ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

КВАЛІФІКАЦІЙНА РОБОТА

Тема: «Кросплатформений плеєр, з використанням алгоритмів рекомендацій»

Ступінь вищої освіти – бакалавр
Спеціальність – 122 «Комп’ютерні науки»
Освітня програма «Комп’ютерні науки»

ПОЯСНЮВАЛЬНА ЗАПИСКА

Виконав: здобувач 4 курсу
групи КН-21
Юрій ПОНОМАРЕНКО

Керівники: доцент кафедри комп’ютерних наук, кандидат технічних наук
Олександр ПОЛІЩУК
асистент кафедри комп’ютерних наук
Микита ФЕШЕНКО

Засвідчую, що кваліфікаційна робота оформлена відповідно до ДСТУ 3008:2015 та не містить запозичень з праць інших авторів без відповідних посилань.

Здобувач: _____
(підпис)

м. Київ – 2025 рік

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД

«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

ЗАТВЕРДЖУЮ:
завідувач кафедри
комп'ютерних наук
_____Сергій МІЧКІВСЬКИЙ
«____»____20___р

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Пономаренко Юрій Андрійович

Тема роботи	Кросплатформений плеєр, з використанням алгоритмів рекомендацій
Номер та дата наказу про затвердження теми	№121-7 від 24 грудня 2024 року
Коротка постановка завдання	Розробити кросплатформену програму музичного плеєра з алгоритмами рекомендацій для локальної музичної бібліотеки.
Посилання на джерела інформації (не більше п'яти найменувань, які рекомендує науковий керівник)	Шаповалов С.М., Мічківський С.М. Розробка рекомендаційної системи для підбору співрозмовника в соціальній мережі // XII Міжнародна науково-технічна конференція «Проблеми Інформатизації» (м. Київ, 13 грудня 2018 року). – К.: Державний університет телекомунікацій, 2018. URL: https://dut.edu.ua/uploads/l_1701_65845854.pdf (дата звернення: 24.01.2025) Koren Y., Bell R., Volinsky C. Matrix Factorization Techniques for Recommender Systems // Науковий журнал «Computer». – 2009. - Том 42, № 8. – С 30-37 DOI: 10.1109/MC.2009.263
Вимоги до кваліфікаційної роботи	Кваліфікаційна робота має містити теоретичне, системотехнічне або експериментальне дослідження за темою роботи, яку слід розглядати як складне спеціалізоване завдання або практичну проблему в галузі комп'ютерних наук, яка характеризується комплексністю та невизначеністю умов і потребує застосування теорій і методів інформаційних технологій.

Дата видачі завдання «27» грудня 2024 р.

Керівник Олександр ПОЛІЩУК
Керівник Микита ФЕЩЕНКО

Здобувач освітнього ступеня бакалавра Юрій ПОНОМАРЕНКО

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання	Примітка
	Підготовчий етап		
1	Вибір напряму дослідження	02.12.2024 р.	*виконано*
2	Формування теми та призначення керівника	16.12.2024 р.	*виконано*
3	Затвердження теми кваліфікаційної роботи	23.12.2024 р.	*виконано*
4	Затвердження завдання на кваліфікаційну роботу	27.12.2024 р.	*виконано*
	Основний етап		
5	Розробка концепції кваліфікаційної роботи	13.01.2025 р.	*виконано*
6	Підбір та вивчення джерел інформації з напряму дослідження. Огляд існуючих аналогів	20.01.2025 р.	*виконано*
7	Затвердження розширеної постановки завдання. Підготовка та подання керівникові розділу 1 кваліфікаційної роботи	10.03.2025 р.	*виконано*
8	Проєктування. Підготовка та подання керівникові розділу 2 кваліфікаційної роботи	24.03.2025 р.	*виконано*
9	Підготовка доповіді для експертизи стану виконання кваліфікаційної роботи (проміжний контроль)	31.03-04.04.2025 р.	*виконано*
10	Реалізація. Підготовка та подання керівникові розділу 3 кваліфікаційної роботи	07.04.2025 р.	*виконано*
11	Підготовка та подання керівнику першого варіанту всієї кваліфікаційної роботи	14.04.2025 р.	*виконано*
12	Доопрацювання кваліфікаційної роботи з урахуванням зауважень керівника та представлення керівникові доопрацьованого варіанту кваліфікаційної роботи	21.04.2025 р.	*виконано*
	Завершальний етап		
13	Представлення рукопису для перевірки на плагіат	28.04-04.05.2025 р.	*виконано*
14	Підготовка презентації та доповіді на передзахист	05.05-11.05.2025 р.	*виконано*
15	Передзахист кваліфікаційної роботи	12.05-16.05.2025 р.	*виконано*
16	Доопрацювання роботи за результатами передзахисту	19.05-06.06.2025 р.	*виконано*
17	Експертиза роботи керівником та зовнішнім експертом	09.06-15.06.2025 р.	*виконано*
18	Доопрацювання доповіді та презентації для захисту	09.06-15.06.2025 р.	*виконано*
19	Захист кваліфікаційної роботи	16.06-22.06.2025 р.	*виконано*

Керівник Олександр ПОЛІЩУК

Керівник Микита ФЕЩЕНКО

Здобувач освітнього ступеня бакалавра Юрій ПОНОМАРЕНКО

Пономаренко Ю.А. Кросплатформений плеєр, з використанням алгоритмів рекомендацій.

Пояснювальна записка кваліфікаційної роботи за спеціальністю 122 – Комп'ютерні науки (освітня програма – Комп'ютерні науки) СО Бакалавр. – ВНЗ «Університет економіки та права «КРОК», Навчально-науковий інститут інформаційних та комунікаційних технологій, кафедра комп'ютерних наук, Київ, 2025.

Досліджено розробку кросплатформеної програми плеєра і алгоритмів рекомендацій на прикладі популярних стрімінгових платформ. Включає аналіз предметної області, конкурентних рішень і MVP продукту (плеєра) з реалізацією ключового функціоналу.

Ключові слова: кросплатформена розробка, плеєр, музика, рекомендаційні алгоритми.

Табл. 2. Рис. 24. Бібліограф.: 30 найм.

Ponomarenko Y.A. Cross-platform player with recommendation algorithms implementation

Explanatory note of the qualification work in the specialty 122 – Computer Science (educational program – Computer Science) Bachelor's degree. – Higher Educational Institution «University of Economics and Law »KROK», Educational and Scientific Institute of Information and Communication Technologies, Department of Computer Science, Kyiv, 2025.

The development of a cross-platform player program and recommendation algorithms has been researched using popular streaming platforms as an example. It includes an analysis of the subject area, competitive solutions, and the MVP product (player) with the implementation of key functionality.

Keywords: cross-platform development, player, music, recommendation algorithms.

Tabl. 2. Fig. 24. Bibliography: 30 Items.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

Бургер-меню – це елемент інтерфейсу користувача, який використовується для приховування навігаційного меню на вебсайті або в мобільному додатку. Його назва походить від схожості з бургером: три горизонтальні лінії, які символізують шари бургеру.

Кросплатформеність – це здатність програмного забезпечення працювати на різних апаратних платформах або операційних системах.

Офлайн – це термін, який означає «не в мережі» або «поза межами інтернету». Спочатку це слово було англіцизмом «offline», утвореним від англійських слів «off» і «line», що означає «поза лінією» або «відключений від мережі».

Плагін – незалежний програмний модуль, який підключається до основної програми та розширює її функціонал і можливості.

Плейлист – це список відтворення музичних файлів або відеороликів, що впорядкований за певними правилами.

Стрімінговий сервіс – це сервіс, який передає контент від провайдера до користувача, надаючи безперервний доступ до музики, фільмів, книг та іншого вмісту безкоштовно або за певну плату. Весь контент вже завантажений на сторонньому сервері, а кінцевий користувач не потрібно нічого завантажувати для перегляду або прослуховування. Контент транслюється в режимі реального часу, швидкість завантаження безпосередньо залежить від швидкості інтернету користувача.

Трек – окремий музичний запис у складі музичного альбому або плейлиста.

Фреймворк – це програмна платформа, яка визначає структуру програмного забезпечення, може складатись з наборів інструментів, бібліотек і правил для написання коду.

ВСТУП

Актуальність теми. З моменту виходу iTunes в 2003 році, музика стала легкодоступною для всіх, а для її придбання та прослуховування людям не потрібно було йти в магазин і купувати CD-диск чи платівку. Зараз музика є невід'ємною частиною життя багатьох людей.

З того часу багато чого помінялось і нині основним способом прослуховування музики є стрімінгові сервіси, які пропонують централізований спосіб прослуховування треків більшості виконавців світу.

Такі сервіси поміж своїх однозначних переваг, таких як можливість заплатити малу підписку в обмін на доступ до дуже великої бібліотеки, в порівнянні з купівлею альбомів чи пісень, кожна з яких може вартувати як місячна підписка, також мають і свої недоліки. Такі як повна залежність від цієї компанії, щоб та мала конкретну пісню в доступі, чи навпаки не прибрала її, або ж неможливість в деяких сервісах слухати пісні високої якості.

Зважаючи на це попит на офлайн плеєри лишається, але на ринку відсутні рішення які консистентні на різних платформах і вбирають в себе найкраще з двох світів.

Метою проєкту є створення кросплатформеного офлайн музичного плеєра, який здатен програвати музику високої якості і має вбудовані алгоритми рекомендації.

Завдання дослідження:

- проаналізувати ринок;
- сформувати перелік вимог та функціоналу;
- дослідити ключові рекомендаційні алгоритми;
- сформулювати рекомендаційні алгоритми;
- розробити плеєр.

Об'єктом дослідження є кросплатформені фреймворки розробки і рекомендаційні алгоритми щодо підбору музики.

Предметом дослідження є сфера стрімінгових сервісів, що надають послуги з прослуховування музики та офлайн плеєрів на різних платформах.

Практична цінність проєкту полягає у розробці музичного плеєру, який є консистентним між платформами, інтегрує в себе корисні рекомендаційні алгоритми.

Структура та обсяг пояснювальної записки. Пояснювальна записка до проєкту складається зі вступу, трьох розділів, висновків, списку посилань (30 найменувань) та 0 додатків. Пояснювальна записка містить 24 рисунка, 2 таблиці. Загальний обсяг пояснювальної записки складає 58 сторінок, основний зміст викладено на 55 сторінках.

РОЗДІЛ 1
ПОСТАНОВКА ЗАВДАННЯ НА РОЗРОБКУ КРОСПЛАТФОРМЕНОГО ПЛЕЄРУ

1.1 Опис предметної області

Переважна більшість людей слухає музику, чи то на комп'ютері, чи то на телефоні, чи то на програвачі фізичних носіїв, таких як платівки або диски. За часів цифровізації фізичні носії поступилися місцем MP3 плеєрам, а згодом і телефонам [1].

На даний момент, люди користуються стрімінговими сервісами для прослуховування музики, а не купує її в форматі файлів чи фізичних носіїв [2]. Данна тенденція породжує зсув уваги гігантів IT індустрії від виробництва якісних продуктів для прослуховування особистої музичної колекції, на розробку онлайн платформ, які повністю залежать від наявності підключення до серверів. Звісно дане підключення не має бути перманентним, але користувач зобов’язаний на регулярній основі підключатись до серверів постачальника послуги для підтвердження наявності підписки, до прикладу Spotify дозволяє перебувати застосунку в офлайн режимі 30 днів, після чого треба підключатись до їх серверів [3].

Необхідність бути постійно онлайн для прослуховування треків не є єдиною проблемою нинішньої індустрії. Регулярно користувачі стикаються з регіональними обмеженнями, які не дозволяють слухати ту чи іншу пісню, у випадках, коли ліцензію не було узгоджено у відповідному регіоні з тієї чи іншої причини [4]. Це призводить до того, що користувач має чи порушувати користувацьку угоду і користуватись сервісом з іншого регіону, при цьому не знаходячись в цьому регіоні, чи користуватись кількома різними сервісами, чи банально лишитись без можливості прослуховувати даний трек або цілого виконавця.

Стрімінгові сервіси також мають переваги. Більше за все привертає увагу наявність алгоритмів рекомендацій, що є побічним ефектом війни за

увагу [5]. Коли користувач слухає музику, алгоритм вираховує яку пісню йому можна запропонувати і створює чергу з таких треків. Це регулярно призводить до приємних знахідок, і дозволяє малознайомим артистам знайти своїх перших фанатів.

Вакуум на ринку в ніші офлайн плеєрів породжує низку дрібних проєктів, які виконують найпростішу функцію – грати музику. Однак цей досвід або не дуже якісний, або обмежений однією платформою, або не вбирає в себе найкраще з індустрії, частіше за все – одразу все.

1.2 Огляд аналогів

Після аналізу ринку, було виявлено кілька аналогів, які в містять у собі ключові критерії якісного плеєра. Також було розглянуто стрімінгові сервіси, так як вони мають свої плеєри.

До цього списку входять:

- AIMP [6];
- Spotify [7];
- Youtube Music [8];
- Foobar2000 [9].

Далі наведено детальний опис аналогів, їх ключові переваги та недоліки.

AIMP. Вперше випущений ще в 2006 році. Досить популярний офлайн плеєр, з більше ніж 10 мільйонами завантажень, і який досі продовжують використовувати.

Цей музичний плеєр надає можливість перегляду усієї файлової системи при цьому він відбирає папки з музичними файлами і підчас роботи з однією основною папкою можливо в середині плеєру заходити до дочірніх папок не покидаючи основну. Наприклад, у рисунку 1.1: всередині основної папки «Jpop» є різні пісні та дві дочірні папки, що є альбомами групи «KIRINJI» з назвами «Neo» та «Cherish», у яких всередині також є пісні, а при відповідному натисканні на них розгорнеться весь перелік пісень.

AIMP має кілька вбудованих рекомендаційних плейлистів, ключовими з яких є наступні (рис. 1.1):

– «Забуті треки»;
– «Топ 100».
– «Ще не програні».

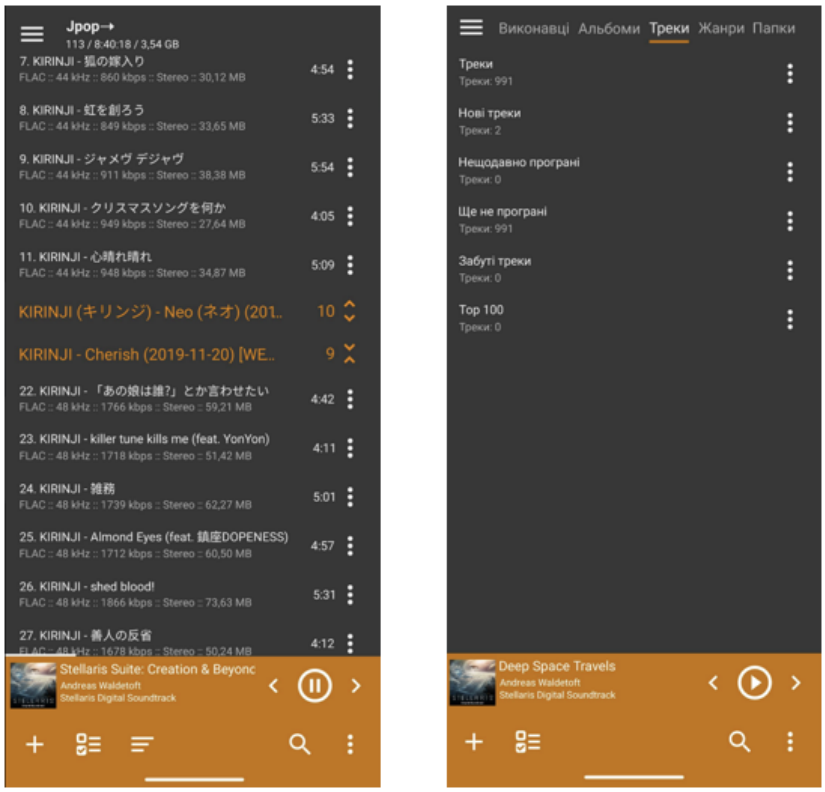

Рисунок 1.1 – Ліворуч зображено роботу AIMP з папками, а праворуч – рекомендаційні плейлисти

Джерело: AIMP [6]

Дизайн не є ключовою частиною цієї роботи, але не можливо оминути використання бокової шторки, яку неможливо відкрити провівши пальцем з боку екрану, у випадку, коли користувач застосовує жести для навігації системою замість кнопок. Відповідно єдиним способом відкрити це меню, в якому прихована майже вся ключова інформація є кнопка «бургер-меню», яке можна побачити зверху зліва на рисунку 1.1. Такий спосіб навігації дуже незручний коли у користувача великий телефон чи малі долоні.

Spotify є один з найбільших стрімінгових сервісів в світі. Має дуже обширну систему рекомендації. Свого часу цією компанією було створено Spotify Wrapped, функцію яка раз на рік, нагадує користувачу про пісні які він більше за все слухав минулого року. Spotify Wrapped є трендовим функціоналом, який кожного року заполоняє стрічки соціальних мереж (рис. 1.2).

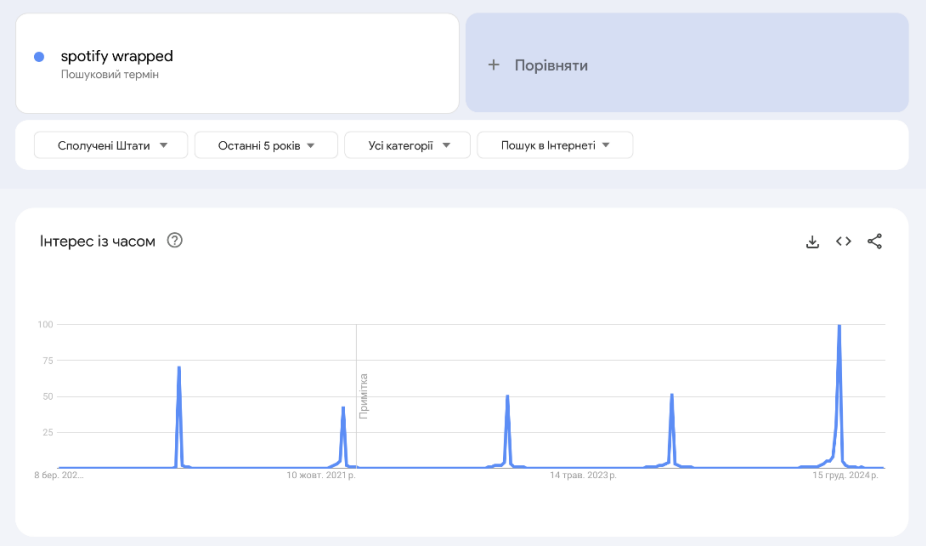

Рисунок 1.2 – Всплески пошукових запитів Spotify Wrapped

Джерело: Google trends[10]

Spotify має кілька недоліків:

- регіональні обмеження;
- нав'язливий функціонал «штучного інтелекту» (далі ШІ);
- відсутність музики в високій якості.

Розберемо їх детальніше.

Регіональні обмеження. Крім того, що деякі пісні відсутні через відсутність ліцензії чи порушень правил платформи, є більш обширне обмеження по регіону в якому можна користуватись сервісом. До прикладу, аккаунт користувача зареєстрований в Україні і за правилами сервісу користувач може оплачувати використання сервісу виключно українською платіжною карткою чи подарунковими картами українського регіону. Також у

платформи є обмеження на використання сервісу за кордоном терміном в 14 діб. Тобто, якщо у користувача запланована відпустка на термін довший за 14 діб, його акаунт буде тимчасово обмежено до того моменту коли він підключиться до серверів Spotify з українського IP.

Нав'язливий функціонал «ШІ». Після набрання популярності великих текстових ШІ-моделей, ринок відреагував так, що багато компаній почали додавати префікс «AI» чи впроваджувати нейронні мережі в свої продукти. Spotify не є виключенням. У 2023 році, на платформу було додано «розумне перемішування», яке розширює функціонал класичного перемішування плейлиста або черги [11]. Дана кнопка додає третю позицію для кнопки «перемішати» яка вже давно всім відома (рис. 1.3), додаючи запит на сервера сервісу, які за допомогою нейронних мереж створюють підібрану під користувача чергу треків, яка є сумішшю пісень з його плейлистів і тих треків, які користувач ще не чув, чи не додав до конкретного плейлиста.

Рисунок 1.3 – Іконка «розумного перемішування»
Джерело: Spotify[11]

Це є доречним місцем для застосуванням нейронної мережі та рекомендаційного алгоритму. Однак, у цьому випадку є проблема, що полягає в тому, що користувач не може вимкнути опцію розумного перемішування і заблокований від звичайного перемішування [12]. Іншими словами, якщо споживач попередньо користувався «розумним перемішування», то при кожній спробі перемішати плейлист буде викликатись ця функція доки сервер не сформує йому нову чергу, а спроба виклику сервісу відбувається навіть тоді, коли у користувача відсутнє інтернет з'єднання.

Відсутність музики в високій якості: платформа не дає можливості меломанам прослуховувати музику високої якості, максимум який надає

платформа є 320 кбіт/сек, що є максимальною роздільною здатністю формату MP3. Звісно для більшості людей це є задовільною якістю, але це обмежує людей яким висока якість є важливою.

YouTube Music є стрімінговим сервісом, що збирає усю музику з YouTube в одному зручному місці. У цього сервісу відсутня версія для ПК, а сайт має кілька відмінностей від YouTube, наприклад, нижча якість музичних кліпів і коротша черга (50 наступних пісень замість 200).

Мобільна версія сервісу не має таких обмежень з чергою як на сайті, і коли користувач добирається до кінця списку – довантажується ще 50 пісень, а якщо в плейлисті не лишилось пісень, алгоритм рекомендує ще кілька пісень за загальними вподобаннями користувача.

Також на мобільній версії присутня вкладка «Samples», в якій можна подивитися короткі фрагменти з кліпів пісень і зберегти собі в бібліотеку ті, що сподобались. Загальний стиль кліпів є відео стандартного формату співвідношення екрану 16:9 з обрізанням центрального фрагменту. Роздільна здатність відео теж є обмеженою, вона точно не вказується, але на перший погляд виглядає як 480p.

Foobar2000. Не дуже відомий плеєр, з довгою історією. Перша версія була випущена ще в 2001 році і плеєр продовжує оновлюватись до сьогодні. У цього плеєра лишились старі архітектурні рішення (такі як відсутність черги) і він вже є застарілим, хоча має і низку переваг, таких як:

- велика кількість підтриманих форматів музики;
- система плагінів;
- можливість під’єднати інтернет радіо;
- відкритий Software development kit (далі SDK).

Велика кількість підтриманих форматів музики та система плагінів. У людей, що полюбляють колекціонувати музику періодично з’являються малопоширені формати музичних файлів, які не кожен плеєр здатен програти. Окрім основного переліку форматів, які підтримує більшість плеєрів (mp3, flac, wav, ogg, тощо), даний плеєр має можливість встановлювати плагіни

розроблені користувачами як для розширення переліку форматів рідкісними форматами (tta, alac, ac3, тощо), так і для додавання відображення тексту пісень або для інтеграції роботи з iPod.

Можливість під'єднати інтернет радіо. Якщо почути слово радіо, то перше, що приходить на думку це радіо в машині, але це функція в плеєрі, що дозволяє підключатись до інтернет радіо, яке має свої переваги, такі як краща якість звуку, відображення інформації треку, якщо їх постачає станція, та відсутність обмежень по регіону.

Відкритий SDK. SDK – це набір засобів розробки, утиліт і документації, яка дозволяє створювати продукт на його основі. Це дозволяє розробникам, які хочуть додати підтримку якісної обробки звукових файлів в свій програмний продукт полегшує цей процес.

Недоліком Foobar2000 є його застарілість, загальна архітектура плеєру лишилась на рівні часів його випуску, а мобільним застосунком користуватись дуже важко, через купу вкладених меню, частина з яких є дублюванням функціоналу, який вже розташовано на сторінці програвання пісні (рис. 1.4).

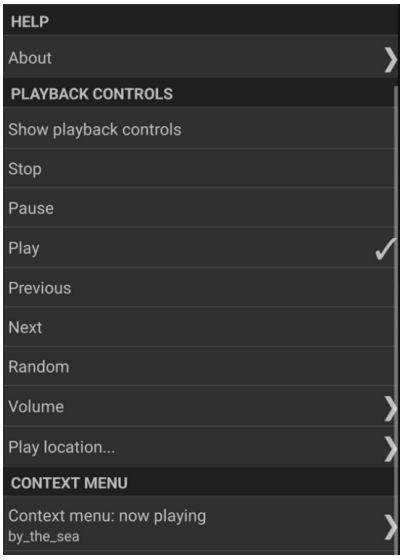

Рисунок 1.4 – Кнопки для керування програванням приховані в одному з контекстних меню

Джерело: Foobar2000[13]

1.3 Постановка завдання на кваліфікаційну роботу

Завдання кваліфікаційної роботи – розробити музичний плеєр, який буде максимально уніфікованим між різними платформами та буде використовувати алгоритми персональної рекомендації на основі діяльності користувача, що має вирішити проблему відсутності локального прослуховування музики з задовільним рівнем персоналізації.

Далі буде наведено перелік вхідних і вихідних даних (табл. 1.1), а також функціональних і нефункціональних вимоги (табл. 1.2).

Таблиця 1.1 – Вхідні та вихідні дані

Вхідні дані	**Вихідні дані**
Музична бібліотека	Персоналізовані плейлисти і черги
Поведінка користувача при прослуховуванні музики	Інформація про треки
	Прослуховування пісень

Продовження Таблиці 1.2 – Функціональні та нефункціональні вимоги

Функціональні вимоги	**Нефункціональні вимоги**
Відтворення музики	Навігація має бути зручною як на мобільній так і на версії для ПК.
Плеєр має бути кросплатформеним для уніфікації досвіду на різних платформах.	Швидка обробка великих музичних бібліотек.
Має відбуватись аналіз поведінки користувача в прослуховуванні музики.	Обробка різних файлових форматів.
Під час прослуховування музики користувач має мати можливість розширювати завершені плейлисти рекомендаціями, або генерувати абсолютно нові плейлисти.	Надання алгоритмом якісних рекомендацій.
Відображати на регулярній основі скільки пісень прослухав користувач і які з них було частіше за все відтворено.	

Висновок до розділу 1

Було виявлено проблемну ситуацію, в якій користувачі не мають продуктів, які б надали їм персоналізований досвід прослуховування особистої музичної бібліотеки. Сформульовано ключову перевагу продукту над конкурентами

Проведено аналіз конкурентних продуктів, їх сильні і слабкі сторони, що буде враховано при розробці продукту для використання кращих рішень та не припуститися помилок.

РОЗДІЛ 2
ПРОЄКТУВАННЯ КРОСПЛАТФОРМЕНОГО ПЛЕЄРУ

2.1 Моделювання поведінки продукту

В даному розділі було проведено моделювання поведінки продукту для кращого розуміння роботи його систем. Першочергово було розроблено діаграму використання продукту для візуалізації основних сценаріїв використання плеєру і загальної поведінки системи (рис. 2.1).

Рисунок 2.1 – Діаграма варіантів використання

Джерело: розроблено автором

Нижче наведено детальний опис кожного сценарію. Загальна назва системи: «плеєр з алгоритмами рекомендацій». Ліворуч фігуркою людини зазначений актор, що є користувачем.

Першим варіантом використання є надання доступу до локальної музичної бібліотеки. Користувач отримує доступ до своєї локально збереженої музичної колекції на пристрої. Послідовність дій: користувач запускає програвач, і той відображає список файлів музики, що зберігаються на пристрої. Зв'язок з іншим варіантом використання: після отримання доступу до бібліотеки, метадані пісень можуть бути оброблені рекомендаційним алгоритмом, який пояснений у другому варіанті використання.

Другий варіант використання є первинна обробка метаданих пісень рекомендаційним алгоритмом. Алгоритм аналізує інформацію про музичні файли (жанр, виконавець, рік випуску, тощо) для формування початкових рекомендацій. Це відбувається автоматично після надання доступу до бібліотеки. Послідовність дій: алгоритм зчитує метадані пісень [14] і використовує їх для створення первинних рекомендаційних зав’язків. На рисунку 2.1 зв’язок пунктирною лінією вказує на те, що це автоматичний процес, який відбувається у фоновому режимі та не потребує прямої взаємодії з користувачем.

Створення списків відтворення (плейлистів). Користувач створює власні плейлисти з локальної бібліотеки. Послідовність дій: користувач вибирає пісні, додає їх до нового або існуючого плейлиста, може перейменувати плейлист чи видалити його.

Прослуховування бібліотеки. Користувач відтворює музику з локальної бібліотеки. Послідовність дій: користувач вибирає пісню або плейлист для прослуховування, програвач починає відтворення. Є зв’язок з наустпним пунктом, коли під час прослуховування алгоритм може надавати рекомендації пісень.

Рекомендація пісень. Алгоритм рекомендує користувачу нові пісні на основі його музичних уподобань та історії прослуховувань. Послідовність дій:

алгоритм аналізує історію прослуховування користувача (які пісні були відтворені, як часто, які плейлисти використовувалися), порівнює їх з іншими піснями в локальній бібліотеці та генерує список рекомендованих треків. Рекомендації відображаються користувачеві подовжуючи наявну чергу прослуховування. На рисунку 2.1 зв'язок пунктирною лінією вказує на те, що це автоматичний процес, який відбувається під час прослуховування.

Оцінка роботи алгоритму. Користувач оцінює якість рекомендацій наданих алгоритмом. Послідовність дій: користувач може ставити «палець вгору» або «палець вниз» рекомендованим пісням, або просто ігнорувати їх. Ці дії впливають на роботу алгоритму рекомендацій, а саме на частоту рекомендацій оціненої пісню в подібних чергах відтворення майбутньому.

Отримання статистики прослуховування. Програма збирає дані про те, які пісні слухав користувач, як часто він їх слухав, та інші показники. На основі зібраних даних, програвач надає користувачеві щомісячний звіт про його музичні звички. Послідовність дій: програма автоматично записує інформацію про відтворення музики протягом місяця. Наприкінці місяця генерується звіт, який включає статистику про найчастіше відтворені пісні, жанри, виконавців та інші показники. Звіт відображається користувачеві у зручному форматі, наприклад, графіку чи таблиці.

Далі було розроблено діаграму діяльності, для того щоб формально відобразити очікуваний від користувача спосіб використання застосунку (рис. 2.2).

Користувач надає доступ до бібліотеки пісень, і може почати одразу її слухати, а програма на фоні буде збирати інформацію про те, які пісні користувач слухає, і у випадку, коли плейлист завершується може запропонувати деяку кількість пісень для подовження черги. Також в полі «збереження інформації про відтворення в базі даних алгоритму» також мається на увазі можливість користувача напряму вплинути на оцінку алгоритму.

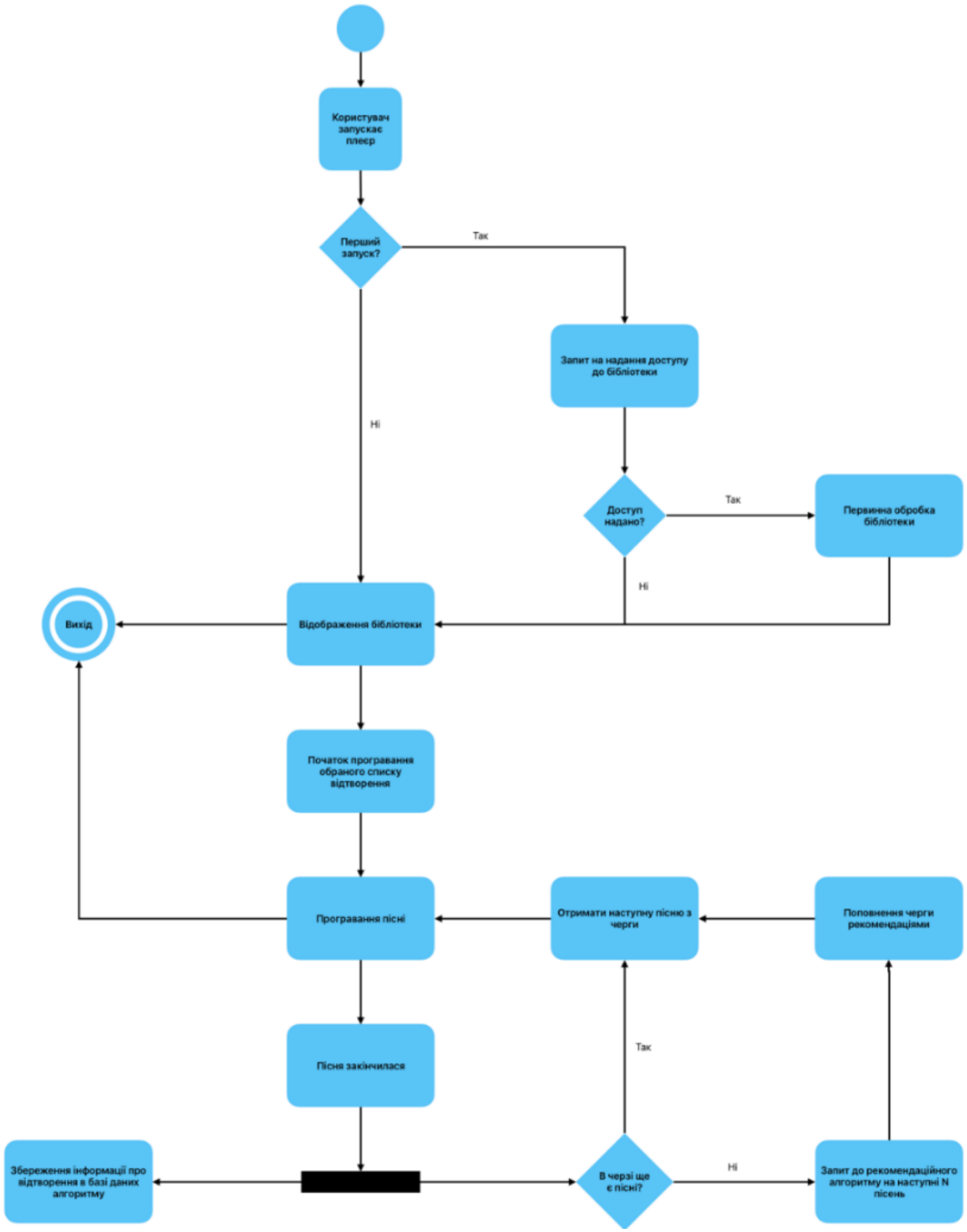

Рисунок 2.2 – Діаграма діяльності

Джерело: розроблено автором

Покроковий опис:

1. початок. Користувач запускає плеєр: процес починається з того, що користувач запускає програму;
2. перевірка чи це перший запуск: плеєр перевіряє чи це перший запуск програми, якщо так то перехід на пункт 3, інакше 6;
3. запит на доступ до бібліотеки: програма просить надати доступ до локальної папки з музикою;
4. перевірка чи надано доступ: плеєр перевіряє, чи надано доступ до бібліотеки, якщо так то перехід на пункт 5, інакше 6;
5. первинна обробка бібліотеки: відбувається первинна обробка файлів. Вони зберігаються в локальну базу даних, а рекомендаційний алгоритм робить первинні зв’язки пісень на основі метаданих, таких як автор, виконавець, жанр і тд.;
6. відображення бібліотеки: користувач бачить інтерфейс програвача з інформацією про поточний трек (і чергу якщо музика програється на пк);
7. початок програвання списку відтворення;
8. програвання пісні: плеєр починає програвати вибраний трек;
9. пісня закінчилась: програвання пісні завершилось, система виконує два наступних кроки (10 і 11) одночасно;
10. збереження інформації про відтворення в базі даних алгоритму: інформація про прослуханий трек (id, час прослуховування, попередня пісня, оцінка рекомендації за наявності) зберігається у внутрішній базі даних плеєра. Це використовується для покращення алгоритму рекомендацій;
11. перевірка чи є в черзі наступна пісня: якщо так, то перехід на пункт 12, інакше 13 пункт;
12. отримати наступну пісню з черги: після завершення поточного треку, плеєр отримує наступну пісню з черги;

13. запит до рекомендаційного алгоритму на наступні N-кількість пісень: алгоритм має підготувати N-кількість пісень, які на його думку є найбільш підходящими для подовження черги. В даному випадку N-кількість означає фіксоване число, яке буде залежати від швидкості роботи алгоритму, і розміру бібліотеки, але не буде перевищувати 20 пісень;
14. поповнення черги рекомендаціями: алгоритм рекомендацій додає до черги підготовлену N-кількість пісень. Повернення до пункту 8, відтворення продовжується з першої рекомендованої пісні;
15. вихід: процес завершується.

Щоб проілюструвати взаємодію між об'єктами та компонентами системи під час виконання цього процесу, було розроблено діаграму послідовності (рис. 2.3). В цій діаграмі візуалізовано виклики компонентів програми під час стандартного сценарію роботи з програмою.

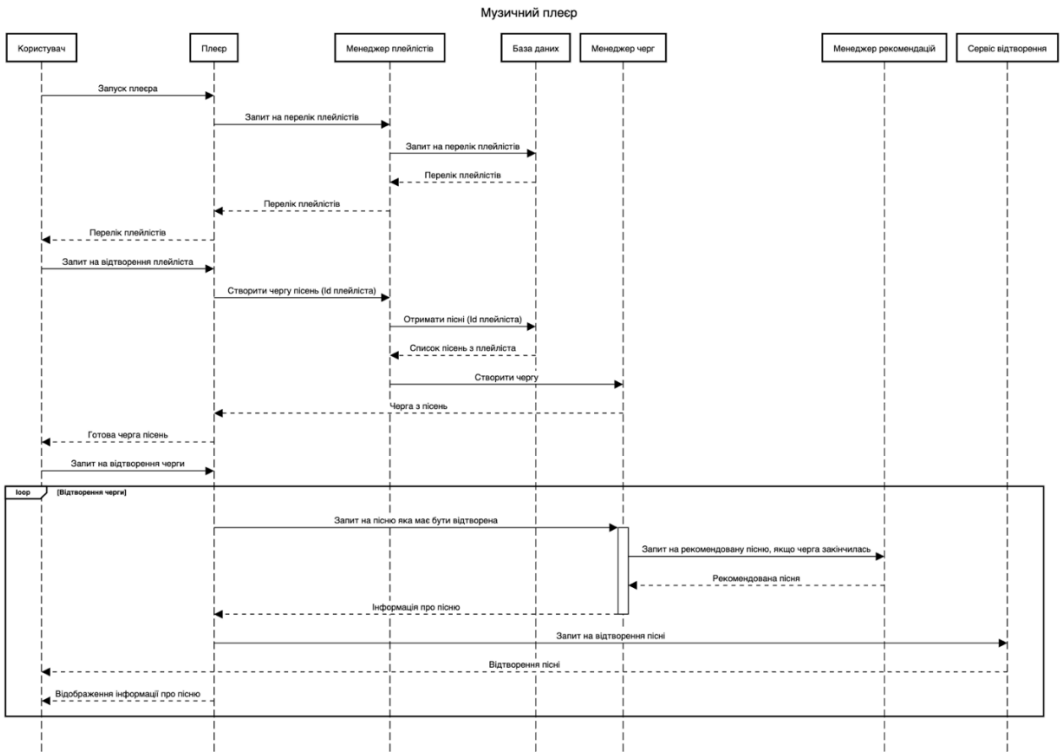

Рисунок 2.3 – Діаграма послідовності

Джерело: розроблено автором

Актори діаграми послідовності:

- користувач: людина, яка використовує музичний плеєр;
- плеєр: основна програма-плеєр, що відповідає за інтерфейс користувача та керування процесом відтворення;
- менеджер плейлистів (Playlist Manager): відповідає за отримання та управління списками відтворення;
- база даних: зберігає інформацію про плейлисти, пісні тощо.
- менеджер черг (Queue Manager): створює та керує чергою пісень для відтворення;
- менеджер рекомендацій (Recommendation Manager): надає рекомендовані пісні, коли черга закінчується;
- сервіс відтворення (Playback Service): відповідає за фактичне відтворення музики.

Більш детально розглянуто цикл відтворення черги:

- запит на пісню для відтворення: плеєр запитує у менеджера черг наступну пісню для відтворення;
- активація менеджера черг: позначає, що менеджер черги активний та обробляє запит;
- запит на рекомендовану пісню: якщо черга закінчилася, менеджер черг звертається до менеджера рекомендацій для отримання наступної пісні;
- рекомендована пісня: менеджер рекомендацій надає рекомендовану пісню менеджеру черги;
- інформація про пісню: менеджер черги передає інформацію про пісню плеєру;
- деактивація менеджера черг: позначає, що Менеджер черги завершив обробку запиту;
- запит на відтворення пісні: плеєр надсилає запит сервісу відтворення для відтворення отриманої пісні;

- відтворення пісні: сервіс відтворення відтворює пісню та передає звук користувачу;
- відображення інформації про пісню: плеєр відображає інформацію про поточну пісню користувачу.

Цей цикл повторюється до тих пір, поки черга не буде повністю відтворена або користувач не зупинить програму. Деякі функції виконуються асинхронно, такі як відтворення пісень і робота рекомендаційного алгоритму, щоб не створювати блокування головного потоку, яке призведе до зависання графічного інтерфейсу.

На завершення, щоб показати конкретні зв'язки та обмін інформацією між компонентами системи, було розроблено діаграму комунікацій (рис. 2.4). В ній відображено основні запити між різними елементами програми під час її виконання.

Кожен компонент відповідає за певну функціональність:

- менеджер пісень (Songs Manager) керує списком пісень (отримання, інформація);
- плеєр (player) є основним компонент, який керує відтворенням музики та взаємодіє з іншими компонентами;
- менеджер плейлистів (Playlists Manager) керування списками відтворення;
- менеджер черг (Queue Manager) керування чергою пісень для відтворення;
- менеджер рекомендацій (Recommendation Manager) надає рекомендації щодо пісень.

2.2 Моделювання структури продукту

У цьому розділі буде детально розписано внутрішню структуру продукту. Нижче представлено класову діаграму, що містить ключові елементи (рис. 2.5).

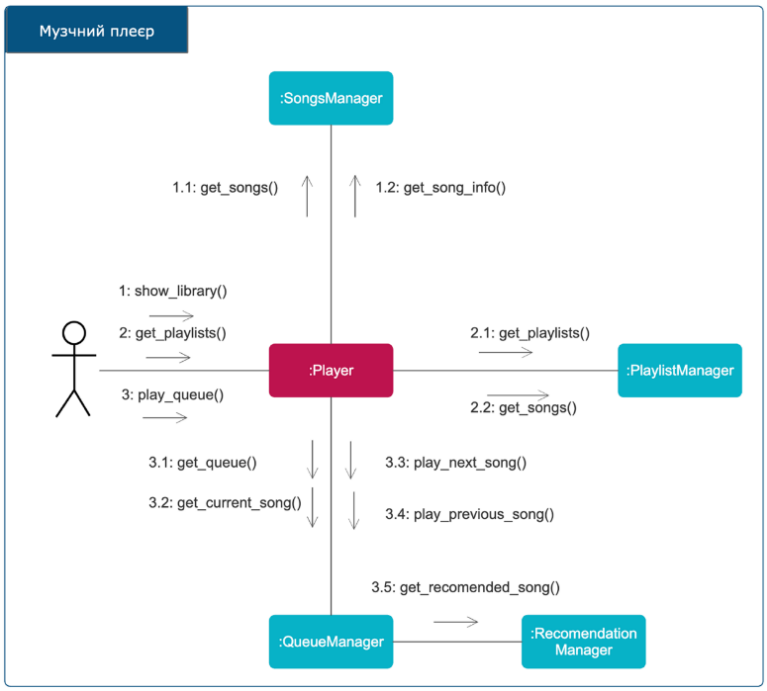

Рисунок 2.4 – Діаграма комунікацій

Джерело: розроблено автором

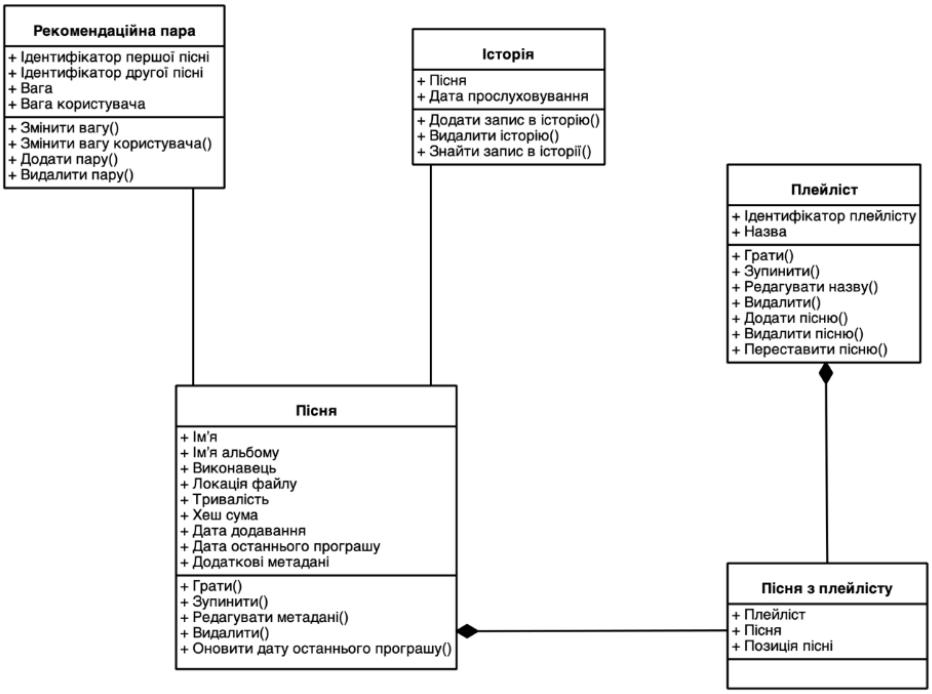

Рисунок 2.5 – Класова діаграма

Джерело: розроблено автором

Нижче детально описано класи, їх атрибути та методи.

Пісня:

- відповідальність: представляє окрему пісню в бібліотеці музики;
- атрибути: ім'я (назва пісні), ім'я альбому, виконавець, локація файлу, тривалість, жанр, дата додавання, дата останнього програшу, додаткові метадані;
- методи: грати(), зупинити(), редагувати метадані(), видалити(), оновити дату останнього програшу().

Плейлист:

- відповідальність: представляє список пісень, організований користувачем;
- атрибути: ідентифікатор плейлисту, назва;
- методи: грати(), зупинити(), редагувати назву(), видалити(), додати пісню(), видалити пісню(), переставити пісню().

Історія:

- відповідальність: зберігає історію прослуховування музики користувачем;
- атрибути: пісня, дата прослуховування;
- методи: додати запис в історію(), видалити історію(), знайти запис в історії().

Рекомендаційна пара:

- відповідальність: представляє рекомендовану пару пісень для користувача;
- атрибути: ідентифікатор першої пісні, ідентифікатор другої пісні, ваговий коефіцієнт метаданих, ваговий коефіцієнт поведінки користувача;
- методи: змінити вагу(), додати пару(), видалити пару().

Пісня з плейлисту:

- відповідальність: зв'язує пісню з плейлистом та визначає її позицію в плейлисті;
- атрибути: плейлист, пісня, позиція пісні.

Зв'язки між класами:

- агрегація: плейлист містить пісні з плейлисту. Тобто плейлист має колекцію пісень, але пісня може існувати незалежно від плейлиста;
- композиція: історія містить пісні. Це означає, що запис в історії не може існувати без пісні;
- асоціація: рекомендаційна пара пов’язана з піснями (через їхні ідентифікатори).

Було розроблено діаграму об’єктів для відображення вигляду структури програми в конкретний момент виконання (рис. 2.6).

Діаграма представляє структуру даних плеєра. Player View Model відображає поточну пісню та чергу. Об’єкт Queue. Recommendation Manager формує рекомендації на основі інформації з черги, пропонуючи список рекомендованих пісень. Об'єкт Queue містить список пісень (Song Model) та вказує поточну пісню та її індекс у списку. Кожен об’єкт Song Model зберігає дані про окрему пісню: назву, виконавця, альбом, шлях до файлу, тривалість, хеш, дату додавання, дату останнього відтворення та додаткові метадані.

Для організації великої кількості класів та компонентів, було розроблено діаграму пакетів (рис. 2.7), щоб розглянути їх групування.

Опис діаграми продуктів:

Пакет «Player» (Плеєр): основний пакет, що містить компоненти для функціональності плеєра.

Пакет «Queue» (Черга): містить дані про пісні:

- songs: колекція пісень в черзі;
- song: опис окремої пісні.
- Queue Manager: керує чергою пісень, використовуючи Queue Service для роботи з чергою.

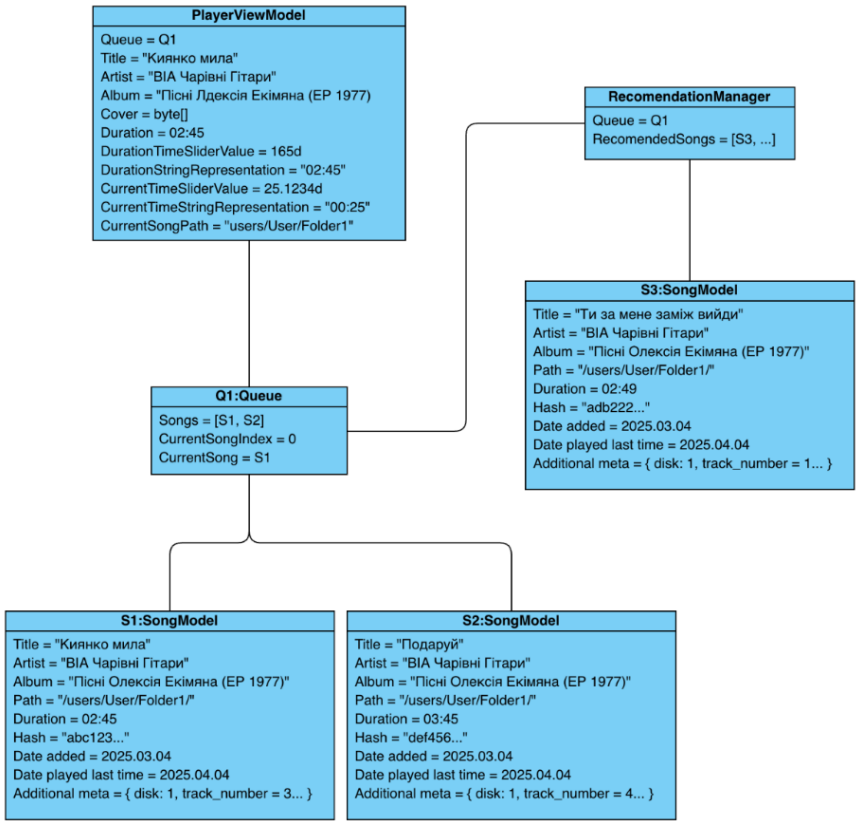

Рисунок 2.6 – Діаграма об’єктів

Джерело: розроблено автором

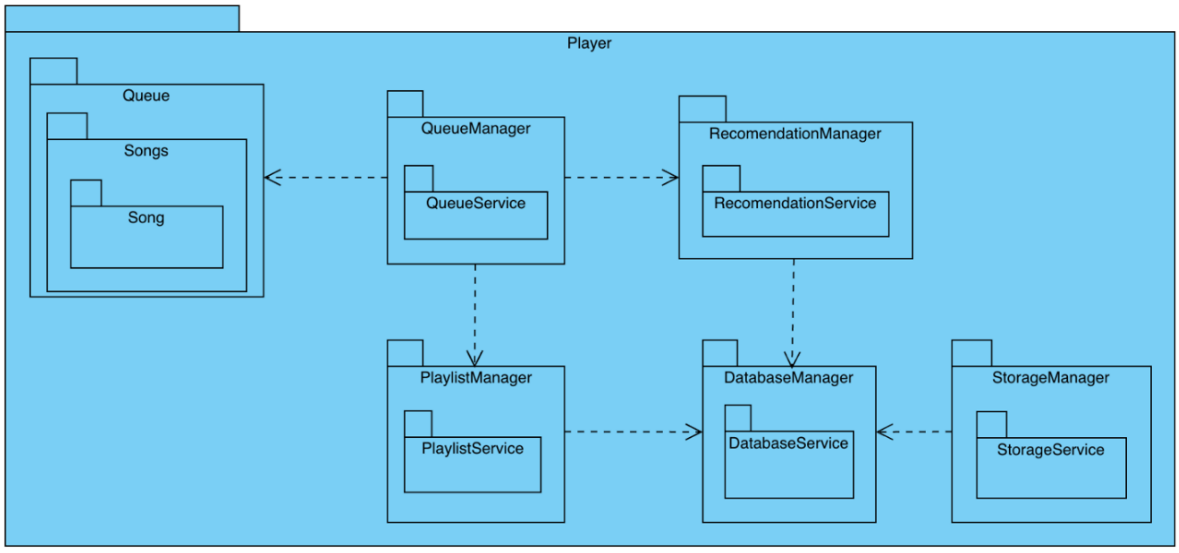

Рисунок 2.7 – Діаграма пакетів

Джерело: розроблено автором

Recommendation Manager: надає рекомендації, використовуючи Recommendation Service.

Playlist Manager: керує плейлистами, використовуючи Playlist Service.

Database Manager: взаємодіє з базою даних через Database Service.

Storage Manager: зберігає та отримує дані через Storage Service.

Пояснення різниці між менеджером і сервісом буде надано в розділі 2.3

Також було розроблено грубий ескіз інтерфейсу для версії для ПК, яку розглянемо нижче (рис. 2.8).

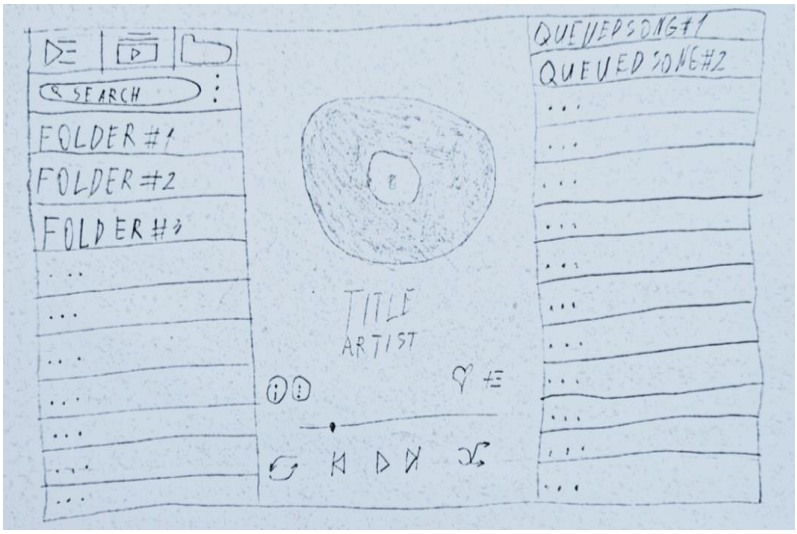

Рисунок 2.8 – Ескіз дизайну версії для ПК.

Джерело: розроблено автором

Плеєр складається з одної сторінки, і ця сторінка розділена на три частини. Поділимо екран на три стовпчики відповідно до рисунку 2.8.

Лівий стовпчик містить 3 перемикачі зверху. Кнопка «Черга» – це перелік черг, які створив користувач. Черги дозволяють гнучко змінювати перелік пісень не вносячи змін до існуючих і більш сталих плейлистів. Кнопка «Плейлисти» – це перелік списків відтворення. Кнопка «Папки» – це розділення пісень по їх розташуванню в файловій системі. Під цими кнопками є рядок пошуку, який дозволяє знайти щось конкретне в переліку нижче. Праворуч від нього розташована іконка з вертикальними трьома крапками, яка

перенаправляє користувача на базові налаштування плеєру. Під пошуком розташований перелік елементів які були обрані описаними вище кнопками.

Центральна стовпчик є панеллю керування програвачем. Тут можна: перемотувати пісні, перемішувати черги, чи задавати цим чергам правила повтору. Дізнатися інформацію про пісню чи додати її в плейлист.

Правий стовпчик відповідає за відображення нинішньої черги програвання, за допомогою якої користувач може пропустити одразу кілька пісень чи переставити їх місцями.

Далі буде представлено ескіз мобільного дизайну (рис. 2.9). Він повторює елементи ПК версії, але розділений на вкладки, щоб рівномірно розділити елементи, які не вміщуються на малому екрані і уникнути конфліктів з системною навігацією жестами. Також було використано багато пустого простору зверху екрану, щоб користувачу не треба було тягнутись до верхньої частини екрану, що є не зручною дією на великому екрані чи якщо у людини малі руки.

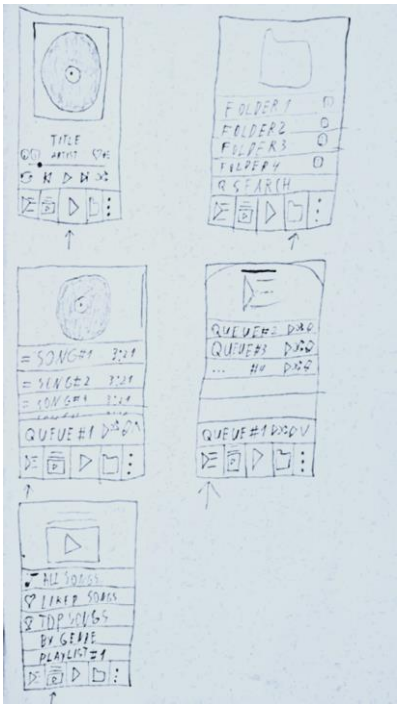

Рисунок 2.9 – Ескіз дизайну версії для телефону

Джерело: розроблено автором

2.3 Опис архітектури продукту

В даному розділі буде представлено простір імен, класів і їх методів.

Кросплатформений офлайн плеєр музики було вирішено назвати «MAUSIC» на честь фреймворку .NET MAUI та англійського слова «music», що означає музика.

Простір імен MAUSIC містить усі попередньо описані простори імен. Цей простір імен також включає файли App та AppShell (розділені на два файли як сторінки), а також файл MauiProgram. Файл App ініціалізується першим при запуску програми. AppShell представляє собою головну «сторінку» застосунку, яка містить інші сторінки. Файл MauiProgram відповідає за ініціалізацію основного функціоналу програми, включаючи моделі сторінок, менеджери та сервіси.

Простір імен Data включає в себе все що пов’язано в з конкретними даними, такими як константи і сутності. В відповідно тримає два простори імен, Constants і Entities. В просторі Constants розташовані файли констант, а в просторі Entities, містить в собі конкретні сутності, такі як пісні, плейлисти, історію, тощо, на основі яких побудовані таблиці в базі даних. Ці сутності наслідують базову сутність BaseEntity, яка знаходиться в просторі імен Data.Entities.Abstract, і описує базове поле основного ключа Id.

Простір імен Managers містить класи, що забезпечують координацію операцій з даними та бізнес-логікою. Менеджери є абстракціями вищого рівня, які інкапсулюють взаємодію з сервісами для виконання більш комплексних завдань. Сервіси відповідають за вузькоспеціалізовані операції, наприклад, отримання даних з бази даних, тоді як менеджери обробляють результати цих операцій та інтегрують їх в бізнес-процеси.

Простір імен Managers включає наступні класи: DatabaseManager, HistoryManager, PlaylistManager, QueueManager, RecommendationManager, SongsManager, StorageManager.

Менеджер баз даних побудовані на абстракції сутностей, що дозволяє уникати створення спеціалізованих класів для роботи з кожною окремою

таблицею бази даних. Це забезпечує гнучкість та спрощує підтримку структури даних.

Простір імен Mappers містить статичні класи (ті яким не треба ініціалізація), котрі надають статичні методи Map() для перетворення об'єктів сутностей у відповідні моделі та навпаки. Включає наступні класи: FolderMapper, PlaylistMapper, SongMapper.

Простір імен Models містить класи, що представляють собою репрезентацію сутностей для відображення даних користувачу. Ці моделі адаптовані для зручного представлення інформації в інтерфейсі користувача.

Простір імен Models включає наступні класи: FolderModel, PlaylistModel, SongModel, SongsQueue. Всі моделі наслідують абстрактний класу BaseModel.

Простір імен PageModels містить класи, що представляють собою моделі сторінок, які використовуються для відображення даних користувачу в повноцінних сторінках програми. На відміну від простих моделей, призначених для представлення окремих елементів інтерфейсу, PageModels інтегруються з повним функціоналом сторінки та керують її поведінкою.

Простір імен PageModels включає наступні класи: FoldersPageModel, PlayerPageModel, PlaylistsPageModel. Всі моделі сторінок наслідуються від абстрактного класу BasePageModel. PageModels мають власні методи, що дозволяють їм самостійно викликати інші елементи програми, такі як менеджери, мапери або інші моделі.

Простір імен Pages містить класи, що представляють собою сторінки програми. Кожна сторінка складається з пари файлів: .xaml, який визначає розмітку інтерфейсу користувача у форматі, схожому на XML, та .xaml.cs, який обробляє події інтерфейсу (натискання кнопок, перетягування слайдеру, тощо) і взаємодіє з відповідною PageModel для виконання більш складної логіки.

Простір імен Pages включає наступні класи: FoldersPage, PlayerPage, PlaylistsPage, QueuePage. Всі сторінки унаслідуються від абстрактного класу BasePage, який визначає загальні властивості та методи для всіх сторінок.

Простір імен Platforms: містить простори імен з файлами, специфічними для різних платформ. Ці файли переважно не використовуються безпосередньо в коді, але слугують для здійснення запитів на дозволи до системних функцій, зокрема, доступ до сховища. Підпростори імен включають Android, iOS, MacCatalyst, Tizen та Windows, кожен з яких представляє окрему платформу розробки.

Простір імен Resources містить файли ресурсів застосунку. Він включає підпростори імен AppIcon, Fonts, Images, Raw, Splash та Styles. Кожен з цих підпросторів зберігає відповідні ресурси для застосунку, такі як іконки, шрифти, зображення, необроблені дані, екрани завантаження та стилі.

Простір імен Views містить файли представлень застосунку (views). Він включає файли FolderView, PlaylistView, QueuedSongView, а також SongView. Файли як і файли сторінок поділені на .xaml та .cs.

Ці компоненти реалізовані як одинаки. Тобто це патерн проєктування, який гарантує, що клас має лише один екземпляр, та надає глобальну точку доступу до нього [15].

Висновок до розділу 2

Проєктування музичного плеєра з алгоритмами рекомендацій було виконано з акцентом на детальний аналіз функціональності та структурування системи. Була розроблена архітектура, що враховує моделювання поведінки продукту, що підкріплено діаграмами використання, діяльності, послідовності та класів.

Основні результати роботи:

- функціональне моделювання: детально описано основні сценарії використання плеєра, включаючи доступ до бібліотеки музики,

створення плейлистів, рекомендації та збір статистики прослуховування.

- структурне проєктування: створено класову діаграму, що визначає ключові класи та їх атрибути/методи, діаграму об’єктів яка дозволяє уявити програму в процесі виконання, а також діаграму пакетів для організації компонентів системи.
- ескізи інтерфейсу користувача: створено ескізи desktop- та мобільної версій плеєра, для зручного використання на різних платформах.
- архітектурний план: розроблено структуру просторів імен для забезпечення модульності та чіткого поділу відповідальності між компонентами.

Проведена робота заклала основу для подальшої реалізації системи музичного плеєра з алгоритмами рекомендацій.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ КРОСПЛАТФОРМЕНОГО ПЛЕЄРУ

3.1 Реалізація та конструювання програмного продукту

В даному розділі буде більш детально описано засоби реалізації програмного продукту, і його особливості.

Для розробки було обрано мову програмування *C#*. Це високорівнева, об’єктно-орієнтована мова програмування зі строгою типізацією.

Є декілька причин чому було зроблено саме цей вибір, ключовою є наявність в мене п’яти років досвіду розробки цією мовою програмування, наявність цього досвіду дає мені змогу писати якісний код сильно не занурюючись в документацію. Другою причиною є тим, що це одна з небагатьох мов яка підтримує фреймворки кросплатформеної розробки.

Кросплафтормена розробка – це вид розробки програмного забезпечення, який дозволяє писати спільний код який може виконуватись на різних платформах.

Фреймворком програмного продукту було обрано *.NET MAUI*. Це фреймворк розроблений Microsoft для кросплатформеної розробки, що дозволяє економити ресурси при розробці, дозволяючи наймати всього один штат працівників, на відміну від нативної розробки, яка потребує окремого штату програмістів для кожної платформи.

Нативна розробка – створення програмного забезпечення спеціально під якусь платформу (Windows, Linux, MacOS, Android, iOS). Для цього виду розробки зазвичай використовують офіційно підтримувані мови програмування, такі як Swift для MacOS та iOS, чи Kotlin для Android.

Перевагами цього фреймворку є можливість написання бізнес-логіки і інтерфейсу в одному місці, а фреймворк збирає проєкт під відповідну платформу використовуючи нативні елементи інтерфейсу і, за можливістю, елементи керування системними запитами (такі як системні дозволи, сповіщення, файлова система, тощо). Також важливим уточненням до

сказаного мною вище, про п’ять років досвіду є те, що більшість з цього часу я розробляв застосунки з використанням фреймворку Xamarin і Xamarin.Forms, що є попередниками MAUI. Вони відрізняються тим, що в Xamarin, була спільна бізнес логіка, але графічний інтерфейс і передавання даних зі спільної частини в інтерфейс і назад потребували окремої роботи з кожною платформою, що досить сильно збільшує кількість роботи. З урахуванням того, що Xamarin давав змогу працювати з кожною платформою окремо – це дозволяло проводити більш гнучку і глибоку роботу з кожною платформою. Xamarin.Forms, є більш високорівневою абстракцією, інтерфейс вже створюється в одному місці для всіх платформ, але це має свої недоліки. Переважно це досить важкий доступ до нативних особливостей кожної з платформ, розмір фінального проєкту на платформі займав дуже багато пам’яті, і продуктивність самого продукту було досить низька. Також підтримка цих фреймворків була припинена Microsoft, які зараз активно розвивають MAUI. Потенційним конкурентом до MAUI, був Flutter віл Google, через свою надзвичайну продуктивність і підтримку платформи Linux, але для роботи з ним треба знати мову програмування Dart, на вивчення якої знадобилось би витратити досить багато часу.

Основними платформами для фокусу під час розробки і тестування є MacOS та Android, хоча з урахуванням особливостей MAUI, програмний продукт має запуститися і на Windows з iOS, хоча й передбачаються баги, так як цим платформам не було приділено уваги.

При розробці плеєру було обрано наступні бібліотеки:

CommunityToolkit.Maui.MediaElement [16]. Ця бібліотека надає можливість відтворювати медіафайли, як музичні так і відео, та додає можливість використовувати системне сповіщення для керування музикою (рис. 3.1).

Основною перевагою цієї бібліотеки є те що вона розроблена Microsoft, і відповідно матиме довгу підтримку, та не треба буде перейматись про доброчесність написаного коду.

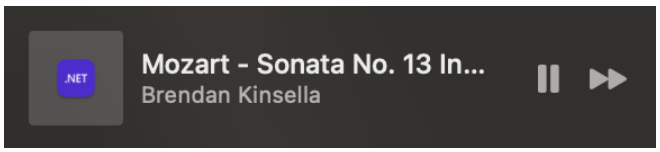

Рисунок 3.1 – Системне сповіщення під час відтворення музики на MacOS

Джерело: розроблено автором

TagLibSharp [17]. Бібліотека призначена для зчитування метаданих пісні (назва, автор, обкладинка, тощо). Під час дослідження можливих варіантів зчитування метаданих, користувачі на форумах переважно посилались на цю бібліотеку, що є оптимальним рішенням в порівнянні з вивченням особливостей записування метаданих з великої кількості різноманітних музичних форматів та написанням цієї логіки з нуля.

CommunityToolkit.Mvvm [18]. Ця бібліотека дозволяє під час розробки використовувати патерн Model-View-ViewModel (рис. 3.2).

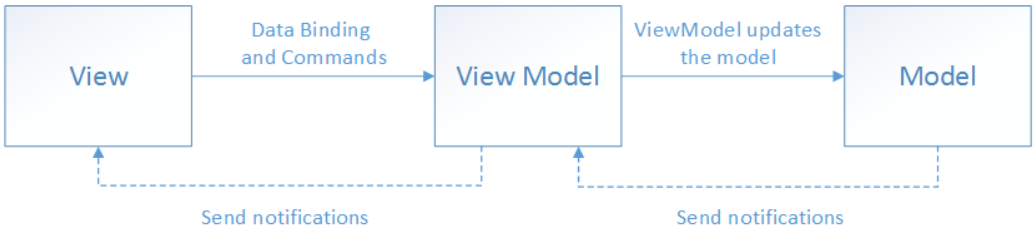

Рисунок 3.2 – Патерн MVVM

Джерело: Microsoft learn [19]

Це рекомендований Microsoft патерн програмування який дозволяє розділяти бізнес логіку від інтерфейсу. Основними перевагами цього патерну які виділяють Microsoft є:

- створення юніт-тестів без використання інтерфейсу;
- можливість змінювати дизайн без внесення змін в логіку ViewModel чи Model;
- ViewModel є адаптером між моделлю і View, це виносить логіку роботи з моделлю в ViewModel і зменшує ризики під час змін в

бізнес логіку, в порівнянні з випадками коли бізнес логіка знаходиться в моделі.

CommunityToolkit.Maui [20]. Офіційна білбіотека, яка додає розширений перелік анімацій, контролів (елементи інтерфейсу), поведінок, конверторів, тощо. Ця бібліотека надає змогу працювати з файловою системою і відображати спливаючі вікна.

Також є кілька бібліотек які необхідні для роботи інших бібліотек, чи є важливими для роботи самої програми.

Вбудовані бібліотеки:

Microsoft.Maui.Controls [21]. Основний фреймворк MAUI для створення UI елементів та керування взаємодією з користувачем.

CommunityToolkit.Maui.Core [22]. Основний набір компонентів Community Toolkit для MAUI.

Microsoft.Extensions.Logging.Debug [23]. Бібліотека для запису логів інформації про роботу програми з метою налагодження та моніторингу.

Microsoft.Net.ILLink.Tasks [24]. Інструмент для оптимізації розміру виконуваного файлу MAUI додатку шляхом видалення невикористаного коду.

На рисунку 3.3 зображено фінальну файлову систему проєкту.

Також необхідно приділити увагу алгоритму рекомендацій. При додаванні пісні в бібліотеку він створює рекомендаційні пари. Кожна пара має в собі ключі двох пісень (у першої завжди Id має бути меншим за другу) таким чином вибудовується, верхня діагональна матриця без самої діагоналі, так як алгоритм не передбачає рекомендацію пісень, що вже є у черзі.

Для підрахунку вагового коефіцієнту метаданих, тобто зв’язку пари пісень було використано кореляцію. З файлу пісні зчитуються метадані і по таким полям як виконавець або гурт, альбом, жанри (перелік через «;»), кількість ударів на секунду, усі виконавці та гурти (перелік через «;») відбувається порівняння. Базова ваговий коефіцієнт метаданих становить «0.5», а кожне порівняння додає чи віднімає до нього «0.1» в залежності від результативності співпадіння (рис. 3.4).

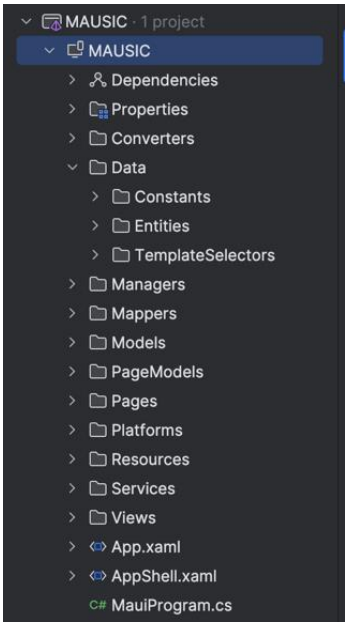

Рисунок 3.3 – Фінальна структура проєку

Джерело: розроблено автором

```
2 usages  Yurii *
private float CalculateWight(SongEntity firstSongEntity, SongEntity secondSongEntity)
{
    var weight :float  = RecommendationConstants.DefaultWeight;

    weight = firstSongEntity.Artist == secondSongEntity.Artist
        ? weight + RecommendationConstants.AutoGeneratedWeight
        : weight - RecommendationConstants.AutoGeneratedWeight;

    weight = firstSongEntity.Album == secondSongEntity.Album
        ? weight + RecommendationConstants.AutoGeneratedWeight
        : weight - RecommendationConstants.AutoGeneratedWeight;

    try
    {
        weight = (firstSongEntity.Genres != null && secondSongEntity.Genres != null) &&
                 firstSongEntity.Genres.Split(';').ToHashSet().Overlaps( other: secondSongEntity.Genres.Split(';'))
            ? weight + RecommendationConstants.AutoGeneratedWeight
            : weight - RecommendationConstants.AutoGeneratedWeight;


        weight = (firstSongEntity.BPM != 0 && secondSongEntity.BPM != 0) &&
                 Math.Abs((int)firstSongEntity.BPM - secondSongEntity.BPM) <= 20
            ? weight + RecommendationConstants.AutoGeneratedWeight
            : weight - RecommendationConstants.AutoGeneratedWeight;

        weight = (firstSongEntity.Performers != null && secondSongEntity.Performers != null) &&
                 firstSongEntity.Performers.Split(';').ToHashSet().Overlaps( other: secondSongEntity.Performers.Split(';'))
            ? weight + RecommendationConstants.AutoGeneratedWeight
            : weight - RecommendationConstants.AutoGeneratedWeight;
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        return RecommendationConstants.DefaultWeight;
    }

    return weight;
}
```

Рисунок 3.4 – Код підрахунку вагового коефіцієнту метаданих пари пісень

Джерело: розроблено автором

Також у користувача є можливість напряму впливати на роботу алгоритму. Для цього в алгоритмі існує ваговий коефіцієнт користувача і його базове значення «0.5».

У зв’язку з тим, що користувацький зворотній зв’язок є більш цінним у порівнянні зі зміною вагового коефіцієнту метаданих, було прийнято рішення про те, що ваговий коефіцієнт користувача має змінюватись на «0.4», а це є високим значенням. Тобто, рекомендовані пісні мають кнопки «палець вгору» та «палець вниз» на сторінці плеєра, і якщо користувачу не подобається дана рекомендація він може натиснути на кнопку «палець вниз» і ваговий коефіцієнт користувача зміниться на «-0.4» і навпаки. Зміна відбувається до всіх пар «пісня з черги» та «рекомендована пісня».

Якщо виникне потреба відкалібрувати алгоритм змінні, що відповідають за вагові коефіцієнти винесені в окремий файл константами. Тому, якщо «0.4» все ж виявиться завеликим значенням, можна буде змінити змінну в одному місці.

Коли у користувача лишається всього дві пісні в черзі і не встановлено повторне відтворення черги по колу, плеєр викликає алгоритм з запитом на кілька рекомендованих пісень. Алгоритм отримує з бази даних всі можливі пари, відсікає пари пісень, які вже є у черзі і підраховує у кожної пісні фінальну вагу, з наступною пропорцією: 40% – ваговий коефіцієнт метаданих, 60% – ваговий коефіцієнт користувача (рис. 3.5). Якщо сумарна нормалізована вага не становить як мінімум «0.4», пісня видаляється з переліку. Алгоритм підбирає пісні з найбільшим коефіцієнтом і повертає їх у тій кількості, яку запросив плеєр.

Цей алгоритм не є фінальною версією. Потенційним місцем для покращення є періодичне повернення пар з нижчим рейтингом, але треба бути обережним, щоб випадково не повернути якусь пісню в жанрі «heavy metal» коли користувач слухає пісні в жанрі «Lo-Fi». Також фінальні пісні зараз повертаються з найбільшою вагою по спаданню, має сенс змінити в

майбутньому цю систему, на повернення більш хаотичне, рекомендувати кількість пісень більш розбіжною вагою і перемішувати фінальний результат.

```
1 usage  Yurii
private List<RecommendationPairEntity> TrimRecommendationPairEntities(
    List<RecommendationPairEntity> pairEntities,
    int count)
{
    var dictionary = new Dictionary<int, float>();

    pairEntities.ForEach(model :RecommendationPairEntity  =>
    {
        float normalizedSum = (model.AlgorithmWeight * 0.8f + model.UserWeight * 1.2f) / 2;

        if (normalizedSum >= 0.4f)
        {
            dictionary[model.Id] = normalizedSum;
        }
    });

    dictionary = dictionary.OrderByDescending(pair => pair.Value).ToDictionary();

    return dictionary.Select((pair) => pairEntities.First((entity) => entity.Id == pair.Key)).Take(count).ToList();
}
```

Рисунок 3.5 – Нормалізація ваги потенційних пар і повернення запитаної кількості рекомендацій

Джерело: розроблено автором

Окрім загальних засобів розробки та алгоритму рекомендації ключовим елементом є база даних, вона зберігає всю необхідну для коректного функціонування застосунку інформацію.

База даних складається з наступних таблиць:

Folder: зберігає інформацію про папки з музикою, надані користувачем.

Поля (будуть позначатись як і в коді програми, типами даних C#, причина такого підходу буде описана нижче):

- id: int, primary key;
- path: string, шлях до папки;

Playlist: містить дані про плейлисти

Поля:

- id: int, primary key;
- title: string, назва плейлисту;

Playlist song: встановлює зв'язок між плейлистами та піснями.

Поля:

- id: int, primary key;
- playlistId: int, foreign key, вказує на плейлист;
- songId: int, foreign key, вказує на пісню;

Song: зберігає інформацію про кожну пісню.

Поля:

- id: int, primary key;
- title: string, назва пісні;
- artist: string, виконавець;
- album: string, альбом;
- duration: string, тривалість;
- path: string, шлях;
- isFavourite: bool, чи є в обраному;
- genres: string, жанри, розділені знаком «;»;
- performers: string, виконавці, розділені знаком «;»;
- BPM: uint, кількість бітів на секунду (відображає швидкість або інтенсивність пісні).

Поле hash, яке раніше з’являлось в роботі відсутнє, так як хеш планувалось використовувати для функції порівняння пісень, щоб можна було відслідковувати переміщення пісень та потенційні дублікати файлів. Ця функція поки імплементована не була.

Recommendation pair: містить пари пісень для рекомендацій з відповідними вагами алгоритму та користувача.

Поля:

- id: int, primary key
- firstSongId: int, foreign key, id першої пісні.
- secondSongId: int, foreign key, id другої пісні

– algorithmWeight: float, вага імовірності сумісності пісень, визначається алгоритмом.
– userWeight: вага float, вага імовірності сумісності пісень, визначається поведінкою і зворотнім зв’язком користувача.

Нижче представлено діаграму фінальної версії бази даних (рис. 3.6). Вийшла база даних третього рівня нормалізації. Потенційною дією для покращення і оптимізації, було би винести наступні поля таблиці пісень, які можуть повторюватись в окремі таблиці, це може бути альбом, жанри і виконавці. Рушієм бази даних було обрано SQLite, це досить легка і мала база даних в порівнянні з аналогами, яка повністю покриває всі запити цього проєкту.

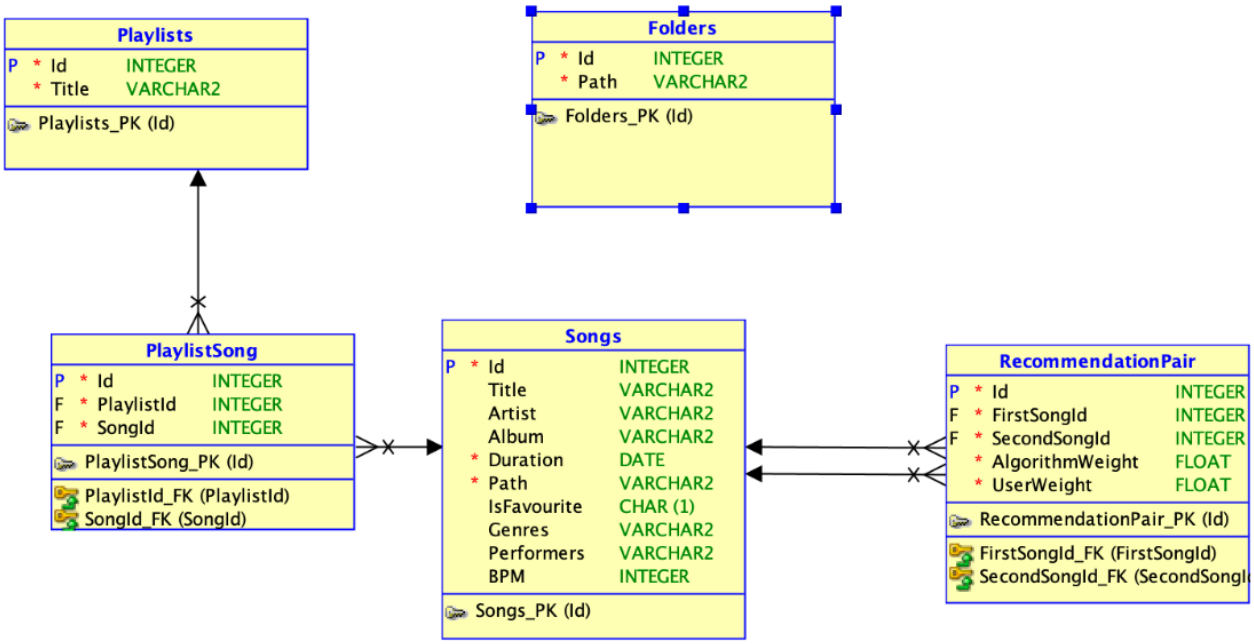

Рисунок 3.6 – UML діаграма бази даних

Джерело: розроблено автором

Для роботи з базою даних було використано бібліотеку SQLite-Net-PCL [25]. Це бібліотека котра свого часу розроблялась для Xamarin.IOS, але з часом розрослася до підтримки кросплатформеної розробки. Основною перевагою цієї бібліотеки є можливість взаємодіяти з базою даних в форматі

схожому на LINQ. LINQ – Language Integrated Query, дозволяє взаємодіяти з перелічувальними типами даних (список, словник, колекція, тощо), за допомогою функцій схожих на мову SQL. Це дозволяє взаємодіяти з базою даних знайомою мовою, без потреби в написані додаткового коду і запитів. Бібліотека вміє працювати з типами даних c#, для створення таблиці використовується клас, на полів якого і створюються колонки в таблиці. Важливою особливістю є те, що поля можуть бути виключно примітивами (int, float, bool, string, тощо), інші класи використовувати не можна, а перетворення примітива в відповідний тип сумісний з базою даних відбувається автоматично.

3.2 Тестування програмного продукту

Функціональне тестування забезпечує якість продукту, відповідність його до заявлених функцій і перевіряє фактичну якість роботи кінцевої програми.

Частина тестування відбувалась під час написання коду, класичним методом є «Happy path», яке саме по собі є простим методом тестування, в якому користувач іде по заявленому, очікуваному шляху виконання програми [26]. Це тестування періодично вказувало на регресії, тобто, те що працювало раніше і перестало через зміни в коді (похідне від регресивного тестування [27]). До прикладу під час роботи зі спливаючими вікнами періодично ламалась черга. Ці помилки одразу ж виправлялись.

Більш детальне тестування плеєру призвело до виявлення наступних проблем:

Шторка з відображенням пісні працює некоректно на обох платформах. На мобільній версії відсутня кнопка перемотування пісні на наступну, а кнопка перемотування назад повертає трек на стартову позицію. На версії для комп’ютера перемотування теж відсутнє. Після детальнішого дослідження проблема була виявлена в бібліотеці CommunityToolkit.Maui.MediaElement. Можливість це виправити після перегляду документації і форумів не виявлена.

Під час перемотування треків з високою роздільною здатністю на комп’ютері, починається неправильне відображення поточного часу треку, тож відображається ніби трек грає на 5-10 секунд довше ніж його заявлена довжина. Дослідження знову привело до бібліотеки згаданої вище, це баг який вже досить давно існує але ще не був виправлений [28].

Рекомендаційний алгоритм при першому спрацьовуванні іноді повертає ідентичні рекомендації двічі. Проблема полягає у самій взаємодії алгоритму і плеєра.

На телефоні, в переліку пісень (будь якому) досить низька продуктивність, і прокручування списку досить сильно зависає. Дослідивши проблему було виявлено те, що в метаданих пісень лежать дуже великі картинки, які сильно впливають на продуктивність колекцій на мобільних девайсах.

Загалом тестування виявило кілька багів, які потребують усунення. Якісь будуть легкими, як то проблема з рекомендаціями, а деякі будуть потребувати досить кропіткої роботи, такої як зменшення роздільної здатності картинок і заміну бібліотеки відтворення музики. Хоча це і є досить об’ємною задачею, враховуючи кількість проблем які створює бібліотека MediaElement, є сенс відійти від неї і знайти гарний аналог, навіть якщо доведеться доробити функціонал якого не вистачає.

Далі було проведено більш детальне модульне тестування. Була протестована загальна швидкість виконання основних класів відповідальних за бізнес логіку, та правильність даних повернених ними.

Було помічено аномалію, в якій перший запит в базу даних завжди відпрацьовував довше за інші. Перший займає виконується 500-2000 мс, в той час як другий запит зазвичай виконується за 100-150мс. На рисунках 3.7 і 3.8 проілюстровано як одні і ті ж запити переставлені місцями виконуються один і той самий час, якщо брати до уваги погрішність, при тому що запит «GetPlaylistSongs» більш важкий в виконанні, так як треба знайти всі пісні і перетворити їх в необхідний формат, а не взяти першу можливу пісню.

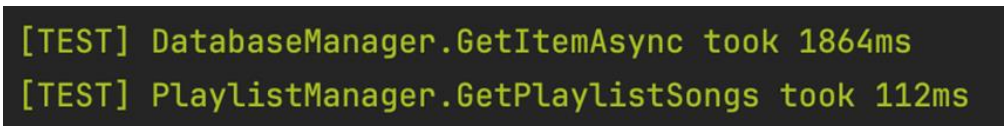

Рисунок 3.7 – Результат виконання запиту в базу данних

Джерело: розроблено автором

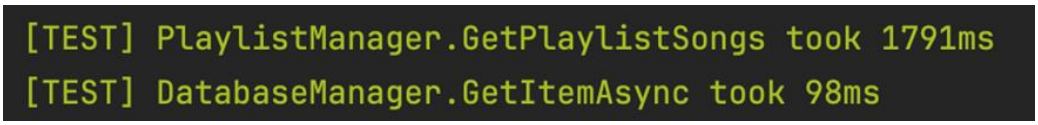

Рисунок 3.8 – Результат виконання запитів в базу даних у зворотньому порядку

Джерело: розроблено автором

Скоріше за все причина затримки полягає в тому що під час тестування відкривається база даних і одразу відбувається запит в неї, що викликає досить велику затримку, доки база даних остаточно не відкриється.

Далі було проведено тестування на великій кількості файлів (трохи менше однієї тисячі). Це вказало на досить серйозну проблему, обробка файлів займає приблизно 100 секунд, а створення рекомендацій не завжди доходить кінця через обмеження системи. Звісно це не блокує головний процес, так як це обробка відбувається в фоновому режимі, але користувач не буде чекати доки на фоні відбудеться якийсь довгий процес.

Для вирішення цієї проблеми був переглянутий код збереження пісень і створення пар. Після поверхневого огляду проблема одразу стала зрозумілою. Всі ці обчислення відбуваються синхронно, тобто один за одним (рис. 3.9).

Звідси виходить просте розуміння, що ~1000 запитів в файлову систему по черзі витягуючи дані з різних файлів є дуже повільною процедурою, а створення рекомендаційних пар так тим поче, адже як було описано раніше, кожна пісня має мати пару з кожною іншою піснею, хоч ітерації і не є такими ж важкими як зчитування файлів, для 1000 пісень загальна кількість пар

складатиме приблизно 499,500 пар згідно формулі суми чисел до числа N-1 де sum = (n*n)/2.

```
public async Task<int> TryCreateRecommendationPairs(List<SongEntity> songEntities)
{
    var newPairs = new List<RecommendationPairEntity>();

    for (int i = 0; i < songEntities.Count; i++)
    {
        for (int j = i + 1; j < songEntities.Count; j++)
        {
            var pairEntity = new RecommendationPairEntity()
            {
                FirstSongId = songEntities[i].Id,
                SecondSongId = songEntities[j].Id,
                AlgorithmWeight = CalculateWight(songEntities[i], songEntities[j]),
                UserWeight = RecommendationConstants.DefaultWeight
            };

            newPairs.Add(pairEntity);
        }
    }

    var existingPairs :List<RecommendationPairEntity> = await _recommendationService.GetAllRecommendationPairs();

    int result = 0;

    if (existingPairs.Count == 0)
    {
        result = await _recommendationService.AddRecommendationPairs(newPairs);

        return result;
    }

    var songs :List<SongEntity> = await _songsManager.GetAllSongs();

    foreach (var song in songEntities)
    {
        songs.Remove(song);
    }

    for (int i = 0; i < songs.Count; i++)
    {
        foreach (var songEntity in songEntities)
        {
            var pairEntity = new RecommendationPairEntity()
            {
                FirstSongId = songs[i].Id,
                SecondSongId = songEntity.Id,
                AlgorithmWeight = CalculateWight(songEntity, songs[i]),
                UserWeight = RecommendationConstants.DefaultWeight
            };

            newPairs.Add(pairEntity);
        }
    }
```

Рисунок 3.9 – Нинішня реалізація створення рекомендаційних пар

Джерело: розроблено автором

Вирішенням цієї проблеми є оптимізація коду, запити мають виконуватись максимально асинхронно, і поєднуватись в один потік тільки коли виконання закінчується. Це є дуже цікавою задачею по оптимізації, так як на повільніших мобільних пристроях обробка великих об’ємів інформації може займати ще довше.

3.3 Використання програмного продукту

Першої користувача зустрічає сторінка плеєру (рис. 3.10), яка призначена для відтворення музики та надання користувачеві контролю над процесом.

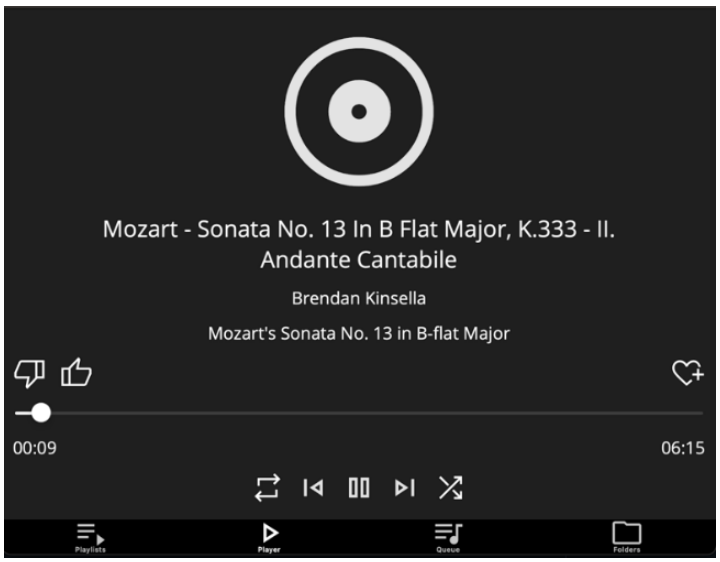

Рисунок 3.10 – Сторінка плеєра
Джерело: розроблено автором

1. *Відображення інформації*: на екрані відображається обкладинка альбому, назва пісні, виконавець та альбом, що зараз відтворюється;
2. *Керування плейлистом*:
   - кнопка «Обране»: по натисканню на кнопку додає поточну пісню до списку обраного або видаляє її з нього;
   - кнопки рекомендацій (якщо пісня рекомендована): кнопки «Палець вгору» та «Палець вниз», допомагають алгоритму рекомендацій краще підібрати музику.
3. *Керування відтворенням*:
   - індикатор прогресу відтворення: перетягування повзунка на переміщує до потрібного моменту пісні;

- кнопка «Повтор черги»: вмикає або вимикає повторення поточної черги відтворення;
- кнопка «Попередня пісня»: переходить до попередньої пісні в черзі, або починає відтворення пісня з самого початку якщо вже було відтворено більше 5и секунд;
- кнопка «Пауза/Продовжити»: призупинить або відновіть відтворення музики;
- кнопка «Наступна пісня»: переходить до наступної пісні в черзі;
- кнопка «Перемішати чергу»: вмикає або вимикає перемішування пісень у поточній черзі.

Зліва від сторінки плеєра розташована сторінка плейлистів, вона призначена для керування і відтворення плейлистів створених користувачем.

1. *Відображення*: відображається список усіх доступних плейлистів, включаючи системні («Всі пісні» та «Улюблені пісні»);
2. *Кнопка «Створити плейлист»*: по натисканню відкриває вікно для введення назви нового плейлиста;
3. *Перегляд плейлиста*: по натисканню на плейлист відображається перелік пісень, що входять до нього;
4. *Відтворення плейлиста*: відтворення плейлиста запускає чергу відтворення пісень у порядку їх відображення починаючи з обраної пісні.

Першою, справа від сторінки плеєра, розташована сторінка черги (рис. 3.11). Вона дозволяє побачити пісні які будуть відтворюватись далі чи вже були відтворені.

1. *Відображення*: відображається перелік пісень у поточній черзі відтворення;
2. *Індикація відтворення*: пісня, що зараз відтворюється, виділена зеленою рамкою;
3. *Індикація рекомендації*: пісні, додані алгоритмом рекомендацій, мають синій фон;

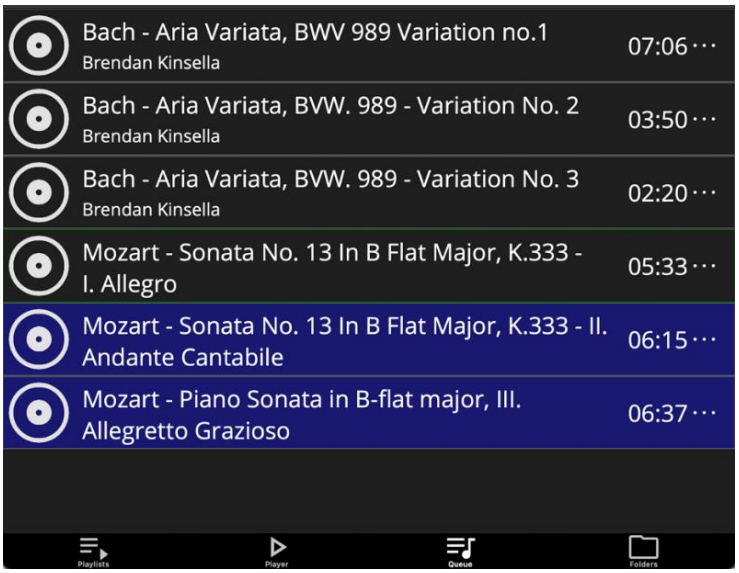

Рисунок 3.11 – Сторінка черги

Джерело: розроблено автором

4. *Дії з піснею (через меню)*: натиснувши на кнопку меню біля інформації про тривалість треку, відкривається вікно з наступними опціями керування піснею (ці опції доступні по всьому застосунку):
   - «Додати в плейлист»: додає пісню до існуючого плейлиста;.
   - «Видалити з плейлиста»: видаляє пісню з існуючого плейлиста;.
   - «Обране»: додає пісню до списку обраного або видаляє її з нього;.
   - «Додати в чергу»: додає пісню до поточної черги відтворення;.
   - «Видалити з черги»: видаляє пісню з поточної черги відтворення.

Остання сторінка яка залишилась це сторінка папок, користувач може переглядати файлову систему і програвати пісні з конкретної обраної папки.

1. *Кнопка «Додати папку з музикою»*: по натисканню дозволяє вибрати основну папку для зберігання музичних файлів;
2. *Відображення*: відображається перелік дочірніх папок відносно обраної основної папки;

3. *Навігація*: натискання на папку розгортає її вміст, дозволяючи «заходити» в дочірні папки;
4. *Відображення пісень*: якщо у дочірній папці присутні музичні файли, вони відображаються у списку;

Також треба звернути увагу на рекомендаційний алгоритм. Він призначений для підбору музики, яка може підійти для прослуховування наступною на основі аналізу бібліотеки і оцінки користувача.

1. *Аналіз музичної бібліотеки:* алгоритм сканує музичну бібліотеку, визначаючи жанри, виконавців, альбоми та інші характеристики треків;
2. *Рекомендації:* на основі аналізу бібліотеки, алгоритм пропонує пісні, які можуть сподобатися. Рекомендації надаються лише за умови:
   - черга відтворення не перебуває в режимі повторення;
   - у базі даних є пісні для рекомендацій;
   - присутні достатньо релевантні пісні.
3. *Зворотній зв'язок користувача:* ви можете налаштувати алгоритм, використовуючи кнопки «палець вгору» та «палець вниз» біля рекомендованих треків:
4. *Кнопка «Палець вгору»:* сигналізує алгоритму про те, що пісня сподобалася, і він повинен частіше висувати цю пісню в чергах де присутні нинішні пісні;
5. *Кнопка «Палець вниз»:* сигналізує алгоритму про те, що пісня не відповідає контексту нинішньої черги, і він повинен уникати подібних рекомендацій у майбутньому.

Висновок до розділу 3

Розроблений програмний продукт реалізовано з використанням мови програмування C# та фреймворку .NET MAUI. Обраний стек технологій зумовлений наявністю досвіду розробки, підтримкою кросплатформної розробки та економією ресурсів при розробці. Для забезпечення

функціональності застосунку використано декілька бібліотек, зокрема CommunityToolkit.Maui.MediaElement для відтворення файлів музики, TagLibSharp для зчитування метаданих та CommunityToolkit.Mvvm для реалізації патерну MVVM. Алгоритм рекомендацій базується на кореляційному аналізі метаданих пісень з можливістю ручного коригування користувачем. Проведено тестування, яке виявило декілька проблем, пов’язаних з використанням бібліотек та оптимізацією обробки великих обсягів даних, що потребують подальшого вдосконалення.

ВИСНОВКИ

У ході виконання даної кваліфікаційної роботи було успішно розроблено концепцію музичного плеєра з персоналізованими рекомендаціями, спрямованого на вирішення проблеми відсутності якісних офлайн плеєрів та частково реалізовано продукт. Напрямок роботи плеєру без використання інтернету є відходженням від тенденції етапу розвитку інформаційних технологій, який настав в 90х роках, в якому компанії-гіганти почали збирати інформацію користувачів, що частково це зумовлено загальним зростанням потужності комп'ютерів, яке надає змогу не надавати свої особисті данні третім особам [29, 30].

Проведено ретельний аналіз існуючих рішень, що дозволило визначити їхні сильні та слабкі сторони, а також сформулювати ключові вимоги до розроблюваного продукту. Особливу увагу було приділено підтримці кросплатформеності, уникненню застарілих патернів дизайну та реалізації алгоритму рекомендацій на основі аналізу поведінки користувача та його бібліотеки.

Використання C# та .NET MAUI дозволило створити кросплатформенний додаток з потенціалом для розгортання на різних операційних системах. Застосування патерну MVVM сприяло покращенню структури коду та полегшенню його підтримки та подальшого розвитку.

Проте, в процесі розробки виявлено ряд проблем, пов'язаних з використанням бібліотеки CommunityToolkit.Maui.MediaElement та оптимізацією обробки великих обсягів даних. Ці проблеми потребують подальшої уваги та вирішення для забезпечення стабільної роботи та високої продуктивності плеєра.

Основні результати виконаної роботи:

- сформовано чітке розуміння потреб ринку в офлайн-музичних плеєрах з персоналізованими рекомендаціями;

- розроблено концепцію та архітектуру програмного продукту, що враховує ключові вимоги до функціональності та зручності використання;
- реалізовано базовий функціонал плеєра, включаючи відтворення музики, створення плейлистів, рекомендації та збір статистики прослуховування;
- визначено потенційні напрямки для подальшого розвитку продукту, зокрема оптимізація алгоритму рекомендацій, покращення роботи з великими бібліотеками музики та розширення функціональності інтерфейсу користувача.

В цілому, виконана робота заклала основу для створення конкурентоспроможного музичного плеєра, який зможе задовольнити потреби аудиторії меломанів, які віддають перевагу офлайн прослуховуванню музики і хотіли б мати вбудовану систему рекомендацій. Подальша розробка та вдосконалення продукту потребуватиме додаткових зусиль та ресурсів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Ipsos. Portable MP3 player ownership reaches new high [Електронний ресурс]. URL: https://www.ipsos.com/en-us/portable-mp3-player-ownership-reaches-new-high (дата звернення 26.04.2025);
2. Statista. Distribution of music industry revenue in the United States from 2017 to 2023, by source [Електронний ресурс]. URL: https://www.statista.com/statistics/186304/revenue-distribution-in-the-us-music-industry/ (дата звернення 26.04.2025);
3. Spotify. Listen offline [Електронний ресурс]. URL: https://support.spotify.com/us/article/listen-offline/ (дата звернення 26.04.2025);
4. Korkeakivi S. The expansion of Spotify and international copyright law: impact on artists // Юридичний журнал «Michigan Journal Of International Law». - 2021. – Том 43. URL: https://www.mjilonline.org/the-expansion-of-spotify-and-international-copyright-law-impact-on-artists/ (дата звернення 26.04.2025);
5. Forbes. The attention war [Електронний ресурс]. URL: https://www.forbes.com/sites/onmarketing/2012/10/19/the-attention-war/ (дата звернення 26.04.2025);
6. Google Play Market. AIMP [Електронний ресурс]. URL: https://play.google.com/store/apps/details?id=com.aimp.player (дата звернення: 14.03.2025);
7. Spotify. Web player: music for everyone [Електронний ресурс]. URL: https://open.spotify.com (дата звернення 26.04.2025);
8. YouTube Music [Електронний ресурс]. URL: https://music.youtube.com (дата звернення 26.04.2025);
9. Foobar2000 [Електронний ресурс]. URL: https://www.foobar2000.org (дата звернення 26.04.2025);
10. Google trends. Пошукові тренди по запиту «spotify wrapped» [Електронний ресурс]. URL: https://trends.google.com/trends/explore?date=today%205-y&geo=US&q=spotify%20wrapped (дата звернення 14.03.2025);
11. Spotify. Smart shuffle breaths new life into your spotify playlist [Електронний ресурс]. URL: https://newsroom.spotify.com/2023-03-08/smart-shuffle-new-life-spotify-playlists/ (дата звернення 14.03.2025);
12. Spotify Forum. How to permanently turn off smart shuffle? [Електронний ресурс]. URL: https://community.spotify.com/t5/Desktop-Windows/How-

to-permanently-turn-off-smart-shuffle/td-p/6141137 (дата звернення 14.03.2025);
13. Foobar2000. Foobar2000 mobile [Електронний ресурс]. URL: https://www.foobar2000.org/mobile (дата звернення 14.03.2025);
14. ID3. ID3 tags provide the Title, Artist, Year, Genre and other great information when you're listening to music. [Електронний ресурс]. URL: https://id3.org (дата звернення 26.04.2025);
15. Refactoring Guru. Суть патерна «Одинак» [Електронний ресурс]. URL: https://refactoring.guru/uk/design-patterns/singleton (дата звернення 26.04.2025);
16. Nuget. CommunityToolkit.Maui.MediaElement [Електронний ресурс]. URL: https://www.nuget.org/packages/CommunityToolkit.Maui.MediaElement (дата звернення 26.04.2025);
17. Nuget. TagLibSharp [Електронний ресурс]. URL: https://www.nuget.org/packages/TagLibSharp (дата звернення 26.04.2025);
18. Nuget. CommunityToolkit.Mvvm [Електронний ресурс]. URL: https://www.nuget.org/packages/CommunityToolkit.Mvvm (дата звернення 26.04.2025);
19. Microsoft learn. Model-View-ViewModel (MVVM) [Електронний ресурс]. URL: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm (дата звернення: 23.04.2025);
20. Nuget. CommunityToolkit.Maui [Електронний ресурс]. URL: https://www.nuget.org/packages/CommunityToolkit.Maui (дата звернення 26.04.2025);
21. Nuget. Microsoft.Maui.Controls [Електронний ресурс]. URL: https://www.nuget.org/packages/Microsoft.Maui.Controls (дата звернення 26.04.2025);
22. Nuget. CommunityToolkit.Maui.Core [Електронний ресурс]. URL: https://www.nuget.org/packages/CommunityToolkit.Maui.Core (дата звернення 26.04.2025);
23. Nuget. Microsoft.Extensions.Logging.Debug [Електронний ресурс]. URL: https://www.nuget.org/packages/Microsoft.Extensions.Logging.Debug (дата звернення 26.04.2025);
24. Nuget. Microsoft.NET.ILLink.Tasks [Електронний ресурс]. URL: https://www.nuget.org/packages/Microsoft.NET.ILLink.Tasks (дата звернення 26.04.2025);

25. Nuget. sqlite-net-pcl [Електронний ресурс]. URL: https://www.nuget.org/packages/sqlite-net-pcl (дата звернення 26.04.2025);
26. xUnit patterns. Happy path [Електронний ресурс]. URL: http://xunitpatterns.com/happy%20path.html (дата звернення 26.04.2025)
27. Basu A., Software quality assurance, testing and metrics (2015), Індія, Делі: видавництво «PHI Learning Pvt. Ltd.», 2015. С. 150. ISBN 10: 8120350685;
28. GitHub CommunityToolkit. MediaElement get wrong position or seek to wrong position with high res audio files. [Електронний ресурс]. URL: https://github.com/CommunityToolkit/Maui/issues/1766 (дата звернення 26.04.2025);
29. Пономаренко Ю. Рач В. Глобальна цифровізація як основа нового етапу розвитку галузі «Інформаційні технології». // III Міжнародна конференція «Держава, регіони, підприємництво: інформаційні, суспільно-правові, соціально-економічні аспекти розвитку» (м. Київ, 2 грудня 2021 року). – Університет економіки та права «КРОК», 2021. URL: https://conf.krok.edu.ua/SRE/SRE-2021/paper/view/781 (дата звернення 26.04.2025);
30. CPU Benchmark. Year on year performance. [Електронний ресурс]. URL: https://www.cpubenchmark.net/year-on-year.html (дата звернення 26.04.2025);