Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра Програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система рекомендацій музики на основі вподобань групи користувачів. Back-end
(тема)

Виконав:
студент 4 курсу, групи ПЗПІ-20-10

Пушкар А. П.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник доц. кафедри ПІ Афанасьєва І.В.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра Програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Пушкарю Артему Дмитровичу
(прізвище, ім’я, по батькові)

1. Тема роботи Програмна система рекомендацій музики на основі вподобань групи користувачів. Back-end
Затверджена наказом по університету від 20.05. 2024р. № 471 Ст.
2. Термін подання студентом роботи до екзаменаційної комісії 12.06.2024
3. Вихідні дані до роботи В програмній системі рекомендацій музики на основі вподобань групи користувачів передбачити розробку серверної частини проекту за допомогою ASP .NET Web API та серверу штучного інтелекту розробленого на Python.
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі, огляд існуючих рішень, вибір найбільш придатних аналогів	08.04.2024 – 11.04.2024	виконано
2	Створення специфікації ПЗ, затвердження специфікації ПЗ керівником кваліфікаційної роботи	12.04.2024 – 16.04.2024	виконано
3	Проектування ПЗ	17.04.2024 – 24.04.2024	виконано
4	Розробка ПЗ	25.04.2024 – 17.05.2024	виконано
5	Тестування ПЗ	18.05.2024 – 24.05.2024	виконано
6	Оформлення пояснювальної записки	25.05.2024 – 01.06.2024	виконано
7	Перевірка роботи на антиплагіат	02.06.2024	виконано
8	Нормоконтроль	07.06.2024	виконано
9	Оцінка роботи рецензентом, отримання відгуку від керівника кваліфікаційної роботи	07.06.2024	виконано
10	Здача роботи у електронний архів, допуск роботи до захисту завідувачем кафедри	07.06.2024	виконано
11	Участь у демо-виставці	09.06.2024	виконано
12	Захист кваліфікаційної роботи	12.06.2024	виконано

Дата видачі завдання 5 березня 2024р.

Студент (ка)_____ Пушкар А.Д.
(підпис)

Керівник роботи _____ доц. кафедри ПІ Афанасьєва І.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи бакалавра, 72 с., 22 рис., 2 табл., 4 додатка, 11 джерел

МУЗИКА, РЕКОМЕНДАЦІЇ, РЕЛЯЦІЙНА БД, СЕРВЕРНА ЧАСТИНА, ШТУЧНИЙ ІНТЕЛЕКТ, AZURE

Об’єкт розробки – програмна система рекомендації музики на основі вподобань групи користувачів.

Мета розробки – створення серверної частини застосунку для рекомендації музики.

Метод рішення – середовище розробки Visual Studio та платформа AZURE, мова програмування C#, T-SQL та python 3.12, фреймворк .NET 7.

У результаті роботи було спроектовано та розроблено застосунок для рекомендації музики. Це рішення дозволяє користувачам отримувати персоналізовані рекомендації, що робить процес вибору музики більш зручним та ефективним. Впровадження даного застосунку покращує взаємодію користувачів з музичним контентом та сприяє відкриттю нових музичних творів.

MUSIC, RECOMMENDATIONS, RELATIONAL DATABASE, SERVER SIDE, ARTIFICIAL INTELLIGENCE, AZURE

Development object – a music recommendation software system based on the preferences of a group of users.

Development goal – to create the server-side component of the music recommendation application.

Solution method – Visual Studio development environment and AZURE platform, C#, T-SQL та python 3.12, .NET 7 framework.

As a result of the work, an application for music recommendation was designed and developed. This solution allows users to receive personalized recommendations, making the process of selecting music more convenient and efficient. The implementation of this application enhances user interaction with music content and facilitates the discovery of new musical pieces.

Я, Пушкар Артем Дмитрович, студент гр. ПЗПІ-20-10, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Програмна система рекомендацій музики на основі вподобань групи користувачів. Back-end», що буде представлена до екзаменаційної комісії для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови до допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API – Application Programming Interface;

FTP – File Transfer Protocol;

HTTP – Hypertext Transfer Protocol;

IP – Internet Protocol;

JWT – JSON Web Token;

ОС – Операційна система;

SQL – Structured Query Language;

TCP – Transmission Control Protocol.

ВСТУП

В сучасному світі для користувачів доступний величезний обсяг музичного контенту за різними вподобаннями. Користувачі бажають швидко знаходити музику, яка відповідає їхнім уподобанням, тому для них важливо мати доступ до ефективних методів сортування музичного контенту за власними критеріями.

Тому програми рекомендації музики стали необхідним елементом для багатьох платформ. Шляхом аналізу вподобань, вони відкривають користувачам новий музичний контент.

Рекомендаційні системи музики, на основі вподобань групи користувачів, відіграють важливу роль у вдосконалені споживання музичного контенту та користування стрімінговими платформами.

Перевагою таких систем є здатність адаптуватися до індивідуальних музичний уподобань кожного користувача, надавати персональні рекомендації та роботи процес пошуку нової музики більш зручним.

Метою цього кваліфікаційного проекту є розробка та реалізація Back-end частини програмної системи рекомендації музики на основі вподобань групи користувачів.

Предметом дослідження є аналіз та моделювання вподобань користувачів, обробка великого обсягу музичних даних, реалізацію системи збереження та управління даними в базі даних, а також розробку API для взаємодії з front-end компонентами та іншими сервісами.

Методи розробки базуються на технології ASP.NET Core WebAPI та мови програмування C#. База даних сервера написана на SQL Azure за допомогою T-SQL. Брокер повідомлень створений за допомогою RabbitMQ. Сервіс зберігання фотокарток створений на платформі Azure як blob-сховище.

В результаті реалізації кваліфікаційного проекту була розроблена та впроваджена Back-end частина програмної системи рекомендації музики на основі вподобань групи користувачів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної області

Музика – унікальна універсальна мова, яка об'єднує суспільство по-різному. Вона сприяє комунікації між людьми різних культур та походжень і відіграє важливу роль у емоційному зв'язку, зміцнюючи зв'язки між націями через обмін музичними враженнями. Музика також, важливо, дозволяє нам відчувати майже всі емоції, які ми відчули в житті. Однак виклик полягає в тому, що музичні уподобання у групі людей не завжди збігаються.

Зараз музика є дуже сильним інструментом в тому числі і для заробітку. Тому великі компанії відкривають стрімінгові платформи для її прослуховування. Кожна з цих платформ має свій алгоритм рекомендації пісень, що робить їх ще більш привабливими.

Система рекомендацій – це фільтруюча система, метою якої є передбачення вподобань, які користувач надав би певному елементу, у нашому випадку - пісні. Це ядро великих двигунів, які працюють за допомогою певних алгоритмів рекомендацій та пропонують один елемент або набір елементів користувачам на основі таких передбачень.

Чи ми усвідомлюємо це чи ні, різноманітні системи рекомендацій стали невід'ємною частиною нашої повсякденної рутина останнім часом. Починаючи від точно націленої реклами та пропозицій товарів і закінчуючи персоналізованими відео чи музичними плейлистами, складеними спеціально для нас - системи рекомендацій альбомів, здається, охоплюють нашу повсякденну діяльність з кожного кутка цифрового простору.

Отже, предметна область даної роботи є злиттям рекомендаційних систем, інформаційних технологій та музикальної індустрії для забезпечення швидкого процесу рекомендації музики для компанії.

1.2 Виявлення та вирішення проблем

У музичній індустрії системи рекомендацій є частиною великого двигуна стрімінгових застосунків, таких як Spotify, YouTube Music, Deezer і подібних. Ці сервіси використовують потужні алгоритми, щоб надавати персоналізовані рекомендації, які налаштовані на індивідуальні смаки користувача. Таким чином, вони можуть запропонувати нову музику, яка ідеально відповідає смаку кожного слухача.

Серед них немає прямих конкурентів застосунку рекомендації музики на основі вподобань групи користувачів «Melody Fusion». Їх рекомендації налаштовані під конкретного користувача на основі його прослуховувань та вподобань. Наприклад стрімінгова платформа YouTube Music (див. рис. 1.1) має широкий спектр функціонала, а саме:

– Підбір пісень за енергією, жанрами, настроєм та іншим;

– Розповсюдження плейлистів інших користувачів;

– Рекомендація нових пісень/ на основі вподобань;

– Зберігання особистих плейлистів;

– Перегляд хіт-парадів у світі або окремих країнах;

– Режим караоке;

– Перегляд відповідних музикальних кліпів до кожної з пісень у YouTube.

«Melody Fusion» привертає увагу саме за рахунок рекомендації музики на основі вподобань кількох користувачів, що не можуть зробити великі стрімінгові сервіси.

Застосунок зможе задовільнити потреби компанії людей в підборі пісень та спростить цей процес. Такий підхід до рекомендацій може виявитися особливо привабливим для користувачів, які шукають спосіб спільного відкриття нової музики та обміну музичними враженнями з іншими.

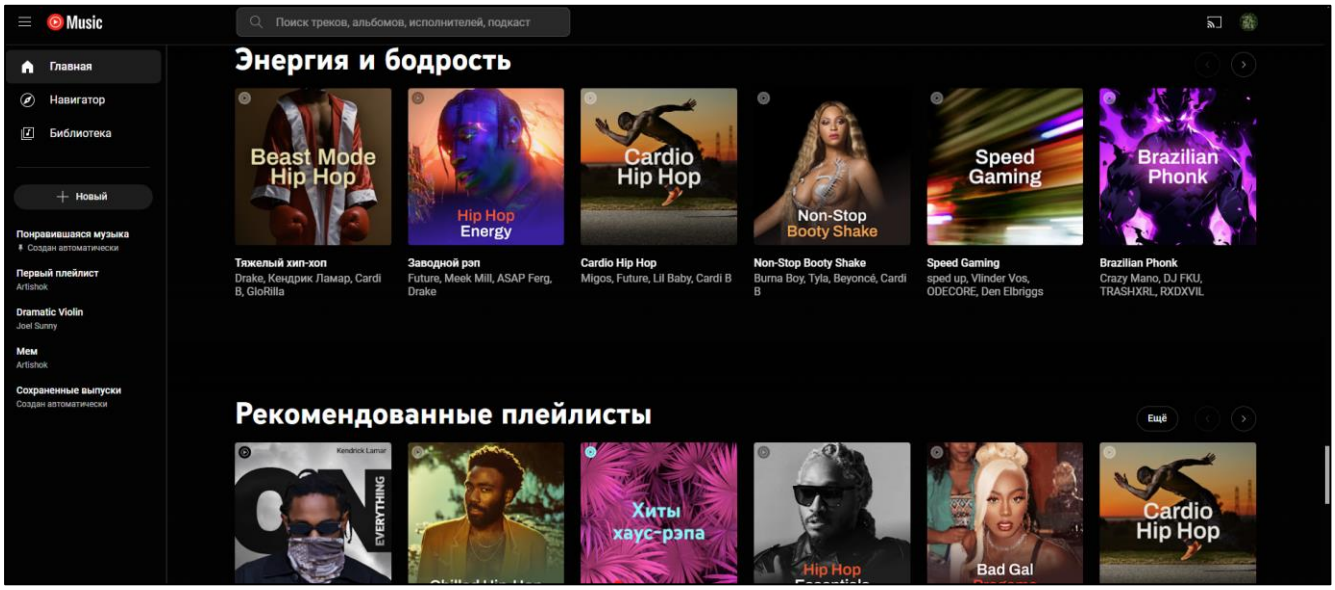


Рисунок 1.1 – Стрімінгова платформа пісень YouTube Music (за даними [1])

На ринку вже існують схожі програмні застосунки, такі як система рекомендацій фільмів для закоханих «Date Night Movies».

Цей сайт відомий своєю простотою та точністю в рекомендаціях. Інтерфейс цього сайту складається лише з однієї функціональної сторінки, що робить його використання максимально простим. Користувач обирає два фільми на основі яких підбирається третій.

Такі приклади показують, що існує попит на системи рекомендацій, які орієнтуються не лише на індивідуальні вподобання, але й на групові. Такий підхід може бути особливо цінним для груп друзів, сімей або пар, які бажають насолодитися спільним музичним досвідом.

Він дозволяє не лише знайти нову музику, але і створити атмосферу спільності та спільного відкриття для групи людей з різними музичними смаками. Таким чином, система рекомендацій «Melody Fusion» може стати цінним ресурсом для тих, хто шукає спосіб насолодитися музикою разом з друзями та близькими (див. рис. 1.2).

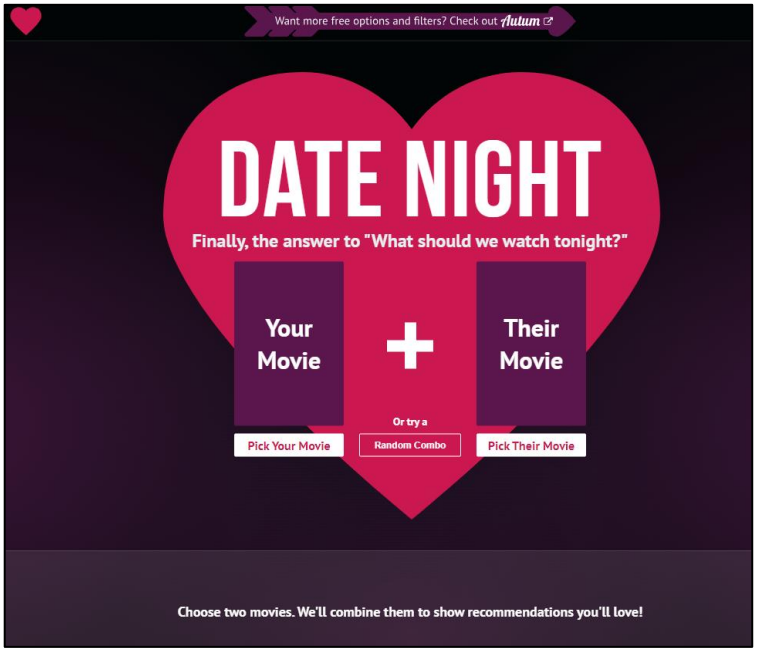


Рисунок 1.2 – Система рекомендацій фільмів для закоханих «Date Night Movies» (за даними [2])

Попри всі переваги продукт «Melody Fusion» та потенційні його можливості, важливо вказати, що існує бізнес-ризики, які безпосередньо впливають на успішність проекту та його подальшу релевантність (див. Таблиця 1.1).

Таблиця 1.1– Бізнес ризики (таблиця виконана самостійно)

Ризики	Складність	Пом’якшення
Поява конкурентів	Висока	Забезпечення якісного продукту та постійне оновлення баз даних свіжими даними
Недостатня кількість клієнтів	Середня	Розповсюдження таргетованої реклами
Низька продуктивність сервера	Висока	Горизонтальне та вертикальне масштабування серверів і баз даних

Кінець таблиці 1.1

Ризики	Складність	Пом’якшення
Проблеми з авторськими правами	Низька	Ліцензування контенту та використання стрімінгових платформ
Відсутність фінансування	Висока	Пошук інвесторів, презентація проекту для широкої аудиторії для знаходження інвестування

Також серед основних проблем для запуску програмної системи рекомендацій музики на основі вподобань групи користувачів є так званий «холодний старт».

Як і будь-який штучний інтелект, система рекомендацій повинна мати хоча б мінімальну базу даних для тренування та подальшого аналізу пісень. Вирішення цієї проблеми частково чіпає саме обраний підхід до рекомендацій, а саме Content-based filtering (фільтрація на основі вмісту), який не аналізує рейтинги, виставлені пісням користувачами.

Content-based filtering використовує метадані пісень, такі як жанр, темп, стиль, інструменти та інші характеристики, для формування рекомендацій. Це дозволяє системі надавати рекомендації навіть за відсутності великої кількості користувацьких даних або історії прослуховувань. Таким чином, система може почати працювати і надавати релевантні рекомендації відразу після запуску, використовуючи доступні дані про самі пісні.

Вибір набору даних для фільтрації по метаданим є дуже важливою складовою подальшої коректної роботи платформи. Тож для цієї цілі релевантно використання глобальних стрімінгових платформ пісень з відкритими джерелами та базами даних. Такою є платформа Spotify, а саме Spotify API.

Spotify надає доступ до великої кількості музичних даних через свій API. Ці дані включають детальні метадані про пісні, такі як темп, танцювальність, мелодичність, інструментальність, емоційна тональність та інші характеристики.

Основні переваги використання даних Spotify включають:

– різноманітність даних. Стрімінгова платформа пропонує широкий спектр пісень з різним набором критеріїв, такі як темп, танцювальність, мелодичність, інструментальність, емоційна тональність та інші характеристики;

– доступність та великий обсяг. Spotify має одну з найбільших баз даних у світі, що є великою перевагою для використання її як основної платформи для подальшої обробки даних;

– актуальність. Дані цієї бази даних регулярно оновлюються, що забезпечує актуальність даних;

– аналітичні дані. Крім метаданих є можливість отримати додаткові дані про взаємодію користувачів з музикою, що у подальшому можна використати для покращення алгоритмів рекомендацій.

1.2.1 Регуляторні та стандартизовані аспекти

Музична індустрія, як і багато інших, має багато законів та регуляцій, які стосуються різних її аспектів (інтелектуальної власності, ліцензування, авторських прав та інше).

Вище наведені аспекти повинні бути враховані, тому програмна система рекомендацій музики на основі вподобань групи користувачів має наступні обмеження:

– неможливість прослуховування пісень прямо з платформи, через відсутність авторських прав на них;

– географічні обмеження стають проблемою для відображення музичних композицій для користувачів застосунку. Це означає, що користувачі з різних країн мають різний доступ до музичної бібліотеки Spotify.

1.3 Постановка задачі

Для вирішення вище описаних проблем, потрібно розробити back-end компонента програмної системи рекомендації музики на основі вподобань групи користувачів.

Для досягнення цієї мети необхідно виконати наступні завдання:

– розробити алгоритми для аналізу музичних вподобань користувачів, які зможуть визначити їхні індивідуальні вподобання та стиль музики.

– розробити механізми зберігання та обробки даних користувачів, включаючи їхню історію прослуховування музики та їхню взаємодію з контентом на ній.

– забезпечити асинхронну обробку запитів для швидкого доступу до рекомендацій та даних для користувачів.

– підключити серверну частину до інших сервісів, таких як Spotify або Azure, щоб отримати додаткову інформацію про музичний контент та користувачів.

– забезпечити високий рівень безпеки та захисту особистих даних користувачів за допомогою відповідних методів шифрування та аутентифікації.

– протестувати розроблені функції та алгоритми на предмет їхньої функціональності та точності зроблених рекомендацій.

– постійно моніторити продуктивність системи з метою оптимізації, аналізуючи її з часу на час, тим самим постійно вдосконалюючи механізм рекомендацій.

Для того, щоб розробити серверну частину запропонованого застосунку було обрано кросплатформений фреймворк ASP.NET CORE від Microsoft та інтеграцію з сервісами Azure та Spotify [4]. Система доступна на таких ОС, як Windows, Linux, MacOS.

1.3.1 Опис потрібних підсистем програмного комплексу

Перелік необхідних підсистем:

– система управління базами даних SQL та blob -сховищ;

– серверна система для обробки запитів користувачів, від клієнтських частин проекту (back-end);

– серверна частина для обробки запитів на надання рекомендацій музики (штучний інтелект);

– система відправки та отримання повідомлень для взаємозв'язку серверів штучного інтелекту та back-end частини.

1.3.2 Система управління базами даних SQL та blob-сховищ

Система управління базами даних (СУБД) є ключовим компонентом програмного застосунку, яка забезпечує зберігання, обробку та керування структурованими даними.

Основні функції СУБД включають:

– зберігання реляційних та нереляційних даних у достатньо великому обсязі;

– обробка великої кількості одночасних запитів;

– надання зручних інтерфейсів для підтримки роботи;

– забезпечення надійного збереження даних у сховищі з механізмом контролю доступу та резервного копіювання;

– можливість горизонтального та вертикального масштабування системи;

Також система використовується для зберігання великих файлів картинок з зображенням фотокарток користувачів.

Для даних задач підходить платформа Azure з її сервісами зберігання реляційних та нереляційних даних.

1.3.3 Серверна система для обробки запитів користувачів (back-end)

Підсистема відповідає за прийом, бізнес логіку та відповіді на запити від клієнтських частин проекту. Вона є посередником між клієнтською частиною а базою даних. Основні функції включають:

– отримання HTTP/HTTPS – запитів від клієнтських частин;

– обробка запитів, яка включає в себе регістрація, аутентифікація, адміністрування, тощо;

– відправка відповідей у форматі json або статусів запитів до клієнтської сторони.

1.3.4 Серверна частина для обробки запитів на надання рекомендацій музики (штучний інтелект)

Підсистема використовує алгоритми штучного інтелекту для самостійної генерації рекомендації пісень. Основні функції включають:

– генерація рекомендацій за допомогою алгоритмів штучного інтелекту;

– інтеграція з серверної частиною (back-end) для прийому та відправці даних;

– аналіз даних про усі музичні аспекти переданих пісень.

1.3.5 Система відправки та отримання повідомлень для взаємозв'язку серверів штучного інтелекту та back-end частини

Система використовує черги повідомлень для розвантаження серверних частин від великої кількості запитів від клієнтів. Основними функціями є:

– черги повідомлень для обміну даних між частинами;

– моніторинг та логування для глубокої діагностики станів запитів;

– безпека та надійність в роботі систем навіть в умовах великих навнтажень.

Для даних задач підходить брокер повідомлень RabbitMQ, яка має широкий функціонал для реалізації поставлених задач.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Головна функціональність

Можна виділити наступні основні функціональні вимоги до застосунку рекомендації музики на основі вподобань групи користувачів:

Основна функція – 1: Особистий кабінет;

– реєстрація та аутентифікація;

– авторизація

– управління особистою інформацією;

– Основна функція – 2: Рекомендація пісень;

– безпосередня взаємодія користувача зі штучним інтелектом;

– взаємодія зі стрімінговою платформою Spotify для надання даних для пісень;

– надання великої кількості сучасних пісень для вибору;

– можливість відстежування останніх рекомендацій в особистому профілі;

Основна функція – 3: Здійснення сплати підписки за допомогою сервесів PayPal Braintree;

Основна функція – 4: Адміністрування;

– управління базою користувачів (блокування в системі, видача ролей)

– отримання статистичних даних за певний період часу;

Основна функція – 5: Отримання листів на пошту з підтвердженням персонального акаунту.

2.2 Технології та інструменти

Для розробки програмної системи для рекомендації музики на основі вподобань групи користувачів використовуються наступні технології:

– ASP.NET CORE Web API для створення кросплатформеної серверної частини, яка може запускатись на таких операційних системах як Linux, Windows, MacOS.

– використання Spotify API для отримання детальної інформації про рекомендовані пісні та посилання на них на стрімінговій платформі;

– використання брокеру повідомлень для розвантаження серверу зі штучним інтелектом та перекидання запитів у різні черги для розпаралелювання запитів від користувачів;

– використання Cloud сервісами Azure для збереження даних в реляційних та нереляційних баз даних. Реляційні SQL бази даних є легко масштабованими. Використання нереляційної бази даних для збереження та швидкого пошуку фотокарток користувачів;

– використання PayPal Braintree сервісів для створення оплати в застосунку;

– передача даних через HTTPS протоколи для захищеного надсилання даних на клієнтські частини проекту;

– використання jwt-токенів авторизації для збереження клієнтської інформації;

– використання MailKit для відправки особистих повідомлень на пошту клієнтів [3].

2.3 Продуктивність та оптимізація

При розробці програмної системи рекомендації музики на основі вподобань групи користувачів у Back-end частині, основна увага зосереджена на забезпеченні високої продуктивності та оптимізації проекту. Використання технологій .NET [4] робить розробку набагато швидшою та ефективнішою, а асинхронне програмування [5] дозволяє максимально використовувати ресурси сервера, обробляючи запити з різних потоків одночасно та не дозволяє блокувати систему програми.

Використання бази даних Azure забезпечує швидкий доступ до інформації, а також масштабованість системи при збільшенні обсягу даних. Присутня можливість як горизонтального так і вертикального масштабування. Оптимізація

запитів до баз даних за допомогою кластерних та некластерних індексів грає критичну роль у підвищенні швидкості роботи проекту.

Також розвантаження серверів зі штучним інтелектом є дуже високо пріоритетною задачею. За допомогою брокеру повідомлень RabbitMQ [6] вдалося досягти розпаралелювання запитів від серверу до серверу Python (див. рис. 2.1) зі штучним інтелектом по різним чергам (черги отримання та відправлення запитів).

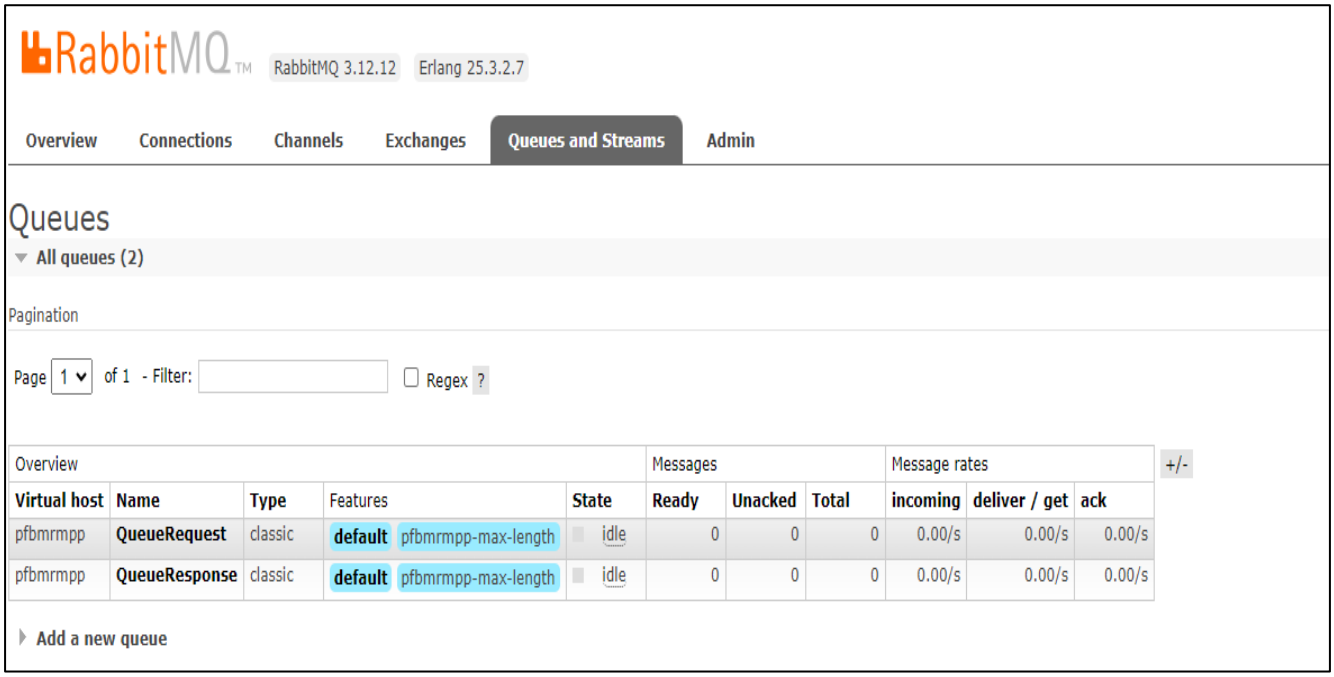


Рисунок 2.1 – Приклад черг у RabbitMQ (рисунок виконаний самостійно)

Крім того, моніторинг застосунків та аналіз продуктивності допомагають вчасно виявляти та усувати можливі неефективності в системі. Регулярні оновлення та вдосконалення коду сприяють забезпеченню високого рівня ефективності.

3 АРХІТЕКТУРА ПРОЕКТУ

3.1 UML проектування ПЗ

3.1.1 UML діаграма прецедентів

Спочатку необхідно ретельно проаналізувати ПЗ для з'ясування всіх вимог до системи. Це включає визначення функціональних та нефункціональних вимог, а також взаємозв'язків між компонентами системи. Застосунок має 2 типи користувачів, а саме: користувач та адміністратор системи. Для їх опису було обрано використати UML діаграму прецедентів.

Для користувача існує два типи дій: які дозволені неавторизованим, та ті, які дозволені авторизованим юзерам. Загальний функціонал включає в себе регістрацію та авторизацію.

Після проведеної авторизації, користувач може використовувати усі доступні для звичайного юзера функції. Такими є: керування особистою інформацією, можливість придбання підписки, перегляд списку раніше рекомендованих пісень, можливість підтвердження електронної пошти. Головним аспектом є те, що він може через серверну частину, взаємодіяти зі штучним інтелектом (див. рис. 3.1).

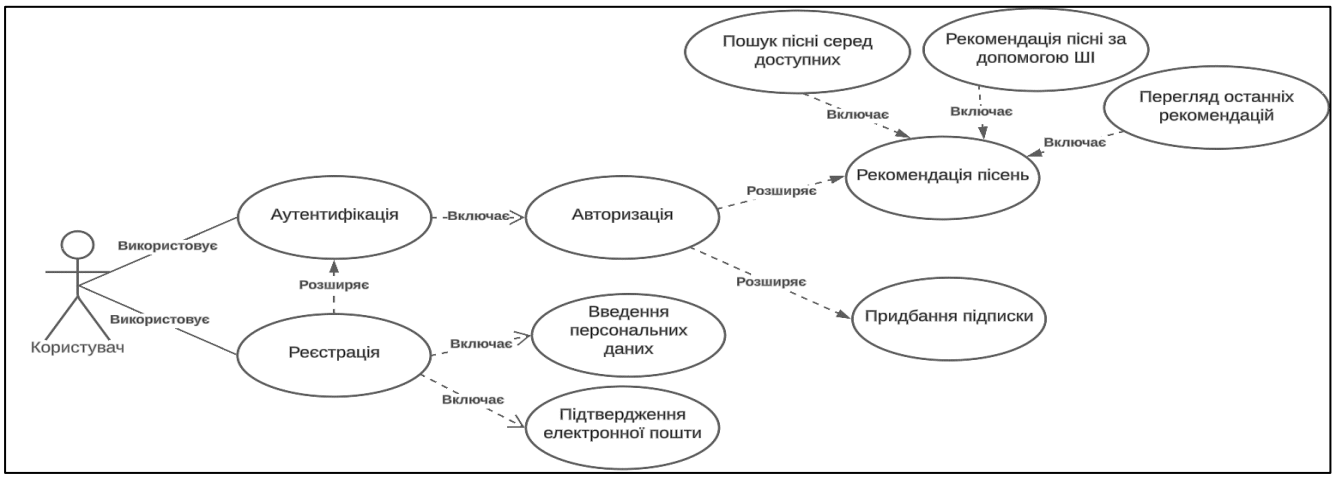


Рисунок 3.1 – UML діаграма прецедентів (Use Case Diagram) для користувача з роллю «Користувач» (рисунок виконаний самостійно)

Користувачі з роллю Адмін (див. рис. 3.2) доповнюють функціонал ролі звичайного юзера наступними діями: управління користувачами, перегляд їх списків, зміна ролей, блокування або розблокування та перегляд статистичних даних.

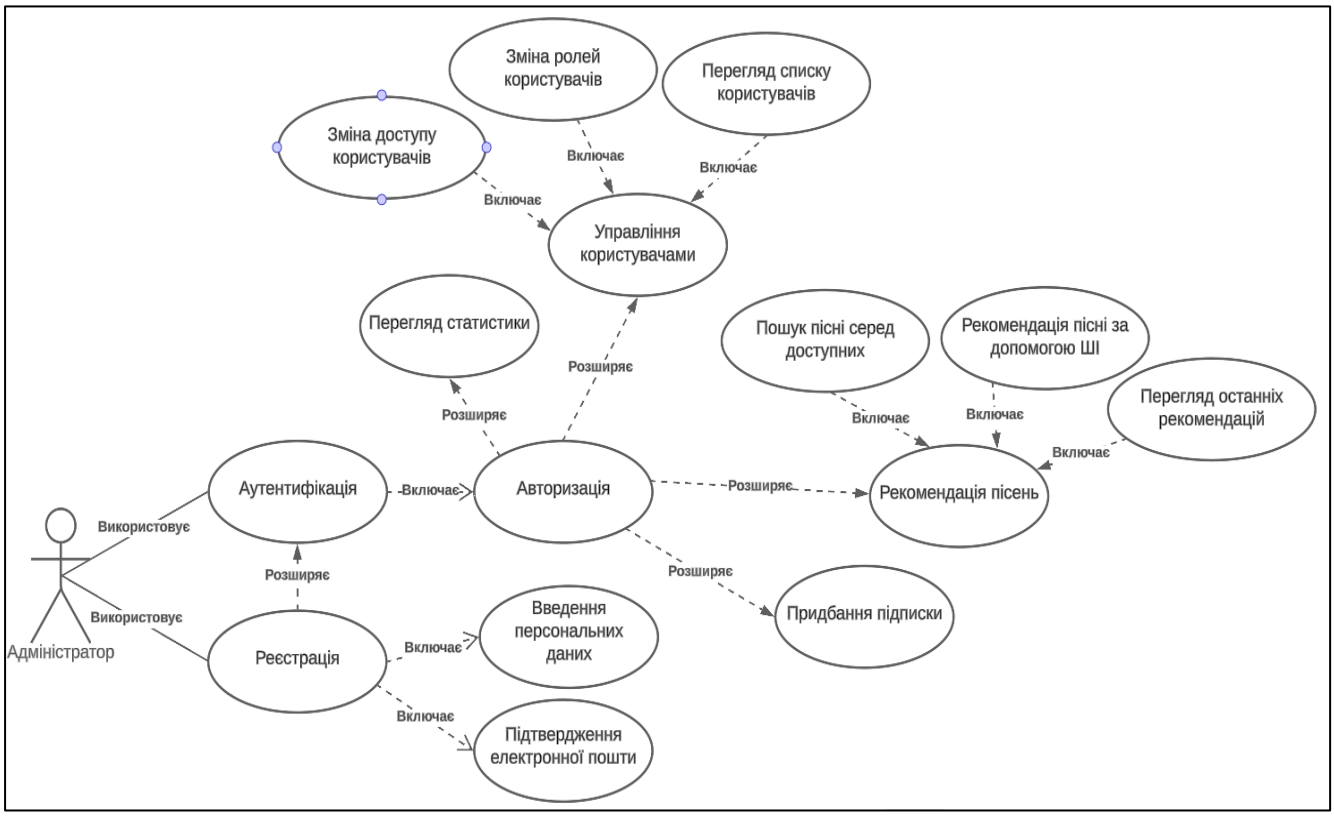


Рисунок 3.2 – UML діаграма прецедентів (Use Case Diagram) для користувача з роллю «Адміністратор» (рисунок виконаний самостійно)

Таким чином за допомогою розділення ролей у застосунку кожен з типів користувачів може взаємодіяти лише зі своїм набором функцій.

3.1.2 UML діаграма пакетів

Було виділено основний пакет рішення, який включає такі пакети, як (див. рис. 3.3), що забезпечують інтегрований підхід до управління проектами, дозволяючи оптимізувати процеси та підвищити ефективність роботи проекту (див. рис. 3.3).

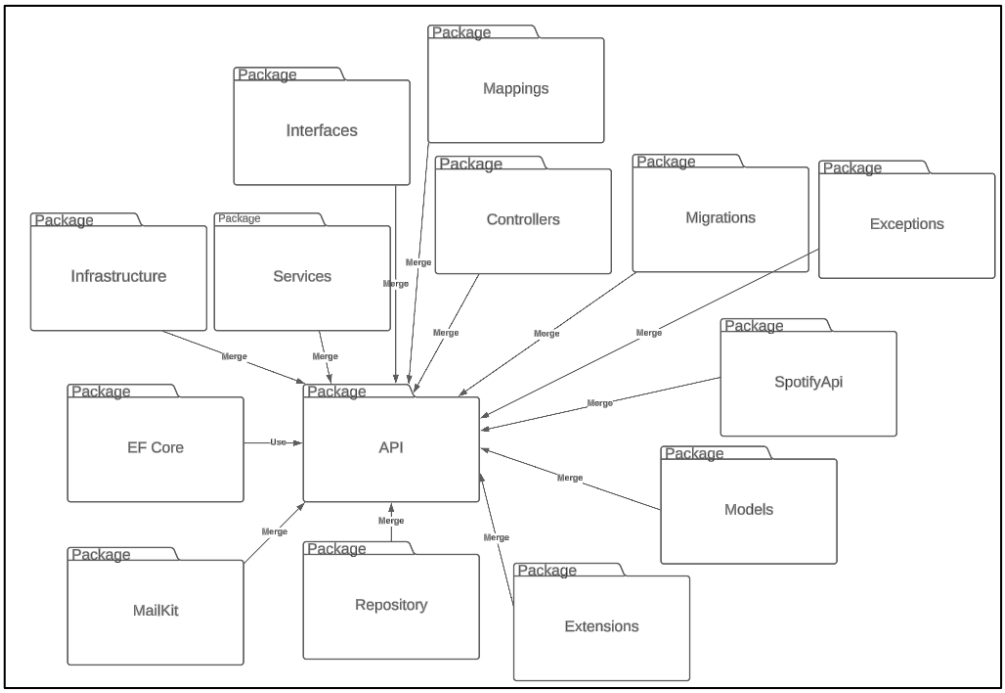


Рисунок 3.3 – UML діаграма пакетів проекту «Програмна система рекомендації музики на основі вподобань групи користувачів. Back-end» (рисунок виконаний самостійно)

– контролери (Controllers) включають точки входу до проекту;

– моделі (Models) містять сутності бази даних та їхні відображення;

– сервіси (Services) обробляють інформацію;

– інфраструктура (Infrastructure) включає компоненти, які взаємодіють з зовнішніми бібліотеками;

– міграції (Migrations) містять класи, створені Entity Framework;

– репозиторії (Repository) містять методи для роботи з базою даних;

– інтерфейси (Interfaces) використовуються для реалізації Dependency Injection;

– відображення (Mappings) допомагає перетворювати одні сутності в інші;

– винятки (Exceptions) включають класи для систематизації обробки помилок;

– розширення (Extensions) включає класи для налаштування ініціалізації identity у хості програми;

– SpotifyApi включає класи та сервіси для взаємодії зі сторонніми сервісами Spotify;

– MailKit включає класи та сервіси для взаємодії з сервісами відправки повідомлень поштою.

3.1.3 UML діаграма розгортання

Програмна система рекомендацій музики на основі вподобань групи користувачів має декілька компонентів (див. рис. 3.4):

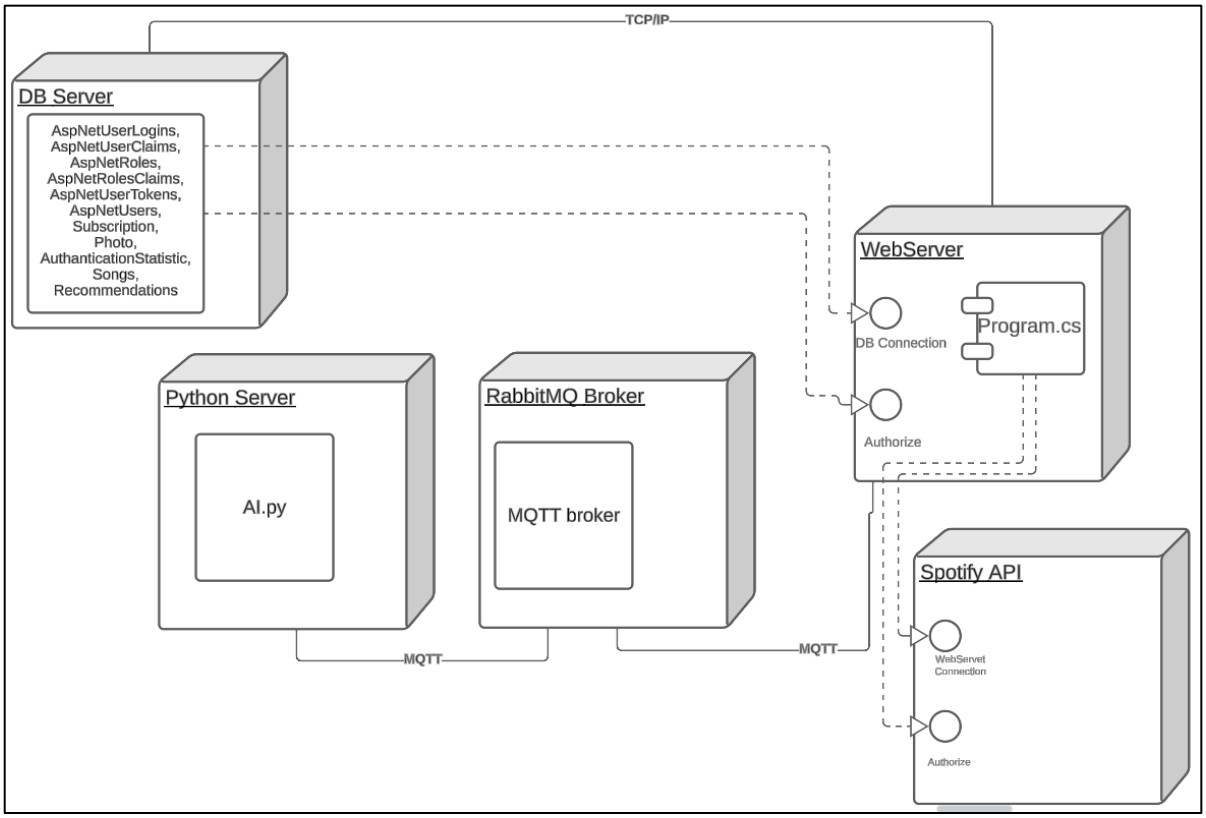


Рисунок 3.4 – UML діаграма розгортання проекту «Програмна система рекомендації музики на основі вподобань групи користувачів. Back-end» (рисунок виконаний самостійно)

Як можемо побачити, програма back-end частини складається з п'яти компонентів:

– база даних;

– Web API сервер;

– сервер штучного інтелекту (Python Server);

– брокер повідомлень RabbitMQ;

– Spotify API сервіс.

3.2 Архітектура системи

Програмна система для рекомендації музики на основі вподобань групи користувачів має 3-рівневу архітектуру. Це розділяє серверну частину на три логічних рівні, а саме:

Рівень доступу даних. Саме він відповідає за взаємодію з базою даних Azure для зберігання даних у відповідних контейнерах. Реалізує патерн Repository для централізованого доступу до функціоналу взємодії з бд. Надає дані до рівню бізнес-логіки.

Рівень бізнес-логіки. Обробляє запити з рівня доступу даних та надсилає їх до рівня презентації. Цей рівень містить в собі сервіси для відправки та приймання повідомлень через брокер RabbitMQ, взаємодії з сервісами Azure та Spotify. Цей рівень готує дані для подальшої передачі на презентаційний рівень.

Презентаційний рівень. Цей рівень використовується для взаємодії з клієнтською частиною для отримання та видачі даних. Він не несе під собою ніякої бізнес логіки.

Програма також має проект, який включає серію тестів для перевірки критичних частин програми. Вони налаштовані за допомогою технології мокування з використанням фреймворку xUnitTests. Такі тести перевіряють правильність роботи окремих компонентів програми – шару доступу до даних, бізнес-логіки та ескізів презентаційного шару. Програма може емулювати можливі сценарії взаємодії з Spotify та Azure завдяки мокам, які дозволяють перевірити, що обробка даних виконується правильно. Усі ці тести сприяють високій якості програмного забезпечення, оскільки воно стає більш стабільним та надійним – більшість слабких місць виявляються та усуваються, забезпечуючи загальну правильність.

3.2.1 Специфікація Rest

Програмний застосунок «Програмна система рекомендації музики на основі вподобань групи користувачів. Back-end» складається з 21 ендпоінту. Специфікація кожного з ендпоінту (див. Таблиця 3.1).

Таблиця 3.1– Специфікація REST API (таблиця виконана самостійно)

Посилання на ендпоінт	Метод	Опис
/api/Authentication/Registration	Post	Реєстрація акаунту
/api/Authentication/Login	Post	Авторизація та аутентифікація
/api/Authentication/mail-confirmation	Get	Підтвердження пошти
/api/Admin/BanUser/{userId}	Patch	Блокування/Розблокування користувача
/api/Admin/AddRole	Patch	Додавання ролі
/api/Admin/DeleteRole	Patch	Видалення ролі
/api/Admin/GetUserList	Get	Отримання переліку користувачів
/api/Azure/GetUserInfo	Get	Отримання відповіді від порталу Azure
/api/EmailSender	Post	Відправка посилання на пошту
/api/Payment/Create	Post	Створення оплати
/api/Payment/GetAllProducts	Get	Отримання продуктів
/api/Song	Get	Пошук пісні зі списку
/api/Song/get-recommendations	Get	Отримання рекомендації від ШІ
/api/Song/get-last-recommendations	Get	Отримання останніх рекомендацій користувача
/api/Song/rating	Post	Виставлення рейтингу
/api/Statistic/GetUserInfo	Put	Статистика по юзерам

Кінець таблиці 3.1

Посилання на ендпоінт	Метод	Опис
/api/Statistic/GetLoginInfo	Put	Статистика авторизацій/реєстрацій
/api/Statistic/GetRecommendationInfo	Get	Статистика рекомендацій
/api/Statistic/GetUserInfo	Put	Статистика по юзерам
/api/Statistic/GetLoginInfo	Put	Статистика авторизацій/реєстрацій
/api/Statistic/GetRecommendationInfo	Get	Статистика рекомендацій
/api/User/GetUserInfo	Get	Отримання інформації про користувача
/api/User/DeleteAccount	Delete	Видалення аккаунту
/api/User/UpdatePassword	Patch	Оновлення паролю
/api/User/UpdateUserInfo	Put	Оновлення інформації про юзера
/api/User/ChangeAvatar	Put	Оновлення аватару

Кожен ендпоінт налаштований з окремими поведінками, які дозволяють користувачам взаємодіяти з системою та отримувати персоналізовані рекомендації, керувати відображенням та профілем, швидко отримувати рекомендації з музичного контенту та покращувати аналіз даних для надання кращого сервісу.

Ця модель бізнес логіки дозволяє забезпечити надійну та ефективну роботу системи, гарантуючи високу продуктивність та масштабованість застосунку «Melody Fusion». Серверна частина системи виконує ключові функції з обробки даних, взаємодії з користувачами та зовнішніми сервісами, що робить її центральним компонентом успішної роботи програмного продукту.

Відображення частини ендпоінтів можна побачити на рисунку 3.5.

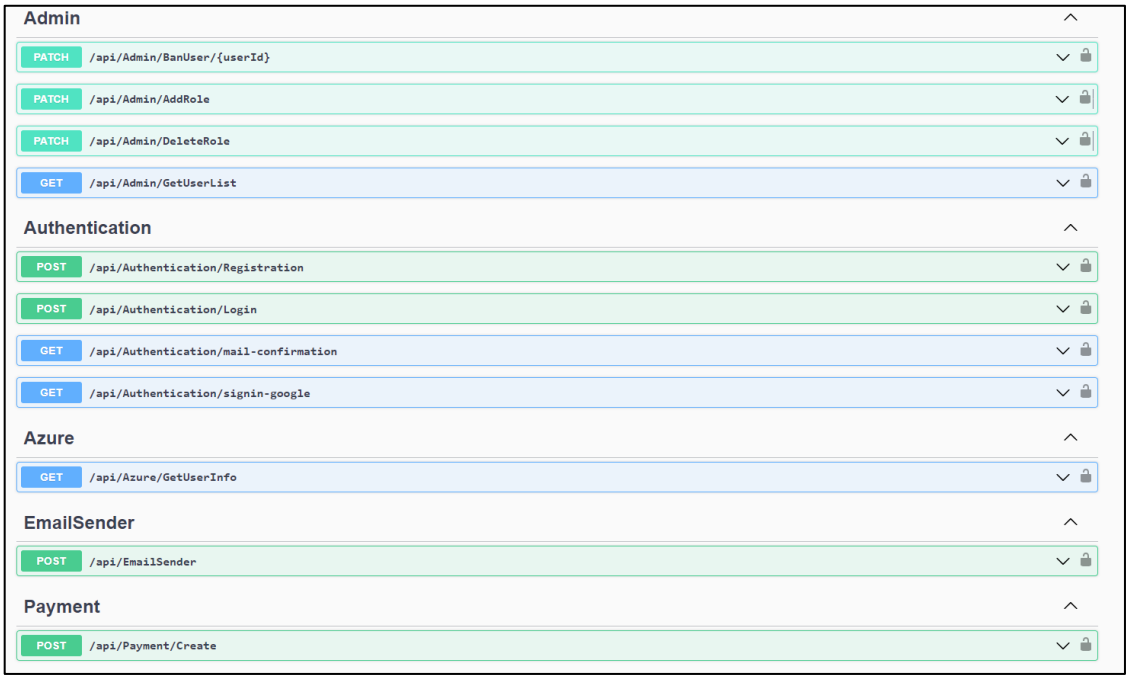


Рисунок 3.5 – Частина ендпоінтів програмної системи рекомендації музики на основі вподобань групи користувачів. Back-end (рисунок виконаний самостійно)

Рисунок показує основні ендпоінти адміністрування та авторизації на у застосунку.

3.3 Проєктування структури зберігання даних

Для проектування бази даних була розроблена діаграма. Вона складається з 11 таблиць (див. рис. 3.6).:

– identity таблиці бази даних: AspNetUserLogins, AspNetUserClaims, AspNetRoles, AspNetRolesClaims, AspNetUserTokens, AspNetUsers;

– власні таблиці бази даних: Subscription, Songs, Photo, AuthenticationStatistic, Recommendations.

Кожна з власних таблиць має наступні поля «CreatedDate» та «LastModifiedDate», які додані для ведення статистичних даних.

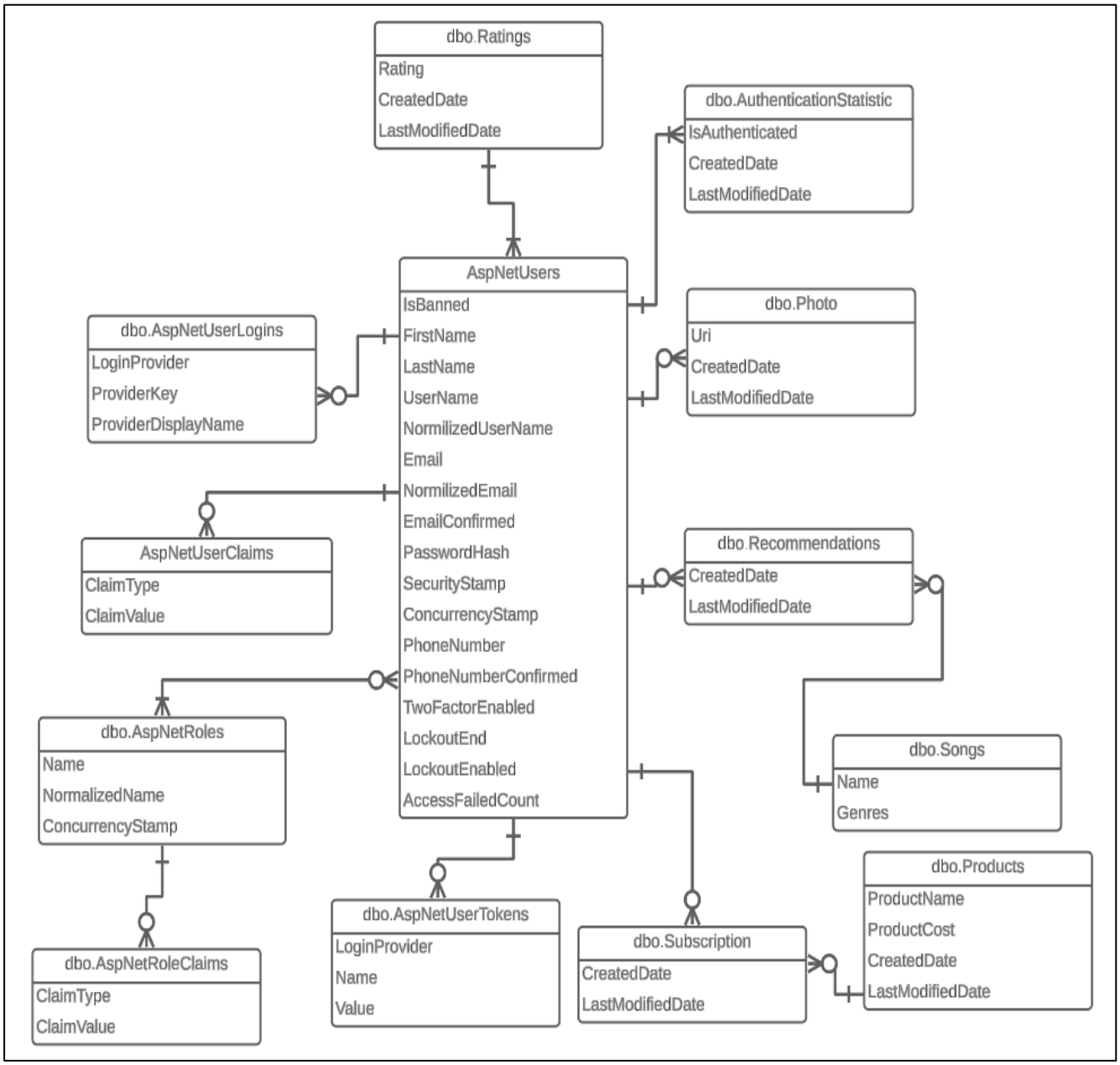


Рисунок 3.6 – ER-модель даних проекту «Програмна система рекомендації музики на основі вподобань групи користувачів. Back-end» (рисунок виконаний самостійно)

Модель є нормалізованою базою даних, що надає цілісність та оптимізацію структури, забезпечує ефективність запитів, надає зручність у майбутній розробці та підтримці [7].

3.4 Приклади найцікавіших алгоритмів та методів

3.4.1 Передумови використання технології

Під час розробки програмного забезпечення виникає питання про те, як воно буде точно вибирати музику, враховуючи уподобання кількох осіб. Для цього використання штучного інтелекту (ШІ) є відповідним, оскільки він може надавати рекомендації з певною точністю на основі наборів даних, представлених у форматі XML або JSON файлів.

Штучний інтелект (ШІ) може аналізувати великі обсяги даних, включаючи індивідуальні музичні переваги, історію прослуховування та навіть контекст, у якому зазвичай слухається музика. Використовуючи алгоритми машинного навчання, ШІ може виявляти закономірності та відносини в даних, які не завжди очевидні для людського аналізу.

3.4.2 Опис підходу розробки

Проаналізувавши сучасні технології рекомендаційних систем представлених у відкритому доступі та наявних баз даних, релевантних для роботи з сучасною музикой, було вирішено використовувати фокус на Content-Based методах, які особливо підходять для цієї задачі.

Фільтрація на основі контенту ґрунтується на ідеї, що користувачам сподобаються елементи, які мають схожі атрибути чи характеристики з елементами, які їм сподобалися раніше. Наприклад, якщо вам сподобалася комедійна стрічка з певним актором, вам можуть сподобатися інші фільми того ж жанру з цим самим актором. Фільтрація на основі контенту використовує інформацію про самі елементи, таку як їх жанри, характеристики або описи, для формування рекомендацій для конкретного користувача. Вона не вимагає жодної інформації про інших користувачів, їхні оцінки або взаємодії.

За основу тестової бази даних була взята робота [8], яка містила в собі п'ять тисяч сучасних пісень зі стрімінгової платформи Spotify.

3.4.3 Аналіз тестової бази даних Spotify

Тестова база даних Spotify містить в собі наступні поля:

– id: ідентифікаційний номер, який співпадає з ідентифікаційним номером у Spotify;

– name: назва трека;

– artists: перелік імен артистів;

– genres: жанри;

– release Year: дата випуску пісні;

– duration: тривалість пісні у мілісекундах;

– popularity: популярність на стрімінговій платформі Spotify;

– dancebility: танцювальність пісні в балах;

– acousticness: Чи пісня/трек акустичний чи ні;

– tempo: Темп пісні/треку, виміряний у вдарах за хвилину (BPM);

– energy: Як енергійна пісня/трек;

– liveness: Чи присутній живий аудиторіум, чи пісня/трек записана в студії;

– valence: Наскільки позитивна музика;

Серед полів є як числові так і нечислові значення. Щоб зрозуміти якість обраних даних, було проведено ретельний аналіз, прибрані нульові значення.

Для покращення коефіцієнту правильності відповідей користувачеві були опрацьовані усі записи та видалені нульові та повторюванні значення.

Після ретельного аналізу та видалення нульових значень, дані в тестовій базі даних Spotify стали більш відповідними для подальшого використання в системі рекомендацій музики. Кожне поле має своє значення і сприяє уточненню характеристик треків, які використовуються для генерації рекомендацій.

Якщо детально розглядати статистичні дані по всіх чисельних параметрах, то можна побачити наступне (див. рис. 3.7):

Result_dataset

Feature	Count	Unique Value Count	Missing Value Count	Min	Max	Mean	Mean Deviation	1st quantile	Median	3rd quantile	Mode	Range
id	5001	5001	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
name	5001	4760	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
artists	5001	3374	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
id_artists	5001	3375	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
genres	4935	2952	66	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
release_year	5001	68	0	1949	2021	2010.737253	9.875136	2007	2017	2019	2020	72
duration_s	5001	4538	0	52.6	682.817	220.043656	41.004798	185.08	212.493	245.64	{180.0, 180.64}	630.217
popularity	5001	6	0	67	72	69.444911	1.099791	68	69	70	68	5
danceability	5001	725	0	0.0	0.975	0.636241	0.123906	0.538	0.651	0.748	0.748	0.975
energy	5001	852	0	2.03e-05	1.0	0.639142	0.157314	0.512	0.659	0.788	0.833	0.9999797
key	5001	12	0	0	11	5.179364	3.097673	2	5	8	1	11
loudness	5001	3856	0	22.736	61.509	53.076804	2.380564	51.744	53.718	55.265	54.74	38.772999999999996
mode	5001	2	0	0	1	0.634273	0.463941	0	1	1	1	1
speechiness	5001	1065	0	0.0	0.884	0.099265	0.073337	0.0364	0.055	0.116	0.0334	0.884
acousticness	5001	1946	0	4.76e-06	0.995	0.260218	0.214649	0.0422	0.172	0.409	0.114	0.99499524
instrumentalness	5001	1780	0	0.0	1.0	0.029813	0.053333	0	0	0.000159	0.0	1.0
liveness	5001	1074	0	0.0134	0.989	0.179789	0.10316	0.0931	0.122	0.221	0.109	0.9756
valence	5001	968	0	0.0	0.989	0.522621	0.198725	0.342	0.52	0.712	{0.354, 0.45, 0.491, 0.543, 0.591, 0.836}	0.989
tempo	5001	4557	0	0.0	220.065	122.426865	24.080319	98.243	121.493	140.974	119.964	220.065

Рисунок 3.7 – Статистика тестової бази даних (рисунок виконаний самостійно)

Дати випуску пісні різняться від 1949 року до 2021, але середнє значення це 2010 рік, отже дані у БД відносно нові. Популярність треку має шкалу від 1 до 100, де 1 це мінімальний, а 100 це максимальний рейтинг. Середня популярність є дуже близькою к двом з границь, тож пісні не мають великої розбіжності, що є дуже добрим результатом. Інші дані також показують, що усі вони мають приблизно однакову розбіжність, наприклад: Key, Mode, Valence, Tempo та інші [9].

3.4.4 Математичні алгоритми розробки

Для правильної роботи з БД, з неї було видалено текстові поля, які не мають під собою ніякого корисного значення для проекту. Такими полями були: ім’я артиста, його ідентифікаційний номер, назва пісні. Текстове поле жанрів має деякий вплив на результат, тож це поле було переформатовано в масив строк, який містить їх перелік . Усі інші поля були обрані як ті, що підходять для роботи.

Для вираховування рекомендації за основу взята векторна модель простору [10]. У цій моделі кожен елемент представляється у вигляді вектора його характеристик у n-вимірному просторі, і кут між векторами обчислюється для визначення схожості між ними [11].

Косинусна міра подібності широко використовується для визначення близькості двох векторів, представлених за цією моделлю, і, в кінцевому підсумку, дає міру схожості між векторами.

Формула для розрахунку косинусної подібності наведено у формулі 3.1.

$$similarity = \frac{A*B}{||A||*||B||}, \tag{3.1}$$

де A * B – скалярний добуток векторів A і B;

||A|| * ||B|| – їхні норми.

Для векторів A=(a_1,a_2,...,a_n) і B=(b_1,b_2,...,b_n), формула скалярного добутоку наведено у формулі 3.2.

$$A \cdot B = a_1 \cdot b_1 + a_2 \cdot b_2 + ... + a_n \cdot b_n, \tag{3.2}$$

де a_n – компонент вектору A;

b_n – компонент вектору B.

Норма вектора A обчислюється як наведено у формулі 3.3.

$$||A|| = \sqrt{a_1^2 + a_2^2 + ... + a_n^2}, \tag{3.3}$$

де a_n – компонент вектору A.

Таким чином, косинусна подібність вимірюється як косинус кута між двома векторами і може приймати значення в діапазоні від – 1 (абсолютно протилежні вектори) до 1 (ідентичні вектори).

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Опис баз даних

База даних проекту є дуже важливою складовою для подальшої успішності проекту. На розгляді до потенційної платформи були декілька варіантів:

Локальна реляційна база даних Microsoft SQL Server. Вона відома своєю надійністю, високою швидкодією та широким спектром функцій. Використання локальної реляційної бази даних Microsoft SQL Server може забезпечити стабільну та ефективну роботу проекту, зручну інтеграцію з іншими продуктами Microsoft, такими як .NET Core. Перевага також в тому, що дана платформа використовувалась в минулих проектах багато разів та э дуже добре вивченою;

Локальна реляційна база даних MySQL Server. Як і Microsoft SQL Server є однією з найпопулярніших відкритих реляційних баз даних у світі. Вона відома своєю простотою в управлінні, високою продуктивністю та широкою підтримкою різних платформ. Була відкинута у використанні через нестачу знань у її експлуатації;

Локальна нереляційна база даних MongoDB. Є дуже гнучкою базою даних, простою у масштабуванні для зберігання даних у вигляді документів в JSON форматі. Важливою перевагою є швидкий пошук на великому наборі даних;

Хмарова реляційна база даних Azure SQL DB. Пропонується Microsoft Azure. Вона забезпечує високий рівень доступності, масштабованість та безпеку даних. Використання хмарної реляційної бази даних Azure SQL DB може забезпечити гнучкість та легкість управління, а також інтеграцію з іншими хмарними сервісами Azure. Має велику кількість функцій направлених на безпеку даних, ведення логування та статистки, легке масштабування;

Хмарова нереляційна база даних Azure (blob-сховище). Це одне із ключових сервісів, які пропонуються платформою Microsoft Azure. Це сховище даних призначене для зберігання великих об'єктів, таких як зображення, відео, аудіо, документи та інші нереляційні дані.

Серед наведених баз даних були обрані три з них: Azure SQL DB, Microsoft SQL Server та нереляційна база даних Azure (blob-сховище).

Цілі використання кожної з них в наступному:

– використання локального серверу Microsoft SQL Server для модульного та навантажувального тестування системи. Не використовує гроші для підтримки, але потрібна сильна локальна машина для опрацювання реального великого потоку запитів від клієнтів;

– використання : Azure SQL DB для хмарного зберігання та опрацювання даних. Ідеально підходить для опрацювання великого потоку запитів від клієнтів та гнучкого масштабування;

– використання Azure (blob-сховища) для хмарного зберігання великих за розміром даних та їх швидкого пошуку. Ідеальний варіант для зберігання фотокарток юзерів.

Структура реляційних баз даних ідентична як на локальній машині так і на хмарному сховищі (див. рис. 4.1).

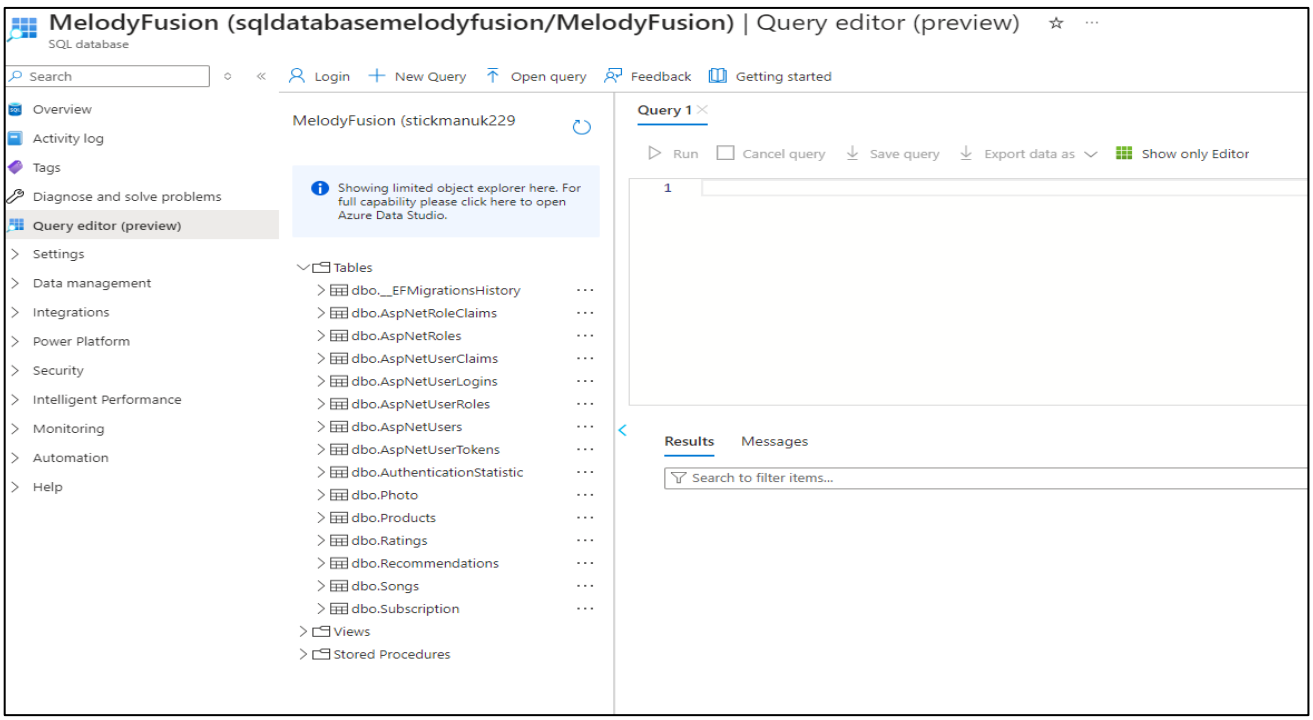


Рисунок 4.1 – Таблична структура реляційних баз даних

(рисунок виконаний самостійно)

Також цікавою є структура нереляційної бази даних, а саме blob-сховища (див. рис. 4.2).

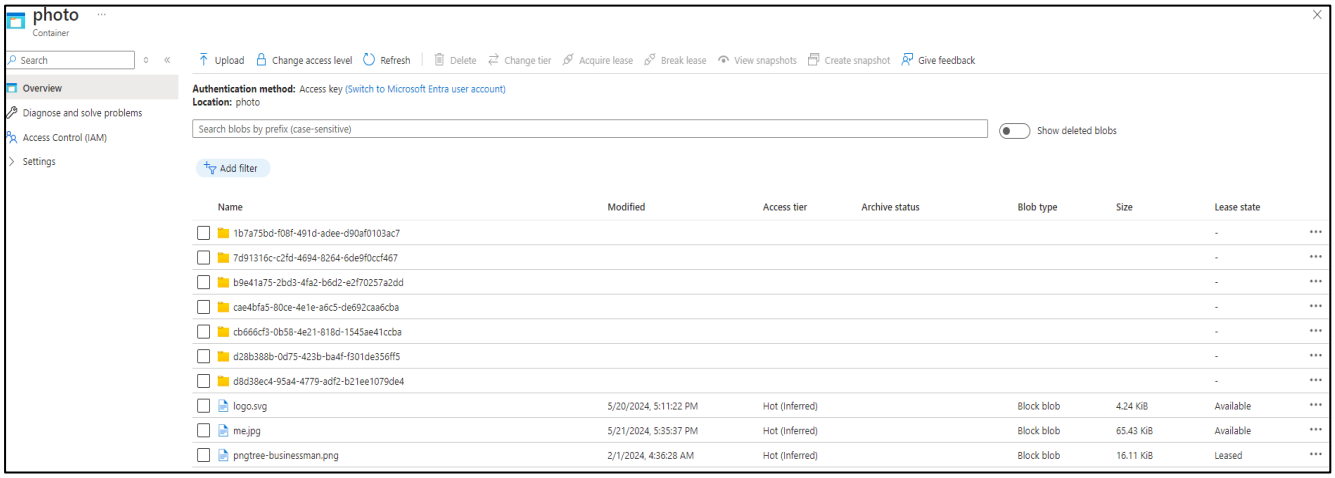


Рисунок 4.2 – Структура нереляційної бази даних
(рисунок виконаний самостійно)

Структура сховища розділена на папки, які належать кожному окремому юзеру. Це сприяє швидкому пошуку інформації в разі потреби. Основні картинки, такі як логотипи, стандартні набори аватарів та інше знаходяться у корні сховища.

4.1.1 Оптимізація роботи бази даних

Для швидкою робити реляційних баз даних були написані некластерні індекси для швидкої роботи з пошуком пісень. Таблиця з переліком музичних творів містить велику кількість даних з самого початку роботи застосунку. Код некластерного індексу виглядає наступним чином (див. рис. 4.3):

```
CREATE NONCLUSTERED INDEX IDX_Songs_Name ON Songs (name);
```

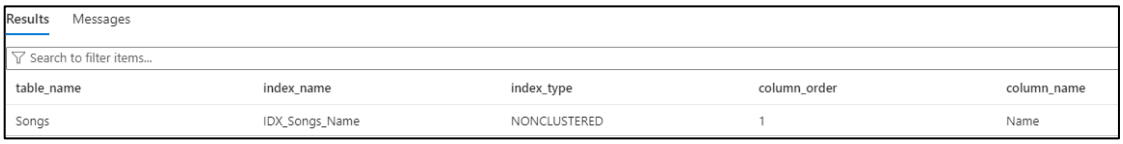


Рисунок 4.3 – Відображення некластерного індексу у Azure
(рисунок виконаний самостійно)

Індекс створений для швидкої праці з найбільш використовуваною функціональністю сайту: пошук пісні за картинкою.

4.2 Опис серверної частини застосунку

Серверна частина застосунку написана на .NET 7. Проект розроблений як Web API частина для взаємодії його з іншими частинами проекту.

4.2.1 Опис роботи з базою даних

Робота з базою даних зроблена на окремому рівні – рівні «Data layer». Розділення застосунку на окремі рівні є важливим принципом архітектури програмного забезпечення, який дозволяє досягти кращої організації коду, зручності в підтримці та тестуванні застосунку. Основні аспекти роботи з Data Layer, його структура та принципи взаємодії будуть описані далі.

Основним принципом Data Layer є інкапсуляція доступу до даних, що служить абстракцією для створення запитів на зчитування, додавання, оновлення та видалення інформації. Це дозволяє змінювати способи зберігання даних, але не міняти сам код програми. Також такий підхід покращує як читабельність коду, так і спрощує тестування – як юніт так і навантажувального тестування.

Наважливішою компонентою в роботі є контекст бази даних (DbContext). За допомогою Entity Framework були налаштовані зв’язки між таблицями. Приклад налаштування зв’язку у коді:

```
public class UserEntityConfiguration : IEntityTypeConfiguration<UserDto>
 {
     public void Configure(EntityTypeBuilder<UserDto> builder)
     {
        builder.HasMany(ur => ur.UserRoles)
             .WithOne(u => u.UserDto)
             .HasForeignKey(x => x.UserId)
             .OnDelete(DeleteBehavior.Cascade);
        builder.HasMany(x => x.Recommendations)
             .WithOne(x => x.User)
             .HasForeignKey(x => x.UserId)
             .OnDelete(DeleteBehavior.Cascade);}}
```

Налаштування всього проекту має загальний вигляд (див. рис. 4.4):

```
Ссылок: 2
public DbSet<SubscriptionDto> Subscription { get; set; }
Ссылок: 2
public DbSet<PhotoDto> Photo{ get; set; }
Ссылок: 2
public DbSet<AuthenticationStatisticDto> AuthenticationStatistic { get; set; }
Ссылок: 2
public DbSet<Recommendations> Recommendations { get; set; }
Ссылок: 2
public DbSet<Product> Products { get; set; }
Ссылок: 2
public DbSet<Ratings> Ratings { get; set; }
Ссылок: 0
public DbSet<Songs> Songs { get; set; }


Ссылок: 0
public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)
{
    Database.Migrate();
}

Ссылок: 0
protected override void OnModelCreating(ModelBuilder builder)
{
    base.OnModelCreating(builder);

    builder.ApplyConfiguration(new RoleEntityConfiguration());
    builder.ApplyConfiguration(new UserEntityConfiguration());
    builder.ApplyConfiguration(new SubscriptionEntityConfiguration());
    builder.ApplyConfiguration(new PhotoEntityConfiguration());
    builder.ApplyConfiguration(new AuthenticationStatisticEntityConfiguration());
    builder.ApplyConfiguration(new SongEntityConfiguration());
    builder.ApplyConfiguration(new ProductEntityConfiguration());
    builder.ApplyConfiguration(new RatingEntityConfiguration());
}
```

Рисунок 4.4 – Демонстрація налаштування зв’язків таблиць бази даних (рисунок виконаний самостійно)

Серед демонстрованих вище конфігурацій є налаштування ролей, користувачів, підписок, фотографій, статистики, пісень, продуктів та рейтингів.

Для коректної взаємодії з базою даних, а саме для створення запитів, використаний патерн проектування «Репозиторій». Цей патерн дозволяє створити абстракцію над джерелом даних, спрощуючи доступ до бази даних і знижуючи залежності між різними компонентами застосунку.

Переваги використання патерну:

– інкапсуляція логіки доступу до даних;

– зручність тествування;

– покращена організація коду;

– легкість у зміні сховища даних.

Інерфейс цього репозиторія можна побачити на наступному фото (див. рис. 4.5):

```
namespace MelodyFusion.DLL.Interfaces
{
    Ссылок: 8
    public interface IRepository<TEntity> where TEntity : BaseEntity
    {
        Ссылок: 2
        Task<List<TEntity>> GetAllAsync();
        Ссылок: 3
        Task<List<TEntity>> GetAsync(Expression<Func<TEntity, bool>> predicate);

        Ссылок: 4
        Task<List<TEntity>> GetAsync(Expression<Func<TEntity, bool>>? predicate = null,
            Func<IQueryable<TEntity>, IOrderedQueryable<TEntity>> orderBy = null,
            string? includeString = null,
            bool disableTracking = true);

        Ссылок: 4
        Task<List<TEntity>> GetAsync(Expression<Func<TEntity, bool>>? predicate = null,
            Func<IQueryable<TEntity>, IOrderedQueryable<TEntity>>? orderBy = null,
            List<Expression<Func<TEntity, object>>>? includes = null,
            bool disableTracking = true);

        Ссылок: 8
        Task<TEntity?> GetByIdAsync(string id);
        Ссылок: 14
        Task<TEntity> AddAsync(TEntity entity);
        Ссылок: 7
        Task UpdateAsync(TEntity entity);
        Ссылок: 1
        Task DeleteById(string id);
        Ссылок: 2
        Task DeleteAsync(TEntity entity);
    }
}
```

Рисунок 4.5 – Інтерфейс репозиторію (рисунок виконаний самостійно)

Цей інтерфейс імплементує сім класів, тобто усі власно створені моделі:

– репозиторій статистики;

– репозиторій фото;

– репозиторій продуктів;

– репозиторій рейтингів;

– репозиторій рекомендацій;

– репозиторій підписки;

– репозиторій пісень.

Код реалізованого запиту додавання продукту до бази даних:

```
public override async Task<Product> AddAsync(Product entity)
{
    await using var transaction = await
_context.Database.BeginTransactionAsync();
    try
    {
        await _context.AddAsync(entity);
        await _context.SaveChangesAsync();
        await transaction.CommitAsync();
        return entity;
    }
    catch (Exception e)
    {
        await transaction.RollbackAsync();
        _logger.LogError("Error while creating product: {EMessage}",
e.Message);
    }
    return new Product();
}
```

Усі запити огорнуті в транзакцію для їх надійності та цілісності передачі даних.

4.2.2 Опис роботи з брокером повідомлень RabbitMQ

Основна функція роботи з модулем брокеру повідомлень – це забезпечення асинхронної передачі повідомлень, що дозволяє ефективно розподіляти завдання та покращувати масштабованість системи.

В системі проекту «Melody Fusion» він використовується для відправки запитів на рекомендацію пісень на сервер штучного інтелекту. Це забезпечує безперебійну роботу застосунку, розвантажуючи основні процеси та дозволяючи швидко та ефективно отримувати музичні рекомендації.

Серед існуючих модулів передачі використовуються черги (Queues). Вони працюють за принципом FIFO (перший прийшов – перший пішов). Повідомлення зберігаються в чергах, доки їх не буде отримано та оброблено споживачем (див. рис. 4.6).

```
Ссылок: 2
public async Task SendMessage(object obj)
{
    var message :string = JsonSerializer.Serialize(obj);
    await SendMessage(message);
}

Ссылок: 2
public async Task SendMessage(string message)
{
    var factory = new ConnectionFactory() { Uri = new Uri(_config.GetValue<string>(key: "RabbitMQ:Host")) };
    using (var connection = factory.CreateConnection())
    using (var channel :IModel? = connection.CreateModel())
    {
        channel.QueueDeclare(queue: _config.GetValue<string>(key: "RabbitMQ:QueueRequest"),
            durable: false,
            exclusive: false,
            autoDelete: false,
            arguments: null);

        var body :byte[] = Encoding.UTF8.GetBytes(message);

        channel.BasicPublish(exchange: "",
            routingKey: _config.GetValue<string>(key: "RabbitMQ:QueueRequest"),
            basicProperties: null,
            body: body);
    }
}
```

Рисунок 4.6 – Частина коду з відправки повідомлень на брокер
(рисунок виконаний самостійно)

Пересилання виконується об’єктів у вигляді JSON строки на окрему чергу з відправки повідомлень.

4.2.3 Опис роботи бізнес логіки серверної частини

Бізнес логіка серверної частини системи є центральною компонентою, яка відповідає за обробку та управління даними, реалізацію правил та процесів, що визначають функціонування застосунку. У випадку проекту «Melody Fusion», серверна частина обробляє запити від клієнтів, здійснює взаємодію з базою даних через рівень доступу до БД, зовнішніми сервісами та брокером повідомлень RabbitMQ для забезпечення рекомендацій музики на основі запитів користувачів.

У роботі широко використовується патерн «Впровадження залежностей». Dependency Injection (DI) – це патерн проектування, який дозволяє об'єктам отримувати залежності від зовнішнього джерела, замість того, щоб створювати їх

самостійно. DI сприяє слабкому зв'язуванню компонентів у системі, що покращує тестованість, зручність у супроводі та гнучкість коду.

Для налаштування роботи проекту також використовуються DI-контейнери. Код налаштування контейнеру бізнес логіки наведений нижче:

```
public static IServiceCollection AddBusinessLayer(this
IServiceCollection services,IConfiguration configuration)
{
    services.AddDataLayer(configuration);
    services.AddAutoMapper(Assembly.GetExecutingAssembly());
    services.AddScoped<IRoleInitializer, RoleInitializer>();
    services.AddScoped<IProductInitializer, ProductInitializer>();
    services.AddScoped<IRegistrationService, RegistrationService>();
    services.AddScoped<ILoginService, LoginService>();
    services.AddScoped<IUserService, UserService>();
    services.AddScoped<IAdminService, AdminService>();
    services.AddTransient<IBraintreeService, BraintreeService>();
    services.AddScoped<IPaymentService, PaymentService>();
    services.AddScoped<IAzureBlobService, AzureBlobService>();
    services.AddScoped<IStatisticService, StatisticService>();
    services.AddScoped<IEmailSender, EmailSenderService>();
    services.AddScoped<IConfirmationEmailService,
ConfirmEmailService>();
    services.AddScoped<ISpotifyService, SpotifyService>();
    services.AddScoped<IRabbitMqService, RabbitMqService>();
    services.AddScoped<IRatingService, RatingService>();
    services.AddScoped<JwtHandler>();
    return services;
}
```

Кожен інтерфейс зв’язаний парою з класом, що його реалізує. Зроблено це для конструкторної ін’єкції. Конструкторна ін'єкція: Залежності передаються через конструктор класу. Це найбільш поширений метод, оскільки дозволяє чітко визначити необхідні залежності під час створення об'єкта.

4.3 Опис серверу штучного інтелекту

Сервер штучного інтелекту в проекті «Melody Fusion» відповідає за генерацію рекомендацій музичних треків на основі аналізу існуючого музичного контенту та вподобань користувачів. Цей сервер використовує кілька потужних

бібліотек Python для обробки даних, перетворення тексту у вектори та визначення подібності між музичними треками.

Для читання та обробки файлів з даними використовується бібліотека pandas. Вона дозволяє ефективно працювати з великими обсягами даних, зберігаючи їх у форматі DataFrame, що забезпечує зручний доступ і маніпуляції.

Функція song_recommender займається безпосередньо рекомендаціями пісень(див. рис. 4.7).

```
def song_recommender(first_song_id, second_song_id):
    try:
        num_cols = ['release_year', 'duration_s', 'popularity', 'danceability', 'energy', 'key', 'loudness',
                    'mode', 'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo']

        text_vec1 = song_vectorizer.transform(song_library[song_library['id'] == str(first_song_id)]['genres']).toarray()
        text_vec2 = song_vectorizer.transform(song_library[song_library['id'] == str(second_song_id)]['genres']).toarray()

        num_vec1 = song_library[song_library['id'] == str(first_song_id)][num_cols].to_numpy()
        num_vec2 = song_library[song_library['id'] == str(second_song_id)][num_cols].to_numpy()

        sim_scores = []

        for index, row in song_library.iterrows():
            name = row['id']

            text_vec_other = song_vectorizer.transform(song_library[song_library['id'] == name]['genres']).toarray()

            num_vec_other = song_library[song_library['id'] == name][num_cols].to_numpy()

            text_sim1 = cosine_similarity(text_vec1, text_vec_other)[0][0]
            text_sim2 = cosine_similarity(text_vec2, text_vec_other)[0][0]

            num_sim1 = cosine_similarity(num_vec1, num_vec_other)[0][0]
            num_sim2 = cosine_similarity(num_vec2, num_vec_other)[0][0]

            sim = (text_sim1 + num_sim1 + text_sim2 + num_sim2) / 4
            sim_scores.append(sim)

        song_library['similarity'] = sim_scores

        song_library.sort_values(by=['similarity', 'popularity', 'release_year'], ascending=[False, False, False],
                                 inplace=True)

        recommended_songs = song_library[['id']][2:7]

        return recommended_songs
```

Рисунок 4.7 – Алгоритм роботи штучного інтелекту
(рисунок виконаний самостійно)

Вона приймає два ідентифікатори пісень, які користувач прослухав, і обчислює схожість інших пісень з ними. Спочатку для кожної з вхідних пісень створюються текстові та числові вектори. Потім для кожної пісні у бібліотеці обчислюється середня подібність за допомогою косинусної подібності як для текстових, так і для числових векторів.

4.4 Опис взаємодії з Spotify API

Для надання найактуальнішої інформації про надані пісні використовується взаємодія з сервісом Spotify API. Використання цього API дозволяє отримувати детальну інформацію про треки, включаючи метадані, такі як назва, артист, альбом, жанр, популярність та інші атрибути, які використовуються для покращення рекомендаційного сервісу.

Код взаємодії з спотіфай наведений нижче.

```
    public async Task<SongSpotifyResponse> GetSpotifySongById(string
songId)
    {
        var spotifyClient = await this.GetSpotifyClient();
        var track = await spotifyClient.Tracks.Get(songId);

        var result = _mapper.Map<FullTrack,
SongSpotifyResponse>(track);

        return result;
    }

private async Task<SpotifyClient> GetSpotifyClient()
{
    var config = SpotifyClientConfig.CreateDefault();
    var clientId =
_config.GetValue<string>("SpotifyCredentials:Key");

    var request = new
ClientCredentialsRequest(_config.GetValue<string>("SpotifyCredentials
:Key"), _config.GetValue<string>("SpotifyCredentials:ClientSecret"));
    var response = await new
OAuthClient(config).RequestToken(request);

    return new SpotifyClient(config.WithToken(response.AccessToken));
}
```

Метод GetSpotifySongById приймає ідентифікатор пісні songId як параметр та асинхронно викликає метод GetSpotifyClient для отримання клієнта. Після цього за допомогою методу Tracks.Get об'єкта spotifyClient виконується запит до API для отримання інформації про трек з вказаним ідентифікатором. Отримані дані представляються у вигляді об'єкта FullTrack, який містить всі деталі про пісню.

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Для забезпечення надійності та продуктивності розробленого програмного забезпечення було обрано навантажувальне тестування. Це тестування дозволяє оцінити, як система поводиться під високим навантаженням, ідентифікувати можливі вузькі місця та гарантувати стабільну роботу під час пікових навантажень. Для проведення навантажувальних тестів було використано інструмент Apache JMeter, який є потужним і широко розповсюдженим засобом для такого роду тестування.

Apache JMeter дозволяє створювати сценарії, що імітують одночасні запити від великої кількості користувачів до серверної частини системи. Було проведено тестування критичних секцій системи, таких як обробка запитів на отримання рекомендацій, взаємодія з зовнішніми API, зокрема робота з брокером повідомлень RabbitMQ. Ці секції були обрані через їх важливість для забезпечення швидкої та коректної роботи всієї системи.

В ході тестування було змодельовано різні сценарії навантаження, включаючи поступове збільшення кількості запитів. Це дозволило оцінити стійкість та ефективність системи в умовах, наближених до реальних експлуатаційних.

Існує кілька типів навантажувального тестування, кожен з яких має свою специфіку та застосовується для різних цілей.

Навантажувальне тестування (Load Testing): Цей тип тестування допомагає визначити, як система поводиться під очікуваним навантаженням. Мета полягає в тому, щоб переконатися, що застосунок може обробляти певну кількість запитів за заданий проміжок часу без зниження продуктивності.

Стрес-тестування (Stress Testing): Використовується для визначення меж продуктивності системи, піддаючи її навантаженням, які перевищують очікувані рівні. Це допомагає ідентифікувати точки відмови та перевірити, як система відновлюється після збою.

Об'ємне тестування (Volume Testing): Зосереджується на обробці великих обсягів даних, щоб перевірити, як система справляється з значними розмірами даних у базі даних або великою кількістю записів.

Тестування продуктивності (Performance Testing): Включає оцінку швидкості, чуйності та стабільності системи під різними навантаженнями. Метою є виявлення потенційних проблем з продуктивністю до їх появи в реальних умовах.

Тестування стабільності (Stability Testing) або Тестування надійності (Reliability Testing): Перевіряє, чи може система стабільно працювати протягом тривалого періоду під постійним навантаженням.

Тестування пікового навантаження (Spike Testing): Імітує різкі сплески навантаження, щоб перевірити, як система справляється з раптовим збільшенням кількості запитів та чи може вона швидко адаптуватися до змін у навантаженні.

Для тестування було обрано саме тестування продуктивності (Performance Testing). Проект ще не дійшов до фінального етапу з можливістю розміщення його на платформи, тому цей вид підходить найбільшим чином.

В ході роботи було проведено кілька тестів продуктивності критичних секцій. Першою з них є перевірка на отримання користувачем даних про особистий профіль. Для тесту було обрано число в 100 одночасних підключень юзерів (див. рис. 5.1).

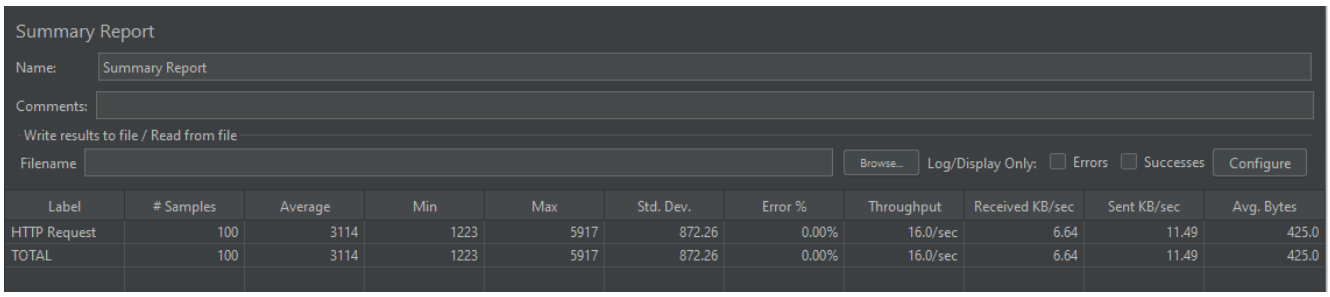


Рисунок 5.1 – Статистика навантажувального тесту отримання власної інформації (рисунок виконаний самостійно)

Як можна побачити, що збоїв у програмі ніяких не було, та середньо статистичний час відповіді від серверу даних доволі швидка для кожного юзера.

Наступний тест перевіряє пошук пісень за їх назвою. Було проведено два тести: до написання некластерного індексу до бази даних по полю імені пісні (див. рис. 5.2) та після його написання (див. рис. 5.3).

Рисунок 5.2 – Статистика навантажувального тесту без некластерного індексу (рисунок виконаний самостійно)

Рисунок 5.3 – Статистика навантажувального тесту з некластерним індексом (рисунок виконаний самостійно)

Як можна побачити, що збоїв у програмі ніяких не було, а середньо статистичний час відповіді від серверу даних виріс майже на 3 секунди.

Наступний тест перевіряє швидкість отримання статистичних даних, бо їх генерація є дуже ресурсномісткою операцією (див. рис. 5.4).

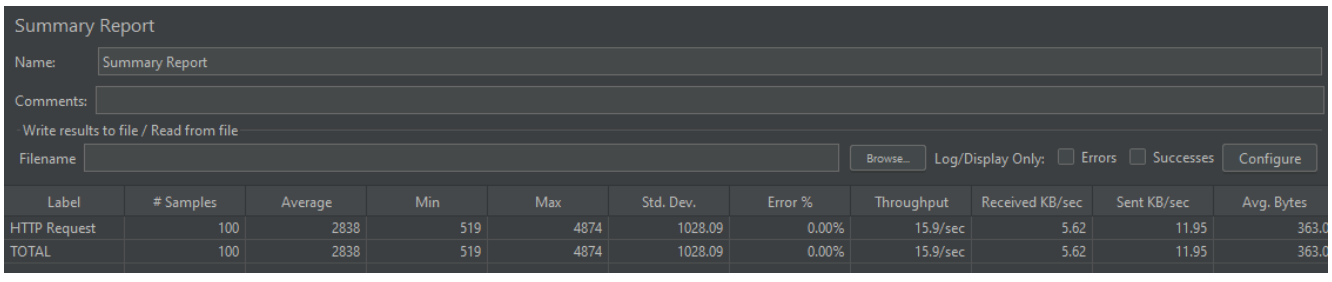


Рисунок 5.4 – Статистика навантажувального тесту отримання статистики (рисунок виконаний самостійно)

Через роботу з асинхронними блоками коду та оптимізованими запитами до БД, швидкість отримання цих даних доволі швидка та безпомилкова.

Останній тест навантажувального тестування був зроблений для перевірки головного функціоналу серверу, а саме рекомендація пісень. Він торкається декількох аспектів роботи застосунку, а саме:

– швидкість створення запиту від сервера та надсилання його на сервер;

– швидкість обробки отриманих з брокеру повідомлень даних сервером штучного інтелекту;

– швидкість отримання даних сервером та виведення їх у вигляді JSON повідомлення.

Через велику навантаженість на один комп'ютер двома серверами та брокером, кількість одночасних запитів була зменшена до десяти людей (див. рис. 5.5).

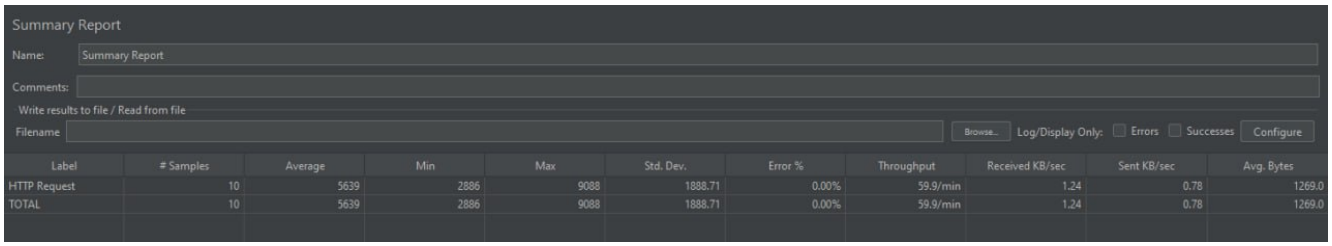


Рисунок 5.5 – Статистика навантажувального тесту отримання рекомендацій пісень (рисунок виконаний самостійно)

Таким чином, проведене навантажувальне тестування за допомогою Apache JMeter забезпечило високу якість та надійність розробленого програмного забезпечення, підтвердивши його готовність до роботи в умовах високих навантажень та великих обсягів даних.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи була розроблена програмна система рекомендації музики на основі вподобань групи користувачів, а саме її серверна частина та штучний інтелект.

Використані та закріплені навички у використанні наступних технологій: ASP.NET Web API, мові програмування C#, мові запитів до баз даних T-SQL, використані різні архітектурні підходи для структуризації, оптимізації та модифікації окремих компонентів. Проведена робота з обробкою даних від сторонніх сервісів, таких як Spotify API.

Були проаналізована література, зокрема робота Yamac Eren Ay, база даних якого була використана в роботі як тестова. Були проведені ознайомлені роботи з використаними технологіями за допомогою відповідних документацій.

Вирішена проблема з оптимізованим зберіганням та створенням запитів інформації на різних типах баз даних:

– реляційних – база даних для основного проекту на SQL;

– нереляційних – блоб сховище Azure для зберігання фотокарток.

Також були використані оптимізаційні платформи, а саме брокер повідомлень RabbitMQ та асинхронне програмування C#.

Були виконані усі поставлені задачі, а саме:

– створені алгоритми для аналізу музичних вподобань користувачів;

– створені механізми зберігання та обробки даних користувачів;

– забезпечені асинхронна обробка запитів;

– платформа інтегрована зі сторонніми сервісами (Spotify та Azure);

– забезпечений високий рівень безпеки та захисту користувачів за допомогою методів хешування паролів, jwt-авторизації та іншого;

– протестовані критичні секції та алгоритми за допомогою навантажувального тестування;

– використана підсистема управління базами даних SQL та blob-сховищ;

– створена система для обробки запитів користувачів (back-end);

– створена серверна частина для обробки запитів на надання рекомендацій музики (штучний інтелект);

– створена система відправки та отримання повідомлень для взаємозв'язку серверів штучного інтелекту та back-end частини.

Програмна система рекомендації музики на основі вподобань групи користувачів не є кінцевою версією продукту. В подальшому цей проект буде розвиватися і удосконалюватися для виходу на ринок.

Дана робота використовує передові технології сучасності, а саме штучний інтелект, для вирішення поставленої задачі, що дозволило розширити знання про розробку програмного забезпечення та практично застосувати їх у сфері рекомендаційних систем на основі схожих пісень за певними критеріями.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Youtube Music. URL: https://music.youtube.com/ (дата звернення: 03.06.2024).

2. Movie Recommendations for Date Night. URL: https://datenightmovies.com/ (дата звернення: 03.06.2024).

3. Introduction. MimeKit. URL: https://mimekit.net (date of access: 14.05.2024).

4. Web API | Spotify for Developers. Home | Spotify for Developers. URL: https://developer.spotify.com/documentation (дата звернення: 13.05.2024).

5. Dobovizki N. C# Concurrency. Lugano, Switzerland, 2023. 350 p.

6. RabbitMQ Documentation | RabbitMQ. RabbitMQ: One broker to queue them all | RabbitMQ. URL: https://www.rabbitmq.com/docs дата звернення: 13.05.2024).

7. Azure SQL documentation - Azure SQL. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/azure/azure-sql/?view=azuresql (дата звернення: 07.02.2024).

8. YAMAC E. A. Spotify Dataset 1921-2020, 600k+ Tracks. Kaggle: Your Machine Learning and Data Science Community. URL: https://www.kaggle.com/datasets/yamaerenay/spotify-dataset-19212020-600k-tracks/data (дата звернення: 07.03.2024).

9. Azure AI services documentation. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/azure/ai-services/ (дата звернення: 10.04.2024).

10. Yiting Yuan, Youyang Qin, Zekai Yu. A Content-based Movie Recommendation System. The International Conference on Computing Innovation and Applied Physics, Rochester, NY, 18 February 2022 (дата звернення: 10.04.2024).

11. A systematic review and research perspective on recommender systems - Journal of Big Data. SpringerOpen. URL: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-022-00592-5 (дата звернення: 02.04.2024).

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ

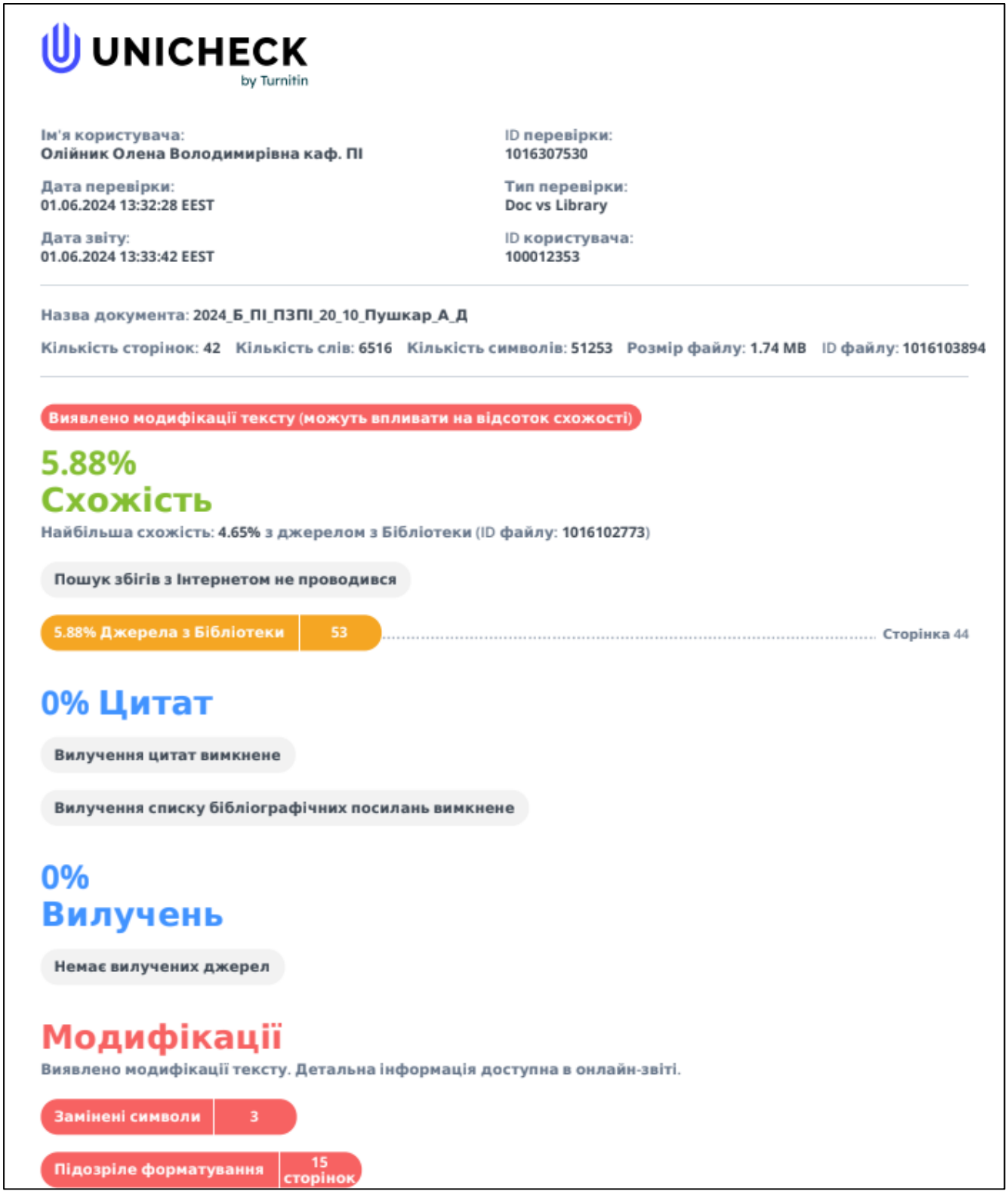


Рисунок А.1 – Результат перевірки на унікальність тексту

ДОДАТОК Б

Слайди презентації

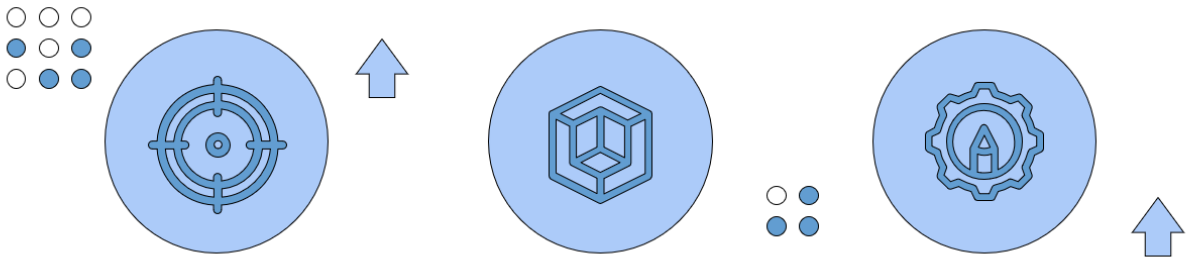
Харківський національний університет радіоелектроніки
Кваліфікаційна робота бакалавра

Програмна система рекомендацій музики на основі вподобань групи користувачів. Back-end

Виконав: студент 4 курсу, ПЗПІ-20-10 Пушкар А.Д.

Керівник: доц. кафедри ПІ Афанасьєва І.В.

1



Мета

Створення серверної частини застосунку для рекомендації музики

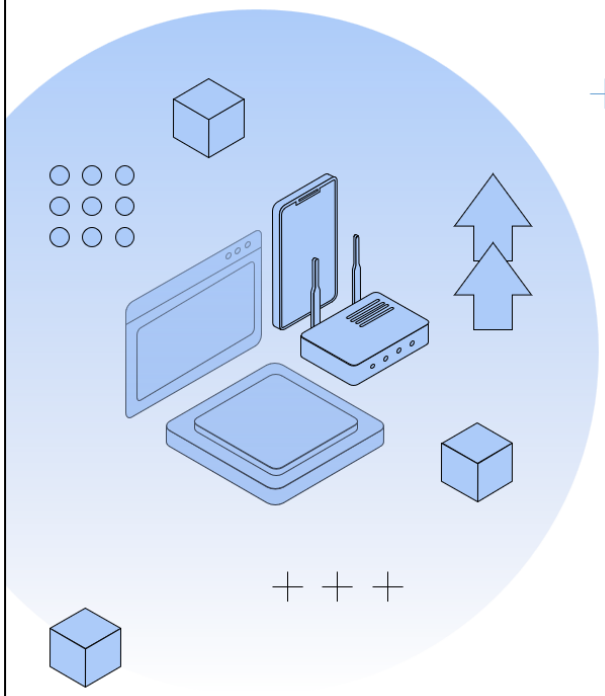
Об’єкт розробки

Програмна система рекомендації музики на основі вподобань користувачів

Метод рішення

C#, T-SQL, python 3.12, середовище розробки Visual Studio та платформа AZURE

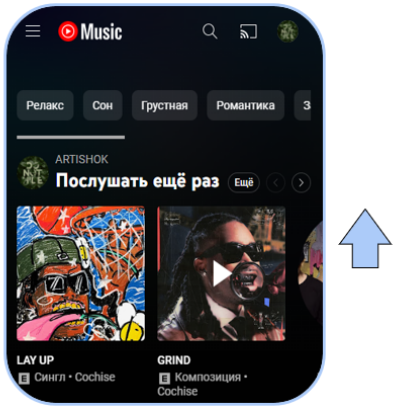
2



Предметна область

є злиттям рекомендаційних систем, інформаційних технологій та музикальної індустрії для забезпечення швидкого процесу рекомендації музики для компанії

Аналіз предметної галузі



Youtube music

Date Night Movie

Постановка задачі

Крок 1

Алгоритми аналізу музичних вподобань

Крок 2

Механізми зберігання та обробки даних користувачів

Крок 3

забезпечити асинхронну обробку запитів

Крок 4

Забезпечити інтеграцію зі Spotify та Azure

Крок 5

забезпечити високий рівень безпеки

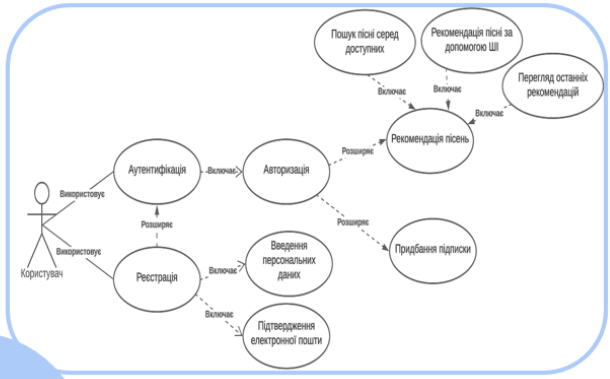
Крок 6

протестувати розроблені функції

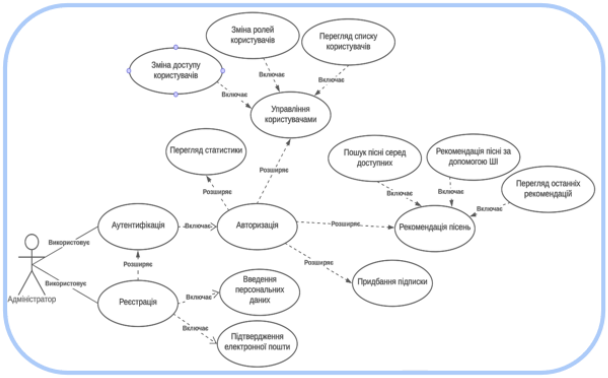
5

Ролі користувачів застосунку

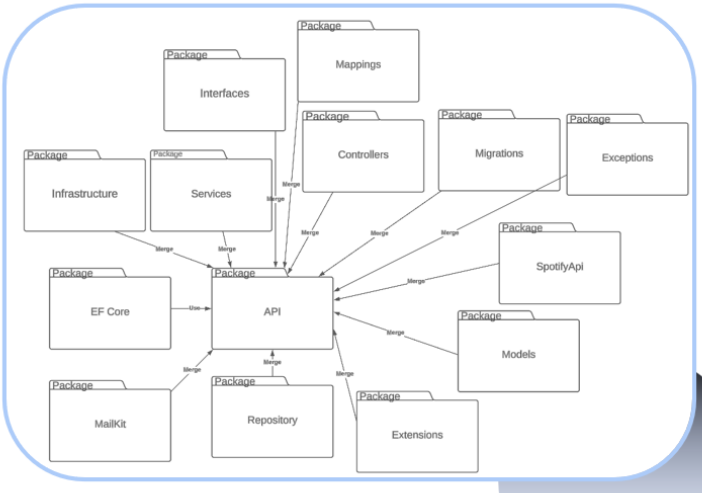
Користувач



Адміністратор

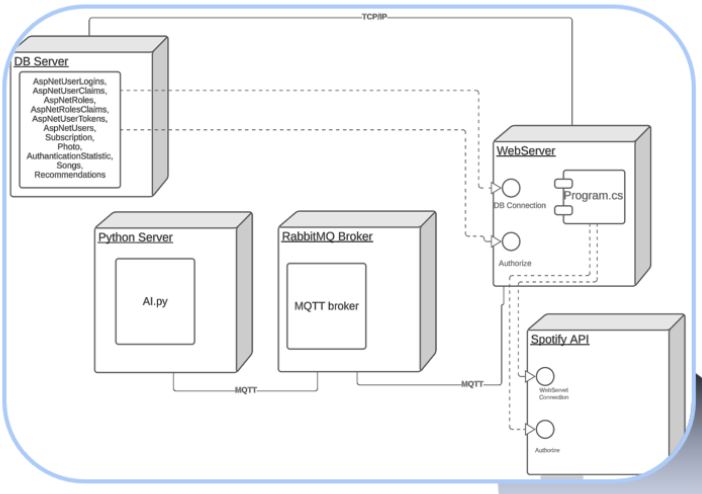


6



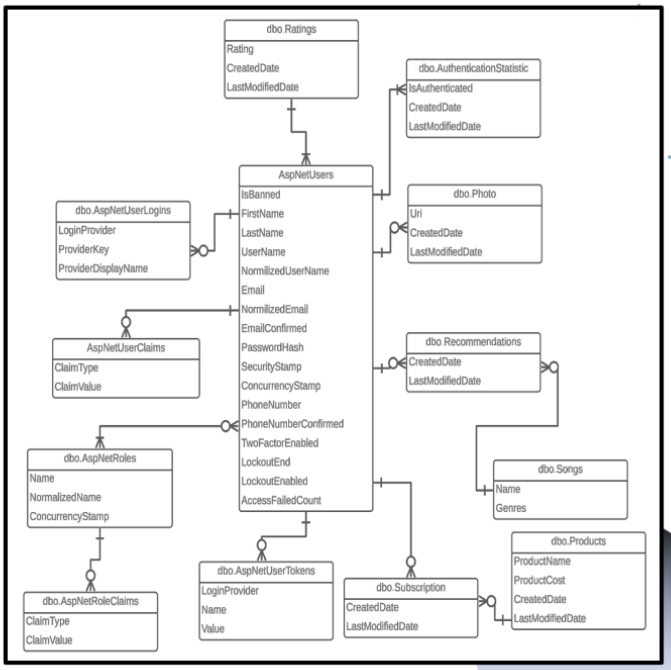
Діаграма пакетів проекту

7



UML діаграма розгортання

8



ER-модель даних проекту

9

Специфікація REST

Admin
- PATCH /api/Admin/BanUser/{userId}
- PATCH /api/Admin/AddRole
- PATCH /api/Admin/DeleteRole
- GET /api/Admin/GetUserList

Authentication
- POST /api/Authentication/Registration
- POST /api/Authentication/Login
- GET /api/Authentication/mail-confirmation
- GET /api/Authentication/signin-google

Azure
- GET /api/Azure/GetUserInfo

EmailSender
- POST /api/EmailSender

Payment
- POST /api/Payment/Create
- GET /api/Payment/GetAllProducts

Song
- GET /api/Song
- GET /get-recommendation
- GET /get-last-recommendations
- POST /rating

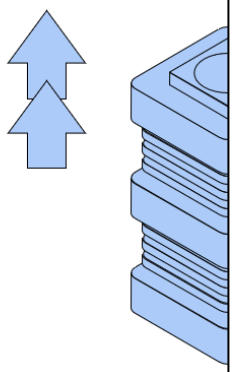
Statistic
- GET /api/Statistic/GetUserInfo
- GET /api/Statistic/GetLoginInfo
- GET /api/Statistic/GetRecommendationInfo

UserAccount
- GET /api/UserAccount/GetUserInfo
- PATCH /api/UserAccount/UpdatePassword
- DELETE /api/UserAccount/DeleteAccount
- PUT /api/UserAccount/UpdateUserInfo
- PUT /api/UserAccount/ChangeAvatar

Кожен ендпоінт налаштований з окремими поведінками, які дозволяють користувачам взаємодіяти з системою та отримувати персоналізовані рекомендації, керувати відображенням та профілем, швидко отримувати рекомендації з музичного контенту та покращувати аналіз даних для надання кращого сервісу

10

Опис роботи штучного інтелекту



11

Опис підходу розробки

Алгоритм рекомендації сфокусований на Content-Based методах. Ґрунтується на ідеї, що користувачам сподобаються елементи, які мають схожі атрибути чи характеристики

Name	Genre	Actors	…

«Хочу рекомендацію того, що мені сподобається»

Користувач

Алгоритм рекомендації

Item	Score
1	0.9
2	0.88
…	…

12

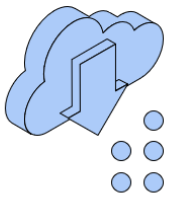
Аналіз тестової бази даних

Result_dataset

Feature	Count	Unique Value Count	Missing Value Count	Min	Max	Mean	Mean Deviation	1st quantile	Median	3rd quantile	Mode	Range
id	5001	5001	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
name	5001	4760	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
artists	5001	3374	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
id_artists	5001	3375	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
genres	4935	2952	66	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
release_year	5001	68	0	1949	2021	2010.737253	9.875136	2007	2017	2019	2020	72
duration_s	5001	4538	0	52.6	682.817	220.043656	41.004798	185.08	212.493	245.64	(180.0, 180.64)	630.217
popularity	5001	6	0	67	72	69.444911	1.099791	68	69	70	68	5
danceability	5001	725	0	0.0	0.975	0.636241	0.123906	0.538	0.651	0.748	0.748	0.975
energy	5001	852	0	2.03e-05	1.0	0.639142	0.157314	0.512	0.659	0.788	0.833	0.9999797
key	5001	12	0	0	11	5.179364	3.097673	2	5	8	1	11
loudness	5001	3856	0	22.736	61.509	53.076804	2.380564	51.744	53.718	55.265	54.74	38.772999999999996
mode	5001	2	0	0	1	0.634273	0.463941	0	1	1	1	1
speechiness	5001	1065	0	0.0	0.884	0.099265	0.073337	0.0364	0.055	0.116	0.0334	0.884
acousticness	5001	1946	0	4.76e-06	0.995	0.260218	0.214649	0.0422	0.172	0.409	0.114	0.99499524
instrumentalness	5001	1780	0	0.0	1.0	0.029813	0.053333	0	0	0.000159	0.0	1.0
liveness	5001	1074	0	0.0134	0.989	0.179789	0.10316	0.0931	0.122	0.221	0.109	0.9756
valence	5001	968	0	0.0	0.989	0.522621	0.198725	0.342	0.52	0.712	[0.354, 0.45, 0.491, 0.543, 0.591, 0.836]	0.989
tempo	5001	4557	0	0.0	220.065	122.426865	24.080319	98.243	121.493	140.974	119.964	220.065

13

Алгоритм рекомедації



01 Косинусна подібність

$$similarity = \frac{A*B}{||A||*||B||},$$

де A * B – скалярний добуток векторів A і B;
||A|| * ||B|| – їхні норми.

02 Формула скалярного добутку

$$A \cdot B = a1 \cdot b1 + a2 \cdot b2 + ... + an \cdot b_n,$$

де an – компонент вектору A;
b_n – компонент вектору B.

03 Норма вектору

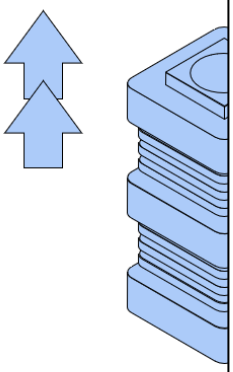
$$||A|| = \sqrt{a_1^2 + a_2^2 + ... + a_n^2},$$

де an – компонент вектору A.

Висновок:
Косинусна подібність вимірюється як косинус кута між двома векторами і може приймати значення в діапазоні від – 1 (абсолютно протилежні вектори) до 1 (ідентичні вектори).

14

Опис роботи з базою даних



15

Опис баз даних

MS SQL Server

Для модульного та навантажувального тестування системи

Azure SQL DB

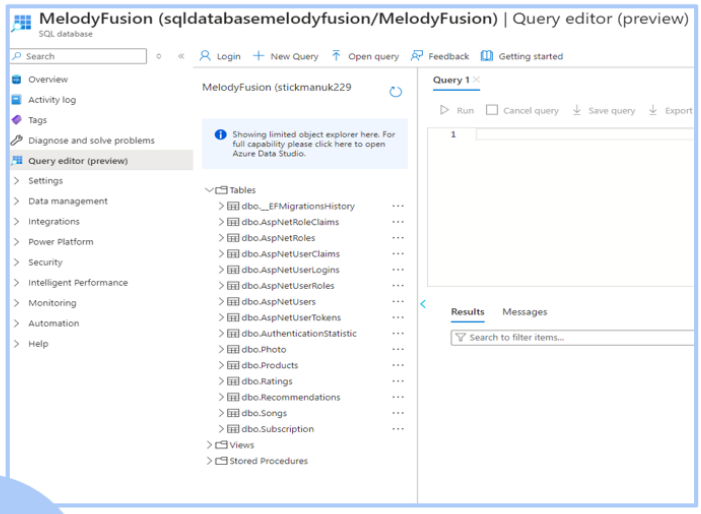
Для опрацювання великого потоку запитів від клієнтів

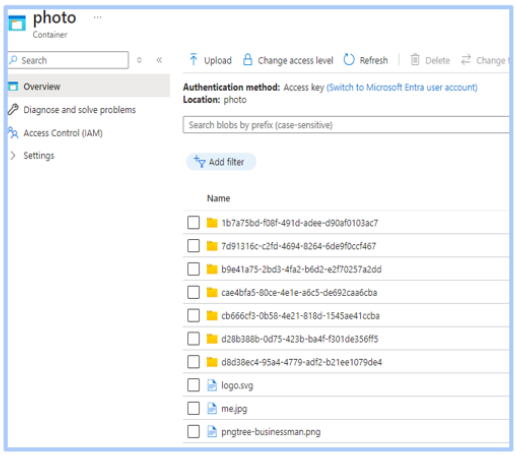
Blob-storage

Для збереження великих за розміром даних зокрема фотокартки

16

Структура БД





17

Структура реляційної БД

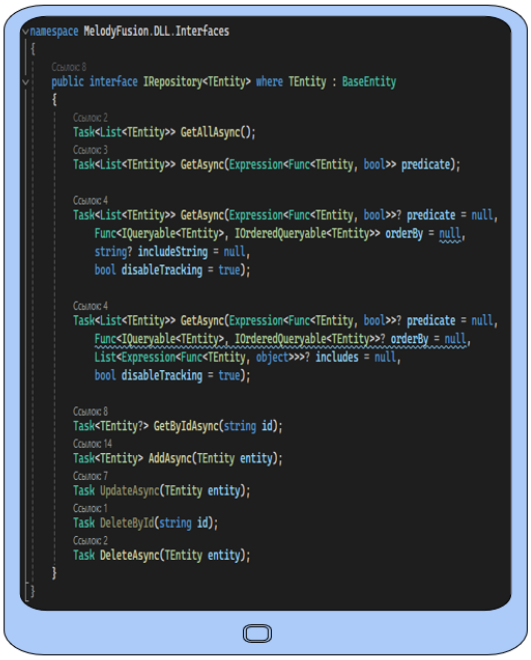
Структура нереляційної БД

```
using MelodyFusion.DLL.Entities.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace MelodyFusion.DLL.EntityConfigurations
{
    Ссылок: 1
    public class UserEntityConfiguration : IEntityTypeConfiguration<UserDto>
    {
        Ссылок: 0
        public void Configure(EntityTypeBuilder<UserDto> builder)
        {
            builder.HasMany(ur => ur.UserRoles)
                .WithOne(u => u.UserDto)
                .HasForeignKey(x => x.UserId)
                .OnDelete(DeleteBehavior.Cascade);

            builder.HasMany(x => x.Recommendations)
                .WithOne(x => x.User)
                .HasForeignKey(x => x.UserId)
                .OnDelete(DeleteBehavior.Cascade);
        }
    }
```

Приклад налаштуання зв’язків

За допомогою Entity Framework були налаштовані зв’язки між таблицями у Data Layer

18



```
namespace MelodyFusion.DLL.Interfaces
{
    Ссылок: 8
    public interface IRepository<TEntity> where TEntity : BaseEntity
    {
        Ссылок: 2
        Task<List<TEntity>> GetAllAsync();
        Ссылок: 3
        Task<List<TEntity>> GetAsync(Expression<Func<TEntity, bool>> predicate);

        Ссылок: 4
        Task<List<TEntity>> GetAsync(Expression<Func<TEntity, bool>>? predicate = null,
            Func<IQueryable<TEntity>, IOrderedQueryable<TEntity>> orderBy = null,
            string? includeString = null,
            bool disableTracking = true);

        Ссылок: 4
        Task<List<TEntity>> GetAsync(Expression<Func<TEntity, bool>>? predicate = null,
            Func<IQueryable<TEntity>, IOrderedQueryable<TEntity>>? orderBy = null,
            List<Expression<Func<TEntity, object>>>? includes = null,
            bool disableTracking = true);

        Ссылок: 8
        Task<TEntity?> GetByIdAsync(string id);
        Ссылок: 14
        Task<TEntity> AddAsync(TEntity entity);
        Ссылок: 7
        Task UpdateAsync(TEntity entity);
        Ссылок: 1
        Task DeleteById(string id);
        Ссылок: 2
        Task DeleteAsync(TEntity entity);
    }
}
```

Інтерфейс репозиторію

– репозиторій статистики;

– репозиторій фото;

– репозиторій продуктів;

– репозиторій рейтингів;

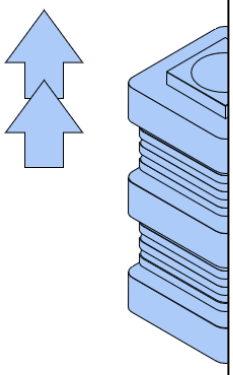
– репозиторій рекомендацій;

– репозиторій підписки;

– репозиторій пісень.

19

Опис роботи з брокером повідомлень RabbitMQ



20

```
public async Task SendMessage(object obj)
{
    var message = JsonSerializer.Serialize(obj);
    await SendMessage(message);
}

public async Task SendMessage(string message)
{
    var factory = new ConnectionFactory() { Uri = new Uri(_config.GetValue<string>("RabbitMQ:Host")) };
    using (var connection = factory.CreateConnection())
    using (var channel = connection.CreateModel())
    {
        channel.QueueDeclare(queue: _config.GetValue<string>("RabbitMQ:QueueRequest"),
            durable: false,
            exclusive: false,
            autoDelete: false,
            arguments: null);

        var body = Encoding.UTF8.GetBytes(message);

        channel.BasicPublish(exchange: "",
            routingKey: _config.GetValue<string>("RabbitMQ:QueueRequest"),
            basicProperties: null,
            body: body);
    }
```

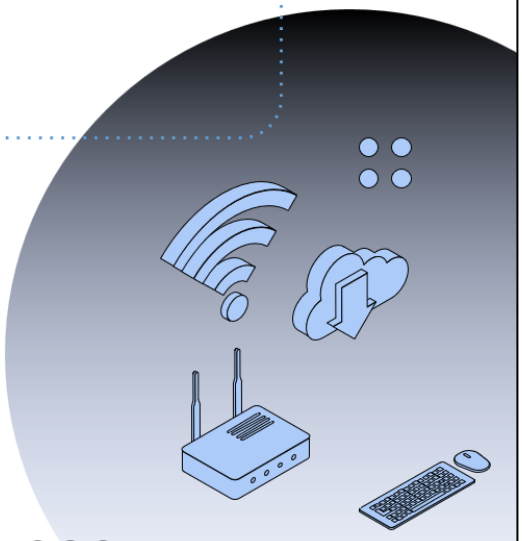
Опис роботи з брокером повідомлень RabbitMQ

Серед існуючих модулів передачі використовуються черги (Queues). Вони працюють за принципом FIFO (перший прийшов – перший пішов).

Опис роботи бізнес логіки серверної частини

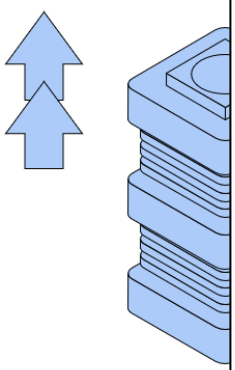
Бізнес логіка

Є центральною компонентою, яка відповідає за обробку та управління даними, реалізацію правил та процесів, що визначають функціонування застосунку.
У роботі широко використовується патерн «Впровадження залежностей».
Для налаштування роботи проекту також використовуються DI-контейнери.



23

Опис серверу штучного інтелекту



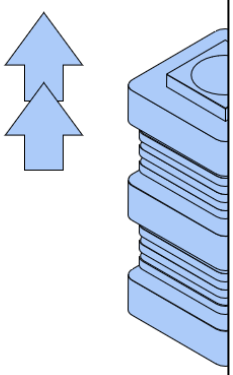
24

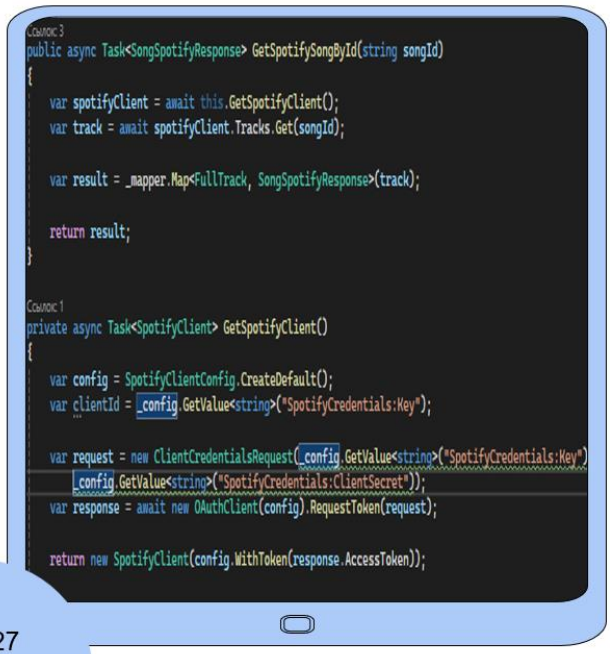
```
def song_recommender(first_song_id, second_song_id):
    try:
        num_cols = ['release_year', 'duration_s', 'popularity', 'danceability', 'energy', 'key', 'loudness',
                    'mode', 'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo']

        text_vec1 = song_vectorizer.transform(song_library[song_library['id'] == str(first_song_id)]['genres']).toarray(
        text_vec2 = song_vectorizer.transform(song_library[song_library['id'] == str(second_song_id)]['genres']).toarray

        num_vec1 = song_library[song_library['id'] == str(first_song_id)][num_cols].to_numpy()
        num_vec2 = song_library[song_library['id'] == str(second_song_id)][num_cols].to_numpy()

        sim_scores = []

        for index, row in song_library.iterrows():
            name = row['id']

            text_vec_other = song_vectorizer.transform(song_library[song_library['id'] == name]['genres']).toarray()

            num_vec_other = song_library[song_library['id'] == name][num_cols].to_numpy()

            text_sim1 = cosine_similarity(text_vec1, text_vec_other)[0][0]
            text_sim2 = cosine_similarity(text_vec2, text_vec_other)[0][0]

            num_sim1 = cosine_similarity(num_vec1, num_vec_other)[0][0]
            num_sim2 = cosine_similarity(num_vec2, num_vec_other)[0][0]

            sim = (text_sim1 + num_sim1 + text_sim2 + num_sim2) / 4
            sim_scores.append(sim)

        song_library['similarity'] = sim_scores

        song_library.sort_values(by=['similarity', 'popularity', 'release_year'], ascending=[False, False, False],
                                 inplace=True)

        recommended_songs = song_library[['id']][2:7]

        return recommended_songs
```


Робота функції рекомендації пісень

- Приймає два ідентифікатори;
- Створення текстових та чисельних векторів;
- Обчислюється середня косинусна подібність.

Опис взаємодії з Spotify API





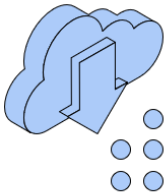
Опис роботи з Spotify API

Метод GetSpotifySongById приймає ідентифікатор пісні songId як параметр та асинхронно викликає метод GetSpotifyClient, після чого отримує дані пісні від сервісів Spotify.

27

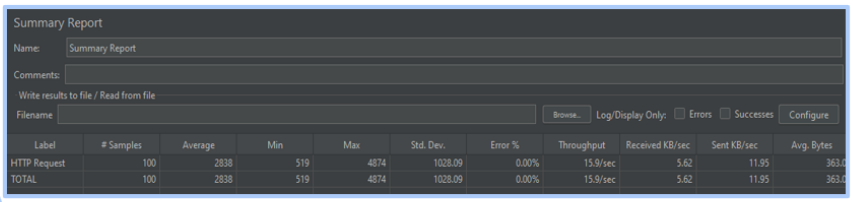
Тестування розробленого програмного забезпечення

Performance Testing



Для тестування критичних секцій обрано навантажувальне тестування за допомогою Apache JMeter. Проведено наступні навантажувальні тести:

- отримання власної інформації;
- пошук пісні по імені в базі даних без та з кластерним індексом;
- отримання статистичних даних;
- отримання рекомендацій пісень.



Статистика навантажувального тесту отримання статистики

29

Висновки

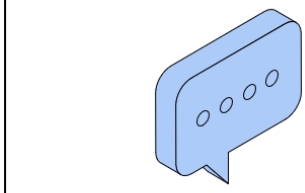
У ході виконання кваліфікаційної роботи була проаналізована предметна область проекту та розроблена програмна система рекомендації музики на основі вподобань групи користувачів, а саме її серверна частина та штучний інтелект.

- Використані та закріплені навички у використанні технологій: ASP.NET Web API, мові програмування C#, мові запитів до баз даних T-SQL, інтеграція зі сторонніми сервісами Spotify API та Azure;
- Проаналізована література зокрема робота Yamac Eren Ay, база даних якого була використана в роботі як тестова. Проведені ознайомчі роботи з документаціями використаних технологій;
- Створені підсистеми управління базами даних SQL та blob-сховищ, система для обробки запитів користувачів, система надання рекомендацій, система інтеграцій та відправки та отримання повідомлень для взаємозв'язку серверів ШІ та back-end частини;
- Платформа протестована за допомогою навантажувального тестування;
- Вирішена проблема з оптимізованим зберіганням та створенням запитів інформації на різних типах баз даних.

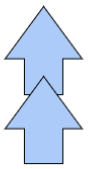
Збірки тез доповідей

AI-POWERED MUSIC RECOMMENDATION SERVICE USING .NET IMPLEMENTATION. X INTERNATIONAL CONFERENCE Information Technology and Implementation (Satellite), Kyiv, 21 November 2023. P. 48–49.

30



Дякую за увагу



ДОДАТОК В

Апробація результатів роботи

TARAS SHEVCHENKO NATIONAL UNIVERSITY OF KYIV
(FACULTY OF INFORMATION TECHNOLOGY,
FACULTY OF COMPUTER SCIENCE AND CYBERNETICS)
NATIONAL TECHNICAL UNIVERSITY OF UKRAINE
"IGOR SIKORSKY KYIV POLYTECHNIC INSTITUTE"
VIKTOR GLUSHKOV INSTITUTE OF CYBERNETICS OF THE NAS OF UKRAINE
INSTITUTE FOR INFORMATION RECORDING OF THE NAS OF UKRAINE
INSTITUTE OF SOFTWARE SYSTEMS OF THE NAS OF UKRAINE
THE COUNCIL OF YOUNG SCIENTISTS OF THE FACULTY OF COMPUTER SCIENCE
AND
CYBERNETICS AND THE FACULTY OF INFORMATION TECHNOLOGY OF
TARAS SHEVCHENKO NATIONAL UNIVERSITY OF KYIV

X INTERNATIONAL CONFERENCE

Information Technology and Implementation (Satellite)

21 November, 2023

Conference Proceedings

Kyiv

2023

Рисунок В.1 – Титульна сторінка конференції

Artem Pushkar, Student of Software Engineering Department
Iryna Afanasieva, PhD, Associate Professor of Software Engineering Department
Kharkiv National University of Radioelectronics

AI-POWERED MUSIC RECOMMENDATION SERVICE USING .NET IMPLEMENTATION

Abstract

Research in the field of computer science provides the ability to address complex tasks related to individual preferences and collective music selection for groups of people. This research work explores methods and technologies for creating systems, taking into account aspects of data processing, preference analysis and recommendation systems. The technical implementation of the system, based on ASP.NET Core Web Application, is also considered.

Keywords: music, artificial intelligence, automatic selection, recommendation system

Music is a unique universal language that unites society in various ways. It promotes communication among people of different cultures and backgrounds and plays a vital role in emotional bonding, strengthening connections between nations through the exchange of musical experiences. Music also, importantly, allows us to feel almost all the emotions we have experienced in life. However, the challenge lies in the fact that musical preferences in a group of people do not always align [1].

Therefore, it is necessary to develop a system that allows for music selection taking into account the diverse preferences of multiple individuals. There exists a system [2] with a similar principle, albeit designed for film recommendations. Spotify has also introduced a "Synthesize" feature, which assesses musical taste overlaps but does not add new tracks to existing playlists.

During the development of the software service, the question arises as to how it will precisely select music considering the preferences of multiple people. For this purpose, the utilization of artificial intelligence (AI) [3, 4] is appropriate, as it can provide recommendations with a certain degree of accuracy based on datasets represented in the form of XML or JSON files.

In this case, since we are addressing the issue of recommendations based on specific criteria, it is quite logical to employ an approach known as Content-Based Filtering [5], which is a main part of AI. Each user selects songs from the existing catalog based on their individual preferences. The "Spotify Dataset" consists of several tables, including "data," "data_by_artist," "data_by_genres," "data_by_year," and "data_w_genres" [6]. Each of them has its own characteristics, with the most common ones being artists, loudness, energy, genre, danceability, and more. As Content-Based Filtering relies on the analysis of object characteristics (in this case, music tracks), vectorization is a crucial step in creating content-based recommendation services. It helps measure the similarity between objects and provides more accurate and personalized recommendations for users.

The software service should provide recommendations for objects that align with the user's preferences. To achieve this, it is necessary to first select a similarity metric,

Рисунок В.2 – Перша сторінка тез

such as the scalar product. Subsequently, the system should be configured to evaluate each potential object based on this similarity metric.

The question arises: how will the user interact with Artificial Intelligence (AI) For the development of the project's backend, it is logical to utilize .NET Core, specifically ASP.NET Core Web Application, to enable interaction with the software service from a client application in a web browser. After authentication and system authorization, users will easily engage with AI by sending requests to obtain recommendations and receiving personalized suggestions for music selection, making interaction with artificial intelligence convenient and accessible. This approach offers several advantages:

- The server can be deployed on any platform;
- Asp.Net Core facilitates additional flexibility concerning languages;
- ASP.NET Core Web framework is intended from scratch and maintains the performance in mind;

Therefore, the software service will be in demand due to the lack of full-fledged alternatives, and the task of recommendations will be performed by Content-Based Filtering method, which has its advantages in scalability and performance. It will be integrated with the backend part built on ASP.NET Core Web Application, providing a flexible and reliable structure for creating various categories of high-quality web applications.

References:

1. Forgeard V. Unveil 35 Ways Music Impacts Our Life: Get Ready to Be Amazed!. Brilliantio. URL: https://brilliantio.com/how-does-music-impact-our-life/ (date of access: 17.10.2023).
2. Recommendation Systems: Content-Based Filtering. medium.com. URL: https://medium.com/mlearning-ai/recommendation-systems-content-based-filtering-e19e3b0a309e (date of access: 21.10.2023).
3. P. Melville, V. Sindhwani; Recommender systems; Encyclopedia of Machine Learning, 2010 (date of access: 3.9.2023).
4. X. Su, T.M. Khoshgoftaar; A Survey of Collaborative Filtering Techniques; Advances in Artificial Intelligence, 2009 (date of access: 3.9.2023).
5. Recommendation Systems: Content-Based Filtering. medium.com. URL: https://medium.com/mlearning-ai/recommendation-systems-content-based-filtering-e19e3b0a309e (date of access: 21.10.2023).
6. Music Recommendation System using Spotify Dataset. Kaggle: Your Machine Learning and Data Science Community. URL: https://www.kaggle.com/code/vatsalmavani/music-recommendation-system-using-spotify-dataset/input (date of access: 06.11.2023).

Рисунок В.3 – Остання сторінка тез