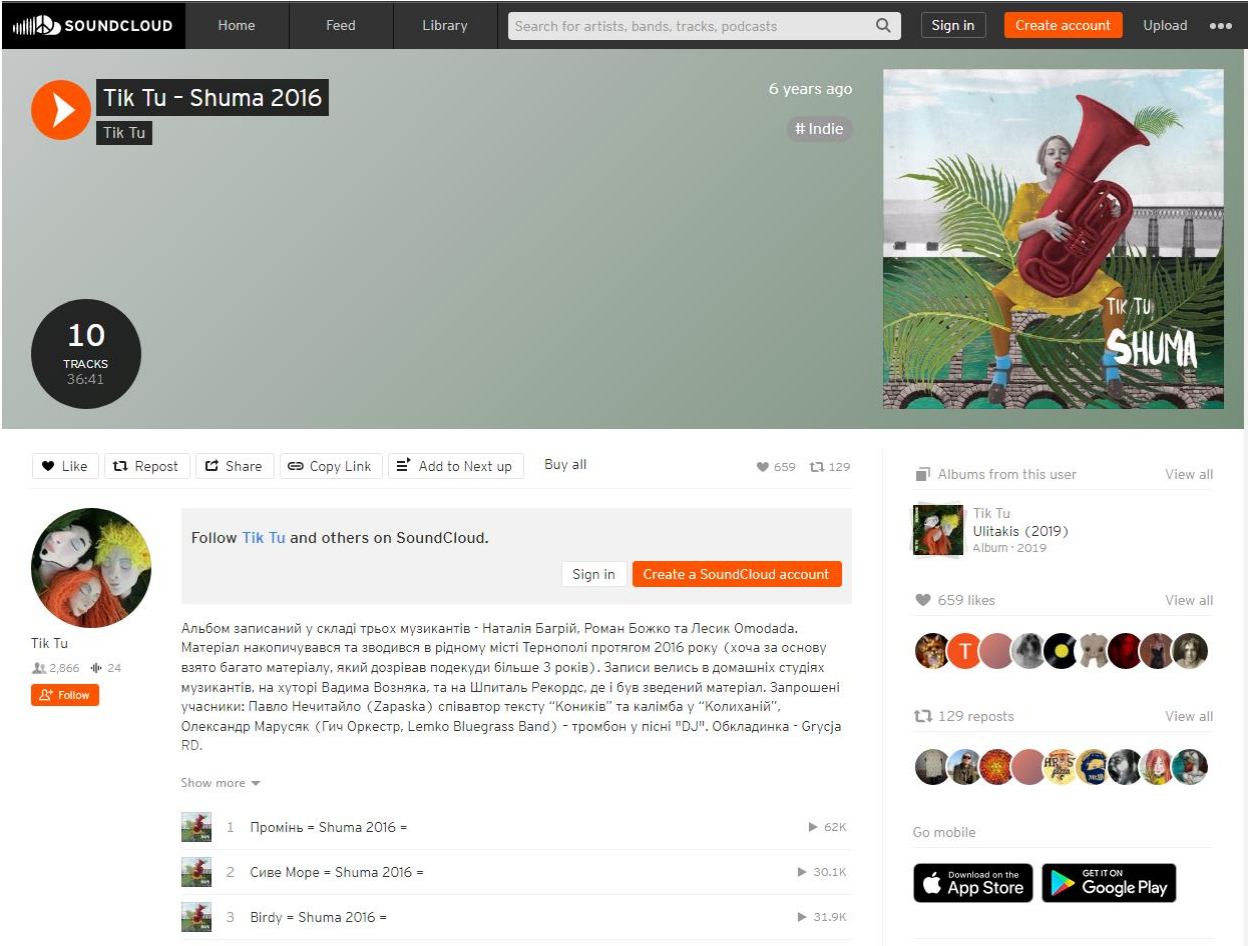


Рисунок 1.2 – Інтерфейс Soundcloud

Таблиця 1.1 – Порівняння програмних аналогів

Критерій	Spotify	Soundcloud	Розробка
Прослуховування альбомів, композицій	+	+	+
Створення і надання доступу до переліків композицій/альбомів	+/-	+/-	+/+
Відправлення персональних порад	—	—	+
Визначення популярності альбомів	—	—	+

1.3 Висновки за розділом 1

Встановленими завданнями дипломної кваліфікаційної роботи магістра є:

- дослідити методи визначення популярності альбомів музичних виконавців;
- спроектувати програмне забезпечення прослуховування музичних альбомів і композицій;
- реалізувати програмне забезпечення.

У першому розділі розглянуто основні характеристики композицій, музичних альбомів, питання індексу популярності композицій, виконано огляд подібних програм, порівняння розробки з ними.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Задача визначення популярності альбомів музичних виконавців

Використовуючи приведені результати в розділі 1, популярність музичних альбомів розглядається саме на основі індексу, який використовується Spotify. Враховуючи логіку його практичного застосування і для підвищення ефективності практичного застосування створеного рішення зокрема, було вирішено розглядати задачу визначення популярності альбомів музичних виконавців як задачу класифікації.

У такому випадку дана задача полягає в тому, щоб визначити, до якого класу належить альбом, популярність якого ще не відома. Це може бути або клас популярних альбомів, або клас непопулярних альбомів. До популярних альбомів віднесені альбоми, які за індексом популярності отримуються рівень вище 30, відповідно альбоми з меншим значенням даного індексу вважаються непопулярними. Ця інформація може використовуватися менеджерами гуртів для того, щоб зрозуміти перспективи альбому на музичному ринку і прийняти адекватне рішення про варіант маркетингової компанії, який буде застосований в підсумку.

Якщо альбом віднесений до непопулярних, то в такому випадку можна прийняти рішення про збільшення витрат на маркетингову компанію, щоб досягти бажаних результатів, підбираючи відповідні механізми, адекватні саме для даного альбому.

Якщо альбом віднесений до популярних, то по-перше, менеджер музичного гурту, сам музичний гурт, менеджери сервісу прослуховування музики можуть розуміти, що він може принести певну фінансову вигоду. В сучасних умовах представники гурту можуть розуміти, що на ринку альбом має певний шанс, альбом принесе певні кошти в підсумку, а також можуть розуміти, що інвестування коштів в маркетингову компанію альбому є не просто необхідним як в першому випадку, а є доцільним і обґрунтованим. У

свою чергу менеджери сервісу розуміють, що цей альбом можна з більшою вірогідністю пропонувати користувачам для прослуховування, що пропонування даного альбому на сервісі може принести додаткову вигоду самому сервісу.

Таким чином, задача визначення популярності альбомів музичних виконавців є задачею класифікації музичного альбому за його потенційною популярністю. При цьому альбом характеризується набором вхідних ознак, значення яких є вхідними даними, а вихідними даними – клас потенційної популярності альбому. Вхідні дані визначають характеристики альбому, які доцільно формувати на основі характеристик окремих композицій з використанням системи критеріїв, яка була описана в розділі 1.

2.2 Дерева рішень

Дерево рішень – це структура, яка містить вузли (візуально зазвичай позначаються колами або прямокутниками) і ребра (стрілки) і яка побудована з набору даних, що представляється загальною певною таблицею, яка складається зі стовпців, що представляють атрибути даних, а рядки відповідають екземплярам даних, при цьому кожен вузол використовується або для прийняття рішення (вузол рішення), або для представлення результату (листовий вузол) [15].

Алгоритм побудови дерев рішень Iterative Dichotomiser 3 (ID3) загальною визначається наступною процедурою:

- створити кореневий вузол для дерева, яке містить увесь набір даних;
- знайти найкращий атрибут, використовуючи деяку міру вибору атрибутів, яка відома як міра вибору атрибутів;
- розділити увесь набір даних на підмножини відповідно до вибраного атрибута;
- рекурсивно створювати нові дерева рішень, використовуючи підмножини набору даних, створеного на попередньому кроці: цей процес

триває доти, доки не настане етап, коли вже не можна буде далі класифікувати вузли (розділяти загальний простір пошуку) і тоді останній вузол листовим вузлом [16].

При побудові дерева рішення головне питання полягає в тому, як вибрати найкращий атрибут для кореневого вузла або вузлів інших рівнів, адже випадковим вибором вузла, який буде кореневим, цю проблему неможливо розв'язати (випадковий підхід може дати погані результати з низькою точністю), а для розв'язання цього питання використовуються дві основні техніки:

- ентропія – це міра випадковості інформації, що обробляється: чим більша ентропія у вузлі, тим менше інформації відомо про класифікацію даних на цьому етапі дерева: гілка з ентропією 0 є чистим листовим вузлом, а гілка з ентропією більше нуля потребує подальшого поділу;

- приріст інформації – це статистична властивість, яка використовується для вимірювання того, на скільки добре заданий атрибут відокремлює навчальні дані залежно від цільової класифікації: це різниця між ентропією до виконання розділення та середньою ентропією після розділення для даного набору даних на основі заданих значень атрибута, приріст інформації базується на зменшенні ентропії після того, як набір даних розділено за атрибутом, відповідно потрібно знайти атрибут, який повертає найбільший приріст інформації та найменшу ентропію, це найбільш однорідна гілка [16].

2.3 Метод визначення популярності альбомів музичних виконавців

Метод визначення популярності альбомів музичних виконавців дозволяє розв'язати загальним чином сформульовану в цьому розділі задачу. Для розв'язання задачі спочатку було прийнято рішення про сукупність ознак, якої характеризується кожен музичний альбом для виконання першого

етапу, на якому необхідно було сформулювати вибірку даних для навчання моделі.

Кожен альбом представляється набором, що сформований зі значень наступних ознак:

- danceability – мінімальне, максимальне і середнє значення за критерієм танцювальності серед всіх композицій альбому;
- energy – мінімальне, максимальне і середнє значення за критерієм енергія серед всіх композицій альбому;
- key – мінімальне, максимальне і середнє значення за критерієм тональність серед всіх композицій альбому;
- mode – значення за критерієм модальність, що найчастіше зустрічається серед всіх композицій альбому;
- speechiness – мінімальне, максимальне і середнє значення за критерієм речитативність серед всіх композицій альбому;
- acousticness – мінімальне, максимальне і середнє значення за критерієм акустичність серед всіх композицій альбому;
- instrumentalness – мінімальне, максимальне і середнє значення за критерієм інструментальність серед всіх композицій альбому;
- liveness – мінімальне, максимальне і середнє значення за критерієм виконання наживо серед всіх композицій альбому;
- valence – мінімальне, максимальне і середнє значення за критерієм позитивність серед всіх композицій альбому;
- tempo – мінімальне, максимальне і середнє значення за критерієм темп серед всіх композицій альбому;
- timesignature – мінімальне, максимальне і середнє значення за критерієм тактовий розмір серед всіх композицій альбому.

Тобто для того щоб отримати значення відповідних ознак потрібно спочатку отримати ці значення для кожної музичної композиції з цього альбому. Тоді знаючи всі ці окремі значення, потрібно звести їх до більш обмеженого простору, що складається з 3 значень для всіх критеріїв окрім

модальності. Оскільки в альбомі зазвичай трохи більше 10 композицій, хоча можуть зустрічатися і більші альбоми, наприклад, подвійні, то така кількість ознак є скороченням простору ознак.

Усі наявні екземпляри даних, для яких відомі значення всіх ознак, описані вище, формують у підсумку навчальну вибірку.

На другому етапі роботи методу потрібно створити модель визначення популярності альбому за допомогою застосування дерева рішень. В якості базового алгоритму навчання дерева рішень використовувався алгоритм ID3, хоча він може бути замінений і на інший алгоритм підтримки роботи дерев рішень, який здатен виконувати класифікацію на основі вибірки, описаної вище. За результатами виконання цього етапу побудована модель має бути збережена для повторюваного використання на наступних етапах.

Етапи 3-4 виконуються тоді, коли з'являється новий музичний альбом або змінюються значення ознак неопублікованого альбому.

На етапі 3 необхідно на основі композицій альбому сформувати екземпляр даних, який характеризує музичний альбом, у відповідності зі структурою, описаною вище. Тоді на четвертому етапі має бути подано цей екземпляр на вхід моделі і отримано клас популярності для цього екземпляру альбому.

Клас популярності (популярний або непопулярний альбом) є підсумковим результатом роботи методу.

2.4 Висновки за розділом 2

Сформульовано загальну постановку задачі визначення популярності альбомів музичних виконавців як задачу класифікації. Розглянуто дерева рішень в якості моделі для розв'язання задачі.

Наукова новизна роботи сформована в методі визначення популярності альбомів музичних виконавців, який виконує класифікацію

музичних альбомів на популярні і непопулярні, застосовуючи алгоритм ID3 на основі групи ознак, що характеризують музичні композиції в альбомі за різними критеріями, що дозволяє приймати рішення стосовно керування маркетинговою компанією альбому до його видання.

3 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір рішень для розробки програмного забезпечення

Вимоги віддаленого доступу різних користувачів, використання різних пристроїв для доступу, реалізації інтелектуальних засобів спершу призвели до прийняття рішення про необхідність створення вебдодатку, а потім було прийнято рішення про використання мови Python, що дозволяє створити інтелектуально-орієнтовані вебдодатки в підсумку. Третій етап прийняття рішень для розробки полягав у виборі Python-фреймворка для реалізації вебдодатків, основними з яких є Flask і Django. Результати за виконаним порівнянням представлені в таблиці 3.1.

Таблиця 3.1 – Порівняння фреймворків для розробки вебдодатку

Критерій	Flask	Django
Тип фреймворку	Легковісний	Повний стек
Тип проєктів	Малий	Великий, середній
Взаємодія з системами керування базами даних	Потребує підключення зовнішніх засобів	Підтримує популярні системи MySQL, PostgreSQL, Oracle
Зв'язування об'єктів даних з таблицями бази даних	Вручну або підключенням зовнішніх засобів	Вбудоване
Вбудоване вирішення адміністративних завдань	Ні	Так
Підтримка динамічних вебсторінок	Ні	Так

Для реалізації обрано фреймворк Django, тому що він дозволяє створювати складніші додатки, а даний додаток є багатофункціональним і

доволі складним, має багато вбудованих засобів, тобто не потребує додаткового вивчення і підключення сторонніх засобів.

3.2 Визначення функціональних вимог до програми

Як було представлено в технічному завданні, програмне забезпечення повинно мати широкий набір функціональних можливостей. Використовуючи цей перелік, було визначено сценарії роботи основних користувачів системи. Серед основних користувачів системи не було розглянуто окремо адміністратора, оскільки він повинен мати можливість змінювати, створювати будь-які дані, а це автоматично реалізується на основі вибору, виконаного в попередньому підпункті. Тобто Django вимагає тільки реєстрації моделей додатку для того, щоб з ними можна було працювати через адміністративну панель. Тому неавторизованих користувачів, авторизованих користувачів-слухачів і менеджерів сервісу розглянуто в якості акторів системи.

Неавторизовані користувачі мають такий набір сценаріїв (рис. 3.1):

- перегляд композицій альбому;
- прослуховування композиції;
- перегляд сторінки музичного гурту;
- перегляд переліку композицій;
- перегляд переліку музичних гуртів;
- перегляд переліку альбомів;
- відтворення переліку найкращих композицій;
- відтворення переліку найкращих альбомів;
- відтворення переліку найкращих музичних гуртів;
- перегляд доступних переліків композицій користувача-слухача;
- перегляд доступних переліків альбомів користувача-слухача;
- перегляд доступних переліків музичних гуртів користувача-слухача;
- реєстрація, що має альтернативним сценарієм авторизацію;

- перегляд сторінки виконавця;
- перегляд альбомів виконавця;
- перегляд альбомів музичного гурту;
- пошук музичних виконавців;
- пошук музичних гуртів.

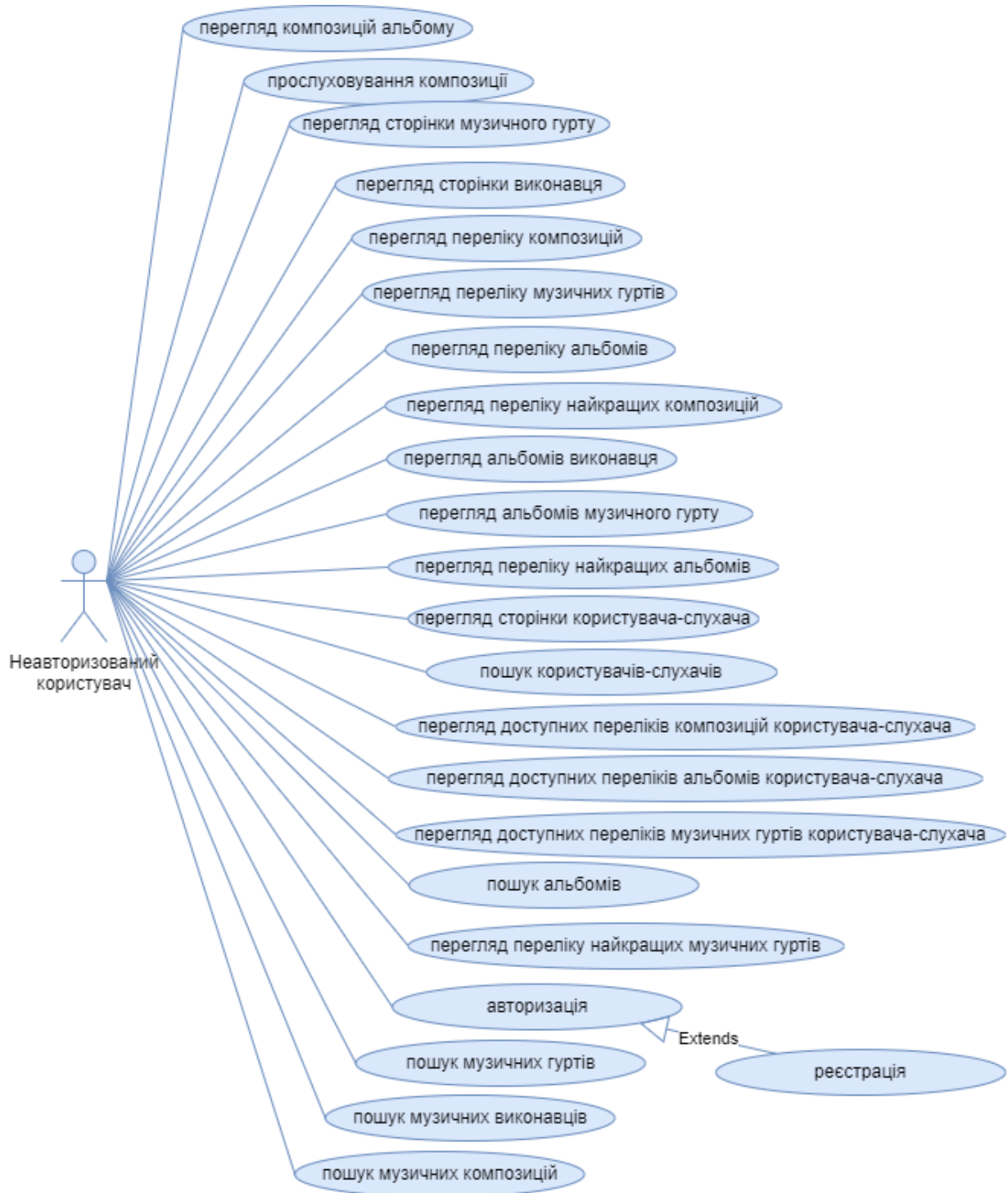


Рисунок 3.1 – Діаграма прецедентів неавторизованого користувача

До сценаріїв неавторизованого користувача також належать:

- пошук музичних композицій;
- пошук альбомів;
- перегляд сторінки користувача-слухача;
- пошук користувачів-слухачів.

Авторизовані користувачі-слухачі мають такий набір сценаріїв окрім сценаріїв неавторизованих користувачів:

- збереження альбому до власного переліку, альтернативним сценарієм до якого є вилучення альбому з власного переліку альбомів, якщо він вже був збережений раніше в ньому;
- створення власного переліку альбомів;
- збереження композиції до власного переліку, альтернативним сценарієм до якого є вилучення композиції з власного переліку композицій, якщо вона вже була збережений раніше в ньому;
- створення власного переліку композицій;
- збереження музичного гурту до власного переліку, альтернативним сценарієм до якого є вилучення музичного гурта з власного переліку гуртів, якщо він вже був збережений раніше в ньому;
- створення власного переліку музичних гуртів;
- перегляд власних переліків композицій, альтернативним сценарієм до якого є встановлення рівня доступу до власного переліку композицій, якщо користувач бажає змінити доступну аудиторію;
- перегляд власних переліків альбомів, альтернативним сценарієм до якого є встановлення рівня доступу до власного переліку альбомів, якщо користувач бажає змінити доступну аудиторію;
- перегляд власних переліків музичних гуртів, альтернативним сценарієм до якого є встановлення рівня доступу до власного переліку музичних гуртів, якщо користувач бажає змінити доступну аудиторію;
- перегляд підписаних переліків композицій;
- перегляд підписаних переліків альбомів;

- перегляд підписаних переліків музичних гуртів;
- відправлення поради користувачу-слухачу прослухати альбом;
- відправлення поради користувачу-слухачу прослухати музичну композицію;

- відтворення адресованих порад прослухати музичні композиції;
- відтворення адресованих порад прослухати альбоми;
- відтворення власних порад прослухати музичні композиції;
- відтворення власних порад прослухати альбоми;
- уподобання альбому;
- уподобання музичної композиції;
- збереження переліку композицій іншого користувача-слухача;
- збереження переліку альбомів іншого користувача-слухача;
- збереження переліку музичних гуртів іншого користувача-слухача;
- відтворення сторінок підписаних користувачів-слухачів;
- підписування на користувача-слухача;
- відтворення переліку підписників користувача-слухача;
- відтворення нових альбомів музичних гуртів з переліку;
- відтворення переліку власних уподобаних композицій;
- відтворення переліку власних уподобаних альбомів;
- скасування уподобання альбому;
- скасування уподобання композиції;
- відписування від користувача-слухача (рис. 3.2).

Менеджери сервісу окрім сценаріїв авторизованого користувача-слухача мають такий набір сценаріїв:

- авторизація;
- публікація переліків альбомів авторства сервісу;
- публікація переліків композицій авторства сервісу;
- додавання музичного гурту;
- редагування даних музичного гурту;
- публікація альбому;

- редагування даних альбому;
- публікація композиції;
- публікація інформації про виконавця;
- перегляд каталогу виконавців, що має альтернативним сценарієм редагування інформації виконавця.

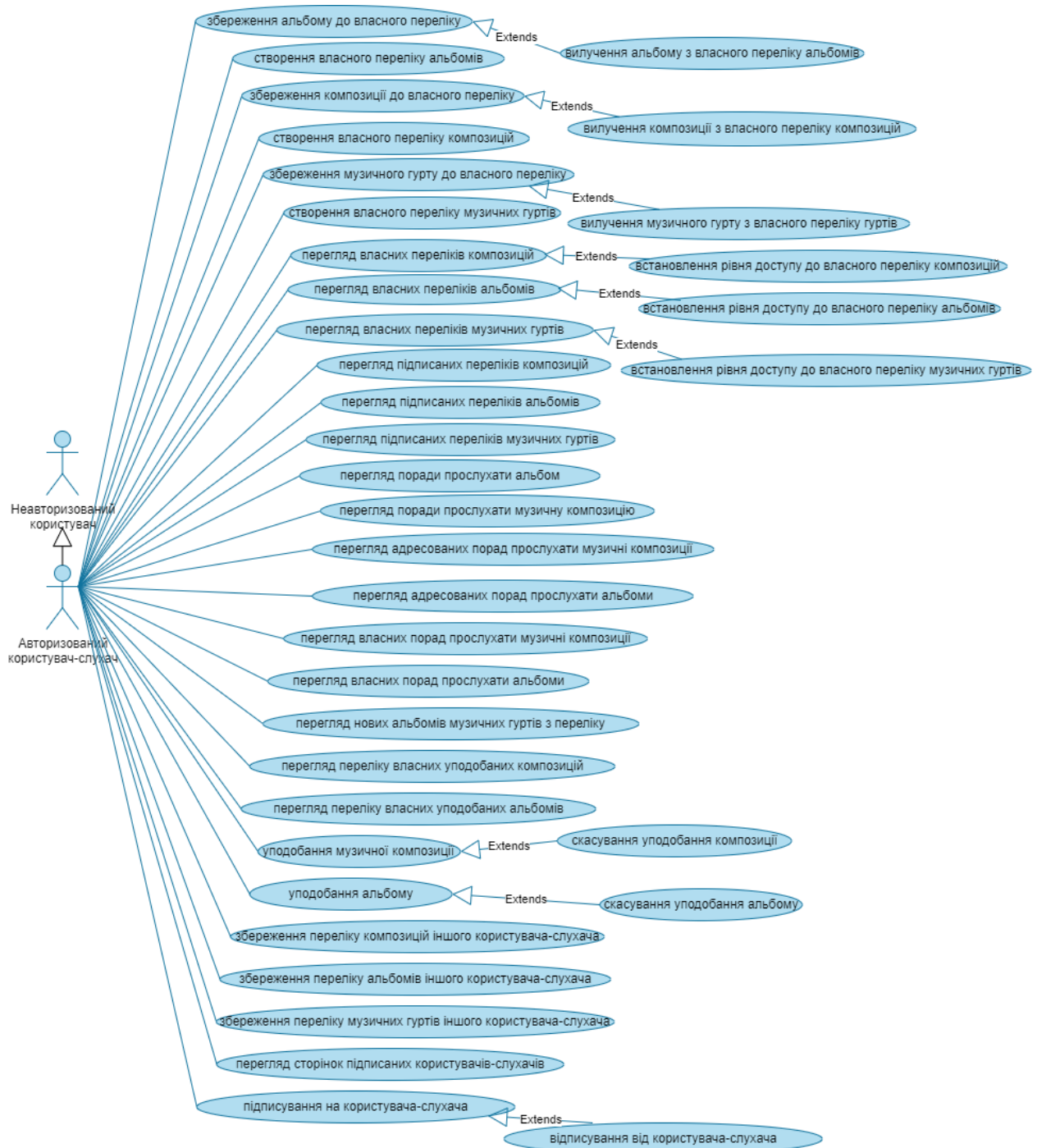


Рисунок 3.2 – Діаграма прецедентів авторизованого користувача-слухача

До сценаріїв менеджера сервісу (рис. 3.3) також належать:

- перегляд каталогу композицій, що має альтернативним сценарієм редагування композиції;
- перегляд каталогу музичних гуртів, що має альтернативним сценарієм редагування даних музичного гурту;
- перегляд каталогу альбомів, що має альтернативним сценарієм редагування альбому.



Рисунок 3.3 – Діаграма прецедентів менеджера сервісу

3.3 Визначення структури бази даних

База даних (БД) дозволяє зберігати в ній усі дані, які стосуються роботи з програмою. MySQL при цьому є достатньою системою керування БД, до того ж це безкоштовне рішення.

Музиканти можуть об'єднуватися в гурти, а гурти створювати композиції, але також можуть бути і окремі виконавці, які створюють музику, не об'єднуючись з іншими музикантами. Фактично ці дві сутності представляють колектив або одиночного виконавця, які відносно авторства в

альбомі, а також інформації про них мало чим відрізняються. Їх характеристики фактично співпадають, що зручності роботи над подальшою функціональністю сприяє, адже тоді всі переліки, підписки будуть повторюватися по два рази і передбачати необхідність витягання даних окремо і їх об'єднати в єдину сутність для відображення на вебсторінках. Тому було вирішено створити єдину таблицю `mustasterapp_musband` в БД для цих двох сутностей, а вже для записів у таблиці, які представляють музичні гурти, окремо визначено склад цих гуртів, щоб зберігати дані про них як про єдину одиницю. Таблиця має таку структуру полів:

- `id` – ідентифікаційний номер музичного гурта;
- `musname` – назва музичного гурта;
- `site` – вебсайт музичного гурта;
- `origin` – країна походження музичного гурта, представлена зовнішнім ключем з таблиці `mustasterapp_origin` з забезпеченням зв'язку один-до-багатьох, де одна країна походження має різні музичні гурти, при цьому звичайно музичний гурт може відноситися тільки до однієї країни;
- `presentation` – текстова презентація музичного гурта;
- `muspicture` – візуальна презентація музичного гурта (стандартно фотографія гурта);
- `inlists` – кількість переліків музичних гуртів, у які користувачі-слухачі внесли цей гурт;
- `playings` – кількість прослуховувань будь-яких композицій цього музичного гурта.

Враховуючи описану концепцію про об'єднання в попередній таблиці гуртів з виконавцями, створено таблицю для визначення учасників музичного гурта (відповідно для індивідуального виконавця такі дані не зберігаються в цій таблиці), що має назву `mustasterapp_musbandperformer` і таку структуру полів:

- `id` – ідентифікаційний номер співвідношення виконавця з музичним гуртом;

- `musband` – музичний гурт, у якому бере участь цей виконавець, представлений зовнішнім ключем з таблиці `mustasterapp_musband` з забезпеченням зв'язку один-до-багатьох;

- `performer` – виконавець, який є учасником даного музичного гурта, представлений зовнішнім ключем з таблиці `mustasterapp_origin` з забезпеченням зв'язку один-до-багатьох;

- `role` – роль виконавця як учасника в гурті (співак, бас-гітарист, соло-гітарист і т.д.).

Як видно зі структури, таблиця `mustasterapp_musbandperformer` реалізує зв'язок між таблицями `mustasterapp_musband` і `mustasterapp_musbandperformer`, адже він представляє тип багато-до-багатьох, оскільки можливі ситуації, коли один і той же виконавець бере участь в різних музичних гуртах, а відповідно гурт складається більше ніж з одного виконавця.

Розглянутий тут виконавець є окремою сутністю і представлений таблицею, що має назву `mustasterapp_performer` і має таку структуру полів:

- `id` – ідентифікаційний номер виконавця музики;
- `appname` – ім'я виконавця;
- `origin` – країна походження виконавця, представлена зовнішнім ключем з таблиці `mustasterapp_origin` з забезпеченням зв'язку один до багатьох, де одна країна походження має різних виконавців, при цьому звичайно виконавець може походити тільки з однієї країни;
- `site` – персональний вебсайт з інформацією про виконавця;
- `presentation` – текстова презентація музичного виконавця;
- `muspicture` – візуальна презентація музичного виконавця (стандартно фотографія виконавця).

Країна походження виконавців і музичних гуртів є також окремою таблицею, що використовується для зручності роботи з цими даними і яка має назву `mustasterapp_origin` і таку структуру полів:

- `id` – ідентифікаційний номер країни походження;

– `countryname` – назва країни походження.

Альбоми, в які композиції об'єднуються гуртами і видаються для концептуального прослуховування, представлені таблицею, що має назву `mustasterapp_longplay` і таку структуру полів:

- `id` – ідентифікаційний номер альбому музичного гурта;
- `musband` – музичний гурт, який створив цей альбом, представлений зовнішнім ключем з таблиці `mustasterapp_musband` з забезпеченням зв'язку один-до-багатьох, де один музичний гурт може мати багато альбомів, але один альбом може бути створений як основним тільки одним музичним гуртом;
- `lptitle` – назва альбому музичного гурта;
- `inlikes` – кількість уподобань користувачами-слухачами цього альбому музичного гурта;
- `inlists` – кількість переліків, у які різні користувачі-слухачі внесли цей альбом музичного гурта;
- `inadvices` – кількість персональних порад, наданих користувачами-слухачами іншим користувачам, щодо прослуховування цього альбому музичного гурта;
- `inlistenings` – кількість прослуховувань користувачами-слухачами цього альбому музичного гурта;
- `popularity` – індекс популярності цього альбому музичного гурта;
- `release` – дата і час видання альбому.

У записі альбомів також можуть брати участь запрошені виконавці, які не входять до складу гурта або які допомагають індивідуальному виконавцю записати альбом. Ці дані знаходяться в таблиці, що має назву `mustasterapp_longplayperformer` і таку структуру полів:

- `id` – ідентифікаційний номер участі виконавця в записі музичного альбому;

- `longplay` – музичний альбом, в запису якого взяв участь виконавець як запрошений, представлений зовнішнім ключем з таблиці `mustasterapp_longplay` з забезпеченням зв'язку один-до-багатьох;

- `performer` – виконавець, який взяв участь в записі цього альбому, представлений зовнішнім ключем з таблиці `mustasterapp_performer` з забезпеченням зв'язку один-до-багатьох;

- `role` – роль, з якою взяв участь виконавець в записі альбому (співак, бас-гітарист і т.д.).

Таблиця `mustasterapp_longplayperformer` реалізує зв'язок багато-до-багатьох між таблицями `mustasterapp_longplay` і `mustasterapp_performer`, адже один альбом може мати багато запрошених виконавців для запису, а один виконавець може взяти участь в записах різних альбомів.

Спеціальні позначки, які використовуються для визначення необхідних особливостей альбомів, знаходяться в таблиці, що має назву `mustasterapp_mark` і таку структуру полів:

- `id` – ідентифікаційний номер позначки;
- `shortening` – текст позначки.

Зв'язок позначок з музичними альбомами належить типу багато-до-багатьох, оскільки одна позначка має використовуватися для різних альбомів, а один альбом за необхідності може бути позначений різними позначками. Ці дані знаходяться в таблиці, що має назву `mustasterapp_longplaymark` і таку структуру полів:

- `id` – ідентифікаційний номер позначки альбому;
- `longplay` – альбом, якому було встановлено цю позначку, представлений зовнішнім ключем з таблиці `mustasterapp_longplay` з забезпеченням зв'язку один-до-багатьох;
- `mark` – позначка, яка була встановлена цьому альбому, представлений зовнішнім ключем з таблиці `mustasterapp_mark` з забезпеченням зв'язку один-до-багатьох.

Користувачі-слухачі можуть уподобати альбоми, щоб виказати свою симпатію та можливо повернутися потім до їхнього прослуховування. Ці дані знаходяться в таблиці, що має назву `mustasterapp_longplaylike` і таку структуру полів:

- `id` – ідентифікаційний номер уподобання користувачем-слухачем альбому;
- `longplay` – альбом, який сподобався користувачу-слухачу, представлений зовнішнім ключем з таблиці `mustasterapp_longplay` з забезпеченням зв'язку один-до-багатьох, де один альбом може мати багато уподобань, при цьому звичайно уподобання може відноситися тільки до одного альбому в одній ситуації;
- `listenuser` – користувач-слухач, якому сподобався альбом, представлений зовнішнім ключем з таблиці `mustasterapp_listenuser` з забезпеченням зв'язку один-до-багатьох, де одне уподобання може бути сформовано тільки одним користувачем-слухачем одночасно, але кожен користувач-слухач в різний час може створити багато уподобань різних альбомів;
- `happened` – дата і час, коли користувач-слухач уподобав альбом.

Музичні композиції, які формують альбоми або видаються самостійно, знаходяться в таблиці, що має назву `mustasterapp_muscomposition` і таку структуру полів:

- `id` – ідентифікаційний номер музичної композиції;
- `longplay` – музичний альбом, до складу якого входить композиція: значення є зовнішнім ключем з таблиці `mustasterapp_longplay`, яке може бути пропущено, якщо музична композиція не входить до складу альбому, зв'язок з таблицею типу один-до-багатьох, де одна музична композиція входить тільки до одного альбому, а один альбом може містити деяку кількість музичних композицій;
- `mctitle` – повна назва музичної композиції;
- `audio` – аудіодані музичної композиції;

- `ordernumber` – порядковий номер композиції в альбомі;
- `inlikes` – кількість уподобань даної музичної композиції різними користувачами-слухачами;
- `inlists` – кількість переліків, до складу яких різні користувачі-слухачі внесли дану композицію;
- `inlistenings` – кількість прослуховувань цієї музичної композиції;
- `inadvices` – кількість персональних порад, наданих користувачами-слухачами іншим користувачам, щодо прослуховування цієї композиції;
- `danceability` – танцювальність;
- `energy` – енергія;
- `key` – тональність;
- `mode` – модальність;
- `speechiness` – речитативність;
- `acousticness` – акустичність;
- `instrumentalness` – інструментальність;
- `liveness` – виконання наживо;
- `valence` – позитивність;
- `tempo` – темп;
- `timesignature` – тактовий розмір.

Користувачі-слухачі можуть уподобати не тільки альбоми загалом, але і окремі композиції, щоб виказати свою симпатію та можливо повернутися потім до прослуховування цієї композиції. Ці дані знаходяться в таблиці, що має назву `mustasterapp_muscompositionlike` і має таку структуру полів:

- `id` – ідентифікаційний номер уподобання користувачем-слухачем музичної композиції;
- `muscomposition` – музична композиція, яку уподобав користувач-слухач, представлена зовнішнім ключем з таблиці `mustasterapp_muscomposition` з забезпеченням зв'язку один-до-багатьох, де

одна композиція може мати різну кількість уподобань, а одне уподобання може стосуватися тільки однієї музичної композиції;

- `listenuser` – користувач-слухач, який уподобав композицію, представлена зовнішнім ключем з таблиці `mustasterapp_listenuser` з забезпеченням зв'язку один-до-багатьох, де один користувач-слухач може мати різні уподобання музичних композицій, при цьому кожне уподобання виконується тільки одним користувачем-слухачем одночасно;

- `happened` – дата і час, коли користувач-слухач уподобав композицію.

Переліки музичних композицій, які складають користувачі-слухачі, знаходяться в таблиці, що має назву `mustasterapp_muscompositionlist` і таку структуру полів:

- `id` – ідентифікаційний номер переліку композиційних композицій;

- `listenuser` – користувач-слухач, який створив цей перелік музичних композицій;

- `accessmode` – доступ до переліку музичних композицій: приватний, для користувачів-слухачів, які підписалися на автора переліку, публічний (для всіх) або рівень всього застосунку (переліки, які створюються менеджерами сервісу і мають знаходитися на базових сторінках застосунку);

- `mcldtitle` – назва переліку музичних композицій;

- `happened` – дата і час, коли створено цей перелік музичних композицій;

- `presentation` – текстова презентація переліку його автором;

- `muspicture` – візуальна презентація переліку музичних композицій.

- `infollowers` – кількість випадків, коли будь-який користувач-слухач підписався на цей перелік.

Будь-який перелік складається з окремих музичних композицій. У певний момент їх може не бути в переліку, це може бути одна композиція, але зазвичай в переліку міститься певна кількість композицій. Відповідно кожна композиція може входити до складу різних переліків, тому тип зв'язку

між таблицями багато-до-багатьох. Через це було створено окрему таблицю, що має назву `mustasterapp_muscompositionlistpart` і таку структуру полів:

- `id` – ідентифікаційний номер входження композиції до відповідного переліку;
- `muscompositionlist` – перелік музичних композицій, до якого входить дана, представлений зовнішнім ключем з таблиці `mustasterapp_muscompositionlist` з забезпеченням зв'язку один-до-багатьох;
- `muscomposition` – музична композиція, яка входить до цього переліку, представлена зовнішнім ключем з таблиці `mustasterapp_muscomposition` з забезпеченням зв'язку один-до-багатьох.

Користувачі-слухачі можуть підписуватися на переліки музичних композицій інших користувачів. Ці дані знаходяться в таблиці, що має назву `mustasterapp_muscompositionlistfollower` і таку структуру полів:

- `id` – ідентифікаційний номер збереження списку користувачем-слухачем до себе;
- `muscompositionlist` – перелік музичних композицій, який вирішив зберегти користувач, представлений зовнішнім ключем з таблиці `mustasterapp_muscompositionlist` з забезпеченням зв'язку один-до-багатьох, де один перелік може мати багато збережень різними користувачами, при цьому кожне збереження стосується тільки одного переліку одночасно;
- `listenuser` – користувач-слухач, який зберіг цей перелік музичних композицій, представлений зовнішнім ключем з таблиці `mustasterapp_listenuser` з забезпеченням зв'язку один-до-багатьох, де один користувач-слухач може зберегти різні переліки композицій, але одне збереження виконується тільки одним користувачем одночасно;
- `happened` – дата і час, коли користувач-слухач зберіг цей перелік музичних композицій.

Окрім переліків композицій дозволяється створювати переліки альбомів, які складають користувачі-слухачі. Вони знаходяться в таблиці, що має назву `mustasterapp_longplaylist` і таку структуру полів:

- id – ідентифікаційний номер переліку альбомів;
- listenuser – користувач-слухач, який створив цей перелік;
- accessmode – доступ до переліку альбомів: приватний, для користувачів-слухачів, які підписалися на автора переліку, публічний (для всіх) або рівень всього застосунку (переліки, які створюються менеджерами сервісу і мають знаходитися на базових сторінках застосунку);
- lp1title – назва переліку альбомів;
- happened – дата і час, коли було створено перелік;
- presentation – текстова презентація переліку альбомів його автором;
- muspicture – візуальна презентація переліку альбомів.
- infollowers – кількість випадків, коли будь-який користувач-слухач підписався на цей перелік.

Ці переліки складаються з окремих альбомів. У певний момент їх може не бути в переліку, це може бути один альбом, але зазвичай в переліку міститься певна кількість альбомів. Відповідно кожен альбом може входити до складу різних переліків, тому тип зв'язку між таблицями багато-до-багатьох. Через це було створено окрему таблицю, що має назву `mustasterapp_longplaylistpart` і таку структуру полів:

- id – ідентифікаційний номер входження альбому до відповідного переліку;
- longplaylist – перелік альбомів, до якого внесено цей альбом, представлений зовнішнім ключем з таблиці `mustasterapp_longplaylist` з забезпеченням зв'язку один-до-багатьох;
- longplay – альбом, який внесено до переліку, представлений зовнішнім ключем з таблиці `mustasterapp_longplay` з забезпеченням зв'язку один-до-багатьох.

Користувачі-слухачі можуть підписуватися на переліки альбомів інших користувачів. Ці дані знаходяться в таблиці, що має назву `mustasterapp_longplaylistfollower` і має таку структуру полів:

- `id` – ідентифікаційний номер збереження користувачем-слухачем переліку альбомів собі;

- `longplaylist` – перелік альбомів, який зберіг собі користувач-слухач, представлений зовнішнім ключем з таблиці `mustasterapp_longplaylist` з забезпеченням зв'язку один-до-багатьох, де одна підписка стосується одного переліку альбомів, при цьому на кожен перелік може бути оформлено багато підписок різними користувачами-слухачами;

- `listenuser` – користувач-слухач, який зберіг цей перелік альбомів, представлений зовнішнім ключем з таблиці `mustasterapp_listenuser` з забезпеченням зв'язку один-до-багатьох, де одна підписка на перелік оформляється тільки одним користувачем-слухачем, при цьому користувач-слухач може мати багато підписок на різні переліки альбомів;

- `happened` – дата і час, коли користувач-слухач зберіг цей перелік альбомів.

Також слухач може створювати переліки музичних гуртів. Вони необхідні для тих ситуацій, коли користувач-слухач не відносить гурт до улюблених, проте хоче в певний момент повертатися до музики деяких виконавців, отримувати перелік їх нових альбомів. Відповідні дані знаходяться в таблиці, що має назву `mustasterapp_musbandlist` і має таку структуру полів:

- `id` – ідентифікаційний номер переліку музичних гуртів;

- `listenuser` – користувач-слухач, який створив цей перелік музичних гуртів, представлений зовнішнім ключем з таблиці `mustasterapp_listenuser` з забезпеченням зв'язку один-до-багатьох, де один користувач-слухач може створити декілька різних переліків музичних гуртів, при цьому один перелік має тільки одного автора;

- `accessmode` – доступ до переліку музичних гуртів: приватний, для користувачів-слухачів, які підписалися на автора переліку, публічний (для всіх);

- `mbtitle` – назва переліку музичних гуртів;

- happened – дата і час, коли було створено перелік;
- presentation – текстова презентація переліку музичних гуртів його автором;
- muspicture – візуальна презентація переліку музичних гуртів.
- infollowers – кількість випадків, коли будь-який користувач-слухач підписався на цей перелік.

Ці переліки складаються з окремих музичних гуртів (або індивідуальних виконавців). У певний момент їх може не бути в переліку, це може бути один гурт, але зазвичай в переліку міститься певна кількість гуртів. Відповідно кожен гурт може входити до складу різних переліків, тому тип зв'язку між таблицями багато-до-багатьох. Через це було створено окрему таблицю, що має назву `mustasterapp_musbandlistpart` і таку структуру полів:

- id – ідентифікаційний номер входження гурту до відповідного переліку;
- musbandlist – перелік, до якого входить музичний гурт, представлений зовнішнім ключем з таблиці `mustasterapp_musbandlist` з забезпеченням зв'язку один-до-багатьох;
- musband – музичний гурт, який входить до переліку, представлений зовнішнім ключем з таблиці `mustasterapp_musband` з забезпеченням зв'язку один-до-багатьох.

Користувачі-слухачі можуть підписуватися на переліки музичних гуртів. Ці дані знаходяться в таблиці, що має назву `mustasterapp_musbandlistfollower` і має таку структуру полів:

- id – ідентифікаційний номер збереження користувачем-слухачем переліку гуртів собі;
- musbandlist – перелік гуртів, який зберіг собі користувач-слухач, представлений зовнішнім ключем з таблиці `mustasterapp_musbandlist` з забезпеченням зв'язку один-до-багатьох, де одна підписка може бути

оформлена на один перелік музичних гуртів, при цьому на один перелік може бути багато підписок;

- `listenuser` – користувач-слухач, який зберіг цей перелік гуртів, представлений зовнішнім ключем з таблиці `mustasterapp_listenuser` з забезпеченням зв'язку один-до-багатьох, де одна підписка на перелік оформляється тільки одним користувачем-слухачем, при цьому користувач-слухач може мати багато підписок на різні переліки гуртів;

- `happened` – дата і час, коли користувач-слухач зберіг цей перелік гуртів.

Користувачі-слухачі є основними учасниками процесів, які формують життєвий цикл всієї програмної платформи. Дані про них знаходяться в таблиці, що має назву `mustasterapp_listenuser` і таку структуру полів:

- `id` – ідентифікаційний номер користувача-слухача;
- `appname` – повне ім'я користувача-слухача, яке він бажає відобразити в програмі (може використовуватися і певний нік);
- `muspicture` – зображення або фотографія, які асоціює користувач-слухач зі своїм профілем;
- `login` – логін користувача-слухача;
- `password` – пароль користувача-слухача;
- `infollowers` – кількість користувачів-слухачів, що підписались на сторінку цього користувача;
- `origin` – країна походження користувача-слухача: може бути і пропущена, якщо користувач не бажає її визначати, представлена зовнішнім ключем з таблиці `mustasterapp_origin` з забезпеченням зв'язку один-до-багатьох, де одна країна походження може мати різних користувачів-слухачів у програмі, при цьому кожен користувач-слухач може мати одну або жодної країни походження;
- `email` – адреса електронної пошти користувача-слухача.

Підписки для швидкого повернення до сторінок користувачів-слухачів, а також для доступу до напівприхованих переліків композицій,

альбомів та музичних гуртів реалізуються між користувачами-слухачами і знаходяться в таблиці, що має назву `mustasterapp_listenuserfollower` і таку структуру полів:

- `id` – ідентифікаційний номер підписки користувача-слухача;
- `listenuser` – користувач-слухач, на якого підписався даний користувач, представлений зовнішнім ключем з таблиці `mustasterapp_listenuser` з забезпеченням зв'язку один-до-багатьох, де на одного користувача-слухача можуть підписатися різні інші користувачі, при цьому одна підписка оформляється тільки на одного конкретного користувача-слухача;
- `follower` – користувач-слухач, який підписався на даного користувача, представлений зовнішнім ключем з таблиці `mustasterapp_listenuser` з забезпеченням зв'язку один-до-багатьох, де один користувач-слухач може мати багато підписок, при цьому одна підписка оформляється тільки одним користувачем одночасно;
- `happened` – дата і час, коли користувач-слухач підписався на іншого користувача.

Поради прослухати музичну композицію, від одного користувача-слухача іншому знаходяться в таблиці, що має назву `mustasterapp_muscompositionadvice` і таку структуру полів:

- `id` – ідентифікаційний номер поради з музичною композицією;
- `listenuser` – користувач-слухач, який створив цю пораду, представлений зовнішнім ключем з таблиці `mustasterapp_listenuser` з забезпеченням зв'язку один-до-багатьох, де один користувач-слухач може створити декілька порад, при цьому одна порада має одного автора;
- `addresser` – користувач-слухач, якому адресований ця порада, представлений зовнішнім ключем з таблиці `mustasterapp_listenuser` з забезпеченням зв'язку один-до-багатьох, де одна порада може бути адресована тільки одному користувачу-слухачу, при цьому один користувач-слухач може мати багато адресованих йому порад;

- `muscomposition` – музична композиція, якої стосується порада, представлена зовнішнім ключем з таблиці `mustasterapp_muscomposition` з забезпеченням зв'язку один-до-багатьох, де одна музична композиція може бути використана в різних порадах, при цьому одна порада стосується тільки однієї музичної композиції;

- `happened` – дата і час створення поради.

Поради прослухати альбом знаходяться в таблиці, що має назву `mustasterapp_longplayadvice` і таку структуру полів:

- `id` – ідентифікаційний номер поради з альбомом;

- `listenuser` – користувач-слухач, який створив цю пораду, представлений зовнішнім ключем з таблиці `mustasterapp_listenuser` з забезпеченням зв'язку один-до-багатьох, де один користувач-слухач може створити декілька порад, при цьому одна порада має одного автора;

- `addresser` – користувач-слухач, якому адресована ця порада, представлена зовнішнім ключем з таблиці `mustasterapp_listenuser` з забезпеченням зв'язку один-до-багатьох, де одна порада може бути адресована тільки одному користувачу-слухачу, при цьому один користувач-слухач може мати багато адресованих йому порад;

- `longplay` – альбом, з порадою прослухати який і була створена ця порада, представлений зовнішнім ключем з таблиці `mustasterapp_longplay` з забезпеченням зв'язку один-до-багатьох, де один альбом може бути використаний в різних порадах, при цьому одна порада стосується тільки одного альбому;

- `happened` – дата і час створення поради.

Схема структури БД, що відповідає описаним таблицям і зв'язкам між ними, подана на рис. 3.4.

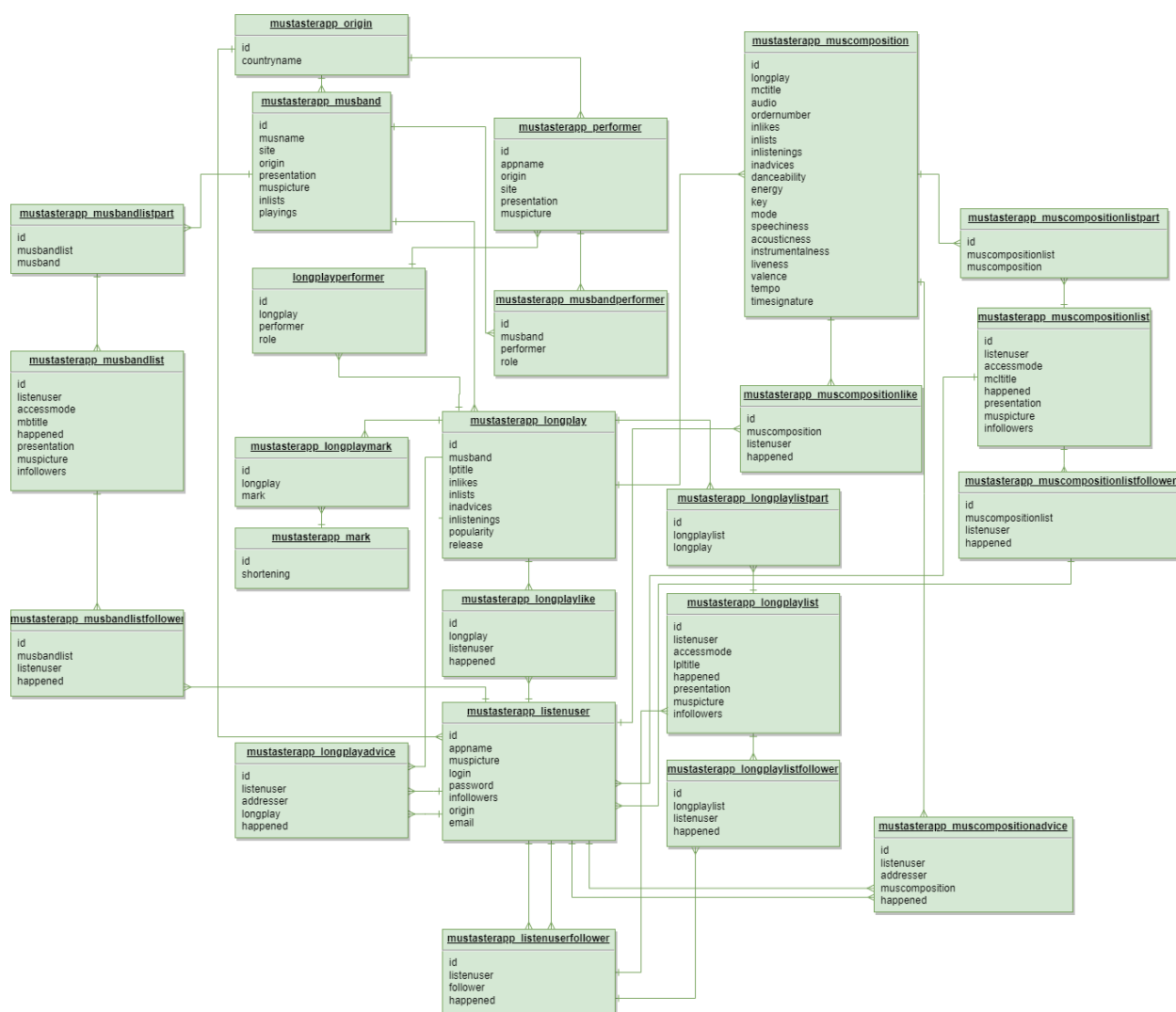


Рисунок 3.4 – Схема структури БД

3.4 Висновки за розділом 3

У третьому розділі описано логіку створення вебдодатку і використання мови Python для реалізації цього вебдодатку, виконано порівняння фреймворків для цієї мови, Flask і Django, обрано Django для проєктування і реалізації вебдодатку, що в підсумку визначає основні технології, які використовувались для створення вебдодатку, виконано моделювання функціональних вимог з діаграмами прецедентів для неавторизованого користувача, користувача-слухача і для менеджера сервісу, визначено структуру роботи з даними в БД, обрано систему керування БД MySQL для підтримки роботи з даними.

4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

4.1 Опис структури розробленої програмної системи

Розробка програмного забезпечення була виконана через застосування фреймворку Django, який побудований на основі подібного до Model-View-Controller шаблону проєктування, який часто інтерпретується як шаблон Model-View-Template. Це означає, що стандартні для Model-View-Controller контролери реалізуються через представлення вебдодатку, а ось представлення фактично можна інтерпретувати як шаблони в Model-View-Template.

Представлення відповідно підпадають під ті сценарії, які було представлено в розділі 3. Для зручності роботи з ними вони були розділені на 4 частини:

- `lp_view.py` для представлень роботи з альбомами музичних гуртів;
- `band_view.py` для представлень роботи з музичними гуртами;
- `compos_view.py` для представлень роботи з композиціями;
- `views.py` для представлень роботи з обліковими записами, питаннями авторизації та допоміжними представленнями.

Для основної кількості представлень створені шаблони вебсторінок, що знаходяться в окремій директорії `templates`. Для представлень створення і змінювання об'єктів створено по одному шаблону і в них використовуються форми, які знаходяться в файлі `forms.py`. Зв'язування адрес вебсторінок з представленнями виконано в файлі `urls.py`.

Розроблені моделі в чіткій відповідності зі структурою БД знаходяться в файлі `models.py`.

Програмну реалізацію запропонованого методу розміщено в директорії `popularity`. Реалізацію забезпечено через окремі файли: `trainer.py`, що забезпечує навчання моделі прогнозування популярності альбомів, `newres.py`, що забезпечує прогнозування популярності нових альбомів.

4.2 Основні реалізовані класи

При реалізації вебдодатку було використано створені за результатами виконання другого розділу діаграми прецедентів. Абсолютно під кожен сценарій було створено відповідний клас представлення і шаблон вебсторінки. Також було реалізовано всі необхідні додаткові складові вебдодатків Django та забезпечено реалізацію описаного методу. Усі моделі реалізовані класами, структура яких відповідає структурі таблиць БД, яка була детально описана в розділі 3.

Відображення даних альбому підтримується створенням шаблону вебсторінки `longplay.html`, дані до якого завантажуються з класу представлення `LongplayView`. Оскільки це сторінка, яка в термінології Django виконує деталізацію, то наслідування класу `DetailView` з Django забезпечило основу для його реалізації. Для цього класу основним методом, який підтримує його роботу, є метод `get`, де як раз для шаблону передаються потрібні дані. Дані включають зокрема дані самого альбому, пов'язаних з ним позначок і виконавців, які були залучені до запису альбому.

Важливою особливістю цієї сторінки є те, що в ній мають бути відображені різні елементи керування в залежності від того, який досвід роботи з цим альбомом має користувач-слухач. Тобто якщо користувач-слухач раніше вподобав цей альбом, то йому має графічно відображатися можливість скасувати це вподобання, а якщо він ще не уподобав його або раніше скасував уподобання, то має відображатися відповідно можливість уподобання. Для цього серед даних в результаті формується відповідна позначка, яка в складі словника з даними знаходиться за ключем `liked`. Якщо користувач не вподобав цей альбом, то міститься значення `False`, а якщо вподобав, то `True`. Для цього вибираються уподобання альбому, а серед них перевіряється, чи є уподобання авторизованого користувача-слухача. Якщо

воно є, то вноситься значення True, а якщо немає, то залишається значення False.

За аналогічною логікою організована робота з переліками авторизованого користувача. Тобто з одного боку користувач повинен мати сам перелік з власними переліками альбомів, куди він може додати цей альбом, а з іншого боку він повинен знати, чи внесено в цей перелік цей альбом раніше. Для цього відбувається витягання всіх переліків альбомів користувача-слухача, а потім реалізується циклічне проходження по всіх елементах і для кожного визначається, чи конкретно цей перелік містить в своїх об'єктах моделі Longplaylistpart, тобто в своїх складових (альбомах) цей альбом. Якщо це так, то до відповідного об'єкту додається атрибут inlist зі значенням True або зі значенням False, якщо навпаки. Всі переліки в підсумку містяться в словнику з результуючими даними за ключем lists.

Відображення композицій зі складу альбому підтримується створенням шаблону вебсторінки longplaylist.html, дані до якого завантажуються з класу представлення LongplaylistDetailView. Це також відповідно представлення деталізації даних. Особливе це представлення тим, що в ньому відбувається перевірка, чи має доступ відповідний користувач до відповідного переліку. Відповідним користувачем вважається авторизований користувач-слухач. У методі get в параметрах міститься параметр pk, що визначає номер переліку альбомів. За цим номером завантажуються дані самого об'єкта переліку альбомів. Вони долучаються до результуючого словника, яким наповнюється шаблон вебсторінки, за ключем longplaylist.

Оскільки перелік має 4 можливі варіанти доступу до нього, то перевірка відбувається співставленням цих варіантів. За варіанта з номером 0 відбувається доступ без обмежень до переліку, який створений користувачем-слухачем. За варіанта з номером 3 відбувається доступ без обмежень до переліку, який асоціюється з самим сервісом загалом. Тобто за цих двох варіантів дані переліку повністю передаються вебсторінці. У випадку значення 3 доступ приватний, тобто тільки самим користувачем-

слухачем, тому дані повертаються тільки тоді, якщо автор співпадає з авторизованим користувачем-слухачем. У випадку значення 2 доступ відбувається тільки для підписників цього користувача. Тому спочатку перевіряється, чи є авторизований користувач-слухач підписником користувача-слухача, який є автором цього переліку альбомів. Для цього витягаються дані об'єкту `Listenuserfollower`. Якщо вони є і варіант доступу дорівнює 1, то дані переліку також повертаються. Також через параметр `ordering`, переданий в адресі сторінки, вибирається варіант впорядкування:

- значення 0 відповідає впорядкуванню за кількістю уподобань цього альбому всіма користувачами-слухачами сумарно;
- значення 1 відповідає впорядкуванню за кількістю випадків, коли цей перелік зберіг інший користувач-слухач (який не є його автором);
- значення 2 відповідає впорядкуванню за кількістю порад, в яких було рекомендовано цей альбом користувачам-слухачам особисто;
- значення 3 відповідає впорядкуванню за кількістю прослуховувань всіх композицій альбому;
- значення 4 відповідає впорядкуванню за популярністю альбому;
- значення 5 відповідає впорядкуванню за датою випуску альбому.

Аналогічні системи впорядкуванню використовуються в багатьох інших представленнях. Якщо у відповідної моделі якісь з цих параметрів відсутні, то і у впорядкуванні потім відповідно користувачу вони не доступні, а в коді вони не обробляються.

Саме ж внесення менеджером сервісу альбомів, змінювання їх параметрів відбувається на основі реалізованих класів `LongplayCreateView`, `LongplayUpdateView` відповідно. Вони реалізовані на основі взаємодії на сторінці з відповідною формою `LongplayForm`, клас якої визначає наявність полів, що визначають всі атрибути альбому. Так само зокрема побудовано процес створення власних переліків. Наприклад, перелік альбомів створюється на основі класу представлення `LongplaylistCreateView`, а змінюється цей перелік на основі класу `LongplaylistUpdateView`.

Уподобання альбому підтримується класом представлення `LongplayLikeView`, яке вимагає авторизації попередньо і відповідно реалізовано на основі базового класу представлення `View`. Підтримка цієї функціональності відбувається через створення методу `post`, тобто відкрита передача даних за посиланням тут не виконується. У даному методі через параметр `pk` встановлюється сам номер альбому, за яким витягається з БД відповідний об'єкт. За цим альбомом і для цього авторизованого користувача-слухача перевіряється, чи є вже уподобання у нього цього альбому. Якщо є, то уподобання вилучається, а в об'єкті самого альбому кількість уподобань зменшується на 1. Якщо уподобання немає, то створюється відповідний об'єкт і зберігається в БД, а в об'єкті самого альбому кількість уподобань зменшується на 1.

Збереження альбому до переліку альбомів користувача підтримується класом представлення `LongplayToListView`, яке вимагає авторизації попередньо і реалізовано на основі базового класу представлення `View`. Підтримка цієї функціональності відбувається через створення методу `post`, в якому за параметром `request` за ключем (параметром) `longplaylist` витягається з БД відповідний об'єкт переліку альбомів з цим номером. Тоді витягається альбом, номер якого відповідає параметру `pk`, переданого до методу `post`. За цим альбомом і переліком перевіряється, чи входить цей альбом вже до даного переліку, через модель `Longplaylistpart`. Якщо такий об'єкт є, то він вилучається і кількість переліків, до яких внесено цей альбом, що зберігається в об'єкті самого альбому, зменшується на 1, після цього зміни фіксуються в БД. Якщо такого об'єкту немає, то він створюється, а кількість переліків, до яких внесено цей альбом, збільшується на 1, а тоді зміни також фіксуються в БД.

Збереження переліку альбомів іншого користувача реалізується через клас представлення `FollowLongplaylistView`. Це представлення, як і попередні, особливі тим, що вони не потребують шаблону вебсторінки, тобто вони призводять до виконання певних дій, про успішність виконання яких

відправляється відповідь. У даному класі основним є метод `post`, де витягається перелік альбомів за номером `pk` параметру методу, потім для даного переліку визначається, чи зберіг авторизований користувач-слухач цей перелік собі, для чого виконується витягання об'єкту моделі `Longplaylistfollower`. Якщо він є, то він вибуває з БД, а кількість підписників на цей перелік зменшується на 1. Якщо його немає, то він створюється і зберігається в БД, а кількість підписників на цей перелік збільшується на 1.

Шаблон вебсторінки `longplay_look.html` відповідає за підтримку пошуку альбомів, забезпечувану через клас представлення `LongplayLookupView`. Загалом клас представлення створений через наслідування класу представлення `ListView` для роботи з переліками. Самі результати пошуку визначаються через метод `get_queryset`, де формується сам набір даних за запитом. Передача параметрів виконується через параметри GET-запиту, де параметр `title` відповідає текстовій фразі, пошук якої відбувається, параметр `likes` відповідає мінімальній кількості уподобань альбому, параметр `ordering` визначає порядок впорядкування результатів, а параметр `origin` визначає номер країни походження музичного гурту, який створив альбом.

Пошук текстової фрази з параметру `title` відбувається через перевірку наявності її в назві музичного колективу, який створив цей альбом, в імені виконавця, або в назві альбому. Якщо хоча в одному з цих варіантів буде знайдений альбом, то він буде доданий до результатів пошуку. Після цього буде перевірено, чи цей альбом має не меншу ніж задану кількість уподобань та чи цей альбом був створений музичним гуртом з заданої країни походження, після чого весь набір впорядковано необхідним чином.

Шаблон вебсторінки `band_longplays.html` відповідає за відображення переліку альбомів заданого музичного колективу, використовуючи для цього клас представлення `BandLongplaysView`. Метод `get_queryset` виконує підготовку даних, спочатку за параметром `band` з заданої адреси вебсторінки

витягається номер музичного гурту, витягаються дані про нього з БД, а далі формується перелік всіх альбомів цього музичного гурту, впорядковуючи його за напрямком, заданим в параметрі `ordering`. Цей клас створений на основі класу представлення `ListView`, який надається самим Django. На основі даного класу побудовані всі подібні відображення переліків.

Шаблон вебсторінки `new_list_longplays.html` підтримує відображення нових альбомів за переліком, який відображається користувачу. Номер переліку передається через параметр `list` адреси вебсторінки. За цим переліком витягаються всі альбоми, які в нього входять через дані з моделі `Longplaylistpart` і які були опубліковані не раніше ніж 30 днів тому від поточного моменту.

Перелік усіх уподобаних альбомів формується через клас представлення `LikedLongplaysView`. Останні уподобані альбоми розташовуються в переліку раніше.

Особливість програми, поради, підтримуються через наступні вебсторінки. Створення порад забезпечується через клас представлення `LongplayadviceCreateView` і вебсторінку `new_longplay_advice.html`. Під час створення поради в методі `post` автоматично вноситься автор поради на основі даних авторизованого користувача-слухача. Також автоматично змінюється кількість порад за альбомом.

Забезпечення перегляду відправлених авторизованим користувачем-слухачем порад виконується на основі шаблону вебсторінки `longplay_advices.html` за підтримки його наповнення класом представлення `LongplayAdvicesView`. Усі об'єкти впорядковують за часом створення, тобто останні створені поради розташовуються раніше.

Забезпечення перегляду адресованих авторизованому користувачу-слухачу порад виконується на основі шаблону вебсторінки `longplay_listadvices.html` за підтримки його наповнення класом представлення `LongplayListenerAdvicesView`. Усі об'єкти впорядковують за часом створення, тобто останні створені поради розташовуються раніше.

4.3 Висновки за розділом 4

У четвертому розділі описано результати реалізації програми стосовно розробленого коду в складі відповідного програмного проєкту. Описано структуру програмного забезпечення, включаючи сам вебдодаток і файли, пов'язані з реалізацією методу, представлено основні реалізовані класи з описом програмної логіки.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

5.1 Опис роботи з програмою

Робота з програмою неавторизованих користувачів, користувачів-слухачів, а загалом і менеджерів сервісу є інтуїтивно зрозумілою. Для неавторизованого користувача доступна дуже обмежена кількість розділів, що включають тільки розділ «BEST», звідки надається перелік найкращих альбомів, музичних композицій, музичних гуртів за різними критеріями (кількість прослуховувань, уподобань і т.д.) і переліки, створені самим сервісом для користувачів, та розділ «Пошук», звідки можна виконати пошук потрібного альбому, виконавця, музичного гурту або композиції. Для визначення варіанту пошуку потрібно обрати відповідний варіант з переліку. У випадку пошуку альбому пошук також здійснюється з визначенням альбомів, які могли бути створені виконавцем або гуртом, назву якого вказано, тобто заданий текст в пошуку використовується для пошуку не тільки в самій назві альбому.

Після авторизації користувача-слухача одразу кількість розділів розширюється. Серед розділів з'являються:

- розділ «Переліки», що включає всі власні переліки користувача-слухача, для керування якими використовуються кнопки, які визначають альбоми, композиції і гурти: при натисканні відображається перелік створених переліків, а обравши конкретний, натиснувши на нього, відкривається сторінка з конкретним переліком альбомів, гуртів, композицій відповідно, звідси ж доступний перелік з улюбленими композиціями, альбомами, тобто всі ті композиції, альбоми, які були уподобані на всіх окремих сторінках раніше;

- розділ «Підписки», що включає всі оформлені підписки на інших користувачів-слухачів і на інші переліки інших користувачів-слухачів: доступні кнопки, що визначають альбоми, композиції, гурти, слухачів та

підписників (у цьому випадку відображається перелік користувачів-слухачів, які підписалися на авторизованого користувача);

– розділ «Штурханці», який містить поради прослухати альбом або композицію від іншого користувача-слухача та поради, які відправив авторизований користувач-слухач іншим користувачам для прослуховування альбомів або композицій: всього доступно дві групи кнопок, перша з яких визначає, чи це власні поради, чи це отримані поради, а друга відповідно – чи це поради альбомів чи музичних композицій.

Однією з основних одиниць впорядкування всіх цих елементів є сторінка музичного альбому (рис. 5.1). З цієї сторінки доступна назва музичного гурту (за назвою можна перейти на сторінку музичного гурту), назва альбому, перелік позначок, зображення. Над зображенням наявні кнопки для того, щоб додати альбом у перелік (тоді спливає зверху віконечко, де можна обрати потрібний перелік з власних переліків альбомів) або відправити пораду іншому користувачу прослухати цей альбом («Відправити штурханець»).

Нижче доступні іконки, які виводять відповідну інформацію про альбом загалом та дозволяють отримати розширену інформацію або змінити певну. Спочатку визначається кількість уподобань альбому: натискаючи на іконку, можна або додати, або вилучити з уподобань альбом, а натискаючи на кількість уподобань, можна відобразити спливаюче віконечко з посиланнями на сторінки користувачів, які уподобали цей альбом. Правіше розташована іконка з кількістю входжень альбому у переліки альбомів інших користувачів. Аналогічно, натискаючи на кількість, можна відтворити переліки, які містять цей альбом. Ще правіше розташована кількість порад, які були відправлені іншим користувачам відносно даного альбому. Самі поради не відображаються, тільки кількість. Останньою розташована сумарна кількість прослуховувань всіх музичних композицій альбому.

Нижче на сторінці приводиться перелік композицій з альбому, кожен з яких можна прослухати через відповідний відтворювач, а також за кожною

є можливість її одразу порадити або додати до власного переліку композицій. За кожною композицією доступна кількість уподобань, входження в переліки, порад та прослуховувань. Дії, які можна виконати, аналогічні діям, описаним вище для альбому.

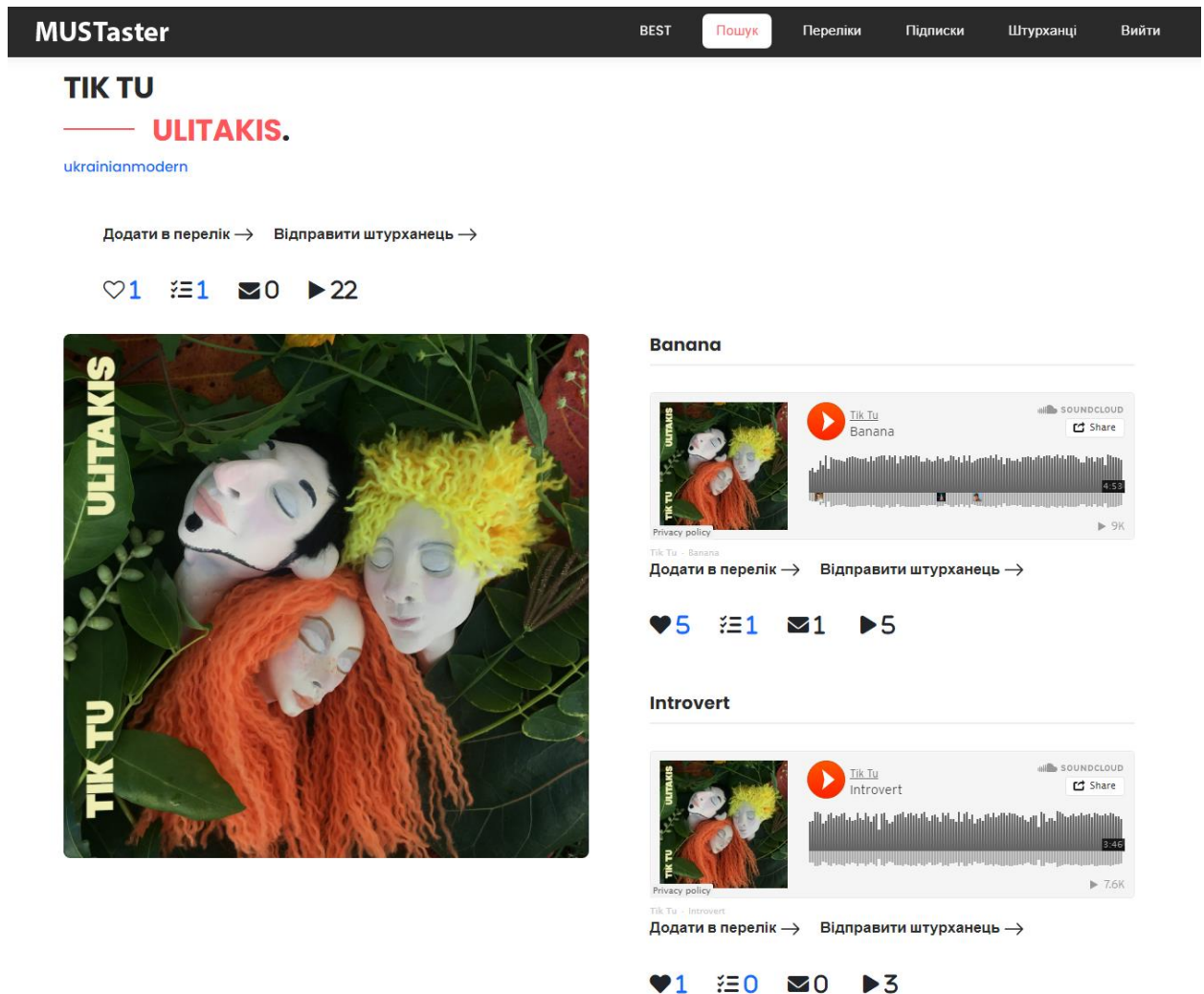


Рисунок 5.1 – Сторінка відтворення для слухача музичного альбому

Ще нижче на сторінці розташована інформація про дату видання альбому, про всіх виконавців, які брали участь в записі альбому.

Робота з порадами відбувається через розділ «Штурханці». Як було описано вище, спочатку потрібно обрати режим роботи, а нижче при цьому відображається перелік відповідних порад. Для кожної поради (рис. 5.2) доступне або посилання на сторінку альбому, або відтворювач з

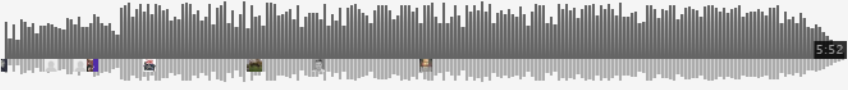
композицією, її назва, назва гурту, кількість уподобань, переліків, в яких вона міститься, порад та прослуховувань.

MUSTaster
BESTПошукПерелікиПідпискиШтурханціВийти

Dakh Daughters – Інше Місто




Dakh Daughters
Inshe Misto


5:52

 Share

45.7K

Privacy policy

Dakh Daughters · Inshe Misto

Додати В Перелік → Відправити Штурханець →

4.12.2022 Від AlexG

♥4 📋2 ✉1 ▶7

Христина Соловій – Про Весну




Khrystyna Soloviy
Христина Соловій - Про Весну


4:06

 Share

45.2K

Privacy policy

Khrystyna Soloviy · Христина Соловій - Про Весну

Додати В Перелік → Відправити Штурханець →

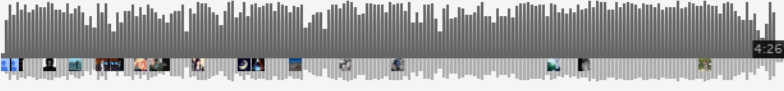
4.12.2022 Від AlexG

♥5 📋1 ✉1 ▶6

Бумбокс – Люди (Акустична Версія)



mfmlabel
БУМБОКС - ЛЮДИ (Acoustic Version)


4:26

 Share

171K

Privacy policy

mfmlabel · БУМБОКС - ЛЮДИ (Acoustic Version)

Додати В Перелік → Відправити Штурханець →

3.12.2022 Від Khrystyа

♥1 📋0 ✉1 ▶1

Рисунок 5.2 – Сторінка перегляду порад прослуховування композицій, направлених авторизованому користувачу-слухачу

Користувачу надається можливість додати композицію або альбом до свого переліку, відправити пораду іншому користувачу.

Окрім всього цього також є інформація про те, коли ця порада була створена і ким. Це дозволяє перейти до сторінки відповідного користувача-слухача. Також на сторінку користувача-слухача можна перейти зі сторінки переліку, де він вказується як автор, якщо цей перелік доступний публічно, а також з переліку уподобань композиції, альбому і т.д. Після переходу на сторінку користувача-слухача можна або підписатися, або відписатися від нього, а також переглянути його переліки.

Менеджер сервісу має той самий перелік розділів, але до нього також додається розділ «Каталог», звідки він може керувати даними композицій, альбомів і т.д. Звідси доступні кнопки, які дозволяють обрати потрібний елемент. Також менеджеру зі сторінок кожного альбому, виконавця і т.д. (куди наприклад, можна перейти з пошуку) доступна кнопка редагування відповідних даних.

За кожним альбомом менеджеру відображається така ж інформація, що і користувачу-слухачу, але перед цим відображається інформація про виконаний прогноз (рис. 5.3). Одразу згори вказано, чи очікується, що цей альбом стане популярним, чи ні. А нижче вказано набір всіх оцінок альбому за кожним критерієм. Для кожного критерію вказано три значення, що відповідають мінімальному, потім середньому, а в кінці – максимальному значенню. Це не стосується модальності. Зі стандартної інформації нижче менеджер може дізнатися про кількість прослуховувань, уподобань альбому, якщо альбом вже пробув на платформі достатню кількість часу. Але інформація про популярність альбому доступна тільки менеджеру, звичайним користувачам-слухачам вона не доступна, так само як і значення з критеріями альбому та окремих композицій, які доступні менеджеру. Значення за кожним критерієм для кожної композиції відображаються менеджеру біля відтворювача для прослуховування цієї композиції.

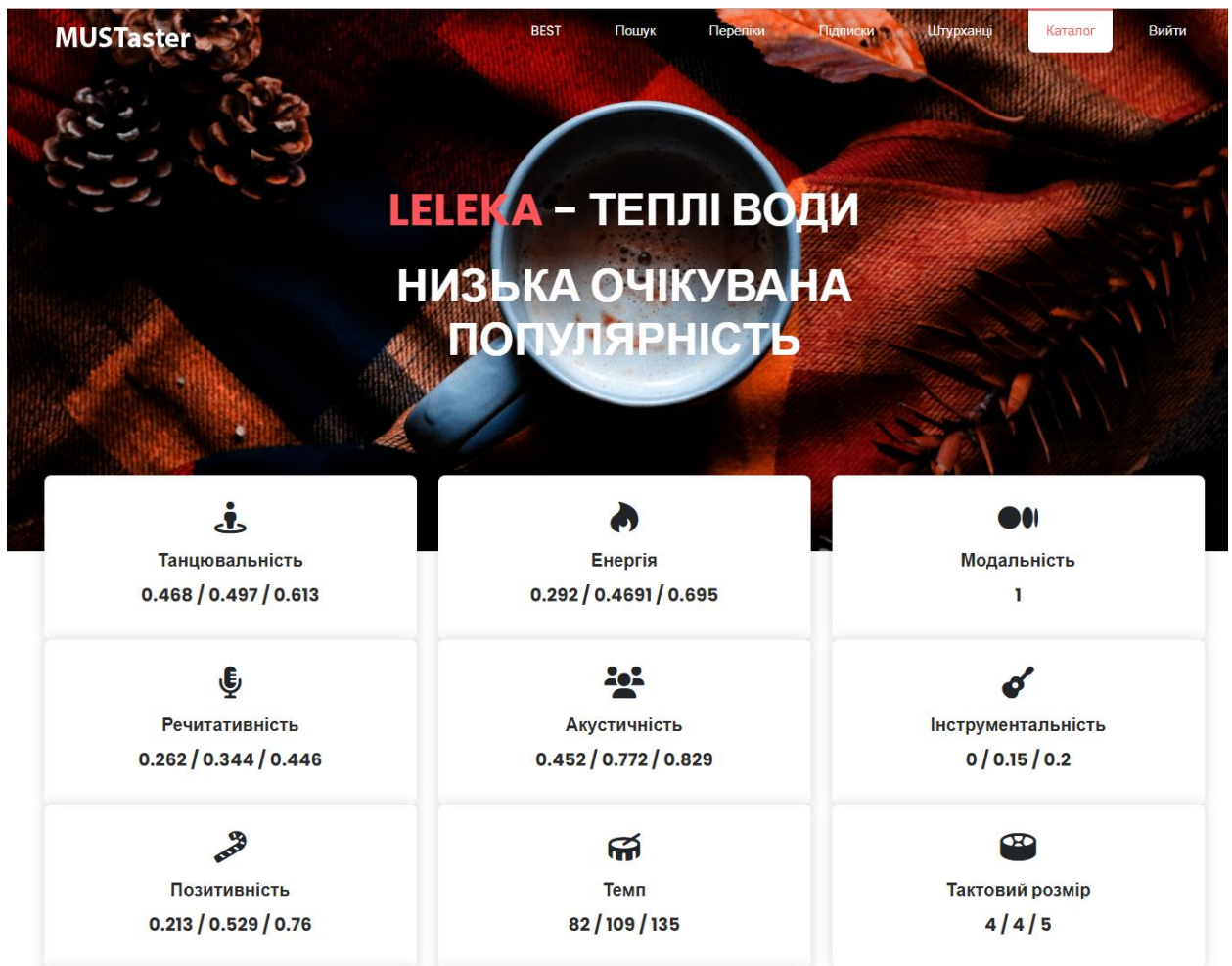


Рисунок 5.3 – Сторінка альбому для менеджера сервісу

5.2 Експериментальне дослідження

Експериментальне дослідження поведено на базі вибірки даних з джерела [8], де було зібрано дані з сервісу потокового прослуховування музики Spotify. Вибірka складає більше 181 тисячі екземплярів. Кожен екземпляр характеризує музичний альбом. Дані музичного альбому включають:

- ідентифікатор альбому;
- назву альбому;
- дату видання альбому;
- перелік виконавців альбому;
- кількість музичних композицій на альбомі;

- назви композицій;
- тривалість композицій;
- три значення танцювальності;
- три значення енергії;
- три значення тональності;
- три значення модальності;
- три значення речитативності;
- три значення акустичності;
- три значення інструментальності;
- три значення виконання наживо;
- три значення позитивності;
- три значення темпу;
- три значення тактового розміру;
- популярність альбому.

Перед визначенням набору ознак, які використовуються в моделі всередині запропонованого методу, було вивчено вибірку даних, зокрема проведено дослідження залежності розподілу частки популярних і непопулярних альбомів серед всіх альбомів за різного рівня відповідних ознак. Отримані результати лягли в основу методу. Наприклад, можна побачити, що форма функції залежності частки популярних і непопулярних альбомів від значень середньої акустичності (рис. 5.4) абсолютно відрізняється від форми функції залежності частки популярних і непопулярних альбомів від значень середньої танцювальності (рис. 5.5), що говорить про те, що кожен з цих показників по-різному впливає на загальний результат і відповідно є важливим для визначення класу популярності альбомів.

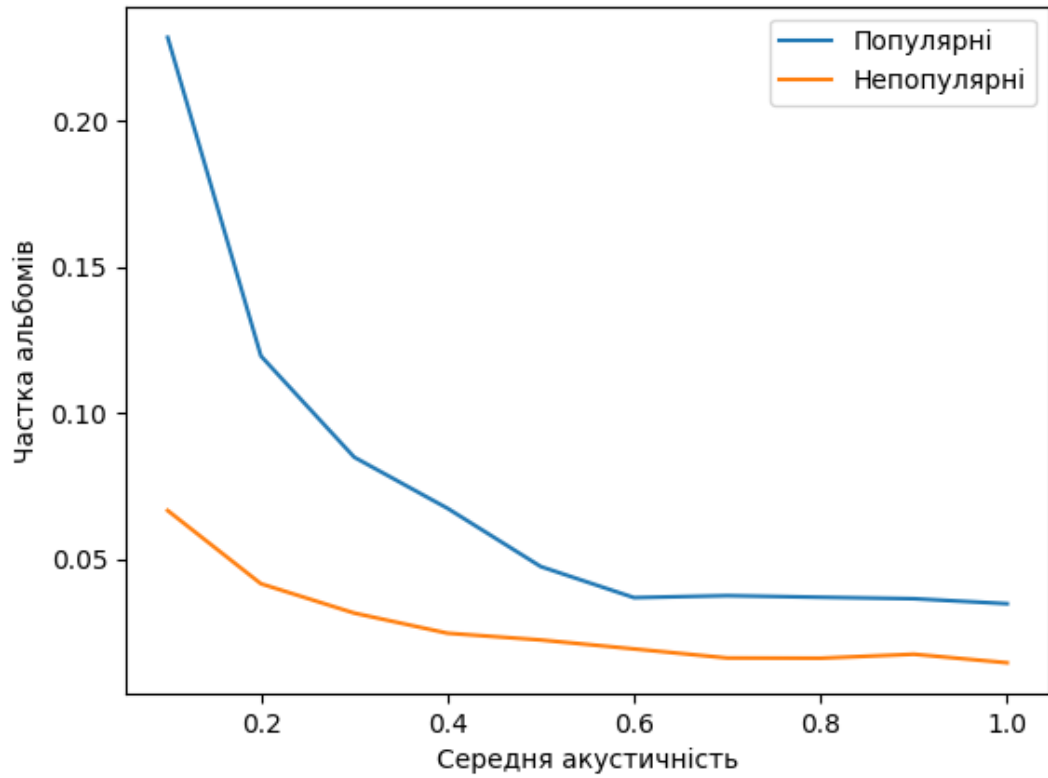


Рисунок 5.4 – Графік залежності частки популярних і непопулярних альбомів від значень середньої акустичності

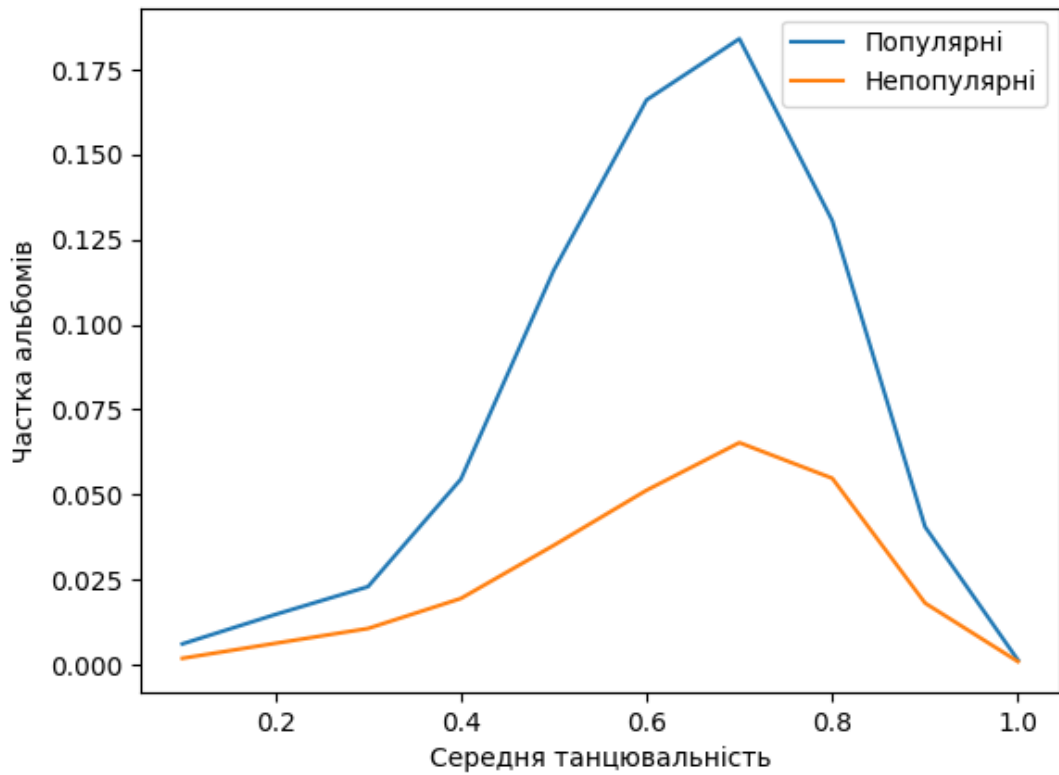


Рисунок 5.5 – Графік залежності частки популярних і непопулярних альбомів від значень середньої танцювальності

Експериментальне дослідження проводилось, порівнюючи власний метод визначення популярності альбомів музичних виконавців та метод k-найближчих сусідів. Якщо метод k-найближчих сусідів призвів до отримання похибки в 37,29 %, то запропонований метод – 15,71 %, що значно менше і окрім того є прийнятним для практики результатом (табл. 5.1).

Таблиця 5.1 – Результати експериментального дослідження

Метод / модель	Похибка, %
Метод k-найближчих сусідів	37,29
Метод визначення популярності альбомів музичних виконавців	15,71

5.3 Висновки за розділом 5

У п'ятому розділі виконано опис результируючих можливостей програмного забезпечення та процедур використання цих можливостей під час роботи з програмним забезпеченням. Експериментальне дослідження виконано на вибірці даних за сервісом Spotify з отриманням похибки класифікації в 15,71 % у запропонованого методу, що менше ніж в 2 рази ніж у методу k-найближчих сусідів.

ВИСНОВКИ

За результатами виконання роботи розроблено програмне забезпечення прослуховування музичних альбомів і композицій для впорядкування даного процесу зі сторони слухачів і підтримки менеджерів з авторами альбомів. Програму створено на основі використання мови Python, бібліотеки scikit-learn, фреймворку веброзробки Django, системи керування базами даних MySQL.

Представлене за результатами роботи програмне забезпечення дозволяє надавати слухачам інформацію про альбоми, композиції, музичні гурти, шукати їх, об'єднувати їх у власні переліки, підписуватися на переліки інших слухачів, ділитися персональними порадами, висловлювати власні уподобання композиціям і альбомам, визначати популярність альбомів. Перелік уподобаних композицій, альбомів може переглядати тільки користувач-слухач, який виконав ці уподобання. Для кожного переліку композицій, гуртів або альбомів автор може визначити рівень доступу, який може включати публічний доступ, доступ тільки для власних підписників або відсутність доступу неавторів. Передбачено створення переліків від імені сервісу.

Наукова новизна роботи сформована в методі визначення популярності альбомів музичних виконавців, який виконує класифікацію музичних альбомів на популярні і непопулярні, застосовуючи алгоритм ID3 на основі групи ознак, що характеризують музичні композиції в альбомі за різними критеріями, що дозволяє приймати рішення стосовно керування маркетинговою компанією альбому до його видання. Проведено експериментальне дослідження запропонованого методу, що підтвердило можливість його застосування на практиці.

Виконання роботи було побудовано на основі сформульованого переліку завдань, кожен пункт якого було виконано.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Clark, B. What's the Difference Between a Single, EP, and LP? [Electronic resource] / B. Clark. – Access mode : <https://www.musicianwave.com/difference-between-single-ep-lp/>.
2. Blake, A. Who Listens to Albums Anymore? [Electronic resource] / A. Blake. – Access mode : <https://www.rollingstone.com/music/music-features/who-listens-to-albums-anymore-1082287/>.
3. Razat, M. 5 Reasons to Listen to Full Albums [Electronic resource] / M. Razat. – Access mode : <https://www.lifehack.org/326997/5-reasons-listen-full-albums>.
4. Roberts, R. The lost art of deep listening [Electronic resource] : Choose an album. Lose the phone. Close your eyes / R. Roberts. – Access mode : <https://www.latimes.com/entertainment-arts/music/story/2020-03-17/coronavirus-deep-listening-music-albums>.
5. Modern Listeners Shifting Towards Playlists Over Albums Study Shows [Electronic resource]. – Access mode : <https://mn2s.com/news/label-services/modern-listeners-shifting-towards-playlists-over-albums-study-shows/>.
6. Samuelson, K. Americans Listening to Playlists Over Albums, Study Finds [Electronic resource] / K. Samuelson. – Access mode : <https://time.com/4505600/playlists-albums-loop-music-business/>.
7. Web API Reference. Spotify for Developers [Electronic resource] : Get Tracks' Audio Features. – Access mode : <https://developer.spotify.com/documentation/web-api/reference/#/operations/get-several-audio-features>.
8. Music Albums Popularity Prediction [Electronic resource]. – Access mode : <https://www.kaggle.com/datasets/elemento/music-albums-popularity-prediction?select=test.csv>.
9. Alexis, M. What is the Spotify Popularity Index? [Electronic resource] / M. Alexis. – Access mode : <https://twostorymelody.com/spotify-popularity-index/>.

10. Eichler, O. Spotify Popularity – A unique insight into the Spotify algorithm and how to influence it [Electronic resource] / O. Eichler. – Access mode : <https://lab.songstats.com/spotify-popularity-a-unique-insight-into-the-spotify-algorithm-and-how-to-influence-it-93bb63863ff0>.

11. Robley, C. The Spotify Algorithm [Electronic resource] : What Musicians Need to Know / C. Robley. – Access mode : <https://diymusician.cdbaby.com/music-career/spotify-algorithm/#:~:text=Spotify's%20Popularity%20index%20is%20a,get%20you%20onto%20Discover%20Weekly>.

12. What is the Spotify Popularity Index? [Electronic resource]. – Access mode : <https://lodgecove.com/what-is-the-spotify-popularity-index/>.

13. Spotify – вебпрогравач [Electronic resource]. – Access mode : <https://open.spotify.com/>.

14. Discover the top streamed music and songs online on Soundcloud [Electronic resource]. – Access mode : <https://soundcloud.com/discover>.

15. Sakka, Y. Decision Trees [Electronic resource] : ID3 Algorithm Explained / Y. Sakka. – Access mode : <https://towardsdatascience.com/decision-trees-for-classification-id3-algorithm-explained-89df76e72df1>.

16. Decision Tree [Electronic resource] : ID3 Algorithm. – Access mode : <https://www.programsbuzz.com/article/decision-tree-id3-algorithm>.

ДОДАТОК А
Технічне завдання

Вступ

Застосування інформаційних технологій для прослуховування музики свого часу змінило дану галузь і сприяло її подальшому активному розвитку. Але одним з етапів цього розвитку є збільшення аудіо-даних, серед яких важко знайти ту музику, яка зацікавить кожного користувача особисто, а окрім того важко визначити ще й, на скільки буде популярним той чи інший альбом виконавця. Саме на ці особливості має бути зорієнтована розробка в цій роботі.

A.1 Підстави для розробки

Розробка дипломної кваліфікаційної роботи магістра виконується на основі завдання за темою «Дослідження та програмна реалізація методів визначення популярності альбомів музичних виконавців», затвердженою наказом № 363 від 4 листопада 2022 р. за Національним університетом «Запорізька політехніка».

A.2 Призначення розробки

Програма призначена для упорядкування альбомів, композицій і музичних гуртів для зручного прослуховування, обміну інформацією про музику між людьми з однаковими інтересами, підтримки викладання нових музичних матеріалів і прогнозування популярності альбомів.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Для забезпечення власного призначення програма повинна дозволяти виконувати функції:

- перегляд композицій альбому;
- збереження альбому до власного переліку;
- створення власного переліку альбомів;
- вилучення альбому з власного переліку альбомів;
- прослуховування композиції;
- збереження композиції до власного переліку;
- створення власного переліку композицій;
- вилучення композиції з власного переліку композицій;
- збереження музичного гурту до власного переліку;
- створення власного переліку музичних гуртів;
- вилучення музичного гурту з власного переліку гуртів;
- перегляд сторінки музичного гурту;
- перегляд переліку композицій;
- перегляд переліку музичних гуртів;
- перегляд переліку альбомів;
- перегляд власних переліків композицій;
- перегляд переліків композицій іншого користувача-слухача;
- перегляд власних переліків альбомів;
- перегляд переліків альбомів іншого користувача-слухача;
- перегляд власних переліків музичних гуртів;
- перегляд переліків музичних гуртів іншого користувача-слухача;
- перегляд підписаних переліків композицій;
- перегляд підписаних переліків альбомів;
- перегляд підписаних переліків музичних гуртів;
- відправлення поради користувачу-слухачу прослухати альбом;
- відправлення поради користувачу-слухачу прослухати музичну композицію;
- відтворення адресованих порад прослухати музичні композиції;
- відтворення адресованих порад прослухати альбоми;
- відтворення власних порад прослухати музичні композиції;

- відтворення власних порад прослухати альбоми;
- відтворення переліку найкращих композицій;
- відтворення переліку найкращих альбомів;
- відтворення переліку найкращих музичних гуртів;
- реєстрація, авторизація користувачів-слухачів;
- авторизація менеджерів сервісу;
- публікація переліків альбомів авторства сервісу;
- публікація переліків композицій авторства сервісу;
- встановлення рівня доступу до власного переліку композицій;
- встановлення рівня доступу до власного переліку музичних гуртів;
- встановлення рівня доступу до власного переліку альбомів;
- уподобання альбому;
- уподобання музичної композиції;
- перегляд сторінки виконавця;
- перегляд альбомів виконавця;
- перегляд альбомів музичного гурту;
- пошук музичних гуртів;
- пошук музичних виконавців;
- пошук музичних композицій;
- пошук альбомів;
- збереження переліку композицій іншого користувача-слухача;
- збереження переліку альбомів іншого користувача-слухача;
- збереження переліку музичних гуртів іншого користувача-слухача;
- відтворення сторінок підписаних користувачів-слухачів;
- підписування на користувача-слухача;
- відтворення переліку підписників користувача-слухача;
- відтворення нових альбомів музичних гуртів з переліку;
- відтворення переліку власних уподобаних композицій;
- відтворення переліку власних уподобаних альбомів;
- скасування уподобання альбому;

- скасування уподобання композиції;
- додавання музичного гурту;
- редагування даних музичного гурту;
- публікація альбому;
- редагування даних альбому;
- публікація композицій;
- редагування композицій;
- публікація інформації про виконавця;
- редагування інформації виконавця;
- перегляд сторінки користувача-слухача;
- пошук користувачів-слухачів;
- перегляд каталогу виконавців;
- перегляд каталогу композицій;
- перегляд каталогу музичних гуртів;
- перегляд каталогу альбомів.

А.3.2 Вимоги до складу та параметрів технічних засобів

Доступ з клієнтського пристрою користувача, який виконує будь-яку роль у вебдодатку, виконується за наявності підключення до мереж інтернет та за наявності актуальної версії будь-якого сучасного браузера.

Для запуску вебсервера, який буде підтримувати роботу вебдодатку безпосередньо, зокрема запускаючи і виконуючи метод, який має бути запропонований в роботі, необхідно скористатися послугами хостингу з можливостями підтримки роботи з системою керування БД MySQL і запуску скриптів за розкладом, або підтримувати власний вебсервер, при цьому він повинен бути з оперативною пам'яттю з мінімальним обсягом 4 Гб, жорстким диском з мінімальним обсягом 1 Гб та щонайменше чотирьохядерним процесором (частота не менше 2 ГГц).

A.3.3 Умови експлуатації

Для розгортання вебдодатку на сервері з подальшим доступом користувачів знадобиться вебсервер. В якості стандартного рішення даної проблеми для додатків, розроблених за допомогою Django, використовується uWSGI. Вебсервер використовується як шлюз і фактично перенаправляє запити, які надходять до нього з клієнтських пристроїв, у Django-додаток.

A.3.4 Вимоги до програмної документації

Технічне завдання і текст програми, що повинен вміщувати фрагменти програми, оформлюються як окремі програмні документи, а опис роботи з програмою представляється в розділі 5 пояснювальної записки.

A.4 Порядок контролю та приймання

Контроль і приймання роботи мають відбуватися за календарним планом, який входить до складу завдання на дипломну кваліфікаційну роботу магістра.

ДОДАТОК Б
Текст програми

Б.1 Текст файла lp_view.py

```

from django.shortcuts import get_object_or_404
from django.contrib.auth.mixins import LoginRequiredMixin
from django.http import HttpResponse
from django.shortcuts import render, redirect
from django.views.generic import DetailView, ListView
from django.views import View
from mustasterapp.forms import LongplayForm,
LongplayLookForm, LongplayadviceForm, LongplaylistForm
    from mustasterapp.models import Longplaylike, Listenuser,
Longplaylist, Longplaylistpart, Longplaylistfollower, Origin,
Musband, Performer
    from django.urls import reverse
    from django.db.models import Q
    import datetime
    import timedelta

class LongplayView(DetailView):
    model = Longplay
    template_name = 'mustasterapp/longplay.html'

    def get(self, request, pk):
        longplayer = Longplay.objects.
select_related('performers').select_related('marks').get(id =
pk)
        lplikes = Longplaylike.objects.filter(longplay =
longplayer)
        listener_user = self.request.user
        listen_user = Listenuser.objects.get(user =
listener_user)
        liked = False
        if lplikes.filter(listenuser = listen_user):
            liked = True
        longplay_lists =
Longplaylist.objects.filter(listenuser = listen_user)
        for lplay in longplay_lists:
            part =
Longplaylistpart.objects.filter(longplaylist = lplay, longplay =
longplayer)
            if part:
                lplay.inlist = True
            else:
                lplay.inlist = False
        longplay_data = {}

```

```

        longplay_data['longplay'] = longplayer
        longplay_data['lplikes'] = lplikes
        longplay_data['liked'] = liked
        longplay_data['lists'] = longplay_lists
        return render(request, self.template_name,
longplay_data)

class LongplaylistDetailView(DetailView):
    model = Longplay
    template_name = 'mustasterapp/longplaylist.html'

    def get(self, request, pk):
        longplaylist = Longplaylist.objects.get(id = pk)
        listener_user = self.request.user
        listen_user = Listenuser.objects.get(user =
listener_user)
        luf = Listenuserfollower.objects.get(listenuser =
longplaylist.listenuser, follower = listen_user)
        if (longplaylist.accessmode == 1 and luf) or
longplaylist.accessmode == 0 or longplaylist.accessmode == 3 or
(longplaylist.accessmode == 2 and listen_user ==
longplaylist.listenuser):
            parts =
Longplaylistpart.objects.filter(longplaylist = longplaylist)
            longplays = []
            for part in parts:
                longplays.append(part.longplay.id)
            longplays = Longplay.objects.filter(id__in =
longplays)
            ordering =
int(self.request.GET.get('ordering'))
            if ordering == 0:
                longplays = longplays.order_by('-inlikes')
            elif ordering == 1:
                longplays = longplays.order_by('-inlists')
            elif ordering == 2:
                longplays = longplays.order_by('-
inadvices')
            elif ordering == 3:
                longplays = longplays.order_by('-
inlistenings')
            elif ordering == 4:
                longplays = longplays.order_by('-
popularity')
            elif ordering == 5:
                longplays = longplays.order_by('-release')

```

```

        else:
            longplaylist = None
            longplays = []
            longplaylist_data = {}
            longplaylist_data['longplaylist'] = longplaylist
            longplaylist_data['longplays'] = longplays
            return render(request, self.template_name,
longplaylist_data)

class LongplayCreateView(LoginRequiredMixin, View):
    template_name = 'mustasterapp/new_longplay.html'

    def get(self, request, pk = None):
        LPform = LongplayForm()
        lp_data = {'lpform': LPform }
        return render(request, self.template_name, lp_data)

    def post(self, request, pk = None):
        LPform = LongplayForm (request.POST, request.FILES
or None)
        if not LPform.is_valid():
            lp_data = {'lpform': LPform }
            return render(request, self.template_name,
lp_data)

        lp = LPform.save(commit = False)
        lp.save()
        return redirect(reverse('mustasterapp:bandlps',
args = [lp.musband.id]))

class LongplayUpdateView (LoginRequiredMixin, View):
    template_name = 'mustasterapp/new_longplay.html'

    def get(self, request, pk):
        longplayer = get_object_or_404(Longplay, id = pk)
        LPform = LongplayForm (instance = longplayer)
        lp_data = {'lpform': LPform }
        return render(request, self.template_name, lp_data)

    def post(self, request, pk):
        longplayer = get_object_or_404(Longplay, id = pk)
        LPform = LongplayForm (request.POST, request.FILES
or None, instance = longplayer)
        if not LPform.is_valid():
            lp_data = {'lpform': LPform }
            return render(request, self.template_name,
lp_data)

```

```

        lp = LPform.save(commit = False)
        lp.save()
        return redirect(reverse('mustasterapp:bandlps',
args = [lp.musband.id]))

    @method_decorator(csrf_exempt, name='dispatch')
    class LongplayLikeView(LoginRequiredMixin, View):

        def post(self, request, pk):
            longplayer = get_object_or_404(Longplay, id = pk)
            listener_user = self.request.user
            listen_user = Listenuser.objects.get(user =
listener_user)
            lplike = Longplaylike.objects.get(longplay =
longplayer, listenuser = listen_user)
            if lplike:
                lplike.delete()
                longplayer.inlikes = longplayer.inlikes - 1
                longplayer.save()
            else:
                lplike = Longplaylike(longplay = longplayer,
listenuser = listen_user)
                lplike.save()
                longplayer.inlikes = longplayer.inlikes + 1
                longplayer.save()
            return HttpResponse()

    class LongplayLookView(ListView):
        model = Longplay
        template_name = 'mustasterapp/longplay_look.html'

        def get_queryset(self, *args, **kwargs):
            longplays = super().get_queryset(*args, **kwargs)
            title_phrase = str(self.request.GET.get('title'))
            if title_phrase:
                bands =
Musband.objects.filter(musname__icontains = title_phrase)
                cond_for_bands = Q(musband__in = bands)
                cond_for_title = Q(lptitle__icontains =
title_phrase)
                if title_phrase:
                    performers =
Performer.objects.filter(appname__icontains = title_phrase)
                    cond_for_performers = Q(musbandperformers__in =
performers)

```

```

        longplays = longplays.filter(cond_for_bands |
cond_for_title | cond_for_performers)
        like_count = int(self.request.GET.get('likes'))
        longplays = longplays.filter(inlikes__gte =
like_count)
        origin_id = int(self.request.GET.get('origin'))
        if origin_id:
            origins = Origin.objects.get(id = origin_id)
            longplays = longplays.filter(musband_origin =
origins)
        ordering = int(self.request.GET.get('ordering'))
        if ordering == 0:
            longplays = longplays.order_by('-inlikes')
        elif ordering == 1:
            longplays = longplays.order_by('-inlists')
        elif ordering == 2:
            longplays = longplays.order_by('-inadvices')
        elif ordering == 3:
            longplays = longplays.order_by('-inlistenings')
        elif ordering == 4:
            longplays = longplays.order_by('-popularity')
        elif ordering == 5:
            longplays = longplays.order_by('-release')
        return longplays

    def get_context_data(self, **kwargs):
        look_data = super(LongplayLookView,
self).get_context_data(**kwargs)
        look_data['lp_look'] = LongplayLookForm (initial={
            'title': self.request.GET.get('title', ''),
            'likes': self.request.GET.get('likes', '0'),
            'ordering': self.request.GET.get('ordering',
'0'),
            'origin': self.request.GET.get('origin', '')
        })
        return look_data

    @method_decorator(csrf_exempt, name='dispatch')
    class LongplayToListView(LoginRequiredMixin, View):

        def post(self, request, pk):
            longplaylist_id = request.data.get("longplaylist")
            lpl = Longplaylist.objects.get(id =
longplaylist_id)
            longplayer = get_object_or_404(Longplay, id = pk)

```

```

        lp_lpart = Longplaylistpart.objects.get(longplaylist
= lp1, longplay = longplayer)
        if not lp_lpart:
            lp_lpart = Longplaylistpart(longplaylist = lp1,
longplay = longplayer)
        try:
            lp_lpart.save()
            longplayer.inlists = longplayer.inlists + 1
            longplayer.save()
        except IntegrityError as ierr:
            pass
    else:
        lp_lpart.delete()
        longplayer.inlists = longplayer.inlists - 1
        longplayer.save()
    return HttpResponse()

```

```

class LongplaylistCreateView(LoginRequiredMixin, View):
    template_name = 'mustasterapp/new_longplay_list.html'

    def get(self, request, pk = None):
        LPform = LongplaylistForm()
        lp_data = {'lpform': LPform }
        return render(request, self.template_name, lp_data)

    def post(self, request, pk = None):
        LPform = LongplaylistForm (request.POST,
request.FILES or None)
        if not LPform.is_valid():
            lp_data = {'lpform': LPform }
            return render(request, self.template_name,
lp_data)

        lp = LPform.save(commit = False)
        listener_user = self.request.user
        listen_user = Listenuser.objects.get(user =
listener_user)
        lp.listenuser = listen_user
        lp.save()
        return redirect(reverse('mustasterapp:bandlps',
args = [lp.musband.id]))

```

```

class LongplaylistUpdateView (LoginRequiredMixin, View):
    template_name = 'mustasterapp/new_longplay_list.html'

```

```

def get(self, request, pk):
    longplaylist = get_object_or_404(Longplaylist, id =
pk)

    LPform = LongplaylistForm (instance = longplaylist)
    lp_data = {'lpform': LPform }
    return render(request, self.template_name, lp_data)

def post(self, request, pk = None):
    longplaylist = get_object_or_404(Longplaylist, id =
pk)

    LPform = LongplaylistForm (request.POST,
request.FILES or None, instance = longplaylist)
    if not LPform.is_valid():
        lp_data = {'lpform': LPform }
        return render(request, self.template_name,
lp_data)

    lpl = LPform.save(commit = False)
    lpl.save()
    return redirect(reverse('mustasterapp:bandlps',
args = [lp.musband.id]))

@method_decorator(csrf_exempt, name='dispatch')
class FollowLongplaylistView(LoginRequiredMixin, View):

    def post(self, request, pk):
        longplaylist_id = request.data.get("longplaylist")
        lpl = Longplaylist.objects.get(id = pk)
        listener_user = self.request.user
        listen_user = Listenuser.objects.get(user =
listener_user)

        lplfollow =
Longplaylistfollower.objects.get(longplaylist = lpl, listenuser
= listen_user)

        if not lplfollow:
            lplfollow = Longplaylistfollower (longplaylist
= lpl, listenuser = listen_user)
            try:
                lplfollow.save()
                lpl.infollowers = lpl.infollowers + 1
                lpl.save()
            except IntegrityError as ierr:
                pass
        else:
            lplfollow.delete()
            lpl.infollowers = lpl.infollowers - 1
            lpl.save()

```

```

        return HttpResponse()

class BandLongplaysView(ListView):
    model = Longplay
    template_name = 'mustasterapp/band_longplays.html'

    def get_queryset(self, *args, **kwargs):
        longplays = super().get_queryset(*args, **kwargs)
        band_id = int(self.request.GET.get('band'))
        musband = get_object_or_404(Musband, id = band_id)
        longplays = longplays.filter(musband = musband)
        ordering = int(self.request.GET.get('ordering'))
        if ordering == 0:
            longplays = longplays.order_by('-inlikes')
        elif ordering == 1:
            longplays = longplays.order_by('-inlists')
        elif ordering == 2:
            longplays = longplays.order_by('-inadvices')
        elif ordering == 3:
            longplays = longplays.order_by('-inlistenings')
        elif ordering == 4:
            longplays = longplays.order_by('-popularity')
        elif ordering == 5:
            longplays = longplays.order_by('-release')
        elif ordering == 6:
            longplays = longplays.order_by('-lptitle')
        return longplays

class PerformerLongplaysView(ListView):
    model = Longplay
    template_name = 'mustasterapp/performer_longplays.html'

    def get_queryset(self, *args, **kwargs):
        longplays = super().get_queryset(*args, **kwargs)
        performer_id =
int(self.request.GET.get('performer'))
        performer = get_object_or_404(Performer, id =
performer_id)
        longplays =
longplays.filter(longplayperformers__contains = performer)
        ordering = int(self.request.GET.get('ordering'))
        if ordering == 0:
            longplays = longplays.order_by('-inlikes')
        elif ordering == 1:
            longplays = longplays.order_by('-inlists')
        elif ordering == 2:

```

```

        longplays = longplays.order_by('-inadvices')
    elif ordering == 3:
        longplays = longplays.order_by('-inlistenings')
    elif ordering == 4:
        longplays = longplays.order_by('-popularity')
    elif ordering == 5:
        longplays = longplays.order_by('-release')
    elif ordering == 6:
        longplays = longplays.order_by('-lptitle')
    return longplays

class MyLongplaylistsView(LoginRequiredMixin, ListView):
    model = Longplaylist
    template_name = 'mustasterapp/my_longplaylists.html'

    def get_queryset(self, *args, **kwargs):
        longplaylists = super().get_queryset(*args,
**kwargs)

        listener_user = self.request.user
        listen_user = Listenuser.objects.get(user =
listener_user)

        longplaylists =
Longplaylist.objects.filter(listenuser = listen_user)
        ordering = int(self.request.GET.get('ordering'))
        if ordering == 0:
            longplaylists =
longplaylists.order_by('lpltitle')
        elif ordering == 1:
            longplaylists =
longplaylists.order_by('happened')
        elif ordering == 2:
            longplaylists = longplaylists.order_by('-
infollowers')

        return longplaylists

class ListenerLongplaylistsView(ListView):
    model = Longplaylist
    template_name =
'mustasterapp/listener_longplaylists.html'

    def get_queryset(self, *args, **kwargs):
        longplaylists = super().get_queryset(*args,
**kwargs)

        listener_id = int(self.request.GET.get('listener'))
        listen_user = Listenuser.objects.get(id =
listener_id)

```

```

        longplaylists =
Longplaylist.objects.filter(listenuser = listen_user,
accessmode__in = [0, 1])
        ordering = int(self.request.GET.get('ordering'))
        if ordering == 0:
            longplaylists =
longplaylists.order_by('lplttitle')
        elif ordering == 1:
            longplaylists =
longplaylists.order_by('happened')
        elif ordering == 2:
            longplaylists = longplaylists.order_by('-
infollowers')
        listener_user2 = self.request.user
        listen_user2 = Listenuser.objects.get(user =
listener_user2)
        luf = Listenuserfollower.objects.get(listenuser =
listen_user, follower = listen_user2)
        accessed_longplaylists = []
        for longplaylist in longplaylists:
            if (longplaylist.accessmode == 1 and luf) or
longplaylist.accessmode == 0 or (longplaylist.accessmode == 3
and listen_user == longplaylist.listenuser):
                accessed_longplaylists.append(longplaylist)
        return accessed_longplaylists

    class FollowedLongplaylistsView(LoginRequiredMixin,
ListView):
        model = Longplaylist
        template_name =
'mustasterapp/followed_longplaylists.html'

        def get_queryset(self, *args, **kwargs):
            longplaylists = super().get_queryset(*args,
**kwargs)
            listener_user = self.request.user
            listen_user = Listenuser.objects.get(user =
listener_user)
            lplf = Longplaylistfollower.objects.get(listenuser
= listen_user)
            lplf = lplf.order_by('-happened')
            lpls = []
            for lpl in lplf:
                lpls.append(lpl.longplaylist)
            longplaylists = longplaylists.filter(id__in = lpls)
            ordering = int(self.request.GET.get('ordering'))

```

```

        if ordering == 0:
            longplaylists =
longplaylists.order_by('lplttitle')
        elif ordering == 1:
            longplaylists =
longplaylists.order_by('happened')
        elif ordering == 2:
            longplaylists = longplaylists.order_by('-
infollowers')
        return longplaylists

class BestLongplaysView(ListView):
    model = Longplay
    template_name = 'mustasterapp/best_longplays.html'

    def get_queryset(self, *args, **kwargs):
        longplays = super().get_queryset(*args, **kwargs)
        origin_id = int(self.request.GET.get('origin'))
        if origin_id:
            origins = Origin.objects.get(id = origin_id)
            longplays = longplays.filter(musband_origin =
origins)

        ordering = int(self.request.GET.get('ordering'))
        if ordering == 0:
            longplays = longplays.order_by('-inlikes')
        elif ordering == 1:
            longplays = longplays.order_by('-inlists')
        elif ordering == 2:
            longplays = longplays.order_by('-inadvices')
        elif ordering == 3:
            longplays = longplays.order_by('-inlistenings')
        elif ordering == 4:
            longplays = longplays.order_by('-popularity')
        elif ordering == 5:
            longplays = longplays.order_by('-release')
        elif ordering == 6:
            longplays = longplays.order_by('-lplttitle')
        return longplays[:100]

class LongplayadviceCreateView(LoginRequiredMixin, View):
    template_name = 'mustasterapp/new_longplay_advice.html'

    def get(self, request, pk = None):
        LPform = LongplayadviceForm()
        lp_data = {'lpform': LPform }
        return render(request, self.template_name, lp_data)

```

```

        def post(self, request, pk = None):
            LPform = LongplayadviceForm (request.POST,
request.FILES or None)
            if not LPform.is_valid():
                lp_data = {'lpform': LPform }
                return render(request, self.template_name,
lp_data)

            lp = LPform.save(commit = False)
            listener_user = self.request.user
            listen_user = Listenuser.objects.get(user =
listener_user)
            lp.listenuser = listen_user
            lp.save()
            longplayer = lp.longplay
            longplayer.inadvices = longplayer.inadvices + 1
            return redirect(reverse('mustasterapp:lpadvices',
args = [listen_user.id]))

    class LongplayAdvicesView(ListView):
        model = Longplayadvice
        template_name = 'mustasterapp/longplay_advices.html'

        def get_queryset(self, *args, **kwargs):
            advices = super().get_queryset(*args, **kwargs)
            listener_user = self.request.user
            listen_user = Listenuser.objects.get(user =
listener_user)
            advices = Longplayadvice.objects.filter(listenuser
= listen_user)
            advices = advices.order_by('-happened')
            return advices

    class LongplayListenerAdvicesView(ListView):
        model = Longplay
        template_name =
'mustasterapp/longplay_listadvices.html'

        def get_queryset(self, *args, **kwargs):
            longplays = super().get_queryset(*args, **kwargs)
            listener_user = self.request.user
            listen_user = Listenuser.objects.get(user =
listener_user)
            advices = Longplayadvice.objects.filter(addresser =
listener_user)
            advices = advices.order_by('-happened')
            longplay_ids = []

```

```

for advice in advices:
    longplay_ids.append(advice.longplay)
longplays.filter(id__in = longplay_ids)
ordering = int(self.request.GET.get('ordering'))
if ordering == 0:
    longplays = longplays.order_by('-inlikes')
elif ordering == 1:
    longplays = longplays.order_by('-inlists')
elif ordering == 2:
    longplays = longplays.order_by('-inadvices')
elif ordering == 3:
    longplays = longplays.order_by('-inlistenings')
elif ordering == 4:
    longplays = longplays.order_by('-popularity')
elif ordering == 5:
    longplays = longplays.order_by('-release')
elif ordering == 6:
    longplays = longplays.order_by('-lptitle')
return longplays

class NewLongplaysFromListView(ListView):
    model = Longplay
    template_name = 'mustasterapp/new_list_longplays.html'

    def get_queryset(self, *args, **kwargs):
        longplays = super().get_queryset(*args, **kwargs)
        longplaylist_id = int(self.request.GET.get('list'))
        lpl = get_object_or_404(Longplaylist, id =
longplaylist_id)
        new_time = datetime.datetime.now()
        new_time = new_time - timedelta(days = 30)
        parts =
Longplaylistpart.objects.filter(longplaylist = lpl,
longplay_release__gte = new_time)
        longplayids = []
        for part in parts:
            longplayids.append(part.longplay)
        longplays = longplays.filter(id__in = longplayids)
        ordering = int(self.request.GET.get('ordering'))
        if ordering == 0:
            longplays = longplays.order_by('-release')
        elif ordering == 1:
            longplays = longplays.order_by('-inlikes')
        elif ordering == 2:
            longplays = longplays.order_by('-inadvices')
        elif ordering == 3:

```

```

        longplays = longplays.order_by('-inlistenings')
    elif ordering == 4:
        longplays = longplays.order_by('-popularity')
    elif ordering == 5:
        longplays = longplays.order_by('-release')
    return longplays

class ServiceLongplaylistsView(ListView):
    model = Longplaylist
    template_name =
'mustasterapp/service_longplaylists.html'

    def get_queryset(self, *args, **kwargs):
        longplaylists = super().get_queryset(*args,
**kwargs)

        longplaylists =
Longplaylist.objects.filter(accessmode = 3)
        longplaylists = longplaylists.order_by('-happened')
        ordering = int(self.request.GET.get('ordering'))
        if ordering == 0:
            longplaylists =
longplaylists.order_by('lplttitle')
        elif ordering == 1:
            longplaylists = longplaylists.order_by('-
happened')
        elif ordering == 2:
            longplaylists = longplaylists.order_by('-
infollowers')
        return longplaylists

class LikedLongplaysView(ListView):
    model = Longplay
    template_name = 'mustasterapp/liked_longplays.html'

    def get_queryset(self, *args, **kwargs):
        longplays = super().get_queryset(*args, **kwargs)
        listener_user = self.request.user
        listen_user = Listenuser.objects.get(user =
listener_user)
        longplaylikes =
Longplaylike.objects.filter(listenuser = listen_user)
        longplaylikes = longplaylikes.order_by('-happened')
        longplays = []
        for lplike in longplaylikes:
            longplays.append(lplike.longplay)
        return longplays

```

Б.2 Текст файла band_view.py

```

from django.shortcuts import get_object_or_404
from django.contrib.auth.mixins import LoginRequiredMixin
from django.shortcuts import render, redirect
from django.views.generic import DetailView, ListView
from django.views import View
from django.urls import reverse
from django.http import HttpResponse
from mustasterapp.models import Musband, Longplay,
Longplaylist, Listenuserfollower, Musbandlist, Musbandlistpart,
Musbandlistfollower
    from mustasterapp.forms import MusbandForm,
MusbandLookForm, MusbandlistForm

class MusbandView(DetailView):
    model = Musband
    template_name = 'mustasterapp/musband.html'

    def get(self, request, pk):
        musband = Musband.objects.
select_related('performers').get(id = pk)
        lps = Longplay.objects.filter(musband = musband)
        lp_liked = lps.order_by('-inlikes')[:5]
        lp_last = lps.order_by('-release')[:3]
        musband_data = {}
        musband_data['musband'] = musband
        musband_data['likedlps'] = lp_liked
        musband_data['lastlps'] = lp_last
        return render(request, self.template_name,
musband_data)

class MusbandCreateView(LoginRequiredMixin, View):
    template_name = 'mustasterapp/new_musband.html'

    def get(self, request, pk = None):
        MBform = MusbandForm()
        mb_data = {'mbform': MBform }
        return render(request, self.template_name, mb_data)

    def post(self, request, pk = None):
        MBform = MusbandForm (request.POST, request.FILES
or None)

        if not MBform.is_valid():
            mb_data = {'mbform': MBform }

```

```

        return render(request, self.template_name,
mb_data)
        mb = MBform.save(commit = False)
        mb.save()
        return redirect(reverse('mustasterapp:band', args =
[mb.id]))

class MusbandUpdateView (LoginRequiredMixin, View):
    template_name = 'mustasterapp/new_musband.html'

    def get(self, request, pk):
        musband = get_object_or_404(Musband, id = pk)
        MBform = MusbandForm (instance = musband)
        mb_data = {'mbform': MBform }
        return render(request, self.template_name, mb_data)

    def post(self, request, pk):
        musband = get_object_or_404(Musband, id = pk)
        MBform = MusbandForm (request.POST, request.FILES
or None, instance = longplayer)
        if not LPform.is_valid():
            mb_data = {'mbform': MBform }
            return render(request, self.template_name,
mb_data)

            mb = MBform.save(commit = False)
            mb.save()
            return redirect(reverse('mustasterapp:band', args =
[mb.id]))

class MusbandlistDetailView(DetailView):
    model = Musbandlist
    template_name = 'mustasterapp/musbandlist.html'

    def get(self, request, pk):
        musbandlist = Musbandlist.objects.get(id = pk)
        listener_user = self.request.user
        listen_user = Listenuser.objects.get(user =
listener_user)
        luf = Listenuserfollower.objects.get(listenuser =
musbandlist.listenuser, follower = listen_user)
        if (musbandlist.accessmode == 1 and luf) or
musbandlist.accessmode == 0 or musbandlist.accessmode == 3:
            parts =
Musbandlistpart.objects.filter(musbandlist = musbandlist)
            musbandso = []
            for part in parts:

```

```

        musbandso.append(part.musband.id)
        musbands = Musband.objects.filter(id__in =
musbands)

        ordering =
int(self.request.GET.get('ordering'))
        if ordering == 0:
            musbands = musbands.order_by('-inlists')
        elif ordering == 1:
            musbands = musbands.order_by('- musname')
        elif ordering == 2:
            musbands = musbandso
    else:
        musbandlist = None
        musbands = []
        musbandlist_data = {}
        musbandlist_data['musbandlist'] = musbandlist
        musbandlist_data['musbands'] = musbands
        return render(request, self.template_name,
musbandlist_data)

class MusbandLookView(ListView):
    model = Musband
    template_name = 'mustasterapp/musband_look.html'

    def get_queryset(self, *args, **kwargs):
        musbands = super().get_queryset(*args, **kwargs)
        title_phrase = str(self.request.GET.get('title'))
        if title_phrase:
            musbands =
Musband.objects.filter(musname__icontains = title_phrase)
        playings_count = int(self.request.GET.get('plays'))
        musbands = musbands.filter(playings__gte =
playings_count)
        origin_id = int(self.request.GET.get('origin'))
        if origin_id:
            origins = Origin.objects.get(id = origin_id)
            musbands = musbands.filter(origin = origins)
        ordering = int(self.request.GET.get('ordering'))
        if ordering == 0:
            musbands = musbands.order_by('-playings')
        elif ordering == 1:
            musbands = musbands.order_by('musname')
        elif ordering == 2:
            musbands = musbands.order_by('-inlists')
        return musbands

```

```

    def get_context_data(self, **kwargs):
        look_data = super(MusbandLookView,
self).get_context_data(**kwargs)
        look_data['lp_look'] = MusbandLookForm (initial={
            'title': self.request.GET.get('title', ''),
            'plays': self.request.GET.get('plays', '0'),
            'ordering': self.request.GET.get('ordering',
'0'),
            'origin': self.request.GET.get('origin', '')
        })
        return look_data

@method_decorator(csrf_exempt, name='dispatch')
class MusbandToListView(LoginRequiredMixin, View):

    def post(self, request, pk):
        musbandlist_id = request.data.get("musbandlist")
        mbl = Musbandlist.objects.get(id = pk)
        musbander = get_object_or_404(Musband, id = pk)
        mblpart = Musbandlistpart.objects.get(musbandlist =
mbl, musband = musbander)
        if not mblpart:
            mblpart = Musbandlistpart(musbandlist = mbl,
musband = musbander)
        try:
            mblpart.save()
            musbander.inlists = musbander.inlists + 1
            musbander.save()
        except IntegrityError as ierr:
            pass
        else:
            mblpart.delete()
            musbander.inlists = musbander.inlists - 1
            musbander.save()
        return HttpResponse()

class MusbandlistCreateView(LoginRequiredMixin, View):
    template_name = 'mustasterapp/new_musband_list.html'
    def get(self, request, pk = None):
        MBLform = MusbandlistForm()
        mbl_data = {'mblform': MBLform }
        return render(request, self.template_name,
mbl_data)

```

```

        def post(self, request, pk = None):
            MBLform = MusbandlistForm (request.POST,
request.FILES or None)
            if not MBLform.is_valid():
                mbl_data = {'mblform': MBLform }
                return render(request, self.template_name,
mbl_data)

            mbl = MBLform.save(commit = False)
            listener_user = self.request.user
            listen_user = Listenuser.objects.get(user =
listener_user)
            mbl.listenuser = listen_user
            mbl.save()
            return redirect(reverse('mustasterapp:mbandlist',
args = [mbl.id]))

    class MusbandlistUpdateView (LoginRequiredMixin, View):
        template_name = 'mustasterapp/new_musband_list.html'

        def get(self, request, pk):
            musbandlist = get_object_or_404(Musbandlist, id =
pk)

            MBLform = MusbandlistForm (instance = musbandlist)
            mbl_data = {'mblform': MBLform }
            return render(request, self.template_name,
mbl_data)

        def post(self, request, pk = None):
            musbandlist = get_object_or_404(Musbandlist, id =
pk)

            MBLform = MusbandlistForm (request.POST,
request.FILES or None, instance = musbandlist)
            if not MBLform.is_valid():
                mbl_data = {'mblform': MBLform }
                return render(request, self.template_name,
mbl_data)

            mbl = MBLform.save(commit = False)
            mbl.save()
            return redirect(reverse('mustasterapp:mbandlist',
args = [mbl.id]))

    class MyMusbandlistsView(LoginRequiredMixin, ListView):
        model = Musbandlist
        template_name = 'mustasterapp/my_musbandlists.html'

```

```

        def get_queryset(self, *args, **kwargs):
            musbandlists = super().get_queryset(*args,
**kwargs)

            listener_user = self.request.user
            listen_user = Listenuser.objects.get(user =
listener_user)
            musbandlists = musbandlists.filter(listenuser =
listen_user)

            ordering = int(self.request.GET.get('ordering'))
            if ordering == 0:
                musbandlists =
musbandlists.order_by('mbltitle')
            elif ordering == 1:
                musbandlists =
musbandlists.order_by('happened')
            elif ordering == 2:
                musbandlists = musbandlists.order_by('-
infollowers')

            return musbandlists

    class ListenerMusbandlistsView(ListView):
        model = Musbandlist
        template_name =
'mustasterapp/listener_musbandlists.html'

        def get_queryset(self, *args, **kwargs):
            musbandlists = super().get_queryset(*args,
**kwargs)

            listener_id = int(self.request.GET.get('listener'))
            listen_user = Listenuser.objects.get(id =
listener_id)
            musbandlists = musbandlists.filter(listenuser =
listen_user, accessmode__in = [0, 1])
            ordering = int(self.request.GET.get('ordering'))
            if ordering == 0:
                musbandlists =
musbandlists.order_by('mbltitle')
            elif ordering == 1:
                musbandlists =
musbandlists.order_by('happened')
            elif ordering == 2:
                musbandlists = musbandlists.order_by('-
infollowers')

            listener_user2 = self.request.user
            listen_user2 = Listenuser.objects.get(user =
listener_user2)

```

```

        luf = Listenuserfollower.objects.get(listenuser =
listen_user, follower = listen_user2)
        accessed_musbandlists = []
        for musbandlist in musbandlists:
            if musbandlist.accessmode == 1 and luf or
musbandlist.accessmode == 0 or musbandlist.accessmode == 3:
                accessed_musbandlists.append(musbandlist)
        return accessed_musbandlists

    class FollowedMusbandlistsView(LoginRequiredMixin,
ListView):
        model = Musbandlist
        template_name =
'mustasterapp/followed_musbandlists.html'

        def get_queryset(self, *args, **kwargs):
            musbandlists = super().get_queryset(*args,
**kwargs)

            listener_user = self.request.user
            listen_user = Listenuser.objects.get(user =
listener_user)
            mblf = Musbandlistfollower.objects.get(listenuser =
listen_user)
            mblf = mblf.order_by('-happened')
            mbls = []
            for mbl in mblf:
                mbls.append(mbl.musbandlist)
            musbandlists = musbandlists.filter(id__in = mbls)
            ordering = int(self.request.GET.get('ordering'))
            if ordering == 0:
                musbandlists =
musbandlists.order_by('mbltitle')
            elif ordering == 1:
                musbandlists =
musbandlists.order_by('happened')
            elif ordering == 2:
                musbandlists = musbandlists.order_by('-
infollowers')
            return musbandlists

    class BestMusbandsView(ListView):
        model = Musband
        template_name = 'mustasterapp/best_mesbands.html'

        def get_queryset(self, *args, **kwargs):
            musbands = super().get_queryset(*args, **kwargs)

```

```

origin_id = int(self.request.GET.get('origin'))
if origin_id:
    origins = Origin.objects.get(id = origin_id)
    musbands = musbands.filter(origin = origins)
    ordering = int(self.request.GET.get('ordering'))
    if ordering == 0:
        musbands = musbands.order_by('-playings')
    elif ordering == 1:
        musbands = musbands.order_by('-inlists')
    return musbands[:100]

@method_decorator(csrf_exempt, name='dispatch')
class FollowMusbandlistView(LoginRequiredMixin, View):

    def post(self, request, pk):
        mbl = Musbandlist.objects.get(id = pk)
        listener_user = self.request.user
        listen_user = Listenuser.objects.get(user =
listener_user)
        mblfollow =
Musbandlistfollower.objects.get(musbandlist = mbl, listenuser =
listen_user)
        if not mblfollow:
            mblfollow = Musbandlistfollower (musbandlist =
mbl, listenuser = listen_user)
            try:
                mblfollow.save()
                mbl.infollowers = mbl.infollowers + 1
                mbl.save()
            except IntegrityError as ierr:
                pass
        else:
            mblfollow.delete()
            mbl.infollowers = mbl.infollowers - 1
            mbl.save()
        return HttpResponse()

```

ДОДАТОК В
Слайди презентації

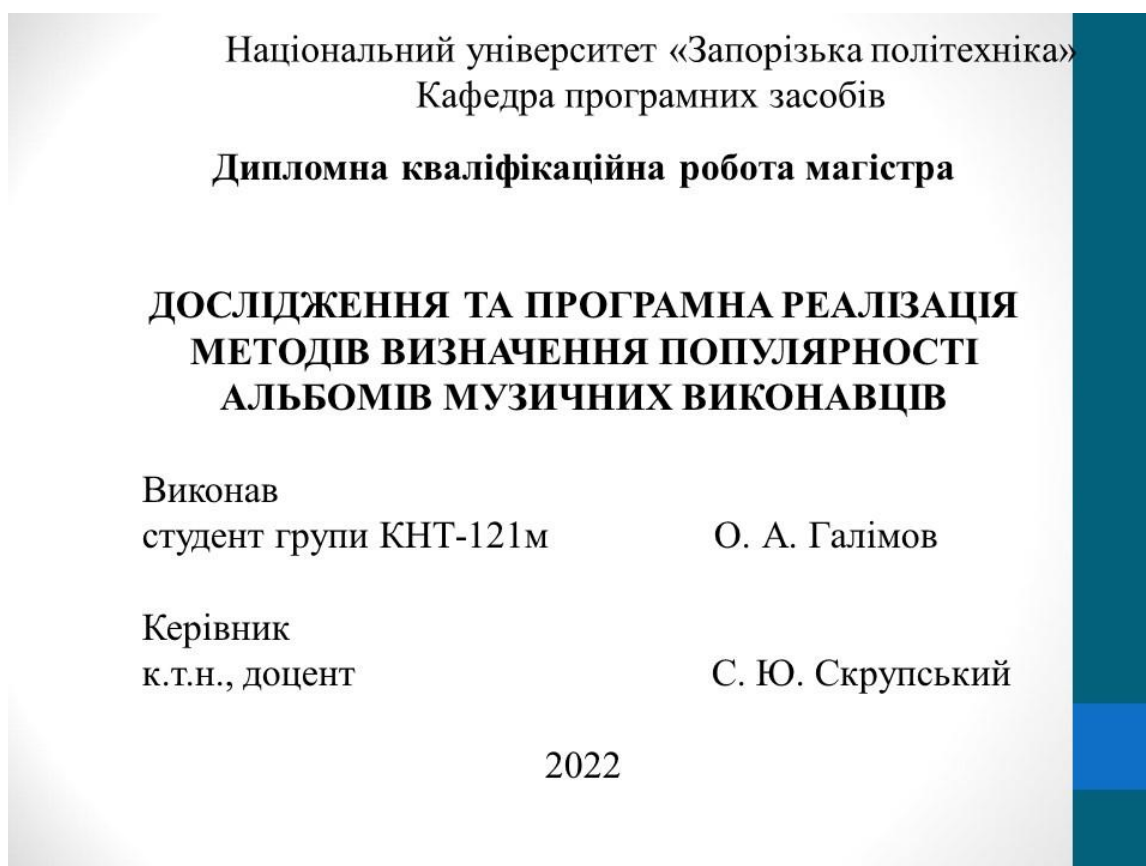


Рисунок В.1 – Титульний слайд

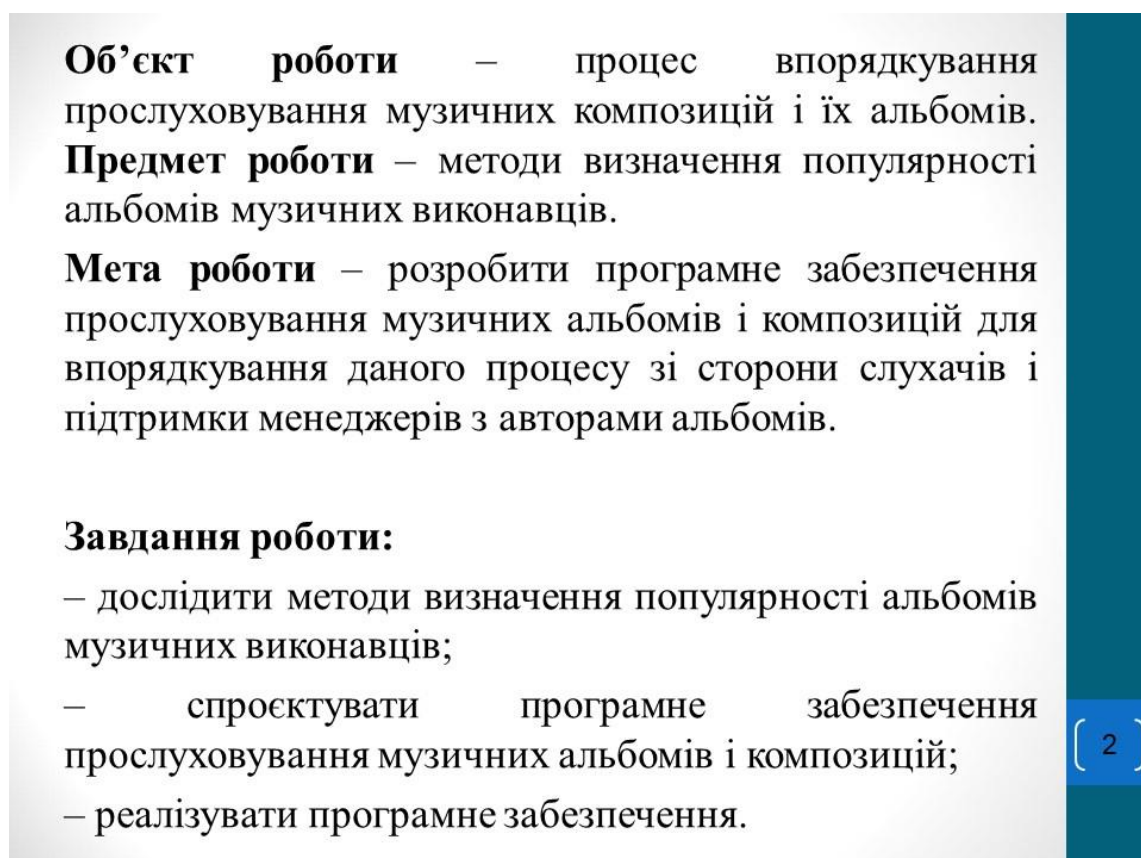


Рисунок В.2 – Об'єкт, предмет, мета роботи

Порівняння програмних аналогів

Критерій	Spotify	Soundcloud	Розробка
Прослуховування альбомів, композицій	+	+	+
Створення і надання доступу до переліків композицій/альбомів	+/-	+/-	+/+
Відправлення персональних порад	-	-	+
Визначення популярності альбомів	-	-	+

[3]

Рисунок В.3 – Порівняння програмних аналогів

Метод визначення популярності альбомів музичних виконавців

На першому етапі необхідно сформувати вибірку даних, яка складається з екземплярів, що представляють альбоми. Для того щоб отримати значення ознак альбому потрібно спочатку отримати ці значення для кожної музичної композиції з цього альбому. Тоді знаючи всі ці окремі значення, потрібно звести їх до більш обмеженого простору ознак, які характеризують весь альбом.

На другому етапі роботи потрібно створити модель визначення популярності альбому за допомогою застосування дерева рішень. В якості базового алгоритму навчання дерева рішень використовувався алгоритм ID3. За результатами виконання цього етапу побудована модель має бути збережена для повторюваного використання на наступних етапах.

Етапи 3-4 виконуються тоді, коли з'являється новий музичний альбом або змінюються значення ознак неопублікованого альбому.

На етапі 3 необхідно на основі композицій альбому сформувати екземпляр даних, який характеризує музичний альбом, у відповідності зі структурою, описаною вище. Тоді на четвертому етапі має бути подано цей екземпляр на вхід моделі і отримано клас популярності для цього екземпляру альбому.

[4]

Рисунок В.4 – Метод визначення популярності альбомів музичних виконавців

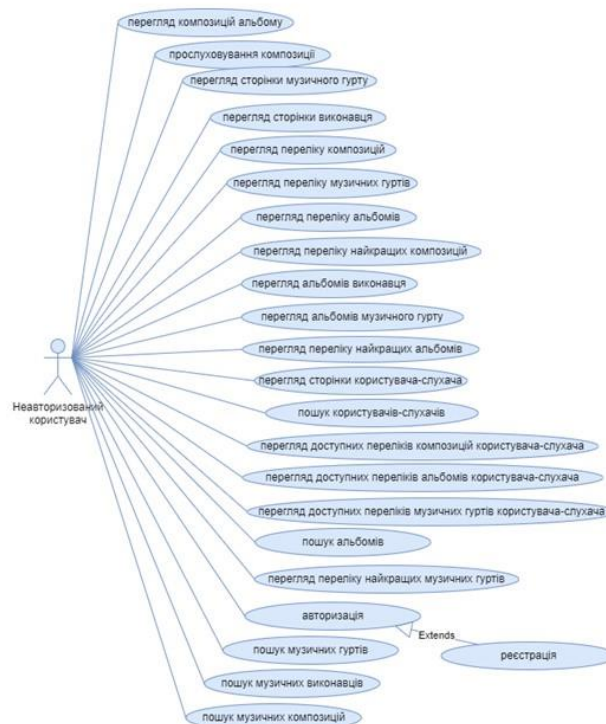
Порівняння фреймворків для розробки вебдодатку

Критерій	Flask	Django
Тип фреймворку	Легковісний	Повний стек
Тип проєктів	Малий	Великий, середній
Взаємодія з системами керування базами даних	Потребує підключення зовнішніх засобів	Підтримує популярні системи MySQL, PostgreSQL, Oracle
Зв'язування об'єктів даних з таблицями бази даних	Вручну або підключенням зовнішніх засобів	Вбудоване
Вбудоване вирішення адміністративних завдань	Ні	Так
Підтримка динамічних вебсторінок	Ні	Так

5

Рисунок В.5 – Порівняння фреймворків для розробки вебдодатку

Діаграми прецедентів



6

Рисунок В.6 – Діаграми прецедентів

Діаграми прецедентів

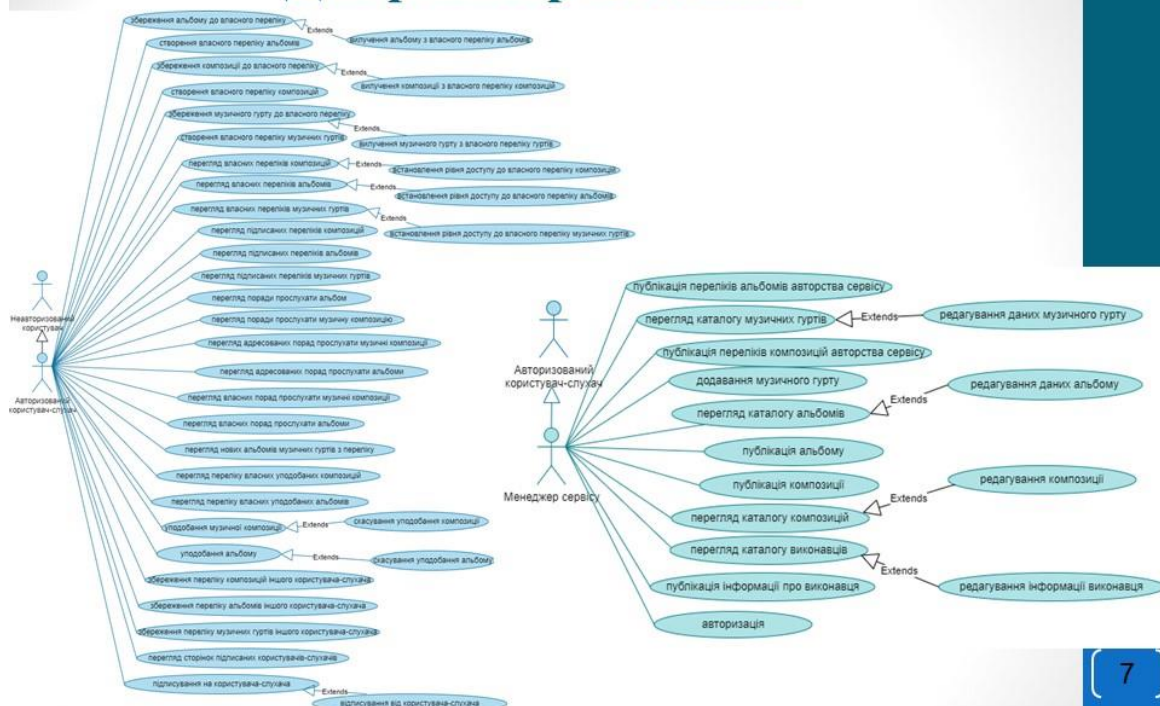


Рисунок В.7 – Діаграми прецедентів

Схема структури бази даних

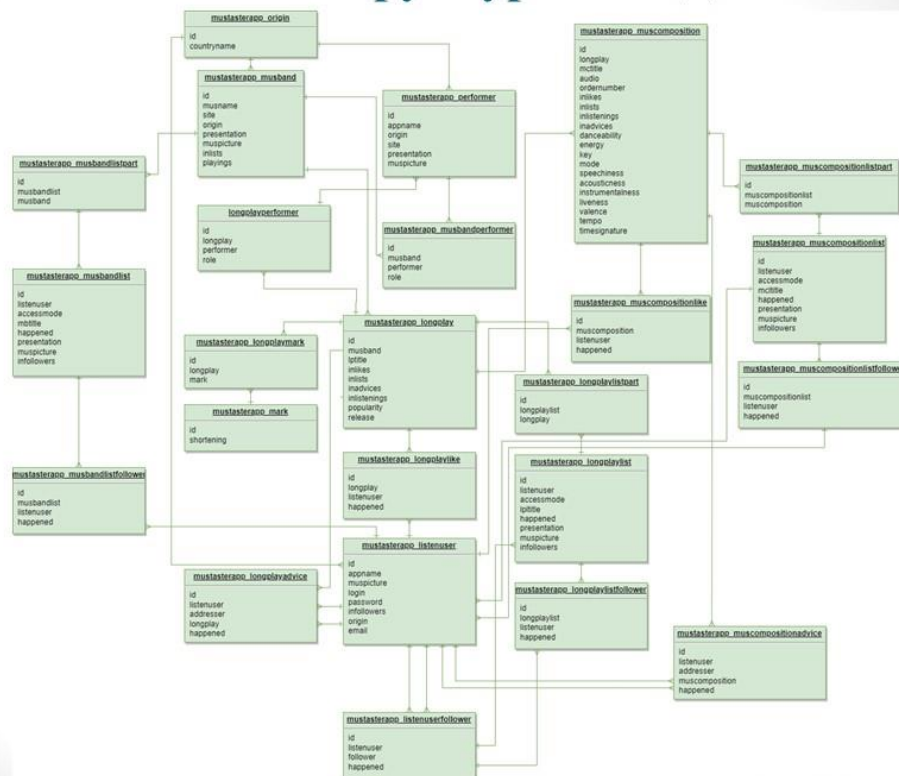


Рисунок В.8 – Схема структури бази даних

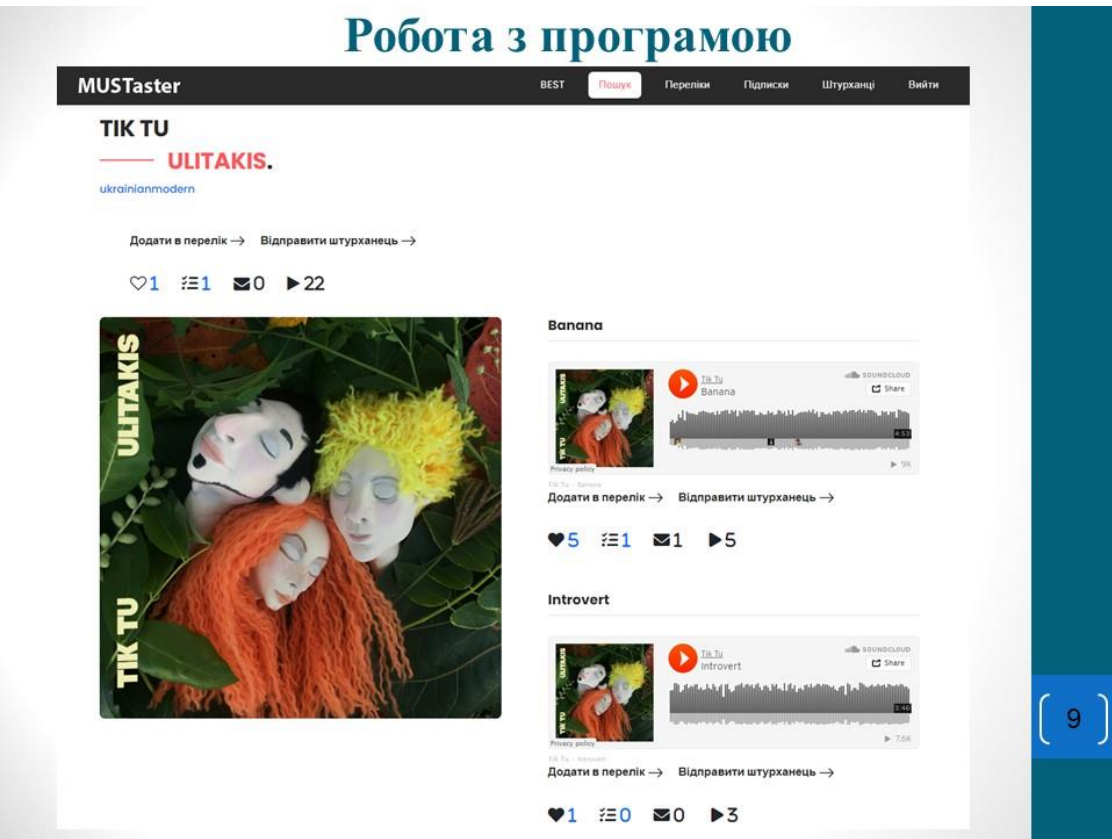


Рисунок В.9 – Робота з програмою

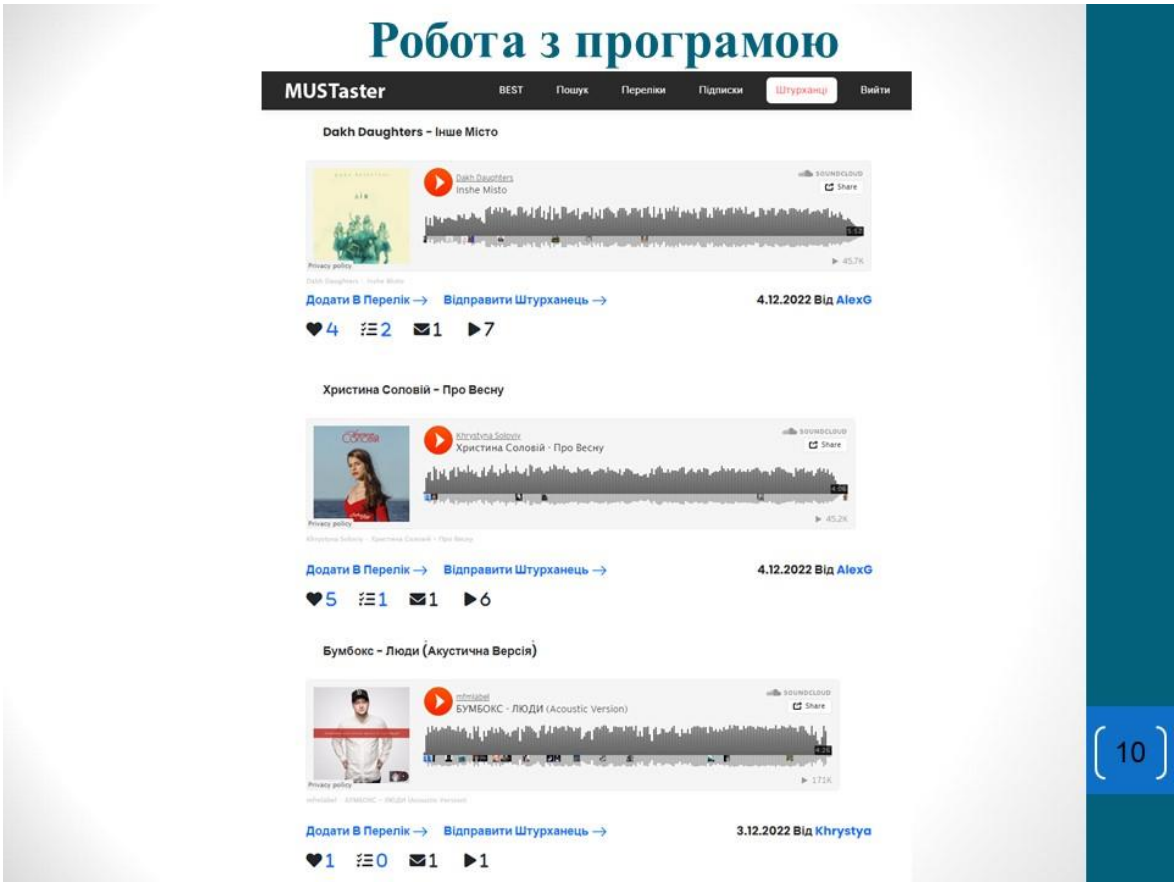
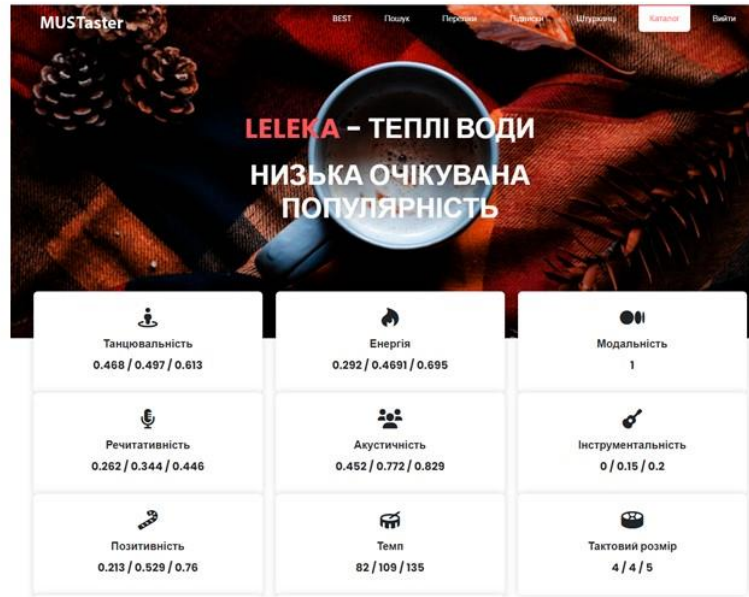


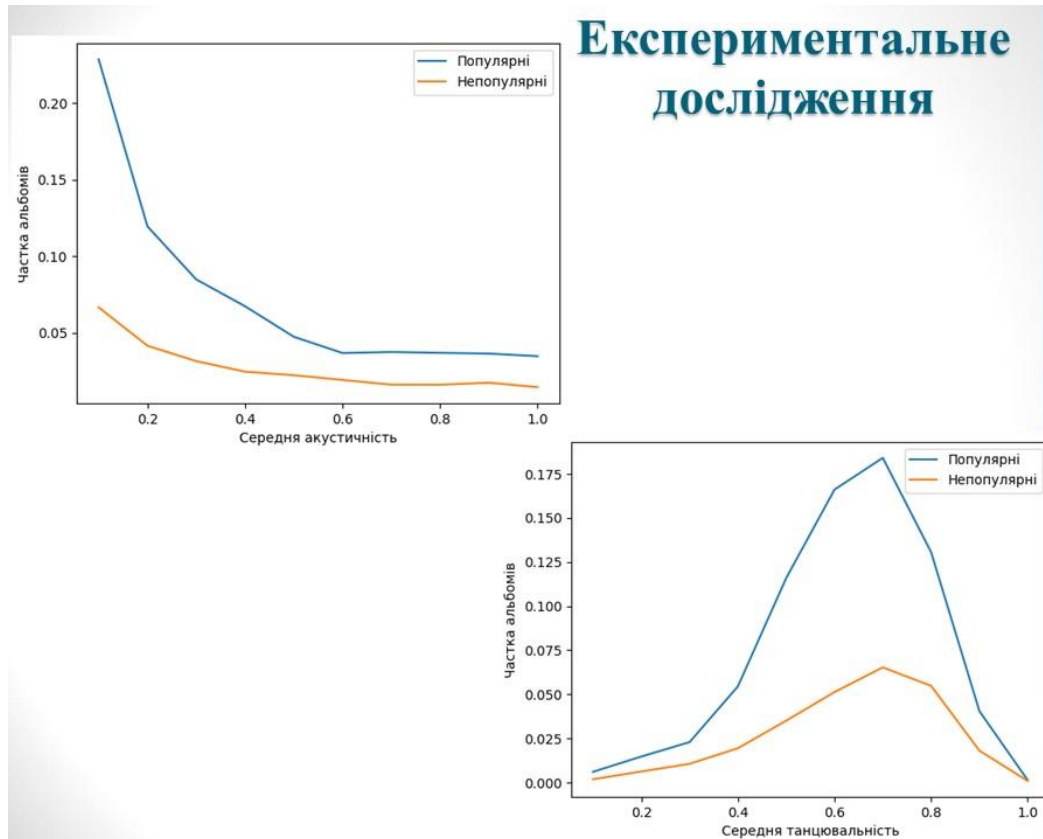
Рисунок В.10 – Робота з програмою

Робота з програмою



[11]

Рисунок В.11 – Робота з програмою



[12]

Рисунок В.12 – Експериментальне дослідження

Результати експериментального дослідження

Метод / модель	Похибка, %
Метод k-найближчих сусідів	37,29
Метод визначення популярності альбомів музичних виконавців	15,71

[13]

Рисунок В.13 Результати експериментального дослідження

Висновки

Представлене за результатами роботи програмне забезпечення дозволяє надавати слухачам інформацію про альбоми, композиції, музичні гурти, шукати їх, об'єднувати їх у власні переліки, підписуватися на переліки інших слухачів, ділитися персональними порадами, висловлювати власні уподобання композиціям і альбомам, визначати популярність альбомів. Перелік уподобаних композицій, альбомів може переглядати тільки користувач-слухач, який виконав ці уподобання. Для кожного переліку композицій, гуртів або альбомів автор може визначити рівень доступу, який може включати публічний доступ, доступ тільки для власних підписників або відсутність доступу неавторів. Передбачено створення переліків від імені сервісу.

Наукова новизна роботи сформована в методі визначення популярності альбомів музичних виконавців, який виконує класифікацію музичних альбомів на популярні і непопулярні, застосовуючи алгоритм ID3 на основі групи ознак, що характеризують музичні композиції в альбомі за різними критеріями, що дозволяє приймати рішення стосовно керування маркетинговою компанією альбому до його видання. Проведено експериментальне дослідження запропонованого методу, що підтвердило можливість його застосування на практиці.

Виконання роботи було побудовано на основі сформульованого переліку завдань, кожен пункт якого було виконано.

[14]

Рисунок В.14 – Висновки